

PERCAKAPAN ANTARMUKA PENGGUNA (User Interface) DALAM CHATBOT



PERCAKAPAN ANTARMUKA PENGGUNA (User Interface) DALAM CHATBOT

Penulis :

Dr. Budi Raharjo, S.Kom., M.Kom., MM.

ISBN : 978-623-8120-66-6

Editor :

Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

Penyunting :

Dr. Joseph Teguh Santoso, M.Kom.

Desain Sampul dan Tata Letak :

Irdha Yuniyanto, S.Ds., M.Kom

Penebit :

Yayasan Prima Agus Teknik

Redaksi :

Jl. Majapahit no 605 Semarang

Telp. (024) 6723456

Fax. 024-6710144

Email : penerbit_ypat@stekom.ac.id

Distributor Tunggal :

Universitas STEKOM

Jl. Majapahit no 605 Semarang

Telp. (024) 6723456

Fax. 024-6710144

Email : info@stekom.ac.id

Hak cipta dilindungi undang-undang

**Dilarang memperbanyak karya tulis ini dalam bentuk dan dengan cara
apapun tanpa izin dari penulis**

KATA PENGANTAR

Puji Syukur kepada Tuhan Yang Maha Esa, Karena atas Rahmat -Nya, Penulis dapat menyelesaikan buku ajar yang berjudul **“PERCAKAPAN ANTAR MUKA PENGGUNA (User Interface) DALAM CHATBOT”**. Buku ini ditulis untuk membantu pengajar atau dosen dan mahasiswa yang membutuhkan berbagai materi tentang “membuat chatbot diberbagai platform dan sumber”.

Di era di mana teknologi terus mendefinisikan ulang cara kita berinteraksi dengan dunia digital, chatbot telah muncul sebagai inovasi yang menarik dan sangat diperlukan. Agen virtual yang didukung kecerdasan buatan ini telah melampaui kebaruan awal mereka untuk menjadi bagian integral dari kehidupan kita sehari-hari, mulai dari layanan pelanggan dan e-commerce hingga bantuan pribadi dan seterusnya. Chatbots, pada intinya, adalah program komputer canggih yang dirancang untuk terlibat dalam percakapan bahasa alami dengan manusia, memberikan informasi, menjawab pertanyaan, dan bahkan melakukan tugas. Chatbot telah merevolusi cara bisnis dan individu berkomunikasi, meningkatkan efisiensi, aksesibilitas, dan kenyamanan.

Chatbot hadir dalam berbagai bentuk, mulai dari sistem berbasis aturan yang mengikuti skrip yang telah ditentukan hingga model berbasis AI yang lebih canggih yang mampu memahami dan merespons nuansa bahasa manusia. Fleksibilitas dan kemampuan beradaptasi mereka telah menjadikannya alat yang ampuh di berbagai domain, membantu organisasi merampingkan operasi, meningkatkan pengalaman pengguna, dan meningkatkan layanan dukungan mereka.

Pengenalan ini akan menggali lebih dalam ke dunia chatbot, mengeksplorasi evolusi, aplikasi, dan dampak besar yang mereka miliki pada lanskap digital kita. Bergabunglah bersama kami saat kami memulai perjalanan melalui dunia chatbot yang rumit, di mana teknologi bertemu dengan percakapan untuk menciptakan interaksi yang mulus dan personal.

Program komputer yang dirancang untuk mensimulasikan percakapan manusia, biasanya melalui interaksi berbasis teks atau suara. Chatbot didukung oleh algoritme kecerdasan buatan (AI), yang memungkinkannya untuk memahami dan merespons pertanyaan atau perintah pengguna dengan cara percakapan. Chatbot dapat diintegrasikan ke dalam berbagai platform, termasuk situs web, aplikasi perpesanan, dan asisten suara, untuk memberikan informasi, bantuan, dan terlibat dalam percakapan yang bermakna dengan pengguna.

Fungsi utama chatbot meliputi: *Menjawab Pertanyaan*: Chatbots dapat memberikan jawaban yang cepat dan akurat untuk pertanyaan pengguna, sehingga berguna untuk dukungan pelanggan, pencarian informasi, dan FAQ. *Mengotomatiskan Tugas*: Mereka dapat melakukan tugas-tugas seperti mengatur pengingat, mengirim pemberitahuan, membuat reservasi, dan memesan produk atau layanan atas nama pengguna. *Personalisasi*: Chatbots dapat menganalisis data dan perilaku pengguna untuk menawarkan rekomendasi atau konten yang dipersonalisasi, meningkatkan pengalaman pengguna. *Keterlibatan*: Chatbots digunakan untuk melibatkan dan menghibur pengguna. Mereka dapat bermain game, bercerita, atau

mensimulasikan percakapan untuk hiburan dan persahabatan. *Mengumpulkan Informasi*: Chatbot dapat mengumpulkan informasi pengguna, umpan balik, atau tanggapan survei, yang dapat digunakan untuk pemasaran, peningkatan produk, atau penelitian. *Bantuan dan Bimbingan*: Chatbots dapat memandu pengguna melalui proses, seperti memecahkan masalah teknis atau membantu mereka melengkapi formulir dan aplikasi.

Dalam bab pertama buku ini berisi pendahuluan, pada bab kedua membahas ChatBot Tercipta, dan pada bab ketiga mengenai Membuat ChatBot. Bab keempat mengenai Membangun ChatBot Dengan Cara Mudah, dan pada bab kelima mengenai Membangun ChatBot Dengan Cara Yang Sulit. Selanjutnya di bab ke enam Merancang Dan Membangun ChatBot, pada bab ketujuh mengenai Merancang Dan Mengimplementasikan ChatBot dan bab kedelapan Membangun ChatBot Menggunakan Bahasa Alami.

Bab kesembilan mengenai Pemrosesan Bahasa Alami Untuk ChatBot, bab kesepuluh Membangun ChatBot Di Beberapa Platform, bab kesebelas mengenai Menyebarkan ChatBot Di Web dan bab keduabelas Menjelajahi Percakapan dari Bot Builder. Bab keempat belas berisi mengenai Membangun Keterampilan Di Asisten Google dan bab kelima belas sebagai bab terakhir membahas mengenai Pembuatan Bot Tingkat Lanjut.

Demikian buku ajar ini kami buat dengan harapan agar pembaca dapat memahami informasi dan juga mendapatkan wawasan mengenai bidang teori desain serta dapat bermanfaat bagi Masyarakat dalam arti luas. Terima kasih.

Semarang, Oktober 2023

Penulis

Dr. Budi Raharjo, S.Kom., M.Kom., MM.

DAFTAR ISI

Halaman Judul	i
Kata Pengantar	ii
Daftar Isi	iv
BAB 1 PENDAHULUAN	1
1.1. Antarmuka Pengguna Percakapan	1
1.2. Sejarah Singkat Chatbot	2
1.3. Perkembangan Terkini	3
1.4. Alat Pengembangan Chatbot	8
1.5. Layanan Direktori	10
1.6. Monetisasi	10
1.7. Manfaat chatbot	10
1.8. Chatbot Akan Tetap Ada	11
BAB 2 CHATBOT TERCIPTA	13
2.1. Popularitas Penggunaan Chatbots	14
2.2. Kebutuhan Chatbot	16
2.3. Perspektif Bisnis	17
2.4. Chatbots Mendatangkan Pendapatan	17
2.5. Sekilas Penggunaan Chatbot	19
2.6. Pelanggan Lebih Memilih Chatbots	20
2.7. Rilis Fitur dan Perbaiki Bug	21
2.8. Industri yang Terkena Dampak Chatbots	22
2.9. Garis Waktu Singkat Chatbots	22
2.10. Dimulai Dengan Chatbots	27
2.11. Pohon Keputusan di Chatbots	27
2.12. Komponen Chatbot dan Terminologi yang Digunakan	31
BAB 3 MEMBUAT CHATBOT	34
3.1. Menggunakan Chatfuel	34
3.2. Blok Bangunan Dasar	38
3.3. Mengelola Konteks	47
3.4. Kartu Masukan Pengguna	48
3.5. Variabel Kontekstual Default	50
3.6. Pemrosesan Bagian Belakang	53
3.7. Template Bot	60
BAB 4 MEMBANGUN CHATBOTS DENGAN CARA MUDAH	61
4.1. Pengantar Dialogflow	61
4.2. Mulai Membuat Chatbot	62
4.3. Membangun Chatbot Pemesanan Makanan	63

4.4.	Memahami Dan Membalas Kembali Ke Pengguna	70
4.5.	Menyebarkan Dialogflow Chatbot di Web	74
4.6.	Integrasikan Dialogflow Chatbot di Facebook Messenger	77
4.7.	Menyiapkan Konsol Dialogflow	78
4.8.	URL Panggilan Balik	79
4.9.	Mengonfigurasi Webhook	80
4.10.	Menguji Bot Messenger	81
4.11.	Pemenuhan	84
4.12.	Memeriksa Respon	87
BAB 5	MEMBANGUN CHATBOTS DENGAN CARA YANG SULIT	90
5.1.	Pendahuluan	90
5.2.	Apa itu Rasa NLU?	90
5.3.	Mengapa Saya Harus Menggunakan Rasa NLU?	91
5.4.	Menyelam Langsung Ke Rasa NLU	91
5.5.	Memutuskan Saluran Pipa di Rasa	92
5.6.	Melatih dan Membangun Chatbot Dari Awal	93
5.7.	Mempersiapkan Data untuk Chatbot	94
5.8.	Memvisualisasikan Dan Memodifikasi Format Data JSON Rasa	96
5.9.	Konfigurasi Yaml	98
5.10.	Menulis Kode Python untuk Melatih Model dan Memprediksi	99
5.11.	Melatih Model	100
5.12.	Manajemen Dialog Menggunakan Rasa Core	101
5.13.	Memahami Konsep Rasa	104
5.14.	Slot	105
5.15.	Templat	106
5.16.	Membuat File Domain untuk Chatbot	106
5.17.	Menulis Tindakan Kustom dari Chatbot	108
5.18.	Persiapan Data untuk Pelatihan Bot	111
5.19.	Membuat Data Cerita	112
5.20.	Melatih Model Agen Chatbot	114
5.21.	Rasa Core Terbaru	116
5.22.	Mengekspor Percakapan Sebagai Cerita	126
BAB 6	MERANCANG DAN MEMBANGUN CHATBOT	129
6.1.	Menjelajahi API Transportasi	129
6.2.	Membuat Akun Pengembang	130
6.3.	Desain Percakapan	134
6.4.	Membangun Bot SMS Sederhana	136
6.5.	Menjadwalkan Tugas	143
6.6.	Membangun Chatbot Dua Arah	145
6.7.	Memahami Ucapan Pengguna	150

BAB 7	MERANCANG DAN MENGIMPLEMENTASIKAN CHATBOT	163
7.1.	Tugas Percakapan	163
7.2.	Desain Percakapan	164
7.3.	Tugas Latar Belakang	165
7.4.	Membangun Antarmuka Backend	168
7.5.	Menerapkan Chatbot	172
7.6.	Membuat Layanan Web Chatbot	178
7.7.	Membuat Aplikasi Facebook	187
7.8.	Membuat Aplikasi Web Antarmuka Facebook	190
7.9.	Menghubungkan Aplikasi Facebook Dan Aplikasi Antarmuka	193
7.10.	Menguji Chatbot Anda	194
7.11.	Mempublikasikannya	195
7.12.	Lebih banyak format pesan Facebook	196
BAB 8	MEMBANGUN CHATBOT MENGGUNAKAN BAHASA ALAMI	203
8.1.	Memperkenalkan Dialogflow	203
8.2.	Menyiapkan Dialogflow	203
8.3.	Menyebarkan Chatbot	207
8.4.	Integrasi Situs Web	208
8.5.	Entitas	216
8.6.	Agen Impor Dan Ekspor	227
BAB 9	PEMROSESAN BAHASA ALAMI UNTUK CHATBOT	239
9.1.	Niat	239
9.2.	Entitas	256
BAB 10	PEMROSESAN BAHASA ALAMI UNTUK CHATBOTS	261
10.1.	Apa Itu Spacy?	262
10.2.	Pembelajaran Mendalam	263
10.3.	Instalasi Dan Prasyarat	264
10.4.	Metode Dasar NLP Untuk Membangun Chatbots	267
10.5.	Penguraian Ketergantungan	271
10.6.	Visualisasi Interaktif Untuk Parsing Ketergantungan	278
10.7.	Potongan Kata Benda	281
10.8.	Menemukan Kesamaan	282
10.9.	Senang Mengetahui Hal-Hal Di NLP Untuk Chatbots	284
BAB 11	INTEGRASI CHATBOT PADA TWEETER	287
11.1.	Memulai Dengan Aplikasi Twitter	287
11.2.	Membangun Bot Twitter Pertama Anda	290
11.3.	Menjelajahi Sdk Twitter	293
11.4.	Membuat Database Minat Pengguna	305
BAB 12	MENJELAJAHI PERCAKAPAN DARI BOT BOUILDER	317
12.1.	Kerangka Bot Ms	317
12.2.	Membangun Bot	318

12.3.	Alur Percakapan	331
12.4.	Menanggapi Ucapan Pengguna	334
12.5.	Peralihan Konteks	338
12.6.	Nlu Kontekstual	340
12.7.	Mengakhiri Percakapan	341
12.8.	Memulai Zomato	342
12.9.	Bot Pencarian Restoran	345
12.10.	Menghubungkan Ke Skype	353
BAB 13	MENJELAJAHI PERANGKAT ASISTEN PINTAR DENGAN AI	358
13.1.	Mengonfigurasi Keahlian Anda	360
13.2.	Model Interaksi	362
13.3.	Manajer Percakapan – Kutipan	369
13.4.	Membangun Model Interaksi	375
13.5.	Intent Dan Jenis Slot Bawaan	387
BAB 14	MEMBANGUN KETERAMPILAN DI ASISTEN GOOGLE	389
14.1.	Todois	389
14.2.	Membuat Aplikasi Web	399
14.3.	Menguji agen di Dialogflow	402
14.4.	Mengintegrasikan dengan Asisten Google	405
14.5.	Uji Di Beranda Google	409
BAB 15	PEMBUATAN BOT TINGKAT LANJUT	410
15.1.	Prinsip Desain	410
15.2.	Pesan Multimedia	415
15.3.	Menampilkan Hasil Produk	418
15.4.	Mengintegrasikan Maksud Pencarian Lokasi	430
15.5.	Menyiapkan Pesan	434
15.6.	Membangun Pengklasifikasi	439
15.7.	Bisnis Dan Monetisasi	449
15.8.	Jumlah Pengguna	452
15.9.	Analisis Niat	455
15.10.	Kasus Penggunaan Chatbot	456
15.11.	Cara Komunikasi	457
15.12.	Chatbots Menurut Vertikal Industri	458
Daftar Pustaka	463

BAB 1

PENDAHULUAN

Era mesin cerdas telah tiba, dan antarmuka percakapan menjadi yang terdepan. Selama beberapa tahun terakhir, kita telah dikerumuni oleh sejumlah jenis mesin dan perangkat lunak baru yang dikenal sebagai bot. Bot adalah mesin perangkat keras atau perangkat lunak otomatis yang didukung oleh kemajuan teknologi Kecerdasan Buatan (AI). Perkembangan terkini dalam algoritme pembelajaran mesin, seperti pembelajaran mendalam dan pembelajaran penguatan mendalam, telah meningkatkan kinerja tugas AI seperti Pengenalan Ucapan Otomatis (ASR), Pemahaman Bahasa Alami (NLU), Sintesis Teks ke Ucapan (TTS), dan Pengenalan Gambar. Hal ini telah mempercepat perjalanan umat manusia menuju singularitas teknologi, suatu titik ketika AI melampaui kecerdasan alami manusia dengan pesat.

Salah satu tujuan jangka panjang di bidang AI adalah membangun sistem komputer yang dapat melakukan percakapan seperti manusia dengan pengguna. Dengan kemajuan teknologi AI terkini, kita kini selangkah lebih dekat untuk mencapai tujuan ini. Kini, bukan lagi fiksi bahwa kita bisa berinteraksi dengan perangkat dan gadget di rumah dan kantor kita hanya dengan menggunakan suara. Jalan kita masih panjang untuk menciptakan standar dan membangun makhluk digital yang mampu melakukan percakapan bahasa alami dengan lancar. Namun, lonjakan minat dan investasi besar-besaran baru-baru ini dalam mewujudkan ide-ide ini menunjukkan bahwa kita berada pada jalur yang tepat untuk mengembangkan standar global tersebut. Jika Anda tertarik dengan perkembangan terkini dalam teknologi AI dan otomasi, buku ini cocok untuk Anda. Kita akan memulai perjalanan menuju titik waktu di masa depan yang oleh pakar desain Mark Curtis disebut sebagai singularitas percakapan, ketika perangkat percakapan menghilang dan percakapan antara manusia dan mesin berjalan mulus dan alami.

Ini adalah buku untuk programmer yang mulai membangun antarmuka percakapan. Saat ini, chatbot berbasis tombol dasar dapat dibuat bahkan tanpa harus menulis satu baris kode pun. Dalam buku ini, kita akan mulai dari situ. Kami secara bertahap akan beralih ke arsitektur yang lebih kompleks dan fleksibel, dan kami akan menjelajahi saluran yang akan digunakan, seperti Facebook Messenger, SMS, dan Twitter. Kami juga akan mengeksplorasi alat untuk memahami bahasa alami dan manajemen percakapan seiring dengan langkah kami. Terakhir, kami akan mengakhiri perjalanan kami dengan membuat bot berkemampuan suara di platform seperti Amazon Alexa dan Google Assistant.

1.1 Antarmuka Pengguna Percakapan

Antarmuka pengguna percakapan sama tuanya dengan komputer modern itu sendiri. ENIAC, komputer tujuan umum pertama yang dapat diprogram, dibuat pada tahun 1946. Pada tahun 1950, Alan Turing, seorang ilmuwan komputer Inggris, mengusulkan untuk mengukur tingkat kecerdasan dalam mesin menggunakan tes percakapan yang disebut tes

Turing. Pengujian tersebut melibatkan mesin yang bersaing dengan manusia sebagai mitra dialog dengan sekelompok juri manusia (manusia lainnya). Para juri akan berinteraksi dengan masing-masing peserta (manusia dan mesin) menggunakan antarmuka tipe teks yang tidak berbeda dengan kebanyakan aplikasi obrolan perpesanan modern. Melalui obrolan, juri seharusnya mengidentifikasi siapa di antara dua peserta yang merupakan mesin. Jika minimal 30% juri tidak dapat membedakan kedua peserta, maka mesin dianggap lulus ujian. Ini adalah salah satu pemikiran manusia paling awal mengenai antarmuka percakapan dan pengaruhnya terhadap tingkat kecerdasan mesin yang memiliki kemampuan tersebut. Namun, upaya untuk membangun antarmuka seperti itu tidak terlalu berhasil selama beberapa dekade berikutnya.

Selama sekitar 35 tahun, sejak tahun 1980an, Graphical User Interface (GUI) telah mendominasi cara kita berinteraksi dengan mesin. Dengan perkembangan AI terkini dan semakin besarnya kendala seperti menyusutnya ukuran gadget (dari laptop hingga ponsel), berkurangnya ruang di layar (jam tangan pintar), dan kebutuhan akan antarmuka yang tidak terlihat (rumah pintar dan robot), komunikasi antarmuka pengguna sekali lagi menjadi kenyataan. Misalnya, cara terbaik untuk berinteraksi dengan robot seluler yang mendistribusikan gadget di rumah pintar adalah menggunakan suara. Oleh karena itu, sistem harus mampu memahami permintaan dan tanggapan pengguna dalam bahasa alami manusia. Kemampuan sistem seperti itu dapat mengurangi upaya manusia dalam mempelajari dan memahami antarmuka kompleks saat ini.

Antarmuka pengguna percakapan dikenal dengan beberapa nama: antarmuka bahasa alami, sistem dialog lisan, chatbot, agen virtual cerdas, asisten virtual, dan sebagainya. Perbedaan sebenarnya antara sistem-sistem ini adalah dalam hal integrasi backend (misalnya, database, dan modul tugas/kontrol), modalitas (misalnya, teks, suara, dan avatar visual), dan saluran tempat sistem diterapkan. Namun, salah satu tema umum di antara sistem ini adalah kemampuannya untuk berinteraksi dengan pengguna dalam percakapan menggunakan bahasa alami.

1.2 Sejarah Singkat Chatbot

Asal usul chatbot modern dapat ditelusuri kembali ke tahun 1964 ketika Joseph Weizenbaum di Massachusetts Institute of Technology (MIT) mengembangkan chatbot bernama Eliza. Itu menggunakan aturan percakapan sederhana dan mengulangi sebagian besar perkataan pengguna untuk mensimulasikan terapis Rogerian. Meskipun hal ini menunjukkan bahwa pengguna yang naif mungkin tertipu dan berpikir bahwa mereka sedang berbicara dengan terapis sebenarnya, sistem itu sendiri tidak memahami masalah pengguna. Setelah itu, pada tahun 1991, hadiah Loebner dilembagakan untuk mendorong peneliti AI membangun chatbot yang dapat mengalahkan uji Turing dan memajukan AI. Meskipun tidak ada chatbot yang mampu mengalahkan ujian tersebut hingga tahun 2014, banyak chatbot terkenal yang memenangkan hadiah karena memenangkan tantangan terbatas lainnya. Ini termasuk ALICE, JabberWacky, Rose, dan Mitsuku. Namun, pada tahun 2014, dalam kompetisi tes Turing untuk menandai peringatan 60 tahun kematian Alan Turing, sebuah chatbot bernama Eugene Goostman, yang memerankan seorang anak

berusia 13 tahun, berhasil menipu 33% juri sehingga memenangkan tes tersebut. Artificial Intelligence Markup Language (AIML) dan ChatScript dikembangkan sebagai cara untuk menyusun pengetahuan dan konten percakapan untuk sebagian besar chatbot ini. Skrip yang dikembangkan menggunakan bahasa skrip ini kemudian dapat dimasukkan ke dalam penerjemah untuk menciptakan perilaku percakapan. Chatbots yang dikembangkan untuk mengalahkan tes Turing sebagian besar cerewet hanya dengan satu tujuan—untuk mengalahkan tes Turing. Hal ini tidak dianggap oleh banyak orang sebagai kemajuan dalam AI atau pengembangan asisten percakapan yang berguna.

Di sisi lain, penelitian di bidang kecerdasan buatan, khususnya pembelajaran mesin dan pemrosesan bahasa alami, memunculkan berbagai antarmuka percakapan seperti sistem penjawab pertanyaan, antarmuka bahasa alami ke database, dan sistem dialog lisan. Tidak seperti chatbot yang dibuat untuk mengalahkan uji Turing, sistem ini memiliki tujuan yang sangat jelas.

Sistem penjawab pertanyaan memproses pertanyaan bahasa alami dan menemukan jawaban dalam kumpulan data teks tidak terstruktur. Antarmuka Bahasa Alami ke Sistem Basis Data (NLIDBS) adalah antarmuka ke database SQL besar yang menafsirkan kueri database yang diajukan dalam bahasa alami seperti bahasa Inggris, mengubahnya menjadi SQL, dan mengembalikan hit sebagai respons.

Sistem Dialog Lisan (SDS) adalah sistem yang dapat mempertahankan percakapan kontekstual dengan pengguna untuk menangani tugas percakapan seperti memesan tiket, mengendalikan sistem lain, dan membimbing pelajar. Ini adalah cikal bakal chatbot modern dan antarmuka percakapan.

1.3 Perkembangan Terkini

Pada tahun 2011, Apple merilis asisten cerdas bernama Siri sebagai bagian dari iPhone mereka. Siri dimodelkan untuk menjadi asisten pribadi pengguna, melakukan tugas-tugas seperti melakukan panggilan, membaca pesan, dan mengatur alarm dan pengingat. Ini adalah salah satu peristiwa paling penting di masa lalu yang menghidupkan kembali kisah antarmuka percakapan. Pada masa awal Siri, pengguna hanya menggunakannya beberapa kali dalam sebulan untuk melakukan tugas seperti mencari di internet, mengirim SMS, dan melakukan panggilan telepon. Meskipun baru, Siri dianggap sebagai pekerjaan yang sedang berjalan dengan lebih banyak fitur yang akan ditambahkan di tahun-tahun berikutnya. Pada masa-masa awal, Siri memiliki banyak klon dan persaingan di Android dan platform ponsel pintar lainnya. Sebagian besar dimodelkan sebagai asisten dan tersedia sebagai aplikasi seluler.

Pada tahun yang sama (2011), IBM memperkenalkan Watson, sistem penjawab pertanyaan yang berpartisipasi dalam acara permainan bernama Jeopardy dan memenangkannya melawan pemenang manusia sebelumnya, Brad Rutter dan Ken Jennings. Hal ini menandai tonggak sejarah AI karena Watson mampu memproses pertanyaan bahasa alami domain terbuka dan menjawabnya secara real time.

Sejak itu, Watson telah diubah menjadi sebuah perangkat dengan serangkaian alat layanan kognitif untuk pemahaman bahasa alami, analisis sentimen, manajemen dialog, dan sebagainya.

Mengikuti Siri dan Watson, pengumuman besar berikutnya datang dari Microsoft pada tahun 2013, ketika mereka memperkenalkan Cortana sebagai fitur standar pada ponsel Windows dan kemudian pada tahun 2015 pada OS Windows 10. Seperti Siri, Cortana adalah asisten pribadi yang mengatur tugas-tugas seperti mengatur pengingat, menjawab pertanyaan, dan sebagainya.

Pada bulan November 2014, Amazon mengundang anggota Perdana untuk mencoba asisten pribadinya yang disebut Alexa. Alexa tersedia di produk Amazon sendiri yang disebut Echo. Echo adalah speaker pintar pertama yang memiliki asisten seperti "hantu" di dalam mesin. Meski disebut speaker, sebenarnya ini adalah komputer kecil dengan suara sebagai satu-satunya antarmuka, tidak seperti ponsel pintar, tablet, dan komputer pribadi. Pengguna dapat berbicara dengan Alexa menggunakan suara, memintanya melakukan tugas seperti mengatur pengingat, memutar musik, dan lain sebagainya.

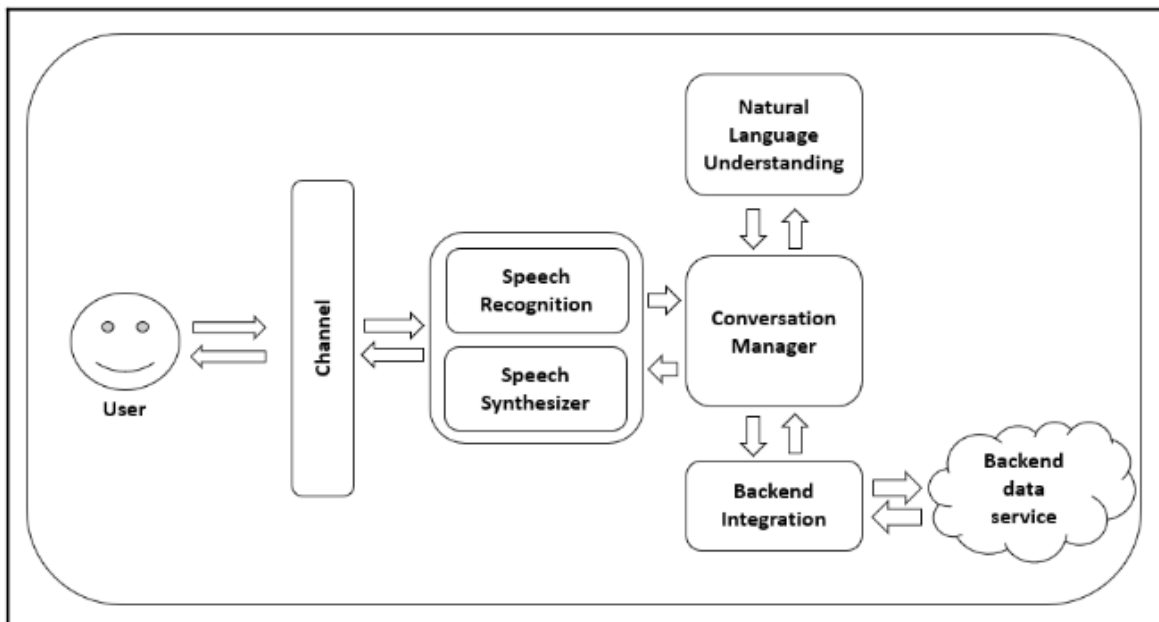
Baru-baru ini, pada bulan April 2016, Facebook mengumumkan bahwa mereka membuka platform Messenger populernya untuk chatbots. Ini adalah pendekatan antarmuka percakapan yang sangat berbeda dibandingkan dengan Siri, Alexa, dan Cortana. Berbeda dengan asisten pribadi ini, pengumuman Facebook mengarah pada terciptanya chatbot yang dibuat khusus dan diberi merek. Bot ini sangat mirip dengan Siri, Cortana, dan Alexa, tetapi dapat disesuaikan dengan kebutuhan bisnis yang membangunnya. Chatbots kini siap untuk mendisrupsi beberapa pasar, termasuk layanan pelanggan, penjualan, pemasaran, dukungan teknis, dan sebagainya. Banyak platform perpesanan, seperti Skype, Telegram, dan lainnya, juga membuka chatbot pada waktu yang hampir bersamaan.

Pada bulan Mei 2016, Google mengumumkan Asisten, versi chatbot pribadinya yang dapat diakses di berbagai platform seperti aplikasi Allo dan Google Home (speaker pintar seperti Echo). Semua asisten seperti Siri, Cortana, Alexa, dan Google Assistant juga telah dibuka sebagai saluran untuk kemampuan percakapan pihak ketiga. Jadi, kini Alexa dan Asisten Google Anda dapat dipersonalisasi dengan menambahkan kemampuan percakapan (disebut keterampilan atau tindakan) dari perpustakaan solusi pihak ketiga. Sama seperti merek yang dapat mengembangkan chatbot mereka sendiri untuk berbagai layanan perpesanan (misalnya, Skype dan Facebook Messenger), mereka juga dapat mengembangkan keterampilan untuk Alexa atau tindakan untuk Asisten Google. Speaker pintar milik Apple, Homepod, yang didukung oleh Siri, dijadwalkan akan dirilis pada tahun 2018.

Sejalan dengan perkembangan ini, terdapat juga pertumbuhan besar dalam hal alat yang tersedia untuk membangun dan menghosting chatbots. Selama dua tahun terakhir, terdapat pertumbuhan eksponensial dalam alat untuk merancang, meniru, membangun, menerapkan, mengelola, dan memonetisasi chatbots. Hal ini menghasilkan terciptanya ekosistem yang merancang dan membangun antarmuka percakapan khusus untuk bisnis, badan amal, pemerintahan, dan organisasi lain di seluruh dunia.

Arsitektur antarmuka pengguna percakapan

Di bagian ini, mari kita lihat arsitektur dasar antarmuka percakapan:



Gambar 1.1 Arsitektur Dasar Antarmuka Percakapan

Modul inti dari antarmuka percakapan adalah pengelola percakapan. Modul ini mengontrol alur percakapan. Dibutuhkan representasi semantik dari apa yang dikatakan pengguna sebagai masukan, dan memutuskan respons sistem yang seharusnya. Ini akan mempertahankan representasi konteks percakapan dalam beberapa bentuk, misalnya sekumpulan pasangan nilai kunci, untuk melakukan percakapan secara bermakna dalam beberapa putaran antara pengguna dan sistem.

Representasi semantik dari input pengguna dapat langsung dimasukkan dengan menekan tombol. Dalam sistem yang dapat memahami bahasa, ucapan pengguna akan diterjemahkan ke dalam representasi semantik, yang terdiri dari maksud dan parameter pengguna (slot dan entitas), melalui modul pemahaman bahasa alami. Modul ini mungkin perlu dilatih sebelumnya untuk memahami serangkaian maksud pengguna yang diidentifikasi oleh pengembang terkait dengan tugas percakapan yang ada.

Antarmuka berkemampuan suara yang menerima masukan ucapan pengguna juga memerlukan modul pengenalan ucapan yang dapat mentranskripsikan ucapan menjadi teks sebelum memasukkannya ke dalam modul pemahaman bahasa alami. Secara simetris, di sisi lain, diperlukan modul penyintesis ucapan (atau mesin text-to-speech) yang mengubah respons teks sistem menjadi ucapan.

Manajer percakapan akan berinteraksi dengan modul backend. Ini bisa berupa database atau sumber data online yang ditanyakan untuk menjawab pertanyaan pengguna (misalnya, jadwal TV) atau layanan online untuk melaksanakan instruksi pengguna (misalnya, memesan tiket).

Saluran adalah tempat chatbot bertemu dengan pengguna. Tergantung pada salurannya, mungkin ada satu atau lebih modul yang membentuk lapisan ini. Misalnya, jika

chatbot ada di Facebook Messenger, lapisan ini terdiri dari Halaman Facebook dan Aplikasi Facebook yang terhubung ke modul chatbot lainnya yang dibungkus sebagai aplikasi web.

Klasifikasi

Antarmuka pengguna percakapan telah diterapkan dalam berbagai skenario. Penerapannya dapat diklasifikasikan secara luas menjadi dua kategori: asisten perusahaan dan asisten pribadi.

Asisten Perusahaan adalah chatbot dan antarmuka pengguna percakapan lainnya yang meniru perwakilan layanan pelanggan dan asisten toko. Sama seperti perwakilan layanan pelanggan manusia, bot melibatkan pelanggan dalam percakapan melakukan tugas pemasaran, penjualan, dan dukungan. Sebagian besar chatbot yang diterapkan di saluran seperti Facebook Messenger, Skype, Slack, dan banyak lagi adalah asisten perusahaan. Mereka dirancang dan dibangun untuk melakukan tugas-tugas yang akan dilakukan oleh asisten toko dan perwakilan layanan pelanggan. Asisten perusahaan sedang dikembangkan di banyak sektor bisnis, mengotomatiskan berbagai tugas percakapan.

Di sisi lain, asisten pribadi adalah bot seperti Alexa, Siri, dan Cortana, yang bertindak sebagai asisten pribadi pengguna, melakukan tugas-tugas seperti mengelola kalender, mengirim teks, menerima panggilan, dan memutar musik. Asisten pribadi ini dapat diperluas kemampuannya. Misalnya, Alexa memungkinkan peningkatan tersebut dengan membiarkan pengembang membangun keterampilan yang dapat dipilih pengguna untuk ditambahkan ke Alexa mereka sendiri. Oleh karena itu, merek dapat mengembangkan keterampilan untuk Alexa atau tindakan untuk Asisten Google yang memungkinkan Alexa dan Asisten berinteraksi dengan layanan TI merek dan melakukan tugas seperti melakukan pemesanan, memeriksa status pengiriman, dan banyak lagi. Misalnya, merek populer seperti PizzaHut, Starbucks, dan Domino's telah mengembangkan keterampilan yang dapat diaktifkan di Alexa.

Aplikasi

Meskipun chatbot telah dikembangkan setidaknya selama beberapa dekade, hingga saat ini, chatbot belum menjadi saluran utama untuk keterlibatan pelanggan. Selama dua tahun terakhir, berkat upaya serius yang dilakukan oleh raksasa industri seperti Apple, Google, Microsoft, Facebook, IBM, dan Amazon, serta investasi mereka selanjutnya dalam mengembangkan perangkat, chatbot, dan antarmuka percakapan telah menjadi pesaing serius bagi saluran kontak pelanggan lainnya. Saat ini, chatbots telah diterapkan di berbagai sektor dan berbagai skenario percakapan dalam sektor-sektor ini: ritel, perbankan dan keuangan, pemerintahan, kesehatan, hukum dan sektor ketiga, dan banyak lagi.

Di ritel, chatbots telah diterapkan untuk pemasaran produk, keterlibatan merek, bantuan produk, penjualan, dan percakapan dukungan. Chatbot keterlibatan merek menawarkan tips dan saran kepada pelanggan setia suatu merek terkait dengan penggunaan produk yang dijual oleh merek tersebut. Misalnya, chatbot Sephora memberi saran kepada pengguna tentang cara memilih lipstik ideal mereka. Demikian pula, chatbot TK-Maxx membantu pengguna dalam memilih hadiah untuk teman dan keluarga mereka selama Natal 2016. Salah satu pengecer pertama yang mengeksplorasi chatbot untuk penjualan adalah H&M. Chatbot H&M membantu pengguna menelusuri katalog produk dan menambahkan

produk ke keranjang belanja mereka. Produsen mobil seperti Tesla, Kia, dan Mercedes telah mengembangkan chatbot yang dapat membantu pengguna mobil mendapatkan informasi mengenai mobilnya.

Chatbots sangat sukses di industri perbankan dan keuangan. Perbankan adalah salah satu sektor pertama yang bereksperimen dengan antarmuka percakapan. Chatbot perbankan dapat menjawab pertanyaan umum tentang produk keuangan, perbankan yang aman, dan sebagainya, serta memberikan informasi spesifik dan personal tentang akun pengguna. Banyak bank global dan penyedia layanan keuangan termasuk Bank of America, ICICI bank, HSBC, Royal Bank of Scotland, Capital One, Mastercard, dan sebagainya telah menerapkan chatbots untuk membantu pelanggan mereka. Banyak perusahaan fintech yang membangun chatbot yang dapat bertindak sebagai asisten keuangan bagi pengguna. Ernest.ai dan Cleo adalah chatbot yang dapat terhubung ke rekening bank Anda dan berbicara dengan Anda tentang pengeluaran, saldo, dan juga memberikan tips untuk menghemat uang. Chatbots juga banyak digunakan di sektor asuransi, di mana mereka bertindak sebagai asisten yang dapat memberi Anda penawaran harga yang disesuaikan (misalnya, SPIXII).

Chatbots juga digunakan di sektor hukum, kesehatan, pemerintahan, dan sektor ketiga. Chatbot bernama DoNotPay telah membantu orang menantang tiket parkir di London dan New York dalam lebih dari 160.000 kasus. Setelah ini, lebih banyak chatbots telah dikembangkan untuk membantu masyarakat mengakses keadilan dan layanan hukum: penilaian kejahatan (LawBot), penggabungan bisnis (LawDroid), bantuan penyewa (RentersUnion), bantuan dengan pertanyaan dan dokumentasi hukum (Lisa, LegaliBot, Lexi, DocuBot), dan temukan pengacara (BillyBot).

Di sektor ketiga, chatbots telah digunakan untuk menyebarkan kesadaran akan isu-isu yang menjadi perhatian badan amal. Stoptober adalah chatbot Facebook yang dikembangkan oleh National Health Services (NHS) di Inggris untuk membantu perokok berhenti. Chatbot lainnya, Yeshi, dikembangkan untuk menarik kesadaran akan krisis air di Ethiopia. Chatbots juga mulai memasuki layanan kesehatan. Chatbots seperti Your.MD dan HealthTap dirancang untuk mendiagnosis masalah kesehatan berdasarkan gejala. Emily adalah chatbot yang dirancang oleh LifeFolder untuk membantu membuat keputusan akhir hidup (misalnya, dokumentasi hukum, dukungan hidup, donasi organ, dan banyak lagi).

Chatbots tidak hanya digunakan untuk menghadapi pelanggan tetapi juga secara internal, untuk menghadapi karyawan. Chatbots, dalam arti tertentu, menjadi rekan kerja dengan membantu sesama karyawan melakukan tugas-tugas yang berulang, biasa-biasa saja, dan membosankan. Layanan perpesanan seperti Slack dan Microsoft Teams telah mendorong chatbot di platform mereka untuk mengotomatiskan komunikasi kantor. Bot ini bertujuan untuk melibatkan rekan kerja dalam obrolan tentang tugas-tugas yang menyenangkan dan penting. Misalnya, terdapat bot untuk membantu rekan kerja berbagi pengetahuan (Obie.ai), mengakses layanan lain seperti GDrive (WorkBot), mengatur pertemuan (Meekan), mendiskusikan makan siang (LunchTrain), dan bahkan membantu pengambilan keputusan (ConcludeBot, SimplePoll).

Jika Anda tertarik untuk mengetahui lebih banyak kasus penggunaan, saya sarankan Anda melihat beberapa layanan direktori bot seperti botlist.co dan www.chatbots.org, di mana Anda dapat menemukan lebih banyak informasi dan inspirasi.

Perangkat pengembang

Selama beberapa tahun terakhir, ekosistem alat dan layanan telah berkembang berdasarkan gagasan antarmuka percakapan. Ada sejumlah alat yang dapat kita pasang dan mainkan untuk merancang, mengembangkan, dan mengelola chatbots.

Alat maket

Maket dapat digunakan untuk menunjukkan kepada klien tentang tampilan dan perilaku chatbot. Ini adalah alat yang mungkin ingin Anda pertimbangkan untuk digunakan selama desain percakapan, setelah memberikan contoh percakapan antara pengguna dan bot di balik serbet. Alat mockup memungkinkan Anda memvisualisasikan percakapan antara pengguna dan bot dan menampilkan dinamika pergantian percakapan. [BotSociety.io](https://botsociety.io/) (<https://botsociety.io/>) dan [BotMock.com](https://botmock.com/) (<https://botmock.com/>) adalah beberapa alat mockup yang populer. Beberapa alat ini memungkinkan Anda mengeksport desain mockup dan membuat video.

Saluran

Saluran mengacu pada tempat di mana pengguna dapat berinteraksi dengan chatbot. Ada beberapa saluran penerapan di mana bot Anda dapat diekspos ke pengguna. Ini termasuk layanan perpesanan seperti Facebook Messenger, Skype, Kik, Telegram, WeChat, dan Line; layanan obrolan kantor dan tim seperti Slack, Microsoft Teams, dan banyak lagi; saluran tradisional seperti web chat, SMS, dan panggilan suara; dan speaker pintar seperti Amazon Echo dan Google Home. Pilih saluran berdasarkan pengguna Anda dan persyaratan proyek. Misalnya, jika Anda membangun chatbot yang menasar konsumen, Facebook Messenger bisa menjadi saluran terbaik karena semakin banyak pengguna yang menggunakan layanan ini untuk tetap berhubungan dengan teman dan keluarga. Menambahkan chatbot Anda ke daftar kontak mereka mungkin lebih mudah daripada meminta mereka mengunduh aplikasi Anda. Jika pengguna perlu berinteraksi dengan bot menggunakan suara di lingkungan rumah atau kantor, saluran speaker pintar bisa menjadi pilihan ideal. Dan terakhir, ada tools yang dapat menghubungkan chatbot ke banyak saluran secara bersamaan (misalnya integrasi Dialogflow, MS Bot Service, dan Smooch.io, dan sebagainya).

1.4 Alat Pengembangan Chatbot

Ada banyak alat yang dapat Anda gunakan untuk membuat chatbot tanpa harus membuat kode satu baris pun: Chatfuel, ManyChat, Dialogflow, dan sebagainya. Chatfuel memungkinkan desainer menciptakan alur percakapan menggunakan elemen visual. Dengan ManyChat, Anda dapat membangun alur menggunakan peta visual yang disebut FlowBuilder. Elemen percakapan seperti ucapan bot dan tombol respons pengguna dapat dikonfigurasi menggunakan elemen UI seret dan lepas. Dialogflow dapat digunakan untuk membangun chatbot yang memerlukan pemahaman bahasa alami tingkat lanjut untuk berinteraksi dengan pengguna.

Di sisi lain, ada bahasa scripting seperti Artificial Intelligence Markup Language (AIML), ChatScript, dan RiveScript yang dapat digunakan untuk membangun chatbots. Skrip ini akan berisi konten dan alur percakapan yang kemudian perlu dimasukkan ke dalam program penerjemah atau mesin aturan untuk menghidupkan chatbot. Penerjemah memutuskan bagaimana melanjutkan percakapan dengan mencocokkan ucapan pengguna dengan templat dalam skrip. Meskipun mudah untuk membangun chatbot percakapan menggunakan pendekatan ini, akan sulit untuk membangun chatbot transaksional tanpa menghasilkan representasi semantik eksplisit dari ucapan pengguna. PandoraBots adalah platform berbasis web populer untuk membangun chatbot AIML.

Alternatifnya, terdapat pustaka SDK yang dapat digunakan untuk membuat chatbot: MS Bot Builder, BotKit, BotFuel, dan sebagainya menyediakan SDK dalam satu atau lebih bahasa pemrograman untuk membantu pengembang dalam membangun modul manajemen percakapan inti. Kemampuan mengkodekan manajer percakapan memberi pengembang fleksibilitas untuk membentuk percakapan dan mengintegrasikan bot ke tugas-tugas backend lebih baik daripada platform tanpa kode dan skrip. Setelah dibangun, pengelola percakapan kemudian dapat dihubungkan ke layanan lain seperti pemahaman bahasa alami untuk memahami ucapan pengguna.

Analisis

Seperti solusi digital lainnya, chatbots mendapatkan keuntungan dari pengumpulan dan analisis statistik penggunaannya. Meskipun Anda dapat membangun platform analitik khusus dari awal, Anda juga dapat menggunakan perangkat siap pakai yang tersedia secara luas saat ini. Tersedia banyak perangkat analisis siap pakai yang dapat dihubungkan ke chatbot, yang dapat digunakan untuk mencatat dan memeriksa pesan masuk dan keluar. Alat-alat ini memberi tahu pembuat dan pengelola chatbot jenis percakapan yang sebenarnya terjadi antara pengguna dan chatbot. Data tersebut akan memberikan informasi berguna seperti tugas percakapan yang populer, tempat di mana pengalaman percakapan terganggu, ucapan yang tidak dipahami oleh bot, dan permintaan yang masih perlu ditingkatkan oleh chatbot. Dashbot.io, BotAnalytics, dan Google's Chatbase adalah beberapa perangkat analitik yang dapat Anda gunakan untuk menganalisis kinerja chatbot Anda.

Pemahaman Bahasa Alami

Chatbots dapat dibangun tanpa harus memahami ucapan pengguna. Namun, menambahkan kemampuan pemahaman bahasa alami tidaklah terlalu sulit. Ini adalah salah satu fitur khas yang membedakan chatbots dari rekan-rekan digitalnya seperti situs web dan aplikasi dengan elemen visual. Ada banyak modul pemahaman bahasa alami yang tersedia sebagai layanan cloud. Pemain TI besar seperti Google, Microsoft, Facebook, dan IBM telah menciptakan alat yang dapat Anda sambungkan ke chatbot Anda. Dialogflow Google, Microsoft LUIS, IBM Watson, SoundHound, dan Wit.ai Facebook adalah beberapa alat NLU yang dapat Anda coba. Kita akan menjelajahi Dialogflow (sebelumnya disebut Api.AI) di beberapa bab.

1.5 Layanan Direktori

Salah satu tantangan dalam membangun bot adalah membuat pengguna menemukan dan menggunakannya. Chatbots tidak sepopuler situs web dan aplikasi seluler, sehingga calon pengguna mungkin tidak tahu di mana harus mencari bot tersebut. Setelah chatbot Anda diterapkan, Anda perlu membantu pengguna menemukannya. Ada direktori yang mencantumkan bot dalam berbagai kategori. Chatbots.org adalah salah satu layanan direktori tertua yang telah mencantumkan chatbots dan asisten virtual sejak tahun 2008. Yang populer lainnya adalah Botlist.co, BotPages, BotFinder, dan ChatBottle. Direktori ini mengkategorikan bot berdasarkan tujuan, sektor, bahasa yang didukung, negara, dan sebagainya. Selain itu, saluran seperti Facebook dan Telegram memiliki direktori sendiri untuk bot yang dihosting di saluran mereka. Dalam kasus Facebook, Anda dapat membantu pengguna menemukan bot Messenger Anda menggunakan layanan Discover mereka.

1.6 Monetisasi

Chatbots dibuat untuk berbagai tujuan: untuk menciptakan kesadaran, untuk mendukung pelanggan setelah penjualan, untuk menyediakan layanan berbayar, dan banyak lagi. Selain semua ini, chatbots dengan konten menarik dapat melibatkan pengguna untuk waktu yang lama dan dapat digunakan untuk menghasilkan uang melalui iklan hasil personalisasi yang ditargetkan. Layanan seperti CashBot.ai dan AddyBot.com dapat berintegrasi dengan chatbot Anda untuk mengirimkan iklan dan rekomendasi yang ditargetkan kepada pengguna, dan ketika pengguna terlibat, chatbot Anda menghasilkan uang.

Hal di atas bukanlah daftar alat yang lengkap dan layanan juga tidak tercantum di bawah masing-masing jenis. Alat-alat ini berkembang seiring waktu seiring chatbot menemukan ceruk pasarnya. Daftar ini memberi Anda gambaran tentang betapa multidimensi ekosistem dan membantu Anda menjelajahi ruang dan memenuhi pikiran kreatif Anda.

1.7 Manfaat chatbot

Antarmuka pengguna percakapan menghadirkan yang terbaik dari kedua dunia: interaksi alami seperti manusia dikombinasikan dengan manfaat teknologi digital.

- a) Ketersediaan: Seperti teknologi digital otomatis lainnya, antarmuka percakapan berbiaya rendah dan tersedia 24/7. Ini seperti memiliki seseorang yang menjaga meja obrolan web sepanjang waktu sehingga pelanggan selalu memiliki seseorang untuk mendapatkan jawaban.
- b) Pengalaman yang dipersonalisasi: Tidak seperti situs web dan aplikasi ponsel cerdas, chatbots dapat memberikan pengalaman yang sangat personal karena sifat interaksinya yang bersifat percakapan. Pengaturan percakapan satu lawan satu memberikan banyak kesempatan untuk memahami dan beradaptasi dengan tujuan, preferensi, dan batasan pengguna.

- c) Biaya rendah: Chatbots adalah solusi digital dan oleh karena itu menyediakan layanan dukungan pelanggan setidaknya sepuluh kali lebih murah daripada manusia yang melakukan tugas yang sama.
- d) Konsistensi: Chatbots bisa konsisten dalam layanan, yang mungkin sulit dicapai oleh operator manusia dan mungkin sangat penting di sektor tertentu. Waktu respons yang cepat: Tidak seperti sistem berbasis manusia, waktu respons untuk chatbots jauh lebih cepat. Pengguna tidak lagi harus menunggu panggilan mereka diangkat dan selama percakapan, respons chatbot akan lebih cepat daripada respons manusia, terutama ketika operator manusia ditugaskan untuk melakukan lebih dari satu obrolan secara bersamaan (terkadang hingga lima obrolan). Kemampuan chatbots untuk menangani percakapan simultan juga menghilangkan hambatan bandwidth dukungan pelanggan yang terbatas dan oleh karena itu membantu meningkatkan skala bisnis.
- e) Peningkatan skala: Chatbots dapat dengan mudah ditingkatkan untuk menangani lalu lintas yang meningkat dan musiman, yang tidak mudah dilakukan jika menggunakan penasihat langsung. Musim liburan khususnya dapat meningkatkan permintaan akan dukungan pelanggan. Pada saat seperti itu, chatbots dapat digunakan untuk menangani tugas-tugas berprioritas rendah dan mudah, sehingga mengurangi beban penasihat langsung dan bantuan manusia dapat digunakan secara bijaksana untuk menangani percakapan bernilai tinggi.

1.8 Chatbot Akan Tetap Ada

Teknologi antarmuka pengguna percakapan saat ini menjadi salah satu topik tren teratas dalam bisnis teknologi. Sebagian besar merek besar sudah mulai merumuskan strategi chatbot mereka dalam strategi AI dan otomatisasi yang lebih besar. Inovasi seperti chatbot, speaker pintar, dan mobil tanpa pengemudi mendorong pengambilan kebijakan besar tersebut. Dunia bersiap menghadapi serangan teknologi otomatisasi yang siap menggantikan manusia dalam tugas yang berulang dan terstruktur.

Munculnya chatbot baru-baru ini dipicu oleh banyak faktor:

- Generasi milenial terus beralih ke chat sebagai saluran pilihan mereka untuk berinteraksi dengan merek. Survei kontak pelanggan menunjukkan bahwa orang ingin menggunakan saluran obrolan web bila tersedia, dibandingkan saluran tradisional lainnya, seperti email dan telepon, untuk menghubungi bisnis.
- Pertumbuhan aplikasi pemesanan chat di ponsel pintar dan perangkat lain telah melampaui penggunaan aplikasi media sosial seperti Facebook dan Twitter. Kini orang-orang menghabiskan lebih banyak waktu pada aplikasi pemesanan, mengobrol dengan teman, keluarga, kolega, dan bahkan bisnis.
- Meningkatnya permintaan pelanggan pada obrolan memberikan tekanan yang luar biasa pada merek. Kurangnya sumber daya manusia yang terampil untuk menangani peningkatan lalu lintas obrolan juga merupakan kontributor penting munculnya chatbots.

- Ketersediaan alat layanan kognitif untuk pemahaman bahasa alami, pengenalan ucapan, sintesis ucapan, manajemen percakapan, analitik, dan sebagainya telah menjadikan desain dan pengembangan chatbot lebih mudah dibandingkan beberapa tahun yang lalu.
- Terbukanya saluran perpesanan dan cara baru yang inovatif, seperti speaker pintar, telah menjadikan penyampaian layanan melalui chatbots menjadi kenyataan. Fakta bahwa terdapat peningkatan minat terhadap aplikasi dan perangkat perpesanan seperti smart speaker menghadirkan peluang menarik bagi merek untuk membangun chatbot guna memanfaatkan pengguna yang sudah tersedia di saluran ini.

Ada beberapa survei dan statistik yang menunjukkan bahwa antarmuka percakapan akan tetap ada. Melalui daftar berikut, kami menawarkan beberapa temuan survei dan prediksi paling menarik yang menunjukkan bahwa chatbots akan ada dalam jangka panjang:

- Gartner(<https://www.gartner.com/smarterwithgartner/gartnerstrategicpredictions-for-2018-and-beyond/>) memperkirakan bahwa pada tahun 2021, merek yang mendesain situs webnya dengan menyertakan penelusuran suara dan visual akan meningkatkan pendapatan sebesar 30% dan lebih dari 50% bisnis akan menghabiskan lebih banyak uang untuk chatbots dibandingkan aplikasi seluler tradisional.
- Dalam survei Oracle, 80% responden (eksekutif tingkat C) mengatakan bahwa mereka berencana memperkenalkan layanan chatbot pada tahun 2020, jika belum [ATAU].
- Penelitian Juniper memperkirakan bahwa penggunaan chatbot akan menghasilkan penghematan biaya tahunan sebesar USD 8 miliar pada tahun 2022, naik dari USD 20 juta pada tahun 2017 [JR].
- Survei Hubspot menemukan bahwa sekitar 47% konsumen terbuka untuk membeli barang melalui chatbot dan sekitar 40% tidak peduli apakah mereka berbicara dengan chatbot atau manusia selama mereka mendapatkan bantuan dengan mudah dan cepat [HB].
- Terakhir, menurut laporan Grand View Research baru-baru ini, pasar chatbot global diperkirakan akan mencapai angka USD 1,25 miliar pada tahun 2025, dengan pertumbuhan CAGR sebesar 24,3% dari USD 190 juta pada tahun 2016 [GVR].

Jadi, apakah Anda siap untuk memulai dan membuat beberapa chatbot? Saya harap saya telah memberi Anda pengenalan yang baik tentang dunia chatbots di bab ini. Kami membahas sejarah dan perkembangan terkini, klasifikasi chatbot, penerapannya di berbagai sektor, manfaatnya, masa depannya, dan arsitektur dasarnya. Selama delapan bab berikutnya, saya akan memperkenalkan Anda pada beberapa alat, teknik, dan konsep yang akan memungkinkan Anda membangun antarmuka percakapan yang menakjubkan. Biarkan perjalanan dimulai!

BAB 2

CHATBOT TERCIPTA

Saat Anda mulai membuat chatbot, sangat penting untuk memahami apa fungsi chatbot dan tampilannya. Anda pasti pernah mendengar tentang Siri, IBM Watson, Google Allo, dll. Masalah dasar yang coba dipecahkan oleh bot ini adalah menjadi perantara dan membantu pengguna menjadi lebih produktif. Mereka melakukan ini dengan memungkinkan pengguna untuk tidak terlalu khawatir tentang bagaimana informasi akan diambil dan tentang format input yang mungkin diperlukan untuk mendapatkan data tertentu. Bot cenderung menjadi semakin cerdas saat mereka menangani masukan data pengguna dan mendapatkan lebih banyak wawasan darinya. Chatbots sukses karena memberikan apa yang Anda inginkan.

Apakah Anda kesal atau frustrasi ketika Anda harus memasukkan nama, ID email, alamat, dan kode pin yang sama setiap kali di situs web yang berbeda? Bayangkan sebuah bot yang melakukan tugas Anda misalnya, memesan makanan dari vendor yang berbeda, berbelanja online dari berbagai perusahaan e-commerce, atau memesan tiket penerbangan atau kereta api dan Anda tidak perlu memberikan ID email yang sama, pengiriman alamat, atau informasi pembayaran setiap saat.

Chatbot adalah salah satu metode AI yang penggunaan dan aplikasinya semakin berkembang. Asisten virtual seperti Amazon Alexa dan Google Assistant, serta layanan perpesanan seperti WeChat dan Facebook Messenger adalah beberapa contoh teknologi chatbot. Chatbot bisa menyederhanakan proses belanja dan pembayaran. Anda bisa menggunakan bot untuk mengirimkan katalog produk yang pelanggan butuhkan sesuai permintaan tertentu. Termasuk mengirimkan invoice yang memudahkan pelanggan membayar.

Bot sudah memiliki kemampuan untuk mengetahui informasi ini dan cukup cerdas untuk mengambil apa yang diperlukan ketika Anda menanyakannya dalam bahasa Anda sendiri atau dalam apa yang dalam ilmu komputer dikenal sebagai Natural Language.

Pengembangan chatbots jauh lebih mudah dibandingkan beberapa tahun yang lalu, namun chatbots juga ada beberapa dekade yang lalu; namun, popularitas chatbots telah meningkat secara eksponensial dalam beberapa tahun terakhir.

Jika Anda adalah orang teknis atau memiliki gambaran tentang cara kerja aplikasi web atau aplikasi seluler, Anda pasti pernah mendengar istilah API. Segala jenis data yang Anda perlukan saat ini tersedia untuk digunakan dalam bentuk API yang disediakan oleh berbagai penyedia layanan dan institusi. Jika Anda mencari informasi cuaca, memesan tiket, memesan makanan, mendapatkan informasi penerbangan, mengubah satu bahasa ke bahasa lain, atau memposting di Facebook atau Twitter, semua ini dapat dilakukan dengan menggunakan API. API ini digunakan oleh aplikasi berbasis web atau seluler untuk melakukan tugas-tugas ini. Chatbots juga dapat menggunakan API ini untuk mencapai tugas yang sama berdasarkan permintaan kami.

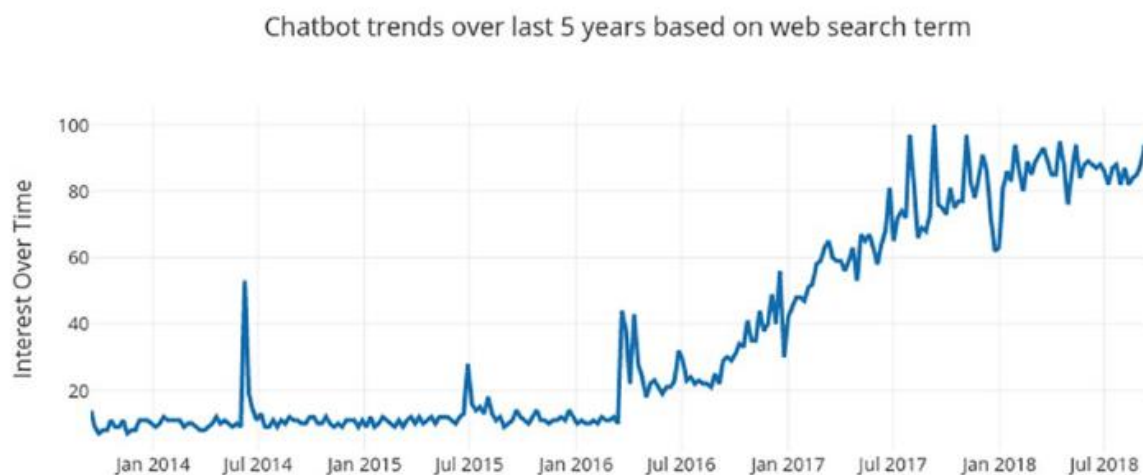
Alasan Chatbots lebih unggul dibandingkan metode tradisional dalam menyelesaikan sesuatu secara online adalah Anda dapat melakukan banyak hal dengan bantuan chatbot. Ini bukan sekedar chatbot, ini seperti asisten pribadi virtual Anda. Anda dapat membayangkan memesan kamar hotel di booking.com serta memesan meja di restoran terdekat di hotel tersebut, tetapi Anda dapat melakukannya menggunakan chatbot Anda. Chatbots memenuhi kebutuhan akan multiguna dan karenanya menghemat banyak waktu dan uang.

Dalam buku ini kita akan belajar bagaimana membangun pengalaman percakapan alami menggunakan bot dan bagaimana mengajarkan bot untuk memahami bahasa alami kita dan membuatnya melakukan tugas untuk kita dari satu antarmuka.

Bot secara umum tidak lain hanyalah sebuah mesin yang cukup cerdas untuk memahami permintaan Anda dan kemudian merumuskan permintaan Anda sedemikian rupa sehingga dapat dimengerti oleh sistem perangkat lunak lain untuk meminta data yang Anda butuhkan.

2.1 Popularitas Penggunaan Chatbots

Chatbots telah menjadi populer seperti apa pun di masa lalu. Mari kita coba melihat Gambar 1-1, yang menggambarkan kebangkitan chatbots, dan juga mencoba memahami mengapa ada permintaan yang besar untuk membangun chatbots.



Gambar 2.1. Angka pada sumbu Y mewakili minat penelusuran relatif terhadap titik tertinggi pada grafik di semua kategori di seluruh dunia

Jawaban sederhana yang terlintas dalam pikiran adalah bahwa ini bukanlah perangkat lunak yang rumit dan dapat digunakan oleh siapa saja. Saat kami membangun perangkat lunak, kami menargetkan audiens yang akan menggunakannya, namun ketika digunakan oleh orang lain, perangkat lunak tersebut menjadi sulit dan tidak dapat digunakan. Saat kami membuat chatbot, kami ingat bahwa chatbot akan digunakan oleh orang-orang dari semua kelompok umur. Hal ini hanya terjadi pada chatbots, di mana perangkat lunaknya mencoba berperilaku seperti orang bodoh (namun cerdas) dan

membiarkan penggunanya menjadi dirinya sendiri. Di semua perangkat lunak lainnya, Anda akan menyadari bahwa Anda harus mengetahui beberapa terminologi atau secara bertahap mengetahui cara memanfaatkannya secara optimal, tetapi tidak demikian halnya dengan chatbots. Jika Anda tahu cara berbicara dengan seseorang, Anda tidak akan mengalami masalah apa pun dalam menggunakan chatbot.

Permintaan terhadap chatbot terus meningkat. Namun, belum banyak penelitian yang secara empiris mencoba mencari tahu motivasi di balik penggunaan chatbots. Dalam penelitian baru-baru ini, kuesioner online menanyakan pengguna chatbot berusia 16 hingga 55 tahun di AS untuk menjelaskan kebutuhan mereka menggunakan chatbot dalam kehidupan sehari-hari. Survei tersebut mengungkapkan “produktivitas” menjadi faktor motivasi utama dalam menggunakan chatbots.

Zen dari Python dan Mengapa Ini Diterapkan pada Chatbots?

Saya ingat Zen dari Python, yang mengatakan, “Sederhana lebih baik daripada Kompleks,” dan itu berlaku di banyak tempat dalam perangkat lunak.

Zen of Python adalah kumpulan 20 prinsip perangkat lunak yang mempengaruhi desain Bahasa Pemrograman Python.

—Tim Peters

Ingin tahu “Apa itu Zen dari Python?” Coba langkah-langkah di bawah ini.

Jika Anda sudah menginstal Python di komputer Anda. Buka saja juru bahasa Python Anda dan impor ini:

```
Python 2.7.15 (default, May 1 2018, 16:44:08)
[GCC 4.2.1 Compatible Apple LLVM 9.1.0 (clang-902.0.39.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.

Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one—and preferably only one—obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
```

Now is better than never.
 Although never is often better than *right* now.
 If the implementation is hard to explain, it's a bad idea.
 If the implementation is easy to explain, it may be a good idea.
 Namespaces are one honking great idea—let's do more of those!

Anda mungkin tidak dapat memahami semua poin di atas terkait dengan chatbot, tetapi Anda pasti dapat memahami sebagian besar poin tersebut.

Nah, kembali ke topik kita, saya ingat menemukan kesulitan untuk mulai menggunakan Antarmuka Pengguna Facebook saat menggunakan latar belakang Orkut. Jika Anda belum pernah menggunakan Orkut, Anda tidak akan memahaminya, namun coba pikirkan situasi dalam hidup Anda saat Anda mulai menggunakan beberapa perangkat lunak atau aplikasi baru dan Anda kesulitan memahaminya. Mungkin beralih dari Windows ke MacOS/Linux atau sebaliknya? Saat Anda menggunakan aplikasi baru, Anda perlu mempelajari beberapa hal, dan perlu waktu untuk membiasakannya serta mengetahui fungsi dan cara kerjanya. Terkadang Anda mengetahui beberapa fitur aplikasi bahkan setelah bertahun-tahun menggunakannya. Jika Anda menggunakan MacOS, coba Shift + Option + Volume Naik/Turun dan lihat apa yang terjadi. Beri tahu saya jika ini membuat Anda takjub, jika Anda belum mengetahuinya.

Dalam kasus chatbots, komunikasi antara pengguna dan server atau sistem backend cukup sederhana. Ini seperti berbicara dengan orang lain menggunakan aplikasi perpesanan.

Anda cukup mengetik apa yang Anda inginkan, dan bot akan dapat memberikan apa yang Anda inginkan atau memandu Anda cara mendapatkannya. Dengan kata lain, ini akan mengarahkan Anda ke informasi yang benar dengan memberi Anda tautan atau dokumen. Waktunya telah tiba di mana bot bahkan dapat menggali informasi dari sebuah artikel dan dokumen dan memberikannya kepada pengguna.

Kemajuan signifikan dalam AI yang dilakukan oleh perusahaan seperti Google, Facebook, dan IBM serta oleh layanan pembelajaran mesin seperti Amazon Lex, wit.ai, api.ai, luis.ai, IBM Watson, Amazon Echo, dll. telah menghasilkan pertumbuhan dan permintaan yang luar biasa robot seperti itu.

2.2 Kebutuhan Chatbot

Sekarang, kami akan mencoba melihat kebutuhan dan permintaan chatbot di era penciptaan dan pengambilan informasi yang berkembang pesat ini dari dua perspektif berbeda: sudut pandang bisnis dan sudut pandang pengembang. Jadi, jika Anda seorang manajer produk, manajer penjualan, atau dari bagian pemasaran atau domain terkait apa pun yang menggerakkan bisnis secara langsung, maka Anda tidak boleh melewatkan perspektif bisnis dari chatbots. Ini akan memberi Anda gambaran jelas bahwa bisnis saat ini perlu mengadopsi teknologi ini untuk mendorong lebih banyak pendapatan.

2.3 Perspektif Bisnis

Kami akan mencoba melihat perspektif bisnis dari chatbots. Apakah baik bagi bisnis untuk memiliki chatbot atau memigrasikan banyak hal untuk dilakukan oleh chatbots?

Waktunya telah tiba bagi bisnis untuk memperlakukan chatbots sebagai salah satu alat pemasaran generasi ini.

- *Aksesibilitas*: Mudah diakses. Konsumen dapat membuka situs web dan mulai mengajukan pertanyaan atau mulai menyelesaikan pertanyaan mereka tanpa harus menghubungi nomor dan mengikuti cara buruk “Tekan 1 untuk ini dan Tekan 2 untuk itu” di IVR. Mereka dapat dengan cepat menyampaikan maksudnya hanya dengan serangkaian informasi dasar.
- *Efisiensi*: Pelanggan dapat duduk di meja mereka di kantor atau di sofa di ruang tamu sambil menonton pertandingan dan mendapatkan status permohonan kartu kredit, mengetahui status pesanan makanan, atau mengajukan keluhan tentang masalah apa pun. Jika Anda menjadikan pelanggan efisien dan produktif, mereka mulai mencintai Anda. Bot melakukan hal itu dan membantu meningkatkan bisnis.
- *Ketersediaan*: Chatbots tersedia 24 jam per hari, 7 hari per minggu. Mereka tidak akan pernah meminta cuti atau merasa lelah seperti karyawan manusia. Mereka akan melakukan tugas yang sama atau tugas baru setiap saat dengan efisiensi dan kinerja yang sama. Anda pasti merasa frustrasi ketika beberapa nomor telepon layanan pelanggan mengatakan, “Silakan hubungi kami antara jam 9:00 dan 18:00,” hanya untuk sekedar informasi. Bot Anda tidak akan pernah mengatakan ini.
- *Skalabilitas*: Satu Bot => 1 juta karyawan. Anda melihat ini? Ya, jika bot Anda dapat melakukan apa yang dibutuhkan pelanggan, bot tersebut dapat dengan mudah menangani ratusan ribu pertanyaan pelanggan secara bersamaan tanpa bersusah payah. Anda tidak perlu membuat pelanggan Anda menunggu dalam antrian sampai perwakilan pelanggan bebas.
- *Biaya*: Tentu saja hal ini menghemat banyak biaya bagi bisnis. Siapa yang tidak suka menghemat uang? Ketika bot melakukan itu untuk Anda, tidak ada alasan mengapa Anda tidak menyukainya.
- *Wawasan*: Perwakilan penjualan Anda mungkin tidak dapat mengingat perilaku pengguna dan memberi Anda wawasan eksklusif tentang pola perilaku konsumen, namun bot Anda dapat menggunakan teknik pembelajaran mesin dan ilmu data terbaru.

2.4 Chatbots Mendatangkan Pendapatan

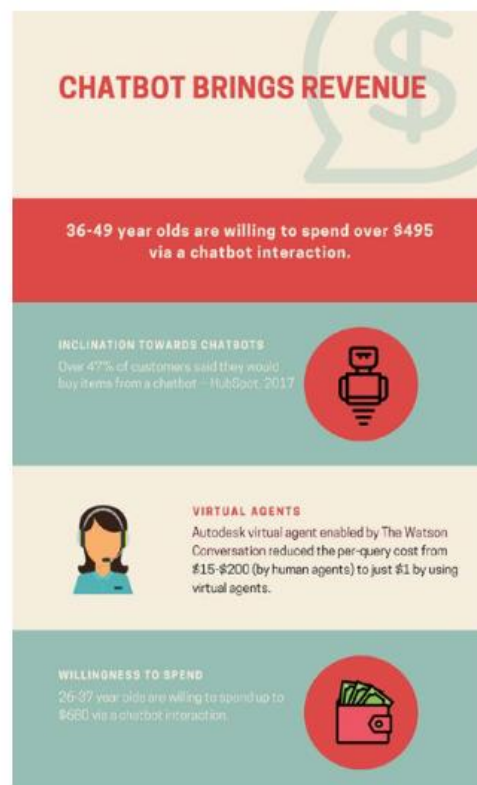
Chatbots telah terbukti berhasil mendatangkan lebih banyak pendapatan bagi bisnis. Bisnis yang dimulai dengan dukungan chatbot atau membuat chatbot baru untuk mendukung pertanyaan pelanggan memiliki kinerja yang baik di pasar dibandingkan dengan pesaing mereka.

Berdasarkan salah satu postingan blog di stanfy.com, dalam 2 bulan pertama setelah memperkenalkan chatbot Facebook, 1-800-Flowers.com melaporkan bahwa lebih dari 70 persen pesanan Messenger berasal dari pelanggan baru. Pelanggan baru ini umumnya juga berusia lebih muda dibandingkan pembeli pada umumnya karena mereka sudah familiar

dengan aplikasi Facebook Messenger. Hal ini secara signifikan meningkatkan pendapatan tahunan mereka.

Salah satu nilai tambah terbesar dari chatbots adalah menggunakannya untuk menghasilkan prospek. Anda dapat menjangkau klien potensial Anda secara langsung di tempat yang menjadi perhatian mereka (utusan) dan menyajikan kepada mereka produk, layanan, atau barang terbaru Anda. Ketika pelanggan ingin membeli suatu produk/jasa, ia dapat melakukan pembelian melalui chatbot, termasuk proses pembayarannya. Bot, seperti 1-800flowers.com, eBay, dan Fynd telah membuktikannya.

—Julien Blancher, Salah Satu Pendiri @ Recast.AI



Gambar 2.2. Chatbot mendatangkan pendapatan

Dalam sebuah artikel oleh Stefan Kojouharov, pendiri ChatbotsLife, dia menyebutkan bagaimana berbagai perusahaan menghasilkan lebih banyak uang daripada tanpa chatbots. Dia berkata, Ruang e-commerce telah mulai menggunakan chatbots dalam sejumlah cara yang dengan cepat menambah keuntungan mereka. Mari kita lihat kisah sukses awalnya:

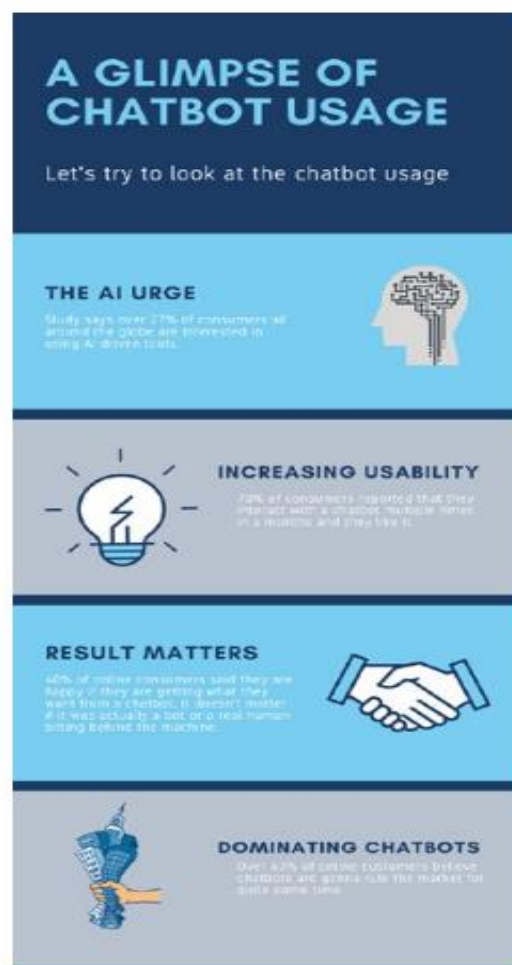
- ✚ 1-800-Bunga: melaporkan bahwa lebih dari 70 persen pesanan Messenger berasal dari pelanggan baru!
- ✚ Sephora: meningkatkan janji temu perubahan sebesar 11 persen melalui chatbot Facebook Messenger.
- ✚ Nitro Café: meningkatkan penjualan sebesar 20 persen dengan chatbot Messenger mereka, yang dirancang untuk kemudahan pemesanan, pembayaran langsung, dan komunikasi dua arah instan.

- ✚ Sun's Soccer: chatbot mendorong hampir 50 persen penggunanya kembali ke situs mereka di seluruh liputan sepak bola tertentu; 43 persen pelanggan chatbot mengklik selama periode terbaiknya.
- ✚ Asos: meningkatkan pesanan sebesar 300 persen menggunakan chatbot Messenger dan mendapatkan laba atas pembelanjaan sebesar 250 persen sekaligus menjangkau 3,5 kali lebih banyak orang.

Gambar 2-2 mencoba memberi Anda gambaran mengapa ada korelasi langsung antara chatbots dan pendapatan. Mari kita lihat Gambar 2-2 untuk mendapatkan gambaran tentang hal itu.

2.5 Sekilas Penggunaan Chatbot

Kami akan mencoba melihat betapa bermanfaatnya chatbot bagi konsumen karena kegunaannya dan efisiensi yang diberikannya. Semua orang di era TI yang dinamis ini ingin cepat dalam segala hal, dan menggunakan chatbot membuat pekerjaan Anda lebih mudah dan cepat setiap hari. Itu dipersonalisasi sedemikian rupa agar tidak mengulangi hal-hal yang sudah jelas; hal ini membuat kami memikirkan kembali penggunaan perangkat lunak secara tradisional. Gambar 2-3 memberikan ilustrasi yang akan memberi Anda gambaran yang adil tentang penggunaan chatbot.



Gambar 2.3. Sekilas tentang penggunaan Chatbot oleh konsumen

2.6 Pelanggan Lebih Memilih Chatbots

Chatbots bukan sekedar software di era modern. Chatbots seperti asisten pribadi kita yang memahami kita dan dapat dikonfigurasi secara mikro. Mereka mengingat suka dan tidak suka kita dan tidak pernah mengecewakan kita dengan melupakan apa yang telah kita ajarkan kepada mereka, dan inilah alasan mengapa semua orang menyukai chatbot. Lain kali Anda bertemu seseorang atau bertemu pelanggan Anda, jangan lupa bertanya apakah mereka lebih suka perangkat lunak konvensional atau chatbots mutakhir. Mari kita lihat Gambar 2-4 untuk memahami alasan mengapa pelanggan lebih memilih chatbots dibandingkan dengan sistem perangkat lunak lain untuk interaksi manusia dan komputer.



Gambar 2.4. Pelanggan lebih memilih chatbot

Di bagian selanjutnya dari bab ini kita akan membahas mengapa chatbots adalah hal besar berikutnya bagi pengembang pemula. Baik Anda seorang pengembang baru, pengembang tingkat menengah, atau UKM berpengalaman, Anda harus memahami apa yang tersedia bagi pengembang saat membangun chatbots.

Perspektif Pengembang

Pernahkah Anda merasakan sakitnya ketika harus memperbarui OS komputer atau ponsel atau aplikasi lain yang mungkin Anda gunakan untuk menggunakan fitur-fitur baru? Bagaimana jika aplikasi tidak perlu diperbarui setiap saat untuk menggunakan fitur baru? Atau katakanlah, alih-alih memiliki banyak aplikasi, seseorang dapat memiliki satu aplikasi yang melakukan sebagian besar hal yang saat ini dilakukan oleh banyak aplikasi?

Bot untuk pengembang menyenangkan untuk dibuat. Ini seperti mengajari anak Anda berjalan, berbicara, berperilaku, dan melakukan sesuatu. Anda suka menjadikannya lebih

cerdas dan mandiri. Dari sudut pandang pengembang, chatbots adalah topik yang sangat penting untuk diketahui.

2.7 Rilis Fitur dan Perbaikan Bug

Banyak fitur yang dapat ditambahkan ke chatbot tanpa mengharuskan pengguna memperbarui aplikasi chatbot Anda. Mungkin akan merepotkan jika Anda merilis versi aplikasi dengan beberapa bug, dan Anda harus memperbaikinya dan merilisnya lagi di AppStore untuk mendapatkan persetujuan, dan, yang paling penting, pengguna harus memperbarui aplikasi tersebut. Jika mereka tidak memperbarui, pelanggan akan terus mengeluh tentang masalah tersebut, yang mengakibatkan hilangnya produktivitas semua orang. Di chatbots, semuanya berbasis API, jadi Anda tinggal memperbaiki masalah di backend, menerapkan perubahan di produksi, dan woah masalah diperbaiki untuk pengguna Anda tanpa khawatir. Anda juga menghemat banyak waktu dari bug yang dilaporkan pengguna.

Bayangkan Anda membuat bot untuk mencari restoran dan kemudian Anda ingin menambahkan kemampuan mencari hotel, penerbangan, dll. Pengguna dapat dengan mudah meminta informasi tersebut, dan sistem chatbot backend Anda akan menangani semuanya.

Misalkan Anda sedang membuat chatbot Facebook Messenger; Anda dapat mengontrol hampir semuanya, termasuk antarmuka yang dilihat pengguna di aplikasinya, langsung dari backend Anda. Di bot Facebook Messenger, Anda dapat memilih apakah pengguna mengklik tombol untuk mengatakan Ya/Tidak atau hanya memasukkan teks sederhana.

Permintaan pasar

Lima puluh empat persen pengembang di seluruh dunia menggunakan chatbot untuk pertama kalinya pada tahun 2016. Ada permintaan besar untuk membangun chatbot sederhana yang dapat digunakan oleh perusahaan, dan mereka mencari pengembang yang dapat membangunnya untuk mereka. Setelah Anda menyelesaikan Bab 3 buku ini, saya yakin Anda dapat mulai menjual jasa Anda ke perusahaan dengan mudah. Anda juga dapat memulai startup Anda sendiri di bidang keahlian Anda dengan memperkenalkan chatbot untuk domain tersebut. Mampu membangun chatbot end-to-end adalah keterampilan baru yang harus dimiliki, dan itulah alasan rata-rata gaji pasar juga sangat baik bagi pengembang chatbot.

Meningkatnya permintaan chatbots dapat dilihat dari banyaknya chatbots yang dikembangkan di platform pengembang seperti Facebook. Facebook memiliki 100.000 bot aktif bulanan di platform Messenger, dan terus bertambah. Anda akan terkejut mengetahui bahwa Messenger memiliki 600 juta pengguna pada bulan April 2015, tumbuh menjadi 900 juta pada bulan Juni 2016, 1 miliar pada bulan Juli 2016, dan 1,2 miliar pada bulan April 2017.

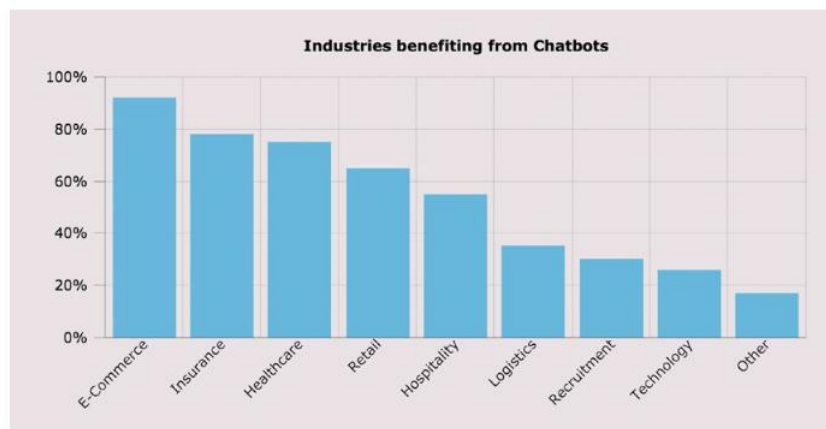
Kurva Pembelajaran

Baik Anda berlatar belakang frontend/backend atau hanya tahu sedikit pemrograman, ada kemungkinan besar untuk mempelajari hal-hal baru saat Anda membuat

atau belajar membuat chatbot. Dalam proses ini Anda akan belajar banyak hal. Misalnya, Anda dapat mempelajari lebih lanjut tentang Human Computer Interaction (HCI), yang membahas tentang desain dan penggunaan teknologi komputer, yang berfokus pada antarmuka antara manusia dan komputer. Anda akan mempelajari cara membuat atau menggunakan API atau layanan web, menggunakan API pihak ketiga seperti Google API, Twitter API, Uber API, dll. Anda akan memiliki peluang besar untuk mempelajari tentang Natural Language Processing, pembelajaran mesin, perilaku konsumen, dan banyak hal teknis dan non teknis lainnya.

2.8 Industri yang Terkena Dampak Chatbots

Mari kita lihat sekilas industri yang paling diuntungkan dari chatbots. Sebuah studi penelitian oleh Mindbrowser bekerja sama dengan Chatbots Journal mengumpulkan data dari 300+ individu yang berpartisipasi dari beragam industri termasuk ritel online, penerbangan, logistik, rantai pasokan, e-commerce, perhotelan, pendidikan, teknologi, manufaktur, dan pemasaran & periklanan. Jika kita melihat grafik pada Gambar 2-5, terlihat jelas bahwa e-commerce, asuransi, layanan kesehatan, dan ritel adalah industri yang paling diuntungkan dari chatbots. Industri-industri ini sangat bergantung pada daya tanggap tim layanan pelanggan dengan cara yang efisien sehingga menghemat waktu. Mengingat fakta bahwa chatbot pandai dalam hal itu, jelas bahwa chatbot akan segera diterima di industri-industri ini.



Gambar 2.5. Industri teratas yang akan mendapatkan keuntungan terbesar dari chatbots

Saat ini, chatbot masih mendapatkan daya tarik di sektor-sektor baru dalam berbagai bentuk. 5 hingga 10 tahun ke depan akan sangat penting bagi chatbots untuk menyebarkan berita di berbagai industri yang tidak memiliki pengalaman bekerja dengan chatbots.

2.9 Garis Waktu Singkat Chatbots

Mari kita lihat sejarah singkat kronologi bagaimana chatbots dirumuskan. Sangat penting untuk mengetahui dari mana teknologi chatbot berasal dan bagaimana teknologi itu dibentuk. Chatbots tentu saja mendapatkan popularitas baru-baru ini, tetapi upaya tersebut dilakukan dengan menggunakan teknologi ini selama puluhan tahun. Sejarah chatbots tentu

Percakapan Antarmuka Pengguna (User Interface) Dalam Chatbot – Dr. Budi Raharjo

akan membuat Anda takjub mengenai seberapa jauh kemajuan kami sejak kami memulainya.

a) 1950

Tes Turing dikembangkan oleh Alan Turing. Itu adalah ujian terhadap kemampuan mesin untuk menunjukkan perilaku cerdas yang setara, atau tidak dapat dibedakan, dengan perilaku manusia.

b) 1966

Eliza, chatbot pertama, diciptakan oleh Joseph Weizenbaum, dirancang untuk menjadi seorang terapis. Ini digunakan untuk mensimulasikan percakapan dengan menggunakan “pencocokan pola” dan metodologi substitusi yang memberikan ilusi pemahaman kepada pengguna dari pihak bot.

c) 1972

Parry, sebuah program komputer oleh psikiater dan ilmuwan Stanford Kenneth Colby, mencontohkan perilaku penderita skizofrenia paranoid.

d) 1981

Chatbot Jabberwocky dibuat oleh programmer Inggris Rollo Carpenter. Ini dimulai pada tahun 1981 dan diluncurkan di internet pada tahun 1997. Tujuan dari chatbot ini adalah untuk “mensimulasikan obrolan alami manusia dengan cara yang menarik, menghibur, dan lucu.”

e) 1985

Mainan robot nirkabel, Tomy Chatbot, mengulangi pesan apa pun yang terekam di kasetnya.

f) 1992

Sbaitso, chatbot yang dibuat oleh Creative Labs untuk MS-DOS, “berbicara” dengan pengguna seolah-olah pengguna adalah seorang psikolog dengan suara digital. Kata-kata umpatan yang berulang-ulang dan masukan yang salah dari pengguna menyebabkan Dr. Sbaitso “rusak” dalam “PARITY ERROR” sebelum dapat mengatur ulang sendiri.

g) 1995

A.L.I.C.E (Entitas Komputer Internet Linguistik Buatan) dikembangkan oleh pemenang Hadiah Nobel Richard Wallace.

h) 1996

Hex, dikembangkan oleh Jason Hutchens, didasarkan pada Eliza dan memenangkan Hadiah Loebner pada tahun 1996.

i) 2001

Smarterchild, bot cerdas yang dikembangkan oleh ActiveBuddy, didistribusikan secara luas di seluruh jaringan pesan instan dan SMS global. Implementasi aslinya dengan cepat tumbuh untuk menyediakan akses cepat ke berita, cuaca, informasi saham, jadwal film, daftar halaman kuning, dan data olahraga terperinci, serta berbagai alat (asisten pribadi, kalkulator, penerjemah, dll.).

j) 2006

Ide Watson muncul dari meja makan; itu dirancang untuk bersaing di acara TV “Jeopardy.” Pada tahap pertama, ia hanya mampu mendapatkan sekitar 15 persen

jawaban yang benar, namun kemudian Watson mampu mengalahkan kontestan manusia secara rutin.

k) 2010

Siri, asisten pribadi yang cerdas, diluncurkan sebagai aplikasi iPhone dan kemudian diintegrasikan sebagai bagian dari iOS. Siri adalah spin-out dari SRI International Artificial Intelligence Center. Mesin pengenalan ucapannya disediakan oleh Nuance Communications, dan Siri menggunakan teknologi pembelajaran mesin canggih agar dapat berfungsi.

l) 2012

Google meluncurkan chatbot Google Now. Awalnya diberi nama sandi "Majel" setelah Majel Barrett, istri Gene Roddenberry dan pengisi suara sistem komputer dalam franchise Star Trek; itu juga diberi nama sandi sebagai "asisten".

m) 2014

Amazon merilis Alexa. Kata "Alexa" memiliki konsonan keras dengan X, sehingga dapat dikenali dengan presisi lebih tinggi. Inilah alasan utama Amazon memilih nama ini.

n) 2015

Cortana, asisten virtual yang dibuat oleh Microsoft. Cortana dapat mengatur pengingat, mengenali suara alami, dan menjawab pertanyaan menggunakan informasi dari mesin pencari Bing. Namanya diambil dari karakter fiksi kecerdasan buatan dalam seri video game Halo.

o) 2016

Pada bulan April 2016, Facebook mengumumkan platform bot untuk Messenger, termasuk API untuk membangun chatbots untuk berinteraksi dengan pengguna. Peningkatan selanjutnya dilakukan termasuk bot yang dapat berpartisipasi dalam grup, layar pratinjau, dan kemampuan pemindaian QR melalui fungsi kamera Messenger untuk mengarahkan pengguna langsung ke bot. Pada bulan Mei 2016, Google meluncurkan bot berkemampuan suara pesaing Amazon Echo yang disebut Google Home di konferensi pengembang perusahaan. Ini memungkinkan pengguna untuk mengucapkan perintah suara untuk berinteraksi dengan berbagai layanan.

p) 2017

Woebot adalah agen percakapan otomatis yang membantu Anda memantau suasana hati, mempelajari diri sendiri, dan membuat Anda merasa lebih baik. Woebot menggunakan kombinasi teknik NLP, keahlian psikologis (terapi perilaku kognitif [CBT]), tulisan yang bagus, dan selera humor untuk mengatasi depresi.

Masalah Apa yang Dapat Saya Selesaikan Menggunakan Chatbots?

Pertanyaan ini menjadi menantang ketika Anda tidak mengetahui cakupan bot Anda atau tidak ingin membatasinya untuk menjawab pertanyaan. Sangat penting untuk diingat bahwa ada batasan terhadap apa yang dapat dilakukan chatbots. Selalu terasa bahwa kita sedang berbicara dengan makhluk mirip manusia yang sangat cerdas, tetapi bot tertentu dirancang dan dilatih untuk berperilaku dengan cara tertentu dan menyelesaikan masalah tertentu saja. Ia tidak dapat melakukan segalanya, setidaknya untuk saat ini. Masa depan pasti cerah. Jadi, kita sampai pada pertanyaan untuk mencari tahu apakah pernyataan

masalah Anda benar-benar bagus dan Anda dapat membuat bot untuk mengatasinya. Jika jawaban atas ketiga pertanyaan ini adalah ya, maka Anda siap melakukannya.

Bisakah Masalah Diselesaikan dengan Tanya Jawab Sederhana atau Komunikasi Bolak Balik?

Sangat penting untuk tidak mencoba menjadi pahlawan ketika menyelesaikan masalah apa pun yang sangat baru bagi Anda. Anda harus selalu berusaha untuk membatasi cakupan masalah. Bangun fungsionalitas dasar lalu tambahkan di atasnya. Jangan mencoba membuatnya rumit pada potongan pertama. Ini tidak berfungsi di perangkat lunak.

Bayangkan Mark Zuckerberg berpikir keras dan menghabiskan waktu membangun semua fitur Facebook sejak awal. Menandai teman, memiliki tombol suka, menyukai komentar pengguna, pesan yang lebih baik, video langsung, reaksi terhadap komentar, dll. fitur-fitur ini tidak ada bahkan ketika Facebook didanai oleh lebih dari 1 juta pengguna terdaftar di platform tersebut. Akankah dia benar-benar berhasil jika dia terus membangun fitur-fitur ini terlebih dahulu dan kemudian meluncurkan platformnya?

Jadi, kita harus selalu berusaha membuat fitur-fitur yang hanya dibutuhkan saat ini tanpa harus melakukan rekayasa berlebihan. Sekarang, kembali ke pertanyaan pertama, “Dapatkah masalah diselesaikan dengan tanya jawab sederhana atau komunikasi bolak-balik?”

Anda hanya perlu membatasi cakupan Anda dan jawaban Anda adalah ya. Kita sama sekali tidak membatasi diri kita untuk memecahkan masalah yang kompleks, namun jelas membatasi diri kita untuk menyelesaikan masalah yang kompleks sekaligus. “Anda harus membuat setiap detailnya sempurna. Dan Anda harus membatasi jumlah detailnya.”

—Jack Dorsey

Apakah Ada Masalah yang Sangat Berulang yang Membutuhkan Analisis atau Pengambilan Data?

Pertanyaan ini penting karena baik dari sudut pandang bisnis atau sudut pandang pengembang, apa yang dilakukan dan dilakukan chatbot adalah membuat orang yang menggunakannya menjadi efisien dan produktif. Dan bagaimana Anda melakukannya? Dengan menghilangkan kebutuhan pengguna untuk melakukan hal-hal yang berulang sendiri.

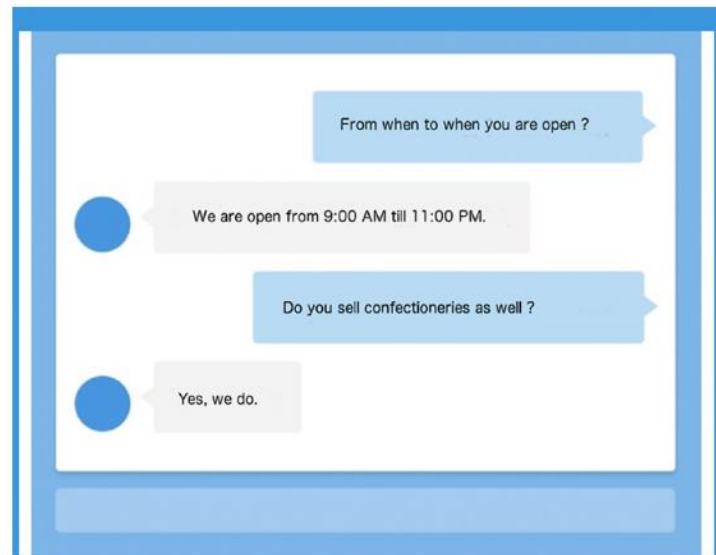
Chatbots jelas lebih mampu mengotomatiskan beberapa hal yang sangat berulang, tetapi Anda akan selalu menemukan bahwa sebagian besar chatbots terutama mencoba memecahkan masalah yang sama baik itu dengan belajar di bawah pengawasan (baca: “Dengan Supervised Learning”) atau otodidak (baca: “Dengan Pembelajaran Tanpa Pengawasan”).

Bisakah Tugas Bot Anda Diotomatiskan dan Diperbaiki?

Kecuali Anda berpikir untuk membuat chatbot hanya untuk tujuan pembelajaran Anda, Anda harus memastikan masalah yang Anda coba selesaikan dapat diotomatisasi. Mesin sudah mulai belajar dan melakukan berbagai hal sendiri, namun ini masih merupakan tahap yang sangat baru. Apa yang Anda anggap tidak dapat diotomatisasi saat ini mungkin akan terotomatisasi dalam beberapa tahun.

Bot QnA

Salah satu contoh pernyataan masalah yang bagus untuk membuat chatbot adalah Bot QnA. Bayangkan sebuah bot yang dilatih untuk memahami berbagai pertanyaan pengguna yang jawabannya sudah tersedia di halaman FAQ sebuah website. Jika Anda kembali dan mencoba mencari jawaban dari ketiga pertanyaan di atas, jawabannya adalah ya. Lihat Gambar 2-6 dan Anda akan menemukan apa yang dilakukan bot FAQ.



Gambar 2.6. Contoh chatbot pertanyaan umum

Ini hanyalah pertanyaan berulang yang mungkin dihubungi oleh pelanggan toko tertentu dan ditanyakan atau coba temukan jawabannya dengan mengunjungi situs web dan menelusuri halamannya.

Pikirkan ketika Anda memiliki chatbot seperti ini dan ia menjawab pertanyaan Anda seperti manusia dalam hitungan detik dan bahkan melakukan lebih dari yang dapat Anda bayangkan. Ini hanyalah sebagian kecil dari kemampuan chatbot.

Sekarang, mari kita coba menganalisis tiga pertanyaan di atas dan jawabannya dalam kasus QnA Bot.

- *Apakah permasalahan dapat diselesaikan dengan tanya jawab sederhana atau komunikasi bolak-balik?*

Ya, FAQ hanyalah pertanyaan sederhana yang sering diajukan dan jawaban relatifnya. Mungkin ada FAQ berbasis konteks, tetapi kecuali Anda memecahkan masalah multidomain menggunakan chatbots, Anda tidak akan mengalami masalah ini. Mungkin ada situasi di mana dua atau lebih pertanyaan tampak serupa, namun Anda selalu dapat merancang bot untuk mengajukan pertanyaan kembali kepada pengguna jika ada keraguan.

- *Apakah ada masalah yang sangat berulang sehingga memerlukan analisis atau pengambilan data?*

Ya, FAQ mengharuskan kita mengambil data dari database dan menampilkan semuanya sekaligus di situs web atau mungkin secara dinamis. Namun pengguna harus

menelusuri semua pertanyaan satu per satu untuk menemukan pertanyaan yang dicarinya dan kemudian melihat jawabannya. Banyak menyisir UI sebelum konsumen benar-benar mendapatkan jawabannya... atau mungkin tidak. Mengapa tidak membiarkan bot kita melakukan itu untuk kita?

➤ *Apakah tugas bot Anda dapat diotomatisasi dan diperbaiki?*

Ya, bot FAQ perlu mendapatkan pertanyaan, menganalisis pertanyaan, mengambil informasi dari database, dan mengembalikannya kepada pengguna. Tidak ada apa pun di sini yang tidak dapat dilakukan dengan menggunakan pengkodean. Dan juga, proses yang diperbaiki tidak akan berubah secara real-time.

2.10 Dimulai Dengan Chatbots

Ada tiga langkah yang harus diikuti sebelum membangun chatbots. Kami akan membahasnya secara singkat di sini.

1. Pikirkan semua skenario atau tugas yang Anda ingin agar chatbot Anda dapat lakukan, dan kumpulkan semua pertanyaan terkait dalam berbagai bentuk yang dapat diminta untuk melakukan tugas tersebut. Setiap tugas yang Anda ingin chatbot Anda lakukan akan menentukan maksudnya.
2. Setiap pertanyaan yang Anda daftarkan atau maksudkan dapat direpresentasikan dalam berbagai cara. Itu tergantung bagaimana pengguna mengekspresikannya.
Misalnya: Alexa, Matikan lampunya. Alexa, bisakah kamu mematikan lampunya? Bisakah Anda mematikan lampunya? Pengguna dapat menggunakan salah satu kalimat berikut untuk memerintahkan bot mematikan lampu. Semua itu mempunyai maksud/tugas yang sama untuk mematikan lampu, namun ditanyakan dengan pengucapan/variasi yang berbeda.
3. Tulis semua logika Anda untuk menjaga pengguna tetap terikat pada aliran yang Anda pilih setelah Anda mengenali maksud pengguna.
Misalnya, Anda membuat bot untuk membuat janji temu dengan dokter. Kemudian Anda meminta pengguna Anda untuk memberikan nomor telepon, nama, dan spesialis, lalu Anda menunjukkan slotnya dan kemudian memesannya.

Dalam hal ini Anda dapat mengharapkan pengguna mengetahui detail ini dan tidak mencoba mengakomodasi semua hal di bot itu sendiri, seperti spesialis masalah telinga yang disebut THT. Namun, melakukan hal ini bukanlah masalah besar. Jadi, kembali lagi untuk menentukan cakupan bot Anda, bergantung pada waktu dan sumber daya yang Anda miliki untuk membangun aplikasi.

2.11 Pohon Keputusan di Chatbots

Jika Anda mengetahui tentang pohon keputusan, itu sangat bagus karena Anda akan sering membutuhkan pengetahuan tersebut saat merancang alur chatbot Anda. Namun jika Anda tidak tahu tentang pohon keputusan, Googling saja akan membantu Anda mempelajari konsep sederhana yang banyak digunakan dalam Ilmu Komputer.

Menggunakan Pohon Keputusan di Chatbots

Dalam konteks chatbots, pohon keputusan hanya membantu kita menemukan jawaban yang tepat atas pertanyaan pengguna. *Pohon keputusan* adalah alat pendukung keputusan yang menggunakan grafik atau model keputusan seperti pohon dan kemungkinan konsekuensinya, termasuk hasil kejadian yang tidak disengaja, biaya sumber daya, dan utilitas. Ini adalah salah satu cara untuk menampilkan algoritma yang hanya berisi pernyataan kontrol bersyarat.

Bagian tersulit saat membuat chatbot adalah melacak blok kode if...else. Semakin besar jumlah keputusan yang harus diambil, semakin sering if...else muncul dalam kode. Namun pada saat yang sama, blok-blok ini diperlukan untuk menyandikan alur percakapan yang kompleks. Jika masalahnya rumit dan membutuhkan banyak if...else di kehidupan nyata, maka hal itu memerlukan kode untuk menyesuaikan dengan cara yang sama.

Bagaimana Pohon Keputusan Membantu?

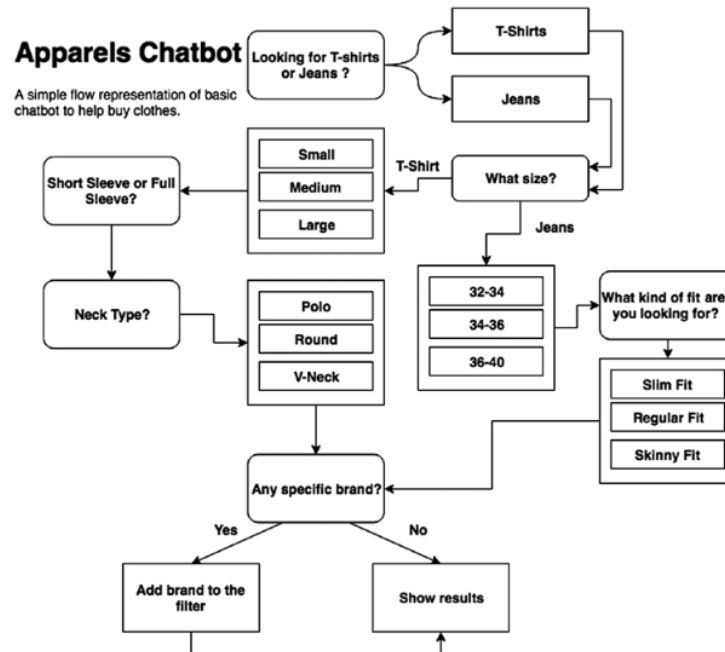
Pohon keputusan sederhana untuk ditulis dan dipahami, namun merupakan representasi kuat dari solusi yang dibuat untuk masalah yang dimaksud. Mereka mewarisi kemampuan unik untuk membantu kita memahami banyak hal.

- Membantu menciptakan gambaran utuh mengenai masalah yang dihadapi. Dengan melihat pohon keputusan, kita dapat dengan mudah memahami apa yang kurang atau apa yang perlu diubah.
- Membantu debug lebih cepat. Pohon keputusan seperti kitab suci pendek atau, katakanlah, representasi visual dari dokumen spesifikasi kebutuhan perangkat lunak, yang dapat dijadikan referensi oleh pengembang, manajer produk, atau pimpinan untuk menjelaskan perilaku yang diharapkan atau membuat perubahan apa pun jika diperlukan.
- AI masih belum berada pada tahap yang dapat dilatih dengan banyak data dan bekerja dengan akurasi 100 persen. Hal ini masih membutuhkan banyak pegangan dalam menulis logika dan aturan bisnis. Pohon keputusan membantu ketika sulit meminta mesin mempelajari dan melakukannya.

Mari kita ambil contoh sederhana dan mencoba memahami bagaimana hal ini membantu dalam membangun chatbots. Lihatlah contoh diagram chatbot yang dimulai dengan pertanyaan apakah pengguna sedang mencari kaos atau jeans, dan berdasarkan masukan tersebut, alur diagram melangkah lebih jauh dengan memberikan opsi terkait produk dengan mengajukan lebih banyak pertanyaan. Anda tidak perlu membuat pohon keputusan yang lengkap, tetapi Anda harus memiliki alur pertanyaan yang jelas di setiap langkah sebelum mulai membuat chatbot.

Misalkan Anda sedang membuat chatbot serupa yang membantu orang membeli pakaian secara online. Hal pertama yang akan Anda lakukan adalah membuat pohon keputusan atau diagram alur serupa untuk membantu chatbot Anda mengajukan pertanyaan yang sesuai pada waktu yang tepat. Hal ini sangat diperlukan untuk mengatur ruang lingkup setiap langkah dan apa yang perlu dilakukan pada tahap tersebut. Anda akan memerlukan diagram status atau diagram alur sederhana nanti saat Anda benar-benar membuat kode chatbot pertama Anda. Ingatlah untuk tidak terlalu ketat saat membuat diagram seperti

Gambar 2-7; buatlah sesederhana mungkin dan kemudian tambahkan fungsionalitas tambahan nanti. Keuntungan dari proses tersebut adalah waktu pengembangan akan dipersingkat, dan nantinya fungsionalitas akan digabungkan secara longgar dan akan mulai masuk akal sebagai komponen. Seperti pada contoh, setelah membuat fungsionalitas dasar, Anda juga dapat menambahkan pilihan warna, kisaran harga, peringkat, dan opsi diskon.



Gambar 2.7. Representasi sederhana dari chatbot pakaian untuk membeli pakaian secara online

Pastinya ada lebih banyak hal yang dapat Anda tambahkan ke kasus penggunaan sebelumnya tergantung pada kebutuhan Anda. Namun Anda harus memastikan bahwa Anda tidak membuatnya terlalu rumit untuk diri Anda sendiri dan juga bagi pengguna.

Pohon keputusan tidak hanya membantu Anda menjaga pengguna tetap terikat pada arus tetapi juga merupakan cara yang sangat efektif untuk mengidentifikasi maksud selanjutnya yang mungkin datang dalam bentuk pertanyaan dari pelanggan. Jadi, bot Anda akan menanyakan serangkaian pertanyaan sesuai dengan pohon keputusan yang telah Anda buat. Setiap node mempersempit tujuan pelanggan melalui maksud chatbot.

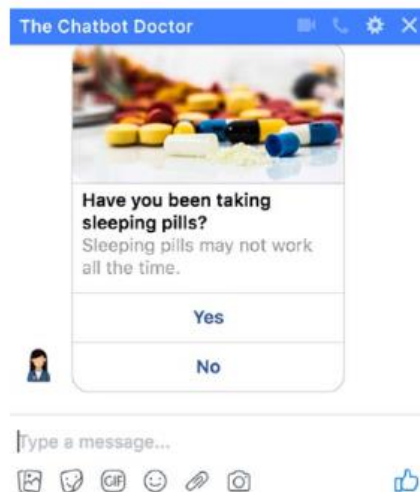
Misalkan Anda membuat chatbot untuk lembaga keuangan misalnya, bank yang dapat melakukan transfer uang berdasarkan permintaan Anda setelah autentikasi. Dalam hal ini, bot Anda mungkin ingin memverifikasi detail akun terlebih dahulu dan meminta pengguna mengonfirmasi jumlahnya, lalu bot mungkin meminta untuk memvalidasi nama akun target, nomor akun, jenis akun, dll.

Anda tidak dapat atau tidak ingin menggunakan API OTP (kata sandi satu kali) kecuali Anda telah memvalidasi apakah saldo akun pengguna lebih dari jumlah yang diminta.

Hal ini terjadi pada kita semua dan juga pada pelanggan. Mereka menjadi frustrasi ketika pertanyaan mereka tidak dijawab dengan benar. Menggunakan pohon keputusan

untuk chatbot Anda pasti akan membuat pengalaman pengguna Anda lebih baik dibandingkan jika Anda tidak menggunakannya.

Sering kali Anda akan menemukan masalah yang menyelesaikan beberapa maksud secara terprogram. Jadi, intinya adalah, “Jika Anda tidak dapat menyelesaikan sesuatu secara terprogram, maka selesaikanlah dengan desain.” Lihat Gambar 2-8 di mana bot mencoba mengikuti kuis kesehatan dan ingin mengetahui apakah antibiotik dapat bekerja untuk segala hal.



Gambar 2.8. Contoh penyelesaian kasus penggunaan berdasarkan desain

Karena jawabannya diharapkan berupa Boolean (Benar/Salah), Anda hanya memberikan dua tombol untuk diklik pengguna alih-alih membiarkan mereka mengetik dan menunggu untuk memperbaiki kesalahan mereka.

Ini diselesaikan dengan desain daripada menulis banyak kode yang akan menangani masukan pengguna yang tidak terduga. Anda akan memiliki begitu banyak skenario saat membangun chatbots di mana hanya dengan memberikan tombol, Anda akan dapat dengan cepat mengetahui maksud pengguna. Penting untuk memahami skenario seperti itu dan menyediakan tombol untuk kenyamanan Anda sendiri serta untuk pengguna yang tidak perlu mengetikkan jawaban opsional yang jelas.

Kerangka Chatbots/Bot Terbaik

- <https://woebot.io/>
 - Dapat melacak suasana hati Anda
 - Membantu Anda merasa lebih baik
 - Memberi Anda wawasan dengan melihat pola suasana hati Anda
 - Mengajarkan Anda bagaimana menjadi positif dan berenergi tinggi
- <https://qnamaker.ai/>
 - Buat, latih, dan publikasikan bot tanya jawab sederhana berdasarkan FAQ, URL, dan dokumen terstruktur dalam hitungan menit.
 - Uji dan sempurnakan tanggapan menggunakan antarmuka obrolan yang familier.
 - <https://dialogflow.com/>

- Sebelumnya dikenal sebagai api.ai dan sangat populer di kalangan penggemar chatbot.
 - Memberikan cara baru kepada pengguna untuk berinteraksi dengan produk Anda dengan membangun antarmuka percakapan berbasis suara dan teks yang menarik dan didukung oleh AI.
 - Terhubung dengan pengguna di Google Assistant, Amazon Alexa, Facebook Messenger, dan platform serta perangkat populer lainnya.
 - Menganalisis dan memahami maksud pengguna untuk membantu Anda merespons dengan cara yang paling berguna.
- <https://core.rasa.ai>
 - Kerangka kerja untuk membangun perangkat lunak percakapan
 - Anda dapat menerapkan tindakan yang dapat dilakukan bot Anda dengan kode Python.
 - Daripada sekumpulan pernyataan if...else, logika bot Anda didasarkan pada model probabilistik yang dilatih berdasarkan contoh percakapan.
 - <https://wit.ai>
 - Wit.ai memudahkan pengembang untuk membuat aplikasi dan perangkat yang dapat Anda ajak bicara atau kirim SMS.
 - Diakuisisi oleh Facebook dalam waktu 21 bulan setelah peluncurannya, tim wit.ai berkontribusi terhadap mesin NLP milik Facebook di dalam Facebook.
 - Anda dapat menggunakan wit.ai untuk membuat chatbot, otomatisasi rumah, dll.
 - Wit.ai mirip dengan cara kerja Dialogflow tetapi tidak sekaya Dialogflow. Orang-orang awalnya menggunakan wit.ai, karena gratis dan Dialogflow tidak, namun kemudian Dialogflow menjadi gratis juga.
 - <https://www.luis.ai/>
 - Layanan berbasis pembelajaran mesin untuk membangun bahasa alami ke dalam aplikasi, bot, dan perangkat IoT.
 - Dengan cepat membuat model khusus yang siap digunakan untuk perusahaan dan terus ditingkatkan.
 - <http://botkit.ai>
 - Pembangun percakapan visual
 - Statistik dan metrik bawaan
 - Dapat dengan mudah diintegrasikan dengan Facebook, Microsoft, IBM Watson, Slack, Telegram, dll.

2.12 Komponen Chatbot dan Terminologi yang Digunakan

Komponen sistem chatbot sangat sedikit. Di bagian ini kita akan membahas secara singkat komponen chatbot yang akan Anda temui di bab selanjutnya. Memiliki pemahaman teoretis dasar tentang sistem apa pun sebelum mendalaminya selalu membantu. Anda akan mendapatkan gambaran yang jelas setelah bagian ini tentang terminologi teknis yang digunakan saat membuat chatbot menggunakan Python. Terminologi ini akan sering digunakan di bab-bab selanjutnya ketika kita benar-benar mulai membangun chatbots.

Maksud

Ketika pengguna berinteraksi dengan chatbot, apa niatnya menggunakan chatbot tersebut/apa yang dia minta?

Misalnya, ketika pengguna mengatakan, “Pesan tiket film,” kepada chatbot, kita sebagai manusia dapat memahami bahwa pengguna ingin memesan tiket film. Ini dimaksudkan untuk bot. Itu bisa diberi nama maksud “book_movie”. Contoh lainnya adalah ketika pengguna mengatakan, “Saya ingin memesan makanan,” atau “Bisakah Anda membantu saya memesan makanan?” Ini bisa disebut maksud “pesanan_makanan”. Demikian pula, Anda dapat menentukan maksud sebanyak yang Anda inginkan.

Entitas

Maksud memiliki metadata tentang maksud yang disebut “Entitas.” Dalam contoh, “Pesan tiket bioskop”, pemesanan tiket dapat berupa niat dan entitasnya adalah “film”, yang juga dapat berupa hal lain, seperti penerbangan, konser, dll.

Anda dapat memberi label entitas umum untuk digunakan di seluruh maksud. Entitas dapat direpresentasikan sebagai kuantitas, jumlah, atau volume. Intent juga dapat memiliki banyak entitas.

✚ Misalnya: Pesankan saya sepatu ukuran 8. Mungkin ada dua entitas di sini:

✚ Kategori: Ukuran Sepatu: 8

Ucapan

Ucapan tidak lain hanyalah bentuk berbeda dari pertanyaan/maksud yang sama yang mungkin ditunjukkan oleh pengguna Anda.

- Ingat kita pernah membahas tentang cara mematikan lampu? Itu adalah contoh bagaimana pengguna dapat menggunakan ucapan berbeda untuk maksud yang sama.
- Disarankan untuk memiliki maksimal 10 ucapan per maksud dan minimal 5 ucapan, namun hal ini tidak dibatasi.

Melatih Bot

Pelatihan pada dasarnya berarti membangun model yang akan belajar dari serangkaian maksud/entitas dan ucapan yang telah ditentukan tentang cara mengkategorikan ucapan-ucapan baru dan memberikan skor keyakinan yang menyertainya. Saat kita melatih sistem menggunakan ucapan, ini disebut pembelajaran terawasi. Kami akan segera belajar lebih banyak tentang melakukan hal ini secara praktis.

Skor Keyakinan

Setiap kali Anda mencoba menemukan maksud suatu ucapan, model Anda akan menghasilkan skor keyakinan. Skor ini menunjukkan seberapa yakin model pembelajaran mesin Anda dalam mengenali maksud pengguna.

Hanya itu yang ingin kami bahas di bab pertama “Pengantar Chatbots.” Anda harus memiliki gagasan yang adil tentang chatbots dari sudut pandang bisnis dan dari sudut pandang teknis. Kami menelusuri jalur sejarah milik chatbots. Sungguh menakjubkan seberapa jauh chatbots telah berevolusi.

Kami belajar tentang bagaimana chatbots telah berevolusi selama periode waktu tertentu dan mengapa chatbots adalah suatu keharusan bagi bisnis untuk berkembang dalam persaingan yang ketat ini. Kami mempelajari berbagai kerangka chatbot dan juga

mengetahui terminologi yang digunakan untuk chatbot melalui contoh. Kami akan menggunakannya di bab-bab mendatang. Anda seharusnya berada pada tahap di mana Anda mengetahui jenis chatbot apa yang ingin Anda buat dan bagaimana perilakunya saat dibuat. Kerjakan semua penulisan dan pohon keputusan Anda, jika diperlukan, dan setelah kita mempelajari dasar-dasar Pemahaman Bahasa Alami di bab berikutnya, kita dapat segera mulai membangun chatbot kami.

Jangan khawatir meskipun Anda tidak memikirkan apa pun. Kami akan mencoba membangun chatbot keren selangkah demi selangkah dengan semua konsep yang dipelajari di bab-bab mendatang.

BAB 3

MEMBUAT CHATBOT

Setelah mengetahui tentang chatbot dan sejarah singkatnya, dalam bab ini kita akan mempelajari cara membuat chatbot pertama Anda. Kami akan membangun chatbot yang akan bertindak sebagai pemandu wisata kota bagi pengguna. Kami akan menggunakan percakapan turis populer, seperti belajar tentang kota, mencari tahu hal yang harus dilakukan, dan mencari tempat makan sebagai contoh tugas. Kita akan mempelajari cara menggunakan berbagai alat dalam toolkit untuk membangun chatbot yang mampu melakukan tugas percakapan ini.

Mari kita jelajahi alat populer, Chatfuel, dan pelajari cara membuat chatbot dari awal. Chatfuel adalah alat yang memungkinkan Anda membuat chatbot tanpa harus membuat kode sama sekali. Ini adalah alat berbasis web dengan editor GUI yang memungkinkan pengguna membuat chatbot secara modular. Selain membangun chatbot, ini juga memungkinkan pengembang memantau dan mengelola kinerja bot menggunakan alat manajemen dan analitik.

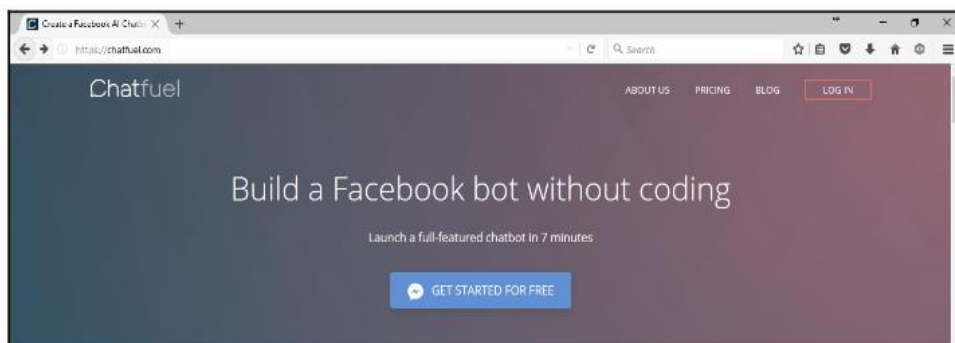
Dalam bab ini, kita akan menjelajahi elemen dasar chatbot. Kami kemudian akan membuat chatbot pemandu wisata dari awal dan menerapkannya di Facebook Messenger. Kami akan mengeksplorasi berbagai fitur yang disediakan oleh Chatfuel untuk memperluas kemampuan chatbot seperti integrasi backend dan penyiaran pesan. Terakhir, kita akan melihat sekilas rangkaian analitik.

Pada akhir bab ini, Anda akan dapat:

- ✓ Pahami dasar-dasar Chatfuel
- ✓ Memahami konsep alur percakapan dan navigasi
- ✓ Pahami blok penyusun: blok, kartu, plugin, dan atribut
- ✓ Kelola konteks menggunakan pasangan nilai kunci dan tangani ucapan pengguna
- ✓ Bangun chatbot dan integrasikan webhook untuk pemrosesan backend
- ✓ Integrasikan chatbot di Facebook Messenger

3.1 Menggunakan Chatfuel

Mari kita mulai. Untuk membuat akun dengan Chatfuel, kunjungi <https://chatfuel.com>:

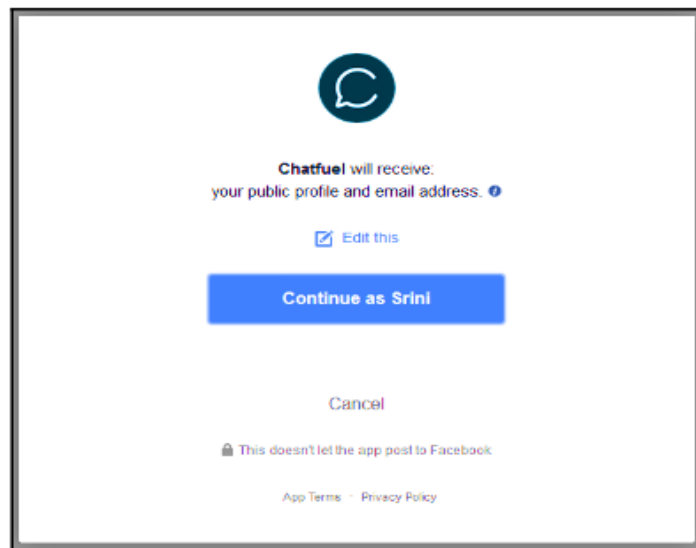


Gambar 3.1 Halaman Website Chatfuel

Klik GET STARTED FOR FREE. Ingat, toolkit Chatfuel saat ini gratis untuk digunakan. Ini akan mengarahkan Anda ke salah satu dari dua opsi berikut:

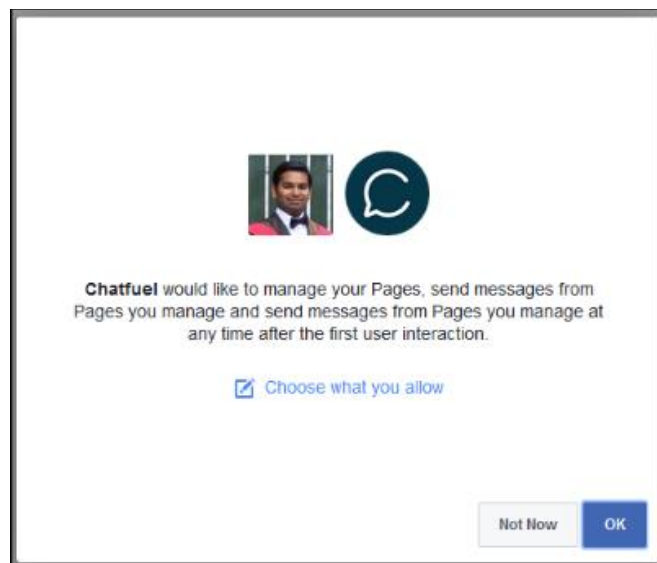
- ✓ Jika Anda masuk ke Facebook, ia akan meminta izin untuk menautkan akun Chatfuel Anda ke akun Facebook Anda
- ✓ Jika Anda belum login, Anda akan diminta login ke Facebook terlebih dahulu sebelum meminta izin

Tautan Chatfuel ke Facebook untuk menyebarkan bot. Jadi memerlukan izin untuk menggunakan akun Facebook Anda:



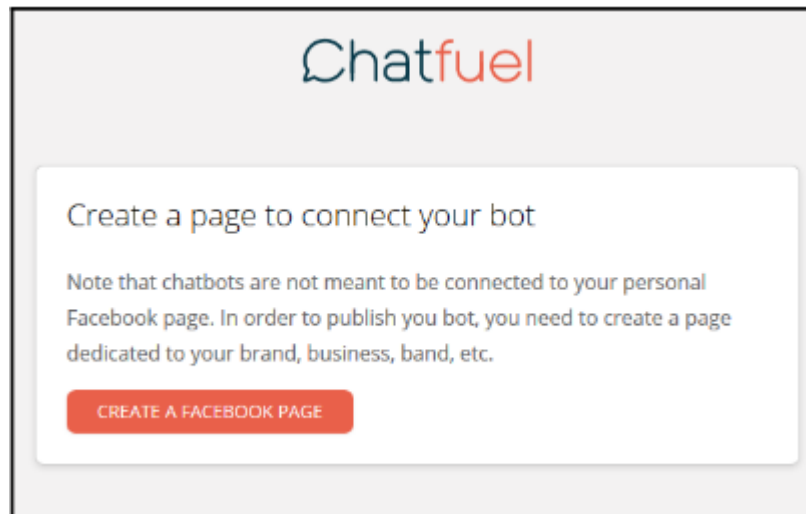
Gambar 3.2 Tautan Halaman Chatfuel

Otorisasi Chatfuel untuk menerima informasi tentang Anda dan menjadi pengelola Halaman Anda:



Gambar 3.3 Otorisasi Halaman Chatfuel

Itu dia! Anda siap membuat bot pertama Anda:

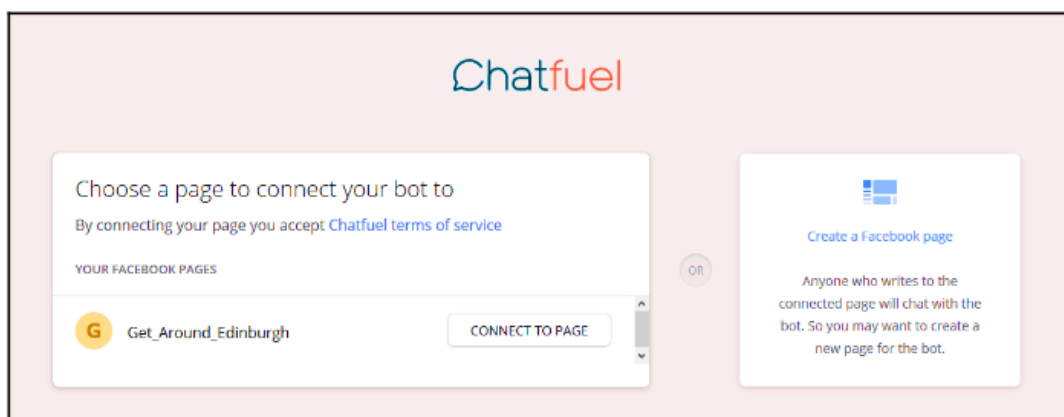


Gambar 3.4 Halaman Pertama membuat Chatfuel

Membangun bot pertama Anda

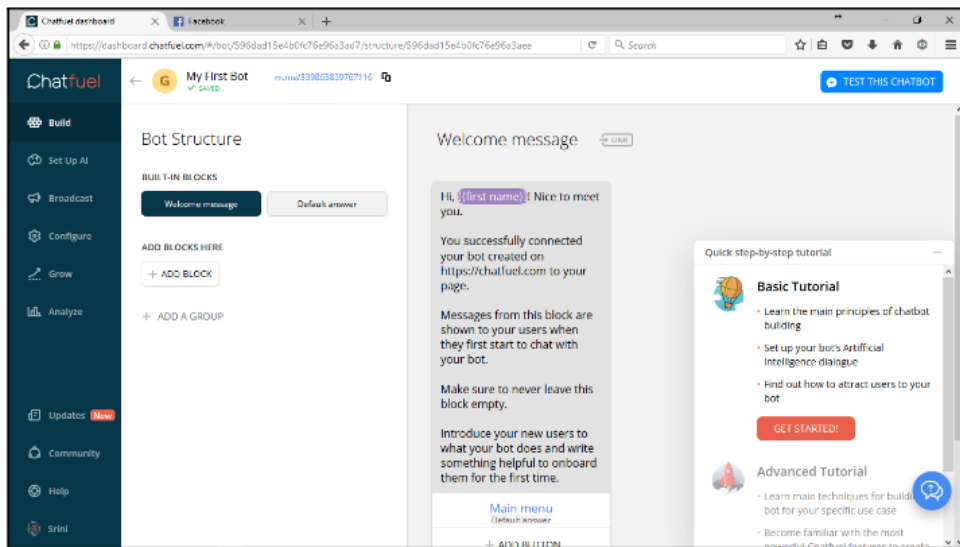
Bot bahan bakar obrolan dapat dipublikasikan di dua platform penerapan: Facebook Messenger dan Telegram. Mari kita buat chatbot untuk Facebook Messenger terlebih dahulu. Untuk melakukan itu, kita perlu membuat Halaman Facebook. Setiap chatbot di Facebook Messenger harus dilampirkan ke sebuah halaman. Inilah cara kita membuat Halaman Facebook:

1. Kunjungi <https://www.facebook.com/pages/create>.
2. Klik kategori (yang sesuai dengan konten halaman). Dalam kasus kami, kami akan menggunakan Merek atau Produk dan memilih Halaman Aplikasi.
3. Beri nama pada halaman tersebut. Dalam kasus kita, mari gunakan Get_Around_Edinburgh. Perhatikan bahwa Facebook tidak memberikan kemudahan untuk mengubah nama halaman. Jadi pilihlah dengan bijak.
4. Setelah halaman dibuat, Anda akan melihat Chatfuel meminta izin untuk terhubung ke halaman tersebut:



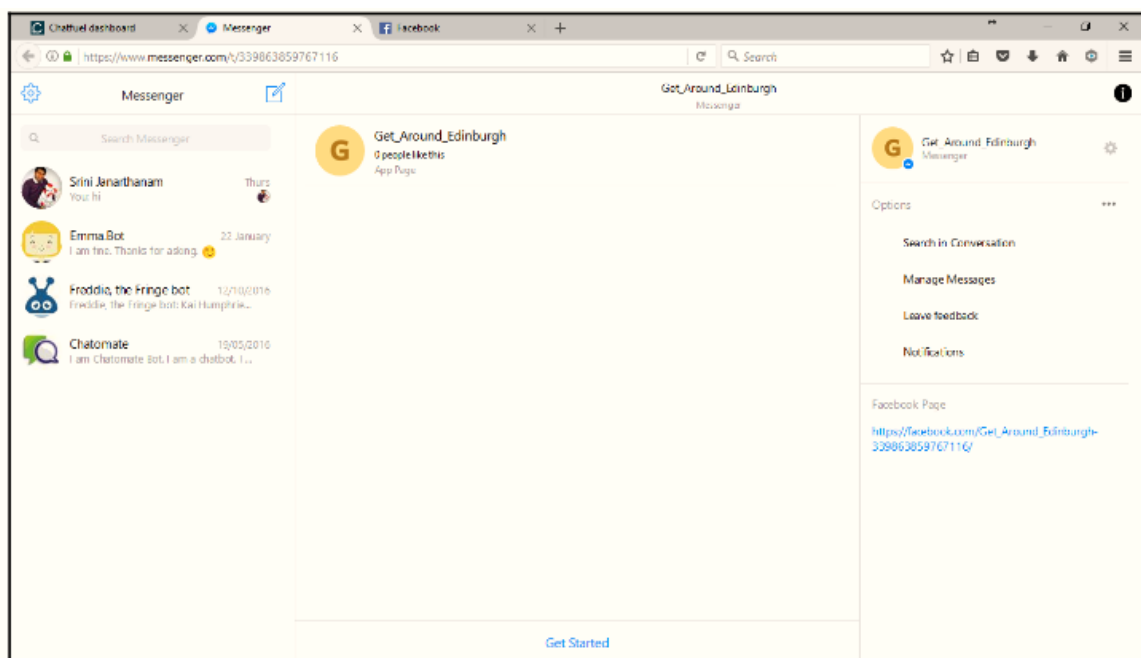
Gambar 3.5 Bagian Menautkan Izin Ke Halaman

5. **Klik HUBUNGAN KE HALAMAN.** Anda akan dibawa ke editor bot.
6. Nama bot diatur ke Bot Pertama Saya. Ini memiliki URL Messenger, yang dapat Anda lihat di samping namanya. URL Messenger dimulai dengan m.me. Anda mungkin memperhatikan bahwa bot juga dilengkapi dengan pesan Selamat Datang yang ada di dalamnya. Di sebelah kiri, Anda melihat menu utama dengan sejumlah opsi, dengan opsi Build dipilih secara default. Kami akan menjelajahi opsi lain di bagian selanjutnya:



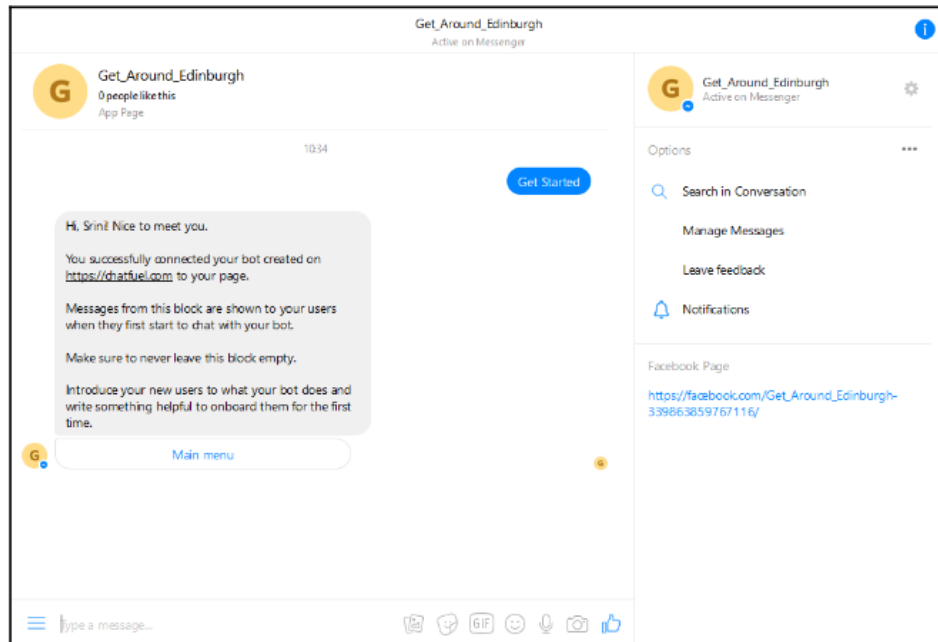
Gambar 3.6 Tampilan Halaman Beranda Chatfuel

7. **Klik URL Messenger** untuk memulai percakapan pertama Anda dengan bot. Ini akan membuka Facebook Messenger di tab browser Anda:



Gambar 3.7 Tampilan Halaman Messenger Chatfuel

8. Untuk memulai percakapan, klik tombol Memulai di bagian bawah jendela obrolan.
9. Ini dia! Percakapan Anda baru saja dimulai. Bot telah mengirimkan Anda pesan selamat datang:



Gambar 3.8 Tampilan Percakapan bot

Perhatikan bagaimana ia menyapa Anda dengan nama Anda. Itu karena Anda telah memberikan bot akses ke informasi Anda di Facebook. Sekarang setelah Anda membuat bot pertama dan melakukan percakapan dengannya, beri tepukan pada diri Anda sendiri. Selamat datang di dunia chatbot!

3.2 Blok Bangunan Dasar

Sebelum kita melanjutkan membuat bot, mari kita lihat elemen dasar:

- 1) Blok
- 2) Kartu-kartu
- 3) Tombol
- 4) Plugin
- 5) Atribut

Blok

Kembali ke editor dan lihat tab Struktur Bot. Di bawah Struktur Bot, Anda akan menemukan dua jenis blok bawaan dan ditentukan pengguna. Pesan selamat datang dan Jawaban default adalah dua blok bawaan. Kami akan membangun blok yang ditentukan pengguna untuk mengimplementasikan kemampuan percakapan chatbot. Setiap blok dapat dianggap sebagai segmen respons dari chatbot. Setiap blok dapat berisi satu atau lebih kartu.

Kartu-kartu

Kartu adalah konstruksi yang digunakan untuk mengirim pesan ke platform penerapan. Ada banyak jenis kartu. Kartu teks adalah tipe dasar kartu; mereka membawa pesan teks dan tombol opsional. Kartu lainnya mencakup gambar, balasan cepat (susunan tombol), daftar, dan galeri. Kami akan menyelidiki masing-masing hal ini nanti.

Tombol

Kebanyakan kartu memiliki tombol. Tombol digunakan untuk memberikan opsi respons kepada pengguna. Pengguna dapat menanggapi pertanyaan dan permintaan chatbot dengan mengklik tombol. Ini adalah pendekatan alternatif yang memungkinkan pengguna memasukkan permintaan dan tanggapan mereka menggunakan teks bahasa alami.

Plugin

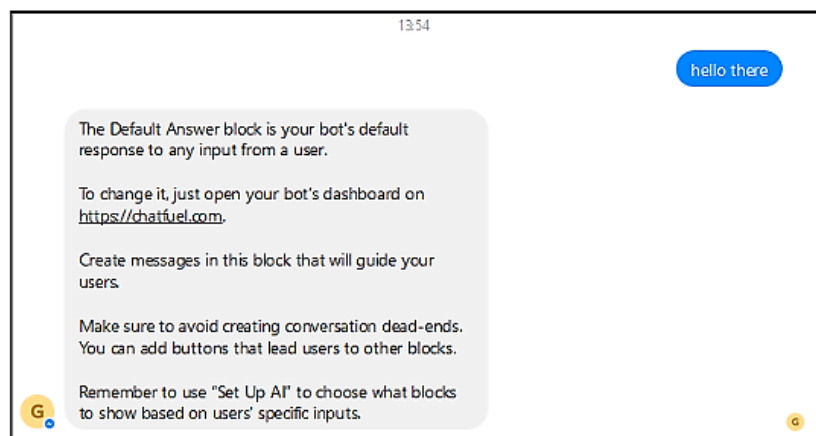
Plugin adalah bagian dari blok di mana chatbot diharuskan melakukan tugas khusus, bukan hanya merespons pengguna. Misalnya, Anda dapat menggunakan plugin untuk mengirim email ke diri Anda sendiri berisi semua informasi yang didapat chatbot dari pengguna. Kita akan melihat sejumlah plugin nanti.

Atribut

Atribut adalah variabel dalam bahasa programmer. Ini adalah placeholder tempat kita dapat menyimpan data sementara selama percakapan. Ada atribut yang sudah ditentukan sebelumnya dan ada pula yang ditentukan pengguna. Dalam hal ini, pengguna sebenarnya berarti pengembang. Ini dapat digunakan untuk mengarahkan percakapan ke satu arah atau yang lain. Atau mereka dapat digunakan untuk mengumpulkan data dari pengguna untuk diproses lebih lanjut dalam modul backend.

Blok bawaan

Ada dua blok yang telah ditentukan sebelumnya: Pesan selamat datang dan Pesan default. Pesan selamat datang dilengkapi dengan kartu default dengan teks selamat datang di dalamnya. Blok selamat datang dipicu saat pengguna pertama kali menemukan chatbot. Di Facebook Messenger, ini dipicu ketika pengguna menekan tombol Memulai. Blok jawaban default dipicu ketika bot tidak mengetahui cara menangani pesan masukan pengguna. Coba sapa bot di Facebook Messenger dan lihat responsnya:

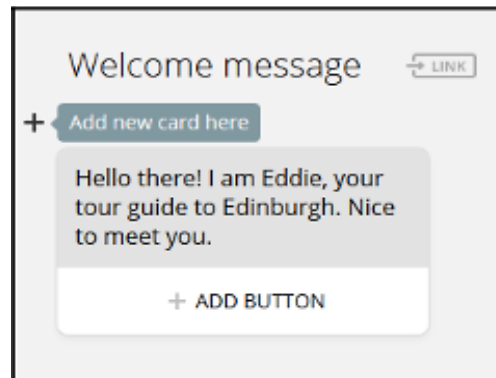


Gambar 3.9 Tampilan Pesan Bot

Langkah selanjutnya

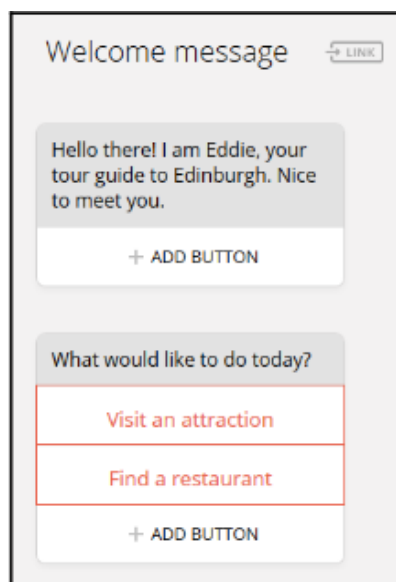
Sekarang setelah kita mempelajari dasar-dasarnya, mari mulai membuat bot kita:

1. Di blok selamat datang, klik teks default dan edit. Mengarahkan mouse di sekitar blok dapat menampilkan opsi seperti menghapus kartu, mengatur ulang urutan kartu, dan menambahkan kartu baru di antara kartu yang ada. Hapus tombol menu Utama:



Gambar 3.10 Tampilan Blok Selamat Datang

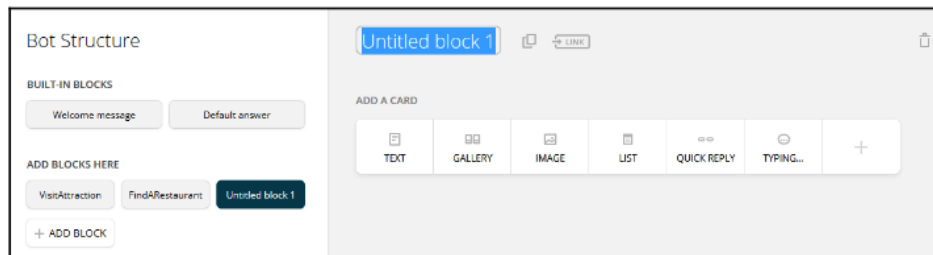
2. Tambahkan kartu Teks. Mari tambahkan kartu teks tindak lanjut dan ajukan pertanyaan kepada pengguna.
3. Tambahkan tombol untuk tanggapan pengguna. **Klik TAMBAHKAN TOMBOL** dan ketik nama tombol. Abaikan nama blok untuk saat ini. Karena tidak lengkap, maka akan muncul warna merah. Ingat, Anda dapat menambahkan hingga tiga tombol ke kartu teks:
- 4.



Gambar 3.11 Tampilan Kartu Teks

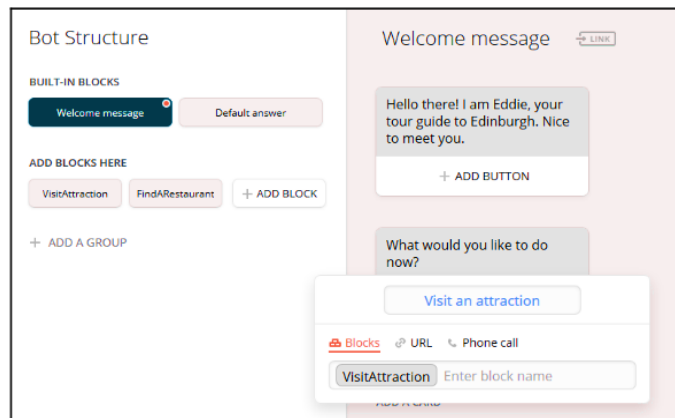
5. Respons tombol harus dikaitkan dengan blok sehingga ketika pengguna menekan tombol, chatbot mengetahui apa yang harus dilakukan atau dikatakan. Mari tambahkan beberapa blok. Untuk menambahkan blok baru, klik TAMBAHKAN BLOK di tab Struktur

Bot. Ini menciptakan blok baru tanpa judul. Di sisi kanan, isikan nama blok. Ulangi hal yang sama untuk setiap blok yang ingin Anda bangun:



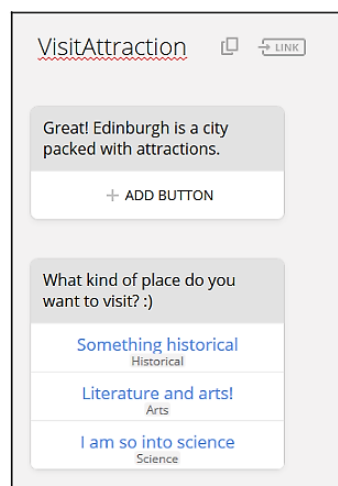
Gambar 3.12 Tampilan Struktur Bot

6. Sekarang, kembali ke tombol dan tentukan nama blok yang akan dihubungkan. Klik tombol, pilih Blok, dan berikan nama blok:



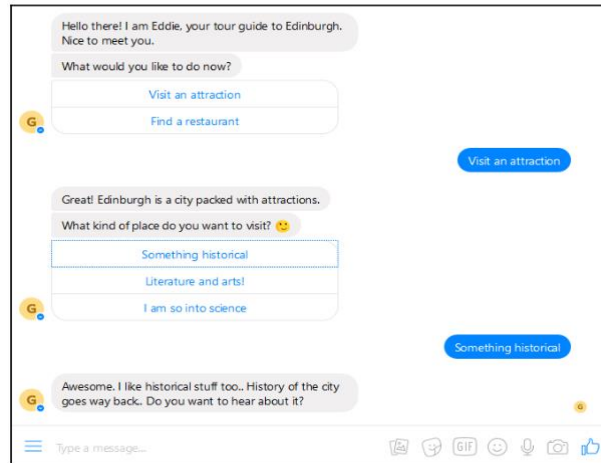
Gambar 3.13 Tampilan Struktur Bot Untuk Menentukan Nama Blok

7. Untuk setiap blok yang Anda buat, tambahkan konten dengan menambahkan kartu yang sesuai. Ingat, setiap blok bisa memiliki lebih dari satu kartu. Setiap kartu akan muncul sebagai respons, satu demi satu:

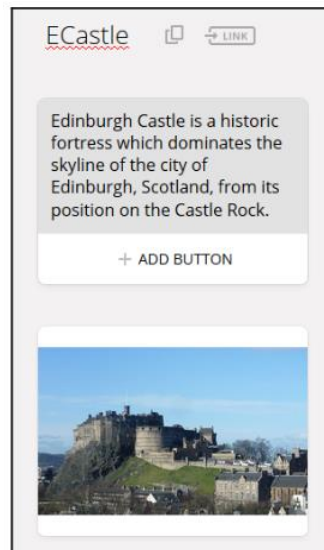


Gambar 3.14 Halaman Kunjungan

Ulangi langkah sebelumnya untuk menambahkan lebih banyak blok dan menghubungkannya ke tombol blok lainnya. Setelah selesai, Anda dapat mengujinya dengan **mengklik tombol UJI CHATBOT** ini di pojok kanan atas editor. Anda sekarang akan melihat pesan selamat datang baru dengan tombol untuk tanggapan. Lanjutkan dan klik salah satunya untuk melakukan percakapan:



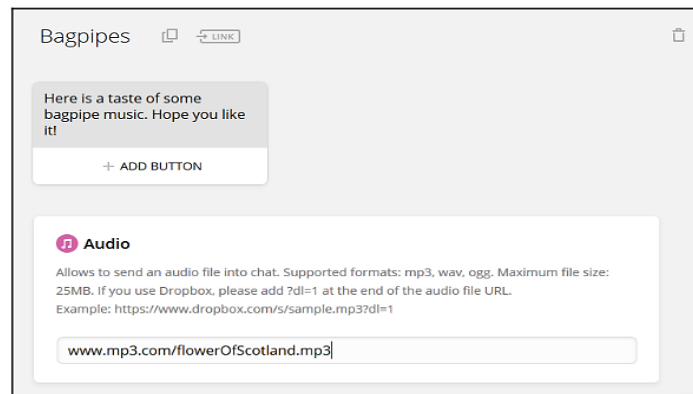
Gambar 3.15 Tampilan Chat Bot



Gambar 3.16 Menambahkan Gambar

Audio

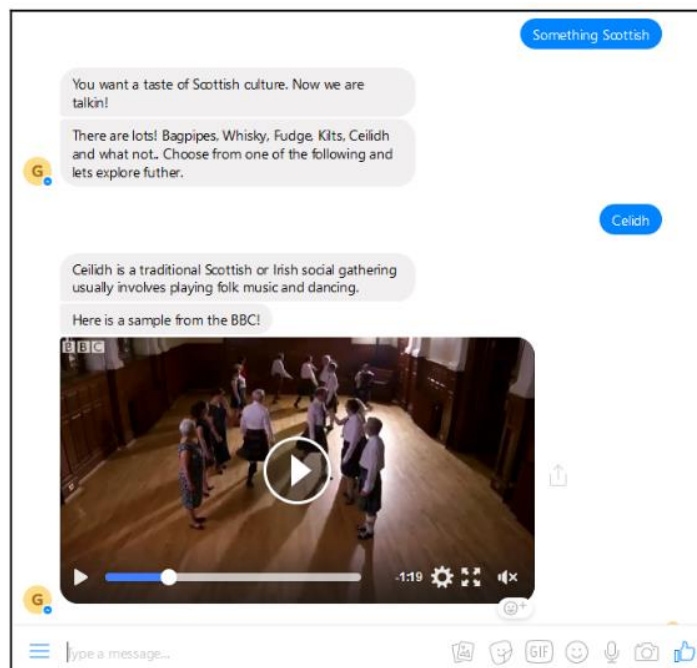
Anda dapat mengirim file audio melalui chat yang dapat langsung diputar oleh pengguna. Untuk mengirim file audio, klik ikon + di akhir menu ADD A CARD. Pilih Audio dan berikan URL file audio. Format seperti MP3, WAV, dan OGG didukung:



Gambar 3.17 Tampilan chat bot untuk upload audio

Video

Menambahkan kartu video mirip dengan menambahkan kartu audio. Klik ikon +, pilih Video, dan tentukan URL file video. Hanya format MP4 yang didukung dan ukuran maksimum video dibatasi hingga 25 MB. Ini tampilannya di chat:



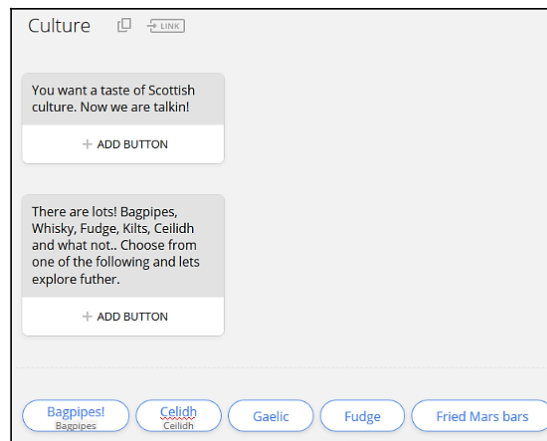
Gambar 3.18 Menambahkan Video dalam chatbot

Balasan cepat

Balasan cepat adalah serangkaian tombol yang ditampilkan secara horizontal. Ini dapat digunakan untuk memperoleh jawaban dari pengguna dengan cara yang sama seperti tombol yang dilampirkan pada kartu teks. Bedanya di sini Anda dapat memiliki hingga 10 tombol dan tidak terpasang pada satu kartu. Tombol-tombol tersebut juga akan hilang setelah pengguna mengklik salah satu tombol tersebut atau mengetikkan permintaan/tanggapannya alih-alih mengklik tombol tersebut. Hal ini sangat berguna ketika

Anda memiliki lebih dari tiga opsi respons atau opsi yang tidak boleh bertahan lama setelah giliran karena akan kehilangan konteks.

Untuk menambahkan serangkaian tombol balasan cepat, klik Balasan Cepat, tambahkan nama tombol. Untuk setiap tombol, Anda dapat menentukan blok tujuan yang dituju. Jika respons diperlakukan sebagai jawaban, respons tersebut juga dapat disimpan sebagai atribut pengguna. Kami akan memeriksa atribut pengguna lebih lanjut:

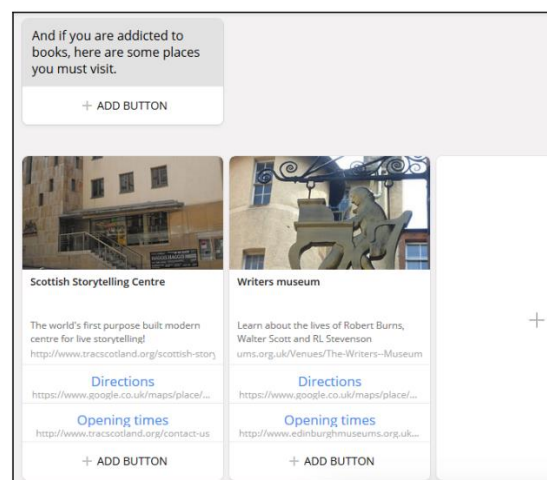


Gambar 3.19 tampilan chat bot balasan cepat

Galeri

Galeri adalah jenis kartu yang memungkinkan kita menciptakan pengalaman yang kaya dalam menyajikan konten. Daripada hanya serangkaian tombol sebagai balasan cepat, konten yang sama dapat disajikan dengan cara yang diperkaya dengan gambar dan deskripsi teks. Untuk menambahkan kartu galeri, klik Galeri di bawah tab TAMBAHKAN KARTU. Kartu galeri dapat dianggap sebagai kartu dengan sejumlah minicard di dalamnya.

Lanjutkan untuk menambahkan gambar, judul, subjudul, URL opsional, dan tombol (maksimal tiga) untuk minicard default. Setelah selesai, tambahkan minicard lagi dengan mengklik ikon + di sebelah kanan:



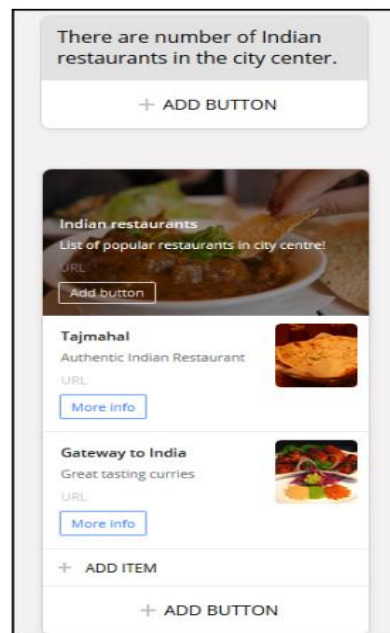
Gambar 3.20 Tampilan menambahkan galeri

Daftar menu

Percakapan Antarmuka Pengguna (User Interface) Dalam Chatbot – Dr. Budi Raharjo

Kartu daftar sangat mirip dengan kartu galeri dalam hal konten. Ini menyediakan cara untuk menyajikan konten yang kaya kepada pengguna dengan gambar dan URL. Namun, tidak seperti kartu galeri, tata letak itemnya vertikal. Untuk menambahkan kartu daftar, klik ikon Daftar di tab TAMBAHKAN KARTU.

Tambahkan gambar sampul, judul, URL, subjudul, dan tombol ajakan bertindak, dan buat header grup. Ulangi hal yang sama untuk setiap item dalam daftar. Faktanya, elemen teratas tidak harus berupa header grup dan cukup menjadi elemen pertama dalam daftar yang ditampilkan secara jelas dengan latar belakang gambar:



Gambar 3.21 tampilan chat bot menu

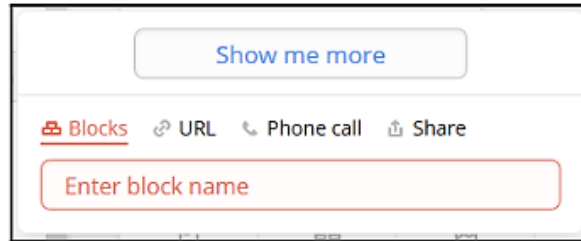
Pada contoh yang ditampilkan, kita memiliki header grup dengan judul Restoran India. Namun, jika Anda tidak ingin memiliki header, Anda cukup menggunakan tempat untuk item pertama dalam daftar. Kartu daftar memiliki minimal dua item dan maksimal empat. Anda juga dapat menambahkan tombol opsional di akhir daftar, misalnya, untuk memberikan opsi "lebih" untuk meminta lebih banyak item.

Navigasi

Bagaimana cara pengguna dan chatbot menavigasi percakapan? Bagaimana mereka merespons satu sama lain dan memajukan pembicaraan? Di bagian ini, kita akan memeriksa perangkat untuk memfasilitasi alur percakapan.

Tombol

Tombol adalah cara yang memungkinkan pengguna merespons chatbot dengan cara yang jelas. Anda dapat menambahkan tombol ke teks, galeri, dan kartu daftar. Tombol memiliki label dan muatan. Label adalah apa yang dapat dilihat pengguna. Payload adalah apa yang terjadi di backend ketika pengguna mengklik tombol:



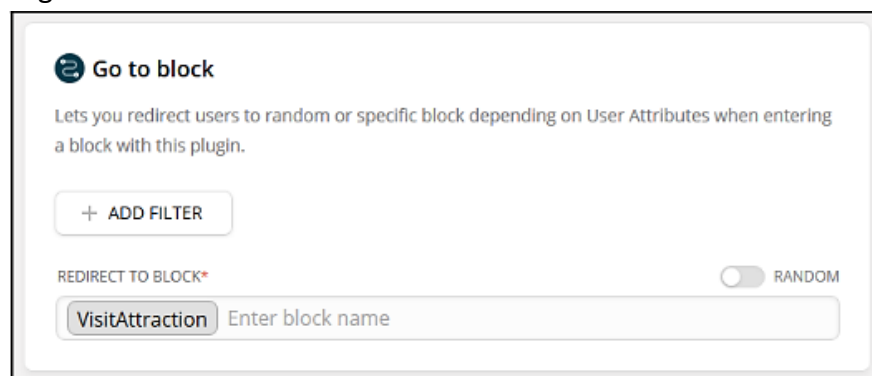
Gambar 3.22 tampilan tombol

Sebuah tombol dapat mengambil salah satu dari empat jenis muatan: blok berikutnya, URL, nomor telepon, atau berbagi. Blok berikutnya diidentifikasi dengan nama bloknnya. Ini akan memberitahu chatbot blok mana yang harus dieksekusi ketika tombol ditekan. URL dapat ditentukan, jika chatbot ingin membuka halaman web di browser web yang tertanam. Karena browser sudah tertanam, ukuran jendela juga dapat ditentukan. Nomor telepon dapat ditentukan, jika chatbot ingin melakukan panggilan suara ke seseorang (misalnya panggilan untuk memesan meja). Dan terakhir, opsi Bagikan dapat digunakan di kartu seperti daftar dan galeri untuk membagikan kartu tersebut dengan kontak pengguna lainnya.

Blok Kartu

Tombol dapat digunakan untuk menavigasi pengguna dari satu blok ke blok lainnya, namun pengguna harus menekan tombol untuk mengaktifkan navigasi. Namun, mungkin ada keadaan di mana navigasi perlu dilakukan secara otomatis. Misalnya, jika chatbot memberikan petunjuk langkah demi langkah kepada pengguna tentang cara melakukan sesuatu, chatbot dapat dibuat dengan meletakkan semua kartu (satu langkah informasi per kartu) dalam satu blok. Namun, mungkin ada baiknya untuk menempatkannya di blok yang berbeda demi modularitas. Dalam kasus seperti ini, kita perlu menyediakan tombol langkah berikutnya kepada pengguna untuk melanjutkan ke langkah berikutnya.

Di Chatfuel, kita dapat menggunakan kartu Go to Block untuk mengatasi masalah ini. Kartu Buka Blokir dapat ditempatkan di akhir blok mana pun untuk membawa chatbot ke blok lain. Setelah chatbot mengeksekusi semua kartu dalam satu blok, ia berpindah ke blok lain secara otomatis tanpa campur tangan pengguna. Dengan menggunakan kartu Go to Block, kita dapat membangun chatbot secara modular. Untuk menambahkan kartu Go to Block di akhir blok, pilih TAMBAHKAN KARTU, klik ikon + dan pilih kartu Go to Block. Isi nama blok untuk pengalihan:



Gambar 3.23 tampilan go blok

Pengalihan juga dapat dilakukan secara acak dan bersyarat. Dengan memilih opsi acak, kita dapat membuat chatbot memilih salah satu blok yang disebutkan secara acak. Hal ini menambah sedikit ketidakpastian dalam percakapan. Namun, hal ini perlu digunakan dengan sangat hati-hati karena konteks percakapan mungkin sulit dipertahankan.

Pengalihan bersyarat dapat dilakukan jika ada kebutuhan untuk memeriksa konteks sebelum pengalihan dilakukan. Mari kita meninjau kembali opsi ini setelah kita membahas konteksnya.

3.3 Mengelola Konteks

Dalam percakapan apa pun, konteks percakapan perlu diatur. Konteks dapat dipertahankan dengan membuat cache lokal dimana informasi yang ditransfer antara dua mitra dialog dapat disimpan. Misalnya, pengguna dapat memberi tahu chatbot tentang preferensi makanan mereka, dan informasi ini dapat disimpan dalam konteks untuk referensi di masa mendatang jika tidak segera digunakan. Contoh lainnya adalah percakapan di mana pengguna mengajukan pertanyaan tentang sebuah cerita. Pertanyaan-pertanyaan ini mungkin tidak lengkap dan mungkin perlu ditafsirkan berdasarkan informasi yang tersedia dalam konteksnya.

Di bagian ini, kita akan mengeksplorasi bagaimana konteks dapat direkam dan dimanfaatkan selama percakapan di Chatfuel. Mari kita ambil tugas mencari restoran sebagai bagian dari chatbot pemandu wisata Anda. Percakapan antara chatbot dan pengguna mungkin berlangsung sebagai berikut:

Pengguna : Carikan Restoran

Bot : Ok. Dimana?

Pengguna : Pusat Kota

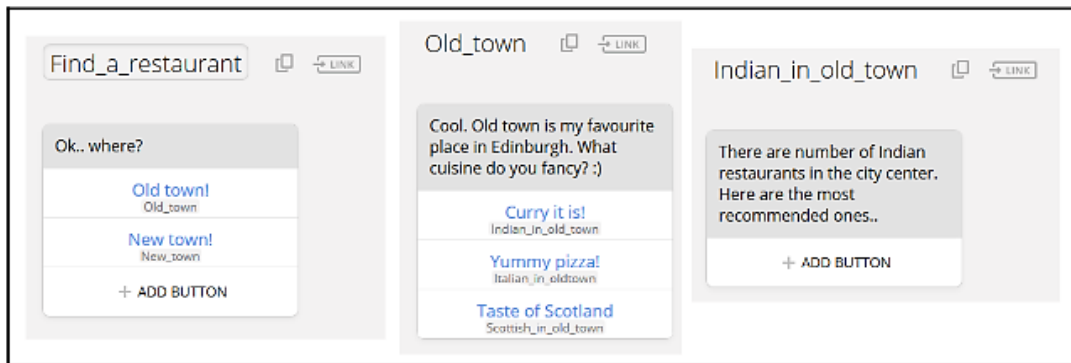
Bot : Ok. Ada Masakan Yang Kamu Suka?

Pengguna : India

Bot : Ok. Biar Kulihat Saya Menemukan Beberapa Restoran India Di Pusat Kota. *Ini Dia.*

Dalam percakapan sebelumnya, hingga ucapan bot terakhir, bot perlu menyimpan informasi secara lokal. Ketika sudah mengumpulkan semua informasi yang dibutuhkan, ia dapat pergi dan mencari database dengan parameter yang sesuai. Perhatikan bahwa informasi tersebut juga perlu digunakan dalam menghasilkan ujaran secara dinamis. Mari kita jelajahi cara melakukan keduanya menghasilkan ucapan secara dinamis dan menelusuri database.

Pertama, kita perlu membangun alur percakapan untuk mengarahkan pengguna melalui percakapan seperti yang kita bahas di bagian Langkah selanjutnya. Mari kita asumsikan bahwa pengguna mengklik tombol Find_a_restaurant di blok selamat datang. Mari buat blok dasar dengan pesan teks dan tombol untuk menavigasi percakapan:

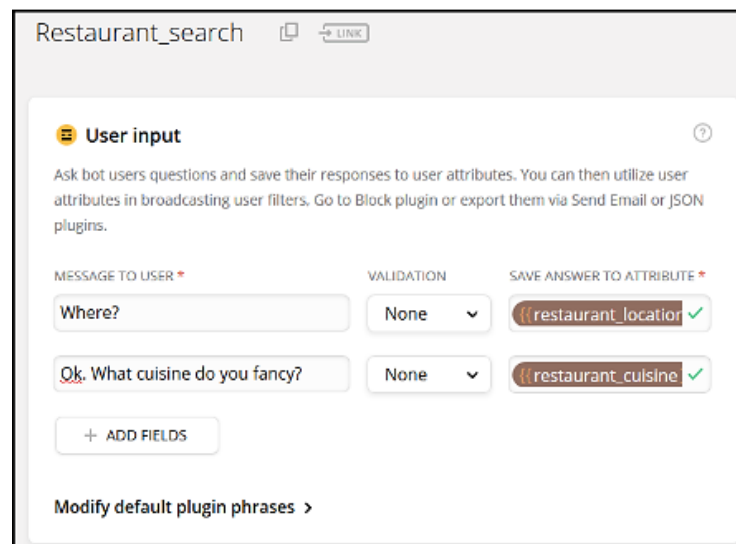


Gambar 3.24 Chat bot dengan navigasi

3.4 Kartu Masukan Pengguna

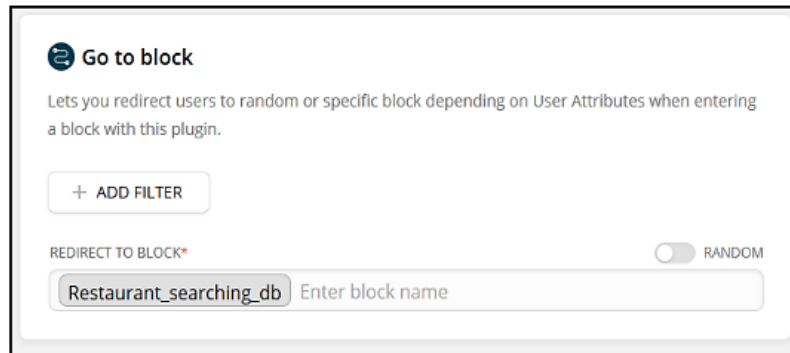
Seperti yang bisa Anda bayangkan, membuat blok untuk setiap kombinasi masakan dan lokasi bisa menjadi tugas yang melelahkan. Mari kita coba membangun fungsionalitas yang sama dengan cara lain bentuk. Untuk menggunakan formulir, kartu masukan pengguna perlu digunakan. Mari kita buat blok baru bernama `Restaurant_search` dan ke dalamnya, mari tambahkan kartu Input Pengguna. Untuk menambahkan kartu Input Pengguna, klik **TAMBAHKAN KARTU**, klik ikon **+**, dan pilih kartu Input Pengguna.

Tambahkan semua pertanyaan yang ingin Anda tanyakan kepada pengguna di bawah *Pesan Kepada Pengguna*. Jawaban atas setiap pertanyaan ini dapat disimpan ke variabel. Sebutkan variabel-variabel yang ada pada setiap pertanyaan. Variabel ini selalu dilambangkan dengan tanda kurung kurawal ganda (misalnya, `{{restaurant_location}}`):



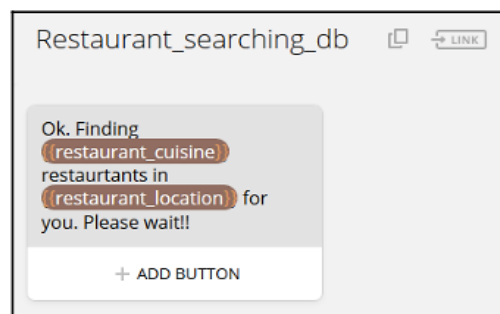
Gambar 3.25 jendela penelusuran

Informasi yang diberikan oleh pengguna juga dapat divalidasi sebelum diterima. Jika informasi yang diperlukan adalah nomor telepon, alamat email, atau nomor, hal ini dapat divalidasi dengan memilih format informasi masukan yang sesuai. Setelah pengguna memasukkan kartu, mari tambahkan kartu Buka Blokir untuk mengarahkan alur ke halaman hasil:



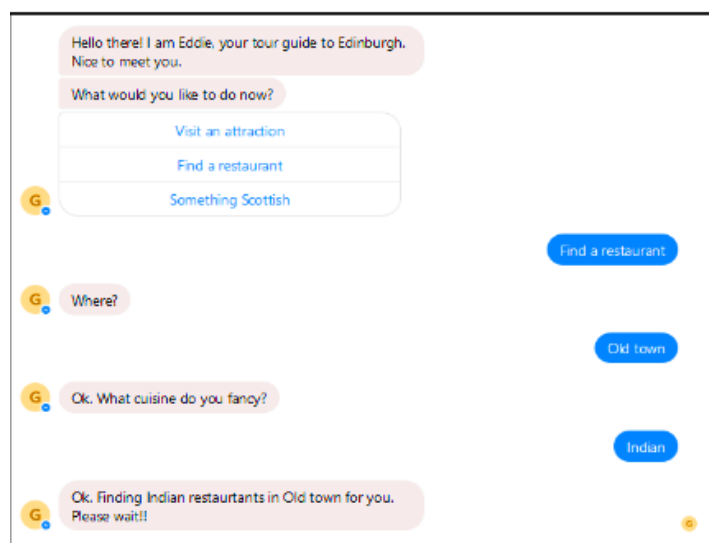
Gambar 3.26 tampilan chat bot blokir

Dan tambahkan blok tempat kami menyajikan hasilnya. Seperti yang Anda lihat di sini, variabel yang menyimpan informasi dapat digunakan dalam ucapan chatbot. Ini akan diganti secara dinamis dari konteks saat percakapan terjadi:



Gambar 3.27 Tampilan percakapan dalam chat bot

Tangkapan layar berikut menunjukkan percakapan sejauh ini di Messenger:



Gambar 3.28 tampilan chat messenger

Mengatur atribut pengguna

Selain kartu masukan pengguna, ada juga cara lain untuk menyimpan informasi dalam konteks. Hal ini dapat dilakukan dengan menggunakan kartu atribut pengguna yang disiapkan. Dengan menggunakan kartu ini, Anda dapat mengatur informasi spesifik konteks kapan saja selama percakapan. Mari kita lihat cara melakukannya. Untuk menambahkan kartu ini, pilih TAMBAHKAN KARTU, klik ikon +, dan pilih kartu Siapkan Atribut Pengguna:

Gambar 3.29 Tampilan menambahkan kartu

Tangkapan layar sebelumnya menunjukkan variabel riwayat-suka-pengguna disetel ke true ketika pengguna meminta atraksi bersejarah. Informasi ini nantinya dapat digunakan untuk mengarahkan percakapan (seperti yang digunakan dalam kartu Buka Blokir) atau untuk memberikan rekomendasi.

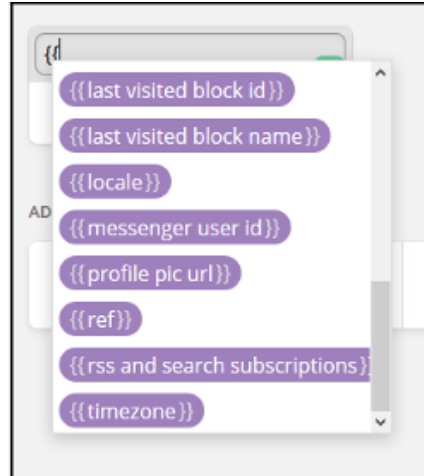
Variabel yang sudah ada dalam konteksnya dapat direset ke nilai baru atau tidak ada nilai sama sekali. Untuk menghapus nilai suatu variabel, gunakan nilai NOT SET khusus dari menu drop-down yang muncul segera setelah Anda mencoba mengisi nilai untuk variabel tersebut. Selain itu, Anda dapat menyetel/mengatur ulang lebih dari satu variabel dalam satu kartu.

3.5 Variabel Kontekstual Default

Selain mendefinisikan variabel kontekstual Anda sendiri, Anda juga dapat menggunakan daftar variabel yang telah ditentukan sebelumnya. Informasi yang terkandung dalam variabel-variabel tersebut antara lain sebagai berikut:

- Informasi yang diperoleh dari platform penerapan (yaitu Facebook) termasuk nama pengguna, jenis kelamin, zona waktu, dan lokasi
- Informasi kontekstual tombol yang terakhir ditekan, nama blok yang terakhir dikunjungi, dan seterusnya

Untuk mendapatkan daftar variabel lengkap, buat kartu teks baru dan ketik {{. Ini akan membuka menu drop-down dengan daftar variabel yang dapat Anda pilih. Daftar ini juga akan menyertakan variabel yang Anda buat:



Gambar 3.30 text variabel yang akan dibuat

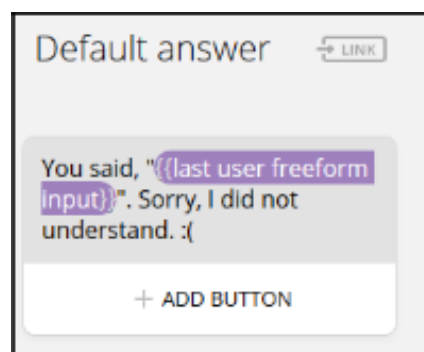
Seperti halnya variabel yang ditentukan pengembang, variabel bawaan ini juga dapat digunakan dalam pesan teks dan pengalihan bersyarat menggunakan kartu Buka Blokir.

Memahami Bahasa Alami

Sejauh ini, kita telah melihat bagaimana alur percakapan terjadi menggunakan tombol. Dalam setiap kasus, pengguna harus menekan tombol atau mengetik informasi untuk mengisi slot, sehingga percakapan dapat berlanjut. Namun, pengguna juga dapat memulai percakapan dan bernavigasi menggunakan masukan bahasa alami (NL) di Chatfuel. Di bagian ini, kita akan mempelajari cara membuat chatbot memahami masukan bahasa alami pengguna dan mengambil tindakan.

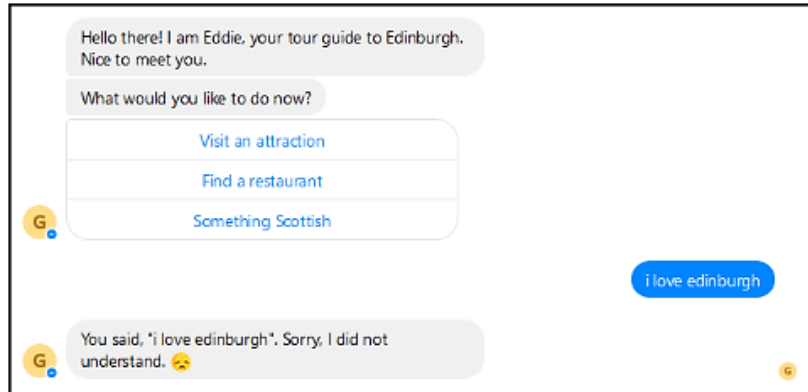
Blok bawaan

Di samping blok Pesan Selamat Datang, ada blok Jawaban default. Blok ini digunakan oleh chatbot sebagai respons terhadap masukan apa pun dari pengguna yang tidak dipahaminya. Silakan ubah isi blok sebagai berikut:



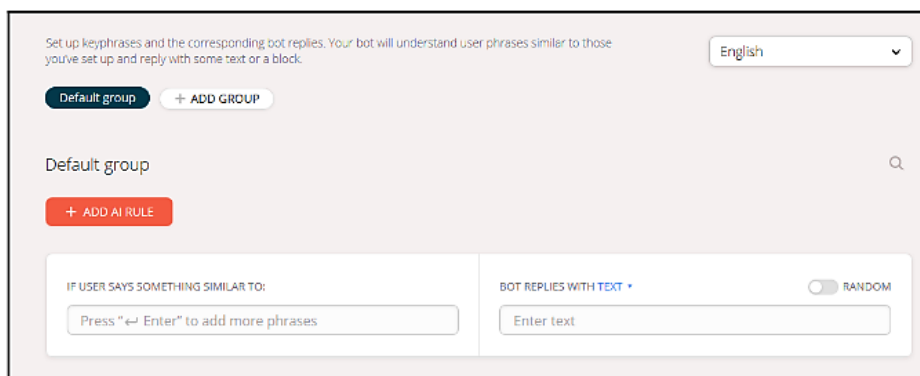
Gambar 3.31 tampilan chat bot selamat datang yang bisa diubah

Sekarang uji bot tersebut dengan mengetikkan pertanyaan ke chatbot di Messenger. Anda dapat melihat bagaimana pesan default digunakan:



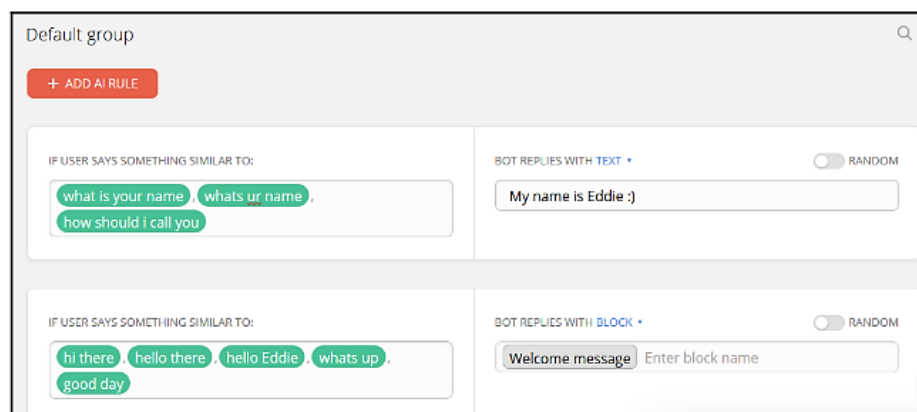
Gambar 3.32 tampilan pertanyaan ke chatbot di Messenger

Sekarang mari kita jelajahi cara menangani masukan NL sehingga pengguna tidak mendapatkan respons default ini. Untuk mengatur chatbot agar memahami input NL, temukan dan klik opsi Siapkan AI di menu paling kiri halaman. Di sinilah kita akan menentukan template masukan NL dan respons chatbot yang sesuai:



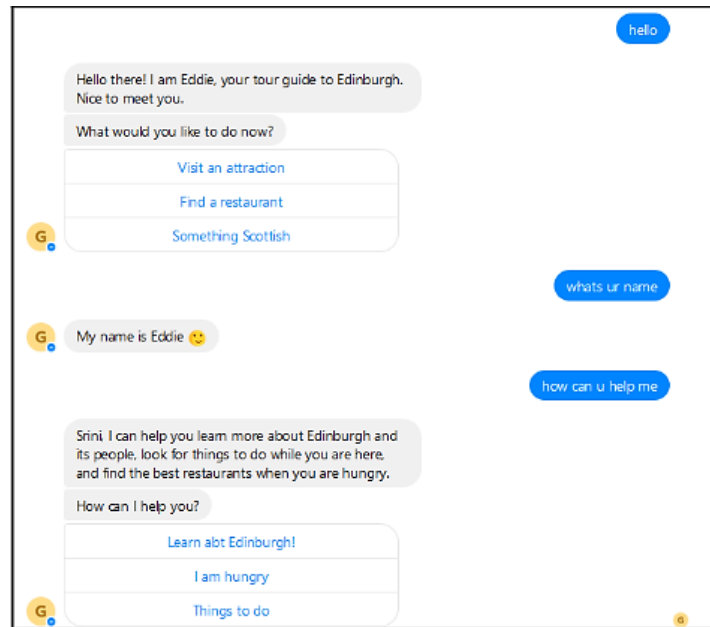
Gambar 3.33 Menentukan template untuk chat bot

Klik **TAMBAHKAN ATURAN AI**. Anda akan melihat dua bidang. Ketik ucapan pengguna dan tanggapan bot di masing-masing bidang:



Gambar 3.34 Tampilan tambahan aturan chat bot

Respons bot dapat berupa teks atau blok. Dengan memberikan nama blok, chatbot dapat dialihkan ke blok yang sesuai ketika pengguna memilih untuk mengetik tanggapan daripada menekan tombol. Setelah menambahkan beberapa aturan, kembali ke Messenger dan coba bot dengan input NL:



Gambar 3.35 respon ujicoba pada chat bot

Anda juga dapat menambahkan elemen ketidakpastian dan keacakan dengan memilih tanda acak pada respons bot dan menambahkan lebih dari satu respons. Bot kemudian akan memilih satu respons secara acak.

3.6 Pemrosesan Bagian Belakang

Tugas percakapan biasanya perlu didukung oleh tugas backend. Misalnya, memesan meja di restoran bukan sekadar percakapan, namun juga melibatkan tindakan memesan meja. Ini adalah tugas backend di mana informasi mengenai pemesanan dikirim ke server pemesanan sebagai permintaan HTTP. Di bagian ini, kita akan mempelajari cara menggunakan kartu API JSON untuk mengaktifkan tugas backend.

Sebelum kita mulai menggunakan kartu API JSON, kita memerlukan URL yang dapat mengambil beberapa parameter dan menjalankan tugas backend. Mari kita buat layanan backend tiruan yang dapat memesan meja berdasarkan nama restoran, jumlah orang, dan waktu. Untuk melakukan ini, kita harus membangun aplikasi web Node.js dan menyimpannya di cloud:

1. Buat direktori bernama Eddie-bot. Di direktori ini, kita memerlukan tiga file: *index.js*, *package.json*, dan *Procfile*.
2. Buat file paket, *package.json*, yang mendeklarasikan paket-paket yang diperlukan untuk aplikasi, seperti yang ditunjukkan di sini:

```
{
```

```

"name": "eddie-server",
"version": "1.0.0"
"description": "eddie- chatfuel",
"main": "index.js",
"scripts": {
"test": " echo \" Error: no test specified\"&& exit 1"
},
"author": "srini Janarthanam",
"license": "ISC"
"dependencies": {
  " body - parser" : "^1.15.2" ,
  "express": "^4.14.0"
  " request" : "^2.72.0"
}
}

```

3. Buat Profil. Procfile adalah file khusus yang digunakan oleh layanan cloud kami, yang disebut Heroku, untuk memberi tahu aplikasi dynos perintah apa yang harus dijalankan untuk memulai aplikasi. Itu harus berisi kode berikut: [web:node index.js](#)
4. Buat file index.js. Terakhir, kita memerlukan program yang akan memproses permintaan di backend. Rekatkan kode berikut:

```

// EddieBot webhooks
Const express = require ('express')
Const bodyparser = require ('body - parser')
Const app = express()
App.set ( 'port', (process.env.PORT ||5000))

//Process application/x-www-form-urlencoded
App.use (bodyParser.urlencoded ( {extended: false} ) )

//Process application/json
App.use(bodyParser.json( ) )
App.use (express.static ('public' ) )

// spin up the server
App.listen (app.get('port') , function () {
  Console.log ('running on port', app.get ('port') )
} )

// Index route
App.get('/', function (req,res) {
  res.send ('Hello world , I am EddieBot webhook. ` ` )
} )

app.post ('/booktable/' , function (req,res) {
  console.log (JSON.stringify ( req.body) ) ;
// YOUR BOOKING CODE GOES HERE !!

  Var out = {

```

```

    " messages" : {
      {"text" : " Thanks for your booking !"},
      {"text" : "See you soon !!" }
    }
    Var outString =JSON.stringify(out) ;
    Console.log(`Out: ` + outString) ;
    Res.send (outstring) ;
  } }

```

Pada program sebelumnya, kita memiliki pegangan yang disebut `booktable`, yang akan kita panggil ketika pengguna ingin memesan meja di restoran tertentu. Untuk menangani ini, kami akan mengirimkan sejumlah parameter yang diperlukan untuk melakukan pemesanan meja seperti nama restoran, jumlah orang, dan waktu. Setelah pemesanan dilakukan, aplikasi mengembalikan pesan sukses ke bot, yang kemudian diteruskan ke pengguna. Pada kode sebelumnya, kami sebenarnya tidak melakukan pemesanan. Kami hanya berasumsi bahwa pemesanan telah dilakukan.

5. Agar aplikasi ini dapat beroperasi, kita perlu mendorongnya ke server cloud. Untuk melakukannya di Heroku, ketikkan perintah berikut di konsol:

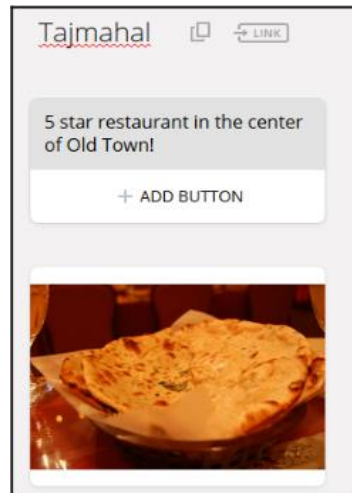
```

C:\Eddie-bot> git init
C:\Eddie-bot> git add
C:\Eddie-bot> git commit -m "Backend Eddie Bot v1"
C:\Eddie-bot> Heroku create eddie-bot-backend
C:\Eddie-bot> git push Heroku master

```

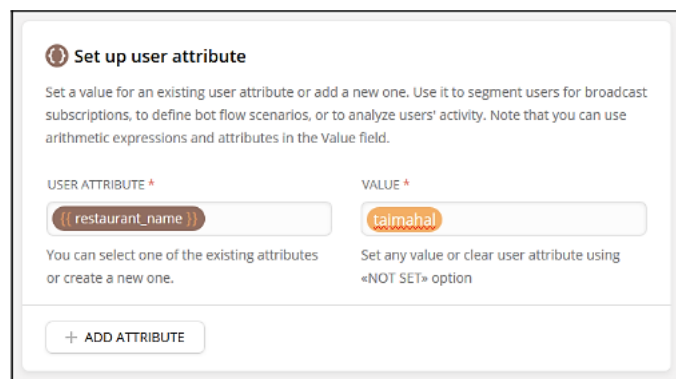
Perintah terakhir mendorong aplikasi ke cloud, membangunnya, dan menjalankan program `index.js` yang memulai server web. Kita akan mempelajari lebih lanjut tentang cara membuat aplikasi web di Node.js dan Java di bab selanjutnya. Sekarang, kita dapat memanggil URL `https://eddie-bot-backend.herokuapp.com/booktable`, dengan parameter yang diperlukan, untuk melakukan pemesanan.

6. Sekarang server pemesanan tiruan sudah diatur, mari kita panggil dan lakukan pemesanan dari chatbot. Ini melibatkan pembuatan alur menggunakan sejumlah jenis kartu. Anggaphlah pengguna mengklik tombol Informasi lebih lanjut di restoran tertentu, Tajmahal, misalnya. Ini perlu mengarahkan pengguna ke blok tempat kami menampilkan informasi tentang restoran seperti masakan, menu, dan lokasi, dengan opsi untuk memesan meja jika pengguna tertarik. Oleh karena itu, mari kita membangun blok dasar untuk Tajmahal:



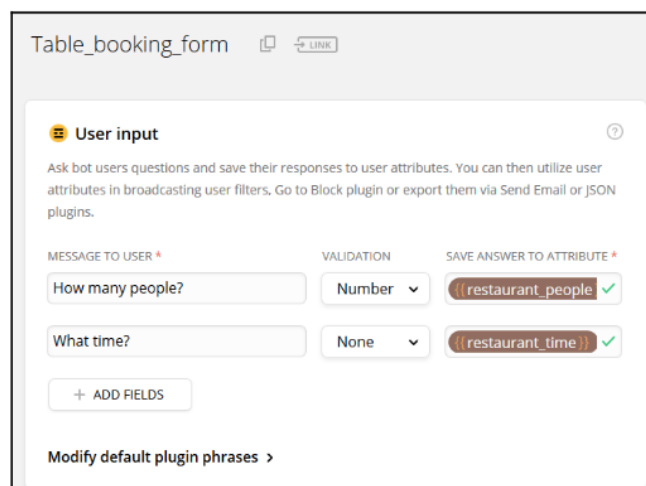
Gambar 3.36 menambahkan blok dasar Taj Mahal

7. Kita perlu menambahkan beberapa kartu lagi ke blok ini. Untuk mengatur konteksnya, kita perlu menambahkan kartu atribut pengguna tempat kita mengatur nama restoran:



Gambar 3.37 Menambahkan kartu atribut pengguna

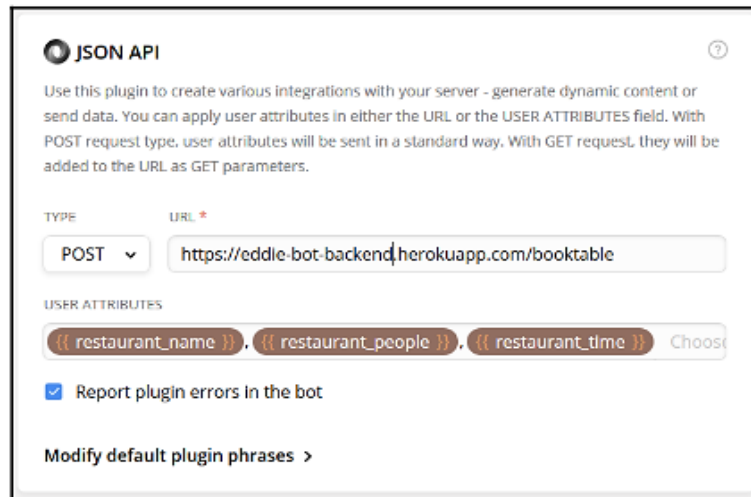
8. Kami kemudian mentransfer percakapan ke blok pemesanan meja umum di mana informasi lain yang diperlukan dapat dikumpulkan. Untuk melakukan ini, mari buat blok Table_booking_form baru:



Gambar 3.38 tampilan form pemesanan meja

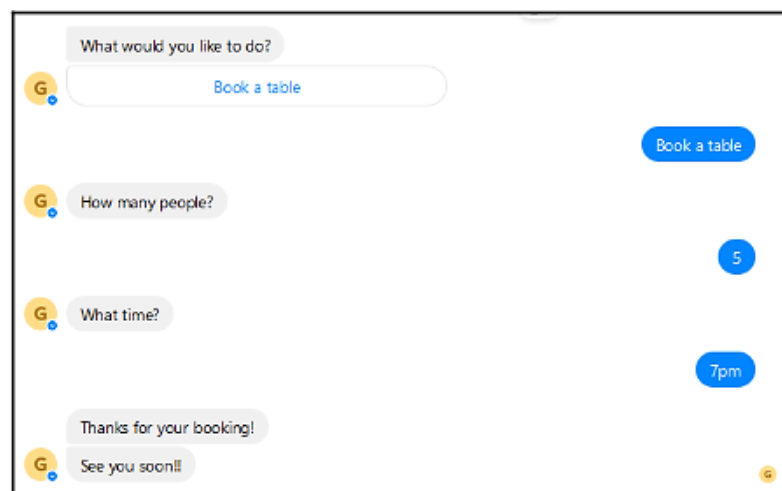
Di blok ini, kami telah menambahkan kartu masukan pengguna di mana informasi mengenai jumlah orang dan waktu dicari dari pengguna. Ini disimpan dalam variabel yang sesuai.

9. Terakhir, pada blok ini, kita menambahkan kartu API JSON, yang membuat chatbot memanggil server eksternal untuk pemrosesan backend:



Gambar 3.39 menambahkan tampilan Jason API

Di kartu ini, kita perlu menyebutkan jenis permintaan (GET/POST), URL yang akan dipanggil, dan parameter yang akan dikirim. Respons dari server backend dikembalikan ke pengguna. Mari kita uji di Messenger:



Gambar 3.40 tampilan respon di messenger

Perhatikan bagaimana pesan teks yang dikembalikan dari server backend berakhir di Messenger. Respons dari server backend harus dalam format tertentu. Di index.js, kami mengirim kembali pesan teks sederhana. Namun, jenis pesan lain juga dapat dikirim. Misalnya, berikut adalah contoh bagaimana Anda dapat mengirim blok teks dengan tombol:

```

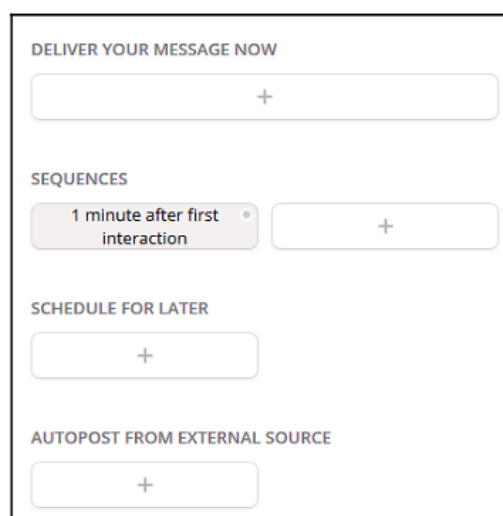
{
  "messages" : [
    {
      "attachment" : {
        "type" : "template",
        "payload" : {
          "template_type": "button" ,
          " text" : "Here are my recommendations!",
          "buttons" : [
            {
              "type" : "show_block" ,
              "block_name" : "TajMahal"
              " title " : " TajMahal (5starts)"
            } ,
            {
              "type" : "show_block" ,
              "block_name" : "TSpicyTandoori" ,
              "title" : "Spicy Tandoori ( 4 starts) "
            }
          ]
        }
      }
    }
  ]
}

```

Jelajahi jenis pesan lain yang dapat Anda kirim untuk membuat kartu percakapan yang lebih dinamis di <https://help.chatfuel.com/facebook-messenger/plugins/json-plugin/>.

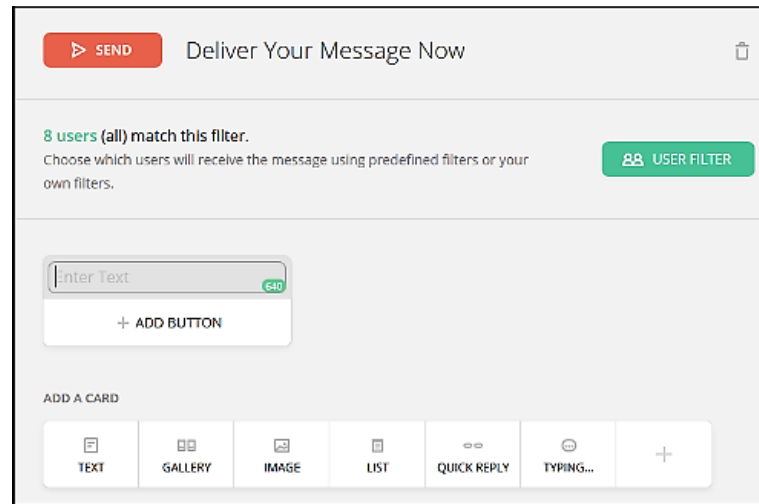
Penyiaran

Fitur menarik lainnya yang disediakan Chatfuel adalah kemampuan mengirim pesan ke lebih dari satu pengguna kapan saja. Ini adalah fasilitas yang dapat Anda gunakan untuk memulai kembali percakapan dengan pengguna atau mengirimkan notifikasi secara berkala. Untuk menggunakan fitur ini, *klik opsi Siaran di menu utama*:



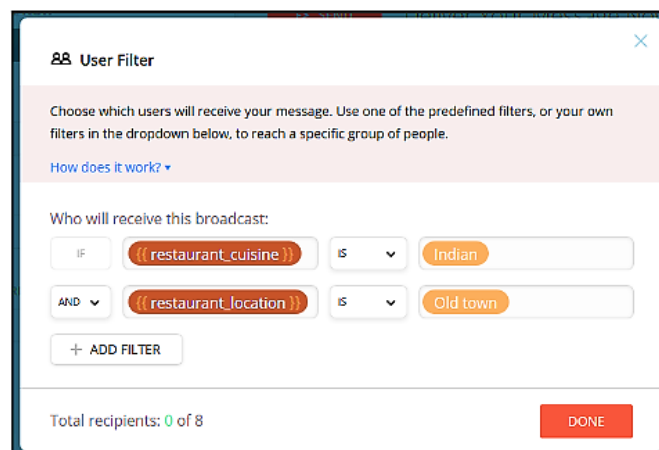
Gambar 3.41 Tampilan siaran menu

Anda akan melihat empat opsi. *Ayo pilih KIRIM PESAN ANDA SEKARANG*. Ada tiga hal yang harus dilakukan buat pesan untuk dikirim, *pilih pengguna*, dan *tekan KIRIM*:



Gambar 3.42 Tampilan untuk pengiriman pesan

Klik *FILTER PENGGUNA* untuk memilih subkumpulan pengguna yang ingin Anda kirim pesan. Ini dilakukan dengan menggunakan variabel dan nilai. Misalnya, Anda dapat memilih untuk mengirim pesan ke semua pengguna yang mengatakan bahwa mereka menyukai masakan India (yaitu, `{{restaurant_cuisine}} = 'India'`) dan bahwa mereka berada di Kota Tua (`{{restaurant_location}} = ' Kota Tua'`). Pesan ini bisa berupa penawaran diskon di restoran India di kota:

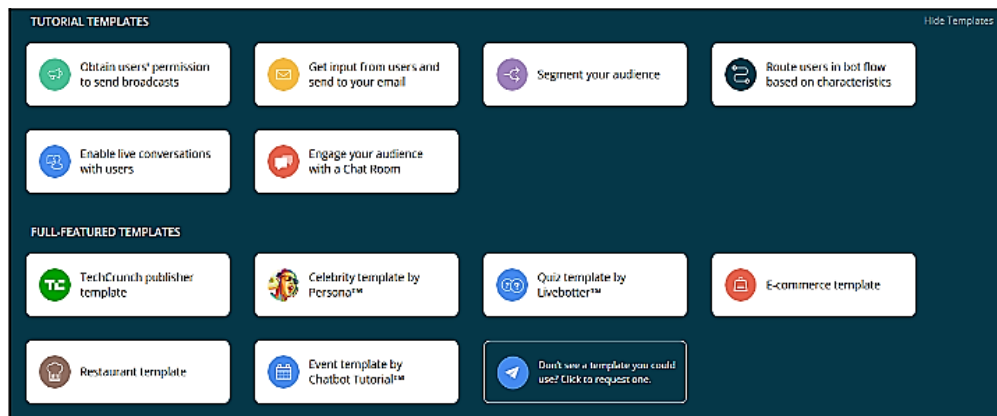


Gambar 3.43 Penawaran restoran

Setelah memilih grup pengguna target, buat blok pesan dan tekan KIRIM. Pesan ini akan segera dikirim ke semua pengguna yang memenuhi syarat. Anda juga dapat mengirim pesan ke grup pengguna target pada waktu yang dijadwalkan, mengurutkan pesan otomatis setelah interaksi pengguna, dan memposting dari sumber eksternal, serta menggunakan fitur siaran.

3.7 Template Bot

Chatbots dapat dibuat dari template. Ini adalah chatbot dasar yang telah dirancang sebelumnya yang dapat dengan mudah diedit dan diubah menjadi chatbot yang Anda perlukan. Ada sejumlah templat untuk Anda mulai. Untuk melakukan ini, buka <https://dashboard.chatfuel.com/#/bots>. Di bawah bagian TEMPLATE TUTORIAL, klik Lihat Semua Template:



Gambar 3.44 tampilan chatfuel

Chatfuel menyediakan sejumlah template dengan beragam kasus penggunaan seperti restoran, asisten selebriti, kuis, dan e-commerce. Jelajahi lebih lanjut untuk mempelajari bagaimana percakapan disusun dan bagaimana berbagai alat digunakan untuk mendukung percakapan tersebut.

Analisis

Chatfuel juga menyediakan analisis pada chatbot Anda. Untuk mengakses analitik, buka dasbor chatbot Anda dan klik Analisis di menu utama. Halaman analitik merangkum statistik penggunaan bot dalam hal pertumbuhan pengguna, aktivitas pengguna, dan retensi pengguna. Hal ini juga menyoroti pola penggunaan, seperti blok yang populer di kalangan pengguna dan masalah yang dihadapi AI dalam hal memahami ucapan bahasa pengguna. Terakhir, ini juga memberi tahu Anda dari belahan dunia mana pengguna Anda berasal.

Dalam bab ini, kita memulai perjalanan menuju pembuatan chatbot yang mengagumkan. Saya harap, dengan menggunakan Chatfuel, Anda telah membuat bot pertama Anda. Dengan menggunakan pemandu tur sebagai kasus penggunaan, kami menjelajahi berbagai topik desain dan pengembangan chatbot desain alur percakapan, blok, jenis konten pesan, menavigasi alur percakapan, memahami ucapan dasar bahasa alami, dan banyak lagi. Kami juga telah mempelajari berbagai elemen dasar yang disediakan oleh Chatfuel untuk membangun, memantau, dan mengelola chatbot.

Pada bab-bab berikutnya, kita akan mengeksplorasi penggunaan berbagai kasus penggunaan lain dan topik lebih lanjut dalam membangun chatbot, seperti memahami nuansa ucapan bahasa alami dan mengelola konteks percakapan yang lebih kompleks.

BAB 4

MEMBANGUN CHATBOTS DENGAN CARA MUDAH

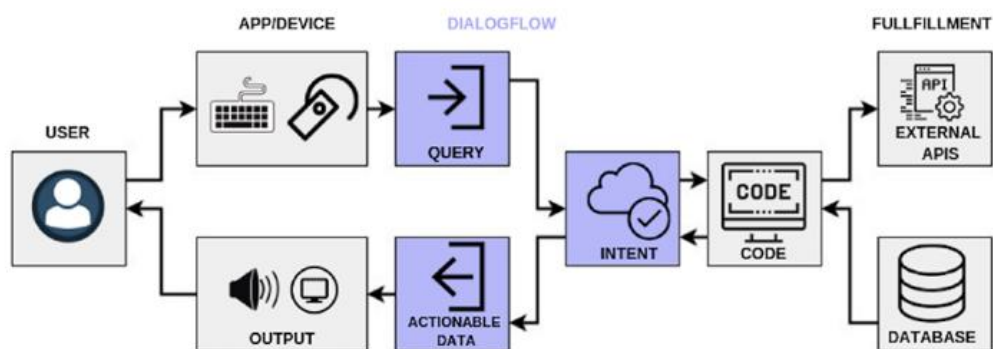
Membangun chatbot dengan cara mudah telah ditulis dengan mengingat bahwa terkadang Anda tidak ingin membangun semuanya dari awal dan hanya ingin menyelesaikan sesuatu. Bab ini tidak mengharuskan Anda melakukan banyak pengkodean tetapi tetap memberi Anda gambaran yang adil tentang cara membuat chatbot dengan cara yang mudah dan menjadikannya publik.

Alasan mengapa bab ini menjadi lebih penting dalam pembelajaran membuat chatbot adalah karena dunia perangkat lunak bergerak terlalu cepat untuk diadaptasi. Terkadang kita perlu membangun aplikasi dengan sangat cepat, dan kita mencoba mencari alat yang tersedia di perpustakaan sumber terbuka yang dapat digunakan untuk membangun aplikasi dengan cepat, tanpa perlu menciptakan kembali rodanya. Terkadang kita tidak cukup pandai dalam coding untuk membangun semuanya dari awal. Meskipun kami ingin membangun aplikasi dari awal, kami tidak bisa karena kurva pembelajarannya akan sangat curam bagi seorang pemula.

Bab ini akan membantu Anda membuat chatbot dengan sangat cepat dan menjadikannya publik agar dapat digunakan oleh dunia. Kami akan mencoba alat yang sebelumnya dikenal sebagai Api.ai. Sekarang dikenal sebagai Dialogflow.

4.1 Pengantar Dialogflow

Dialogflow memberi pengguna metode baru untuk berinteraksi dengan produk mereka dengan membangun antarmuka percakapan berbasis suara dan teks yang menarik, seperti aplikasi suara dan chatbots. Dialogflow didukung oleh AI. Ini membantu Anda terhubung dengan pengguna di situs web, aplikasi seluler, Asisten Google, Amazon Alexa, Facebook Messenger, dan platform serta perangkat populer lainnya. Diagram dari Dialogflow berikut menunjukkan cara mereka menangani permintaan pengguna.



Gambar 4-1. Diagram kerja arsitektur Dialogflow

Inilah yang terjadi:

1. Pengguna berbicara ke perangkat input.

2. Permintaan pengguna masuk ke mesin Dialogflow.
3. Dialogflow mencoba mengenali maksudnya.
4. Berdasarkan maksudnya, dilakukan pemenuhan dan data dikembalikan dari database.
5. Respon dikembalikan ke maksud.
6. Respon diubah menjadi data yang dapat ditindaklanjuti.
7. Permintaan informasi pengguna diberikan kembali ke perangkat keluaran. Ada konsep agen di Dialogflow yang paling tepat digambarkan sebagai Natural

Modul Pemahaman Bahasa (NLU). Ini dapat disertakan dalam aplikasi, produk, atau layanan Anda dan mengubah permintaan alami pengguna menjadi data yang dapat ditindaklanjuti. Transformasi ini terjadi ketika input pengguna cocok dengan salah satu maksud di dalam agen Anda.

Agen juga dapat dirancang untuk mengelola alur percakapan dengan cara tertentu. Hal ini dapat dilakukan dengan bantuan konteks, prioritas maksud, pengisian slot, tanggung jawab, dan pemenuhan melalui webhook.

4.2 Mulai Membuat Chatbot

Mengetahui apa yang telah kita pelajari sejauh ini penting karena alat dan paket gratis yang tersedia dalam sumber terbuka tidak selalu membantu dalam membangun aplikasi chatbot yang lengkap.

Seringkali, Anda mungkin menghadapi situasi ketika Anda ingin membangun semuanya sendiri sehingga Anda memiliki kontrol lebih besar atas aplikasi Anda. Kita akan mempelajarinya di bab berikutnya dan juga menggunakan teknik NLP yang telah dipelajari sebelumnya. Bab ini membahas tentang membuat chatbot sebagai bukti konsep dan membuatnya siap digunakan dunia dengan pemrograman minimal atau tanpa pengalaman pemrograman.

Membangun Chatbot Pemesanan Makanan

Kami akan membuat chatbot dengan bantuan Dialogflow untuk restoran tertentu. Sebut saja OnlineEatsBot. Singkatnya kita bisa menyebutnya produk OnlineEats. Anda dapat memilih kasus penggunaan lain yang ingin Anda buat chatbotnya. Untuk bab ini kita akan membuat chatbot pemesanan makanan.

Memutuskan Ruang Lingkup

Mari kita tentukan cakupan chatbot ini—yaitu, apa yang dapat dilakukannya dan sejauh mana.

- Harus dapat menyapa pengguna secara dinamis.
- Ia harus dapat memahami item menu dan jumlah yang diminta.
- Chatbot harus dapat melakukan pemesanan atas nama pengguna.
- Memberikan status pesanan kepada pengguna saat ditanya.

Maksud Daftar

Di sini kami mencantumkan maksud yang kami ingin agar chatbot kami latih sehingga dapat memahami maksud tersebut ketika ditanya oleh pengguna.

- Niat selamat datang default: saat pengguna mengirim pesan ke chatbot

- Maksud memesan makanan: ketika pengguna meminta bot untuk memesan makanan
- Maksud deskripsi barang: ketika pengguna memberitahukan barang dan jumlah apa yang diinginkannya
- Status pesanan: ketika pengguna ingin mengetahui status pesannya
- Order_ID: bot perlu memahami ID pesanan pengguna untuk dilacak.
- Pengguna berterima kasih: ketika pengguna mengucapkan terima kasih kepada bot

Entitas Pencatatan

Kami akan mencoba membuat daftar semua kemungkinan entitas untuk semua maksud di sini. Nanti di bab ini kita akan melihat definisinya untuk tujuan apa pun penerapannya.

Entitas

- food_items: makanan apa yang ingin dipesan pengguna?
- kuantitas: berapa jumlah item makanan yang ingin dipesan oleh pengguna?
- order_id: order_id dari pemesanan tempat untuk pengguna

4.3 Membangun Chatbot Pemesanan Makanan

Mari kita juga menetapkan ekspektasi chatbot dalam hal hal-hal minimum yang dapat kita lihat dilakukannya. Untuk itu, mari buat skrip percakapan antara chatbot kita dan pengguna. Hal ini membantu kami tetap berpegang pada rencana untuk menyiapkan chatbot dasar yang berfungsi, dengan asumsi pengguna melakukan percakapan dengan cara yang baik dan benar.

Chatbot dan Skrip Percakapan Pengguna:

Pengguna : Halo

OnlineEatsBot : Halo dan Selamat Datang! Apa yang bisa saya bantu?

Pengguna : Saya ingin memesan makanan Cina.

OnlineEatsBot : Tentu, apa yang ingin Anda pesan hari ini?

Pengguna : Satu pangsit ayam dan dua lumpia.

OnlineEatsBot : Selesai. Jumlah akhir Anda adalah XYZ, dan pesanan Anda dilakukan untuk 1 pangsit ayam dan 2 lumpia.

Pengguna : Saya belum menerima pesanan saya. Dimana makanan saya?

OnlineEatsBot : Bisakah Anda membantu saya dengan ID pesanan Anda?

Pengguna : 123456

OnlineEatsBot : Status Pesanan untuk ID Pesanan: 123456. Petugas pengiriman ada di wilayah Anda, makanan Anda akan tiba dalam waktu sekitar 5 menit.

Pengguna : Terima kasih.

OnlineEatsBot : Terima kasih banyak atas kerja sama Anda.

Sekarang kita telah menulis skrip dasar yang harus diikuti untuk membangun chatbot, sekarang kita akan mendalami Dialogflow.

Memulai Dialogflow

Mari ikuti langkah-langkah berikut untuk membuat akun di Dialogflow lalu membuat agen. (Agen hanyalah nama lain dari chatbot.)

1. Buat akun di <https://dialogflow.com>, dan masuk ke akun tersebut.
2. Buat agen.

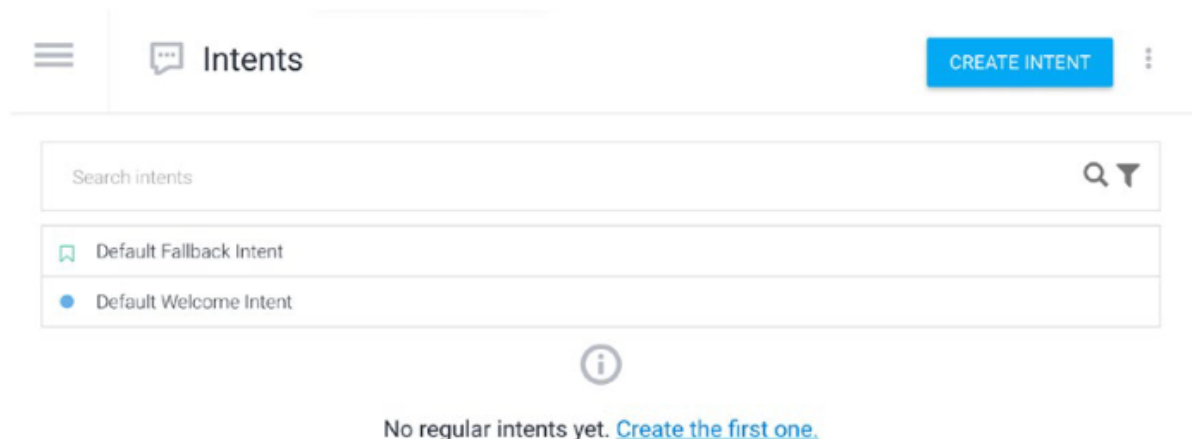
The screenshot shows the 'CREATE' screen in Dialogflow. At the top, the agent name 'OnlineEatsBot' is entered in a text field, followed by a blue 'CREATE' button and a vertical ellipsis menu icon. Below this, there are three main configuration sections:

- DEFAULT LANGUAGE**: Set to 'English – en'. A sub-note reads: 'Primary language for your agent. Other languages can be added later.'
- DEFAULT TIME ZONE**: Set to '(GMT+6:00) Asia/Almaty'. A sub-note reads: 'Date and time requests are resolved using this timezone.'
- GOOGLE PROJECT**: Set to 'Create a new Google project'. A sub-note reads: 'Enables Cloud functions, Actions on Google and permissions management.'

Gambar 4.2 Membuat agen baru di Dialogflow

Masukkan detailnya, seperti nama agen, zona waktu, bahasa default, dan Proyek Google yang ingin Anda pilih atau buat proyek Google baru.

3. Ciptakan niat.
4. Jika Anda melihat Gambar 4.3, Anda akan melihat bahwa kita sudah diberikan dua maksud.
 - Niat penggantian default: Niat penggantian dipicu jika masukan pengguna tidak cocok dengan maksud biasa atau obrolan ringan bawaan yang diaktifkan. Saat Anda membuat agen baru, maksud fallback default dibuat secara otomatis. Anda dapat mengubah atau menghapusnya jika Anda mau.
 - Niat selamat datang default: Kami dapat memperluas niat selamat datang ini untuk chatbot kami sendiri. Anda harus menambahkan beberapa ekspresi pengguna dan respons default Anda sendiri.

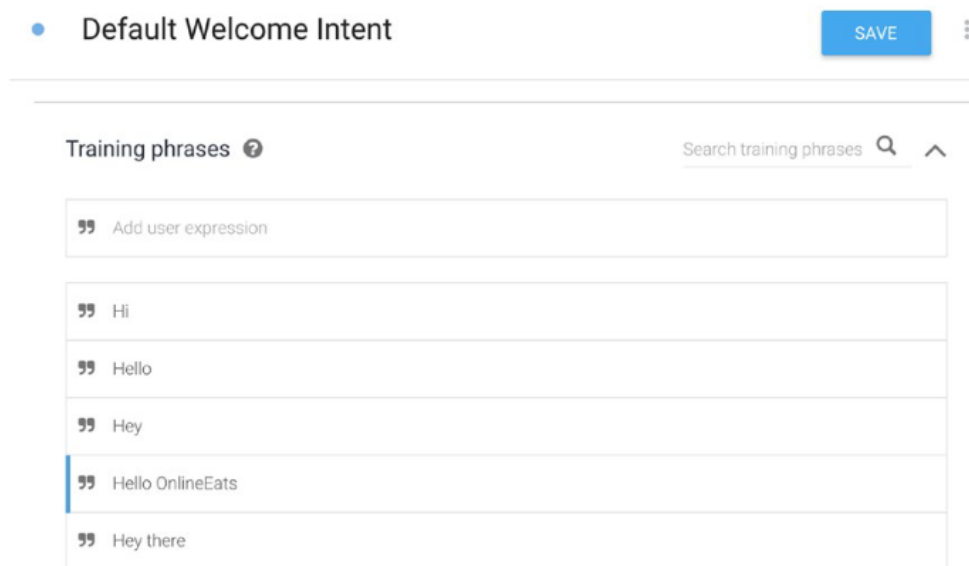


Gambar 4.3. Membuat maksud di Dialogflow

Sebelum kita membuat maksud kita sendiri, pertama-tama mari tambahkan beberapa ucapan dalam maksud selamat datang default dan persiapkan dengan menggunakan langkah-langkah berikut:

1. Klik pada maksud selamat datang default.
2. Tambahkan ekspresi pengguna Anda sendiri di Frasa Pelatihan.
3. Klik SIMPAN.

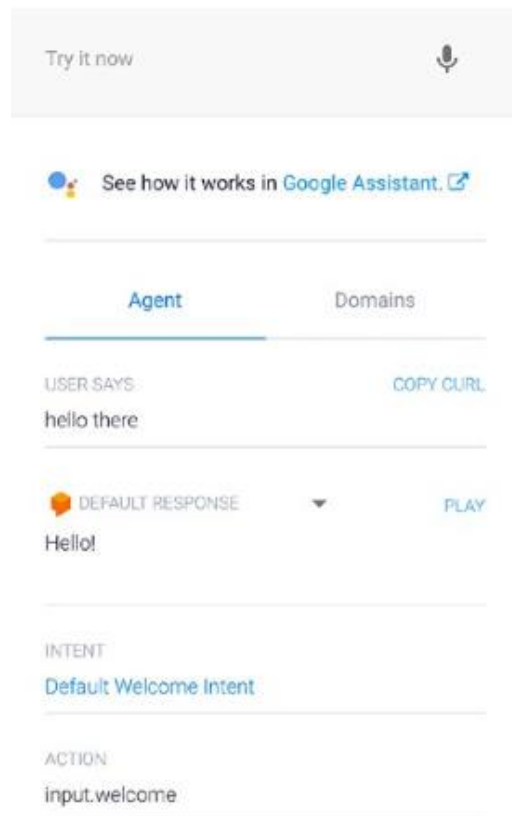
Saat kita mengklik simpan, model pembelajaran mesin di belakang layar menjalankan dan melatih data yang kita berikan (yaitu ekspresi pengguna).



Gambar 4.4. Mendefinisikan maksud selamat datang atau salam default di Dialogflow

Melatih data berarti membiarkan mesin memahami maksudnya berdasarkan data yang kami berikan dan mampu memprediksi kapan kami memberikan data baru ke dalamnya. Misalnya saja jika kita perhatikan Gambar 4.4, di mana kita telah mendefinisikan lima ekspresi pengguna yang sudah diketahui oleh mesin sebagai “niat selamat datang”, bagaimana jika pengguna mengatakan “Halo,” yang tidak didefinisikan di mana pun? Mesin akan tetap mengkategorikan “Halo” sebagai maksud selamat datang default, karena fitur yang digunakan dalam pelatihan dan mesin untuk maksud selamat datang serupa dalam ekspresi pengguna baru.

Mari kita coba melihat apakah niat selamat datang ini berhasil untuk kita. Dengan Dialogflow kita bisa melakukannya di dashboard itu sendiri. Lihat Gambar 4-5.



Gambar 4.5. Menguji maksud selamat datang di Dialogflow

Hal-hal yang Perlu Diingat Saat Membuat Intent

Mari kita lihat beberapa poin penting untuk diketahui saat membuat maksud di Dialogflow.

- Intent Dialogflow juga memiliki fasilitas untuk memiliki respon default dari setiap maksud. Respons default adalah respons yang diberikan kembali kepada pengguna setiap kali maksud tersebut dikenali. Dalam contoh kita, saat pengguna mengatakan “Halo”, kita mendapatkan “Halo!” sebagai respon dari bot.
- Jika diinginkan, Anda dapat menambahkan lebih banyak respons atau menghapus respons yang sudah ada, memiliki lebih dari satu respons akan membuat bot terlihat realistis sehingga tidak selalu membalas dengan respons yang sama dan terasa manusiawi bagi pengguna yang berbicara dengan bot tersebut.
- Intent di Dialogflow juga memiliki kemampuan untuk ditandai sebagai akhir percakapan. Dengan kata lain, Anda dapat membiarkan bot berasumsi bahwa pengguna tidak akan lagi berpartisipasi dalam percakapan, dan bot dapat melakukan tindakan yang diperlukan, berdasarkan informasi ini, untuk mengakhiri percakapan.

Membuat Maksud dan Menambahkan Ucapan

Sekarang kita telah membuat maksud selamat datang, mari buat maksud pesanan. Saya menamakannya tempat_pesanan_maksud. Berikut ini adalah ekspresi pengguna yang saya masukkan:

- *saya ingin makanan*
- *Saya ingin memesan makanan secepatnya*

- *Bisakah Anda mengambil pesanan makanan saya?*
- *Tolong ambil pesanan saya*
- *Saya ingin memesan makanan Cina.*
- *Saya ingin memesan*
- *Maukah Anda membantu saya memesan makanan?*
- *Bisakah kamu memesankan makanan untukku?*
- *Saya ingin memesan makanan*
- *Saya ingin memesan makanan Thailand*
- *Saya ingin memesan makanan Cina*

Sekarang, kami telah membangun maksud untuk mengidentifikasi ekspresi pengguna yang disebutkan di atas atau ekspresi pengguna terkait. Sekarang, saatnya menambahkan respons kembali ke pengguna menggunakan respons default pada maksud.

Menambahkan Respon Default ke Intent

Kami akan menambahkan tiga kemungkinan respons yang akan diberikan kembali kepada pengguna setelah `place_order_intent` ditemukan. Apa yang ingin Anda pesan hari ini? Pastinya, Apa yang ingin Anda miliki hari ini?

Tentu saja, saya akan mencoba membantu Anda dalam hal itu. Apa yang ingin kamu makan hari ini?. Sekarang, langkah selanjutnya adalah menunggu pengguna memasukkan item yang diinginkannya dan mengurai item tersebut.

Sekarang kita akan membuat maksud baru yang memberi tahu kita apa yang sebenarnya ingin dipesan pengguna (yaitu, item makanan). Kami membuat maksud baru bernama `items_description` Pertama, kami menambahkan ekspresi pengguna standar kami. Satu pangsit ayam dan dua lumpia.

Saat kita menambahkan ekspresi pengguna maka kita dapat memilih kata tertentu yang ingin kita tentukan sebagai entitas maksud. Ini bisa berupa kuantitas, tanggal atau waktu, lokasi, dll., yang telah ditentukan sebelumnya, namun kita dapat membuat entitas kita sendiri dengan mengklik tombol Buat Baru di kanan bawah setelah kita mendapatkan kotak pop-up. Soroti kata dalam ucapan yang ingin Anda jadikan kata yang dipilih itu sebagai entitas. Setelah itu, akan terbuka kotak pop-up untuk membuat entitas kita sendiri.

Dalam contoh ini, kita harus bisa mengurai data dalam format yang mudah dibaca sehingga kita bisa menggunakannya menggunakan bahasa pemrograman apa pun. Format JSON adalah format terbaik yang dapat kita miliki dalam aplikasi lintas platform saat ini. Dialogflow mengembalikan data dalam format JSON secara default, yang dapat diurai agar terlihat seperti kode berikut. Selalu disarankan untuk menjaga data Anda seminimal mungkin; jangan membebani respons API dengan memberikan terlalu banyak data. Ingat semua ini membutuhkan biaya dalam skala besar.

```

{
  "food_items": {
    "chicken dumpling": 1,
    "Spring rolls": 2
  }
}

```

Item Deskripsi Maksud dan Entitas Milik

Kita dapat memilih Satu dan Dua dan mendefinisikannya sebagai @sys.number, yang tidak lain adalah tipe datanya. Kami akan membuat entitas baru bernama food_items_entity untuk mengidentifikasi item makanan.

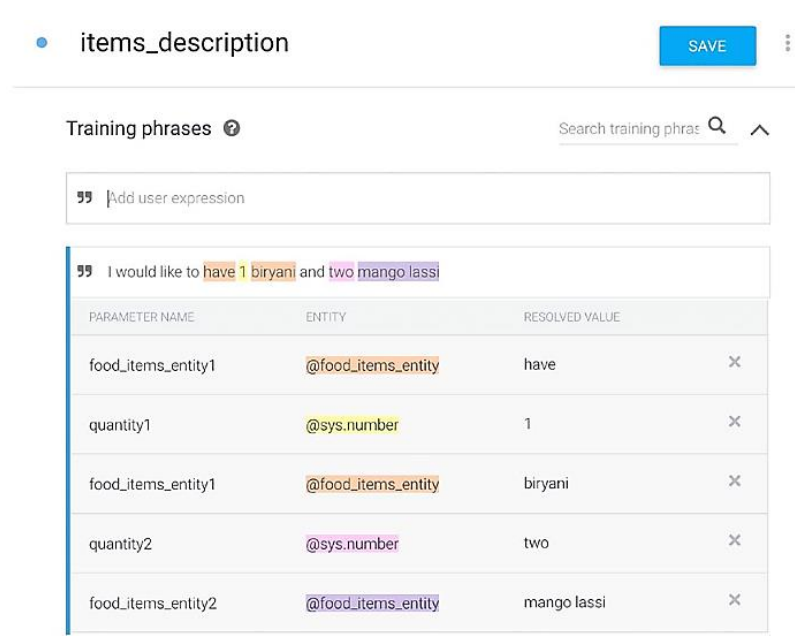
Jika Anda melihat Gambar 4.6, Anda akan menemukan bahwa kita memiliki ENTITY yang diberi nama sebagai entitas_item_makanan, tetapi ketika kita memilih kata-katanya, maka kita menamai parameternya sebagai entitas_item_makanan1 dan entitas_barang_makanan2; Hal ini serupa dengan kuantitas makanan, yaitu angka yang parameter pertama dan kedua kita beri nama masing-masing sebagai kuantitas1 dan kuantitas2.

PARAMETER NAME	ENTITY	RESOLVED VALUE
quantity1	@sys.number	One
food_items_entity1	@food_items_entity	chicken dumpling
quantity2	@sys.number	two
food_items_entity2	@food_items_entity	spring rolls

Gambar 4.6. Maksud deskripsi item

Apa yang kami definisikan di sini akan membantu kami memahami respons JSON, yang akan kami dapatkan setelah niat dipicu. Kita harus memiliki semua nilai-nilai ini di sana untuk bergerak maju dengan aliran chatbot. Jadi, pilih seluruh kata atau kombinasi kata dan klik Buat Baru. Layar baru akan muncul untuk membuat entitas; cukup masukkan nama untuk entitas baru ini dan simpan. Sekarang, kembali ke maksud kita untuk items_description dan Anda akan melihat sesuatu seperti Gambar 4.6. Terus tambahkan lebih banyak ekspresi pengguna dalam frasa pelatihan, dan terus definisikan entitas di dalamnya. Kami telah menambahkan empat ucapan sejauh ini, dan inilah tampilannya. Kami akan menambahkan sebanyak mungkin agar akurasi klasifikasi niat agen kami lebih baik. Dialogflow juga memiliki fitur untuk membagikan data pelatihan agen. Data pelatihan yang digunakan dalam buku ini dapat diakses melalui website Apress, *Percakapan Antarmuka Pengguna (User Interface) Dalam Chatbot – Dr. Budi Raharjo*

<https://github.com/Apress/building-chatbots-with-python>. Seperti yang Anda lihat pada Gambar 4.7 di mana kami mencoba menambahkan beberapa contoh lagi dalam maksud deskripsi item di agen alur dialog kami.



Gambar 4.7. Menambahkan lebih banyak ucapan dalam maksud deskripsi item

Sekarang, pada titik ini kami telah menyimpan niat kami, dan agen kami telah selesai melatih modelnya. Jika kita memasukkan kalimat berikut di sisi kanan, kita akan melihat respon JSON berikut:

Satu pangsit ayam dan dua lumpia

```
{
  "id": "e8cf4a44-6ec9-49ae-9da8-a5542a80d742",
  "timestamp": "2018-04-01T21:22:42.846Z",
  "lang": "en",
  "result": {
    "source": "agent",
    "resolvedQuery": "One chicken dumpling and two spring rolls",
    "action": "",
    "actionIncomplete": false,
    "parameters": {
      "quantity1": 1,
      "food_items_entity1": "chicken dumpling",
      "quantity2": 2,
      "food_items_entity2": "spring rolls"
    }
  },
}
```

```

"contexts": [],
"metadata": {
  "intentId": "0b478407-1b37-4f9a-8779-1866714dd44f",
  "webhookUsed": "false",
  "webhookForSlotFillingUsed": "false",
  "intentName": "items_description"
},
"fulfillment": {
  "speech": "",
  "messages": [
    {
      "type": 0,
      "speech": ""
    }
  ]
},
"score": 1
},
"status": {
  "code": 200,
  "errorType": "success",
  "webhookTimedOut": false
},
"sessionId": "e1ee1860-06a7-4ca1-acae-f92c6e4a023e"
}

```

Jika Anda melihat bagian parameter dari respons JSON yang kita lihat

```

{
  "quantity1": 1,
  "food_items_entity1": "chicken dumpling",
  "quantity2": 2,
  "food_items_entity2": "spring rolls"
}

```

Kita dapat dengan mudah menulis beberapa kode Python untuk mengonversi JSON ke format yang kita inginkan yang telah kita diskusikan.

apakah kamu bisa melakukan ini?

Uji saja kemampuan Python Anda dan coba tulis kode yang membaca JSON seperti sebelumnya dan mengembalikan kuantitas dan item makanan miliknya dalam format JSON lain yang telah kita bahas sebelumnya.

4.4 Memahami Dan Membalas Kembali Ke Pengguna

Sekarang, percakapan berikutnya adalah membuat bot membalas kembali kepada pengguna bahwa perintahnya dipahami bersama dengan informasi baru. Informasi baru dapat berupa `order_id` yang dihasilkan, jumlah pesanan, atau perkiraan waktu pengiriman.

Hal-hal ini akan diisi di sisi server Anda, dan Anda dapat memformulasikannya dengan respons bot untuk mengembalikannya kepada pengguna.

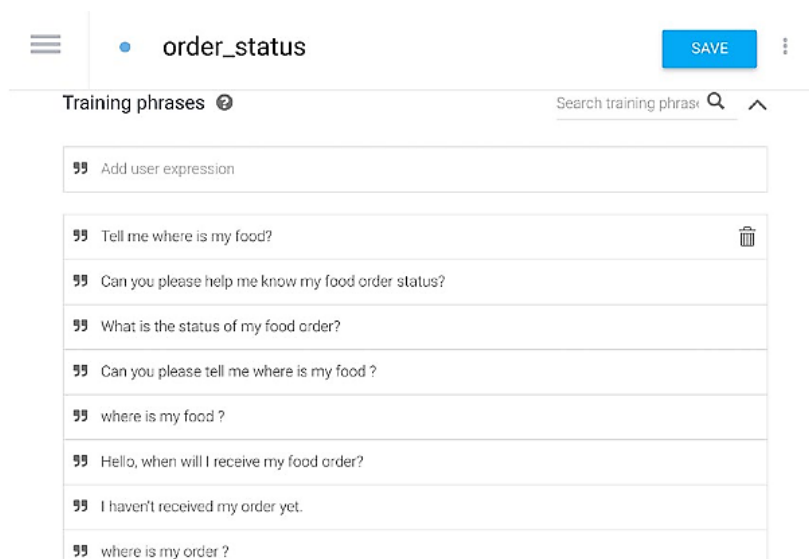
Sekarang, mari kita coba menambahkan jumlah pesanan dalam kasus kita; untuk melakukan itu, kita dapat menggunakan fitur Respons Default Dialogflow dan menambahkan ini ke dalam maksud. Mari kita hardcode jumlahnya untuk saat ini, karena jumlah ini akan bervariasi tergantung pada jenis makanan, kuantitasnya, atau restorannya. Nanti di bab ini, kita akan membahas cara membuatnya dinamis dengan memanggil API.

Hal yang menarik di sini adalah kita dapat mengakses parameter yang kita peroleh dari maksud (yaitu item makanan dan jumlahnya). Respons dapat berisi referensi ke nilai parameter. Kami akan memahaminya sebentar lagi. Jika sebuah parameter ada di tabel parameter, kita dapat menggunakan format berikut untuk merujuk ke nilainya di kolom 'Respons teks': \$parameter_name. Kita dapat menggunakan parameter ini dalam respons default sehingga bot mengonfirmasi pesanan kembali ke pengguna. Tambahkan “Selesai. Jumlah akhir Anda adalah XYZ dan pesanan Anda dilakukan sebesar \$quantity1 \$food_items_entity1 dan \$quantity2 \$food_items_entity2” sebagai responsnya.

Catatan Jika maksud kami tidak dapat mengurai item makanan atau kuantitasnya, kami perlu memberikan respons default berbeda yang meminta untuk menjelaskan apa yang tidak dapat dipahami oleh bot kami atau setidaknya mengonfirmasi. Kita telah mempelajari cara menambahkan respons default ke suatu maksud di bagian “Menambahkan Respons Default ke Maksud.”

Maksud Status Pesanan

Sekarang, mari buat maksud order_status, di mana pengguna mungkin mencoba menanyakan status pesanan setelah pesanan dilakukan. Gambar 4-8 memberikan beberapa frasa pelatihan yang kami tambahkan untuk maksud status pesanan, dan kami beri nama maksud tersebut status_pesanan.



Gambar 4.8. Membuat maksud status pesanan

Sekarang, mari kita coba beberapa status urutan acak yang menanyakan ucapan dan lihat apakah agen kita cukup cerdas untuk mengidentifikasi maksudnya. Saya mencoba, “Belum menerima makanan saya,” dan voila—agen saya benar sekali bahwa ini adalah maksud status_pesanan. Lihat ResolveQuery dan IntentName-nya di JSON pada Gambar 4-9.

JSON

```

1 - {
2   "id": "e68790f6-3d9c-4398-a7b1-5b1f6a3d0f1b",
3   "timestamp": "2018-04-01T21:45:20.386Z",
4   "lang": "en",
5   "result": {
6     "source": "agent",
7     "resolvedQuery": "Haven't received my food yet",
8     "action": "",
9     "actionIncomplete": false,
10    "parameters": {},
11    "contexts": [],
12    "metadata": {
13      "intentId": "a76ae537-b648-4e81-a03d-eca7bc84b136",
14      "webhookUsed": "false",
15      "webhookForSlotFillingUsed": "false",
16      "intentName": "order_status"
17    },
18    "fulfillment": {
19      "speech": "",
20      "messages": [
21        {

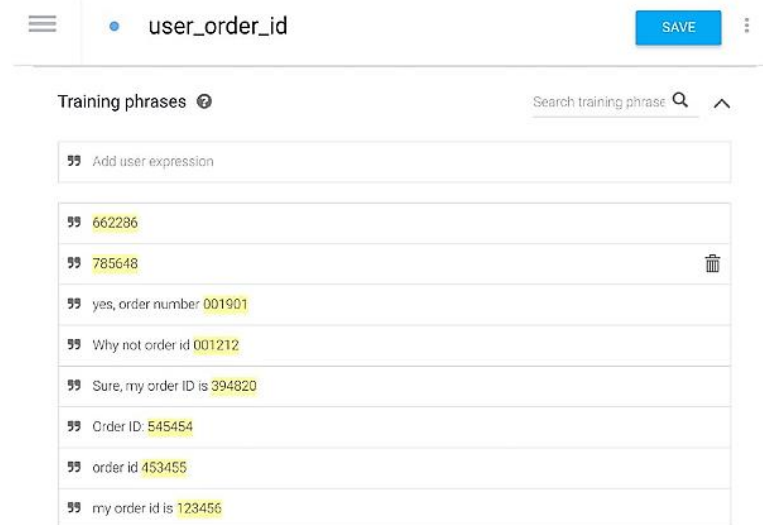
```

Gambar 4.9. Respons JSON dari Dialogflow setelah kueri diselesaikan

Maksud User_Order_ID

Sekarang, selanjutnya adalah menanyakan ID Pesanan kepada pengguna, jadi mari kita atur respons default dari maksud ini untuk menanyakan pertanyaan berikut. Bisakah Anda membantu saya dengan ID pesanan Anda? Sekarang, pengguna akan memberikan ID pesannya, dan tugas kita adalah mengidentifikasinya dan memberikan respons lagi. Jadi, untuk itu kita perlu membuat maksud lain untuk mengidentifikasi saat pengguna membicarakan ID pesanan.

Perhatikan bahwa maksud yang kami buat tidak bergantung satu sama lain. Dalam hal ini kita tahu bahwa pengguna akan memberikan ID pesanan, dan sebagian besar benar. Jika salah, Anda selalu dapat kembali ke pengguna dan bertanya lagi. Kita juga harus mencatat bahwa dalam beberapa kasus, order_id dan nomor telepon keduanya mungkin bilangan bulat. Dalam kasus seperti ini, kita perlu melakukan beberapa validasi, seperti jumlah digit pada order_id atau nomor telepon. Selain itu, berdasarkan konteks pertanyaan sebelumnya, Anda dapat mengetahui apakah pengguna memberikan order_id atau nomor telepon. Seperti yang dibahas di Bab 1, kita selalu dapat menggunakan pohon keputusan untuk pengelolaan aliran chatbot yang lebih baik. Kami juga dapat melacak secara terprogram bahwa setelah maksud order_status kami meminta ID pesanan, dan pengguna akan mengirimkan beberapa ID pesanan (nomor tertentu), yang lebih mudah diurai dalam kode daripada membuat maksud baru sama sekali. Dalam contoh ini, kita akan membuat maksud user_order_id, karena tidak ada konflik. Sekarang, kita membuat maksud baru yang disebut user_order_id. Gambar 3-10 menunjukkan tampilan maksud user_order_id kita.



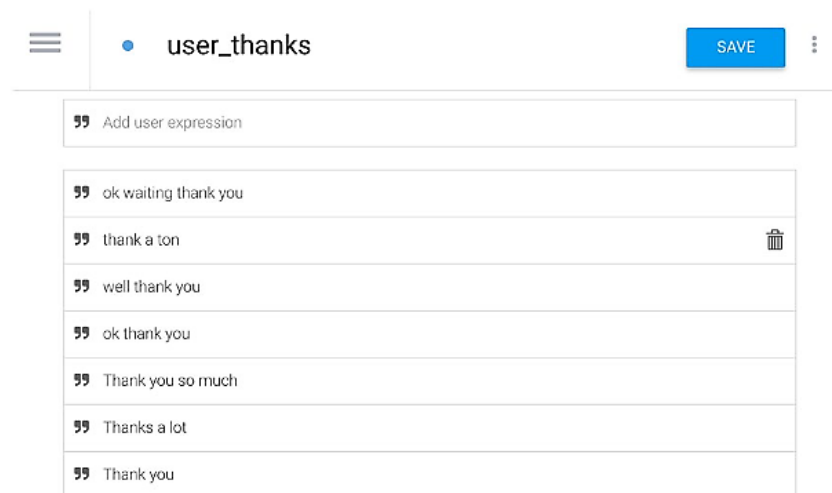
Gambar 4.10. Menentukan maksud ID pesanan pengguna di agen kami

Saya menguji beberapa ekspresi, dan berfungsi dengan baik untuk mengklasifikasikannya dengan benar sebagai maksud `user_order_id`. Selalu uji menggunakan konsol Dialogflow untuk melihat apakah niat Anda berfungsi seperti yang diharapkan. Sekarang, mari kita atur respon default dari `user_order_id` menjadi respon dari bot berikut:

Status Pesanan untuk ID Pesanan: `$order_id`. Pengantar berada di wilayah Anda, makanan Anda akan tiba dalam waktu sekitar 5 menit. Kami kembali menggunakan parameter yang diurai dari maksud `user_order_id` untuk menyiapkan balasan kepada pengguna.

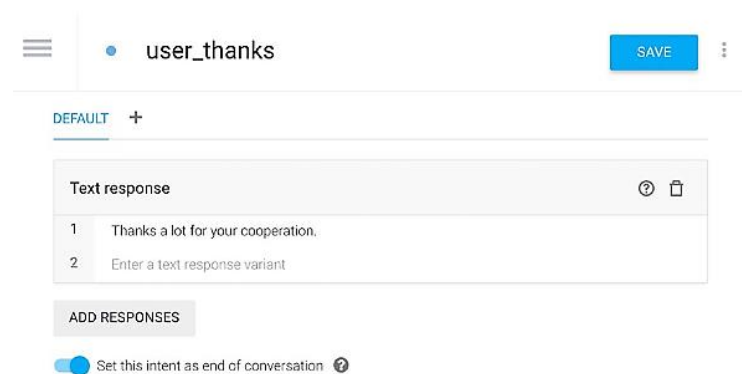
User_Thanks Intent

Selanjutnya, pengguna mungkin akan mengucapkan terima kasih, jika bukan sesuatu yang lain, jadi kami membuat maksud baru yang disebut `user_thanks` untuk mengidentifikasi berbagai cara pengguna mengucapkan terima kasih. Ini penting karena begitu pengguna mengucapkan terima kasih dengan cara tertentu, bot kami akan membalas hal yang sama. Kita tidak boleh hanya mengharapkan pengguna mengucapkan terima kasih setelah respons default status pengiriman dan membalas secara membabi buta, tetapi cobalah mengidentifikasinya menggunakan maksud khusus. Gambar 4-11 menunjukkan tampilan maksud `user_thanks` kita.



Gambar 4.11. Mendefinisikan maksud ketika pengguna mengucapkan terima kasih

Sekarang, saatnya mengucapkan terima kasih kepada pengguna yang menggunakan fitur respons default dan menandai maksud ini sebagai akhir percakapan. Mari tambahkan beberapa teks seperti, “Terima kasih banyak atas kerja sama Anda,” sebagai respons default kita. Kita dapat menambahkan lebih banyak tanggapan seperti itu sehingga bot terlihat lebih realistis (lihat Gambar 4.12).



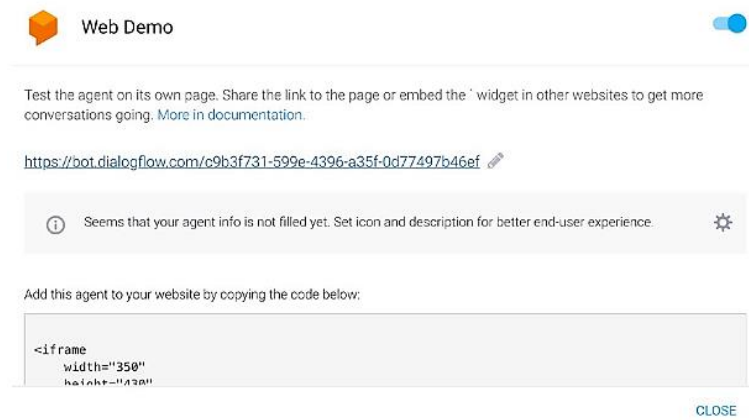
Gambar 4.12. Menambahkan respons terhadap maksud pengguna di agen

Lihatlah Gambar 4.12 dan lihat bahwa kita telah mengaktifkan maksud ini sebagai akhir percakapan. Jika kami mencoba mengintegrasikan bot kami dengan Asisten Google, mengaktifkannya berarti menutup mikrofon di Asisten Google saat niat selesai. Sekarang pada titik ini kami telah membuat bot kami, membuatnya sesuai desain dan skrip awal kami, dan melatihnya. Sekarang saatnya menerapkannya di web dan melihat tampilannya.

4.5 Menyebarkan Dialogflow Chatbot di Web

Pada bagian ini, kita akan mengintegrasikan bot kita dengan berbagai platform seperti Facebook Messenger, Twitter, Slack, dll., dan melihat apakah berfungsi. Masih banyak lagi platform di mana Anda dapat mengintegrasikan bot ini dengan mudah. Kami akan menguji bot kami dengan Web Demo dan Facebook Messenger untuk saat ini. Mari

buka halaman Integrasi di akun Dialogflow kita dan aktifkan Demo Web. Anda akan mendapatkan pop-up seperti Gambar 4.13. Klik tautan di pop-up.

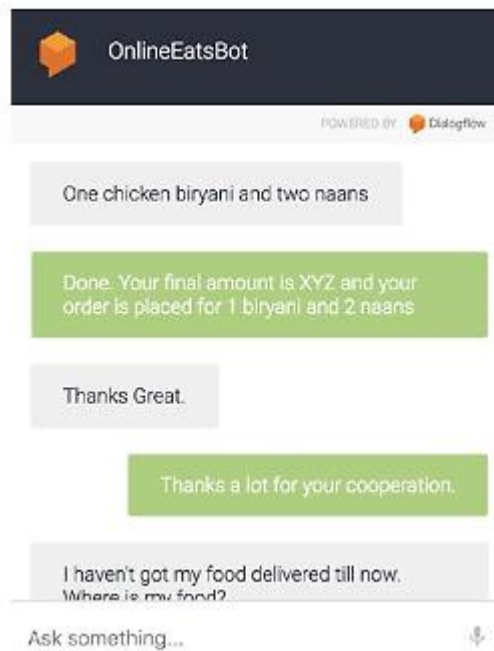


Gambar 4.13. Tautan demo web Dialogflow

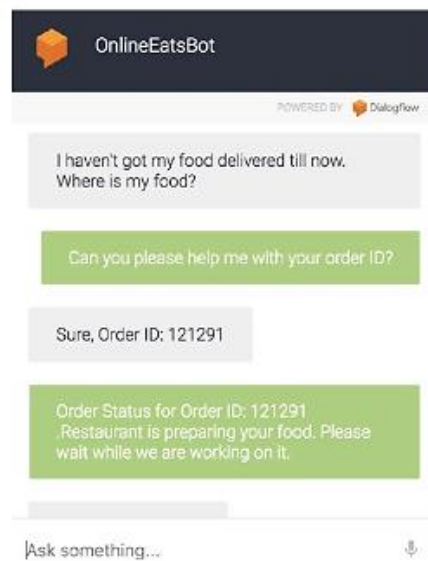
Anda akan melihat sesuatu yang mirip dengan Gambar 4.14.1 hingga 4.14.4. Saya menguji bot saya dengan percakapan yang kami tulis, dan bot saya bekerja dengan sangat baik.



Gambar 4.14.(1) Layar Percakapan Demo OnlineEatsBot I



Gambar 4.14.(2). Layar Percakapan Demo OnlineEatsBot II



Gambar 4.14.(3). Layar Percakapan Demo OnlineEatsBot III



Gambar 4.14.(4). Layar Percakapan Demo OnlineEatsBot IV

Selain itu kami juga dapat menyematkan bot ini di situs web kami menggunakan kode iframe yang terdapat di jendela pop-up. Bicaralah dengan OnlineEatsBot saya di sini: <https://bot.dialogflow.com/c9b3f731-599e-4396-a35f-0d77497b46ef> .

Bagikan juga bot Anda dengan teman dan keluarga, dan lihat cara mereka berinteraksi dengan bot dengan cara yang sah. Jika chatbot Anda tidak melakukan sesuatu yang diharapkan, cobalah memperbaikinya.

4.6 Integrasikan Dialogflow Chatbot di Facebook Messenger

Di bagian ini kami akan mencoba mengintegrasikan chatbot kami yang sama ke Facebook Messenger sehingga pengguna platform Facebook kami juga dapat menggunakannya tanpa harus mengunjungi situs web kami. Mari kembali ke halaman integrasi di dasbor Dialogflow dan aktifkan ikon Facebook Messenger, lalu klik ikon tersebut, yang akan memunculkan pop-up serupa seperti sebelumnya.

Di sini kita perlu pergi ke Facebook, mendaftarkan aplikasi, dan mendapatkan token yang diperlukan.

- Verifikasi Token (string apa pun dan semata-mata untuk tujuan Anda)
- Token Akses Halaman (Masukkan token yang dihasilkan di Konsol Pengembang Facebook)

Integrasi Dialogflow Facebook sangat membantu untuk membuat bot Facebook Messenger dengan mudah dengan NLU, berdasarkan teknologi Dialogflow.

Menyiapkan Facebook

Agar bot kita berfungsi sama seperti di Facebook, kita perlu melakukan hal berikut:

1. Buat akun Facebook jika Anda belum melakukannya.
2. Buat halaman Facebook tempat Anda dapat menambahkan bot Anda.

Saat pengguna mengunjungi halaman Facebook Anda dan mengirimi Anda pesan, mereka akan berbicara langsung dengan bot Anda.

Membuat Aplikasi Facebook

Berikut langkah-langkah membuat aplikasi:

1. Masuk ke Konsol Pengembang Facebook.
2. Klik Aplikasi Saya di pojok kanan atas.
3. Klik Tambah Aplikasi Baru dan masukkan nama dan alamat email kontak.
4. Klik Create App ID seperti terlihat pada Gambar 4-15 di bawah.
5. Di halaman berikutnya, klik tombol Set Up untuk opsi Messenger.
6. Di bawah bagian Pembuatan Token, pilih halaman Facebook yang ingin kita sambungkan dengan bot (lihat Gambar 4-16).

Create a New App ID

Get started integrating Facebook into your app or website

Display Name
OnlineEatsBot

Contact Email
YOUR-CONTACT-EMAIL@example.com

By proceeding, you agree to the Facebook Platform Policies

Cancel Create App ID

Gambar 4.15. Membuat aplikasi baru di Platform Pengembang Facebook

Token Generation

Page token is required to start using the APIs. This page token will have all messenger permissions even if your app is not approved to use them yet, though in this case you will be able to message only app admins. You can also generate page tokens for the pages you don't own using Facebook Login.

Page	Page Access Token
Onlineeatsbot	EAADDh3kSpOEBAMF9lmbMwrmzAXZByAUwsX5MiqZAEVG8e2tv4fNOmlszf13ZBm1KLNKdUY7e8jMGPRKSuwKZCQSQ!

Create a new page

Gambar 4.16. Menghasilkan token dengan memilih halaman Facebook untuk bot Anda

Ini akan menghasilkan Token Akses Halaman. Simpan token ini, karena kita harus memasukkannya ke Dialogflow.

Menyiapkan Konsol Dialogflow

Berikut langkah-langkahnya:

1. Klik opsi Integrasi di konsol Dialogflow Anda di menu sebelah kiri dan aktifkan Facebook Messenger jika Anda belum melakukannya. Pada pop-up yang terbuka, masukkan informasi berikut seperti yang ditunjukkan pada Gambar 4-17 Menyiapkan dan mengintegrasikan Dialogflow dengan Facebook Messenger:
 - Verifikasi Token: ini bisa berupa string apa pun yang Anda inginkan dan untuk tujuan Anda sendiri

- Token Akses Halaman: masukkan token yang dihasilkan di Konsol Pengembang Facebook
2. Klik tombol Mulai.

Gambar 4.17. Menyiapkan dan mengintegrasikan Dialogflow dengan Facebook Messenger

Anda akan mendapat pesan yang mengatakan, “Bot telah dimulai.” Itu berarti kita siap berangkat. Anda pasti bertanya-tanya apa itu URL panggilan balik, Token Verifikasi, dan Token Akses Halaman. Mari kita coba memahaminya.

4.7 URL Panggilan Balik

URL panggilan balik tidak lain hanyalah URL yang dapat diakses publik di mana Facebook akan MEMPOSTING permintaan real-time apa pun yang datang dari halaman Facebook Anda.

Misalkan Anda mencoba melakukan pembayaran makanan Anda di OnlineEats dan kemudian Anda diarahkan ke halaman pembayaran bank. Sekarang, OnlineEats harus memberikan URL panggilan balik ke bank tempat mereka dapat mengarahkan pengguna setelah pembayaran selesai.

Di sini Facebook tidak akan melakukan pengalihan apa pun tetapi akan mengambil semua pesan pengguna kami di kotak obrolan halaman dan POST ke webhook atau URL panggilan balik. Sekarang, setelah kami menerima pesan di server kami, kami melakukan klasifikasi maksud dan penguraian entitas, lalu merumuskan apa yang ingin Anda balas kembali ke pengguna.

Verifikasi Token

Token verifikasi adalah string arbitrer yang dikirim ke titik akhir Anda saat langganan diverifikasi. Alasan mengapa hal ini diperlukan adalah untuk memastikan bahwa server kami mengetahui bahwa permintaan tersebut dibuat oleh Facebook dan terkait dengan langganan yang baru saja kami konfigurasi.

Misalkan orang lain mengetahui webhook Anda dan memposting pesan yang menyamar sebagai Facebook, maka verifikasi_token akan muncul, dan Anda akan memverifikasi apakah sumbernya benar atau tidak. Berdasarkan token ini, Anda juga dapat menangani permintaan POST dari berbagai sumber karena akan ada token berbeda yang ditentukan untuk sumber berbeda tetapi url panggilan baliknya sama.

Akses Token

API Facebook memerlukan Token Akses Halaman untuk mengelola halaman Facebook. Mereka unik untuk setiap halaman, admin, dan aplikasi dan memiliki waktu kedaluwarsa. Catatan Simpan URL panggilan balik dan Token Verifikasi untuk mengonfigurasi webhook sekarang.

4.8 Mengonfigurasi Webhook

Untuk mengonfigurasi webhook bot kita, mari kembali ke Konsol Pengembang Facebook:

1. Klik tombol Pengaturan di bawah bagian Tambahkan produk untuk webhook ketika Anda mengklik dasbor. Jika Anda belum berlangganan webhook, Anda akan mendapatkan opsi yang mengatakan “berlangganan objek ini.” Klik ini untuk mendapatkan pop-up baru dan masukkan informasi berikut:
 - URL Panggilan Balik: ini adalah URL yang disediakan di halaman integrasi Facebook Messenger.
 - Verifikasi Token: ini adalah Token yang Anda buat.
2. Buka Messenger ► Pengaturan ► Atur Webhook. Anda akan mendapatkan pop-up seperti Gambar 4.18. Tambahkan url panggilan balik Anda dan verifikasi token.
3. Periksa opsi messages dan messages_postbacks di bawah Subscription Fields. Anda pasti dapat memilih mana saja yang diperlukan untuk kasus penggunaan Anda.
4. Klik tombol Verifikasi dan Simpan. Periksa Gambar 4.18 untuk referensi.

Anda akan dibawa kembali ke halaman pengaturan dan Webhook seharusnya memiliki status “Selesai”. Pastikan untuk memilih halaman tempat Anda dapat berlangganan webhook Anda untuk acara halaman.

New Page Subscription [X]

Callback URL

Verify Token

Subscription Fields

<input checked="" type="checkbox"/> messages	<input checked="" type="checkbox"/> messaging_postbacks	<input type="checkbox"/> messaging_optins
<input type="checkbox"/> message_deliveries	<input type="checkbox"/> message_reads	<input type="checkbox"/> messaging_payments
<input type="checkbox"/> messaging_pre_checkouts	<input type="checkbox"/> messaging_checkout_updates	<input type="checkbox"/> messaging_account_linking
<input type="checkbox"/> messaging_referrals	<input type="checkbox"/> message_echoes	<input type="checkbox"/> messaging_game_plays
<input type="checkbox"/> standby	<input type="checkbox"/> messaging_handovers	<input type="checkbox"/> messaging_policy_enforcement

Learn more

Gambar 4.18. Menyiapkan webhook di Facebook untuk bot Dialogflow

4.9 Menguji Bot Messenger

Agar bot kami tersedia untuk pengujian, kami perlu menjadikan aplikasi kami publik:

1. Klik Tinjauan Aplikasi di menu sebelah kiri Konsol Pengembang Facebook.
2. Klik tombol di bawah Jadikan <Nama APLIKASI Anda> publik? Jika Anda mendapatkan pesan URL Kebijakan Privasi Tidak Valid, buka tautan Pengaturan Dasar di kotak dialog dan, jika Anda belum melakukannya, masukkan URL apa pun untuk URL Kebijakan Privasi, untuk saat ini, lalu klik Simpan Perubahan. Sekarang, kembali ke halaman Tinjauan Aplikasi dan coba alihkan aplikasi ke publik lagi.
3. Anda akan diminta memilih kategori untuk aplikasi Anda.
4. Pilih Pendidikan dari daftar. Jangan ragu untuk memilih mana yang paling sesuai dengan bot Anda.
5. Klik tombol Konfirmasi seperti yang ditunjukkan pada Gambar 4-19, Menjadikan aplikasi facebook Anda publik.

Make App Public? [X]

Are you sure you want to make your app public? It will become available to everyone.

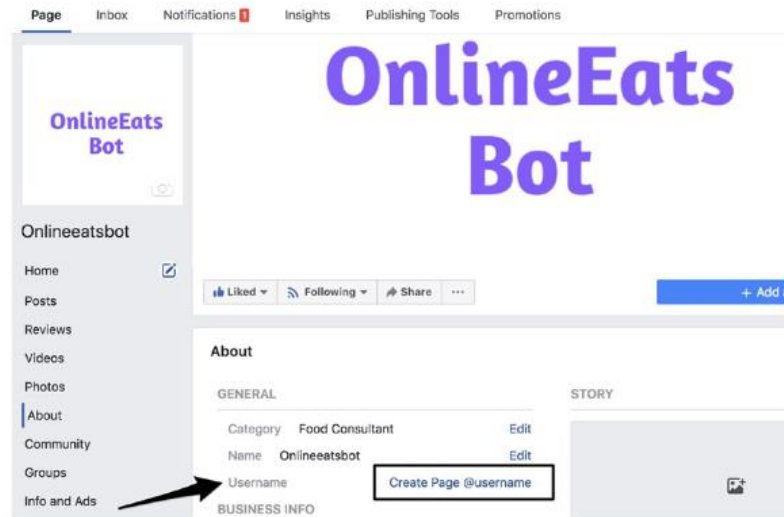
Please select a category before you go public

Category

Gambar 4.19. Menjadikan aplikasi facebook Anda publik

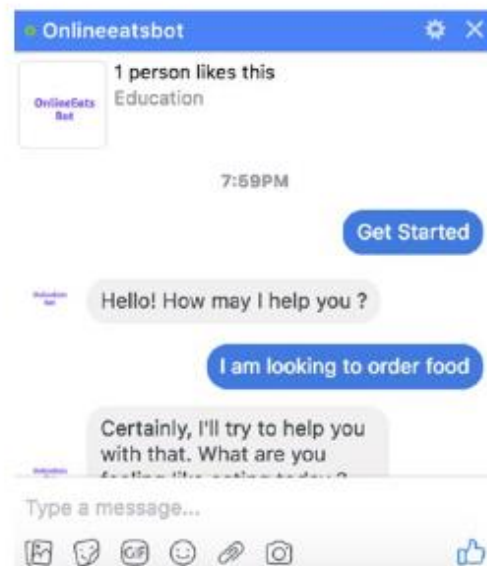
Kita juga perlu membuat nama pengguna untuk halaman kita. Ini adalah nama pengguna yang akan digunakan pengguna untuk ngobrol saat menggunakan bot kami. Untuk mengatur nama pengguna, klik link Buat Halaman @NamaPengguna di bawah bagian

Tentang di halaman Anda, seperti yang ditunjukkan pada Gambar 3-20. Ini berguna untuk membagikan halaman atau bot Anda dengan orang-orang yang hanya menggunakan nama pendek.



Gambar 4.20. Membuat nama pengguna halaman bot Facebook Anda

Mari kita uji aliran bot yang sama di Facebook Messenger yang kami uji di situs web Dialogflow. Anda seharusnya dapat melihat bagaimana bot Facebook Messenger saya merespons dengan mengacu pada Gambar 4.21.(1) hingga Gambar 4.21.(4).



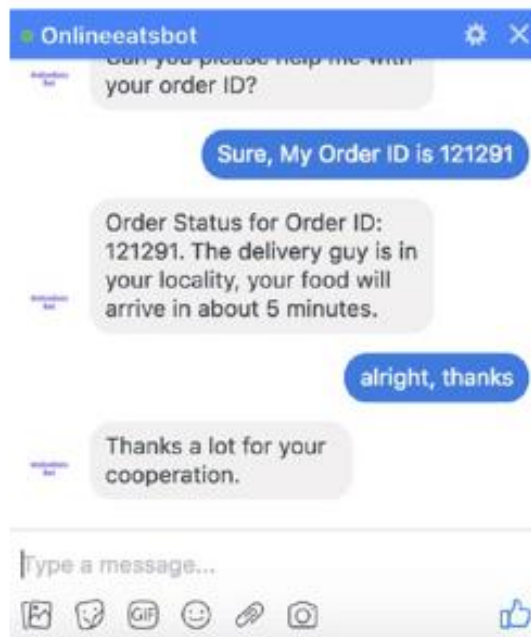
Gambar 4-21.(1). Layar Demo Facebook Messenger OnlineEatsBot I



Gambar 4.21.(2). Layar Demo Facebook Messenger OnlineEatsBot II



Gambar 4-21.(3). Layar Demo Facebook Messenger OnlineEatsBot III



Gambar 4-21.(4). Layar Demo Facebook Messenger OnlineEatsBot IV

Dan itulah cara kami membuat bot. Bab 4 akan menjadi lebih menarik. Di Bab 4, kami akan mencoba mencapai hal yang sama tanpa harus bergantung pada API atau dasbor Dialogflow.

Selalu menyenangkan bila Anda memiliki kendali penuh atas semua yang Anda miliki, bukan?

Catatan: Anda dapat membuka pengaturan akun Anda dan mengekspor agen atau mengimpor agen lain secara langsung. Anda dapat mengunduh file zip (OnlineEatsBot.zip). Anda dapat menggunakan zip ini untuk mengimpornya ke Dialogflow dan bermain dengan chatbot yang kita buat di bab ini. Anda pasti bertanya-tanya, bagaimana jika saya ingin melakukan penempatan pesanan secara real-time dan mengetahui status pesanan menggunakan API vendor/restoran dan membalas pengguna dengan sesuai? Bisa berupa panggilan API apa pun yang ingin Anda lakukan ambil data secara real-time dan merumuskan respons bot. Inilah saatnya mengetahui cara melakukannya sebelum kita menyelesaikan bab ini dan bersiap untuk bab berikutnya.

Mari pelajari tentang sesuatu yang disebut “Pemenuhan” di Dialogflow.

4.10 Pemenuhan

Untuk mendapatkan informasi real-time yang diminta oleh pengguna, kita perlu mengembangkan beberapa API atau menggunakan yang sudah ada. Untuk mencapai hal tersebut menggunakan Dialogflow, kita perlu menyiapkan pemenuhan, yang memerlukan penerapan layanan dan pemanggilan API.

Kami tidak akan membahas seluk beluk pembuatan API dan cara menerapkannya, tetapi jika Anda pernah mencoba menggunakan API Google atau Facebook, Anda setidaknya harus memahami cara memanggilnya.

Saya telah membuat API kecil berbasis Flask dan menerapkannya ke Heroku. Saya akan menggunakannya untuk pemenuhan, yang hanya mengambil order_id apa pun di url dan mengembalikan status_pesanan acak. Jika Anda belum familiar dengan Heroku, jangan khawatir, Anda dapat menjalankan kode yang disediakan di sistem lokal Anda dan mengujinya. Di bab mendatang, kami akan menerapkan banyak aplikasi menggunakan Heroku di mana Anda dapat mempelajari hal-hal terkait dengannya. Dalam kode Anda dapat membaca bagaimana order_identity, intensName, dll., diurai. Temukan kodenya di sini: flask_onlineeats_demo.zip

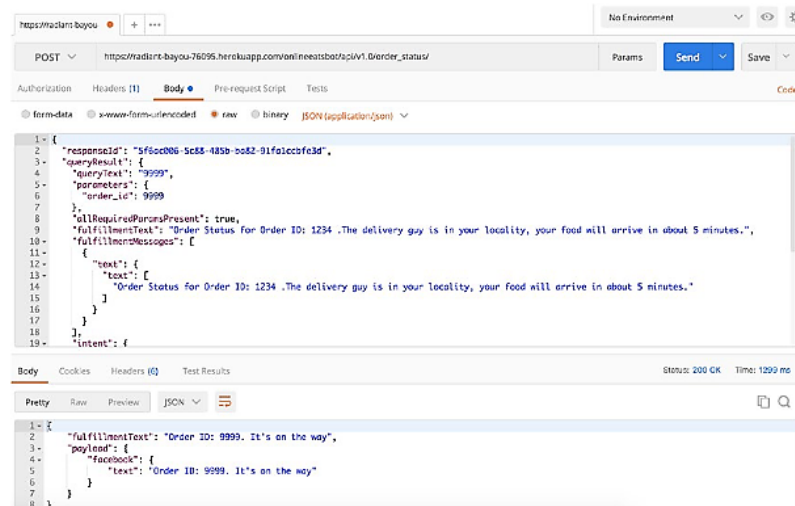
URL Permintaan:

https://radiant-bayou-76095.herokuapp.com/onlineeatsbot/api/v1.0/order_status/

Jadi, dalam Dialogflow Fulfillment akan POST respons JSON dari maksud ke URL ini, yang perlu Anda uraikan untuk mendapatkan entitas yang relevan dan nilainya serta melakukan tindakan tertentu.

Anda juga dapat mencoba menerapkan contoh kode Aplikasi Flask di Heroku dan membuat titik akhir Anda sendiri berfungsi di bot Anda untuk pemenuhan.

Sekarang, Dialogflow akan POST respons JSON dari maksud yang panggilan webhooknya diaktifkan di titik akhir kami. Ia memiliki kode untuk mengurai entitas order_id dan mengambil tindakan berdasarkan itu. Saat ini, kode tersebut hanya mengembalikan status yang dipilih secara acak dari daftar. Untuk menguji apakah API berfungsi, buka POSTMAN dan uji menggunakan data sampel pada Gambar 4-22. Jika Anda menjalankan aplikasi Flask di lokal, gunakan url lokal.



Gambar 4.22. Menguji API pemenuhan yang diterapkan pada heroku di POSTMAN

Mengaktifkan Webhook

Jadi, mari buka halaman pemenuhan di Dialogflow dan coba aktifkan webhook (lihat Gambar 4.23).

Fulfillment

Webhook ENABLED

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the webhook requirements specific to the API version enabled in this agent.

URL*

BASIC AUTH

HEADERS

Content-Type

Enter key Enter value

[+ Add header](#)

DOMAINS

Gambar 4.23. Menyiapkan webhook untuk pemenuhan di Dialogflow

Pastikan Anda telah mengaktifkan panggilan webhook untuk maksud `user_order_id` (lihat Gambar 4-24).

user_order_id SAVE

- 1 Order Status for Order ID: Sorder_id .Restaurant is preparing your food. Please wait while we are working on it.
- 2 Order Status for Order ID: Sorder_id .The delivery guy has picked up your food. Please wait for 15-20 minutes.
- 3 Order Status for Order ID: Sorder_id .The delivery guy is in your locality, your food will arrive in about 5 minutes.
- 4 Enter a text response variant

[ADD RESPONSES](#)

Set this intent as end of conversation [?](#)

Fulfillment [?](#) ^

Enable webhook call for this intent

Enable webhook call for slot filling

Gambar 4.24. Mengaktifkan panggilan webhook untuk maksud tertentu

Dialogflow akan POST isi JSON ke URL webhook Anda, yang akan terlihat seperti Gambar 4.25.

JSON

```

1 - {
2   "responseId": "6e48703b-0259-4d6c-81a1-c6e6b2da0d07",
3   "queryResult": {
4     "queryText": "Order ID is 234",
5     "parameters": {
6       "order_id": 234
7     },
8     "allRequiredParamsPresent": true,
9     "fulfillmentText": "Order ID: 234.0. Restaurant preparing the food",
10    "fulfillmentMessages": [
11      {
12        "text": {
13          "text": [
14            "Order ID: 234.0. Restaurant preparing the food"
15          ]
16        }
17      }
18    ],
19    "webhookPayload": {
20      "facebook": {
21        "text": "Order ID: 234.0. Restaurant preparing the food"
22      }
23    },
24    "intent": {
25      "name": "projects/helloaa-fdfca/agent/intents/9a39f983-74e8-487c-bb67-
26        -a22b728fc3d2",
27      "displayName": "user_order_id"
28    },
29    "intentDetectionConfidence": 0.97,
30    "diagnosticInfo": {
31      "webhook_latency_ms": 200
32    },
33    "languageCode": "en"
34  },
35  "webhookStatus": {
36    "message": "Webhook execution successful"
37  }
38 }

```

CLOSE COPY

Gambar 4.25. Data JSON masuk dari Dialogflow ke titik akhir webhook kami

4.11 Memeriksa Respon

Dialogflow mengharapkan respons dari layanan web Anda dalam format yang ditunjukkan pada Gambar 4.26 setiap kali mereka POST respons JSON suatu maksud (ditunjukkan pada Gambar 4.25) ke layanan web Anda.

```

1 {
2   "fulfillmentText": "This is a text response",
3   "fulfillmentMessages": [
4     {
5       "card": {
6         "title": "card title",
7         "subtitle": "card text",
8         "imageUri": "https://assistant.google.com/static/images/molecule/Molecule-Formation-stop.png",
9         "buttons": [
10          {
11            "text": "button text",
12            "postback": "https://assistant.google.com/"
13          }
14        ]
15      }
16    },
17    {
18      "source": "example.com",
19      "payload": {
20        "google": {
21          "expectUserResponse": true,
22          "richResponse": {
23            "items": [
24              {
25                "simpleResponse": {
26                  "textToSpeech": "this is a simple response"
27                }
28              }
29            ]
30          }
31        },
32        "facebook": {
33          "text": "Hello, Facebook!"
34        },
35        "slack": {
36          "text": "This is a text response for Slack."
37        }
38      }
39    },
40    "outputContexts": [
41      {
42        "name": "projects/${PROJECT_ID}/agent/sessions/${SESSION_ID}/contexts/context name",
43        "lifespanCount": 5,
44        "parameters": {
45          "param": "param value"
46        }
47      }
48    ],
49    "followupEventInput": {
50      "name": "event name",
51      "languageCode": "en-US",
52      "parameters": {
53        "param": "param value"
54      }
55    }
56 }

```

Gambar 4.26. Respons dari URL webhook yang diharapkan oleh Dialogflow

Jika Anda berpikir bahwa respons API Anda harus sama persis dengan format seperti Gambar 4.26, santai saja bukan itu masalahnya. Niat Anda tidak akan menimbulkan kesalahan karena semua kunci di badan JSON bersifat opsional.

Inilah tampilan respons API saya dan berfungsi dengan sempurna:

```
{
  "fulfillmentText": "Order ID: 9999. It's on the way",
  "payload": {
    "facebook": {
      "text": "Order ID: 9999. It's on the way"
    }
  }
}
```

Ketika saya mencoba menggunakan API yang sama lagi, saya mendapatkan teks status pesanan yang berbeda tetapi dengan format yang sama yang diharapkan oleh mesin Dialogflow.

```
{
  "fulfillmentText": "Order ID: 9999. Rider has picked up your food,
please wait for another 10-15 minutes",
  "payload": {
    "facebook": {
      "text": "Order ID: 9999. Rider has picked up your food, please
wait for another 10-15 minutes"
    }
  }
}
```

FulfillmentText adalah kunci yang penting bagi agen untuk membalas sesuatu kepada pengguna. Sekarang, coba bot dengan URL publik atau di agen Dialogflow itu sendiri untuk melihat respons yang datang dari API, bukan respons statis default yang kami tambahkan sebelumnya. Inilah cara kami mengintegrasikan API eksternal atau API kami sendiri menggunakan fitur pemenuhan Dialogflow ke dalam chatbot kami untuk menjadikan segalanya dinamis dan real-time.

Ringkasan

Dalam bab ini kita mempelajari tentang Dialogflow dan cara menggunakan Dialogflow untuk membangun chatbot. Kami belajar mendefinisikan maksud dan entitasnya masing-masing. Kami membangun chatbot pemesanan makanan sederhana yang memahami maksud pemesanan makanan dari pengguna dan juga memahami jenis makanan apa yang diminta pengguna serta jumlahnya. Kami juga menyempurnakan chatbot kami agar pengguna dapat bertanya tentang status pesanan mereka dan mengambil ID pesanan dari mereka serta merumuskan respons dengan status pesanan yang berbeda.

Kami juga mempelajari pemenuhan di Dialogflow, tempat kami mengambil status pesanan dari API kami sendiri dan memberikan respons kepada pengguna berdasarkan status tersebut. Kami belajar membuat demo web chatbot kami, dan kami juga

mengintegrasikan bot kami dengan Messenger. Pada titik ini, Anda seharusnya sudah memiliki gambaran yang jelas tentang cara kerja chatbot secara end-to-end.

Pada bab selanjutnya kita akan mencoba cara yang lebih sulit dalam membangun chatbot. ya, Anda tidak salah dengar. Kami akan menghilangkan ketergantungan pada alat seperti Dialogflow dan membangun semuanya sendiri secara terprogram. Mari bersiap untuk bab selanjutnya, karena akan lebih menyenangkan jika Anda membangun semuanya sendiri dari awal. Ini seperti melatih dan menjinakkan chatbot Anda sendiri. Sampai jumpa di bab berikutnya.

BAB 5

MEMBANGUN CHATBOTS DENGAN CARA YANG SULIT

“Membangun Chatbots dengan Cara yang Sulit” tidak terlalu sulit untuk dipelajari. Ini adalah cara yang sulit dalam membangun chatbot untuk memiliki kendali penuh atas chatbot Anda sendiri. Jika Anda ingin membangun semuanya sendiri, maka Anda mengambil jalan yang sulit. Rute yang lebih sulit memang sulit ketika Anda melewatinya tetapi indah dan jelas ketika Anda melihat ke belakang.

*Ini adalah jalan terjal yang menuju ke puncak keagungan.
Lucius Annaeus Seneca*

5.1 Pendahuluan

Jika Anda memahami Python dengan baik dan memiliki sedikit pemahaman tentang cara menyiapkan paket, dll., Anda tidak akan kesulitan mempelajari bab ini. Jika Anda seorang pengembang, itu akan mudah bagi Anda. Jika Anda seorang manajer atau orang non-teknis, Anda masih dapat melakukan langkah-langkah satu per satu seperti yang disebutkan di setiap bagian dan menyelesaikan semuanya. Saya sangat menyarankan semua orang menyelesaikan bab ini sehingga Anda mempelajari hal-hal inti tentang membangun chatbots.

Bab ini tidak hanya mengajarkan Anda cara membuat chatbot dari awal tetapi juga menunjukkan cara kerja pembelajaran mesin inti (ML) dengan NLP dengan bantuan Rasa NLU. Seperti yang disebutkan di bab pertama buku ini, ada baiknya jika Anda memiliki pohon keputusan saat Anda membuat chatbot. Dalam bab ini, kita hampir tidak akan menggunakan aturan apa pun, namun ML belum berada pada tahap yang dapat diandalkan 100 persen. Jadi, keputusan ini sepenuhnya didasarkan pada kasus penggunaan Anda dan apakah Anda ingin menerapkan logika bisnis bersama dengan model ML Anda atau tidak. Terkadang, model ML Anda mungkin berfungsi dengan baik sehingga Anda tidak memerlukan heuristik sama sekali. Namun menurut pengalaman saya, saat Anda ingin menjual chatbot atau mengkomersialkannya, Anda harus sedikit berhati-hati. Tidak ada fungsi yang lebih baik daripada memiliki fungsi yang tidak masuk akal.

Kami akan menggunakan perpustakaan sumber terbuka yang disebut Rasa NLU untuk mempelajari cara membuat chatbot dari awal tanpa menggunakan layanan cloud apa pun seperti Dialogflow, Watson, wit.ai, dll. Ingat Rasa NLU adalah perpustakaan yang sangat canggih dan memiliki banyak fitur di dalamnya. Kami hanya akan membahas konsep dan fitur yang penting bagi kami untuk membangun chatbot kami.

5.2 Apa itu Rasa NLU?

Rasa NLU adalah pustaka NLP sumber terbuka untuk klasifikasi maksud dan ekstraksi entitas di chatbots. Ini membantu Anda membangun dan menulis NLP khusus untuk chatbot Anda.

Ada dua bagian Rasa yang akan kita bahas di bab ini.

- **Rasa NLU:** Dengan Rasa NLU kita akan belajar menyiapkan data pelatihan untuk chatbot, menulis file konfigurasi, memilih pipeline, dan melatih model. Terakhir, kami akan memprediksi maksud sebuah teks menggunakan model kami. Kami juga akan mempelajari cara mengurai entitas menggunakan Rasa NLU.
- **Rasa Core:** Pada bagian kedua kita akan belajar melatih model manajemen dialog Rasa Core untuk mempersiapkan tanggapan kembali kepada pengguna. Bagian ini menjadi sangat penting ketika Anda memiliki beragam maksud di chatbot Anda dan pertanyaan atau tanggapan tindak lanjutnya. Daripada menulis banyak kondisi di pohon keputusan dan menghabiskan waktu berjam-jam untuk men-debugnya jika ada aplikasi tingkat perusahaan besar, lebih baik mengajarkan model untuk membuat respons. Menarik untuk melihat seberapa baik performa model terlatih kami dalam melakukan hal tersebut. Kami tidak bisa begitu saja memberikan apa pun kepada pengguna dalam bentuk teks; itu seharusnya masuk akal.

5.3 Mengapa Saya Harus Menggunakan Rasa NLU?

Rasa NLU bukan sembarang perpustakaan dengan banyak metode untuk melakukan beberapa hal. Ia memiliki kemampuan membangun hampir semua jenis chatbot yang dapat Anda bayangkan. Rasa memberi Anda kemampuan ajaib dalam melatih mesin untuk memahami makna teks daripada Anda menulis aturan untuk memahaminya.

Mari kita simak poin-poin berikut mengapa kita harus menggunakan Rasa NLU:

- Rasa NLU adalah proyek yang dikelola secara aktif dan memiliki komunitas yang baik untuk mendukungnya.
- Jika kami tidak ingin membagikan data sensitif pengguna kami kepada pihak ketiga, kami harus menggunakan alat sumber terbuka seperti Rasa NLU untuk membuat chatbot dari awal. Dengan cara ini semua data tetap ada dan diproses di server kami sendiri.
- Bergantung pada layanan pihak ketiga untuk melatih data Anda dan menemukan maksud ucapan pengguna, Anda harus memanggil API yang mungkin tidak selalu dapat diandalkan. Apa yang terjadi pada aplikasi chatbot Anda jika servernya sedang down?
- Menggunakan Rasa NLU untuk membuat chatbot akan memberi Anda kendali penuh atas chatbot Anda. Anda dapat melatih, menyempurnakan, dan mengoptimalkannya sesuai keinginan dengan data yang Anda inginkan. Dengan Rasa NLU kita dapat bereksperimen dengan algoritma ML mana yang paling cocok untuk kumpulan data kita daripada bergantung pada algoritma tetap.

5.4 Menyelam Langsung Ke Rasa NLU

Di bagian ini kita akan langsung mempelajari bagian praktiknya dan mencoba memasang tumpukan Rasa dan mulai membangun model ML dengan melatih data kami. Kami akan menggunakan beberapa perpustakaan sumber terbuka yang keren untuk membuat hidup kami lebih mudah.

Memasang Rasa

Untuk menginstal Rasa, jalankan perintah pip berikut yang kami coba di bab sebelumnya untuk menginstal spaCy. Perhatikan bahwa kami akan menggunakan Rasa versi 0.13.2.

Pip install rasa-nlu ==0.13.2

Rasa NLU memiliki banyak komponen untuk mengklasifikasikan maksud dan mengenali entitas. Komponen Rasa yang berbeda memiliki ketergantungannya masing-masing. Saat kami melatih model kami, Rasa NLU memeriksa apakah semua dependensi yang diperlukan telah diinstal. Jika Anda ingin menginstal persyaratan lengkap yang diperlukan untuk sepenuhnya menggunakan perpustakaan Rasa, Anda dapat menjalankan langkah-langkah berikut:

```
git clone https://github.com/RasaHQ/rasa_nlu.git #Clone the repo
cd rasa_nlu #Get into the rasa directory
pip install -r alt_requirements/requirements_full.txt #Install full
requirements
```

5.5 Memutuskan Saluran Pipa di Rasa

Pipeline tidak lain hanyalah sekumpulan algoritme yang akan digunakan untuk melatih model Anda. Rasa NLU memiliki dua pipeline yang banyak digunakan yang disebut `spacy_sklearn` dan `tensorflow_embedding`. Mari kita pelajari sedikit tentang keduanya.

spacy_sklearn

- `spacy_sklearn` pipeline menggunakan vektor kata terlatih baik dari algoritme GloVe atau algoritme yang dikembangkan oleh tim AI Facebook yang disebut fastText.
- `spacy_sklearn` bekerja dengan sangat baik dalam situasi di mana, misalkan Anda memiliki ucapan seperti, “Bagaimana cuaca di Boston?” Saat kita melatih model kita menggunakan contoh ucapan yang sama dan memintanya untuk memprediksi maksud dari, “Bagaimana cuaca di London?” model kita sekarang cukup cerdas untuk mengetahui bahwa kata “Boston” dan “London” memiliki kemiripan, dan memiliki maksud yang sama.
- Pipeline ini sangat berguna dengan kumpulan data yang kecil.

tensorflow_embedding

- Pipeline `tensorflow_embedding` tidak menggunakan vektor kata yang telah dilatih sebelumnya seperti `spacy_sklearn`, namun ia menyesuaikan dirinya sendiri sesuai dengan kumpulan data yang kami sediakan.
- Hal yang baik tentang pipeline `tensorflow_embedding` adalah vektor kata kita akan sesuai dengan domain kita.
- Untuk menjelaskan cara kerja `tensorflow_embedding` dengan sebuah contoh, dalam bahasa Inggris, kata “bermain” mungkin berkaitan erat dengan “olahraga” atau “aktivitas yang menyenangkan atau rekreasi”, dan mungkin terlihat sangat berbeda dengan kata “an bertindak.” Dalam ranah teater, “lakon” dan “suatu lakon” berkaitan erat, di mana “lakon” berarti “suatu bentuk karya sastra yang ditulis oleh seorang penulis naskah

drama,” dan sangat penting untuk memberi tahu model kita agar belajar spesifik pada ranah kita dan bukan menjadi bingung karena beberapa model terlatih.

5.6 Melatih dan Membangun Chatbot Dari Awal

Kami akan membuat bot horoskop yang memahami pertanyaan pengguna dan memberi tahu mereka horoskop mereka untuk hari itu. Jadi, mari kita mulai.

Membangun Bot Horoskop

Dalam contoh membangun chatbot sepenuhnya sendiri menggunakan perpustakaan sumber terbuka Rasa NLU, kita akan membuat Bot Horoskop. Mari kita tentukan cakupan chatbot ini dan lihat apa yang dilakukan dan dapat dilakukannya.

- 1) Bot Horoskop harus dapat memahami salam dan membalas dengan salam.
- 2) Bot harus dapat memahami jika pengguna menanyakan horoskop.
- 3) Bot harus bisa menanyakan tanda horoskop pengguna jika pengguna tidak menyediakannya.
- 4) Jika pengguna tidak mengetahui tanda horoskopnya, bot harus menanyakan tanggal lahir (DOB) pengguna dan mencarikannya untuk mereka.
- 5) Bot harus berlangganan/berhenti berlangganan pengguna untuk mendapatkan ramalan bintang harian.
- 6) Bot harus belajar dari respons yang ada untuk merumuskan respons baru.
- 7) Bot harus mampu menangani kesalahan ejaan yang dilakukan pengguna.

Cukup sederhana apa yang seharusnya dilakukan bot kita di sini.

Kemungkinan niat

- a) Maksud Ucapan: Pengguna memulai dengan salam
- b) Dapatkan Maksud Horoskop: Pengguna meminta horoskop
- c) Maksud Horoskop Pengguna: Pengguna memberitahukan tanda horoskop
- d) Maksud DOB Pengguna: Pengguna memberitahukan DOB-nya
- e) Maksud Berlangganan: Pengguna meminta berlangganan

Bergantung pada cara Anda mendesain bot dan menulis kode, Anda mungkin memerlukan atau tidak memerlukan semua maksud di atas atau Anda mungkin perlu menentukan beberapa maksud tambahan juga untuk mencakup semua kasus penggunaan yang kami ingin bot kami lakukan. Seiring dengan kemajuan kami dalam bab ini, kami akan mencoba membuat bot yang melakukan tugas dasar memberikan horoskop; nanti sebagai bagian dari latihan, kita bisa menambahkan lebih banyak kasus penggunaan ke dalamnya. Ya, inilah yang telah kami lakukan sejauh ini untuk mengenali kemungkinan maksud berdasarkan kemungkinan percakapan sederhana antara bot dan pengguna.

Skrip Percakapan Antara Bot Horoskop dan Pengguna

Mari buat skrip percakapan yang memungkinkan antara chatbot kita dan pengguna. Seperti disebutkan di bab pertama, menulis skrip percakapan seperti menyiapkan dokumen persyaratan perangkat lunak sebelum membangun perangkat lunak. Ini mungkin berubah

nanti, tetapi memberi Anda gambaran keseluruhan tentang apa yang akan terjadi di aplikasi chatbot akhir.

Berikut percakapan kami antara pengguna dan bot:

Pengguna : Halo

Bot Horoskop : Halo! Semoga Anda baik-baik saja.

Pengguna : Saya ingin tahu horoskop saya.

Bot Horoskop : Tentu, bolehkah saya mengetahui tanda horoskop Anda?

Pengguna : Capricorn

Bot Horoskop : Maaf, maksud Anda Capricorn?

Pengguna : Ya.

Bot Horoskop : Baiklah, Ini ramalan bintang Anda hari ini. Kemungkinannya adalah kamu akan merasa optimis dan positif hari ini. Bagikan energi ini dengan orang lain yang merasa kurang puas. Sikap Anda dapat berdampak besar pada teman, pasangan, pasangan, dan anak-anak. Tetangga dan keluarga besar dapat menangkap semangat Anda tanpa Anda sadari. Jika seseorang terluka atau marah, berikan semangat. Semuanya pada akhirnya akan berhasil.

Bot Horoskop : Apakah Anda ingin berlangganan pembaruan harian? Jika ya, katakan saja berlangganan.

Pengguna : Ya, silakan berlangganan.

Bot Horoskop : Bagus, Anda berlangganan sekarang. Semoga harimu menyenangkan.

Percakapan ini hanya untuk mendapatkan gambaran yang adil tentang bagaimana percakapan chatbot kita akan terlihat. Kita mungkin ingin atau tidak ingin mengubah alur pembicaraan berdasarkan kebutuhan kita.

Kita dapat melatih model chatbot kita sendiri untuk menyiapkan respons yang valid alih-alih menulis sekumpulan pernyataan if...else.

5.7 Mempersiapkan Data untuk Chatbot

Rasa NLU memiliki banyak cara untuk menentukan maksud dan entitasnya pada data khusus kami. Ini mendukung data dalam penurunan harga, dalam JSON sebagai satu file, atau sebagai direktori yang berisi banyak file.

Kita akan membahas metode yang paling sulit, namun sangat terukur terlebih dahulu. Membuat file JSON sulit dilakukan secara manual tetapi secara terprogram sangat mudah dan terukur.

Membuat Data untuk Model dalam Format JSON

Format data JSON yang diharapkan Rasa NLU memiliki objek tingkat atas yang disebut `rasa_nlu_data`, dengan kunci `common_examples`, `entitas_synonyms`, dan `regex_features`. Hal terpenting yang akan kita kerjakan adalah `contoh_umum`. Berikut ini adalah bentuk kerangka tampilan data JSON kita:

```
{
  "rasa_nlu_data": {
    "common_examples": [],
    "regex_features": [],
    "entity_synonyms": []
  }
}
```

Kunci `common_examples` dalam data JSON kita adalah tempat sentral yang akan digunakan untuk melatih model kita. Kami akan menambahkan semua contoh pelatihan kami di array `common_examples`.

`regex_features` adalah alat untuk membantu pengklasifikasi maksud mengenali entitas atau maksud dan meningkatkan keakuratan klasifikasi maksud.

Mari mulai menulis file JSON kita. Sebut saja `data.json`.

1. Buat folder bernama `horoscope_bot`.
2. Ubah direktori kerja saat ini menjadi `horoscope_bot`.
3. Mulai Buku Catatan Jupyter #buku catatan jupyter.
4. Buat folder baru bernama `data`.
5. Klik pada folder `data` dan buka "File Teks" di bawah menu Baru di Jupyter Notebook.
6. Klik pada nama file yang dibuat dan ubah namanya menjadi `data.json` dan tulis maksud Anda untuk chatbots Anda.

Untuk Langkah 5 dan 6, silakan gunakan editor favorit Anda seperti Sublime, Notepad++, PyCharm, dll., untuk bekerja dengan file JSON.

Berikut tampilan `data.json` saya di bawah folder `data`:

```
{
  "rasa_nlu_data": {
    "common_examples": [
      {
        "text": "Hello",
        "intent": "greeting",
        "entities": []
      },
      {
        "text": "I want to know my Horoscope",
        "intent": "get_horoscope",
        "entities": []
      },
      {
        "text": "Can you please tell me my horoscope?",
        "intent": "get_horoscope",
        "entities": []
      },
      {
        "text": "Please subscribe me",
        "intent": "subscription"
      }
    ],
    "regex_features": [],
    "entity_synonyms": []
  }
}
```

Seperti yang Anda lihat, kelihatannya sangat kikuk untuk menyiapkan ini dengan tangan. Anda pasti ingat metode bagus dan mudah yang kami miliki di Dialogflow. Jadi, mari kita periksa alat keren dan menarik untuk membuat data pelatihan dalam format yang diharapkan Rasa. Itu dibuat oleh Polgár András, dan juga cukup bagus untuk memeriksa dan memodifikasi data yang sudah kita siapkan sebelumnya. Alat ini menghemat banyak waktu jika Anda mengerjakan proyek kecil di mana Anda harus membuat datanya dengan tangan. Itu selalu merupakan ide bagus untuk memvisualisasikan data dalam aplikasi apa pun yang Anda buat yang sepenuhnya berbasis data. Jadi, simpan saja file data.json yang kita buat sebelumnya hingga kita memperluas pengumpulan data menggunakan metode yang lebih baik.

5.7 Memvisualisasikan Dan Memodifikasi Format Data JSON Rasa

Di bagian ini kita akan menggunakan alat yang disebut pelatih Rasa NLU untuk memvisualisasikan data kita. (yaitu, data yang telah kita buat sejauh ini). Alat ini juga membantu kami dalam membuat anotasi data. Jika Anda ingat ketika antarmuka Dialogflow dijelaskan di Bab 3, sangat mudah untuk mendefinisikan entitas, nama, dan tipenya. Kami akan melakukan hal yang sama menggunakan alat sumber terbuka.

Pelatih Rasa NLU adalah alat yang sangat bagus dan berguna untuk mengedit data pelatihan kami langsung dari browser kami sendiri. Menangani data JSON itu rumit dan juga bisa menyebabkan kesalahan. Dengan alat praktis ini kita dapat dengan mudah menambahkan lebih banyak contoh ke data pelatihan atau mengedit contoh yang sudah ada. Ini menghemat banyak waktu dari membuat anotasi data secara manual. rasa-nlu-trainer adalah alat berbasis javascript, jadi kita perlu menginstal node.js untuk menjalankan alat ini di sistem kita. Tidak perlu lebih dari 5 menit untuk melakukannya. Mari siapkan ini dengan mengikuti langkah-langkah berikut:

1. Buka <https://www.npmjs.com/get-npm> dan unduh node.js.
2. Instal paket pada sistem Anda sesuai panduan di situs web. Setelah terinstal, buka antarmuka terminal/baris perintah baru di sistem Anda dan ketik “npm” untuk melihat apakah itu berfungsi.

Saya telah menginstal LTS versi 8.11.4. Setelah terinstal, jalankan perintah berikut untuk menginstal rasa-nlu-trainer:

Sudo npm I -g rasa-nlu-trainer

Setelah instalasi perintah ini berhasil, Anda akan melihat log yang mirip dengan berikut:

```
[fsevents] Success: "/usr/local/lib/node_modules/rasa-nlu-trainer/
node_modules/fsevents/lib/binding/Release/node-v57-darwin-x64/fse.node"
already installed
Pass --update-binary to reinstall or --build-from-source to recompile
npm WARN slick-carousel@1.8.1 requires a peer of jquery@>=1.8.0 but none is
installed. You must install peer dependencies yourself.

+ rasa-nlu-trainer@0.2.7
added 492 packages in 10.14s
```

Meskipun pesan Anda tidak terlihat seperti ini, selama tidak menimbulkan kesalahan apa pun, jangan khawatir. Kami akan segera mengetahui apakah rasa-nlu-trainer kami berhasil diinstal dan berfungsi dengan baik.

Mari masuk ke folder data yang kita buat sebelumnya di terminal dan jalankan perintah berikut:

Rasa-nlu-trainer

Mengetik perintah ini akan mengaktifkan server lokal pada port 55703 dan membukanya di browser default. Ini akan terlihat seperti Gambar 5-1.



Gambar 5.1. rasa-nlu-pelatih di localhost

Seperti yang Anda lihat, semua data kami yang ada dari data.json diambil oleh alat luar biasa ini untuk kami hapus atau edit dan kami juga dapat menambahkan contoh baru dari sini dan itu akan terus memperluas file data.json.

Saya menyarankan Anda menambahkan lebih banyak data ke maksud Anda untuk pelatihan model yang lebih baik. Anda bisa mendapatkan data.json ini dalam kode sumber zip atau repo yang disediakan oleh penerbit, jika Anda mencoba membuat chatbot yang sama seperti yang dijelaskan dalam bab ini.

Sama seperti kita memilih entitas dalam ucapan untuk didefinisikan di Bab 3 menggunakan Dialogflow, kita dapat melakukan hal yang sama menggunakan alat ini dan memberi nama pada entitas kita untuk diurai nanti. Jadi, klik tombol sakelar pada contoh, pilih teks, dan tambahkan entitas dengan memberinya nama.

Saya telah menambahkan lima hingga enam contoh ucapan di setiap maksud yang telah saya definisikan. Semakin banyak contoh yang kami tambahkan, semakin baik model tersebut dilatih dan memberikan akurasi yang lebih baik. Jika Anda melihat file data.json sekarang, akan ada lebih banyak contoh yang ditambahkan secara otomatis ke dalamnya. Jadi, lanjutkan dan verifikasi file data.json Anda untuk melihat apakah Anda memiliki semua contoh tambahan dari UI rasa-nlu-trainer.

Anda juga akan melihat di file data.json bahwa entitas yang mungkin telah Anda definisikan menggunakan UI rasa-nlu-trainer ditangkap dalam daftar `common_examples` karena memiliki kunci awal dan akhir, yang memberi tahu model pada titik mana nilai entitas tertentu dimulai pada contoh dan kapan berakhir.

Objek kamus yang sama juga menggambarkan nilai entitas dan nama entitas yang kita definisikan. Sebagai contoh kita, tampilannya seperti berikut:

```
{
  "text": "19-01",
  "intent": "dob_intent",
  "entities": [
    {
      "start": 0,
      "end": 2,
      "value": "19",
      "entity": "DD"
    },
    {
      "start": 3,
      "end": 5,
      "value": "01",
      "entity": "MM"
    }
  ]
}
```

Melatih Model Chatbot

Pada bagian ini kita akan melatih model pada data yang telah kita siapkan. Saat kita menggunakan Jupyter Notebook untuk pembuatan dan pengelolaan file, kita akan membuat .ipynb baru dan mulai menulis kode Python untuk melatih model kita dengan memilih salah satu pipeline yang telah kita bahas sebelumnya di bab ini.

Membuat File Konfigurasi

Mari buat file JSON lagi dengan cara yang sama seperti yang kita buat sebelumnya menggunakan Jupyter dan beri nama config.json. Mari kita simpan di luar folder data kita (yaitu, di horoscope_bot yang merupakan direktori proyek kita).

Tambahkan konfigurasi berikut ke dalamnya:

```
{
  "pipeline": "tensorflow_embedding",
  "path": "./models/nlu",
  "data": "./data/data.json"
}
```

5.8 Konfigurasi Yaml

Seperti yang Anda lihat, ada beberapa parameter konfigurasi penting yang dilakukan di file config.json kami. Mari kita coba memahaminya.

- pipeline: Pipeline akan menentukan featureizer atau ekstraktor fitur apa yang akan digunakan untuk mengolah pesan teks dan mengekstrak informasi yang diperlukan. Dalam kasus kami, kami menggunakan tensorflow_embedding.
- path: path pada dasarnya adalah direktori tempat kita menyimpan model setelah pelatihan. Kami akan menyimpan model kami di folder /models/nlu.
- data: data adalah jalur yang perlu kita tentukan; ini pada dasarnya adalah tempat data pelatihan kami berada.

Setelah kita selesai dengan file config.json, mari beralih ke beberapa kode Python untuk melatih model ML kita.

Anda juga dapat menggunakan file .yml sebagai file konfigurasi seperti di bawah ini. Anda bisa mendapatkan contoh file config.yml di repo github.

- **Example 1:**

```
language: "en"
pipeline: "tensorflow_embedding"
```

- **Example 2:**

```
language: "en"
pipeline:
  - name: "nlp_spacy"
  - name: "tokenizer_spacy"
  - name: "intent_entity_featurizer_regex"
  - name: "intent_featurizer_spacy"
  - name: "ner_crf"
  - name: "ner_synonyms"
  - name: "intent_classifier_sklearn"
```

Semua pesan masuk diproses sesuai urutan komponen yang ditentukan. Komponen yang ditentukan dijalankan secara berurutan, satu per satu, dan karenanya disebut pipa pemrosesan. Komponen yang berbeda digunakan untuk tujuan yang berbeda, seperti ekstraksi entitas, klasifikasi maksud, pra-pemrosesan, dll. Keuntungan dari format seperti itu adalah kita dapat menentukan saluran pipa yang telah ditentukan sebelumnya oleh Rasa dengan jelas.

5.9 Menulis Kode Python untuk Melatih Model dan Memprediksi

Mari buka file .ipynb baru dan mulai menulis kode kita. Beri nama ipynb sebagai rasa-nlu.ipynb. Pastikan Anda sudah berhasil menginstal rasa-nlu==0.13.2 untuk versi Python yang Anda gunakan.

Berikut ini tampilan kode kita menggunakan data.json dan config.json dengan Python dan melatih model menggunakan pipeline tensorflow_embedding.

```
from rasa_nlu.training_data import load_data
from rasa_nlu.model import Trainer
from rasa_nlu import config
from rasa_nlu.model import Interpreter

def train_horoscopebot(data_json, config_file, model_dir):
    training_data = load_data(data_json)
    trainer = Trainer(config.load(config_file))
    trainer.train(training_data)
    model_directory = trainer.persist(model_dir, fixed_model_name =
    'horoscopebot')
```

```
def predict_intent(text):
    interpreter = Interpreter.load('./models/nlu/default/horoscopebot')
    print(interpreter.parse(text))
```

Di bagian pertama kode, kami mengimpor semua perpustakaan yang diperlukan dari paket `rasa_nlu`. Kemudian kita mendefinisikan dua metode yang disebut `train_horoscopebot` dan `predict_intent`, di mana metode pertama melatih model dengan data yang diberikan, `config_file`, dan `model_directory` (tempat menyimpan model) dan metode `predict_intent` menggunakan model `Interpreter` dari `rasa_nlu` untuk memuat file model yang telah dilatih sebelumnya dan memberi pengguna kemampuan untuk memprediksi contoh teks baru.

5.10 Melatih Model

Kami menjalankan cuplikan di bawah ini untuk memanggil metode `train_horoscopebot` kami dengan parameter masing-masing

```
train_horoscopebot('./data/data.json', 'config.json', './models/nlu')
```

Setelah menjalankan kode ini di `rasa-nlu.ipynb`, kita akan mendapatkan output seperti ini:

```
Epochs: 100% ██████████ 300/300 [00:01<00:00, 175.69it/s,
loss=0.075, acc=1.000]
```

Kode untuk melatih model chatbot akan membuat folder model, yang dapat Anda lihat menggunakan Jupyter atau menggunakan aplikasi file explorer atau finder Anda. Ini akan membuat banyak file indeks, meta, dan acar di tujuan direktori model yang kami sediakan.

Memprediksi Dari Model

Mari kita panggil metode `predict_intent` dengan meneruskan teks untuk melihat kinerja model terlatih kita.

```
predict_intent("I am looking for my horoscope for today. I am wondering if
you can tell me that.")
```

Metode itu sendiri mencetak hasilnya. Untuk teks di atas, keluaran saya terlihat seperti berikut:

```
{
  "intent": {
    "name": "get_horoscope",
    "confidence": 0.9636583924293518
  },
  "entities": [],
  "intent_ranking": [
    {
      "name": "get_horoscope",
      "confidence": 0.9636583924293518
    },
    {
      "name": "dob_intent",
      "confidence": 0.03462183475494385
    }
  ]
}
```

```

    },
    {
      "name": "greeting",
      "confidence": 0
    },
    {
      "name": "subscription",
      "confidence": 0
    }
  ],
  "text": "I am looking for my horoscope for today. I am wondering if you
  can tell me that."
}

```

Wow! Bukankah ini ajaib? Model kami telah memprediksi teks ini dengan tingkat keyakinan 96 persen. Anda dapat melihat di file `ipynb` yang disediakan bahwa model kami juga berfungsi dengan baik dalam memprediksi maksud lainnya. Inilah kekuatan `tensorflow` dan `ML` secara keseluruhan. Tak perlu dikatakan lagi, perpustakaan `rasa_nlu` membuatnya begitu mudah untuk dipercaya. Jadi, inilah saatnya Anda kembali melakukan retrospeksi, dan jika Anda mengingat Bab 3 buku ini, maka Anda harus ingat setiap kali kita menambahkan contoh baru, `Dialogflow` digunakan untuk melatih ulang model tersebut. Mereka sebenarnya melakukan hal yang sama seperti yang baru saja kami lakukan, di belakang layar. Kami tidak dapat mengubah model atau menyetel parameter apa pun di sana, namun kami dapat melakukan semuanya sekarang dengan kontrol penuh.

Sekarang kita telah berhasil membuat dan melatih model menggunakan `tensorflow` dan mengujinya juga, kita akan beralih ke topik Manajemen Dialog berikutnya. Saya akan meminta Anda menguji semua skenario yang mungkin dihadapi bot Anda sehingga Anda mengetahui titik-titik di mana model Anda tidak berkinerja baik dan, oleh karena itu, Anda dapat menambahkan lebih banyak data atau menyesuaikan parameter jika diperlukan.

Perlu diingat juga bahwa Anda hanya perlu melatih ulang model setiap kali ada perubahan pada data pelatihan. Jika tidak ada perubahan pada data pelatihan, kita dapat memuat model pelatihan yang ada untuk terus melakukan prediksi pada contoh baru.

5.11 Manajemen Dialog Menggunakan Rasa Core

Di bagian ini kita akan memulai dengan melatih model lain untuk manajemen dialog `Rasa Core`. Ingat, saat ini kita memiliki model yang siap memprediksi maksud teks, kita bisa menulis beberapa kode `Python` untuk merumuskan tanggapan, dan kita bisa membalas kembali ke pelanggan. Tapi bagaimana jika kita ingin menambahkan lebih banyak maksud ke bot kita? Apakah itu dapat diskalakan untuk aplikasi yang lebih besar dengan banyak fitur? Jawabannya adalah tidak. Di sinilah manajemen dialog dari `Rasa Core` membantu.

Jika Anda pernah mencoba menggunakan bot apa pun di platform apa pun, Anda pasti melihatnya gagal dalam kondisi tertentu. Ya, kita semua pernah mengalaminya dan masih ada karena bot saat ini gagal mengelola konteks percakapan dengan buruk atau

mengikuti alur percakapan. Dengan bantuan kerangka dialog berbasis ML Rasa Core, kami dapat memperbaiki masalah ini dengan mudah.

Rasa Core telah terbukti baik untuk aplikasi tingkat perusahaan dan digunakan oleh ribuan pengembang karena siap produksi, mudah digunakan dan diperluas, dan yang paling penting, open source.

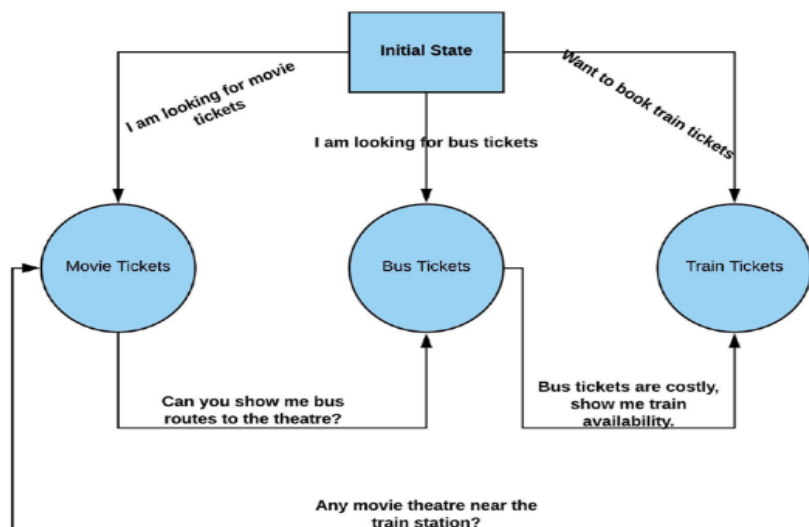
Memahami Lebih Lanjut Inti Rasa dan Sistem Dialog

Sebelum kita benar-benar beralih ke bagian pengkodean Rasa Core untuk model manajemen dialog kita, sangat penting untuk memahami mengapa dan dari mana hal ini berasal. Kami akan mencoba memahami bagaimana kami telah melakukan hal-hal sejauh ini untuk membangun chatbots dan bagaimana hal ini akan berubah selamanya.

Mari kita ambil contoh:

Jika kita membuat chatbot sederhana yang akan membantu pengguna memesan tiket penerbangan/bus/film/kereta api, cara termudah adalah dengan membuat mesin status atau pohon keputusan, menulis sekumpulan if...else, dan itu akan selesai. Ini akan berhasil tetapi tidak akan berkembang. Jika pelanggan pada awalnya memiliki pengalaman yang baik dengan sesuatu, mereka ingin menggunakannya lebih sering. Dengan beberapa heuristik, kita dapat menunjukkan bahwa chatbot cerdas tetapi tidak untuk waktu yang lama. Ketika aliran kontrol kode berpindah dari blok coba ke blok kecuali, kita mulai menggaruk-garuk kepala.

Gambar 5-2 adalah representasi sederhana tentang tampilan mesin negara untuk membangun chatbot ini.



Gambar 5.2. Representasi diagram status untuk chatbot pemesanan tiket

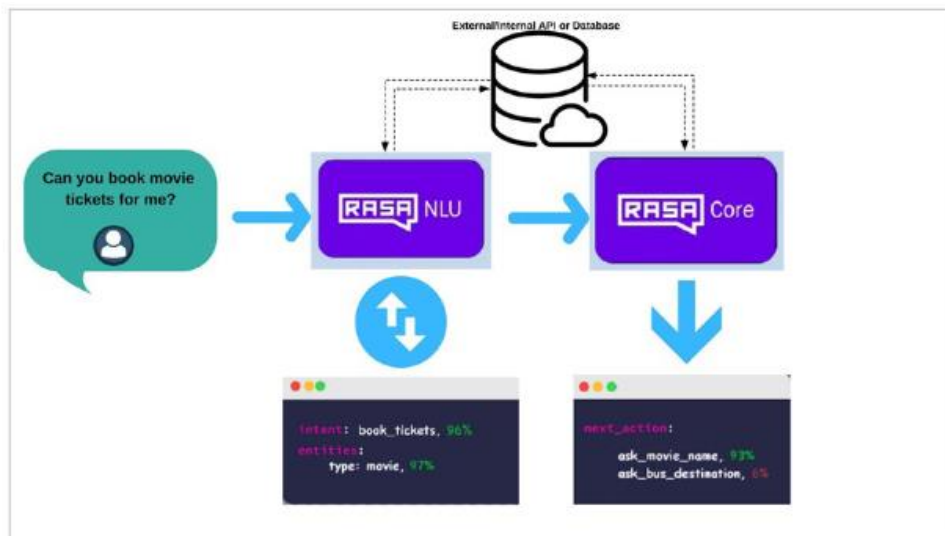
Jika kita melihat diagram keadaan kita, ini mungkin berfungsi untuk percakapan normal di mana pengguna sedang mencari tiket film, bus, atau kereta api atau ingin memesan tiket bus setelah meminta tiket film. Bagaimana jika pengguna meminta tiket bus dan bioskop secara bersamaan? Anda mungkin mengatakan bahwa kita dapat menambahkan beberapa pernyataan if...else lagi dalam kode yang sudah disarangkan untuk

mengatasinya. Jika Anda seorang pengembang yang baik, Anda tidak perlu banyak waktu untuk menulis masukan dari mesin negara atau memperluas pohon keputusan Anda. Tapi pikirkan situasi ketika kondisi ini mulai berkembang secara eksponensial dan Anda harus terus menambahkan kasus untuk mengatasinya dan mereka juga mulai saling mengganggu.

Otak kita bekerja dengan cara kita belajar dan belajar kembali. Jika seorang anak tidak tahu dampak api terhadap dirinya, mereka akan menyentuhnya, namun jika api itu menyakitinya, mereka tidak akan melakukannya.

lagi. Mereka memperkuat fakta bahwa hal itu berbahaya. Cara kerjanya serupa dalam hal imbalan ketika Anda melakukan sesuatu dan mendapatkan sesuatu, Anda mengasosiasikan fakta bahwa melakukan sesuatu mendatangkan imbalan atau imbalan yang lebih baik dan kemudian Anda berniat melakukannya lagi. Hal ini disebut pembelajaran penguatan dalam ML, di mana mesin mempelajari cara berperilaku dalam situasi tertentu dengan melakukan tindakan dan memahami hasilnya. Pembelajaran penguatan terkadang bukan pendekatan terbaik, seperti dalam situasi di mana data tidak cukup untuk dipelajari, kualitas data tidak baik untuk mempelajari skenario penghargaan, dll.

Gambar 4-3 adalah diagram untuk membantu Anda memahami bagaimana Rasa Core cocok dengan Rasa Stack dan perannya bersama Rasa NLU, yang telah kita pelajari sejauh ini.



Gambar 5.3. Diagram yang mewakili cara kerja Rasa NLU dan Rasa Core

Dengan Rasa Core, kita dapat menentukan semua hal yang seharusnya dikatakan atau dilakukan bot kita. Ini disebut tindakan. Salah satu tindakan yang mungkin dilakukan adalah mengucapkan salam kepada pengguna atau menanyakan database, atau bisa juga mengambil beberapa informasi menggunakan beberapa layanan web atau API. Rasa Core menyediakan fasilitas untuk melatih model probabilistik kami untuk memprediksi tindakan apa yang harus diambil berdasarkan riwayat percakapan pengguna atau user. Bayangkan melakukannya tanpa model ML dan menulis ratusan kasus penggunaan untuk mencari tahu respons atau tindakan apa yang harus dikirim atau dijalankan. Secara sederhana, Rasa Core memecahkan masalah ini untuk Anda.

Mari kita lihat percakapan pengguna vs. bot secara berdampingan yang kita buat di Bab 3 saat membuat OnlineEatsBot.

User	OnlineEatsBot
Hello OnlineEats	Hello! How may I help you?
I am looking to order food	Sure, What would you like to order today?
One chicken burger	Done. Your final amount is \$3.99
Thank you	Thanks a lot

Percakapan ini cukup sederhana dan dapat dilakukan tanpa model ML apa pun, meskipun kami menggunakan model terlatih untuk mengenali maksud dan entitas di dalamnya.

Nah, pertanyaan menarik yang terlintas di benak Anda adalah, “Bagaimana jika saya ingin bot saya bisa mengubah urutannya? Bagaimana jika saya ingin bot saya dapat menghapus atau menambahkan lebih banyak item makanan?” Jadi, seperti yang telah dibahas, tambahkan lebih banyak kasus penggunaan, lebih banyak kompleksitas, lebih banyak maksud dan ucapannya, lebih banyak if...else dalam kode untuk menangani kasus-kasus sudut, namun ketika Anda membangun chatbot untuk sebuah bisnis, Anda harus menskalakannya untuk menghasilkan lebih banyak kasus. pendapatan melaluinya. Semua sistem perangkat lunak melakukan hal itu, dan mereka yang tidak gagal untuk bertahan hidup. Intinya adalah, kami tidak bisa terus-menerus mengubah dan menerapkan kode.

Jadi, daripada menangani semua kasus yang terjadi dan melakukannya berulang kali, kita dapat mengambil rute ML untuk mengatasi masalah ini, dengan bantuan Rasa Stack, yang akan memberi tahu kita apa yang harus dilakukan bot selanjutnya berdasarkan konteks pengguna dan keadaan percakapan. Karena model belajar sendiri berdasarkan konteks data percakapan sebelumnya, bot menjadi lebih mudah untuk menjaga percakapan lebih alami dan ramah pengguna daripada memilih empat hingga lima kalimat tetap secara acak.

Rasa merekomendasikan pengguna dengan sedikit atau tanpa data untuk menggunakan pembelajaran interaktif. Kita akan mengetahui lebih banyak tentang pembelajaran interaktif nanti di bab ini. Sebelum kita benar-benar mulai menggunakan Rasa Stack untuk menulis bagian inti bot kita, pertama-tama kita perlu memahami beberapa konsep.

5.12 Memahami Konsep Rasa

Sangat penting bagi kita untuk memahami beberapa konsep spesifik terkait Rasa sebelum benar-benar mencoba menggunakannya dalam kode sepenuhnya. Di bagian ini kita akan mempelajari beberapa konsep Rasa NLU yang penting dan sangat berguna. Pastikan Anda memahami konsep ini sepenuhnya, karena kami akan menggunakannya dalam membangun chatbot internal pertama kami menggunakan format file domain Rasa. Jika kita tidak memahami maksud dari konsep-konsep ini, maka akan sulit untuk mengambil langkah selanjutnya.

Tindakan

Seperti namanya, ini adalah tindakan spesifik yang bisa diambil. Sesuai dengan dokumentasi Rasa, dikatakan, “Tindakan selanjutnya harus diambil sebagai respons terhadap keadaan dialog.” Misalnya, jika pengguna menanyakan horoskop hari ini, bot kami dapat menjalankan tindakan “GetToday’sHoroscope”. Mari kita lihat bagaimana tindakan “GetToday’sHoroscope” akan terlihat dalam bentuk kode.

```
from rasa_core.actions import Action
from rasa_core.events import SlotSet

class GetToday’sHoroscope(Action):
    def name(self):
        return "get_todays_horoscope"

    def run(self, dispatcher, tracker, domain):
        # type: (Dispatcher, DialogueStateTracker, Domain) -> List[Event]

        user_horoscope_sign = tracker.get_slot('horoscope_sign')
        """Write your logic to get today’s horoscope details
        for the given Horoscope sign based on some API calls
        or retrieval from the database"""

    return [SlotSet("horoscope_sign", user_horoscope_sign)]
```

Metode nama mengembalikan nama Tindakan yang akan kita rujuk dalam file domain sebagai nama metode khusus. Metode run melakukan tugas utama untuk menyelesaikan tindakan—yaitu, logika bisnis inti berada di sini. Seperti yang Anda lihat, dibutuhkan tiga parameter: pengiriman, pelacak, dan domain.

Mari kita pahami parameter ini satu per satu:

- *Dispatcher*: Dispatcher digunakan untuk mengirim pesan kembali ke pengguna kami. Kita dapat menggunakan `dispatcher.utter_message()` untuk mencapai hal yang sama.
 - *pelacak*: pelacak status untuk pengguna saat ini. Kita dapat mengakses nilai slot menggunakan `tracking.get_slot(slot_name)`, dan untuk mendapatkan pesan pengguna terbaru kita dapat menggunakan `tracking.latest_message.text`.
 - *domain*: domain bot. Kami akan membahas lebih detail tentang domain nanti di bab ini.
- Metode eksekusi catatan mengembalikan daftar contoh Peristiwa.

5.13 Slot

Slot lah yang membuat bot bertanggung jawab untuk bekerja seperti manusia. Slot seperti ruang penyimpanan tempat informasi yang diberikan oleh pengguna dapat disimpan atau informasi yang diambil sebelumnya dari database atau API juga dapat digunakan.

Ada jenis slot yang berbeda untuk kasus penggunaan yang berbeda:

Misalnya, dalam kasus penggunaan di mana kita ingin membuat HoroscopeBot, kita mungkin ingin menggunakan tipe slot sebagai teks untuk `horoscope_sign` yang disediakan oleh

pengguna. Tergantung pada jenis slot yang ingin Anda simpan, Rasa menyediakan beberapa jenis slot yang telah ditentukan sebelumnya.

Selain teks, Rasa memiliki jenis slot berikut:

- Boolean ⇒ Digunakan untuk Benar/Salah
- Kategorikal ⇒ Digunakan untuk situasi di mana Anda harus memilih salah satu dari beberapa nilai
- Float ⇒ Digunakan untuk nilai kontinu
- Daftar ⇒ Digunakan untuk daftar nilai
- Unggulan ⇒ Digunakan untuk penyimpanan nilai internal yang tidak memengaruhi percakapan

5.14 Templat

Templat adalah kata yang pasti pernah Anda dengar setidaknya sekali dalam hidup Anda, saat mencari templat untuk mengirim email, untuk menyiapkan dokumen, untuk membuat situs web portofolio, atau untuk mengikuti suatu proses. Templat di Rasa digunakan untuk ucapan. Templat ucapan berisi sekumpulan teks prasetel yang akan dikirim ke pengguna saat beberapa tindakan dipicu. Dengan memiliki nama tindakan yang sama dengan ucapan atau tindakan dengan kode khusus, kami dapat mengirimkan pesan berformat dalam templat kepada pengguna.

Representasi sederhana dari templat di dalam file domain mungkin terlihat seperti berikut:

```
templates:
  utter_greet:
    - "hello {name}!" # name can be filled by a slot of same name or by
      custom code
  utter_goodbye:
    - "goodbye"
    - "take care bye" # multiple templates allow the bot to randomly pick
      from them
  utter_default:
    - "Sorry, I didn't get that."
```

Sekarang, setelah kita mempelajari tiga konsep tindakan, slot, dan templat serta mengetahui maksud dan entitas apa yang menjadi bagian dari pembelajaran kita dari Bab 3, sekarang kita siap untuk mendalami Rasa dan mulai menulis kode untuk kita chatbot pertama.

5.15 Membuat File Domain untuk Chatbot

Tugas pertama dan terpenting yang harus dilakukan saat membuat chatbot menggunakan Rasa stack adalah membuat file domain. Sesuai dokumentasi Rasa, "Domain mendefinisikan alam semesta tempat bot Anda beroperasi. Ini menentukan maksud, entitas, slot, dan tindakan yang harus diketahui bot Anda. Secara opsional, ini juga dapat menyertakan template untuk hal-hal yang dapat dikatakan oleh bot Anda."

Sekarang Anda tahu mengapa kami harus bersiap menghadapi situasi ini sebelumnya dengan memahami konsep inti Rasa. Mari buat file DefaultDomain dengan definisi YAML. Rasa menggunakan file .yml untuk menentukan format domain.

Awalnya YAML seharusnya berarti Yet Another Markup Language, yang menggambarkan penggunaannya sebagai bahasa markup, namun kemudian diubah menjadi dipahami sebagai YAML Bukan Bahasa Markup, sebuah akronim rekursif, untuk membedakan tujuannya sebagai berorientasi data dan bukan sebagai markup dokumen bahasa.

Sekarang, mari kembali ke direktori rasa-nlu Jupyter Notebook dan mulai membuat file. Perhatikan bahwa kita dapat menulis semua kode kita dalam file terpisah menggunakan baris perintah dan menggunakan editor untuk mengeditnya. Menurut saya Jupyter Notebook lebih interaktif dan mudah diakses untuk menelusuri file. Apa pun yang Anda rasa nyaman, lakukan saja, tetapi ada baiknya mengetahui sebagian besar fitur yang disediakan Jupyter Notebook.

Buka direktori utama horoscope_bot dan buat file, sebut saja

Horoscope_domain.yml.

Berikut isi horoscope_domain.yml untuk bot kita:

```
slots:
  horoscope_sign:
    type: text
  DD:
    type: text
  MM:
    type: text

  subscribe:
    type: bool

intents:
  - greeting
  - get_horoscope
  - subscription
  - dob_intent

entities:
  - horoscope_sign
  - DD
  - MM
  - subscribe
  - dob_intent
```

```

templates:
  utter_greet:
    - 'Hello! How are you doing today?'
  utter_ask_horoscope_sign:
    - 'What is your horoscope sign?'
  utter_ask_dob:
    - 'What is your DOB in DD-MM format?'
  utter_subscribe:
    - 'Do you want to subscribe for daily updates?'

actions:
  - utter_greet
  - utter_ask_horoscope_sign
  - utter_ask_dob
  - utter_subscribe
  - get_todays_horoscope
  - subscribe_user

```

Seperti yang Anda lihat, file domain terdiri dari lima bagian penting: maksud, entitas, slot, templat, dan tindakan, yang telah kita bahas sebelumnya. Perlu diperhatikan bahwa untuk setiap templat, ada pengucapanAction yang ditentukan, seperti ucapan_sapa, ucapan_ask_horoscope_sign, dan ucapan_ask_dob, kita harus memiliki templat yang ditentukan di bagian templat dengan nama yang sama.

Seperti yang bisa Anda lihat dalam contoh kita, ada lima tindakan yang ditentukan, di mana tiga tindakan pertama hanya untuk mengucapkan teks templat kepada pengguna, namun dua tindakan terakhir mengharuskan kita mengambil data dari database atau membuat API panggilan untuk mendapatkan horoskop hari itu dan mengembalikannya ke pengguna.

Dalam kasus tindakan subscribe_user juga, kita perlu melakukan operasi di mana kita menambahkan pengguna saat ini ke daftar berlangganan di database. Tindakan yang ditentukan pengguna ini disebut tindakan kustom. Untuk mendapatkan tindakan khusus seperti itu, kita perlu menulis apa yang seharusnya dilakukan bot ketika tindakan ini dipicu. Di bagian selanjutnya kita akan mempelajari cara menulis tindakan kustom.

5.16 Menulis Tindakan Kustom dari Chatbot

Seperti yang kita ketahui, setiap kali UtterAction dipicu, bot kita akan merespons kembali dengan teks ucapan yang ditentukan dalam template untuk Action tersebut. Namun apa yang terjadi jika beberapa tindakan kustom dipicu? Di bagian ini kita akan menulis kode Python untuk membuat tindakan khusus, yang bisa kita gunakan untuk membuat panggilan API atau hampir semua hal yang bisa Anda lakukan dengan Python.

Mari buat file baru bernama action.py di direktori proyek kita (yaitu, dalam kasus kita di dalam folder horoscope_bot).

```

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals

import requests
from rasa_core_sdk import Action
from rasa_core_sdk.events import SlotSet

class GetTodaysHoroscope(Action):

    def name(self):
        return "get_todays_horoscope"

    def run(self, dispatcher, tracker, domain):
        # type: (Dispatcher, DialogueStateTracker, Domain) -> List[Event]

        user_horoscope_sign = tracker.get_slot('horoscope_sign')
        base_url = http://horoscope-api.herokuapp.com/horoscope/{day}/{sign}
        url = base_url.format(**{'day': "today", 'sign': user_horoscope_
            sign})
        #http://horoscope-api.herokuapp.com/horoscope/today/capricorn
        res = requests.get(url)
        todays_horoscope = res.json()['horoscope']
        response = "Your today's horoscope:\n{}".format(todays_horoscope)

        dispatcher.utter_message(response)
        return [SlotSet("horoscope_sign", user_horoscope_sign)]

```

Seperti yang bisa kita lihat, ada dua metode dalam tindakan kita yang disebut GetTodaysHoroscope. Nama metode pertama hanya mengembalikan nama tindakan. Metode lainnya dijalankan dan, seperti yang dibahas sebelumnya, merupakan metode yang benar-benar melakukan tugas dengan mengeksekusi logika bisnis yang kita tulis.

Dalam metode kami, kami menggunakan API sumber terbuka dengan kode yang dihosting di github [<https://github.com/tapasweni-pathak/Horoscope-API>] Url APInya terlihat seperti ini:

<http://horoscope-api.herokuapp.com/horoscope/today/capricorn>

yang mengembalikan data dalam format JSON:

```

{
  "date": "2018-08-29",
  "horoscope": "You will be overpowered with nostalgia and may long to get
in touch with old pals. And as Ganesha says, chances are that you may
take a liking to your ex-lover, while simultaneously strengthening your
social standing. All in all, the day will be a productive one.",
  "sunsign": "capricorn"
}

```

Seperti yang Anda lihat di metode run, kami mengonversi respons dari API ke objek Python JSON dan kemudian mengakses kunci 'horoskop' dari JSON untuk mendapatkan horoskop sebenarnya. Setelah mendapatkan horoskop sebenarnya dari JSON, kami

Percakapan Antarmuka Pengguna (User Interface) Dalam Chatbot – Dr. Budi Raharjo

merumuskan respons dan mengirimkannya kembali ke pengguna menggunakan objek operator dan metodenya, `utter_message`.

Terakhir kita atur slotnya menggunakan metode `SlotSet`. `SlotSet` seperti menyimpan variabel yang Anda temukan dari respons pengguna sehingga Anda dapat menggunakannya kapan saja dalam kode Anda selama percakapan berlangsung.

Catatan Dengan menggunakan API di atas kita bisa mendapatkan ramalan bintang hari ini dengan memberikan tanda horoskop. Anda bebas menggunakan API atau database Anda sendiri. Anda hanya perlu mengganti panggilan API dengan beberapa sumber lain yang ingin Anda gunakan.

Sama seperti kami menambahkan tindakan `GetToday'sHoroscope` di file `action.py`, kami juga akan menambahkan tindakan `SubscribeUser`. Kami tidak akan menggunakan database apa pun untuk menyimpan preferensi langganan pengguna, tetapi saat Anda membuat chatbot untuk pengguna sebenarnya, Anda mungkin harus memiliki `user_ids` yang dapat Anda tautkan dengan langganan mereka di database.

Berikut tampilan tindakan `SubscribeUser`:

```
class SubscribeUser(Action):
    def name(self):
        return "subscribe_user"

    def run(self, dispatcher, tracker, domain):
        # type: (Dispatcher, DialogueStateTracker, Domain) -> List[Event]

        subscribe = tracker.get_slot('subscribe')

        if subscribe == "True":
            response = "You're successfully subscribed"
        if subscribe == "False":
            response = "You're successfully unsubscribed"

        dispatcher.utter_message(response)
        return [SlotSet("subscribe", subscribe)]
```

Seperti ini, kita dapat menulis tindakan kustom sebanyak yang diperlukan.

Langkah selanjutnya adalah data. Model manajemen dialog Rasa dilatih berdasarkan percakapan aktual yang dilakukan pengguna dan chatbot. Poin pentingnya di sini adalah percakapan tersebut harus diubah ke dalam format cerita. Sebuah cerita tidak lain hanyalah percakapan aktual antara pengguna dan chatbot di mana masukan pengguna diubah menjadi maksud dan entitas, sementara respons yang dikembalikan dari chatbot diperlakukan sebagai tindakan yang seharusnya dipicu oleh chatbot saat diperlukan. Salah satu contoh bagaimana percakapan nyata antara pengguna dan chatbot terlihat sebagai sebuah cerita diberikan pada tabel di bawah.

Skenario I

User	HoroscopeBot
Hello there!	utter_greet
I want to know my horoscope for today	utter_ask_horoscope_sign
My sign is Capricorn	actions.GetTodaysHoroscope
Can you subscribe me for updates?	actions.SubscribeUser

Skenario II

User	HoroscopeBot
Hello there!	utter_greet
I want to know my horoscope for today	utter_ask_horoscope_sign
I don't know my sign	utter_ask_dob
12-12	actions.GetTodaysHoroscope

Kami belum membahas skenario dalam kode di mana pengguna tidak mengetahui tanda horoskopnya tetapi mengetahui DOB-nya. Di sini, kode kita memerlukan beberapa modifikasi untuk mendapatkan entitas DATE dan MONTH ketika nilai horoscope_sign tidak ditemukan.

Kita dapat menggunakan nilai DD-MM untuk memeriksa tanda horoskop dan kemudian memanggil metode GetTodaysHoroscope secara eksplisit atau melatih model dengan cara tersebut.

5.17 Persiapan Data untuk Pelatihan Bot

Penting untuk selalu memiliki data berkualitas baik sebelum melakukan ML apa pun. Untuk melatih chatbot kami, kami juga membutuhkan data; percakapan antara pengguna dan chatbot adalah data yang kita perlukan untuk melatih model kita. Terkadang menjadi sulit untuk menemukan kumpulan data secara bebas di web yang sesuai dengan kebutuhan kita.

Kita harus meluangkan waktu yang kita perlukan untuk mengumpulkan data. Kami dapat meminta teman dan keluarga kami untuk memberikan contoh teks percakapan tentang bagaimana mereka akan berinteraksi dengan sejenis bot yang Anda buat. Beberapa orang membuat contoh aplikasi untuk hal yang sama dan melakukan crowdsourcing data. Jadi, lebih baik datanya, lebih baik modelnya, dan lebih baik pula respons chatbotnya.

Dalam hal mempersiapkan data, Rasa tidak meninggalkan kebutuhan bisnis yang terlewat dan hadir dengan fitur keren yang disebut pembelajaran interaktif. Ini membantu Anda menghasilkan data cerita dengan mudah dan juga melatih model manajemen dialog saat kami terus menambahkan data cerita. Anda bisa menyebutnya pelatihan ML waktu nyata. Jadi, saat kita terus menambahkan data cerita, kita akan mengetahui apakah model kita menghasilkan keluaran yang benar atau tidak. Yang terpenting, kita bisa melihat apakah modelnya membaik atau menurun ketika kita menambahkan cerita baru. Dalam sebagian besar kasus, hal ini akan menjadi lebih baik karena kita akan melakukan pembelajaran

penguatan di mana kita memerintahkan model ML untuk melupakan dan mempelajari kembali hampir seperti yang dilakukan manusia.

5.18 Membuat Data Cerita

Seperti yang kita ketahui, data cerita hanyalah cara percakapan antara pengguna dan chatbot tentang bagaimana hal itu akan menghasilkan akhir yang logis. Secara umum, semua chatbot dirancang untuk membantu pengguna dengan serangkaian hal yang telah ditentukan sebelumnya; cerita hanya mewakili bagaimana hal itu dilakukan.

Kami akan mencoba menyiapkan beberapa dialog sederhana dalam format yang diharapkan Rasa. Dialog-dialog ini tidak memiliki kewarganegaraan—artinya, dialog-dialog ini tidak bergantung pada dialog-dialog sebelumnya. Kami akan menggunakan cerita tanpa kewarganegaraan buatan tangan kami untuk pembelajaran interaktif. Kami akan meluangkan waktu beberapa menit untuk menyusun sendiri beberapa cerita yang kami ketahui, sehingga kami dapat menyesuaikan diri dengan cara pembuatan data cerita. Pertama mari kita buat file bernama `Stories.md` di folder data kita.

```
## story_001
* greeting
  - utter_greet
* get_horoscope
  - utter_ask_horoscope_sign
* get_horoscope{"horoscope_sign": "Capricorn"}
  - slot{"horoscope_sign": "Aries"}
  - get_todays_horoscope
  - utter_subscribe

## story_002
* greeting
  - utter_greet
* get_horoscope{"horoscope_sign": "Capricorn"}
  - slot{"horoscope_sign": "Cancer"}
  - get_todays_horoscope
  - utter_subscribe
* subscription
  - slot{"subscribe": "True"}
  - subscribe_user

## Horoscope query with horoscope_sign
* greeting
  - utter_greet
```

```

* get_horoscope
  - utter_ask_horoscope_sign
* get_horoscope{"horoscope_sign": "capricorn"}
  - slot{"horoscope_sign": "capricorn"}
  - get_todays_horoscope
  - slot{"horoscope_sign": "capricorn"}
  - utter_subscribe
* subscription{"subscribe": "True"}
  - slot{"subscribe": "True"}
  - subscribe_user
  - slot{"subscribe": true}

## Horoscope with sign provided
* greeting
  - utter_greet
* get_horoscope{"horoscope_sign": "leo"}

  - slot{"horoscope_sign": "leo"}
  - get_todays_horoscope
  - slot{"horoscope_sign": "leo"}
  - utter_subscribe
* subscription{"subscribe": "True"}
  - slot{"subscribe": "True"}
  - subscribe_user
  - slot{"subscribe": true}

## When user directly asks for subscription
* greeting
  - utter_greet
* subscription{"subscribe": "True"}
  - slot{"subscribe": "True"}
  - subscribe_user
  - slot{"subscribe": true}

```

Jika Anda menatap cerita-cerita itu selama beberapa menit, mereka akan menjelaskan kepada Anda apa maksudnya; seharusnya tidak sulit untuk memahami apa yang sedang terjadi. Perbedaan utama antara dua cerita pertama adalah bahwa di cerita pertama pengguna tidak menyebutkan tanda horoskopnya dan bot seharusnya menanyakan tanda horoskop dan kemudian melanjutkan ceritanya.

Pada cerita kedua, pengguna sendiri yang menceritakan tanda horoskop dan kemudian mengakhiri cerita dengan dialog berlangganan. Kami telah menambahkan beberapa cerita lagi yang mencakup lebih banyak kasus penggunaan. Jangan ragu untuk menambahkan cerita Anda sendiri di file yang sama.

Jadi, pada dasarnya Stories adalah file penurunan harga dimana kita bisa menulis cerita sebanyak yang diperlukan dalam format penurunan harga yang ditunjukkan

sebelumnya. Melakukan hal ini tampaknya merupakan tugas yang sulit dengan tangan. Jadi, kami akan mencoba mempelajari cara menggunakan alat pembelajaran interaktif Rasa untuk menghasilkan lebih banyak cerita seperti ini.

Mari kita mulai.

Pembelajaran Interaktif

Sejauh ini kita telah membahas pembelajaran interaktif sedikit demi sedikit, namun inilah saatnya untuk benar-benar menulis beberapa kode dan melakukannya. Pembelajaran interaktif adalah salah satu fitur paling keren yang dimiliki Rasa yang menjadikan bagian ML menyenangkan dan mudah. Ada dua bagian: bagian pertama, kami melatih model dengan memberikan kumpulan data awal menggunakan berbagai kebijakan, dan bagian kedua, kami menguji model, memperbaikinya, dan melatihnya kembali secara interaktif.

5.19 Melatih Model Agen Chatbot

Mari buat file baru bernama `train_initialize.py` di direktori proyek utama kita. Isi `train_initialize.py` terlihat seperti ini:

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals

from rasa_core import utils
from rasa_core.agent import Agent
from rasa_core.policies.keras_policy import KerasPolicy
from rasa_core.policies.memoization import MemoizationPolicy
from rasa_core.policies.sklearn_policy import SklearnPolicy

if __name__ == '__main__':
    utils.configure_colored_logging(loglevel="DEBUG")

    training_data_file = './data/stories.md'
    model_path = './models/dialogue'

    agent = Agent("horoscope_domain.yml",
                  policies=[MemoizationPolicy(), KerasPolicy()])

    training_data = agent.load_data(training_data_file)

    agent.train(
        training_data,
        augmentation_factor=50,
        epochs=500,
        batch_size=10,
        validation_split=0.2
    )

    agent.persist(model_path)
```

Ini adalah kode yang kami tulis di file `train_initialize.py`. Nah, sebelum kita lanjut ke file kode selanjutnya, mari kita coba pahami dulu poin-poin penting di dalamnya.

1. Pertama, kita mengimpor beberapa metode dari modul masa depan. Pernyataan masa depan dalam Python memiliki kegunaan khusus, pernyataan tersebut mengubah cara penguraian modul Python Anda, dan mengubah perilaku metode yang ada.

From_future_import braces

2. Impor metode `utils` dari modul `rasa_core` untuk mengonfigurasi logging.
3. Impor kelas `Agent` dari modul `agent` untuk membuat objek `Agent`.
4. `KerasPolicy`, `MemorizationPolicy` akan diteruskan sebagai parameter kebijakan ke kelas `Agent`.
5. `configure_color_logging`: Metode utilitas yang ditentukan di `utils.py` untuk pencatatan log berwarna menggunakan paket log warna Python.
6. `Agent`: Kelas yang ditentukan oleh Rasa yang menyediakan antarmuka untuk memanfaatkan fungsionalitas Inti Rasa yang paling penting, seperti pelatihan, menangani pesan, memuat model dialog, mengambil tindakan selanjutnya, dan menangani saluran.
7. `load_data`: memuat data pelatihan dari jalur yang diberikan.
8. `train`: melatih kebijakan/ansambel kebijakan tertentu menggunakan data dari file yang disediakan.
9. `training_data`: objek dikembalikan dengan metode `load_data`. Daftar `DialogueStateTracker`. Ini tidak lain adalah file data pelatihan kami.
10. `augmentation_factor`: memberi tahu Rasa berapa banyak cerita tiruan yang harus dibuat berdasarkan rangkaian cerita awal kita. Faktor 10x adalah heuristik untuk putaran augmentasi untuk generator data pelatihan.
11. `epoch`: 1 epoch adalah siklus pelatihan lengkap pada seluruh dataset pelatihan. Jumlah total data pelatihan maju dan mundur.
12. `batch_size`: memberi tahu Anda jumlah sampel pelatihan yang akan digunakan dalam setiap lintasan. 100 contoh dengan `batch_size` 10 akan membutuhkan 10 epoch untuk meninjau seluruh kumpulan data.
13. `validation_split` : persentase data untuk memvalidasi keakuratan model yang tidak bias.
14. `bertahan`: metode ini digunakan untuk mempertahankan objek `Agent` dalam direktori tertentu untuk digunakan kembali.

Pada titik ini Anda sudah cukup memahami apa yang dilakukan masing-masing metode dan apa yang terjadi di dalam kode. Sebelum kita melanjutkan menjalankan skrip, pastikan Anda telah menginstal pustaka `rasa_core` sebelum menjalankan skrip ini.

Anda dapat menginstal `rasa_core` menggunakan perintah berikut:

Pip install rasa_core == 0.11.1

Jika Anda mengikuti contoh chatbot dalam buku ini, pastikan Anda menginstal versi yang disebutkan di atas saja, karena Rasa mungkin tidak kompatibel dengan versi sebelumnya. Mereka dengan cepat menemukan metode yang lebih baru dan lebih optimal.

5.20 Rasa Core Terbaru

Anda juga dapat menginstal `rasa_core` versi terbaru dari repo github. Anda hanya perlu menjalankan serangkaian perintah berikut, yang akan mengambil kode terbaru dari github langsung sebelum menginstal.

```
git clone https://github.com/RasaHQ/rasa_core.git
cd rasa_core
pip install -r requirements.txt
pip install -e .
```

Mari kita coba jalankan file kode ini untuk melatih model kita sesuai parameter yang diberikan.

```
$python train_initialize.py
```

Anda juga dapat menjalankan skrip ini dari Jupyter Notebook itu sendiri menggunakan perintah ajaib Jupyter, seperti yang ditunjukkan di sini:

```
!python train_initialize.py #Gunakan python3 jika Anda telah menginstal rasa untuk python3
```

Diperlukan waktu sekitar 25 hingga 30 detik agar model dilatih pada kumpulan data sekecil milik kita. Saya menambahkan `SklearnPolicy` dalam daftar kebijakan bersama dengan `MemorizationPolicy` dan `KerasPolicy` untuk melatih model saya. Kebijakan yang berbeda mempunyai manfaatnya masing-masing. Baca selengkapnya tentang mereka untuk mengetahui mana yang mungkin lebih cocok untuk kasus penggunaan Anda; untuk kumpulan data saya, `SklearnPolicy` tampaknya berkinerja lebih baik daripada `KerasPolicy`.

Setelah skrip selesai dijalankan, Anda akan melihat beberapa pesan sukses seperti ini:

```
2018-08-30 04:24:31 INFO rasa_core.policies.keras_policy - Done fitting
keras policy model
2018-08-30 04:24:31 INFO rasa_core.featurizers - Creating states and
action examples from collected trackers (by MaxHistoryTrackerFeaturizer)...
Processed trackers: 100%|████████████████████| 96/96 [00:00<00:00,
898.31it/s, # actions=75]
2018-08-30 04:24:31 INFO rasa_core.featurizers - Created 75 action
examples.
2018-08-30 04:24:31 INFO rasa_core.policies.sklearn_policy - Done
fitting sklearn policy model
2018-08-30 04:24:31 INFO rasa_core.agent - Model directory models/nlu
exists and contains old model files. All files will be overwritten.
2018-08-30 04:24:31 INFO rasa_core.agent - Persisted model to
'/Users/sumit/apress_all/Chapter IV/horoscope_bot/models/nlu'
2018-08-30 04:24:31 INFO rasa_core.agent - Model directory models/nlu
exists and contains old model files. All files will be overwritten.
2018-08-30 04:24:31 INFO rasa_core.agent - Persisted model to
'/Users/sumit/apress_all/Chapter IV/horoscope_bot/models/nlu'
```

Anda juga akan menemukan beberapa folder yang dibuat sesuai nama modelnya. Pastikan Anda memilikinya di `model_path` yang Anda berikan di skrip. Di bawah ini adalah folder/file yang saya lihat di dalam folder `model_path` saya.

```
policy_0_MemoizationPolicy
policy_1_KerasPolicy
policy_2_SklearnPolicy
domain.json
domain.yml
Policy_metadata.json
```

Jika Anda telah memvalidasi bahwa model Anda telah berhasil menyelesaikan eksekusi dan telah mempertahankan model tersebut di sistem lokal Anda, maka kita dapat melanjutkan ke langkah pelatihan interaktif berikutnya.

Pelatihan Real-Time dengan Penguatan

Di bagian ini kita akan menulis beberapa kode lagi untuk melatih model dialog kita dan melatihnya kembali ketika model tersebut memberikan keluaran yang salah. Jadi, saat bot kita melakukan kesalahan, kita langsung terjun dan memberi tahu model bahwa prediksinya salah dengan memberi tahu model tersebut apa yang benar. Tanpa harus berhenti, model akan melatih kembali dirinya sendiri, dan setelah selesai, interaksi antara pengguna dan bot akan ditangkap ke dalam file dan ditambahkan ke data pelatihan yang ada. Ini bekerja lebih seperti sistem umpan balik di setiap langkah, daripada menunggu satu imbalan di akhir.

Langkah selanjutnya adalah membuat file baru bernama `endpoints.yml` dengan konten di bawah ini. Kami akan menggunakan file ini dalam file kode Python kami `train_online.py`. Dengan konfigurasi ini kita dapat mengekspos metode Rasa sebagai HTTP API.

```
action_endpoint:
  url: http://localhost:5055/webhook

#nlg:
# url: http://localhost:5056/nlg

core_endpoint:
  url: http://localhost:5005
```

Sekarang, mari buat `train_online.py` untuk tujuan pelatihan online/interaktif.

```

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals

import logging

from rasa_core import utils, train
from rasa_core.training import online
from rasa_core.interpreter import NaturalLanguageInterpreter

logger = logging.getLogger(__name__)

def train_agent(interpreter):
    return train.train_dialog_model(domain_file="horoscope_domain.yml",
                                    stories_file="data/stories.md",
                                    output_path="models/dialog",
                                    nlu_model_path=interpreter,
                                    endpoints="endpoints.yml",
                                    max_history=2,
                                    kwargs={"batch_size": 50,
                                             "epochs": 200,
                                             "max_training_samples": 300
                                             })

if __name__ == '__main__':
    utils.configure_colored_logging(loglevel="DEBUG")
    nlu_model_path = "./models/nlu/default/horoscopebot"
    interpreter = NaturalLanguageInterpreter.create(nlu_model_path)
    agent = train_agent(interpreter)
    online.serve_agent(agent)

```

`max_history` adalah jumlah negara bagian yang harus dilacak oleh model.

Sebelum kita menjalankan skrip terakhir kita, `train_online.py`, kita harus mengetahui dan mempersiapkan diri untuk sesuatu yang disebut `rasa-nlu-sdk`.

rasa-nlu-sdk

Tumpukan Rasa NLU hadir dengan `rasa-nlu-sdk`, yang merupakan SDK Python untuk pengembangan tindakan khusus untuk Rasa Core. Adapun contoh chatbot kita, kita perlu menentukan beberapa tindakan khusus, seperti menekan API untuk mendapatkan horoskop hari ini atau mungkin operasi penulisan database untuk menambahkan pengguna ke daftar berlangganan. Kabar baiknya adalah mereka memiliki perpustakaan terpisah untuk ini, dan kita dapat menginstalnya menggunakan `pip`. Mari kita instal menggunakan perintah berikut:

`pip install rasa-core-sdk ==0.11.0`

Sekarang, kita perlu pergi ke tab lain di terminal kita atau baris perintah baru dan jalankan perintah berikut di direktori proyek kita (tempat di mana file `action.py` kita berada):

```
python -m rasa_core_sdk.endpoint --actions actions

INFO:__main__:Starting action endpoint server...
INFO:rasa_core_sdk.executor:Registered function for 'get_todays_horoscope'.
INFO:rasa_core_sdk.executor:Registered function for 'subscribe_user'.
INFO:__main__:Action endpoint is up and running. on ('0.0.0.0', 5055)
```

Perintah ini akan memulai server tindakan yang akan mendengarkan tindakan kustom apa pun yang diprediksi oleh model. Segera setelah tindakan apa pun dipicu, ia akan menjalankannya dan memberikan respons sesuai metodenya.

Port default untuk titik akhir server tindakan adalah 5055 di localhost. Jika Anda ingin mengubahnya, Anda dapat menambahkan argumen `pass --port` di baris perintah. Sebuah pertanyaan singkat muncul di benak: mengapa? Mengapa saya memerlukan server terpisah untuk itu?

Mengapa kita tidak bisa menggunakan Python biasa? Ya, kita bisa menggunakan Python biasa, tapi misalkan Anda mengembangkan tindakan yang diperlukan dalam bahasa lain atau Anda memiliki beberapa tindakan yang sudah diekspos sebagai API. Sekarang, Anda hanya perlu membuka `endpoints.yml`, yang telah kita buat, dan menggunakannya untuk menyebutkan di mana server tindakan Anda harus digunakan dan di mana server `core_endpoint` Anda seharusnya berada. Dalam sistem produksi, keduanya dapat berupa server berbeda yang memiliki url berbeda.

Sekarang, ketika kita menjalankan skrip berikutnya di mana kita menyebutkan file `endpoints.yml`, Rasa akan membaca file tersebut dan mendapatkan konfigurasi untuk `action_server` kita, yang sudah aktif dan berjalan sesuai konfigurasi yang kita lakukan sebelumnya.

```
action_endpoint:
  url: http://localhost:5055/webhook
```

Mari jalankan `train_online.py` di terminal baris perintah baru.

```
$python3 train_online.py
```

Setelah pelatihan model dialog berhasil, kita akan menerima pesan seperti ini:

```
2018-08-30 07:09:37 INFO    rasa_core.policies.keras_policy - Done
fitting keras policy model
2018-08-30 07:09:37 INFO    rasa_core.agent - Model directory models/nlu
exists and contains old model files. All files will be overwritten.
2018-08-30 07:09:37 INFO    rasa_core.agent - Persisted model to '/Users/
sumit/apress_all/Chapter IV/horoscope_bot/models/nlu'
2018-08-30 07:09:37 INFO    rasa_core.training.online - Rasa Core server
is up and running on http://localhost:5005
Bot loaded. Type a message and press enter (use '/stop' to exit).
127.0.0.1 - - [2018-08-30 07:09:37] "GET /domain HTTP/1.1" 200 996 0.001847
```

Sekarang, disini Anda bisa mulai berbicara dengan bot Anda yang baru saja lahir. Dari sini, sepenuhnya terserah Anda bagaimana Anda melatihnya. Anda dapat memperbaikinya setiap kali ia merespons sesuatu yang buruk atau tidak terduga. Mari kita lakukan itu dan lihat apakah hasilnya membaik.

Saya memasukkan "Hai" sebagai pesan pertama dan bot mengembalikan pesan berikut:

```
-----
Chat history:

  bot did: action_listen

  user said: hi

  whose intent is: {'confidence': 0.8472929307505297, 'name': 'greeting'}

we currently have slots: DD: None, MM: None, horoscope_sign: None,
subscribe: None

-----
The bot wants to [utter_greet] due to the intent. Is this correct?

1.    Yes
2.    No, intent is right but the action is wrong
3.    The intent is wrong
0.    Export current conversations as stories and quit

-----
```

Sekarang, bot tersebut memberi tahu saya apa yang dilakukannya berdasarkan apa yang dikatakan pengguna dan bagaimana dia memahaminya. Sekarang, ini juga memberi kita empat pilihan. Kami akan membicarakannya sebentar lagi. Yang dilakukan bot di sini adalah arti `action_listen`; itu hanya menunggu pengguna memasukkan sesuatu, itu tidak masalah.

Bot mengatakan maksudnya `{'kepercayaan': 0.8472929307505297, 'nama': 'salam'}`, yang benar, jadi kita tekan 1, artinya bot memahami dan mencoba melakukannya dengan benar.

1

Chat history:

bot did: action_listen

user said: hi

whose intent is: {'confidence': 0.8472929307505297, 'name': 'greeting'}

bot did: utter_greet

we currently have slots: DD: None, MM: None, horoscope_sign: None,
subscribe: None-----
The bot wants to [action_listen]. Is this correct?

1. Yes.
2. No, the action is wrong.
0. Export current conversations as stories and quit

Dan ini dia, bot kami setidaknya menyapa pengguna dengan baik. Jika Anda melihat respons yang disebutkan di atas, Anda sebenarnya dapat memeriksa apa yang dilakukan bot kami; ia melacak semua ini saat kami melatihnya.

Bot did : utter_greet

Sekarang, bot mengatakan bahwa hal berikutnya adalah menunggu dan mendengarkan apa yang akan dikatakan pengguna (yaitu menunggu masukan pengguna). Apakah ini benar? Ya benar, jadi kita tekan 1 lagi dan bot memberikan prompt kepada pengguna.

Saya bertanya, "Apa ramalan bintang saya hari ini?"

Chat history:

bot did: action_listen

user said: hi

whose intent is: {'confidence': 0.8472929307505297, 'name':
'greeting'}

bot did: utter_greet

bot did: action_listen

user said: what's my horoscope for today?

whose intent is: {'confidence': 0.8902154738608781, 'name':
'get_horoscope'}we currently have slots: DD: None, MM: None, horoscope_sign: None,
subscribe: None-----
The bot wants to [utter_ask_horoscope_sign] due to the intent. Is this correct?

1. Yes
2. No, intent is right but the action is wrong
3. The intent is wrong
0. Export current conversations as stories and quit

Sekarang, bot dengan sangat tepat mengidentifikasi maksud tersebut menjadi maksud “get_horoscope” dengan akurasi 89 persen, dan ini cukup bagus. Bot juga mengatakan ingin mengucapkan_ask_horoscope_sign, yang lagi-lagi benar karena pengguna belum menyebutkan tanda horoskop apa pun dan slotnya masih Tidak Ada, seperti yang ditunjukkan.

Kami menekan 1 lagi.

```
-----
Chat history:
  bot did: action_listen
  user said: hi
  whose intent is: {'confidence': 0.8472929307505297, 'name': 'greeting'}
  bot did: utter_greet
  bot did: action_listen
  user said: what's my horoscope for today?
  whose intent is: {'confidence': 0.8902154738608781, 'name':
'get_horoscope'}
  bot did: utter_ask_horoscope_sign
we currently have slots: DD: None, MM: None, horoscope_sign: None,
subscribe: None
-----
The bot wants to [action_listen]. Is this correct?
  1.    Yes.
  2.    No, the action is wrong.
  0.    Export current conversations as stories and quit
-----
```

Sekarang, bot menanyakan apakah sudah waktunya menunggu masukan pengguna. Ya, pengguna perlu memberikan tanda horoskop. Kami menekan 1 lagi, dan ketika diminta, kami memasukkan “Capricorn” sebagai input.

Chat history:

bot did: action_listen

user said: hi

whose intent is: {'confidence': 0.8472929307505297, 'name': 'greeting'}

bot did: utter_greet

bot did: action_listen

user said: what's my horoscope for today?

whose intent is: {'confidence': 0.8902154738608781, 'name': 'get_horoscope'}

bot did: utter_ask_horoscope_sign

bot did: action_listen

user said: Capricorn

whose intent is: {'confidence': 0.8456705103362621, 'name': 'get_horoscope'}

with horoscope_sign: capricorn

we currently have slots: DD: None, MM: None, horoscope_sign: capricorn, subscribe: None

The bot wants to [get_todays_horoscope] due to the intent. Is this correct?

1. Yes
 2. No, intent is right but the action is wrong
 3. The intent is wrong
 0. Export current conversations as stories and quit
-

Sekarang, seperti yang bisa kita lihat, slot horoscope_sign secara otomatis disetel di atas ke "Capricorn."

Langkah selanjutnya adalah mengajari bot untuk memanggil get_todays_horoscope dan memanggil tindakan get_todays_horoscope. Saat ini slot kita sudah disetel untuk variabel horoscope_sign yang kita tentukan di horoscope_domain.yml, jadi jika tindakan ini dipanggil lagi, chatbot akan mengembalikan horoskop sebenarnya untuk hari itu. Mari mencoba.

Saya memasukkan "Capricorn".

Bot mencapai titik akhir server tindakan dan mengembalikan hasilnya ke pengguna seperti yang kami tentukan dalam tindakan khusus kami.

Your today's horoscope :

Perjalanan dari kekalahan menuju kesuksesan didorong oleh optimisme. Anda akan sangat optimis hari ini tentang pekerjaan Anda, masa depan, dan segala sesuatu yang dapat membawa Anda menuju kesuksesan, kata Ganesha. Anda juga akan sangat berhati-hati saat

menangani pekerjaan sehari-hari, dan akan memanfaatkan setiap peluang yang dapat membawa Anda selangkah lebih dekat ke tujuan Anda. Tekad dan keberuntungan adalah semua yang Anda butuhkan untuk mewujudkan impian Anda; kamu akan mendapatkan keduanya hari ini.

Ini tampaknya adil, karena saya melewatkan tenggat waktu untuk menyelesaikan bab ini, dan saat ini sudah jam 6 pagi dan saya belum tidur. Lakukan semaksimal mungkin agar uang Anda layak dibelanjakan untuk buku ini. Oke, mari kita kembali melihat apa yang bot kita coba lakukan selanjutnya.

The bot wants to [action_listen]. Is this correct?

1. Yes.
2. No, the action is wrong.
0. Export current conversations as stories and quit

Mungkin kita tidak ingin bot kita mendengarkan sekarang setelah ia mengucapkan horoskop hari ini. Kami ingin bot kami meminta pengguna untuk berlangganan pembaruan sesuai skrip percakapan kami dan juga sesuai cerita yang kami buat.

Jadi, saya akan menekan 2, dan muncul tulisan

what is the next action for the bot?

0	action_listen	1.00
1	action_restart	0.00
2	action_default_fallback	0.00
3	utter_greet	0.00
4	utter_ask_horoscope_sign	0.00
5	utter_ask_dob	0.00
6	utter_subscribe	0.00
7	get_todays_horoscope	0.00
8	subscribe_user	0.00

Karena kita ingin pengguna berlangganan, kita harus menekan 6 sehingga bot menanyakan pengguna apakah dia ingin berlangganan pembaruan harian.

Bot menjawab, "Apakah Anda ingin berlangganan pembaruan harian?" Sekarang bot berkata:

The bot wants to [get_todays_horoscope]. Is this correct?

1. Yes.
2. No, the action is wrong.
0. Export current conversations as stories and quit

Tindakan di sini tidak benar, jadi kita tekan 2 dan latih bot untuk mendengarkan tindakan pengguna di sini.

```
-----
what is the next action for the bot?
```

0	action_listen	0.21
1	action_restart	0.01
2	action_default_fallback	0.01
3	utter_greet	0.21
4	utter_ask_horoscope_sign	0.51
5	utter_ask_dob	0.01
6	utter_subscribe	0.02
7	get_todays_horoscope	0.78
8	subscribe_user	0.22

```
-----
```

Kami menekan 0 ketika bot sedang menunggu untuk mengetahui langkah selanjutnya yang benar. Bot sekarang akan menunggu masukan pengguna dalam proses pembelajaran interaktif kami.

Pengguna mengatakan di sini, "Tolong berlangganan saya," dan bot kami berhasil dengan keyakinan 80 persen mengenali ini sebagai maksud berlangganan dan juga, berdasarkan data pelatihan kami di data.json, memahami dengan mudah bahwa pengguna meminta untuk berlangganan dan tidak berhenti berlangganan. Dengan kata lain, ini menetapkan slot berlangganan ke True, seperti yang Anda lihat di sini:

```
-----
user said: Please subscribe me

whose intent is: {'confidence': 0.795172441763619, 'name':
'subscription'}

with subscribe: True

we currently have slots: DD: None, MM: None, horoscope_sign: capricorn,
subscribe: True
-----
```

Sekarang, bot kita telah memahami maksudnya dan juga menguraikan entitas di dalamnya, saatnya membalas kembali ke pengguna dengan mengatakan, "Dia berhasil berlangganan," dan bingo—kita mendapat pesan dari bot:

You're successfully subscribed

Ini dia! Anda berhasil membuat chatbot internal Anda sendiri. Sebagai bagian dari latihan, saya sarankan Anda menambahkan semua kasus penggunaan yang telah kita diskusikan untuk HoroscopeBot dan melihat apakah itu berhasil. Cobalah untuk menambahkan lebih banyak kasus penggunaan dan lebih banyak fungsi ketika semua yang telah kita lakukan sejauh ini berfungsi dengan baik.

Anda dapat menambahkan lebih banyak pesan ucapan di sini agar bot dapat mengucapkan, "Selamat tinggal, semoga harimu menyenangkan," setelah percakapan selesai. Saya sangat yakin Anda harus berada dalam posisi untuk melakukan itu. Pastikan Anda melakukannya dan beri tahu kami bagaimana hasilnya bagi Anda.

Sekarang, bagian terakhirnya adalah mengembalikan semua pembelajaran ini kepada pengguna sehingga kita tidak perlu mengajarkan cerita yang sama lagi dan bot mencoba belajar dari cerita yang sudah ada dan lebih banyak lagi yang akan datang.

5.21 Mengekspor Percakapan Sebagai Cerita

Sekarang langkah selanjutnya dalam bagian pembelajaran interaktif adalah menekan 0 dan mengeksport percakapan kita saat ini sebagai cerita dan keluar.

Kami menekan 0 untuk itu dan bot menanyakan sesuatu seperti ini:

File to export to (if file exists, this will append the stories) [stories.md]:

Cukup masukkan nama file Stories.md kami, jika Anda menyimpan nama lain, pastikan Anda memberikan nama file yang benar.

Dan Rasa melakukan kerja kerasnya untuk kami dan membuat file Stories.md untuk kami secara otomatis lagi seperti ini:

```
## Generated Story 3797421409943253925
* greeting
  - utter_greet
* I want to know my horoscope
  - utter_ask_horoscope_sign
* get_horoscope

## Generated Story 7304374603374920907
* greeting
  - utter_greet
* what is my todays horoscope?
  - utter_ask_horoscope_sign
* Cancer

## Generated Story -6877464862083507836
* greeting
  - utter_greet
* get_horoscope
  - utter_ask_horoscope_sign
* get_horoscope{"horoscope_sign": "capricorn"}
  - slot{"horoscope_sign": "capricorn"}
  - get_todays_horoscope
  - slot{"horoscope_sign": "capricorn"}
  - utter_subscribe
* subscription{"subscribe": "True"}
  - slot{"subscribe": "True"}
  - subscribe_user
  - slot{"subscribe": true}
```

Anda dapat mengubah sumber file cerita saat melatih model dengan file yang disebutkan di atas atau Anda dapat menyalin dan menempelkan cerita yang baru dibuat ke dalam file Stories.md asli kami. Untuk saat ini, saya akan menyalin dan menempelkan cerita -

cerita ini ke file Stories.md asli kami. Pastikan untuk menjalankan kembali model Anda sebelum menguji atau melakukan pembelajaran interaktif lagi.

Menguji Bot

Menguji bot adalah bagian dari proses pembelajaran interaktif kami. Kami terus memeriksa semua kemungkinan skenario yang awalnya dipikirkan untuk bot dan melatihnya sesuai dengan itu. Saya menguji Bot Horoskop untuk beberapa skenario lain dan melatihnya sesuai dengan itu. Berikut ini adalah salah satunya yang sedikit berbeda dari apa yang kita lakukan sebelumnya di bab ini.

Uji Kasus 1

Mari kita jalankan tes pertama:

Pengguna : Halo!

HoroscopeBot: Halo! Apa yang bisa saya lakukan untuk Anda?

Pengguna : Apa ramalan bintang Capricorn hari ini?

HoroscopeBot: Horoskop Anda hari ini:

Perjalanan dari kekalahan menuju kesuksesan didorong oleh optimisme. Anda akan sangat optimis hari ini tentang pekerjaan Anda, masa depan, dan segala sesuatu yang dapat membawa Anda menuju kesuksesan, kata Ganesha. Anda juga akan sangat berhati-hati saat menangani pekerjaan sehari-hari, dan akan memanfaatkan setiap peluang yang dapat membawa Anda selangkah lebih dekat ke tujuan Anda. Tekad dan keberuntungan adalah semua yang Anda butuhkan untuk mewujudkan impian Anda; kamu akan mendapatkan keduanya hari ini.

HoroscopeBot: Apakah Anda ingin berlangganan pembaruan harian?

Pengguna : Silakan berlangganan saya.

HoroscopeBot: Anda berhasil berlangganan.

Seperti yang Anda lihat dalam percakapan antara pengguna dan chatbot ini, kami menguji apakah bot menangani skenario di mana pengguna sendiri yang memberikan horoscope_sign sambil menanyakan horoskop untuk hari itu. Bot bekerja dengan baik dalam kasus ini dan akhirnya meminta berlangganan dan, berdasarkan respons pengguna, mengambil tindakan yang sesuai. Anda harus melakukan pengujian kewarasan terhadap percakapan bot untuk menentukan apakah aliran normal berfungsi atau tidak.

Saya juga menguji cerita yang kami sebutkan di file Stories.md kami ketika pengguna langsung meminta berlangganan dan itu berhasil juga.

Uji Kasus 2

Sekarang tes kedua:

Pengguna : Hai

HoroscopeBot: Halo! Apa yang bisa saya lakukan untuk Anda?

Pengguna : Tolong berlangganan saya untuk pembaruan harian.

HoroscopeBot: Anda berhasil berlangganan.

Saya juga akan menambahkan beberapa cerita baru agar bot bekerja dengan sempurna.

Namun saat ini kami memiliki bot yang berfungsi penuh. Dalam versi kedua kode di github Anda juga akan menemukan bot untuk memperbaiki kesalahan ejaan untuk tanda

horoskop, mengambil tanda horoskop berdasarkan DOB pengguna, mengucapkan pesan selamat tinggal, dll. Saya sangat menyarankan Anda memeriksa kode bagaimana caranya selesai dan berikan tanggapan Anda. Tetapi sebelum itu, pikirkan bagaimana hal itu akan dilakukan dan ke mana arah perubahannya. Kami sengaja tidak menyebutkan kode untuk kasus penggunaan lainnya dalam bab ini, sehingga tidak menjadi rumit saat Anda berkonsentrasi mempelajari trik-trik perdagangan.

Anda dapat mengunduh kode Python dan Notebook Jupyter terbaru dari repo github kami dan mencoba menginstal paket yang benar untuk menjalankannya. Anda akan menemukan lebih banyak kasus penggunaan yang ditangani dalam kode seperti yang dibahas dalam bab ini.

Ringkasan

Dalam bab ini, kita mempelajari tentang Rasa-NLU dan mengapa Rasa-NLU lebih baik daripada alat sumber terbuka lainnya yang tersedia di pasar. Kami mempelajari cara mengonfigurasi pipeline dengan menggunakan tensorflow, sklearn, dan keras dalam konfigurasi pipeline kami.

Kami belajar membuat semuanya dari awal di sistem lokal kami tanpa bergantung pada layanan apa pun yang mengharuskan Anda menggunakan API mereka, seperti Dialogflow, wit.ai, dll.

Kami juga mempelajari cara membuat cerita dan cara melatih model NLU dan model dialog serta menggunakan keduanya menggunakan Rasa Core untuk membuat bot dengan melatih menggunakan fitur paling keren yaitu pembelajaran interaktif. Kami juga mendapat ide bagus tentang cara membuat data pelatihan dengan mudah dan membuat anotasi dengan mudah menggunakan alat sumber terbuka seperti rasa-nlu-trainer. Saya harap bab ini lebih interaktif bagi Anda dibandingkan bab lainnya. Jika Anda belum merasakan pencapaian, bersiaplah untuk bab berikutnya, di mana kami akan menyajikannya secara langsung kepada audiens dan menunjukkan kepada dunia kemampuan bot. Kita akan belajar mengintegrasikan chatbot dari bab ini ke berbagai platform seperti Facebook dan Slack menggunakan server web kita sendiri. Terus latih bot Anda hingga di bab berikutnya kami menayangkannya secara langsung. Sampai jumpa di bab berikutnya.

BAB 6

MERANCANG DAN MEMBANGUN CHATBOT

Sejauh ini, kami telah membuat chatbot dan menerapkannya pada platform perpesanan seperti Facebook Messenger. Facebook Messenger saat ini merupakan layanan perpesanan paling populer. Namun ada layanan perpesanan yang sudah ada cukup lama dan masih populer digunakan oleh para pebisnis untuk terhubung dengan pelanggan. Layanan Pesan Singkat, yang populer disingkat SMS, adalah platform hebat untuk chatbots. Bisnis di seluruh dunia menggunakan layanan ini untuk mengirimkan ribuan pesan notifikasi kepada pelanggan di berbagai fase perjalanan mereka: pemasaran, penjualan, transaksi, pengiriman, dan sebagainya. Dan bagian terbaiknya adalah tidak memerlukan data internet untuk berkomunikasi.

Dalam bab ini, kami akan merancang dan membangun chatbot untuk membantu pengguna merencanakan perjalanan kereta mereka. Kami akan menggunakan API data Transportasi untuk memperoleh informasi tentang kereta api, waktu kedatangan dan keberangkatannya di stasiun, dan sebagainya, dan untuk menyajikan informasi tersebut, membangun chatbot untuk berinteraksi dengan pengguna dalam bahasa alami. Kami kemudian akan mengeksplorasi bagaimana chatbot dapat diekspos ke platform SMS. Kita akan belajar menggunakan layanan penyedia API komunikasi bernama Twilio. Kita juga akan mempelajari cara memasang perangkat seperti API.AI untuk memahami ucapan pengguna dan mengelola percakapan di chatbot SMS.

Pertama, mari kita lihat Transport API dan data yang ditawarkannya. Berdasarkan data yang tersedia, kami kemudian akan melakukan brainstorming dan merancang beberapa contoh tugas percakapan. Kami kemudian akan membangun bot SMS satu arah sederhana yang dapat mengirimkan pemberitahuan tepat waktu kepada pengguna. Dan kembangkan hal tersebut untuk mengembangkan chatbot dua arah yang mengirimkan informasi kereta api kepada pengguna berdasarkan permintaan mereka dalam bahasa alami. Kami akan mempelajari cara membangun dan mengintegrasikan agen API.AI ke dalam chatbot kami untuk memahami bahasa dan mendorong percakapan.

Pada akhir bab ini, Anda akan dapat:

- Rancang tugas percakapan berdasarkan data
- Buat modul tugas backend menggunakan Transport API
- Bangun bot SMS menggunakan Twilio
- Integrasikan agen Dialogflow untuk memahami ucapan pengguna

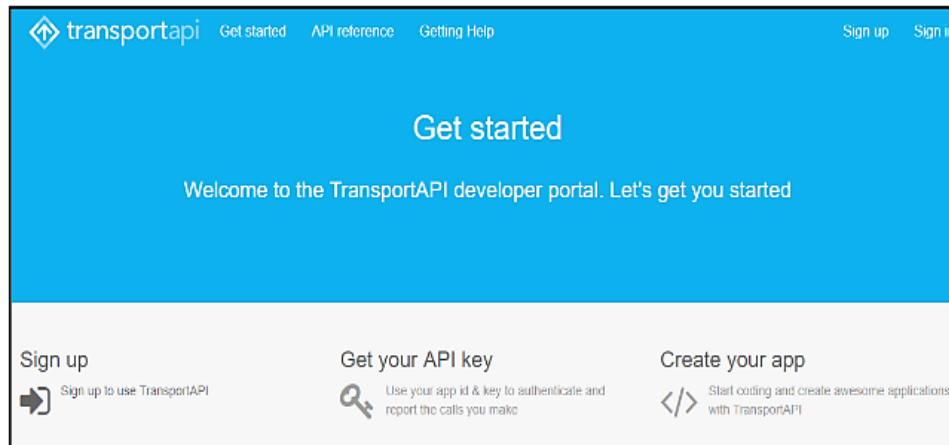
6.1 Menjelajahi API Transportasi

Untuk memulai, mari kita lihat sumber data yang akan kita gunakan dalam bab ini. Transport API adalah layanan data untuk semua layanan transportasi umum di Inggris. Jika Anda berada di negara lain, Anda akan dapat menemukan sesuatu yang mirip dengan negara Anda. Namun Anda masih bisa bermain-main dengan layanan ini. Transport API adalah

platform data untuk data transportasi yang memberikan informasi tentang kedatangan dan keberangkatan langsung, jadwal, perencanaan perjalanan, tarif, indikator kinerja, dan pemetaan tweet komuter. Data disajikan menggunakan layanan web RESTful.

6.2 Membuat Akun Pengembang

1. Untuk memulai, buat akun pengembang di <https://developer.transportapi.com> dan dapatkan kunci aplikasi dan ID aplikasi:



Gambar 6.1 Tampilan transportapi.com

2. Mari kita coba contoh permintaan berikut. Ganti YOUR_APP_ID dan YOUR_APP_KEY dengan ID dan kunci aplikasi Anda. Jalankan permintaan GET berikut dari browser web. Atas permintaan ini, kami mencoba mengambil informasi mengenai stasiun kereta Euston, di London:

http://transportapi.com/v3/uk/places.json?query=euston&type=train_station&app_id=YOUR_APP_ID&app_key=YOUR_APP_KEY

Permintaan sebelumnya akan mengembalikan respons JSON dengan informasi seperti nama lengkap stasiun, koordinat lintang dan bujur, serta kode stasiun:



Menjelajahi kumpulan data

Transport API menyediakan data mengenai kereta api, bus, tube (kereta bawah tanah), dan banyak bentuk transportasi lainnya. Untuk membangun chatbot, mari kita mulai dengan data kereta saja. Di bagian ini, mari kita lihat berbagai jenis data yang tersedia tentang kereta api. Untuk semua permintaan, URL dasarnya adalah <http://transportapi.com/v3/uk/>.

Stasiun kereta di dekat Anda

Titik akhir ini menyediakan daftar stasiun kereta api di dekat lokasi tertentu. Lokasi pencarian harus disediakan dalam koordinat latlon, sebagai berikut: http://transportapi.com/v3/uk/train/stations/near.json?lat=55.9485&lon=-3.2021&app_id=YOUR_APP_ID&app_key=YOUR_APP_KEY

Tanggapan

Respons yang kami dapatkan adalah daftar stasiun di dekat koordinat garis lintang yang diberikan. Untuk setiap stasiun, kami mendapatkan nama, lokasi, dan kode stasiunnya:

```

{
  "minlon": -3.4521,
  "minlat": 55.6985,
  "maxlon": -2.9521,
  "maxlat": 56.1985,
  "searchlon": -3.2021,
  "searchlat": 55.9485,
  "page": 1,
  "rpp": 25,
  "total": 54,
  "request_time": "2017-08-05T06:21:24+01:00",
  "stations": [ ... ] // 25 items
}

"stations": [
  {
    "station_code": "EDB",
    "atcocode": null,
    "tiploc_code": "EDINBUR",
    "name": "Edinburgh Waverley",
    "mode": "train",
    "longitude": -3.188228,
    "latitude": 55.952391,
    "distance": 966
  },
  { - }, // 8 items
  { - }, // 8 items
  { - }, // 8 items

```

Kereta api di daerah tersebut

Stasiun kereta juga dapat dicari dengan menyediakan kotak pembatas. Koordinat kiri atas dan kanan bawah kotak perlu disediakan. Ini sangat berguna jika Anda memerlukan daftar semua stasiun dengan batas kota atau yang serupa. Mari kita coba menggunakan koordinat kotak pembatas untuk Edinburgh:

http://transportapi.com/v3/uk/train/stations/bbox.json?minlon=-3.4521&minlat=55.6985&maxlon=-2.9521&maxlat=56.1985&app_id=YOUR_APP_ID&app_key=YOUR_APP_KEY

Tanggapan

Respon yang kami dapatkan adalah sebagai berikut:

```

{
  "minlon": -3.4521,
  "minlat": 55.6985,
  "maxlon": -2.9521,
  "maxlat": 56.1985,
  "searchlon": -3.2021,
  "searchlat": 55.9485,
  "page": 1,
  "rpp": 25,
  "total": 54,
  "request_time": "2017-08-05T06:58:20+01:00",
  "stations": [
    {
      "station_code": "EDB",
      "atccode": null,
      "tiploc_code": "EDINBUR",
      "name": "Edinburgh Waverley",
      "mode": "train",
      "longitude": -3.188228,
      "latitude": 55.952391,
      "distance": 966
    },
    {
      "station_code": "HYM",
      "atccode": null,
      "tiploc_code": "HAYMRKT",
      "name": "Haymarket",
      "mode": "train",
      "longitude": -3.218449,
      "latitude": 55.945806,
      "distance": 1061
    }
  ]
}

```

Keberangkatan langsung

Status langsung kereta yang tiba dan berangkat di stasiun tertentu dapat diperoleh dengan menggunakan titik akhir berikut. Mari kita coba di stasiun Edinburgh Waverley yang kode stasiunnya EDB:

http://transportapi.com/v3/uk/train/station/EDB/live.json?&app_id=YOUR_APP_ID&app_key=YOUR_APP_KEY

<http://transportapi.com>

[/v3/uk/train/station/EDB/live.json?&app_id=YOUR_APP_ID&app_key=YOUR_APP_ID&app_key=YOUR_APP_KEY](http://transportapi.com/v3/uk/train/station/EDB/live.json?&app_id=YOUR_APP_ID&app_key=YOUR_APP_ID&app_key=YOUR_APP_KEY)

Tanggapan

Inilah tanggapannya:

```

{
  "date": "2017-08-05",
  "time_of_day": "06:00",
  "request_time": "2017-08-05T06:49:44+01:00",
  "station_name": "Edinburgh Waverley",
  "station_code": "EDB",
  "departures": {
    "all": [
      {
        "mode": "train",
        "service": "13560015",
        "train_uid": "G83644",
        "platform": "10",
        "operator": "SR",
        "operator_name": "Scotrail",
        "aimed_departure_time": "06:07",
        "aimed_arrival_time": null,
        "aimed_pass_time": null,
        "origin_name": "Edinburgh Waverley",
        "source": "ATOC",
        "destination_name": "Milngavie",
        "category": "OO",
        "service_timetable": {
          "id": "http://transportapi.com/v3/uk/train/service/train_uid:G83644/2017-08-05/timetable.json?app_id=60ce46ea&app_key=251dc61414961e8ebfe110329ffa367d"
        }
      },
      {
        "mode": "train",
        "service": "22180008",
        "train_uid": "P31962",
        "platform": "7",
        "operator": "XC",
        "operator_name": "CrossCountry",

```

Jadwal stasiun

Jadwal kereta api tiba dan berangkat dari stasiun tertentu pada tanggal dan waktu tertentu dapat diperoleh dengan menggunakan titik akhir berikut. Ayo dapatkan semua kereta yang berangkat dari stasiun Edinburgh Waverley (EDB) pada 05-08-2017 pukul 06:00: http://transportapi.com/v3/uk/train/station/EDB/2017-08-05/06.00/timetable.json?app_id=YOUR_APP_ID&app_key=YOUR_APP_KEY

Tanggapan

Dan inilah jawabannya:

```

{
  "date": "2017-08-05",
  "time_of_day": "06:00",
  "request_time": "2017-08-05T06:49:44+01:00",
  "station_name": "Edinburgh Waverley",
  "station_code": "EDB",
  "departures": {
    "all": [
      {
        "mode": "train",
        "service": "13560015",
        "train_uid": "G83644",
        "platform": "10",
        "operator": "SR",
        "operator_name": "Scotrail",
        "aimed_departure_time": "06:07",
        "aimed_arrival_time": null,
        "aimed_pass_time": null,
        "origin_name": "Edinburgh Waverley",
        "source": "ATOC",
        "destination_name": "Milngavie",
        "category": "OO",
        "service_timetable": {
          "id": "http://transportapi.com/v3/uk/train/service/train_uid:G83644/2017-08-05/timetable.json?app_id=60ce46ea&app_key=251dc61414961e8ebfe110329ffa367d"
        }
      },
      {
        "mode": "train",
        "service": "22180008",
        "train_uid": "P31962",
        "platform": "7",
        "operator": "XC",
        "operator_name": "CrossCountry",

```

Jadwal layanan

Jadwal setiap layanan kereta juga dapat diperoleh. Data ini memungkinkan Anda melihat waktu kedatangan dan keberangkatan layanan kereta api tertentu di berbagai stasiun yang dihubungi pada tanggal dan waktu tertentu. Mari kita coba contoh dengan nomor layanan, 23587103, pada 05-08-2017 pukul 06:00:

http://transportapi.com/v3/uk/train/service/23587103/2017-08-05/06.00/timetable.json?app_id=YOUR_APP_ID&app_key=YOUR_APP_KEY

Tanggapan

Respon yang kami dapatkan terhadap jadwal pelayanan adalah sebagai berikut:

```

{
  "service": "23587103",
  "train_uid": "G82394",
  "headcode": "",
  "toc": {
    "atoc_code": "SR"
  },
  "train_status": "P",
  "origin_name": "Markinch",
  "destination_name": "Edinburgh Waverley",
  "stop_of_interest": null,
  "date": "2017-08-05",
  "time_of_day": "06:00",
  "mode": "train",
  "request_time": "2017-08-05T06:46:22+01:00",
  "category": "OO",
  "operator": "SR",
  "operator_name": "Scotrail",
  "stops": [
    {
      "station_code": "MNC",
      "tiploc_code": "MKIN",
      "station_name": "Markinch",
      "stop_type": "LO",
      "platform": "1",
      "aimed_departure_date": "2017-08-05",
      "aimed_departure_time": "06:19",
      "aimed_arrival_date": null,
      "aimed_arrival_time": null,
      "aimed_pass_date": null,
      "aimed_pass_time": null
    },
    {
      "station_code": "GLT",
      "tiploc_code": "GLNRTHS",
      "station_name": "Glenrothes With Thornton",

```

6.3 Desain Percakapan

Sekarang setelah kita memiliki gambaran bagus tentang data yang kita miliki, mari kita bertukar pikiran tentang tugas percakapan untuk chatbot kita.

- Stasiun terdekat
- Kereta berikutnya
- Waktu kedatangan
- Nanti kereta dari stasiun
- Bagaimana cara saya pergi dari A ke B?

Mari kita mulai dengan hal sederhana: pergi ke stasiun terdekat.

Stasiun terdekat

Untuk mencapai stasiun terdekat bagi pengguna, kami memerlukan lokasinya. Ini bisa dalam bentuk kode pos atau koordinat latLon. Platform seperti Facebook Messenger memungkinkan pengguna untuk berbagi lokasi dalam bentuk koordinat lintang. Namun karena kita akan menggunakan platform SMS, maka gunakan jalur kode pos. Percakapan untuk tugas ini dapat dilakukan melalui salah satu cara berikut:

Pengguna : *di mana stasiun terdekat saya?*
Bot : *bisakah Anda memberi saya kode pos Anda?*
Pengguna : *EH12 9QR*
Bot : *bagus. Stasiun terdekat Anda adalah South Gyle*
Pengguna : *apa stasiun terdekat untuk EH12 9QR?*
Bot : *stasiun terdekat adalah South Gyle*

Kereta berikutnya

Untuk mendapatkan informasi mengenai kereta selanjutnya, percakapan dapat dilakukan dengan cara sebagai berikut:

Pengguna : *kapan jadwal kereta berikutnya ke Glasgow?*
Bot : *dari stasiun mana?*
Pengguna : *Edinburgh Park*
Bot : *kereta berikutnya ke Glasgow Central adalah pukul 10.00*

Pengguna : *kereta berikutnya*
Bot : *Dari mana?*
Pengguna : *Edinburgh Park*
Bot : *mencari kereta berikutnya dari taman edinburgh. Pergi ke?*

Pengguna : *Glasglow*
Bot : *kereta berikutnya ke Glasglow Central adalah pukul 10.00*

Waktu kedatangan

Terkadang pengguna mungkin ingin mengetahui waktu kedatangan kereta dalam konteks di stasiun tertentu. Untuk mendapatkan informasi waktu kedatangan, percakapan dapat dilakukan sebagai berikut:

Pengguna : *Jam berapa kereta tiba di Glasglow Central?*
Bot : *Kereta akan tiba di Glasglow Central pada pukul 11.00*

Masih banyak lagi percakapan yang mungkin dilakukan dalam bidang perjalanan. Semua tugas sebelumnya bersifat reaktif, di mana bot merespons permintaan pengguna. Sebaliknya, tugas proaktif dapat dirancang dengan meminta bot memulai percakapan. Misalnya, bot dapat mengirimkan waktu kereta dan informasi penundaan/pembatalan kepada pengguna pada waktu yang ditentukan sepanjang hari:

Bot : kereta pukul 15.00 ke Edinburgh Waverley ditunda hingga pukul 15.30.

Pengguna : apakah ada kereta yang lebih awal ke Haymarket?

Tugas percakapan seperti daftar kereta selanjutnya dan merencanakan rute Anda lebih kompleks dibandingkan tugas sebelumnya. Saat kita melanjutkan, kita akan melihat cara membuat chatbot yang dapat menangani beberapa tugas yang disebutkan.

6.4 Membangun Bot SMS Sederhana

Sekarang mari kita membangun antarmuka platform SMS untuk chatbot. Untuk mengatasi kerumitan ini, mari kita lakukan ini dalam dua langkah. Pertama, mari kita membuat bot untuk sekadar mengirim pesan teks SMS ke nomor ponsel. Ini bisa berupa pesan mengenai status kereta yang tiba di suatu stasiun atau kereta berikutnya yang menuju tujuan tertentu dari stasiun tertentu. Kedua, kami akan membangun chatbot dua arah yang dapat menerima pesan dari pengguna dan meresponsnya dengan tepat. Untuk melakukan ini, kita akan menggunakan layanan bernama Twilio. Twilio adalah platform pengembang untuk komunikasi yang memungkinkan pengembang menambahkan kemampuan perpesanan, suara, dan video ke perangkat lunak mereka. Kami akan mengeksplorasi bagaimana kami dapat membuat bot notifikasi dan chatbot menggunakan infrastruktur perpesanan Twilio.

Mulai

Untuk memulai dengan Twilio, lakukan langkah-langkah berikut:

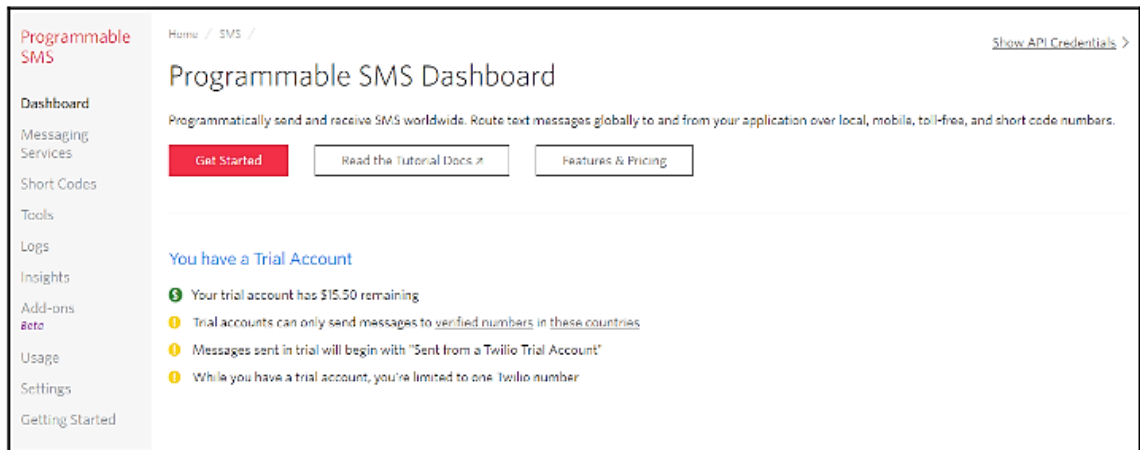
1. Kunjungi www.twilio.com, klik Daftar dan daftarkan diri Anda untuk akun uji coba gratis.
2. Setelah Anda mendaftar, buka halaman konsol di www.twilio.com/console.
3. Salin SID AKUN dan Kunci Akun. Kami akan menggunakannya untuk proyek kami.

Menyiapkan dasbor

Mari kita membuat bot yang mengirimkan notifikasi ke nomor ponsel pengguna. Twilio memiliki empat produk utama: Programmable Chat, Programmable SMS, Programmable Voice, dan Programmable Video. Untuk membangun pengirim notifikasi, kita perlu menggunakan layanan Programmable SMS.

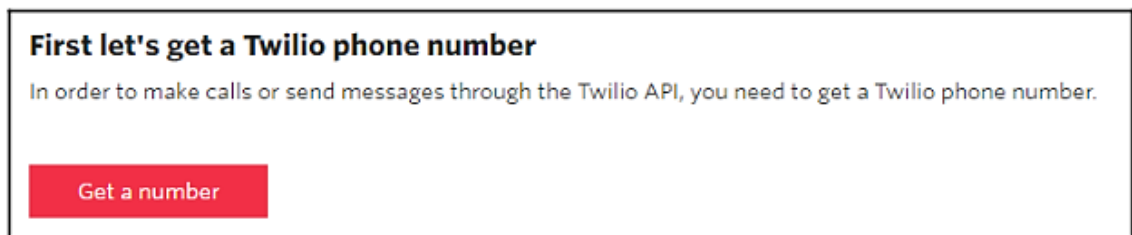
Berikut langkah-langkahnya:

1. Di dasbor konsol, pilih SMS yang Dapat Diprogram.
2. Pada Dasbor SMS yang Dapat Diprogram, klik Memulai:



Gambar 6.2 Dashbord SMS

3. Untuk mengirim pesan SMS, Anda memerlukan nomor telepon. Klik Dapatkan nomor untuk mendapatkannya:



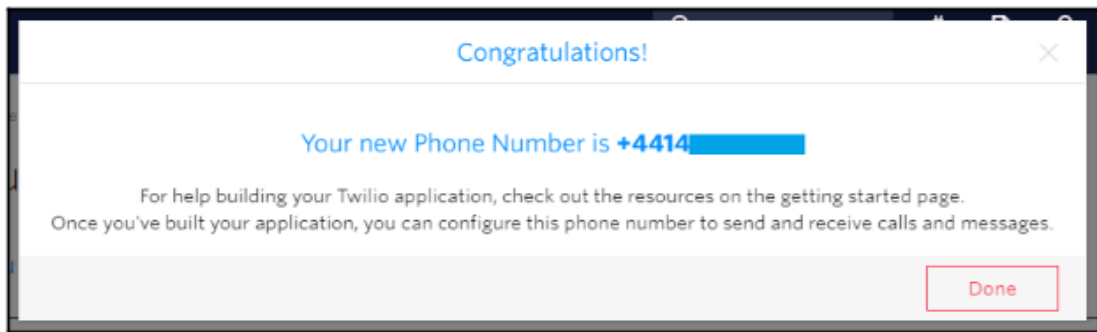
Gambar 6.3 mengirim pesan SMS

4. Ini akan memberi Anda nomor. Anda dapat memilih untuk menerimanya dengan mengklik Pilih Nomor ini. Jika tidak, pilih yang lain. Pastikan nomor tersebut memiliki kemampuan SMS:



Gambar 6.4 Memasukan nomor

5. Anda akan menerima pengakuan bahwa Anda telah diberi nomor:'



Klik Selesai.

Gambar 6.5 berhasil menambahkan nomor telepon.

6. Apakah Anda melihat jendela Kirim Pesan? Mengapa Anda tidak mengirim pesan kepada diri Anda sendiri? Kirim pesan percobaan ke nomor ponsel Anda yang terdaftar (nomor yang Anda gunakan untuk verifikasi akun)

here.'"/>

Gambar 6.6 Verifikasi nomor telepon

7. Periksa ponsel Anda untuk melihat apakah Anda telah menerima pesan tes Anda. Klik Ya untuk memberi tahu Twilio bahwa Anda telah menerima pesan Anda.

Pengirim Pesan Sederhana

Sekarang kita telah menyiapkan akun dan mendapatkan nomor telepon, mari kita lanjutkan membuat bot untuk mengirim notifikasi. Untuk melakukannya, lakukan langkah-langkah berikut:

1. Buka jendela konsol dan buat direktori bernama SMSBot.
2. Di direktori SMSBot, buat proyek Node.js baru menggunakan perintah `npm init`, seperti yang ditunjukkan di sini:

```

C:\Users\Srini\Documents\workspace\SMSBot>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (SMSBot) smsbot
version: (1.0.0)
description: A simple SMS sending bot
entry point: (index.js)
test command:
git repository:
keywords:
author: Srini Janarthanam
license: (ISC)
About to write to C:\Users\Srini\Documents\workspace\SMSBot\package.json:
{
  "name": "smsbot",
  "version": "1.0.0",
  "description": "A simple SMS sending bot",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Srini Janarthanam",
  "license": "ISC"
}

```

Periksa direktori untuk melihat file yang telah dihasilkan. Anda akan melihat file bernama package.json dengan informasi meta mengenai proyek tersebut.

3. Kita perlu menginstal perpustakaan Twilio Node.js untuk membangun bot SMS kita. Di konsol, jalankan perintah `npm install twilio --save`:

```

C:\Users\Srini\Documents\workspace\SMSBot>npm install twilio
npm WARN package.json smsbot@1.0.0 No repository field.
npm WARN package.json smsbot@1.0.0 No README data
twilio@3.6.1 node_modules\twilio
├── deprecate@1.0.0
├── scmp@0.0.3
├── rootpath@0.1.2
├── xmlbuilder@9.0.1
├── q@2.0.3 (weak-map@1.0.5, pop-iterate@1.0.1, asap@2.0.6)
├── request@2.81.0 (aws-sign2@0.6.0, tunnel-agent@0.6.0, forever-agent@0.6.1, oauth-sign@0.8.2, is-type-darray@1.0.0, caseless@0.12.0, safe-buffer@5.1.1, stringstream@0.0.5, aws4@1.6.0, isstream@0.1.2, json-stringify-safe@5.0.1, extend@3.0.1, performance-now@0.2.0, uuid@3.1.0, qs@6.4.0, combined-stream@1.0.5, mime-types@2.1.16, tough-cookie@2.3.2, form-data@2.1.4, hawk@3.1.3, http-signature@1.1.1, har-validator@4.2.1)
├── moment@2.18.1
├── jsonwebtoken@7.4.2 (ms@2.0.0, lodash.once@4.1.1, xtend@4.0.1, jws@3.1.4, joi@6.10.1)
├── lodash@4.0.0

```

4. Mari buat file JS baru bernama index.js. Tambahkan kode berikut ke file:

```

//index.js-SMSBot
// Add your Account SID
Var accountSid = 'your_account_sid';
//Add your Auth Token here

```

```

Var authToken = ` your_auth_token`;

Var twilio = require (`twilio`);
Var client = new twilio (accountSid, authToken) ;

//Create a message with to and from numbers
Clients.message.create ({
  Body : `Srini says hello `,
  To : `+447888999999 `,
  From : `+447888999990
})
.then (message) => console.log(message.sid) `

```

Pada kode sebelumnya, nomor to harus berupa nomor akun percobaan yang terverifikasi. Anda tidak dapat mengirim pesan ke nomor lain kecuali Anda meningkatkan akun Anda. Nomor dari adalah nomor Twilio yang telah Anda peroleh sebelumnya.

5. Simpan file dan jalankan menggunakan perintah node index.js. Ini akan mengirimkan pesan teks ke nomor telepon Anda yang terverifikasi.

Pemberitahu kereta saya

Bayangkan sebuah skenario ketika pengguna bepergian ke tempat kerja setiap hari dari Edinburgh Waverley ke Glasgow Queen Street. Dan mereka kesulitan menentukan jadwal kereta dan waktunya saat bersiap berangkat setiap pagi. Bukankah lebih bagus jika kita bisa menyediakan layanan yang mengirimkan daftar kereta dari stasiun mereka ke tujuan mereka pada suatu saat selama rutinitas pagi mereka?

Dengan menggunakan modul Pengirim Pesan Sederhana sebelumnya, mari kita buat bot yang akan mengirimkan daftar kereta dari stasiun tertentu ke stasiun tujuan tertentu. Mari tambahkan modul untuk mendapatkan daftar kereta dari stasiun tertentu ke tujuan tertentu:

1. Instal perpustakaan permintaan menggunakan npm install request --save.
2. Buat fungsi untuk mengirim notifikasi SMS:

```

Function sendSMS (msg, userPhoneNumber) {
  Var twilio = require (`twilio`);
  Var client = new twilio (accountSID, AUTHtOKEN) ;
  //Create a message with to and from numbers
  Client.message.create ( {
    Body : msg,
    To : userPhoneNumber
    From : `+4414xxxxxx`//YOUR_NUMBER
  } )

```

3. Buat fungsi untuk membuat semua kereta berangkat dari stasiun tertentu:

```
Function getTrains (sourcestation, sourceStationCode,
    destinationStation, userPhoneNumber) {
    var request = require ('request') ;
var url = 'http://transportapi.com/v3/uk/train/station/' +
    sourceStationCode + '/live.json?
    app_id=YOUR_APP_ID&app_key=YOUR_APP_KEY' ;
request (url,function (error,response,body) {
    if (response) {
        var json=JSON.parse(body) ;
        if (json.depatures) {
            //console.log ('Depatures : ',
//JSON.stringify (json.depatures) ) ;
Var dep =
getTrainsToDestination (destinationStation,
json.depatures.all) ;
var summary = summarize (destinationStation,
json.depatures.all ) ;
var summary = summarize (destinationStation,
sourchStation,dep ) ;
console.log('summary : '+ summary ) ;
sendSMS (summary, userPhoneNumber) ;}else {
    console.log ('No Departures Found!') ;
}
}
}
else {
    Console.log ('error : ',error) ; //Print the error if onr
// occuerred
}
}
}
};
}
```

4. Buat fungsi untuk mengambil semua kereta menuju stasiun tujuan tertentu:

```
Function getTrainsToDestination (destination, allDepartures) {
    Var d = [ ] ;
    If ( allDepartures ) {
        For ( var l=0 ; l < allDepartures.lenght ; l++ ) {
            Var service = allDepartures[l] ;
            If ( service.destination_name == destination) {
                d.push(service)
            }
        }
    }
    Return d ;
}
```

5. Buat fungsi untuk memanggil fungsi sebelumnya untuk mengirimkan pemberitahuan kepada pengguna tentang semua kereta yang menuju ke stasiun tujuan tertentu dari stasiun pilihan pengguna:

```

Function summarize ( destinationStation, sourceStation, departures )
{
    Var out = ` ` ;
    If (departures.lenght> 0) {
        Out = `Here are the departures this morning to ` +
            destinationStation
        for(var i=0 ; i< departures.length;i++){
            var service = departures [i];
            var serviceSummary = service.operator _name +"at"+
                service.expected_departure_time ;
            out += serviceSummary +"\n" }
        }
    }else{
        Iut = There are no trains to ` + destinationStation + `
from' +sourceStation;
        }
        Return out ;
    }
}

```

6. Dan atur variabelnya dan panggil modul utama:

```

//Index.js
//Add your Account SID
Var account Sid = ` your_account_sid`;
// your Auth Token here
Var authToken = `your_auth_token `;

Var destinationStation = `Glasgow Queen Street `;
Var userphoneNumber ='+44788899999';
Var sourceStationCode = `EDB` ;
Var sourceStation = `Edinburgh Waverley `;

getTrains (sourceStation,sourceStationCode , destinationStation,
userPhoneNumber) ;

```

7. Jalankan di konsol node index.js untuk melihat apakah berhasil. Ini akan mencetak ringkasan kereta ke stasiun tujuan di konsol dan juga mengirimkannya sebagai pesan pemberitahuan SMS kepada Anda:

```

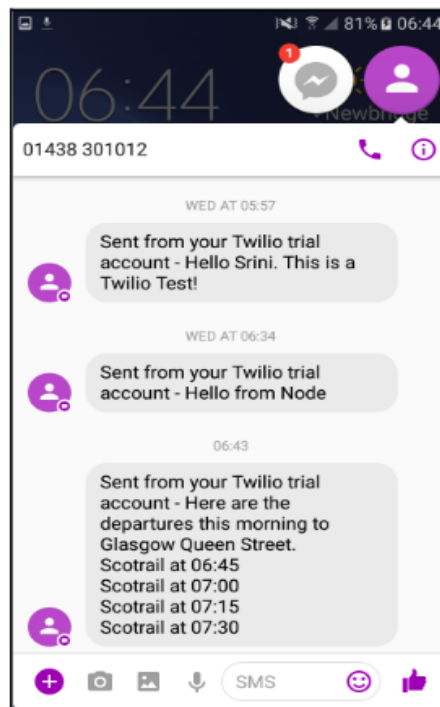
C:\Users\Srini\Documents\workspace\SMSBot>node index.js
Summary: Here are the departures this morning to Glasgow Queen Street.
Scotrail at 06:45
Scotrail at 07:00
Scotrail at 07:15
Scotrail at 07:30

SM79cb0e9ed7e5499f99ec35b3c2f77681

C:\Users\Srini\Documents\workspace\SMSBot>

```

8. Periksa ponsel Anda untuk melihat apakah pesan sudah sampai:



Gambar 6.7 tampilan pesan

9. Selamat! Anda baru saja membuat bot SMS.

6.5 Menjadwalkan Tugas

Sekarang kita memiliki bot yang mengirimkan notifikasi SMS, mari kita coba mengaturnya agar berjalan secara otomatis setiap hari atau setiap jam. Fitur ini akan berguna untuk membuat bot proaktif yang memulai percakapan dengan pengguna pada waktu-waktu tertentu dalam sehari. Untuk melakukannya, ikuti langkah-langkah berikut:

1. Buat direktori bin.
2. Pindahkan file index.js ke direktori bin. Ganti namanya menjadi sendTrainNotification.js.
3. Tambahkan baris kode pertama sebagai berikut:
#!/usr/bin/env node
4. Kembali ke direktori proyek. Kami sekarang akan memasukkan aplikasi ini ke cloud Heroku.
5. Buat repositori Git, tambahkan file, dan komit:

Git init

Git add

Git commit -m "initial commit"

6. Buat aplikasi Heroku:

Percakapan Antarmuka Pengguna (User Interface) Dalam Chatbot – Dr. Budi Raharjo

Heroku create sms-notification-bot

7. Dorong aplikasi ke Heroku:

Git push Heroku master

Kami sekarang memiliki aplikasi yang berjalan

<https://sms-notification-bot.herokuapp.com>.

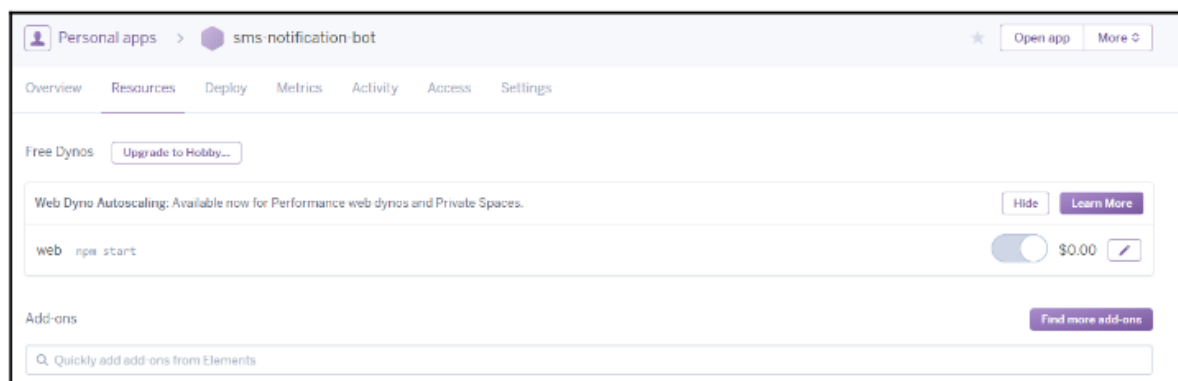
8. Jalankan aplikasi secara lokal:

Heroku run sendTrainNotification.js

Ini harus menjalankan aplikasi web dan mengirim SMS berisi ringkasan kereta ke ponsel pengguna:

```
C:\Users\Srini\Dropbox\_Book\workspace\sms-notification-bot>heroku run sendTrainNotification.js
Running sendTrainNotification.js on sms-notification-bot... up, run.6693 (Free)
Running - SMS Train Notification
Summary: Here are the departures this morning to Glasgow Queen Street.
Scotrail at 16:45
Scotrail at 17:00
Scotrail at 17:15
```

9. Sekarang, kita perlu menjadwalkan tugas tersebut. Untuk melakukan ini, kita perlu mengaturnya di halaman sumber daya Heroku di aplikasi. Buka <https://dashboard.heroku.com/apps/sms-notification-bot/resources> di browser Anda:



Gambar 6.8 halaman sumber daya Heroku

10. Ketik Penjadwal di kotak pencarian Add-on dan pilih Penjadwal Heroku.
 11. Setelah ditambahkan, klik Heroku Scheduler. Ini akan membawa Anda ke <https://scheduler.heroku.com/dashboard>.
 12. Klik Tambahkan pekerjaan baru.
 13. Dalam kotak teks dengan \$, ketikkan nama tugas yang akan dijalankan (itu adalah, sendTrainNotification.js). Pilih Frekuensi (Harian, Setiap Jam, atau Setiap 10 menit) dan klik SAVE:

DYNO SIZE	FREQUENCY	LAST RUN	NEXT DUE
Free	Every 10 minutes	Aug 13 6:57 UTC	Aug 13 7:07 UTC

Gambar 6.9 kotak teks

14. Periksa log di konsol log Heroku. Anda harus memperhatikan bahwa tugas akan berjalan pada interval yang ditentukan secara teratur dan SMS dikirim ke pengguna:

```

2017-08-13T07:07:25.867628+00:00 heroku[scheduler.4555]: Starting process with command 'sendTrainNotification.js'
2017-08-13T07:07:26.517425+00:00 heroku[scheduler.4555]: State changed from starting to up
2017-08-13T07:07:28.022517+00:00 app[scheduler.4555]: Running - SMS Train Notification
2017-08-13T07:07:29.133248+00:00 app[scheduler.4555]: Summary: Here are the departures this morning to Glasgow Queen Street.
2017-08-13T07:07:29.133261+00:00 app[scheduler.4555]: Scotrail at 08:30
2017-08-13T07:07:29.133261+00:00 app[scheduler.4555]: Scotrail at 09:00
2017-08-13T07:07:29.133262+00:00 app[scheduler.4555]: Scotrail at 09:30
2017-08-13T07:07:29.133264+00:00 app[scheduler.4555]:
2017-08-13T07:07:29.133263+00:00 app[scheduler.4555]: Scotrail at 10:00
2017-08-13T07:07:29.884633+00:00 app[scheduler.4555]: SM7f6591a12f8f43eab488c7be5d5a9fbf
2017-08-13T07:07:29.892212+00:00 heroku[scheduler.4555]: State changed from up to complete
2017-08-13T07:07:29.880146+00:00 heroku[scheduler.4555]: Process exited with status 0

```

Gambar 6.10 konsol log Heroku

Selamat! Anda sekarang telah membuat bot SMS proaktif.

6.6 Membangun Chatbot Dua Arah

Sejauh ini, kami telah membangun bot yang dapat mengirimkan notifikasi SMS ke pengguna pada interval waktu yang ditentukan. Meski proaktif, komunikasinya hanya satu arah. Pengguna tidak dapat mengirimkan permintaan apa pun ke bot untuk mengubah sifat atau isi pesan yang dikirimkannya. Mari kita kerjakan itu.

Untuk membangun chatbot yang dapat berkomunikasi dua arah, kita perlu melakukan dua hal: membangun chatbot ke dalam aplikasi web dan mengubah konfigurasi pengaturan di Twilio. Untuk melakukannya, ikuti langkah-langkah berikut:

1. Buat file `index.js` di direktori root proyek.
2. Instal perpustakaan ekspres dan `body-parser`. Pustaka berikut akan digunakan untuk membuat aplikasi web:

```

Npm install body-parser-save
Npm install express --save

```

3. Buat aplikasi web di `index.js`:

```

// Two-way SMS Bot
Const express = require ('express')
Const bodyParser = require ('body-parser')
Const twilio = require ('twilio')

```

```

Cinst app = express ( )

```

Percakapan Antarmuka Pengguna (User Interface) Dalam Chatbot – Dr. Budi Raharjo

```

App.set ( 'port' , (process.env.PORT || 5000) )
// Process application/x-www-form-urlencoded
App.use(bodyParser.urlencoded ( {extended : false } ) )
// Process application/json
App.use(bodyParser.json ( ) )
//Spin up the server
App.listen (app.get ( 'port' ) , function ( ) {
    Console.log( 'running on port' , app.get( 'port' )
} )
// Index route
App.get ( '/' , function (req,res ) {
    res.send ( 'Hello world, I am SMS bot.' )
} )
//Twilio webhook
App.post ( '/sms/' ,function (req,res) {
    Var botSays = `You said : ` +req.body.Body
    Var twiml = new twilio.TwimlResponse ( );
    Twiml.message (botSays) ;
    Res.writeHead (200,{ 'Content-Type' : `text/xml ` } ) ;
    Res.end(twiml.toString ( ) ) ;
} )

```

Kode sebelumnya membuat aplikasi web yang mencari pesan masuk dari pengguna dan meresponsnya. Responsnya saat ini adalah mengulangi apa yang dikatakan pengguna.

4. Dorong ke cloud:

```

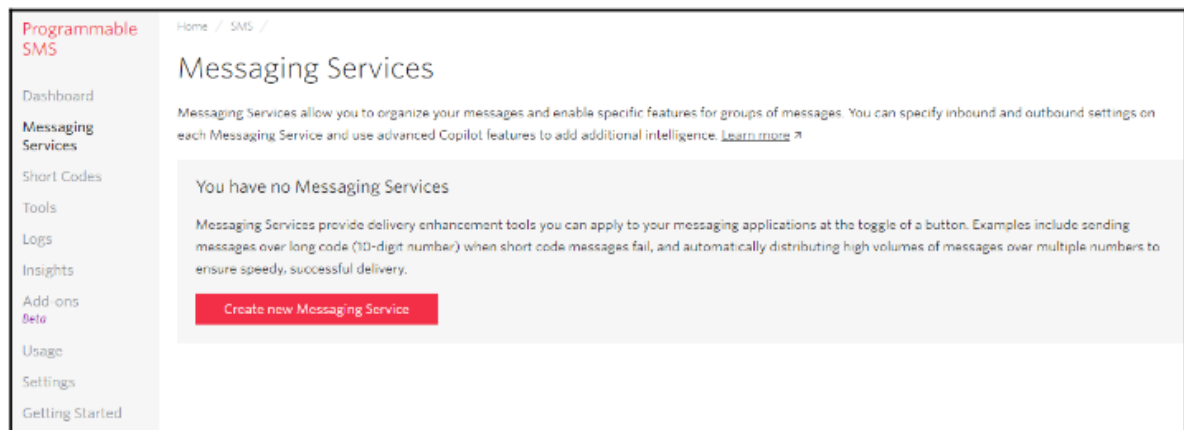
git add .
git commit -m webapp
git push Heroku master

```

Sekarang kami memiliki aplikasi web di cloud di <https://sms-notification-bot.herokuapp.com/sms/> yang dapat dipanggil ketika ada pesan SMS masuk. Aplikasi ini akan menghasilkan respon chatbot yang sesuai dengan pesan masuk.

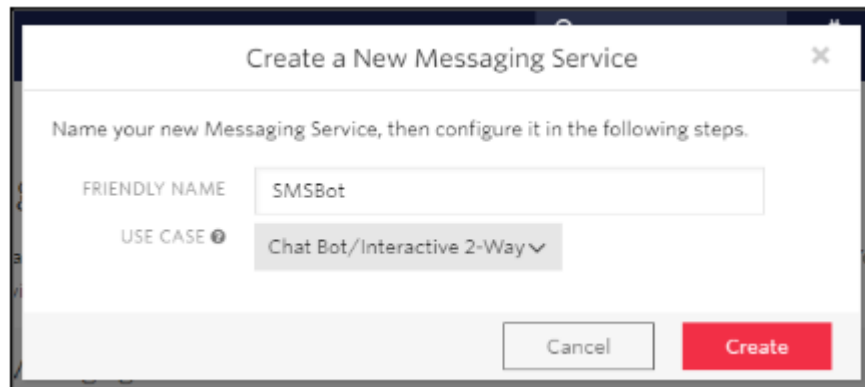
5. Masuk ke halaman Dasbor SMS Twilio yang Dapat Diprogram di <https://www.twilio.com/konsol/sms/dasbor>.

6. Pilih Layanan Pesan pada menu dan klik Buat Layanan Pesan baru:



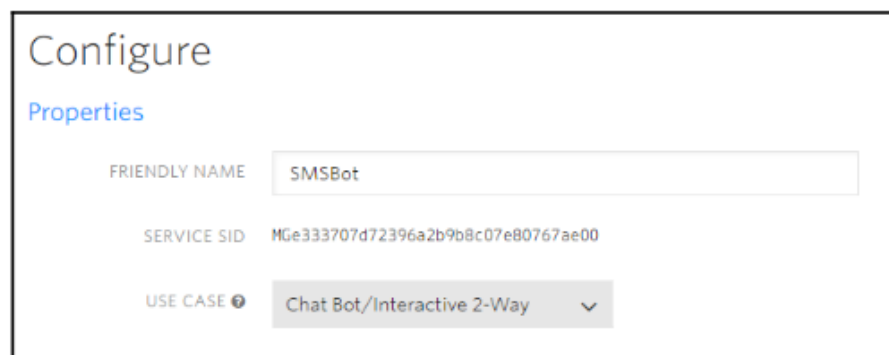
Gambar 6.11 Layanan pesan

7. Beri nama dan pilih Chat Bot/Interactive 2-Way sebagai kasus penggunaan:



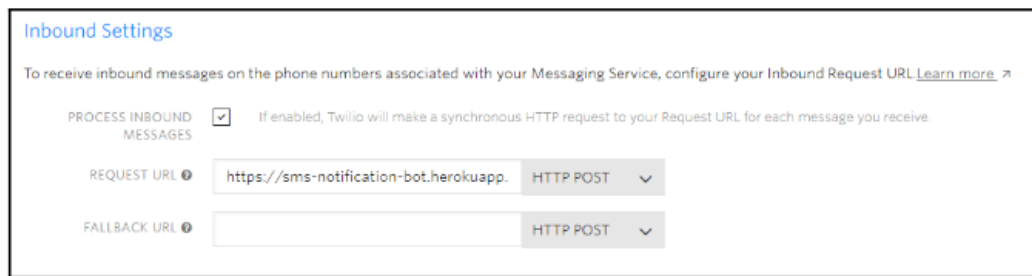
Gambar 6.12 layanan membuat pesan baru

8. Ini akan membawa Anda ke halaman Konfigurasi dengan ID layanan yang baru ditetapkan:



Gambar 6.12 konfigurasi

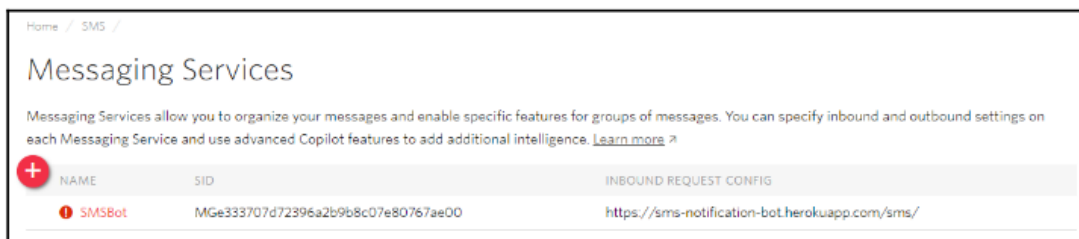
9. Di bawah Pengaturan Masuk, tentukan URL aplikasi web yang telah kita buat di bidang URL PERMINTAAN (yaitu, <https://sms-notification-bot.herokuapp.com/sms/>):



Gambar 6.13 tampilan inbound setting

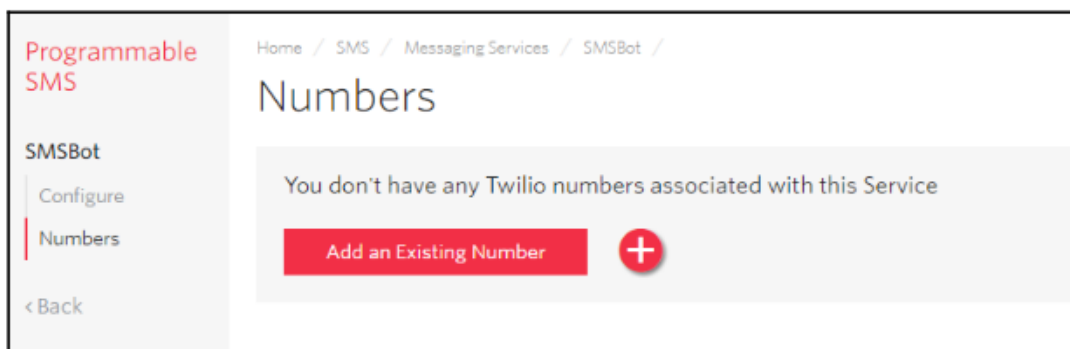
Sekarang semua pesan masuk akan dialihkan ke aplikasi web ini.

10. Kembali ke halaman konsol SMS di <https://www.twilio.com/console/sms/services>. Di sini Anda akan melihat layanan perpesanan baru Anda tercantum bersama dengan URL permintaan masuk:



Gambar 6.14 layanan pesan

11. Klik layanan untuk melampirkan nomor ke layanan:



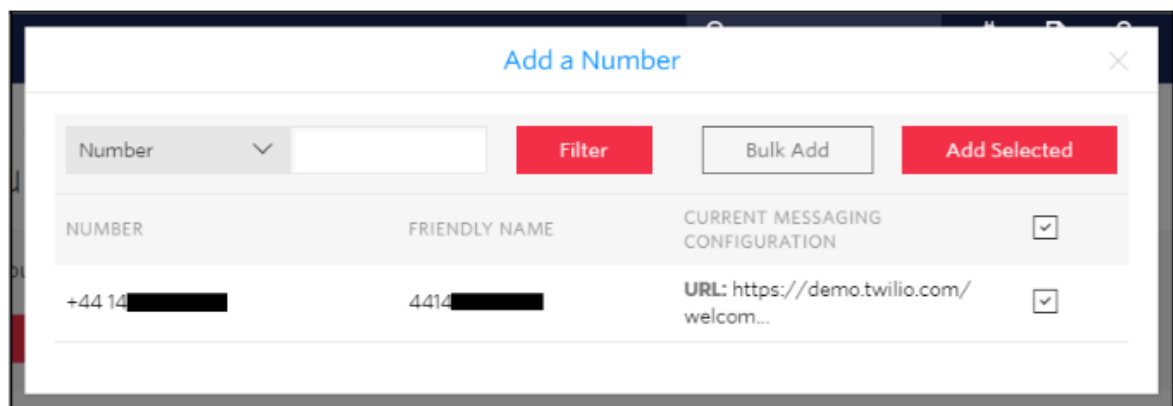
Gambar 6.15 masukan nomor

12. Anda dapat menambahkan nomor baru, dalam hal ini Anda perlu membelinya atau memilih nomor yang sudah Anda miliki. Kami sudah memiliki satu notifikasi pengiriman yang dapat digunakan kembali. Klik Tambahkan Nomor yang Ada.
13. Pilih nomor dengan mencentang kotak di sebelah kanan dan klik Tambah yang Dipilih:



Gambar 6.16 memasukan nomor

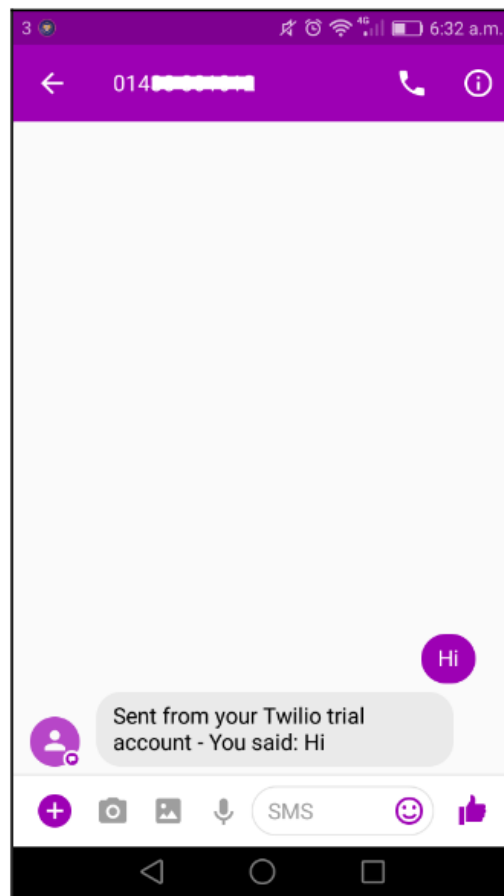
14. Setelah ditambahkan maka akan tercantum pada halaman Numbers sebagai berikut:



Gambar 6.17 menambahkan additional nomor

Dalam pengaturan Lanjutan, kita dapat menambahkan beberapa nomor untuk melayani wilayah geografis yang berbeda dan meminta nomor tersebut merespons seolah-olah chatbot merespons melalui nomor lokal.

15. Langkah terakhir adalah mencoba mengirimkan pesan SMS ke nomor tersebut dan menerima balasan. Kirim pesan menggunakan aplikasi SMS apa pun di ponsel Anda dan amati responsnya:



Gambar 6.18 tampilan chatbot

Selamat! Anda sekarang memiliki chatbot interaktif dua arah.

6.7 Memahami Ucapan Pengguna

Sekarang setelah kita memiliki chatbot dua arah dasar, mari kita lihat bagaimana kita dapat memproses ucapan pengguna, memahami permintaan informasi mereka, dan melayani mereka secara efektif. Untuk memproses ucapan pengguna, kita akan menggunakan toolkit API.AI yang diperkenalkan di Bab 3, Mari Bicara Cuaca. Ucapan pengguna akan diproses dan diubah menjadi maksud dan parameter pengguna oleh API.AI. Ini kemudian akan digunakan untuk mengambil dan menyajikan informasi yang sesuai kepada pengguna.

Membuat agen API.AI

Pertama-tama mari kita buat agen API.AI untuk memahami ucapan pengguna dalam tugas percakapan yang telah kita sebutkan sebelumnya. Karena kita telah membahas pembuatan agen API.AI secara ekstensif di Bab 3, Mari Bicara Cuaca, rangkaian langkah berikut akan dijelaskan secara singkat. Namun, kami akan berinteraksi dengan agen dengan cara yang berbeda dibandingkan dengan yang dilakukan di Bab 3, Mari Bicara Cuaca:

1. Buka api.ai di browser web Anda. Masuk dengan kredensial Google Anda dan klik GO TO CONSOLE.

2. Klik Buat agen baru. Anda akan menemukannya di bagian bawah daftar drop-down yang mencantumkan semua agen yang Anda buat sejauh ini.
3. Beri nama, deskripsi, dan pilih zona waktu. Klik Simpan.
4. Klik Entitas. Buat entitas baru untuk stasiun dengan kode stasiun sebagai nilai referensi dan nama serta alternatif sebagai sinonim:

Code	Description
EDB	EDB, Edinburgh, Edinburgh Waverley, Waverley
GLC	GLC, Glasgow Central
GLQ	GLQ, Glasgow Queen Street, Queen Street
ABD	ABD, Aberdeen
DEE	DEE, Dundee

Gambar 6.19 membuat entitas baru

5. Klik Buat Niat. Buat dua maksud berikut untuk memulai:
 - request_live_departures: Untuk meminta beberapa keberangkatan berikutnya dari stasiun tertentu:

request_live_departures

Contexts

User says: Search in user says

Add user expression

- show me trains leaving aberdeen right now
- can you show me departures at glasgow central
- get me train departures from dundee
- show me live departures at waverley

Events

Action

Enter action name

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST	PROMPTS
<input checked="" type="checkbox"/>	Station	@Station	\$Station	<input type="checkbox"/>	Departures at w...
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>	—

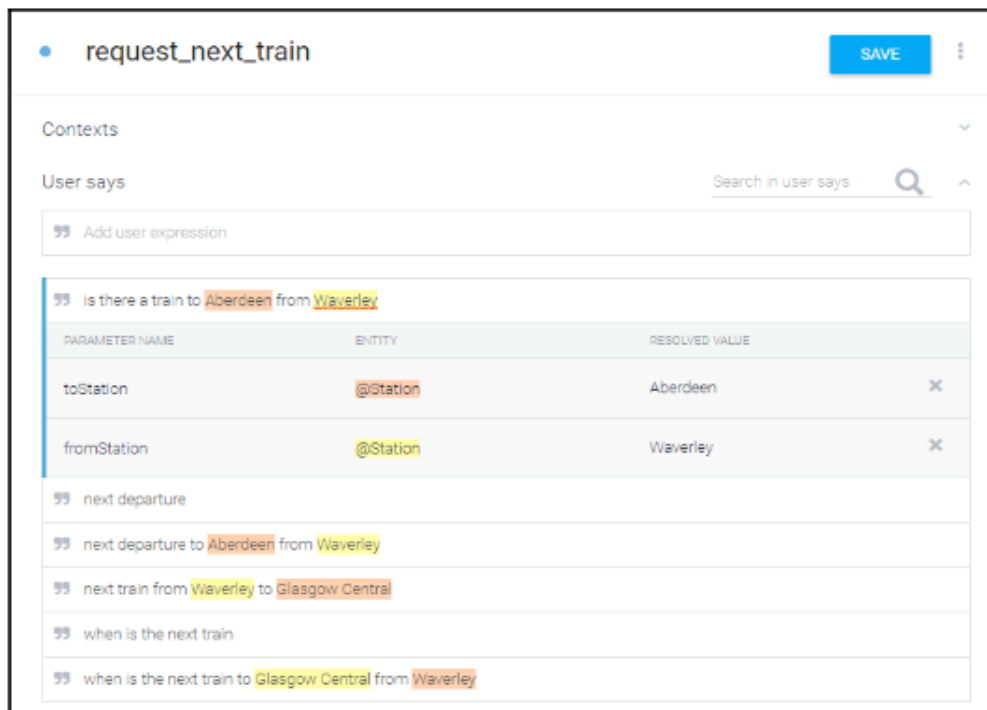
Gambar 6.20 tampilan request_live_departures

Tetapkan parameter Stasiun sesuai kebutuhan dan tentukan perintah yang akan digunakan jika parameter tidak ada dalam ucapan pengguna.



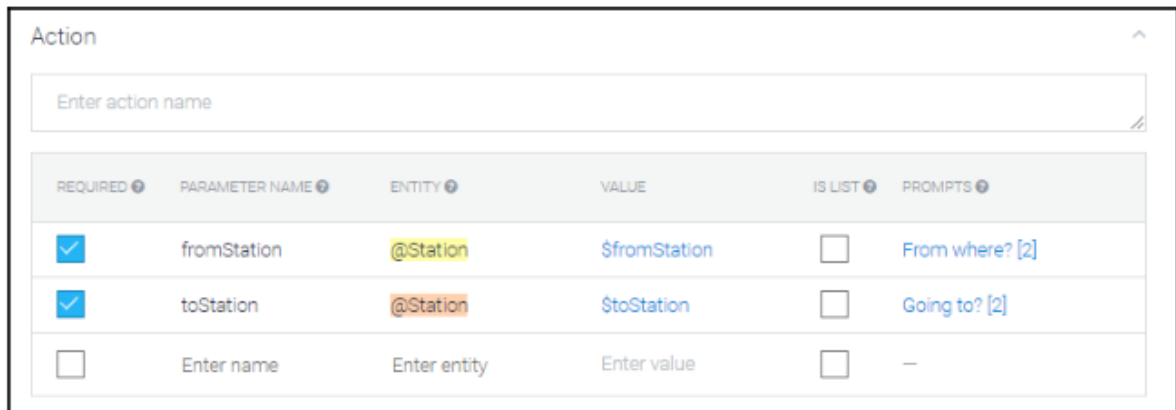
Gambar 6.21 Perintah pencarian stasiun

- request_next_train: Untuk meminta kereta berikutnya dari stasiun tertentu ke stasiun tertentu lainnya:



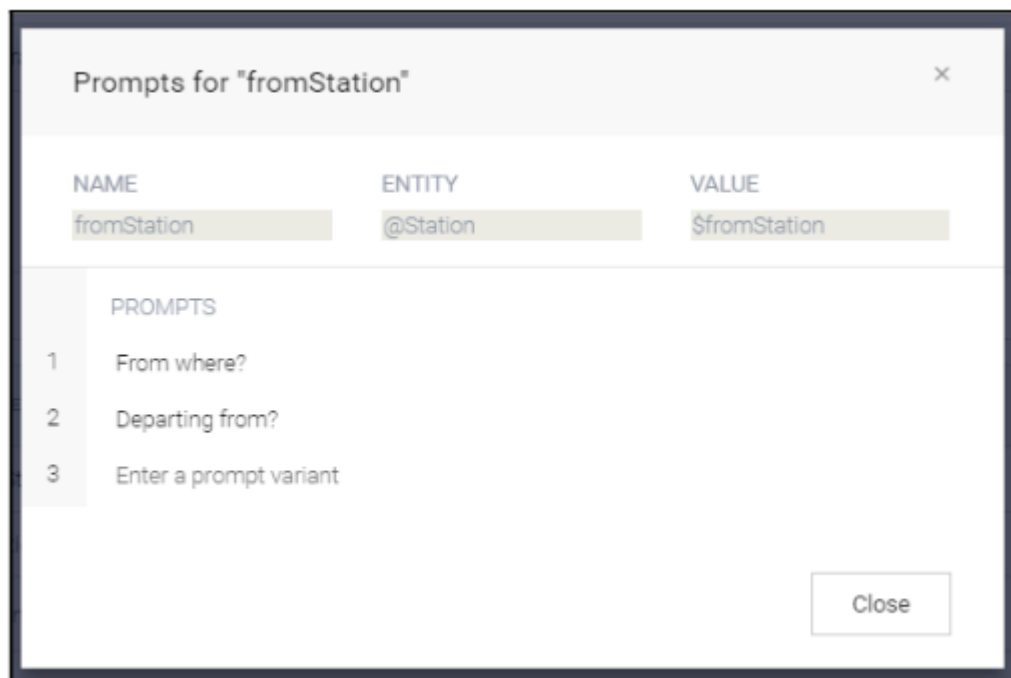
Gambar 6.22 tampilan pilihan pencarian

Tetapkan parameter fromStation dan toStation sesuai kebutuhan:



Gambar 6.23 parameter pencarian

Dan tentukan petunjuk yang akan digunakan jika parameter tidak ada dalam ucapan pengguna. Misalnya, perintah untuk fromStation ditampilkan di sini:



Gambar 6.24 tampilan pencarian "dari stasiun"

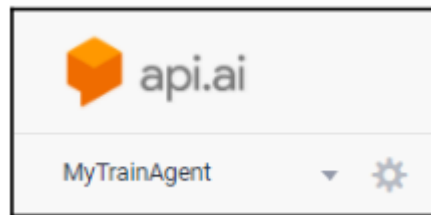
Buat antarmuka Node.js ke agen API.AI

Ada banyak cara untuk terhubung ke agen API.AI. Kita telah membahas beberapa di antaranya dalam Bab 3, Mari Bicara Cuaca. Di bagian ini, kita akan mengeksplorasi satu pendekatan lagi:

1. Buat program Node.js bernama `apiai.js` di direktori root proyek.
2. Instal pustaka API.AI Node.js menggunakan perintah berikut dari konsol:

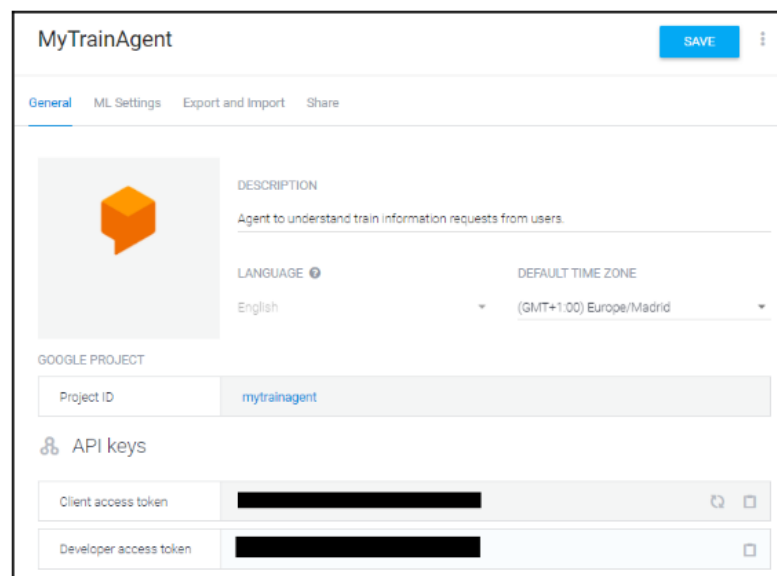
Npm install apiai -- save

3. Kembali ke pengaturan agen (klik ikon pengaturan di sebelah nama agen pada daftar drop-down) di konsol web:



Gambar 6.25 ikon api.ai

4. Di bawah tab Umum, salin token akses Pengembang:



Gambar 6.26 Tampilan pengembang

5. Kembali ke program Node.js, apiai.js, dan buat permintaan untuk mengakses agen. Gunakan token akses pengembang dari konsol dalam kode:

```
//API.AI
var apiai = require('apiai');
var apiai1 = apiai("YOUR_DEVELOPER_ACCESS_TOKEN");

var userUtterance = 'live departures from waverley';

var requestAPIAI = apiai1.textRequest(userUtterance, {
  sessionId: '12345'
});

requestAPIAI.on('response', function(response) {
  console.log(response);
});

requestAPIAI.on('error', function(error) {
  console.log(error);
});

requestAPIAI.end();
```

6. Jalankan program di konsol menggunakan perintah node apiai.js:

```
C:\Users\Srini\Documents\workspace\SMSBot>node apiai.js
{ id: 'a68a406c-9db3-4bc1-acf4-bd711bcac0a6',
  timestamp: '2017-08-15T05:51:31.241Z',
  lang: 'en',
  result:
    { source: 'agent',
      resolvedQuery: 'live departures from waverley',
      action: '',
      actionIncomplete: false,
      parameters: { Station: 'EDB' },
      contexts: [],
      metadata:
        { intentId: 'c35552da-b1b3-41ac-afb4-177533126c08',
          webhookUsed: 'false',
          webhookForSlotFillingUsed: 'false',
          intentName: 'request_live_departures' },
      fulfillment: { speech: '', messages: [Object] },
      score: 0.7699999809265137 },
  status: { code: 200, errorType: 'success' },
  sessionId: '12345' }
```

7. Amati respon yang ditampilkan di konsol. Anda akan menemukan maksud yang diklasifikasikan sebagai maksudNama dan parameter terkait. Dengan menggunakan ini, kita dapat mengambil informasi yang sesuai dari Transport API. Dan perhatikan bahwa tidak ada pidato pemenuhan karena kami tidak menentukannya saat membangun maksud.
8. Mari kita coba beberapa contoh lagi: ubah ucapan pengguna untuk menunjukkan keberangkatan dan jalankan kodenya:

```
C:\Users\Srini\Documents\workspace\SMSBot>node apiai.js
{ id: 'c8ad170d-df2d-4384-b78d-727cfb394259',
  timestamp: '2017-08-15T04:50:21.052Z',
  lang: 'en',
  result:
    { source: 'agent',
      resolvedQuery: 'waverley',
      action: '',
      actionIncomplete: false,
      parameters: { Station: 'EDB' },
      contexts: [],
      metadata:
        { intentId: 'c35552da-b1b3-41ac-afb4-177533126c08',
          webhookUsed: 'false',
          webhookForSlotFillingUsed: 'false',
          intentName: 'request_live_departures' },
      fulfillment: { speech: '', messages: [Object] },
      score: 1 },
  status: { code: 200, errorType: 'success' },
  sessionId: '12345' }
```

9. Amati perbedaannya dengan yang sebelumnya. Dalam ucapan pengguna ini, kami belum menentukan nama stasiun. Oleh karena itu, agen kembali dengan sebuah pertanyaan (dalam pidato pemenuhan): Keberangkatan di stasiun mana?. Perhatikan juga bahwa parameter `Station` kosong dan parameter `actionIncomplete` diatur ke `true`.
10. Sekarang ubah ucapan pengguna menjadi `waverley` dan jalankan lagi:

```
C:\Users\Srini\Documents\workspace\SMSBot>node apiai.js
{ id: 'c8ad170d-df2d-4384-b78d-727cfb394259',
  timestamp: '2017-08-15T04:50:21.052Z',
  lang: 'en',
  result:
    { source: 'agent',
      resolvedQuery: 'waverley',
      action: '',
      actionIncomplete: false,
      parameters: { Station: 'EDB' },
      contexts: [],
      metadata:
        { intentId: 'c35552da-b1b3-41ac-afb4-177533126c08',
          webhookUsed: 'false',
          webhookForSlotFillingUsed: 'false',
          intentName: 'request_live_departures' },
      fulfillment: { speech: '', messages: [Object] },
      score: 1 },
  status: { code: 200, errorType: 'success' },
  sessionId: '12345' }
```

11. Sekarang periksa kembali hasilnya. Tidak ada ucapan pemenuhan dan parameter `Stasiun` diatur ke `EDB` (yang merupakan kode stasiun untuk Edinburgh Waverley). Parameter `actionIncomplete` disetel ke `false`.

Apa yang telah kita lakukan sejauh ini adalah membuat program Node.js untuk memanggil agen API.AI menggunakan perpustakaan Node.js-nya. Kami juga telah menjalankan dan menjalankan kembali program untuk mensimulasikan percakapan. Kami telah menggunakan agen sebagai modul untuk memahami ucapan bahasa alami dan mengelola percakapan. Sekarang kita perlu menghubungkan ini ke program utama Node.js, `index.js` untuk mengambil hasil dari Transform API dan mengaktifkan percakapan melalui SMS.

Mengintegrasikan agen API.AI ke chatbot SMS

Sekarang kita memiliki agen API.AI dan antarmuka Node.js untuk berinteraksi dengannya, mari kita beralih untuk mengintegrasikannya dengan chatbot SMS dua arah yang telah kita buat. Untuk melakukan ini, kita perlu mengunjungi kembali file `index.js` yang merupakan aplikasi web yang merespons pesan SMS yang masuk. Kami akan menyalin dan mengerjakan ulang kode dari `apiai.js` untuk memanggil agen API.AI dari `index.js`:

1. Di webhook Twilio, panggil agen API.AI dengan ucapan pengguna. Kami juga akan menggunakan nomor telepon pengguna sebagai sessionId agar konteks percakapan tidak hilang:

```
//Twilio webhook
app.post('/sms/', function (req, res) {
  //send it to the bot
  var sessionId = req.body.From;
  var userUtterance = req.body.Body;
  //API.AI
  var apiai = require('apiai');
  var apiai1 = apiai("YOUR_DEVELOPER_ACCESS_TOKEN");

  var requestAPIAI = apiai1.textRequest(userUtterance, {
    sessionId: sessionId
  });

  var botSays = '';
  requestAPIAI.on('response', function(response) {
    console.log(response);
    if (response.result.actionIncomplete){
      botSays = response.result.fulfillment.speech;
      console.log('BotSays: ' + botSays);
      var twiml = new twilio.TwimlResponse();
      twiml.message(botSays);
      res.writeHead(200, {'Content-Type': 'text/xml'});
      res.end(twiml.toString());
    }
    else {
      getTrainInfo(
        response.result.metadata.intentName,
        response.result.parameters, res);
    }
  });

  requestAPIAI.on('error', function(error) {
    console.log(error);
  });

  requestAPIAI.end();
})
```

Kami menggunakan tanda `actionIncomplete` dalam respons agen API.AI untuk memutuskan apakah akan menggunakan ucapan pemenuhan agen sebagai respons bot atau memanggil modul backend untuk mencari informasi kereta.

2. Buat modul baru untuk mendapatkan informasi kereta:

```

function getTrainInfo(intent, parameters, res){
  if (intent == 'request_live_departures'){
    return getLiveDepartures(parameters.Station, res);
  }
  else if (intent == 'request_next_train'){
    return getNextTrain(parameters.fromStation,
      parameters.toStation, res);
  }
  else {
    var botSays = 'Working on it...';
    console.log('BotSays: ' + botSays);

    var twiml = new twilio.TwimlResponse();
    twiml.message(botSays);
    res.writeHead(200, {'Content-Type': 'text/xml'});
    res.end(twiml.toString());
  }
}

```

3. Buat modul untuk mendapatkan kereta langsung. Kami menggunakan kembali kode dari `SendTrainNotification.js` yang kami gunakan untuk mengirim notifikasi satu arah:

```

function getLiveDepartures(source, res){
  var request = require('request');
  var url = 'http://transportapi.com/v3/uk/train/station/'
    + source +
    '/live.json?
    app_id=YOUR_APP_ID&app_key=YOUR_APP_KEY';
  request(url, function (error, response, body) {
    if (response){
      var json = JSON.parse(body);
      if (json.departures){
        var botSays = summarize(json.departures.all, 5);
        console.log('BotSays: ' + botSays);

        var twiml = new twilio.TwimlResponse();
        twiml.message(botSays);
        res.writeHead(200, {'Content-Type': 'text/xml'});
        res.end(twiml.toString());
      } else {
        var botSays = 'No Departures found!';
        console.log('BotSays: ' + botSays);

        var twiml = new twilio.TwimlResponse();
        twiml.message(botSays);
        res.writeHead(200, {'Content-Type': 'text/xml'});
        res.end(twiml.toString());
      }
    }
  } else {

```

```

        console.log('error:', error); // Print the error if one
                                     // occurred
        var botSays = 'Error in fetching trains info. Sorry!';
        console.log('BotSays: ' + botSays);

        var twiml = new twilio.TwimlResponse();
        twiml.message(botSays);
        res.writeHead(200, {'Content-Type': 'text/xml'});
        res.end(twiml.toString());
    }
});
}

function summarize(departures, n){
    var out = '';
    if (departures.length > 0){
        out = 'Live departures:\n';
        for (var i=0; i < n; i++){
            var service = departures[i];
            var serviceSummary = service.operator_name + ":" +
                service.destination_name + "@" +
                service.expected_departure_time;
            out += serviceSummary + "\n";
        }
    } else {
        out = 'There are no trains from ' + source;
    }
    return out;
}

```

Dan fungsi untuk mengantarkan kereta selanjutnya dari sumber ke tujuan:

```

function getNextTrain(source, destination, res){
    var request = require('request');
    var url = 'http://transportapi.com/v3/uk/train/station/'
        + source + '/live.json?
            app_id=YOUR_APP_ID&app_key=YOUR_APP_KEY';
    request(url, function (error, response, body) {
        if (response){
            var json = JSON.parse(body);
            if (json.departures){
                var botSays = getNextTrainToDestination(
                    destination,
                    json.departures.all);
                console.log('BotSays: ' + botSays);

                var twiml = new twilio.TwimlResponse();
                twiml.message(botSays);
                res.writeHead(200, {'Content-Type': 'text/xml'});
                res.end(twiml.toString());
            } else {

```

```

        var botSays = 'No Departures found!'
        console.log('BotSays: ' + botSays);

        var twiml = new twilio.TwimlResponse();
        twiml.message(botSays);
        res.writeHead(200, {'Content-Type': 'text/xml'});
        res.end(twiml.toString());
    }
} else {
    console.log('error:', error);
    // Print the error if one
    // occurred
    var botSays = 'Error in fetching trains info. '
        + 'Sorry!';
    console.log('BotSays: ' + botSays);

    var twiml = new twilio.TwimlResponse();
    twiml.message(botSays);
    res.writeHead(200, {'Content-Type': 'text/xml'});
    res.end(twiml.toString());
}
});
}

```

Dan fungsi untuk mengantarkan kereta berikutnya ke tujuan:

```

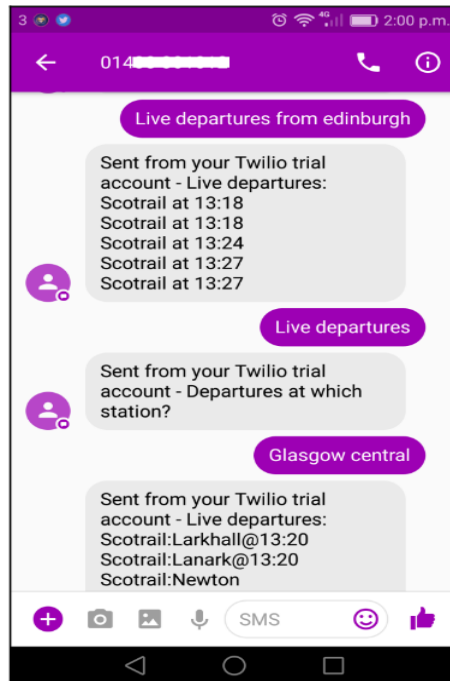
function getNextTrainToDestination(destination, allDepartures){
    if (allDepartures){
        for (var i=0; i < allDepartures.length; i++){
            var service = allDepartures[i];
            if (service.destination_name ==
                getStationName(destination)){
                var serviceSummary = service.operator_name + ":"
                    + service.destination_name + "@" +
                    service.expected_departure_time + "\n";
                return serviceSummary;
            }
        }
    }
    return null;
}

function getStationName(stationCode){
    if (stationCode == 'GLC'){
        return 'Glasgow Central';
    }
    else if (stationCode == 'EDB'){
        return 'Edinburgh';
    }
}

```

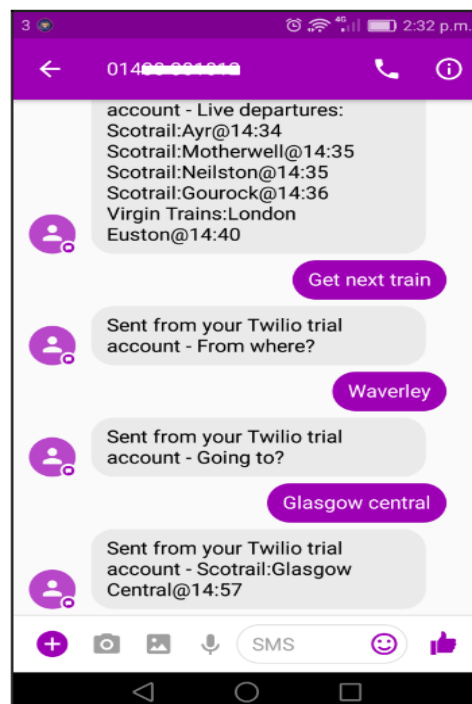
4. Simpan dan masukkan kode yang telah direvisi ke Heroku.

5. Uji dengan mengirimkan SMS. Coba tugas kereta langsung dengan mengatakan, Keberangkatan langsung. Bot akan menanyakan nama stasiun sebelum menampilkan informasi keberangkatan langsung:



Gambar 6.26 tampilan pesan chatbot

Uji tugas kereta berikutnya dengan mengatakan Dapatkan kereta berikutnya. Bot akan menanyakan stasiun sumber dan tujuan sebelum menampilkan hasilnya:



Gambar 6.27 tampilan pesan chatbot

Anda juga dapat mencoba meminta informasi yang sama dengan cara lain untuk menguji seberapa baik agen API.AI memahami maksud pengguna. Selamat! Anda sekarang telah membuat chatbot SMS dua arah yang memahami masukan bahasa alami.

BAB 7

MERANCANG DAN MENGIMPLEMENTASIKAN CHATBOT

Facebook Messenger adalah salah satu platform perpesanan paling populer yang tersedia secara global. Anda dapat mengaksesnya menggunakan akun Facebook Anda. Messenger tersedia secara terpisah untuk Facebook dalam bentuk situs web (<http://messenger.com> dan <http://m.me>) dan sebagai aplikasi ponsel pintar. Hal ini membuatnya sangat mudah diakses. Facebook memiliki lebih dari 1 miliar pengguna global dan semua pengguna media sosial ini juga memiliki akses ke Messenger; jumlah yang mengejutkan ini juga terus bertambah setiap hari. Hal ini menjadikannya pilihan populer bagi pengembang chatbot sebagai platform penerapan. Selain jumlah pengguna, perusahaan ini berinvestasi besar-besaran untuk menjadikannya tempat bagi chatbot untuk berkembang. Fitur terbarunya, seperti tab Temukan dan peningkatan SDK, menjadi bukti komitmen mereka.

Pada bab ini, kami akan merancang dan mengimplementasikan chatbot yang dapat berbicara tentang cuaca. Kami akan membangun chatbot dan mengintegrasikannya dengan layanan backend yang dapat memberikan kami informasi cuaca real-time. Kami kemudian akan melanjutkan untuk mengintegrasikan chatbot di Facebook Messenger. Dan terakhir, lihat fitur eksklusif Messenger yang dapat digunakan untuk membuat chatbot lebih menarik dan menarik.

Pada akhir bab ini, Anda akan dapat:

- ✚ Rancang tugas percakapan untuk membicarakan cuaca
- ✚ Buat integrasi backend menggunakan OpenWeatherMap API,
- ✚ Bangun chatbot di Java 8 dan Node.js dan host di cloud menggunakan Heroku
- ✚ Memahami SDK Facebook Messenger dan jenis templat pesan
- ✚ Integrasikan chatbot di Messenger

7.1 Tugas Percakapan

Pertama-tama mari kita cari tahu tugas yang kita ingin agar dilakukan oleh chatbot. Menurut saya, chatbot yang mampu berbicara tentang cuaca seharusnya dapat melakukannya dengan cara berikut:

- ❖ Cuaca sekarang
- ❖ Cuaca hari ini
- ❖ Cuaca minggu ini
- ❖ Cuaca akhir pekan ini
- ❖ Cuaca di masa depan

Semua laporan yang disebutkan didasarkan pada lokasi geografis. Informasi ini dapat diberikan dalam berbagai format. Kebanyakan orang akan memberikan nama kota/kota, atau kode pos. Namun mengingat fakta bahwa perangkat seluler memiliki kemampuan GPS,

dan aplikasi perpesanan memungkinkan berbagi lokasi, cara lain untuk menemukan lokasi pengguna dapat menggunakan koordinat LatLong (lintang dan bujur). Mari kita perhitungkan semua variasi ini. Selain tugas dan jenis masukan ini, kita perlu mempertimbangkan permintaan klarifikasi yang harus mampu dilakukan oleh chatbot jika informasi lokasi masukan bersifat ambigu.

7.2 Desain Percakapan

Sekarang kita telah mengidentifikasi tugas-tugas yang perlu dilakukan chatbot kita, mari kita lanjutkan dan merancang alur percakapan. Alur percakapan adalah tata letak dasar bagaimana interaksi akan terjadi antara pengguna dan bot.

Mari kita tuliskan beberapa percakapan sederhana antara pengguna dan bot cuaca kita.

Contoh 1:

Pengguna : Hai

Bot : Hai di sana! Saya peramal cuaca, peramal cuaca Anda, Bot cuaca Anda. Apa yang ingin Anda ketahui? Cuaca saat ini atau ramalan cuaca

Pengguna : Cuaca saat ini

Bot : ok, kota mana?

Pengguna : London, Inggris

Bot : ok cuaca sekarang di London, Inggris. Suhu 10 derajat Celcius langit cerah

Pengguna : terima kasih

Bot : tidak masalah ! :)

Mari kita buat satu lagi yang sedikit lebih rumit.

Contoh 2:

Pengguna : hai, ada apa?

Bot : halo! Saya wheaterMan, bot cuaca Anda. apa yang ingin Anda ketahui? cuaca saat ini atau ramalan cuaca?

Pengguna : ramalan cuaca

Bot : ok. Kapan? Besok atau akhir pekan ini?

Pengguna : besok

Bot : ok kota mana?

Pengguna : Edinburgh

Bot : Edinburgh, AS atau Edinburgh, Inggris?

Pengguna : Edinburgh, Inggris

Bot : ok. Cuaca besok di Edinburgh, Inggris. Suhu 12 derajat celcius. Berawan

Dalam ranah pembicaraan cuaca, percakapan yang lebih kompleks dapat dilakukan. Namun, kami akan menggunakan contoh sebelumnya untuk menunjukkan bagaimana Anda dapat membuat chatbot dan menerapkannya di Facebook Messenger pada prinsipnya. Mari kita bahas ucapan dan percakapan yang lebih kompleks nanti.

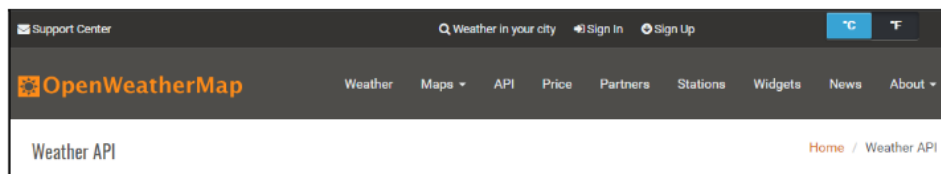
7.3 Tugas Latar Belakang

Sebelum kita melanjutkan implementasi chatbot, mari kita lihat sumber informasi data cuaca, OpenWeatherMap (<https://openweathermap.org>). OpenWeatherMap adalah layanan cloud yang menyajikan informasi cuaca tentang 200.000+ kota di seluruh dunia. Ini memiliki data terkini dan perkiraan serta data historis. Ini memiliki layanan gratis dan berbayar. Navigasikan ke situs web, ketik nama kota Anda, dan cobalah.

1. Mendapatkan data cuaca. Sekarang mari kita membuat akun di layanan OpenWeatherMap untuk mendapatkan data cuaca.
2. Mendapatkan kunci API

Berbeda dengan bab sebelumnya, di mana kita membuat konten untuk chatbot secara manual, kita akan menggunakan data ini langsung dari titik akhir layanan. Untuk melakukan ini, kita perlu mendaftar. Untuk mendaftar, navigasikan ke <https://home.openweathermap.org/> dan buat akun baru.

Untuk memulai dengan data, kita harus berlangganan layanan API yang sesuai. Untuk melakukan ini, navigasikan situs web dengan mengklik opsi API pada menu utama di halaman beranda:



Gambar 9.1 tampilan web

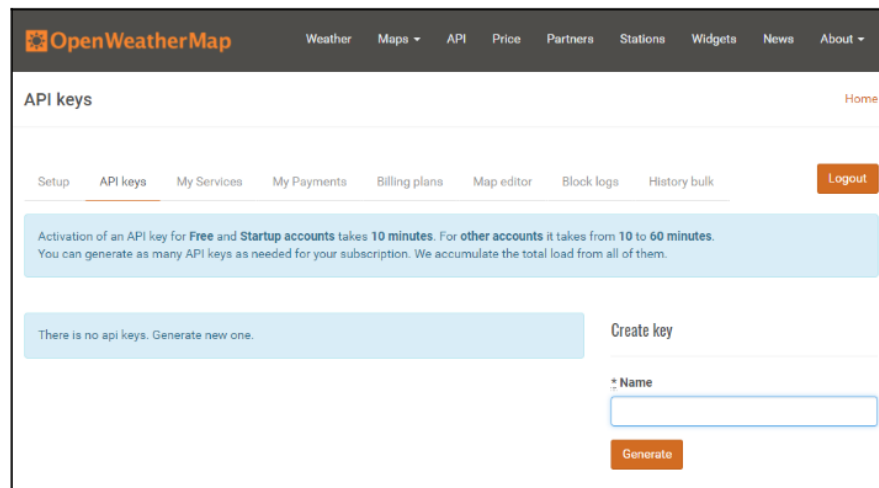
Di halaman API, Anda akan melihat daftar semua layanan API yang tersedia. Mari kita mulai dengan hal dasar: Data cuaca terkini. Klik Berlangganan:

	Free	Startup	Developer	Professional	Enterprise
Price	Free	40 USD / month	180 USD / month	470 USD / month	2,000 USD / month
<small>Price is fixed, no other hidden costs.</small>					
Subscribe	Get API key and Start	Subscribe	Subscribe	Subscribe	Subscribe
<small>Calls per minute (no more than)</small>	60	600	3,000	30,000	200,000
Current weather API	✓	✓	✓	✓	✓
5 days/3 hour forecast API	✓	✓	✓	✓	✓
16 days/daily forecast API	-	-	✓	✓	✓
Weather maps API	✓	✓	✓	✓	✓

Gambar 9.2 Tampilan langganan API

Klik Dapatkan kunci API dan Mulai. Ini akan membawa Anda ke halaman appid (<https://openweathermap.org/appid>) di mana Anda dapat menemukan dokumentasi tentang API (yaitu, cara menggunakan kunci dan batasannya).

Untuk mendapatkan kunci Anda, Anda perlu melangkah lebih jauh. Navigasikan ke halaman kunci API: https://home.Openweathermap.org/api_keys .Anda akan menemukan kunci default. Jika tidak, buatlah:



Gambar 9.3 kunci default api

Sekarang kita sudah punya kuncinya, ayo kita coba!

Mencoba kunci Anda

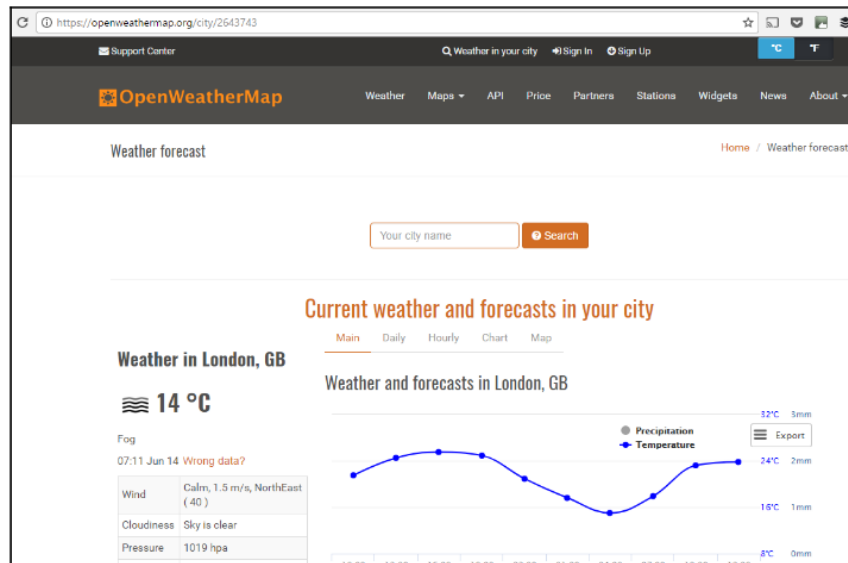
Kita harus membuat URL untuk mendapatkan data yang kita butuhkan. Ini yang mendasar:

http://api.openweathermap.org/data/2.5/forecast?id=<CITY_CODE>&APPID=<YOUR_API_KEY>

Untuk mendapatkan kode kota Anda, navigasikan ke URL berikut:

<https://openweathermap.org/find?q=>

Ketik nama kota Anda dan klik Cari. Di halaman hasil, ambil kode kota Anda dari URL:



Gambar 9.4 tampilan suhu pada suatu daerah

Misalnya kode kota London, GB adalah 2643743. Ganti <YOUR_API_KEY> dengan kunci Anda dan coba URL-nya di browser web. Anda akan mendapatkan objek JSON dengan informasi cuaca saat ini dan ramalan cuaca, mirip dengan berikut ini:

```

{
  "cod": "200",
  "message": 0.003,
  "cnt": 37,
  "list": [ ... ], // 37 items
  "city": {
    "id": 2643743,
    "name": "London",
    "coord": {
      "lat": 51.5085,
      "lon": -0.1258
    },
    "country": "GB"
  }
}

```

Kunci daftar dengan 37 item dapat diperluas, di mana Anda akan menemukan informasi terkini dan perkiraan untuk setiap tiga jam mulai dari waktu saat ini:

```

▼ "list": [
  ▼ {
    "dt": 1497430800,
    ▶ "main": { ... }, // 8 items
    ▼ "weather": [
      ▼ {
        "id": 800,
        "main": "Clear",
        "description": "clear sky",
        "icon": "01d"
      }
    ],
    ▶ "clouds": { ... }, // 1 item
    ▶ "wind": { ... }, // 2 items
    ▶ "sys": { ... }, // 1 item
    "dt_txt": "2017-06-14 09:00:00"
  },
  ▶ { ... }, // 7 items
  ▶ { ... }, // 7 items

```

Cemerlang! Kami sekarang memiliki sumber data untuk dihubungkan ke chatbot kami.

7.4 Membangun Antarmuka Backend

Sekarang mari kita coba beberapa kode! Kami sekarang akan membangun modul di Java yang dapat mengakses data cuaca dari sumber data kami secara real time. Kami kemudian akan menyambungkan modul ini ke chatbot untuk digunakan oleh pengelola percakapan:

- 1) Buka Gerhana.
- 2) Buat proyek Maven baru.
- 3) Pilih Buat proyek sederhana.
- 4) Berikan lokasi proyek dan tekan Berikutnya.
- 5) Pada halaman berikutnya, ketik Group Id, Artifact Id, Name, dan Description. Klik Selesai.

Kami sekarang telah menyiapkan proyek Maven kosong. Sebelum kita melanjutkan ke pengembangan kode backend, mari tambahkan beberapa paket ketergantungan ke proyek kita. Temukan file POM (pom.xml) dan tambahkan dependensi Maven berikut:

```

<dependencies>
  <!--
  https://mvnrepository.com/artifact/com.google.code.gson/gson -->
  <dependency>
    <groupId>com.google.code.gson</groupId>
    <artifactId>gson</artifactId>
    <version>2.8.0</version>
  </dependency>
  <!--

```

<https://mvnrepository.com/artifact/org.apache.httpcomponents/httpclient>-->

```

<dependency>
  <groupId>org.apache.httpcomponents</groupId>
  <artifactId>httpclient</artifactId>
  <version>4.5.3</version>
</dependency>
</dependencies>

```

Kami mungkin memerlukan dependensi lain nanti. Namun untuk kode backend, paket-paket ini sudah cukup. Sekarang mari kita buat kelas Java, Weather.java, untuk mengakses data cuaca. Kode berikut menunjukkan struktur dasar kelas:

```

Package weatherman.weather ;
Import com.google.gson.JsonObject ;
Public class weather {
  Public static void main (string[ ] args) { }
  Public weather ( ) { }
Public JsonObject getWeather(String citycode) {return null ; }
Public JsonObject getCurrentWeather (String cityName ) {return null
; } }

```

Kami sekarang akan menambahkan kode yang diperlukan untuk mendapatkan data aktual dari layanan OpenWeatherMaps:

```

Package weatherman.weather ;
Import java,io,IOException ;

Import org.apache.http.HttpResponse ;
Import org.apache.http.client.ClientProtocolException ;
Import org.apache.http.client.HttpClient ;
Import org.apache.http.client.methods.HttpGet ;
Import org.apache.http.impl.client.HttpClientBuilder ;
Import org.apache.http.util.EntityUtils ;

Import com.google.gson.JsonObject ;
Import com.google.gson.JsonParser ;
Public class weather {
//put your api key here
String apiKey = <Your API key> ;
Public static void main (String [ ] args) {
  Try {
    (new Weather ( ) ) .getWeather ("2643743") ;
  } catch (ClientProtocolException e) {
    e.printStackTrace ( ) ;
  } catch (IOException e) {
    e.printStackTrace ( ) ;
  }
}
Public Weather ( ) { }
Public JsinObject getWeather ( String CityCode)
  Throws ClientProtocolException, IOException {

```

```

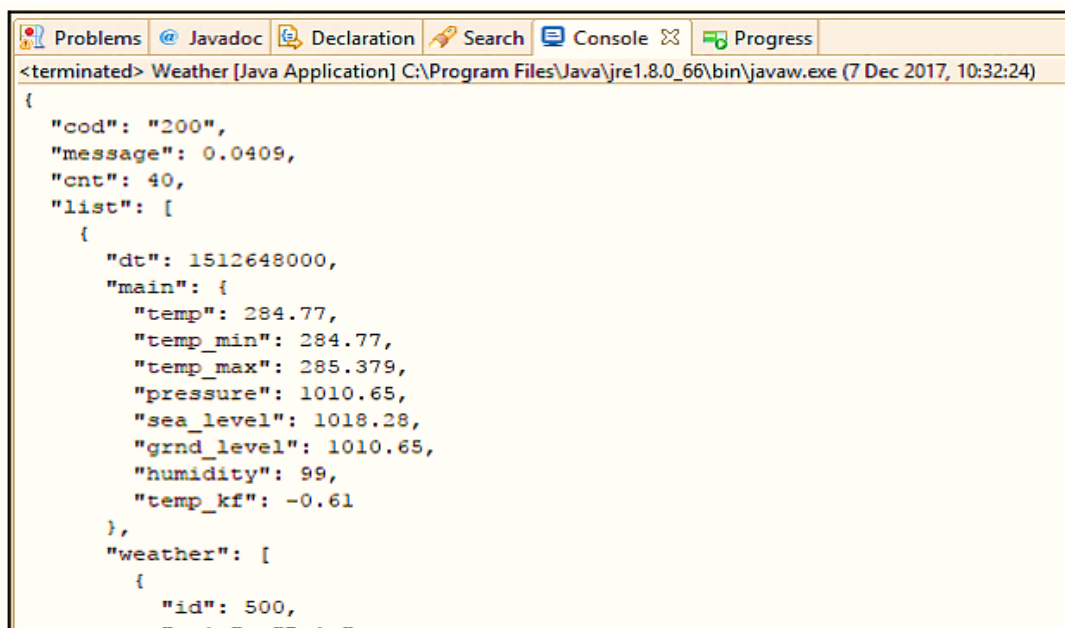
//step1 ; prepare the url
String url = "http ;
//api.openweathermap.org/data/2.5/forecast?="
    + citycode + "&APPID=" + apiKey ;

//step 2 : Create a HTTP client
HttpClient httpClient = HttpClient Builder.create ( ).build( ) ;
//step 3 : Create a HTTPget object and excute the url
HttpGet httpGet = new HttpGet (url) ;
HttpResponse response = httpClient.execute (httpGet ) :

//step 4 : Process the result
JsonObject json = null ;
Int statusCode = response.getStatusLine ( ). getStatusCode ( ) ;
if (statusCode == 200) {
    String response_string = EntityUtils.toString
(response.getEntity ( ) ) ;
    Json= (new JsonParser ( ) ) .parse (response_string)
.getAsJsonObject ( ) ;
    Gson gson = new GsonBuilder ( ) .setPrettyPrinting ( ) .Create
( ) ;
    System.out.println (prettyJson) ;
}
Return json ;
}
}

```

Dalam metode utama, kami telah membuat instance objek Weather untuk mendapatkan cuaca terkini untuk London, GB (2643743). Jalankan kode sebelumnya dan Anda akan mendapatkan hasil berikut di konsol:



```

<terminated> Weather [Java Application] C:\Program Files\Java\jre1.8.0_66\bin\javaw.exe (7 Dec 2017, 10:32:24)
{
  "cod": "200",
  "message": 0.0409,
  "cnt": 40,
  "list": [
    {
      "dt": 1512648000,
      "main": {
        "temp": 284.77,
        "temp_min": 284.77,
        "temp_max": 285.379,
        "pressure": 1010.65,
        "sea_level": 1018.28,
        "grnd_level": 1010.65,
        "humidity": 99,
        "temp_kf": -0.61
      },
      "weather": [
        {
          "id": 500,

```

Hasilnya berupa objek JSON yang bisa kita gunakan untuk merespon pertanyaan cuaca. Dalam daftar item, ada 36 item. Masing-masing item tersebut menyediakan data cuaca, seperti suhu rata-rata (temp), suhu minimum (temp_min), dan suhu maksimum (temp_max) pada waktu tertentu (dt adalah stempel waktu). Sekarang mari kita terapkan dua metode, `getWeatherAtTime()` dan `getWeatherReport()`, untuk menghasilkan laporan cuaca singkat:

```

Public String getWeatherReport (String cityCode,Integer i)
    Throws ClientProtocolException, IOException {

    JSONObject currentWeather = null ;
    If (cityCode !=null) {
        currentWeather = getWeatherAtTime (cityCode, 1) ;
    }
    String weatherReport = null ;
    If ( currentWeather != null) {
        JSONObject weather = currentWeather.get("weather")
            .getAsJSONArray ( ) .get(0).getAsJsonObject ( ) )
        Double avgTemp = Double.valueOf(currentWeather.get("main")
            .getAsJsonObject ( ) .get ("temp").getAsString ( ) ) - 273.15
;
        String avgTempSt = String.valueOf(avgTemp).split (\\.) [0] ;
        weatherReport = " The temperature is " + avgTempSt +
            " degrees Celsius."
            + weather.get ("description"). getAsString( ) +
        ". ";
    }
    System.out.println(weatherReport) ;
    Return weatherReport;
}

Public JSONObject getWeatherAtTime (String cityCode, integer i)
    Throws CloentProtocolException,IOException{
    sonObject json = getWeather (cityCode) ;
    JSONArray list = json.get("list").getAsJSONArray ( ) ;
    JSONObject weatherAtTime = list.get(i).getAsJsonObject( ) ;
    returnweatherAtTime ;
}

```

Anda mungkin memperhatikan bahwa `getWeatherReport()` memerlukan dua parameter kode kota dan waktu. Waktu dilewatkan sebagai bilangan bulat dengan 0 sebagai waktu saat ini, 1 sebagai slot berikutnya, dan seterusnya. Memanggil metode `getWeatherReport()` dari main dengan nama kota dan waktu sebagai parameter akan menghasilkan laporan cuaca tekstual singkat seperti berikut:

Suhu 29,8 Derajat. Langit Cerah

Tentu saja, mengingat kekayaan sumber datanya, kami dapat menambah jumlah tugas backend.

7.5 Menerapkan Chatbot

Sekarang kita sudah menyiapkan tugas backend, mari fokus pada chatbot itu sendiri. Secara umum, chatbot akan mengambil ucapan pengguna sebagai masukan dan merespons dengan ucapannya sendiri.

Namun, karena kami membuat chatbot untuk Facebook Messenger, chatbot kami sebagian besar akan menerima masukan dalam bentuk penekanan tombol dan merespons menggunakan ucapan dan kartu yang menarik secara visual.

Mari kita mulai dengan mengimplementasikan kelas Chatbot.java. Kami akan mulai dengan menyusun algoritma untuk memproses dan merespons ucapan pengguna:

- 1) Memproses masukan pengguna.
- 2) Perbarui konteks.
- 3) Identifikasi maksud bot.
- 4) Menghasilkan ucapan bot dan struktur keluaran.
- 5) Merespon.

Ini adalah algoritma yang sangat sederhana untuk memulai. Pertama, input pengguna, berupa ucapan atau penekanan tombol, diproses. Kemudian konteks pembicaraan diperbarui. Pada langkah berikutnya, kami mengidentifikasi apa yang ingin disampaikan oleh bot. Setelah hal itu ditentukan, kami mencari cara untuk mengatakannya dan meresponsnya. Mari kita mulai dengan menerapkan struktur dasar berdasarkan algoritma yang disebutkan:

```
Package weatherman.chaatbot ;
Import com.google.gson.Json.Object;
Public class chatbot {
    JsonObject context ;
    Public static void main (string [ ] args) { }
    Public Chatbot ( ) {
        Context = new JsonObject ( ) ;
    }
    Public JsonObject process (JsobObject userInput) {
        // step1 : process user input
        JsonObject UserAction = ProcessUserInput (userInput) ;
        //step2 : update context
        updateContext (userAction) ;
        //step3 : identify bot intent
        identifyBotIntent ( ) ;
        //step4 : structure output
        JsonObject out = getBotOutput ( ) ;
        Return out ;
    }
}
```

Sekarang kita akan memodifikasi metode main() untuk menyimulasikan jendela obrolan di mana pengguna dapat mengetikkan permintaan dan tanggapan mereka dan melakukan obrolan dengan bot:

```

Public static void main (string [ ] args ) {
    Chatbot c = new Chatbot ( ) ;
    Scanner scanner = new scanner (system.in) ;
    String userUtterance ;
    Do {
        System.out.print ("user;") ;
        userUtterance = scanner.nextLine ( ) ;
        //end the conversation
        If (userUtterance.equals ("QUIT") ) {break ; }

        JsonObject userInput = new JsonObject ( ) ;
        userInput.add ("userU'tterance", new JsonPrimitive
(userUtterance));
        JsonObject botOutput = c.process (userInput) ;
        String botUtterance = " ";
        If (botOutput !=null && botOutput.has("botUtterance") ) {
            botUtterance = botOutput.get("botUtterance").getAsString (
) ;
        }
        System.out.println("Bot:" + botUtterance) ;
    } while (true) ;
}

```

Sekarang, mari kita fokus pada chatbot itu sendiri. Pertama-tama kami akan membangun modul untuk memahami ucapan pengguna. Kita akan membangun modul yang sangat sederhana menggunakan aturan dan ekspresi reguler untuk menerjemahkan ucapan pengguna menjadi maksud pengguna.

Niat adalah representasi formal yang tidak ambigu tentang apa yang dikatakan pengguna atau bot. Ini menyampaikan makna di balik ucapan atau isyarat.

Pertama-tama mari kita cari tahu daftar awal maksud pengguna untuk tugas yang kita miliki berdasarkan contoh percakapan yang telah kita buat:

- ❖ Salam
- ❖ Request_cuaca_sekarang
- ❖ Informasikan_kota
- ❖ terima kasih

Maksud salam mewakili banyak salam berbeda yang mungkin digunakan pengguna. Oleh karena itu, kami akan menerjemahkan sejumlah ujaran seperti hai, hai, halo, dan halo ke dalam maksud menyapa. Demikian pula, maksud terima kasih mewakili semua cara pengguna berterima kasih kepada bot. Maksud request_current_weather digunakan untuk mewakili ucapan saat pengguna meminta info cuaca terkini dan maksud inform_city adalah saat pengguna menyebutkan nama kota:

```

public JsonObject processUserInput(JsonObject userInput){
    String userUtterance = null;
    JsonObject userAction = new JsonObject();

    //default case
    userAction.add("userIntent", new JsonPrimitive(""));
    if (userInput.has("userUtterance")){
        userUtterance = userInput.get("userUtterance").getAsString();
        userUtterance = userUtterance.replaceAll("%2C", ",");
    }
    if (userUtterance.matches("(hi|hello) (there)?")){
        userAction.add("userIntent", new JsonPrimitive("greet"));
    }
    else if (userUtterance.matches("(thanks)|(thank you)")){
        userAction.add("userIntent", new JsonPrimitive("thank"));
    }
    else if (userUtterance.matches("current weather")||
            userUtterance.matches("weather now")){
        userAction.add("userIntent", new
            JsonPrimitive("request_current_weather"));
    }
    else {
        //contextual processing
        String currentTask = context.get("currentTask").getAsString();
        String botIntent = context.get("botIntent").getAsString();
        if (currentTask.equals("requestWeather") &&

                botIntent.equals("requestPlace")){
            userAction.add("userIntent", new
                JsonPrimitive("inform_city"));
            userAction.add("cityName", new JsonPrimitive(userUtterance));
        }
    }
    return userAction;
}

```

Pada kode sebelumnya, kita mendapatkan maksud pengguna dari ucapannya. Beberapa maksud juga memiliki parameter terkait. Misalnya, maksud `inform_city` memiliki parameter `cityName` terkait yang mewakili nama kota tempat pengguna mencari laporan cuaca. Menariknya, maksud `inform_city` juga merupakan maksud yang kita peroleh dari konteks. Sebab, saat dimintai informasi kota, pengguna hanya menyebutkan nama kotanya saja. Jadi apa pun yang diketik akan dilaporkan sebagai nama kota dan perlu divalidasi nanti. Intent dan parameter terkaitnya dikemas sebagai objek JSON `userAction`. Mari kita lanjutkan memperbaiki konteksnya:

```

public void updateContext(JsonObject userAction){
    //copy userIntent
    context.add("userIntent", userAction.get("userIntent"));
    //
    String userIntent = context.get("userIntent").getAsString();
    if (userIntent.equals("greet")){
        context.add("currentTask", new JsonPrimitive("greetUser"));
    } else if (userIntent.equals("request_current_weather")){
        context.add("currentTask", new JsonPrimitive("requestWeather"));
        context.add("timeOfWeather", new JsonPrimitive("current"));
        context.add("placeOfWeather", new JsonPrimitive("unknown"));
        context.add("placeName", new JsonPrimitive("unknown"));
    } else if (userIntent.equals("inform_city")){
        String cityName = userAction.get("cityName").getAsString();
        JsonObject cityInfo = weather.getCityCode(cityName);
        if (!cityInfo.get("cityCode").isJsonNull()){
            context.add("placeOfWeather", cityInfo.get("cityCode"));
            context.add("placeName", cityInfo.get("cityName"));
        }
    } else if (userIntent.equals("thank")){
        context.add("currentTask", new JsonPrimitive("thankUser"));
    }
}
}

```

Pada kode sebelumnya, kami memperbarui konteks percakapan menggunakan masukan dari pengguna. Di sini, masukan pengguna diterjemahkan menjadi tugas dan parameter. Ide di balik konteks sama dengan bagaimana manusia menjaga konteks pembicaraan saat berbicara dengan seseorang.

Variabel `currentTask` mewakili tugas chatbot saat ini. Maksud seperti salam, terima kasih, dan `request_current_weather` akan mengatur variabel ini. Setiap tugas akan memiliki sejumlah parameter yang perlu diisi. Misalnya, tugas `requestWeather` memiliki dua slot: waktu dan tempat. Mereka harus diisi sebelum bot bisa mendapatkan laporan cuaca. Jika pengguna menanyakan cuaca saat ini, waktu sudah diatur ke saat ini, namun tempatnya masih belum diketahui. Langkah selanjutnya: identifikasi maksud bot!

Setelah memperbarui konteksnya, maksud chatbot perlu ditentukan. Untuk beberapa tugas, seperti menyapa dan berterima kasih kepada pengguna, maksud bot cukup mudah. Untuk tugas kompleks lainnya, maksud ditentukan berdasarkan slot yang perlu diisi dan respons dari tugas backend. Kami memiliki maksud berikut untuk bot:

- ✚ Menyapa Pengguna
- ✚ Terima kasih Pengguna
- ✚ Minta Tempat
- ✚ Menginformasikan Cuaca

`GreetUser` dan `ThankUser` digunakan untuk menyapa dan berterima kasih kepada pengguna. `requestPlace` digunakan ketika bot perlu mengetahui tempat laporan cuaca. `informWeather` digunakan ketika bot berhasil mengambil laporan dari layanan backend yang telah kita buat di bagian sebelumnya:

```

public void identifyBotIntent(){
    String currentTask = context.get("currentTask").getAsString();
    if (currentTask.equals("greetUser")){
        context.add("botIntent", new JsonPrimitive("greetUser"));
    } else if (currentTask.equals("thankUser")){
        context.add("botIntent", new JsonPrimitive("thankUser"));
    } else if (currentTask.equals("requestWeather")){
        if
            (context.get("placeOfWeather").getAsString().equals("unknown")){
            context.add("botIntent", new JsonPrimitive("requestPlace"));
        }
        else {
            Integer time = -1;
            if
                (context.get("timeOfWeather").getAsString().equals("current")){
                time = 0;
            }
            String weatherReport = null;

            try {
                weatherReport = weather.getWeatherReport(
                    context.get("placeOfWeather").getAsString(), time);
            }
            catch (ClientProtocolException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
            if (weatherReport != null){
                context.add("weatherReport", new
                    JsonPrimitive(weatherReport));
                context.add("botIntent", new JsonPrimitive("informWeather"));
            }
        }
    } else {
        context.add("botIntent", null);
    }
}
}

```

Pada kode sebelumnya, perhatikan bagaimana bot memutuskan apa yang harus dilakukan ketika CurrentTask adalah requestWeather. Ia memutuskan untuk menanyakan tempatnya ketika tidak diketahui. Dan ketika waktu dan tempatnya diketahui, laporan akan diambil dan konteksnya diperbarui. Selanjutnya, maksud bot perlu diterjemahkan ke dalam ucapan:

```

public JSONObject getBotOutput(){
    JSONObject out = new JSONObject();
    String botIntent = context.get("botIntent").getAsString();
    String botUtterance = "";

    if (botIntent.equals("greetUser")){
        botUtterance = "Hi there! I am WeatherMan, your weather bot! "
            + "What would you like to know? Current weather or forecast?";
    } else if (botIntent.equals("thankUser")){
        botUtterance = "Thanks for talking to me! Have a great day!!";
    } else if (botIntent.equals("requestPlace")){
        botUtterance = "Ok. Which city?";
    } else if (botIntent.equals("informWeather")){
        String timeDescription =
            getTimeDescription(context.get("timeOfWeather").getAsString());
        String placeDescription = getPlaceDescription();
        String weatherReport = context.get("weatherReport").getAsString();
        botUtterance = "Ok. Weather " + timeDescription + " in " +
            placeDescription + ". " + weatherReport;
    }
    out.add("botIntent", context.get("botIntent"));
    out.add("botUtterance", new JsonPrimitive(botUtterance));

    return out;
}

private String getPlaceDescription() {
    return context.get("placeName").getAsString();
}

private String getTimeDescription(String timeOfWeather) {
    if (timeOfWeather.equals("current")){
        return "now";
    }
    return null;
}
}

```

Sekarang setelah maksud dan ucapan bot teridentifikasi, mari kita jalankan metode utama dan bersenang-senang! Jalankan kelas Chatbot dengan menekan Ctrl + F11. Di konsol, Anda akan diminta untuk memulai percakapan. Perhatikan contoh percakapan berikut:

```

User:hi there
Bot:Hi there! I am WeatherMan, your weather bot! What would you like to
know? Current weather or forecast?
User:current weather
Bot:Ok. Which city?
User:London,GB
Bot:Ok. Weather now in London,GB. The temperature is 291.72 degrees
Fahrenheit. broken clouds.
User:thanks
Bot:Thanks for talking to me! Have a great day!!
User:QUIT

```

Seperti yang mungkin Anda ketahui, ada banyak tugas yang belum selesai di sini. Misalnya, nama kota harus diketik dalam format yang sama (<nama kota,kode negara>) seperti yang ada di database cuaca kami. Ini tidak ideal. Pengguna harus bisa berkata dalam bahasa London dan tetap mendapatkan jawaban. Kita akan membahas hal ini lagi di bab berikutnya, di mana kita akan melihat cara menangani ucapan yang penggunanya dapat memanggil entitas yang sama dengan nama berbeda. Tapi pertama-tama, mari kita buat kode ini berfungsi di Facebook Messenger.

7.6 Membuat Layanan Web Chatbot

Langkah kami selanjutnya adalah membuat chatbot tersedia sebagai layanan web. Hal ini agar platform, seperti Facebook Messenger, dapat mengakses chatbot tanpa harus menghostingnya di server mereka sendiri. Agar chatbot tersedia sebagai layanan web, kode chatbot perlu dikemas sebagai server web dan dihosting di platform cloud. Kami akan menggunakan perpustakaan Java Spark untuk menggabungkan kode chatbot sebagai server web dan platform cloud Heroku untuk menampungnya:

1. Kita perlu menambahkan ketergantungan Spark ke file POM:

```
<!-- - https://mvnrepository.com/artifact/com.sparkjava/spark-core -->
<dependency>
  <groupId>com.sparkjava</groupId>
  <artifactId>spark-core</artifactId>
  <version>2.0.0</version>
</dependency>
```

2. Implementasikan kelas Java bernama `WebServer.java`. Pastikan untuk menempatkannya dalam paket default:

```

public class WebServer {
    public static void main(String[] args) {
        Spark.setPort(getHerokuAssignedPort());
        Spark.staticFileLocation("/public");
        final Chatbot bot = new Chatbot();

        get("/", (req, res) -> "Hello World! I am WeatherMan,
the weather bot!!");

        //post handle for WeatherMan chatbot
        post("/bot", new Route() {
            public Object handle(Request request, Response response) {
                String body = request.body();
                System.out.println("body: " + body);
                String splitChar = "&";
                String keyValueSplitter = "=";
                String[] params = body.split(splitChar);
                String userUtterance = "null";
                for (int i=0; i < params.length; i++){
                    String[] sv = params[i].split(keyValueSplitter);
                    if (sv[0].equals("userUtterance")){
                        if (sv.length > 0){
                            userUtterance = sv[1];
                        } else {

                            userUtterance = "";
                        }
                    }
                    userUtterance = userUtterance.replaceAll("%20",
                    "");
                    userUtterance = userUtterance.replaceAll("%3A",
                    ":");
                }
            }
        })
        if (!userUtterance.equals("null")){
            System.out.println("User says:" + userUtterance);
            JsonObject userInput = new JsonObject();
            userInput.add("userUtterance", new
            JsonPrimitive(userUtterance));
            String botResponse = bot.processFB(userInput);
            System.out.println("Bot says:" + botResponse);
            if (botResponse != null) {
                return botResponse;
            }
        }
    }
}

```

```

        }
    } else {
        return null;
    }
    response.status(400);
    return new ResponseError("Error! POST not handled.");
}
}, json());

after((req, res) -> { res.type("application/json"); });

exception(IllegalArgumentException.class, (e, req, res) -> {
    res.status(400);
    res.body(toJson(new ResponseError(e)));
});
}

static int getHerokuAssignedPort() {
    ProcessBuilder processBuilder = new ProcessBuilder();
    if (processBuilder.environment().get("PORT") != null) {
        return
Integer.parseInt(processBuilder.environment().get("PORT"));
    }
    return 4567;
    //return default port if heroku-port isn't set (i.e. on
    //localhost)
}
}
}

```

3. Mengimplementasikan kelas pembantu server web. `WebServer.java` mengimpor kelas dari paket `Spark` dan `Gson`. Itu juga memerlukan dua kelas pembantu, `JSONUtil.java` dan `ResponseError.java`. Kelas pembantu `JSONUtil` membantu `WebServer` mengubah format respons menjadi JSON:

```

Package weatherman.web.utils ;
Import com.google.gson.Gson ;
Import spark.ResponseTransformer ;

Public class JSONUtil {
    Public static string toJson (Object object) {
        Return new Gson ( ). toJson (object) ;
    }
    Public static ResoinseTransformer json ( ) {
        Return JSONUtil :: toJson ;
    }
}
}

```

Kelas `ResponseError` membantu kelas `WebServer` untuk melaporkan kesalahan dalam format tertentu yang mungkin terjadi selama layanan web:

```

Package weatherman.web.utils ;
Public class ResponseError {
    Private String errorType ;
Public ResponseError(string message, String ...args) {
    This.errorType = String.format (message, args) ;
}
Public ResponseError (Exception e) {
    This.errorType = @.getMessage( ) ;
}
Public String getMessage ( ) {
    Return this.errorType ;
}
}

```

- Ubah kompiler Java menjadi 1.8. File pom.xml memberi tahu Heroku kompiler mana yang akan digunakan saat membangun proyek di cloud. Karena kita menggunakan ekspresi Lambda, kita perlu mengatur compiler ke Java 1.8. Untuk melakukannya, tambahkan kode berikut (elemen build) di file pom.xml tepat setelah elemen dependensi:

```

<build>
  <plugins>
    <plugin>

      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.1</version>
      <configuration>
        <source>1.8</source>
        <target>1.8</target>
      </configuration>
      <executions>
        <execution>
          <id>default-compile</id>
          <phase>compile</phase>
          <goals>
            <goal>compile</goal>
          </goals>
        </execution>
        <execution>
          <id>default-testCompile</id>
          <phase>test-compile</phase>
          <goals>
            <goal>testCompile</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>

```

Simpan file pom.xml. Klik kanan nama proyek, pilih Maven, lalu Perbarui Proyek di menu. Klik OK di kotak dialog. Ini akan memperbarui pengaturan Maven.

5. Tambahkan Profil. Procfile adalah artefak Heroku. Dalam file ini, kami memberi tahu Heroku program mana yang akan dijalankan setelah membangun proyek. Profil Implementasi:

```
Web : java-cp target/classes : target / dependency/*WebServer
```

Baris sebelumnya menginformasikan Heroku untuk menjalankan kelas WebServer dalam paket default di web dyno.

6. Dorong ke cloud. Sekarang kita telah membuat chatbot versi pertama dan membungkusnya sebagai server web, mari kita dorong ke cloud dan membuatnya tersedia sebagai layanan web. Dari jendela konsol, tempat proyek Weatherman Anda berada, jalankan perintah berikut:

```
<git init
>git add
>git commit -m "chatbot for weather v1"
```

Dengan menjalankan perintah sebelumnya, kami membuat dan menyimpan file proyek di repositori GIT lokal. Sekarang mari kita membuat aplikasi Heroku untuk menampung aplikasi web kita:

```
>Heroku create weatherman-bot
```

Anda akan melihat hasil dari perintah ini seperti pada gambar layar berikut. Dikatakan bahwa kami telah membuat aplikasi Heroku bernama bot cuaca dan URL-nya adalah

<https://weatherman-bot.herokuapp.com>:

```

C:\WINDOWS\system32\cmd.exe
C:\Users\Srini\Documents\_Book\workspace>cd weatherman

C:\Users\Srini\Documents\_Book\workspace\weatherman>git init
Initialized empty Git repository in C:/Users/Srini/Documents/_Book/workspace/weatherman/.git/

C:\Users\Srini\Documents\_Book\workspace\weatherman>git add .

C:\Users\Srini\Documents\_Book\workspace\weatherman>git commit -m "chatbot for weather version 1"
[master (root commit) 4c74011] chatbot for weather version 1
 20 files changed, 593 insertions(+)
 create mode 100644 .classpath
 create mode 100644 .project
 create mode 100644 .settings/org.eclipse.jdt.core.preferences
 create mode 100644 .settings/org.eclipse.m2e.core.preferences
 create mode 100644 ProcFile
 create mode 100644 pom.xml
 create mode 100644 src/main/java/weatherman/chatbot/Chatbot.java
 create mode 100644 src/main/java/weatherman/weather/Weather.java
 create mode 100644 src/main/java/weatherman/web/WebServer.java
 create mode 100644 src/main/java/weatherman/web/Utils/JSONUtil.java
 create mode 100644 src/main/java/weatherman/web/Utils/ResponseError.java
 create mode 100644 target/classes/META-INF/MANIFEST.MF
 create mode 100644 target/classes/META-INF/maven/com.sjanarthanam/weatherman/pom.properties
 create mode 100644 target/classes/META-INF/maven/com.sjanarthanam/weatherman/pom.xml
 create mode 100644 target/classes/weatherman/chatbot/Chatbot.class
 create mode 100644 target/classes/weatherman/weather/Weather.class
 create mode 100644 target/classes/weatherman/web/WebServer$.class
 create mode 100644 target/classes/weatherman/web/WebServer.class
 create mode 100644 target/classes/weatherman/web/Utils/JSONUtil.class
 create mode 100644 target/classes/weatherman/web/Utils/ResponseError.class

C:\Users\Srini\Documents\_Book\workspace\weatherman>heroku create weatherman bot
Creating weatherman-bot... done
https://weatherman-bot.herokuapp.com/ | https://git.heroku.com/weatherman-bot.git

C:\Users\Srini\Documents\_Book\workspace\weatherman>

```

Terakhir, kami akan memasukkan kode tersebut ke Heroku dan juga memanggilnya sebagai layanan web. Untuk melakukannya, jalankan perintah berikut:

> git push heroku master

Anda akan melihat pesan berikut, yang berarti kode telah dimasukkan, berhasil dikompilasi, dan diluncurkan juga:

```

remote:      [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus/3.1/plexus-3.1.pom (19 KB at 2019.9 KB/sec)
remote:      [INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest/1.0/plexus-digest-1.0.pom
remote:      [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest/1.0/plexus-digest-1.0.pom (2 KB at 129.3 KB/
sec)
remote:      [INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-components/1.1.7/plexus-components-1.1.7.pom
remote:      [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-components/1.1.7/plexus-components-1.1.7.pom (5 KB
at 607.3 KB/sec)
remote:      [INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.5/plexus-utils-3.0.5.jar
remote:      [INFO] Downloading: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest/1.0/plexus-digest-1.0.jar
remote:      [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-digest/1.0/plexus-digest-1.0.jar (12 KB at 526.4 KB
/sec)
remote:      [INFO] Downloaded: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.5/plexus-utils-3.0.5.jar (226 KB at 9376.
9 KB/sec)
remote:      [INFO] Installing /tmp/build_c8544ea7c3a5ce51ff6859e4aeFa109e/target/weatherman-0.0.1-SNAPSHOT.jar to /app/tmp/cache/.m2/repository/co
m/sjanarthanam/weatherman/0.0.1-SNAPSHOT/weatherman-0.0.1-SNAPSHOT.jar
remote:      [INFO] Installing /tmp/build_c8544ea7c3a5ce51ff6859e4aeFa109e/pom.xml to /app/tmp/cache/.m2/repository/com/sjanarthanam/weatherman/0.0
.1-SNAPSHOT/weatherman-0.0.1-SNAPSHOT.pom
remote:      [INFO] -----
remote:      [INFO] BUILD SUCCESS
remote:      [INFO] -----
remote:      [INFO] Total time: 10.149 s
remote:      [INFO] Finished at: 2017-06-19 14:38:13+00:00
remote:      [INFO] Final Memory: 22M/160M
remote:      [INFO] -----
remote:  ----> Discovering process types
remote:      Procfile declares types -> (none)
remote:
remote:  ----> Compressing...
remote:      Done: 51.1M
remote:  ----> Launching...
remote:      Released v3
remote:      https://weatherman-bot.herokuapp.com/ deployed to Heroku
remote:
remote: Verifying deploy... done.
To https://git.heroku.com/weatherman-bot.git
 * [new branch]      master -> master

```

Sekarang kita telah berhasil membuat chatbot sebagai layanan web, kita perlu membuat bot ini dapat berbicara dengan pengguna di Facebook Messenger.

Publikasi di Facebook Messenger

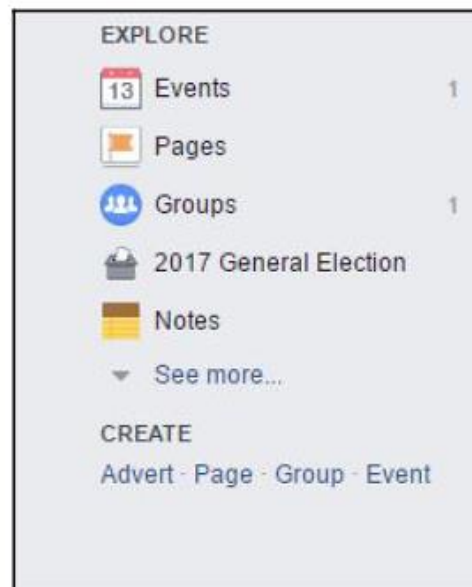
Untuk mempublikasikan chatbot yang dibuat khusus di Facebook, Anda perlu mengikuti langkah-langkah berikut.

Membuat Halaman Facebook

Anda memerlukan Halaman Facebook yang dapat menampung chatbot. Untuk membuat Halaman Facebook, Anda memerlukan akun Facebook. Jika Anda belum memilikinya, silakan buat satu di www.facebook.com.

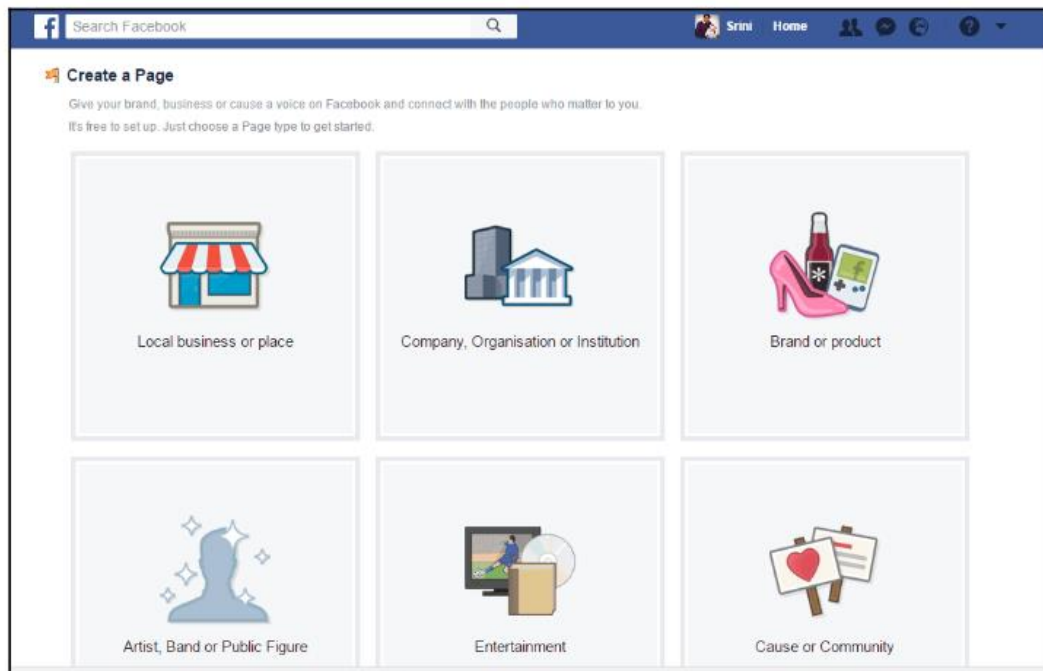
Setelah Anda masuk ke Facebook, Anda seharusnya dapat membuat sejumlah halaman. Untuk membuat halaman, ikuti langkah-langkah berikut:

1. Di sebelah kiri, ada menu dengan tiga tab: SHORTCUTS, EXPLORE, dan CREATE. Di bawah BUAT, klik Halaman:



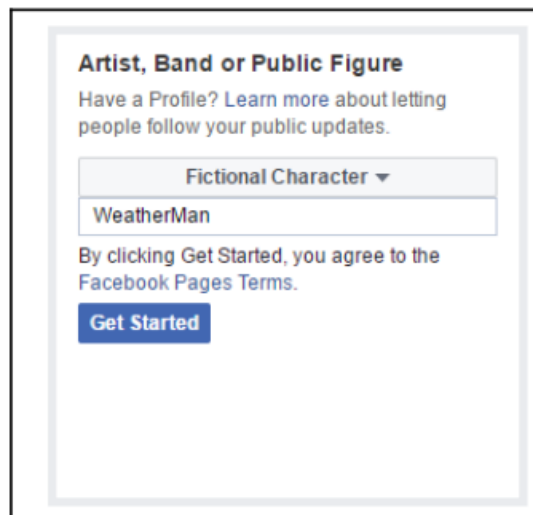
Gambar 7.5 menu pada akun facebook

2. Halaman Facebook dapat dibuat sebagai profil publik untuk mewakili bisnis, merek, selebriti, tujuan publik, dan sebagainya. Anda akan melihat menu berikut:



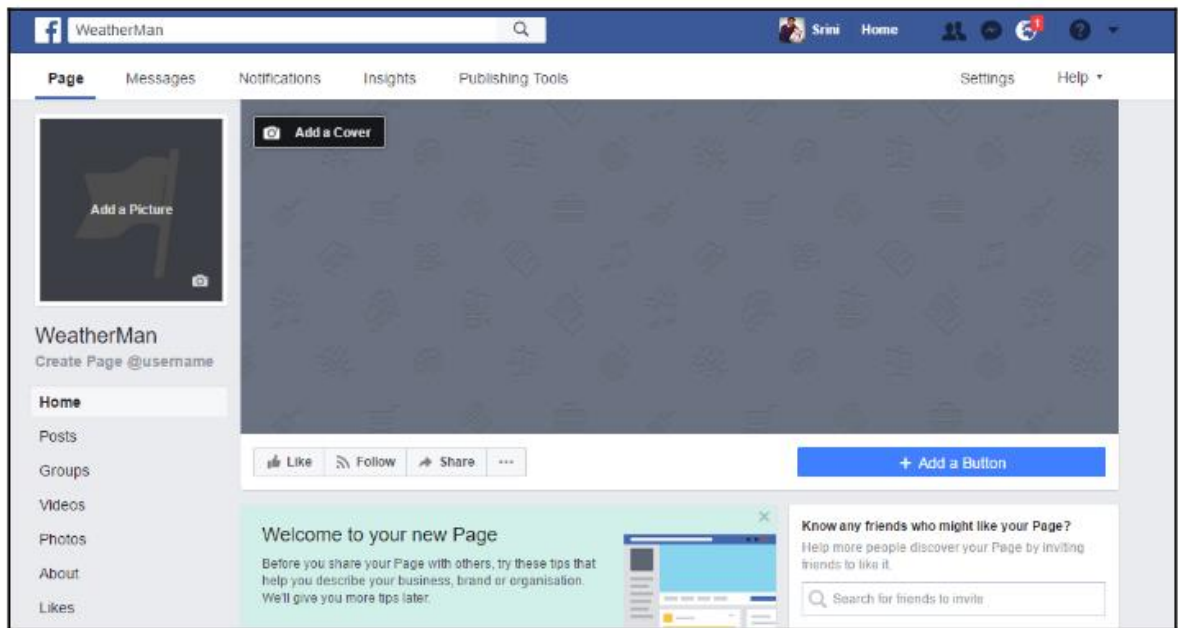
Gambar 9.6 membuat halaman

Pilih Artis, Band atau Tokoh Masyarakat. Karena kita sedang membangun chatbot, mari kita buat sebagai karakter fiksi. Pilih Karakter Fiksi dari menu drop-down, beri nama, dan klik Memulai:



Gambar 9.7 memilih karakter

3. Sekarang kita telah membuat halaman. Mari kita lanjutkan dan konfigurasi:



fb_new_blank_page

Gambar 9.8 konfigurasi halaman

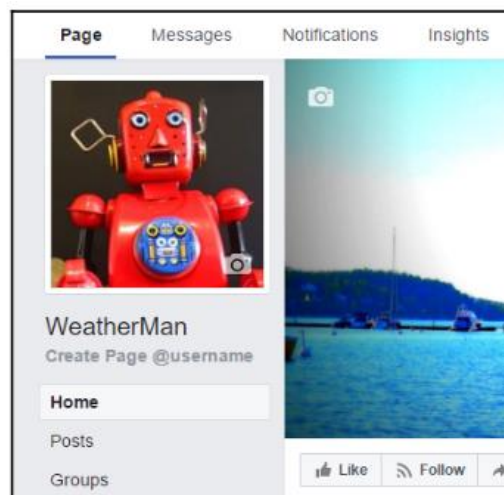
Tandai halaman Anda. Tambahkan gambar profil dengan mengklik Tambahkan Gambar dan gambar spanduk dengan mengklik Tambahkan Sampul (selalu gunakan gambar yang hak ciptanya Anda miliki).

Anda sekarang seharusnya dapat membuat postingan pertama Anda di halaman tersebut.

Membuat nama pengguna

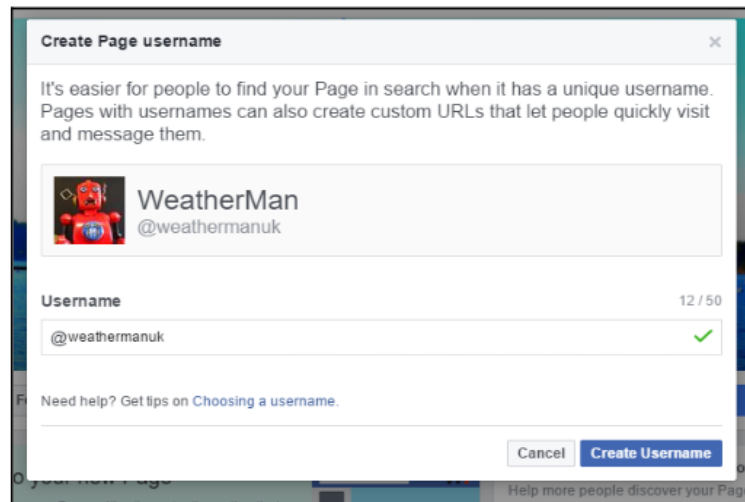
Setelah membuat halaman, mari kita lakukan dua hal lagi sebelum melanjutkan: menambahkan deskripsi halaman dan membuat nama pengguna.

1. Untuk membuat nama pengguna, klik Buat Halaman @namapengguna tepat di bawah gambar profil:



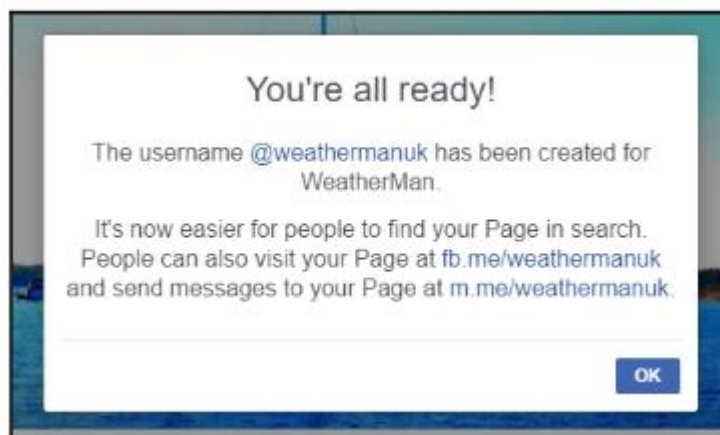
Gambar 9.9 memberi nama pada akun facebook

2. Anda akan dimintai Nama Pengguna. Nama ini akan mengidentifikasi halaman secara unik dan membuat URL mudah diingat. Anda mungkin harus mencoba beberapa nama karena nama yang ada dalam pikiran Anda mungkin sudah terpakai:



Gambar 9.10 mengkonfigurasi nama akun

3. Terakhir, setelah dibuat, Anda akan diberitahu tentang halaman baru dan URL Messenger:



Gambar 9.11 akun siap digunakan

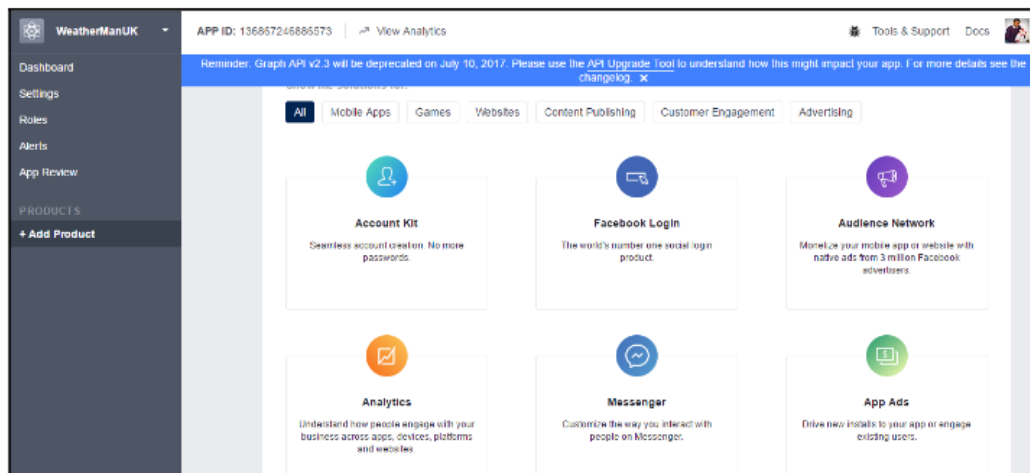
7.7 Membuat Aplikasi Facebook

Anda perlu membuat Aplikasi Facebook yang dapat terhubung ke Halaman Facebook yang baru saja kita buat. Aplikasi Facebook dapat dibuat di portal pengembang Facebook. Ikuti langkah-langkahnya:

1. Navigasikan browser Anda ke <https://developers.facebook.com>.
2. Klik tombol MyApps di pojok kanan atas (di sebelah ikon profil Anda).
3. Klik Tambahkan Aplikasi Baru.
4. Isi nama tampilan aplikasi dan email kontak Anda dan klik Buat ID Aplikasi:

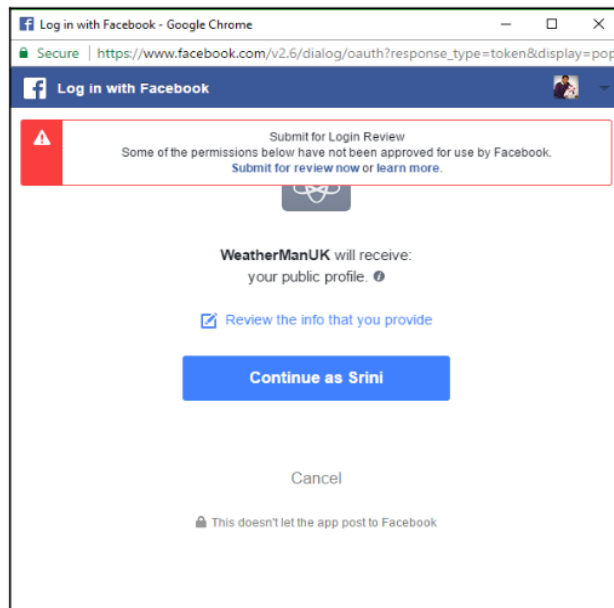
Gambar 9.12 tampilan membuat aplikasi

5. Setelah membuat shell aplikasi, kita perlu menambahkan produk ke dalamnya. Dalam kasus kita, karena kita sedang membangun chatbot, kita perlu menambahkan produk Messenger ke aplikasi:



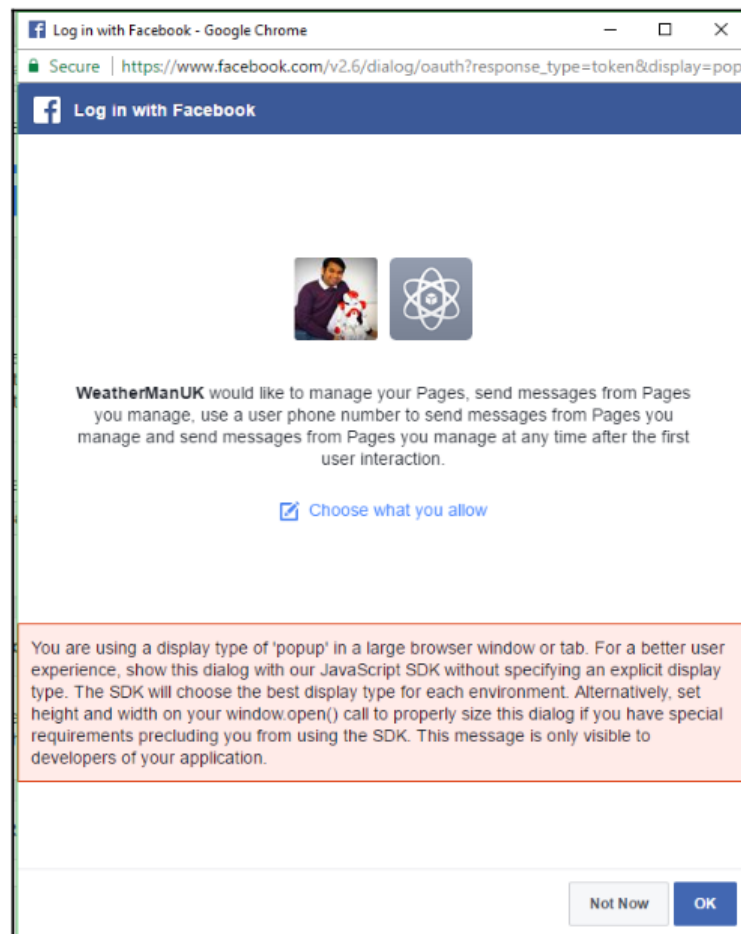
Gambar 9.13 membuat shell aplikasi

6. Arahkan mouse Anda ke Messenger dan klik Set Up. Ini akan menambahkan Messenger ke aplikasi dan membawa Anda ke halaman Pengaturan. Gulir ke bawah halaman ke tab Pembuatan Token. Di sinilah kami menautkan aplikasi ke halaman yang kami buat.
7. Klik Pilih Halaman dan pilih halaman yang Anda inginkan untuk ditautkan dengan aplikasi. Ini akan membuka kotak dialog otentikasi di mana Anda harus mengizinkan aplikasi mengakses halaman tersebut. Klik Lanjutkan sebagai X, dengan X adalah nama depan Anda. Jangan khawatir tentang peringatan untuk saat ini:



Gambar 7.14 login facebook

8. Di halaman berikutnya, klik OK untuk menyelesaikan proses. Ini secara otomatis akan menutup kotak dialog:



Gambar 7.15 menutup kotak dialog setelah login

9. Anda sekarang akan melihat bahwa halaman tersebut diberi token yang disebut Token Akses Halaman. Salin dan sisihkan.
10. Kita harus menyiapkan webhook untuk menyelesaikan proses ini tetapi kita harus melakukannya nanti.

7.8 Membuat Aplikasi Web Antarmuka Facebook

Pada langkah ini, kita akan membuat antarmuka yang memungkinkan chatbot kita berbicara dengan Aplikasi Facebook. Di bagian sebelumnya, kami membuat Aplikasi Facebook. Aplikasi ini harus berkomunikasi dengan antarmuka yang akan kita buat di bagian ini. Info tentang modul ini harus masuk ke pengaturan webhook yang belum kita selesaikan di bagian sebelumnya. Kami akan membuat proyek baru untuk menampung aplikasi antarmuka ini. Sebut saja ahli cuaca-fbmi. Di konsol Anda, buat direktori terpisah (weatherman-fbmi) untuk menampung proyek ini. Di direktori ini, kita akan membuat tiga file berikut:

- ❖ Index.js
- ❖ Package.json
- ❖ Procfile

index.js adalah program Node.js yang bertindak sebagai server web untuk mengirim dan menerima pesan ke dan dari Aplikasi Facebook kami:

```
'use strict';
const express = require('express')
const bodyParser = require('body-parser')
const request = require('request')

const app = express()

// set your weatherman fb page access token
const token = '<YourFBPageAccessToken>';

const botServerUrl = 'https://weatherman-bot.herokuapp.com/bot';

app.set('port', (process.env.PORT || 5000))

// Process application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({extended: false}))

// Process application/json
app.use(bodyParser.json())

app.use(express.static('public'))

// Index route
app.get('/', function (req, res) {
  res.send('Hello world, I am Weatherman!.')
})

// Spin up the server
```

```
app.listen(app.get('port'), function() {
  console.log('running on port', app.get('port'))
})
```

Anda harus mengganti <YourFBPageAccesToken> dengan token akses halaman yang Anda dapatkan dari Langkah 9 di bagian Membuat Aplikasi Facebook. Anda mungkin juga harus mengganti botServerURL dengan milik Anda sendiri.

Kode sebelumnya cukup untuk memutar server saat dimasukkan ke Heroku. Namun kami memerlukan server untuk benar-benar menjembatani Aplikasi Facebook dan chatbot kami. Kami sekarang akan menerapkan rute yang diperlukan ke server web sebelumnya. Tambahkan kode berikut ke index.js:

```
// for Facebook verification
app.get('/webhook/', function (req, res) {
  if (req.query['hub.verify_token'] === 'iam-weatherman-bot') {
    res.send(req.query['hub.challenge'])
  }
  res.send('Error, wrong token')
})

//FBM webhook
app.post('/webhook/', function (req, res) {
  console.log(JSON.stringify(req.body));
  let messaging_events = req.body.entry[0].messaging
  for (let i = 0; i < messaging_events.length; i++) {
    let event = req.body.entry[0].messaging[i]
    let sender = event.sender.id
    let recipient = event.recipient.id
    let time = req.body.entry[0].time

    // we call the chatbot here..
    if (event.message && event.message.text) {
      let text = event.message.text
      //send it to the bot
      request({
        url: botServerUrl,
        method: 'POST',
        form: {
          'userUtterance':text
        }
      }, function (error, response, body) {
        //response is from the bot
        if (!error && response.statusCode == 200) {
          // Print out the response body
          body = body.substring(1,body.length-1);
          body = body.replace(/\//g, '')
        }
      })
    }
  }
})
```

```

        let botOut = JSON.parse(body)
        if (botOut.botUtterance != null){
            sendTextMessage(sender, botOut.botUtterance)
        }
        } else {
            sendTextMessage(sender, 'Error!')
        }
    }); //request ends
}
}
res.sendStatus(200)
})
function sendTextMessage(sender, text) {
    if (text != 'null'){
        let messageData = { 'text':text }
        request({
            url: 'https://graph.facebook.com/v2.6/me/messages',
            qs: {access_token:token},
            method: 'POST',
            json: {
                recipient: {id:sender},
                message: messageData,
            }
        }, function(error, response, body) {
            if (error) {
                console.log('Error sending messages: ', error)
            } else if (response.body.error) {
                console.log('Error: ', response.body.error)
            }
        })
    }
}
}
}

```

Kode sebelumnya membuat rute POST yang disebut webhook yang akan dipanggil oleh Aplikasi Facebook dengan ucapan pengguna. Ini akan dikirim ke chatbot menggunakan URL-nya (botServerURL) sebagai permintaan POST lainnya. Respons bot dikirim kembali ke Messenger dalam metode sendTextMessage(). Ada format pesan lain yang dapat digunakan untuk mengirim tanggapan yang lebih kaya dari bot. Kita akan melihatnya di bagian X.

Mari kita buat file package.json untuk memberi tahu Heroku paket-paket yang diperlukan untuk index.js. Buat file package.json di editor teks Anda dengan JSON berikut:

```

{
  "name" : "weatherman-bot-server",
  "version" : "1.0.0" ,
  "description" ; "Weatherman bot fbml server",
  "main" : "index.js" ,
  "scripts" : {
    "test" : "echo \ "Error:no test specified\ "&& exit 1"
  } ,
  "author" : "Srini Janarthanam" ,
  "license" : "ISC",
  "dependencies " : {
    "body-parser" : "^1.15.2" ,
    "express" : "^4.14.0" ,
    "request" : "^2.72.0"
  }
}

```

```
}

```

Sebagai langkah terakhir, kita perlu membuat Procfile untuk memberitahu Heroku untuk menjalankan index.js. Buat Procfile di editor teks Anda dengan perintah berikut.

```
Web : node index.js

```

Ikuti langkah-langkah berikut untuk mendorong program antarmuka ke cloud. Dari jendela konsol, tempat proyek antarmuka Weatherman Anda berada, jalankan perintah berikut untuk membuat aplikasi Heroku dan dorong aplikasi antarmuka ke cloud:

```
>git inti
>git add.
>git commit -m "fbmi for weatherman v1"
>Heroku create weatherman-bot-fbmi
> git push Heroku master

```

Anda akan menerima pesan dengan URL aplikasi di dalamnya. Dalam kasus saya, ini adalah <https://weatherman-bot-fbmi.herokuapp.com>.

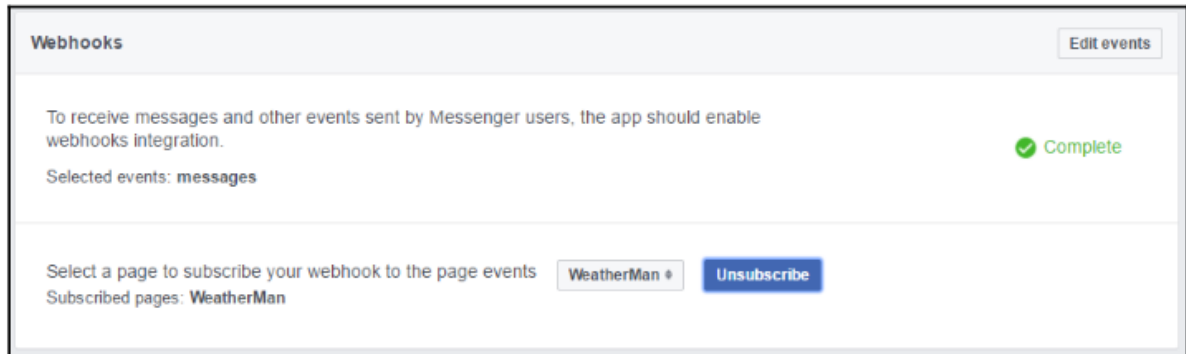
7.9 Menghubungkan Aplikasi Facebook Dan Aplikasi Antarmuka

Terakhir, kita harus menyiapkan webhook yang tidak lengkap pada Langkah 10 saat membuat Aplikasi Facebook. Anda mungkin telah mengamati bahwa kami telah membuat dua rute untuk dijadikan webhook. Rute GET digunakan untuk memverifikasi aplikasi dan rute POST digunakan untuk mengkomunikasikan pesan ke dan dari chatbot:

The image shows a 'New Page Subscription' dialog box. It has a title bar with a close button. The 'Callback URL' field is filled with 'https://weatherman-bot-fbmi.herokuapp.com/webhook'. The 'Verify Token' field is filled with 'iam-weatherman-bot'. Below this, there is a section for 'Subscription Fields' with a grid of checkboxes: 'messages' (checked), 'message_deliveries', 'messaging_pre_checkouts', 'messaging_referrals', 'messaging_postbacks', 'message_reads', 'messaging_checkout_updates', 'message_echoes', 'messaging_optins', and 'messaging_payments'. At the bottom right, there are 'Cancel' and 'Verify and Save' buttons. A 'Learn more' link is also present at the bottom left.

Gambar 9.16 halaman baru subscription

Berlangganan ke Halaman Facebook WeatherMan untuk menerima pesan yang dikirim oleh pengguna Messenger di layanan web:

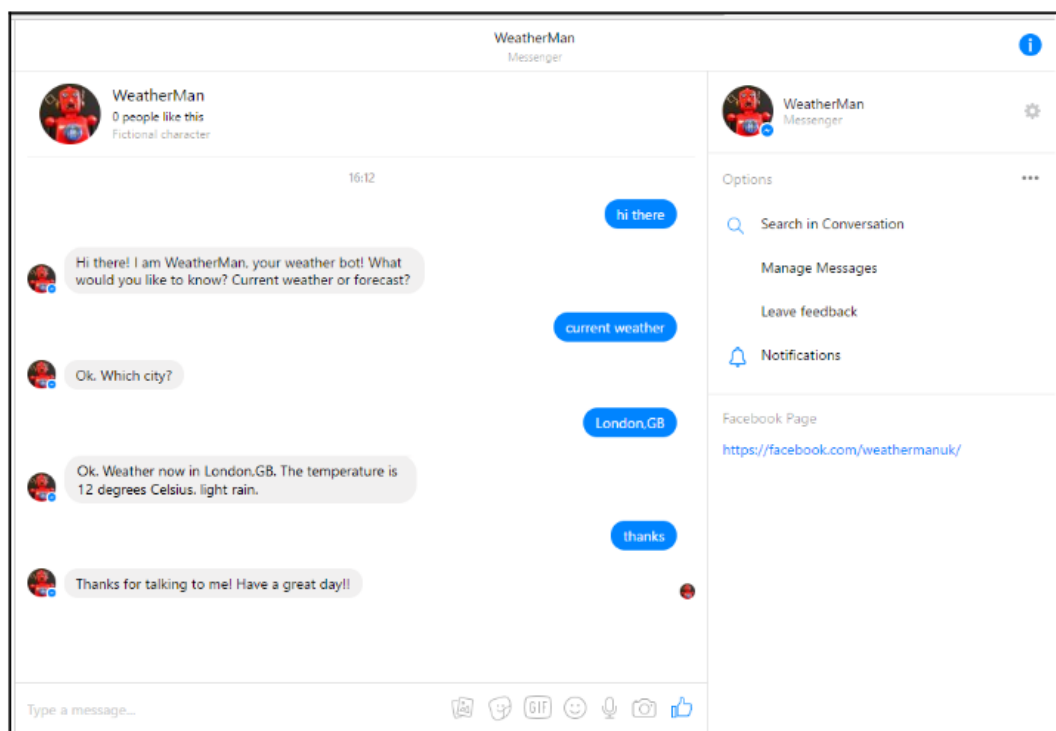


Gambar 9.17 layanan webhooks

Kami sekarang siap untuk menguji chatbot!

7.10 Menguji Chatbot Anda

Sekarang chatbot sudah aktif dan berjalan, kita dapat mengujinya di Halaman Facebook yang menampungnya. Nama pengguna untuk halaman tersebut dibuat di bagian Membuat nama pengguna. Navigasi ke <https://m.me/weathermanuk> di browser web. Ini akan membuka Facebook Messenger, tempat Anda dapat mengobrol dengan bot WeatherMan kami. Silakan dan sapa yang di sana:



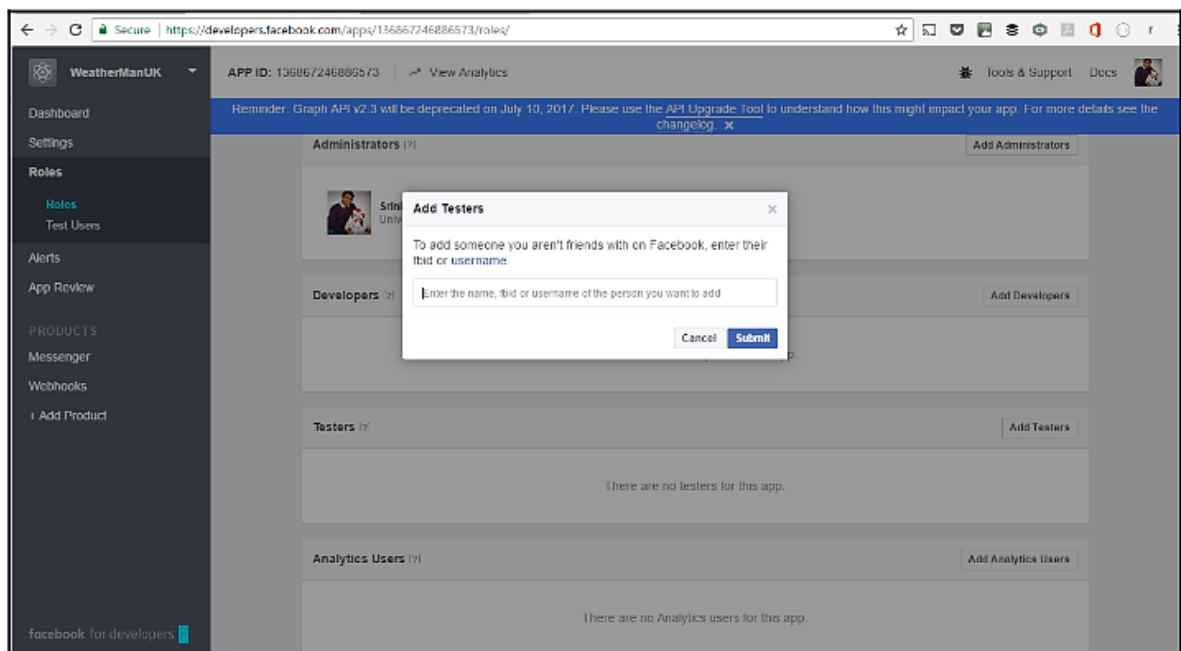
Gambar 9.18 tampilan chat pengujian

Menambahkan lebih banyak penguji

Anda mungkin memperhatikan bahwa hanya Anda yang memiliki akses ke chatbot. Siapa pun yang membuka URL yang sama dan menyapa tidak akan ditanggapi oleh chatbot. Ini karena chatbotnya belum dipublikasikan. Anda dapat menambahkan lebih banyak penguji untuk membantu Anda menguji chatbot. Inilah cara Anda melakukannya:

1. Navigasikan browser Anda ke <https://developers.facebook.com>.
2. Klik aplikasi Anda.
3. Di Dasbor, pilih Peran.
4. Di bawah tab Testers, Anda dapat menambahkan pengguna Facebook dengan menyebutkan nama pengguna mereka.

Pengguna yang ditambahkan akan memiliki akses ke chatbot. Namun ingat, bot Anda masih belum dipublikasikan:



Gambar 9.19 memberikan link publikasi chatbot

7.11 Mempublikasikannya

Langkah terakhir dari proses ini adalah mengajukan permohonan ke Facebook untuk menjadikan chatbot Anda publik. Langkah terakhir ini akan membuat chatbot Anda tersedia untuk semua orang di URL Messenger (<https://m.me/weathermanuk>).

1. Di dasbor aplikasi Anda, klik Pengaturan lalu Dasar di menu Dasbor.
2. Unggah ikon untuk mewakili chatbot. Ini harus berupa gambar 1124 x 1024 piksel.
3. Pilih kategori: Aplikasi untuk Messenger.
4. Tambahkan URL ke kebijakan privasi. Kebijakan ini harus memberi tahu pengguna apa yang terjadi pada data pengguna yang dikumpulkan. Ada beberapa layanan yang dapat membantu Anda membuat kebijakan privasi Aplikasi Facebook:

Percakapan Antarmuka Pengguna (User Interface) Dalam Chatbot – Dr. Budi Raharjo

- *IUBENDA*: <https://www.iubenda.com/blog/privacy-policy-facebook-app/>
 - *TermsFeed*: <https://termsfeed.com/blog/privacy-policy-url-facebook-app/>
5. Pada menu Dasbor, klik Tinjauan Aplikasi.
 6. Jadikan aplikasi Anda publik dengan memilih Ya.
 7. Pada menu Dasbor, pilih Messenger di bawah PRODUK.
 8. Di sebelah kanan, temukan tab Tinjauan Aplikasi Untuk Messenger.
 9. Tambahkan halaman_pesanan ke pengiriman.
 10. Turun ke tab Pengiriman Saat Ini.
 11. Klik Edit Catatan terhadap halaman_messaging.
 12. Pilih Halaman Facebook tempat chatbot berlangganan.
 13. Pilih Pengalaman Messenger Anda mencakup balasan otomatis dan berikan contoh interaksi tentang apa yang dapat dikatakan pengguna dan apa yang akan ditanggapi oleh bot.
 14. Tambahkan catatan lain untuk pengulas jika Anda mau. Ini akan membantu pengulas memahami chatbot dan menetapkan ekspektasi yang tepat.
 15. Klik Simpan. Kembali ke Pengaturan Messenger. Anda akan menemukan tombol Kirim untuk ditinjau diaktifkan. Klik untuk pengiriman. Jika tidak diaktifkan, Anda akan menemukan daftar item yang perlu Anda kerjakan sebelum Anda dapat mengirimkannya.

Anda akan segera menerima pesan yang dapat Anda lihat di halaman Peringatan. Klik Peringatan pada menu Dasbor. Anda akan melihat pesan yang menyatakan bahwa aplikasi sedang ditinjau. Peninjauan biasanya memakan waktu beberapa hari. Setelah ditinjau dan disetujui, chatbot Anda siap digunakan publik.

7.12 Lebih banyak format pesan Facebook

Di bagian ini, kita akan menjelajahi format pesan Facebook lainnya selain teks. Pada contoh sebelumnya, kami memprogram chatbot untuk mengirimkan tanggapan teks kembali ke pengguna. Namun, Facebook memiliki lebih banyak format yang dapat digunakan untuk memberikan pengalaman yang lebih kaya kepada pengguna. Anda dapat melakukan ini dengan mengubah konten elemen pesan dalam permintaan POST dalam metode `sendTextMessage()` di file `index.js`.

Tipe Konten

Selain teks, jenis konten lain seperti gambar, klip audio dan video, serta file dapat dikirim ke pengguna.

Lampiran gambar

Gambar dalam format JPEG, PNG, dan GIF dapat dikirim ke pengguna menggunakan format isi pesan berikut:

```
"message" : "  
  "attachment" : {  
    "type" : "image" ,
```

```

        "payload" : {
          "url" : "https://myimagelibrary.com/justwannadrawsomething.jpg"
        }
      }
    }
  }
}

```

Lampiran audio

Klip audio dapat dikirim ke pengguna menggunakan elemen pesan berikut:

```

"message" : {
  "attachment" : {
    "type" : "audio" ,
    "payload" : {
      " url " : "https://mymp3libraey.com/ijustwannasing.mp3 "
    }
  }
}

```

Lampiran video

File video dapat dikirim melalui Messenger dengan menentukan URL-nya dengan isi pesan berikut:

```

"message" : {
  "attachment" : {
    "type" : "video" ,
    "payload" :{
      "url": "https://myvidiolibrary.com/ijustwannadance.mp4"
    }
  }
}

```

File

File dapat dikirim ke pengguna menggunakan format pesan berikut dengan menentukan URL file:

```

"message" : {
  "attachment" : {
    "type" : "file" ,
    "payload" :{
      "url": https://myfileslibrary.com/ijustwannawrite.pdf
    }
  }
}

```

Templat

Selain mengirim teks sederhana dan tipe konten lainnya, Facebook menyediakan cara untuk mengirim kartu. Ini adalah elemen GUI yang menarik secara visual. Terdapat juga opsi

untuk menambahkan tombol ke kartu yang berfungsi sebagai respons yang dapat diberikan pengguna.

Templat tombol

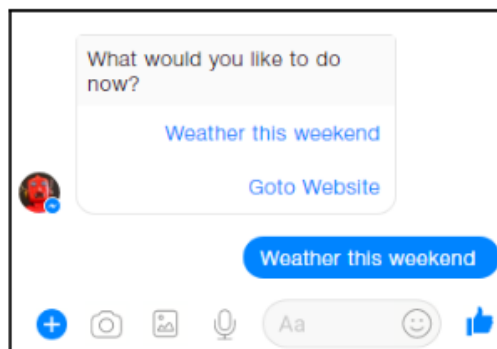
Templat tombol dapat dibuat menggunakan format pesan berikut. Pesan berikut akan menampilkan kartu dengan dua tombol. Label pada tombolnya adalah Beli sekarang dan Info lebih lanjut. Mengklik Beli sekarang akan mengirimkan payload (PAYLOAD_BUY) sebagai pesan respons kembali ke bot. Mengklik tombol Info lebih lanjut akan membawa pengguna ke halaman web yang ditentukan dalam jendela tampilan web tertanam:

```

"message" : {
  "attachment" : {
    "type" : "video" ,
    "payload" :{
      "template_type" : "button",
      "text":"what would you like to do now?",
      "buttons" : [
        \{
          "type" : "postback" ,
          "title" : "Weather this weekend" ,
          "payload" : " PAYLOAD_WEEKEND_LONDON"
        }
        {
          "Type" : "Goto website" ,
          "url" : "https:// myweather.com/London",
          "title" : "More Info"
        }
      ]
    }
  }
}

```

Pesan sebelumnya akan ditampilkan seperti yang ditunjukkan pada tangkapan layar berikut:



Gambar 9.20 tangkapan dari chatbot yang terbaru

Templat daftar

Templat daftar dapat digunakan untuk membuat daftar sejumlah item dalam daftar vertikal. Dengan menggunakan format pesan berikut, Anda bisa membuat daftar yang disajikan dengan cara visual yang menarik:

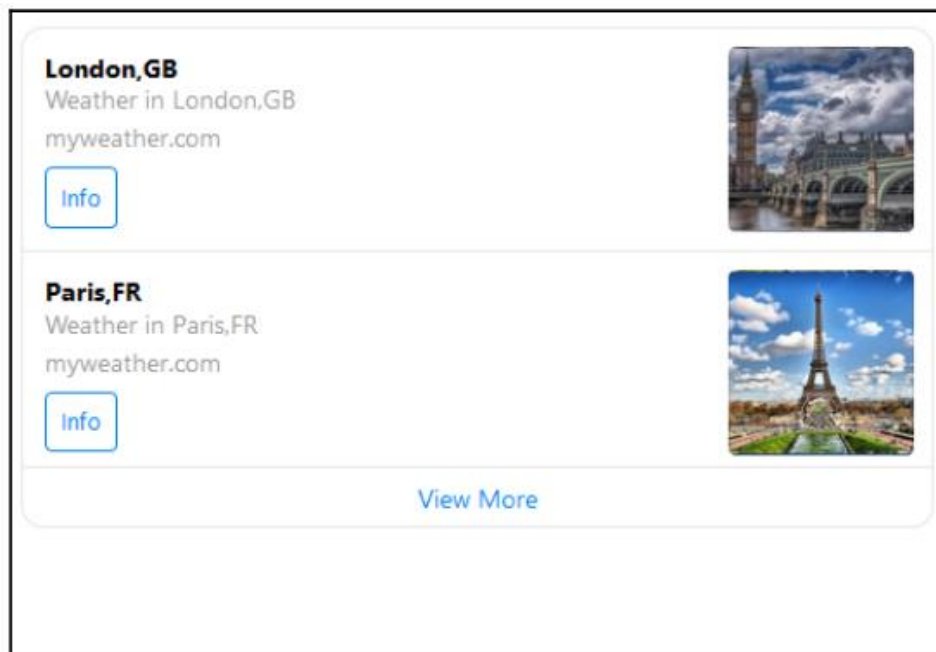
```

"message" : {
  "attachment" : {
    "type" : "template" ,
    "payload" :{
      "template_type" : "list",
      "top_element_style":"compact",
      "elements" : [
        \{
          "title" : "London GB" ,
          "image_url" :
"https://myweather.com/london.png" ,
          "subtitle" : " weather in london ,GB",
          "default_action" : {
            "type": "web_url" ,
            "url" :https://myweather.com/london
            "messenger_extensions" : true,
            "webview_height_ratio":"tall",
            "fallback_yrl" : "https://myweather.com?"
          }
          "buttons" : {
            {
              "Title" : "info" ,
              "type" : "web_url",
              "url" :
https://myweather.com/london/weather",
              "messenger_extansions" : true,
              "webview_height_ratio" : "tall",
              "fallback_url":"https://myweather.com/"
            }
          }
        }
      ]
    }
  }
  {
    "title" : "paris,FR",
    "image_url" : "https://myweather.com/paris.pmg",
    "subtitle":"weather in paris,FR",
    "default_action" : {
      "type" : "web_url",
      "url":"https://myweather.com/paris/weather",
      "messenger_extensions":true
      "webview_height_ratio": "tail",
      "fallback_url": https://myweather.com?,
    }
  }
}
"buttons" : [
  {
    "title" : "View_More",
    "type" : "postback" ,
    "payload" : "payload"
  }
]
}

```

Kode sebelumnya juga berfungsi sebagai contoh bagaimana Anda dapat menggunakan templat daftar di chatbot cuaca kami. Anda dapat membuat daftar kota-kota terkemuka atau mengizinkan pengguna membuat kota favoritnya sendiri. Daftar ini akan memberikan jalan pintas untuk mendapatkan info cuaca pada daftar kota. Pada kode sebelumnya, hal ini dilakukan dengan menambahkan info berpusat pada kota ke daftar array elemen JSON. Setiap elemen memiliki dua URL. Salah satunya adalah saat pengguna menyentuh elemen itu sendiri dan yang lainnya adalah saat pengguna mengeklik tombol ajakan bertindak.

Pada kode sebelumnya, tombol diatur untuk mengarahkan pengguna ke URL web daripada mengirim pesan payload ke bot. Namun keduanya bisa dilakukan. Anda hanya perlu mengubah jenis tombolnya. Dan inilah tampilannya di Messenger:



Gambar 9.21 pilihan info cuaca

Templat umum

Templat generik adalah salah satu templat pertama yang diperkenalkan oleh Facebook untuk membuat kartu yang menarik secara visual. Ini juga disebut korsel. Hal ini memungkinkan Anda menyajikan daftar item sebagai daftar kartu horizontal, di mana pengguna dapat menggulir item tersebut dan merespons menggunakan tombol. Setiap kartu dapat memiliki maksimal tiga tombol. Untuk membuat templat umum, gunakan format pesan berikut:

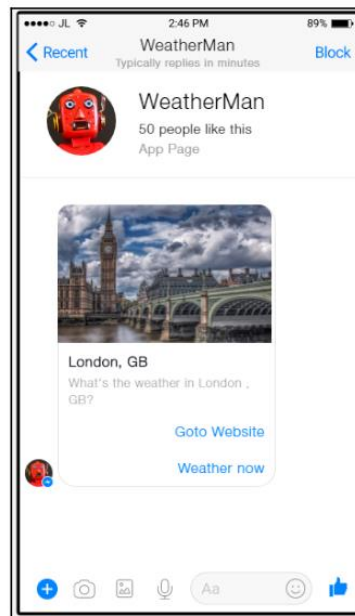
```
"message" : {
  "attachment" : {
    "type" : "template" ,
    "payload" :{
      "template_type" : "generic",
```

```

"elements" : [
{
  "title" : "London, GB" ,
  "image_url":
"https://myweather.com/london.png" ,
  "subtitle" : " What's the weather in london
,GB",
  "default_action" : {
    "type": "web_url" ,
    "url" :https://myweather.com/london" ,
    "messenger_extensions" : true,
    "webview_height_ratio":"tall",
    "fallback_url" : "https://myweather.com/"
  }
  "buttons" : {
    {
      "type":"web_url",
      "url":https://myweather.com/london/weather,
      "title":Goto Website"
    },{
      "type":"postback",
      "title":"Weather now",
      "payload":"PAYLOAD_CURRENT_WEATHER_LONDON
"
    }
  }
}
}

```

Contoh sebelumnya hanya memiliki satu kartu. Tambahkan lebih banyak ke daftar elemen untuk membuat lebih banyak kartu, mungkin satu untuk setiap kota. Perhatikan bahwa kartu tersebut memiliki dua tombol. Salah satunya adalah tampilan web, di mana menekan tombol berlabel Goto Website akan membawa Anda ke situs web dengan informasi cuaca di London pada browser web yang tertanam. Dan yang berikutnya adalah tombol postback, yang akan mengirimkan pesan postback PAYLOAD_CURRENT_WEATHER_LONDON sebagai respons ke chatbot:



Gambar 9.34 tampilan template sederhana

Dengan menggunakan kombinasi elemen ini dan lainnya, Anda dapat menciptakan pengalaman percakapan yang menakjubkan bagi pengguna di Facebook Messenger. API Facebook Messenger berkembang pesat dan elemen baru ditambahkan untuk membantu pengembang menciptakan percakapan chatbot yang menakjubkan di platform. Jadi perhatikan perkembangannya.

Ringkasan

Dalam bab ini, kita telah melihat cara membuat chatbot menggunakan sumber informasi eksternal dan mengintegrasikannya dengan Facebook Messenger. Kami telah melihat cara mempublikasikan bot di Messenger dan fitur eksklusif, seperti tombol dan templat, yang dapat membantu kami membangun pengalaman percakapan yang kaya dan menarik bagi pengguna di Messenger.

Pada bab berikutnya, kita akan beralih dari interaksi berbasis tombol dan mengeksplorasi cara memproses ucapan pengguna untuk membuat chatbot lebih fleksibel bagi pengguna. Kami akan menjelajahi perangkat pemahaman bahasa alami yang disebut Dialogflow dan membuat bot persona yang akan meniru percakapan dengan Albert Einstein.

BAB 8

MEMBANGUN CHATBOT MENGGUNAKAN BAHASA ALAMI

Dalam bab ini, kita akan mengeksplorasi bagaimana kita dapat memproses masukan bahasa alami dari pengguna menggunakan alat dari Google yang disebut Dialogflow (sebelumnya disebut API.AI) untuk membangun chatbot yang dapat memahami masukan bahasa alami dari pengguna. Kami akan menerapkan chatbot pada dua platform: web chat dan Facebook Messenger. Kita akan mengeksplorasi konsep-konsep dalam bahasa alami seperti maksud, entitas, dan konteks, dan kita akan membahas bagaimana sebenarnya merancang dan menerapkan pengalaman percakapan yang kaya menggunakan ide-ide ini. Dengan pemahaman dasar tentang konsep-konsep ini, kami akan merancang dan membangun persona bot yang meniru fisikawan favorit dunia, Albert Einstein.

Pada akhir bab ini, Anda akan dapat:

- ❖ Pahami dasar-dasar Dialogflow
- ❖ Memahami konsep seperti maksud, entitas, dan konteks
- ❖ Buat agen Dialogflow
- ❖ Integrasikan agen Dialogflow ke obrolan web dan Facebook Messenger
- ❖ Perluas kemampuan agen menggunakan webhook

8.1 Memperkenalkan Dialogflow

Dialogflow (sebelumnya disebut API.AI) adalah platform pembuat agen percakapan dari Google. Ini adalah platform berbasis web yang dapat diakses dari browser web apa pun.

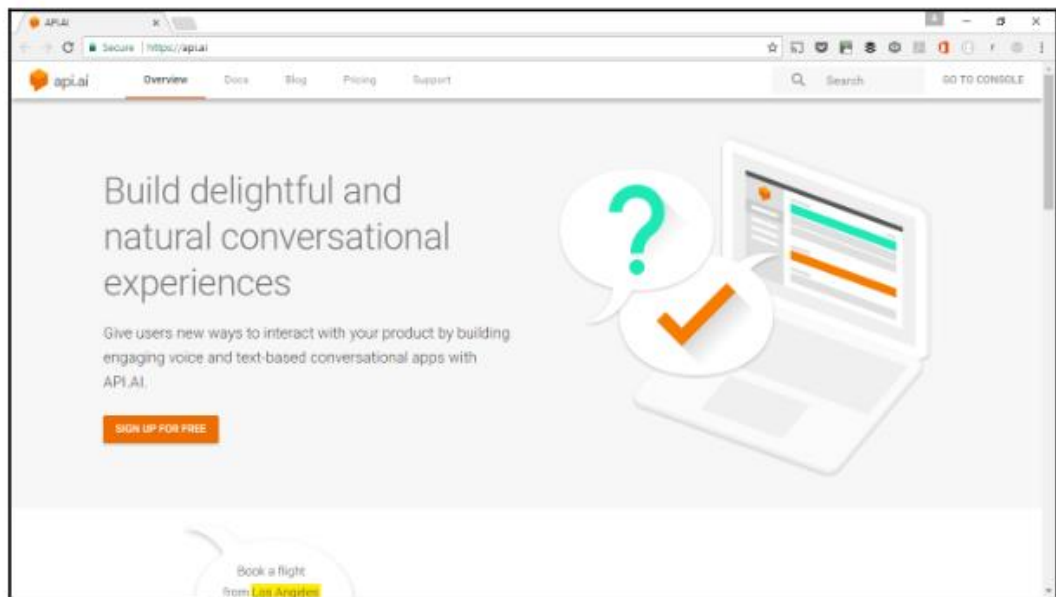
Alat ini telah berkembang seiring waktu dari apa yang dibuat sebagai jawaban terhadap Apple Siri untuk platform Android. Itu disebut SpeakTolt, sebuah aplikasi Android yang menciptakan pengalaman percakapan seperti Siri di ponsel pintar Android mana pun. Teknologi AI dan bahasa alami yang mendukung aplikasi SpeakTolt dibuka untuk pengembang sebagai API.AI pada tahun 2015.

API.AI memungkinkan pengembang untuk menciptakan pengalaman percakapan dengan menyediakan alat untuk melakukan dua jenis tugas: memahami dan menghasilkan ucapan bahasa alami dan mengelola percakapan. Meskipun ada banyak alat yang bersaing dengan API.AI yang tersedia di pasar, kami memilih API.AI karena ini adalah salah satu alat yang pertama, sangat matang dan terus berkembang, serta mudah digunakan dan diintegrasikan dengan layanan lain.

8.2 Menyiapkan Dialogflow

Pertama, mari kita buat akun pengembang di API.AI (sekarang disebut Dialogflow).

1. Buka <https://api.ai/>:



Gambar 8.1 halaman api.ai

2. **Klik BUKA KONSOL** di pojok kanan atas.
3. Masuk. Anda mungkin perlu menggunakan akun Google Anda untuk masuk.

Membuat agen dasar

Mari kita buat agen pertama kita di Dialogflow. Mari kita mulai dengan fitur dasar dan kemudian menambahkan fitur yang lebih kompleks:

1. Untuk membuat agen baru, klik menu drop-down di sebelah kiri halaman beranda (api.ai) dan klik Buat agen baru.
2. Isi formulir di sebelah kanan. Beri nama dan deskripsi. Pilih zona waktu dan klik BUAT.
3. Ini akan membawa Anda ke halaman dengan daftar maksud. Anda akan melihat bahwa sudah ada dua maksud: Maksud Penggantian Default dan Maksud Selamat Datang Default.
4. Mari tambahkan maksud pertama Anda. Maksud adalah apa yang ingin disampaikan oleh pengguna atau bot menggunakan ucapan atau penekanan tombol. Maksud adalah representasi simbolik dari suatu ucapan. Kita perlu niat karena ada banyak cara untuk meminta hal yang sama. Proses mengidentifikasi maksud adalah dengan memetakan berbagai cara secara jelas menuju suatu maksud. Misalnya, pengguna dapat menanyakan cuaca di kotanya menggunakan ucapan berikut:

"Bagaimana Cuaca Di London"

"Bagaimana Cuaca Di London"

"Cuaca Di London"

"Apakah Di Luar Cerah Sekarang"

Pada ujaran sebelumnya, pengguna meminta laporan cuaca di kota London. Dalam beberapa ucapan ini, mereka juga menyebutkan waktu (yaitu sekarang). Di negara lain,

hal ini tersirat. Langkah pertama algoritme kami adalah memetakan banyak ucapan ini ke dalam satu maksud: *request_weather_report*.

Nama Intent sesuai dengan maksud pengguna. Jadi beri nama mereka dari sudut pandang pengguna. Mari tambahkan maksud *user_greet* yang sesuai dengan tindakan menyapa chatbot oleh pengguna. Untuk menambahkan maksud, klik tombol CREATE INTENT.

5. Anda akan melihat halaman berikut di mana Anda dapat membuat maksud baru:

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>

Gambar 8.2 membuat INTENT

Beri nama pada maksud tersebut (misalnya, *user_greet*).

6. Tambahkan contoh ucapan pengguna di kolom teks Kata pengguna. Ini adalah contoh ucapan yang akan membantu agen mengidentifikasi maksud pengguna. Mari tambahkan beberapa ucapan salam yang mungkin diucapkan pengguna ke chatbot kita:

Halo.

Halo.

Halo albert

Halo dokter

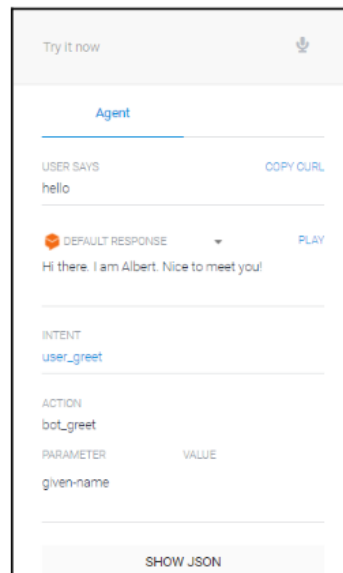
Selamat siang dokter

7. Abaikan tab Acara untuk saat ini dan lanjutkan ke tab Tindakan. Tambahkan nama untuk mengidentifikasi maksud sistem di sini (misalnya, `bot_greet` untuk mewakili sapaan chatbot kepada pengguna).
8. Di tab Response, tambahkan respon bot ke pengguna. Ini adalah ucapan sebenarnya yang akan dikirimkan bot kepada pengguna. Mari tambahkan ucapan berikut di kolom Respon teks. Anda dapat menambahkan lebih banyak respons sehingga agen dapat memilih satu respons secara acak agar tidak terlalu berulang dan membosankan:

`Hi there. I am Albert. Nice to meet you!`

Anda juga dapat menambahkan hingga 10 tanggapan tambahan dengan mengklik tambahkan isi pesan.

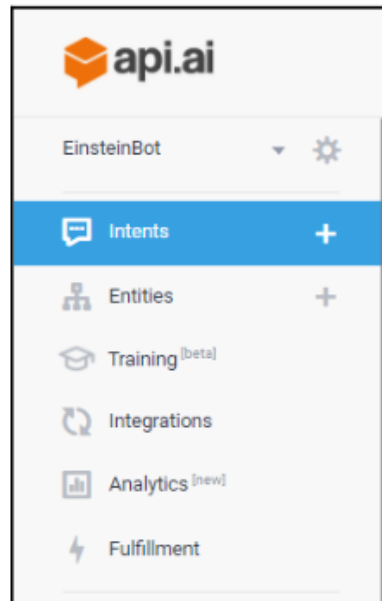
9. Klik tombol SIMPAN di pojok kanan atas untuk menyimpan maksud. Anda telah membuat niat pertama Anda untuk agen.
10. Uji dengan menggunakan simulator di sisi kanan halaman. Di kotak Coba sekarang, ketik hello dan tekan Enter:



Gambar 8.3 simulator api.ai

Anda akan melihat chatbot mengenali ucapan yang Anda ketik dan merespons dengan tepat.

11. Sekarang lanjutkan dan tambahkan beberapa maksud lagi dengan mengulangi langkah 5 hingga 10. Untuk membuat maksud baru, klik tanda + di samping opsi Maksud pada menu di sebelah kiri:



Gambar 8.4 menambahkan intent lebih banyak

Pikirkan informasi seperti apa yang akan ditanyakan pengguna ke chatbot dan buat daftarnya. Ini akan menjadi maksud pengguna. Berikut ini adalah contoh daftar untuk Anda mulai:

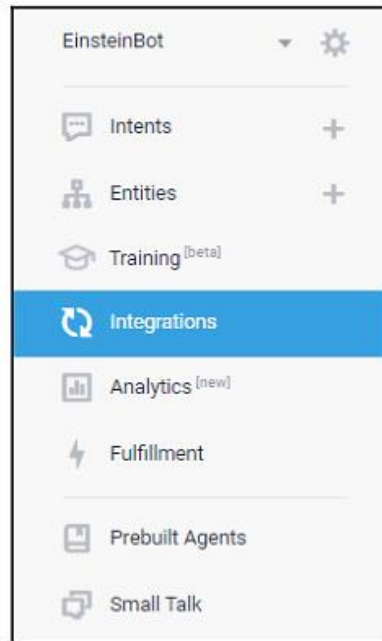
- ❖ Request_name
- ❖ Request_birth_info
- ❖ Request_parents_name
- ❖ Request_first_job_experience
- ❖ Request_info_on_hobbies
- ❖ Request_info_patent_job
- ❖ Request_info_lecturer_job_bern

Tentu saja, daftar ini tidak ada habisnya. Jadi lanjutkan dan bersenang-senanglah.

Setelah Anda memasukkan cukup banyak fakta dalam format yang disebutkan, Anda dapat menguji chatbot di simulator seperti yang dijelaskan pada langkah 10.

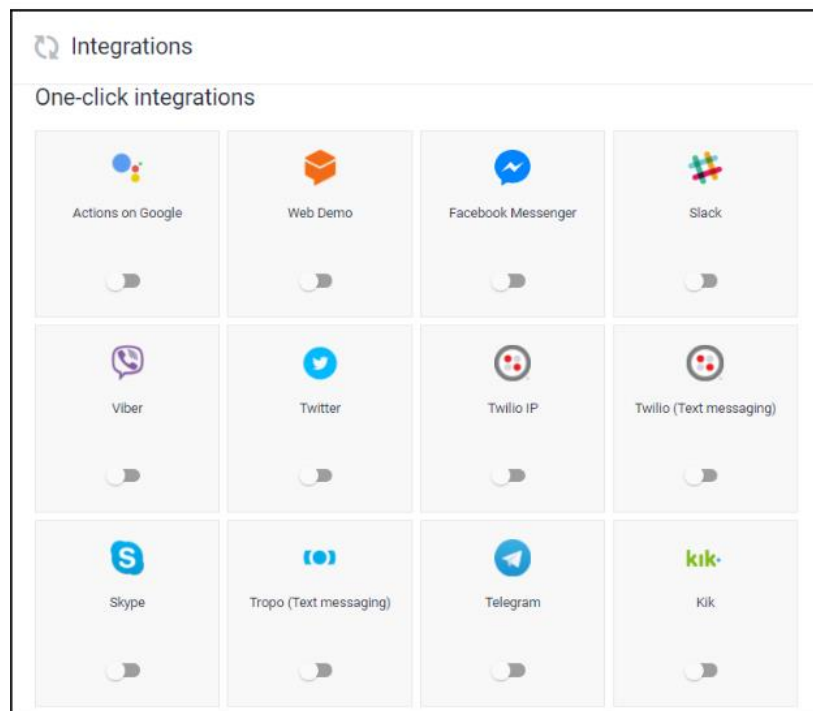
8.3 Menyebarkan Chatbot

Sekarang kita sudah memiliki chatbot, mari kita publikasikan pada platform di mana pengguna benar-benar dapat menggunakannya. Dialogflow memungkinkan Anda mengintegrasikan chatbot (yaitu agen) dengan banyak platform. Klik Integrasi untuk melihat semua platform yang tersedia:



Gambar 4.5 mengintegrasikan chatbot

Di bagian ini, kita akan mengeksplorasi dua integrasi platform: situs web dan Facebook. Kami akan mengeksplorasi lebih lanjut di bagian selanjutnya:

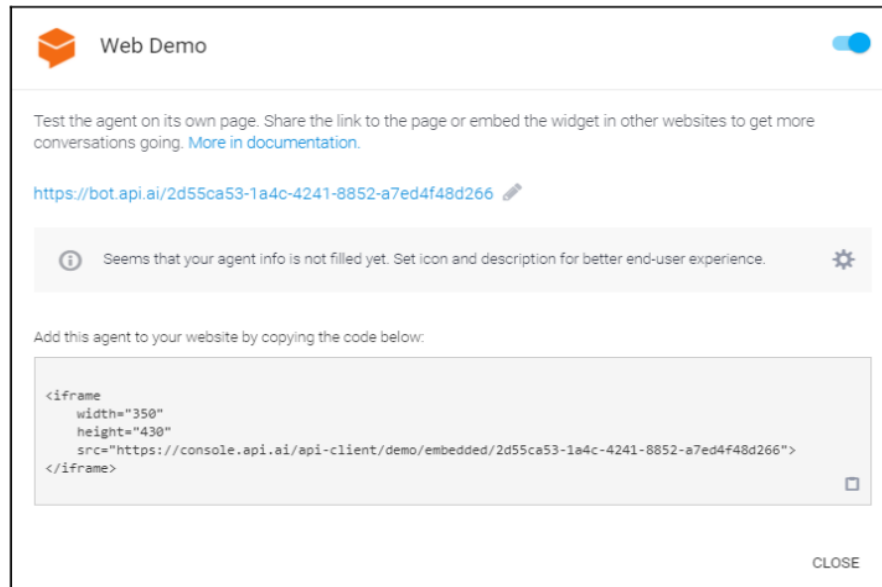


Gambar 8.6 mengintegrasikan ke beberapa social media

8.4 Integrasi Situs Web

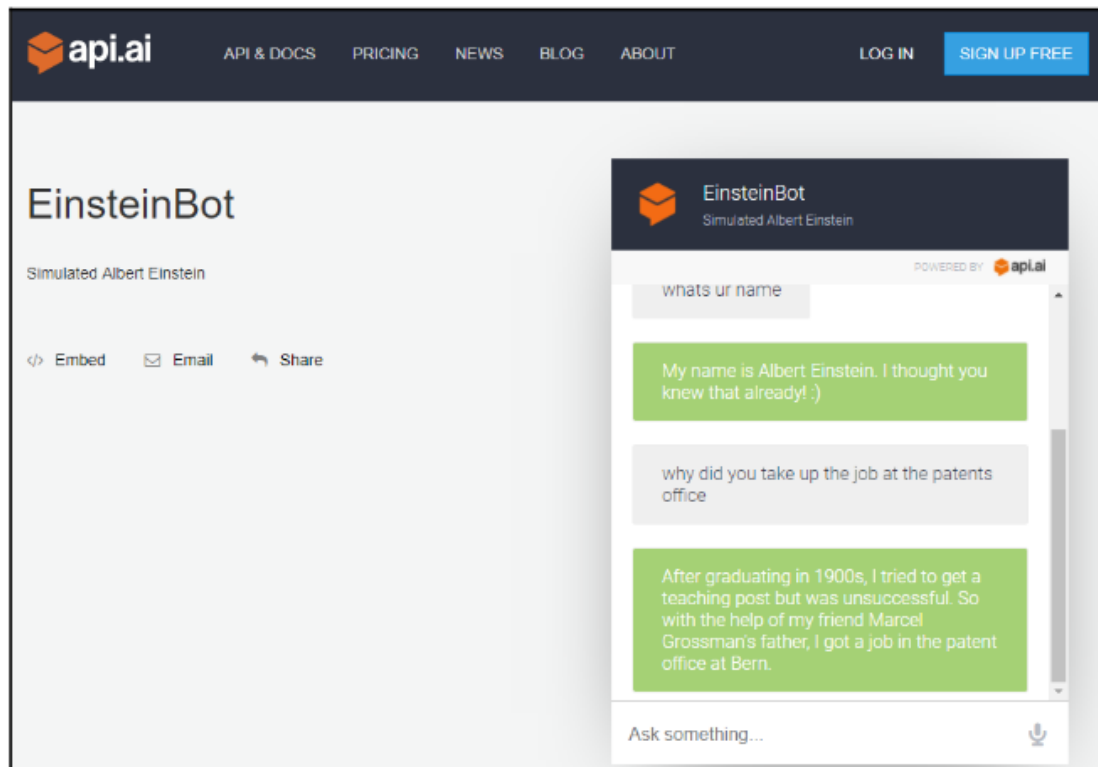
Integrasi situs web memungkinkan Anda menempatkan chatbot tersebut di situs web. Pengguna dapat berinteraksi dengan chatbot di situs web seperti yang mereka lakukan dengan agen obrolan langsung.

1. Pada halaman Integrasi, temukan platform Demo Web dan geser tombol dari mati ke aktif.
2. Klik Web Demo untuk membuka kotak dialog pengaturan berikut:



Gambar 8.7 tampilan demo

3. Klik URL **bot.dialogflow.com** untuk membuka contoh halaman web tempat Anda dapat menemukan bot pada widget obrolan yang tertanam di halaman tersebut. Coba ngobrol dengannya:



Gambar 8.8 tampilan bot.dialogflow

Anda dapat membagikan bot secara pribadi melalui email atau di media sosial dengan mengklik Email dan opsi Bagikan.

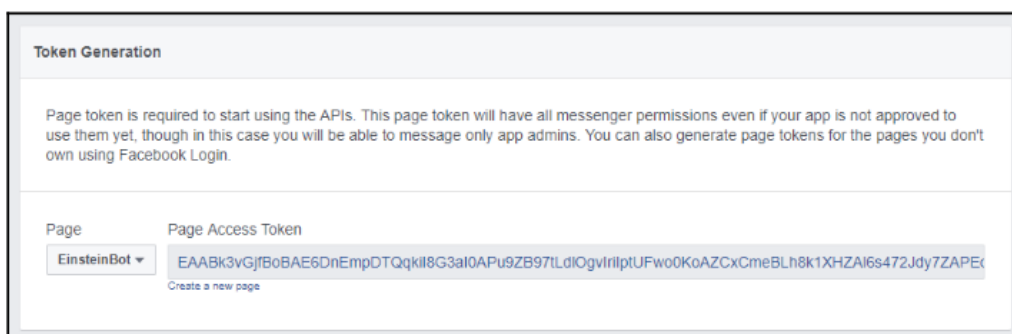
4. Widget obrolan juga dapat disematkan di situs web mana pun dengan menggunakan kode penyematkan iframe yang terdapat di kotak dialog pengaturan. Salin dan tempel kode ke halaman HTML dan coba di browser web:

```
<iFrame
  Width="350"
  Height="430"
  Src="https://console.api.ai/api-client/demo/embedded
    2d55ca53-1a4c-4241-8852-a7ed4f48d266">
</iframe>
```

Integrasi Facebook

Untuk mempublikasikan chatbot API.AI di Facebook Messenger, kita memerlukan Halaman Facebook untuk memulai. Kami juga memerlukan aplikasi Facebook Messenger untuk berlangganan halaman tersebut. Langkah-langkah membuat Halaman Facebook dan aplikasi Facebook Messenger disajikan secara rinci di bagian berjudul Publikasi di Facebook Messenger pada Bab 3, Mari Bicara Cuaca. Mari kita bahas langkah selanjutnya di sini:

- 1) Setelah membuat aplikasi Facebook Messenger, dapatkan Token Akses Halamannya. Anda bisa mendapatkannya di tab Pengaturan Messenger aplikasi:



Gambar 8.9 pengaturan akses halaman

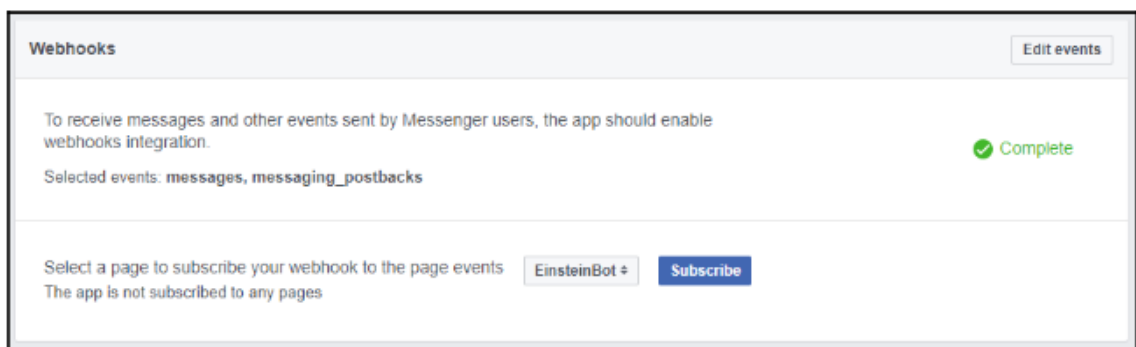
- 2) Di tab yang sama, klik Siapkan Webhook. Kotak dialog bernama Langganan Halaman Baru akan terbuka. Tetap buka di satu tab browser.
- 3) Di tab browser lain, dari halaman Integrasi API.AI, klik Facebook Messenger:

Gambar 8.10 integrasi dengan messenger

- 4) Salin URL di kolom teks URL Panggilan Balik. Ini adalah URL agen API.AI yang akan dihubungi dari aplikasi Messenger. Tempelkan ini di kolom teks URL Panggilan Balik pada kotak dialog Langganan Halaman Baru di aplikasi Facebook Messenger.
- 5) Ketikkan token verifikasi. Bisa apa saja asalkan cocok dengan yang ada di sisi lain. Mari kita ketik iam-einstein-bot.
- 6) Berlangganan pesan dan message_postbacks di bagian Bidang Langganan. Dan tunggu! Jangan klik Verifikasi dan Simpan dulu:

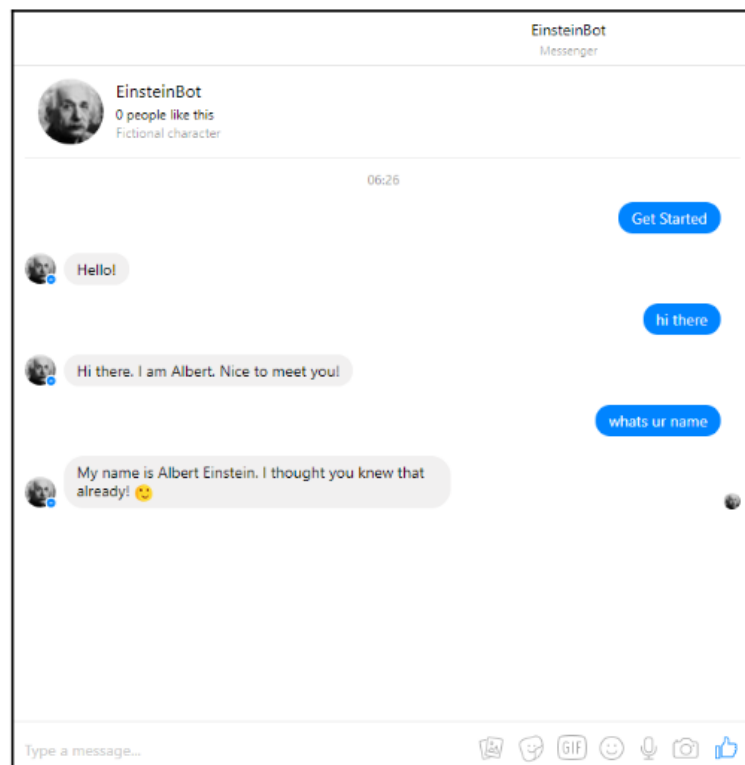
Gambar 8.11 tampilan untuk integrase dengan messenger

- 7) Di tab browser API.AI, Anda akan membuka pengaturan integrasi. Geser tombol ke hidup dari posisi mati di pojok kanan atas.
- 8) Ini akan memungkinkan Anda untuk mengedit pengaturan. Ketik Token Verifikasi. Ini harus sama dengan yang digunakan di pengaturan Aplikasi Facebook Messenger pada langkah 5.
- 9) Tempel Token Akses Halaman Dan Klik Mulai.
- 10) Sekarang kembali ke aplikasi Facebook Messenger dan klik Verifikasi dan Simpan. Ini akan menghubungkan aplikasi ke agen (chatbot).
- 11) Sekarang di halaman pengaturan Facebook Messenger, di bawah Webhook, pilih halaman Facebook yang benar untuk berlangganan aplikasi dan tekan Berlangganan:



Gambar 8.12 berlangganan webhook

Anda sekarang dapat membuka halaman Facebook, klik Kirim Pesan, dan ngobrol dengan chatbot:



Gambar 8.13 tampilan pengujian chatbot

Cemerlang! Sekarang Anda telah berhasil membuat chatbot di API.AI dan menerapkannya di dua platform: web dan Facebook Messenger. Selain platform tersebut, API.AI memungkinkan integrasi agen Anda dengan beberapa platform perpesanan populer seperti Slack, Skype, Cisco Spark, Viber, Kik, Telegram, dan bahkan Twitter.

Konteks

Sejauh ini, chatbot kami menjawab pertanyaan dalam satu kesempatan. Pengguna mengajukan pertanyaan dan chatbot menjawab. Bagaimana jika ada pertanyaan yang memerlukan percakapan lebih lanjut? Bagaimana jika chatbot perlu menanyakan pertanyaan lanjutan sebelum benar-benar dapat menjawab pertanyaan pengguna? Tugas-tugas seperti itu memerlukan konteks percakapan.

Mari kita cermati percakapan berikut ini:

Pengguna : mengapa Anda mengambil pekerjaan di kantor paten?

Bot : setelah lulus, saya mencoba untuk mendapatkan posisi mengajar tetapi tidak berhasil.

(jadi dengan bantuan ayah teman saya marcel grossman, saya mendapat pekerjaan di kantor paten di bern)

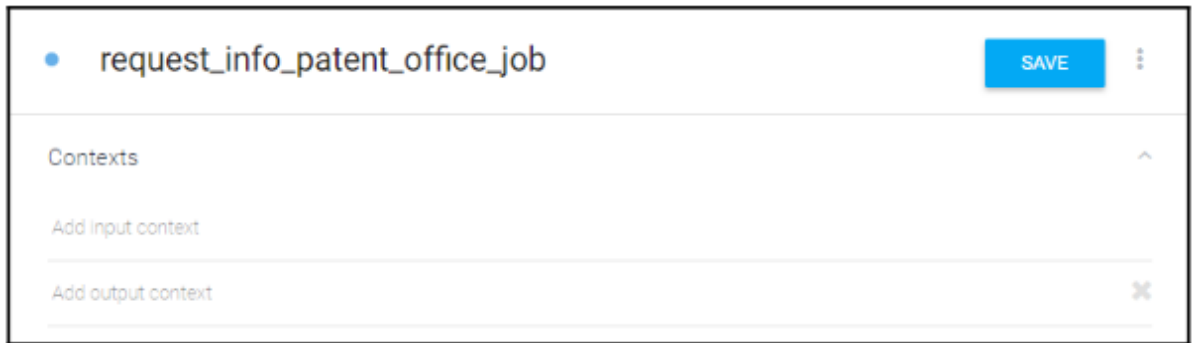
Pengguna : kapan Anda mendapatkan pekerjaan itu?

Pertanyaan kedua pengguna berkaitan dengan pertanyaan pertama dan jawaban dari chatbot. Ini bukanlah pertanyaan independen. Kata pekerjaan dapat berarti pekerjaan apa pun yang mungkin pernah dilakukan Einstein semasa hidupnya, namun sesuai dengan konteksnya, jelas apa yang dibicarakan pengguna. Oleh karena itu, chatbot perlu mempertimbangkan hal ini untuk memahami apa yang sebenarnya dimaksud pengguna.

Di API.AI, kita dapat mengaktifkan agen untuk menyimpan catatan konteks percakapan menggunakan tab Konteks saat membuat maksud. Setiap maksud harus menentukan konteks masukan dan keluaran. Konteks masukan menentukan konteks di mana maksud akan dipertimbangkan. Konteks keluaran adalah apa yang ditetapkan ketika maksud cocok dan berhasil dieksekusi dengan ucapan respons.

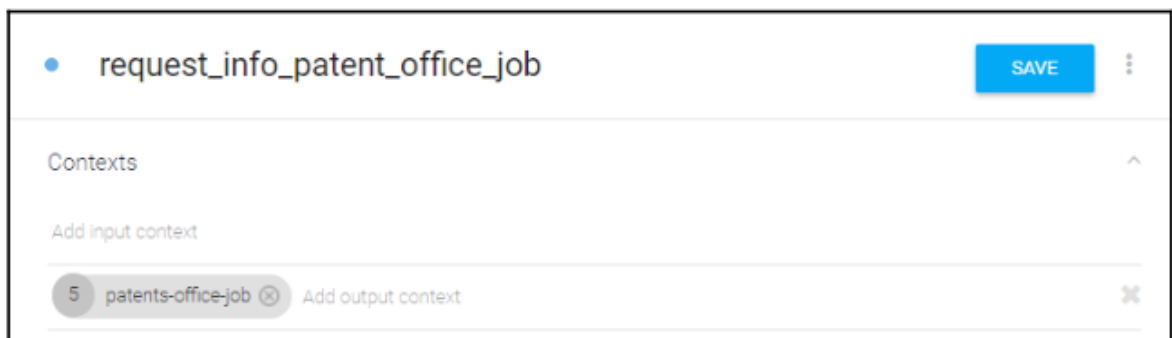
Agar berhasil menjalankan percakapan sebelumnya, agen perlu mencatat informasi kontekstual dalam konteks keluaran untuk maksud `request_info_patent_job` yang mana pertanyaan tentang pekerjaan kantor paten Einstein akan terjawab. Hal ini kemudian akan berfungsi sebagai konteks masukan untuk pertanyaan lanjutan seperti, "Kapan Anda mendapatkan pekerjaan itu?" atau, "Berapa bayaran pekerjaan itu?" dan masih banyak lagi. Inilah cara Anda melakukannya:

- 1) Buka maksud `request_info_patent_job`. Klik tab Konteks. Anda akan melihat tab diperluas seperti yang ditunjukkan pada tangkapan layar berikut:



Gambar 8.14 tangkapan layar `request_info_patent_job`

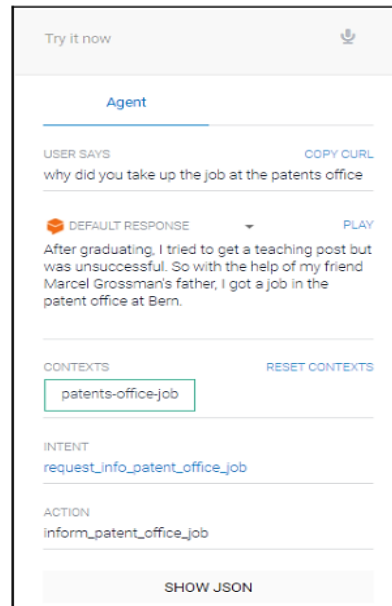
- 2) Ketikkan nama item kontekstual yang Anda ingin agen simpan di bidang konteks keluaran. Karena kita ingin agen mengetahui bahwa pertanyaan pengguna adalah tentang pekerjaan kantor paten, mari kita ketikkan nama item kontekstual (yaitu, pekerjaan kantor paten) dalam konteks keluaran:



Gambar 8.15 kontekstual pemilihan konten

Terlampir pada setiap item kontekstual adalah jangka waktu konteks tersebut. Ini diatur ke 5 secara default. Ini berkurang satu menjadi nol dengan setiap ucapan pengguna yang masuk. Jika sudah mencapai nol, otomatis akan terhapus dari konteks percakapan.

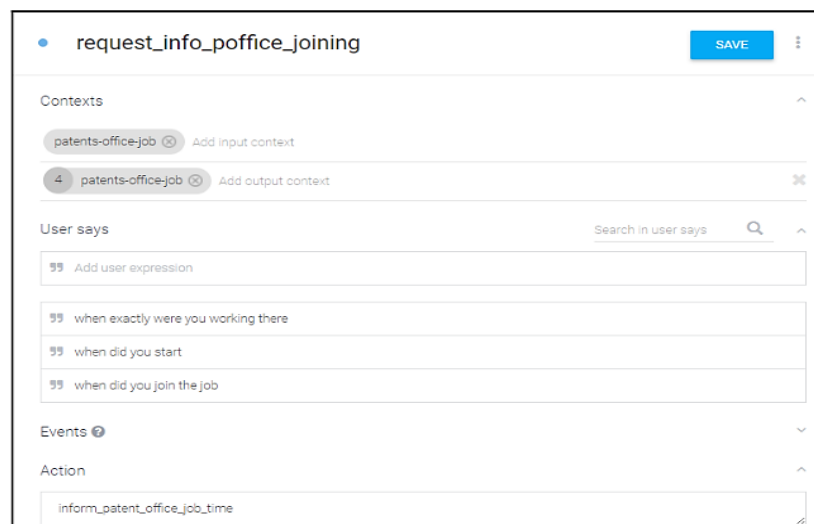
- 3) Kami telah menentukan konteks keluaran untuk maksud tersebut. Jadi, jika agen mengenali maksud ini dari ucapan pengguna, agen akan merespons dengan tepat dan juga mengatur konteks keluaran. Mari kita coba ini di simulator:



Gambar 8.16 simulasi kontekstual keluaran

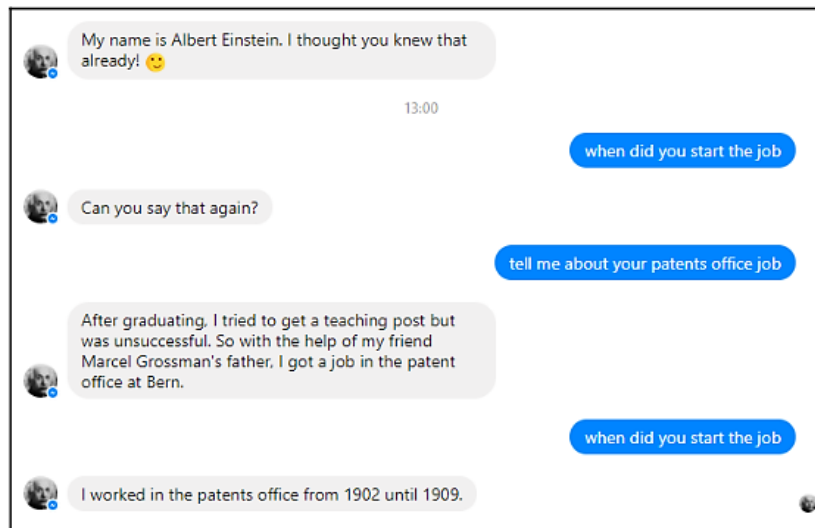
Tangkapan layar sebelumnya menunjukkan bagaimana konteksnya kini berubah dari tidak ada menjadi pekerjaan kantor paten. Setiap ucapan baru dari pengguna kini akan diproses dalam konteks ini.

- 4) Untuk menggunakan konteks, item kontekstual perlu ditentukan sebagai konteks masukan dalam maksud pengguna berikutnya. Mari kita coba dengan membuat maksud pengguna `request_info_poffice_joining` baru. Niat ini akan diketahui ketika pengguna menanyakan kapan Anda memulai pekerjaan Anda di kantor paten atau yang serupa. Namun meskipun pengguna menanyakan kapan Anda memulai pekerjaan yang tidak menyebutkan kantor paten, kita seharusnya dapat mengenalinya selama percakapan tentang pekerjaan kantor paten. Untuk melakukan itu, kami akan menyebutkan isi percakapan dalam konteks masukan:



Gambar 8.17 konteks masukan item

Pada tangkapan layar sebelumnya, Anda dapat melihat cara item kontekstual kantor paten disebutkan di bidang konteks masukan. Jadi, ucapan seperti kapan Anda memulai dalam konteks pekerjaan kantor paten akan dikenali sebagai `request_info_poffice_joining`. Ucapan yang sama dalam konteks lain tidak akan dikenali sebagai maksud `request_info_poffice_joining`:



Gambar 8.18 pengujian untuk ucapan yang sama

Tangkapan layar sebelumnya menunjukkan bagaimana ucapan yang sama pada titik percakapan yang berbeda memberikan hasil yang berbeda karena agen kini memproses masukan dari pengguna secara kontekstual.

Umur item kontekstual dalam konteks keluaran dapat ditambah atau dikurangi. Jika item kontekstual tidak lagi diperlukan, item tersebut dapat dihapus dengan mengatur umur ke nol. Ini bisa menjadi peretasan yang berguna ketika pengguna berterima kasih kepada bot setelah menyelesaikan suatu tugas. Dalam kasus seperti itu, chatbot dapat dibuat untuk mengatur ulang konteks percakapan.

Item yang lebih kontekstual dapat ditambahkan di kolom input dan output jika diperlukan. Ini mungkin berguna dalam menciptakan percakapan yang lebih kompleks. Meskipun menggunakan item kontekstual untuk mendorong percakapan adalah ide yang bagus, kami merekomendasikan penggunaan item kontekstual yang disebut global untuk membuat dan mengelola item kontekstual global di mana parameter global dapat disimpan dan digunakan. Item ini akan tersedia untuk semua maksud dalam masukannya dan diteruskan ke konteks keluaran juga.

8.5 Entitas

Entitas menambah lapisan kompleksitas pada percakapan bahasa alami sehingga membuat chatbot lebih kuat dan fleksibel. Objek apa pun yang menarik dalam percakapan chatbot dengan pengguna dapat dianggap sebagai entitas. Misalnya, dalam chatbot yang

menyalurkan informasi cuaca, kota besar dan kecil dapat dianggap sebagai entitas. Entitas lain yang diminati adalah jenis informasi yang diinginkan pengguna: suhu, kemungkinan hujan, kecepatan angin, dan sebagainya. Dalam hal suhu, metrik yang digunakan, baik Fahrenheit atau Celsius, juga dapat diperlakukan sebagai entitas.

Entitas dapat diatur berdasarkan nama dan nilai parameter. Misalnya, di domain cuaca, Anda dapat menambahkan untuk meminta laporan cuaca spesifik kota dengan cara berikut:

Intent = request_weather
City_name=London_gb

Format sebelumnya setara dengan pertanyaan pengguna, "Seperti apa cuaca di London?" Jadi, daripada menggunakan satu maksud per kota (misalnya, *request_weather_london*), kita dapat mengatur permintaan pengguna dengan lebih baik berdasarkan maksud dan entitas. Hal ini juga membuat pengenalan maksud pengguna lebih mudah dikelola.

Mari kita lihat beberapa contoh lainnya:

Intent = change_metric
Metric= celcius

Contoh sebelumnya mewakili ucapan seperti, "Berapa satuan Celsius?" dan, "Bisakah Anda melaporkan dalam skala Celsius?"

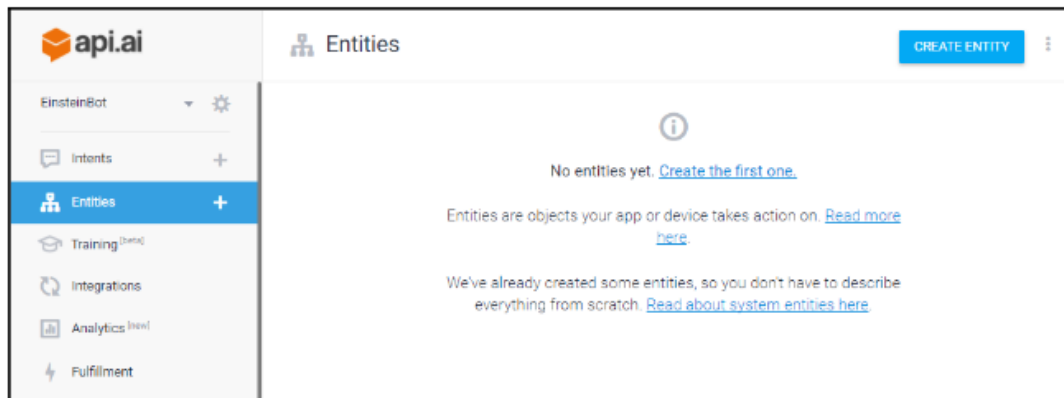
Dalam kasus bot Einstein, kita dapat merepresentasikan karya-karya besarnya, seperti Teori Relativitas Khusus, Teori Relativitas Umum, dan Gerakan Brown sebagai entitas. Entitas tersebut dapat digunakan dalam maksud terkait seperti *request_info_on_work* dan *request_publications*. Ambil contoh, spesifikasi maksud berikut:

Intent = request_info_on_work
Work = special_theory_of_relativity

Maksud sebelumnya dapat berarti bahwa pengguna berkata, "Ceritakan tentang pekerjaan Anda pada teori relativitas khusus" atau sesuatu yang serupa.

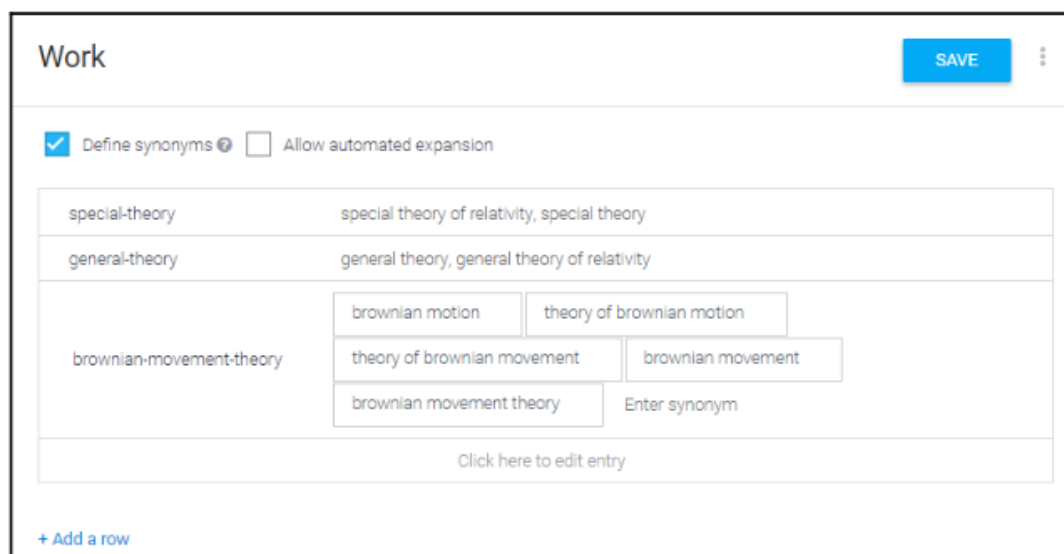
Untuk memanfaatkan entitas, kita perlu menghitungnya di agen. Untuk melakukannya, ikuti langkah-langkah berikut:

1. Klik opsi Entitas di menu utama. Anda akan menemukannya di bawah opsi Intents yang digunakan untuk membuat maksud pengguna:



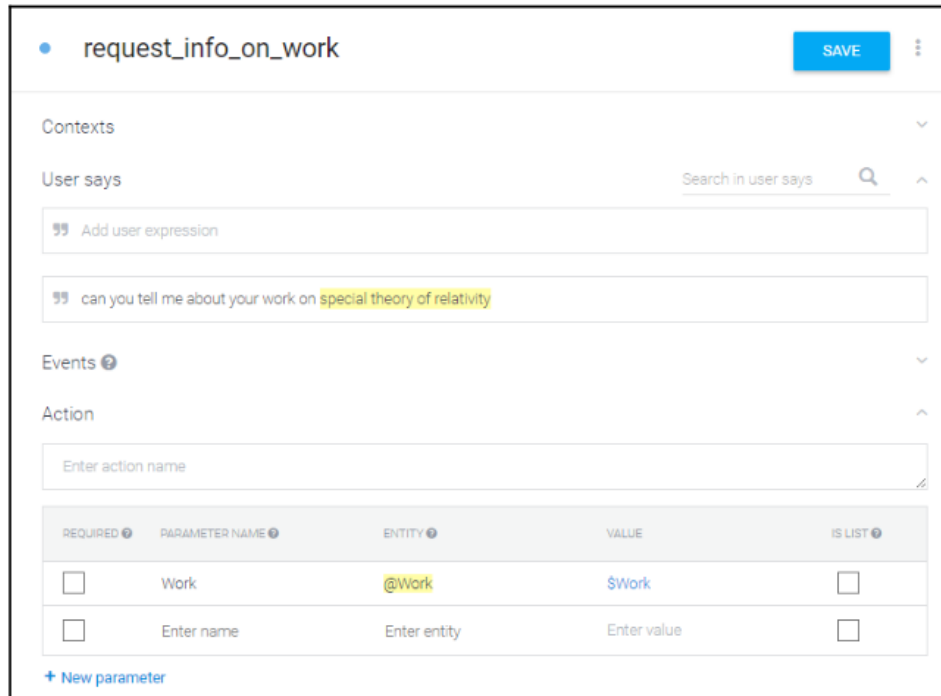
Gambar 8.19 membuat entitas pada api.ai

2. Klik BUAT ENTITAS.
3. Beri nama tipenya. Dalam domain cuaca, London, Paris, atau New York mungkin merupakan entitas sebenarnya tetapi mereka dapat dikelompokkan berdasarkan nama tipe yang disebut Kota. Dalam kasus bot Einstein, sebut saja Work karena entitas sebenarnya adalah karyanya.
4. Ada tabel dengan dua kolom. Di sebelah kiri, Anda harus mengisi nilai referensi untuk entitas dan di sebelah kanan, Anda akan mencantumkan sinonim dari entitas tersebut. Untuk menjelaskan hal ini, mari kita kembali ke contoh kota. Setiap kota akan memiliki nilai referensi (misalnya, newyork-us) untuk mewakilinya secara internal dengan cara yang jelas. Setiap nilai referensi dapat dinyatakan dalam bentuk kata atau frasa dalam bahasa alami. Dalam hal ini, New York bisa disebut “the Big Apple”, “New York City”, “The City of New York”, dan seterusnya. Ini perlu dicantumkan sebagai sinonim. Untuk bot kita, mari kita tambahkan tiga karya Einstein—Teori Relativitas Khusus, Teori Relativitas Umum, dan Teori Gerakan Brown. Untuk masing-masing karya ini, mari kita tambahkan juga sinonim:



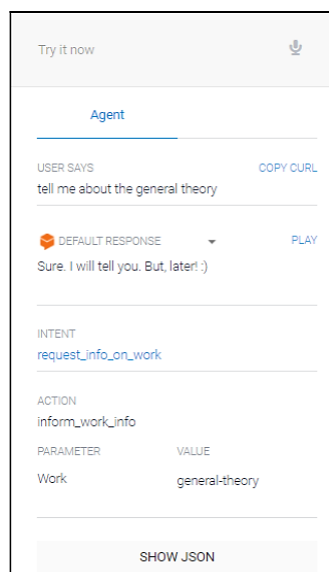
Gambar 8.20 menambahkan sinonim

5. Setelah membuat entitas, sekarang mari kita manfaatkan entitas tersebut dengan membuat maksud yang akan menggunakannya. Mari kita buat maksud yang disebut `request_info_on_work`:



Gambar 8.21 memanfaatkan entitas

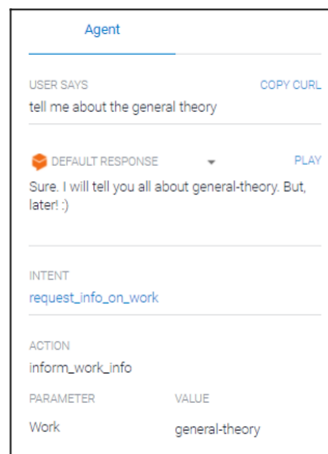
Amati pada tangkapan layar sebelumnya bagaimana kata teori relativitas khusus disorot. Hal ini dilakukan secara otomatis oleh editor API.AI berdasarkan daftar entitas yang kita buat. Ini mengidentifikasi entitas dalam contoh ucapan yang kami berikan untuk setiap maksud. Ia kemudian menambahkan entitas yang teridentifikasi tersebut ke daftar parameter:



Gambar 8.22 menambahkan entitas ke daftar parameter

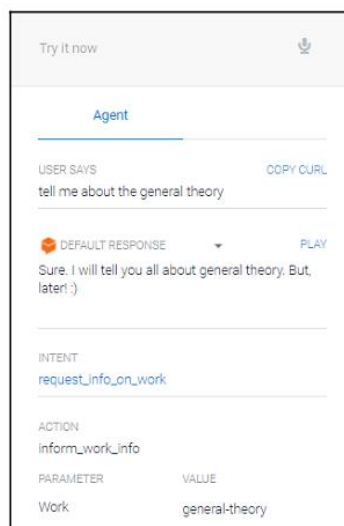
6. Coba ini di simulator dan amati keluaran dari agen. Seperti yang ditunjukkan pada tangkapan layar sebelumnya, Anda akan melihat bahwa agen telah mengambil nama karya dan telah membuat parameter dengan nama (yaitu, Karya) dan nilai (yaitu, teori umum). Di bidang nilai, Anda akan menemukan nilai referensi entitas, apa pun sinonim yang digunakan dalam ucapan.

Sekarang, setelah mengekstrak parameter dan nilai, mari kita cari tahu cara menggunakannya dalam ucapan respons. Parameter tersebut dapat direferensikan dalam ucapan respon menggunakan notasi \$. Dalam hal ini, nilai referensi parameter Work dapat diperoleh dengan menggunakan \$Work. Coba yang Pasti. Saya akan memberi tahu Anda semua tentang \$Work. Tapi nanti! 😊 respon ucapan dan lihat apa yang terjadi:



Gambar 8.23 respon entitas

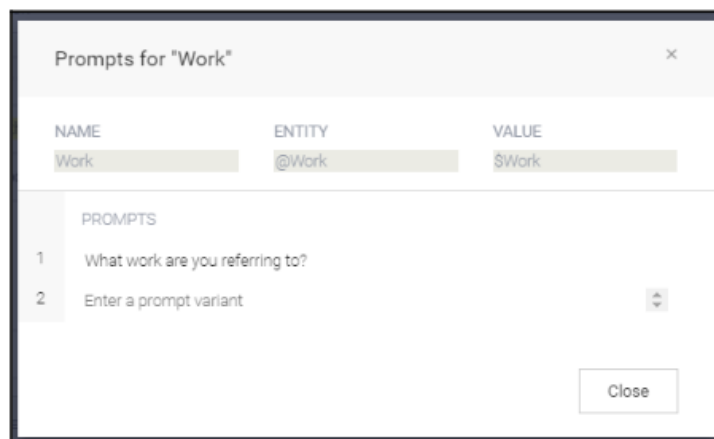
7. Karena nilai referensi kita mempunyai tanda hubung di dalamnya, nilai tersebut tidak sesuai dengan ucapannya. Jadi kita bisa mencoba menggunakan sinonim yang digunakan pengguna. Untuk melakukannya, ubah \$Work menjadi \$Work.original. Simpan niatnya dan coba simulatornya:



Gambar 8.24 menyimpan dialog

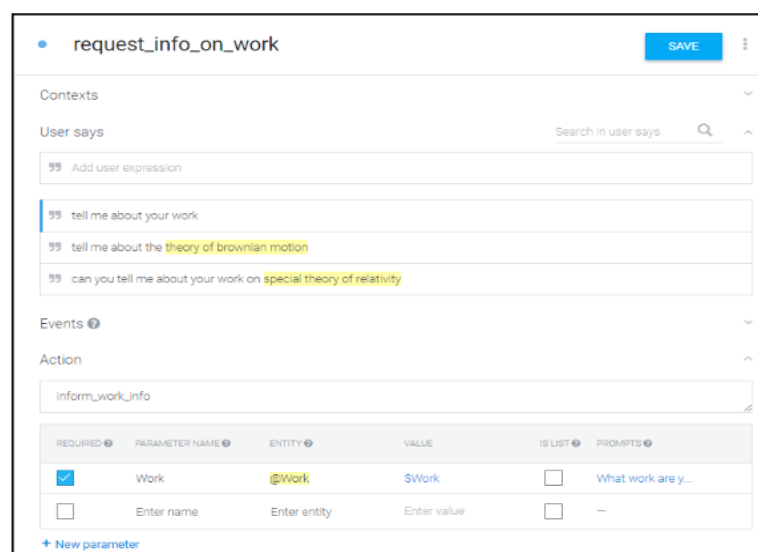
8. Cobalah ucapan serupa namun spesifik sebagian, seperti ceritakan tentang pekerjaan Anda. Ini memetakan ke maksud tetapi tidak memiliki parameter yang diperlukan. Bagaimana agen akan menangani ucapan ini? Jika Anda mencobanya, Anda akan melihat bahwa niatnya telah dikenali tetapi responsnya kosong. Hal ini karena agen tidak dapat menemukan nilai \$Work.original.

Untuk memastikan bahwa agen mendapatkan nilai untuk parameter Pekerjaan, Anda harus menjadikan parameter tersebut sebagai parameter yang diperlukan. Caranya dengan mencentang kotak REQUIRED untuk parameternya. Klik Tentukan petunjuknya untuk memberikan perintah kepada agen untuk digunakan jika pengguna lupa menyebutkan nilai parameter dalam ucapannya:



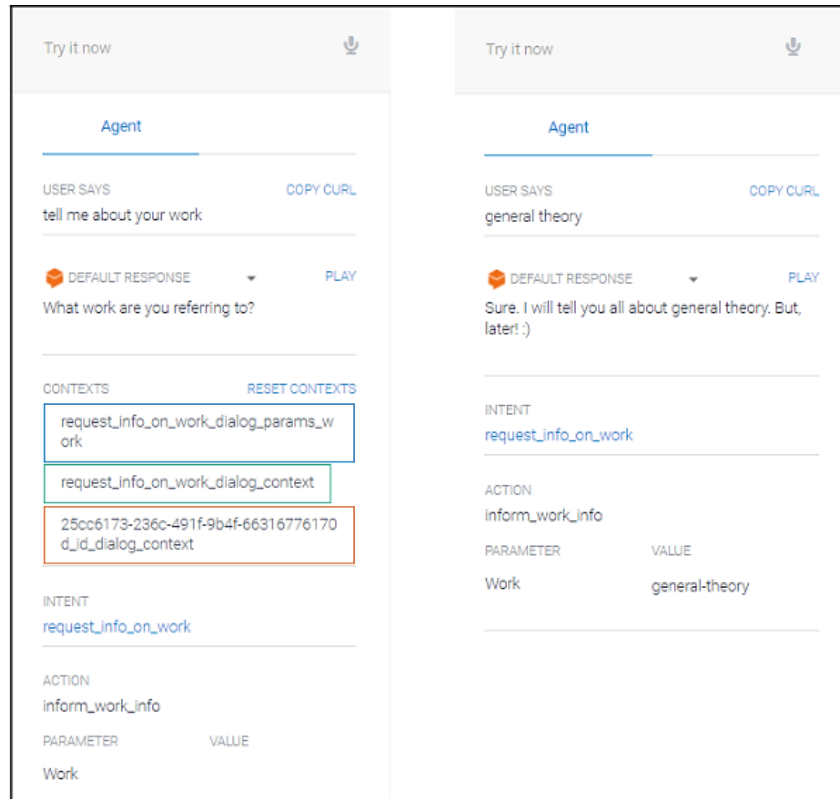
Gambar 8.25 mencoba entitas pencarian

Klik SIMPAN setelah Anda menentukan entitas, mewajibkannya, dan menentukan petunjuknya:



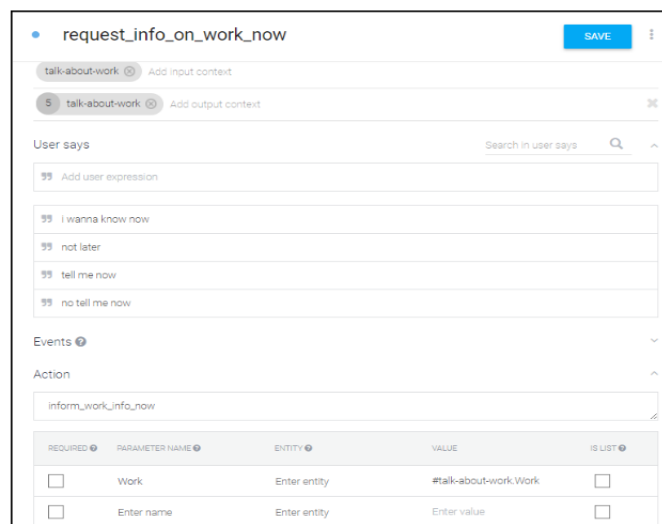
Gambar 8.26 menentukan entitas

9. Coba simulator sekali lagi dengan ceritakan tentang ucapan pekerjaan Anda. Anda akan menemukan agen meminta Anda kembali dengan respon cepat untuk memberi nama karya tersebut. Setelah ini, agen akan merespons dengan tepat:



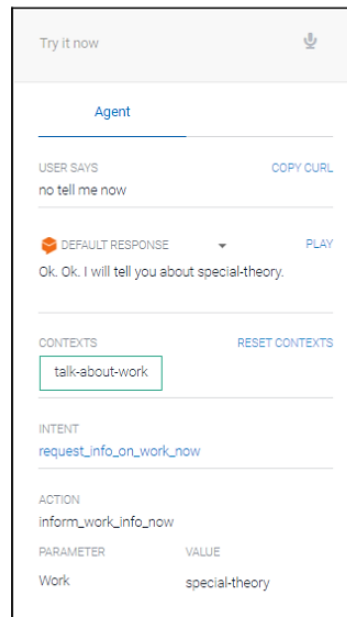
Gambar 8.27 pengujian untuk ucapan yang sama

10. Entitas dapat disimpan dalam konteks dan digunakan di seluruh ucapan. Untuk melakukan ini, mari kita tambahkan item kontekstual `talk_about_work` dalam konteks output dari maksud `request_info_on_work`. Mari kita tambahkan juga maksud `request_info_on_work_now` lainnya dengan `talk_about_work` sebagai konteks masukan:



Gambar 8.28 menyimpan entitas dalam konteks

Entitas diteruskan dari satu maksud ke maksud lainnya melalui item kontekstual. Kita dapat mengakses entitas dari percakapan sebelumnya dengan menggunakan notasi #. Misalnya, amati bagaimana kita mengisi nilai parameter Pekerjaan menggunakan nilai #talk-about-work.Work. Ini kemudian dapat digunakan dalam ucapan respons sebagai \$Work (misalnya, Baiklah. Saya akan memberi tahu Anda tentang \$Work):

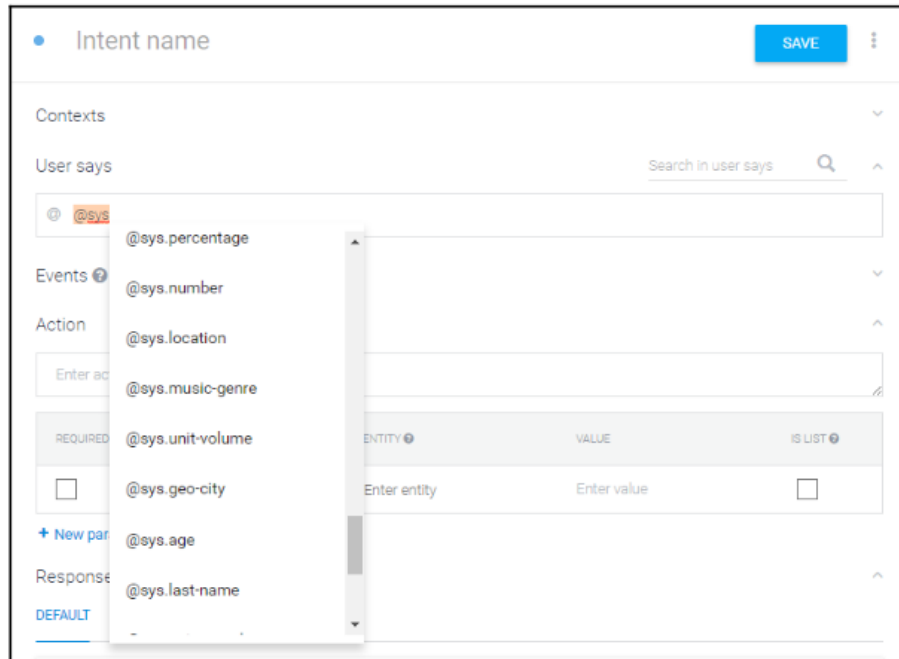


Gambar 8.29 tampilan entitas dari suatu maksud

Kita akan melihat cara menggunakan entitas untuk melakukan pemrosesan lanjutan di backend menjelang akhir bab ini.

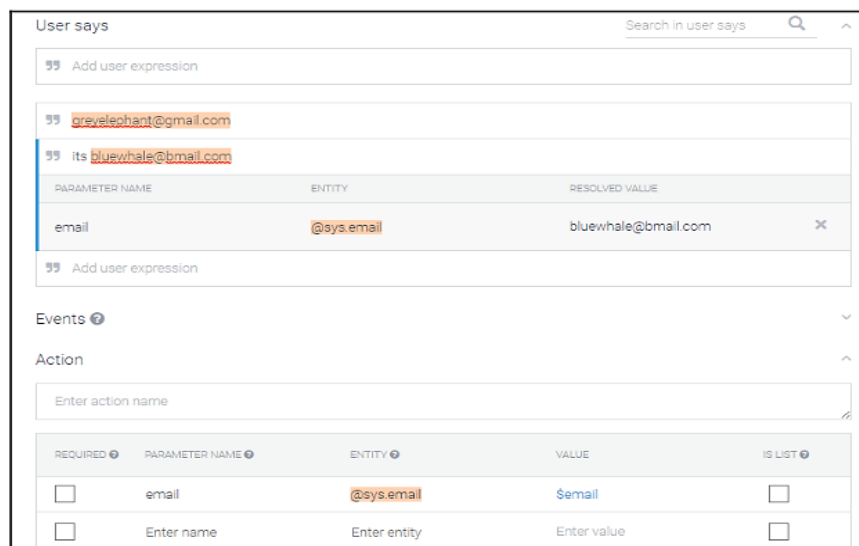
Entitas sistem

Selain entitas yang kita buat (yaitu entitas pengembang) menggunakan pendekatan sebelumnya, ada juga entitas sistem yang telah ditentukan sebelumnya. Ini telah dirancang untuk menangkap entitas yang umum pada sebagian besar domain dan tugas percakapan. Konsep seperti tanggal, nama pribadi, nama kota, nama negara, alamat email, dan nomor telepon dapat dikenali dalam ucapan pengguna menggunakan entitas sistem:



Gambar 8.30 tampilan mengakses menggunakan @sys

Tangkapan layar sebelumnya menunjukkan bagaimana Anda dapat mengaksesnya dengan menggunakan kualifikasi @sys saat memberikan contoh ucapan. Misalnya, Anda ingin mendapatkan alamat email pengguna. Untuk ini, Anda dapat mengatur maksud pengguna dalam konteks percakapan seperti `inform_user_email`. Contoh pengguna ini harus memiliki contoh ucapan, seperti "bluewhale@bmail.com" atau "its bluewhale@bmail.com", yang mana pengguna memberikan emailnya kepada chatbot. Anda akan melihat bahwa alamat email dikenali sebagai @sys.email. Jika tidak dikenali secara otomatis, Anda dapat menentukan bahwa itu adalah email dengan memilihnya dan memilih @sys.email. Ini memberi tahu agen bahwa semua ucapan dengan alamat email yang valid harus dikenali sebagai maksud `inform_user_email`:



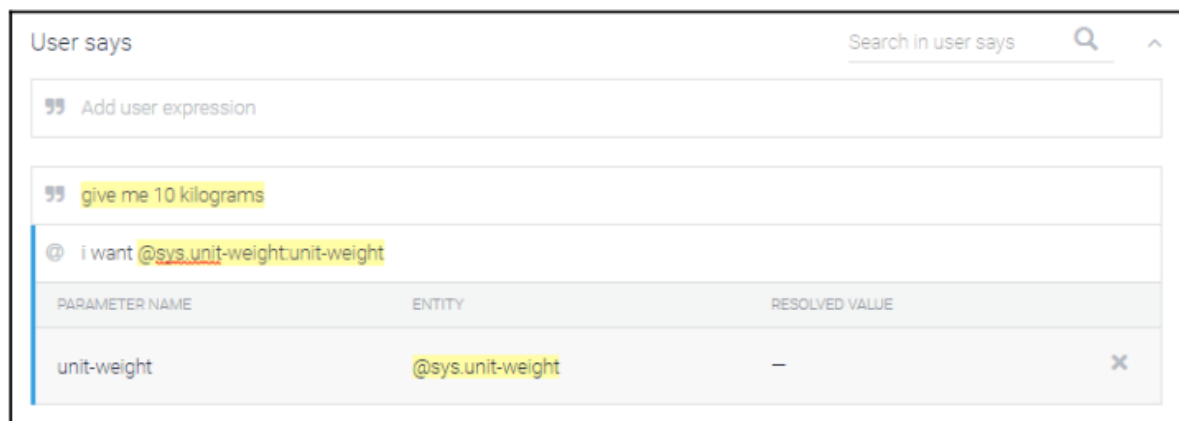
Gambar 8.31 memberikan masukan email pada @sys

Setelah dikenali, entitas akan diteruskan sebagai parameter untuk ditambahkan ke maksud. Entitas sistem seperti @sys.geo-city dan @sys.geo-country dapat digunakan untuk mengenali nama kota dan nama negara dalam ucapan pengguna tanpa harus menentukan semua nama sebagai entitas pengembang.

Untuk mengenali unsur-unsur gabungan dalam ujaran bahasa alami seperti penyebutan besaran (berat, panjang, dan sebagainya), kita dapat menggunakan entitas sistem gabungan. @sys.unit-weight:unit-weight dapat digunakan untuk mengenali ucapan seperti 10 kg, 15 kilo, dan 20 kilogram. Ini akan diselesaikan ke dalam parameter berikut untuk tindakan lebih lanjut:

```
{“amount”:10,“unit”:“kg”}
```

Tipe entitas tidak dapat ditentukan secara langsung dalam contoh ucapan. Misalnya, @sys.email-nya tidak dapat ditentukan sebagai contoh ucapan. Sebaliknya, hal ini perlu disediakan sebagai templat. Untuk memberikan contoh templat ke maksud, alihkan tanda kutip ganda ke tanda @.



Gambar 8.32 mengalihkan tanda kutip menjadi @

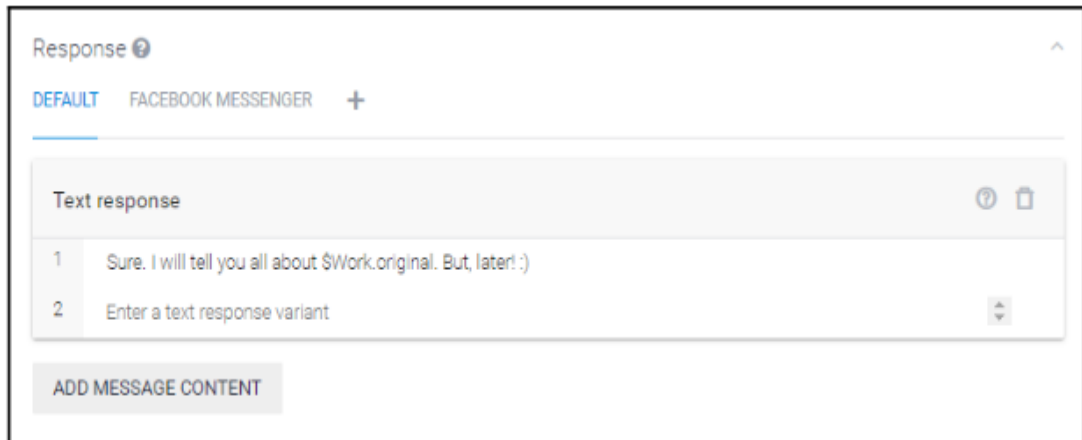
Pada tangkapan layar sebelumnya, amati bahwa ada contoh ucapan (yaitu, beri saya 10 kilogram) dan contoh templat yang saya inginkan @sys.unit-weight:unit-weight. Jelajahi entitas sistem lebih lanjut untuk melihat bagaimana Anda dapat menggunakan entitas yang telah ditentukan sebelumnya di chatbot Anda.

Format respons yang kaya

Selain tanggapan teks, Anda juga dapat memberikan jenis pesan kaya seperti kartu, carousel, dan yang kami gunakan di Bab 2, Pemandu Wisata untuk Kota Anda, saat membuat bot Facebook Messenger untuk cuaca. Jenis respons yang dapat kita tambahkan ke agen bergantung pada integrasi yang diaktifkannya. Setiap integrasi memungkinkan serangkaian jenis respons yang berbeda.

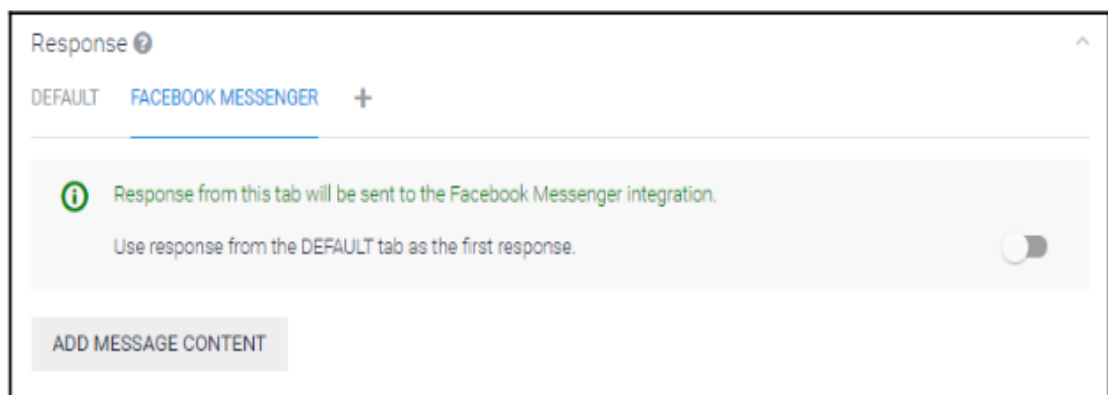
Karena kami telah mengintegrasikan Facebook Messenger dengan bot Einstein, kami dapat menambahkan konten pesan khusus untuk platform tersebut. Untuk melakukannya, ikuti langkah-langkah berikut:

1. Pilih maksud pengguna mana pun yang ingin Anda tambahkan format respons kaya.
2. Gulir ke bawah ke tab Respons:



Gambar 8.33 tap respons

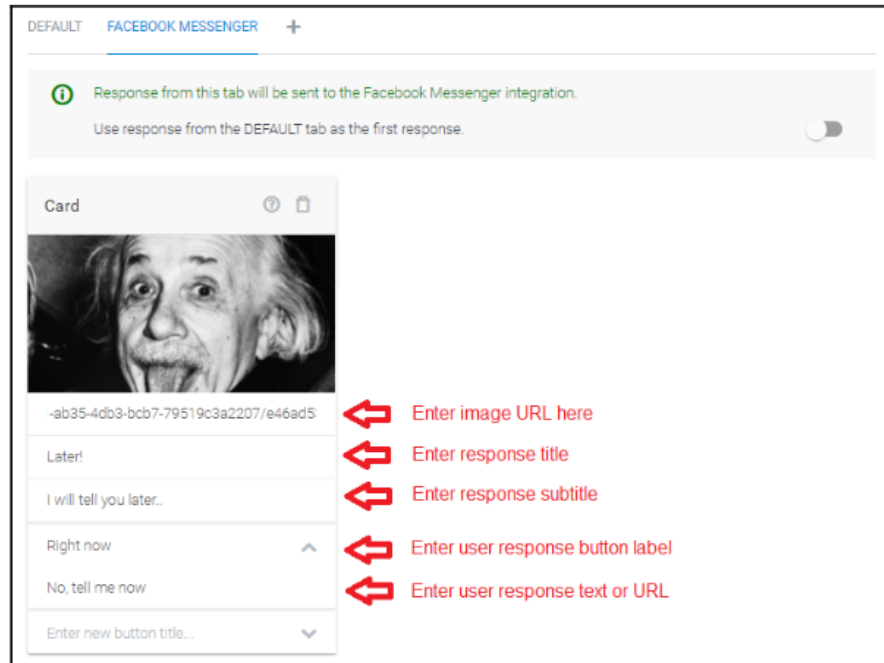
3. Di samping DEFAULT, akan ada tab untuk integrasi yang telah diaktifkan (misalnya, FACEBOOK MESSENGER). Jika Anda tidak melihat integrasi Anda, Anda dapat menambahkannya dengan mengklik menu +.
4. Pada setiap integrasi, Anda akan melihat hal berikut:



Gambar 8.34 tap respon terintegrasi

5. Jika Anda ingin respons default muncul pertama kali pada platform integrasi, aktifkan tombol tersebut.
6. Klik TAMBAHKAN KONTEN PESAN dan pilih jenis tanggapan. Dalam integrasi Facebook Messenger, Anda akan menemukan jenis respons seperti pesan teks, gambar, kartu, balasan cepat, dan muatan khusus.
7. Masukkan informasi spesifik format respons.

Berikut ini contoh respon jenis kartu untuk integrasi Facebook Messenger:



Gambar 8.35 contoh kartu ucapan pada messenger

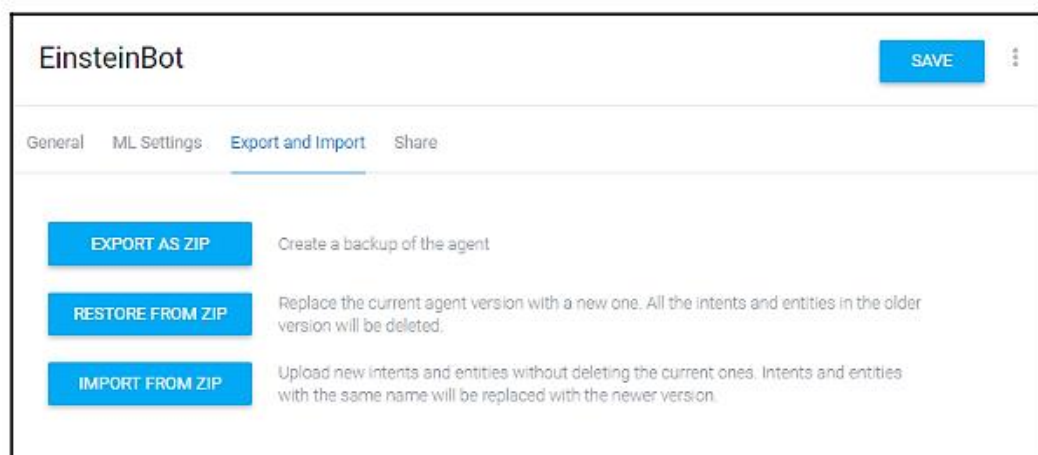
Mengklik tombol Sekarang akan mengirimkan pesan teks Tidak, beri tahu saya sekarang kembali ke agen sebagai respons pengguna terhadap bot.

Sekarang setelah Anda memiliki gagasan tentang cara membuat respons Anda lebih kaya, lanjutkan dan mainkan format berbeda yang tersedia untuk integrasi Anda.

8.6 Agen Impor Dan Ekspor

Semua agen di API.AI dapat diekspor dan dicadangkan. Opsi ini juga memungkinkan Anda memindahkan agen dari satu akun API.AI ke akun lainnya. Mengekspor, memulihkan, dan mengimpor agen inilah cara Anda mengekspor agen.

1. Klik ikon Pengaturan di samping nama agen di menu sebelah kiri.
2. Klik Ekspor dan Impor:



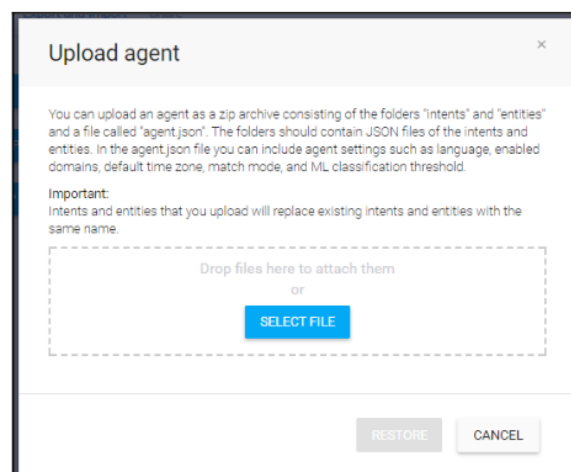
Gambar 8.36 tampilan Einstein bot

3. Klik EKSPOR SEBAGAI ZIP.

File ZIP yang diunduh akan diatur berdasarkan pengaturan agen, entitas, dan maksud dalam format file JSON. Berikut format JSON setting agen bot Einstein kita:

```
{
  "description" : "Simulated Albert Einstein",
  "language":"en"
  "googleAssistant" :{
    "googleAssistantCompatible":false,
    "project":"einsteinbot-dce04",
    "welcomeIntentSignRequired":false,
    "startIntents": [ ],
    "systemintents" : [ ],
    "endIntentIds" : [ ] ,
    "oAuthLinking" : [
      "required" : false
      "grantType" : "AUTH_CODE_GRANT"
    ] ,
    "voiceType" : "MALE_1",
    "capabilities": [ ],
    "protocolVersion" : "V2"
  },
  "defaultTimezone":"Africa/Casablanca",
  "webhook": {
    "avaible":false,
    "useForDomains":false
  },
  "isPrivate":true,
  "customClassifierMode":"use.after"
  "mlMinConfidence" : 0,3
}
```

Anda juga dapat memulihkan agen dari file cadangan menggunakan opsi RESTORE FROM ZIP. Semua maksud dan entitas akan dihapus dan dipulihkan dari file ZIP:



Gambar 8.37 mengupload entitas

Jika Anda ingin memperluas agen Anda dari file ZIP, hal itu dapat dilakukan menggunakan opsi IMPOR. Hal ini memungkinkan Anda menambahkan maksud dan entitas tambahan ke agen yang sudah ada. Semua entitas dan maksud di agen dengan nama yang sama dengan yang ada di file akan diganti dengan yang ada di file ZIP.

Maksud dan entitas dapat diekspor secara terpisah dan satu per satu tanpa harus mengekspor agen seluruhnya. Hal ini dapat dilakukan dengan mengklik ikon cloud_download yang tercantum pada setiap maksud dan entitas:



Gambar 8.38 Mengekspor dan mengimpor maksud dan entitas

Berikut ini contoh maksud yang diunduh dari bot Einstein:

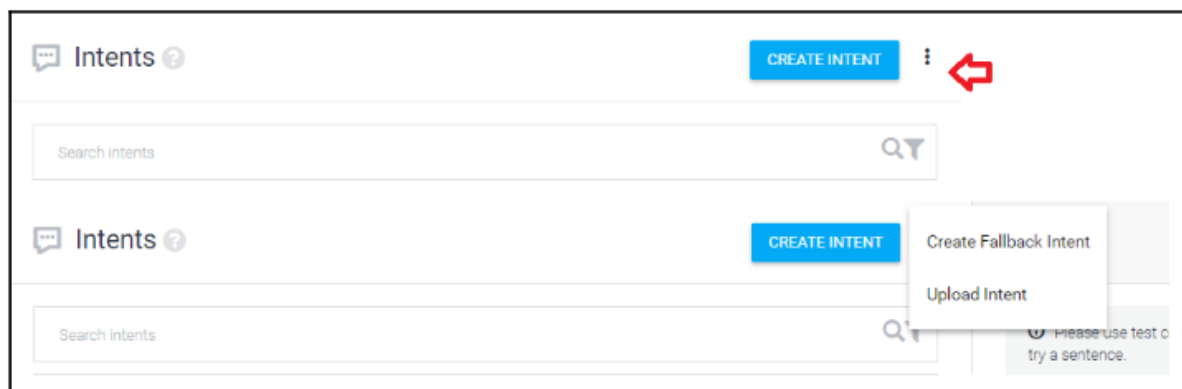
```
{
  "userSays": [
    {
      "id": "aa492d7b-6119-4d4c-a626-03aea6868882",
      "data": [
        {
          "text": "what was your job at the patent office"
        }
      ],
      "isTemplate": false,
      "count": 0
    },
    {
      "id": "f8a43750-c215-4750-aac8-1a0f977611d8",
      "data": [
        {
          "text": "why did you take up the job at the patent office"
        }
      ],
      "isTemplate": false,
      "count": 0
    },
    {
      "id": "2b48f3e6-7949-4cf6-b25c-ebb4691e4812",
      "data": [
        {
          "text": "what did you do at the patent office"
        }
      ],
      "isTemplate": false,
      "count": 0
    }
  ],
}
```

```

"id": "38b83408-16ae-4b05-b841-187b7205e116",
"name": "request_info_patent_office_job",
"auto": true,
"contexts": [],
"responses": [
  {
    "resetContexts": false,
    "action": "inform_patent_office_job",
    "affectedContexts": [
      {
        "name": "patents-office-job",
        "parameters": {},
        "lifespan": 5
      }
    ],
    "parameters": [],
    "messages": [
      {
        "type": 0,
        "speech": "After graduating, I tried to get a teaching post
but was unsuccessful. So with the help of my friend Marcel
Grossman\u0027s father, I got a job in the patent office at Bern."
      }
    ]
  }
],
"priority": 500000,
"webhookUsed": false,
"webhookForSlotFilling": false,
"fallbackIntent": false,
"events": []
}

```

Intent juga dapat diunggah menggunakan spesifikasi dalam format JSON yang ditunjukkan sebelumnya. Untuk melakukannya, buat file JSON, klik opsi lainnya di tab Intents, dan klik Upload Intent:



Gambar 8.39 spesifikasi dalam format JSON

Saat diminta, berikan file maksud dalam format JSON, seperti yang ditunjukkan di sini:

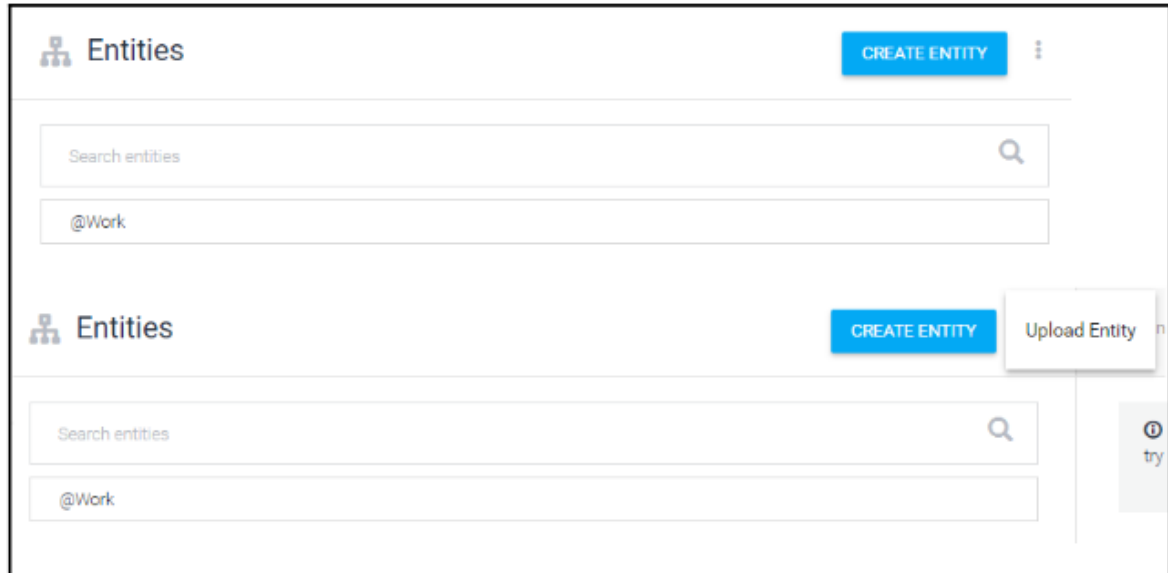


Gambar 8.40 memilih file untuk diupload

Berikut adalah file entitas dari bot Einstein (dari tipe entitas Kerja). Nama tipe entitasnya adalah Pekerjaan dan setiap entitas dimasukkan sebagai entri dengan nilai referensi dan sinonim:

```
{
  "id" : "9ed8010-6ecc-4006-bf7f-7c49bcb03ef",
  "name" : "work" ,
  "isOverridable" : true,
  "entries": [
    {
      "value" : "special-theory" ,
      "synonyms" : [
        "special theory of relativity" ,
        "special theory "
      ]
    },
    {
      "value" : "general-theory" ,
      "synonyms" : [
        "general theory"
        "general theory of relativity"
      ]
    },
    {
      "value" : "brownian motion" ,
      "theory of Brownian motion"
      "theory of Brownian movement"
      "Brownian movement" ,
      "Brownian movement theory"
    }
  ]
  "isEnum" : false ,
  "automatedExpansion" : false
}
```

Tipe entitas baru dapat ditambahkan ke agen dengan membuatnya dalam format sebelumnya dan mengunggahnya. Pendekatan yang sama dapat digunakan untuk mengedit entitas yang ada:



Gambar 8.41 mengupload entity

Pemenuhan

Meskipun Anda dapat membangun agen lengkap di API.AI, Anda masih perlu menggunakan sumber eksternal untuk tugas backend yang seharusnya dilakukan oleh chatbot. Tugas backend adalah tugas yang dilakukan chatbot di belakang layar saat melakukan percakapan dengan pengguna. Hal ini dapat berkisar dari membuat rekaman percakapan di CRM, mengidentifikasi prospek, hingga menanyakan database untuk informasi yang diminta oleh pengguna. Ini disebut pemenuhan.

Contoh pemenuhan dalam chatbot perencanaan perjalanan adalah menanyakan database untuk memeriksa dan mengambil apakah ada penerbangan yang tersedia untuk tanggal tertentu, ke dan dari sumber, dan kota tujuan. Tugas ini akan dipenuhi oleh aplikasi web yang dapat dipanggil oleh chatbot setelah memiliki informasi yang diperlukan untuk melakukan kueri. Contoh lainnya adalah ketika diperlukan untuk melakukan pemesanan.

Sekarang mari kita buat tugas untuk bot Einstein untuk mengeksplorasi hal ini lebih jauh. Mari kita membuat modul kalkulator berdasarkan persamaan Einstein yang terkenal di dunia, $E=mc^2$.

Persamaan ini menghitung jumlah energi yang akan dihasilkan jika massa suatu sistem hilang seluruhnya. Untuk melakukan perhitungan ini, kita perlu mendapatkan massa, m , sistem dari pengguna. Kami akan meneruskannya ke aplikasi web, di mana jumlah energi, E , yang dihasilkan dari massa, m , dihitung. Mari kita lihat contoh percakapannya:

Pengguna : hei. bisakah Anda memberi tahu saya berapa banyak energy yang dihasilkan dari sebuah atom?

Bot : ya, tentu. berapa beratnya?

Pengguna : katakanlah itu 2 gram.




Bot : ok. sistem yang beratnya 2 gram akan berubah menjadi 179751036 MJ energi

Untuk menambahkan webhook ke bot, lakukan hal berikut:

1. Buat aplikasi web yang dapat mengambil parameter, melakukan tugas backend, dan merespons dengan ucapan.
2. Simpan di cloud.
3. Konfigurasi webhook untuk agen.
4. Aktifkan webhook dalam maksud jika diperlukan.

Untuk membangun dan menghosting aplikasi web, lakukan hal berikut:

1. Di konsol Anda, buat direktori untuk menampung proyek ini. Di direktori ini, kita akan membuat tiga file berikut:

-  index.js
-  package.json
-  procfile

index.js adalah program Node.js yang bertindak sebagai aplikasi web untuk menerima parameter dari chatbot, melakukan operasi backend, dan mengirimkan kembali ucapan:

```
const express = require('express')
const bodyParser = require('body-parser')
const request = require('request')

const app = express()

app.set('port', (process.env.PORT || 5000))

// Process application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({extended: false}))

// Process application/json
app.use(bodyParser.json())

app.use(express.static('public'))

// Index route
app.get('/', function (req, res) {
  res.send('Hello world, I am EinsteinBot!.')
})
```

```

app.post('/emc2/', function (req, res) {
  console.log(JSON.stringify(req.body));
  var weight = req.body.result.parameters.weight;
  var m = weight.amount;
  var weight_unit = weight.unit;
  //convert weight into kg
  if (weight_unit == 'g'){
    m = m/1000.0;
  }
  var c2 = 9 * 10^16; //in m^2/s^2
  var e = m * c2;
  res.setHeader('Content-Type', 'application/json');

  var botSpeech = "Energy that the system can create is " + e
  + " Joules.";
  out = {speech: botSpeech,
        displayText: botSpeech,
        data: null};
  var outString = JSON.stringify(out);
  console.log('Out:' + outString);
  res.send(outString);
})

// Spin up the server
app.listen(app.get('port'), function() {
  console.log('running on port', app.get('port'))
})

```

emc2 adalah pegangan POST yang menangani permintaan dari chatbot untuk mengubah massa menjadi energi.

2. Mari kita membuat file package.json untuk memberi tahu Heroku paket-paket yang diperlukan untuk index.js. Buat file package.json di editor teks Anda dengan JSON berikut:

```

{
  "name" : "einstein-bot-wehbook"
  "version" : "1.0.0" ,
  "description" : 'Einstein Bot Wehbook" ,
  "main" : "index.js",
  "script" : {
    "test": "echo\`Error:no test Specified\`" && exit 1"
  },
  "author" : "Srini Janarthanam",
  "license" : "ISC"
  "dependencies" : {
    "body-parser": "^1.15.2"
    "express" : "^4.14.0",
    "request" : "^2.72.0"
  }
}

```

3. Sebagai langkah terakhir, kita perlu membuat Procfile untuk memberitahu Heroku untuk menjalankan index.js. Buat Procfile di editor teks Anda dengan perintah berikut:

```
Web :Node Index.Js
```

Lakukan langkah-langkah berikut untuk mendorong program antarmuka ke cloud. Dari jendela konsol, tempat proyek antarmuka Weatherman Anda berada, jalankan perintah berikut untuk membuat aplikasi Heroku dan dorong aplikasi antarmuka ke cloud:

- git init
- git add
- git commit -m "webhook for einstein bot v1"
- Heroku create einstein-bot
- Git push Heroku master

Anda akan menerima pesan dengan URL aplikasi di dalamnya. Dalam kasus saya, ini adalah <https://einstein-bot.herokuapp.com>.

Untuk mengonfigurasi webhook untuk agen, lakukan hal berikut:

1. Klik opsi Pemenuhan di menu sebelah kiri.
2. Aktifkan webhook. Anda akan melihat formulir spesifikasi webhook berikut:

Gambar 8.42 formulir spesifikasi webhook

3. Isi URL aplikasi web dan klik Simpan. Dalam contoh kita, ini adalah <https://einstein-bot.herokuapp.com/emc2/>. Untuk saat ini, kami tidak akan mengkhawatirkan otentikasi dan header lainnya.

Untuk mengaktifkan webhook dalam maksud, gulir ke bawah sepenuhnya pada halaman maksud. Di bawah tab Pemenuhan, klik Gunakan webhook. Meluas dari aplikasi web yang dirancang sebelumnya untuk mengubah massa menjadi energi menggunakan persamaan Einstein yang terkenal, mari kita lihat bagaimana persamaan tersebut dapat disematkan ke dalam agen. Untuk melakukannya, pertama-tama mari kita buat maksud request_emc, yang akan mengenali ucapan yang meminta konversi (misalnya, "Bisakah Anda mengubah massa 5 gram menjadi energi?"). Maksud ini memerlukan parameter (yaitu massa). Ini ditentukan sebagai entitas sistem (@sys.unit-weight):

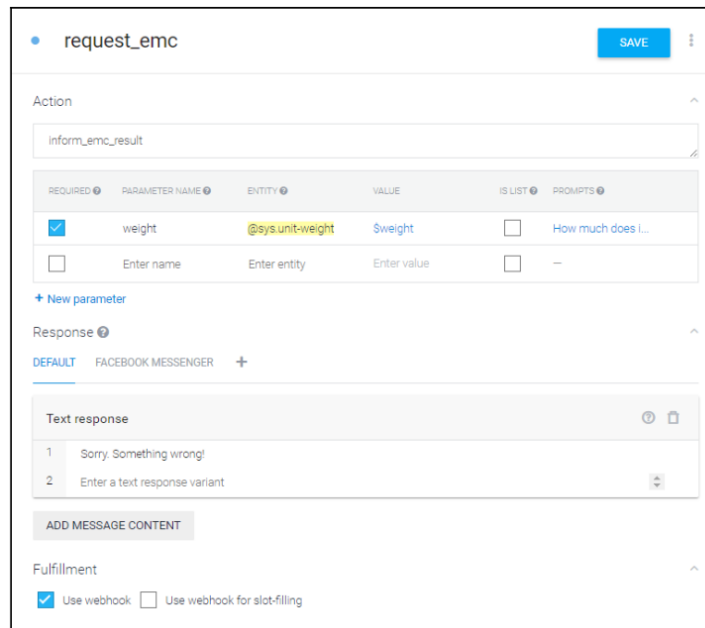
The screenshot shows the Rasa interface for creating a webhook intent named "request_emc". The interface includes a "SAVE" button, a "Contexts" section, a "User says" section with a search bar, and a list of user expressions. Below the expressions is an "Events" section and an "Action" section with a text input field. At the bottom, there is a table of parameters.

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST	PROMPTS
<input checked="" type="checkbox"/>	weight	@sys.unit-weight	\$weight	<input type="checkbox"/>	How much does i...
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>	--

+ New parameter

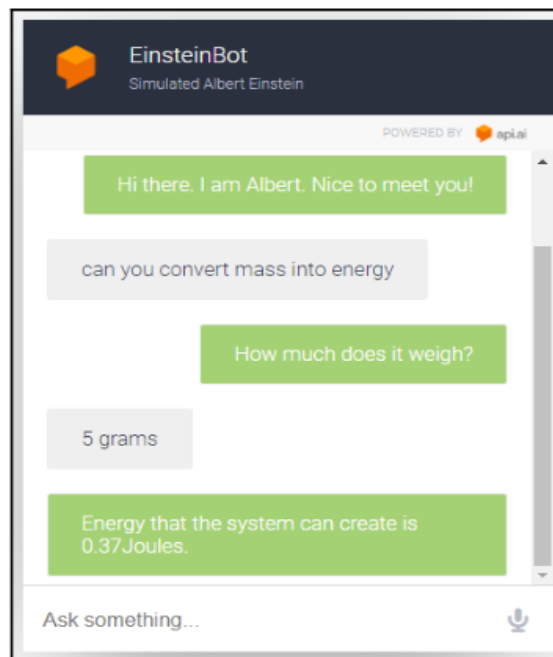
Gambar 8.43 mengaktifkan webhook

Respons default harus berupa sesuatu yang dapat digunakan jika webhook gagal. Aktifkan webhook sehingga aplikasi web akan dipanggil untuk memproses dan memberikan ucapan respons kepada agen:



Gambar 8.44 Respons default jika webhook gagal

Tangkapan layar berikut menunjukkan cara kerja webhook dalam percakapan di platform obrolan web:



Gambar 8.45 Tangkapan layar cara kerja webhook

Ringkasan

Kerja bagus! Dalam bab ini, kami telah merancang dan membangun chatbot percakapan kaya yang meniru Einstein menggunakan API.AI. Jelas sekali, chatbot yang kami buat sangat ramping dan untuk membangun chatbot yang dapat meniru seseorang bahkan hingga tingkat yang cukup baik akan membutuhkan banyak waktu dan tenaga. Namun,

latihan ini berguna untuk menunjukkan konsep yang mendasari pemahaman bahasa alami dalam percakapan seperti maksud, entitas, dan konteks. Kami telah mendemonstrasikan, dengan contoh praktis, cara melakukan integrasi dengan platform penerapan seperti Facebook Messenger dan obrolan web. Kami telah menunjukkan bagaimana tugas backend dapat dikelola dan diintegrasikan dengan lancar ke dalam pengalaman percakapan.

Saya mendorong Anda untuk mencoba agen bawaan yang dapat diimpor ke agen mana pun untuk meningkatkan keterampilan percakapannya. Misalnya, modul obrolan ringan dapat ditambahkan ke agen Anda agar lebih menarik tanpa harus merancang dan mengkodekan semua maksud dan tanggapan yang diperlukan untuk obrolan ringan. Ada alat serupa dari penyedia lain. Hal ini dapat dieksplorasi dengan cara yang sama. Struktur sebagian besar alat ini sangat mirip. Berikut daftar alat yang dapat Anda jelajahi lebih jauh:

- MS LUIS
- IBM Watson
- Facebook WIT.AI
- Amazon Lex

BAB 9

PEMROSESAN BAHASA ALAMI UNTUK CHATBOT

Dalam bab ini, Anda akan mempelajari lebih lanjut tentang cara membuat chatbot pertama Anda dan beberapa prinsip desain. Jika membahas topik agen chatbot, topik yang penting adalah maksud, entitas, konteks, dan entitas. Ini adalah batu bata untuk membangun chatbot Anda. Dalam bab ini, Anda akan mempelajari masing-masingnya dan cara menggunakannya secara efisien.

9.1 NIAT

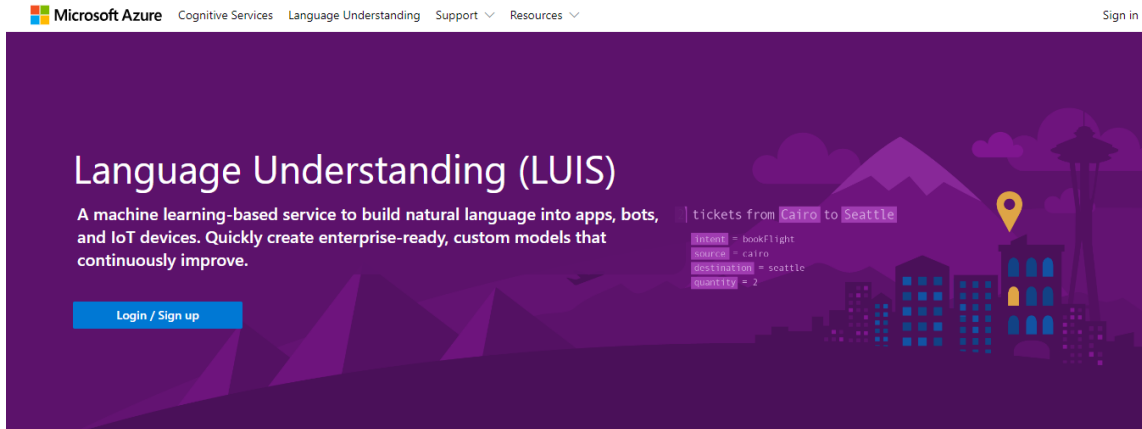
Niat adalah istilah yang digunakan untuk mengidentifikasi niat orang yang menggunakan chatbot secara terprogram. Chatbot harus dapat melakukan beberapa tindakan berdasarkan “niat” yang dideteksinya dari pesan pengguna. Katakanlah Anda sedang membuat chatbot untuk toko yang menjual produk terkait fashion. Sebelum Anda mulai membuat chatbot, Anda perlu mengingat tindakan apa yang dapat dilakukan chatbot Anda. Dalam hal ini, Anda ingin chatbot Anda merespons pengguna dengan informasi tekstual dan visual yang sesuai ketika pengguna ingin melihat produk yang dijual toko dengan mengatakan, misalnya, “Saya ingin membeli baju merah.” Selain itu, saat pengguna mengirimkan pesan chatbot seperti “Apakah Anda punya toko di Berlin?” itu harus dapat menemukan semua toko terdekat untuk lokasi tertentu. Untuk melakukan setiap tindakan ini, chatbot perlu memutuskan apakah pengguna mencari produk atau lokasi toko dari pesan obrolan. Jadi, bisa dibilang chatbot Anda akan memiliki dua maksud: pencarian produk dan pencarian lokasi.

Mendeteksi maksud dari pesan pengguna adalah masalah pembelajaran mesin yang umum. Hal ini dilakukan dengan menggunakan teknik yang disebut klasifikasi teks di mana tujuan dari program ini adalah untuk mengklasifikasikan dokumen/kalimat ke dalam beberapa kelas (dalam hal ini maksud). Kami akan menunjukkan kepada Anda bagaimana Anda dapat membuat pengklasifikasi sederhana menggunakan NodeJS nanti di buku ini, namun untuk saat ini Anda akan menggunakan platform LUIS.ai yang melakukan tugas berat untuk Anda. Botframework mendukung beragam bahasa dan SDK untuk memudahkan proses pembuatan chatbot.

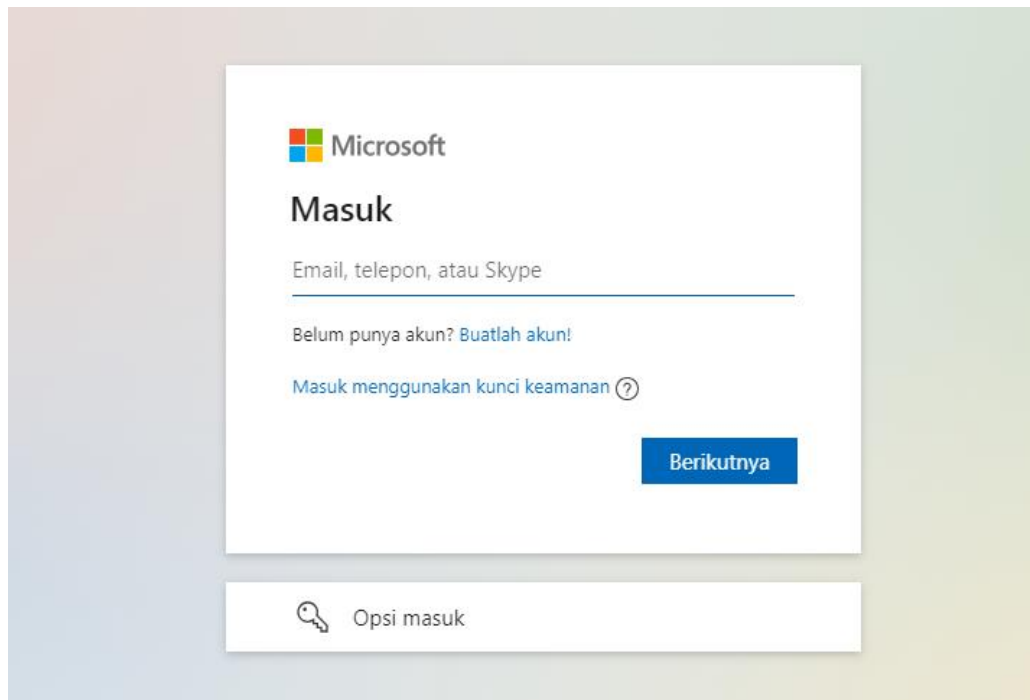
Jika Botframework tidak mendukung atau tidak memiliki SDK untuk bahasa pilihan Anda, Anda dapat membuat chatbot menggunakan REST API yang disediakan oleh Botframework dalam bahasa apa pun.

Sekarang pergilah ke LUIS.ai dan buat akun Anda. LUIS sendiri tidak memerlukan akun terpisah. Jika Anda sudah memiliki akun Microsoft, Anda dapat menggunakannya untuk masuk ke LUIS. Jika Anda tidak memiliki akun Microsoft, Anda dapat mengklik halaman Sign In di halaman beranda LUIS (Gambar 9-1), yang akan membawa Anda ke halaman login akun

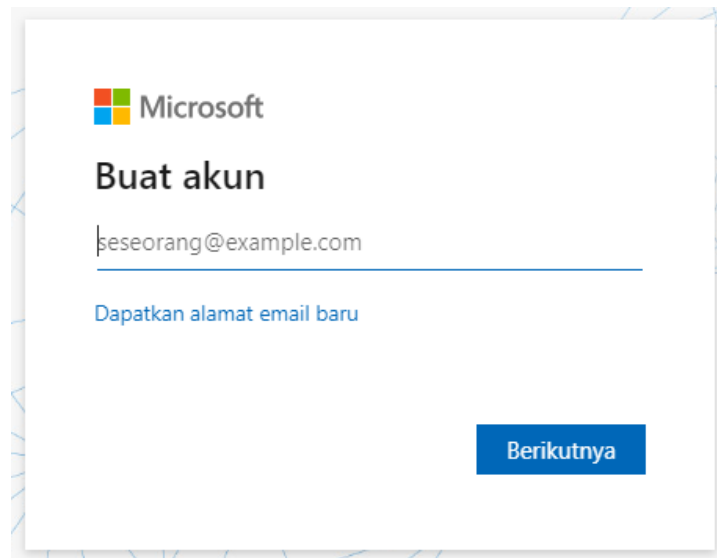
Microsoft. Dari sana, Anda dapat mengklik link “Buat akun Microsoft baru” (Gambar 9-2). Setelah Anda login (Gambar 9-3), buat aplikasi baru dengan mengklik tombol Aplikasi Baru.



Gambar 9.1. halaman beranda LUIS



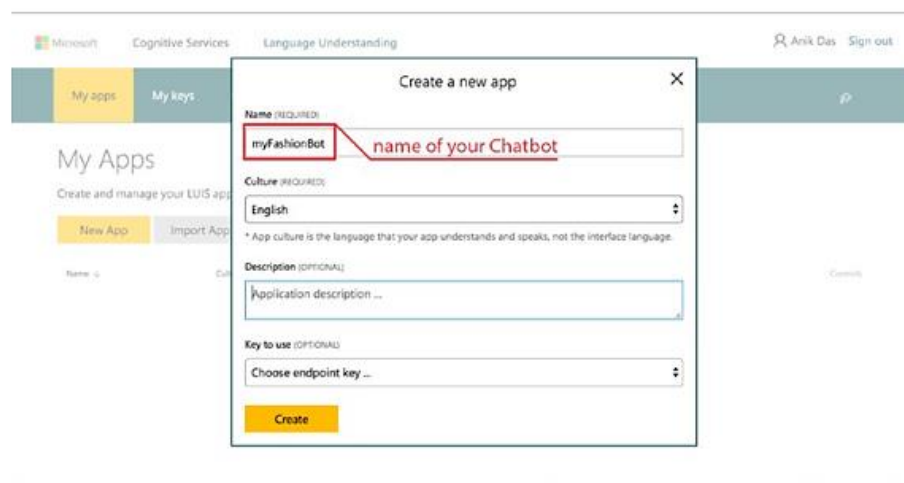
Gambar 9.2. Halaman masuk akun Microsoft



Gambar 9.3. Halaman pendaftaran akun Microsoft

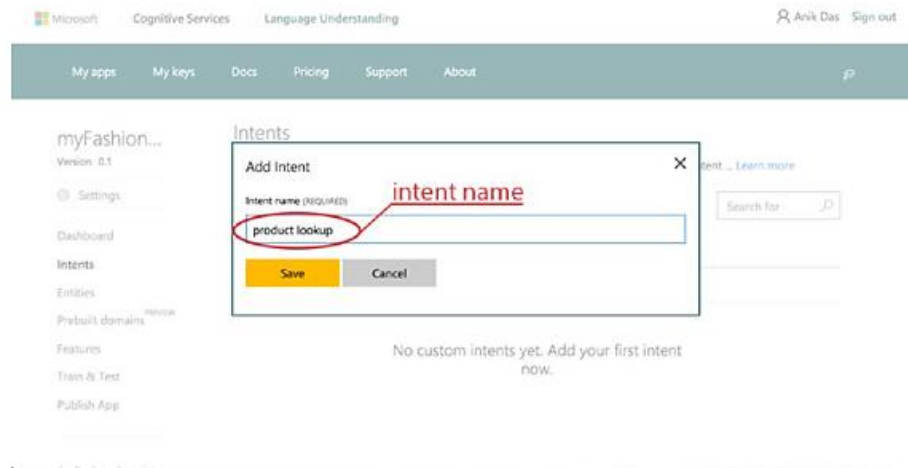
Saat Anda membuat aplikasi, pastikan untuk memberinya nama yang tepat, seperti yang ditunjukkan pada Gambar 9-4. Ini membantu Anda merujuknya dengan mudah saat Anda memiliki banyak aplikasi. Setelah aplikasi dibuat, Anda akan melihat beberapa hal di halaman seperti maksud, entitas, dan sebagainya.

Anda akan belajar lebih banyak tentang mereka saat Anda mengikuti tutorial di bab ini. Untuk saat ini, mari fokus pada suatu niat. Karena chatbot Anda memiliki dua maksud, mari segera buat maksud pertama Anda. Untuk membuat Intent baru, Anda harus masuk ke halaman Intents dengan mengklik Intents di panel kiri. Selanjutnya, klik Add Intent, dan Anda akan melihat dialog Add Intent muncul.



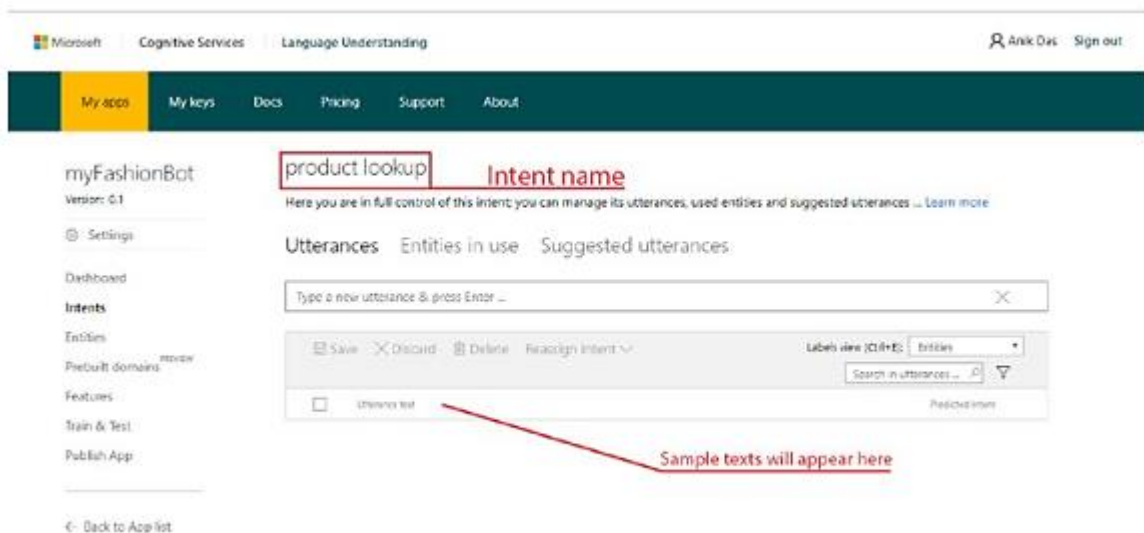
Gambar 9-4. Aplikasi baru LUIS.ai

Masukkan nama produk pencarian, seperti yang ditunjukkan pada Gambar 9-5, lalu klik Simpan.



Gambar 9-5. Tambahkan dialog Intent

Setelah Anda membuat maksud, Anda akan diarahkan ke halaman untuk maksud tersebut. Pada pandangan pertama, Anda akan melihat beberapa hal: ucapan, entitas yang digunakan, ucapan yang disarankan, dan sebagainya (lihat Gambar 9-6).



Gambar 9-6. Halaman beranda

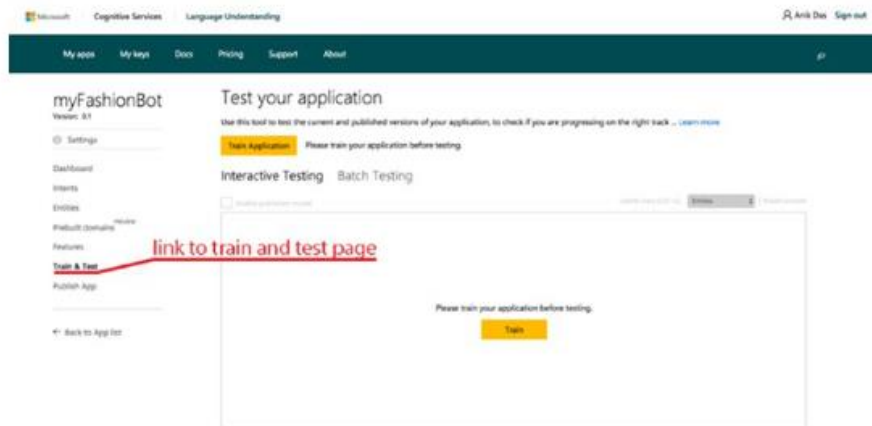
Area yang menjadi fokus pada titik ini adalah ujaran. Ucapan di setiap maksud adalah contoh tampilan pesan pengguna untuk maksud tersebut. Silakan tambahkan contoh berikut ke tab Ucapan Anda untuk maksud pencarian produk:

- Saya ingin membeli kaos.
- Apakah Anda punya celana pendek berwarna abu-abu?
- Tunjukkan padaku beberapa celana chino merah.
- Saya mencari kemeja formal.
- Bolehkah saya melihat celana jins?

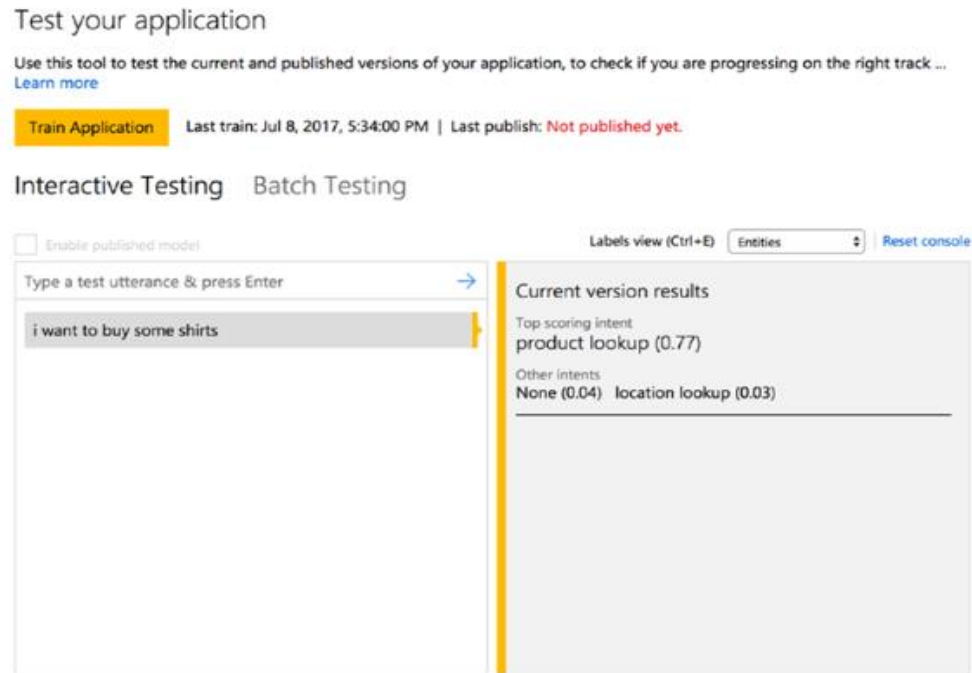
Setelah menambahkan ucapan, klik tombol Simpan untuk menyimpannya dalam maksud. Setelah itu, buat maksud kedua yang disebut pencarian lokasi dan tambahkan ucapan berikut ke maksud tersebut:

- Apakah Anda mempunyai cabang?
- Dimana toko Anda?
- Apakah Anda memiliki toko di Berlin?
- Toko di dekat saya.
- Saya ingin mengunjungi salah satu toko Anda.

Simpan maksud setelah menambahkan ucapan. Sekarang Anda memiliki dua maksud dan beberapa sampel di masing-masing maksud. Untuk melatih model dengan dua maksud ini, klik Latih & Uji di panel kiri, dan Anda akan diarahkan ke halaman yang ditunjukkan pada Gambar 9-7.



Gambar 9-7. Tempat pelatihan dan pengujian LUIS.ai



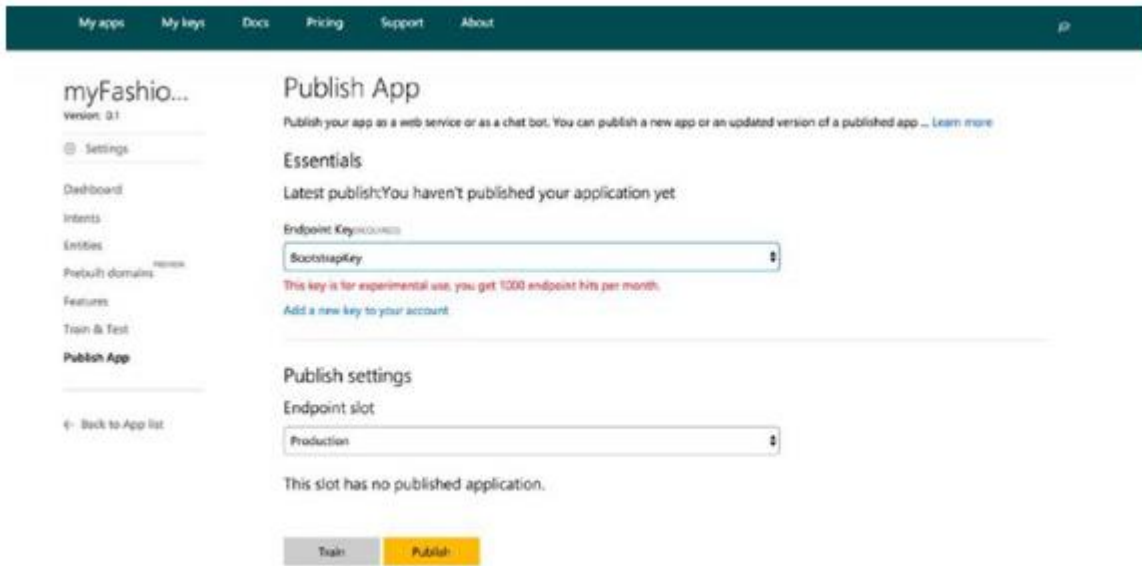
Gambar 9-8. Bagian pengujian interaktif di LUIS.ai

Klik tombol Latih Aplikasi untuk melatih model pertama Anda. Di balik layar, LUIS memproses setiap sampel dari maksud dan menganalisisnya. Hasilnya, Anda mendapatkan REST API yang dapat Anda kueri dengan pesan pengguna, dan REST API akan memberi Anda informasi maksud dari pesan pengguna tersebut. Dengan menggunakan informasi tersebut, Anda dapat dengan mudah menghasilkan respons untuk setiap maksud tersebut. Kita akan membahasnya nanti di bab ini.

Setelah melatih model, Anda dapat menguji model tersebut dengan sampel Anda sendiri untuk memeriksa performa model. Uji model Anda dengan memasukkan Saya ingin membeli beberapa kaos di bagian Pengujian Interaktif. Anda akan melihat sesuatu seperti Gambar 9-8.

Anda akan melihat bahwa sistem mengidentifikasi maksud dengan benar hanya dengan beberapa sampel. Angka di dalam kurung kurawal setiap maksud adalah skor keyakinan yang menentukan seberapa yakin sistem untuk setiap maksud. Anda ingin skornya setinggi mungkin untuk tujuan yang diinginkan. Saatnya mempublikasikan aplikasi pertama Anda dan mulai mengkode chatbot Anda.

Klik Publikasikan Aplikasi di panel kiri (lihat Gambar 9-9).



Gambar 9-9. Publikasikan bagian aplikasi di LUIS.ai

Pilih BootstrapKey di daftar Kunci Titik Akhir dan klik Publikasikan. Setelah Anda memublikasikan aplikasi, Anda akan melihat URL titik akhir yang serupa dengan ini:

https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/<APPLICATION_ID>?subscription-key=<KEY>&verbose=true&timezoneOffset=0&q=

(Kami telah mengganti ID aplikasi dan kunci langganan kami dengan APPLICATION_ID dan KEY dari URL.) Jika Anda menggulung URL berikut, Anda akan menerima JSON dari API dengan maksud yang terdeteksi dan beberapa informasi lainnya.

url: `https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/<APPLICATION_ID>?subscription-key=<KEY>&verbose=true&timezoneOffset=0&q=i want to buy shirts`

response:

```
{
  query: "i want to buy shirts",
  topScoringIntent: {
    intent: "product lookup",
    score: 0.6905002
  },
  intents: [
    {
```

```

        intent: "product lookup",
        score: 0.6905002
    },
    {
        intent: "None",
        score: 0.0479716361
    },
    {
        intent: "location lookup",
        score: 0.0385578275
    }
  ],
  entities: [ ]
}

```

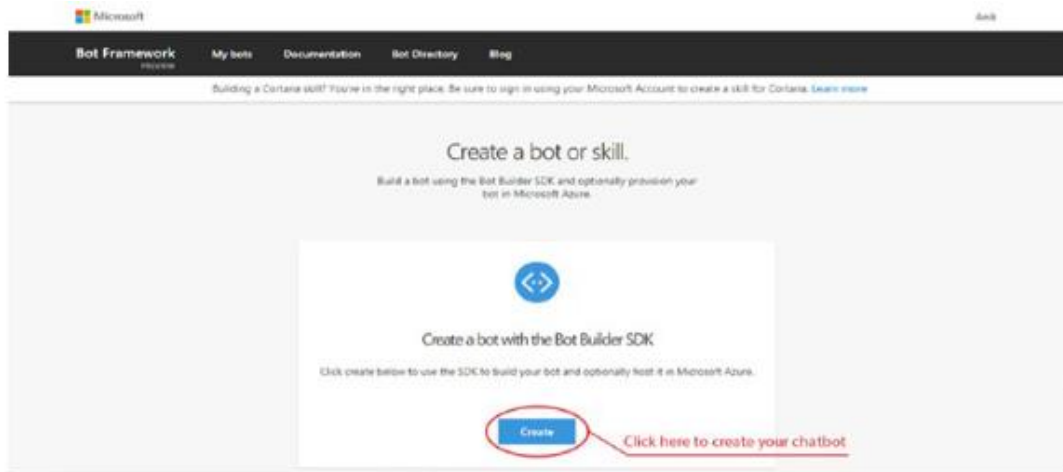
Di URL, ganti APPLICATION_ID dan KEY dengan ID aplikasi dan kunci langganan Anda sendiri.



Gambar 9-10. Membuat chatbot baru (jika Anda sudah punya)

Sekarang Anda harus menyiapkan chatbot Anda di Microsoft Botframework sebelum Anda dapat mulai mengkode bot Anda. Kunjungi <http://dev.botframework.com> dan daftar akun. Setelah Anda mendaftar dan masuk ke akun Botframework Anda, buka bagian Bot Saya. Anda akan diminta untuk mendaftarkan bot Anda (chatbot). Klik Daftar dan isi bagian pertama formulir.

Jika Anda sudah memiliki chatbot yang dibuat dengan Microsoft Botframework, Anda dapat mengklik tombol di bagian kanan atas layar, seperti yang ditunjukkan pada Gambar 9-10. Setelah Anda mengklik tombol "Buat bot", Anda akan melihat halaman seperti yang ditunjukkan pada Gambar 9-11.



Gambar 9-11. Membuat halaman bot di Botframework

Klik tombol Buat, yang akan memunculkan jendela. Dari jendela, pilih “Daftarkan bot yang sudah ada yang dibuat menggunakan Bot Builder SDK” lalu klik tombol OK. Setelah Anda mengklik OK, Anda akan diarahkan ke halaman seperti yang ditunjukkan pada Gambar 9-12 dengan formulir yang harus diisi tentang chatbot Anda.

Gambar 9-12. Pendaftaran chatbot kerangka bot

Di bagian Konfigurasi pada formulir (Gambar 9-13), klik “Buat ID Aplikasi Microsoft dan kata sandi”, yang akan membawa Anda ke halaman yang ditunjukkan pada Gambar 9-14.

Configuration

Messaging endpoint

https URL

Register your bot with Microsoft to generate a new App ID and password

Create Microsoft App ID and password

* Paste your app ID below to continue

Microsoft App ID from the Microsoft App registration portal

Gambar 9-13. Bagian konfigurasi dalam formulir

Microsoft Application Registration Portal Docs Feedback

Generate App ID and password

App name

myFashionBot

App ID

5515aecd-c4a5-4653-9ece-8c37b2eddc74

Generate an app password to continue

English Contact us Terms of use Privacy statement © Microsoft 2017

Gambar 9.14. Halaman pembuatan ID Aplikasi

Salin ID aplikasi dari bagian ini dan tempelkan di bagian Konfigurasi, seperti yang ditunjukkan pada Gambar 9-15.

Configuration

Messaging endpoint

https URL

Register your bot with Microsoft to generate a new App ID and password

Manage Microsoft App ID and password

Paste your app ID below to continue

5515aecd-c4a5-4653-9ece-8c37b2eddc74

Paste your app id here

Gambar 9.15. Bagian konfigurasi

Sekarang daftarkan chatbot Anda dengan mengklik tombol Daftar di bagian bawah halaman. Untuk memahami cara kerja kode sebelumnya, Anda perlu memahami teknologi yang mendasarinya. Aplikasi yang akan Anda kembangkan akan dihosting di server.

Server akan berkomunikasi dengan Botframework. Semua saluran perpesanan Anda seperti Facebook Messenger, Skype, dan Kik akan terhubung ke Botframework.

Setiap kali pengguna mengirim pesan ke bot Anda di saluran mana pun, Botframework akan berkomunikasi dengan aplikasi Anda dengan pesan yang dikirim pengguna. Aplikasi akan memproses pesan dan mengirimkan respons. Apa pun saluran masuk yang digunakan pengguna untuk mengirim pesan ke chatbot, kode Anda akan tetap sama. Itulah keindahan Botframework; itu menciptakan platform komunikasi omnichannel.

Untuk berkomunikasi dengan Botframework RESTfull, Anda akan menggunakan modul NodeJS bernama Restify yang memungkinkan Anda membuat layanan web RESTful dalam beberapa baris. Selain itu, Anda akan menggunakan BotBuilder NodeJS SDK Microsoft untuk berkomunikasi dengan Botframework.

Mari buat direktori bernama MyFashionChatbot di komputer Anda dan buat file bernama app.js. Di dalam app.js, Anda akan menulis logika Anda untuk chatbot. Rekatkan beberapa baris berikut ke app.js:

```
var restify = require('restify');
var builder = require('botbuilder');

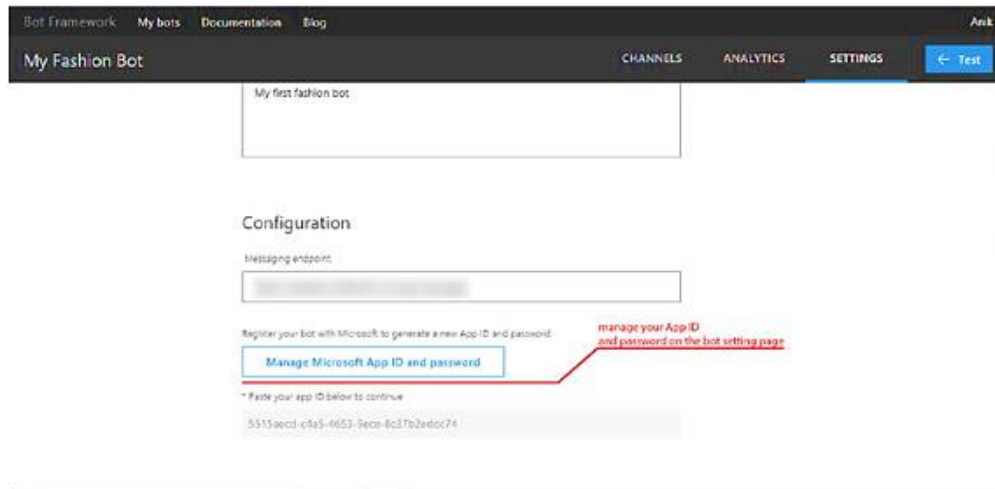
// Setup Restify Server
var server = restify.createServer();
server.listen(process.env.port || process.env.PORT || 3978, function () {
  console.log('%s listening to %s', server.name, server.url);
});
```

Dua baris pertama mencakup restify dan botbuilder. `restify.createServer()` membuat instance server HTTP yang Anda tetapkan ke variabel `server` Anda. Memanggil `server.listen()` memulai server, yang menggunakan port server sebagai argumen pertama dan fungsi panggilan balik sebagai argumen kedua. Server restify akan berjalan pada nomor port 3978, tetapi Anda dapat memilih nomor port Anda sendiri untuk aplikasi Anda. Sekarang tambahkan bagian kode berikut ini ke file app.js Anda:

```
// Create chat connector for communicating with the Bot Framework Service
var APPLICATION_ID = '<APP_ID>';
var APPLICATION_PASSWORD = '<APP_PASSWORD>';
var connector = new builder.ChatConnector({
  appId: APPLICATION_ID,
  appPassword: APPLICATION_PASSWORD
});
```

Kode ini membuat turunan dari `builder.ChatConnector`. Konektor ini akan menangani semua komunikasi dari Botframework dengan aplikasi Anda. Pada kode sebelumnya, `APPLICATION_ID` adalah ID aplikasi yang Anda buat saat mendaftarkan bot di Botframework. Kata sandi aplikasi

dapat dibuat dengan masuk ke pengaturan bot dan kemudian mengklik “Kelola ID Aplikasi Microsoft dan kata sandi” (Gambar 9-16). Setelah Anda membuat kata sandi aplikasi, simpanlah di tempat yang aman karena Anda tidak akan dapat mengambilnya lagi.



Gambar 9-16. Mengelola ID aplikasi dan kata sandi Anda di halaman pengaturan bot

Setelah Anda memiliki ID aplikasi dan kata sandi aplikasi, ganti `APPLICATION_ID` dan `APPLICATION_PASSWORD` dengan keduanya. Tempelkan sisa kode berikut ke file `app.js` Anda:

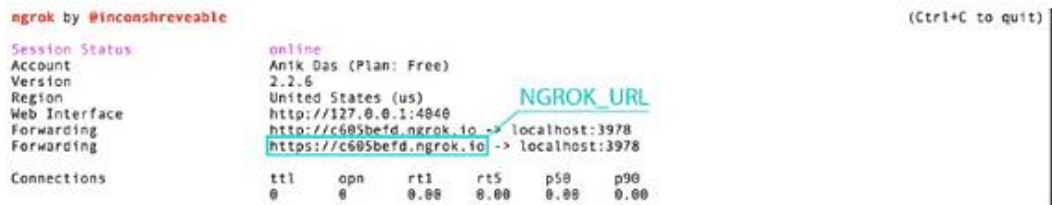
```
// Listen for messages from users
server.post('/api/messages', connector.listen());

// Receive messages from the user and respond by echoing each message back
// (prefixed with 'You said:')
var bot = new builder.UniversalBot(connector, function (session) {
  session.send("You said: %s", session.message.text);
});
```

Argumen pertama dari `server.post`, yaitu `/api/message`, adalah jalur titik akhir HTTP aplikasi tempat Botframework akan POSTing data ketika ada pesan baru yang masuk. Pernyataan `builder.UniversalBot` yang baru membuat instance `UniversalBot`, yang menggunakan konektor Anda sebagai argumen pertama dan fungsi yang menyimpan logika bot Anda sebagai argumen kedua. Kode sebelumnya mengimplementasikan echo chatbot; itu membalas dengan apa pun yang dikatakan pengguna yang diawali dengan “Kamu bilang.” Perhatikan argumen sesi yang diteruskan ke fungsi. Ini menyimpan semua informasi tentang percakapan yang sedang berlangsung dengan pengguna. Misalnya, Anda dapat mengakses pesan pengguna saat ini menggunakan `session.message.text`.

Anda akan menguji chatbot Anda sekarang. Buka Terminal atau baris perintah, navigasikan ke folder/direktori `MyFashionChatbot`, lalu masukkan `node app.js`. Ini akan memulai aplikasi RESTful Anda, yang mendengarkan nomor port 3978. Selanjutnya, jika Anda mengingat

bab sebelumnya, Anda telah menginstal alat bernama ngrok, yang memungkinkan Anda berkomunikasi ke layanan Anda dari dunia luar. Buka jendela Terminal/baris perintah lain dan ketik `ngrok http 3978`. Perhatikan bahwa 3978 adalah nomor port yang Anda gunakan dalam kode Anda. Jika Anda menggunakan port lain, gantilah dengan 3978. Setelah menjalankan perintah, Anda akan melihat sesuatu seperti Gambar 9-17.

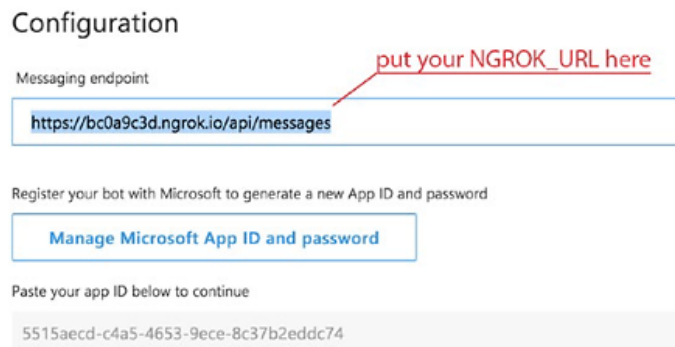


```

ngrok by @Inconshreveable
Session Status: online
Account: Anik Das (Plan: Free)
Version: 2.2.6
Region: United States (us)
Web Interface: http://127.0.0.1:4040
Forwarding:
  http://c605befd.ngrok.io -> localhost:3978
  https://c605befd.ngrok.io -> localhost:3978
Connections:
  ttl    opn    rt1    rt5    p50    p90
   0     0     0.00  0.00  0.00  0.00
  
```

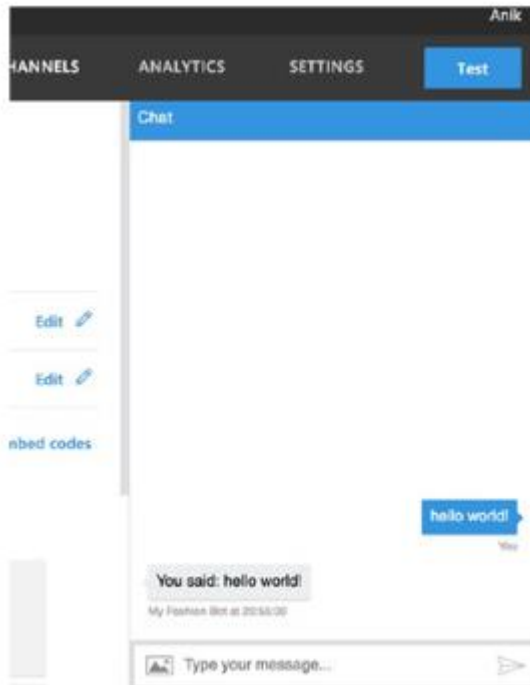
Gambar 9-17. Menjalankan contoh ngrok

Sekarang server lokal dapat diakses dari luar melalui tautan HTTP dan tautan aman HTTPS. Salin tautan dengan awalan HTTPS, buka portal Botframework, dan buka pengaturan chatbot yang baru saja Anda daftarkan. Gulir ke bawah ke bagian konfigurasi dan isi kotak “Messaging endpoint” dengan `NGROK_URL/api/messages`, di mana `NGROK_URL` adalah URL yang baru saja Anda salin (ditunjukkan pada Gambar 9-18).



Gambar 9-18. Menambahkan URL ngrok ke pengaturan bot Botframework

Anda sudah siap. Klik tombol Uji di sudut kanan atas halaman, dan dialog akan muncul. Ketik `halo dunia!` dan tekan Enter. Chatbot Anda akan langsung merespons dengan “Anda berkata: Halo dunia!” Selamat! Anda telah membuat chatbot pertama yang menggemakan pesan pengguna, seperti yang ditunjukkan pada Gambar 9-19.



Gambar 9-19. Menguji chatbot Anda

Sekarang Anda akan mengintegrasikan chatbot Anda dengan maksud yang baru saja Anda buat di LUIS.ai. Sasaran Anda adalah menampilkan semua produk yang tersedia jika maksud pencarian produk terdeteksi dan menampilkan daftar semua toko jika maksud pencarian toko terdeteksi. Untuk melakukan itu, mari ubah file `app.js` Anda. Tambahkan fungsi ini tepat sebelum Anda mendeklarasikan variabel `bot`:

```

LUIS_APPLICATION_ID = '84e62b56-2c91-478c-b382-320a2985720e';
LUIS_SUBSCRIPTION_KEY = '51ee504e1ac14572b84a07ce9e098dbe';
LUIS_URL = 'https://westus.api.cognitive.microsoft.com/luis/v2.0/
apps/'+LUIS_APPLICATION_ID;
function getIntentFromLuis(text, callback) {
  request.get({
    url: LUIS_URL,
    qs: {
      'subscription-key': LUIS_SUBSCRIPTION_KEY,
      'timezoneOffset':0,
      'verbose':true,
      'q': text
    },
  },

```

```

        json: true
      }, function(error, response, data) {
        if(error) {
          callback(error);
        }else{
          callback(null, data);
        }
      });
    }
  }
}

```

Fungsi ini memanggil layanan web LUIS, mengambil informasi maksud terkait teks yang telah diteruskan sebagai argumen kedua, dan memanggil fungsi callback ketika menerima data dari API. Anda dapat menemukan informasi LUIS_APPLICATION_ID dan LUIS_SUBSCRIPTION_KEY Anda di tab Publikasikan Aplikasi di panel kiri portal LUIS Anda. Selanjutnya, mari kita ubah kode bot Anda seperti yang ditunjukkan di samping untuk menggunakan fungsi getIntentOfLuis untuk menghasilkan respons berdasarkan maksud yang teridentifikasi:

```

var bot = new builder.UniversalBot(connector, function (session) {
  getIntentFromLuis(session.message.text, function(error, luisData) {
    var intent = luisData.topScoringIntent.intent;
    if (intent == 'product lookup'){
      session.send("Sure I will show you all the
        products!");
    }else if (intent == 'location lookup'){
      session.send("We have 10 stores across the
        country");
    }else{
      session.send("I did not understand you. I am
        still learning! Can you rephrase?");
    }
  });
});

```

Pada kode sebelumnya, Anda memanggil fungsi getIntentFromLuis dengan pesan pengguna sebagai salah satu argumennya. Berdasarkan maksud yang terdeteksi, Anda menghasilkan respons yang relevan dengan maksud tertentu tersebut. Sekarang masuk ke akun Botframework Anda dan uji bot Anda. Selamat, Anda baru saja membuat bot pertama yang didukung AI (lihat Gambar 9-20)! Sekarang file app.js Anda akan terlihat seperti ini:

```

var restify = require('restify');
var builder = require('botbuilder');
var request = require('request');

```

```

// Setup Restify Server
var server = restify.createServer();
server.listen(3978, function () {
  console.log('%s listening to %s', server.name, server.url);
});

// Create chat connector for communicating with the Bot Framework Service
var APPLICATION_ID = '5515aecd-c4a5-4653-9ece-8c37b2eddc74';
var APPLICATION_PASSWORD = 'bGyKhnsG7gXkQaa6oMxzmvf';
var connector = new builder.ChatConnector({
  appId: APPLICATION_ID,
  appPassword: APPLICATION_PASSWORD
});

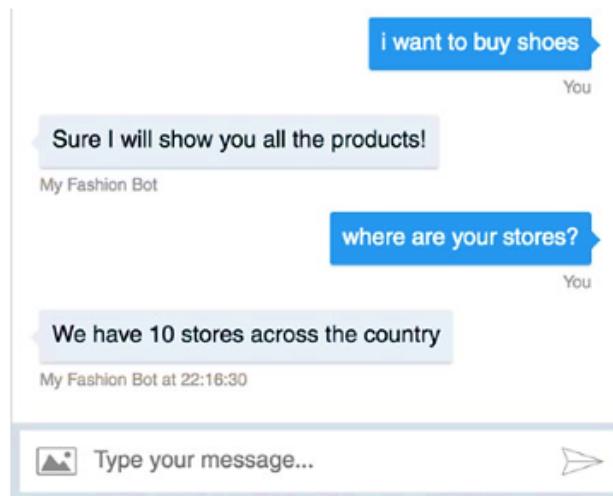
// Listen for messages from users
server.post('/api/messages', connector.listen());

LUIS_APPLICATION_ID = '84e62b56-2c91-478c-b382-320a2985720e';
LUIS_SUBSCRIPTION_KEY = '51ee504e1ac14572b84a07ce9e098dbe';
LUIS_URL = 'https://westus.api.cognitive.microsoft.com/luis/v2.0/apps/' + LUIS_APPLICATION_ID;

// Call LUIS API to get the intent of the user message
function getIntentFromLuis(text, callback) {
  request.get({
    url: LUIS_URL,
    qs: {
      'subscription-key': LUIS_SUBSCRIPTION_KEY,
      'timezoneOffset': 0,
      'verbose': true,
      'q': text
    },
    json: true
  }, function(error, response, data) {
    if(error) {
      callback(error);
    } else {
      callback(null, data);
    }
  });
}

// Receive messages from the user and respond by echoing each message back
// (prefixed with 'You said:')
var bot = new builder.UniversalBot(connector, function (session) {
  getIntentFromLuis(session.message.text, function(error, luisData) {
    var intent = luisData.topScoringIntent.intent;
    if (intent == 'product lookup'){
      session.send("Sure I will show you all the products!");
    } else if (intent == 'location lookup'){
      session.send("We have 10 stores across the country");
    } else {
      session.send("I did not understand you. I am still learning! Can you rephrase?");
    }
  });
});

```



Gambar 9-20. Mengobrol dengan bot Anda

Tapi ada sedikit masalah. Coba kirimkan “Hai” atau apa pun yang berada di luar jangkauan bot Anda. Bot secara acak memilih salah satu maksud antara pencarian produk atau pencarian lokasi atau terkadang merespons dengan “Saya tidak memahami Anda. Saya masih belajar! Bisakah Anda mengulanginya?” Ini bukanlah perilaku yang dapat diterima. Bot Anda harus yakin dengan apa yang dibalasnya. Untuk memastikannya, Anda akan menambahkan ambang batas sebesar 0,30 untuk skor `topScoringIntent`. Perhatikan bahwa ambang batas yang Anda pilih didasarkan pada metode heuristik sepenuhnya. Anda dapat menaikkan atau menurunkan ambang batas berdasarkan kebutuhan Anda, tetapi 0,30 adalah awal yang baik. Mari kita ubah kode bot untuk menambahkan ambang batas.

```
var bot = new builder.UniversalBot(connector, function (session) {
  getIntentFromLuis(session.message.text, function(error, luisData) {
    var intent = luisData.topScoringIntent.intent;
    var score = luisData.topScoringIntent.score;
    if (score > 0.3 && intent != 'None'){
      if (intent == 'product lookup'){
        session.send("Sure I will show you all
          the products!");
      }else if (intent == 'location lookup'){
        session.send("We have 10 stores across
          the country");
      }
    }else{
      session.send("I did not understand you. I am
        still learning! Can you rephrase?");
    }
  });
});
```

Sekarang bot Anda cukup percaya diri saat merespons maksud apa pun. Namun, jika Anda perhatikan lebih dekat, bot Anda merespons secara umum semua pertanyaan terkait produk. Itu tidak dapat mengidentifikasi produk yang dicari pengguna Anda, tetapi itu sangat penting di sini. Bot Anda perlu mengidentifikasi topik pesan. Dalam istilah teknis, Anda menyebut kata-kata penting ini sebagai entitas. Di bagian selanjutnya, Anda akan mempelajari dan memasukkan entitas ke dalam chatbot Anda.

9.2 ENTITAS

Entitas adalah kata kunci/frasa penting yang dicari chatbot Anda dalam pesan pengguna. Entitas ini membantu chatbot mengidentifikasi subjek percakapan dan menyampaikan informasi yang ditargetkan kepada pengguna, sehingga memberikan pengalaman yang lebih baik. Bayangkan ketika Anda mengirim pesan ke chatbot dengan “Saya ingin membeli kemeja” dan chatbot tersebut memahami bahwa Anda ingin membeli produk fashion tetapi tidak dapat “mengidentifikasi” apa yang ingin Anda beli, sehingga memberi Anda informasi tentang semua jenis produk yang ingin Anda beli. Anda tidak tertarik (seperti yang baru saja Anda kembangkan). Jika ia dapat mendeteksi bahwa Anda mencoba membeli kaos dan hanya memberi Anda informasi tentang kaos, seberapa hebatkah pengalaman itu?

Anda dapat menyampaikan pengalaman semacam ini melalui chatbot menggunakan teknik yang disebut pengenalan entitas (NER), yang merupakan metode terkenal untuk mengekstraksi informasi penting dari teks dan mengkategorikannya ke dalam kategori yang telah ditentukan sebelumnya. Anda tidak akan menerapkan NER dalam buku ini, namun Anda akan menggunakan modul Entitas LUIS, yang menangani semua jargon teknis sementara Anda berfokus untuk membuat pengalaman pengguna lebih kaya.

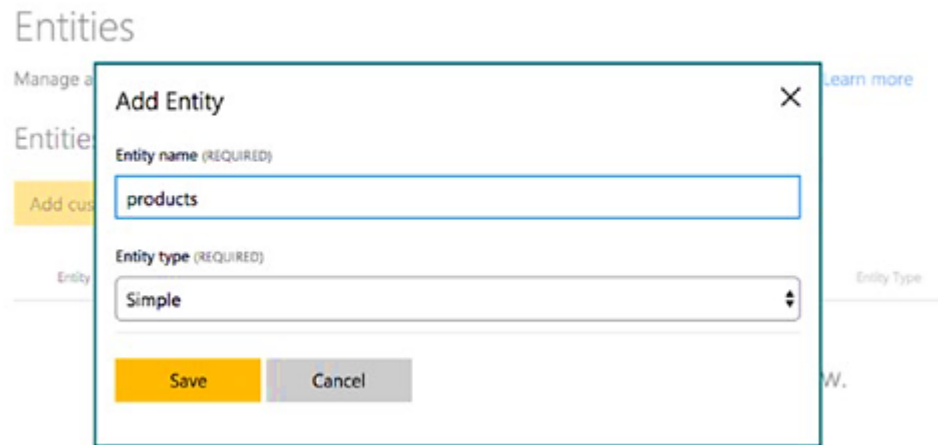
Mari buat dua entitas ini di chatbot Anda untuk saat ini:

- Produk
- Lokasi

Salah satunya adalah entitas kustom (produk), dan yang lainnya adalah entitas bawaan yang disediakan oleh LUIS. Entitas kustom adalah jenis gagasan yang secara eksklusif terkait dengan aplikasi Anda.

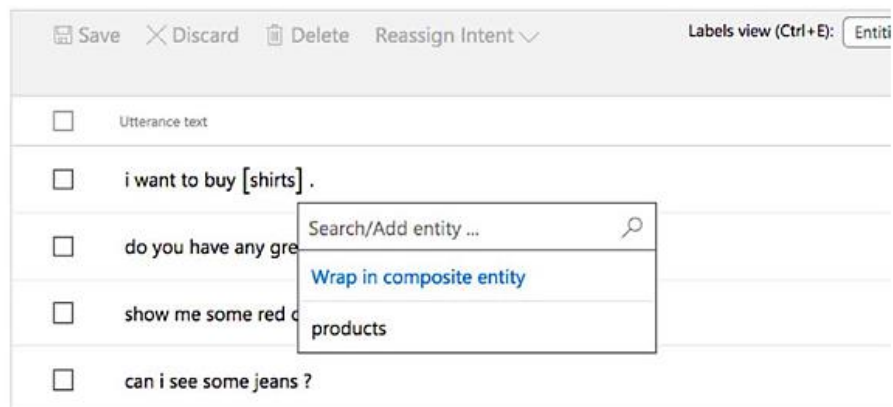
Anda akan membuat entitas produk dari awal. Selain dapat membuat entitas kustom, LUIS juga menyediakan beberapa entitas sistem yang merupakan konsep pengetahuan umum seperti lokasi, tanggal/waktu, dimensi, mata angin, dan sebagainya. Anda akan menggunakan entitas kustom dan entitas bawaan.

Buka portal LUIS Anda dan klik Entitas di panel kiri. Halaman ini berisi semua informasi tentang entitas yang akan Anda gunakan di chatbot Anda. Dari halaman Entitas, klik tombol “Tambahkan entitas kustom”. Modal seperti yang ditunjukkan pada Gambar 9-21 akan muncul. Masukkan produk dan buat entitas kustom pertama Anda dengan mengklik Simpan.



Gambar 9-21. Membuat entitas kustom

Sekarang Anda akan membuat lokasi entitas kedua Anda. Klik “Tambahkan entitas bawaan” dan pilih geografi. Lalu klik Simpan untuk membuat entitas kedua Anda. Setelah membuat kedua entitas ini, Anda perlu melatih entitas kustom Anda. Buka bagian Maksud dan buka maksud pencarian produk. Tandai semua nama produk seperti kemeja, celana pendek, celana chino, jeans, dan sebagainya, sebagai entitas produk, seperti yang ditunjukkan pada Gambar 9-22.



Gambar 9-22. Menandai entitas produk

Setelah menandai semua sampel, klik Simpan untuk menyimpan sampel yang diberi tag dengan entitas produk. Sekarang Anda perlu melatih dan memublikasikan aplikasi Anda. Klik Publikasikan Aplikasi di panel kiri dan klik tombol Latih untuk melatih aplikasi Anda; lalu klik tombol Publikasikan untuk memublikasikan aplikasi. Tanpa memublikasikan aplikasi Anda, Anda tidak akan dapat mengidentifikasi entitas.

Setelah melatih aplikasi, jika Anda memanggil URL API yang sama dengan teks “Saya ingin membeli celana chino”, Anda akan mendapatkan respons seperti ini:

```
{
  "query": "i want to buy chinos",
  "topScoringIntent": {
    "intent": "product lookup",
    "score": 0.6903023
  },
  "intents": [
    {
      "intent": "product lookup",
      "score": 0.6903023
    },
    {
      "intent": "None",
      "score": 0.04735379
    },
    {
      "intent": "location lookup",
      "score": 0.0416906923
    }
  ],
  "entities": [
    {
      "entity": "chinos",
      "type": "products",
      "startIndex": 14,
      "endIndex": 19,
      "score": 0.5390309
    }
  ]
}
```

Demikian pula, Anda akan mendapatkan respons seperti yang ditunjukkan di sini untuk “Toko di Berlin”:

```
{
  "query": "Stores in Berlin",
  "topScoringIntent": {
    "intent": "location lookup",
    "score": 0.956251264
  },
  "intents": [
    {
      "intent": "location lookup",
      "score": 0.956251264
    },
    {
      "intent": "product lookup",
      "score": 0.06805955
    }
  ],
}
```

```

    {
      "intent": "None",
      "score": 0.0558308251
    }
  ],
  "entities": [
    {
      "entity": "berlin",
      "type": "builtin.geography.city",
      "startIndex": 10,
      "endIndex": 15,
      "score": 0.9632844
    }
  ]
}

```

Untuk kedua kasus ini, sistem Anda dapat mendeteksi produk dan entitas lokasi (geografi). Mari ubah file `app.js` Anda sedemikian rupa sehingga dapat mendeteksi dan menggunakan entitas sambil menghasilkan respons. Ubah kode bot sebagai berikut:

```

// Receive messages from the user and respond by echoing each message back
// (prefixed with 'You said:')
var bot = new builder.UniversalBot(connector, function (session) {
  getIntentFromLuis(session.message.text, function(error, luisData) {
    var intent = luisData.topScoringIntent.intent;
    var score = luisData.topScoringIntent.score;
    var entities = luisData.entities;
    if (score > 0.3 && intent !== 'None'){
      if (intent === 'product lookup'){
        if(entities.length > 0){
          var products = [];
          for (var productIterator in
            entities){
            products.push(entities
              [productIterator].entity);
          }
          var message = "Sure I will show you
            " + products.join(', ');
          session.send(message);
        }else{
          session.send("Sure I will show you
            all the products!");
        }
      }else if (intent === 'location lookup'){
        // considering only first location
        if (entities.length > 0){
          var location = entities[0].entity;
          session.send("We have 10 stores across "
            + location);
        }else{
          session.send("We have 50 stores across
            the country.");
        }
      }
    }
  });
});

```

```

    }
  }else{
    session.send("I did not understand you. I am
    still learning! Can you rephrase?");
  }
});
});

```

Anda telah memodifikasi kode sedemikian rupa sehingga akan merespons pengguna berdasarkan produk yang telah diidentifikasi untuk maksud pencarian produk. Untuk maksud pencarian lokasi akan merespon berdasarkan lokasi yang telah ditentukan pengguna. Jika pengguna bertanya tentang lebih dari satu produk, Anda telah menggabungkan produk tersebut dengan koma, sehingga pengguna tahu bahwa Anda memiliki pengetahuan tentang topik pembicaraan.

Anda juga dapat mencoba menambahkan beberapa entitas lagi ke aplikasi seperti warna, ukuran, dan sebagainya, untuk memberikan pengalaman kesadaran informasi yang lebih baik kepada pengguna.

Maksud dan entitas adalah dua blok dasar chatbot. Semakin baik deteksi maksud dan entitas, semakin baik pengalaman yang diberikan chatbot. Kunci kesuksesan chatbot adalah melatih maksud dan entitas dengan jumlah sampel yang banyak, dengan variasi di setiap sampel. Variasi sampel memastikan generalisasi yang lebih baik dari model pembelajaran mesin yang berfungsi. Dalam bab ini, Anda mempelajari dan menerapkan kedua konsep utama chatbot. Konsep-konsep ini akan membantu Anda memahami topik yang akan kita bahas di bab-bab berikutnya.

BAB 10

PEMROSESAN BAHASA ALAMI UNTUK CHATBOTS

Bab ini adalah untuk membantu Anda memulai Pemrosesan Bahasa Alami (NLP) menggunakan Python yang diperlukan untuk membangun chatbots. Anda akan mempelajari metode dan teknik dasar NLP menggunakan perpustakaan sumber terbuka menggunakan yang disebut spaCy. Jika Anda seorang pemula atau menengah dalam ekosistem Python, jangan khawatir, karena Anda akan melakukan setiap langkah yang diperlukan untuk mempelajari NLP untuk chatbots. Bab ini tidak hanya mengajarkan Anda tentang metode dalam NLP tetapi juga mengambil contoh kehidupan nyata dan mendemonstrasikannya dengan contoh pengkodean. Kami juga akan membahas mengapa metode NLP tertentu mungkin diperlukan untuk chatbots. Perhatikan bahwa NLP itu sendiri adalah sebuah keterampilan.

Kita akan mencermati penandaan POS, stemming, deteksi entitas, stopwords, penguraian ketergantungan, dan potongan kata benda serta menemukan kesamaan antar kata. Semua metode ini akan sangat membantu Anda saat membuat chatbot untuk kasus penggunaan Anda.

Ada lebih banyak metode NLP daripada yang dibahas dalam bab ini. Berdasarkan kebutuhan Anda akan chatbot yang Anda buat, Anda dapat mencoba mempelajarinya. Pustaka SpaCy yang akan kita pelajari penggunaannya pada akhir bab ini akan memberi Anda cukup gambaran tentang bagaimana meningkatkan basis pengetahuan Anda tentang NLP dan pemahamannya. Jadi, mari kita mulai dan coba pahami NLP untuk chatbots terlebih dahulu di bagian selanjutnya.

Mengapa Saya Perlu Mengetahui Pemrosesan Bahasa Alami untuk Membangun Chatbot?

Untuk memahami jawaban pertanyaan ini, mari kita pahami dulu Natural Language Processing (NLP). *Natural Language Processing (NLP)* adalah bidang Kecerdasan Buatan yang memungkinkan komputer menganalisis dan memahami bahasa manusia.

Nah, untuk melakukan NLP atau katakanlah *Natural Language Understanding (NLU)*, kita punya banyak sekali metode yang akan kita bahas selanjutnya. Anda pernah mendengar istilah baru Natural Language Understanding (NLU)—apa itu sekarang?

Secara sederhana, *NLU* adalah bagian dari gambaran NLP yang lebih besar, sama seperti pembelajaran mesin, pembelajaran mendalam, NLP, dan penambangan data adalah bagian dari gambaran yang lebih besar dari Kecerdasan Buatan (AI), yang merupakan istilah umum untuk komputer mana pun. program yang melakukan sesuatu yang cerdas. Aturan praktis yang baik adalah menggunakan istilah *NLU* untuk mengekspresikan kemampuan mesin dalam memahami bahasa alami dalam bentuk yang disediakan oleh manusia.

Sekarang, muncul pertanyaan apakah Anda benar-benar perlu mengetahui NLP untuk membangun chatbot jawabannya adalah Ya dan Tidak. Bingung? Anda tidak salah dengar, bukan

berarti Anda tidak bisa membangun chatbot sama sekali jika Anda tidak mengetahui metode dan teknik NLP, tetapi cakupan Anda akan agak terbatas. Anda tidak akan dapat menskalakan aplikasi dan menjaga kode tetap bersih pada saat yang bersamaan. NLP memberi chatbot Anda sayap untuk terbang ketika ia tidak bisa berjalan dan berlari.

Chatbots bagi orang awam hanyalah sebuah cara untuk berkomunikasi dengan mesin cerdas di ujung sana. Mesin ini bisa berbasis suara atau berbasis teks, dimana pengguna akan memberikan masukan dalam bahasanya sendiri, yang dalam ilmu komputer umumnya disebut bahasa alami.

Kita tahu bahwa tidak ada kotak hitam yang mampu melakukan keajaiban, dan semuanya berfungsi dengan baik. Kita harus tahu bahwa tidak ada yang buatan dalam AI; itu sebenarnya mesin algoritma pembelajaran dan pembelajaran mendalam yang ditulis oleh orang-orang hebat, berjalan di bawah tenda. Mesin belum mencapai tahap di mana mereka bisa berpikir serupa dengan manusia untuk memiliki kecerdasannya sendiri. Sistem AI saat ini apa yang mereka lakukan dan cara mereka berperilaku adalah hasil dari cara kami melatih mereka.

Jadi, untuk memahami bahasa alami pengguna dalam bahasa apa pun, atau apa pun bentuk masukannya (teks, suara, gambar, dll.), kita harus menulis algoritme dan menggunakan teknik NLP. NLP dianggap sebagai otak chatbot yang memproses data mentah, melakukan pemrosesan, membersihkannya, dan kemudian bersiap untuk mengambil tindakan yang tepat. NLP sendiri merupakan topik yang sangat besar dan membutuhkan waktu serta ketekunan untuk mempelajarinya secara menyeluruh, namun ada beberapa metode yang perlu diketahui oleh pengembang chatbot, yang akan kita pelajari di bab ini.

10.1 Apa itu spaCy?

spaCy adalah perpustakaan perangkat lunak sumber terbuka untuk NLP tingkat lanjut, ditulis dengan Python dan Cython, dibuat oleh Matthew Honnibal. Ini menyediakan API intuitif untuk mengakses metode yang dilatih oleh model pembelajaran mendalam.

spaCy menawarkan parser sintaksis tercepat di dunia. Diambil langsung dari dokumentasi *spaCy*, mereka mempunyai beberapa hasil benchmarking yang luar biasa, yang ditunjukkan di bawah ini.

Hasil Tolok Ukur spaCy

Dua makalah tinjauan sejawat pada tahun 2015 mengonfirmasi bahwa *spaCy* menawarkan pengurai sintaksis tercepat di dunia dan akurasi berada dalam 1% dari yang terbaik yang tersedia. Beberapa sistem yang lebih akurat 20 kali lebih lambat atau lebih. Mari kita coba lihat Gambar 10-1 yang menunjukkan hasil benchmarking *spaCy* berdasarkan kecepatan dan akurasi dibandingkan dengan perpustakaan lain.

SYSTEM	YEAR	LANGUAGE	ACCURACY	SPEED (WPS)
spaCy v2.x	2017	Python / Cython	92.6	n/a ?
spaCy v1.x	2015	Python / Cython	91.8	13,963
ClearNLP	2015	Java	91.7	10,271
CoreNLP	2015	Java	89.6	8,602
MATE	2015	Java	92.5	550
Turbo	2015	C++	92.4	349

Gambar 10.1. hasil benchmarking spaCy

spaCy juga menawarkan model jaringan saraf statistik untuk berbagai bahasa seperti Inggris, Jerman, Spanyol, Portugis, Prancis, Italia, Belanda, dan NER multi-bahasa. Ini juga menyediakan tokenisasi untuk berbagai bahasa lainnya. Tabel ini menunjukkan kecepatan yang diukur oleh Choi et al., jadi tidak adil jika membandingkan spaCy v2.x yang diukur pada perangkat keras yang berbeda. Inilah alasan Anda tidak melihat nilai kolom kecepatan untuk spaCy v2.x.

Apa yang Disediakan spaCy?

Ada tiga hal utama yang diklaim disediakan oleh spaCy dan sangat membantu. Mari kita coba melihatnya dan memahami mengapa seseorang harus mengetahui dan menggunakan spaCy sebagai modul masuk untuk melakukan NLP.

Perpustakaan Tercepat di Dunia

spaCy bekerja dengan sangat baik dalam mengekstraksi informasi berskala besar. Itu ditulis dari awal dengan sangat hati-hati terhadap memori dengan bantuan perpustakaan Cython.

Selesaikan Segalanya

spaCy dirancang dengan mempertimbangkan “menyelesaikan sesuatu”. Ini membantu kita dalam mencapai skenario NLP dunia nyata. Dokumentasi yang bersih menghemat banyak waktu bagi pengembang dan penggemar linguistik komputasional dan membuat mereka lebih produktif. Instalasinya mudah, sama seperti paket Python lainnya.

10.2 Pembelajaran Mendalam

spaCy adalah salah satu perpustakaan terbaik yang tersedia di komunitas sumber terbuka untuk memproses teks untuk algoritma pembelajaran mendalam. Ia berkolaborasi secara lancar dengan TensorFlow, PyTorch, scikit-learn, Gensim, dan teknologi terkait Python lainnya. Pengembang pembelajaran mendalam dapat dengan mudah membuat model statistik yang canggih secara linguistik untuk berbagai masalah NLP/NLU.

Fitur spaCy

Tidak ada perpustakaan NLP lain yang menyediakan API yang sangat beragam untuk melakukan hampir semua hal, seperti yang dilakukan spaCy. Hal terbaik tentang perpustakaan ini adalah ia terus berkembang dan menjadi lebih baik dan lebih baik lagi. Mari kita intip fitur-fitur spaCy seperti yang disebutkan di situs resminya [<https://spacy.io/>].

- ❖ Tokenisasi non-destruktif
- ❖ Pengakuan entitas bernama
- ❖ Dukungan untuk 28+ bahasa
- ❖ 13 model statistik untuk 8 bahasa
- ❖ Vektor kata terlatih
- ❖ Integrasi pembelajaran mendalam yang mudah
- ❖ Penandaan bagian dari ucapan
- ❖ Penguraian ketergantungan berlabel
- ❖ Segmentasi kalimat berdasarkan sintaksis
- ❖ Visualisator bawaan untuk sintaksis dan NER
- ❖ Pemetaan string-ke-hash yang mudah
- ❖ Ekspor ke array data numpy
- ❖ Serialisasi biner yang efisien
- ❖ Pengemasan dan penerapan model yang mudah
- ❖ Kecepatan tercanggih
- ❖ Akurasi yang kuat dan dievaluasi secara ketat

Sekarang, mari selami modul luar biasa ini untuk NLP dengan Python: spaCy.

10.3 INSTALASI DAN PRASYARAT

Sebelum kita mendalami spaCy dan cuplikan kode, pastikan Anda telah menginstal Python di OS Anda. Jika tidak, lihat.

Anda dapat menggunakan versi Python mana pun yang Anda rasa nyaman. Sebagian besar sistem saat ini sudah diinstal sebelumnya dengan Python versi 2.7.x default. Kami akan menggunakan Python 3 di bab ini. Jadi, jika Anda ingin menggunakan Python 3, silakan instal Python 3 di sistem operasi Anda dengan mengunduhnya dari <https://www.python.org/downloads/>. Jika Anda sudah menginstal Python 2, Anda juga dapat menggunakannya; mungkin memerlukan atau tidak memerlukan sedikit perubahan kode.

Kami akan menginstal spaCy melalui pip. Kami akan menggunakan lingkungan virtual dan menginstal spaCy ke direktori pengguna.

Jika Anda menggunakan macOS/OSX/Linux, ikuti langkah-langkah berikut:

Step 1 : `python3 -m pip install -u virtualenv`

Step 2 : `virtualenv venv -p /usr/local/bin/python3 #Make Sure You Use Your own OS path for python 3 executable.`

Step 3 : source venv/bin/activate

Step 4 : pip3 install -u spacy # We'll be using spaCy version 2.0.11.

Langkah terakhir mungkin memerlukan waktu, jadi tunggulah dengan sabar. Jika Anda menggunakan Windows, ubah saja Langkah 3 menjadi

Venv\Scripts\activate

Sekarang, kita akan menginstal Jupyter Notebook di dalam lingkungan virtual kita, yang kita aktifkan pada Langkah 3. Menggunakan Jupyter Notebook jauh lebih mudah dan produktif daripada interpreter Python standar. Kami akan mengeksekusi semua cuplikan di Jupyter Notebook di bab-bab mendatang.

Untuk menginstal Jupyter Notebook, jalankan perintah pip berikut:

Pip3 install jupyter

Perintah ini akan menginstal Jupyter Notebook di sistem Anda.

Pada titik ini Anda seharusnya sudah menginstal SpaCy dan Jupyter Notebook di virtualenv Anda. Mari kita verifikasi apakah semuanya berhasil diinstal.

1. Buka antarmuka baris perintah Anda dan ketik yang berikut ini dan Anda akan melihat server sedang dimulai dan membuka url di browser default Anda.

#jupyter notebook

Url defaultnya adalah <http://localhost:8888/tree>. Seharusnya terlihat seperti Gambar 10-2.

2. Klik Baru seperti yang ditunjukkan pada Gambar 10-2, dan pilih Python 3.

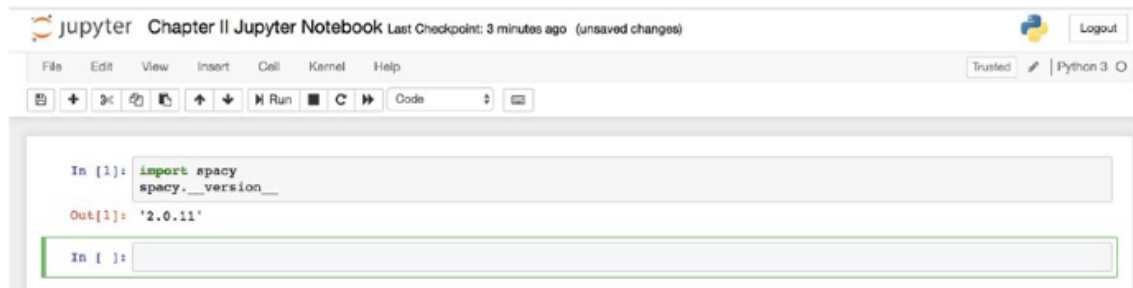
Ini akan membuka tab baru di browser Anda saat ini dan membuat buku catatan baru untuk Anda, tempat Anda bisa bermain-main dengan kode Python.

Anda dapat menjalankan kode Python apa pun, mengimpor perpustakaan, diagram plot, dan sel penurunan harga.



Gambar 10.2. Tampilan pertama Notebook Jupyter

3. Ketik `import spaCy` dan jalankan sel dengan mengklik tombol “Run” atau dengan menekan Shift + Enter. Seharusnya terlihat seperti Gambar 10-3.



```

jupyter Chapter II Jupyter Notebook Last Checkpoint: 3 minutes ago (unsaved changes)
Files Edit View Insert Cell Kernel Help Trusted Python 3
In [1]: import spacy
        spacy.__version__
Out[1]: '2.0.11'
In [ ]:

```

Gambar 10.3. Memverifikasi instalasi spaCy

Jika Langkah 3 tidak memunculkan pesan kesalahan apa pun, maka Anda telah berhasil menginstal modul spaCy di sistem Anda. Anda akan melihat versi spaCy yang terinstal di buku catatan Anda. Jika Anda ingin menginstal spaCy versi yang sama maka Anda dapat menentukan versinya saat menginstal spaCy melalui pip.

Pip3 install -u spacy == 2.0.11

Apa Itu Model SpaCy?

Model SpaCy sama seperti model pembelajaran mesin atau pembelajaran mendalam lainnya. Model adalah hasil dari suatu algoritma atau, katakanlah, sebuah objek yang dibuat setelah data pelatihan menggunakan algoritma pembelajaran mesin. spaCy memiliki banyak model yang dapat ditempatkan langsung di program kami dengan mengunduhnya sama seperti paket Python lainnya.

Sekarang, kita akan menginstal model spaCy sebagai paket Python. Untuk melakukan itu, kita akan menjalankan perintah berikut di notebook dengan menggunakan perintah ajaib notebook. Dengan memberi awalan! [Operator Seru] sebelum perintah shell, kita juga dapat menjalankan perintah shell dari Jupyter Notebooks. Mari kita lihat tampilannya.

!python3 -m spacy download en

Anda mungkin mendapatkan masalah izin saat menggunakan Jupyter Notebook untuk mengunduh model spaCy untuk Python 3. Buka terminal Anda dan jalankan perintah berikut:

Sudo python3 -m download en

Lihat Gambar 10-4 untuk referensi.

```

Sumit:Chapter II geospark-device-3$ sudo python3 -m spacy download en
Password:
Collecting https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-2.0.0/en_core_web_sm-2.0.0.tar.gz
  Downloading https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-2.0.0/en_core_web_sm-2.0.0.tar.gz
    100% |#####| 37.4MB 59.0MB/s
Requirement already satisfied (use --upgrade to upgrade): en-core-web-sm==2.0.0 from https://github.com/explosion/spacy-models/releases/download/en_core_web_sm-2.0.0/en_core_web_sm-2.0.0.tar.gz in /usr/local/lib/python3.6/site-packages
You are using pip version 10.0.1, however version 18.0 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.

Linking successful
/usr/local/lib/python3.6/site-packages/en_core_web_sm -->
/usr/local/lib/python3.6/site-packages/spacy/data/en

You can now load the model via spacy.load('en')

```

Gambar 10.4. Mengunduh model spaCy

Seperti yang Anda lihat pada Gambar 10-4, spaCy mencoba mengunduh beberapa file inti dan menginstalnya sebagai paket Python.

Catatan! [Operator Seru] hanya berfungsi di Notebook Jupyter. Untuk menginstal model spaCy langsung dari terminal, Anda harus menghapusnya! [Operator Seru]; jika tidak maka akan mengakibatkan kesalahan.

- [1] <https://www.python.org/downloads/>
- [2] <https://packaging.python.org/tutorials/installing-packages/#installing-from-pypi>
- [3] <http://docs.python-guide.org/en/latest/dev/virtualenvs/>

10.4 METODE DASAR NLP UNTUK MEMBANGUN CHATBOTS

Sangat penting untuk menguasai dasar-dasar untuk menjadi ahli dalam sesuatu dan melakukannya secara efektif dan efisien. Untuk membangun chatbots, kita perlu mengetahui metode dasar NLP. Metode ini membantu kita memecah masukan menjadi beberapa bagian dan memahaminya. Di bagian selanjutnya, kita akan mempelajari beberapa metode NLP yang paling sering digunakan yang tidak hanya akan membantu Anda menguasai NLP tetapi juga pandai membuat chatbot yang keren. Semakin kami dapat memproses teks masukan dengan lebih baik dan efisien, semakin baik kami dapat merespons pengguna.

Penandaan POS

Penandaan part-of-speech (POS) adalah proses di mana Anda membaca beberapa teks dan menetapkan bagian-bagian ucapan untuk setiap kata atau token, seperti kata benda, kata kerja, kata sifat, dll. Penandaan POS menjadi sangat penting ketika Anda ingin mengidentifikasi suatu entitas dalam kalimat tertentu. Langkah pertama adalah melakukan penandaan POS dan melihat isi teks kita. Mari kita mulai dengan beberapa contoh penandaan POS yang sebenarnya.

Output:

```
( 'I', 'PRON' )
( 'am', 'VERB' )
( 'going', 'VERB' )
( 'to', 'ADP' )
( 'London', 'PROPN' )
( 'next', 'ADJ' )
( 'week', 'NOUN' )
( 'for', 'ADP' )
( 'a', 'DET' )
( 'meeting', 'NOUN' )
( '.', 'PUNCT' )
```

Example 1:

```
nlp = spacy.load('en') #Loads the spacy en model into a python object
doc = nlp(u'I am learning how to build chatbots') #Creates a doc object
for token in doc:
    print(token.text, token.pos_) #prints the text and POS
```

Output:

```
( 'I', 'PRON' )
( 'am', 'VERB' )
( 'learning', 'VERB' )
( 'how', 'ADV' )
( 'to', 'PART' )
( 'build', 'VERB' )
( 'chatbots', 'NOUN' )
```

Example 2:

```
doc = nlp(u'I am going to London next week for a meeting.')
for token in doc:
    print(token.text, token.pos_)
```

Seperti yang bisa kita lihat, saat kita mencetak token dari objek Doc yang dikembalikan dari metode nlp, yang merupakan wadah untuk mengakses anotasi, kita mendapatkan POS yang diberi tag dengan setiap kata dalam kalimat.

Tag ini adalah properti milik kata yang menentukan kata tersebut digunakan dalam kalimat yang benar secara tata bahasa. Kita dapat menggunakan tag ini sebagai fitur kata dalam penyaringan informasi, dll.

Mari kita coba mengambil contoh lain di mana kita mencoba mengeksplorasi berbagai atribut token yang berasal dari objek Doc.

Example 3:

```
doc = nlp(u'Google release "Move Mirror" AI experiment that matches your
pose from 80,000 images')
```

```
for token in doc:
```

```
    print(token.text, token.lemma_, token.pos_, token.tag_, token.dep_,
          token.shape_, token.is_alpha, token.is_stop)
```

Output:

Text	Lemma	POS	Tag	Dep	Shape	Alpha	Stop
Google	google	PROPN	NNP	compound	Xxxxx	True	False
Release	release	NOUN	NN	nmod	xxxx	True	False
"	"	PUNCT	``	punct	"	False	False
Move	move	PROPN	NNP	nmod	Xxxx	True	False
Mirror	mirror	PROPN	NNP	nmod	Xxxxx	True	False
"	"	PUNCT	"	punct	"	False	False
AI	ai	PROPN	NNP	compound	XX	True	False
Experiment	experiment	NOUN	NN	ROOT	xxxx	True	False
That	that	ADJ	WDT	nsubj	xxxx	True	True
Matches	match	VERB	VBZ	relcl	xxxx	True	False
Your	-PRON-	ADJ	PRP\$	poss	xxxx	True	True
Pose	pose	NOUN	NN	dobj	xxxx	True	False
From	from	ADP	IN	prep	xxxx	True	True
80,000	80,000	NUM	CD	nummod	dd,ddd	False	False
Images	image	NOUN	NNS	pobj	xxxx	True	False

Example 4:

```
doc = nlp(u'I am learning how to build chatbots')
```

```
for token in doc:
```

```
    print(token.text, token.lemma_, token.pos_, token.tag_, token.dep_,
          token.shape_, token.is_alpha, token.is_stop)
```

Output:

TEXT	LEMMA	POS	TAG	DEP	SHAPE	ALPHA	STOP
I	-PRON-	PRON	PRP	nsubj	X	True	False
am	be	VERB	VBP	aux	xx	True	True
learning	learn	VERB	VBG	ROOT	xxxx	True	False
how	how	ADV	WRB	advmod	xxx	True	True
to	to	PART	TO	aux	xx	True	True
build	build	VERB	VB	xcomp	xxxx	True	False
chatbots	chatbot	NOUN	NNS	dobj	xxxx	True	False

Lihat tabel di bawah ini untuk mengetahui arti dari setiap atribut yang kami cetak dalam kode.

TEXT	Actual text or word being processed
LEMMA	Root form of the word being processed
POS	Part-of-speech of the word
TAG	They express the part-of-speech (e.g., VERB) and some amount of morphological information (e.g., that the verb is past tense).
DEP	Syntactic dependency (i.e., the relation between tokens)
SHAPE	Shape of the word (e.g., the capitalization, punctuation, digits format)
ALPHA	Is the token an alpha character?
Stop	Is the word a stop word or part of a stop list?

Anda dapat merujuk tabel di bawah ini untuk memahami arti setiap nilai atribut POS dari objek token. Daftar ini memberikan gambaran mendetail tentang tag part-of-speech yang ditetapkan oleh model spaCy.

POS	DESCRIPTION	EXAMPLES
ADJ	adjective	<i>big, old, green, incomprehensible, first</i>
ADP	adposition	<i>in, to, during</i>
ADV	adverb	<i>very, tomorrow, down, where, there</i>
AUX	auxiliary	<i>is, has (done), will (do), should (do)</i>
CONJ	conjunction	<i>and, or, but</i>
CCONJ	coordinating conjunction	<i>and, or, but</i>
DET	determiner	<i>a, an, the</i>
INTJ	interjection	<i>psst, ouch, bravo, hello</i>
NOUN	noun	<i>girl, cat, tree, air, beauty</i>
NUM	numeral	<i>1, 2017, one, seventy-seven, IV, MMXIV</i>
PART	particle	<i>'s, not,</i>
PRON	pronoun	<i>I, you, he, she, myself, themselves, somebody</i>
PROPN	proper noun	<i>Mary, John, London, NATO, HBO</i>
PUNCT	punctuation	<i>., (,), ?</i>
SCONJ	subordinating conjunction	<i>if, while, that</i>
SYM	symbol	<i>%, \$, ©, +, -, ×, ÷, =, :, ½</i>
VERB	verb	<i>run, runs, running, eat, ate, eating</i>
X	other	<i>sfpkdspsxmsa</i>
SPACE	space	

Jadi mengapa penandaan POS diperlukan untuk chatbots?

Jawaban: untuk mengurangi kerumitan pemahaman suatu teks yang tidak dapat dilatih atau dilatih dengan kurang percaya diri. Dengan menggunakan penandaan POS, kita dapat mengidentifikasi bagian dari input teks dan melakukan pencocokan string hanya untuk bagian tersebut. Misalnya, jika Anda ingin mengetahui apakah suatu lokasi ada dalam sebuah kalimat, maka penandaan POS akan menandai kata lokasi tersebut sebagai KATA BENDA, sehingga Anda dapat mengambil semua KATA BENDA dari daftar yang diberi tag dan melihat apakah itu salah satu lokasi dari daftar prasetel Anda. atau tidak.

Stemming dan Lemmatisasi

Stemming adalah proses mereduksi kata-kata yang diinfleksikan menjadi kata dasar, bentuk dasar. Algoritme stemming mereduksi kata “mengatakan” menjadi akar kata “mengatakan”, sedangkan “mungkin” menjadi anggapan. Seperti yang Anda lihat, ini mungkin atau mungkin tidak selalu 100% benar.

Lemmatisasi erat kaitannya dengan stemming, namun lemmatisasi adalah proses algoritmik untuk menentukan lemma suatu kata berdasarkan makna yang dimaksudkan. Misalnya, dalam bahasa Inggris, kata kerja “to walk” dapat muncul sebagai “walk”, “walked”, “walks”, atau “walking”. Bentuk dasar, “berjalan”, yang mungkin dicari di kamus, disebut lemma untuk kata tersebut. spaCy tidak memiliki stemmer bawaan, karena lemmatisasi dianggap lebih tepat dan produktif.

Perbedaan Stemming dan Lemmatisasi

- Stemming melakukan tugasnya dengan cara yang kasar dan heuristik yang memotong akhir kata, dengan asumsi bahwa kata yang tersisa adalah kata yang sebenarnya kita cari, namun sering kali mencakup penghilangan imbuhan pembentukan kata.
- Lemmatisasi mencoba melakukan pekerjaan dengan lebih elegan dengan menggunakan kosa kata dan analisis morfologi kata. Ia mencoba yang terbaik untuk menghilangkan akhiran infleksional saja dan mengembalikan bentuk kamus dari sebuah kata, yang dikenal sebagai lemma.

Meskipun hanya sedikit perpustakaan yang menyediakan metode stemming dan juga lemmatisasi, praktik terbaiknya adalah menggunakan lemmatisasi untuk mendapatkan akar kata dengan benar.

Mari kita coba mendalami lemmatisasi dengan mengambil beberapa contoh:

Example 1:

```
from spacy.lemmatizer import Lemmatizer
from spacy.lang.en import LEMMA_INDEX, LEMMA_EXC, LEMMA_RULES
lemmatizer = Lemmatizer(LEMMA_INDEX, LEMMA_EXC, LEMMA_RULES)
lemmatizer('chuckles', 'NOUN') # 2nd param is token's part-of-speech tag
```

Output:

```
[u'chuckle']
```

Example 2:

```
lemmatizer('blazing', 'VERB')
```

Output:

```
[u'blaze']
```

Example 3:

```
lemmatizer('fastest', 'ADJ')
```

Output:

```
[u'fast']
```

Jika Anda ingin melihat perbandingan antara stemmer dan lemmatizer, maka Anda perlu menginstal salah satu perpustakaan paling populer untuk Python: Natural Language Toolkit (NLTK). spaCy mendapatkan popularitas baru-baru ini, tetapi NLTK-lah yang membuat setiap penggemar NLP terjun ke lautan NLP dan teknik-tekniknya.

Periksa contoh berikut di mana kami mencoba menggunakan dua teknik stemming yang disediakan oleh NLTK. Pertama, kita mencoba mendapatkan kata “tercepat” menggunakan PorterStemmer dan kemudian menggunakan SnowBallStemmer. Keduanya memberikan hasil yang sama yaitu “tercepat” namun ketika kita melakukan lemmatisasi menggunakan metode spaCy, kita mendapatkan “cepat” sebagai akar dari “tercepat”, yang lebih bermakna dan benar.

```
from nltk.stem.porter import *
from nltk.stem.snowball import SnowballStemmer
porter_stemmer = PorterStemmer()
snowball_stemmer = SnowballStemmer("english")
print(porter_stemmer.stem("fastest"))
print(snowball_stemmer.stem("fastest"))

fastest
fastest
```

Catatan Pastikan untuk menginstal paket nltk menggunakan pip3 sebelum mencoba menjalankan kode ini. Karena Anda cukup mengetahui apa yang dilakukan stemming atau lemmatization di NLP, Anda harus dapat memahami bahwa setiap kali Anda menemukan situasi di mana Anda memerlukan bentuk akar kata, Anda perlu melakukan lemmatization di sana. Misalnya saja sering digunakan dalam membangun mesin pencari. Anda pasti bertanya-tanya bagaimana Google memberi Anda artikel di hasil pencarian yang ingin Anda dapatkan meskipun teks pencarian tidak dirumuskan dengan benar.

Di sinilah seseorang memanfaatkan lemmatisasi. Bayangkan Anda menelusuri dengan teks, “Kapan Game of Thrones musim berikutnya akan dirilis?”

Sekarang, misalkan mesin pencari melakukan pencocokan frekuensi kata dokumen sederhana untuk memberi Anda hasil pencarian. Dalam kasus ini, kueri yang disebutkan di atas

mungkin tidak akan cocok dengan artikel dengan teks “Tanggal rilis Game of Thrones musim depan”.

Jika kita melakukan lemmatisasi pada pertanyaan awal sebelum mencocokkan masukan dengan dokumen, maka kita mungkin mendapatkan hasil yang lebih baik.

Kami akan mencoba menguji teori ini juga di bagian selanjutnya.

Pengakuan Entitas Bernama

Pengenalan entitas bernama (NER), juga dikenal dengan nama lain seperti identifikasi entitas atau ekstraksi entitas, adalah proses menemukan dan mengklasifikasikan entitas bernama yang ada dalam teks tertentu ke dalam kategori yang telah ditentukan sebelumnya.

Tugas NER sangat bergantung pada basis pengetahuan yang digunakan untuk melatih algoritme ekstraksi NE, sehingga tugas ini mungkin berfungsi atau tidak bergantung pada kumpulan data yang disediakan tempat ia dilatih.

spaCy hadir dengan model pengenalan entitas yang sangat cepat yang mampu mengidentifikasi frasa entitas dari dokumen tertentu. Entitas dapat memiliki jenis yang berbeda, seperti orang, lokasi, organisasi, tanggal, angka, dll. Entitas ini dapat diakses melalui properti `.ents` dari objek `doc`.

Mari kita coba menemukan entitas bernama dengan mengambil beberapa contoh dengan bantuan kemampuan penandaan NER yang kuat dari spaCy.

Example 1:

```
my_string = u"Google has its headquarters in Mountain View, California
having revenue amounted to 109.65 billion US dollars"
doc = nlp(my_string)
```

```
for ent in doc.ents:
    print(ent.text, ent.label_)
```

Output:

```
('Google', 'ORG')
('Mountain View', 'GPE')
('California', 'GPE')
('109.65 billion US dollars', 'MONEY')
```

Kita dapat melihat betapa indah dan otomatisnya model spaCy dapat dengan mudah mengidentifikasi bahwa kata Google sebagai sebuah Organisasi, California adalah entitas Geopolitik, dan dalam kalimat tersebut kita berbicara tentang 109,65 miliar dolar AS, yang sebenarnya tentang uang.

Mari kita coba jelajahi beberapa contoh lagi.

Example 2:

```
my_string = u"Mark Zuckerberg born May 14, 1984 in New York is an American
technology entrepreneur and philanthropist best known for co-founding and
leading Facebook as its chairman and CEO."
```

```
doc = nlp(my_string)

for ent in doc.ents:
    print(ent.text, ent.label_)
```

Output:

```
('Mark Zuckerberg', 'PERSON')
('May 14, 1984', 'DATE')
('New York', 'GPE')
('American', 'NORP')
('Facebook', 'ORG')
```

Example 3:

```
my_string = u"I usually wake up at 9:00 AM. 90% of my daytime goes in
learning new things."
```

```
doc = nlp(my_string)
for ent in doc.ents:
    print(ent.text, ent.label_)
```

Output:

```
('9:00 AM', 'TIME')
('90%', 'PERCENT')
```

Seperti yang Anda lihat, ekstraktor entitas dapat dengan mudah mengekstrak informasi waktu dari string yang diberikan. Juga seperti yang Anda lihat, ekstraktor entitas tidak hanya mencoba mengidentifikasi angka tetapi juga nilai PERCENTAGE yang tepat.

Sesuai dokumentasi spaCy, model yang dilatih pada korpus OntoNotes 51 mendukung tipe entitas berikut.

TYPE	DESCRIPTION
PERSON	People, including fictional
NORP	Nationalities or religious or political groups
FAC	Buildings, airports, highways, bridges, etc.
ORG	Companies, agencies, institutions, etc.
GPE	Countries, cities, states
LOC	Non-GPE locations, mountain ranges, bodies of water
PRODUCT	Objects, vehicles, foods, etc. (not services)
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK_OF_ART	Titles of books, songs, etc.
LAW	Named documents made into laws
LANGUAGE	Any named language
DATE	Absolute or relative dates or periods
TIME	Times smaller than a day
PERCENT	Percentage, including “%”
MONEY	Monetary values, including unit
QUANTITY	Measurements, as of weight or distance
ORDINAL	“first,” “second,” etc.
CARDINAL	Numerals that do not fall under another type

Kapan pun kami ingin membangun agen percakapan atau chatbot secara sederhana, kami selalu memikirkan domain. Misalnya, kita ingin chatbot membuat janji dengan dokter, memesan makanan, membayar tagihan, mengisi aplikasi perbankan, e-Commerce, dll. Chatbot juga dapat memecahkan kombinasi dari masalah-masalah ini. Dengan mengetahui entitas dalam pertanyaan, seseorang dapat memperoleh gambaran yang adil tentang konteks pertanyaan yang diajukan.

Mari kita coba memahaminya dengan mengambil contoh dua kalimat yang kata-katanya mirip dan maknanya berbeda.

```
my_string1 = u"Imagine Dragons are the best band."
my_string2 = u"Imagine dragons come and take over the city."

doc1 = nlp(my_string1)
doc2 = nlp(my_string2)

for ent in doc1.ents:
    print(ent.text, ent.label_)
```

Perulangan for di atas objek doc1 memberikan keluaran:

```
('Imagine Dragons', 'ORG')
```

Luar biasa, bukan? Ini akan menjadi lebih menarik ketika Anda menyadari bahwa pengenalan entitas tidak mengenali entitas apa pun di string ke-2. Jalankan kode berikut dan doc2 tidak menghasilkan output apa pun.

```
for ent in doc2.ents:
    print(ent.text, ent.label_)
```

Sekarang, bayangkan Anda mengekstrak konteks dari dua string di atas dalam lingkungan hidup. Apa yang akan kamu lakukan? Dengan bantuan Entity Extractor, seseorang dapat dengan mudah mengetahui konteks pernyataan dan dengan cerdas melanjutkan percakapan.

Hentikan Kata-kata

Kata-kata berhenti adalah kata-kata berfrekuensi tinggi seperti a, an, the, to dan juga yang terkadang ingin kita saring dari sebuah dokumen sebelum diproses lebih lanjut. Kata-kata berhenti biasanya memiliki sedikit kandungan leksikal dan tidak memiliki banyak makna.

Di bawah ini adalah daftar 25 kata henti semantik non-selektif yang umum di Reuters-RCV1.

```
a    an    and    are    as    at    be    by    for
from has he  in    is    it    its   of    on
that the to   was   were  will  with
```

Mari masuk ke beberapa kode dan mencoba memahami cara kerjanya. Untuk melihat semua kata yang didefinisikan sebagai kata berhenti di spaCy Anda dapat menjalankan baris kode berikut:

```
from spacy.lang.en.stop_words import STOP_WORDS
print(STOP_WORDS)
```

Anda akan melihat sesuatu seperti:

```
set(['all', 'six', 'just', 'less', 'being', 'indeed', 'over', 'move',
'anyway', 'fifty', 'four', 'not', 'own', 'through', 'using', 'go', 'only',
'its', 'before', 'one', 'whose', 'how',
.....
.....
.....
.....
'whereby', 'third', 'i', 'whole', 'noone', 'sometimes', 'well', 'together',
'yours', 'their', 'rather', 'without', 'so', 'five', 'the', 'otherwise',
'make', 'once'])
```

Ada sekitar 305 kata-kata berhenti yang didefinisikan dalam daftar kata-kata berhenti spaCy. Anda selalu dapat menentukan kata-kata berhenti Anda sendiri jika diperlukan dan mengganti daftar yang ada.

Untuk melihat apakah suatu kata merupakan stop word atau bukan, Anda dapat menggunakan objek nlp dari spaCy. Kita dapat menggunakan atribut `is_stop` objek nlp.

Example 1:

```
nlp.vocab[u'is'].is_stop
```

Output:

```
True
```

Example 2:

```
nlp.vocab[u'hello'].is_stop
```

Output:

```
False
```

Example 3:

```
nlp.vocab[u'with'].is_stop
```

Output:

```
True
```

Kata-kata berhenti adalah bagian yang sangat penting dari pembersihan teks. Ini membantu menghilangkan data yang tidak berarti sebelum kita mencoba melakukan pemrosesan sebenarnya untuk memahami teks.

Misalkan Anda berada dalam situasi di mana Anda membuat bot untuk membuat orang bahagia dengan menilai suasana hati mereka. Sekarang, kita perlu menganalisis sentimen dalam teks yang dimasukkan pengguna sehingga respons yang benar dapat dirumuskan. Di sini,

sebelum melakukan analisis sentimen dasar, kita harus menghilangkan noise pada data yang ada dalam bentuk stop word.

10.5 PENGURAIAN KETERGANTUNGAN

Penguraian ketergantungan adalah salah satu fitur spaCy yang lebih indah dan kuat yang cepat dan akurat. Parser juga dapat digunakan untuk mendeteksi batas kalimat dan memungkinkan Anda mengulangi frasa kata benda dasar, atau “potongan”.

Fitur spaCy ini memberi Anda pohon parsing yang menjelaskan hubungan orang tua-anak antara kata atau frasa dan tidak bergantung pada urutan kemunculan kata.

Mari kita ambil contoh di mana Anda harus mengurai kalimat berikut:

Pesankan saya penerbangan dari Bangalore ke Goa

Example 1:

```
doc = nlp(u'Book me a flight from Bangalore to Goa')
blr, goa = doc[5], doc[7]
list(blr.ancestors)
```

Output:

```
[from, flight, Book]
```

Output di atas dapat memberi tahu kita bahwa pengguna ingin memesan penerbangan dari Bangalore. Mari kita coba membuat daftar nenek moyang objek goa.ancestors:

```
list(goa.ancestors)
```

Output:

```
[to, flight, Book]
```

Keluaran ini dapat memberi tahu kita bahwa pengguna ingin memesan penerbangan ke Goa.

Apa Leluhur dalam Penguraian Ketergantungan?

Leluhur adalah token paling kanan dari turunan sintaksis token ini. Seperti pada contoh di atas untuk objek asal nenek moyang, penerbangan, dan Buku. Ingat Anda selalu dapat membuat daftar leluhur item objek dokumen menggunakan atribut leluhur.

List(doc[4].ancestors) #doc[4]= =flight

Kode di atas akan menampilkan:

```
[flight,Book]
```

Untuk memeriksa apakah item objek doc merupakan nenek moyang dari item objek doc lain secara terprogram, kita dapat melakukan hal berikut:

Doc[3].is_ancestor(doc[5])

Di atas mengembalikan True karena doc[3] (yaitu, penerbangan) adalah nenek moyang dari doc[5] (yaitu, Bangalore). Anda dapat mencoba lebih banyak contoh seperti ini untuk

mendapatkan pemahaman yang lebih baik tentang penguraian ketergantungan dan konsep leluhur. Jika kita mencoba memikirkan skenario dunia nyata yang mungkin kita hadapi saat mencoba membuat chatbot, kita mungkin menemukan beberapa kalimat seperti Saya ingin memesan taksi ke hotel dan meja di restoran. Dalam kalimat ini, penting untuk mengetahui tugas apa yang diminta dan di mana targetnya (yaitu, apakah pengguna ingin memesan taksi ke hotel atau restoran). Mari kita coba melakukannya menggunakan kode berikut:

Example 1:

```
doc = nlp(u'Book a table at the restaurant and the taxi to the hotel')
tasks = doc[2], doc[8] #(table, taxi)
tasks_target = doc[5], doc[11] #(restaurant, hotel)

for task in tasks_target:
    for tok in task.ancestors:
        if tok in tasks:
            print("Booking of {} belongs to {}".format(tok, task))
    break
```

Output:

```
Booking of table belongs to restaurant
Booking of taxi belongs to hotel
```

Apa Anak-anak yang Mengurai Ketergantungan?

Anak-anak adalah ketergantungan sintaksis langsung dari token. Kita dapat melihat turunan suatu kata dengan menggunakan atribut anak seperti yang kita gunakan pada leluhur.

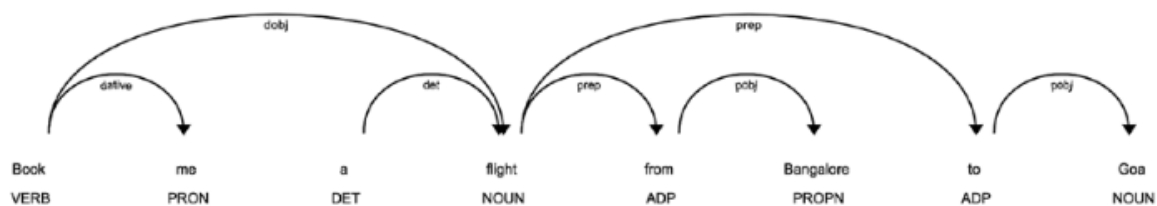
List(doc[3].children)

akan menghasilkan

[a,from,to]

10.6 VISUALISASI INTERAKTIF UNTUK PARSING KETERGANTUNGAN

Sangat sulit untuk memahami konsep penguraian ketergantungan secara lengkap untuk pertama kalinya. spaCy memberikan cara yang sangat mudah dan interaktif untuk memahami penguraian ketergantungannya. spaCy v2.0+ memiliki modul visualisasi di mana kita dapat meneruskan Doc atau daftar objek Doc ke displaCy dan memanggil metode servis displaCy untuk menjalankan server web.



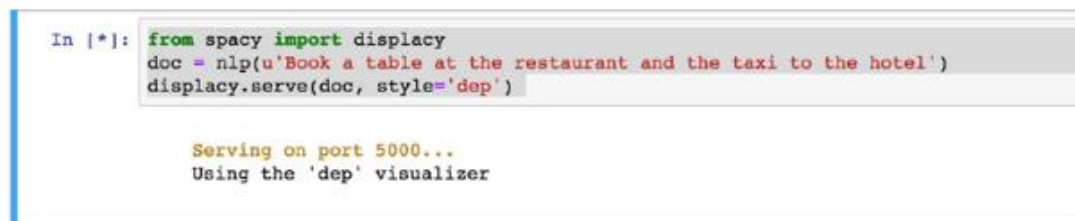
Gambar 11.5 Menunjukkan Bagaimana Visualisasi Interaktif Akan Mencari Penguraian Ketergantungan.

Anda juga dapat membuat visualisasi penguraian ketergantungan pada Gambar 10-5. Untuk membuat visualisasi seperti ini, jalankan kode berikut lalu buka <http://localhost:5000> di browser Anda.

Mari kita coba melakukan visualisasi contoh tugas dan target tugas kita.

```
from spacy import displacy
doc = nlp(u'Book a table at the restaurant and the taxi to the hotel')
displacy.serve(doc, style='dep')
```

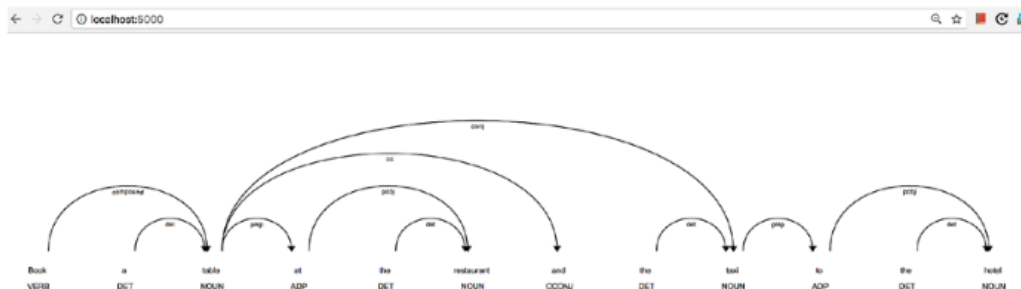
Menjalankan kode ini akan menghasilkan keluaran seperti pada Gambar 10-6. Jika Anda mendapatkan sesuatu yang serupa, buka tab lain di browser Anda dan masukkan <http://localhost:5000>. Kami mendapatkan visualisasi penguraian ketergantungan untuk string ini dalam kode (ditunjukkan pada Gambar 10.7).



```
In [*]: from spacy import displacy
doc = nlp(u'Book a table at the restaurant and the taxi to the hotel')
displacy.serve(doc, style='dep')

Serving on port 5000...
Using the 'dep' visualizer
```

Gambar 10.6. Memulai server penguraian ketergantungan di localhost



Gambar 10.7. Contoh penguraian ketergantungan

Mari kita ambil satu lagi contoh penguraian ketergantungan di mana kita berasumsi pengguna menanyakan kalimat berikut:

Apa saja tempat untuk dikunjungi di Berlin dan menginap di Lübeck?

Pertama-tama kita akan membuat objek doc seperti yang ditunjukkan di sini:

Doc = nlp(u"what are some places to visit in berlin and stay in Lubeck")

Sekarang, kami mendapatkan tempat yang sedang dibicarakan dan tindakan yang diinginkan pengguna:

Place = [doc[7], doc[11]] #[Berlin,Lubeck]

Actions=[doc[5],doc[9]] #[visit,stay]

Karena Anda sudah mengetahui penandaan POS dan ekstraksi entitas, Anda dapat dengan mudah mendapatkan tempat dan tindakan secara otomatis.

Sekarang setelah kita mengetahui tempatnya, mari kita ulangi masing-masing leluhurnya dan periksa apakah ada leluhur yang ditemukan dalam tindakan. Induk tempat pertama yang ditemukan dalam daftar tindakan haruslah tindakan untuk tempat tersebut.

```
for place in places:
    for tok in place.ancestors:
        if tok in actions:
            print("User is referring {} to {}".format(place, tok))
            break
```

Output:

```
User is referring: Berlin to visit
User is referring: Lubeck to stay
```

Seperti yang kita lihat dalam contoh ini, penguraian ketergantungan membuatnya cukup mudah untuk memahami apa yang dimaksud pengguna. Kami melihat bahwa dalam kasus dua tugas yang berbeda juga, kami dapat mengetahui ekspektasinya dan, berdasarkan hal tersebut, merumuskan respons selanjutnya.

Apa Kegunaan Parsing Ketergantungan di Chatbots?

Penguraian ketergantungan adalah salah satu bagian terpenting saat membangun chatbot dari awal. Ini menjadi jauh lebih penting ketika Anda ingin mengetahui arti masukan teks dari pengguna ke chatbot Anda. Ada kalanya Anda belum melatih chatbot Anda, namun Anda tetap tidak ingin kehilangan pelanggan atau membalas seperti mesin bodoh.

Dalam kasus ini, penguraian ketergantungan sangat membantu menemukan relasi dan menjelaskan lebih banyak tentang apa yang mungkin diminta pengguna.

Jika kami membuat daftar hal-hal yang membantu penguraian ketergantungan, beberapa di antaranya mungkin:

- Membantu dalam menemukan hubungan antara kata-kata dalam kalimat yang benar secara tata bahasa.
- Dapat digunakan untuk mendeteksi batas kalimat.
- Sangat berguna untuk mengetahui apakah pengguna berbicara tentang lebih dari satu konteks secara bersamaan.

Anda perlu menulis NLP khusus Anda sendiri untuk memahami konteks pengguna atau chatbot Anda dan, berdasarkan itu, mengidentifikasi kemungkinan kesalahan tata bahasa yang dapat dilakukan pengguna.

Secara keseluruhan, Anda harus siap menghadapi skenario di mana pengguna akan memasukkan nilai sampah atau kalimat yang salah secara tata bahasa. Anda tidak dapat menangani semua skenario tersebut sekaligus, namun Anda dapat terus meningkatkan chatbot

Anda dengan menambahkan kode NLP khusus atau dengan membatasi input pengguna berdasarkan desain.

10.7 POTONGAN KATA BENDA

Potongan kata benda atau NP-chunking pada dasarnya adalah “frasa kata benda dasar”. Kita dapat mengatakan bahwa itu adalah frasa datar yang memiliki kata benda sebagai kepalanya. Anda dapat menganggap potongan kata benda sebagai kata benda dengan kata-kata yang mendeskripsikan kata benda tersebut.

Mari kita coba mengambil contoh dan memahaminya dengan lebih baik.

Example 1:

```
doc = nlp(u"Boston Dynamics is gearing up to produce thousands of robot
dogs")
list(doc.noun_chunks)
```

Output:

```
[Boston Dynamics, thousands, robot dogs]
```

Meskipun memiliki potongan kata benda dari kalimat tertentu sangat membantu, spaCy menyediakan atribut lain yang juga dapat membantu. Mari kita coba jelajahi beberapa di antaranya.

Example 2:

```
doc = nlp(u"Deep learning cracks the code of messenger RNAs and protein-
coding potential")
for chunk in doc.noun_chunks:
    print(chunk.text, chunk.root.text, chunk.root.dep_,
          chunk.root.head.text)
```

Output:

TEXT	ROOT.TEXT	ROOT.DEP_	ROOT.HEAD.TEXT
deep learning	learning	nsubj	cracks
the code	code	dobj	cracks
messenger RNAs	RNAs	pobj	of
protein-coding potential	potential	conj	RNAs

Seperti yang bisa kita lihat dari tabel ini, kita mendapatkan potongan kata benda dan atributnya. Tabel berikut akan membantu Anda memahami setiap kolom.

Column	Meaning
Text	Text of the original noun chunk
Root text	Text of the original word that connects the noun chunk with remaining parse
Root dep	Dependency relation that connects the root to its head
Root head text	Text of the root token's head

10.8 MENEMUKAN KESAMAAN

Menemukan kesamaan antara dua kata adalah kasus penggunaan yang sering Anda temukan saat bekerja dengan NLP. Terkadang menjadi sangat penting untuk mengetahui apakah dua kata itu serupa. Saat membuat chatbot, Anda akan sering menghadapi situasi di mana Anda tidak hanya perlu menemukan kata-kata yang mirip tetapi juga seberapa dekat keterkaitan dua kata secara logis.

spaCy menggunakan vektor kata berkualitas tinggi untuk menemukan kesamaan antara dua kata menggunakan algoritma GloVe (Global Vectors for Word Representation). GloVe adalah algoritma pembelajaran tanpa pengawasan untuk mendapatkan representasi vektor untuk kata-kata. Algoritme GloVe menggunakan statistik kemunculan kata-kata global gabungan dari korpus untuk melatih model. Mari kita coba melihat nilai sebenarnya di dalam vektor spaCy menggunakan atribut vektor dari token.

```
doc = nlp(u'How are you doing today?')
for token in doc:
    print(token.text, token.vector[:5])
```

Output:

```
(u'How', array([-0.29742685,  0.73939574,
-0.04001453,  0.44034013,  2.8967502 ],      dtype=float32))
(u'are', array([-0.23435134, -1.6145049 ,  1.0197453 ,  0.9928169 ,
 0.28227055],      dtype=float32))(u'you', array([ 0.10252178,
-3.564711 ,  2.4822793 ,  4.2824993 ,  3.590245 ],      dtype=float32))
(u'doing', array([-0.6240922 , -2.0210216 , -0.91014993,  2.7051923 ,
 4.189252 ],      dtype=float32))(u'today', array([ 3.5409122 ,
-0.62185854,  2.6274266 ,  2.0504875 ,  0.20191991],      dtype=float32))
(u'?', array([ 2.8914998 , -0.25079122,  3.3764176 ,  1.6942682 ,
 1.9849057 ],      dtype=float32))
```

Melihat keluaran ini, tidak banyak masuk akal dan bermakna. Dari sudut pandang aplikasi, yang paling penting adalah seberapa mirip vektor-vektor dari kata-kata yang berbeda yaitu, makna kata itu sendiri.

Untuk menemukan persamaan antara dua kata di spaCy, kita dapat melakukan hal berikut.

Example 1:

```
hello_doc = nlp(u"hello")
hi_doc = nlp(u"hi")
hella_doc = nlp(u"hella")
print(hello_doc.similarity(hi_doc))
print(hello_doc.similarity(hella_doc))
```

Output:

```
0.7879069442766685
0.4193425861242359 ,
```

Jika melihat kata hello lebih berkaitan dan mirip dengan kata hi, padahal hanya ada perbedaan karakter antara kata hello dan hella. Mari kita ambil satu contoh kalimat lagi dan pelajari bagaimana spaCy melakukan perbandingan kesamaan. Ingat contoh Game of Thrones di bagian sebelumnya? Kami akan mencobanya dan melihat kesamaannya.

Code:

```
GoT_str1 = nlp(u"When will next season of Game of Thrones be releasing?")
GoT_str2 = nlp(u"Game of Thrones next season release date?")
GoT_str1.similarity(GoT_str2)
```

Output:

```
0.785019122782813
```

Seperti yang bisa kita lihat dalam contoh ini, kemiripan antara kedua kalimat tersebut adalah sekitar 79%, yang cukup untuk mengatakan bahwa kedua kalimat tersebut cukup mirip, dan itu memang benar. Ini dapat membantu kami menghemat banyak waktu untuk menulis kode khusus saat membuat chatbot. Jadi, kita sampai pada fakta bahwa spaCy memberi kita kesamaan yang bermakna antara dua kata menggunakan vektor kata, bukan hanya melihat ejaan atau abjadnya.

Kami akan mengambil contoh yang sangat sederhana dan mencoba menemukan kesamaan antar kata.

```
example_doc = nlp(u"car truck google")

for t1 in example_doc:
    for t2 in example_doc:
        similarity_perc = int(t1.similarity(t2) * 100)
        print "Word {} is {}% similar to word {}".format(t1.text,
            similarity_perc, t2.text)
```

Output:

```

Word car is 100% similar to word car
Word car is 71% similar to word truck
Word car is 24% similar to word google
Word truck is 71% similar to word car
Word truck is 100% similar to word truck
Word truck is 36% similar to word google
Word google is 24% similar to word car
Word google is 36% similar to word truck
Word google is 100% similar to word google

```

Menemukan kesamaan antara kata atau kalimat menjadi sangat penting ketika kita ingin membangun aplikasi apa pun yang sangat bergantung pada implementasi NLP. Jika Anda pernah menggunakan StackOverFlow, setiap kali kami mencoba mengajukan pertanyaan baru, ia mencoba membuat daftar pertanyaan serupa yang sudah ditanyakan di platform. Ini adalah salah satu contoh terbaik di mana menemukan kesamaan antara dua rangkaian kalimat mungkin bisa membantu. Keyakinan spaCy untuk menemukan kesamaan antara dua kata berdasarkan model yang sudah dilatih sepenuhnya bergantung pada jenis asumsi umum yang diambil.

Saat membuat chatbot, menemukan kesamaan bisa sangat berguna untuk situasi berikut:

- Saat membuat chatbot untuk mendapatkan rekomendasi
- Menghapus duplikat
- Membangun pemeriksa ejaan

Hal-hal yang kami pelajari ini sangat penting saat membangun chatbot sehingga kami mengetahui cara mengurai masukan pengguna agar masuk akal saat menulis logika bisnis di dalam kode.

10.9 SENANG MENGETAHUI HAL-HAL DI NLP UNTUK CHATBOTS

Di bagian ini kita akan mempelajari beberapa topik menarik yang sering berguna ketika Anda berencana menulis metode NLP kustom Anda sendiri untuk menangani skenario tertentu. Pastikan Anda mempelajarinya, karena saat Anda paling tidak mengharapkannya, hal itu paling dibutuhkan. Kami akan membahas secara singkat tentang tokenisasi dan penggunaan ekspresi reguler dalam skenario chatbot.

Tokenisasi

Tokenisasi adalah salah satu konsep NLP yang sederhana namun mendasar di mana kami membagi teks menjadi segmen-segmen yang bermakna. spaCy pertama-tama memberi token pada teks (yaitu, mengelompokkannya menjadi kata-kata, lalu tanda baca dan hal lainnya). Sebuah pertanyaan mungkin muncul di benak Anda: Mengapa saya tidak bisa menggunakan metode `split` bawaan bahasa Python dan melakukan tokenisasi? Metode pemisahan Python hanyalah metode mentah untuk membagi kalimat menjadi token yang diberi pemisah. Itu tidak

memperhitungkan makna apa pun, sedangkan tokenisasi mencoba mempertahankan maknanya juga. Mari kita coba beberapa kode dan lihat cara kerja tokenisasi.

Example 1:

```
doc = nlp(u'Brexit is the impending withdrawal of the U.K. from the
European Union.')
for token in doc:
    print(token.text)
```

Output:

```
Brexit
is
the
the
impending
withdrawal
of
the
U.K.
from
the
EuropeanUnion
```

Jika Anda melihat output di atas, Inggris muncul sebagai satu kata setelah proses tokenisasi, yang masuk akal, karena Inggris adalah nama negara dan pemisahannya akan salah. Bahkan setelah ini jika Anda tidak puas dengan tokenisasi spaCy, Anda dapat menggunakan metode `add_special_case` untuk menambahkan aturan Anda sendiri sebelum sepenuhnya mengandalkan metode tokenisasi spaCy.

Ekspresi Reguler

Anda pasti sudah mengetahui tentang ekspresi reguler dan penggunaannya. Buku ini berasumsi Anda pasti sudah familiar dengan ekspresi reguler secara umum. Di bagian ini, kita hanya akan melihat beberapa contoh dan melihat bagaimana ekspresi reguler dapat bermanfaat dan bermanfaat saat membangun chatbot. Analisis dan pemrosesan teks merupakan subjek besar tersendiri. Terkadang kata-kata bermain bersama sedemikian rupa sehingga sangat sulit bagi mesin untuk memahami dan melatihnya.

Ekspresi reguler dapat berguna untuk beberapa penggantian model pembelajaran mesin. Ia memiliki kekuatan pencocokan pola, yang dapat memastikan bahwa data yang kita proses benar atau salah. Sebagian besar chatbot awal yang dibahas di Bab 1 pada bagian Sejarah Chatbots sangat bergantung pada pencocokan pola.

Mari kita ambil dua contoh di bawah ini yang cukup mudah dipahami. Kami akan mencoba menggunakan ekspresi reguler untuk mengekstrak informasi dari kedua kalimat. Pesankan saya metro dari Stasiun Bandara ke Stasiun Hong Kong. Pesankan saya taksi dari Bandara Hong Kong ke AsiaWorld-Expo. Ini kodenya:

```

sentence1 = "Book me a metro from Airport Station to Hong Kong Station."
sentence2 = "Book me a cab to Hong Kong Airport from AsiaWorld-Expo."

import re
from_to = re.compile('.* from (.*?) to (.*?)')
to_from = re.compile('.* to (.*?) from (.*?)')

from_to_match = from_to.match(sentence2)
to_from_match = to_from.match(sentence2)

if from_to_match and from_to_match.groups():
    _from = from_to_match.groups()[0]
    _to = from_to_match.groups()[1]
    print("from_to pattern matched correctly. Printing values\n")
    print("From: {}, To: {}".format(_from, _to))

elif to_from_match and to_from_match.groups():
    _to = to_from_match.groups()[0]
    _from = to_from_match.groups()[1]

    print("to_from pattern matched correctly. Printing values\n")
    print("From: {}, To: {}".format(_from, _to))

```

Output:

```

to_from pattern matched correctly. Printing values
From: AsiaWorld-Expo., To: Hong Kong Airport

```

Coba ubah kalimat2 menjadi kalimat1 dan lihat apakah kodenya berfungsi dengan baik untuk mengidentifikasi polanya. Mengingat kekuatan pembelajaran mesin saat ini, ekspresi reguler dan pencocokan pola sudah mengalami kemunduran, namun pastikan Anda mempelajarinya sedikit karena hal ini mungkin diperlukan kapan saja untuk mengurai detail spesifik dari kata, kalimat, atau teks dokumen.

BAB 11

INTEGRASI CHATBOT PADA TWEETER

Dalam bab ini, kita akan mengeksplorasi bagaimana chatbots dapat diintegrasikan ke dalam Twitter, sebuah platform media sosial. Ada banyak platform media sosial serupa, seperti Facebook dan LinkedIn, yang melayani konteks dan pasar berbeda. Apa yang akan kita jelajahi dalam bab ini secara konseptual berlaku untuk semua platform ini.

Twitter adalah platform berita dan media sosial online tempat pengguna dapat memposting dan berinteraksi dengan pesan yang disebut tweet. Ini memiliki sekitar 328 juta pengguna aktif bulanan di seluruh dunia, dengan sekitar 317 juta pengguna aktif bulanan men-tweet sekitar 500 juta tweet setiap hari (Statista 2017). Twitter digunakan oleh bisnis sebagai alat untuk keterlibatan merek, pengumuman produk, dan layanan pelanggan.

Dalam bab ini, kita akan mempelajari cara membangun chatbots di Twitter sehingga tweet dapat diposting dan ditanggapi secara otomatis dan cepat. Kami akan membangun bot berita di platform Twitter yang men-tweet berita terkini di timeline-nya secara teratur. Ini juga akan mempersonalisasi pengalaman para pengikutnya dengan mengirim mereka berita dari topik yang mereka minati. Pertama-tama kami akan menjelajahi API Twitter dan membangun modul inti untuk mengirim tweet, mencari, dan me-retweet. Kemudian, kita akan menjelajahi sumber data berita di seluruh dunia. Kami kemudian akan membuat bot sederhana yang men-tweet berita utama di timeline-nya. Terakhir, kami akan membangun bot berita yang dipersonalisasi yang mendengarkan tweet masuk dari pengguna tentang topik yang mereka minati dan men-tweet berita spesifik pengguna kepada mereka.

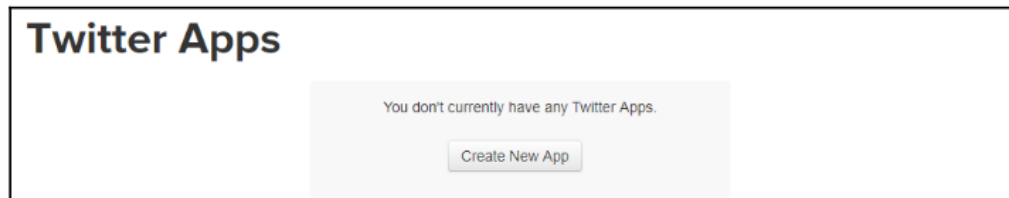
Pada akhir bab ini, Anda akan dapat:

- Pahami dasar-dasar Twitter API
- Buat bot yang mendengarkan hashtag
- Bangun bot Twitter yang men-tweet dan me-retweet
- Integrasikan NewsAPI dan tweet berita utama
- Integrasikan MongoDB untuk melacak minat pengguna
- Bangun bot percakapan untuk menarik minat pengguna dan menyajikan berita yang dipersonalisasi
- Gunakan Penjadwal Heroku untuk men-tweet berita yang dipersonalisasi secara teratur

11.1 MEMULAI DENGAN APLIKASI TWITTER

Untuk memulai, mari kita jelajahi platform pengembang Twitter. Mari kita mulai dengan membuat aplikasi Twitter dan kemudian mengeksplorasi bagaimana kita dapat men-tweet artikel berita ke pengikut berdasarkan minat mereka:

1. Masuk ke Twitter di www.twitter.com. Jika Anda belum memiliki akun di Twitter, buatlah akun.
2. Kunjungi <https://apps.twitter.com/> yang merupakan dasbor pengelolaan aplikasi Twitter.
3. Klik tombol Buat Aplikasi Baru:



Gambar 11.1 pengelolaan aplikasi twitter

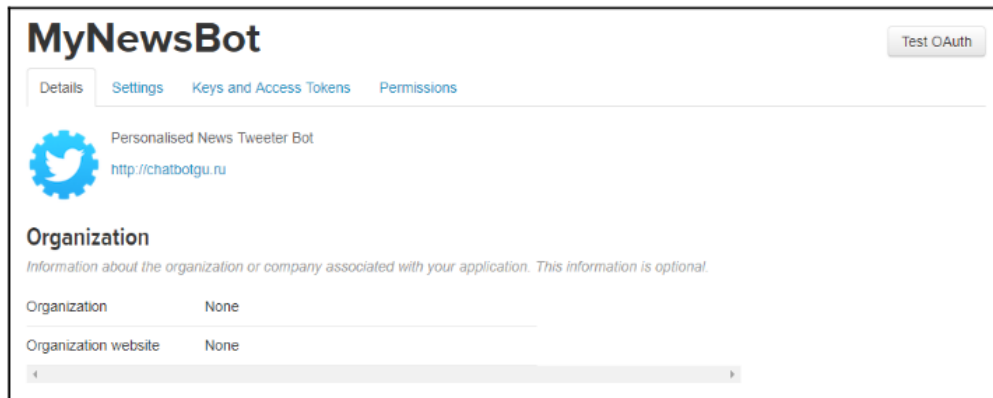
4. Buat aplikasi dengan mengisi formulir yang berisi nama, deskripsi, dan situs web (URL lengkap). Baca dan setuju Perjanjian Pengembang dan tekan Buat aplikasi Twitter Anda:

 A screenshot of the "Create an application" form on Twitter. The form is titled "Create an application" and is divided into "Application Details". It contains four main sections:

- Name ***: A text input field containing "MyNewsBot". Below it, a note says: "Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters max."
- Description ***: A text input field containing "Personalised News Tweeter Bot". Below it, a note says: "Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max."
- Website ***: A text input field containing "http://chatbotgu.ru". Below it, a note says: "Your application's publicly accessible home page, where users can go to download, make use of, or find out more information about your applications. This fully-qualified URL is used in the source attribution for tweets created by your application and will be shown in user-facing authorization screens. (If you don't have a URL, yet, just put a placeholder here but remember to change it later.)"
- Callback URL**: An empty text input field. Below it, a note says: "Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback URL on the request token step, regardless of the value given here. To restrict your application from using callbacks, leave this field blank."

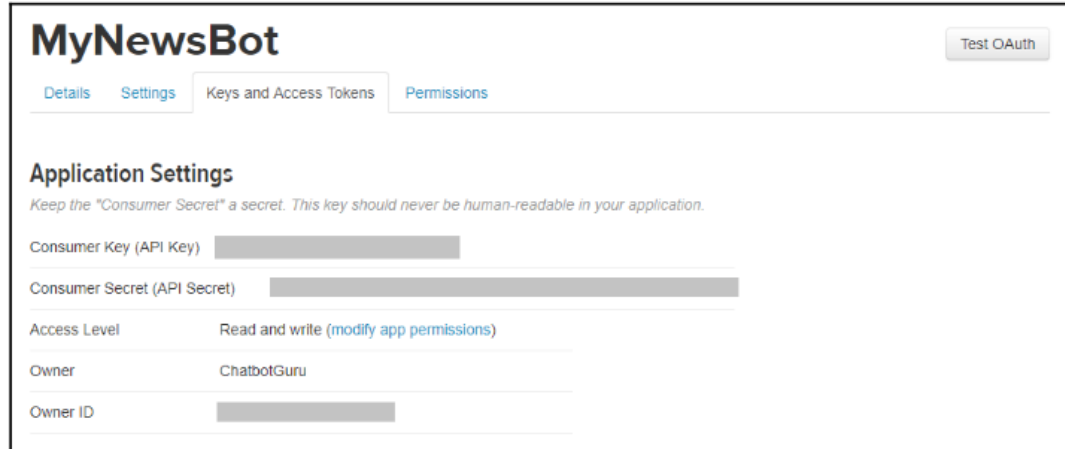
Gambar 11.2 membuat aplikasi di tweeter

5. Anda sekarang akan melihat dashboard aplikasi Anda. Jelajahi tab:



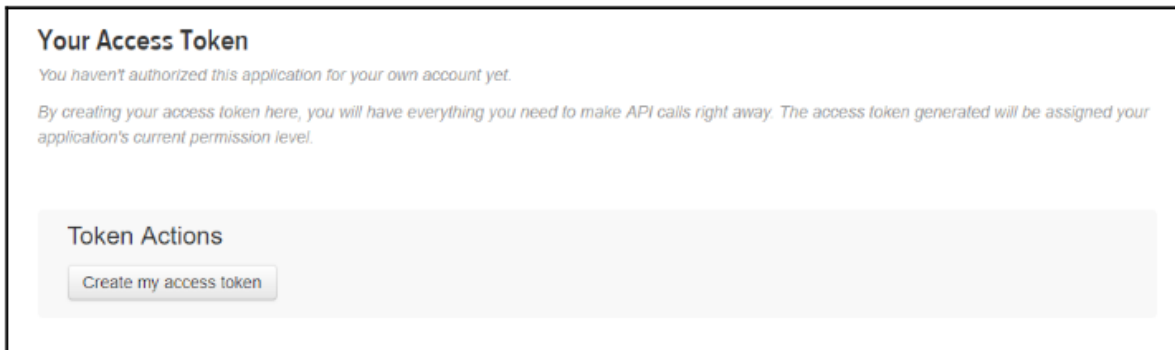
Gambar 11.3 details dashboard MyNewsBot

6. Klik Kunci dan Token Akses:



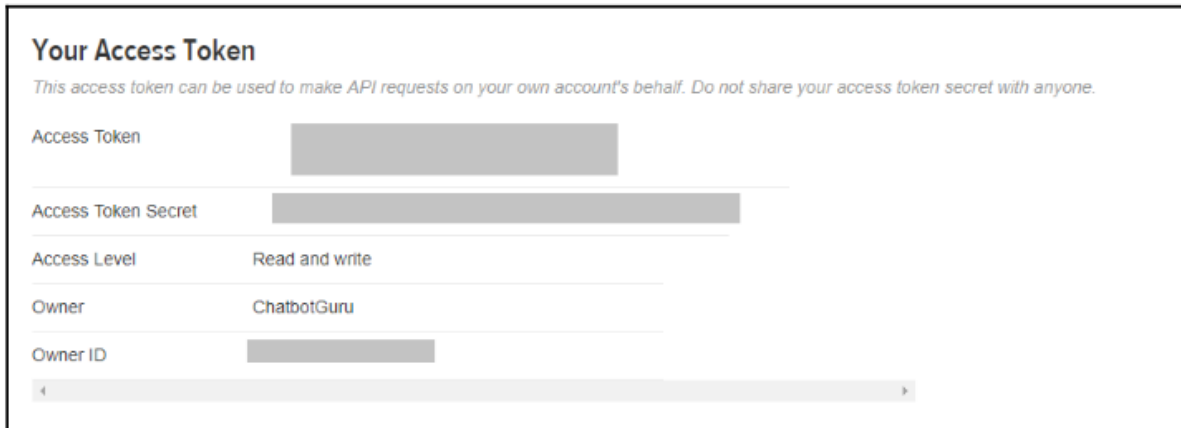
Gambar 11.4 keys and access token

7. Salin kunci konsumen dan rahasia konsumen dan simpanlah.
8. Gulir ke bawah ke Token Akses Anda:



Gambar 11.5 tampilan token akses

9. Klik Buat token akses saya untuk membuat token baru untuk aplikasi Anda:



Gambar 11.6 Membuat Token Akses Baru

10. Salin Access Token dan Access Token Secret dan simpan di sana.

Sekarang, kita memiliki semua kunci dan token yang kita perlukan untuk membuat aplikasi Twitter.

11.2 MEMBANGUN BOT TWITTER PERTAMA ANDA

Mari kita membuat bot Twitter sederhana. Bot ini akan mendengarkan tweet dan memilih tweet yang memiliki hashtag tertentu. Semua tweet dengan hashtag tertentu akan dicetak di konsol. Ini adalah bot yang sangat sederhana untuk membantu kami memulai. Di bagian berikut, kita akan menjelajahi bot yang lebih kompleks.

1. Buka direktori root dan buat program Node.js baru menggunakan npm init:

```
C:\Users\Srini\Dropbox\_Book\workspace\twitterbot>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (twitterbot)
version: (1.0.0)
description: my news bot
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\Users\Srini\Dropbox\_Book\workspace\twitterbot\package.json:
{
  "name": "twitterbot",
  "version": "1.0.0",
  "description": "my news bot",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Gambar 11.7 membuat program Node.js dengan npm init

2. Jalankan perintah `npm install twitter --save` untuk menginstal perpustakaan Node.js Twitter:

```
C:\Users\Srini\Dropbox\_Book\workspace\twitterbot>npm install twitter --save
npm WARN package.json twitterbot@1.0.0 No repository field.
npm WARN package.json twitterbot@1.0.0 No README data
twitter@1.7.1 node_modules\twitter
├── deep-extend@0.5.0
├── request@2.81.0 (aws-sign2@0.6.0, oauth-sign@0.8.2, forever-agent@0.6.1, tunnel-agent@0.6.0, is-typedarray@1.0.0, caseless@0.12.0, safe-buffer@5.1.1, stringstream@0.0.5, isstream@0.1.2, aws4@1.6.0, json-stringify-safe@5.0.1, extend@3.0.1, performance-now@0.2.0, uuid@3.1.0, qs@6.4.0, combined-stream@1.0.5, mime-types@2.1.16, tough-cookie@2.3.2, form-data@2.1.4, hawk@3.1.3, http-signature@1.1.1, har-validator@4.2.1)
```

Gambar 11.8 Npm install request

Jalankan `npm install request --save` untuk menginstal perpustakaan Permintaan juga. Kami akan menggunakan ini di masa depan untuk membuat permintaan HTTP GET ke sumber data berita.

3. Jelajahi file `package.json` Anda di direktori root:

```
{
  "name": "twitterbot",
  "version": "1.0.0",
  "description": "my news bot",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "request": "^2.81.0",
    "twitter": "^1.7.1"
  }
}
```

4. Buat file `index.js` dengan kode berikut:

```
//index.js

var TwitterPackage = require('twitter');
var request = require('request');

console.log("Hello World! I am a twitter bot!");
```

```

var secret = {
  consumer_key: 'YOUR_CONSUMER_KEY',
  consumer_secret: 'YOUR_CONSUMER_SECRET',
  access_token_key: 'YOUR_ACCESS_TOKEN_KEY',
  access_token_secret: 'YOUR_ACCESS_TOKEN_SECRET'
}

```

```
var Twitter = new TwitterPackage(secret);
```

Pada kode sebelumnya, masukkan kunci dan token yang Anda simpan ke dalam variabel yang sesuai. Kami belum memerlukan paket permintaan, tetapi nanti kami akan membutuhkannya.

5. Sekarang mari kita buat pendengar hashtag untuk mendengarkan tweet pada hashtag tertentu:

```

//Twitter stream

var hashtag = '#brexit'; //put any hashtag to listen e.g. #brexit
console.log('Listening to:' + hashtag);
Twitter.stream('statuses/filter', {track: hashtag},
function(stream) {
  stream.on('data', function(tweet) {
    console.log('Tweet:@' + tweet.user.screen_name +
      '\t' + tweet.text);
    console.log('-----')
  });
  stream.on('error', function(error) {
    console.log(error);
  });
});

```

Ganti #brexit dengan hashtag yang ingin Anda dengarkan. Gunakan yang populer sehingga Anda dapat melihat kode beraksi.

6. Jalankan file index.js dengan perintah node index.js.
7. Anda akan melihat aliran tweet dari pengguna Twitter di seluruh dunia yang menggunakan hashtag:



```

C:\Users\Srini\Dropbox\Book\workspace\twitterbot>node index.js
Hello World! I am a twitter bot!
Listening to:#brexit
Tweet:@belanisiya RT @welveconomics: The UK economy grew faster after joining the EU but populism (no facts!) made people believe the opposite...
-----
Tweet:@BazzieSmith RT @jlivingstone100: Brexit support fading it seems -'Marketplace did not encounter any farmers at the show who admitted voting for Brexit...
-----
Tweet:@frankietaggart RT @brexitcountdown1: Brexit is 13850 hours away. #brexit
-----
Tweet:@pm_kristin RT @laute_europaeer: Ist nach Abschluss der #Brexit -Verhandlungen ein 2. Referendum über das Abkommen zwischen #UK und #EU notwendig?
http://t.co/...
-----
Tweet:@GillianAdams RT @nickreeves9876: Shocking how seldom the question of Freedom of Movement is couched in terms of the loss of our Right to live & work in...
-----
Tweet:@hpw_1lp RT @TheLawSociety: What is the European Court of Justice and why does it matter? A Brexit Q&A https://t.co/COVi6J0fm4 https://t.co/BankRoV...
-----
Tweet:@h1llbillies RT @LeedsEurope: Leeds For Europe Stop Brexit Day of Action! 16 Sep - Rally, Question Time & Social with @RCorbettMEP @emmyzen & mor...
-----
Tweet:@KarenMc10 RT @trevdick: Britain 2017 with Brexit looming is peddling backwards to 1971.
Memories are not made of this...
#Brexit is Barmy https://t...

```

Gambar 11.8 Aliran Tweet Dengan Hastag Dari Para Pengguna Twiter

Selamat! Anda telah membuat bot Twitter pertama Anda. Kita akan menggunakan modul mendengarkan hashtag nanti di bab ini untuk membangun bot yang lebih kompleks.

11.3 MENJELAJAHI SDK TWITTER

Di bagian sebelumnya, kita mempelajari cara mendengarkan tweet berdasarkan hashtag. Sekarang mari kita jelajahi Twitter SDK untuk memahami kemampuan yang dapat kita berikan pada bot Twitter kita.

Memperbarui status Anda

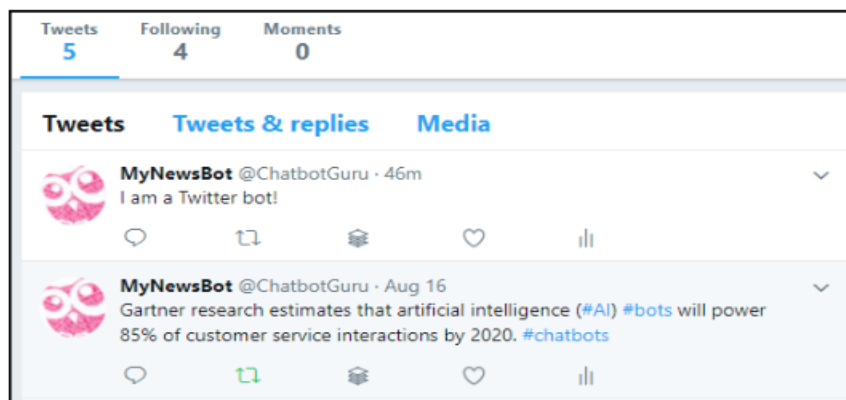
Anda juga dapat memperbarui status di timeline Twitter Anda dengan menggunakan kode modul pembaruan status berikut:

```
tweet ('I am a Twitter Bot!', null, null);

function tweet(statusMsg, screen_name, status_id){

  console.log('Sending tweet to: ' + screen_name);
  console.log('In response to:' + status_id);
  var msg = statusMsg;
  if (screen_name != null){
    msg = '@' + screen_name + ' ' + statusMsg;
  }
  console.log('Tweet:' + msg);
  Twitter.post('statuses/update', {
    status: msg
  }, function(err, response) {
    // if there was an error while tweeting
    if (err) {
      console.log('Something went wrong while TWEETING...');
      console.log(err);
    }
    else if (response) {
      console.log('Tweeted!!!');
      console.log(response)
    }
  });
}
```

Komentari kode pendengar hashtag dan tambahkan kode pembaruan status sebelumnya dan jalankan. Saat dijalankan, bot Anda akan memposting tweet di timeline Anda:



Gambar 11.9 pembaruan status di timeline

Selain men-tweet di timeline Anda, Anda juga dapat men-tweet sebagai tanggapan terhadap tweet lain (atau pembaruan status). Argumen `screen_name` digunakan untuk membuat respons. `screen_name` adalah nama pengguna yang memposting tweet. Kita akan membahasnya nanti.

Retweet ke pengikut Anda

Anda dapat me-retweet tweet ke pengikut Anda menggunakan kode status retweet berikut:

```
var retweetId = '899681279343570944';
retweet(retweetId);

function retweet(retweetId){
  Twitter.post('statuses/retweet/', {
    id: retweetId
  }, function(err, response) {
    if (err) {
      console.log('Something went wrong while RETWEETING...');
      console.log(err);
    }
    else if (response) {
      console.log('Retweeted!!!');
      console.log(response)
    }
  });
}
```

Mencari Tweet

Anda juga dapat mencari tweet terbaru atau populer dengan hashtag menggunakan kode hashtag pencarian berikut:

```
search('#brexit', 'popular')
function search(hashtag, resultType){
  var params = {
    q: hashtag, // REQUIRED
    result_type: resultType,
    lang: 'en'
  }

  Twitter.get('search/tweets', params, function(err, data) {
    if (!err) {
      console.log('Found tweets: ' + data.statuses.length);
      console.log('First one: ' + data.statuses[1].text);
    }
    else {
      console.log('Something went wrong while SEARCHING...');
    }
  });
}
```

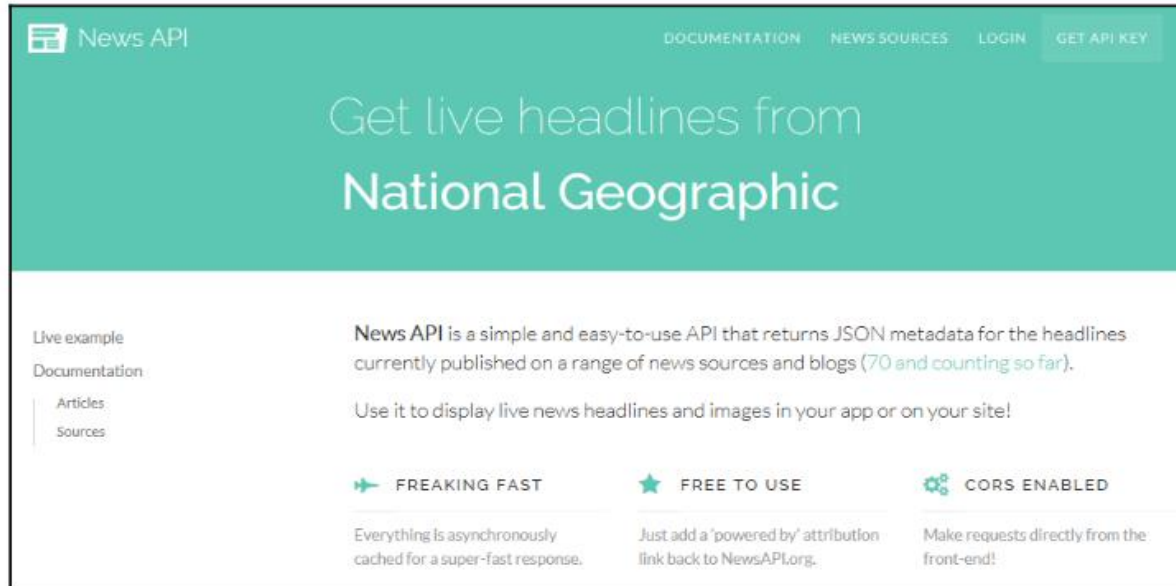
Menjelajahi Layanan Data Berita

Sekarang mari kita buat bot yang akan men-tweet artikel berita ke pengikutnya secara berkala. Kami kemudian akan memperluasnya agar dapat dipersonalisasi oleh pengguna melalui percakapan yang terjadi melalui pesan langsung dengan bot. Untuk membangun bot berita, kita memerlukan sumber dimana kita bisa mendapatkan artikel berita. Kita akan menjelajahi layanan berita bernama NewsAPI.org di bagian ini. NewsAPI adalah layanan yang mengumpulkan artikel berita dari sekitar 70 surat kabar di seluruh dunia.

Menyiapkan NewsAPI

Mari kita buat akun dengan layanan data NewsAPI dan dapatkan kunci API:

1. Kunjungi newsapi.org:



Gambar 11.10 Layanan Data Newsapi

2. Klik Dapatkan kunci API.
3. Daftar menggunakan email Anda.
4. Dapatkan kunci API Anda.
5. Jelajahi sumbernya:

[http:// newsapi.org/v1/sources?apiKey=YOUR_API_KEY](http://newsapi.org/v1/sources?apiKey=YOUR_API_KEY).

Terdapat sekitar 70 sumber dari seluruh dunia termasuk yang populer seperti BBC News, Associated Press, Bloomberg, dan CNN. Anda mungkin memperhatikan bahwa setiap sumber memiliki tag kategori yang dilampirkan. Pilihan yang memungkinkan adalah: bisnis, hiburan, permainan, umum, musik, politik, sains dan alam, olahraga, dan teknologi. Anda mungkin juga memperhatikan bahwa setiap sumber juga memiliki tag bahasa (en, de, fr) dan negara (au, de, gb, in, it, us). Berikut informasi dari sumber BBC-News:

```
{
  "id": "bbc-news",
  "name": "BBC News",
  "description": "Use BBC News for up-to-the-minute news,
  breaking news, video, audio and feature stories.
  BBC News provides trusted World and UK news as well as
  local and regional perspectives. Also entertainment,
  business, science, technology and health news.",
```

```

"url": "http://www.bbc.co.uk/news",
"category": "general",
"language": "en",
"country": "gb",
"urlsToLogos": {
  "small": "",
  "medium": "",
  "large": ""
},
"sortByAvailable": [
  "top"
]
}

```

6. Dapatkan sumber untuk kategori, bahasa, atau negara tertentu menggunakan:

https://newsapi.org/v1/sources?category=business&apiKey=YOUR_API_KEY

Berikut ini adalah bagian respons terhadap kueri sebelumnya yang menanyakan semua sumber dalam kategori bisnis:

```

"sources": [
  {
    "id": "bloomberg",
    "name": "Bloomberg",
    "description": "Bloomberg delivers business

```

```

and markets news, data, analysis, and video
to the world, featuring stories from Businessweek
and Bloomberg News.",
"url": "http://www.bloomberg.com",
"category": "business",
"language": "en",
"country": "us",
"urlsToLogos": {
  "small": "",
  "medium": "",
  "large": ""
},
"sortByAvailable": [
  "top"
]
},
{
  "id": "business-insider",
  "name": "Business Insider",
  "description": "Business Insider is a fast-growing
business site with deep financial, media, tech, and
other industry verticals. Launched in 2007, the
site is now the largest business news site on the web.",

  "url": "http://www.businessinsider.com",
  "category": "business",
  "language": "en",
  "country": "us",
  "urlsToLogos": {
    "small": "",
    "medium": "",
    "large": ""
  },
  "sortByAvailable": [
    "top",
    "latest"
  ]
},
...
]

```

7. Jelajahi artikel:

https://newsapi.org/v1/articles?source=bbc-news&apiKey=YOUR_API_KEY

https://newsapi.org/v1/articles?source=bbc-news&apiKey=YOUR_API_KEY

Berikut contoh tanggapannya:

```

"articles": [
  {
    "author": "BBC News",
    "title": "US Navy collision: Remains found in
             hunt for missing sailors",
    "description": "Ten US sailors have been missing since Monday's
                  collision with a tanker near Singapore.",
    "url": "http://www.bbc.co.uk/news/world-us-canada-41013686",
    "urlToImage":
    "https://ichef1.bbci.co.uk/news/1024/cpsprodpb/80D9/
           production/_97458923_mediaitem97458918.jpg",
    "publishedAt": "2017-08-22T12:23:56Z"
  },
  {
    "author": "BBC News",
    "title": "Afghanistan hails Trump support in 'joint struggle'",
    "description": "President Ghani thanks Donald Trump for
                  supporting Afghanistan's battle against the
                  Taliban.",
    "url": "http://www.bbc.co.uk/news/world-asia-41012617",
    "urlToImage":
    "https://ichef.bbci.co.uk/images/ic/1024x576/p05d08pf.jpg",
    "publishedAt": "2017-08-22T11:45:49Z"
  },
  ...
]

```

Untuk setiap artikel, penulis, judul, deskripsi, url, urlToImage,, dan kolom diterbitkanDi disediakan. Sekarang kita telah menjelajahi sumber data berita yang menyediakan berita terkini dalam berbagai kategori, mari kita lanjutkan dengan membuat bot berita.

Membangun bot berita Twitter

Kini setelah kita menjelajahi NewsAPI, sumber data untuk pembaruan berita terkini, dan sedikit tentang apa yang dapat dilakukan oleh API Twitter, mari kita gabungkan keduanya untuk membuat bot yang men-tweet berita menarik, pertama di timeline-nya sendiri, lalu secara spesifik kepada setiap pengikutnya:

1. Mari kita buat modul tweeter berita yang men-tweet artikel berita teratas berdasarkan sumbernya. Kode berikut menggunakan fungsi tweet() yang kita buat sebelumnya:

```
topNewsTweeter('cnn', null);
```

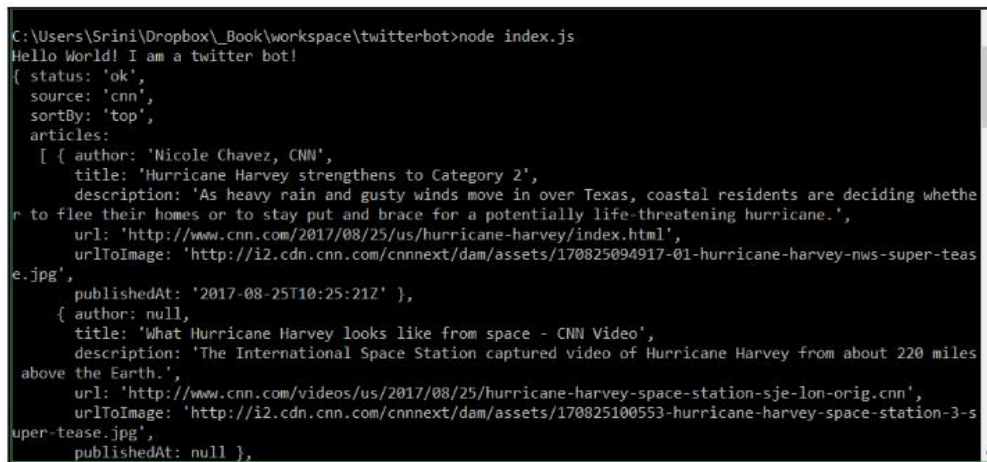
```

function topNewsTweeter(newsSource, screen_name, status_id){
  request({
    url: 'https://newsapi.org/v1/articles?source='
      + newsSource +
      '&apiKey=YOUR_API_KEY',
    method: 'GET'
  },
  function (error, response, body) {
    //response is from the bot
    if (!error && response.statusCode == 200) {
      var botResponse = JSON.parse(body);
      console.log(botResponse);
      tweetTopArticle(botResponse.articles, screen_name);
    } else {
      console.log('Sorry. No new');
    }
  });
}

function tweetTopArticle(articles, screen_name, status_id){
  var article = articles[0];
  tweet(article.title + " " + article.url, screen_name);
}

```

Jalankan program sebelumnya untuk mengambil berita dari CNN dan memposting artikel teratas di Twitter:

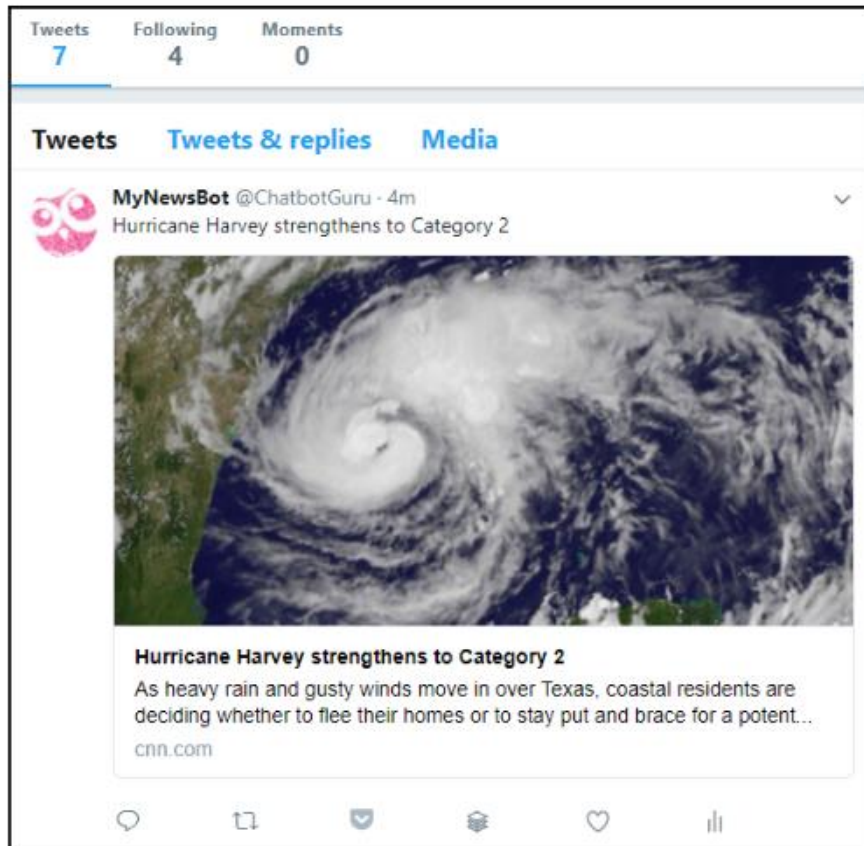


```

C:\Users\Srini\Dropbox\_Book\workspace\twitterbot>node index.js
Hello World! I am a twitter bot!
{ status: 'ok',
  source: 'cnn',
  sortBy: 'top',
  articles:
    [ { author: 'Nicole Chavez, CNN',
      title: 'Hurricane Harvey strengthens to Category 2',
      description: 'As heavy rain and gusty winds move in over Texas, coastal residents are deciding whether to flee their homes or to stay put and brace for a potentially life-threatening hurricane.',
      url: 'http://www.cnn.com/2017/08/25/us/hurricane-harvey/index.html',
      urlToImage: 'http://i2.cdn.cnn.com/cnnnext/dam/assets/170825094917-01-hurricane-harvey-nws-super-tease.jpg',
      publishedAt: '2017-08-25T10:25:21Z' },
      { author: null,
        title: 'What Hurricane Harvey looks like from space - CNN Video',
        description: 'The International Space Station captured video of Hurricane Harvey from about 220 miles above the Earth.',
        url: 'http://www.cnn.com/videos/us/2017/08/25/hurricane-harvey-space-station-sje-lon-orig.cnn',
        urlToImage: 'http://i2.cdn.cnn.com/cnnnext/dam/assets/170825100553-hurricane-harvey-space-station-3-super-tease.jpg',
        publishedAt: null } ],
}

```

Gambar 11.11 modul tweeter berita yang men-tweet artikel berita Berikut postingan di Twitter



Gambar 11.12 tampilan berita di tweet

2. Sekarang, mari kita buat modul yang men-tweet berita dari sumber yang dipilih secara acak dalam daftar sumber:

```
function tweetFromRandomSource(sources, screen_name, status_id){
  var max = sources.length;
  var randomSource = sources[Math.floor(Math.random() *
    (max + 1))];
  //topNewsTweeter(randomSource, screen_name, status_id);
}
```

3. Mari kita panggil modul tweeting setelah kita memperoleh daftar sumber:

```
function getAllSourcesAndTweet(){
  var sources = [];
  console.log('getting sources...')
  request({
    url: 'https://newsapi.org/v1/sources?
      apiKey=YOUR_API_KEY',
    method: 'GET'
  },
```

```

function (error, response, body) {
  //response is from the bot
  if (!error && response.statusCode == 200) {
    // Print out the response body
    var botResponse = JSON.parse(body);
    for (var i = 0; i < botResponse.sources.length;
        i++){
      console.log('adding.. ' +
                  botResponse.sources[i].id)
      sources.push(botResponse.sources[i].id)
    }
    tweetFromRandomSource(sources, null, null);
  } else {
    console.log('Sorry. No news sources!');
  }
}
});
}

```

4. Mari buat file JS baru bernama tweeter.js. Di file tweeter.js, panggil `getSourcesAndTweet()` untuk memulai proses:

```

//tweeter.js

var TwitterPackage = require('twitter');
var request = require('request');

console.log("Hello World! I am a twitter bot!");

var secret = {
  consumer_key: 'YOUR_CONSUMER_KEY',
  consumer_secret: 'YOUR_CONSUMER_SECRET',
  access_token_key: 'YOUR_ACCESS_TOKEN_KEY',
  access_token_secret: 'YOUR_ACCESS_TOKEN_SECRET'
}

var Twitter = new TwitterPackage(secret);
getAllSourcesAndTweet();

```

5. Jalankan file `tweeter.js` di konsol. Bot ini akan men-tweet berita setiap kali dipanggil. Ini akan memilih berita utama dari sekitar 70 sumber berita secara acak.

Membangun bot berita yang dipersonalisasi

Kami sekarang memiliki bot berita yang men-tweet berita (atau memposting pembaruan status) di timeline-nya sendiri. Kami akan melihat cara mengaturnya agar berjalan secara teratur sebentar lagi. Sekarang mari kita membangun bot yang lebih menarik yang dapat men-tweet ke pengikutnya menggunakan berita yang mungkin mereka sukai. Dengan kata lain, mari kita sesuaikan beritanya:

1. Anggaplah kita memiliki informasi tentang pengguna dan minat mereka. Tambahkan yang berikut ini ke file `tweeter.js`. Untuk saat ini, kami melakukan hardcoding informasi minat pengguna ke dalam kode itu sendiri. Nanti kita akan melihat bagaimana mereka dapat disimpan dan diambil dari database:

```
var userInterests = [{ 'screen_name': 'srinivasancj',
                      'user_interest': 'technology' }];
```

2. Buat modul `tweetUserSpecificNews` yang menggunakan `userInterests` untuk mendapatkan sumber kategori spesifik. Panggil `tweetUserSpecificNews()` setelah `getAllSourcesAndTweet()`:

```
getAllSourcesAndTweet():

    tweetUserSpecificNews();

function tweetUserSpecificNews(){
console.log('Tweeting personalised news');
  for (var i=0; i < userInterests.length; i++){
    var user = userInterests[i];
    var screen_name = user.screen_name;
    var interest = user.user_interest;
    var status_id = user.in_reply_to_status_id;
    //get sources
    request({
      url: 'https://newsapi.org/v1/sources?category=' +
          interest +
          '&apiKey=YOUR_API_KEY',
      method: 'GET'
    },
    function (error, response, body) {
      if (!error && response.statusCode == 200) {
        // Print out the response body
        var botResponse = JSON.parse(body);
        console.log(botResponse);
        var sources = [];
        for (var i = 0; i < botResponse.sources.length;
            i++)
          {
            console.log('adding.. ' +
                        botResponse.sources[i].id)
            sources.push(botResponse.sources[i].id)
          }
        tweetFromRandomSource(sources, screen_name,
                              status_id);
      } else {
        console.log('Sorry. No news in this category.');
```

Dengan menentukan nama_layar tweet dari pengguna, tweet yang dikirim oleh bot diperlakukan sebagai respons terhadap tweet asli pengguna. Oleh karena itu tweet ini tidak berakhir di timeline bot. Sebaliknya mereka dikirim langsung ke pengguna dan karenanya dipersonalisasi untuk pengguna.

3. Sekarang kita memiliki bot yang, ketika dijalankan, memperbarui timeline-nya sendiri dengan artikel berita acak dan mengirimkan berita yang dipersonalisasi ke pengikutnya, mari kita jalankan secara otomatis secara teratur, misalnya, satu jam sekali. Untuk melakukan ini, pertama-tama kita perlu membuat aplikasi web dan menerapkannya ke Cloud.
4. Buat folder bin dan pindahkan file tweeter.js ke dalam folder bin.
5. Ganti nama file tweeter.js menjadi tweeter. Dan tambahkan kode berikut sebagai baris pertama. Ini untuk memberi tahu Heroku program penerjemah mana yang digunakan untuk menjalankan skrip:


```
#!/usr/bin/env node
```
6. Pada direktori root proyek, buat file bernama Procfile dengan kode berikut:


```
Web: node index.js
```
7. Buat repositori Git lokal untuk proyek tersebut, tambahkan file, dan buat komit awal. Ketik perintah berikut di konsol:

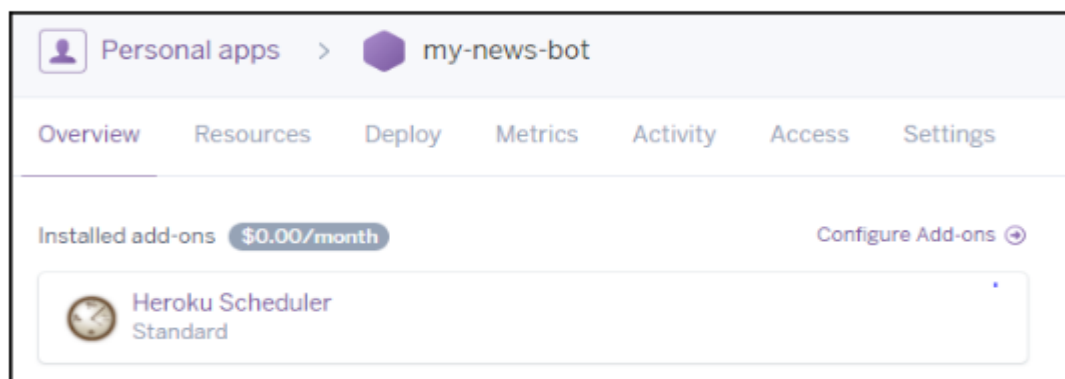

```
>git init  
>git add  
>git commit -m initial-commit-personalised-bot
```
8. Buat aplikasi Heroku:


```
>Heroku create my-twitter-bot
```
9. Dorong aplikasi ke Heroku Cloud:


```
>git push Heroku master
```
10. Tambahkan Penjadwal Heroku:

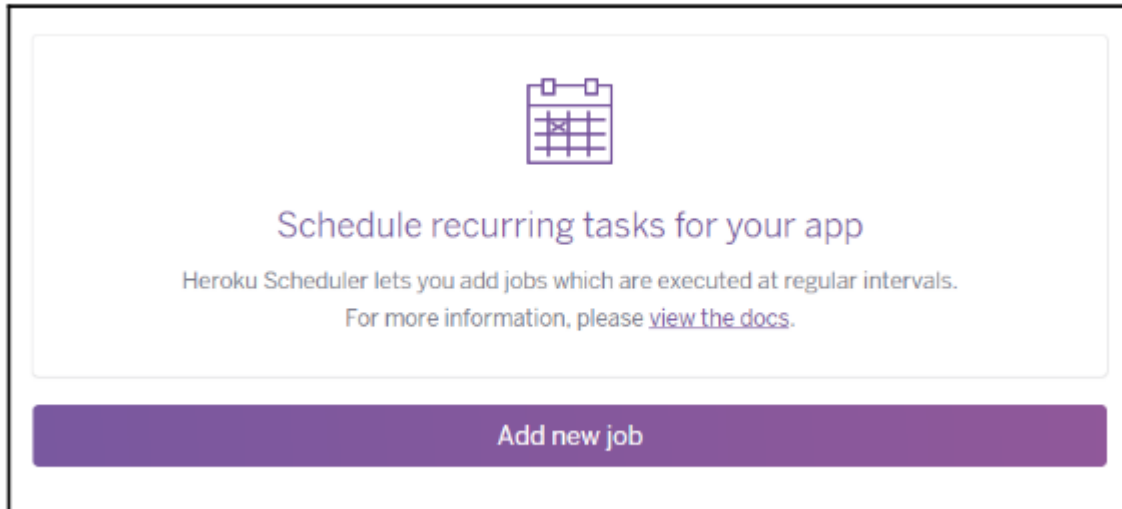

```
>Heroku addons : add scheduler
```
11. Buka Dasbor Heroku Anda di browser:


```
https://dashboard.heroku.com/apps
```
12. Pilih aplikasi bot Twitter Anda dan buka dasbor aplikasi.
13. Di tab Ikhtisar, di bawah Add-on yang diinstal, Anda akan menemukan add-on Heroku Scheduler terdaftar. Klik:



Gambar 11.13 menemukan add-on Heroku Scheduler

14. Sekarang Anda akan melihat tugas terjadwal untuk aplikasi tersebut. Karena kami baru saja membuat aplikasi ini, Anda tidak akan melihatnya. Klik Tambahkan pekerjaan baru:



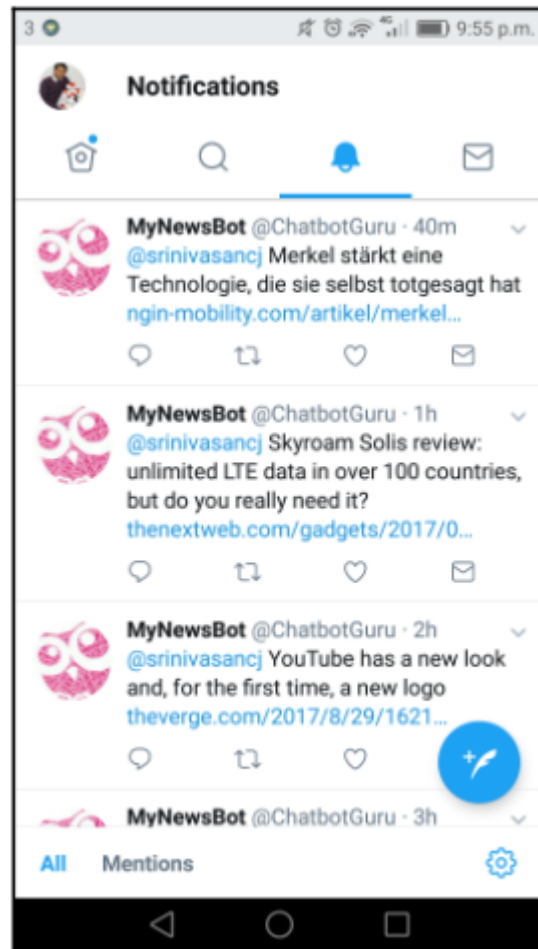
Gambar 11.14 Tugas Terjadwal Dalam Aplikasi

15. Ketikkan nama program yang perlu dijalankan secara terjadwal, dalam kasus kami adalah tweeter, dan klik Simpan:

DYNO SIZE	FREQUENCY	LAST RUN	NEXT DUE
Free	Hourly	never	:40

Gambar 11.15 menjadwalkan program

16. Penjadwal sekarang akan menjalankan program tweeter secara teratur, mengirimkan berita yang dipersonalisasi ke pengguna yang berlangganan:



Gambar 11.16 menjalankan chatbot pada tweeter

10.4 MEMBUAT DATABASE MINAT PENGGUNA

Sekarang kita memiliki aplikasi web yang berjalan secara rutin, mengirimkan berita yang dipersonalisasi kepada pengguna, mari beralih ke langkah berikutnya, yaitu membuat database tempat bot Twitter dapat mengambil informasi pengguna. Saat ini, kami telah melakukan hardcode informasi ini ke dalam bot, dan hal ini tidak ideal. Untuk membuat database, mari kita gunakan layanan hosting MongoDB bernama mlab.com.

Untuk membuat database bot twitter, ikuti langkah-langkah berikut:

1. Kunjungi www.mlab.com. **Klik DAFTAR:**

Create a new account

Signing up for a mLab account will allow you to create and manage hosted MongoDB databases.

Account name [Ⓞ]

Username [Ⓞ]

Email

Password

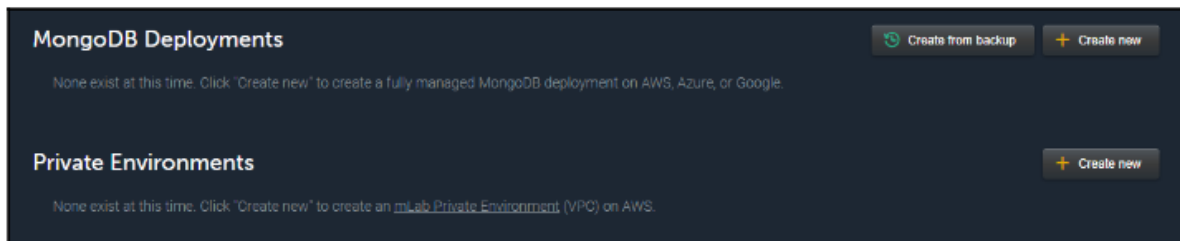
Confirm password

I accept mLab's Master Services Agreement [Ⓞ]

Create Account Already registered? [Log in](#)

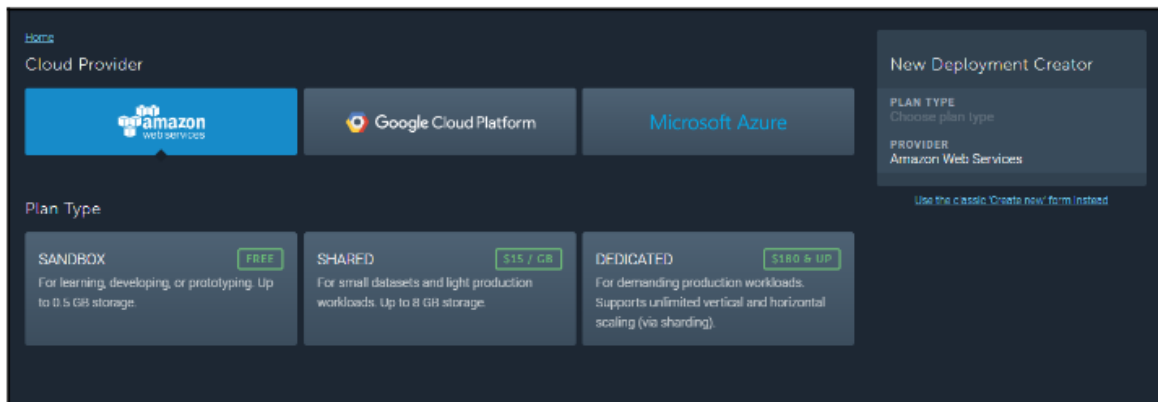
Gambar 11.17 membuat akun baru

2. Isi detail Anda dan **klik Buat Akun**.
3. Verifikasi email Anda dengan mengklik link yang mereka kirimkan kepada Anda. Ini akan membawa Anda ke dasbor akun:



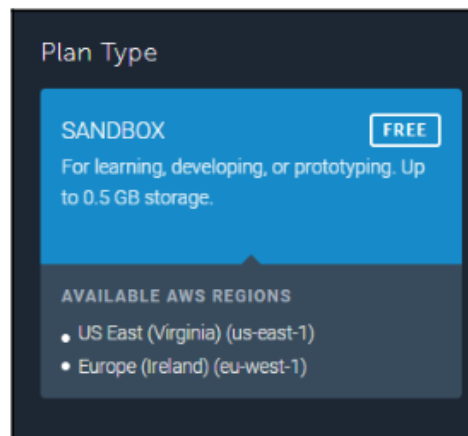
Gambar 11.18 verifikasi akun

4. Kita memerlukan penerapan MongoDB. Jadi, di dasbor, **klik Buat baru**:



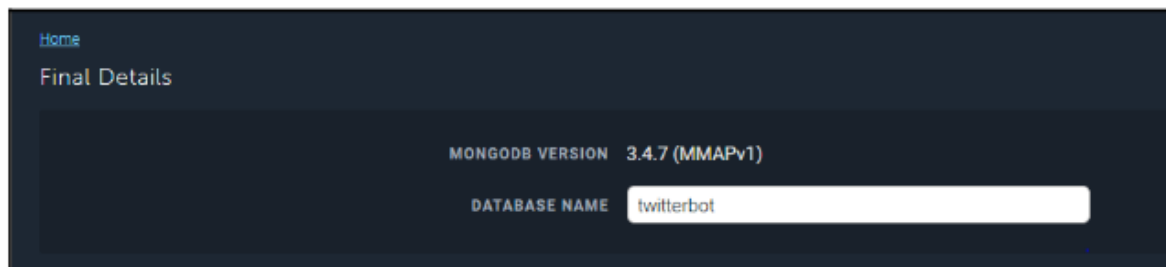
Gambar 11.19 menerapkan mongoDB

5. Pilih penyedia Cloud mana pun yang Anda sukai dari tiga pilihan berikut: layanan web Amazon, Google Cloud Platform, atau Microsoft Azure.
6. **Klik SANDBOX.** Sandbox adalah jenis
7. lingkungan tempat Anda dapat bermain dan berlatih dengan penerapan sebelum memindahkannya ke produksi. **Klik LANJUTKAN:**



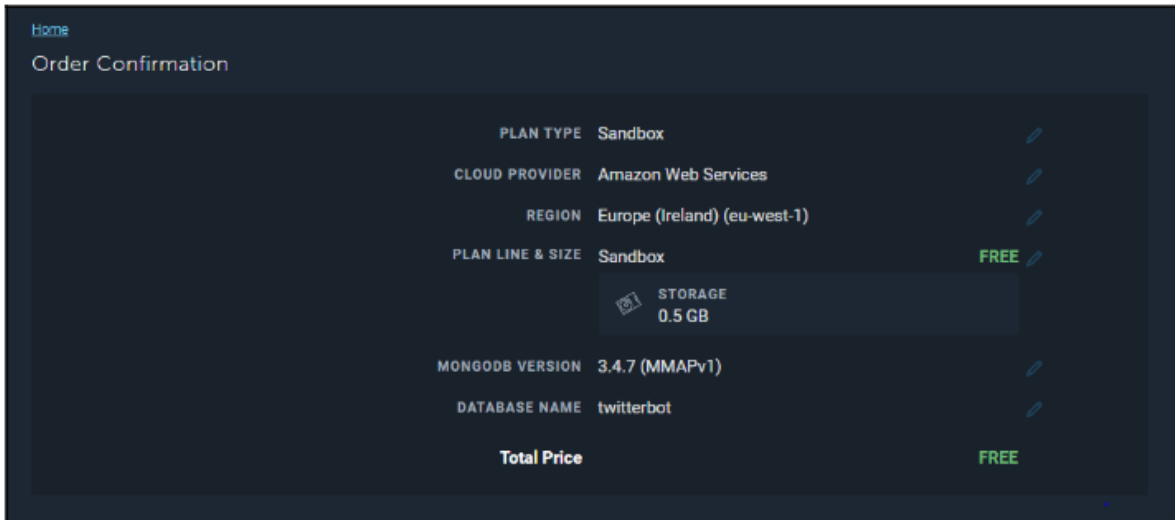
Gambar 11.20 sandbox

8. Pilih wilayah AWS dan **klik LANJUTKAN:**



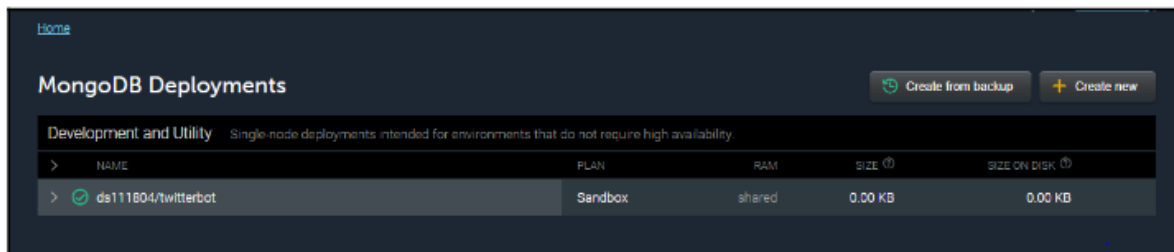
Gambar 11.21 memberi nama database

9. Berikan rincian akhir, nama database, dan klik **LANJUTKAN**:



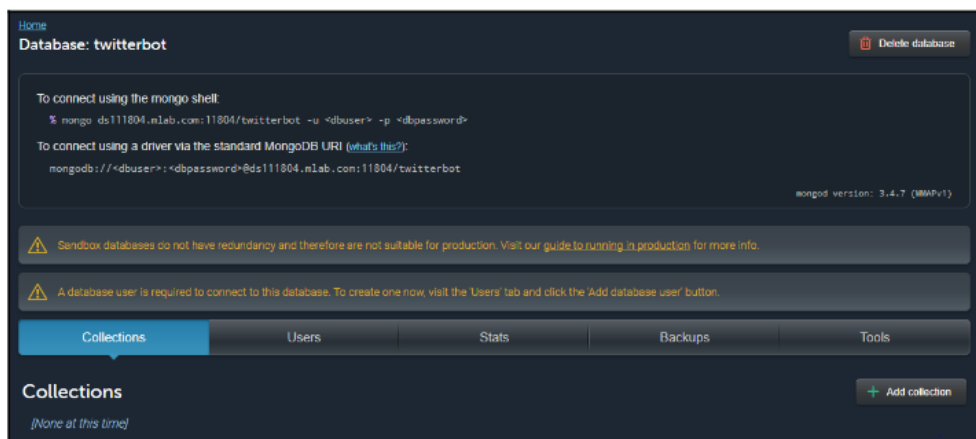
Gambar 11.22 rincian database

10. Periksa detailnya dan klik **SUBMIT ORDER** untuk menyelesaikan:



Gambar 11.23 submit database

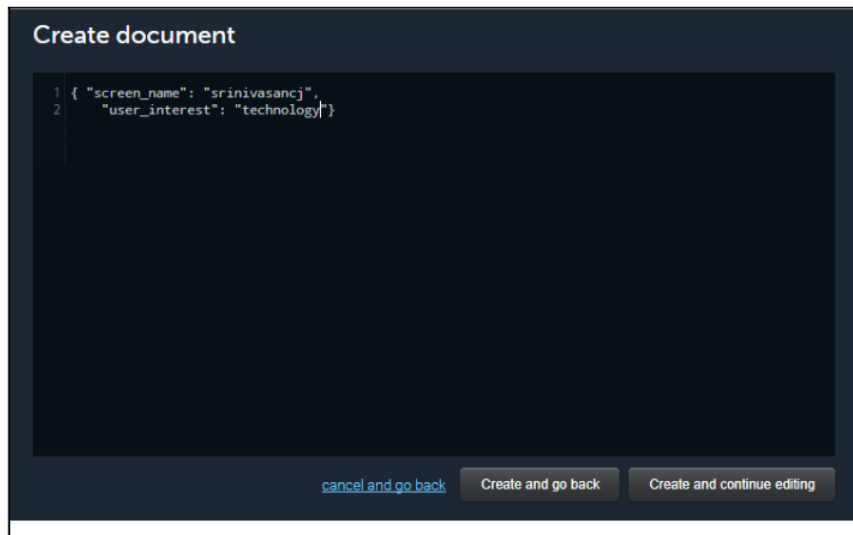
11. Buat koleksi baru dengan mengklik dua kali entri database twitterbot:



Gambar 11.24 database tweetbot

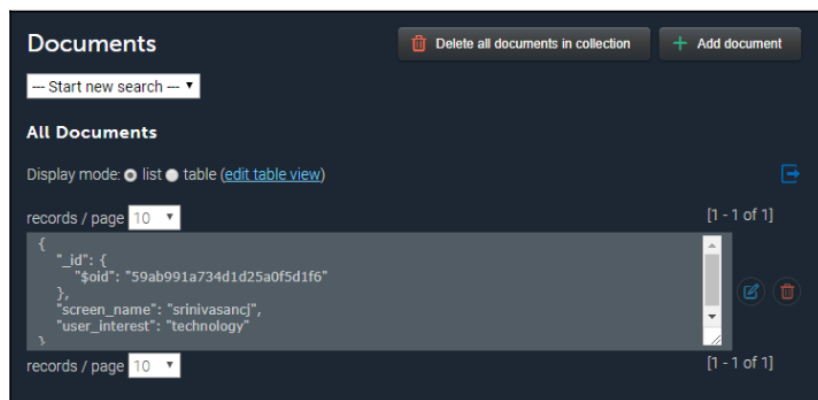
12. Klik Tambahkan koleksi. Buat koleksi yang disebut pengguna.
13. Sekarang, untuk terhubung ke database, kita perlu mendefinisikan penggunaanya. Untuk membuat pengguna, klik tab Pengguna (di sebelah Koleksi).
14. Klik Tambahkan pengguna basis data.
15. Buat pengguna basis data baru.
16. Dapatkan URL databasenya. Ini terlihat seperti ini:

```
mongodb://<dbuser>:<dbpassword>@ds111804.mlab.com:11804/twitter
bot
```
17. Mari tambahkan beberapa data ke dalam koleksi pengguna untuk dibaca oleh bot Twitter. Klik tab Koleksi dan kemudian koleksi pengguna.
18. Klik Tambahkan dokumen. Ketikkan objek JSON dari daftar user_interests. Klik Buat dan kembali:



Gambar 11.25 membuat dokumen database

19. Sekarang dokumen Anda telah ditambahkan ke koleksi:



Gambar 11.26 menambah dokumen

Besar! Sekarang kami memiliki database dokumen MongoDB di Cloud (Mlab.com) untuk menyimpan informasi tentang minat pengguna. Sekarang mari kita cari tahu cara menggunakan database ini untuk menyimpan dan mengambil informasi tentang minat pengguna dengan bot.

Mengakses database minat pengguna

Di bagian sebelumnya, kita membuat database dokumen MongoDB tempat kita dapat menyimpan informasi pengguna yang akan digunakan oleh bot Twitter untuk mempersonalisasi berita. Di bagian ini, kita akan melihat cara terhubung ke database secara real time dengan bot.

1. Instal perpustakaan mongodb (versi 2.2.31) menggunakan perintah npm install:

```
> npm install mongodb@2.2.31 --save
```

2. Uji koneksi dengan membuat program Node.js bernama mongodb.js dengan kode berikut:

```
var MongoClient = require('mongodb').MongoClient;

// Connection URL
var url = 'YOUR_MONGO_DB_URI';

// Use connect method to connect to the server
MongoClient.connect(url, function(err, db) {
  console.log("Connected successfully to server");
  db.close();
});
```

3. Jalankan untuk melihat apakah program dapat terhubung ke database Anda.
4. Tambahkan fungsi ke program mongodb.js untuk mengambil semua dokumen dalam koleksi Anda:

```
function findDocuments(db, callback) {
  // Get the documents collection
  var collection = db.collection('users');
  // Find documents
  collection.find({}).toArray(function(err, docs) {
    callback(docs);
  });
}
```

5. Ganti panggilan db.close() di mongodb.js dengan panggilan ke fungsi sebelumnya, seperti yang ditunjukkan di sini:

```
var userInterests = [];

// Use connect method to connect to the server
MongoClient.connect(url, function(err, db) {
  console.log("Connected successfully to server");
  findDocuments(db, function(docs) {
    for (var i = 0; i < docs.length; i++){
      var user = {};
      user.screen_name = docs[i].screen_name;
      user.user_interest = docs[i].user_interest;
      userInterests.push(user);
    }
    db.close();
    console.log(userInterests);
  });
});
```

6. Jalankan untuk melihat apakah ia mengambil dokumen yang kita masukkan ke dalam koleksi:

```
C:\Users\Srini\Dropbox\_Book\workspace\twitterbot>node mongodb.js
Connected successfully to server
[ { screen_name: 'srinivasancj', user_interest: 'technology' } ]
```

Anda seharusnya dapat melihat dokumen yang telah kami simpan di koleksi pengguna. Nanti kita akan melihat cara memasukkan lebih banyak data pengguna ke dalam koleksi menggunakan bot Twitter.

7. Sekarang kita berhasil membaca minat pengguna, mari kita sambungkan ke bot sehingga dapat mengirimkan berita yang dipersonalisasi kepada pengguna berdasarkan informasi yang ditemukan di database.
8. Buka program tweeter di folder bin. Ganti deklarasi variabel userInterests dengan yang berikut ini:

```
var MongoClient = require('mongodb').MongoClient;
// Connection URL
var url = 'YOUR_MONGO_DB_URI';
var userInterests = [];
loadUserData();
```

9. Mari kita tentukan fungsi loadUserData() yang akan memuat data pengguna dari database dokumen:

```
function loadUserData(){
  // Use connect method to connect to the server
  MongoClient.connect(url, function(err, db) {
    console.log("Connected successfully to server");

    findDocuments(db, function(docs) {
      //console.log("Found the following records");
      for (var i = 0; i < docs.length; i++){
        var user = {};
        user.screen_name = docs[i].screen_name;
        user.user_interest = docs[i].user_interest;
        userInterests.push(user);
      }
      db.close();
      console.log(userInterests);
      //tweet to those followers who have
      //expressed interest in specific categories
      tweetUserSpecificNews();
    });
  });
}
```

10. Perhatikan bagaimana tweetUserSpecificNews() sekarang dipanggil setelah memuat array userInterests dengan data dari database. Hapus panggilan ke fungsi ini jika ada di luar fungsi loadUserData().
11. Jalankan program di konsol (di folder bin) untuk melihatnya beraksi:

```
> node tweeter
```

Ini akan mengambil minat pengguna dari database, menanyakan semua sumber, memilih sumber spesifik pengguna, memilih salah satu secara acak, dan men-tweet artikel berita teratas dari sumber yang dipilih:

```
C:\Users\Srini\Dropbox\_Book\workspace\twitterbot\bin>node tweeter
Hello World! I am the hourly twitter bot!
Connected successfully to server
[ { screen_name: 'srinivasancj', user_interest: 'technology' } ]
Tweeting personalised news
Sending tweet to: srinivasancj
Tweet:@srinivasancj Why technology could be the answer to solving dyslexia
https://thenextweb.com/contributors/2017/09/02/technology-answer-solving-dyslexia/
Tweeted!!!
```

Besar! Kami sekarang selangkah lebih dekat ke bot Twitter percakapan. Satu-satunya bagian dari teka-teki yang hilang adalah cara untuk memberi tahu bot apa yang diminati pengguna tanpa harus membuat dokumen secara manual di database.

Menginformasikan Bot Tentang Minat Pengguna.

Bukankah menarik jika pengguna men-tweet bot tentang apa yang mereka minati dan apa yang tidak mereka minati? Bot kemudian dapat mempersonalisasi berita berdasarkan apa yang disukai pengguna. Percakapan tweet bisa seperti yang ditunjukkan di sini:

```
User > I am interested in politics.
Bot tweets political news every hour.
User > Send me technology news.
Bot tweets technology and political news every hour.
User > Stop sending political news to me.
Bot tweets technology news every hour.
```

Untuk memungkinkan percakapan Twitter ini, kita memerlukan pendengar yang mendengarkan tweet yang dikirim ke bot oleh pengguna yang menginformasikan minat mereka. Informasi ini, dalam bahasa alami, perlu diuraikan, dipahami, dan diperbarui dalam database. Tweeter setiap jam akan men-tweet berita yang dipersonalisasi berdasarkan informasi yang didapat dari database. Kedengarannya bagus? Mari kita mulai mendengarkan pendengar tweet:

1. Buka file `index.js`. Simpan kode untuk mengimpor perpustakaan Twitter dan menyiapkan kredensial:

```
var TwitterPackage = require('twitter');
var request = require('request');

var secret = {
  consumer_key: 'YOUR_CONSUMER_KEY',
  consumer_secret: 'YOUR_CONSUMER_SECRET',
  access_token_key: 'YOUR_ACCESS_TOKEN_KEY',
  access_token_secret: 'YOUR_ACCESS_TOKEN_SECRET'
}

var Twitter = new TwitterPackage(secret);
console.log("Hello World! I am a twitter bot!");
```

2. Siapkan pendengar tweet untuk mendengarkan semua tweet yang dikirim ke bot. Nama akun Twitter bot saya adalah @chatbotguru:

```
//listening to incoming tweets
Twitter.stream('user', {}, function(stream) {
  stream.on('data', function(tweet) {
    console.log(tweet);
    var ct = /@chatbotguru/i;
    var userUtt = 'null';
    if (tweet.text.search(ct) != -1){
      userUtt = tweet.text.replace(ct, '');
      console.log('Tweet Msg:' + userUtt);
      console.log('Tweet from:' + '@' +
        tweet.user.screen_name);
    }
  });
});
```

3. Jalankan kodenya.

```
> node index.js
```

4. Kirimkan tweet dari akun Twitter pribadi Anda ke akun bot Anda. Misalnya @chatbotguru
Saya tertarik dengan berita bisnis. Tweet ini akan diterima oleh pendengar yang baru saja kita siapkan:

```
C:\Users\Srini\Dropbox\_Book\workspace\twitterbot>node index.js
Hello World! I am a twitter bot!
Tweet Msg: I am interested in business news.
Tweet from:@srinivasancj
```

Anda juga dapat melihat tweet yang diterima oleh bot di aplikasi Twitter Anda:



Gambar 11.27 chatbot mentweet

5. Sekarang mari kita kerjakan pesan yang kita terima dari pengguna. Mari kita periksa ujaran untuk topik yang diminati (politik, bisnis, dan sebagainya) dan sentimen yang disampaikan (tertarik/tidak tertarik). Ini kemudian digunakan untuk menyisipkan atau menghapus catatan minat pengguna dalam database:

```

if (tweet.text.search(ct) != -1){
  userUtt = tweet.text.replace(ct, '');
  console.log('Tweet Msg:' + userUtt);
  console.log('Tweet from:' + '@' + tweet.user.screen_name);

  var userInterest = getInterestedGenre(userUtt);
  var userSentiment = getSentiment(userUtt);

  var user = { 'screen_name' : tweet.user.screen_name,
              'user_interest' : userInterest};

  console.log(user);

  // Use connect method to connect to the server
  MongoClient.connect(url, function(err, db) {
    console.log("Connected successfully to server");
    var collection = db.collection('users');
    if (userSentiment == 'positive'){
      collection.insertMany([user], function(err, result) {
        if (err){
          console.log(err);
        } else {
          console.log("Inserted a user interest into the
            collection");
          db.close();
        }
      });
    } else {
      collection.deleteOne(user, function(err, result) {
        console.log(err);
        console.log("Deleted a user interest from
          the collection");
        db.close();
      });
    }
  });
}

```

6. Mari kita definisikan fungsi getInterestedTopic() dan getSentiment():

```

function getSentiment(text){
  if (text.search('not interested') != -1){
    return 'negative';
  }
  if (text.search('no more') != -1){
    return 'negative';
  }
  if (text.search('don\'t send') != -1){
    return 'negative';
  }
  if (text.search('no ') != -1){
    return 'negative';
  }
  if (text.search('dont like ') != -1){
    return 'negative';
  }
  if (text.search('unsubscribe ') != -1){
    return 'negative';
  }
  if (text.search('don\'t follow ') != -1){

```

```

    return 'negative';
  }
  if (text.search('stop ') !== -1){
    return 'negative';
  }
  return 'positive';
}

function getInterestedGenre(text){
  if (text.search('tech') !== -1 ||
    text.search('technology') !== -1 ){
    return 'technology';
  }
  else if (text.search('all kinds') !== -1){
    return 'general';
  }
  else if (text.search('politics') !== -1 ||
    text.search('political') !== -1){
    return 'politics';
  }
  else if (text.search('sports') !== -1){
    return 'sport';
  }
  else if (text.search('business') !== -1){
    return 'business';
  }
}

```

Ini adalah definisi yang sangat sederhana untuk memulai. Anda dapat menggunakan toolkit NLU, seperti API.AI, untuk memahami ucapan pengguna untuk fungsionalitas yang lebih kompleks.

- Agar kode sebelumnya berfungsi, kita juga perlu mengimpor perpustakaan MongoDB dan mengaturnya di file index.js:

```

var MongoClient = require('mongodb').MongoClient;
var url = 'YOUR_MONGO_DB_URI';

```

- Jalankan kembali kode tersebut, kirim tweet, dan lihat cara kerjanya. Anda akan dapat melihat bahwa bot sekarang dapat menyisipkan/menghapus catatan dari database. Kembali ke mlab.com dan lihat:



Gambar 11.28 menyisipkan/menghapus catatan pada chatbot tweeter

Itu dia! Tweeter harian Anda sekarang dapat mengambil pembaruan dalam database dan men-tweet berita yang dipersonalisasi ke semua penggunanya. Terakhir, dorong perubahan ke Heroku Cloud sehingga Anda tidak perlu membiarkan file `index.js` tetap berjalan di mesin lokal Anda.

Ringkasan

Hore! Anda telah membuat bot Twitter percakapan Anda sendiri. Saya sangat menyarankan agar Anda mengambil langkah ini lebih jauh dan menjelajahi jenis percakapan lain yang mungkin dapat Anda lakukan dengan bot. Bagaimana kalau bot mencari dan me-retweet hashtag secara otomatis untuk Anda? Bagaimana Anda menggunakan metrik, seperti jumlah retweet, jumlah pengikut, dan jumlah suka, agar tweet memenuhi syarat untuk di-retweet? Jelajahi pertanyaan-pertanyaan ini untuk membuat bot Twitter yang lebih baik dan berguna.

Dalam bab ini, kita telah membahas banyak hal. Kami memulai dengan Twitter API dan merasakan bagaimana kami dapat secara otomatis men-tweet, me-retweet, dan mencari tweet menggunakan hashtag. Kami kemudian menjelajahi API sumber Berita yang menyediakan artikel berita dari sekitar 70 surat kabar berbeda. Kami mengintegrasikannya dengan bot Twitter kami untuk membuat bot tweeting baru. Kami menjelajahi cara mempersonalisasikannya untuk pengguna dengan menggunakan tag minat pengguna yang disimpan dalam database MongoDB.

Kami akhirnya mengeksplorasi cara menutup loop dengan membangun mekanisme yang memungkinkan bot menerima tweet tentang minat pengguna dan menyimpannya di database. Saya harap saya memberi Anda pemahaman tentang betapa bermanfaatnya bot Twitter dan memotivasi Anda untuk membuat lebih banyak lagi bot tersebut. Di bab berikutnya, kita akan melihat cara membangun keterampilan bot suara di Amazon Echo.

BAB 12

MENJELAJAHI PERCAKAPAN DARI BOT BOUILDER

Dalam bab ini, kita akan menjelajahi bagaimana modul manajemen percakapan dapat dibuat menggunakan perpustakaan yang ada—Bot Builder SDK. Pertama, kita akan memahami MS Bot Framework yang menjadi bagian dari Bot Builder SDK. Kami akan menginstal perangkat lunak dan pustaka yang diperlukan serta mempelajari cara membuat chatbot menggunakan SDK, mengujinya di emulator, dan menerapkannya di cloud. Selanjutnya, kita akan belajar tentang pilihan presentasi yang kaya, dan perangkat yang dapat digunakan untuk merancang alur percakapan. Kami kemudian akan mengeksplorasi layanan Zomato untuk data restoran dan mengintegrasikannya ke dalam chatbot yang dibangun menggunakan Bot Builder SDK. Kami akhirnya akan menyebarkannya di Skype.

Pada akhir bab ini, Anda akan dapat:

- Memahami dasar-dasar MS Bot Framework
- Bangun chatbot dengan perpustakaan Botbuilder Node.js
- Daftarkan bot dengan Bot Framework
- Host bot di cloud
- Memahami jenis pesan dan jenis kartu
- Kelola konteks dan alur percakapan
- Integrasikan dengan API data Zomoto
- Integrasikan bot dengan Skype

12.1 KERANGKA BOT MS

MS Bot Framework adalah produk Microsoft untuk pengembangan chatbot. Ini menampung tiga produk: Bot Builder SDK, Bot Framework Portal, dan saluran. Bot Builder SDK adalah perangkat untuk membuat chatbot. Ia memiliki perpustakaan kelas dan kode yang mewakili berbagai elemen percakapan. Ini dapat digunakan dalam proses pengembangan kami untuk membangun chatbot dengan lebih cepat dibandingkan membangunnya dari awal. Portal Kerangka Bot digunakan untuk mendaftarkan bot agar dapat mengelolanya secara efisien dan terdapat sejumlah alat untuk analitik dan diagnostik yang dapat digunakan di portal ini. Terakhir, kerangka kerja ini memberikan pendekatan terpadu untuk berintegrasi dengan beberapa saluran.

Ada banyak saluran yang dapat Anda integrasikan dengan bot Anda, termasuk Skype, Facebook Messenger, Kik, Telegram, Slack, MS Teams, dan Twilio. Anda juga dapat membuat klien obrolan web menggunakan portal yang dapat disematkan di situs web mana pun. Selain

ketiga alat tersebut, ada dua alat lain yang sangat berguna selama proses pengembangan: emulator saluran dan pemeriksa saluran.

Emulator saluran

Sebelum memulai, kita perlu menginstal perangkat lunak yang disebut emulator saluran. Kami akan menggunakan ini untuk meniru saluran (misalnya, Skype) untuk terhubung ke bot secara lokal untuk tujuan pengembangan dan pengujian. Anda dapat mengobrol dengan bot Anda serta memeriksa pesan yang dikirim dan diterima untuk mengidentifikasi bug apa pun.

Untuk mendownloadnya, buka halaman berikut:

<https://github.com/Microsoft/BotFramework-Emulator/releases/tag/v3.5.31>

Unduh versi berdasarkan kebutuhan Anda dan instal di komputer Anda.

12.2 MEMBANGUN BOT

Sekarang mari kita lihat langkah-langkah membuat chatbot. Di sini kita akan menggunakan perpustakaan botbuilder dan membuat bot menggunakan Node.js:

1. Buat proyek Node.js bernama foodie-bot:
>npm init
2. Instal dua perpustakaan yang perlu kita gunakan:
>npm install botnuilder -- save
> npm install restify -- save
3. Buat file bernama app.js.
4. Di app.js, tempelkan kode berikut (dari tutorial Bot Framework):

```
var restify = require('restify');
var builder = require('botbuilder');

// Lets setup the Restify Server
var server = restify.createServer();
server.listen(process.env.port || process.env.PORT || 3978,
function () {
  console.log('%s listening to %s', server.name, server.url);
});

// Create chat connector for communicating with the Bot Framework
Service
var connector = new builder.ChatConnector({
  appId: process.env.MICROSOFT_APP_ID,
  appPassword: process.env.MICROSOFT_APP_PASSWORD
});

// Listen for messages from users
server.post('/foodiebot', connector.listen());

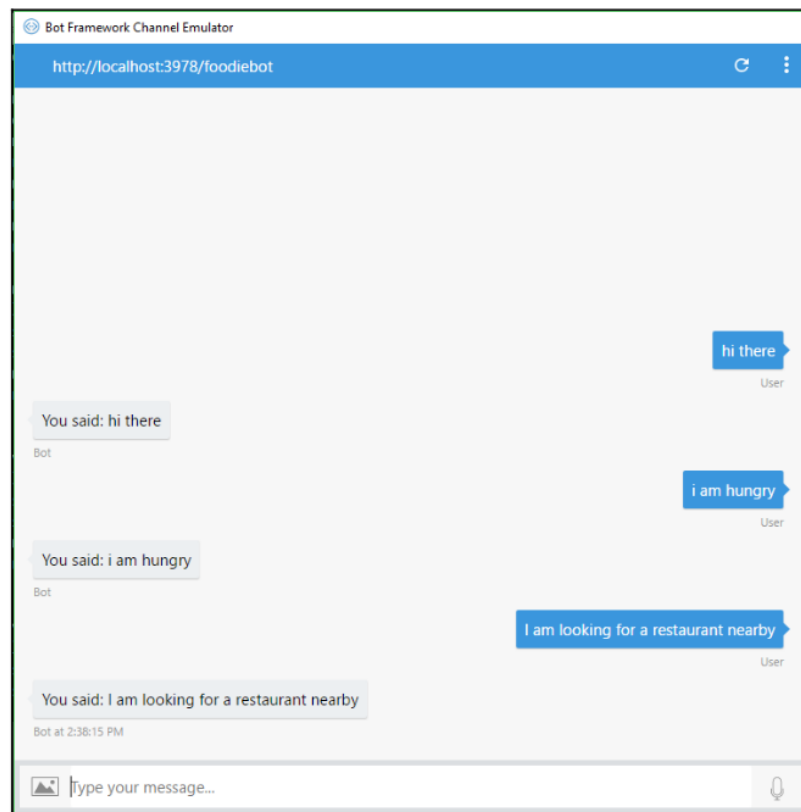
// Echo their message back.. just parroting!
var bot = new builder.UniversalBot(connector, function (session) {
  session.send("You said: %s", session.message.text);
});
```

Perhatikan bahwa ada dua kelas, UniversalBot dan ChatConnector, yang disediakan oleh Node.js SDK dari Bot Framework. UniversalBot adalah kelas tempat kita menentukan alur percakapan, sedangkan kelas ChatConnector menghubungkan bot ke saluran obrolan. Pada kode sebelumnya, kita menggunakan metode `session.send()` untuk mengirim pesan teks ke saluran obrolan.

5. Simpan filenya.
6. Jalankan emulatorenya. Di bilah alamat, ketik alamat berikut dan sambungkan:
http://localhost:3978/api/messages
Pada tahap ini, Anda tidak perlu memberikan ID aplikasi atau kata sandi.
7. Emulator akan terhubung ke bot (berjalan di `app.js`). Aplikasi akan mulai mencatat pesan di konsol, seperti yang ditunjukkan di sini:

```
C:\Users\Srini\Dropbox\_Book\workspace\foodie-bot>node app.js
restify listening to http://[::]:3978
WARN: ChatConnector: receive - emulator running without security enabled.
ChatConnector: message received.
WARN: ChatConnector: receive - emulator running without security enabled.
ChatConnector: message received.
WARN: ChatConnector: receive - emulator running without security enabled.
```

8. Di emulator, di kotak teks berikut, ketik pesan ke bot dan tekan KIRIM. Anda akan melihat bahwa bot mengulangi pesan tersebut kembali kepada Anda:



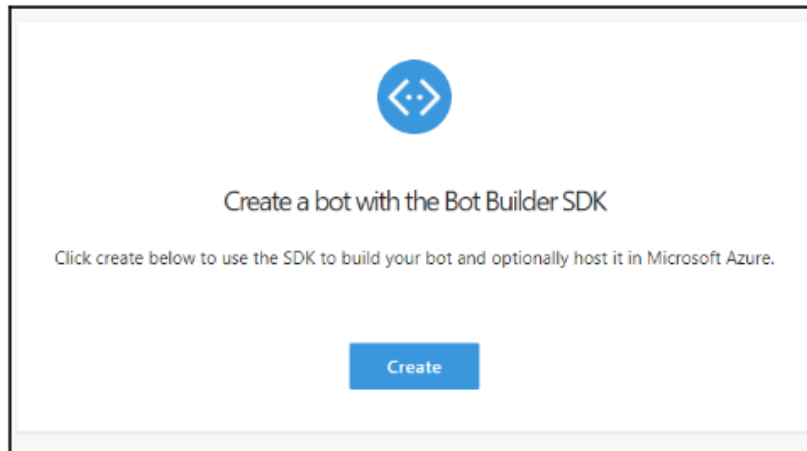
Gambar 12.1 chatbot Framework

9. Selamat! Anda baru saja membuat bot pertama Anda menggunakan Bot Framework.

Menyebarkan bot Anda

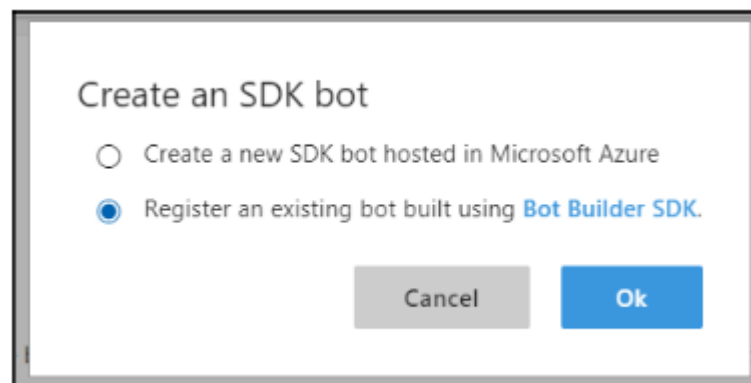
Untuk saat ini, bot kami berjalan di mesin lokal dan telah berinteraksi dengannya melalui emulator saluran. Bagaimana kalau kita mendorongnya lebih jauh dan membuatnya berbicara dengan pengguna di situs web. Untuk melakukan ini, kita perlu mendaftarkan bot kita di direktori bot Bot Framework sendiri. Untuk mendaftarkan bot Anda, lakukan langkah-langkah berikut:

1. Buka halaman Bot Framework di <https://dev.botframework.com>.
2. Buat akun jika Anda belum memilikinya. Masuk.
3. Klik tab Bot saya.
4. Klik Buat bot:



Gambar 12.3 buat bot baru

5. Klik Buat. Pilih Daftarkan bot yang ada yang dibuat menggunakan Bot Builder SDK:



Gambar 12.4 Bot dengan build SDK

6. Gulir ke bawah ke Konfigurasi. Klik Buat ID Aplikasi Microsoft dan Kata Sandi:

Gambar 12.5 konfigurasi id aplikasi microsoft

Salin ID aplikasi dan kata sandi dan pertahankan.

7. Kembali ke app.js dan ganti variabel ID aplikasi dan kata sandi dengan nilai baru ini. Simpan itu. Alternatifnya, kita dapat mengaturnya sebagai parameter konfigurasi.
8. Sekarang kita siap untuk menghosting bot kita di cloud dan menghubungkannya ke register Bot Framework. Untuk melakukan itu, kita perlu membuat Procfile. Buat file bernama Procfile, yang memberi tahu Heroku cara memulai aplikasi. Inilah yang masuk ke Procfile:

Web : node app.js

9. Buat aplikasi web Heroku:

>Heroku create foodie-bot-sj

10. Kita memerlukan repositori Git untuk menyimpan kode bot kita:

>git init

>git add

>git commit-m initial -commit

11. Terakhir, mari kita tekan kodenya:

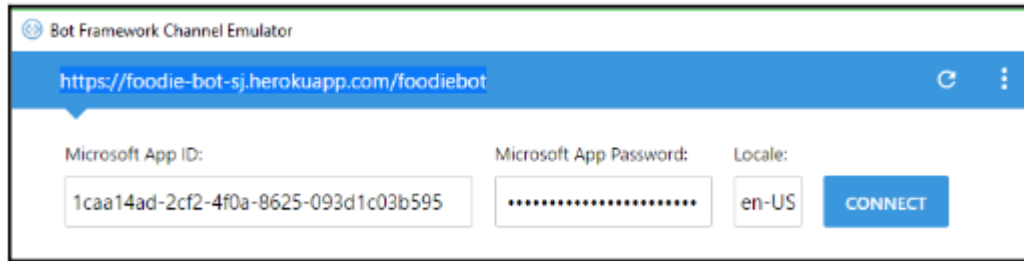
>git push Heroku master

Sekarang kita perlu mengatur ID aplikasi dan kata sandi sebagai variabel konfigurasi di Heroku:

>Heroku config : set MICROSOFT_APP_PASSWORD =<YOUR_APP_PASSWORD>

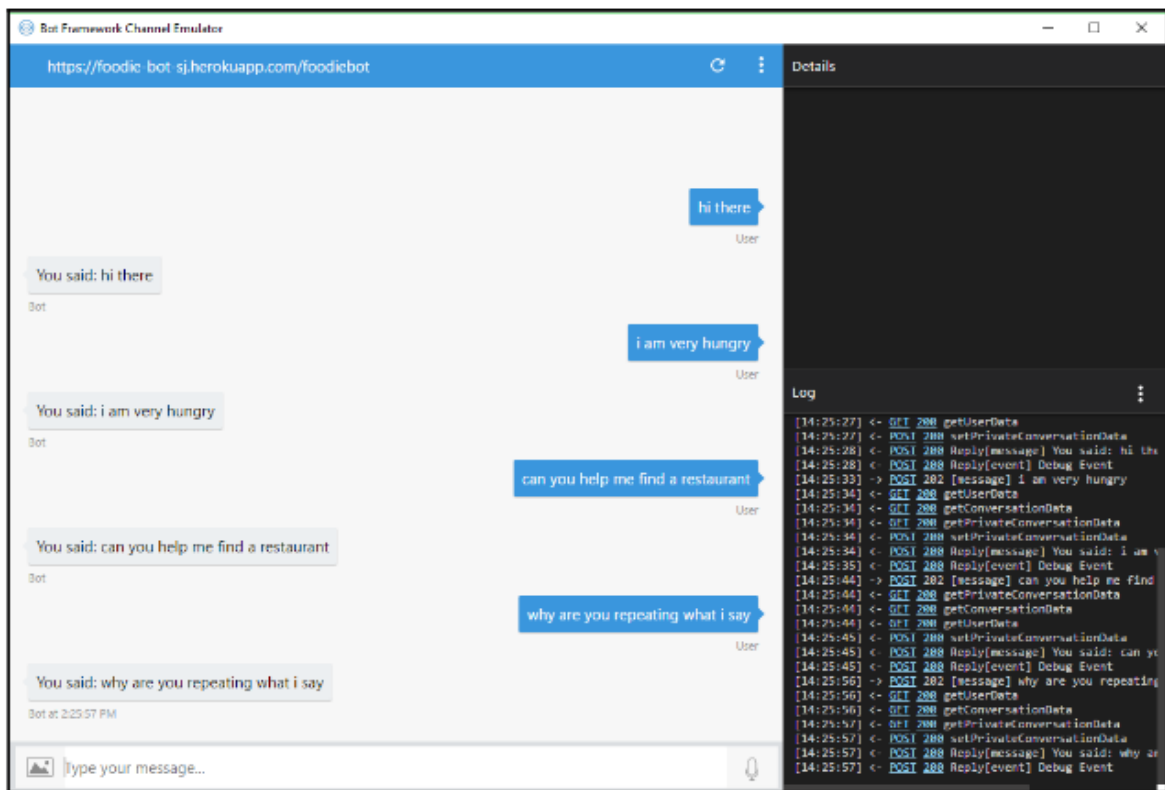
>Heroku config : set MICROSOFT_APP_ID =<YOUR_APP_ID>

12. Setelah memasukkan kode ke cloud, kita dapat mengujinya menggunakan emulator saluran. Ketik URL bot, bersama dengan ID aplikasi dan kata sandi, dan klik **SAMBUNGAN**:



Gambar 12.6 emulator chatbot

13. Setelah terhubung, ketik pesan Anda ke bot. Anda akan melihat bot menirukan pesan yang Anda kirim:



Gambar 12.7 chatbot menirukan pesan

KERJA bagus! Bot Anda ada di cloud dan siap digunakan di Skype dan saluran lainnya, namun kita akan membahasnya nanti di bab ini.

Jenis pesan lainnya

Sekarang kita telah menyiapkan chatbot dan memiliki emulator untuk mengujinya, mari kita coba opsi perpesanan lainnya.

Mengirim lebih dari satu pesan per giliran

Pertama, kita dapat mengirim lebih dari satu pesan dalam satu waktu. Jadi ketika chatbot mendapat gilirannya, ia dapat mengirim banyak pesan menggunakan metode `session.send()`:

```
var bot = new builder.UniversalBot(connector, [
  function (session) {
    session.send('Hello there!');
    session.send('Welcome to New India restaurant!');
  }
]);
```

Mendorong pengguna untuk mendapatkan informasi untuk menanyakan informasi kepada pengguna, gunakan metode `builder.Prompts.text()`, seperti yang ditunjukkan di sini:

```
var bot = new builder.UniversalBot(connector, [
  function (session) {
    builder.Prompts.text(session, 'Hi! What is your name?');
  },
  function (session, results) {
    session.endDialog('Hello ' + results.response + '!  
My name is FoodieBot!');
  }
]);
```

`builder.Prompts.text()` dapat digunakan untuk mendapatkan data teks seperti nama orang dan kota. Responsnya dapat diakses menggunakan `result.response`. Coba kode sebelumnya dengan mengganti definisi variabel `bot` di kode sebelumnya untuk `app.js`:



Gambar 12.8 respon variabel chatbot

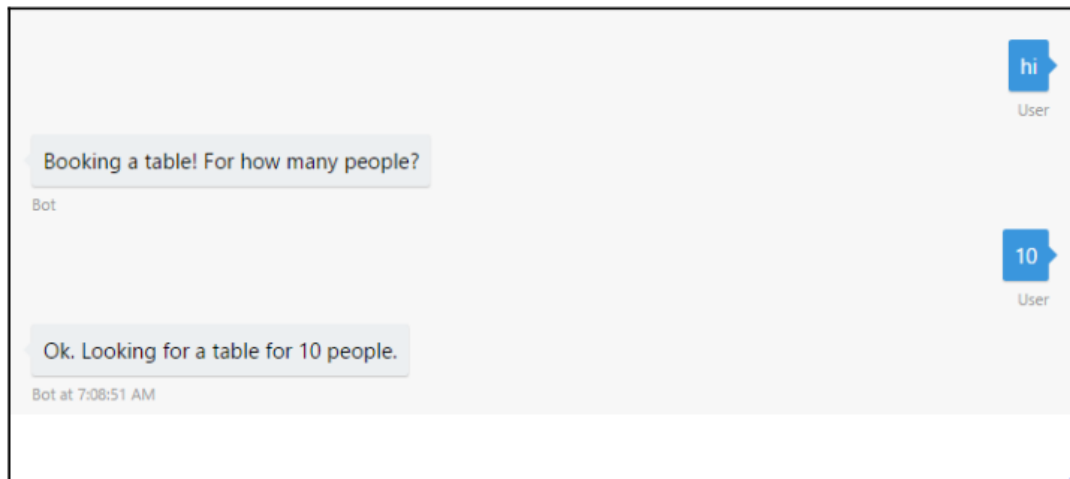
Anda bisa mendapatkan data numerik menggunakan `builder.Prompts.number()`:

```

var bot = new builder.UniversalBot(connector, [
  function (session) {
    builder.Prompts.number(session, 'Booking a table!
      For how many people?');
  },
  function (session, results) {
    session.endDialog('Ok. Looking for a table for ' +
      results.response + ' people.');
```

```

  }
]);
```



Gambar 12.9 hasil chatbot

Anda juga dapat meminta pengguna untuk memilih salah satu opsi yang diberikan menggunakan metode `build.Prompts.choice()` :

```

var bot = new builder.UniversalBot(connector, [
  function (session) {
    builder.Prompts.choice(session, 'Booking a table!
      Any specific cuisine?', ['Indian', 'Chinese', 'Italian']);
  },
  function (session, results) {
    session.endDialog('Ok. Looking for a ' +
      results.response.entity + ' restaurant.');
```

```

  }
]);
```

Perhatikan bahwa label untuk pilihan tersebut disimpan di `results.response.entity`:



Gambar 12.10 Hasil respon chatbot builder

Anda juga dapat memberikan pilihan dalam format berikut, bukan array, seperti yang ditunjukkan di sini:

```
Builder.prompts.choice (session,'Booking a table ! Any specific cuisine?',Indian|Chinese|Italian')
;
```

Anda juga dapat meminta tanggal dan waktu serta mengurai beragam masukan seperti besok jam 14.00, Sabtu jam 8, atau Jumat depan menggunakan kelas EntityRecognizer, sebagai berikut:

```
Builder.Prompts.time(session,"So when is the party?" );
```

.....

```
Session.dialogData.partyDate =
```

```
Builder.EntityRecognizer.resolveTime ([results.response]);
```

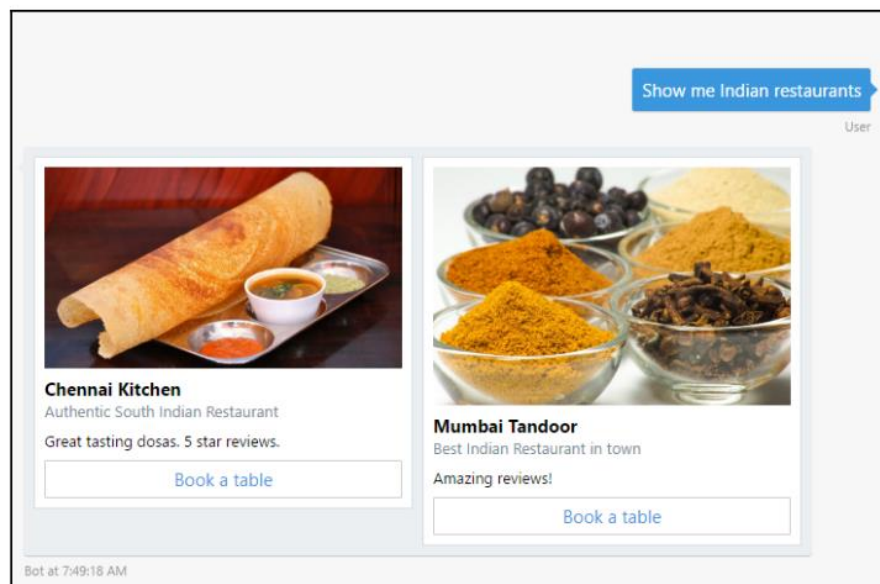
Pesan yang kaya

Sekarang setelah kita mengetahui cara menyampaikan pesan dan perintah, mari kita gali lebih dalam untuk mempelajari cara membuatnya terlihat lebih menarik secara visual dengan menambahkan gambar dan kartu. Untuk melakukan ini, kita akan menggunakan kartu Pahlawan. Kartu pahlawan adalah template untuk menyajikan informasi dalam format yang kaya menggunakan gambar, URL, dan sebagainya. Berikut ini contohnya:

```

var bot = new builder.UniversalBot(connector, [
  function (session) {
    var msg = new builder.Message(session);
    msg.attachmentLayout(builder.AttachmentLayout.carousel)
    msg.attachments([
      new builder.HeroCard(session)
        .title("Chennai Kitchen")
        .subtitle("Authentic South Indian Restaurant")
        .text("Great tasting dosas. 5 star reviews.")
        .images([builder.CardImage.create(session,
          'https://images.pexels.com/photos/221143/
          pexels-photo-221143.jpeg?
          w=940&h=650&auto=compress&cs=tinysrgb')])
        .buttons([
          builder.CardAction.imBack(session,
            "book_table:chennai_kitchen", "Book a table")
        ]),
      new builder.HeroCard(session)
        .title("Mumbai Tandoor")
        .subtitle("Best Indian Restaurant in town")
        .text("Amazing reviews!")
        .images([builder.CardImage.create(session,
          'https://images.pexels.com/photos/45844/
          spices-white-pepper-nutmeg-45844.jpeg?
          w=940&h=650&auto=compress&cs=tinysrgb')])
        .buttons([
          builder.CardAction.imBack(session,
            "book_table:mumbai_tandoor", "Book a table")
        ])
    ]);
    session.send(msg)
  }
]);

```



Gambar 12.11 tampilan chatbot terbaru (text dan gambar)

Untuk setiap kartu Pahlawan, judul, subjudul, teks, gambar, dan respons tombol dapat ditentukan. Pada contoh sebelumnya, tombol telah diprogram untuk mengirim pesan respons kembali ke bot menggunakan metode `imBack()`. Namun, Anda juga dapat memprogramnya untuk membuka halaman web menggunakan metode `openUrl()`, sebagai berikut:

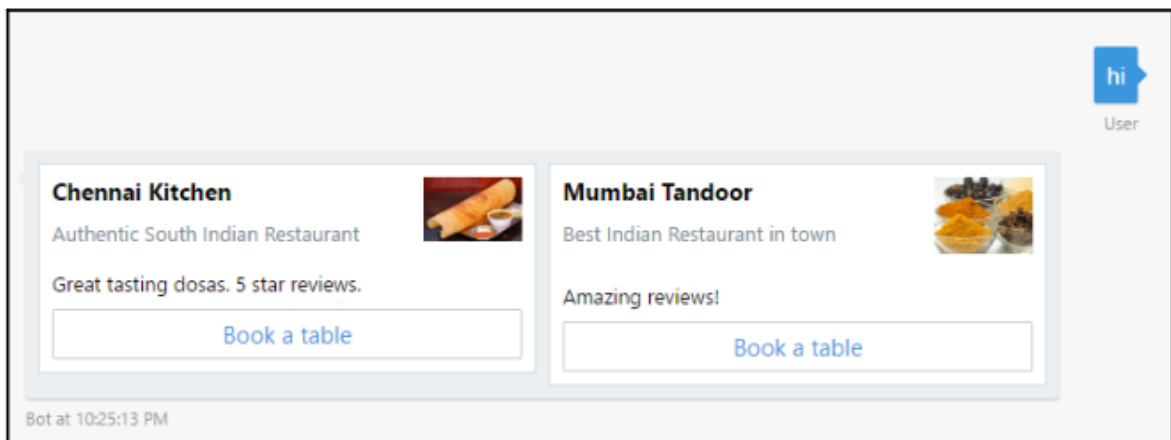
```
Builder.CardAction.openUr1(session,  
'https://mumbaitandoor.com/bookTable',Book a Table');
```

Ada juga jenis kartu lainnya: Kartu thumbnail, Kartu adaptif, Kartu audio, dan Kartu animasi, misalnya. Untuk daftar lengkap kartu, lihat dokumentasi Bot Framework di <https://docs.microsoft.com/en-us/bot-framework/nodejs/bot-builder-nodejs-send-rich-cards>.

Kartu thumbnail mirip dengan kartu Pahlawan tetapi lebih kecil. Anda dapat membuat kartu Thumbnail menggunakan kelas `ThumbnailCard`, seperti yang ditunjukkan di sini:

```
new builder.ThumbnailCard(session)
    .title("Chennai Kitchen")
    .subtitle("Authentic South Indian Restaurant")
    .text("Great tasting dosas. 5 star reviews.")
    .images([builder.CardImage.create(session,
        'https://images.pexels.com/photos/221143/
        pexels-photo-221143.jpeg?w=940&h=650&
        auto=compress&cs=tinysrgb')])
    .buttons([
        builder.CardAction.imBack(session,
            "book_table:chennai_kitchen",
            "Book a table")
    ])
})
```

Mari jalankan kode sebelumnya di emulator:



Gambar 12.13 tampilan emulator terbaru

Mari buat kartu untuk menampilkan gambar GIF. Kelas `AnimationCard` dapat digunakan untuk menampilkan gambar animasi:

```

new builder.AnimationCard(session)
  .title('Microsoft Bot Framework')
  .subtitle('Animation Card')
  .image(builder.CardImage.create(session,
'https://makeYourOwnCurry.com/curryAnimation.jpeg'))
  .media([
    { url: 'http://i.giphy.com/Ki55RUbOV5njy.gif' }
  ])

```

Kartu Audio dan Video dapat digunakan untuk menyajikan informasi audio dan video:

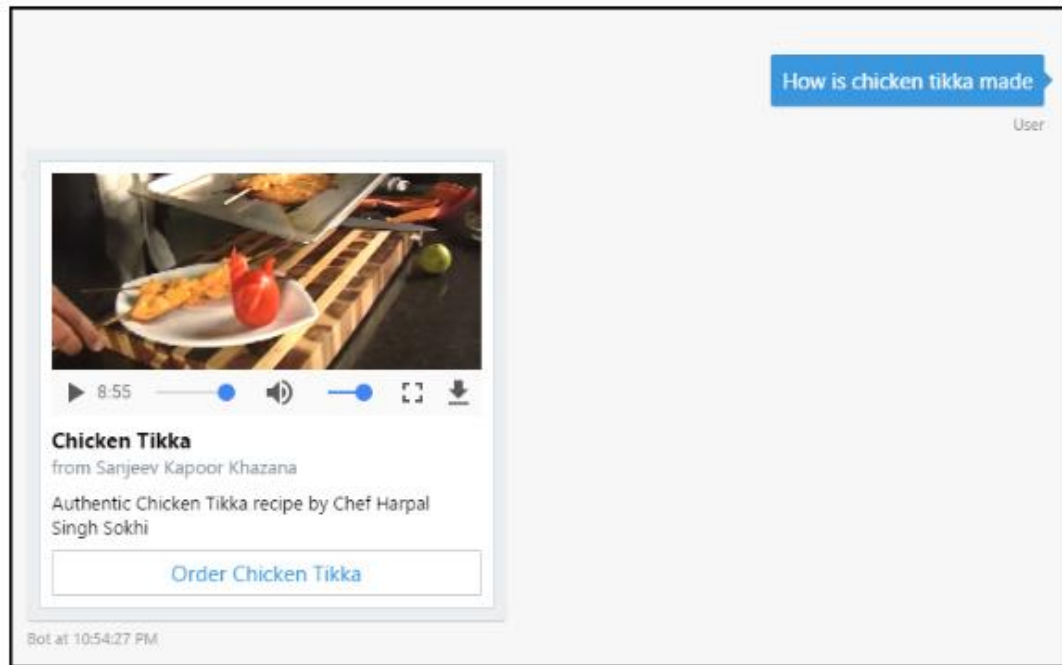
```

//Video card
new builder.VideoCard(session)
  .title('Chicken Tikka')
  .subtitle('from Sanjeev Kapoor Khazana')
  .text('Authentic Chicken Tikka recipe by Chef
Harpal Singh Sokhi')
  .image(builder.CardImage.create(session,
'https://commons.wikimedia.org/wiki/
File:Chicken_Tikka_(1).jpg'))
  .media([
    { url: 'http://fakevideourl.com/makingofchickentikka.mp4' }
  ])
  .buttons([
    builder.CardAction.imBack(session, "order:chicken_tikka",
"Order Chicken Tikka")
  ])
//Audio card

new builder.AudioCard(session)
  .title('Delicious Chicken Tikka')
  .subtitle('Must have at Mumbai Tandoor')
  .text('User')
  .image(builder.CardImage.create(session,
'https://commons.wikimedia.org/wiki/
File:Chicken_Tikka_(1).jpg'))
  .media([
    { url: 'http://fakeaudiourl.com/reviews1.wav' }
  ])
  .buttons([
    builder.CardAction.imBack(session, "order:chicken_tikka",
"Order Chicken Tikka")
  ])

```

Mari kita lihat tampilannya di emulator:



Gambar 12.14 tampilan emulator pada chatbot (menambahkan video)

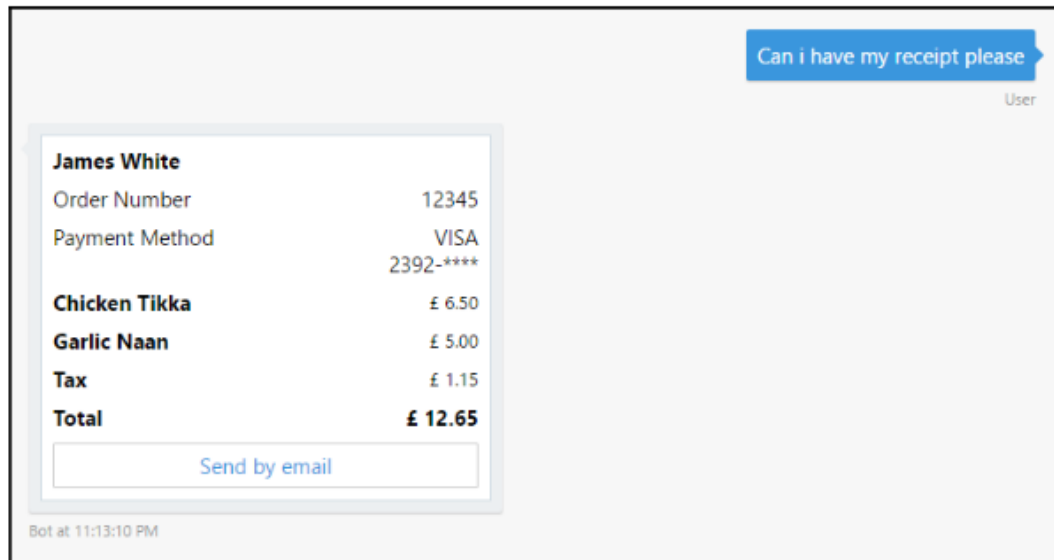
Selain kartu tersebut, terdapat kartu khusus bernama Kartu Resi yang akan menyajikan informasi dalam format resi. Dapat digunakan untuk menyajikan tagihan yang diperinci dengan informasi pembayaran, sebagai berikut:

```

new builder.ReceiptCard(session)
  .title('James White')
  .facts([
    builder.Fact.create(session, '12345', 'Order Number'),
    builder.Fact.create(session, 'VISA 2392-****',
      'Payment Method')
  ])
  .items([
    builder.ReceiptItem.create(session, '£ 6.50', 'Chicken Tikka')
      .quantity(1),
    builder.ReceiptItem.create(session, '£ 5.00', 'Garlic Naan')
      .quantity(2)
  ])
  .tax('£ 1.15')
  .total('£ 12.65')
  .buttons([
    builder.CardAction.imBack(session, 'sendemail',
      'Send by email')
  ])
  )

```

Mari kita jalankan di emulator:

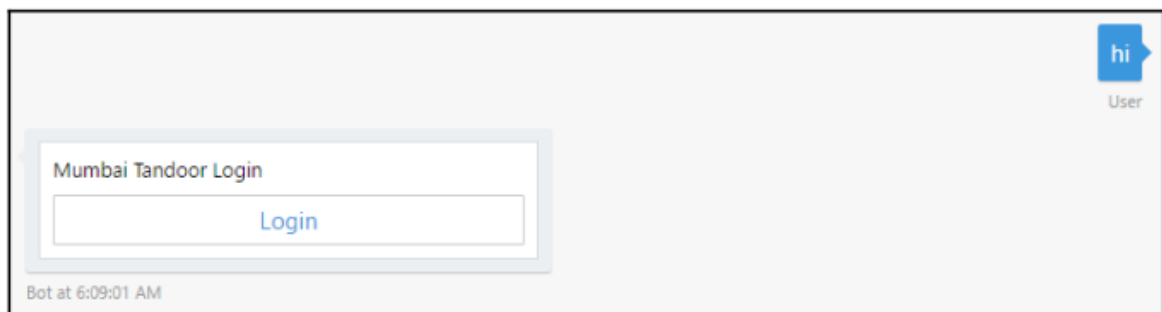


Gambar 12.15 tampilan emulator pada chatbot (menambahkan harga)

Terakhir, terdapat kartu yang dapat digunakan untuk mengautentikasi pengguna dengan meminta mereka masuk. Alur ini dapat dimulai menggunakan kartu Masuk:

```
New builder.SigninCard (session)  
.text('Mumbai Tandoor Login')  
.button ('Login','https://mumbaitandoor.com/login')  
]);
```

Mengklik kartu Masuk akan membawa pengguna ke halaman web tempat pengguna dapat diautentikasi:



Gambar 12.16 halaman login

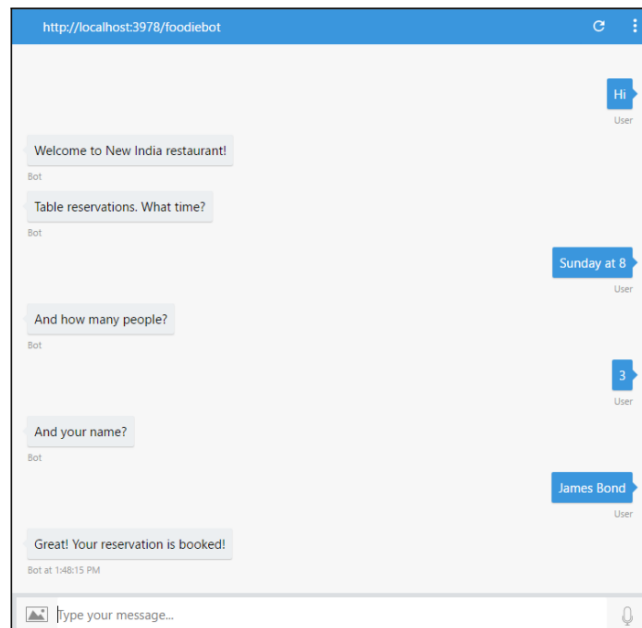
Sekarang setelah kita menjelajahi kartunya, mari beralih ke penerapan alur percakapan.

12.3 ALUR PERCAKAPAN

Sekarang kita telah menyiapkan untuk menguji chatbot dan menjelajahi berbagai cara menyajikan informasi kepada pengguna, mari kita periksa cara mengelola alur percakapan. Model dasar yang tersedia bagi kita adalah model air terjun, dimana percakapan terdiri dari serangkaian langkah. Mari kita ambil contoh pemesanan meja di sebuah restoran dimana percakapan berlangsung dengan cara berikut: dapatkan waktu reservasi, jumlah orang di meja, dan nama pengguna:

```
// Bot Dialogs
var bot = new builder.UniversalBot(connector, [
  function (session) {
    session.send('Welcome to New India restaurant!');
    builder.Prompts.time(session, 'Table reservations.
      What time?');
  },
  function (session, results) {
    session.dialogData.timeOfReservation =
      builder.EntityRecognizer.resolveTime([results.response]);
    builder.Prompts.number(session, "And how many people?");
  },
  function (session, results) {
    session.dialogData.numberOfPeople = results.response;
    builder.Prompts.text(session, "And your name?");
  },
  function (session, results) {
    session.dialogData.nameOnReservation = results.response;
    session.send('Great! Your reservation is booked!');
  }
]);
```

Mari kita coba di emulator:



Gambar 12.17 pemesanan meja

Mari kita membedah kodenya sedikit untuk memahami apa yang terjadi. Kita mulai dengan membuat bot menggunakan kelas UniversalBot. Saat kami membuat bot, kami menentukan langkah-langkah percakapan air terjun sebagai serangkaian fungsi. Ini adalah dialog akarnya. Setiap fungsi merupakan langkah dalam percakapan. Di setiap langkah, bot mengatakan atau meminta pesan kepada pengguna. Jika ada perintah, ia mengharapkan pengguna untuk merespons. Responsnya disimpan di `results.response`, yang diperbarui ke status dialog yang dikelola di `session.dialogData`. Penugasan ini terjadi pada langkah selanjutnya, bot membuat ucapan berikutnya atau meminta informasi lebih lanjut. Seperti yang telah kita bahas sebelumnya, ada berbagai cara untuk mendapatkan dan memverifikasi informasi.

Tidak selalu mungkin untuk memetakan keseluruhan percakapan sebagai serangkaian fungsi. Bagaimana jika ada bagian percakapan yang berulang? Sebagai pemrogram, kami menangani situasi ini menggunakan fungsi dan metode. Metode adalah bagian kode yang terdefinisi dengan baik yang melakukan tugas tertentu dan dapat dipanggil kapan pun diperlukan oleh metode utama atau metode lain. Mari kita ambil contoh, tugas pembayaran saat melakukan pemesanan. Baik Anda sedang makan atau memesan makanan untuk dibawa pulang, Anda harus melakukan pembayaran dengan cara yang sama. Serangkaian pertanyaan yang sama akan ditanyakan: membayar dengan kartu atau tunai, nomor kartu, nama pada kartu, nomor CVV, dan seterusnya. Bayangkan dialog pembayaran antara pengguna dan bot. Akankah dialog ini digunakan di lebih dari satu skenario? Bukankah lebih baik jika langkah percakapan dalam dialog pembayaran dipisahkan dan memanggil prosesnya kapan pun pembayaran perlu dilakukan? Inilah yang bisa kita capai dengan menggunakan metode `dialog()`.

Bot yang kita buat menggunakan kelas UniversalBot dapat diberikan keterampilan percakapan untuk melaksanakan berbagai tugas, seperti pembayaran dan daftar produk, menggunakan metode `dialog()`. Ini kemudian dapat dipanggil bila diperlukan dari dialog akar. Setiap metode `dialog()` dapat digunakan untuk mendefinisikan sub-dialog, dan secara struktural akan menjadi dialog air terjun yang independen. Sekarang mari kita membangun dialog dasar dan menyematkan di dalamnya dua sub-dialog yang meminta pesanan dan meminta pembayaran:

```

//Main dialogue
var bot = new builder.UniversalBot(connector, [
  function (session) {
    session.send("Welcome to New India restaurant.");
    session.beginDialog('askForOrder');
  },
  function (session) {
    session.beginDialog('askForPayment');
  },
  function (session) {
    session.send('Thanks for your order!');
    session.send(`Order summary:
    ${session.conversationData.order}<br/>` +
    `Payment card number:

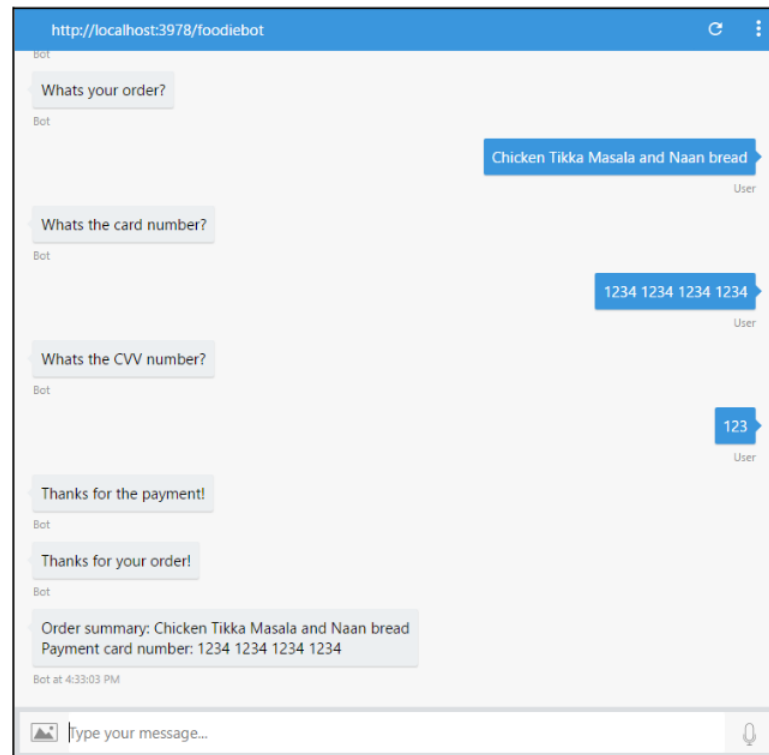
    ${session.conversationData.cardNumber}<br/>`);
    session.endDialog();
  }
]);

// Ask for Order
bot.dialog('askForOrder', [
  function (session) {
    builder.Prompts.text(session, 'Whats your order?');
  },
  function (session, results) {
    session.conversationData.order = results.response;
    session.endDialog();
  }
]);

// Ask for payment
bot.dialog('askForPayment', [
  function (session) {
    builder.Prompts.text(session, 'Whats the card number?');
  },
  function (session, results) {
    session.conversationData.cardNumber = results.response;
    builder.Prompts.text(session, 'Whats the CVV number?');
  },
  function (session, results) {
    session.conversationData.cardCVVNumber = results.response;
    session.send('Thanks for the payment!');
    session.endDialog();
  }
]);

```

Pada kode sebelumnya, Anda dapat melihat tiga dialog: root, meminta pesanan, dan meminta pembayaran. Dalam dialog root, kami menggunakan `session.beginDialog()` untuk memanggil sub-dialog:



Gambar 12.18 chatbot untuk meminta pembayaran

Selain itu, perhatikan bahwa sejauh ini kita telah menggunakan `session.dialogData` untuk menyimpan informasi dari ucapan pengguna. Namun sekarang, kami menggunakan `session.conversationData`. Kami akan mengeksplorasi perbedaan di antara keduanya nanti.

Pada titik ini, kita perlu memahami konsep tumpukan dialog. Pada awalnya, tumpukan dialog berisi dialog root. Saat subdialog dipanggil dari akar, subdialog tersebut ditumpuk di atas dialog akar. Sub-dialog sendiri dapat menyebut sub-dialog lainnya. Ini, pada gilirannya, akan ditumpuk di atasnya. Ketika sub-dialog selesai, bot mengembalikan dialog berikutnya di tumpukan dan terus melakukannya hingga tidak ada lagi dialog.

12.4 MENANGGAPI UCAPAN PENGGUNA

Apa yang kita miliki sekarang adalah percakapan default yang dimulai dengan cara yang sama, apa pun yang dikatakan pengguna. Anda dapat menyapa, atau membantu, atau ucapan lainnya dan bot akan menjawab dengan pesan selamat datang. Cara lain untuk memulai percakapan adalah berdasarkan apa yang dikatakan pengguna. Sekarang mari kita jelajahi cara merespons ketika pengguna mengatakan bantuan di tengah percakapan:

```

bot.dialog('help', function (session, args, next) {
    session.endDialog("Hi. I can take food orders.<br/>
    Say 'continue' to continue?");
})
.triggerAction({
    matches: /^help$/i,
});

```

Menanggapi ucapan pengguna dapat dilakukan dengan menambahkan `triggerAction()` dengan ucapan yang ditentukan sebagai ekspresi reguler di klausa kecocokan. Tambahkan kode sebelumnya ke `app.js` dan mulai ulang server. Sekarang percakapannya mungkin seperti berikut:



Gambar 12.19 tanggapan untuk info

Kode yang kami tambahkan memungkinkan bot merespons ucapan pengguna bantuan. Perhatikan bagaimana hal ini mengesampingkan ekspektasi informasi pesanan saat ini yang ditunggu-tunggu oleh bot.

Ada dua cara lain untuk menafsirkan ucapan pengguna: pengenalan khusus dan menggunakan layanan NLU seperti LUIS. Mari kita coba pengenalan khusus terlebih dahulu. Ke bot Anda, lampirkan pengenalan berikut:

```

bot.recognizer({
  recognize: function (context, done) {
    var intent = { score: 0.0 };
    if (context.message.text) {
      switch (context.message.text.toLowerCase()) {
        case 'help':
          intent = { score: 1.0, intent: 'get-help' };
          break;
        case 'goodbye':
          intent = { score: 1.0, intent: 'say-goodbye' };
          break;
      }
    }
    done(null, intent);
  }
});

```

Dan buatlah sub-dialog yang sesuai dengan maksudnya:

```

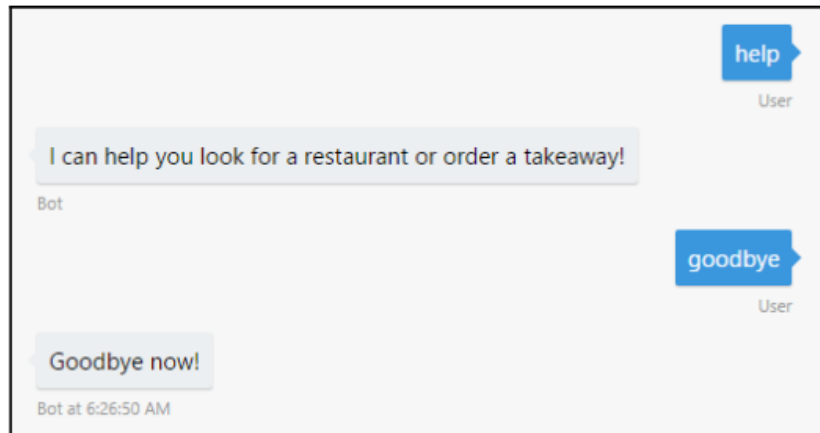
bot.dialog('help', [
  function (session) {
    session.send('I can help you look for a
      restaurant or order a takeaway!');
    session.endDialog();
  }
]).triggerAction({
  matches: 'get-help'
});

bot.dialog('goodbye', [
  function (session) {
    session.send('Goodbye now!');
    session.endConversation();
  }
]).triggerAction({
  matches: 'say-goodbye'
});

```

Meskipun pola ucapan dapat ditentukan menggunakan opsi kecocokan untuk setiap subdialog, akan lebih baik jika mengaturnya sebagai maksud menggunakan pengenalan global untuk semua subdialog.

Hal ini untuk memastikan bahwa kita tidak perlu menduplikasi ekspresi reguler. Setelah maksud teridentifikasi, maksud tersebut dapat digunakan untuk memicu sub-dialog yang sesuai, seperti yang ditunjukkan di sini:



Gambar 12.20 duplikasi ekspresi reguler

LUIS adalah layanan pemahaman bahasa alami Microsoft. Ini mirip dengan API.AI Google dan Lex Amazon. Salah satu alat ini dapat diintegrasikan dengan bot untuk menyediakan layanan NLU jika pengenalan khusus, seperti sebelumnya, tidak memadai. Untuk menjelajahi opsi ini lebih lanjut, lihat dokumentasi resmi di <https://docs.microsoft.com/en-us/bot-framework/nodejs/bot-builder-nodejs-recognize-intent-luis>.

Menjaga konteks tetap utuh

Memproses ucapan pengguna menggunakan subdialog dapat membuat percakapan keluar dari konteks:



Gambar 12.21 sub dialog percakapan agar menjaga tetap utuh

Apa yang terjadi dengan percakapan saat pengguna mengetik lanjutkan, seperti yang disebutkan dalam pesan bantuan? Apakah pembicaraan berlanjut? Tidak, tidak. Bot tersebut

sepertinya sudah benar-benar lupa dengan apa yang dilakukannya sebelumnya. Hal ini karena tumpukan dialog dihapus ketika ucapan pengguna diproses. Mungkin ideal untuk menghapus tumpukan saat pengguna ingin mengubah topik pembicaraan, namun tidak saat pengguna meminta bantuan.

Ada cara untuk menjaga konteks tetap utuh bahkan ketika pengguna menyela dengan pertanyaan dan komentar. Hal ini dapat dilakukan dengan menambahkan opsi `onSelectAction` ke subdialog yang dipanggil. Ini akan menjaga tumpukan dialog tetap utuh dan tidak menghapusnya:

```
bot.dialog('help', function (session, args, next) {
  session.endDialog("Hi there. I can take food orders.");
})
.triggerAction({
  matches: /^help$/i,
  onSelectAction: (session, args, next) => {
    session.beginDialog(args.action, args);
  }
});
```

Jadi sebenarnya merupakan bagian dari keputusan desain untuk menentukan di mana tumpukan dialog perlu dibersihkan dan di mana tidak. Misalnya, ketika pengguna meminta bantuan, lebih baik tidak menghapus konteksnya karena permintaan bantuan tersebut mungkin terkait dengan konteksnya. Namun, jika pengguna tampaknya beralih ke tugas lain (misalnya, meminta pemesanan meja padahal sebenarnya mereka sedang memesan makanan), sebaiknya hapus konteksnya karena tidak pantas untuk kembali mengambil makanan tersebut. pesan setelah meja sudah dipesan.

12.5 PERALIHAN KONTEKS

Namun, mungkin ada kasus di mana pengguna ingin beralih dari satu tugas ke tugas lainnya. Dalam kasus seperti ini, kami tidak ingin tumpukan dialog tetap utuh. Dengan tidak menggunakan opsi `onSelectAction`, kita dapat menghapus tumpukan dialog. Namun, ada baiknya juga memberi tahu pengguna bahwa bot akan meninggalkan tugas saat ini untuk mengambil tugas berikutnya. Hal ini dapat dilakukan dengan menggunakan opsi `konfirmasiPrompt` dalam metode `triggerAction()`:

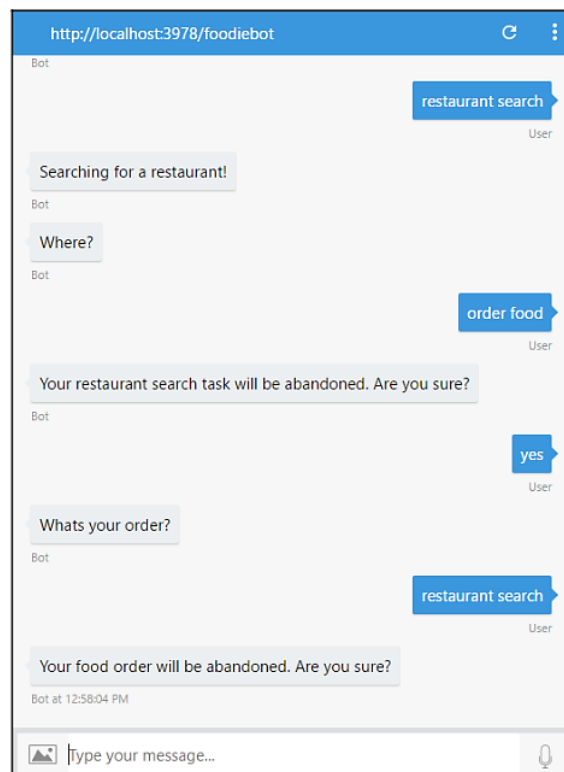
```

bot.dialog('askForOrder', [
  ...
])
.triggerAction({
  matches: /^order food$/i,
  confirmPrompt: "Your food order will be abandoned. Are you sure?"
});

// Search for a restaurant
bot.dialog('searchRestaurant', [
  function (session) {
    session.send('Searching for a restaurant!');
    builder.Prompts.text(session, 'Where?');
  },
  function (session, results) {
    session.conversationData.searchLocation = results.response;
    session.endDialog();
  }
])
.triggerAction({
  matches: /^restaurant search$/i,
  confirmPrompt: 'Your restaurant search task
will be abandoned. Are you sure?'
});

```

Dengan menambahkan metode `triggerAction()` ke dialog `askForOrder`, kita akan dapat merespons permintaan pengguna untuk memesan makanan kapan saja dalam percakapan. Namun, bot akan secara proaktif memberitahukan mereka bahwa tugas lain yang sedang dilakukan (misalnya, memesan meja) akan ditinggalkan:



Gambar 12.22 tampilan pemesanan meja

Perhatikan bagaimana konfirmasi konfirmasi dipanggil ketika pengguna beralih dari satu tugas ke tugas lainnya.

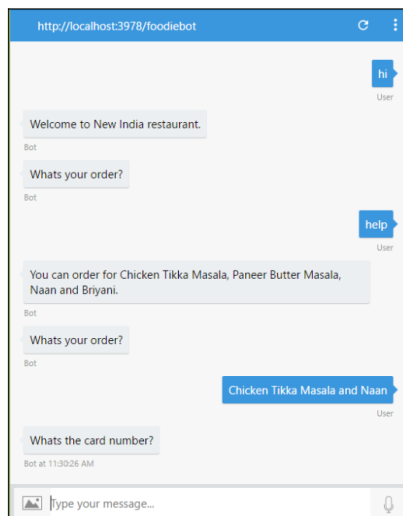
12.6 NLU KONTEKSTUAL

Mungkin juga ideal untuk memberikan bantuan dengan cara yang kontekstual. Misalnya, berikan menu kepada pengguna ketika mereka meminta bantuan saat memesan. Mari kita lihat bagaimana ujaran dapat diproses secara kontekstual. Untuk melakukan ini, kita perlu membuat sub-dialog bantuan dan menambahkannya ke dialog yang ada sehingga dapat terpicu ketika apa yang dikatakan pengguna cocok dengan template yang disediakan:

```
// Ask for Order
bot.dialog('askForOrder', [
  function (session) {
    builder.Prompts.text(session, 'Whats your order?');
  },
  function (session, results) {
    session.conversationData.order = results.response;
    session.endDialog();
  }
])
.beginDialogAction('orderHelpAction', 'orderHelp',
  { matches: /^help$/i });

// Contextual help for ordering
bot.dialog('orderHelp', function(session, args, next) {
  var msg = "You can order for Chicken Tikka Masala,
  Paneer Butter Masala, Naan and Briyani.";
  session.endDialog(msg);
})
```

Perhatikan bagaimana kita menggunakan metode `BeginDialogAction()` untuk menghubungkan subdialog `orderHelp`. `orderHelp` terpicu ketika pengguna mengatakan bantuan selama langkah pemesanan makanan:



Gambar 12.23 tampilan menggunakan bantuan pemesanan

Dalam percakapan sebelumnya, meminta bantuan tidak menghasilkan respons standar. Sebaliknya, kita mendapatkan yang kontekstual.

12.7 MENGAKHIRI PERCAKAPAN

Terakhir, merupakan praktik yang baik untuk mengakhiri percakapan setelah tugas selesai. Hal ini dilakukan dengan memberi tahu pengguna bahwa tugas telah selesai, menghapus tumpukan dialog, dan menyetel ulang objek `session.conversationData`. Untuk melakukannya, gunakan metode `session.endConversation()`. Jadi, mari kita tulis ulang dialog root kita dengan metode `session.endConversation()` :

```
//Main dialogue
var bot = new builder.UniversalBot(connector, [
  function (session) {
    session.send("Welcome to New India restaurant.");
    session.beginDialog('askForOrder');
  },
  function (session) {
    session.beginDialog('askForPayment');
  },
  function (session) {
    session.send('Thanks for your order!');
    session.send(`Order summary:
    ${session.conversationData.order}<br/>` +
      `Payment card number:
    ${session.conversationData.cardNumber}<br/>`);
    session.endConversation();
  }
]);
```

Anda juga dapat menyetel dialog default yang terpicu saat pengguna mengucapkan Selamat Tinggal dan mengakhiri percakapan:

```
bot.dialog('endConversation', [
  session.endConversation("Goodbye!")
])
.endConversationAction(
  "endTasks", "Ok. Goodbye.",
  {
    matches: /^goodbye$/i,
    confirmPrompt: "Cancelling current task. Are you sure?"
  }
);
```

Sekarang mari kita lihat bagaimana kita menyimpan konteks percakapan.

Keadaan Percakapan

Keadaan percakapan dapat disimpan dalam bentuk pasangan kunci/nilai. Ada empat penyimpanan data yang tersedia untuk melakukan hal ini. Ini ditempatkan di dalam objek sesi:

dialogData: Ingat, percakapan dibagi menjadi beberapa dialog (dialog akar dan subdialog dimulai dengan `BeginDialog()`). Dalam setiap dialog, keadaan dapat dipertahankan

secara terpisah. Hal ini dilakukan dengan menggunakan `session.dialogData`, yang telah kita gunakan pada contoh sebelumnya. Ini menyimpan data yang berkaitan dengan sub-dialog saat ini, dan setiap sub-dialog memiliki salinan `dialogData` sendiri. Ketika dialog selesai (yaitu, `endDialog()` dijalankan) dan dihapus dari tumpukan dialog, data ini akan dihapus.

ConversationData: Ini menyimpan data yang berkaitan dengan keseluruhan percakapan dan dibagikan di antara semua anggota (yaitu, pengguna) dalam percakapan. Itu akan dihapus ketika percakapan berakhir atau ketika `endConversation()` dijalankan. Data ini dapat diakses menggunakan `session.conversationData`.

privateConversationData: Ini menyimpan data yang berkaitan dengan keseluruhan percakapan sebagai `ConversationData`, namun bersifat pribadi untuk setiap anggota percakapan. Ini tidak dibagikan kepada anggota lain yang berpartisipasi dalam percakapan. Itu akan dihapus ketika percakapan berakhir atau ketika `endConversation()` dijalankan. Data ini dapat diakses menggunakan `session.privateConversationData`.

userData: Data pribadi yang berkaitan dengan pengguna dapat disimpan di sini. Hal ini terus terjadi di seluruh percakapan. Data seperti nama pengguna, umur, jenis kelamin, alamat, nomor telepon, email, dan info pembayaran dapat disimpan di sini dan digunakan dalam percakapan. Data ini dapat diakses menggunakan `session.userData`.

Sejauh ini, kita telah membahas sejumlah konsep dalam perangkat Pembuat Bot dengan contoh-contoh yang telah dikerjakan. Kita telah memeriksa cara menyiapkan dialog dasar dan memasukkan alur percakapan ke dalam sub-dialog. Kami telah mempelajari cara menangani ucapan pengguna secara lokal dan global, serta penggunaan berbagai objek data untuk menyimpan data pengguna dan percakapan. Sekarang mari kita lanjutkan membuat bot dengan data restoran.

12.8 MEMULAI ZOMATO

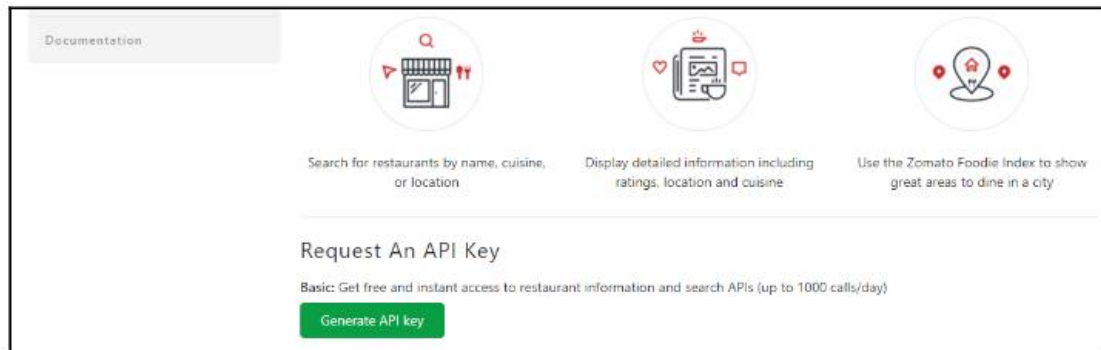
Pertama, kita perlu membuat akun dengan Zomato dan mendapatkan kunci API:

1. Kunjungi <https://developers.zomato.com/api>:



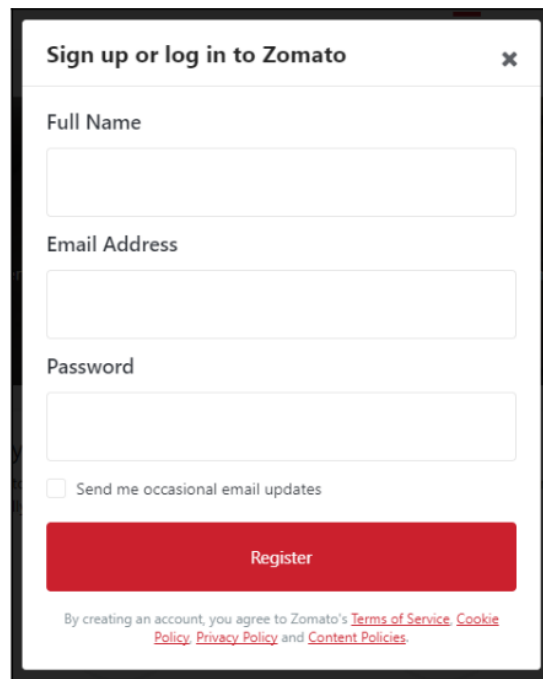
Gambar 12.23 halaman awal Zomato

2. Klik Hasilkan Kunci API:



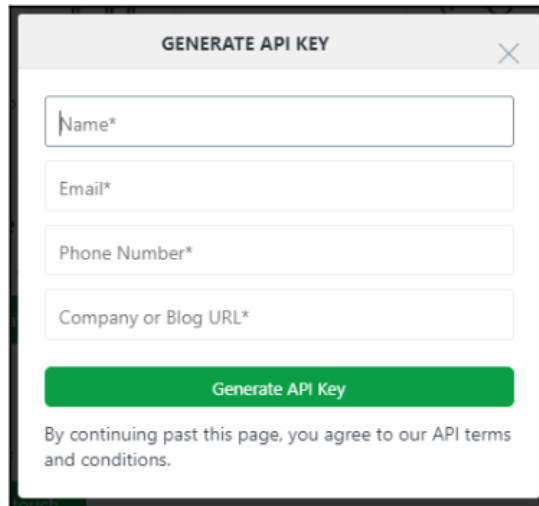
Gambar 12.24 hasil kunci API

3. Mendaftar:

The image shows a screenshot of the Zomato registration form. The form is titled 'Sign up or log in to Zomato' and has a close button (X) in the top right corner. It contains three input fields: 'Full Name', 'Email Address', and 'Password'. Below the 'Email Address' field is a checkbox labeled 'Send me occasional email updates'. At the bottom of the form is a red button labeled 'Register'. Below the button, there is a line of text: 'By creating an account, you agree to Zomato's [Terms of Service](#), [Cookie Policy](#), [Privacy Policy](#) and [Content Policies](#)'.

Gambar 12.25 Mendaftar Zomato

4. Konfirmasikan alamat email Anda dan klik Hasilkan Kunci API lagi:



Gambar 12.26 Mengisi data user Zamato

5. Salin kunci API.

Mendapatkan data dari Zomato

URL dasar untuk mengambil data adalah <https://developers.zomato.com/api/v2.1/>. Setiap query yang dikirim ke server Zomato harus memiliki kunci API yang dikirimkan sebagai bagian dari header, sebagai berikut:

```
curl -X GET --header "Accept: application/json" --header "user-key: YOURAPIKEY" "https://developers.zomato.com/api/v2.1/categories"
```

Dalam panggilan HTTP GET sebelumnya, kategori menentukan pegangan yang menampilkan berbagai kategori restoran yang tersedia. Kategorinya meliputi pengantaran, bawa pulang, dan makan di tempat. Berikut daftar layanan yang tersedia:

Menangani	Keterangan
/categories	Kategori restoran: makan di tempat, bawa pulang, pesan antar, dan banyak lagi.
/cities	Dapatkan info tentang kota-kota di seluruh dunia. Dapatkan ID kota Zomato dari kueri ini untuk digunakan dalam kueri penelusuran nanti. Pencarian menggunakan nama atau lintang dan bujur (latLong) kota.
/collections	Dapatkan daftar restoran berdasarkan ID kota atau koordinat garis lintang.
/cuisines	Dapatkan daftar masakan di kota.
/establishments	Dapatkan daftar kategori restoran yang tersedia di kota.
/locations	Dapatkan ID lokasi untuk dicari. Ini adalah ID pada skala berbeda: kota, subzona, zona, landmark, dan sebagainya. Ini akan digunakan untuk

	mencari restoran.
/search	Cari restoran di ID lokasi tertentu, masakan, kategori, dan sebagainya.
/reviews	Dapatkan ulasan tentang restoran tertentu.

Sekarang kita telah menjelajahi data yang dimiliki Zomato tentang restoran, mari beralih ke membuat bot menggunakan Zomato dan Bot Builder SDK.

12.9 BOT PENCARIAN RESTORAN

Mari kita membangun chatbot yang dapat mencari restoran berdasarkan tujuan dan preferensi pengguna. Mari kita mulai dengan membuat modul Node.js untuk mendapatkan data dari Zomato berdasarkan preferensi pengguna. Buat file bernama `zomato.js`. Tambahkan modul permintaan ke perpustakaan Node.js menggunakan perintah berikut di konsol:

>npm install request --save

Di `zomato.js`, tambahkan kode berikut untuk memulai:

```
var request = require('request');

var baseUrl = 'https://developers.zomato.com/api/v2.1/';
var apiKey = 'YOUR_API_KEY';

var categories = null;
var cuisines = null;

getCategories();
getCuisines(76);
```

Ganti `YOUR_API_KEY` dengan kunci Zomato Anda. Mari buat fungsi untuk mendapatkan daftar kategori dan masakan saat startup. Kueri ini tidak perlu dijalankan saat pengguna meminta penelusuran restoran karena informasi ini cukup statis:

```
function getCuisines(cityId){
  var options = {
    uri: baseUrl + 'cuisines',
    headers: {
      'user-key': apiKey
    },
    qs: {'city_id':cityId},
    method: 'GET'
  }
  var callback = function(error, response, body) {
    if (error) {
      console.log('Error sending messages: ', error)
    } else if (response.body.error) {
      console.log('Error: ', response.body.error)
    } else {
      console.log(body);
      cuisines = JSON.parse(body).cuisines;
    }
  }
  request(options, callback);
}
```

Kode sebelumnya akan mengambil daftar masakan yang tersedia di kota tertentu (diidentifikasi dengan ID kota Zomato). Mari kita tambahkan kode untuk mengidentifikasi daftar kategori:

```
function getCategories(){
  var options = {
    uri: baseURL + 'categories',
    headers: {
      'user-key': apiKey
    },
    qs: {},
    method: 'GET'
  }
  var callback = function(error, response, body) {
    if (error) {
      console.log('Error sending messages: ', error)
    } else if (response.body.error) {
      console.log('Error: ', response.body.error)
    } else {
      categories = JSON.parse(body).categories;
    }
  }
  request(options, callback);
}
```

Sekarang kita sudah menyelesaikan fungsi dasarnya, mari kita membuat kode di kode pencarian restoran:

```
function getRestaurant(cuisine, location, category){
  var cuisineId = getCuisineId(cuisine);
  var categoryId = getCategoryId(category);

  var options = {
    uri: baseURL + 'locations',
    headers: {
      'user-key': apiKey
    },
    qs: {'query':location},
    method: 'GET'
  }
  var callback = function(error, response, body) {
    if (error) {
      console.log('Error sending messages: ', error)
    } else if (response.body.error) {
      console.log('Error: ', response.body.error)
    } else {
      console.log(body);
      locationInfo = JSON.parse(body).location_suggestions;
    }
  }
}
```

```

        search(locationInfo[0], cuisineId, categoryId);
    }
}
request(options, callback);
}

function search(location, cuisineId, categoryId){
    var options = {
        uri: baseUrl + 'search',
        headers: {
            'user-key': apiKey
        },
        qs: {'entity_id': location.entity_id,
            'entity_type': location.entity_type,
            'cuisines': [cuisineId],
            'categories': [categoryId]},
        method: 'GET'
    }
    var callback = function(error, response, body) {
        if (error) {
            console.log('Error sending messages: ', error)
        } else if (response.body.error) {
            console.log('Error: ', response.body.error)
        } else {
            console.log('Found restaurants:')
            var results = JSON.parse(body).restaurants;
            console.log(results);
        }
    }
    request(options, callback);
}
}

```

Kode sebelumnya akan mencari restoran di lokasi, masakan, dan kategori tertentu. Misalnya, Anda dapat mencari daftar restoran India di Newington, Edinburgh yang melayani pengiriman. Kita sekarang perlu mengintegrasikannya dengan kode chatbot. Mari kita buat file terpisah bernama `index.js`. Mari kita mulai dengan dasar-dasarnya:

```

var restify = require('restify');
var builder = require('botbuilder');
var request = require('request');

var baseUrl = 'https://developers.zomato.com/api/v2.1/';
var apiKey = 'YOUR_API_KEY';

var categories = null;
var cuisines = null;

```

```

getCategories();
//setTimeout(function(){getCategoryId('Delivery')}, 10000);

getCuisines(76);
//setTimeout(function(){getCuisineId('European')}, 10000);

// Setup Restify Server
var server = restify.createServer();
server.listen(process.env.port || process.env.PORT || 3978, function () {
  console.log('%s listening to %s', server.name, server.url);
});

// Create chat connector for communicating with
// the Bot Framework Service
var connector = new builder.ChatConnector({
  appId: process.env.MICROSOFT_APP_ID,
  appPassword: process.env.MICROSOFT_APP_PASSWORD
});

// Listen for messages from users
server.post('/foodiebot', connector.listen());

```

Tambahkan kode dialog bot untuk melakukan pencarian restoran. Mari kita rancang bot untuk menanyakan masakan, kategori, dan lokasi sebelum melanjutkan ke pencarian restoran:

```

var bot = new builder.UniversalBot(connector, [
  function (session) {
    session.send("Hi there! Hungry? Looking for a restaurant?");
    session.send("Say 'search restaurant' to start searching.");
    session.endDialog();
  }
]);

// Search for a restaurant
bot.dialog('searchRestaurant', [
  function (session) {
    session.send('Ok. Searching for a restaurant!');
    builder.Prompts.text(session, 'Where?');
  },
  function (session, results) {
    session.conversationData.searchLocation = results.response;
    builder.Prompts.text(session, 'Cuisine? Indian, Italian, or anything else?');
  },
  function (session, results) {
    session.conversationData.searchCuisine = results.response;
  }
]);

```

```

        builder.Prompts.text(session, 'Delivery or Dine-in?');
    },
    function (session, results) {
        session.conversationData.searchCategory = results.response;
        session.send('Ok. Looking for restaurants..');
        getRestaurant(session.conversationData.searchCuisine,
                      session.conversationData.searchLocation,
                      session.conversationData.searchCategory,
                      session);
    }
    ])
    .triggerAction({
    matches: /^search restaurant$/i,
    confirmPrompt: 'Your restaurant search task will be abandoned.
                  Are you sure?'
    });
});

```

Perhatikan bahwa kita memanggil fungsi `getRestaurant()` dengan empat parameter. Tiga di antaranya telah kami definisikan: masakan, lokasi, dan kategori. Untuk ini, kita harus menambahkan yang lain: sesi. Ini meneruskan penunjuk sesi yang dapat digunakan untuk mengirim pesan ke emulator ketika data sudah siap. Perhatikan bagaimana hal ini mengubah fungsi `getRestaurant()` dan `search()`:

```

function getRestaurant(cuisine, location, category, session){
    var cuisineId = getCuisineId(cuisine);
    var categoryId = getCategoryId(category);
    var options = {
        uri: baseUrl + 'locations',
        headers: {
            'user-key': apiKey
        },
        qs: {'query':location},
        method: 'GET'
    }
    var callback = function(error, response, body) {
        if (error) {
            console.log('Error sending messages: ', error)
        } else if (response.body.error) {
            console.log('Error: ', response.body.error)
        } else {
            console.log(body);
            locationInfo = JSON.parse(body).location_suggestions;
            search(locationInfo[0], cuisineId,
                  categoryId, session);
        }
    }
    request(options, callback);
}

```

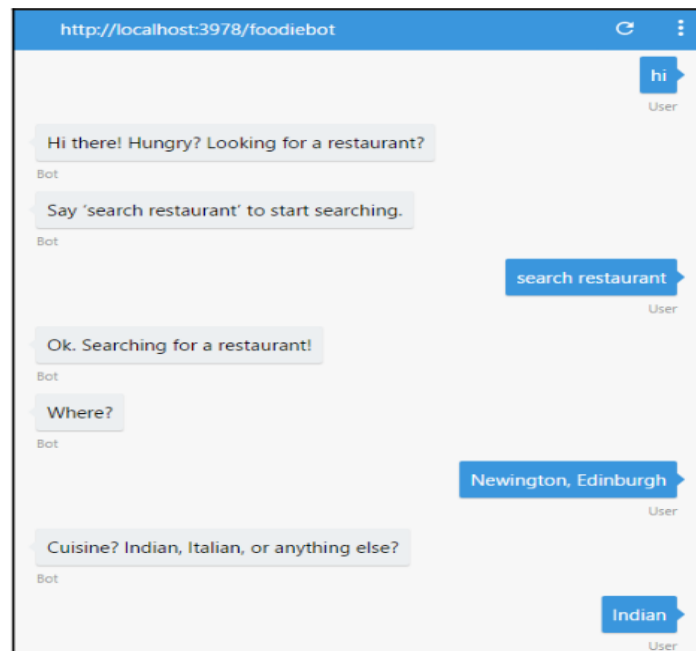
```

}

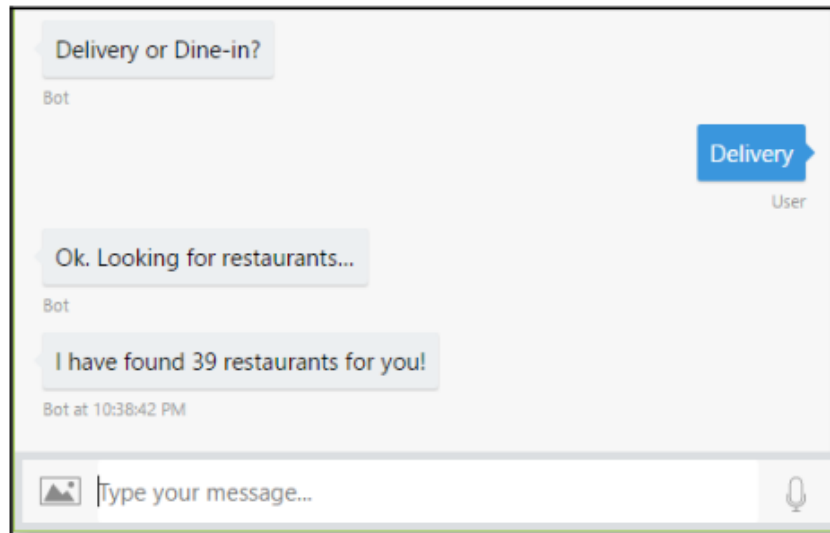
function search(location, cuisineId, categoryId, session){
  var options = {
    uri: baseURL + 'search',
    headers: {
      'user-key': apiKey
    },
    qs: {'entity_id': location.entity_id,
        'entity_type': location.entity_type,
        'cuisines': [cuisineId],
        'category': categoryId},
    method: 'GET'
  }
  var callback = function(error, response, body) {
    if (error) {
      console.log('Error sending messages: ', error)
    } else if (response.body.error) {
      console.log('Error: ', response.body.error)
    } else {
      console.log('Found restaurants:')
      console.log(body);
      //var results = JSON.parse(body).restaurants;
      //console.log(results);
      var resultsCount = JSON.parse(body).results_found;
      console.log('Found:' + resultsCount);
      session.send('I have found ' + resultsCount +
        ' restaurants for you!');
      session.endDialog();
    }
  }
  request(options, callback);
}

```

Setelah hasilnya diperoleh, bot merespons menggunakan `session.send()` dan mengakhiri dialog:



Gambar 12.27 Respon dan mengakhiri percakapan



Gambar 12.28 Chatbot Pencarian restoran

Sekarang setelah kita mendapatkan hasilnya, mari kita sajikan dengan cara yang lebih menarik secara visual menggunakan kartu. Untuk melakukan ini, kita memerlukan fungsi yang dapat mengambil hasil pencarian dan mengubahnya menjadi array kartu:

```
function presentInCards(session, results){
  var msg = new builder.Message(session);
  msg.attachmentLayout(builder.AttachmentLayout.carousel)
  var heroCardArray = [];
  var l = results.length;
  if (results.length > 10){
    l = 10;
  }
  for (var i = 0; i < l; i++){
    var r = results[i].restaurant;
    var herocard = new builder.HeroCard(session)
      .title(r.name)
      .subtitle(r.location.address)
      .text(r.user_rating.aggregate_rating)
      .images([builder.CardImage.create(session, r.thumb)])
      .buttons([
        builder.CardAction.imBack(session,
          "book_table:" + r.id,
          "Book a table")
      ]);
    heroCardArray.push(herocard);
  }
  msg.attachments(heroCardArray);
  return msg;
}
```

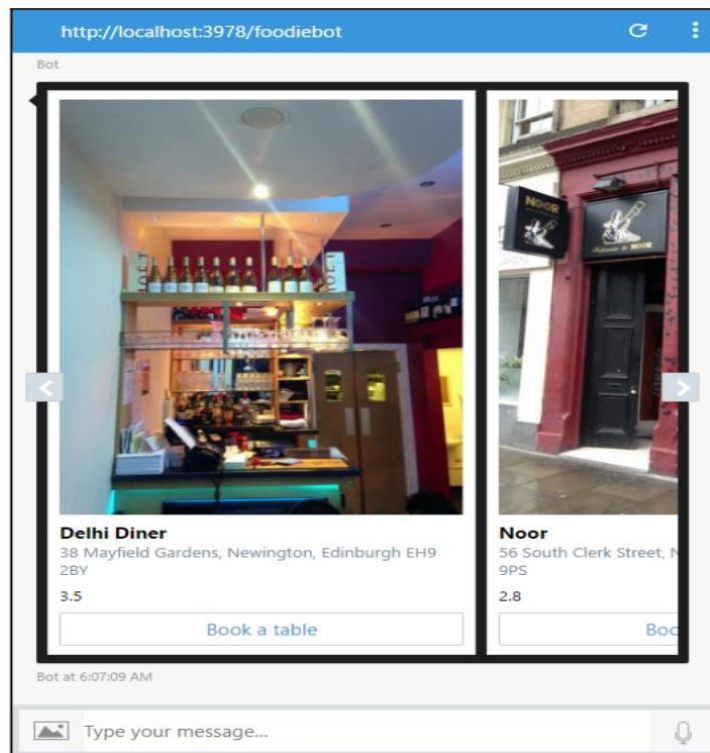
Dan kami memanggil fungsi ini dari fungsi search() :

```

function search(location, cuisineId, categoryId, session){
  var options = {
    uri: baseUrl + 'search',
    headers: {
      'user-key': apiKey
    },
    qs: {'entity_id': location.entity_id,
        'entity_type': location.entity_type,
        'cuisines': [cuisineId],
        'category': categoryId},
    method: 'GET'
  }
  var callback = function(error, response, body) {
    if (error) {
      console.log('Error sending messages: ', error)
    } else if (response.body.error) {
      console.log('Error: ', response.body.error)
    } else {
      console.log('Found restaurants:')
      console.log(body);
      var results = JSON.parse(body).restaurants;
      var msg = presentInCards(session, results);
      session.send(msg);
      session.endDialog();
    }
  }
  request(options, callback);
}

```

Berikut tampilannya:

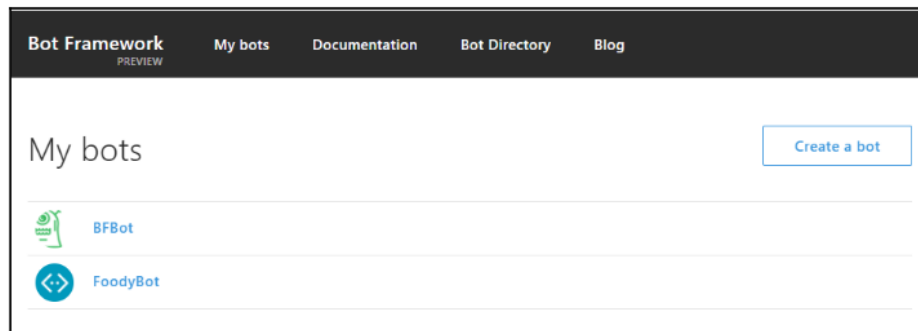


Gambar 12.29 tampilan hasil pencarian

12.10 MENGHUBUNGGAN KE SKYPE

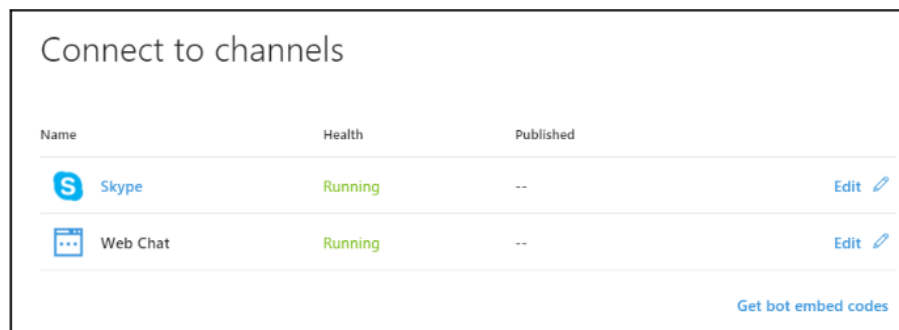
Sekarang kita memiliki chatbot untuk mencari restoran, dorong kembali ke cloud Heroku seperti sebelumnya. Ingatlah untuk mengubah Procfile karena kita perlu menjalankan index.js dan bukan app.js. Setelah menyiapkan chatbot sebagai aplikasi web di Heroku, kami siap untuk menerapkan bot di Skype dan saluran lainnya.

1. Buka dasbor Anda di Bot Framework di <https://dev.botframework.com/bots>:



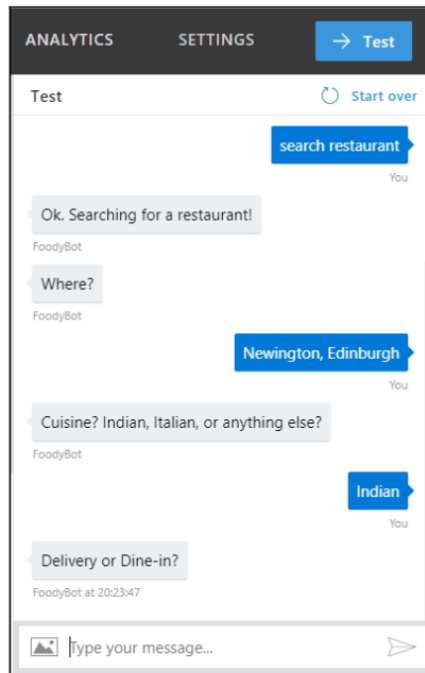
Gambar 12.30 dasbor MyBots

2. Pilih bot Anda. Anda akan melihat bahwa bot Anda sudah terhubung ke dua saluran, Skype dan Web Chat:



Gambar 12.31 Pilihan saluran

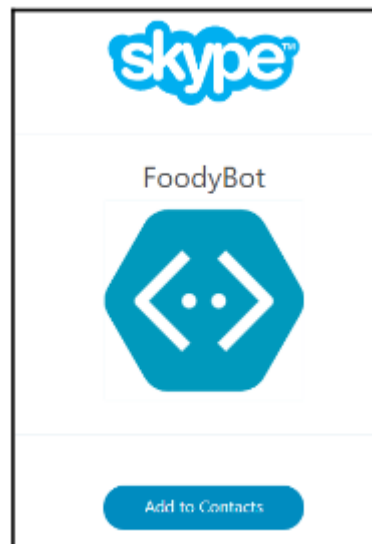
3. Anda juga akan dapat melihat klien obrolan di sisi kanan. Jika Anda tidak melihatnya, Anda dapat membukanya dengan menekan tombol Test. Ini mirip dengan emulator. Jadi, lanjutkan dan ucapkan Hai kepada bot:



Gambar 12.32 ujicoba Chatbot

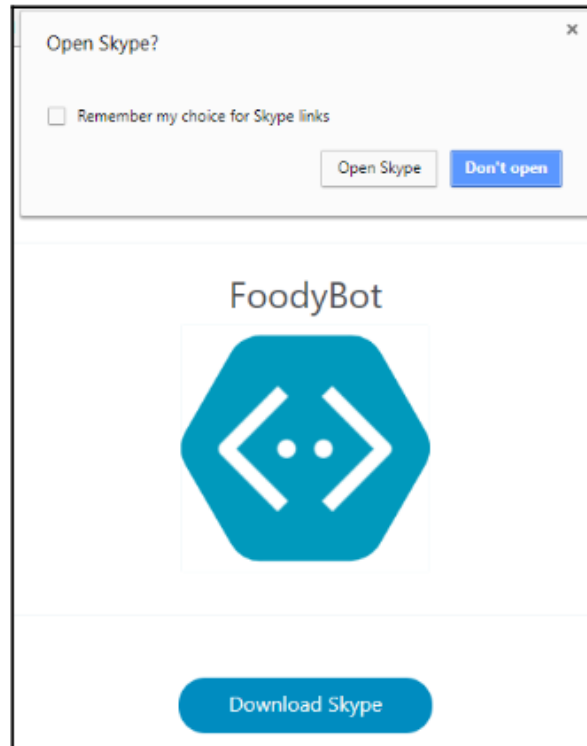
Anda seharusnya dapat mengobrol dengan bot di sini karena bot tersebut berada di cloud dan bukan di localhost.

4. Klik Skype:



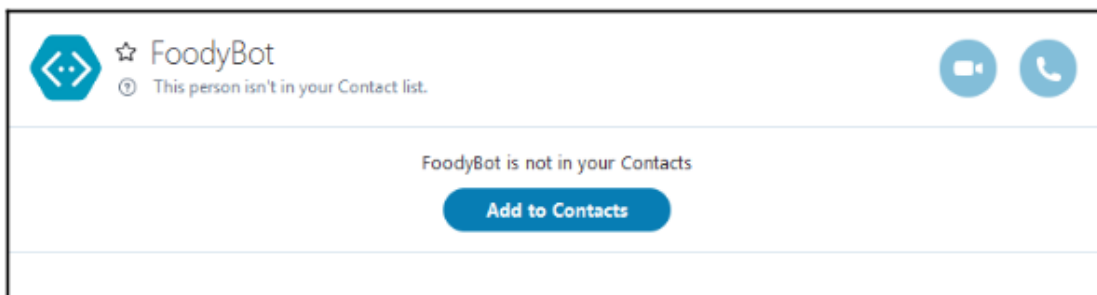
Gambar 12.33 menghubungkan kedalam Skype

5. Klik Tambahkan ke Kontak:



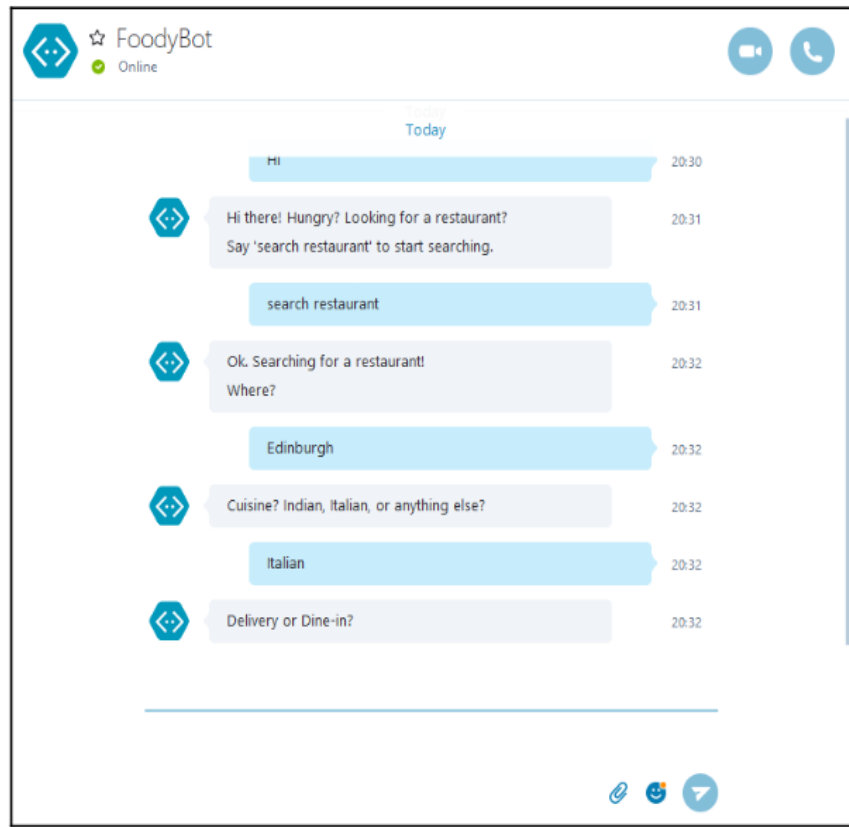
Gambar 12.34 menambahkan ke Kontak skype

6. Klik Buka Skype:

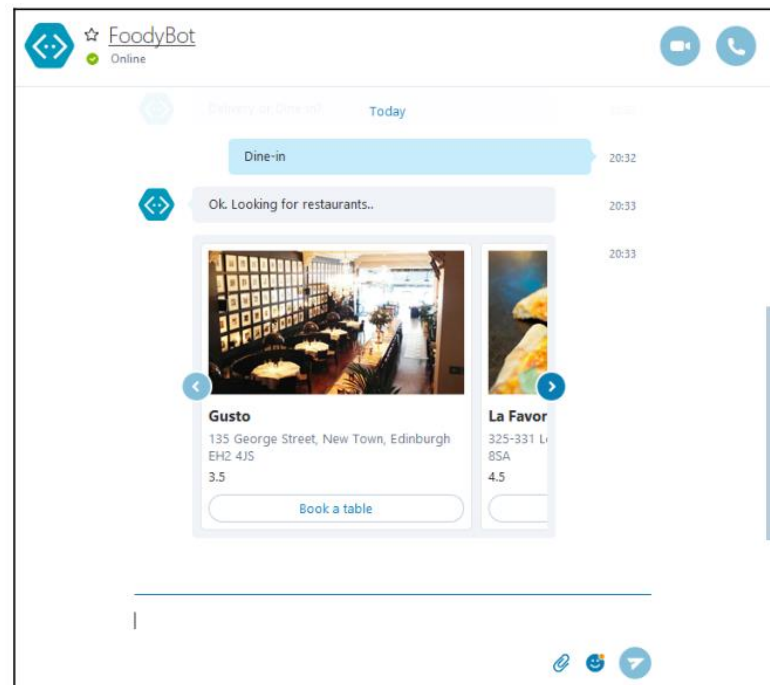


Gambar 12.35 membuka skype setelah dihubungkan

7. Tambahkan bot ke kontak Anda dan mulai mengobrol:



Gambar 12.36 mencoba dialog chatbot



Gambar 12.37 mencoba dialog chatbot untuk membuka gambar

Selamat atas bot baru Anda di Skype. Sekarang kita telah membuat bot menggunakan Bot Builder SDK dan menyebarkannya di Skype, jelajahi semua opsi lain yang kita pelajari tetapi tidak sempat kita coba. Tambahkan dialog pembuatan profil pengguna tempat bot mempelajari preferensi pengguna. Perluas dialog pencarian restoran dengan mengajukan pertanyaan lanjutan berdasarkan ulasan, penilaian, dan kisaran harga setelah restoran ditemukan dan memerlukan penyortiran lebih lanjut. Cobalah untuk menambahkan dukungan bahasa alami sehingga pengguna dapat beralih antar tugas dengan mudah.

Ringkasan

Pada bab sebelumnya, kita telah menjelajahi dan mempelajari cara membuat chatbot menggunakan berbagai alat. Ini termasuk lingkungan pengembangan seperti Chatfuel, alat pemrosesan bahasa alami seperti API.AI, dan SDK khusus saluran seperti Messenger SDK. Namun, ketika mengkodekan alur percakapan untuk mengelola percakapan, kami menggunakan alat berbasis formulir atau membuatnya dari awal. Namun, ada jalan tengahnya. MS Bot Framework menawarkan Bot Builder SDK yang dapat digunakan untuk mengembangkan modul manajemen percakapan yang mengatur bagaimana dialog mengalir antara bot dan pengguna. SDK memodelkan elemen percakapan dengan cara yang elegan sehingga memungkinkan pengembang membuat chatbot dengan cepat dan mudah. Hal ini memberikan fleksibilitas yang lebih besar kepada pengembang dibandingkan alat seret dan lepas, serta menghemat waktu dan tenaga dibandingkan membuat bot dari awal.

BAB 13

MENJELAJAHI PERANGKAT ASISTEN PINTAR DENGAN AI

Dalam bab ini, kita akan menjelajahi genre asisten baru yang mengambil alih dunia, asisten rumah tangga bertenaga suara. Ini adalah chatbot bertenaga AI yang mengambil fungsi asisten Anda di rumah. Mereka duduk di meja di ruang tamu Anda dan berinteraksi dengan Anda menggunakan suara. Mereka disebut speaker pintar tetapi bukan sekedar speaker. Mereka adalah asisten, sama seperti yang telah kita jelajahi di bab-bab sebelumnya, namun dengan perwujudannya sendiri. Salah satu yang lebih populer disebut Alexa, dan dibuat oleh Amazon. Sedangkan perwujudannya yaitu pembicara disebut Echo, asistennya disebut Alexa.

Ada speaker pintar lain di pasaran (misalnya, Google Home). Namun, Amazon Echo tampaknya memimpin persaingan dengan pangsa pasar sekitar 70% pada tahun 2017. Penggunaan asisten rumah speaker pintar juga diproyeksikan tumbuh secara eksponensial seiring berjalannya waktu (<https://www.emarketer.com/Article/Alexa-Ucapkan-Apa-Penggunaan-Speaker-Suara-Bertumbuh-Hampir-130-Tahun-Ini/1015812>). Dalam bab ini, mari kita jelajahi cara memperluas keahlian Alexa dengan membangun keterampilan kita sendiri. Pertama-tama kami akan membangun keterampilan sederhana yang dapat mengambil kutipan inspiratif untuk mencerahkan hari Anda. Kemudian kita akan melanjutkan untuk membangun keterampilan di mana Alexa dapat membantu Anda mengatur jadwal TV hari itu.

Di akhir bab ini, Anda akan dapat:

- Memahami dasar-dasar Amazon Alexa
- Memahami dan membangun model slot, maksud, dan interaksi
- Memahami slot dan maksud bawaan
- Bangun server manajemen percakapan di Heroku dan integrasikan dengan Alexa
- Integrasikan sumber data, Quote.rest, dan TVMaze API ke pengelola percakapan
- Terapkan keterampilan di Amazon Echo

Amazon Alexa

Amazon Alexa memiliki platform pengembangannya sendiri yang memungkinkan pengembang mengajarkan keterampilan baru. Keterampilan tidak lain hanyalah kemampuan percakapan yang dapat dibangun secara khusus ke dalam Alexa menggunakan kombinasi perangkat Amazon dengan milik kita. Anda sebenarnya tidak memerlukan speaker pintar Amazon Echo untuk memulai, tetapi alangkah baiknya jika Anda memilikinya. Amazon Echo hadir dalam tiga ukuran: Amazon Echo, Amazon Echo Dot, dan Amazon Echo Show. Echo Dot adalah versi paling dasar dan dihargai £49 pada saat penulisan. Ini cukup bagi pengembang untuk mengembangkan dan menguji keterampilan yang dikembangkan secara khusus:



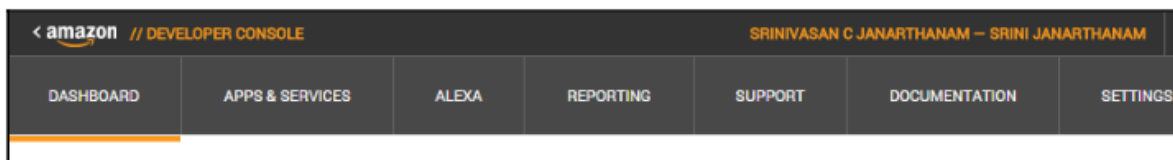
Gambar 13.1 perangkat amazon

Ekosistem Alexa juga mencakup penyimpanan keterampilan tempat keterampilan khusus dapat dipublikasikan. Keterampilan yang dipublikasikan dapat dijelajahi dan dipasang di Alexa Anda, sehingga menambah keahlian Alexa. Hal ini dapat dilakukan di situs web pendamping Alexa (alexa.amazon.com), di aplikasi Amazon Alexa di Google Play/iTunes store, atau situs Amazon (www.amazon.com/alexa-skills/). Alexa terhubung ke akun Amazon Anda dan dapat ngobrol dengan Anda tentang banyak hal: mengatur timer, info cuaca, nilai mata uang, membuat daftar tugas atau belanja, memutar musik, membaca buku, memesan dari Amazon, dan sebagainya. Lanjutkan dan coba yang berikut ini dengan Alexa, setelah Anda mengaturnya:

- ❖ Alexa, jam berapa sekarang?
- ❖ Alexa, bagaimana cuacanya?
- ❖ Alexa, apa kabar terkininya?
- ❖ Alexa, ceritakan sebuah lelucon.
- ❖ Alexa, tambahkan telur ke daftar belanjaku.

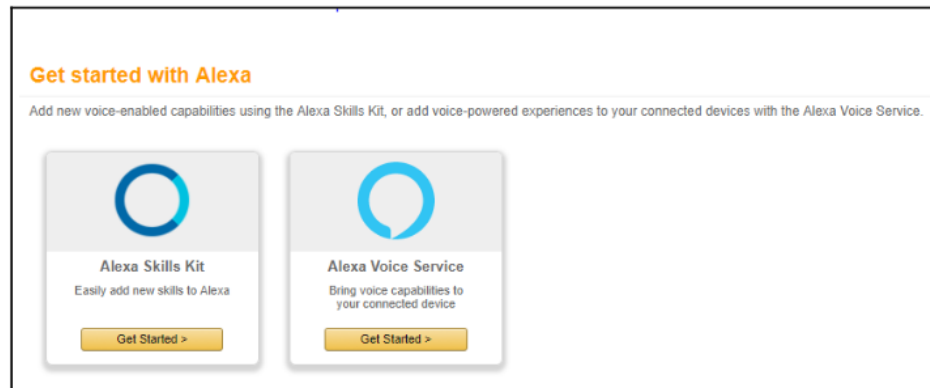
Pemanasan dengan bot kutipan

1. Untuk memulai, Anda memerlukan akun pengembang Amazon. Daftar di URL berikut: <https://developer.amazon.com/home.html>
2. Setelah Anda mendaftar, login.
3. Di dasbor, klik ALEXA:



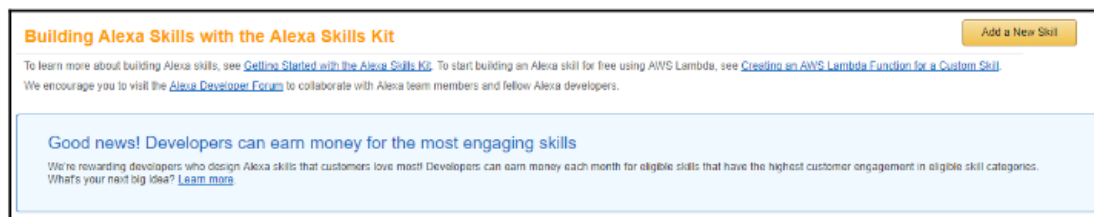
Gambar 13.2 dashbord amazon

4. Pilih Alexa Skills Kit dan klik Memulai:



Gambar 13.3 memulai alexa

5. Untuk membuat skill baru, klik Add a New Skill:



Gambar 13.4 membuat skull baru alexa

Membuat keterampilan Alexa adalah proses dua fase:

1. Mengonfigurasi keterampilan di portal pengembang.
2. Membuat modul manajemen percakapan.

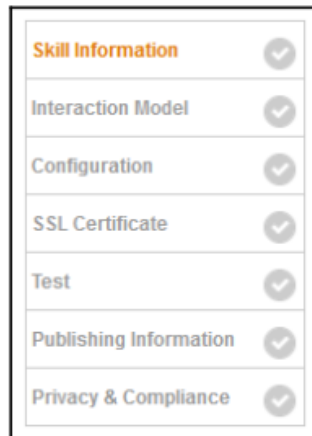
13.1 Mengonfigurasi keahlian Anda

Untuk menjelajahi perangkat ini, kita akan menggunakan contoh skenario membangun keterampilan yang memberikan kutipan inspiratif setiap hari. Jadi percakapan antara pengguna dan Alexa mungkin seperti berikut:

User :Alexa! Ask quote master to inspire me

Alexa:The Whole is greater than the sum of its parts

Di portal pengembang, kita harus melalui proses tujuh langkah berikut untuk mengonfigurasi keterampilan dan model interaksinya:



Gambar 13.5 memilih informasi kemampuan

1. Dalam formulir konfigurasi keterampilan, biarkan jenis keterampilan sebagai Model Interaksi Kustom:

Gambar 13.6 formulir konfigurasi keterampilan

2. Pilih bahasa. Ini akan menjadi bahasa keahlian Anda. Saat ini Alexa mendukung bahasa Inggris (AS, Inggris, dan India) dan Jerman.
3. Beri nama, misalnya Quote Master. Ini akan menjadi nama keterampilan seperti yang muncul di aplikasi Alexa untuk diinstal oleh pengguna Alexa lainnya di perangkat Echo mereka sendiri.
4. Beri nama doa, katakanlah, Quote Master. Nama pemanggilan adalah bagaimana skill tersebut akan dipanggil oleh Alexa. Panduan memilih nama pemanggilan tersedia di URL berikut:

<https://developer.amazon.com/docs/custom-skills/choose-the-invocation-name-for-a-custom-skill.html#invocation-name-requirements>

Anda dapat menginstal sejumlah keterampilan di Alexa Anda, namun dua keterampilan tidak boleh memiliki nama pemanggilan yang sama. Jika terjadi konflik, ini akan disorot saat memasang keterampilan baru yang nama pemanggilannya sudah digunakan.

5. Biarkan kolom lainnya ke nilai defaultnya, klik Simpan, lalu klik Berikutnya:

For successful Alexa Skills Certification, please review and follow our [Invocation Name Guidelines](#) as well as our [Certification Requirements](#).

Global Fields

These fields apply to all languages supported by the skill.

Audio Player
Does this skill use the audio player directives? Yes No
[Learn more](#)

Video App
Does this skill use the video app directives? [Learn more](#) Yes No

Render Template
Does this skill use the Render Template directives? Yes No
[Learn more](#)

Save

Gambar 13.6 menyimpan pilihan

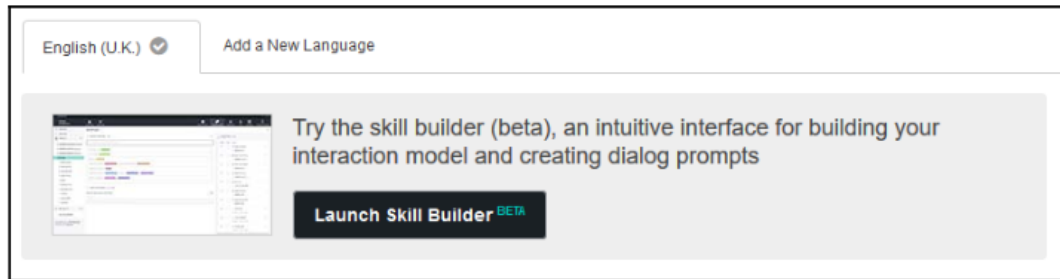
13.2 Model interaksi

Sebelum kita mulai membangun model interaksi, mari kita pahami beberapa konsep dasar. Model interaksi adalah tempat kami menentukan jenis ucapan yang Alexa perlukan untuk dapat memecahkan kode keterampilan tersebut. Hal ini ditentukan dalam maksud dan contoh. Maksud mewakili jumlah permintaan dan respons yang mungkin dimiliki pengguna saat berinteraksi dengan keterampilan. Setiap niat dapat diungkapkan dengan berbagai cara. Misalnya, pengguna mungkin ingin memberikan salam, dan dapat mengatakan salah satu hal berikut:

- ✓ **Hi**
- ✓ **Hi there**
- ✓ **Helo there**
- ✓ **Whats up**
- ✓ **Good day**

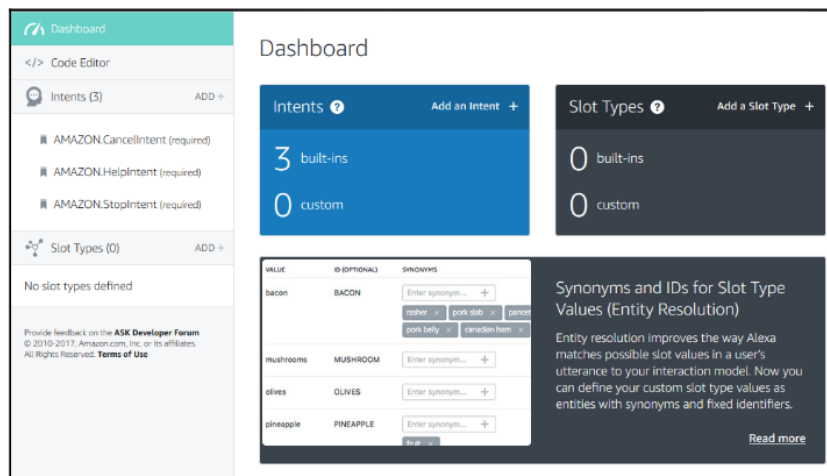
Yang disebutkan di atas adalah contohnya, sedangkan niat secara kolektif dapat disebut sebagai salam:

1. Klik Luncurkan Skill Builder:



Gambar 13.7 skill builder

2. Tindakan ini akan meluncurkan perangkat pengembangan keterampilan dan menampilkan dasbor:



Gambar 13.8 perangkat pengembangan keterampilan

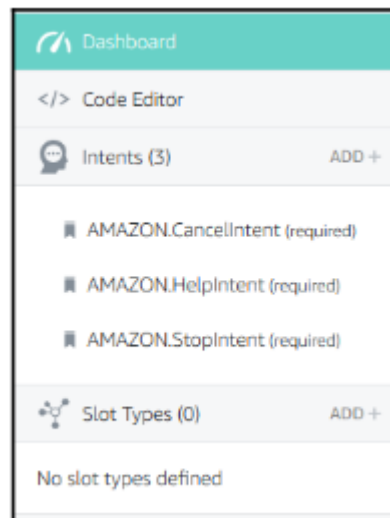
Di bawah tab Intents, Anda dapat melihat bahwa keterampilan sudah memiliki tiga niat yang ditambahkan secara default: CancelIntent, HelpIntent, dan StopIntent. Ketiganya adalah maksud bawaan dan ditambahkan secara default ke setiap keterampilan. Artinya, skill Anda sudah bisa memahami permintaan pembatalan, bantuan, dan penghentian.

Alternatif tampilan dasbor adalah melihat informasi yang sama dalam format kode dengan mengklik opsi Editor Kode:



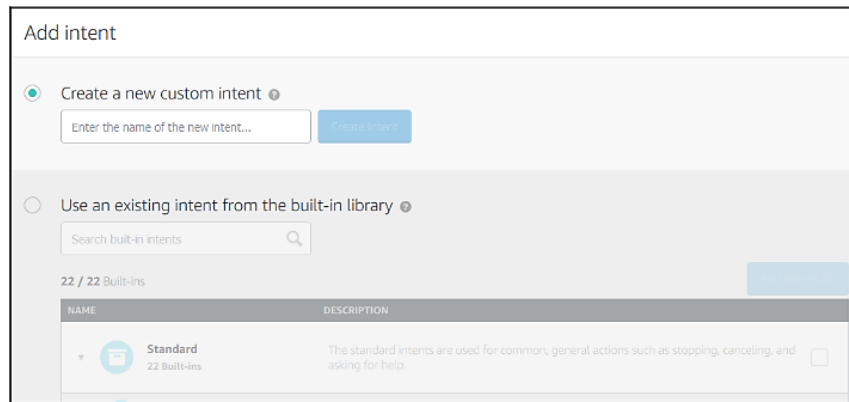
Gambar 13.9 dasbor editor

3. Untuk membuat maksud baru, **klik TAMBAHKAN +** di sepanjang tab Maksud:



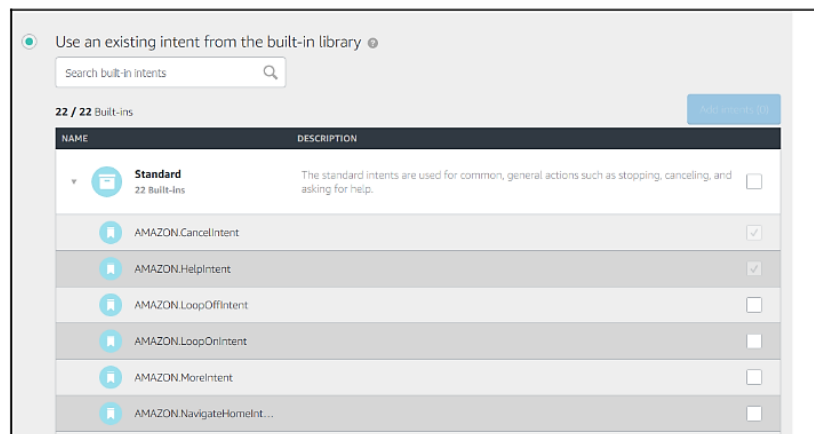
Gambar 13.10 menambahkan skill

4. Ada dua macam niat yang bisa dibuat atau ditambahkan pada skill. Ada maksud bawaan yang dapat ditambahkan. Ini adalah maksud yang telah ditentukan sebelumnya dan dapat digunakan kembali dengan mudah. Hal ini menghemat banyak waktu bagi pengembang untuk mendefinisikan ulang setiap keterampilan yang mereka bangun. Di sisi lain, kita dapat merancang maksud khusus yang disesuaikan dengan kebutuhan keahlian kita:



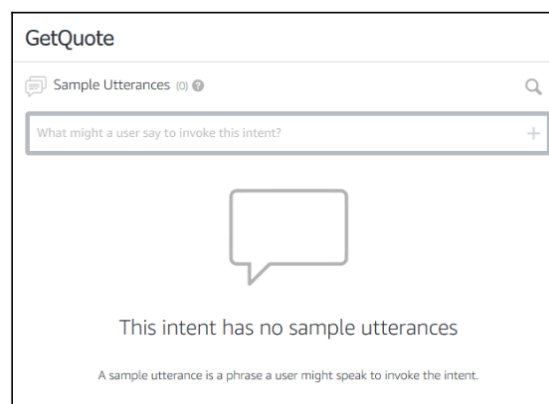
Gambar 13.11 menambahkan intent skill

5. Klik Gunakan maksud yang ada dari pustaka bawaan untuk menelusuri daftar maksud yang telah ditentukan sebelumnya:



Gambar 13.12 memilih intent dari library bawaan

6. Klik Buat maksud kustom baru. Ketikkan nama maksudnya, misalnya, GetQuote dan klik Buat Maksud:



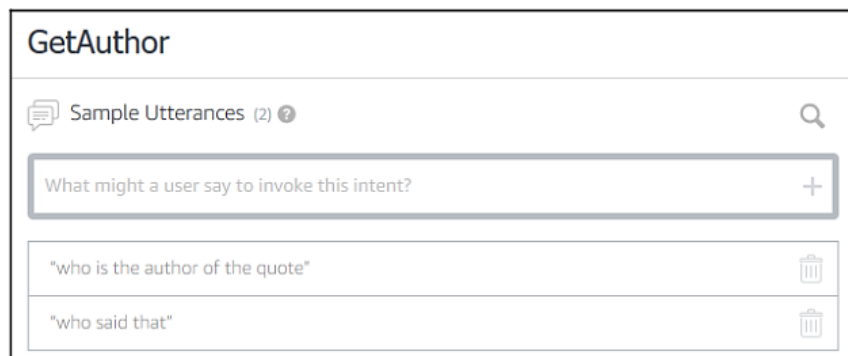
Gambar 13.13 membuat percakapan

7. Tambahkan beberapa contoh ucapan. Ini adalah contoh ucapan yang telah kita bahas sebelumnya. Ketik setiap ucapan dan klik + atau ketik return untuk menambahkannya ke daftar contoh ucapan untuk maksud tertentu:



Gambar 13.14 menambahkan beberapa dialog

8. Mari tambahkan satu maksud lagi untuk membuat ini menarik: GetAuthor. Niat ini akan meminta Alexa untuk mengambil nama penulis kutipan yang baru saja diucapkannya:



Gambar 13.15 menambahkan dialog lagi

9. Klik Simpan Model, lalu Bangun Model, dan tunggu beberapa menit untuk membuat model interaksi



Gambar 13.16 memilih pembuatan modal

10. Sekarang kita telah menentukan permintaan dalam bentuk contoh ucapan dan maksud pengguna, klik Informasi Keterampilan untuk kembali ke panel Konfigurasi Keterampilan. Di menu samping, klik Konfigurasi:

English (U.K.) ✓ Add a New Language

Global Fields

These fields apply to all languages supported by the skill.

Endpoint

Service Endpoint Type: AWS Lambda ARN (Amazon Resource Name) ⓘ HTTPS

Recommended

AWS Lambda is a server-less compute service that runs your code in response to events and automatically manages the underlying compute resources for you.

[More info about AWS Lambda](#)
[How to integrate AWS Lambda with Alexa](#)

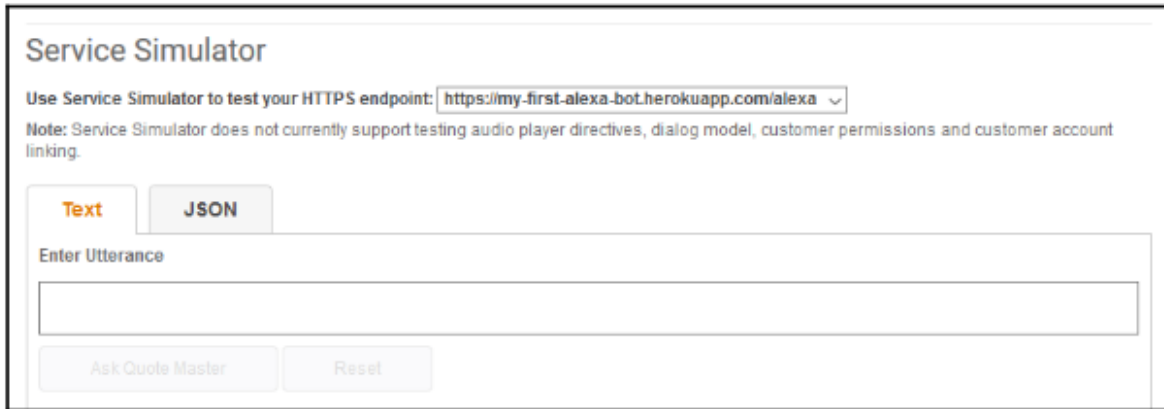
Default

Provide geographical region endpoints? (Optional) ⓘ Yes No

Gambar 13.17 konfigurasi keterampilan

11. Hal ini membawa kita ke bagian pengembangan berikutnya, di mana kita perlu membangun sebuah modul untuk mengelola percakapan. Pada langkah ini, kami mengonfigurasi titik akhir tempat pengelola percakapan dapat ditemukan. Kami memiliki dua opsi: AWS dan HTTPS. Dalam kasus AWS, Anda perlu membuat layanan AWS Lambda untuk menghosting titik akhir server. Dalam hal HTTPS, kita dapat menghosting sebagai server web di platform cloud apa pun yang kita inginkan. Kami akan mengambil opsi HTTPS di sini.
12. Di kolom teks Default berikut, masukkan URL aplikasi web untuk pengelola percakapan keahlian Anda. Karena kami belum membuatnya, masukkan URL indikatif untuk saat ini (misalnya, <https://my-first-alexa-bot.herokuapp.com/alexa>). Abaikan parameter lainnya untuk saat ini dan klik Berikutnya.
13. Pada panel sertifikat SSL, klik Berikutnya dan lanjutkan ke Tes.

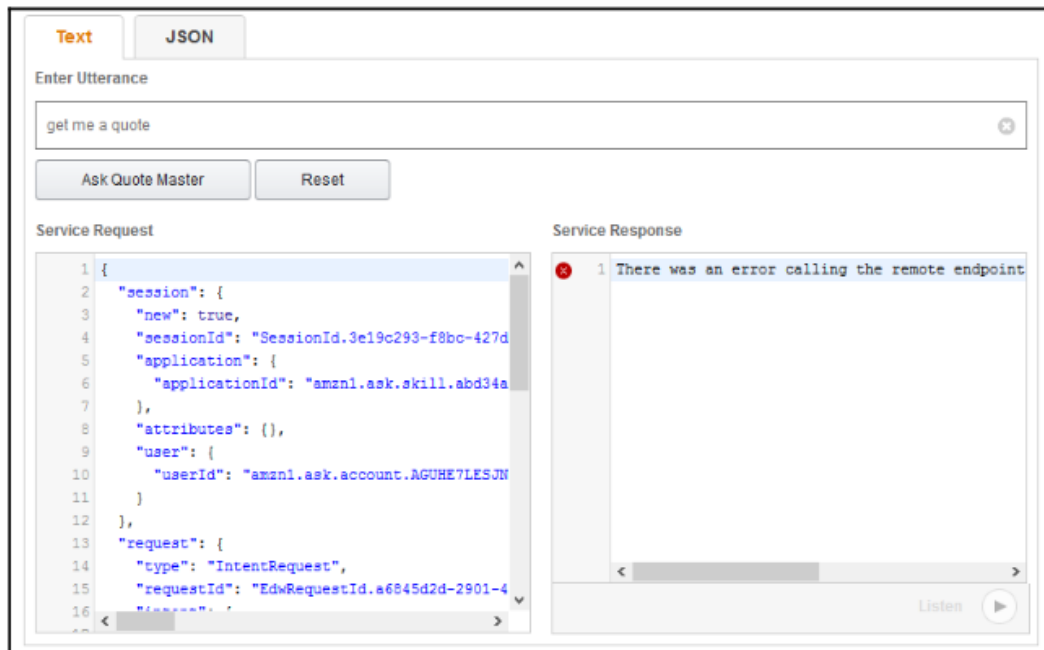
14. Pada panel Tes, kita dapat menguji apakah Alexa dapat mengenali ucapan yang kita ingin dia kenali dan mengklasifikasikan maksudnya secara akurat. Gulir ke bawah ke tab Simulator Layanan:



Gambar 13.18 menguji pada tab simulator

Anda akan melihat titik akhir HTTPS yang Anda atur di Titik Akhir Server ditentukan.

15. Ketik ucapan di tab Teks dan **klik tombol Tanya <skillname>** di bawahnya:



Gambar 8.19 mencoba bertanya pada simulasi

16. Anda akan melihat bahwa Alexa telah menguraikan ucapan dan mengidentifikasi maksudnya. Ini disajikan dalam bentuk objek JSON di sebelah kiri (Permintaan Layanan):

```

...
"request": {
  "type": "IntentRequest",
  "requestId": "EdwRequestId.a6845d2d-2901-4e8c-b260-
    3a3a89bd5068",
  "intent": {
    "name": "GetQuote",
    "slots": {}
  },
  "locale": "en-GB",
  "timestamp": "2017-10-11T07:45:24Z"
},
...

```

Di sebelah kanan (Service Response) akan ada pesan error, karena endpoint yang kita tentukan belum ada. Titik akhir ini perlu memproses permintaan JSON dan merespons dengan ucapan yang dapat diucapkan kembali oleh Alexa kepada pengguna.

13.3 MANAJER PERCAKAPAN - KUTIPAN

Manajer percakapan adalah aplikasi web yang dapat menerima hasil maksud Alexa dan merespons dengan respons bot. Mari kita membuat aplikasi web Node.js yang dapat menerima maksud yang telah kita tentukan dalam keahlian kita dan merespons dengan tepat. Untuk membangun modul, lakukan langkah-langkah berikut:

1. Buat file package.json menggunakan npm init:
 - >npm init**
2. Instal express, request, body-parser, dan hashmap:
 - **Npm install request - - save**
 - **Npm install express - - save**
 - **Npm install body-parser - -save**
 - **Npm install hasmap - -save**
3. Buat file index.js:

```

/*
 * Alexa
 */

```

```

const request = require('request')
const express = require('express')
const bodyParser = require('body-parser')

const app = express()

const Hashmap = require('hashmap');

var userContexts = new Hashmap();

app.set('port', (process.env.PORT || 5000))

// Process application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({extended: false}))

// Process application/json
app.use(bodyParser.json())

// Index route
app.get('/', function (req, res) {
  res.send('Hello world, I am Quote Master.')
})

// Spin up the server
app.listen(app.get('port'), function() {
  console.log('running on port', app.get('port'))
})

```

4. Tambahkan pengendali rute POST untuk menangani permintaan dari Alexa. Kode berikut memiliki placeholder yang akan kita isi dengan kode yang sesuai pada langkah selanjutnya:

```

app.post('/alexa', function (req, res) {
  console.log('Request from Alexa:');
  console.log(req.body);
  //1. add context code here

  //2. get userIntent from request

  //3. set default return json object
  //4. dialogue management code to decide what to
  //say in response to user intents.
})

```

5. Mari buat variabel konteks lokal untuk menyimpan semua informasi yang kita perlukan. Variabel ini dapat digunakan untuk mengumpulkan semua data yang perlu dibagikan di seluruh sesi. Untuk menyimpan informasi secara lokal selama belokan, kami menggunakan

variabel `sessionContext`. Ini dimuat dari variabel global `userContexts`, yang merupakan peta hash yang menampung semua konteks sesi. Alexa menyajikan maksud pengguna setiap kali dengan ID sesi yang sama di seluruh sesi. Jadi, kita dapat menggunakan ID sesi untuk menyimpan dan mengambil konteksnya dalam variabel global `userContexts`:

```

var sessionContext = {
    'lastUserIntent': 'null',
    'lastQuote': null
};
var sessionId = req.body.session.sessionId;
if (!userContexts.get(sessionId)){
    userContexts.set(sessionId, sessionContext);
} else {
    sessionContext = userContexts.get(sessionId);
}

```

6. Kami mengidentifikasi maksud pengguna dari permintaan (`req.body.request.intent.name`). Namun, ketika ada batas waktu dan Alexa mengakhiri sesi, ia mengembalikan `SessionEndedRequest`, yang perlu ditangani dengan tepat. Kami menanganinya serupa dengan `Amazon.StopIntent` berikut:

```

var userIntent = '';
if (req.body.request.type === 'SessionEndedRequest'){
    userIntent = 'AMAZON.StopIntent';
} else {
    userIntent = req.body.request.intent.name;
}
console.log('UserIntent:' + userIntent);

```

7. Sekarang mari siapkan JSON default untuk dikembalikan. Berikut ini adalah format pengembalian JSON. Kunci `ssml` berisi respons Speech Synthesis Markup Language (SSML) dengan teks tersemat. Teks inilah yang disintesis Alexa sebagai respons terhadap permintaan pengguna:

```

var rjson = {
    "version": "1.0",
    "response": {
        "shouldEndSession": false,
        "outputSpeech": {
            "type": "SSML",
            "ssml": "<speak> Whole is larger than
                    the sum of its parts!</speak>"
        }
    }
};

```

8. Respons SSML default perlu ditimpa dengan respons yang sesuai dengan permintaan pengguna. Jadi jika pengguna meminta kutipan, kita perlu memberi mereka kutipan baru yang menginspirasi, tetapi jika mereka menanyakan penulisnya, kita perlu merespons dengan nama penulisnya. Logika manajemen dialog inilah yang akan kita kodekan selanjutnya. Jika maksud pengguna adalah mendapatkan penawaran baru, kami membuat permintaan GET untuk mengambil penawaran baru dari `quotes.rest`. Perhatikan bagaimana

kami menyimpan kutipan dalam konteks sehingga jika pengguna menanyakan penulis kutipan, Alexa mengetahui jawabannya dari konteks:

```

if (userIntent === 'AMAZON.StopIntent'){
    rjson.response.shouldEndSession = true;
    rjson.response.outputSpeech.ssml = '<speaks> Ok.
    Have a good day! </speaks>';
    console.log(rjson);
    res.json(rjson);
}
else if (userIntent === 'GetQuote'){
    request({
        url: 'http://quotes.rest/qod.json?category=inspire',
        method: 'GET'
    },
    function (error, response, body) {
        //response is from the bot
        if (!error && response.statusCode == 200) {
            var q = JSON.parse(body).contents.quotes[0];
            var quote = {'quote': q.quote,
                'author': q.author}
            sessionContext.lastQuote = quote;
            userContexts.set(sessionId, sessionContext);
            rjson.response.outputSpeech.ssml = '<speaks>' +
            quote.quote + '</speaks>';
            console.log(rjson);
            res.json(rjson);

        } else {
            console.log('Error: ' + error)
            console.log('Statuscode: ' + response.statusCode)
        }
    });
}
else if (userIntent === 'GetAuthor'){
    rjson.response.outputSpeech.ssml =
        '<speaks>' + sessionContext.lastQuote.author
        + '</speaks>';
    console.log(rjson);
    res.json(rjson);
}
else {
    rjson.response.shouldEndSession = true;
    rjson.response.outputSpeech.ssml = '<speaks>
    Sorry. I did not get you! </speaks>';
    console.log(rjson);
    res.json(rjson);
}
}

```

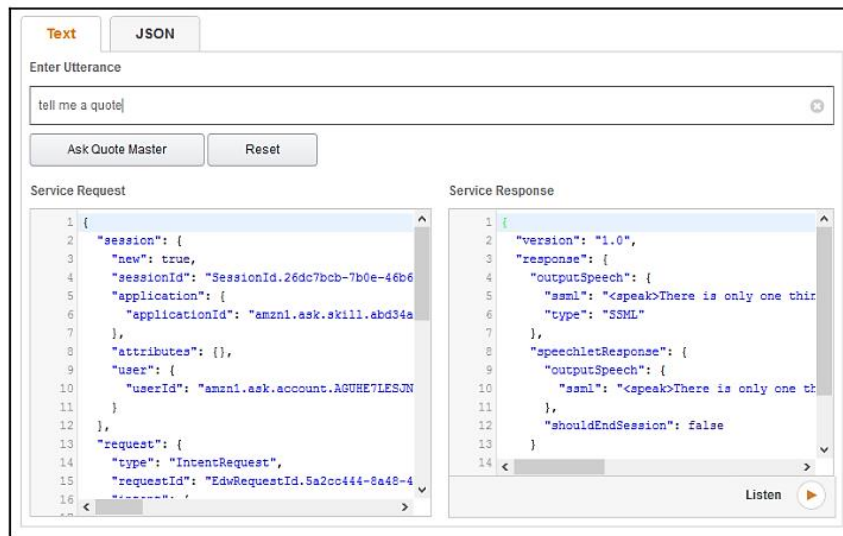
9. Sekarang kita memiliki aplikasi web untuk mengelola percakapan, mari kita dorong di cloud ke titik akhir server yang dikonfigurasi di panel Konfigurasi Keterampilan:

- **Git init**
- **Git add**
- **Git commit -m alexa-quote-bot**
- **Heroku create my-first-alex-bot**

➤ **Git push Heroku master**

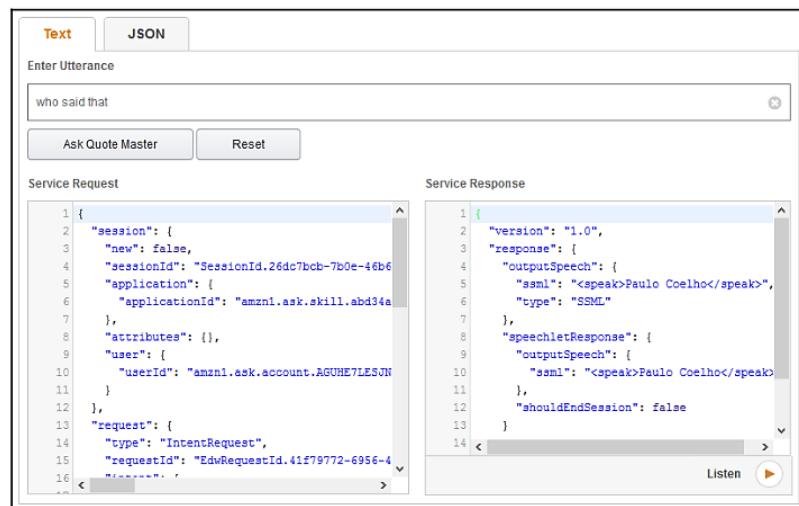
Setelah menjalankan instruksi sebelumnya di konsol Anda, Anda akan menjalankan pengelola percakapan <https://my-first-alexa-bot.herokuapp.com/alexa>. Di sinilah Alexa akan mengirimkan permintaan POST. Responsnya akan diuraikan dan ucapan SSML akan disintesis kepada pengguna.

10. Sekarang kembali ke panel tes, ketikkan contoh ucapan (misalnya, beri tahu saya sebuah kutipan), dan klik Ask Quote Master:



Gambar 13.20 memberikan contoh ucapan dialog

11. Lanjutkan pembicaraan. Ketik siapa yang mengatakan itu dan klik Ask Quote Master:



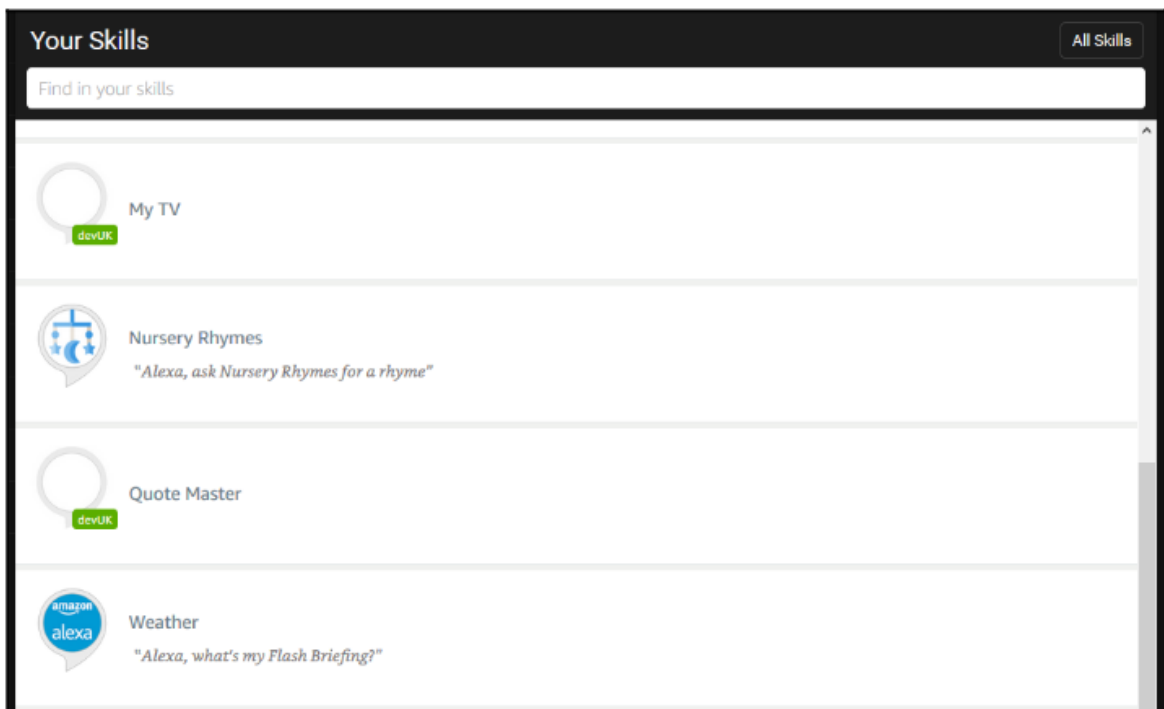
Gambar 13.21 melanjutkan percakapan

Permintaan ini akan diidentifikasi sebagai maksud GetAuthor dan ditanggapi dengan nama penulis kutipan.

- Perhatikan bahwa sessionId untuk kedua permintaan sebelumnya adalah sama. Sekarang klik Reset untuk mengatur ulang percakapan. Setiap permintaan baru akan memiliki sessionId yang berbeda.

Uji pada Alexa

Selamat! Keterampilan Alexa pertama Anda sudah siap. Anda dapat mengujinya di Alexa Anda jika Anda memilikinya di rumah. Semua keterampilan yang Anda kembangkan akan ditambahkan ke Alexa Anda secara default. Anda dapat melihatnya di dashboard Alexa Anda (alexa.amazon.com), klik tab Skills di menu, dan Your Skills di kanan atas:



Gambar 13.22 dialog Alexa

Anda akan melihat bahwa Alexa menunggu setelah membacakan kutipan untuk Anda. Hal ini untuk memungkinkan pengguna membuat permintaan tindak lanjut. Ini diaktifkan dengan menyetel `ShouldEndSession` ke `false` secara default. Ini disetel ke benar hanya ketika pengguna mengatakan berhenti atau waktu habis tanpa mengatakan apa pun saat Alexa menunggu respons.

Panduan TV saya

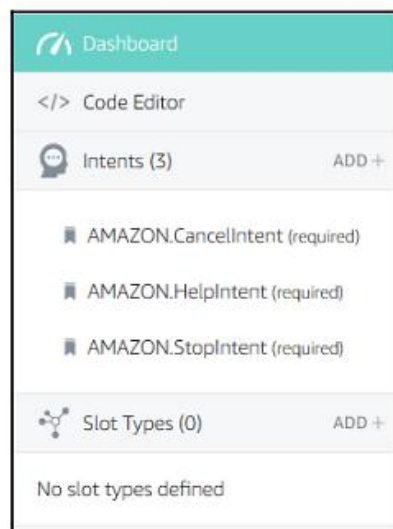
Mari kita beralih ke proyek yang lebih menantang: membangun keterampilan Alexa untuk mendapatkan info tentang acara TV. Dalam tugas ini, kita akan membangun keterampilan TV yang memungkinkan Alexa menjawab pertanyaan tentang acara TV. Misalnya, Alexa akan mampu menjawab pertanyaan-pertanyaan berikut.

- Apa selanjutnya di BBC One?
- Apa yang ada di ITV sekarang?
- Tentang apa acaranya?
- Ada apa di ITV jam 2 siang?

13.4 MEMBANGUN MODEL INTERAKSI

Selain konsep yang kita jelajahi di keterampilan QuoteMaster, dalam tugas ini kita akan mengeksplorasi elemen percakapan yang lebih kompleks: slot. Mari kita ikuti proses tujuh langkah dari sebelumnya untuk menciptakan keterampilan baru yang disebut TV Saya:

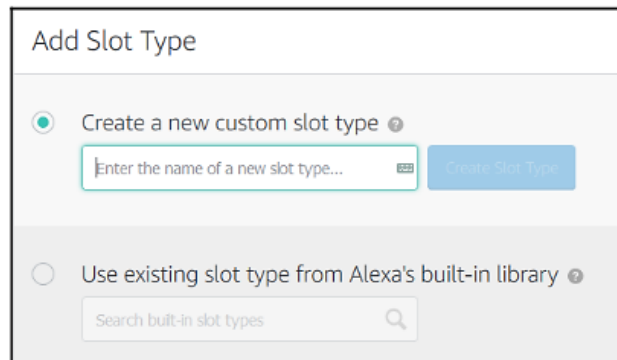
1. Isi Informasi Keterampilan dan lanjutkan ke halaman Model Interaksi.
2. Klik Luncurkan Skill Builder dan buka dasbor. Seperti sebelumnya, Anda akan melihat bahwa ada tiga maksud dalam model secara default.
3. Seperti yang mungkin Anda ketahui, ada elemen dalam pertanyaan sebelumnya yang dapat disebutkan dan diklasifikasikan sebagai tipe entitas. Misalnya, BBC One, BBC Two, dan ITV dapat diklasifikasikan sebagai jaringan. Di Alexa, pengetahuan ini dapat dimasukkan ke dalam sistem sebagai jenis slot. Untuk membuat jenis slot, berikut yang perlu Anda lakukan: Di bawah Intents pada menu sebelah kiri, Anda akan menemukan Jenis Slot. Klik TAMBAHKAN +:



Img: alexa_slot_types

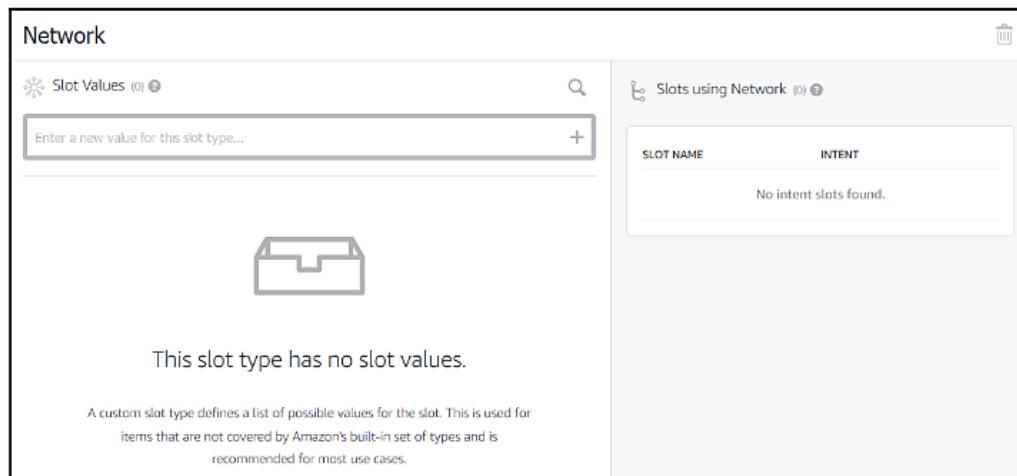
Gambar 13.23 menambahkan dialog percakapan

4. Terkait dengan maksud, ada dua jenis slot—slot bawaan dan slot khusus:



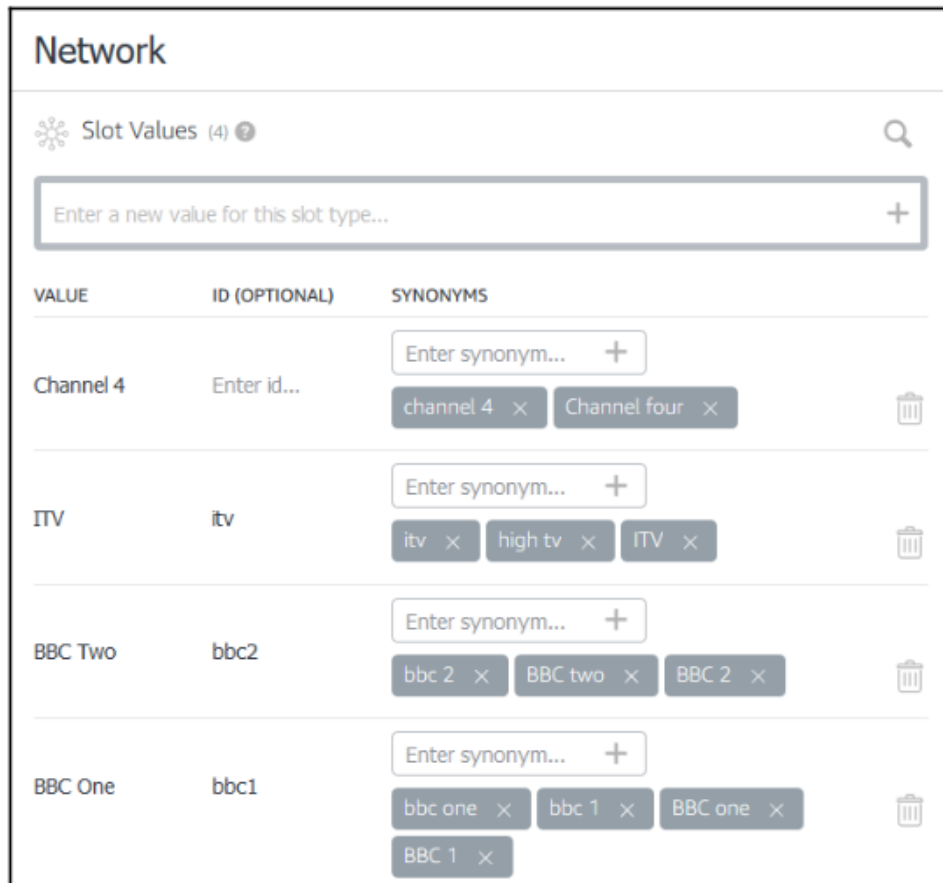
Gambar 13.24 menambahkan jenis slot bawaan dan khusus

Kami akan membuat slot khusus yang disebut Jaringan. Klik Buat Jenis Slot:



Gambar 13.25 membuat slot khusus

5. Di halaman ini, kita harus menentukan semua jaringan yang kita ingin Alexa identifikasi selama percakapan dengan pengguna. Mari kita buat daftar kecil: Channel 4, BBC One, BBC Two, dan ITV. Kita dapat menambahkan lebih banyak lagi nanti.
6. Mari tambahkan yang pertama: BBC One. Klik + di akhir bidang teks. Tambahkan semua variasi nilai BBC One, BBC 1, BBC One, dan seterusnya sebagai sinonimnya.
7. Tambahkan semua jaringan lainnya dengan cara yang sama:



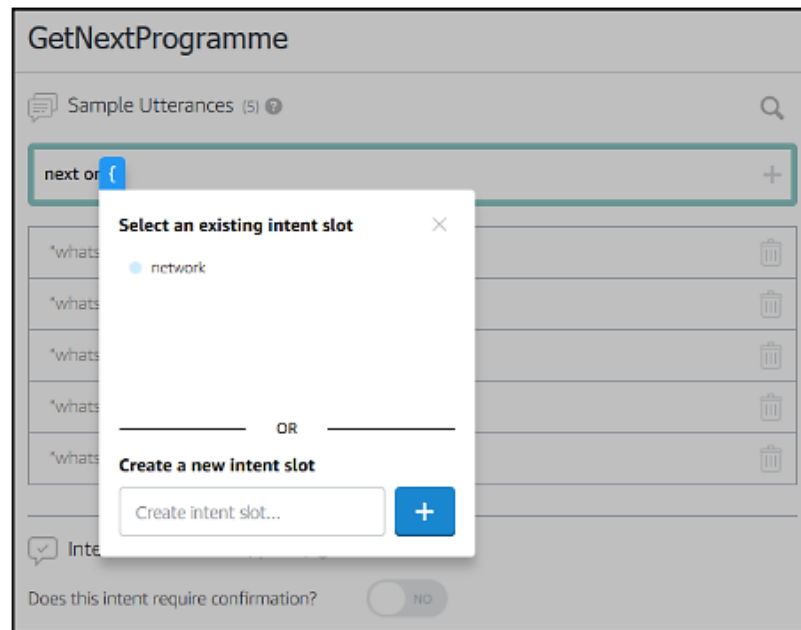
Img: alexa_slot_network_filled

Gambar 13.26 tampilan slot jaringan tivi alexa

8. Sekarang mari kita buat maksud yang menggunakan tipe slot Jaringan. Mari kita mulai dengan `GetNextProgramme`. Maksud ini sesuai dengan semua ucapan di mana pengguna meminta program berikutnya dalam jaringan tertentu. Contoh ucapannya adalah, apa program selanjutnya di BBC One?. Klik Maksud + dan tambahkan contoh ucapan. Namun alih-alih menyebutkan jaringan dalam ucapannya, kita perlu menggunakan tipe slot (menggunakan notasi `{}`), seperti yang ditunjukkan di sini:

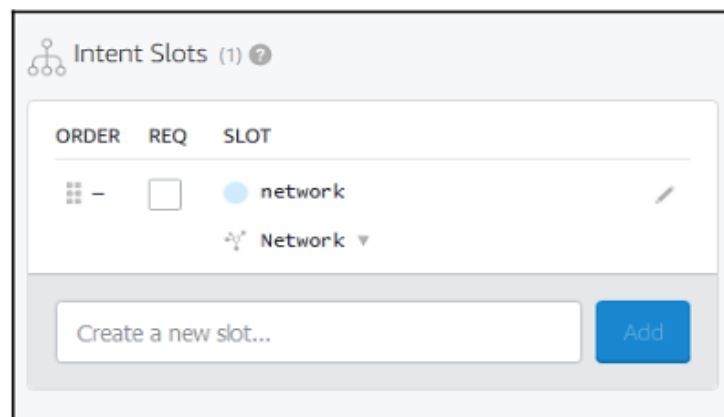
```
whats on {network} now?
what the next show on {network}
whats next on {network}?
```

Saat Anda mulai memasukkan jenis slot dalam ucapan, daftar jenis slot akan muncul dan dapat dipilih, seperti yang ditunjukkan di sini:



Gambar 13.27 Menambahkan program dialog

Meskipun jaringan slot diperlukan untuk melaksanakan tugas tersebut, biarkan jaringan tersebut tidak dicentang sebagai slot yang diperlukan (kotak centang di bawah PERMINTAAN) untuk saat ini:



Gambar 13.28 kontak dialog yang diperlukan

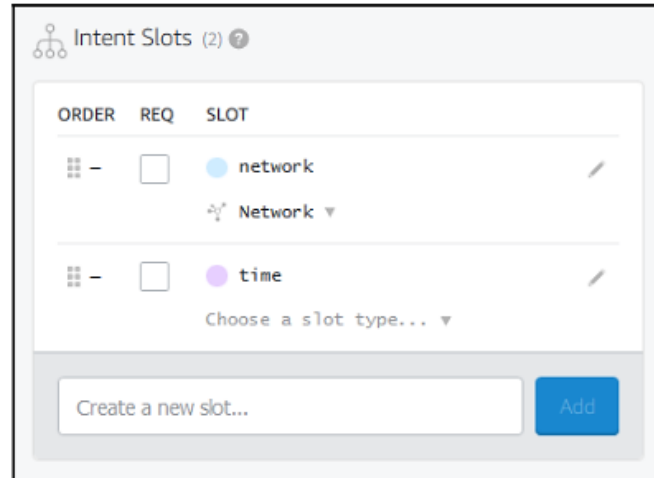
- Mari tambahkan maksud lain: `GetProgrammeAtTime`. Yang ini untuk ucapan di mana pengguna meminta acara di jaringan tertentu pada waktu tertentu. Contoh ucapannya adalah, acara apa yang ditayangkan di BBC One pada jam 2 siang.

Buat maksud baru dan tambahkan yang berikut ini sebagai contoh ucapan:

What show is on {network} at {time}

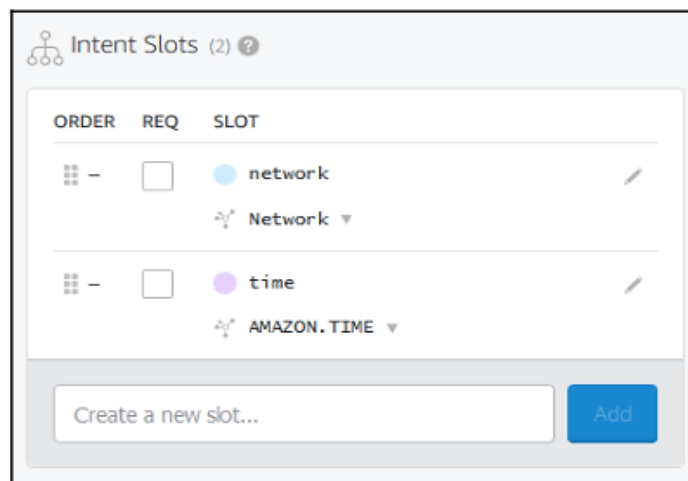
Whats on {network} at {time}

Karena waktu bukanlah slot yang ditentukan, kita perlu menentukannya nanti. Semua slot yang terhubung ke maksud akan dicantumkan di sebelah kanan, seperti yang ditunjukkan di sini:



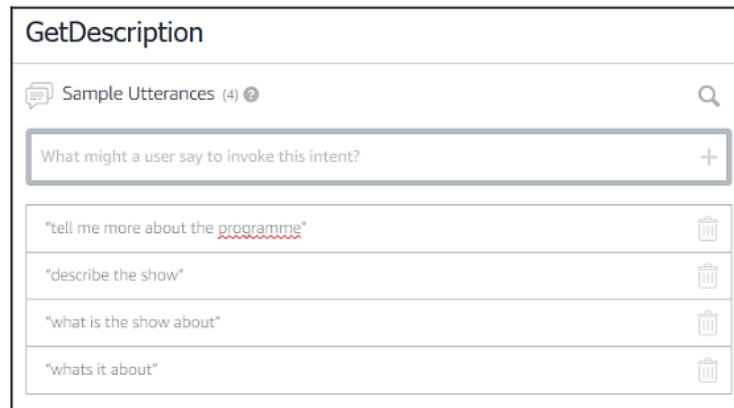
Gambar 13.29 menambahkan dialog waktu

10. Klik Pilih jenis slot... dan pilih AMAZON.TIME. Ini salah satu jenis slot yang telah ditentukan sebelumnya. Klik kotak centang untuk membuat slot diperlukan dan memberikan prompt dan respons, seperti yang kami lakukan untuk slot jaringan:



Gambar 13.30 memilih waktu amazon

11. Tambahkan ucapan lain dan simpan modelnya.
12. Tambahkan maksud akhir: GetDescription. Maksudnya adalah untuk ucapan yang membuat pengguna ingin mengetahui lebih banyak tentang suatu acara. Misalnya, pengguna dapat mengatakan, tentang apa acara tersebut sebagai pertanyaan lanjutan dari dua maksud sebelumnya:



Gambar 13.31 Mengisi deskripsi

Yang ini tidak memiliki slot apa pun karena kami akan mencoba menyimpan informasi yang diperlukan dalam konteks percakapan.

13. Tambahkan ucapan lain untuk maksudnya dan simpan modelnya:

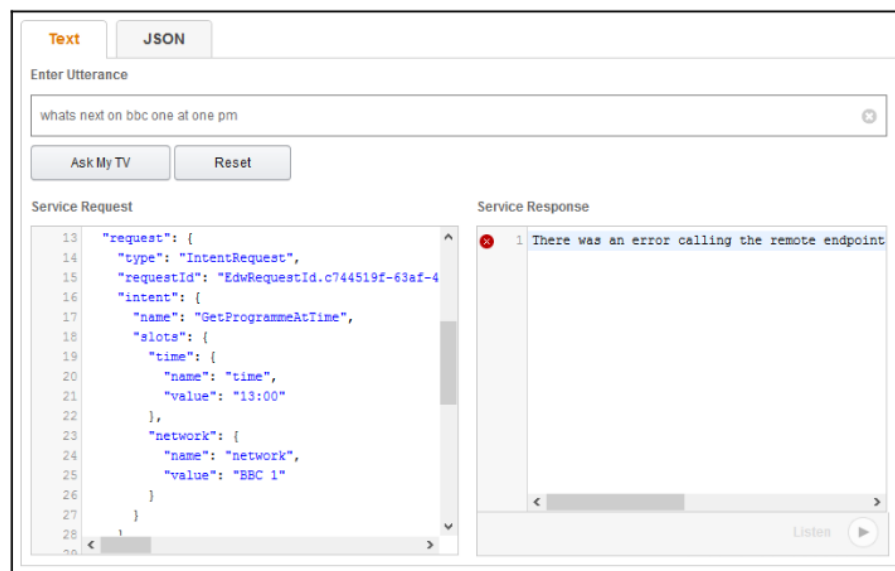
Tell me more about the show

Describe the show

What is the show about

Whats it about

14. Mari kita uji model seperti yang kita lakukan dengan skill QuoteMaster. Di panel tes, ketikkan beberapa ucapan dan lihat bagaimana Alexa menganalisisnya:



Gambar 13.32 menguji skill QuoteMaster

Karena Anda tidak memiliki server yang disetel untuk merespons, responsnya akan berupa kesalahan.

Menyiapkan server keterampilan TV

Sekarang kita memiliki model interaksi, kita memerlukan pengelola percakapan yang dapat merespons maksud pengguna. Sekarang mari kita siapkan server untuk merespons pengguna:

1. Pada halaman Konfigurasi Keterampilan, klik konfigurasi dan pilih opsi HTTPS.
2. Buat proyek Node.js menggunakan npm dan tambahkan perpustakaan default, seperti yang kita lakukan untuk server QuoteMaster yang dibahas sebelumnya.
3. Buat file index.js dan impor pustaka request, express, body-parser, dan hashmap.
4. Buat server web menggunakan perpustakaan Express, seperti yang Anda lakukan sebelumnya:

```
const request = require('request')
const express = require('express')
const bodyParser = require('body-parser')

const Hashmap = require('hashmap');

var usercontexts = new Hashmap();

const app = express()

app.set('port', (process.env.PORT || 5000))

// Process application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({extended: false}))

// Process application/json
app.use(bodyParser.json())

app.use(express.static('public'))

// Index route
app.get('/', function (req, res) {
  res.send('Hello world, I am the TV Guide.')
})
```

5. Kita memerlukan perpustakaan lain untuk memproses tanggal dan waktu, yang akan kita sebut momen. Instal menggunakan npm install dan tambahkan kode berikut ke index.js. Ini digunakan untuk membuat tanggal hari ini untuk mengambil jadwal TV pada hari itu:

```
const moment = require('moment')

var todaysSchedule = {};
const now = new Date();
var date = now.getDate();
if (date < 10) { date = '0' + date; }
var month = now.getMonth() + 1;
if (month < 10) { month = '0' + month; }
const todaysDate = now.getFullYear() + '-' + month + '-' + date;

loadTodaysSchedule(todaysDate);
```

6. Buat fungsi `loadTodaysSchedule()` untuk mendapatkan jadwal TV hari itu. Ini dapat disimpan secara lokal dan ditanyakan untuk menjawab pertanyaan pengguna. Perlu diperhatikan bahwa jadwal tidak akan berubah setiap hari secara otomatis:

```
function loadTodaysSchedule(todaysDate){
  var url = 'http://api.tvmaze.com/schedule?country=GB&date='
    + todaysDate;
  console.log('URL:' + url);
  request({
    url: url,
    method: 'GET'
  },
  function (error, response, body) {
    //response is from the bot
    if (!error && response.statusCode == 200) {
      // Print out the response body
      todaysSchedule = JSON.parse(body);
      console.log('Todays Schedule: ' +
        todaysSchedule.length);
      console.log(todaysSchedule);
    } else {
      console.log('Error: ' + error)
      console.log('Statuscode: ' + response.statusCode)
    }
  });
}
```

7. Berikut adalah template untuk titik akhir respons sebenarnya:

```
app.post('/alexa', function (req, res) {
  console.log('Request from Alexa:');
  console.log(req.body);
  //1. add context code here

  //2. get userIntent from request

  //3. set default return json object
  //4. dialogue management code to
  //decide what to say in response to user intents.
})

// Spin up the server
app.listen(app.get('port'), function() {
  console.log('running on port', app.get('port'))
})
```

8. Mari simpan konteksnya dan memuatnya seperti yang kita lakukan pada skill QuoteMaster. Kami akan menyimpan informasi acara yang diminta pengguna sehingga kami dapat menjelaskannya pada giliran berikutnya:

```
var sessionContext = {
  'show': null
};
var sessionId = req.body.session.sessionId;
if (!usercontexts.get(sessionId)){
  usercontexts.set(sessionId, sessionContext);
} else {
  sessionContext = usercontexts.get(sessionId);
}
```

9. Mari kita dapatkan maksud pengguna dari permintaan tersebut dan siapkan respons JSON default:

```

var userIntent = '';
if (req.body.request.type === 'SessionEndedRequest'){
  userIntent = 'AMAZON.StopIntent';
} else {
  userIntent = req.body.request.intent.name;
}
console.log('UserIntent:' + userIntent);
var rjson = {
  "version": "1.0",
  "response": {
    "shouldEndSession": false,
    "outputSpeech": {
      "type": "SSML",
      "ssml": "<speack>Hello, I am your TV Guide!</speack>"
    }
  }
};

```

10. Dan berikut adalah kode untuk merespon maksud pengguna. Respons pengguna JSON disetel ulang dengan informasi yang diambil dari jadwal hari itu:

```

if (userIntent === 'AMAZON.StopIntent'){
  rjson.response.shouldEndSession = true;
  rjson.response.outputSpeech.ssml =
    '<speack> Ok. Have a good day! </speack>';
  console.log(rjson);
  res.json(rjson);
}
else {
  if (userIntent === 'GetNextProgramme'){
    var network =
      req.body.request.intent.slots.network.value;
    var e = getNextProgrammeInNetwork(network);
    sessionContext.show = e;
    usercontexts.set(sessionId, sessionContext);
    rjson.response.shouldEndSession = false;
    rjson.response.outputSpeech.ssml =
      '<speack> The next show on ' + network + ' is ' +
      e.show.name + ' at ' + e.airtime + '! </speack>';
    console.log(rjson);
    res.json(rjson);
  }
  else if (userIntent === 'GetProgrammeAtTime'){
    var network =
      req.body.request.intent.slots.network.value;
    var time = req.body.request.intent.slots.time.value;
    var e = getProgrammeAtTime(network, time);
    sessionContext.show = e;
    usercontexts.set(sessionId, sessionContext);
    rjson.response.shouldEndSession = false;
    rjson.response.outputSpeech.ssml = '<speack> On '
    + network +
      ' at ' + e.airtime +
      ' is ' + e.show.name +
      '! </speack>';
    console.log(rjson);
    res.json(rjson);
  }
  else if (userIntent === 'GetDescription'){
    var e = sessionContext.show;
    var desc = getDescription(e);
    rjson.response.shouldEndSession = true;
    rjson.response.outputSpeech.ssml = '<speack> ' + desc +
      ' </speack>';
    console.log(rjson);
  }
}

```

```

        res.json(rjson);
    }
    else if (userIntent === 'Thank'){
        rjson.response.shouldEndSession = true;

        rjson.response.outputSpeech.ssm1 = '<speack>
        You are most welcome. Goodbye! </speack>';
        console.log(rjson);
        res.json(rjson);
    }
    else {
        rjson.response.shouldEndSession = true;
        //rjson.response.outputSpeech.ssm1 = '<speack> Ok.
        //Have a good day! </speack>';
        console.log(rjson);
        res.json(rjson);
    }
}
}

```

11. Terakhir, bagian kode yang mengambil informasi yang diperlukan dari jadwal:

```

function getNextProgrammeInNetwork(networkName){
    console.log('Getting next programme in : ' + networkName);
    for (var i=0; i < todaysSchedule.length; i++){
        var e = todaysSchedule[i];
        var showtime = moment(e.airdate + ' ' + e.airtime);
        moment().format();

        var a = moment();
        var b = moment(showtime);
        if (a.diff(b, 'minutes') < 0){
            if (e.show.network.name === networkName){
                console.log(e.show.name + ' on ' +
                    e.show.network.name + ' at ' + e.airtime);
                return e;
            }
        }
    }
}

function getProgrammeAtTime(networkName, time){
    console.log('Getting next programme in : ' + networkName + '
        at ' + time);
    for (var i=0; i < todaysSchedule.length; i++){
        var e = todaysSchedule[i];
        if (e.show.network.name === networkName){
            var showtime = moment(e.airdate + ' ' + e.airtime);

```

```

var requestedtime = moment(e.airdate + ' ' + time);
moment().format();

var a = moment(requestedtime);
var b = moment(showtime);
if (a.diff(b, 'minutes') > -30 && a.diff(b, 'minutes')
    < 30 ){
    console.log(e.show.name + ' on ' +
        e.show.network.name + ' at ' + e.airtime);
    return e;
}
}
}

function getDescription(episode){
    console.log('Getting episode info:' + episode.show.name);
    console.log(episode.show.summary);
    return episode.show.summary;
}

```

12. Sekarang, mari pindahkan aplikasi web ini ke Cloud:

```

> git init
> git add .
> git commit -m alexa-tv-bot
> heroku create my-alexa-tv-bot
> git push heroku master

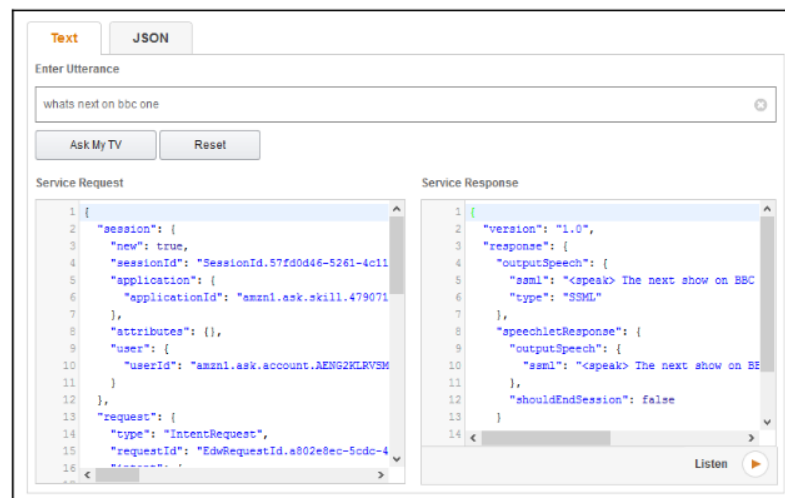
```

Setelah mendorong aplikasi web ke cloud menggunakan instruksi sebelumnya, kita akan menjalankan pengelola percakapan <https://my-alexa-tv-bot.herokuapp.com/alexa>. Di sinilah Alexa akan mengirimkan permintaan POST. Responsnya akan diuraikan dan ucapan SSML akan disintesiskan kepada pengguna.

Menguji keterampilan TV

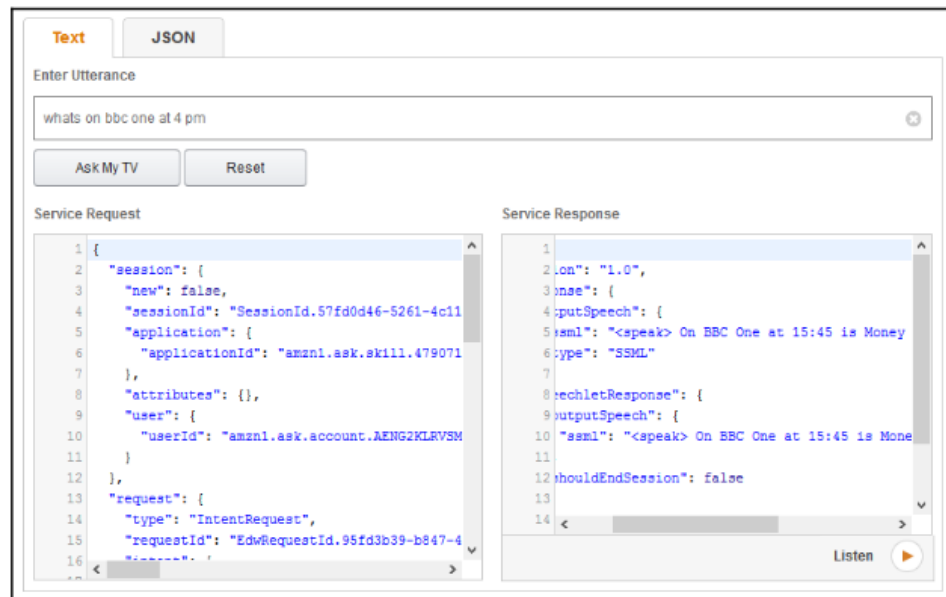
Sekarang mari kita kembali ke panel pengujian dan menguji beberapa contoh ucapan untuk melihat cara kerjanya:

1. Ketik apa selanjutnya di bbc one dan klik Tanya TV Saya:



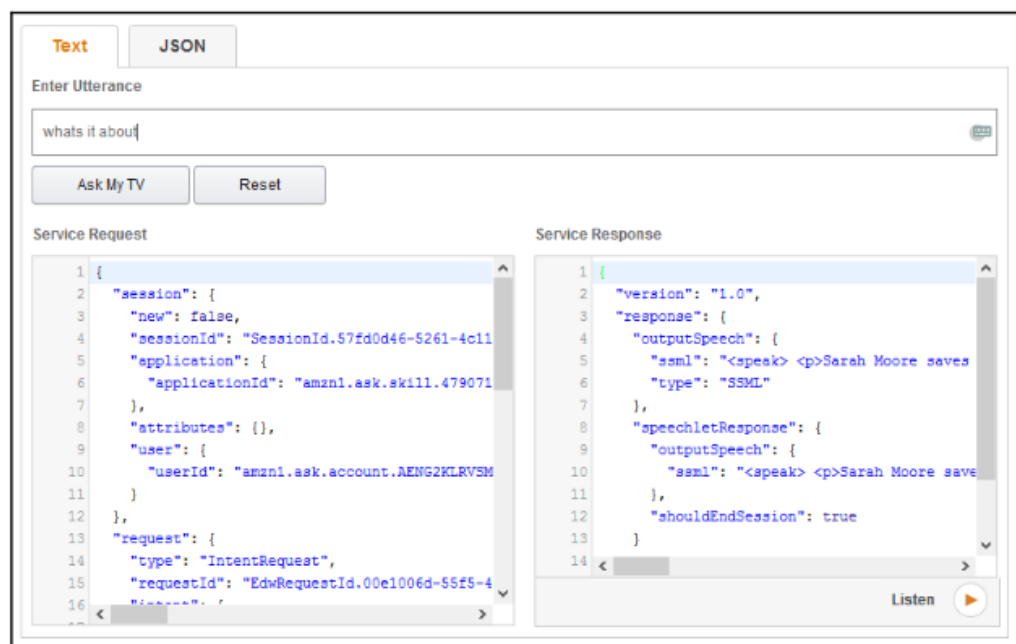
Gambar 13.33 mencoba dialog TV

2. Coba apa yang ada di bbc two pada jam 4 sore dan klik Ask My TV:



Gambar 13.34 dialog chatbot bbc jam 4 sore

3. Terakhir, mari kita coba mendapatkan lebih banyak informasi tentang acara tersebut dengan permintaan tindak lanjut. Ingatlah untuk tidak menekan tombol Reset sebelum membuat permintaan tindak lanjut karena dapat mengatur ulang konteks percakapan:



Gambar 13.35 mengatur ulang konteks percakapan

Respons yang Anda dapatkan akan berupa penjelasan singkat tentang acara yang sebelumnya dikeluarkan Alexa. Berikut adalah bagian dari keluaran JSON:

```
"outputSpeech": {
  "ssml": "<speack> <p>Sarah Moore saves three items from the
tip in each episode and transforms them into a much more
valuable piece, before returning the profits back to the
original owners of the item.</p> </speack>",
  "type": "SSML"
},
```

- Setelah menguji skill di simulator, ujilah di Echo. Ingat, Anda perlu menyebutkan nama pemanggilan sebelum permintaan. Misalnya, Alexa, tanyakan pada TV saya, apa selanjutnya di BBC One.

13.5 INTENT DAN JENIS SLOT BAWAAN

Alexa memiliki perpustakaan maksud dan jenis slot bawaan yang dapat digunakan dalam keterampilan apa pun yang kami bangun. Menggunakan niat bawaan memiliki keuntungan tersendiri. Sebagai pengembang, Anda tidak perlu memberikan contoh ucapan dan setiap kali Alexa diperbarui dengan lebih banyak contoh ucapan, keterampilan Anda akan meningkat. Selain itu, memudahkan pengguna untuk mengingat serangkaian frasa sederhana di seluruh keterampilan untuk menyelesaikan sesuatu dengan Alexa.

Berikut ini serangkaian maksud bawaan standar yang dapat Anda gunakan:

Built-in intents	Keterangan
Amazon.YesIntent	Setiap kali pengguna mengatakan ya, ya, atau sesuatu yang serupa
Amazon.NoIntent	Saat pengguna mengatakan tidak, tidak, terima kasih, atau hal serupa
Amazon.CancelIntent	Ketika pengguna meminta untuk membatalkan tugas saat ini, seperti membatalkan, sudahlah, atau lupakan saja
Amazon.HelpIntent	Ketika pengguna meminta bantuan (bantuan, bisakah Anda membantu saya)
Amazon.StopIntent	Saat pengguna mengatakan berhenti, mati, atau tutup mulut

Alexa juga memiliki beberapa jenis slot yang bisa Anda gunakan:

Jenis slot bawaan	Keterangan
AMAZON.DATE	Deskripsi tanggal apa pun dapat diambil menggunakan jenis slot ini. Misalnya hari ini (10-

	10-2017), sekarang, akhir pekan ini (W40-2017-WE), tahun depan (2018), atau musim dingin ini. Frasa ini akan diterjemahkan ke dalam format tanggal ISO 8601.
AMAZON.TIME	Referensi waktu apa pun, seperti jam dua, besok pagi, atau siang hari, akan diambil dan disimpan dalam format waktu ISO 8601.
AMAZON.DURATION	Durasi, misalnya sepuluh menit, enam jam, atau tujuh tahun. Info yang diterjemahkan direpresentasikan dalam format durasi ISO 8601.
AMAZON.NUMBER	Angka berapa pun, misalnya satu, seratus, atau dua tiga empat.
AMAZON.FOUR_DIGIT_NUMBER	Nomor empat digit apa pun.

Ada jenis slot lainnya, seperti AMAZON.AggregateRating (untuk peringkat seperti terbaik, bintang lima, sepuluh besar), AMAZON.Animal (untuk hewan), AMAZON.AT_CITY (untuk kota-kota di seluruh dunia), AMAZON.Color (untuk warna), dan AMAZON.Country (untuk negara), yang dapat digunakan. Jelajahi daftar lengkap di URL berikut:

<https://developer.amazon.com/docs/custom-skills/slot-type-reference.html>

Ringkasan

Cemerlang! Saya yakin Anda bersenang-senang membangun dua keterampilan Alexa pertama Anda di bab ini. Kami menjelajahi dasar-dasar pengembangan keterampilan—niat dan slot. Kami menjelajahi proses membangun model interaksi pada platform pengembangan Alexa serta membangun aplikasi web pengelola percakapan di cloud. Kami telah membangun dua keterampilan—satu untuk kutipan dan satu lagi untuk jadwal TV—memberi kami banyak peluang untuk mengeksplorasi dan memahami berbagai elemen proses pengembangan keterampilan Alexa. Saya akan mendorong Anda untuk mengambil ini sebagai titik awal dan mengeksplorasi lebih jauh. Ada sekitar 10.000 keterampilan di toko Alexa dan terus berkembang. Dengan proyeksi pertumbuhan penggunaan speaker pintar bantuan rumah, akan ada pasar yang besar untuk keterampilan Alexa yang dapat disesuaikan di tahun-tahun mendatang. Di bab berikutnya, kita akan mempelajari cara membuat tindakan untuk Asisten Google dan menjalankannya di Google Home.

BAB 14

MEMBANGUN KETERAMPILAN DI ASISTEN GOOGLE

Dalam bab ini, kita akan menjelajahi proses membangun keterampilan di Asisten Google. Asisten Google adalah tawaran chatbot bantuan pribadi dari Google seperti Siri dari iPhone dan Alexa dari Amazon. Anda dapat berinteraksi dengan Asisten Google di berbagai saluran seperti aplikasi seluler Allo, Allo untuk web, dan Google Home.

Google Home adalah speaker pintar yang mirip dengan Amazon Echo dan ditenagai oleh Google Assistant. Dengan kata lain, Google Home adalah perwujudan tempat tinggal Asisten. Ini sangat mirip dengan hubungan antara Alexa dan Echo atau antara Siri dan iPhone. Google Home adalah pesaing langsung Amazon Echo dan saat ini memegang 24% pangsa pasar. Seperti Alexa, keterampilan (atau tindakan, begitu mereka menyebutnya) Asisten dapat ditingkatkan.

Dalam bab ini, kami akan membuat tindakan untuk Asisten Google. Dalam upaya mewujudkan man Friday, mari kita mulai dengan mengintegrasikannya ke dalam layanan online yang dapat digunakan untuk mengelola tugas sehari-hari Anda dengan mudah. Kami akan mengeksplorasi bagaimana tindakan dapat diaktifkan untuk berinteraksi dengan layanan ini guna menambah dan mengambil tugas. Pertama, mari jelajahi layanan online bernama Todoist dan periksa cara menambahkan dan mengambil tugas menggunakan REST API. Kemudian, kami akan membangun agen dengan toolkit Dialogflow dan mengintegrasikannya dengan Asisten Google sebagai tindakan. Terakhir, kami akan mengintegrasikan tindakan dengan layanan Todoist untuk menyelesaikan perulangan.

Pada akhir bab ini, Anda akan dapat:

- Bangun agen Dialogflow untuk menangani daftar tugas
- Integrasikan ke layanan Todoist menggunakan webhook pemenuhan
- Buat Google Actions dan integrasikan dengan Asisten Google
- Uji tindakannya di Google Home

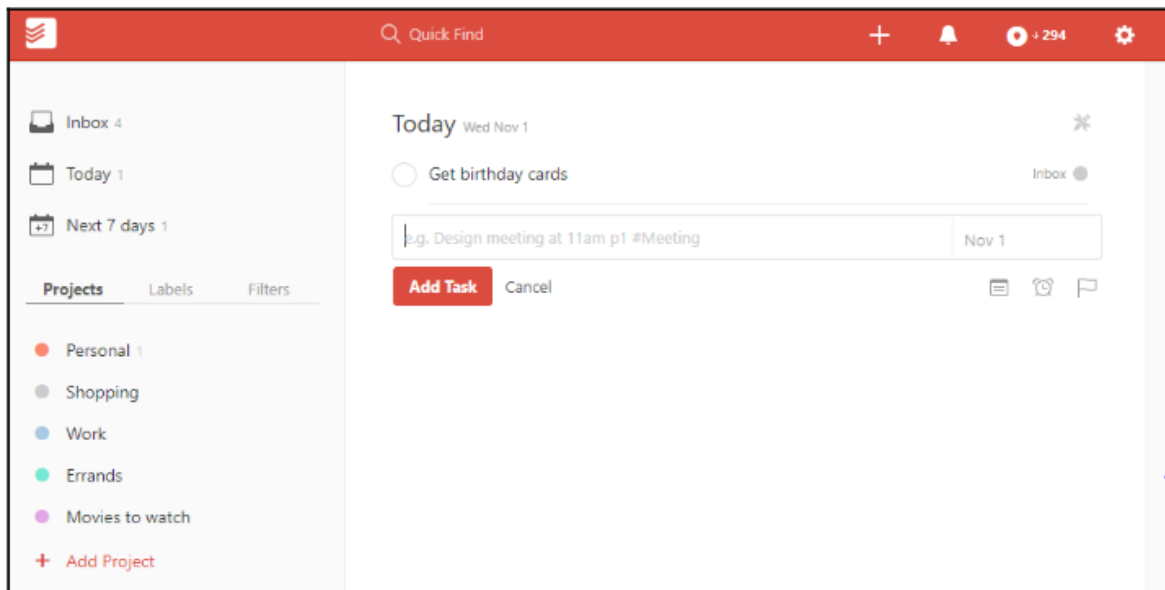
14.1 Todois

Todoist adalah layanan online untuk menyimpan dan mengatur daftar item yang Anda rencanakan dalam waktu dekat. Pengguna dapat menambahkan tugas ke akun mereka, mengaturnya berdasarkan proyek, dan menetapkan tenggat waktu dan pengingat untuk tugas mereka. Akun tersebut dapat diakses di situs web mereka serta aplikasi di perangkat Android dan iOS. Layanan Todoist juga dapat diakses menggunakan API pengembangnya. Mari kita periksa dua layanan dasar: menambahkan tugas dan mengambil daftar tugas untuk hari itu.

Mendapatkan kuncinya

Untuk mengakses Todoist, kita memerlukan kunci API. Untuk mendapatkannya, ikuti langkah-langkah berikut:

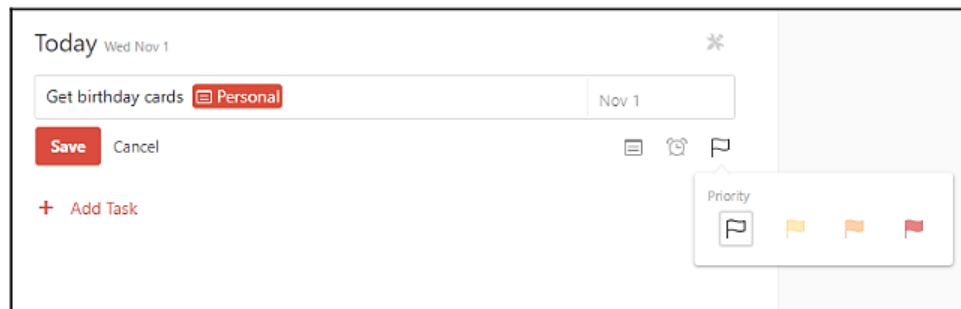
1. Kunjungi <https://todoist.com/>.
2. Mendaftar untuk sebuah akun.
3. Masuk ke akun dan lihat dashboard.
4. Coba tambahkan tugas dengan mengklik Tambah Tugas:



Gambar 14.1 menambah tugas pada todois

Anda dapat melihat bahwa tugas dapat diatur ke dalam proyek. Anda dapat membuat proyek baru jika Anda mau. Secara default, tugas akan ditambahkan ke kotak masuk Anda dan memiliki tanggal jatuh tempo pada hari pembuatannya.

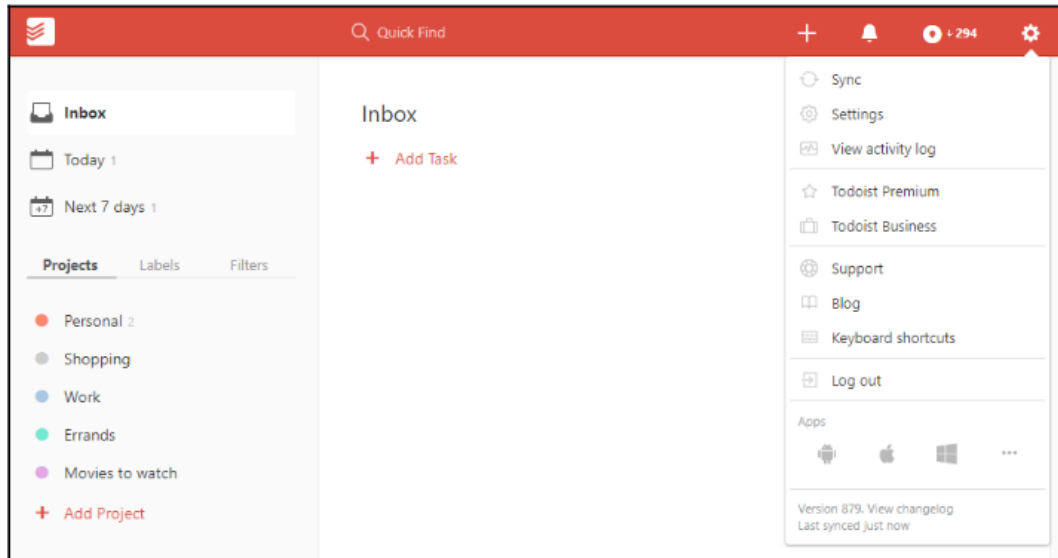
5. Klik tugas yang baru saja Anda buat dan jelajahi pilihannya:



Gambar 14.2 memilih tugas baru

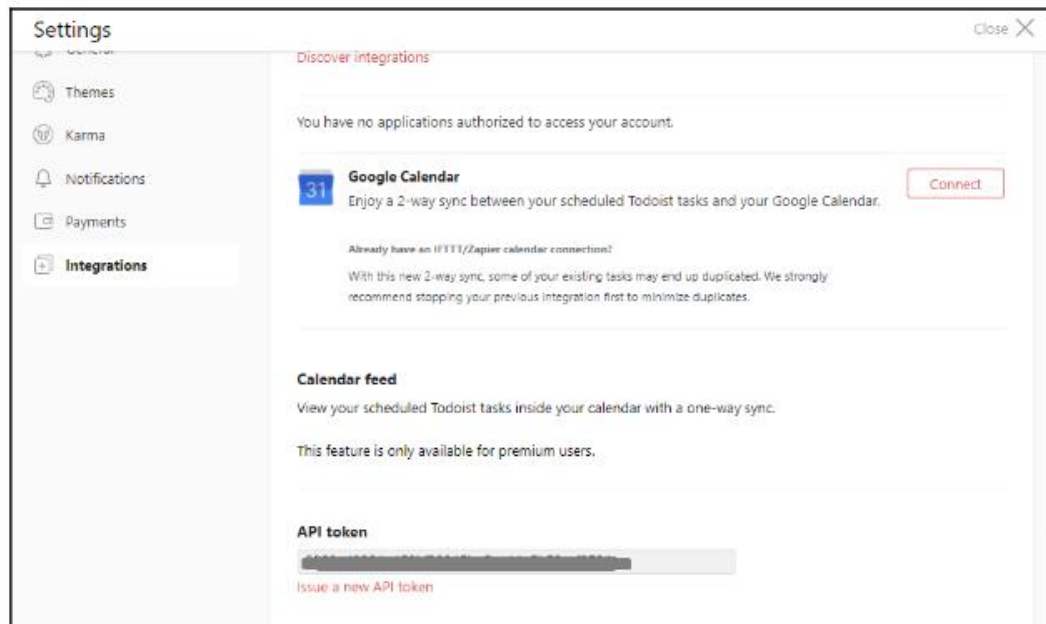
Anda dapat mengubah tanggal jatuh tempo, proyeknya, prioritas tugas, dan mengatur pengingat.

6. Sekarang kita telah menjelajahi kemampuan platform layanan, mari dapatkan kunci API ke akun Anda. Klik ikon pengaturan di sudut kanan atas:



Gambar 14.3 Pengaturan pada todois

7. Klik Pengaturan.
8. Pada menu samping, klik Integrasi.
9. Gulir ke bawah ke token API:



Gambar 14.4 mengatur token API

Salin token Anda dan simpan dengan aman.

Sekarang mari kita menulis program Node.js untuk menambahkan item dan mengambil daftar.

Menambahkan tugas

Dengan menggunakan kunci tersebut, kita dapat menambah dan mengambil daftar tugas yang harus dilakukan. Berikut adalah kode Node.js untuk menambahkan tugas ke daftar tugas Anda:

```
var task = {'type': 'item_add',
  'temp_id': uuid.v4(),
  'uuid': uuid.v4(),
  'args': {
    'content': taskDescription
  }
};

var url = 'https://todoist.com/api/v7/sync?token=' +
  todoist_token +
  '&sync_token=*&resource_types=["items"]&commands=['
  + JSON.stringify(task) + ']'
  console.log('URL:' + url);

request({
  url: url,
  method: 'GET'
},
function (error, response, body) {
  //response is from the bot
  if (!error && response.statusCode == 200) {
    console.log(JSON.parse(body));
  } else {
    console.log('Error: ' + error)
    console.log('Statuscode: ' + response.statusCode)
  }
});
```

Mengambil semua tugas untuk hari itu untuk mengambil semua tugas dalam daftar tugas hari itu, gunakan kode Node.js berikut:

```
var url = 'https://todoist.com/api/v7/sync?token='
  + todoist_token +
  '&sync_token=*' +
  '&resource_types=["items"]';
console.log('URL:' + url);
var resp = syncrequest('GET', url);
var allItems = JSON.parse(resp.getBody('utf8')).items;
var allItemsSummary = '';
for (var i = 0; i < allItems.length; i++){
  allItemsSummary += allItems[i].content;
  if (i < allItems.length - 2){
    allItemsSummary += ', ';
  }
  else if (i == allItems.length - 2) {
    allItemsSummary += ' and ';
  }
}
```

```

    else {
      allItemsSummary += '.';
    }
  }

  response = 'You have ' + allItems.length + ' in your list. '

      + allItemsSummary;
  console.log(response);

```

Kode sebelumnya nantinya akan ditanamkan ke dalam web service yang akan dipanggil dari Dialogflow untuk mendapatkan respon ucapan yang dihasilkan secara dinamis dari webhook. Untuk menjelajahi berbagai kemungkinan tindakan dalam Todoist, lihat dokumentasi di sini: <https://developer.todoist.com>.

Membangun suatu tindakan

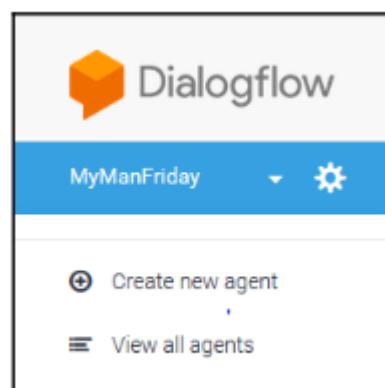
Untuk membuat Google Assistant Action, kita harus memulai dari Dialogflow. Secara garis besar langkah-langkahnya adalah sebagai berikut:

1. Bangun agen Dialogflow.
2. Buat aplikasi web yang mengakses layanan Todoist.
3. Integrasikan agen dengan Google Actions.
4. Uji di simulator.
5. Uji di Beranda Google.

Membangun agen Dialogflow

Untuk membangun agen Dialogflow, ikuti langkah-langkah berikut:

1. Buka dialogflow.com dan masuk menggunakan akun Google Anda.
2. Klik menu drop-down di sebelah kiri dan klik Buat agen baru:



Gambar 14.5 membuat agen dialogflow

3. Tambahkan info agen dan klik SIMPAN:

The screenshot shows a configuration form for an agent. At the top, there is a text input field labeled 'Agent name' with a blue 'SAVE' button and a vertical ellipsis menu icon to its right. Below this is a 'DESCRIPTION' section with a text area and the placeholder text 'Describe your agent'. The next section is 'ADD SAMPLE DATA' with a dropdown menu and the text 'Select sample data to be loaded into new agent or leave unselected for empty agent'. Below that are two dropdown menus: 'DEFAULT LANGUAGE' set to 'English - en' and 'DEFAULT TIME ZONE' set to '(GMT0:00) Africa/Casablanca'. The final section is 'GOOGLE PROJECT' with a dropdown menu and the text 'Create a new Google project'.

Gambar 14.6 Menambahkan info agen

4. Ini akan membuat proyek Google baru yang ditugaskan ke agen. Anda dapat melihat informasi pengaturan dengan mengklik ikon Pengaturan di sebelah nama agen pada daftar drop-down.

The screenshot shows the configuration page for an agent named 'MyManFriday'. At the top, there is a blue 'SAVE' button and a vertical ellipsis menu icon. Below the agent name, there are tabs for 'General', 'Languages', 'ML Settings', 'Export and Import', and 'Share'. The 'General' tab is selected. The page features a large orange cube icon on the left. To the right of the icon is the 'DESCRIPTION' section with a text area and the placeholder text 'Describe your agent'. Below this are two dropdown menus: 'DEFAULT LANGUAGE' set to 'English - en' and 'DEFAULT TIME ZONE' set to '(GMT+1:00) Europe/Madrid'. The 'GOOGLE PROJECT' section shows a 'Project ID' field with the value 'myga-e4798 (Google Cloud | Actions on Google)'. Below this is the 'API keys' section, which contains two rows: 'Client access token' and 'Developer access token', each with a masked value and a refresh icon.

Gambar 14.7 membuat proyek google baru

5. Mari tambahkan empat maksud pada agen untuk memulai:
- Greet
 - Add_task
 - Inform_task_description

- List_all_tasks

Kami berasumsi di sini bahwa pengguna akan memulai percakapan dengan salam (maksud salam) diikuti dengan permintaan untuk menambahkan tugas (add_task) atau membuat daftar semua tugas untuk hari itu (list_all_tasks). Dalam permintaan penambahan tugas berikut, agen akan meminta pengguna untuk menjelaskan tugas yang akan ditanggapi oleh pengguna dengan deskripsi tugas (inform_task_description).

6. Tambahkan beberapa contoh ucapan untuk maksud salam:

- Hi
- Hello there

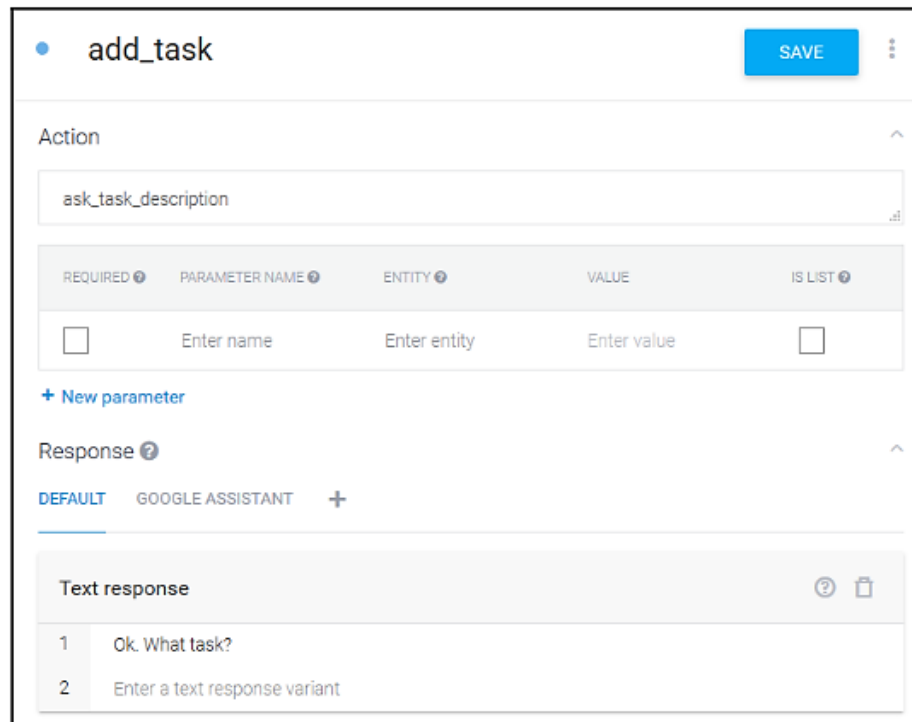
7. Mari kita kerjakan maksud add_task. Ini adalah maksud yang akan digunakan pengguna untuk menambahkan tugas ke daftar ini. Sekarang tambahkan beberapa ucapan di sini:

- ***I want to add a task***
- ***Add new task***
- ***Add a task to my list***



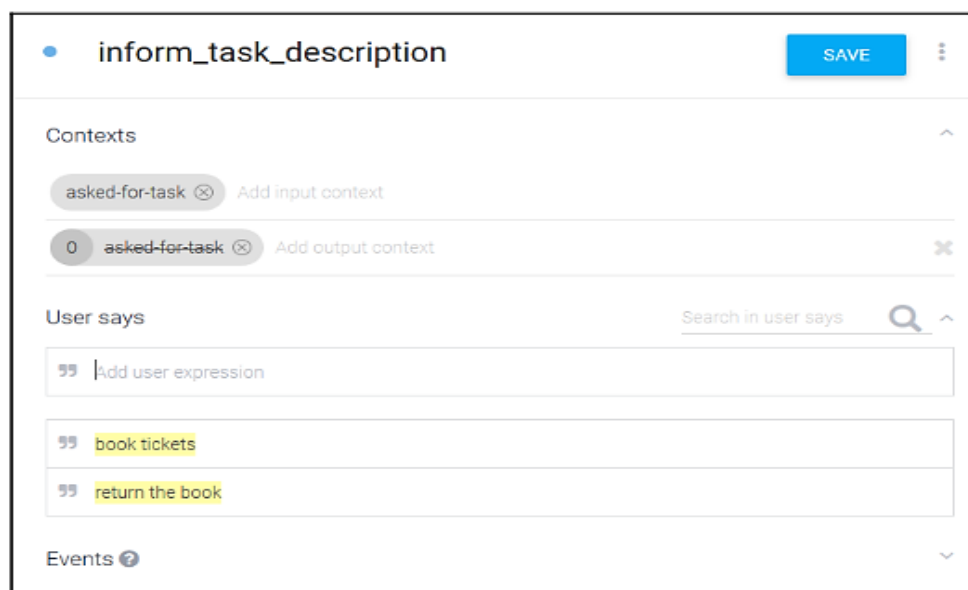
Gambar 14.8 Menambahkan list dialog

8. Kita perlu menindaklanjutinya dengan permintaan uraian tugas yang perlu ditambahkan. Jadi mari kita minta deskripsi tugas sebagai respons dan atur juga konteksnya ke tugas yang diminta:



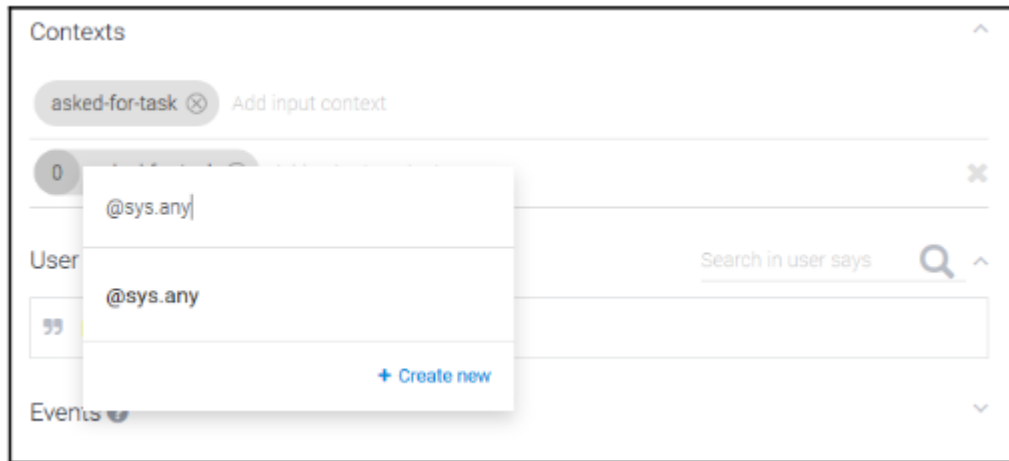
Gambar 14.9 menambahkan list respon dan dialog

9. Pengguna secara alami akan merespons dengan deskripsi tugas. Kami perlu menangkap deskripsi tersebut dan mengirimkannya ke aplikasi web yang dapat mencatatnya di daftar Todoist Anda. Mari kita lakukan ini dengan maksud `inform_task_description` dalam konteks tugas yang diminta:



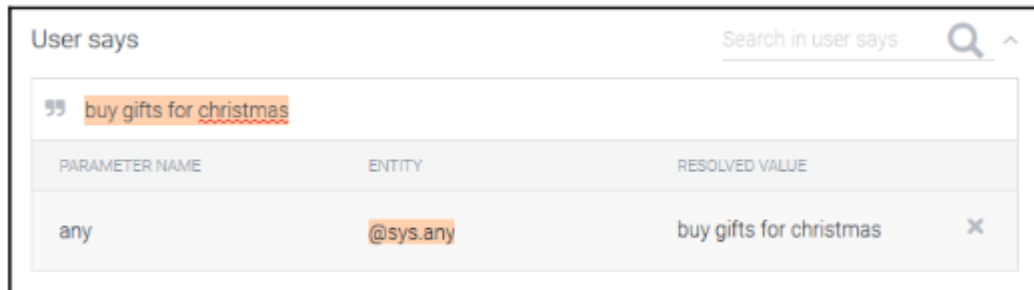
Gambar 14.10 memberikan informasi deskripsi

10. Semua ucapan di `inform_task_descriptions` dapat dianggap sebagai deskripsi tugas dan oleh karena itu dikirim ke Todoist apa adanya. Untuk menangkap seluruh ucapan dalam sebuah parameter (misalnya tugas), pilih seluruh ucapan dan pilih `@sys.any` di menu drop-down yang muncul:



Gambar 14.11 konten todois

11. Ini akan membuat parameter bernama `any` yang terkait dengan maksud. Mari kita ganti namanya menjadi tugas. Klik apa saja dan ubah menjadi tugas:



Gambar 14.12 mengubah tugas

12. Lakukan hal yang sama untuk beberapa contoh ucapan lainnya. Berikut ini adalah uraian tugas:
- Buy gifts for Christmas
 - Send a birthday card to jones
 - Have tablet after lunch

13. Tambahkan Maaf. Saya tidak dapat menambahkan tugas baru. sebagai respons terhadap niat tersebut. Respons ini akan dikirimkan hanya ketika agen gagal menambahkan tugas baru menggunakan aplikasi web. Kami akan mengaktifkan webhook untuk tujuan ini sebentar lagi. Simpan maksudnya dengan mengklik tombol SAVE.
14. Buat maksud list_all_tasks baru untuk mencantumkan semua tugas dalam daftar tugas pengguna. Tambahkan ucapan berikut:
- What tasks do I have still
 - List all my tasks
 - Get me the list off all tasks
- Beri nama tindakan respons: inform_all_tasks dan tetapkan respons default ke Maaf. Saya tidak bisa mendapatkan daftar tugas sekarang. Kami juga akan mengaktifkan webhook untuk maksud ini nanti.
15. Sekarang mari kita siapkan layanan web untuk memanggil maksud list_all_tasks dan add_task. Klik Pemenuhan pada menu sebelah kiri.
16. Aktifkan webhook dan ketik URL aplikasi web yang akan dihubungi. Karena kita belum memilikinya, buat saja placeholder-nya (<https://my-home-bot.herokuapp.com/home>):

Fulfillment

Webhook ENABLED

Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#).

[Webhook example](#)

URL*

BASIC AUTH

HEADERS

[+](#) Add header

Gambar 14.13 Aktifkan webhook dan ketik URL aplikasi web

Gulir ke bawah hingga akhir halaman dan **klik SIMPAN**.

17. Kembali ke maksud `list_all_tasks` dan `add_task`. Untuk masing-masing maksud ini, gulir ke bawah dan klik Pemenuhan. Hal ini memastikan bahwa maksud dan parameter dikirim ke aplikasi web yang menghasilkan ucapan respons yang sesuai.

14.2 Membuat Aplikasi Web

Sekarang kita memiliki agen Dialogflow untuk memahami ucapan pengguna, kita perlu membangun aplikasi web yang dapat berinteraksi dengannya. Agen akan mengirimkan maksud dan parameter pengguna ke aplikasi web sehingga respons yang sesuai dapat dihasilkan. Misalnya, ketika pengguna ingin menambahkan tugas baru, deskripsi tugas perlu dikirim ke aplikasi web, yang pada gilirannya akan menambahkan tugas baru ke daftar tugas Anda menggunakan Todoist API. Untuk membuat aplikasi web, ikuti langkah-langkah berikut:

1. Buat proyek Node.js baru menggunakan npm dan buat proyek bot rumah:

>npm init

2. Instal perpustakaan yang diperlukan:

- Npm install request == save
- Npm install sync-request -- save
- Npm install express -- save
- Npm install body-parser -- save
- Npm install moment -- save
- Npm install node-uuid -- save

`request` dan `sync-request` adalah perpustakaan yang akan digunakan untuk berinteraksi dengan layanan Todoist. `express` dan `body-parser` akan digunakan untuk menjalankan server web. `moment` akan digunakan untuk perhitungan terkait waktu/tanggal. Terakhir, `node-uuid` akan digunakan untuk menghasilkan UUID untuk tugas.

3. Buat file bernama `index.js`. Tambahkan pernyataan import, seperti yang ditunjukkan di sini:

```
'use strict'

/*
 * Google Home Bot
 */

const request = require('request')
const syncrequest = require('sync-request')
const express = require('express')
const bodyParser = require('body-parser')
const moment = require('moment');
const uuid = require('node-uuid');
```

4. Untuk menjalankan server, kita perlu menambahkan kode berikut:

```
const app = express()

app.set('port', (process.env.PORT || 5000))

// Process application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({extended: false}))

// Process application/json
app.use(bodyParser.json())

app.use(express.static('public'))

// Index route
app.get('/', function (req, res) {
  res.send('Hello world, I am your man Friday.')
})

// Handle the intents
app.post('/home', function (req, res) {
  // Handling the intent code
})

// Spin up the server
app.listen(app.get('port'), function() {
  console.log('running on port', app.get('port'))
})
```

5. Tambahkan kode berikut pada handle /home POST untuk menangani maksudnya:

```
console.log('Request from DialogFlow:');
console.log(req.body);
var intent = req.body.result.metadata.intentName;
var botSpeech = 'hello world';
if (intent === 'inform_task_description'){
  var taskDescription = req.body.result.parameters.task;
  addTask(taskDescription);
  botSpeech = 'Adding new task.';
}
else if (intent === 'list_all_tasks'){
  botSpeech = getAllTasks();
}

var out = {speech: botSpeech,
  displayText: botSpeech,
  data: null};
var outString = JSON.stringify(out);
console.log('Out:' + outString);
res.send(outString);
```

6. Terakhir, kita perlu membuat dua fungsi yang dipanggil dari /homehandle. Untuk menambahkan tugas baru, pertama kita tambahkan kode berikut:

```
function addTask(taskDescription) {
  var todoist_token = 'YOUR-TODOISTI-TOKEN';
  var task = {'type': 'item_add',
    'temp_id': uuid.v4(),
    'uuid': uuid.v4(),
    'args': {
      'content': taskDescription
    }
  };

  var url = 'https://todoist.com/api/v7/sync?token='
    + todoist_token +
    '&sync_token=*' + '&resource_types=["items"]&commands=['
    + JSON.stringify(task) + ']'

  console.log('URL:' + url);
  request({
    url: url,
    method: 'GET'
  },
  function (error, response, body) {
    //response is from the bot
    if (!error && response.statusCode == 200) {
      console.log(JSON.parse(body));
    } else {
      console.log('Error: ' + error)
      console.log('Statuscode: ' + response.statusCode)
    }
  })
}
```

- Dan selanjutnya, untuk mendapatkan ringkasan semua tugas, kami menambahkan kode berikut:

```
function getAllTasks(){
  var todoist_token = 'YOUR-TODOISTI-TOKEN';
  var url = 'https://todoist.com/api/v7/sync?token='
    + todoist_token +
    '&sync_token=*' +
    '&resource_types=["items"]';

  console.log('URL:' + url);
  var resp = syncrequest('GET', url);
  var allItems = JSON.parse(resp.getBody('utf8')).items;
  var allItemsSummary = '';
  for (var i = 0; i < allItems.length; i++){
    allItemsSummary += allItems[i].content;
    if (i < allItems.length - 2){
      allItemsSummary += ', ';
    }
    else if (i == allItems.length - 2) {
      allItemsSummary += ' and ';
    }
    else {
      allItemsSummary += '.';
    }
  }
}
```

```

    }
  }
  var alltasks = 'You have ' + allItems.length + '
                in your list. '
                + allItemsSummary;
  return alltasks;
}

```

7. Mari kita dorong ke cloud sekarang:

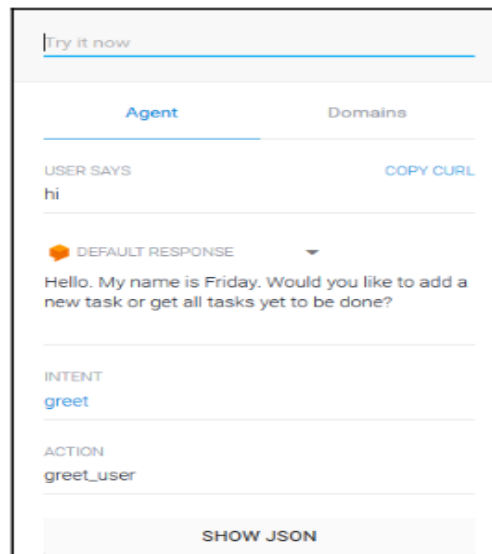
- Git init
- Git add
- Git commit -m initial -commit
- Heroku create my-home-bot
- Git push Heroku master

Setelah didorong, aplikasi web siap untuk berkomunikasi dengan agen Dialogflow kami.

14.3 Menguji agen di Dialogflow

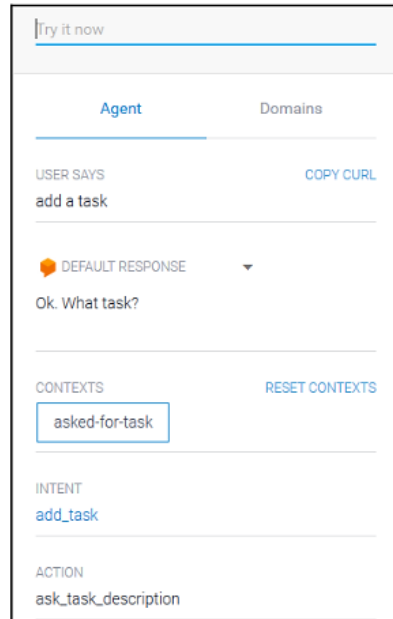
Sekarang kami siap untuk menguji orang kami pada hari Jumat di Dialogflow. Di Dialogflow, Anda dapat menguji agen Anda menggunakan simulator. Jika Anda telah membuat agen Dialogflow (sebelumnya disebut API.AI) di salah satu bab sebelumnya, Anda mungkin sudah mengetahui cara melakukan proses ini. Jika tidak, ikuti langkah-langkah berikut:

1. Kunjungi dialogflow.com.
2. Di sebelah kanan, Anda akan melihat bingkai dengan kolom teks Coba sekarang di bagian atas. Ketik hai dan tekan kembali. Di sinilah kita akan mengetik masukan kita kepada agen. Agen akan mengembalikan respons tekstual serta maksud pengguna dan tindakan sistem:



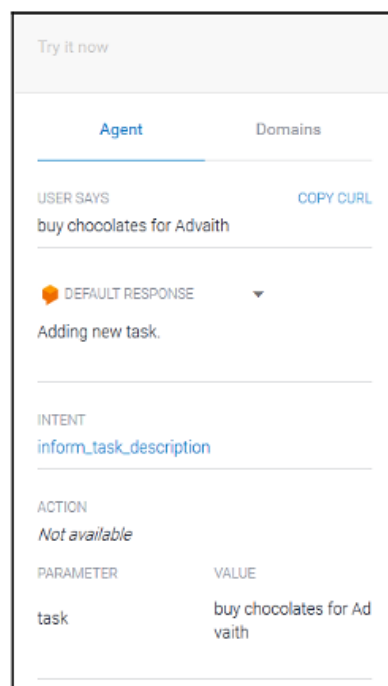
Gambar 14.14 kolom teks Coba

3. Sekarang tambahkan tugas. Coba tambahkan ucapan tugas baru. Agen sekarang harus meminta deskripsi tugas. Anda juga akan melihat bahwa konteksnya diperbarui menjadi tugas yang diminta:



Gambar 14.15 Menambahkan tugas

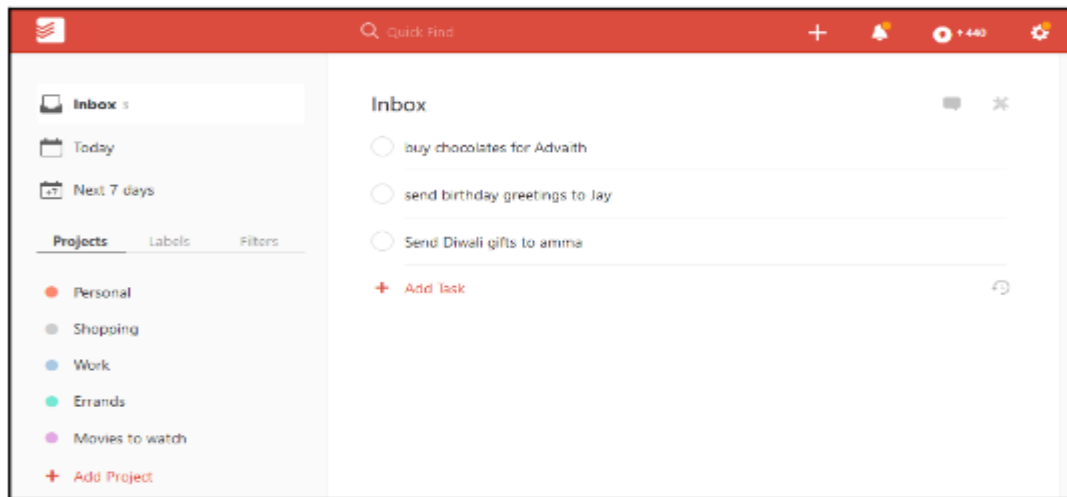
4. Berikan deskripsi tugas. Agen sekarang akan menambahkan tugas baru ke daftar tugas Anda dan kembali dengan pemberitahuan:



Gambar 14.16 menambahkan tugas baru ke daftar tugas

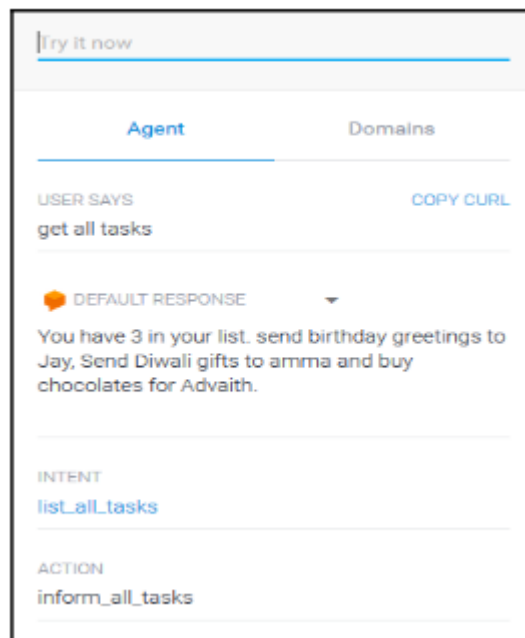
Jika gagal menghubungi layanan Todoist karena alasan tertentu, pesan respons default akan dikembalikan.

5. Tambahkan beberapa tugas lagi dengan cara yang sama.
6. Sekarang kita dapat memeriksa tugas-tugas yang ditambahkan ke daftar tugas kita dengan masuk ke todoist.com atau menggunakan aplikasi Todoist:



Gambar 14.17 menambahkan ke daftar tugas kita dengan masuk ke todoist.com atau menggunakan aplikasi Todoist

7. Sekarang coba ambil semua tugas. Coba ucapkan dapatkan semua tugas:

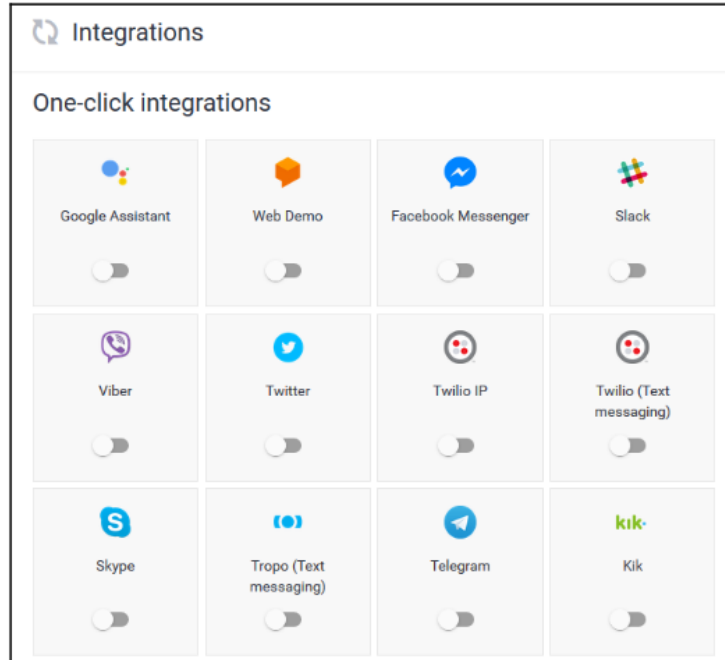


Gambar 14.18 mencoba ucapan pada semua tugas

14.4 Mengintegrasikan dengan Asisten Google

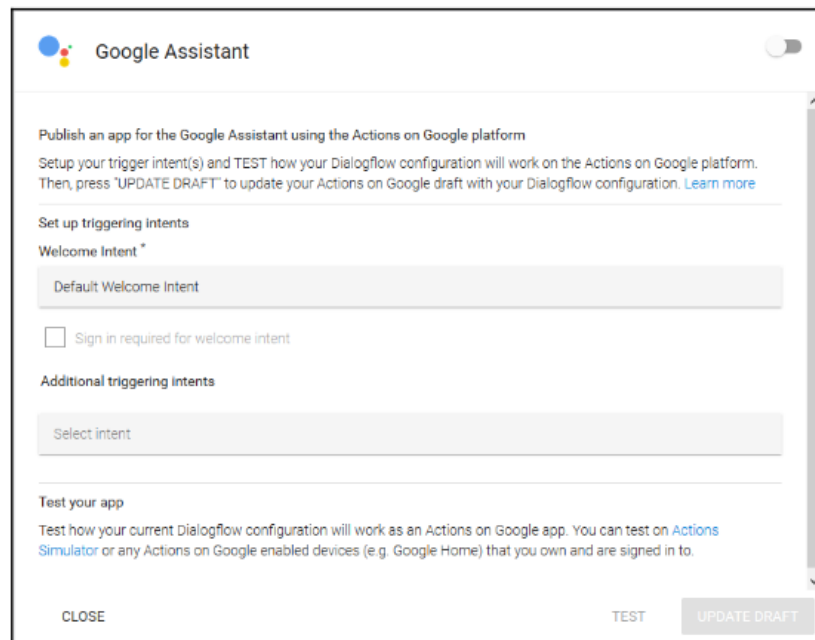
Sekarang agennya sudah siap, langkah kami selanjutnya adalah mengintegrasikannya dengan Asisten Google. Lakukan langkah-langkah berikut ini untuk melakukan integrasi ini:

1. Pada menu Dialogflow di sebelah kiri, klik Integrasi:



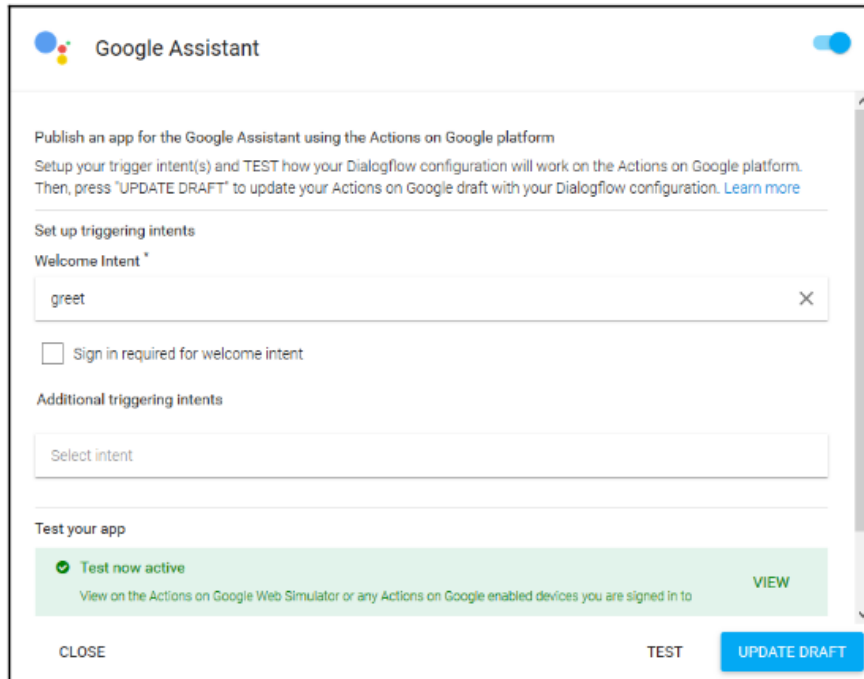
Gambar 14.19 mengintegrasikan ke beberapa platform

2. Klik Asisten Google:



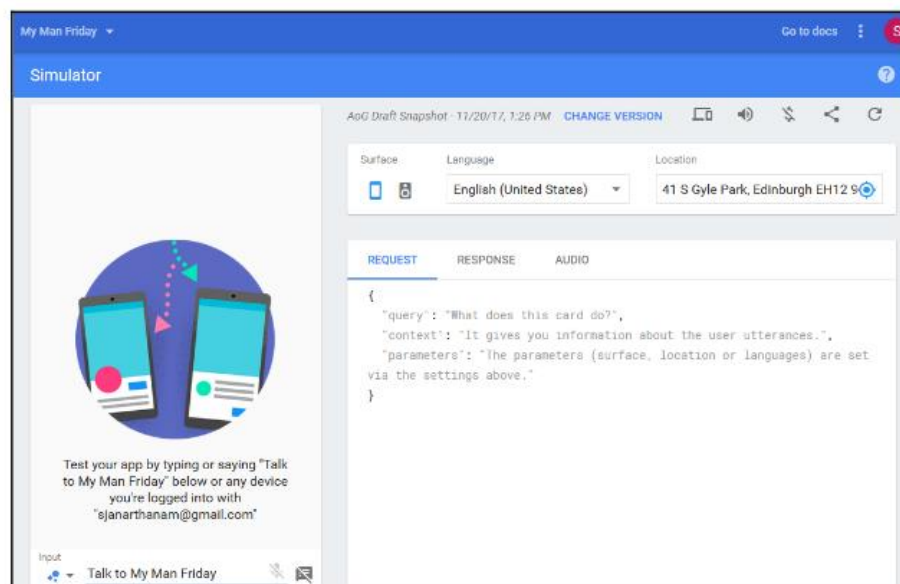
Gambar 14.20 Asisten google

3. Pada kotak dialog Asisten Google, gerakkan penggeser di pojok kanan atas ke posisi aktif.
4. Pilih niat Selamat Datang. Pilih maksud salam yang telah kita tentukan.
5. Klik UJI lalu LIHAT untuk membuka Simulator Tindakan:



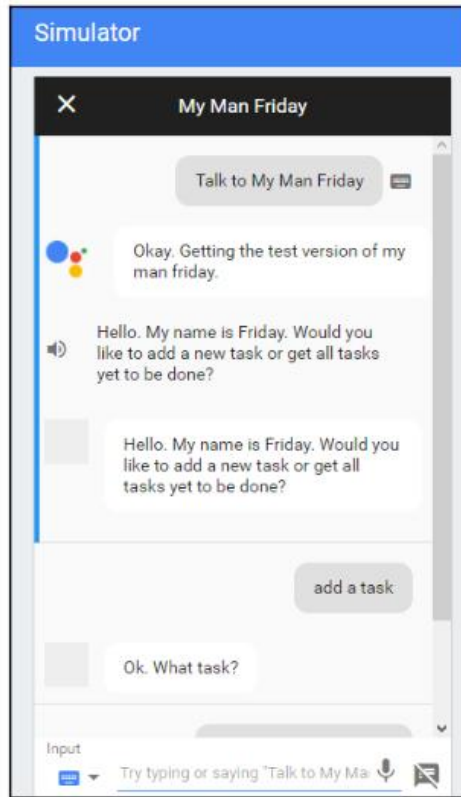
Gambar 14.21 pengujian pada google assiten

Ini akan membuka tab lain dan membawa Anda ke console.actions.google.com tempat agen dapat diuji dalam simulasi sebagai tindakan di Asisten Google:



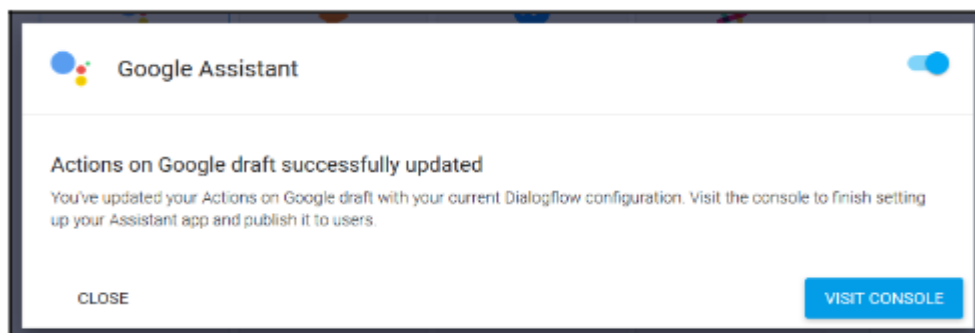
Gambar 14.22 simulasi chatbot

6. Ubah tipe Surface menjadi Phone dan tipe Input menjadi Keyboard.
7. Ketik Talk to my test app dan tekan return. Ini akan membuka versi uji tindakan dan Anda seharusnya dapat melakukan percakapan yang sama dengan agen seperti yang kami lakukan di Dialogflow. Semua tugas akan ditambahkan ke daftar tugas Anda, yang dapat diperiksa dengan mengunjungi todoist.com atau meminta untuk mendapatkan semua tugas setelah tugas tes ditambahkan:



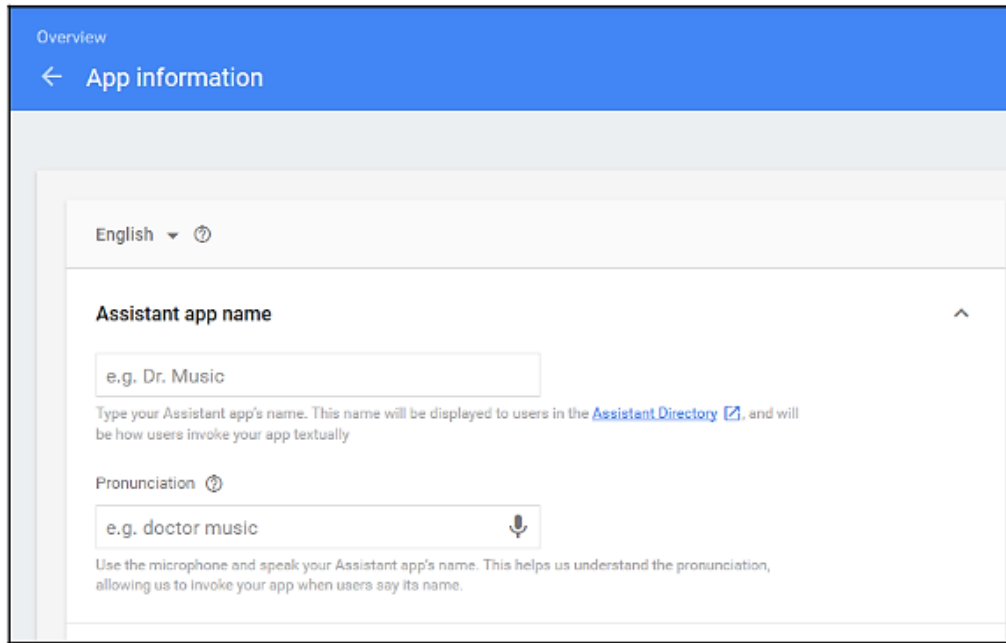
Gambar 14.23 simulasi todois

8. Setelah menguji agen, kembali ke Dialogflow, klik Perbarui Draf, dan klik Kunjungi Konsol di kotak dialog yang terbuka:



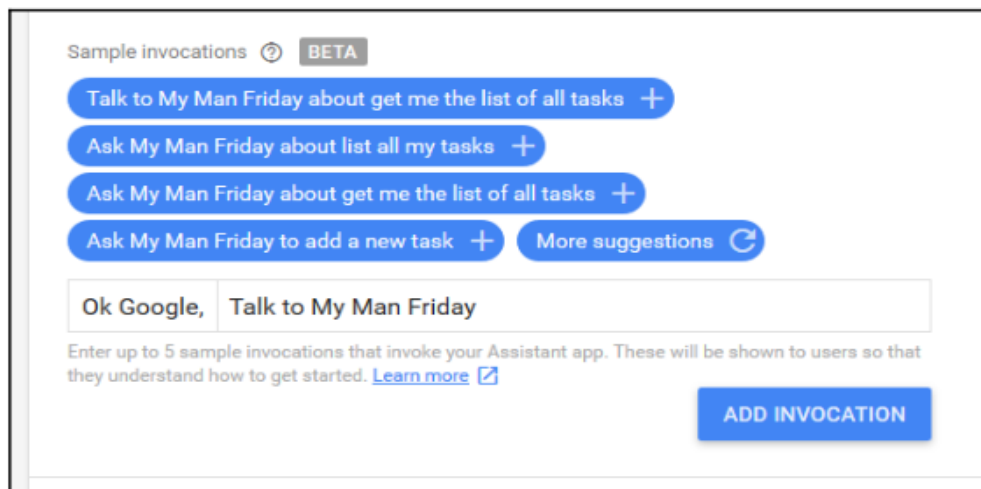
Gambar 14.24 memperbarui draf di kotak dialog

9. Pada halaman Ikhtisar tindakan, Anda akan melihat bahwa tindakan tersebut ditautkan ke Dialogflow. Langkah selanjutnya adalah memberikan beberapa informasi aplikasi penting.
10. Klik Sunting:



Gambar 14.25 menyesuaikan pada halaman ikhtisar tindakan

11. Ketikkan nama, pengucapan, dan deskripsi.
12. Gulir ke bawah untuk menemukan pemanggilan default untuk tindakan tersebut. Anda dapat menambahkan lebih banyak jika perlu:



Gambar 14.26 mencoba pemanggilan default

13. Tambahkan gambar untuk ikon dan spanduk. Ini dapat dikosongkan selama tahap pengembangan.
14. Gulir ke bawah dan klik SIMPAN.

Kini tindakan tersebut tersedia untuk menguji Akun Google yang terkait dengan pengembang. Untuk membuatnya tersedia bagi pengguna lain, Anda harus mengirimkan tindakan untuk persetujuan di URL berikut:

<https://developers.google.com/actions/distribute>

14.5 UJI DI BERANDA GOOGLE

Sekarang tindakannya sudah siap, Anda dapat mengujinya di perangkat Google Home Anda. Anda harus menyiapkan Akun Google yang sama yang digunakan untuk membuat agen dan tindakan, karena masih dalam pengembangan dan belum dipublikasikan. Jadi, nyalakan Google Home Anda dan ucapkan, Hai Google, bicaralah dengan teman saya hari Jumat:



Gambar 14.26 mikrofon google asisten

Anda sekarang seharusnya dapat melakukan percakapan yang sama seperti yang Anda lakukan di simulator, di Google Home menggunakan suara. Meskipun keakuratan pengenalan ucapan cukup tinggi dalam hal kata dan frasa umum dalam bahasa Inggris, alat ini mungkin masih tidak dapat mengenali nama yang tidak biasa dan konsep non-Barat.

Percakapan akan dimulai dengan pesan penafian yang menyatakan bahwa tindakan tersebut masih dalam versi uji coba. Coba tambahkan tugas dan tinjau semua tugas yang masih harus Anda selesaikan. Buka aplikasi Todoist untuk menyelesaikan tugas yang telah diselesaikan dan meninjaunya kembali di perangkat. Selamat bersenang-senang. Selamat, Anda telah membuat tindakan Anda sendiri untuk Asisten Google!

BAB 15

PEMBUATAN BOT TINGKAT LANJUT

Di bab sebelumnya, Anda telah mempelajari tentang elemen dasar chatbot, termasuk maksud dan entitas. Memahami maksud dan mengekstrak entitas penting dari pesan pengguna adalah bagian terpenting dalam membangun chatbot. Setelah Anda menangani kedua hal ini, inilah waktunya untuk menyajikan kepada pengguna balasan yang paling intuitif dan estetis.

Dalam bab ini, Anda akan mempelajari cara membuat chatbot lintas platform siap produksi dengan standar industri terbaik dan cara membuat bot yang akan berinteraksi dengan layanan API pihak ketiga untuk mendapatkan informasi dinamis. Anda kemudian akan mempelajari cara menyimpan pesan yang dipertukarkan antara pengguna dan bot di MongoDB yang Anda siapkan sebelumnya di buku ini. Terakhir, Anda akan melanjutkan untuk membuat pengklasifikasi niat Anda sendiri berdasarkan algoritma naïve Bayes.

Pertama, dalam bab ini, kita akan membahas berbagai elemen UI yang tersedia untuk digunakan pada berbagai platform perpesanan dan cara memanfaatkannya untuk mendapatkan pengalaman pengguna (UX) yang memuaskan.

15.1 PRINSIP DESAIN

Anda semua tahu bahwa chatbots adalah teknologi yang sama sekali baru. Ini seperti usia awal Web. Segalanya masih goyah namun berkembang dengan kecepatan cahaya. Meskipun demikian, setiap orang memiliki pendapatnya masing-masing tentang cara membangun chatbot yang memberikan kepuasan pengguna yang maksimal.

Katakanlah Anda ingin membuat aplikasi seluler baru. Industri aplikasi seluler sudah sangat matang pada saat ini sehingga Anda mungkin langsung memikirkan antarmuka dengan tombol navigasi, tab, dan kanvas yang penuh dengan informasi berguna untuk berinteraksi.

Hal yang sama berlaku ketika Anda mengunduh aplikasi baru ke ponsel Anda atau menjelajahi situs web. Setelah Anda mendarat di layar beranda aplikasi atau situs web, Anda segera mencari beberapa elemen penting seperti tombol navigasi, formulir, bagian konten, dan sebagainya, yang merupakan elemen di sebagian besar aplikasi/situs web. Untuk industri chatbot, kasusnya tidak sama. Belum ada spesifikasi atau panduan yang dapat membantu Anda dalam pengembangan. Namun, bukan tidak mungkin menemukan kesamaan di antara chatbot yang telah dibuat. Komunitas yang secara aktif membangun chatbot mengikuti beberapa pola umum, yang pada akhirnya akan memudahkan pengguna akhir untuk menemukan kesamaan di semua chatbot. Kami pikir hal ini akan berperan besar dalam mendemokratisasi chatbots kepada masyarakat luas. Berikut ini adalah saran utama kami dalam hal UX saat membangun chatbot.

Meskipun Anda mungkin ingin menyampaikan banyak hal, jangan membombardir pengguna dengan teks yang banyak karena mereka tidak akan membacanya. Buatlah sesingkat mungkin. Jika diperlukan, bagi pesan menjadi beberapa bagian dan kirimkan dalam beberapa bagian, tetapi jangan hanya menyajikan pesan yang panjang. Anda juga dapat meringkas pesan dan mengirimkan pesan ringkasan dengan sebuah tombol, yang akan mengirimkan pesan terperinci ketika diklik. Misalnya, teknik ini berfungsi dengan baik saat Anda mengirimkan tagihan ke pengguna; pada pesan pertama, cukup kirim jumlah kumulatif dan kirim tombol yang bertuliskan Lihat Tagihan Terperinci. Setelah tombol diklik, kirimkan elemen kaya yang bagus yang menunjukkan detail tagihan.

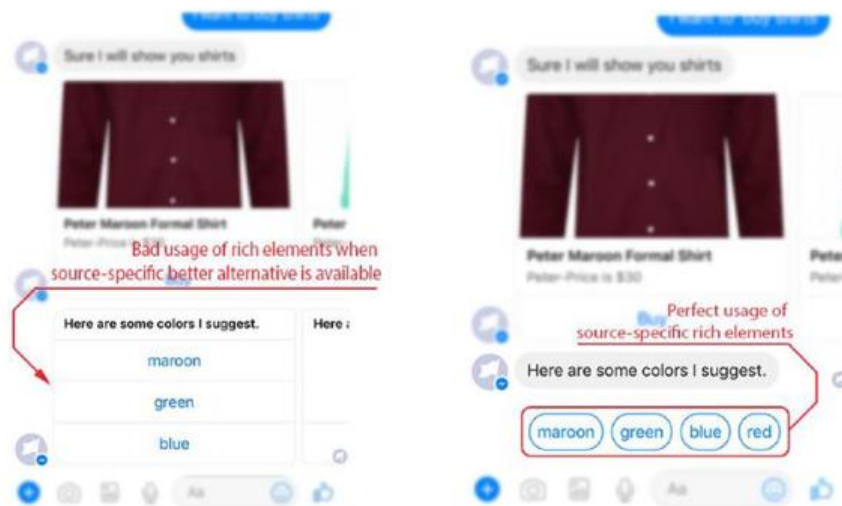
Manfaatkan Elemen Kaya

Ada sejumlah platform perpesanan yang dapat Anda manfaatkan untuk menerapkan chatbot Anda, dan banyak di antaranya mendukung pengiriman pesan kaya seperti carousel, tombol, gambar, video, dan sebagainya, bersama dengan pesan teks biasa. Anda dapat menggunakan elemen pesan kaya berbasis data ini untuk menciptakan pengalaman pengguna yang tak tertandingi. Salah satu kelebihan chatbot adalah mereka dapat mengambil data dari berbagai sumber data seperti database, API, dan sebagainya, lebih cepat daripada manusia dan dapat menghasilkan elemen pesan yang kaya dan interaktif. Misalnya, chatbot yang mewakili rumah mode akan dapat mengambil dan menampilkan beberapa opsi dari database dan menyajikannya kepada pengguna dalam carousel yang bagus dengan tombol interaktif. Jadi, Anda harus menggunakan elemen pesan kaya khusus sumber. Nanti di bagian ini, kami akan membahas sejumlah elemen kaya secara mendetail untuk memberi Anda gambaran umum.

Hormati Sumbernya

Saat pesan dikirim ke chatbot, ia mengetahui saluran sumber pesan tersebut. Terkadang sumber juga menyediakan beberapa data tentang pengguna, seperti nama, wilayah, zona waktu, lokal, dan lain sebagainya, yang dapat dimanfaatkan. Misalnya, saat menyapa pengguna, Anda selalu dapat memulai dengan “Hai, {first_name}.” Selain itu, jika lokal untuk pengguna tersedia, Anda dapat mengganti bahasa chatbot jika chatbot mendukung lokal pengguna. Selain itu, setiap platform memiliki opsi perpesanan sendiri-sendiri. Facebook Messenger memiliki jangkauan format pesan kaya terluas. Slack, Telegram, Skype, Viber, dan sebagainya, juga memiliki banyak opsi pemformatan pesan yang kaya. Itu selalu baik untuk memformat pesan sesuai dengan sumbernya. Misalnya, saat Anda meminta pengguna untuk memilih dari beberapa opsi atau memberi mereka saran tentang apa yang akan mereka katakan selanjutnya, Anda dapat mengirimkan daftar tombol untuk semua saluran perpesanan. Jika Anda melakukannya, pengalamannya mungkin berbeda-beda berdasarkan sumbernya. Misalnya, Facebook Messenger mendukung pengiriman balasan cepat yang dapat digulir secara horizontal, yang dimaksudkan untuk melakukan hal tersebut: memberikan saran kepada pengguna tentang apa yang mungkin mereka tanyakan atau katakan selanjutnya. Jika Anda

mengingat sumbernya dan menyusun pesan berdasarkan asal pesan, hal ini benar-benar dapat meningkatkan pengalaman pengguna (lihat Gambar 8-1).



Gambar 15.1. Contoh Penggunaan Penggunaan Elemen Khusus Sumber

Gunakan Penyerahan Manusia

Chatbots tidak sempurna. Kita semua tahu itu. Mereka dirancang untuk menanggapi serangkaian pertanyaan terbatas dan membuat kehidupan agen manusia lebih mudah. Oleh karena itu, chatbot mungkin gagal menjawab pertanyaan tertentu dari pengguna. Selain itu, pengguna mungkin tersesat di tengah percakapan.

Skenario ini sering terjadi dan dapat membuat pengguna akhir frustrasi. Mengingat hal tersebut, sebaiknya gunakan failover manusia dalam kasus tertentu ketika chatbot tidak dapat menjawab pertanyaan pengguna. Ini akan menyelamatkan Anda dari meninggalkan kesan buruk terhadap chatbot oleh pengguna akhir.

Jangan Membuat Pisau Swiss Army

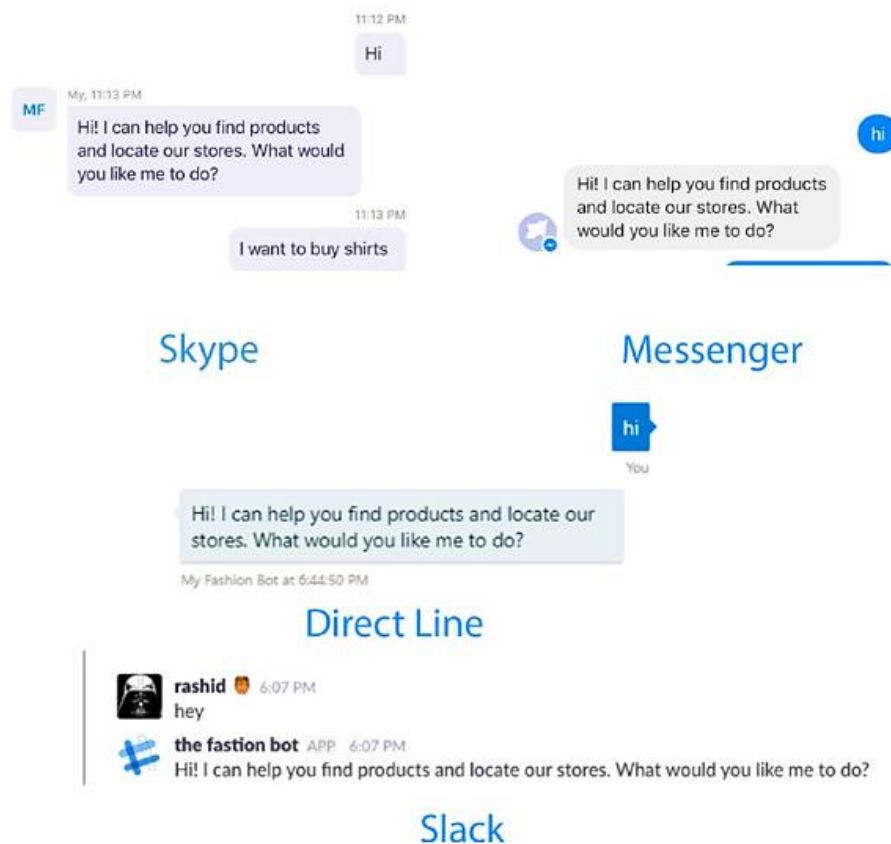
NLP dan AI belum mampu memahami bahasa manusia seperti yang Anda pahami. Jadi, mencoba membangun chatbot yang melakukan segalanya hanya akan menjadi mimpi buruk. Pengguna akan merasa frustrasi, dan pada akhirnya tujuan membangun chatbot menjadi tidak jelas. Cobalah untuk meminimalkan jumlah hal yang dapat dilakukan chatbot jika memungkinkan. Berdasarkan pengalaman kami, ketika jumlah tindakan yang dapat dilakukan chatbot melebihi tiga, chatbot mulai kebingungan saat mengambil keputusan.

Elemen Umum

Karena saluran perpesanan yang berbeda telah membuka platform mereka bagi pengembang untuk membuat chatbot, mereka juga memperkenalkan banyak elemen UI yang dapat digunakan tergantung pada situasinya. Kami akan membahas semua elemen UI umum yang saat ini tersedia di seluruh platform dan kemudian mendiskusikan elemen penting yang khusus untuk platform tersebut. Elemen UI berikut tersedia di semua platform.

Pesan Teks Biasa

Pesan teks biasa adalah elemen paling umum dari chatbot mana pun (lihat Gambar 15.2). Elemen ini adalah mode utama penyampaian informasi kepada pengguna. Semua platform mendukung pengiriman pesan teks biasa kepada pengguna. Saat mengirim pesan teks biasa kepada pengguna, Anda harus ingat bahwa pesan ini tidak boleh panjang. Pesan-pesan tersebut harus singkat, tepat, dan langsung pada sasaran. Sebagian besar platform mendorong hal ini, dan banyak dari mereka memiliki batas ukuran pesan maksimum. Misalnya, Facebook Messenger memiliki batasan 320 karakter per pesan.



Gambar 15.2. Pesan teks biasa pada platform berbeda

Berikut beberapa hal yang harus dilakukan:

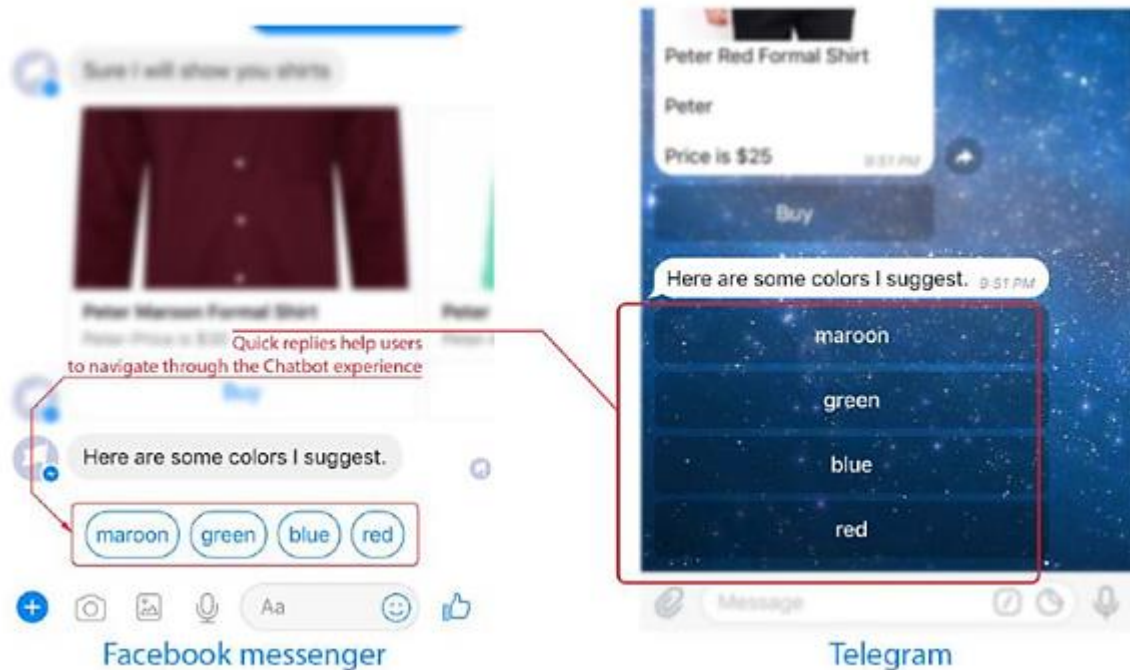
- Buatlah singkat.
- Buatlah ringkas.
- Gunakan emoji.

Berikut beberapa hal yang tidak boleh dilakukan:

- Jangan mengirim banyak pesan berturut-turut.

Balasan Cepat

Balasan cepat adalah tombol untuk memberikan petunjuk kepada pengguna tentang apa yang diharapkan chatbot sebagai balasan atas respons sebelumnya (lihat Gambar 8-3). Balasan cepat sangat bagus karena memberikan gambaran sekilas tentang apa yang akan dipahami dan apa yang tidak dipahami oleh chatbot. Ini adalah cara terbaik untuk berinteraksi dengan pengguna saat chatbot mengharapkan jawaban dari sekumpulan informasi. Meskipun balasan cepat adalah cara yang bagus untuk berinteraksi dengan pengguna, perlu diingat juga bahwa ketika pengguna diperlihatkan balasan cepat, hal itu menciptakan ilusi pilihan yang terbatas. Untuk mengatasinya, pengguna dapat diberikan opsi di quick reply untuk memasukkan “sesuatu yang lain”. Secara umum, balasan cepat ini membantu pengguna menghindari keharusan mengetik pesan yang sama berulang kali.



Gambar 15.3. Balasan cepat di Facebook Messenger

Berikut beberapa hal yang harus dilakukan:

- Buatlah elemen balasan cepat menjadi singkat.
- Pasti menggunakannya untuk mendapatkan konfirmasi.
- Memberikan opsi kepada pengguna untuk memasukkan informasi secara manual.

Berikut beberapa hal yang tidak boleh dilakukan:

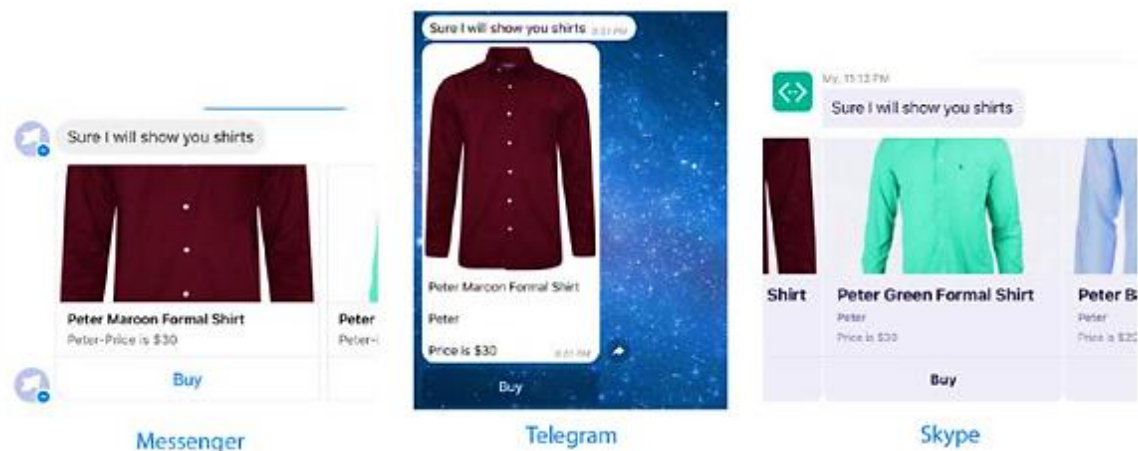
- Hindari pesan balasan cepat yang panjang.
- Hindari daftar balasan cepat yang panjang.

Korsel

Carousel adalah item yang dapat di-scroll secara horizontal, dengan setiap item berisi gambar, judul, deskripsi item (opsional), dan tombol untuk melakukan tindakan pengguna (opsional). Karsel adalah cara terbaik untuk menampilkan informasi serupa dengan cara yang elegan, memberikan pengguna fleksibilitas untuk melihat item dan mengambil tindakan pada masing-masing item.

Sebagai contoh, TechCrunch menggunakan carousel untuk menampilkan berita di chatbotnya, IFB menggunakan hal yang sama untuk melihat resep, dan seterusnya. Anda juga akan menggunakan banyak carousel di chatbot Anda. Karsel juga berguna untuk menampilkan “cara” untuk chatbot. Sama seperti setiap aplikasi yang memiliki proses orientasi yang memperkenalkan fitur-fitur berbeda kepada penggunanya, carousel di awal chatbot dapat digunakan untuk melihat pratinjau kemampuannya dengan visual. Ada banyak cara carousel dapat digunakan.

Terseher imajinasi Anda bagaimana Anda ingin menggunakannya (lihat Gambar 8-4). Semua bidang berbeda (gambar, deskripsi, judul) dapat dipadukan dan dicocokkan untuk menghasilkan kombinasi berbeda. Perhatikan bahwa, untuk membuat carousel, tidak wajib untuk menempatkan semua bidang di setiap item.



Gambar 15.4. Penggunaan carousel di berbagai platform

Berikut beberapa hal yang harus dilakukan:

- Dapat digunakan untuk memamerkan produk
- Dapat digunakan sebagai proses orientasi. Berikut beberapa hal yang tidak boleh dilakukan:
- Jangan mengirim banyak carousel sekaligus. Jika lebih dari sepuluh carousel telah dikirimkan ke pengguna, Anda perlu mengevaluasi pengambilan keputusan pengguna.

15.2 PESAN MULTIMEDIA

Beberapa platform chatbot mendukung pengiriman berbagai jenis pesan multimedia seperti gambar, video, file, dan sebagainya. Elemen-elemen ini dapat digunakan untuk mengirim gambar produk, PDF tagihan, dan sebagainya, untuk menciptakan pengalaman pengguna yang luar biasa.

Gambar-gambar

Gambar, terutama GIF, adalah cara yang luar biasa untuk meningkatkan keterlibatan dengan chatbot Anda. Dengan menggunakan sedikit animasi, chatbot dapat dengan mudah menyampaikan pesannya. GIF dapat digunakan secara inovatif. Banyak chatbot yang memasukkan beberapa GIF rekaman layar tentang cara menggunakan chatbot untuk memberikan gambaran umum. Ini adalah cara yang jauh lebih menarik untuk menjelaskan cara menggunakan chatbot daripada menampilkan beberapa teks mengenai penggunaan chatbots.

Video

Sebagian besar platform chatbot memungkinkan Anda mengirim video. Meskipun video tidak populer di komunitas, tidak ada yang bisa mengalahkan video menakjubkan untuk ikhtisar produk. Jika Anda ingin membuat chatbot untuk suatu produk, video pengantar dapat sangat membantu. Misalnya, saat pengguna menanyakan detail suatu produk, video yang menjelaskan berbagai keunikan dan fitur dapat dikirimkan.

File

Beberapa platform (Facebook Messenger, Viber, dan sebagainya) memungkinkan Anda mengirim jenis file tertentu. Untuk mengirimkan tanda terima dalam format PDF atau file khusus bisnis, ini bisa sangat berguna. Kami sebagian besar telah membahas semua jenis elemen UI yang akan Anda gunakan untuk membangun chatbot Anda. Sejujurnya, dalam sebagian besar kasus, elemen inilah yang akan Anda gunakan. Itu juga tergantung pada jenis chatbot yang Anda buat. Tidak ada batasan dalam menggunakan kombinasi berbeda dari elemen-elemen ini. Kami akan mendemonstrasikan beberapa kombinasi di bab ini.

Sekarang mari lanjutkan membangun chatbot yang Anda mulai di bab sebelumnya. Chatbot Anda saat ini memiliki dua tujuan.

- Pencarian lokasi
- Pencarian produk

Dalam kebanyakan kasus, orang memulai percakapan dengan salam, jadi mari tambahkan maksud salam ke proyek LUIS Anda. Nanti Anda akan mengintegrasikan maksud salam ke dalam chatbot Anda.

Berikut ini adalah contoh yang akan Anda tambahkan ke maksud salam Anda. (Maksud adalah maksud pengguna. Saat pesan diterima oleh chatbot, pesan tersebut akan dipetakan ke salah satu maksud dalam cakupan chatbot.)

- Hai
- Halo
- Halo

- Hai
- Ada apa
- Halo
- Hlw

Anda dapat menambahkan salam sebanyak yang Anda inginkan. Anda dapat mengikuti tutorial di bab sebelumnya untuk menambahkan maksud ke aplikasi LUIS Anda. Selanjutnya mari kita latih aplikasi untuk memperbarui model pembelajaran mesin. Setelah melatih model, Anda harus mempublikasikannya. Anda dapat mempublikasikan model terbaru Anda dengan menavigasi ke Publikasikan Aplikasi di panel kiri. Setelah Anda menambahkan maksud salam ke aplikasi LUIS dan melatihnya, Anda perlu mengintegrasikan fungsionalitas untuk merespons salam. Di file `app.js` Anda, perbarui fungsi bot Anda seperti ini:

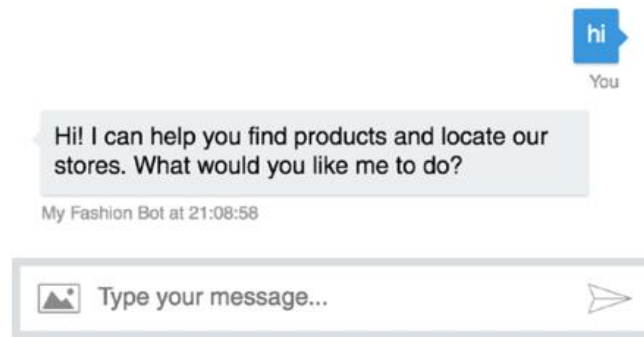
```
var bot = new builder.UniversalBot(connector, function (session) {
  getIntentFromLuis(session.message.text, function(error, luisData) {
    // Intent detected by LUIS
    var intent = luisData.topScoringIntent.intent;
    // Confidence score of the Intent detected
    var score = luisData.topScoringIntent.score;
    // Entities extracted by LUIS
    var entities = luisData.entities;
    // We are setting a threshold of 0.3 for the confidence score
    // If the confidence score is less that 0.3 we are considering that
    // the Chatbot has failed to understand the user message
    if (score > 0.3 && intent != 'None'){
      // Check if the user is looking for any product
      if (intent == 'product lookup'){
        // check if any entities are found
        if(entities.length > 0){
          var products = [];
          for (var productIterator in entities){
            products.push(entities[productIterator].entity);
          }
          var message = "Sure I will show you " + products.join(', ');
          session.send(message);
        }else{
          session.send("Sure I will show you all the products!");
        }
      }
      // Check if the user is looking for location to a store
    }else if (intent == 'location lookup'){
      // considering only first location
      if (entities.length > 0){
        var location = entities[0].entity;
        session.send("We have 10 stores across " + location);
      }else{
        session.send("We have 50 stores across the country.");
      }
    }
  }
});
```

```

    // Check if user has greeted the Chatbot
  } else if (intent == 'greetings') {
    session.send("Hi! I can help you find products and locate
    our stores. What would you like me to do?");
  }
} else {
  session.send("I did not understand you. I am still learning!
  Can you rephrase?");
}
});
});
});

```

Perhatikan bahwa sekarang chatbot Anda dapat merespons salam pengguna. Anda dapat menguji fitur ini dengan masuk ke akun Botframework Anda dan mengirimkan “hai” ke chatbot Anda. Anda akan menerima pesan seperti yang ditunjukkan pada Gambar 8-5.



Gambar 15.5. Satu tanggapan terhadap salam

15.3 MENAMPILKAN HASIL PRODUK

Katakanlah Anda ingin chatbot Anda menampilkan beberapa produk saat pengguna menanyakan produk. Untuk melakukan itu, Anda perlu menyimpan beberapa informasi produk tiruan yang dapat Anda cari dan ambil. Untuk chatbot ini, Anda akan menggunakan layanan bernama Algolia. Algolia adalah produk pencarian sebagai layanan. Hal ini memungkinkan Anda membuat API penelusuran yang tangguh dan canggih di sekitar data Anda. Untuk saat ini, Anda akan menggunakan layanan yang telah kami buat. Nanti di bab ini kami akan mendemonstrasikan cara membuat aplikasi Anda sendiri di Algolia. Jadi, buat file bernama `search.js` di direktori yang sama dengan file `app.js` Anda. File `search.js` ini akan berisi semua fungsi untuk mengambil produk Anda dari API pencarian. Di file `search.js` itu, buat fungsi bernama `searchByProduct`. Fungsi ini mengambil serangkaian nama produk sebagai atribut dan memanggil fungsi panggilan balik dengan serangkaian hasil pencarian. Seperti inilah tampilan file `search.js` Anda:

```

var request = require('request');

// This function returns products based on the categories
function searchByProduct(categories, callback){

    var options = {
        method: 'POST',
        url: 'http://7m299rv0cn.algolia.net/1/indexes/products/query',
        headers: {
            'x-algolia-application-id': '7M299RV0CN',
            'x-algolia-api-key': '5e3422fe5d9733c5dd8f6da9868f6f5c'
        },
        body: '{ "params": "query=' + categories.join(' ') + '&restrictSearchableAttributes=category" }'
    };

    request(options, function (error, response, body) {
        // Check for any error
        if (error){
            callback(error);
        }else{
            // Parse the text to JSON format
            var bodyJson = JSON.parse(body);
            callback(null, bodyJson.hits);
        }
    });
}

```

```

// This function fetches all the products
function showAllProducts(categories, callback){

    var options = {
        method: 'POST',
        url: 'http://7m299rv0cn.algolia.net/1/indexes/products/query',
        headers: {
            'x-algolia-application-id': '7M299RV0CN',
            'x-algolia-api-key': '5e3422fe5d9733c5dd8f6da9868f6f5c'
        },
        body: '{ "params": "query=&hitsPerPage=10" }'
    };

    request(options, function (error, response, body) {
        // Check for any error
        if (error){
            callback(error);
        }else{
            // Parse the respons to JSON
            var bodyJson = JSON.parse(body);
            callback(null, bodyJson.hits);
        }
    });
}

// Export these functions as modules
module.exports = {
    searchByProduct: searchByProduct,
    showAllProducts: showAllProducts
};

```

Pada dasarnya Anda telah membuat modul yang dapat Anda gunakan dari file JavaScript lainnya. Anda akan menggunakan modul pencarian ini di aplikasi Anda. Setiap kali Anda menemukan entitas produk dalam kueri pengguna, Anda akan menggunakan fungsi `searchByProduct` untuk mengambil produk. Anda juga dapat membuat fungsi Anda sendiri untuk mendapatkan informasi produk dari sumber lain (misalnya database, API, dan sebagainya). Dalam hal ini, Anda telah membuat toko e-commerce tiruan dan mengekspos API pencarian melalui Algolia. Karena tujuan utama buku ini adalah fokus murni pada chatbots, kami tidak akan mendalami topik tersebut lebih dalam. Pada fungsi sebelumnya, Anda memanggil API permintaan pencarian layanan Algolia untuk mencari inventaris toko untuk produk yang cocok. Mari integrasikan hasilnya ke dalam chatbot Anda.

Untuk memudahkan pemahaman, Anda akan membuat fungsi yang mengirimkan informasi produk ke pengguna terkait. Selain itu, fungsi kedua, `showAllProducts`, mengambil sepuluh produk pertama.

Pertama, Anda akan “memerlukan” file `search.js` Anda di bagian atas file `app.js` bersama dengan semua pernyataan `require` modul lainnya.

```
var search = require('./search');
```

Lalu, letakkan fungsi ini di bagian bawah file app.js Anda:

```
//This function sends the user information about the products in carousels
function sendProductInformation(session, products) {

    // Create a message object
    var message = new builder.Message(session);
    message.attachmentLayout(builder.AttachmentLayout.carousel);
    var cards = [];

    // For each product create a carousel element
    for (var productIterator = 0; productIterator < products.length;
    ++productIterator){
        var product = products[productIterator];

        // Create a carousel element (Knows as HeroCard in Botframework)
        var heroCard = new builder.HeroCard(session);

        // Add product name as the carousel title
        heroCard.title(product.product_name);
        // Add the product brand as the carousel subtitle
        heroCard.subtitle(product.brand);
        // Use the carousel text element for showing price
        heroCard.text("Price is " + product.price);
        // Add product image to the carousel by creating CardImage Object
        with the image URL
        heroCard.images([builder.CardImage.create(session, product.image)]);
        heroCard.buttons([builder.CardAction.imBack(session, "i want to buy
        " + product.category, "Buy")]);

        cards.push(heroCard);
    }

    message.attachments(cards);
    session.send(message);
}
```

Fungsi sebelumnya mengambil sesi pengguna dan produk sebagai argumen dan mengirimkan informasi produk kepada pengguna. Fungsi `session.send` mengambil objek pesan sebagai input. Objek pesan ini dapat berupa pesan teks biasa atau pesan UI kaya seperti carousel, tombol, atau lainnya. Dalam hal ini, Anda mengirimkan daftar elemen HeroCard, yang pada dasarnya adalah carousel. Sekarang mari kita jelaskan fungsinya baris demi baris.

```
var message = new builder.Message(session);
message.attachmentLayout(builder.AttachmentLayout.carousel);
```

Pada baris pertama, Anda meminta pembuat untuk membuat pesan kosong dan menyimpannya dalam variabel pesan. Di baris berikutnya, Anda menyetel tata letak pesan ke carousel dengan menyatakannya secara eksplisit sebagai argumen ke fungsi

message.attachmentLayout. Pada titik ini, mungkin tampak agak sulit untuk dipahami, tetapi percayalah bahwa Anda akan sering menggunakannya dan pada akhirnya akan terbiasa. Sekarang mari kita lihat perulangan for. Dalam perulangan for, Anda membuat setiap elemen HeroCard seperti ini:

```
var heroCard = new builder.HeroCard(session);
heroCard.title(product.product_name);
heroCard.subtitle(product.brand);
heroCard.text("Price is " + product.price);
heroCard.images([builder.CardImage.create(session, product.image)]);
heroCard.buttons([builder.CardAction.imBack(session, "i want to buy " +
product.category, "Buy")]);
```

Memanggil pembangun baru.HeroCard(sesi) menciptakan sebuah instance dari elemen HeroCard yang kosong. Anda perlu mengisi instance dengan informasi terkait. Di baris berikut, Anda mengisi elemen HeroCard dengan judul, subtitle, teks, gambar, dan tombol. Hal yang menarik di sini adalah pembuatan tombolnya. Di setiap kartu Anda akan meletakkan satu tombol untuk saat ini. Merupakan praktik terbaik untuk menggunakan tidak lebih dari tiga tombol per kartu. Dengan bertambahnya jumlah tombol, hal ini meningkatkan tingkat kebingungan, dan terdapat juga masalah kompatibilitas lintas platform. Facebook Messenger tidak mengizinkan Anda menambahkan lebih dari tiga tombol per elemen carousel. Di chatbot ini, Anda juga akan menggunakan tiga tombol per kartu. Sekarang Botframework mendukung pembuatan berbagai jenis tombol. Untuk saat ini, Anda akan menggunakan tombol builder.CardAction.imBack. Perhatikan argumen konstruktor builder.CardAction.imBack. Argumen kedua pada dasarnya adalah string yang akan dikirim sebagai pesan teks, dan argumen ketiga adalah judul tombol.

```
message.attachments(cards);
session.send(message);
```

Di baris sebelumnya, Anda melampirkan kartu ke pesan Anda dan mengirimkannya ke pengguna. Sekarang ubah fungsi bot di file app.js tempat maksud pencarian produk ditangani seperti ini:

```

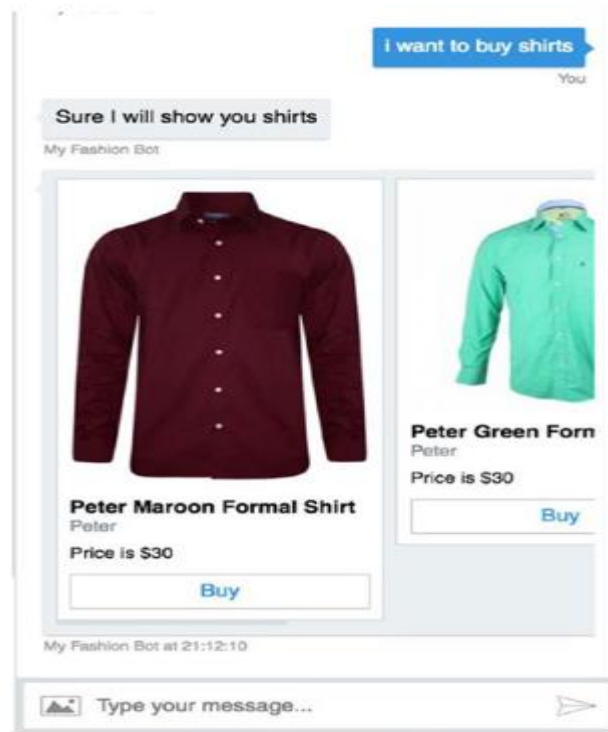
if (intent == 'product lookup'){
  if(entities.length > 0){
    var products = [];
    for (var productIterator in entities){
      products.push(entities[productIterator].entity);
    }

    session.send("Sure I will show you " + products.join(', '));
    search.searchByProduct(products, function (error, productResult) {
      sendProductInformation(session, productResult);
    });
  }else{
    session.send("Sure I will show you all the products!");

    search.showAllProducts(products, function (error, productResult) {
      sendProductInformation(session, productResult);
    });
  }
}

```

Setiap kali maksud pencarian produk terdeteksi, Anda memeriksa apakah entitas terkait terdeteksi. Jika suatu entitas ditemukan, Anda mengirim pesan yang mengakui fakta bahwa Anda telah memahami maksud pengguna, dan kemudian Anda mengambil produk terkait menggunakan fungsi modul pencarian Anda, `searchByProduct`. Ketika Anda tidak dapat menemukan produk untuk dicari (alias ketika tidak ada entitas yang ditemukan dalam permintaan pengguna), Anda masih menampilkan sepuluh produk dari inventaris Anda kepada pengguna (lihat Gambar 8-6). Sekarang mari kita uji chatbot Anda.



Gambar 15.6. Chatbot menanggapi pertanyaan produk

Chatbot Anda sekarang memiliki kecerdasan dan integrasi dengan sistem Anda. Masih ada satu topik penting yang kurang; itu tidak dapat menyimpan informasi tentang pengguna mana pun. Saat Anda membuat chatbot mode, orang-orang akan memfilter produk berdasarkan warna. Sekarang Anda akan menambahkan fungsionalitas ke chatbot Anda sehingga dapat mengingat produk dan warna apa yang dicari pengguna. Pada dasarnya, Anda akan memiliki chatbot untuk menyimpan informasi yang peka konteks.

Mari buat chatbot Anda mengingat produk apa yang dicari pengguna Anda. Untuk melakukan itu, Anda akan menggunakan fitur bawaan Botframework. Botframework memungkinkan Anda menyimpan data khusus percakapan. Di file `app.js` Anda, mari tambahkan kode ini tepat sebelum Anda memanggil fungsi pencarian Anda `searchByProduct`.

```
session.conversationData.products = products;
session.save();
```

Sekarang Botframework akan mengingat produk apa yang dicari pengguna. Selanjutnya, Anda akan memberikan fleksibilitas kepada pengguna untuk memfilter hasil dengan preferensi warna mereka, menjaga produk tetap dalam konteks. Mari dengan cepat membuat maksud filter warna dan entitas warna di aplikasi LUIS Anda. Kemudian mari tambahkan contoh berikut ke maksud filter warna Anda, dengan memberi tag entitas warna di masing-masing sampel.

- Saya ingin yang berwarna hitam.
- Tunjukkan padaku sesuatu yang berwarna merah.

- Apakah ini berwarna hijau?
- Saring berdasarkan warna biru.
- Saya lebih suka warna abu-abu.
- Bisakah saya mendapatkannya dalam warna merah marun?

Setelah Anda selesai menambahkan sampel ke maksud Anda, Anda akan melatih dan mempublikasikan aplikasi LUIS Anda, sehingga membuat maksud baru tersedia untuk aplikasi Anda. Selanjutnya, mari kita integrasikan fungsionalitas untuk mengatasi maksud baru. Tambahkan fungsi baru ini ke file search.js Anda:

```
function searchByProductFilterByColor(categories, colors, callback){
    for (var i=0; i< colors.length; ++i){
        colors[i] = 'color:'+colors[i]
    }

    var filterQuery = colors.join(' OR ');

    var options = {
        method: 'POST',
        url: 'http://7m299rv0cn.algolia.net/1/indexes/products/query',
        headers: {
            'x-algolia-application-id': '7M299RV0CN',
            'x-algolia-api-key': '5e3422fe5d9733c5dd8f6da9868f6f5c'
        },
        body: '{ "params": "query=' + categories.join(' ') + '&filters='+
filterQuery + '&restrictSearchableAttributes=category" }'
    };

    request(options, function (error, response, body) {
        if (error){
            callback(error);
        }else{
            var bodyJson = JSON.parse(body);
            callback(null, bodyJson.hits);
        }
    });
}
```

Anda juga harus menambahkan fungsi ini ke modul, seperti yang ditunjukkan di sini:

```
module.exports = {
    ...
    searchByProductFilterByColor:searchByProductFilterByColor
};
```

Mari ubah juga fungsi bot Anda di file app.js untuk menangani maksud filter warna.

```

...
} else if (intent == 'greetings') {
    session.send("Hi! I can help you find products and locateour stores.
    What would you like me to do?");
} else if (intent == 'color filter') {

    var colors = [];
    for (var colorIterator in entities){
        if (entities[colorIterator].type == 'color'){
            colors.push(entities[colorIterator].entity);
        }
    }
    var productsFromContext = session.conversationData.products;

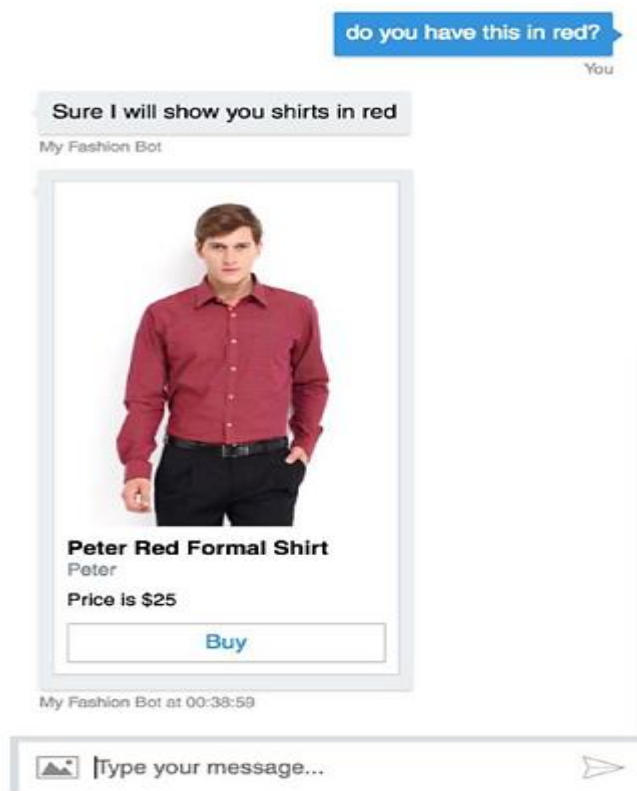
    session.send("Sure I will show you " +
        productsFromContext.join(', ') +
        ' in ' +
        colors.join(' '));

    search.searchByProductFilterByColor(
        productsFromContext,
        colors,
        function (error, productResult) {
            sendProductInformation(session, productResult);
        }
    );
}
...

```

Pada kode sebelumnya, pada dasarnya Anda mencari entitas warna ketika maksud terdeteksi. Kemudian Anda mencari katalog produk Anda melalui API pencarian yang Anda integrasikan dalam fungsi `searchByProductFilterByColor` di modul pencarian. Ayo uji chatbot Anda dengan konteks yang terintegrasi.

Sekarang chatbot Anda dapat memahami pertanyaan terkait produk dan dapat memfilter hasilnya berdasarkan warna, Anda dapat menambahkan sedikit kecerdasan ke dalamnya. Ketika daftar produk ditampilkan, Anda juga dapat menyarankan beberapa warna kepada pengguna Anda (lihat Gambar 8-7). Namun ada kendala di sini: di luar kotak, Botframework tidak mengizinkan Anda mengirim tombol. Jadi, Anda akan menggunakan elemen HeroCard tanpa gambar untuk mengirimkan saran warna ini. Namun tombol-tombol ini tidak akan dimuat di beberapa platform, sehingga mengganggu pengalaman pengguna di beberapa platform. Untuk menghindari hal tersebut, Anda dapat mengirimkan payload khusus platform. Misalnya, Facebook Messenger memungkinkan Anda mengirim balasan cepat yang indah, alih-alih mengirim elemen HeroCard. Untuk Facebook Messenger, Anda akan mengirim balasan cepat.



Gambar 15.7. Chatbot membalas pertanyaan terkait warna

Pertama, Anda akan menambahkan fungsionalitas untuk mengirimkan saran warna kepada pengguna. Untuk melakukannya, mari tulis fungsi bernama `sendColorSuggestion` seperti ini:

```
function sendColorSuggestion(session, products){
  var colors = [];

  for (var productIterator = 0; productIterator < products.length;
  ++productIterator) {
    var product = products[productIterator];
    if (colors.indexOf(color) == -1){
      colors.push(product.color);
    }
  }
}
```

```

var message = new builder.Message(session);
message.attachmentLayout(builder.AttachmentLayout.carousel);

var heroCard = new builder.HeroCard(session);
var buttons = [];

for (var colorIterator = 0; colorIterator < colors.length; ++colorIterator) {
    var color = colors[colorIterator];

    // To make sure the buttons are not duplicated
    var button_text = "Do you have this in " + color;
    var button = builder.CardAction.imBack(session, button_text, color);

    buttons.push(button);
}

heroCard.title('Here are some colors I suggest.');
```

```

heroCard.buttons(buttons);

message.attachments([heroCard]);
session.send(message);
}

```

Fungsi sebelumnya mengambil array sesi dan produk sebagai argumen dan membuat array baru yang disebut warna dari array produk dan kemudian mengirimkan elemen HeroCard hanya dengan judul dan tombol di dalamnya. Mari kita panggil fungsi ini ketika maksud pencarian produk dipicu.

```

search.searchByProduct(products, function (error, productResult) {
    sendProductInformation(session, productResult);
    sendColorSuggestion(session, productResult);
});

```

Mari tambahkan muatan khusus Facebook Messenger untuk mengirim balasan cepat. Anda akan membuat fungsi lain yang disebut `sendColorSuggestionFB`, yang akan Anda gunakan untuk mengirim balasan cepat saran warna ke pengguna Facebook Messenger.

```

function sendColorSuggestionFB(session, products){
    var colors = [];

    for (var productIterator = 0; productIterator < products.length;
    ++productIterator) {
        var product = products[productIterator];
        colors.push(product.color);
    }
}

```

```

var message = new builder.Message(session);
var quickReplies = [];

// Limit the maximum number of quick replies to 10 (10 is the limit set
by messenger
for (var colorIteraror = 0; colorIteraror < 10 && colorIterator
< colors.length; ++colorIteraror) {
    var color = colors[colorIteraror];

    var quickReply = {
        "content_type": "text",
        "title": color,
        "payload": "do you have this in " + color
    }
    quicReplies.push(quickReply)
}

message.sourceEvent({
    facebook: {
        text: 'Here are some colors I suggest.',
        quick_replies: quicReplies
    }
})

session.send(message);
}

```

Selain itu, Anda harus mengubah logika chatbot Anda untuk memeriksa sumber facebook dan mengirim balasan cepat yang sesuai. Sumber pesan dapat diakses sebagai sesi.pesan.sumber. Mari kita ubah logika chatbot sebagai berikut:

```

...
search.searchByProduct(products, function (error, productResult) {
    sendProductInformation(session, productResult);

    if (session.message.source == 'facebook'){
        sendColorSuggestionFB(session, productResult);
    }else{
        sendColorSuggestion(session, productResult);
    }
});
...

```

Mari selami kodenya lebih dalam. Jika sumbernya adalah facebook, Anda akan memanggil fungsi sendColorSuggestionFB dengan sesi terkait dan array productResult. Di dalam fungsi tersebut Anda membuat array quickReplies dengan mengikuti dokumentasi platform Facebook Messenger. Contoh objek balasan cepat terlihat seperti berikut:

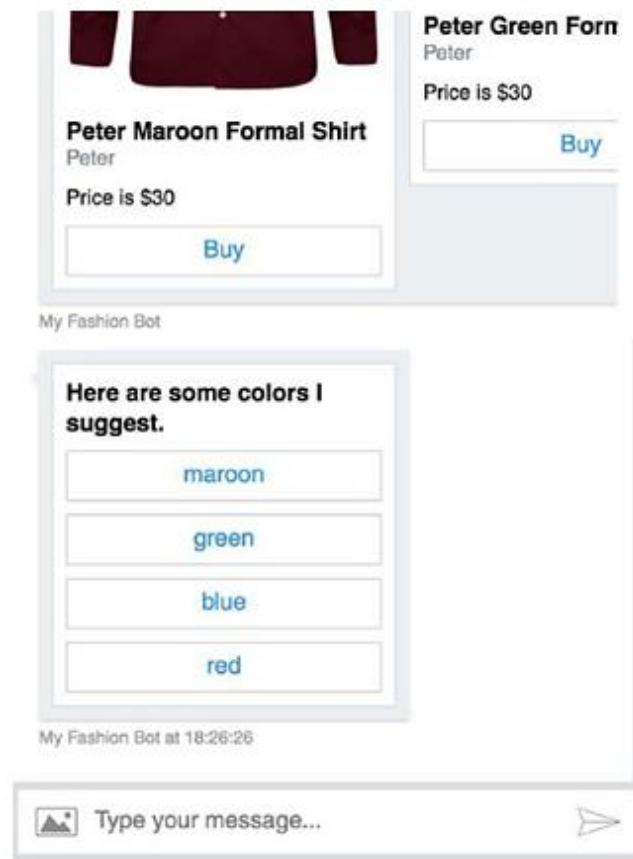
```
{
  "content_type": "text",
  "title": "Search",
  "payload": "<POSTBACK_PAYLOAD>",
  "image_url": "http://example.com/img/red.png"
}
```

Judul adalah apa yang akan terlihat oleh pengguna pada tombol. Ketika tombol diklik, Anda akan menerima payload sebagai postback. Jadi, untuk setiap tombol balasan cepat berwarna, Anda menyetel payload balasan cepat menjadi “Apakah Anda memiliki ini di <warna>?” Setiap kali pengguna mengklik salah satu balasan cepat, Anda mendapatkan muatan ini sebagai pesan teks biasa.

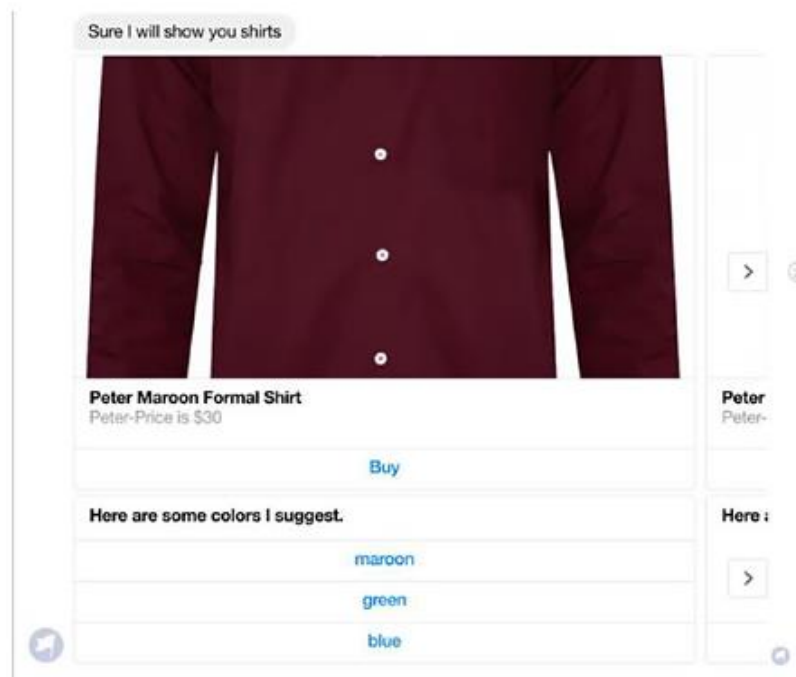
Sekarang kami akan menjelaskan cara mengirim objek khusus ini ke pengguna Facebook. Botframework memungkinkan Anda mengirim data khusus platform khusus saat Anda mengirim pesan. Anda dapat melakukannya menggunakan fungsi `message.sourceEvent`. Fungsi ini memungkinkan Anda menyetel objek khusus untuk platform tertentu dalam format tertentu, seperti yang ditunjukkan di sini:

```
{
  facebook: {
    ...
  }
}
```

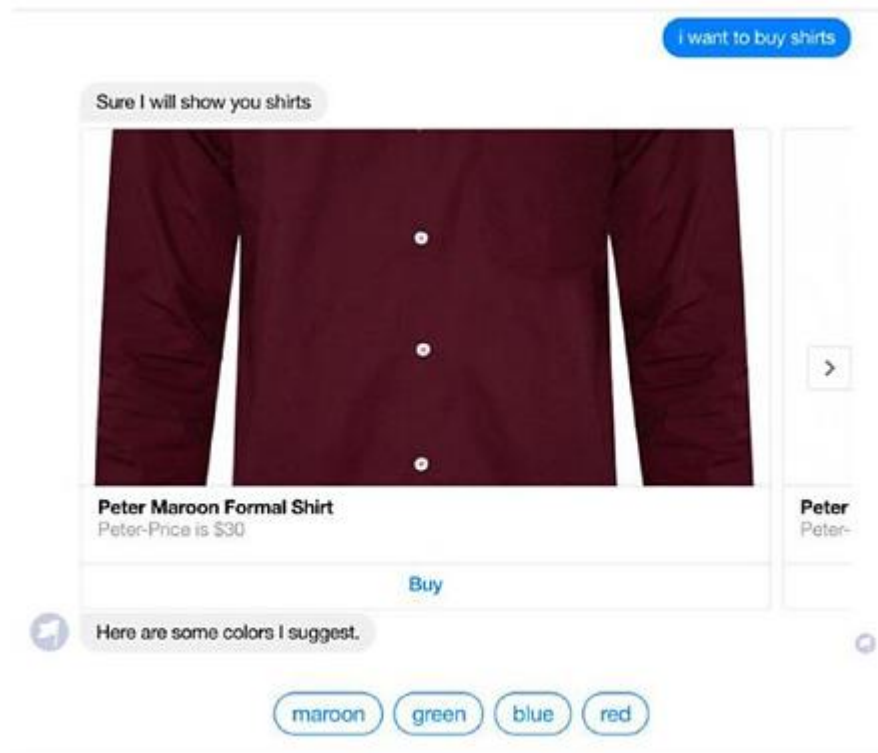
Seperti yang ditunjukkan sebelumnya, Anda dapat mengirim payload khusus untuk meningkatkan UX untuk masing-masing platform. Kami sangat menyarankan untuk mencoba-coba fungsi ini untuk memberikan pengalaman pengguna terbaik untuk chatbot Anda kepada pengguna (lihat Gambar 8-8, Gambar 8-9, dan Gambar 8-10).



Gambar 15.8. Chatbot memberikan saran warna



Gambar 15.9. Saran warna di Messenger sebelum menambahkan balasan cepat melalui sourceEvent



Gambar 15.10. Saran warna di Messenger setelah menambahkan balasan cepat

15.4 MENGINTEGRASIKAN MAKSUD PENCARIAN LOKASI

Anda selesai mengintegrasikan maksud produk. Sekarang Anda akan mengintegrasikan maksud pencarian lokasi yang Anda buat di bab sebelumnya dengan data lokasi penyimpanan yang Anda miliki. Kami telah membuat API yang memungkinkan Anda mencari toko yang tersedia di suatu kota. Anda akan menggunakan API itu untuk mengambil toko yang dicari pengguna Anda dan mengirimkan informasinya. Sekali lagi, ini hanyalah API demo; Anda selalu dapat menghubungkan sumber data Anda sendiri seperti dari database atau API apa pun ke chatbot Anda. Seperti biasa, Anda akan membuat fungsi pencarian di file `search.js` Anda untuk mencari.

```

function searchStoreByCity(cities, callback){

  var options = {
    method: 'POST',
    url: 'http://7m299rv0cn.algolia.net/1/indexes/store_locations/query',
    headers: {
      'x-algolia-application-id': '7M299RV0CN',
      'x-algolia-api-key': '5e3422fe5d9733c5dd8f6da9868f6f5c'
    },
    body: '{ "params": "query=' + cities.join(' ') + '&restrictSearchableAttributes=city" }'
  };

  request(options, function (error, response, body) {
    if (error){
      callback(error);
    }else{
      var bodyJson = JSON.parse(body);
      callback(null, bodyJson.hits);
    }
  });
}

```

API mengembalikan serangkaian objek yang sesuai dengan lokasi toko di suatu kota. Juga, mari tambahkan fungsi ini ke modul dengan fungsi lainnya.

```

module.exports = {
  ...
  searchStoreByCity:searchStoreByCity
};

```

Dengan cara ini, fungsi `searchStoreByCity` Anda akan dapat diakses oleh semua modul lain dalam aplikasi Anda. Mari kita panggil fungsi tersebut setiap kali maksud pencarian lokasi dipicu dan entitas lokasi ditemukan. Sekarang Anda akan membuat perubahan pada logika chatbot Anda di file `app.js`. Mari buat fungsi yang mengambil daftar entitas kota yang ditemukan dalam kueri pengguna, memanggil API Anda, lalu mengirimkan daftar elemen HeroCard ke masing-masing pengguna.

```

function sendStoreList(session, cities){

    var message = new builder.Message(session);
    message.attachmentLayout(builder.AttachmentLayout.carousel);
    var cards = [];

    search.searchStoreByCity(cities, function(err, stores) {
        if (stores.length > 0){
            session.send('Sure we have got stores in '+ cities.join(', '));
            for (var storeIterator=0; storeIterator < stores.length;
                ++storeIterator){

                var store = stores[storeIterator];

                var heroCard = new builder.HeroCard(session);

                heroCard.title(store.city);
                heroCard.text(store.street);

                cards.push(heroCard);
            }
            message.attachments(cards);
            session.send(message);

        }else{
            session.send('Sorry we do not have any stores in '+ cities.
                join(', '));
        }
    });
}

```

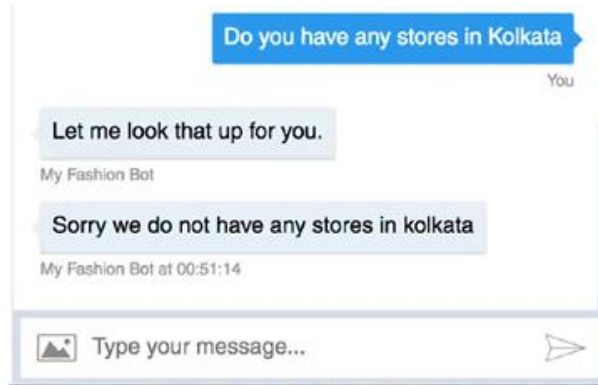
Fungsi sebelumnya cukup mudah. Dibutuhkan sesi pengguna saat ini dan daftar kota sebagai argumen, membuat elemen HeroCard untuk setiap lokasi dengan detail toko (alamat jalan dan nama kota), dan mengirimkannya kembali ke pengguna.

Pada Gambar 15.11, chatbot membalas pengguna dengan informasi yang relevan tentang lokasi toko bisnis yang berbeda.



Gambar 15.11. Chatbot merespons pertanyaan terkait lokasi toko

Pada Gambar 15.12, Anda dapat melihat bahwa jika tidak ada toko yang ditemukan untuk lokasi yang diminta oleh pengguna, chatbot akan kembali dengan nada meminta maaf.



Gambar 15.12. Chatbot merespons tidak ada toko yang ditemukan

Selanjutnya, Anda akan menambahkan fungsi lain yang disebut `sendCitySuggestions` yang mengirimkan daftar kota terpopuler dengan toko terpopuler. Dengan fungsi ini, meskipun chatbot Anda tidak dapat mengidentifikasi kota tempat pengguna mencari toko, Anda masih dapat memberi mereka opsi untuk menelusuri daftar kota populer. Mari kita terapkan fungsi berikut:

```

function sendCitySuggestions(session) {
    session.send('No city found in your query to look up our stores! But
    these are the cities where our stores are most popular.');
```

```

    var cityList = [
        "London",
        "Bangkok",
        "Singapore",
        "New York",
        "Kuala Lumpur",
        "Hong Kong",
        "Dubai"
    ]

    var message = new builder.Message(session);
    message.attachmentLayout(builder.AttachmentLayout.carousel);

    var heroCard = new builder.HeroCard(session);
    var buttons = [];

    for (var cityIterator = 0; cityIterator < cityList.length;
    ++cityIterator) {
        var city = cityList[cityIterator];

        // To make sure the buttons are not duplicated
        var button_text = "Do you have stores in " + city + '?';
        var button = builder.CardAction.imBack(session, button_text, city);
        buttons.push(button);
    }

    heroCard.title('Cities');
    heroCard.buttons(buttons);

    message.attachments([heroCard]);
    session.send(message);
}

```

Hebat, sekarang Anda memiliki semua alat untuk menangani permintaan pengguna untuk pencarian lokasi. Ubah logika chatbot untuk maksud pencarian lokasi sebagai berikut:

```

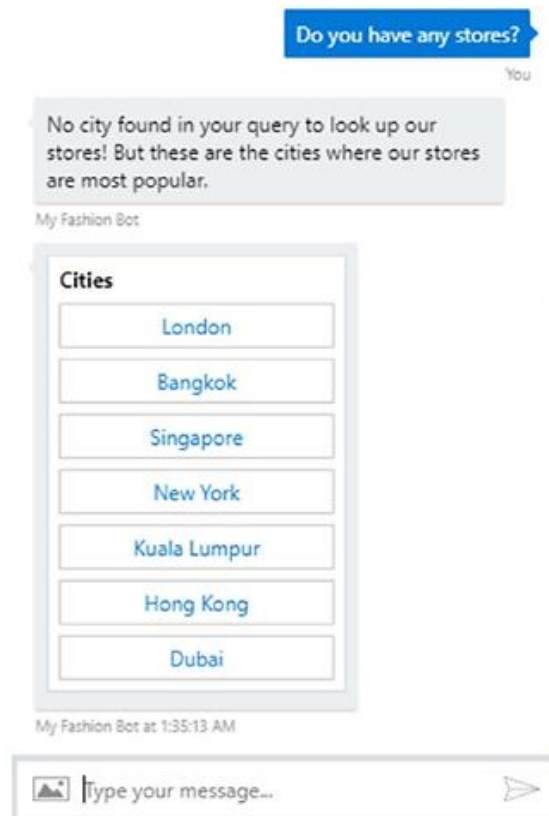
...
if (intent == 'location lookup') {
  // considering only first location
  var cities = [];

  if (entities.length > 0) {
    for (var locationIterator = 0; locationIterator < entities.length;
        ++locationIterator) {
      var entityObject = entities[locationIterator];
      if (entityObject.type == 'builtin.geography.city') {
        cities.push(entityObject.entity);
      }
    }
  }

  if (cities.length > 0) {
    session.send('Let me look that up for you.')
    sendStoreList(session, cities);
  } else {
    sendCitySuggestions(session);
  }
}
...

```

Dengan ini, Anda selesai mengimplementasikan chatbot cerdas end-to-end pertama Anda yang dapat menampilkan saran produk dengan manajemen konteks dan juga dapat memberikan informasi tentang toko yang tersedia di kota tertentu. Jangan ragu untuk mengotori tangan Anda dengan semua fitur yang tidak dapat dibahas di halaman buku ini. Chatbots memiliki potensi yang tak terbatas untuk mengganggu banyak solusi yang ada karena inovasi terkini, pengalaman pengguna yang luar biasa, aksesibilitas, kemudahan penggunaan, dan banyak fitur lainnya (lihat Gambar 15.13).



Gambar 15.13 informasi yang muncul toko pada negara lain

Ada beberapa peringatan dalam penerapan chatbot ini. Kami akan menyebutkannya secara singkat, Anda harus mencoba memperbaikinya sendiri, yang akan membantu Anda mendapatkan pemahaman lebih dalam tentang cara kerja semuanya.

15.5 MENYIAPKAN PESAN

Anda membuat chatbot lengkap di bagian sebelumnya, yang bukan hanya bot berbasis tombol tetapi merupakan chatbot lengkap yang memahami bahasa alami. Ini memberi pengguna kebebasan untuk mengetik apa yang ada di pikiran mereka tanpa terlalu khawatir. Anda juga meluncurkan chatbot di berbagai saluran dan melihat bagaimana pola UX/UI setiap saluran berguna saat mendesain chatbot Anda. Salah satu aspek terpenting dari chatbot yang diterapkan sepenuhnya adalah analitik yang disediakan.

Nah, Anda akan menggunakan instance MongoDB Anda dan menyimpan pesan yang dikirim oleh pengguna dan pesan yang dikirim oleh bot sebagai respons. Setelah Anda menyimpan semua interaksi antara pengguna dan chatbot, Anda kemudian dapat memasukkan analitik ke dalam modul UI dan memvisualisasikannya. Untuk cakupan chatbot dalam buku ini, Anda akan membatasi diri Anda sendiri untuk hanya menyimpan data. Dalam kode yang ada di GitHub, Anda akan menambahkan API untuk mengeksport data dalam format JSON yang kemudian dapat digunakan oleh aplikasi UI Anda untuk visualisasi.

Catatan Kami telah menggunakan MongoDB untuk penyimpanan pesan dalam buku ini, namun Anda bebas menggunakan database pilihan Anda. Pilihan kami didasarkan pada kemudahan pengaturan dan kompatibilitas dengan Node.js.

Menjadi Luwak

MongoDB adalah database NoSQL yang berdiri sendiri yang memiliki banyak ODM yang dikembangkan oleh para penggemar open source. Untuk menghubungkan back end contoh aplikasi bot ke MongoDB, kami akan menunjukkan cara menggunakan Mongoose ODM, yang dirilis oleh orang baik Automattic (perusahaan di balik WordPress). Mari lanjutkan dan instal Mongoose di aplikasi Anda.

```
$ cd <path_to_your_application>
$ npm install --save mongoose
```

Anda dapat menginstal Mongoose menggunakan modul NPM, yang telah kita bahas di Bab 2. Ketergantungan ditambahkan ke file package.json Anda, yang akan berguna saat Anda mendorong bot Anda ke produksi.

Membangun Model Pesan

MongoDB adalah database berorientasi dokumen, yang memberi Anda fleksibilitas untuk menyimpan objek JavaScript apa pun sebagaimana adanya di database. Mongoose menyediakan API yang terdefinisi dengan baik untuk mencari data yang disimpan dengan kemampuan untuk menerapkan filter. Mari lanjutkan dan tentukan bidang yang perlu Anda simpan di database.

- Sumber: Sumber pesan (Facebook, Slack, Skype, dll.)
- Dari: Siapa yang mengirim pesan (ID unik orang/bot)
- Kepada: Kepada siapa pesan dikirimkan (ID unik orang/bot)
- Payload: Payload yang diterima dari pesan masuk/keluar untuk diperiksa jenis pesan dan lampirannya
- Waktu Pembuatan: Kemampuan untuk memahami pola saat pengguna berinteraksi dengan chatbot

Menambahkan File Model

Merupakan praktik terbaik untuk memodulasi kode dalam beberapa file, dengan setiap file hanya menangani satu fungsi luas. Seperti yang Anda lakukan saat membuat search.js, lanjutkan dan buat file mode.js di editor teks favorit Anda di direktori root proyek Anda. Hal pertama yang perlu Anda lakukan adalah mengimpor modul Mongoose, yang Anda instal di bagian sebelumnya, dan mengkonfigurasinya untuk menggunakan instalasi lokal database MongoDB.

Model.js file

```
var mongoose = require ('mongoose');
var schema = mongoose.Schema;
mongoose.connect ('mongodb://localhost:27017/DB_NAME');
```

Sekarang aplikasi Anda dapat terhubung ke database MongoDB menggunakan Mongoose ODM. Sebagai pengganti DB_NAME, berikan nama apa pun untuk database. Pastikan nama database yang diberikan sudah dibuat menggunakan klien Mongo pada baris perintah. Sekarang mari kita tentukan skema pesan Anda, yang akan Anda simpan sebagai koleksi di database MongoDB. (Setiap objek dalam database disebut koleksi di MongoDB.) Tambahkan kode berikut dalam file mode.js setelah penanganan koneksi sebelumnya:

```
var BOT_NAME = 'shop_assistant';
var messageSchema = new mongoose.Schema({
  from: String,
  to: String,
  createTime: Date,
  source: String,
  payloadObject: schema.Types.Mixed
});
```

Anda akan menyadari bahwa Anda telah membuat instance objek baru dari tipe Skema dan telah memberi tahu Mongoose bagaimana tampilan objek koleksi Anda. Merupakan praktik yang baik untuk memberi tahu Mongoose terlebih dahulu tentang jenis variabel yang akan Anda simpan; ini juga membantu pengembang lain membaca kode Anda untuk memahami apa yang disimpan setiap bidang dalam koleksi. Skema jenis Campuran pada dasarnya memberi tahu Mongoose bahwa Anda akan menyimpan objek JavaScript (JSON) di lapangan. String dan Tanggal adalah tipe JavaScript primitif yang disediakan langsung oleh lingkungan Node. Sekarang mari kita lanjutkan dan tentukan model pesan Anda untuk Mongoose.

```
var messageModel = mongoose.model ('Messages', messageSchema);
```

Skema dan model didefinisikan, yang saling terkait. Sekarang mari kita tentukan dua fungsi yang akan diekspor ke app.js. Fungsi pertama adalah saveSendMessage, yang akan dipanggil ketika bot merespons permintaan pengguna. Fungsi saveSendMessage mengambil satu argumen, yang diberi nama payload. Ini adalah objek yang dihasilkan oleh Botbuilder. Fungsi kedua adalah saveIncomingMessage, yang dipanggil setiap kali pengguna mengirim pesan ke bot; saveIncomingMessage juga mengambil satu argumen yang disebut payload, yang disediakan oleh Botbuilder.

```

function saveSendMessage (payload) {
  var sentmessage = new messageModel ({
    from: BOT_NAME,
    to: payload.address.user.id,
    createTime: new Date(),
    source: payload.source,
    payloadObject: payload
  });
  sentMessage.save (function(err){
    if (err) {
      console.log (`Error saving message: Message to
        ${payload.user.id}`);
    } else {
      console.log (`Message saved successfully`);
    }
  });
}

```

Anda telah melakukan hardcode pada variabel BOT_NAME di bagian atas file. Ini akan membantu Anda menanyakan database dengan mudah berdasarkan BOT_NAME di bidang Kepada atau Dari. Mari tambahkan implementasi fungsi saveIncomingMessage.

```

function saveIncomingMessage (payload) {
  var incomingMessage = new messageModel ({
    from: payload.address.user.id,
    to: BOT_NAME,
    createTime: new Date(),
    source: payload.source,
    payloadObject: payload
  });
  incomingMessage.save(function (err) {
    if (err) {
      console.log (`Error saving message: Message from
        ${payload.user.id}`);
    } else {
      console.log (`Message saved successfully`);
    }
  });
}

```

Anda telah mengimplementasikan model Anda, dan sekarang Anda perlu mengekspor dua fungsi yang dapat digunakan oleh app.js.

```

module.exports = {
  saveSentMessage: saveSentMessage,
  saveIncomingMessage: saveIncomingMessage
};

```

Mengintegrasikan Model ke dalam Aplikasi

Karena Anda telah memodulasi aplikasi Anda menjadi beberapa file, Anda perlu mengimpor file model yang baru saja Anda buat di app.js. Untuk melakukannya, tambahkan cuplikan kode berikut setelah pernyataan persyaratan awal Anda. Sorotan tebal menunjukkan baris yang ditambahkan ke app.js sebelumnya:

```

var restify = require ('restify');
var builder = require ('builder');
var request = require ('request');
var search = require ('./search');
var model = require('./model');

```

Selanjutnya Anda perlu menambahkan dua event pendengar; satu pendengar peristiwa harus dipicu setiap kali Anda menerima pesan dari pengguna mana pun ke bot Anda, dan pendengar peristiwa kedua harus dipicu setiap kali bot mengirim pesan ke pengguna. Botbuilder memunculkan dua peristiwa ini secara otomatis; Anda hanya perlu menambahkan fungsi handler untuk fungsi-fungsi ini. Di app.js, Anda akan menambahkan dua event listening di akhir file, pastikan Anda menambahkan kode di app.js.

```

bot.on ('incoming', function(data) {
  model.saveIncomingMessage (data);
  console.log ('-incoming message-');
});

bot.on('outgoing', function(data) {
  model.saveSentMessage (data);
  console.log ('-outgoing message-');
});

```

bot memancarkan dua peristiwa bernama masuk dan keluar pada setiap pesan yang diterima atau dikirim. Anda telah menambahkan pendengar acara yang memanggil fungsi yang telah Anda tetapkan di model.js untuk menyimpan pesan yang sesuai. Anda mungkin bertanya-tanya seperti apa data untuk setiap kasus. Nah, Anda harus mencatat datanya untuk melihat seperti apa. Berikut data pesan masuk. Anda harus mencatat data pada output standar menggunakan console.log untuk melihat tampilan payload saat Anda mengirim pesan.

```

{
  "type": "message",
  "timestamp": "2017-08-28T06:03:35.9092036Z",
  "textFormat": "plain",
  "text": "HEy",
  "textLocale": "en-GB",
  "sourceEvent": {
    "clientActivityId": "1503900130494.4429384200782791.8"
  },
  "attachments": [],
  "entities": [],
  "address": {
    "id": "1deef1cc612b4bc181c6449a94a52417|0000007",
    "channelId": "webchat",
    "user": {
      "id": "HtFISdImqn",
      "name": "You"
    },
  },
  "conversation": {
    "id": "1deef1cc612b4bc181c6449a94a52417"
  },
  "bot": {
    "id": "cbbooktest@qVEjtpz7vWc",
    "name": "chatbot-book-test"
  },
  "serviceUrl": "https://webchat.botframework.com/"
},
"source": "webchat",
"agent": "botbuilder",
"user": {
  "id": "HtFISdImqn",
  "name": "You"
}
}

```

Sekarang Anda memiliki infrastruktur untuk mendukung bot dan kemampuan melacak pesan untuk analisis lebih lanjut dan memahami pola penggunaan. Anda dapat menambahkan fungsionalitas Anda sendiri ke bot dengan menentukan maksud dan entitas Anda sendiri di LUIS.ai dan cukup masukkan kode logika bisnis di app.js.

15.6 MEMBANGUN PENGKLASIFIKASI

Di bagian sebelumnya, Anda membuat chatbot dan menggunakan LUIS.ai untuk menangani kueri bahasa alami yang disajikan oleh pengguna. LUIS.ai menyediakan pengawas abstraksi serangkaian algoritme pembelajaran mesin untuk mendukung kueri Anda. Di bagian ini, Anda akan membuat pengklasifikasi maksud kecil yang dapat dipasang ke chatbot apa pun. Mengembangkan pengklasifikasi maksud akan membantu Anda memahami cara kerja LUIS.ai atau mesin NLP apa pun dan akan membantu Anda melatih bot Anda dengan lebih baik. Gagasan kami bukan untuk menggantikan LUIS.ai tetapi untuk melengkapi penggunaan LUIS.ai dengan membahas cara membuat pengklasifikasi maksud.

Agar penyiapan tetap sederhana dan fokus pada pembuatan pengklasifikasi maksud, Anda akan memulai proyek baru dan melakukan pengembangan dari awal. Anda bebas mengintegrasikan pengklasifikasi ke dalam bot yang Anda buat di bagian sebelumnya.

Apa Itu Pengklasifikasi?

Pengklasifikasi adalah program komputer yang dapat mengklasifikasikan masukan tertentu ke dalam keranjang yang sudah ada/yang telah ditentukan sebelumnya. Wikipedia mendefinisikan pengklasifikasi sebagai berikut:

Algoritma yang mengimplementasikan klasifikasi, terutama dalam implementasi konkrit disebut dengan classifier. Istilah “pengklasifikasi” terkadang juga mengacu pada fungsi matematika, yang diterapkan oleh algoritma klasifikasi, yang memetakan data masukan ke suatu kategori.

Anda akan membangun pengklasifikasi dengan mengikuti model statistik yang telah terbukti yang disebut algoritma naïve Bayes. Untuk melatih pengklasifikasi, Anda memerlukan kumpulan data berlabel. Untuk contoh ini, data yang diberi label harus dalam format berikut:

`<utterance>,<label>`

Sebelum masuk ke implementasi dan penggunaan kode, Anda memerlukan pemahaman tentang algoritma naïve Bayes. Algoritma naïve Bayes adalah pengklasifikasi probabilistik sederhana berdasarkan penerapan teorema Bayes dengan asumsi independensi yang kuat antar fitur. Ini adalah metode populer untuk kategorisasi teks, dan klasifikasi maksud sangat cocok untuk masalah yang dapat diselesaikan dengan algoritma naïve Bayes. Algoritme ini diperkenalkan dengan nama berbeda ke dalam komunitas pengambilan teks pada awal tahun 1960an.

Teorema Bayes dinamai Thomas Bayes, dan bekerja berdasarkan probabilitas bersyarat. Probabilitas bersyarat adalah probabilitas bahwa sesuatu akan terjadi jika sesuatu yang lain telah terjadi. Anda mencoba menggunakan pengetahuan sebelumnya untuk memprediksi kemungkinan kejadian saat ini. Berikut rumus menghitung probabilitas bersyarat:

$$P(H|E) = \frac{P(E|H) \times P(H)}{P(E)}$$

Di mana:

- $P(H)$ adalah probabilitas hipotesis H benar. Ini dikenal sebagai probabilitas sebelumnya.
- $P(E)$ adalah probabilitas bukti (terlepas dari hipotesisnya).
- $P(E|H)$ adalah probabilitas bukti jika hipotesis tersebut benar.
- $P(H|E)$ adalah probabilitas hipotesis jika terdapat bukti.

Pengklasifikasi naïve Bayes menggunakan teorema Bayes untuk memprediksi probabilitas keanggotaan setiap fitur. Kelas dengan probabilitas tertinggi dianggap sebagai kelas yang paling mungkin, yang disebut Maksimum A Posteriori (MAP). Rumus yang digunakan untuk menghitung MAP (hipotesis) diberikan di sini:

$$\text{MAP}(H) = \text{Max} (P(H|E)) = \max \frac{P(E|H) \times P(H)}{P(E)} = \max (P(E|H) \times P(H))$$

Algoritma naïve Bayes adalah algoritma yang cepat dan sangat skalabel. Ini mudah diterapkan dan merupakan pilihan tepat untuk masalah klasifikasi teks. Ini banyak digunakan dalam deteksi spam email. Bagian terbaik dari algoritme ini adalah ia dapat dilatih pada kumpulan data kecil. Salah satu kelemahan algoritma ini adalah menganggap semua fitur tidak berhubungan; oleh karena itu, ia tidak dapat memprediksi hubungan antar fitur. Anda dapat meningkatkan akurasi algoritma naïve Bayes dengan melakukan preprocessing pada data. Praktik yang baik adalah menghapus kata-kata berhenti sebelum menjalankan pengklasifikasi. Kata-kata penghenti adalah kata-kata umum yang tidak menambah nilai apa pun pada kategorisasi; itu adalah kata-kata seperti a, mampu, baik, lain, selamanya, dan seterusnya. Teknik pembersihan lain yang diterapkan disebut lemmatisasi. Dalam lemmatisasi, Anda menemukan akar kata untuk melatih model kalimat tertentu. Para ahli telah melaporkan bahwa keakuratan algoritme dapat ditingkatkan dengan menggunakan TF-IDF, bukan sekadar menghitung setiap kata. TF-IDF adalah singkatan dari Term Frekuensi-Invers Dokumen Frekuensi, jadi alih-alih menghitung frekuensi setiap kata, Anda melakukan pemrosesan lebih lanjut seperti memberi penalti pada kata yang lebih sering muncul dalam data pelatihan.

Sekarang, Anda telah memiliki pemahaman teoritis tentang algoritma naïve Bayes. Mari lanjutkan dan buat pengklasifikasi Anda sendiri sekarang dan lihat kinerjanya dibandingkan dengan LUIS.ai.

Mengkodekan Pengklasifikasi

Anda akan memulai dari awal, jadi mari buat folder proyek baru bernama my-classifier dan bootstrap proyek menggunakan NPM.

```
$ mkdir my-classifier
$ cd my-classifier
$ npm init .
```

Berikan semua nilai default ketika perintah init dijalankan untuk NPM. Anda tidak akan mengimplementasikan pengklasifikasi naïve Bayes tetapi menggunakannya dari modul Node. Alasan untuk melakukan ini adalah untuk menghindari kesalahan saat membuat model matematika Anda. Dalam pembelajaran mesin, membuat algoritme dan melakukan rekayasa fitur lebih berharga daripada menerapkan algoritme.

Untuk membuat pengklasifikasi naïve Bayes, Anda akan menggunakan perpustakaan Natural, yang memiliki implementasi algoritme sumber terbuka. Perpustakaan Natural tersedia di NPM dan merupakan fasilitas bahasa umum untuk Node.js. Ini menyediakan hal-hal berikut sebagai bagian dari penawarannya:

- Tokenisasi
- Membendung
- Klasifikasi
- Fonetik
- TF-IDF
- WordNet
- Kesamaan string

Memasang Modul natural

Mari kita instal modul Natural melalui NPM. Jalankan perintah berikut di jendela Terminal/prompt perintah Anda:

```
$ npm install --save natural
```

Modul Pengklasifikasi

Seperti yang telah dibahas di bagian sebelumnya, kami sangat menyukai modularisasi kode menjadi beberapa file untuk memudahkan mengekspor fungsionalitas dan meningkatkan penggunaan kembali kode daripada menduplikasi kode. Mari buat classifier.js di direktori my-classifier. Pengklasifikasi Anda harus memiliki dukungan untuk fungsi berikut:

- Memuat dan menyimpan file pengklasifikasi dari/ke sistem file atau database untuk data yang telah dilatih. Anda tidak ingin harus melatih model setiap kali Anda memulai aplikasi.
- Menyediakan API untuk menambahkan pelatihan saat bepergian. API dapat berupa pemanggilan fungsi.
- Menyediakan API prediksi untuk mengembalikan hasil berupa daftar kemungkinan klasifikasi dengan keyakinan untuk setiap klasifikasi.

Dari persyaratan sebelumnya, Anda dapat dengan mudah menyimpulkan bahwa Anda memerlukan fungsi berikut untuk diterapkan di file classifier.js Anda:

- `saveClassifier`
- `loadClassifier`
- `addTraining`
- `predict`

Mari terapkan fungsi-fungsi ini di file classifier.js Anda dan ekspor sebagai modul untuk digunakan oleh aplikasi lain.

```

classifier.js
var natural = require('natural');

function getNewClassifier () {
  return new natural.BayesClassifier();
}

function loadClassifier (sourceFile) {
  return new Promise (function(resolve, reject) {
    natural.BayesClassifier.load (sourceFile, null, function(err,
    loadedClassifier) {
      if (loadedClassifier) {
        return resolve(loadedClassifier);
      } else {
        reject (err);
      }
    });
  });
}

function saveClassifier (classifier, destinationFile) {
  return new Promise(function (resolve, reject) {
    classifier.save (destinationFile, null, function(err,
    savedClassifier) {
      if (savedClassifier) {
        return resolve (savedClassifier);
      } else {
        return reject (err);
      }
    });
  });
}

function addTraining (classifier, utterance, label) {
  classifier.addDocument (utterance, label);
}

```

```

function train(classifier) {
  return new Promise (function (resolve) {
    classifier.train ();
    return resolve ();
  });
}

function predict (classifier, utterance) {
  return JSON.stringify(classifier.classify(utterance));
}

module.exports = {
  loadClassifier: loadClassifier,
  saveClassifier: saveClassifier,
  addTraining: addTraining,
  predict: predict,
  train: train,
  getNewClassifier: getNewClassifier
};

```

Anda sekarang telah membuat API yang mudah digunakan melalui perpustakaan Natural. Bersamaan dengan empat fungsi yang disebutkan sebelumnya, Anda telah mengimplementasikan dua fungsi tambahan (train, getNewClassifier) untuk mengaktifkan pelatihan maksud saat ini dan pengambilan pengklasifikasi baru ketika tidak ada pengklasifikasi. Mari kita lanjutkan dan gunakan fungsi-fungsi ini untuk melatih beberapa maksud dan memprediksinya setelah itu. Buat file bernama app.js di direktori root proyek my-classifier.

```

app.js
var mlModel = require('./classifier');

function trainModel () {
  var myClassifier = mlModel.getNewClassifier();
  mlModel.addTraining(myClassifier, "Hello! How are you?", "greeting");
  mlModel.addTraining(myClassifier, "Hi", "greeting");
  mlModel.addTraining(myClassifier, "Hey", "greeting");
  mlModel.addTraining(myClassifier, "What's up", "greeting");
  mlModel.addTraining(myClassifier, "How are you?", "greeting");
  mlModel.addTraining(myClassifier, "I want to buy a shirt", "buy-shirt");
  mlModel.addTraining(myClassifier, "I am looking for shirts", "buy-shirt");
  mlModel.addTraining(myClassifier, "Do you have any shirts?", "buy-shirt");
  mlModel.addTraining(myClassifier, "Help", "help");
  mlModel.addTraining(myClassifier, "Main Menu", "help");
  mlModel.saveClassifier(myClassifier, 'classifier.json');
}

```

```

function loadModel () {
  return new Promise(function(resolve, reject){
    mlModel.loadClassifier('classifier.json').then(function(classifier){
      console.log ('Classifier Loaded');
      mlModel.train(classifier).then(function () {
        console.log ('Model is trained');
        console.log (`Hey!: ${mlModel.predict(classifier,"Hey")}`);
        console.log (`Do you have shirts?: ${mlModel.
          predict(classifier,"I need help looking for shirts")}`);
        return resolve();
      });
    }, function (){
      console.log ('Could not load the Model');
      return reject();
    });
  });
}

loadModel().then(() => {
  console.log ('Finished Execution');
}, () => {
  console.log ('Error with Execution');
});

```

Tambahkan maksud berikut untuk dilatih berdasarkan pengklasifikasi Naïve Bayes Anda:

- Salam
- Beli baju
- Membantu

Sebelum menjalankan file `app.js` sebelumnya, pastikan Anda telah menjalankan file tersebut saat baru memanggil fungsi `trainModel`. Ini membuat pengklasifikasi kosong, menambahkan data pelatihan ke dalamnya, dan akhirnya menyimpannya dalam file di direktori akar proyek `classifier.json`. Saat Anda memanggil fungsi `loadModel`, file `classifier.json` dimuat dari sistem file, dan pengklasifikasi dibuat darinya. Anda kemudian melanjutkan untuk memprediksi maksud data pengujian.

15.7 BISNIS DAN MONETISASI

Segala sesuatu yang tidak disimpan akan hilang.

—Pesan “Keluar dari Layar” Nintendo

Ini merupakan perjalanan yang mengasyikkan kami telah mempelajari semua aspek teknis dalam membangun chatbot. Pada bab sebelumnya, Anda fokus pada pembuatan bot end-to-end dengan integrasi ke API pihak ketiga, menghubungkan ke beberapa layanan, dan akhirnya menerapkannya di Facebook Messenger, Skype, dan Slack.

Dalam bab ini, kita akan membahas permasalahan yang ada: membangun bisnis menggunakan chatbot yang memanfaatkan kecerdasan buatan dan memberikan nilai kepada pengguna. Salah satu langkah paling penting untuk memonetisasi teknologi yang baru lahir adalah dengan membangun analisis yang hebat.

Kecuali pola penggunaan dan analisis dasar dapat dilacak, sulit untuk memahami apa yang disukai atau tidak disukai pengguna Anda.

Kami memulai bab ini dengan menelusuri analitik yang harus Anda lacak. Selanjutnya kami jelaskan dimana chatbots dapat berperan besar untuk memberikan kemudahan kepada pengguna. Kami menjelajahi berbagai kasus penggunaan di banyak vertikal dan industri untuk menunjukkan contoh di mana Anda dapat memanfaatkan chatbot dengan baik.

■ Tip Ide dan solusi bisnis terbaik sering kali diakibatkan oleh kesulitan seseorang dalam melakukan aktivitas yang seharusnya mudah dilakukan. Sebaiknya berhenti sejenak di sini dan buat daftar semua masalah yang Anda hadapi secara online dan offline saat berkomunikasi dengan bisnis atau merek; lalu lihat apakah masalah tersebut dapat diselesaikan dengan menggunakan teknologi dan data chatbot.

Analisis: Mengapa dan Bagaimana?

Data adalah hal terpenting di dunia saat ini; ini membantu kita membuat keputusan yang lebih baik dan tepat. Faktanya, perusahaan teknologi terbesar di dunia membangun dominasinya karena data yang mereka miliki. Google mengindeks seluruh World Wide Web untuk dicari orang. Facebook memiliki informasi tentang sebagian besar teman dan keluarga kita dan menyediakan cara mudah untuk bersosialisasi secara online. Data menjelaskan banyak hal tentang penggunaan aplikasi dan biasanya merupakan indikator keberhasilan ketika ditafsirkan dan dikerjakan.

Analytics adalah ilmu yang mengekstraksi pola, tren, dan informasi yang dapat ditindaklanjuti dari data yang tersedia. Jumlah data yang dihasilkan setiap tahun di Internet telah tumbuh secara eksponensial dan dapat dikaitkan dengan tiga faktor utama: bandwidth, penyimpanan digital, dan kekuatan pemrosesan. Data yang tersedia harus dimanfaatkan dan dianalisis, yang akan membantu Anda menjaga merek Anda tetap terdepan dalam menghadapi gangguan. Hal ini juga dapat membantu Anda meningkatkan posisi kompetitif Anda dibandingkan dengan perusahaan lain di pasar tempat Anda beroperasi.

Saat bisnis Anda mulai berkembang, ada beberapa metrik yang harus terus dipantau untuk membantu Anda menandai kemajuan Anda setiap minggu atau bulanan. Untuk bisnis yang berhubungan dengan konsumen, metrik ini mencakup jumlah pengguna aktif, pengguna yang kembali, rata-rata waktu yang dihabiskan pada aplikasi, dan sebagainya. Untuk bisnis yang menghadapi bisnis, metrik yang paling umum untuk diukur adalah skor promotor bersih (NPS). Alat yang dapat membantu Anda mengukur metrik ini antara lain Google Analytics, Piwik, FireStats, dan sebagainya.

Chatbots saat ini berada pada tahap awal, dan ekosistem perangkat untuk mengukur pertumbuhan chatbots masih belum matang. Solusi analitik yang ada seperti Google Analytics tidak dapat digunakan karena metrik yang diperlukan untuk mengukur kriteria keberhasilan chatbot sangat bervariasi pada aplikasi berbasis web atau seluler. Berikut adalah beberapa perbedaan utama antara aplikasi tradisional (web dan seluler) dan aplikasi perpesanan:

- Obrolan bersifat asinkron. Jumlah sesi per pengguna tinggi, namun rata-rata waktu yang dihabiskan per sesi jauh lebih rendah. Ini karena pengguna mengirim pesan dan kembali ke apa yang mereka lakukan dan kemudian memeriksa responsnya nanti.
- Obrolan berfungsi dengan baik dalam kasus penggunaan dukungan pelanggan, dan dalam kasus ini jumlah pengguna yang kembali akan lebih sedikit karena kemungkinan besar pertanyaan pengguna akan diselesaikan dalam satu percakapan.
- Penurunan mungkin terjadi karena berbagai alasan, dan pelacakan halaman tidak akan banyak membantu. Anda perlu melacak pesan di mana penurunan terjadi dan menggali polanya.

Untuk mengatasi tantangan ini, Anda memerlukan modul analisis percakapan mandiri yang dapat menangani kompleksitas pengukuran pertumbuhan chatbot Anda. Di bagian selanjutnya, kita akan membahas analitik yang diperlukan untuk mengikuti lintasan kesuksesan Anda.

Pada bab sebelumnya, Anda menyiapkan aplikasi chatbot untuk menyimpan pesan yang dipertukarkan antara pengguna dan chatbot. Anda menambahkan pengendali peristiwa yang dipicu pada setiap pesan yang dikirim atau diterima oleh chatbot. Di beberapa bagian berikutnya, Anda akan fokus pada pemahaman metrik yang berperan besar dalam keberhasilan chatbot; sebagian besar dapat dengan mudah dilacak dari data yang Anda masukkan ke MongoDB.

Analisis Teratas

Agar Anda sebagai merek dapat memperoleh wawasan dan meningkatkan keterlibatan di chatbot Anda, Anda harus melacak analitik dan metrik tertentu apa pun jenis chatbotnya. Untuk sebagian besar metrik yang disebutkan di sini, angka yang lebih tinggi menunjukkan bahwa chatbot bekerja dengan baik.

■ **Tip Saat mengukur analitik dari berbagai poin penting, angka absolut tidak menjadi masalah.** Yang penting adalah perubahan relatif angka dalam jangka waktu tertentu. Memiliki 100.000 pengguna di chatbot dengan jumlah tersebut tumbuh 5 persen setiap minggunya jauh lebih baik daripada memiliki 500.000 pengguna yang konstan.

Kami telah memilih beberapa metrik teratas untuk dicakup; mereka mungkin berbeda untuk bisnis Anda tergantung pada kasus penggunaannya. Namun jika metriknya berubah secara positif dari minggu ke minggu atau dari bulan ke bulan, Anda melakukannya dengan baik. Di bagian berikutnya, kami akan membahas analisis, menjelaskan maksudnya, dan memberikan beberapa saran berdasarkan pengalaman kami tentang cara meningkatkan metrik tersebut.

15.8 JUMLAH PENGGUNA

Jumlah total pengguna chatbot merupakan indikasi yang baik tentang bagaimana bot tersebut dipandang oleh kelompok sasaran Anda. Dalam kebanyakan kasus, memiliki lebih banyak pengguna berarti bot yang memberikan nilai kepada pengguna. Selain itu, memiliki banyak pengguna menyediakan data yang Anda perlukan untuk memvalidasi hipotesis Anda tentang metrik tertentu; juga, memiliki banyak pengguna menghilangkan kesalahan atau anomali apa pun yang mungkin terjadi karena ukuran sampel yang kecil. Jumlah pengguna chatbot dapat dianggap setara dengan jumlah unduhan aplikasi seluler atau jumlah kunjungan unik di situs web. Perubahan metrik saat mengukur jumlah pengguna harus terus bertambah selama jangka waktu yang dipilih. Memperoleh lebih banyak pengguna berkorelasi langsung dengan kesadaran merek atau bisnis Anda di antara audiens atau kelompok target Anda. Memiliki chatbot akan membantu Anda meningkatkan kesadaran merek karena ini merupakan sumber saluran akuisisi pengguna.

Penyimpanan

Sisi lain dari cerita untuk aplikasi apa pun adalah siklus hidup suatu interaksi. Retensi menunjukkan kepada Anda “kelengketan” chatbot yang diberikan. Chatbot dengan retensi yang baik adalah yang melibatkan audiens saat ini hingga mereka melakukan transaksi. Transaksi didefinisikan sebagai tindakan yang harus dilakukan oleh pengguna untuk membantu merek/bisnis mencapai metrik. Misalnya, untuk chatbot e-niaga, transaksinya didefinisikan sebagai pengguna yang membeli suatu item, sedangkan untuk chatbot pemesanan film, transaksinya didefinisikan sebagai pengguna yang akhirnya memesan film tersebut.

Strategi retensi yang baik adalah dengan terus menargetkan ulang pengguna berdasarkan pola penggunaan mereka. Jika dengan chatbot e-niaga pengguna telah melihat beberapa produk dan berhenti setelah itu, cara yang baik untuk berinteraksi dengan pengguna adalah dengan mengirimkan pesan pengingat lembut dan menampilkan item serupa bersamanya. Untuk chatbot yang menyediakan

mekanisme untuk memesan tiket ke suatu acara, strategi retensi yang baik adalah dengan mengirimkan pembaruan acara melalui pesan dan membuat pengguna tetap terlibat dengan mengirimkan kuis atau fakta tentang acara tersebut. Strategi retensi Anda akan memainkan peran besar dalam membuat pengguna Anda senang. Jika bot mengirimkan terlalu banyak pesan atau pembaruan dalam sehari, bot tersebut mungkin ditandai sebagai spam atau mengganggu pengguna.

- Tip Berikan pengguna cara untuk memilih tidak menerima pembaruan dengan cara yang dapat diakses.

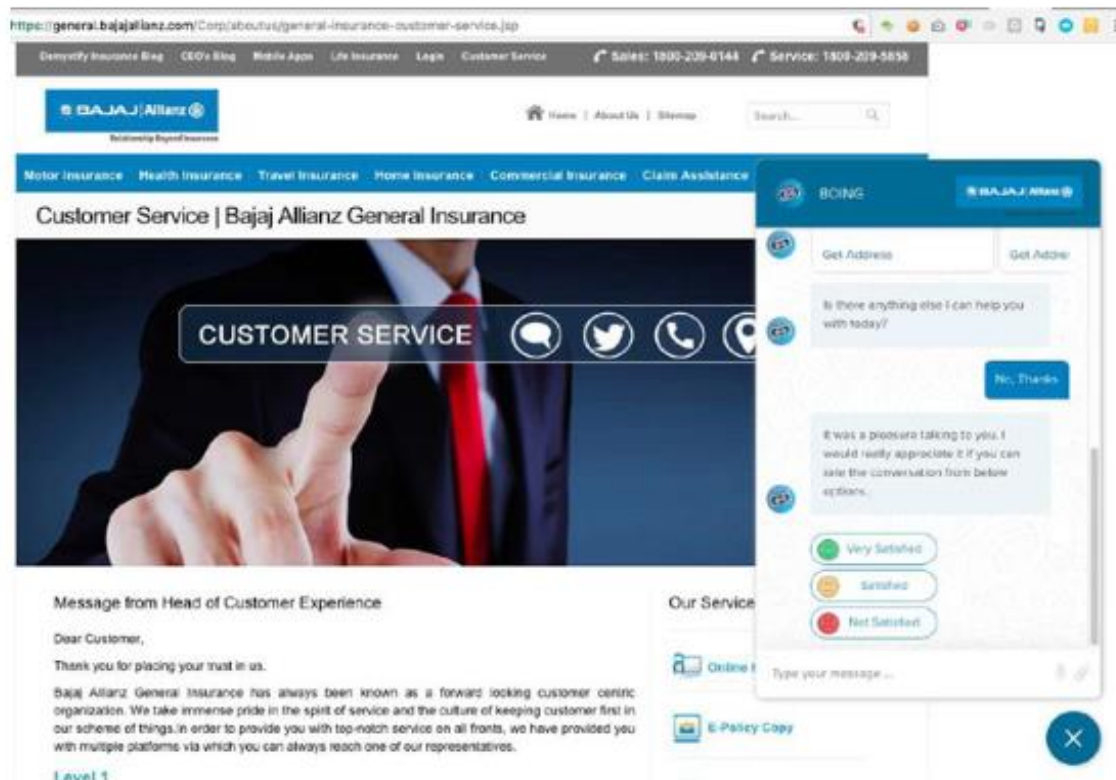
Sentimen

Analisis sentimen membantu Anda memahami perasaan pengguna saat menggunakan chatbot. Anda harus selalu bertujuan untuk memberikan perasaan senang kepada pelanggan

Anda saat berinteraksi dengan chatbot Anda. Terjebak dalam satu lingkaran dan tidak dapat memahami apa yang ingin dikatakan bot adalah cara paling umum untuk membuat sentimen pengguna menjadi negatif.

Rancang pengalaman pengguna Anda sedemikian rupa sehingga jika pengguna terhenti pada suatu saat atau tidak dapat melanjutkan, pengguna akan melihat opsi navigasi untuk kembali ke menu utama atau mungkin melanjutkan tindakan yang sedang dilakukan. Analisis sentimen memungkinkan Anda memecahkan masalah menggunakan data sentimen yang diberi tag dalam jumlah besar. Anda dapat menemukan set pelatihan untuk melatih model Anda dan menggunakan algoritme naïve Bayes untuk mendapatkan sentimen pengguna untuk setiap pesan. Memiliki lebih banyak umpan balik negatif dan netral daripada positif adalah tanda bahaya, dalam hal ini Anda harus meluangkan waktu untuk fokus pada perjalanan pengguna.

Setelah sentimen negatif terdeteksi oleh chatbot, sebaiknya beri tahu pengguna bahwa mereka didengarkan dan ambil tindakan untuk mentransfer obrolan ke agen manusia yang dapat membantu percakapan lebih lanjut. Sentimen pengguna dapat ditangkap melalui model pembelajaran mesin, seperti yang dibahas sebelumnya, yang bersifat implisit (Gambar 15-1). Umpan balik pengguna yang eksplisit pada chatbot juga dapat membantu mengidentifikasi sentimen, seperti yang ditunjukkan pada Gambar 15-2.



Gambar 15.14 Chatbot Bajaj Allianz General Insurance dengan fungsi umpan balik untuk menangkap sentimen secara eksplisit



Gambar 15.15 Chatbot Bajaj Allianz General Insurance dengan deteksi sentimen implisit

Seperti disebutkan, Anda dapat menggunakan teknik deteksi sentimen implisit dan eksplisit untuk memahami perilaku pengguna. Seperti yang ditunjukkan pada Gambar 15.14 dan Gambar 15.15, segera setelah pengguna mengetikkan sentimen negatif, bot akan merespons dengan mengambil tindakan yang sesuai. Chatbot ini diterapkan oleh Bajaj Allianz General Insurance Group di India dan dapat diakses di <https://general.bajajallianz.com/Corp/aboutus/general-asuransi-customer-service.jsp>.

Kecepatan Respons

Salah satu alasan mengapa chatbots memiliki keunggulan dibandingkan obrolan orang-ke-orang biasa dalam kasus penggunaan bisnis adalah semakin lamanya waktu yang dibutuhkan manusia untuk memahami apa yang dikatakan pengguna dan menghasilkan respons yang sesuai. Agen manusia yang merespons satu pesan memerlukan waktu beberapa detik jika tidak berbicara dengan orang lain. Jika agen manusia menangani beberapa obrolan pada waktu tertentu, waktu penyelesaian rata-rata meningkat menjadi lebih dari beberapa menit. Dalam kasus chatbot yang merespons pertanyaan pengguna, idealnya tidak memerlukan waktu lebih dari satu atau dua detik untuk menghubungi pengguna dengan informasi yang diminta.

Dengan memasuki pasar chatbots, ekspektasi rata-rata pengguna meningkat drastis. Jika chatbot Anda membutuhkan waktu lebih dari beberapa detik untuk merespons pertanyaan pengguna, mungkin ada baiknya meluangkan waktu untuk memahami alasannya. Jika bot Anda melakukan tindakan yang menghabiskan banyak waktu, sebaiknya beri tahu pengguna tentang

hal ini. Jika Anda mencoba mengambil data dari API atau layanan pihak ketiga untuk menyajikan data ke chatbot, praktik yang baik adalah memberi tahu pengguna tentang hal ini dan selalu menangani kasus edge ketika API gagal; dengan kata lain, berikan pesan kesalahan yang sesuai kepada pengguna agar mereka mengetahui masalahnya.

Bayangkan sebuah skenario dengan bot berbasis pemesanan di mana Anda telah memilih kursi untuk film yang ingin Anda tonton selama akhir pekan. Bagian belakang pemesanan kursi kelebihan beban, sehingga chatbot tidak dapat merespons Anda dengan nomor kursi. Jika operasi memakan waktu terlalu lama untuk diselesaikan, sebaiknya beri tahu pengguna bahwa permintaan ada dalam antrean dan akan diproses oleh server back-end sesegera mungkin. Jika pesan seperti itu tidak diberikan, maka pengguna mungkin mengira ada yang salah dengan chatbot dan akan meninggalkan sesi tersebut dan membuat permintaan pemesanan lain melalui media lain.

Durasi Sesi

Durasi sesi adalah metrik yang sulit untuk ditangani. Pengguna harus menghabiskan lebih banyak waktu di chatbot Anda sambil melakukan sesuatu yang produktif. Jika pengguna mengalami kebuntuan dan menghabiskan banyak waktu antara dua interaksi berturut-turut, berarti ada yang salah dengan UX chatbot. Durasi sesi saja tidak dapat dikaitkan dengan suatu kesimpulan. Durasi sesi akan berbeda untuk industri dan keadaan tertentu. Bot e-commerce, misalnya, akan memiliki durasi sesi yang lebih lama dibandingkan dengan bot yang memberikan suhu di areanya kepada pengguna. Durasi sesi bersifat kontekstual, dan tidak boleh ada kesimpulan yang diambil hanya dari durasi sesi besar/kecil.

15.9 ANALISIS NIAT

Menangkap data niat penting bagi bisnis dan pengembang bot. Karena niat berkorelasi langsung dengan tindakan yang dilakukan oleh pengguna di chatbot, analisis niat memberikan ukuran yang baik tentang layanan dan tindakan teratas yang dilakukan di chatbot. Sebagai sebuah bisnis, dengan memantau maksud yang dilakukan di chatbot, Anda akan memiliki keunggulan strategis dibandingkan pesaing Anda. Analisis niat menunjukkan layanan yang paling populer dan tidak populer. Dengan mengoptimalkan sumber daya dan memberikan pengalaman terbaik untuk layanan populer, Anda dapat meningkatkan retensi dan jumlah pengguna secara keseluruhan.

Pada akhirnya, sebagai pengembang chatbot, Anda harus mengetahui tindakan teratas yang dilakukan oleh pengguna di chatbot Anda. Selain itu, mencatat waktu di mana tindakan ini dilakukan biasanya memberikan banyak informasi tentang penggunaan chatbot.

Menangkap niat yang “tidak ada” sama pentingnya dengan menangkap niat lainnya. Frekuensi maksud “tidak ada” akan memberi Anda detail tentang seberapa baik NLP di chatbot Anda bekerja. Biasanya, chatbot dengan maksud “tidak ada” yang berlebihan kemungkinan besar merupakan bot yang berkinerja buruk, dengan semua data analitik lainnya menunjukkan

kehancurannya. Mengirimkan pesan kesalahan dan menangani situasi dengan baik merupakan praktik desain yang baik.

Jenis Kelamin dan Usia

Pembuatan profil pengguna membantu dalam memahami basis pengguna Anda. Gender dan usia berperan besar dalam menentukan cara memasarkan chatbot kepada audiens yang dituju. Cara menjangkau tiap gender dan kelompok umur berbeda-beda, dan mekanisme pemasaran yang berbeda harus diterapkan untuk menjangkau audiens. Facebook menyediakan beberapa informasi pengguna seperti jenis kelamin dan rentang kelompok usia, sedangkan di platform lain, mungkin bermanfaat untuk mengumpulkan informasi yang sama.

Pembuatan profil ini juga akan membantu Anda sebagai pebisnis untuk mengkorelasikan pola penggunaan di berbagai jenis kelamin dan kelompok umur. Untuk chatbot e-niaga, penandaan kelompok usia setiap pengguna akan membantu Anda memahami tindakan berbagai kelompok usia dengan lebih baik.

Wilayah

Untuk sebagian besar kasus penggunaan, Anda akan menentukan geografi chatbot yang akan diluncurkan. Merupakan praktik yang baik untuk melacak lokasi pengguna chatbot untuk mendapatkan wawasan tentang pola penggunaan pengguna di berbagai wilayah. Jika Anda meluncurkan chatbot yang akan digunakan di beberapa wilayah, pastikan Anda menyediakan dukungan bahasa untuk semua wilayah.

Kami telah membahas kumpulan analisis dasar yang dapat Anda gunakan untuk memperoleh wawasan bernilai tambah bagi Anda sebagai pengembang chatbot atau untuk merek atau bisnis Anda. Tingkat pertumbuhan dan adopsi chatbot Anda adalah metrik yang baik untuk diingat ketika membangun kasus penggunaan bisnis-ke-konsumen (B2C).

15.10 KASUS PENGGUNAAN CHATBOT

Bagi orang yang memegang palu, segalanya tampak seperti paku.

—Mark Twain

Antarmuka obrolan adalah salah satu antarmuka pengguna paling sederhana yang pernah dirancang. Ini terdiri dari beberapa gelembung pesan di kedua sisi jendela dan area input teks di bagian bawah layar. Untuk seseorang atau organisasi yang mempelajari kasus penggunaan chatbot, sepertinya antarmuka obrolan dapat menyelesaikan semua masalah yang dihadapi pengguna pada antarmuka pengguna “biasa”. Faktanya, antarmuka obrolan terasa sangat alami bagi kami, karena otak kami sudah terbiasa dengan cara kerja obrolan, berkat WhatsApp, yang telah memainkan peran utama dalam penerapan obrolan sebagai saluran komunikasi peer-to-peer.

Obrolan dapat mengganggu antarmuka yang telah ada selama berabad-abad, dan hal ini tampaknya lebih mungkin terjadi saat ini karena kemajuan teknologi dalam pembelajaran mesin

dan kecerdasan buatan. Antarmuka obrolan unggul dibandingkan antarmuka lainnya ketika fungsi yang akan dijalankan bersifat spesifik atau dapat dipersempit ke opsi tertentu dalam beberapa langkah. Beberapa contoh di mana obrolan dapat mengungguli antarmuka pengguna lainnya adalah mengajukan tiket untuk suatu masalah, meminta data masa lalu, dan melakukan pembayaran tagihan utilitas.

Pada bagian bab ini, kita akan membahas berbagai mode komunikasi yang ada di dunia saat ini, dan setelah itu kita akan membahas kasus penggunaan di setiap vertikal/sector. Saat ini sudah ada chatbot yang digunakan dalam berbagai kasus penggunaan ini.

15.11 CARA KOMUNIKASI

Seseorang sering kali memakai banyak topi sepanjang hari. Pada bagian ini, kita akan membahas berbagai peran yang dimainkan seseorang sepanjang hari dan menunjukkan bagaimana seseorang berinteraksi dengan orang lain dalam ekosistem. Anda kemudian akan berada dalam posisi yang lebih baik untuk memahami berbagai skenario penggunaan di mana chatbots dapat diterapkan.

Bisnis-ke-Bisnis (B2B)

Bisnis biasanya diwakili oleh satu orang dalam organisasi kecil atau sekelompok orang dalam organisasi yang lebih besar. Sebuah bisnis biasanya berinteraksi dengan bisnis lain di domainnya atau di luar domainnya karena berbagai alasan. Sebuah bisnis mungkin membeli beberapa produk/layanan dari bisnis lain untuk operasi sehari-harinya. Chatbots dalam bentuk asisten digital dapat digunakan dalam kasus penggunaan seperti itu, dimana chatbot menangani komunikasi untuk bisnis yang menyediakan produk atau layanan. Asisten dapat memberikan informasi seperti jam buka dan tutup, lokasi berbagai kantor, informasi produk, informasi kontak, dan lain sebagainya.

Bisnis-ke-Konsumen (B2C)

Dalam sebagian besar kasus penggunaan, suatu bisnis secara langsung menyediakan produk dan layanannya kepada konsumen. Frekuensi konsumen yang menggunakan layanan ini berbeda-beda, bergantung pada jenis dan geografi bisnis. Salah satu contoh chatbot yang paling umum untuk kasus penggunaan B2C adalah chatbot e-commerce. Chatbot e-niaga menyediakan semua informasi produk dan layanan tentang bisnis. Dalam beberapa kasus, konsumen mungkin tertarik dengan kegunaan lain seperti menanyakan harga, mendaftarkan tiket produk yang rusak atau tidak terkirim tepat waktu, dan lain sebagainya.

Konsumen-ke-Konsumen (C2C)

Orang yang berinteraksi dengan konsumen lain melalui obrolan termasuk dalam kategori ini. Ini adalah percakapan yang cukup sulit untuk diotomatisasi, dan chatbots pada saat ini sepertinya tidak terlalu berguna. Dalam skenario tertentu, chatbot mungkin digunakan untuk meningkatkan kualitas percakapan. Skenario seperti ini biasanya termasuk dalam kategori

belanja sosial. Lebih banyak platform perpesanan perlu bermunculan dan menyediakan lebih banyak kemampuan untuk memungkinkan pengalaman sosial berjalan lancar.

Bisnis-ke-Karyawan (B2E)

Dalam beberapa tahun terakhir, saluran yang melaluinya sebuah bisnis dapat berkomunikasi dengan karyawannya telah terbuka. Munculnya jaringan sosial pribadi dapat dikaitkan dengan meningkatnya interaksi tersebut. Banyak interaksi antara karyawan dan organisasi dapat diotomatisasi melalui chatbots. Aplikasi populer termasuk memiliki chatbot lengkap untuk pertanyaan terkait HR yang dicolokkan ke sistem HR utama. Chatbot semacam ini mengurangi bolak-balik saat mengetahui kebijakan SDM, meminta waktu liburan, dan sebagainya.

Karyawan-ke-Karyawan (E2E)

Dengan munculnya teknologi seperti Slack, Skype for Business, dan Microsoft Teams, percakapan antar karyawan meningkat di media obrolan. Produk-produk ini memberikan dukungan langsung terhadap bot, yang berarti saat ini terdapat peluang besar untuk membangun aplikasi yang meningkatkan produktivitas karyawan dalam suatu organisasi.

15.12 CHATBOTS MENURUT VERTIKAL INDUSTRI

Kami sekarang akan fokus pada vertikal dan mendiskusikan skenario chatbot seperti apa yang dapat dibangun. Kami terutama akan berfokus pada vertikal B2C karena vertikal tersebut didefinisikan dengan sangat baik dan ada banyak masalah yang harus diselesaikan. Saat ini, pendapatan pasar dukungan pelanggan saja sudah lebih dari \$20 miliar. Di sebagian besar bagian, kami membahas beberapa kasus penggunaan karena wajar jika suatu merek memiliki satu chatbot yang memberikan rekomendasi produk dan dukungan pelanggan kepada pengguna.

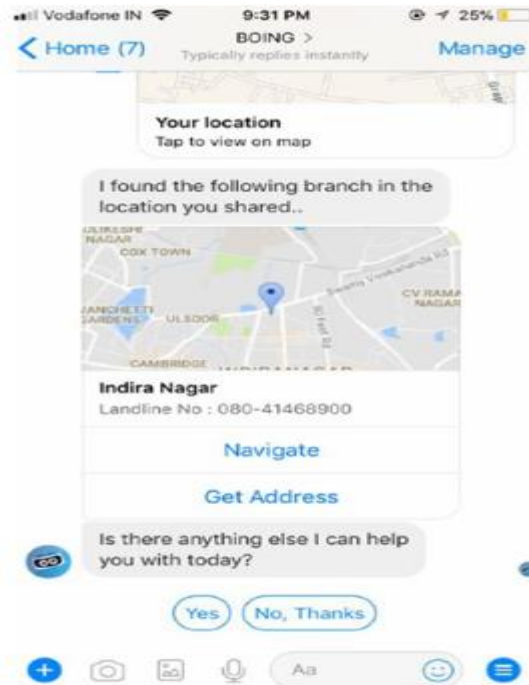
Perbankan, Jasa Keuangan, dan Asuransi (BFSI)

Cara kita berinteraksi dengan bank dan perusahaan asuransi telah berubah secara drastis. Sektor BFSI adalah pionir dalam adopsi teknologi baru. Sebelumnya, kami harus mengunjungi cabang bank atau menghubungi manajer hubungan kami bahkan untuk meminta buku cek baru. Saat ini, semua layanan ini hanya dengan sekali klik di situs web atau aplikasi seluler. Gelombang adopsi teknologi berikutnya telah dimulai oleh beberapa bank dan perusahaan asuransi terbesar di dunia, dimana mereka mengadopsi chatbot untuk kasus penggunaan tertentu dan menerapkannya dalam skala besar. Mari kita lihat aplikasi-aplikasi yang populer di sektor BFSI.

Bank internet

Proses perbankan normal dapat diakses melalui antarmuka obrolan, termasuk aktivitas seperti mencari cabang terdekat, memeriksa saldo, meminta transfer uang ke rekening lain, dan seterusnya (lihat Gambar 5-3). Kasus penggunaan dukungan pelanggan seperti meminta kartu baru atau memblokir kartu kredit curian dapat dilakukan dengan mudah melalui chatbot.

Chatbot berinteraksi langsung dengan sistem back-end sistem perbankan saat ini dan diberikan izin yang tepat untuk melakukan tindakan atas nama pengguna.



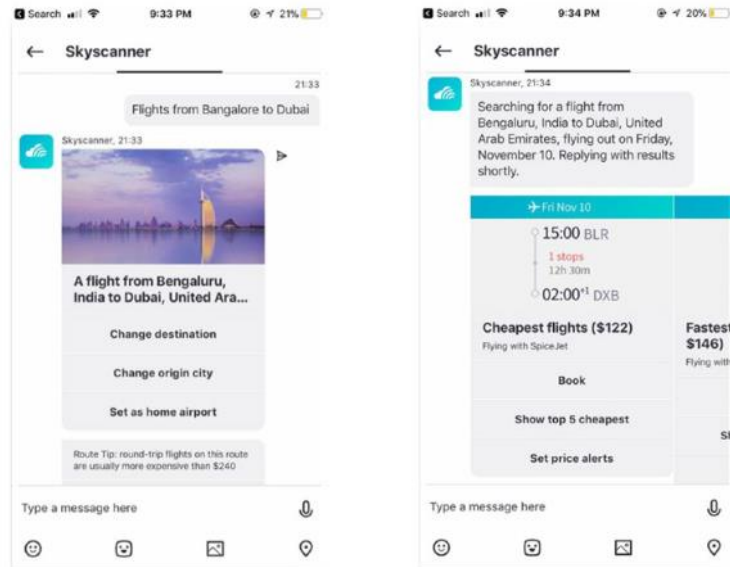
Gambar 15.16. Chatbot Bajaj Allianz General Insurance di Facebook Messenger membantu pengguna menemukan cabang terdekat

Pertanggung

Kegiatan asuransi melibatkan banyak bolak-balik antara pelanggan dan perusahaan asuransi. Data yang dipertukarkan antara kedua pihak untuk sebagian besar interaksi terstruktur dan dapat diotomatisasi. Beberapa kasus penggunaan dimana kita melihat adopsi chatbots di domain asuransi adalah mendaftarkan klaim asuransi, mengetahui status klaim, dan mendapatkan informasi tentang produk asuransi lainnya. Selain itu, chatbots memberikan kemampuan kepada perusahaan untuk melakukan cross-sell berbagai produk lain berdasarkan pola pembelian penggunanya. Di sinilah analisis yang kita bahas di awal bab ini berguna. Memahami dan membangun pola pembelian akan memungkinkan perusahaan memanfaatkan data yang ada untuk menyarankan produk dengan lebih baik kepada pengguna. Kasus penggunaan kedua di mana chatbot dapat membantu pengguna adalah memutuskan rencana yang tepat berdasarkan beberapa pertanyaan awal. Seringkali pengguna tidak menyadari penawaran yang mungkin berhak mereka terima, dan chatbots dapat membantu meningkatkan penjualan dengan menangkap dan memanfaatkan data penjualan.

Perjalanan: Bot Pemesanan

Perjalanan adalah pasar besar di mana banyak interaksi pelanggan terjadi sebelum penjualan dilakukan. Salah satu pendorong utama penjualan di bidang perjalanan adalah harga; pengguna selalu mencari cara untuk mengoptimalkan harga yang mereka bayarkan saat memesan hotel atau penerbangan.



Gambar 15.17 Chatbot Skyscanner di Skype yang membantu pengguna memesan tiket perjalanan

Perusahaan seperti Skyscanner dan Hipmunk menyediakan harga penerbangan dan hotel secara real-time. Salah satu kasus penggunaannya adalah dengan mengintegrasikan dan membangun chatbot yang berkomunikasi dengan beberapa pihak untuk mendapatkan harga penerbangan dan hotel serta mengawasi semua harga. Segera setelah harga kursi tertentu naik atau turun, pemberitahuan dapat dipicu. Keuntungan obrolan adalah semua konteks pencarian sebelumnya terlihat di layar pertama, dan perubahan apa pun dapat dilacak dengan mudah. Di aplikasi atau situs web, segera setelah Anda menutup dan membuka kembali situs web/aplikasi, konteks baru dimuat dengan riwayat Anda sebelumnya yang tidak mudah terlihat.

Kasus penggunaan lain yang dapat diintegrasikan pada chatbot adalah merekomendasikan tempat untuk dikunjungi atau dilihat saat berlibur (lihat Gambar 15.16). Kita sering melakukan banyak pencarian di beberapa blog untuk menemukan hal yang tepat untuk dilakukan di kota yang kita kunjungi. Sering kali, rekomendasi ini sudah ketinggalan zaman atau terlalu klise. Chatbot dapat mengatasi masalah ini dengan melakukan crowdsourcing data untuk

kota tertentu. Pengguna memberikan informasi terbaru tentang suatu tempat, dan chatbot menyusun semua rekomendasi dan menyajikannya kepada pengguna sesuai kebutuhan.

Makanan dan Restoran

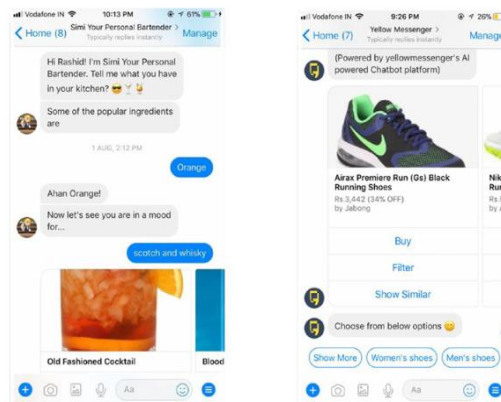
Kami telah melihat banyak kasus penggunaan yang dapat diotomatisasi menggunakan chatbot di industri makanan. Ini adalah kasus penggunaan yang mudah digunakan dan dibuat, dan kami mendorong Anda untuk mencoba membuat salah satu chatbot yang dijelaskan di bagian ini. Salah satu kategori utama pertanyaan untuk industri makanan terkait dengan reservasi meja; bahkan saat ini sebagian besar reservasi meja ditangani melalui telepon. Chatbot sepertinya cocok untuk masalah ini; akan lebih mudah untuk mengakses chatbot dan memesan meja untuk sejumlah orang saat bepergian.

Berdasarkan pengalaman kami membangun chatbot selama lebih dari dua tahun, kami menemukan beberapa skenario menarik untuk chatbot di industri makanan. Salah satu klien kami menginginkan Bot Bartender, yang tersedia saat ini. Pengguna memasukkan beberapa bahan ke dalam chatbot, dan chatbot kemudian menyarankan berbagai cocktail yang bisa dibuat. Bersamaan dengan saran-sarannya, chatbot juga memberikan resep cara membuat cocktail. Tantangan utama dalam membangun chatbot semacam ini adalah sumber datanya. Namun, jika data tersedia untuk Anda dan dapat digunakan oleh program komputer, Anda dapat dengan mudah mengubah data tersebut menjadi chatbot yang cantik.

Perdagangan elektronik

Dalam kasus penggunaan e-commerce, ada dua fungsi utama yang dapat dilakukan oleh chatbot: pencarian produk dan dukungan pelanggan (lihat Gambar 15.18).

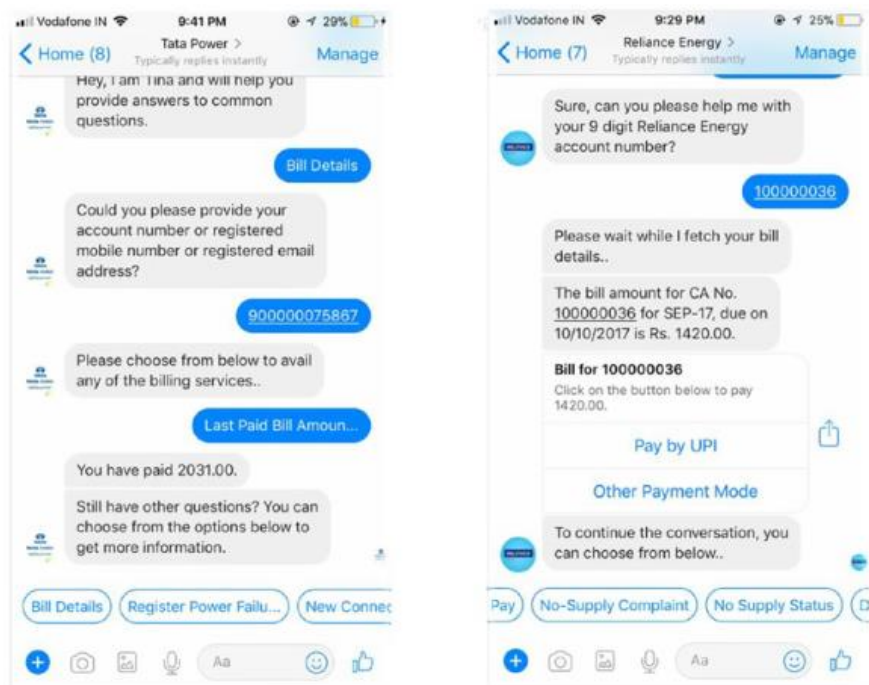
Mengotomatiskan dukungan pelanggan untuk e-niaga adalah pasar yang sangat besar, dan dengan kemajuan dalam pemahaman bahasa komputer, semua pertanyaan dukungan pelanggan akan segera ditangani oleh sistem otomatis. Mengotomatiskan dukungan untuk kasus penggunaan level 0 atau level 1 dapat dilakukan oleh chatbot. Sistem tiket dapat diintegrasikan ke dalam chatbot, yang kemudian dapat diekspos ke pengguna.



Gambar 15.18. Bot Simi Bartender (kiri), yang membantu pengguna menemukan resep tentang koktail.

Utilitas dan Tagihan

Layanan utilitas digunakan oleh semua orang, dan membayar tagihan adalah kasus penggunaan dimana otomatisasi chatbot dapat membantu (lihat Gambar 5-6). Berdasarkan pengalaman kami, chatbot yang membantu pengguna mengelola utilitas mereka adalah salah satu area bot yang tumbuh paling cepat. Bot ini memiliki retensi yang baik dan dengan beberapa integrasi yang solid dapat memberikan banyak nilai kepada pelanggan akhir. Perusahaan telekomunikasi dan perusahaan listrik bisa mendapatkan keuntungan dengan meluncurkan chatbot untuk penggunaannya di berbagai platform (situs web, Facebook, Skype) dan menyediakan layanan pengambilan tagihan dasar serta integrasi solusi pembayaran.



Gambar 15.19. Di sisi kiri adalah utilitas chatbot yang dirilis oleh Tata Power. Di sebelah kanan adalah chatbot pembayaran yang diaktifkan oleh Reliance Energy yang diterapkan di Facebook Messenger.

DAFTAR PUSTAKA

- A. Swarnkar and A. Swarnkar, „Artificial Intelligence Based Optimization Techniques: A Review,‘Intelligent Computing Techniques for Smart Energy Systems.‡ Lecture Notes in Electrical Engineering, 607 (2020).
- Adamopoulou, E., & Moussiades, L. (2020b). Chatbots: History, technology, and applications. *Machine Learning with Applications*, 2, 100006. <https://doi.org/10.1016/j.mlwa.2020.100006>
- Arsovski S., Muniru I., Cheok A. Analysis of the chatbot open source languages aiml and chatscript: A review (2017). doi: 10.13140/RG.2.2.34160.15367
- Brouillette, T., (2016), “Social Media Messaging: The Trend to Embrace in 2017”, Retrieved from <https://www.linkedin.com/pulse/social-media-messaging-trend-embrace-2017-tami-brouillette/>
- Cardona, D. R., Schönborn, S., Werth, O., & Breitner, M. H. (2019). A mixed methods analysis of the adoption and diffusion of chatbot technology in the German insurance sector. 25th Americas Conference on Information Systems, AMCIS 2019, 1–10.
- Chatfuel (2017), “Why every business should embed a Messenger bot on their website”, Retrieved from <https://blog.chatfuel.com/customer-chat/>
- Constine, J., (2015), “Facebook Launches Messenger Platform for Content Tools and Chat with Businesses” retrieved from <http://social.techcrunch.com/2015/03/25/facebooklaunches-messenger-platform-with-content-tools-and-chat-with-businesses/>
- Dekay, S. H., (2012), “How large companies react to negative Facebook comments”, *Corporate Communications: an International Journal*, vol. no. 3, pp. 289-299.
- Dialogue in a Virtual Tutor for Educational Software. *Research in Computing Science* 56: 19-27
- Facebook (2017), “Introduction to Messenger platform”, Retrieved from <https://developers.facebook.com/docs/messenger-platform/introduction>
- Fingas, J., (2015), “Facebook Messenger no longer needs a Facebook account” Retrieved from <http://www.engadget.com/2015/06/24/facebook-messenger-without-an-fbaccount/>
- Hansson, L., and Wrangmo, A., (2013), “Optimal ways for companies to use Facebook as a
- Happiest Minds and Chat bots Insights. (2017), Retrieved from <https://www.happiestminds.com/Insights/chatbots/>
- Haristiani, N. (2019). Artificial Intelligence (AI) Chatbot as Language Learning Medium: An inquiry. *Journal of Physics: Conference Series*, 1387(1), 012020. <https://doi.org/10.1088/1742-6596/1387/1/012020>

- Hiremath, G. (2020). *Chatbot for education system*. December, 36–43.
- Hudiyono, R. F. (2022). Exploring Indonesian Companies' Chatbots to Gather Customer Experience. In the Proceedings of the 5th International Conference on Vocational Education Applied Science and Technology 2022. Available at: <https://www.mdpi.com/2504-3900/83/1/1/htm>. Retrieved on: March 13, 2023.
- Idaputra, M., & Mursyidah, M. (2019). Pembuatan Chatbot Bahasa Aceh Menggunakan Artificial Intelligence Markup Language. *Jurnal Infomedia: Teknik Informatika, Multimedia & Jaringan*, 4(1), 42-49
- Janarthanam, S. (2017). *Hands-on chatbots and conversational UI development: build chatbots and voice user interfaces with Chatfuel, Dialogflow, Microsoft Bot Framework, Twilio, and Alexa Skills*. Packt Publishing Ltd.
- Jia J. CSIEC: A computer assisted english learning chatbot based on textual knowledge and reasoning *Knowledge-Based Systems*, 22 (4) (2009), pp. 249- 255. doi: 10.1016/j.knosys.2008.09.001
- Johannsen F., Leist S., Konadl D., Basche M. Comparison of commercial chatbot solutions for supporting customer interaction *ECIS* (2018)
- Jurafsky, D., & Martin, J. H. (2020). *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*. Pearson.
- Komawar, O., Thakar, P., Shetty, R., Bartakke, A., & Desai, P. M. (2015). An Internet Relay Chat Bot using AIML. *International Journal of Science and Research*, 4(10), 2014–2016.
- Lalwani, T., Bhalotia, S., Pal, A., Rathod, V., & Bisen, S. (2018). Implementation of a Chatbot System using AI and NLP (SSRN Scholarly Paper No. 3531782). <https://doi.org/10.2139/ssrn.3531782>
- Lexico Dictionaries. (n.d.). CHATBOT: Definition of CHATBOT by Oxford Dictionary on Lexico.com also meaning of CHATBOT. Lexico Dictionaries | English. <https://www.lexico.com/en/definition/chatbot>
- Lin L., D'Haro L., Banchs R. A web-based platform for collection of human-chatbot interactions (2016), pp. 363-366. doi: 10.1145/2974804.2980500
- Lokman, A. S., Ameen, M. A., & Kapoor, S. (2019). Modern Chatbot Systems: A Technical Review . *Proceedings of the Future Technologies Conference (FTC) 2018*. https://link.springer.com/chapter/10.1007/978-3-030-02683-7_75.
- Lutze, H., (2010), "Using Facebook as a business-building tool", *Agency Sales*, vol. 40, pp. 46-48.
- marketing channel", *Journal of Information Communication and Ethics*, vol. 11, p.118.
- Mohamad Suhaili, S., Salim, N., & Jambli, M. N. (2021). Service chatbots: A systematic review. In *Expert Systems with Applications* (Vol. 184). Elsevier Ltd. <https://doi.org/10.1016/j.eswa.2021.115461>

- Molnár, G., Zoltán, S.: The role of chatbots in formal education. Presented at the 15 September 2018
- Musthafa, M. F., Buliali, J. L., & Hariadi, V. (2018). Pemodelan Multilabel Tweet Media Sosial Mahasiswa Untuk Klasifikasi Keluhan. *Jurnal Teknik ITS*, 7(1), A247–A252.
- Nimavat, K., & Champaneria, T. (2017, July). Chatbots: An overview Types, Architecture, Tools and Future Possibilities. https://www.researchgate.net/profile/KetakeeNimavat/publication/320307269_Chatbots_An_overview_Types_Architecture_Tools_and_Future_Possibilities/links/59dcf2fdaca2729031b77acf/Chatbots-An-overview-TypesArchitecture-Tools-and-Future-Possibilities.pdf.
- Nuruzzaman M., Hussain O.K. A survey on chatbot implementation in customer service industry through deep neural networks. 2018 IEEE 15th international conference on ebusiness engineering, IEEE, Xi'an (2018), pp. 54-61. doi: 10.1109/ICEBE.2018.00019
- Pietroszek, Krzysztof. "Providing Language Instructor with Artificial Intelligence Assistant". *International Journal of Emerging Technologies in Learning* (2007). Found at: https://www.researchgate.net/publication/26490167_Providing_Language_Instructor_with_Artificial_Intelligence_Assistant
- R. Juwita, "Media Sosial dan Perkembangan Komunikasi Korporat," *J. Penelit. Komun.*, vol. 20, no. 1, pp. 47–60, 2017, doi: 10.20422/jpk.v20i1.136.
- Sari, A. C., Virnilia, N., Susanto, J. T., Phiedono, K. A., & Hartono, T. K. (2020). Chatbot developments in the business world. *Advances in Science, Technology and Engineering Systems*, 5(6), 627–635. <https://doi.org/10.25046/aj050676>
- Srinivasa-Desikan, B. (2018). *Natural Language Processing and Computational Linguistics: A practical guide to text analysis with Python, Gensim, spaCy, and Keras*. Birmingham: Packt Publishing
- Trivedi A., Gor V., Thakkar Z. Chatbot generation and integration: A review *International Journal of Advance Research, Ideas and Innovations in Technology*, 5 (2) (2019), pp. 1308-1311 XX.XXX/IJARIIT-V5I2-1840
- Wünderlich, N. v., & Paluch, S. (2018). A Nice and Friendly Chat with a Bot: User Perceptions of AIBased Service Agents. *ICIS 2017: Transforming Society with Digital Innovation*, 0–11
- Yang, T., (2012), "The decision behaviour of Facebook users", *Journal of Computer Information Systems*, vol. 52, pp. 50-59

PERCAKAPAN ANTARMUKA PENGGUNA (User Interface) DALAM CHATBOT

Dr. Budi Raharjo, S.Kom., M.Kom., MM.

BIODATA PENULIS



Dr. Budi Raharjo, S.Kom., M.Kom., MM lahir di Semarang, tanggal 22 Februari 1985. Beliau adalah Alumni dari Universitas Bina Nusantara (BINUS University) Jakarta dan juga alumni Universitas Kristen Satya wacana (UKSW) Salatiga. Dr. Budi Raharjo telah menjadi Dosen pada Universitas STEKOM pada mata kuliah Kepemimpinan (Leadership), mata kuliah Pengantar Akuntansi, Manajemen Proses, Manajemen Akuntansi dan Manajemen Resiko Bisnis. Selain sebagai dosen Universitas STEKOM, Dr. Budi Raharjo, M.Kom., MM juga mempunyai bisnis sendiri dalam bidang perhotelan dan juga sebagai wirausaha dalam bidang pemasok unggas (ayam) beku, ke berbagai kota besar, khususnya Jakarta dan sekitarnya.

Pengalaman beliau berwirausaha menjadi bekal utama dalam penulisan buku ajar yang diterbitkan oleh Yayasan Prima Agus Teknik (YPAT) Semarang. Oleh sebab itu bukunya berisi langkah langkah praktis yang mudah diikuti oleh para mahasiswa, saat mahasiswa mengikuti proses perkuliahan pada Universitas Sains dan Teknologi Komputer (Universitas STEKOM). Jabatan struktural yang di embannya saat ini adalah Wakil Rektor 1 (Akademik) Universitas STEKOM Semarang.



YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-623-8120-66-6 (PDF)



Dr. Budi Raharjo, S.Kom., M.Kom., MM.

PERCAKAPAN ANTAR MUKA PENGGUNA (User Interface) DALAM CHATBOT



YAYASAN PRIMA AGUS TEKNIK

YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-623-8120-66-6 (PDF)



9 786238 120666