PENGEMBANGAN SOFTWARE

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

YAYASAN PRIMA AGUS TEKNIK

PENGEMBANGAN SOFTWARE

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

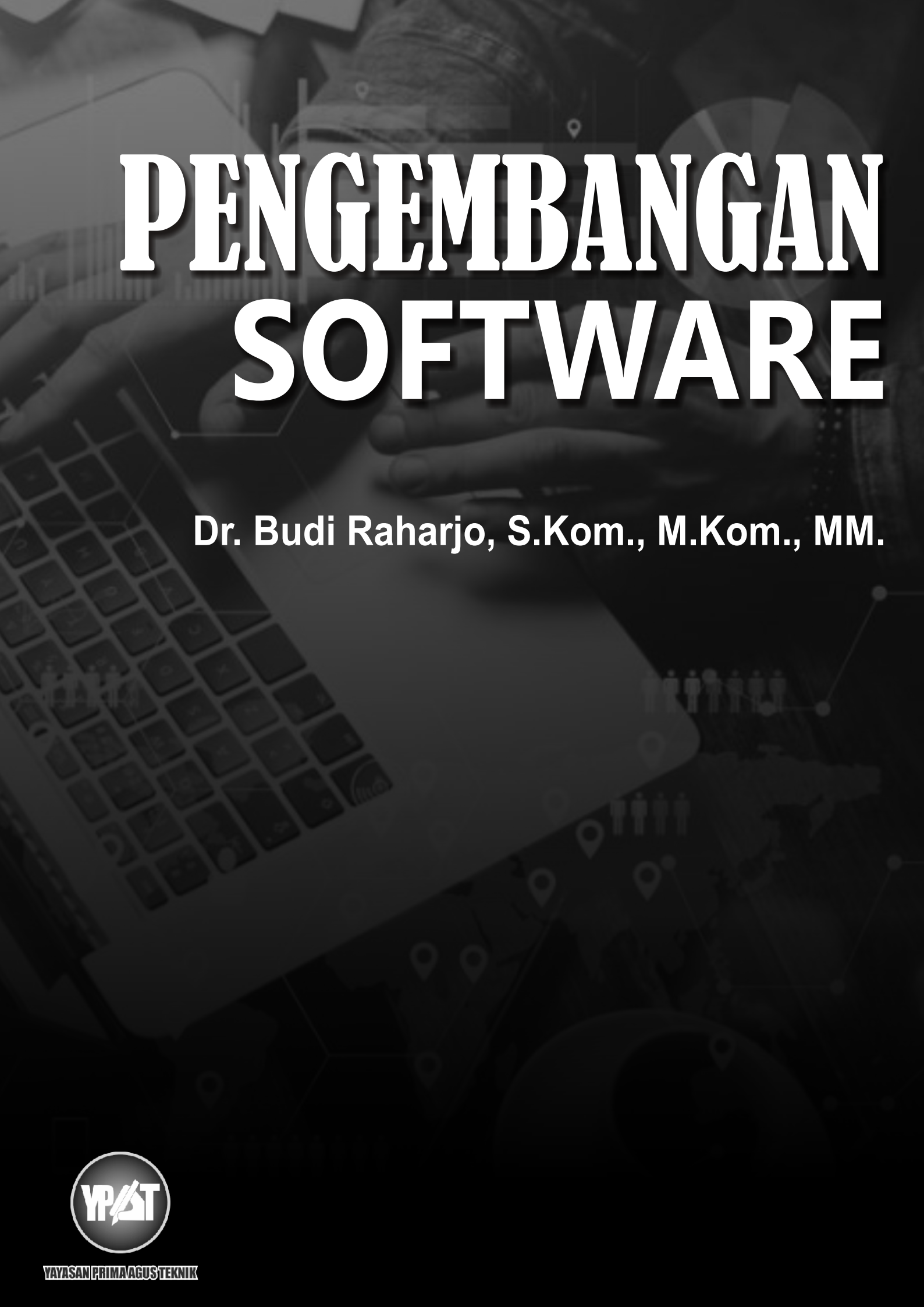
BIODATA PENULIS

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 978-623-5734-05-7 (PDF)



PENGEMBANGAN SOFTWARE

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

YAYASAN PRIMA AGUS TEKNIK

PENGEMBANGAN SOFTWARE

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

ISBN : 9 786235 734057

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

Penyunting :
Dr. Joseph Teguh Santoso, M.Kom.

Desain Sampul dan Tata Letak :
Irdha Yunianto, 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

KATA PENGANTAR

Puji syukur kami panjatkan atas kehadirat Tuhan karena buku yang berjudul "*Pengembangan Software*" Dapat terselesaikan. Dalam buku ini dibahas tentang bagaimana sebuah ide yang terlahir diubah menjadi subuah layanan produk yang menjembatani antara pengguna dan teknologi. Dalam buku ini membahas tidak hanya bagaimana menganalisis persyaratan, tetapi juga bagaimana mengumpulkan persyaratan awal untuk proyek yang akan dibangun serta ke tahap perancangan dari proses pengembangan.

Kenapa perlu adanya pengembangan perangkat lunak yaitu untuk membantu komunikasi antar tim developer dalam pengembangan software. Ini sangatlah penting, karena untuk mengurangi terjadinya kesalahan komunikasi antar tim. Komunikasi antar tim disini sangat diperlukan agar proses pengembangan *Software* dapat berjalan terarah. Fungsi lain dalam pengembambangan *Software* untuk memberikan hasil yang jelas mengenai input dan output dalam tahap pengembangan *Software.*

Dalam buku ini berbicara tentang pengujian kegunaan, dan juga membahas berapa banyak pengguna yang perlu dimiliki untuk menguji produk Anda dengan benar. Kami menjalankan pengujian kegunaan dengan *platform* pendidikan kami. Kami melakukan brainstorming tentang fitur-fitur yang harus dimiliki *platform*. Dalam bab ini juga dibahas tentang bagaimana pengujian setelah *software* tersebut dibuat.

Metode-metode yang digunakan dalam pengembangan software juga akan dibahas dalam buku ini, serta bagaimana mengimplementasikan model-model tersebut dalam pengembangan software. Setelah pengujian software yang dikembangkan, maka *Feedback* yang didapatkan akan diolah. Hasil dari feedback tersebut akan diolah untuk pengembangan perangkat lunak tingkat lanjut. Banyak ilmu menarik yang ada dalam buku ini. Akhir kata semoga buku ini berguna untuk para pembaca.

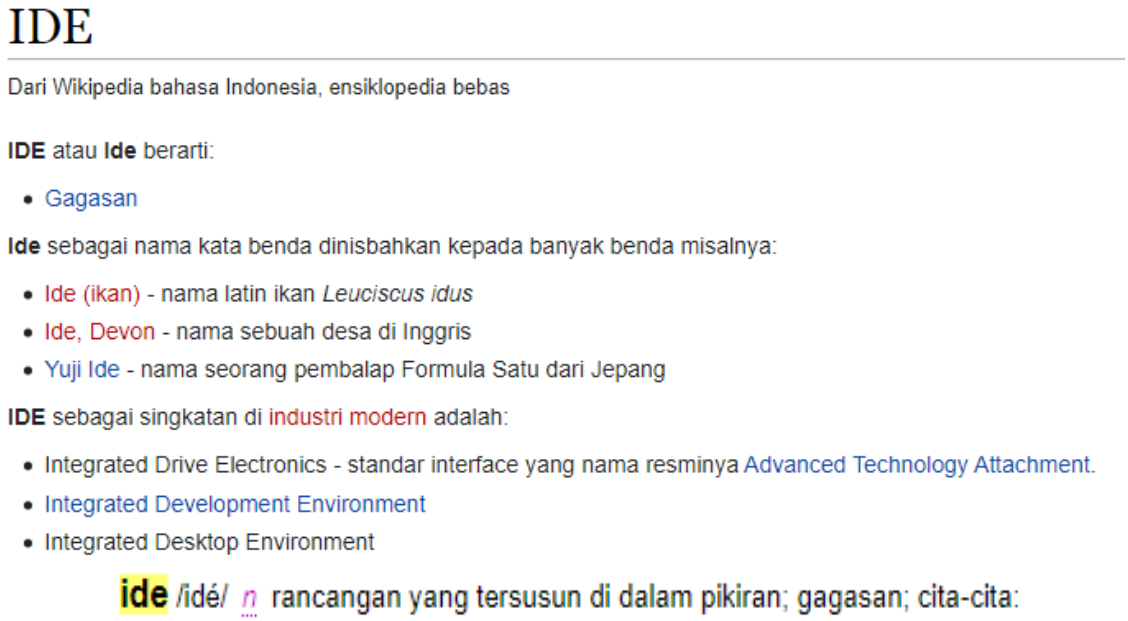
Semarang, November 2021
Penulis,

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

DAFTAR ISI

BAB 1
PENDAHULUAN

Pertama, kita ingin berada di halaman yang sama, tidak hanya tentang buku ini tetapi juga konsep dan gagasannya. Mari kita definisikan kata ide dan menetapkan gagasan dari ide yang akan kita bicarakan dalam buku ini. Cara termudah adalah dengan Google, betul bukan (Gambar 1-1)?


IDE

Dari Wikipedia bahasa Indonesia, ensiklopedia bebas

IDE atau **Ide** berarti:

- Gagasan

Ide sebagai nama kata benda dinisbahkan kepada banyak benda misalnya:

- Ide (ikan) - nama latin ikan *Leuciscus idus*
- Ide, Devon - nama sebuah desa di Inggris
- Yuji Ide - nama seorang pembalap Formula Satu dari Jepang

IDE sebagai singkatan di industri modern adalah:

- Integrated Drive Electronics - standar interface yang nama resminya Advanced Technology Attachment.
- Integrated Development Environment
- Integrated Desktop Environment

ide /idé/ *n* rancangan yang tersusun di dalam pikiran; gagasan; cita-cita:

Gambar 1-1. Definisi Ide Menurut Wikipedia Dan Kbbi

Dalam buku ini saya akan menggunakan definisi pertama sebagai dasarnya. Ide adalah gagasan, pemikiran atau saran yang mengarah pada tindakan. Sinonim yang muncul di bawah definisi dapat sepenuhnya digunakan sebagai kata kunci untuk menggambarkan buku ini. Bersiaplah untuk menghadapi banyak kata ini: rencana, desain, proyek, tujuan. Saya juga menyukai etimologi—"dari dasar idein *'untuk melihat'*". Segala sesuatu yang datang sebagai ide adalah visualisasi mental dari sesuatu. Sesuatu ini bisa benar-benar baru atau bisa berasal dari hal-hal yang sudah ada.

Hukum umum manusia adalah bahwa segala sesuatu yang dapat kita lihat dapat kita bangun, sehingga setiap visualisasi atau imajinasi yang kita miliki sebenarnya dapat diubah menjadi suatu produk. Jadi, dalam buku ini kita akan berbicara tentang ide-ide yang mengarah pada tindakan yang direncanakan dengan hati-hati di mana tujuannya adalah untuk mengubah ide menjadi sebuah produk. Sekarang, mari kita definisikan produk dalam konteks buku ini. Sekali lagi, sobat Google mendefinisikan produk sebagai hasil dari suatu tindakan atau proses. Semua kata ini sama dan sangat penting dalam konteks itu. Hasil, tindakan, proses. Tanpa tindakan, ide Anda akan tetap menjadi ide selamanya. Tanpa proses implementasi yang jelas, Anda tidak akan pernah mencapai tujuan Anda. Dan tentu saja,

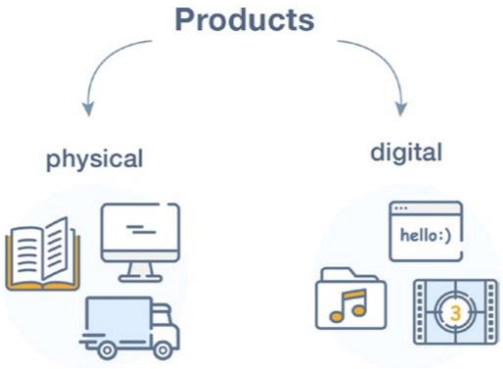
Anda mengharapkan hasil! Anda menentukan hasil yang diharapkan dari ide Anda dan Anda menetapkan proses untuk mencapainya.

Produk adalah hasil dari serangkaian tindakan yang didefinisikan sebagai kebutuhan untuk mengimplementasikan ide.

Ada banyak sekali jenis dan kategori produk, biasanya berupa bisnis, konsumen, atau keduanya; produk konsumen, pada gilirannya, dapat dibagi menjadi empat kelompok yakni belanja, kenyamanan, khusus, dan produk yang tidak dicari. Tujuan buku bukan memberi Anda wawasan tentang bisnis dan pemasaran, sehingga sayaakan mengelompokkan produk menjadi dua jenis (Gambar 1-2):

- Produk fisik: stapler, notebook, pulpen, mobil, mainan, dll.
- Produk digital: program perangkat lunak, film, e-book, dll.

Seperti yang sudah saya tunjukkan, kita tidak akan membicarakan semua jenis produk dalam buku ini. Tujuan dari buku ini adalah untuk menunjukkan bagaimana perangkat lunak dibuat; oleh karena itu kita akan berbicara tentang bagian yang sangat khusus dari kategori produk digital: produk perangkat lunak. Oleh karena itu, ide-ide yang akan kita bicarakan adalah ide-ide yang mengarah pada pembuatan perangkat lunak.



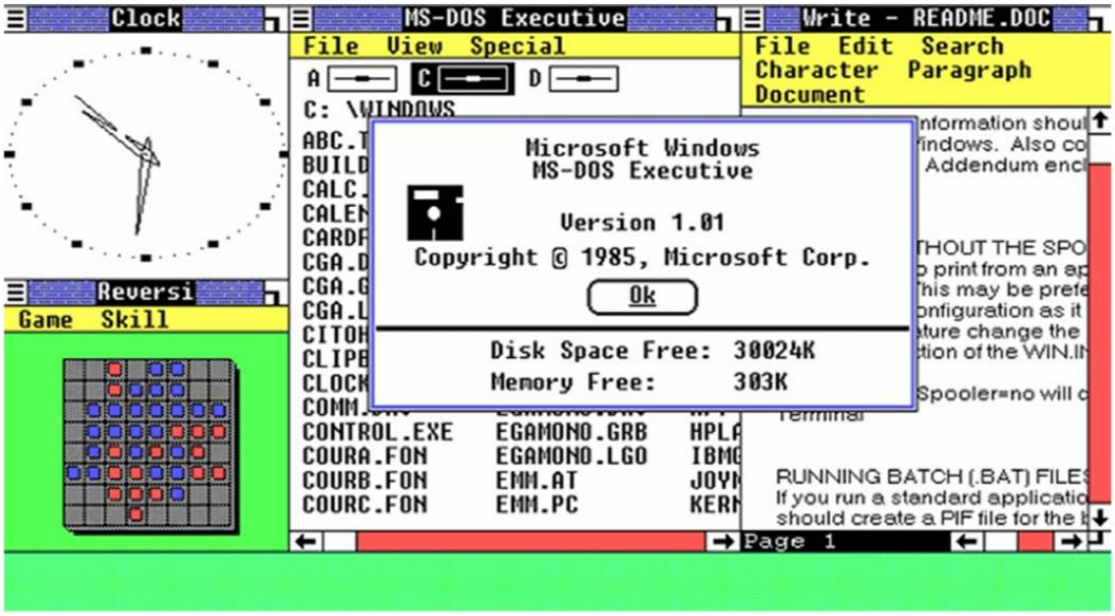
Gambar 1-2. Kategorisasi produk

1.1 Produk Perangkat Lunak dan Ide di Baliknya

Mungkin produk perangkat lunak yang paling banyak digunakan adalah sistem operasi. Sistem operasi Microsoft Windows telah melakukan perjalanan besar sejak versi pertamanya pada tahun 1985 (Gambar 1-3). Pekerjaan sebenarnya pada GUI di Microsoft dimulai pada tahun 1981 ketika Microsoft memutuskan untuk membangun antarmuka perangkat lunak yang dipetakan sedikit. Setelah itu, Bill Gates melihat demonstrasi VisiOn (https://en.wikipedia.org/wiki/Visi_On) perangkat lunak GUI untuk komputer yang kompatibel dengan IBM. Dia juga menyadari betapa canggihnya GUI Apple, dan Microsoft telah bekerja sama dengan Apple pada aplikasi dengan antarmuka pengguna untuk Apple Computer baru. Bill Gates memutuskan antarmuka pengguna Microsoft harus berbeda. Versi pertama MS Windows memiliki beberapa aplikasi seperti kalkulator, kalender, Jam, Notepad, Paint, Reversi, Terminal, Cardfile, dan Write. Ini tampak seperti yang ditunjukkan pada Gambar 1-3.

Sulit membayangkan bagaimana versi modern dari MS windows benar-benar berevolusi. Itu bahkan tidak dianggap sebagai sistem operasi: sampai versi ketiga Windows

dianggap sebagai lingkungan operasi. Versi pertama diberi nama kode "Manajer antarmuka." Selama bertahun-tahun program ini telah ditingkatkan, dikembangkan, dan dipoles. Versi terbaru tidak ada hubungannya dengan versi pertama. Sangat penting untuk disebutkan bahwa terlepas dari peluang bisnis yang besar, Microsoft telah berada di tangannya sebagai salah satu perusahaan terbesar di dunia dan terlepas dari kritik keras yang diterima Windows 1.0, Bill Gates selalu menekankan bahwa Windows adalah produk utama Microsoft yang mendefinisikan seluruh arah dan masa depan. Omong-omong, jika Anda ingin mengerjai teman Anda, terutama pengguna Mac atau Linux, inilah simulator windows XP yang sangat bagus: http://geekprank.com/. Berbicara tentang Microsoft mengingatkan saya betapa mahal layanan dan produk mereka, yang pada gilirannya mengingatkan saya tentang sistem pembayaran digital. Cukup Microsoft untuk saat ini; mari kita bicara tentang PayPal (Gambar 1-4).



Gambar 1-3. Microsoft Windows versi pertama pada tahun 1985

Gambar 1-4. Beranda PayPal saat ini

Banyak orang mengira PayPal adalah salah satu penemuan Elon Musk yang dia jual untuk mulai bermain dengan ide-ide gilanya. Itu tidak sepenuhnya benar. PayPal adalah bagian kecil dari perangkat lunak yang disebut Confinity (https://en.wikipedia.org/wiki/Confinity). Penciptanya memutuskan untuk berkonsentrasi hanya pada bagian ini setelah mereka menyadari bahwa itu memecahkan masalah besar bagi pengguna eBay. Artikel yang ditemukan di https://www.entrepreneur.com/article/228206 menunjukkan dengan cara yang sangat bagus bagaimana seseorang harus fleksibel tentang ide-ide mereka dan tidak takut untuk mengubah arah seluruh perusahaan setelah menyadari bahwa itu mungkin memecahkan masalah nyata; masalah pelanggan.

Apa peran Elon Musk dalam hal ini? Dalam kisah PayPal ini, Elon Musk sebenarnya adalah seorang visioner besar, karena setelah menggabungkan perusahaannya (X.com) dengan Confinity, ia memutuskan untuk menghentikan semua pengembangan lainnya dan hanya fokus pada layanan uang PayPal. Karena kita sudah cukup banyak googling di bab ini, mari kita bicara tentang Google! Kembali di tahun 90-an, dua mahasiswa PhD muda dari Universitas Stanford, Larry Page dan Sergey Brin, mengembangkan algoritma pencarian yang disebut BackRub. Algoritme ini segera menjadi sangat populer di Stanford dan mulai menggunakan begitu banyak bandwidth server Stanford sehingga kedua jenius itu menjatuhkan gelar PhD mereka dan memutuskan untuk membangun perusahaan di sekitarnya. Mengapa BackRub berubah menjadi Google? Pada titik tertentu mereka memutuskan bahwa nama BackRub tidak cukup baik dan membentuk sesi brainstorming. Selama sesi ini Lawrence , mahasiswa pascasarjana saat itu, muncul dengan nama googolplex. Page menyukainya dan mengusulkan untuk mempersingkatnya menjadi Googol. Saat mencari di registri nama domain untuk menentukan apakah nama Googol sudah digunakan, Sean membuat kesalahan dan mengetik "google.com." Domain ini gratis, dan semua orang menyukai kesalahannya. Sehingga langsung terdaftar dan dikenal oleh semua orang sejak saat
itu. Lihat sejarah singkat di balik nama Google di sini: https://graphics.stanford.edu/~dk/google_name_origin.html.

Saat ini kami menggunakan banyak layanan google selain mesin pencari: Google+, Google analytics, solusi cloud Google, Gmail, dll. Seperti yang Anda lihat, perusahaan perangkat lunak besar selalu memulai di suatu tempat, dan biasanya "suatu tempat" ini adalah ide seseorang. Jangan takut dengan ide-ide Anda, jangan takut untuk membagikannya, dan jangan takut untuk maju bersamanya. Satu dari jutaan spermatozoid mau tidak mau menjadi manusia, dengan cara yang sama salah satu dari banyak ide, jika didorong dan direncanakan dengan baik, memiliki kekuatan untuk menjadi Google lain.

1.2 Model yang berbeda

Perusahaan yang berbeda membangun produk yang berbeda yang melayani kebutuhan yang berbeda. Beberapa produk melayani kebutuhan individu (misalnya, sistem operasi), sedangkan produk lain digunakan untuk kebutuhan bisnis perusahaan lain. Misalnya, platform e-commerce yang dapat dijual ke berbagai perusahaan untuk kebutuhan e-commerce mereka.

Produk yang memenuhi kebutuhan pelanggan individu disebut B2C (business to consumer/bisnis ke konsumen) sedangkan yang memenuhi kebutuhan bisnis lain disebut B2B (bisnis ke bisnis).

Ada banyak bentuk pemasaran lainnya—G2B (government to business), B2G (business to government), bahkan C2C (consumer to consumer). Dalam model ini produk memfasilitasi pelanggan untuk memberikan layanan atau menjual produk kepada pelanggan lain. Bagaimana Anda memutuskan apa yang harus dilakukan produk Anda dan jenis kebutuhan apa yang harus ditargetkan sejak awal? Jalankan penelitian. Kemudian kembangkan. Mari kita bicara tentang penelitian dan pengembangan.

1.3 Penelitian dan Pengembangan

Di bagian sebelumnya, kita membahas bagaimana ide mengubah perusahaan yang sudah ada dan bagaimana ide tersebut dapat memunculkan perusahaan baru. Tentu saja, tidak semudah seseorang hanya berbaring di bawah pohon dan menunggu apel jatuh di kepala mereka. Dalam dunia teknologi yang cepat berubah dan sangat kompetitif saat ini, harus ada beberapa struktur organisasi yang mengarah pada generasi ide untuk produk baru atau bahkan ide untuk meningkatkan yang sudah ada, membuatnya menarik di pasar. Bahkan, beberapa perusahaan memiliki departemen Riset dan Pengembangan yang cukup besar.

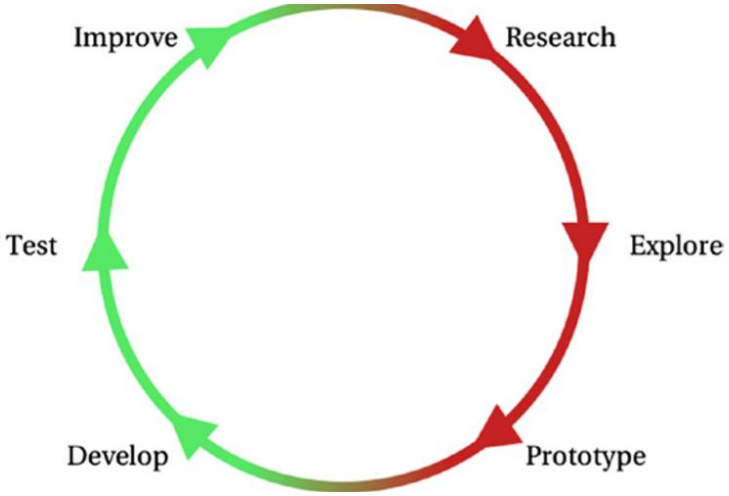
Ada berbagai cara untuk mengatakan "penelitian dan pengembangan." "R-and-D," "Rn'D," "R&D," "R+D," atau bahkan "RTD" (Research and Technological/penelitian dan pengembangan teknologi)—istilah-istilah ini semuanya sama.

Cukup sulit untuk memberikan nilai yang tepat kepada departemen R&D dalam sebuah perusahaan. Penelitian untuk inovasi dan kreativitas biasanya bertujuan untuk kesuksesan jangka panjang; oleh karena itu tidak mudah untuk mengevaluasi nilainya saat ini. Perusahaan harus memutuskan berapa banyak yang akan diinvestasikan ke dalam R&D. Biasanya, semakin teknologi suatu perusahaan, semakin banyak uang yang dihabiskan untuk R&D. Misalnya, di Indonesia, perusahaan industri biasa menghabiskan tidak lebih dari 3% untuk R&D, sedangkan perusahaan berteknologi tinggi menginvestasikan 24% pendapatannya untuk itu (dari Wikipedia: https://en.wikipedia.org/wiki/Research_and_development).

Jumlah (dalam persentase) dari apa yang diinvestasikan perusahaan dalam litbang disebut intensitas litbang.

Tentu saja, berinvestasi lebih banyak dalam R&D tidak berarti bahwa produk Anda akan menjadi yang paling inovatif dan paling kreatif. Namun, beberapa penelitian telah membuktikan bahwa perusahaan dengan proses R&D yang stabil mengungguli perusahaan yang tidak memiliki investasi dalam R&D sama sekali. Seperti apa prosesnya? Ini adalah siklus (Gambar 1-5). Teliti, jelajahi, buat prototipe, kembangkan, uji, tingkatkan, riset, jelajahi....dan seterusnya. Ulangi tindakan ini selamanya karena produk Anda tidak akan pernah sempurna! Proses tak terelakkan yang ditinggalkan kepada kita oleh karya-karya Darwin lama kita. Dunia berkembang setiap hari; orang berubah serta kebutuhan mereka. Hal-hal yang relevan hari ini dapat benar-benar usang besok. Perangkat lunak yang telah

kami gunakan dan sukai dapat terlihat ketinggalan zaman hari ini. Itulah mengapa R&D penting tidak hanya untuk menciptakan produk baru tetapi juga untuk terus meningkatkan produk yang sudah ada. Oleh karena itu, diagram proses ini adalah siklus.



Gambar 1-5. Siklus Hidup Proses Penelitian Dan Pengembangan

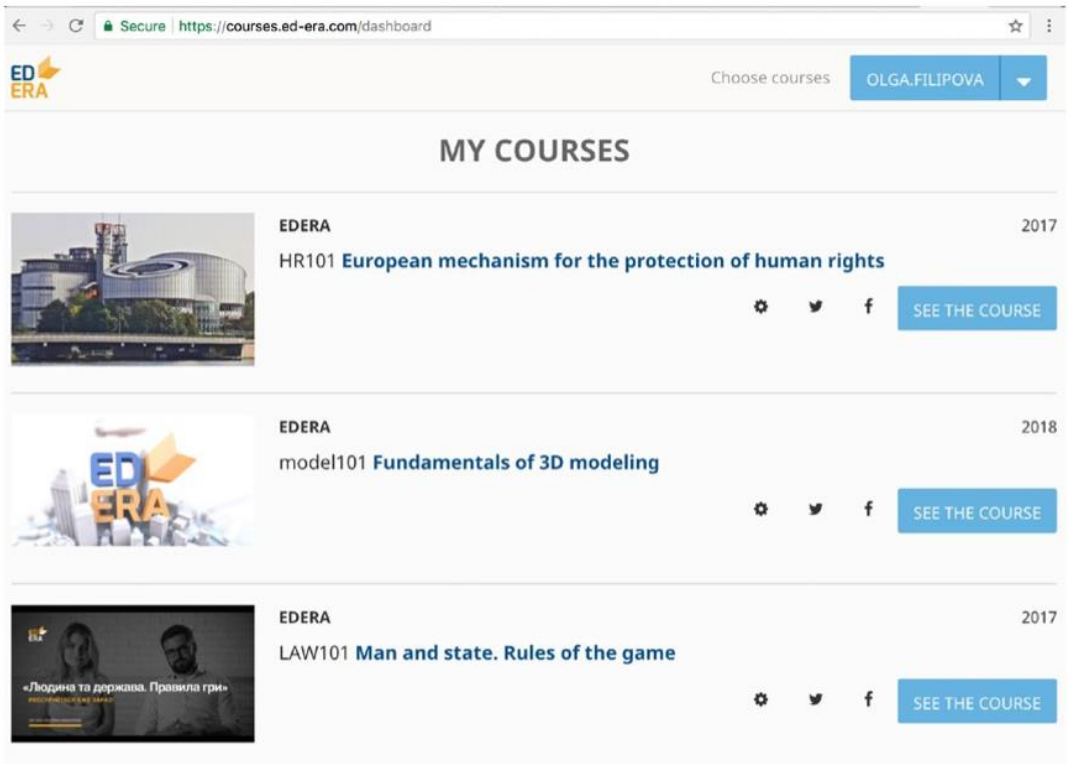
Siapa yang harus menjalankan penelitian? Ini mungkin berbeda dari perusahaan ke perusahaan. Di beberapa perusahaan, penelitian dijalankan oleh pakar pemasaran, pengembangan bisnis, dan analitik; perusahaan lain menangani tugas penelitian kepada para insinyur; beberapa perusahaan tidak memiliki departemen R&D sama sekali tetapi mencoba mengatur proses kerja mereka sedemikian rupa sehingga R&D menjadi bagian dari rutinitas sehari-hari. Kami telah melihat banyak hal berbeda terjadi. Di OptioPay R&D dijalankan oleh pemilik bisnis, pemilik produk, pakar analitik, dan insinyur.

Dasbor analitik dibuat oleh pakar analisis bisnis di atas metrik langsung yang dikumpulkan dari sistem yang berjalan dalam produksi. Metrik ini terus didiskusikan dalam pertemuan mingguan oleh semua pemangku kepentingan. Selama pertemuan ini, kami memikirkan kemungkinan solusi yang akan meningkatkan jumlah metrik yang ada karena kami juga menghasilkan metrik baru untuk mengukur kesuksesan kami. Berdasarkan diskusi tersebut, terkadang muncul ide-ide baru, membuka jalan menuju peluang bisnis baru. Kami juga bekerja di perusahaan tempat penelitian dilakukan oleh para insinyur— mereka memeriksa laporan data yang sangat besar, membandingkan angka dengan produk yang ada dari pesaing, memikirkan solusi untuk ditingkatkan, dan langsung mencobanya karena mereka tahu cara menulis kode. Sangat nyaman, omong-omong.

Di EdEra (perusahaan pendidikan online yang kami wakili sebagai co-founder teknis), R&D dijalankan oleh orang yang berdedikasi. Orang ini menjalankan penelitian mendalam tentang proses pendidikan. Analisis yang cermat terhadap berbagai alat, teknik, dan platform pendidikan, seperti Udemy, Khan academy, Zenius Education, EdX, Duolingo, Ruang Guru, dll., memungkinkan kita untuk membangun ide tidak hanya tentang apa yang ingin dipelajari oleh orang kontemporer modern, tetapi juga bagaimana mereka ingin untuk melakukannya. Ini bukan hanya tentang bagaimana proses kognitif bekerja; ini tentang seberapa cepat masyarakat berubah dan bagaimana kemajuan teknologi memainkan peran yang kuat dalam persepsi kita tentang informasi. Hasilnya, dari studi ini kami mendapatkan

banyak artikel blog tentang gamification, pembelajaran adaptif, pembelajaran mikro, personalisasi, pembelajaran online, pembelajaran campuran, pembelajaran campuran, dll. Tidak dapat dipercaya betapa berbedanya proses pembelajaran sekarang dari yang kita pelajari. Sebenarnya, semua investigasi yang telah kami lakukan selama 3 tahun terakhir membawa kami pada ide untuk membuat platform pendidikan baru.

1.4 Platform Berbagi Pengetahuan



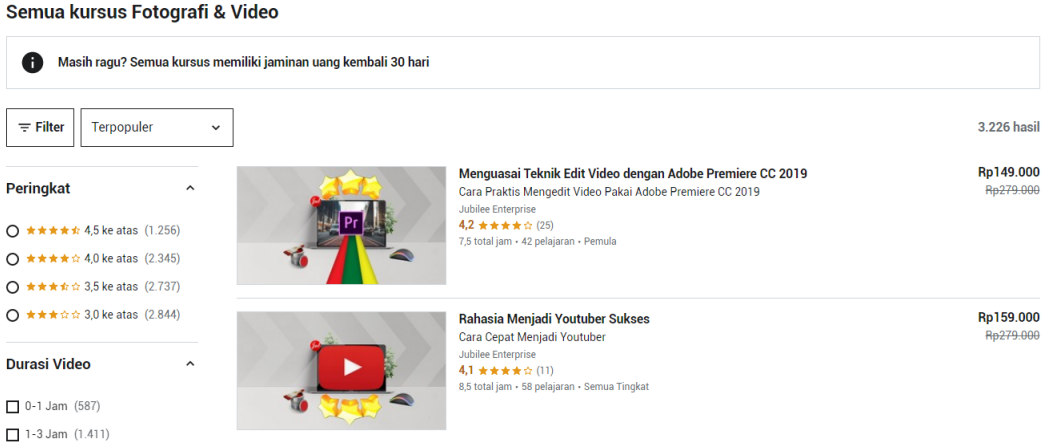
Gambar 1-6a Platform edX Terbuka yang dihosting sendiri: Dasbor kursus EdEra

Saat ini di EdEra kami menggunakan platform sumber terbuka pihak ketiga (Gambar 1-6a) untuk menyelenggarakan dan membagikan kursus online kami. Platform ini disebut Open edX (https://open.edx.org/), dan dikembangkan di MIT. Platform ini dibuat pada tahun 2012, dan sedang dikembangkan oleh lebih dari 300 kontributor dari seluruh dunia dan menawarkan berbagai kemungkinan untuk membuat dan menjalankan MOOC (Massive Open Online Courses). Begitu juga dengan Ruangguru, Udemy dan laiinya. Kami senang menjadi pengguna platform ini, tetapi…

Dimanapun, pasti selalu ada "tapi", kan?

Dalam 3 tahun penggunaan ekstensif platform Open edX, kami terjebak dengan banyak masalah berbeda. Tumpukan teknologi platform itu menakutkan. Basis data, antrean pesan, bahasa pemrograman, dan teknologi yang berbeda terikat bersama, menawarkan sistem yang sangat erat dan sulit untuk diskalakan, dipelihara, dan ditingkatkan secara horizontal. Sistem ini sangat kompleks sehingga Anda dapat menemukan perusahaan di luar sana yang menawarkan layanan pemasangan, peningkatan, dan pemeliharaan Open edX. Layanan ini tidak murah sama sekali. Jadi, apakah Anda menghabiskan banyak waktu untuk

melakukannya sendiri, atau Anda membayar banyak uang kepada perusahaan-perusahaan lain dan mereka akan melakukannya untuk Anda. Dalam setiap kasus, perangkat lunak open source gratis, akan tetapi tidak akan sebebas yang ditawarkan.



Gambar 1-6b Platform Udemy.

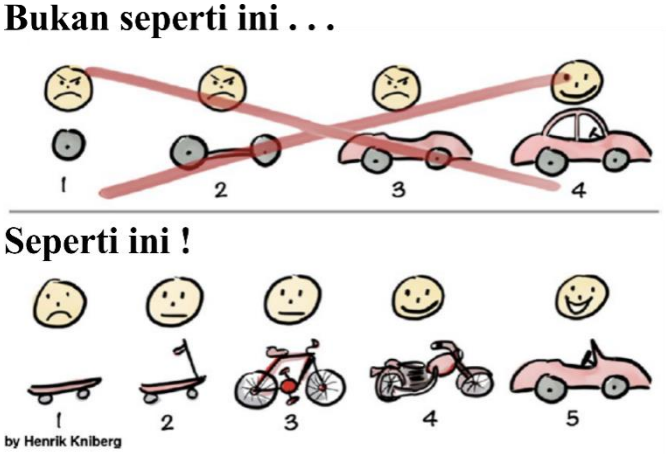
Ide membangun platform in-house lahir sejak lama. Tidak ada R&D yang terlibat, tidak ada apel yang jatuh di kepala, tidak ada bangun di pagi hari dengan pencerahan tiba-tiba. Mengarah pada gagasan yang ada, Anda harus berjuang dengan platform pihak ketiga "Ayo buat milik kita sendiri!" Anda bahkan tidak dapat membayangkan berapa kali kami membicarakan hal itu. Sekarang kami memiliki banyak penelitian terstruktur di atas ide ini yang dapat membantu kami membangun platform pembelajaran yang bagus. Tentu saja, dalam jangka pendek, kami tidak akan dapat membangun platform dengan semua karakteristik pembelajaran adaptif, pembelajaran mikro, augmented reality, gamifikasi dan personalisasi yang mewah, tetapi kami dapat mencoba membangun sistem yang solid yang memungkinkan kami mudah menambahkan fitur yang berbeda dan meningkatkan fungsionalitas yang ada sambil terukur dan dapat dipelihara. Untuk ini kita perlu memahami bahwa kita ingin membangun versi pertama yang sangat mendasar dari suatu produk. Dalam bisnis, versi seperti itu disebut MVP.

1.5 Produk Layak Minimum (*Minimum Viable Product/MVP*)

MVP (*Minimum Viable Product*) adalah produk yang memiliki fitur yang cukup untuk membuktikan konsepnya. Ini juga memungkinkan seseorang untuk membangun fitur di atasnya. Cukup baik untuk ditunjukkan kepada investor dan meminta uang untuk pengembangannya. Cukup baik untuk menggunakannya dengan cara dasar. Pernahkah Anda mendengar ungkapan, “Kita perlu membuat sepatu roda dulu”? Orang sering mengatakannya dengan tepat dalam konteks yang diterapkan pada MVP (Gambar 1-7).

Apa artinya? Jika Anda ingin membuat mobil, Anda bisa mulai dengan membuat ban, lalu roda, lalu bangkai, dan kemudian sampai ke seluruh mobil, sehingga membuat seluruh produk hanya dapat digunakan jika sudah benar-benar dibuat. MVP berarti pertama Anda membuat sepatu roda, lalu Anda membuat skuter tendangan, lalu Anda membuat sepeda, dan kemudian Anda membuat mobil Anda. Proses ini lebih lambat, tetapi Anda memiliki sesuatu yang berfungsi di setiap iterasi. Dalam buku ini kita akan membangun skate kita

untuk platform pembelajaran kita. Apa itu MVP untuk platform pembelajaran? Nah, platform pembelajaran harus menawarkan cara untuk belajar, bukan? Oleh karena itu, kita dapat menulis daftar hal-hal dasar yang diperlukan seseorang untuk dapat belajar.



Gambar 1-7. Ilustrasi terkenal oleh Henrik Kniberg tentang MVP

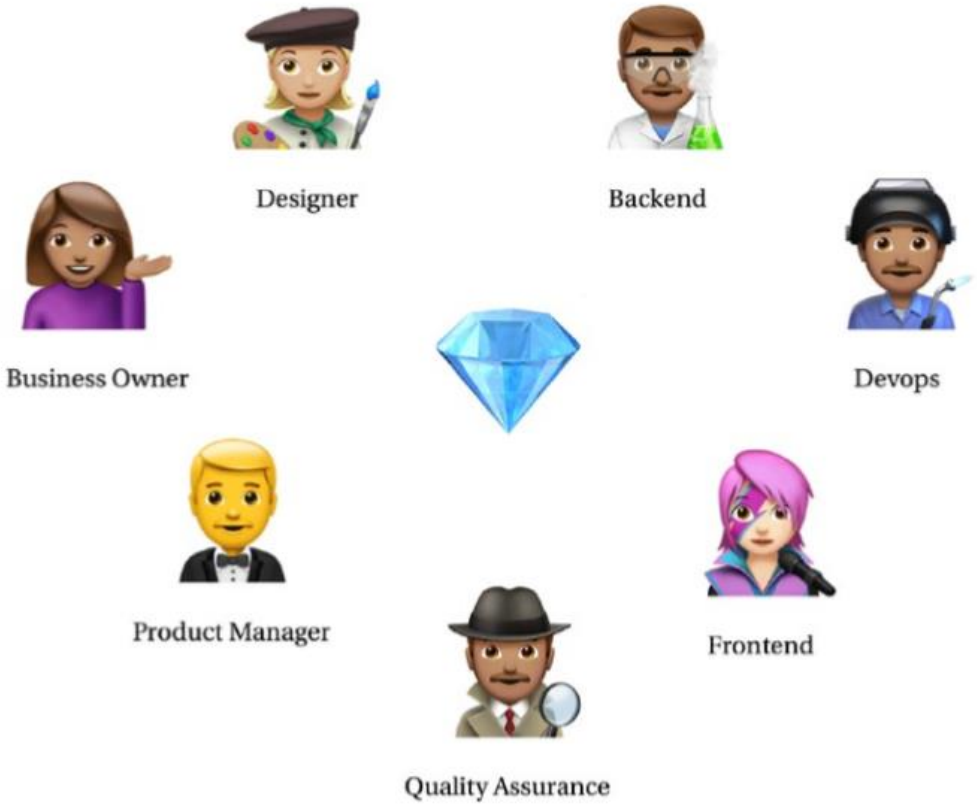
Proses pembelajaran online biasanya terdiri dari dua tahap sederhana—ada beberapa bahan untuk menyerap pengetahuan (bisa berupa video, atau buku, atau hanya sepotong teks), dan ada cara untuk memeriksa pengetahuan yang diperoleh (bisa berupa kuis, tes, daftar pertanyaan atau penilaian peer-to-peer di lingkungan yang lebih kompleks). Saya akan mengatakan bahwa jika kita membangun sesuatu yang memungkinkan pengguna untuk membaca beberapa materi pembelajaran atau menonton video pembelajaran dan kemudian mengikuti kuis, maka kita akan baik-baik saja. Kami akan mendefinisikan daftar persyaratan yang lebih lengkap di bab-bab selanjutnya. Dan kami akan membangun MVP kami dari awal hingga penerapannya. Apakah Anda tahu siapa yang akan membantu kami melakukan ini? Anda, pembaca kami yang terkasih. Anda akan membantu kami dari versi pertama platform pembelajaran, dan dengan kreasi ini Anda akan belajar bagaimana mengubah ide menjadi produk yang berfungsi penuh

1.6 Kesimpulan

Dalam bab ini kita membahas ide-ide yang berbeda dan kita telah melihat bagaimana ide-ide itu diubah menjadi produk. Kami telah mendekati berbagai jenis produk dan berbagai jenis kebutuhan yang dapat mereka layani. Kami menjelaskan bagaimana kami sampai pada ide yang akan kami terapkan dalam buku ini. Kami mempersiapkan Anda untuk bekerja sama dengan kami dalam proyek ini. Apa yang akan menjadi peran Anda? Atau, mungkin, mari kita ajukan pertanyaan ini dari perspektif yang berbeda: peran apa yang diperlukan untuk membangun perangkat lunak? Kami masih berbicara tentang perangkat lunak, meskipun kami berbicara tentang sepatu roda di bab ini. Ada banyak peran berbeda yang dapat terlibat dalam membawa ide Anda ke panggung MVP. Dan Anda akan mencoba setiap peran penting selama penulisan buku ini. Dalam bab berikutnya, kita akan berbicara tentang peran yang topinya akan kita kenakan di sepanjang buku ini.

BAB 2
PERAN, TANGGUNG JAWAB DAN METODOLOGI

Sekarang setelah Anda memiliki ide, saatnya untuk mengumpulkan tim Anda. Secara umum, memiliki satu orang saja untuk mengurus semua bukan ha yang baik. Pertama, sangat tidak mungkin seseorang memiliki semua keterampilan untuk membawa produk melalui semua tahapan ke kehidupan; kedua, bahkan jika secara hipotetis orang seperti itu ada, selalu lebih baik setiap bagian dari produk memiliki setiap orang yang bertanggung jawab sehingga jumlah perhatian yang sama didistribusikan secara adil. Benar juga jika Anda bekerja di perusahaan yang sudah mapan di mana semua tim sudah ada dan menetap, tetapi tentu saja itu tidak berlaku jika Anda hanya punya ide dan ingin membangun perusahaan dari itu. Dalam hal ini, Anda harus siap pada awalnya untuk melakukan sedikit dari segalanya.



Gambar 2-1. Tim Di Sekitar Produk

Bayangkan Anda sedang mencari investor. Ketika Anda melakukan hal seperti itu, seharusnya Anda sudah memiliki beberapa demo atau bukti konsep, sehingga Anda dapat menunjukkannya kepada calon investor Anda, ini merupakan hal yang umum yang berarti bahwa pada titik tertentu Anda dan mitra Anda harus membuat serangkaian fitur yang menurut Anda penting untuk ditampilkan dan mampu menarik perhatian calon investor Anda bersama dengan beberapa desain produk untuk membuatnya menarik dan beberapa implementasi aktual untuk ditampilkan.

Beberapa interaksi harus dilakukan, agar mereka mendapatkan perasaan konkret tentang apa yang ingin Anda capai. Jika Anda ingin ide Anda menjadi nyata, bukti konsep dan demo berjalan adalah yang paling penting, karena ini akan menunjukkan kepada orang luar seberapa serius dan komitmen Anda terhadap proyek tersebut. Bagaimanapun, bab ini mendekati topik ini seolah-olah Anda bekerja di perusahaan di mana semua tim sudah ada. Kami akan memandu Anda melalui peran dan tanggung jawab masing-masing departemen serta bagaimana merencanakan proyek Anda dan semua proses yang terlibat.

2.1 Peran dan Tanggung Jawab

Ada beberapa peran yang dapat Anda identifikasi di sebagian besar perusahaan di mana perangkat lunak adalah pemain yang kuat. Sebagian besar perusahaan saat ini beroperasi melalui Internet dan menjadikannya semacam perusahaan perangkat lunak juga, bahkan jika bisnis utama mereka tidak terkait dengan perangkat lunak. Bayangkan, misalnya, sebuah perusahaan yang memiliki beberapa toko buku tersebar di suatu negara. Ketika Internet mulai menjadi lebih menonjol, bisnis-bisnis ini harus mencoba untuk bertahan hidup, di mana hal yang seharusnya dilakukan adalah membuat toko online untuk toko offline mereka.

Perusahaan sebesar jaringan toko buku sudah memenuhi syarat untuk memiliki departemen produk dan perangkat lunak mereka sendiri dengan solusi internal untuk mengatasi tantangan mereka. Idenya di sini adalah, untuk menunjukkan secara umum, peran yang akan kami uraikan tidak hanya berlaku untuk perusahaan perangkat lunak tetapi juga untuk perusahaan mana pun di mana perangkat lunak memainkan peran penting dalam bisnisnya. Gambar 2-1 menunjukkan tim lintas fungsi yang akan kita bicarakan (perhatikan, kelompok orang ini mungkin sangat bervariasi!).

2.2 Business Owner/Pemilik bisnis

Terkadang orang mengalami kesulitan membedakan antara pemilik bisnis dan manajer produk (dijelaskan secara rinci di bagian selanjutnya), yang dapat dimengerti karena tidak semua perusahaan memiliki yang pertama. Kemungkinan perusahaan kecil tidak memilikinya karena pada awalnya mereka cenderung berfokus terutama untuk mengeluarkan produk di pasar.

Selama ini cukup umum bahwa orang yang menjalankan perusahaan bersama dengan manajer produk entah bagaimana menutupi sebagian besar tanggung jawab pemilik bisnis. Ketika sebuah perusahaan tumbuh dan beberapa bisnis lain ikut bermain didalamnya, maka akan semakin jelas bahwa posisi seperti itu harus ada. Mempertimbangkan contoh platform online, jika kami melihat sekilas, model bisnisnya tampak cukup jelas: kami menjual kursus online kepada pelanggan. Ini adalah model bisnis-ke-konsumen (B2C) yang khas.

Ketika perusahaan menjadi sukses, mereka menjadi magnet bagi bisnis baru, terutama dengan perusahaan lain. Semua orang ingin bergaul dengan anak-anak keren, bukan? Model ini disebut business-to-business (B2B). Seperti yang Anda lihat, sekarang tidak hanya platform kami yang perlu berurusan dengan pelanggan langsung mereka, tetapi perusahaan lain juga semacam pelanggan atau, lebih tepatnya secara politis, mitra. Menurut

kami ini adalah titik balik dalam sebuah perusahaan, ketika satu atau lebih pemilik bisnis mulai dibutuhkan: ketika sebuah perusahaan mulai menjalankan beberapa bisnis secara paralel atau membawa kemitraan baru yang membutuhkan pengembangan perangkat lunak lebih lanjut dan berdedikasi.

Pemilik bisnis biasanya dapat membuat keputusan untuk perusahaan dan bertanggung jawab untuk mengidentifikasi kemungkinan bisnis dan kemitraan baru. Mereka juga bertanggung jawab untuk membuat semacam penilaian dan memahami apakah kemitraan semacam itu akan membawa nilai bagi perusahaan. Nilai di sini tidak berarti pendapatan saja. Perusahaan memiliki beberapa kepentingan, dan pada titik tertentu, bahkan jika suatu kerjasama tidak menghasilkan uang langsung, pada akhirnya mungkin masih menjadi kemitraan yang baik untuk strategi perusahaan. Pemilik bisnis bukanlah orang teknis. Mereka tidak akan menentukan bagaimana hal-hal akan dilakukan. Mereka bekerja pada tingkat bisnis abstrak dengan mendefinisikan masalah, bagaimana hal itu dapat ditangani, dan bagaimana mengukur keberhasilannya

Tanggung jawab lain dari pemilik bisnis adalah menegosiasikan persyaratan kemitraan atau menetapkan harga produk baru. Juga, kelangsungan hidup proyek baru itu sendiri adalah sesuatu yang biasanya dijalankan oleh pemilik bisnis. Ini juga termasuk, misalnya, memeriksa apakah solusi tersebut legal dan apakah perusahaan akan menghadapi masalah hukum di masa depan karena melanggar beberapa paten atau melakukan pendekatan pasar ilegal. Sering dilupakan, tetapi penting, adalah melacak metrik bisnis dan indikator kinerja utama setelah proyek selesai. Tidak ada produk yang sempurna.

Semua dari mereka memiliki kekurangan mereka. Penting bagi pemilik bisnis untuk melacak metrik tersebut untuk meningkatkan produk dan menanggapi kemungkinan perubahan pasar. Ini bisa sesederhana mengambil metrik tentang bagaimana orang menggunakan filter di situs web—ini memberi kita wawasan tentang apa yang sebenarnya dicari orang. Pemilik bisnis bekerja sama dengan manajer produk untuk menghasilkan konsep yang dapat diimplementasikan lebih lanjut oleh tim teknis.

2.3 Manajer produk

Manajer produk dapat dilihat sebagai pengatur antara pemilik bisnis dan tim pengembangan. Tugas umum manajer produk termasuk memprioritaskan tugas, mendukung tim non-teknis seperti pemasaran dan konten, dan mengumpulkan umpan balik, di antara tugas-tugas lainnya. Peran manajer produk penting karena mereka bertanggung jawab untuk menerjemahkan ide dan konsep ke dalam produk atau fitur untuk perusahaan. Karena mereka perlu mengatur antara beberapa tim dalam sebuah perusahaan, itu sering kali merupakan pekerjaan yang membuat stres. Setiap tim dengan caranya sendiri menganggap permintaannya adalah yang paling penting untuk keberhasilan perusahaan dan akan mendorong manajer produk dengan keras sampai mereka mendapatkan apa yang mereka inginkan.

Seorang manajer produk perlu memiliki beberapa keterampilan agar berhasil memenuhi perannya. *Soft Skill* penting karena mereka perlu berurusan dengan beberapa individu yang berbeda tetapi begitu juga keterampilan teknis, karena merekalah yang akan

menentukan sampai batas tertentu bagaimana hal-hal akan diimplementasikan dan kapan. Itu tidak berarti mereka harus memiliki latar belakang di bidang teknik tentunya, tetapi mereka harus memiliki pengetahuan yang kuat di bidang teknologi untuk membuat penilaian yang cepat dan memberikan jawaban yang cepat kepada para pemangku kepentingan.

Selain itu, tanggung jawab yang biasanya diberikan kepada manajer produk adalah mempersiapkan peta jalan suatu produk dan memastikan produk memenuhi kebutuhan pengguna. Penting bahwa pekerjaan ini dilakukan dengan benar sehingga tim pengembangan ditempati dan menghasilkan nilai bagi perusahaan. Salah satu poin kunci keberhasilan antara manajer produk dan tim pengembangan adalah memahami apa itu hutang teknis dan seberapa sering tim pengembangan melakukan sprint/iterasi konsolidasi adalah. Jangan khawatir jika Anda tidak memahami beberapa konsep seperti hutang teknis atau sprint/iterasi, karena akan dibahas secara rinci nanti.

Setiap hari, manajer produk bertanggung jawab untuk menulis cerita pengguna, yang merupakan deskripsi singkat dari kasus penggunaan yang biasanya berpusat di sekitar kebutuhan pengguna. Bersamaan dengan itu, kriteria penerimaan juga biasa diberikan oleh manajer produk. Pada dasarnya mereka menyatakan bagaimana fitur dapat diberikan secara lengkap.

Biasanya tim jaminan kualitas (QA) mengandalkan kriteria penerimaan cerita pengguna untuk memeriksa apakah implementasinya memenuhi persyaratan yang ditentukan. Umumnya di akhir siklus implementasi dan verifikasi, manajer produk memeriksa apakah cerita pengguna benar-benar memenuhi persyaratan yang ditentukan sebelumnya; dalam kasus lain, QA bertanggung jawab penuh atas persetujuan—bergantung pada kematangan tim atau bahkan perusahaan, proses ini dapat bervariasi dari kasus ke kasus.

2.4 Designers/Desainer

Desain mungkin diperlukan, ini tergantung pada cerita pengguna atau fitur yang berasal dari manajer produk. Desainer penting karena mereka tidak hanya bertanggung jawab untuk membuat antarmuka yang akan berinteraksi dengan pengguna, tetapi juga membawa semacam identitas ke suatu produk atau bahkan perusahaan untuk membuatnya konsisten di antara semua platform. Mereka entah bagaimana merupakan jembatan yang menghubungkan pengguna dengan teknologi yang diekspos suatu produk. Memahami bagaimana pengguna berinteraksi dengan produk dan mempertimbangkannya dalam mendesainnya adalah tanggung jawab yang diberikan kepada tim desain. Interaksi dengan produk mencakup bagaimana orang akan menggunakan atau, jika produk sudah keluar, menggunakan produk—apa yang dapat diklik, transisi, status, fungsi drag-and-drop, dan lain-lain.

Konsistensi antar elemen juga sesuatu yang perlu dipertimbangkan; semua tombol yang dapat diklik memiliki warna atau bentuk yang sama; beberapa contoh elemen di mana konsistensi diperlukan adalah tombol berbentuk hati sebagai sesuatu yang dapat “disukai”, tanda plus yang menambahkan beberapa item ke daftar, dan panah yang menavigasi melalui beberapa formulir input. Tim desain bekerja sangat erat dengan tim pengembangan untuk

menilai apa yang layak untuk implementasi, berapa lama waktu yang dibutuhkan, apakah beberapa animasi menunggu diperlukan saat panggilan sisi server dilakukan, dll.

Terkadang hal yang merupakan tantangan untuk menjaga konsistensi produk harus dipertimbangkan, karena saat ini setiap produk adalah multi-platform—browser yang berbeda, versi seluler pada ukuran layar yang berbeda, dan aplikasi seluler pada sistem operasi yang berbeda—, desain mungkin bertentangan dengan pedoman sistem operasi, yang sering dapat mengakibatkan aplikasi ditolak oleh beberapa vendor. Itu adalah sesuatu yang sering dihadapi para desainer; itu adalah bagian dari pekerjaan sehingga mereka hanya harus menerima kenyataan ini dan mengatasi tantangan tersebut dengan cara terbaik dan dengan menjadi kreatif.

Ketika berbicara tentang desain, dua istilah utama sering muncul: UI dan UX. Yang pertama adalah singkatan dari desain antarmuka pengguna dan yang kedua adalah singkatan dari pengalaman pengguna. Mereka terkadang bercampur dan salah, tetapi mereka cukup mudah untuk dijelaskan. UX sebagian besar berfokus pada interaksi yang dimiliki pengguna dengan produk (pengalaman), apa dan kapan sesuatu dapat diklik, umpan balik kepada pengguna ketika beberapa tindakan selesai, dll. UI sebagian besar berfokus pada penampilan, branding, dan konsistensi—oleh karena itu, antarmuka. Anggap itu sebagai semacam rompi atau kulit produk. Tergantung pada ukuran perusahaan, mungkin ada orang yang berbeda melakukan masing-masing.

Dari pengalaman kami, bahkan perusahaan menengah cenderung mempekerjakan orang yang memiliki pengalaman melakukan keduanya, karena seringkali tidak ada tempat untuk memiliki orang yang berdedikasi melakukan UX saja. Setelah desain selesai untuk beberapa cerita pengguna, mereka diserahkan kepada manajer produk untuk persetujuan. Beberapa interaksi akan terjadi di antara tim manajemen produk, desain, dan pengembangan selama proses ini hingga keputusan akhir dibuat dan diserahkan kepada tim pengembangan untuk implementasi lebih lanjut.

2.5 Backend

Backend adalah entitas produk perangkat lunak yang bertanggung jawab untuk menerima permintaan dari aplikasi klien dan menanganinya dengan menjalankan server khusus yang biasanya dihosting di layanan cloud atau penyedia server. Layanan web Amazon, platform Google Cloud, dan komputasi Microsoft Azure Cloud hanyalah beberapa contoh tempat Anda dapat meng-host backend suatu produk. Ada beberapa jenis layanan web backend (mis., RESTful, WSDL, SOAP) yang memaparkan serangkaian operasi yang dapat digunakan oleh aplikasi frontend atau bahkan layanan integrasi, tetapi dalam buku ini, kami akan fokus pada RESTful. Layanan web RESTful saat ini adalah salah satu yang paling populer karena biasanya hanya mengandalkan protokol HTTP yang tidak memiliki lapisan kompleks lainnya, seperti yang dilakukan WSDL dan SOAP, dan sangat mudah untuk dipahami dan diterapkan. Lebih dari itu akan dibahas nanti dalam buku ini.

Karena itu, tim backend fokus pada mengekspos operasi, sehingga aplikasi frontend dapat mengambil, menyimpan, memodifikasi, dan menghapus entitas data aplikasi. Kembali ke platform kursus online kami, tim backend yang menerapkannya akan menghasilkan titik

akhir untuk mendaftar dan memfilter kursus (mengambil), memulai kursus dan mengerjakan kuis (menyimpan atau membuat), mengubah jawaban mereka (memodifikasi), dan menghapus jawaban sebelumnya (menghapus). Ini hanyalah beberapa contoh kasar tentang kemungkinan operasi yang dapat dilakukan oleh backend aplikasi.

Ketika manajer produk datang dengan fitur atau cerita baru, biasanya implementasi pertama dilakukan oleh tim backend, karena mereka diperlukan oleh sisa produk untuk memenuhi cerita. Dalam beberapa kasus, implementasi frontend dan backend akan dimulai secara paralel, karena selalu ada tugas yang dapat dilakukan tanpa backend, seperti implementasi layar, dan kemudian terhubung. Insinyur backend sering ditantang saat menerapkan operasi backend, terutama pada kinerja. Hal ini disebabkan oleh fakta bahwa backend perlu menangani sejumlah besar permintaan secara bersamaan, karena semua aplikasi akan mengarahkannya.

Pengoptimalan adalah topik besar saat merancang titik akhir backend, cara mengambil data, cara mengatur data, membuatnya sedemikian rupa sehingga mereka dapat dengan mudah mengatasi perubahan di masa mendatang, cara mengintegrasikan dengan layanan eksternal, cache, database, dll. Topik lain bahwa backend harus mengurus adalah otentikasi dan otorisasi. Titik akhir publik versus pribadi, peran pengguna (apa yang dapat diakses pengguna atau tidak), pencabutan akses, dll. Harus ditentukan dengan cermat oleh manajer produk, karena merupakan bagian dari konsep produk secara umum.

Ketika aplikasi mengekspos semacam toko online atau layanan berbasis langganan, backend juga bertanggung jawab untuk menangani pembayaran dengan Penyedia Layanan Pembayaran atau, juga umum saat ini, menerima tanda terima dari toko seluler (Apple, Google, Amazon), memvalidasi tanda terima tersebut, dan bertindak sesuai dengan itu. Ini berarti bahwa sering kali layanan backend bekerja bersama dengan layanan backend lainnya. Ini sering disebut sebagai komunikasi sisi server atau komunikasi server-ke-server. Setelah backend Anda siap, saatnya untuk mulai mengimplementasikan aplikasi frontend kami, aplikasi yang akan berinteraksi langsung dengan pengguna kami.

2.6 Frontend

Aplikasi frontend suatu produk adalah aplikasi yang terlihat oleh pengguna akhir. Secara umum, ketika kita merujuk ke frontend, kita memikirkan aplikasi web yang berjalan di browser. Meskipun itu benar, aplikasi apa pun yang menampilkan antarmuka grafis atau bahkan antarmuka baris perintah dapat dianggap sebagai aplikasi frontend. Secara umum, kita dapat menganggap aplikasi frontend sebagai bagian dari perangkat lunak yang berjalan di sisi klien; ini tidak hanya mencakup aplikasi web tetapi juga seluler dan, baru-baru ini, aplikasi TV.

Dalam konteks buku ini ketika kita mengacu pada aplikasi frontend, kita akan menyebutkan aplikasi web yang berjalan di browser menggunakan JavaScript, HTML, dan CSS. Kita dapat membagi pekerjaan frontend dalam dua modul utama: representasi dan logika. Representasi adalah apa yang dilihat pengguna, antarmuka, bagaimana elemen dirender, dan bagaimana berinteraksi dengannya. Logikanya adalah segala sesuatu yang

menjadikannya sebuah aplikasi, seperti mengambil data, mengubahnya untuk disajikan kepada pengguna, dan menangani permintaan, status, validasi input data, dll.

Bergantung pada ukuran tim atau perusahaan, mungkin ada orang yang didedikasikan untuk tugas frontend tertentu, seperti orang yang hanya melakukan representasi menggunakan HTML dan CSS serta orang yang hanya melakukan pengkodean dalam JavaScript. Di perusahaan rintisan ini sangat sulit ditemukan karena biasanya tidak ada cukup pekerjaan untuk memisahkan posisi ini. Bahkan ada beberapa kasus di mana Anda akan menemukan orang melakukan frontend dan backend sama sekali; biasanya orang dengan kemampuan seperti itu disebut full-stack engineer atau full-stack programmer, tergantung dari pengalaman atau gelar akademisnya. Ketika cerita atau fitur pengguna tiba di meja teknisi frontend, tidak sepenuhnya jelas apakah mereka harus memulai dengan logika atau implementasi desain. Dari pengalaman kami, dan ini cukup umum di perusahaan, implementasi frontend dimulai bahkan ketika desainnya belum final. Dalam kasus seperti itu, insinyur frontend cenderung memulai dengan logika.

Namun demikian, tidak ada aturan keras dalam hal topik ini, itu akan selalu tergantung pada dinamika dan pengalaman tim. Singkatnya, tim frontend bertanggung jawab untuk mengimplementasikan antarmuka dan logika aplikasi yang berinteraksi dengan pengguna. Biasanya ini adalah langkah pengembangan terakhir untuk sebagian besar cerita atau fitur pengguna. Setelah implementasi, tiket diserahkan ke QA untuk pengujian, di mana tiket dapat dipantulkan bolak-balik hingga kriteria penerimaan terpenuhi. Juga cukup umum bahwa selama langkah implementasi ini, pengembangan backend lebih lanjut diperlukan. Terkadang gambaran yang jelas hanya dapat dilihat setelah hal-hal benar-benar dilaksanakan. Meskipun hal ini diterima, secara umum pelajaran harus dipetik dari pengalaman tersebut sehingga dapat dimitigasi di masa depan. Saat tim tumbuh dalam pengalaman dengan bekerja bersama dengan manajemen produk, perubahan backend selama pengembangan frontend berhenti menjadi sebuah kecenderungan.

2.7 *Quality Assurance* / Jaminan Kualitas (QA)

Departemen jaminan kualitas (QA) bertanggung jawab untuk memastikan segala sesuatu yang sampai ke pengguna akhir memenuhi persyaratan dan berfungsi dengan baik. Meskipun ini bisa menjadi definisi QA, kualitas dalam suatu produk dimulai pada tahap awal pengembangan. Selama pengembangan, beberapa jenis pengujian akan ditulis, seperti pengujian unit, di mana komponen secara umum akan diuji secara terpisah; tes integrasi, di mana beberapa komponen diuji untuk bekerja sama; dan juga, tes fungsional, untuk memeriksa apakah persyaratan dipenuhi dengan kriteria penerimaan.

Struktur departemen QA juga tergantung pada ukuran perusahaan atau bahkan apa yang mereka lakukan. Misalnya, ada perusahaan di mana tim pengembangan bertanggung jawab penuh atas QA. Secara umum, bahkan untuk pemula, cukup normal untuk memiliki orang yang berdedikasi melakukan pengujian manual untuk memastikan semuanya memenuhi persyaratan, terutama jika produk memperlihatkan antarmuka grafis.

Beberapa tim QA, tergantung pada pengalamannya, juga menyertakan orang-orang yang dapat menulis tes otomatis. Itu membutuhkan lebih banyak pengetahuan dan

keterampilan pemrograman, karena anggota tim QA perlu menulis kode untuk mencapai hasil. Setelah fitur diimplementasikan, fitur tersebut diserahkan ke QA untuk pengujian. Pengujian semacam ini seringkali merupakan pengujian manual.

Anggota QA akan mengambil tiket dan melakukan beberapa tes untuk menegaskan bahwa semuanya dilaksanakan sesuai dengan kriteria penerimaan. Seiring dengan itu dan karena beberapa perubahan sering dilakukan untuk rilis, tim QA juga akan fokus melakukan pengujian regresi. Pengujian regresi pada dasarnya memastikan bahwa serangkaian perubahan (versi baru) tidak merusak sistem secara keseluruhan. Bayangkan sebuah fitur di mana sekarang, selain hanya membayar dengan Visa dan MasterCard, orang juga dapat membayar dengan PayPal.

Pengujian regresi dalam lingkup ini adalah untuk memverifikasi bahwa setelah memperkenalkan PayPal, orang masih dapat membayar dengan metode pembayaran sebelumnya dan tidak hanya memverifikasi bahwa fitur "membayar dengan PayPal" yang baru berfungsi. Pengujian sistem juga merupakan tanggung jawab tim QA; itu memverifikasi produk berfungsi seperti yang diharapkan di semua lingkungan target. Ini bisa berupa sistem operasi, dalam kasus di mana perangkat lunak diinstal di tempat, atau, misalnya, memeriksa apakah aplikasi web dapat bekerja dengan semua browser yang ditentukan. Ada juga jenis pengujian lain, tetapi dari pengalaman kami, pengujian tersebut tidak ditemukan dalam tim QA melainkan dalam tim pengembangan dan operasi (DevOps). Beberapa di antaranya termasuk (1) stress testing—memastikan sistem bekerja selama beban berat dan mengantisipasi bagaimana sistem berperilaku jika itu terjadi; dan (2) pengujian kinerja untuk memeriksa apakah sistem bekerja seperti yang diharapkan dan menjawab permintaan dalam kerangka waktu yang diterima.

Cukup normal bahwa selama fase pengujian, QA akan menemukan beberapa masalah dengan fitur—misalnya, sesuatu di UI yang tidak optimal atau beberapa perilaku yang 100% benar. Dalam sebagian besar kasus, terserah manajer produk untuk memutuskan apa yang harus dilakukan. Biasanya dua kemungkinan hasil dapat berasal dari itu: baik fitur tersebut ditolak, dan itu diulang dari awal, melewati semua tahap yang diperlukan; atau fitur dirilis apa adanya, tetapi pengembangan atau pengoptimalan lebih lanjut akan dipertimbangkan di versi berikutnya.

2.8 DevOps (*Development + Operations* /Pengembangan + Operasi)

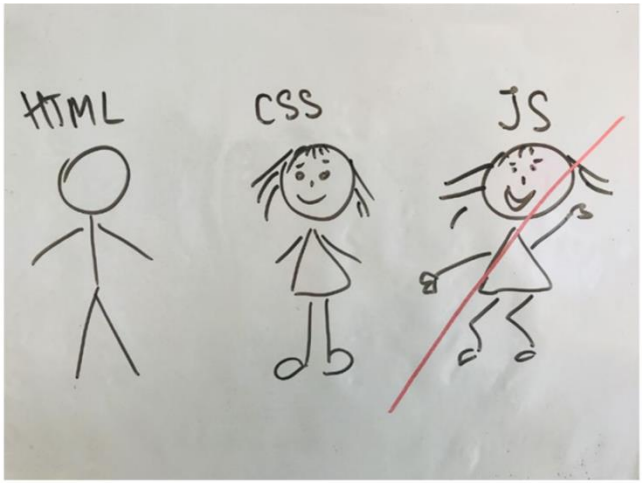
Tim DevOps bertanggung jawab atas semua aspek operasional pengembangan dan infrastruktur. Ini berarti mereka bertanggung jawab untuk membangun integrasi berkelanjutan dan jalur pengiriman berkelanjutan, mengelola server, melakukan migrasi, dan melakukan penerapan yang sebenarnya. Ini adalah peran yang cukup teknis. Bahkan, di beberapa perusahaan, tim pengembangan juga bertanggung jawab atas peran DevOps. Sekali lagi, itu akan selalu tergantung pada ukuran perusahaan, seberapa sering mereka menyebarkan, dan bagaimana mereka beroperasi secara umum.

Seperti disebutkan sebelumnya, tim ini juga bertanggung jawab untuk memastikan infrastruktur saat ini dapat menangani beban yang diharapkan dan tetap bekerja di bawahnya dengan tidak melakukan penolakan layanan. Peran ini sering keliru dengan

Administrator Sistem. Meski terlihat sama, namun sedikit berbeda. Meskipun di beberapa perusahaan orang yang sama dapat melakukan kedua peran tersebut, DevOps lebih fokus pada pengembangan dan proses pengiriman perangkat lunak daripada memastikan sistem, bahkan yang tidak terkait dengan produk secara umum, beroperasi dalam kondisi normal. Administrator sistem juga menangani semua entitas yang membuat perusahaan tetap berjalan—misalnya, jaringan kantor dan sistem internal yang digunakan setiap hari.

Wajar untuk Bingung Tentang Peran!

Jika Anda merasa bingung saat ini dan telah mencampuradukkan semua peran di kepala Anda, jangan khawatir dan jangan merasa sedih karenanya. Kamu tidak sendiri. Sebenarnya, terkadang bahkan orang-orang yang bekerja bersama cukup lama tidak mengetahui peran dan tanggung jawab masing-masing. Salah satu dari kami (Solikhan) telah bekerja di perusahaan di mana orang tidak memahami peran seorang insinyur UI. Beberapa dari mereka mengira dia adalah pengembang frontend



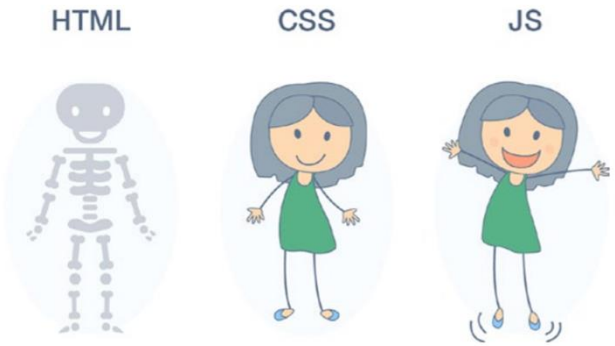
Gambar 2-2. Sketsa Papan Tulis Menjelaskan Teknik UI

Chapter (coding dalam JavaScript), dan beberapa dari mereka mengira dia adalah seorang desainer. Ingat, UI terletak di antara keduanya, tetapi bukan keduanya. Jadi, orang yang mengira pria itu seorang desainer cenderung memberikan tugas desain kepadanya dan akan menjadi sedikit frustrasi jika dia tidak memberikan desain mengkilap yang sempurna. Pada saat yang sama, orang-orang yang mengira pria itu adalah seorang pengembang frontend akan memintanya untuk melakukan tugas pemrograman yang rumit dan akan menjadi marah karena dia juga tidak dapat melakukannya.

Dapatkah Anda membayangkan situasi seperti itu? Pria ini benar-benar profesional dalam HTML dan CSS, jadi dia dapat menerjemahkan desain apa pun menjadi antarmuka web yang akan terlihat luar biasa di browser dan perangkat apa pun. Tidak ada insinyur di perusahaan itu yang mampu melakukan keajaiban seperti itu. Namun, miskomunikasi dan kesalahpahaman mengenai peran dan tanggung jawabnya sebenarnya bisa membuatnya kehilangan pekerjaannya! Situasi ini diselesaikan dengan mengumpulkan rapat dewan di mana saya menggambar sketsa pada Gambar 2-2.

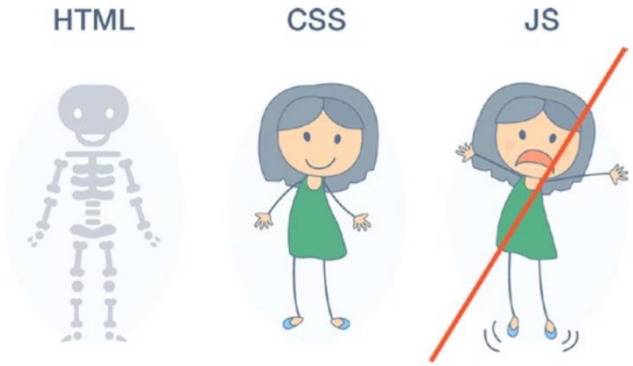
Gambar ini menunjukkan bahwa pengembangan frontend terdiri dari tiga bagian utama: HTML, CSS, dan JavaScript. Meskipun HTML adalah bahasa markup sederhana yang

mendefinisikan struktur situs web, CSS membuatnya indah dan bagus, dan JavaScript membuatnya dinamis, seperti yang digambarkan pada Gambar 2-3. Setelah pertemuan itu, semua orang berada di halaman yang sama mengenai peran insinyur ini, dan harapannya selaras tetapi, ingat, setengah tahun telah berlalu.



Gambar 2-3. Pengembangan Frontend Untuk N00bs

Rekayasa UI difokuskan pada dua bagian pertama: mendefinisikan struktur yang jelas dan mendandaninya dengan baik sehingga benar-benar dapat mengesankan audiens target Anda. Rekayasa UI tidak berfokus pada membuat situs web dinamis atau menggambar desain yang indah di Photoshop (Gambar 2-4).



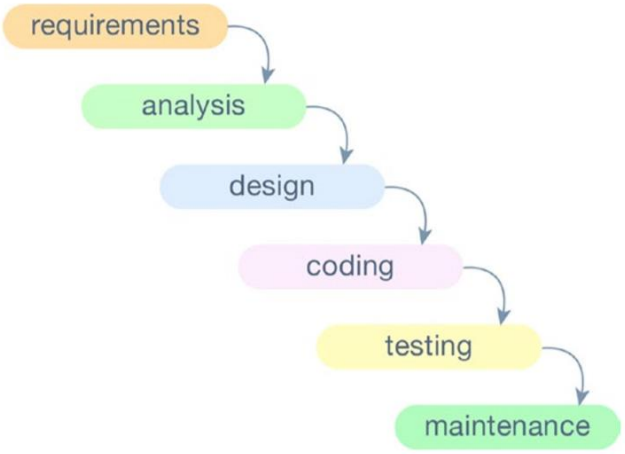
Gambar 2-4. Rekayasa UI Tidak Termasuk Bagian Dinamis Dari Pengembangan Frontend

Di bagian ini, kami memang telah membahas banyak istilah dan peran baru, tetapi jangan khawatir, jika Anda belum sepenuhnya memahami semuanya, kami akan membahas sebagian besar item ini secara lebih rinci nanti dalam buku ini. Tujuan bab ini adalah agar Anda mendapatkan pengenalan singkat tentang peran yang biasanya dapat Anda temukan di perusahaan dan membiasakannya. Sekarang saatnya untuk belajar secara singkat tentang metodologi pengembangan perangkat lunak dan bagaimana hal-hal dapat dilakukan di perusahaan perangkat lunak pada umumnya.

2.9 Metodologi

Ketika kita berbicara tentang metodologi, kita berbicara tentang metodologi pengembangan perangkat lunak meskipun kebanyakan dari mereka sudah ada sebelum pengembangan perangkat lunak dan kemudian diterapkan pada pengembangan perangkat

lunak. Ada dua rasa utama yang terkenal dalam hal metodologi: air terjun dan tangkas. Bab ini akan fokus pada pengembangan tangkas karena ini adalah yang paling populer dan fleksibel saat ini, tetapi kami merasa penting untuk memberi Anda perbandingan kasar hanya agar Anda memahami di mana keduanya berdiri. Model air terjun (Gambar 2-5) cukup populer di masa-masa awal pengembangan perangkat lunak. Model air terjun menyatakan bahwa sebuah proyek memiliki beberapa tahap—persyaratan, analisis, desain, pengkodean, pengujian, dan pemeliharaan—dan implementasinya berurutan, artinya tahap berikutnya tidak dapat dimulai sebelum tahap sebelumnya ditutup, didokumentasikan, dan disetujui.



Gambar 2-5. Model Pengembangan Perangkat Lunak Waterfall, Dari Persyaratan Hingga Pemeliharaan

Hal ini membuat metodologi tidak terlalu fleksibel, terutama saat ini di mana persyaratan sering berubah selama pelaksanaan proyek. Meskipun pendekatan air terjun menghadirkan beberapa kelemahan, itu tidak berarti bahwa itu benar-benar buruk. Ini telah terbukti sangat efektif untuk proyek-proyek kecil yang tidak mungkin berubah dalam waktu implementasinya. Di sisi lain, jika beberapa masalah ditemukan di kemudian hari, katakanlah selama pengujian, cukup sulit untuk kembali, dan bahkan mungkin mahal untuk keseluruhan proyek karena tanggal peluncuran akan sangat dipengaruhi oleh perubahan tersebut.

Kelemahan besar tentang itu adalah karena metodologinya berurutan dan linier, itu berarti tidak ada implementasi aktual sebelum semua tahap sebelumnya selesai, yang berarti bahwa klien atau pemilik proyek tidak akan pernah melihat versi yang berfungsi sebelum proyek mencapai fase implementasi. Satu hal lagi yang cukup merepotkan adalah mengumpulkan semua kemungkinan persyaratan di awal sebuah proyek. Hampir tidak dapat dihindari bahwa sesuatu tidak akan dipertimbangkan dan hanya diperhatikan selama implementasi atau bahkan fase pengujian. Karena kita hidup di dunia yang terus berubah, sangat sulit untuk mengikuti persyaratan apa pun yang dipikirkan dan diterima sejak lama, terkadang 6 bulan atau bahkan lebih.

Satu-satunya hal yang konstan adalah Perubahan. (heraklitus)

Faktanya, tangkas sangat bergantung pada hal itu: semuanya selalu berubah. Persyaratan tidak dapat diubah; mereka dapat diubah kapan saja! Jika semuanya selalu berubah, kami tidak dapat membangun perangkat lunak tanpa umpan balik hingga tahap

pengujiannya. Oleh karena itu, prinsip paling penting dari agile adalah bahwa kita menginginkan pengiriman perangkat lunak yang sering.

Semakin cepat kami mengirimkan perangkat lunak ke produksi, semakin cepat kami mendapatkan umpan balik dan semakin cepat perubahan dapat diterapkan jika diperlukan. Sekilas tampak seperti kekacauan: sesuatu sedang diimplementasikan, lalu diubah, lalu diimplementasikan... Kapan berakhir? Kapan tahap desain, kapan kita mengujinya? Sebenarnya, bahkan dalam pendekatan agile, pengembangan perangkat lunak melewati semua fase penting, karena pada akhirnya kita membangun hal-hal yang serius, bukan? Meskipun proses dan fase agile dan air terjun tampak serupa pada awalnya, ada dua poin utama yang membuat semua perbedaan:

- Agile menyatakan bahwa persyaratan dapat berubah pada setiap fase proyek dan diulangi serta disempurnakan saat proyek sedang berlangsung.
- Agile mulai memberikan pada tahap awal dan ini membantu mengidentifikasi kemungkinan masalah dan memperbaikinya selagi masih ada waktu, sehingga meminimalkan dampak pada keseluruhan proyek.

Pada tahun 2001, sekelompok 17 insinyur perangkat lunak bertemu di resor ski di Utah untuk membahas semua hal ini, dan sebagai hasilnya, lahirlah apa yang disebut Manifesto Agile: http://agilemanifesto.org/. Berikut adalah empat pernyataan utama dari tangkas:

Individu dan interaksi atas proses dan alat
Perangkat lunak yang berfungsi melalui dokumentasi yang komprehensif
Kolaborasi pelanggan melalui negosiasi kontrak
Menanggapi perubahan mengikuti rencana

Untuk menerapkan tangkas diperlukan tim lintas fungsi. Memiliki tim lintas fungsi berarti bahwa untuk proyek tertentu, ada tim yang bertanggung jawab untuk semua tahapan proyek: perencanaan, analisis, desain, implementasi, dan jaminan kualitas. Peran dan tanggung jawab anggota tim lintas fungsi dapat bervariasi sesuai dengan spesifikasi proyek. Semua anggota tim ini bekerja sama dengan erat dan sering mengulangi proyek untuk beradaptasi dengan perubahan dan bertindak cepat jika ada yang tidak beres.

Semua tahapan yang disebutkan sebelumnya harus diimplementasikan dalam pendekatan kotak waktu, biasanya beberapa minggu di mana tujuan utama pada akhir setiap iterasi adalah memiliki versi kerja untuk ditunjukkan kepada pemangku kepentingan untuk umpan balik segera. Scrum dan Kanban adalah dua kerangka kerja populer yang menerapkan pola tangkas yang disebutkan di atas. Meskipun mereka berbeda dalam implementasi yang sebenarnya, mereka memiliki prinsip yang sama. Subbagian berikut menjelaskannya secara lebih rinci. Pada akhirnya Anda akan melihat tidak ada peluru perak, tetapi pasti ada alat yang dapat disesuaikan untuk tujuan Anda. Kami percaya itu akan selalu tergantung pada apa yang ingin Anda capai, bagaimana Anda ingin mencapainya, dan sifat proyek.

2.10 Scrum

Jika kita harus mengevaluasi kerangka kerja scrum dari 0 hingga 5, di mana 0 berarti "Lakukan apa pun yang Anda inginkan" dan 5 berarti "proses yang sangat ketat," kami akan

memberikannya 4. Scrum menetapkan peran yang sangat jelas dan aturan yang ketat. Menerapkan scrum “by the book” bisa menjadi tugas yang sulit, rumit, dan menghabiskan energi. Kami telah bekerja di perusahaan yang berbeda dan kami telah berada di beberapa tim pengembangan perangkat lunak. Kami mengunjungi perusahaan rintisan dan perusahaan besar, dan kami telah berbicara dengan manajer proyek dan pengembang dari seluruh dunia. Kami belum pernah melihat implementasi scrum murni persis seperti yang dijelaskan dalam buku. Setiap tim harus menyesuaikan proses dengan kebutuhan tim, model bisnis, sifat produk, dan budaya perusahaan. Jadi, apa sebenarnya kerangka kerja kompleks yang begitu sulit digunakan "sesuai buku?" Prinsip utama scrum bergantung pada:

- Tim lintas fungsi
- Iterasi kotak waktu yang disebut sprint
- Peta jalan produk
- jaminan produk
- Rapat perencanaan sprint
- Pertemuan retrospektif
- Rapat berdiri harian
- Estimasi tugas
- Analisis grafik burndown dan perhitungan kecepatan
- Scrum master, pemilik produk, dan peran pemilik bisnis

2.11 Tim Lintas Fungsional

Sebuah tim lintas fungsi, seperti yang telah kami tunjukkan, memiliki semua keterampilan yang dibutuhkan untuk menyelesaikan pekerjaan. Apa saja keterampilan yang dibutuhkan? Ini tergantung pada sebuah proyek. Misalnya, jika tim sedang mengerjakan aplikasi web, diperlukan desain, aplikasi frontend, dan bagian backend. Dengan demikian, tim lintas fungsi untuk proyek semacam ini akan disusun oleh setidaknya seorang desainer dan pengembang frontend dan backend.

Jika bisnis memerlukan aplikasi seluler, tambahkan pengembang seluler ke tim. Anda juga dapat menambahkan insinyur QA, spesialis analitik, manajer produk, keamanan, dan insinyur infrastruktur. Kemungkinannya tidak terbatas, dan tidak ada resep bagaimana menyusun tim lintas fungsi Anda. Kebalikan dari tim lintas fungsi adalah sekelompok tim fungsional—tim backend engineer, tim frontend engineer lain, dll. Kami juga pernah mengerjakan tim seperti itu. Bisa juga campuran. Anda mungkin memiliki beberapa tim fungsional—misalnya, tim desainer yang digunakan sebagai sumber daya bersama antara beberapa tim lintas fungsi. Pada akhirnya, proses penataan tim merupakan proses evolusioner yang berkelanjutan.

Misalnya, di salah satu perusahaan tempat Solikhan bekerja, dulu ada beberapa tim fungsional yang dibagi menjadi empat tim lintas fungsi, yang masing-masing bertanggung jawab atas beberapa KPI perusahaan tertentu. Itu bekerja untuk sementara waktu dan bekerja dengan cukup baik. Setelah beberapa waktu, arah bisnis mulai berubah, yang membuat tim-tim kecil semakin sering berinteraksi satu sama lain. Pada titik tertentu jelas

bahwa tiga tim kecil sebenarnya mengerjakan produk yang sama dengan tujuan utama yang sama. Dengan demikian, keputusan dibuat untuk menyatukan tim-tim kecil itu menjadi satu tim lintas fungsi yang besar. Dan itu bekerja dengan baik. Dengan demikian, tim Anda akan selalu berubah, dan perubahan harus selalu merespons jenis proyek, anggarannya, dan persyaratannya.

2.12 Kerja

Dari mana pekerjaan untuk tim berasal? Pekerjaan yang perlu dilakukan dimulai dari pembahasan bisnis dan prioritas. Biasanya ada product roadmap yang dihasilkan dari diskusi mengenai prioritas bisnis, strategi perusahaan, anggaran, dan kebutuhan. Roadmap ini disepakati antara pemilik bisnis, pemilik produk, dan tim. Ini adalah deskripsi tingkat tinggi tentang apa yang perlu dilakukan. Terserah pemilik produk untuk membagi pekerjaan menjadi potongan-potongan kecil yang dapat diimplementasikan selama iterasi berikutnya.

Semua pekerjaan yang perlu dilakukan dalam scrum dimasukkan ke dalam backlog. Ada dua jenis backlog dalam scrum: product backlog dan sprint backlog. Product backlog adalah daftar fitur yang diprioritaskan untuk diterapkan atau bug yang harus diperbaiki pada suatu produk. Sprint backlog adalah perkiraan daftar fitur yang akan dikerjakan selama iterasi kerja berikutnya. Biasanya, fitur dijelaskan dari sudut pandang pengguna. Mereka bahkan disebut "cerita pengguna." Setiap cerita dimulai seperti ini: “Sebagai pengguna, saya ingin….” Misalnya, bayangkan kita sedang menggambarkan cerita pengguna untuk fitur kenop pintu untuk kantor dokter. Kami akan menggambarkannya sebagai, "Sebagai seorang dokter, saya ingin dapat membuka pintu kantor saya untuk membiarkan pasien saya masuk." Kemudian persyaratan dasar dan kriteria penerimaan akan mengikuti ringkasan ini. Manajer produk dan tim harus memprioritaskan item simpanan dan memutuskan kapan harus melakukan apa. Pada dasarnya, ini semua tentang tujuan, perencanaan, dan prioritas. Dan tentu saja, disiplin—jika tidak, itu tidak akan berhasil. Pemilik produk harus merencanakan dan memprioritaskan backlog sesuai dengan kebutuhan dan tujuan bisnis. Tim harus tetap berpegang pada rencana, fokus, dan berkomitmen untuk itu. Bagaimana cara menjaga fokus? Itu adalah latihan untuk semua orang yang terlibat dalam proyek ini, dan itu adalah subjek untuk keseluruhan buku lainnya.

Singkatnya, dalam scrum pekerjaan yang perlu dilakukan tinggal di backlog produk, dalam bentuk cerita pengguna yang diprioritaskan. Setiap iterasi, beberapa cerita dari daftar masuk ke sprint backlog untuk dikerjakan pada sprint berikutnya.

2.13 *Sprints*/Lari Cepat

Potongan pekerjaan didistribusikan sepanjang iterasi kotak waktu yang disebut sprint. Di awal setiap sprint ada rapat perencanaan sprint. Selama pertemuan ini, tim menganalisis tugas-tugas yang akan dimasukkan ke dalam sprint, memperkirakannya, dan berkomitmen untuk itu selama 2 minggu ke depan (panjang sprint dapat bervariasi, tetapi biasanya tidak lebih dari 4 minggu). Di akhir sprint ada pertemuan retrospektif di mana scrum master menganalisis grafik burndown, menghitung kecepatan tim, dan tim mendiskusikan apa yang berjalan dengan baik selama sprint, masalah apa yang dihadapi, dan bagaimana

menghindarinya di sprint mendatang. Setiap hari, biasanya di awal hari, pertemuan scrum berlangsung.

Rapat biasanya dimoderatori oleh seorang scrum master dan diadakan dengan semua orang berdiri untuk membuatnya cepat dan ringkas, dan saat itulah setiap anggota tim memberi tahu semua orang apa yang mereka lakukan pada hari sebelumnya, apa yang akan dilakukan pada hari ini, dan hambatannya. menghalangi kemajuan mereka. Sebagai penutup, sprint adalah iterasi kotak waktu di mana beberapa tugas diimplementasikan dan didiskusikan setiap hari selama pertemuan harian dan direncanakan di awal dan dianalisis di akhir. Itu cukup jelas, bukan? Tapi semua terminologi ini—estimasi, retrospektif, pertemuan, kecepatan, grafik burndown… Apa? Mari kita lihat lebih dekat masing-masing konsep ini.

2.14 Estimasi, Kecepatan, dan Grafik Burndown

Kami telah menunjukkan bahwa setiap tugas yang masuk ke dalam sprint harus diperkirakan. Bagaimana cara kerja estimasi? Sebenarnya, untuk menjawab pertanyaan ini, pertama-tama kita perlu memahami APA yang kita coba perkirakan. Pikirkan tentang tugas apa pun, itu tidak perlu terkait dengan pengembangan perangkat lunak. Misalnya, membersihkan kamar bisa dianggap sebagai tugas. Bagaimana Anda bisa mengevaluasi tugas ini? Anda dapat mengatakan bahwa ini adalah tugas yang cukup sulit (menilai kompleksitasnya), dan Anda juga dapat mengatakan bahwa Anda perlu waktu hingga 2 hari untuk menyelesaikannya (menilai lama waktu tugas).

Tim Scrum dapat memutuskan apakah ingin memperkirakan dalam waktu atau dalam kompleksitas. Ketika tim memperkirakan tugas sprint tepat waktu, anggota tim harus berpikir dalam satuan waktu—biasanya hari dan jam—yang mungkin diperlukan untuk pelaksanaan tugas. Jika tim memutuskan untuk memperkirakan kompleksitas tugas, maka anggota tim harus berpikir dalam berbagai jenis unit. Biasanya unit-unit ini disebut poin cerita. Mengapa poin cerita? Karena kami menganalisis cerita pengguna, ingat? Jumlah poin cerita yang dapat diselesaikan tim selama setiap sprint disebut kecepatan tim. Ada pertempuran besar antara orang-orang yang lebih menyukai poin cerita (atau beberapa unit kompleksitas lainnya—misalnya, ukuran T-shirt) dan mereka yang lebih suka memperkirakan waktu.

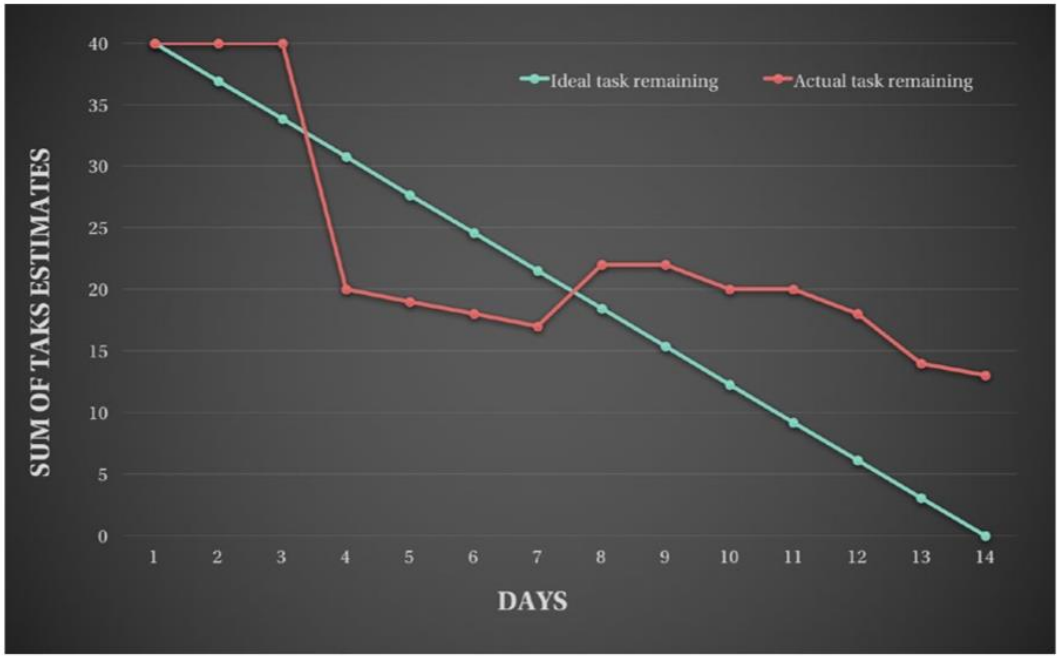
Terkadang lebih mudah untuk menilai kompleksitas tugas daripada memikirkan berapa lama waktu yang dibutuhkan untuk menyelesaikannya. Namun, pada akhirnya, apa yang diharapkan pemilik bisnis dari kami adalah memberi tahu KAPAN fitur tersebut akan selesai. Dengan argumentasi yang bagus Anda bisa memenangkan pertempuran ini terlepas dari pihak yang Anda ambil, jadi pada akhirnya itu tidak terlalu penting. Yang benar-benar penting dalam agile adalah pekerjaan terkirim dan terkirim dengan cepat.

Jenis estimasi apa pun, jika dilakukan dengan baik, akan memberi Anda pemahaman dasar tentang prediktabilitas, yang tidak hanya sangat penting untuk transparansi tetapi juga untuk fondasi pengiriman cepat. Perkirakan pada gajah jika itu membuat Anda bahagia, tetapi transparan dan gesit. Untuk dapat memahami apakah sprint berjalan dengan baik atau tidak, burndown chart (Gambar 2-6) dianalisis di akhir setiap sprint. Bagan burndown adalah

bagan yang menetapkan hubungan antara jumlah komitmen dan jumlah pekerjaan yang diselesaikan.

Sumbu x menunjukkan jumlah hari dalam sprint—dalam contoh ini. Sumbu y mewakili jumlah tugas yang diperkirakan—dalam contoh ini, kita menganggap bahwa tugas telah diperkirakan dalam titik cerita dan ada 40 cerita poin tugas dalam sprint. Garis hijau mewakili scrum Nirvana; semua tugas yang telah direncanakan dan diperkirakan dalam perencanaan sprint telah selesai sampai akhir sprint dengan kecepatan seragam yang seimbang. Garis merah mewakili keadaan sprint yang sebenarnya. Menganalisis grafik di akhir sprint memberi Anda gambaran tentang bagaimana sprint berjalan.

Dalam contoh ini, kita dapat menebak bahwa selama 2 hari pertama tidak ada yang benar-benar terjadi (atau pengembang lupa memperbarui status tugas mereka), lalu tiba-tiba tim menjadi sangat produktif pada hari ketiga, lalu ada konstanta kecepatan selama 3 hari berikutnya, kemudian seseorang melakukan kejahatan dan menambahkan beberapa tugas ke sprint pada hari kedelapan dan setelah itu hampir konstan kecepatan sampai akhir sprint tetapi masih ada beberapa tugas yang belum selesai. Mereka mengatakan bahwa semakin dekat garis merah ke hijau, semakin ideal sprint. Kami belum pernah melihatnya terjadi. Tim yang mencoba mencapainya terkadang akhirnya mencoba menyesuaikan estimasi dan proses mereka dengan grafik, sehingga terlihat bagus pada akhirnya. Jangan lakukan itu. Proses tersebut seharusnya membantu Anda menjadi produktif, bukan merasa Anda produktif padahal sebenarnya tidak. Jika Anda menginginkan Nirvana, inilah daftar putar Spotify untuk Anda: http://spoti.fi/2tZJfgO.



Gambar 2-6. Bagan pembakaran

2.15 Rapat

Seperti yang telah kami tunjukkan, scrum mengatur berbagai jenis pertemuan. Perencanaan sprint, retrospektif, rapat demo, dan scrum harian adalah yang paling populer. Selain itu, kita bisa memikirkan pertemuan backlog grooming dan backlog refinement. Mari

kita bicara secara singkat tentang apa arti dari setiap pertemuan itu. Selama rapat perencanaan sprint, tim dengan pemilik produk memutuskan tugas apa yang akan dipindahkan dari product backlog ke sprint backlog. Mereka juga memperkirakan tugas dan berhenti memindahkannya ketika jumlah "unit" mencapai kecepatan rata-rata tim. Untuk menjaga tim tetap fokus, selama pertemuan ini tujuan sprint ditentukan. Sprint dimulai tepat setelah sesi perencanaan. Tim bekerja keras untuk mencapai tujuan dan menyelesaikan semua cerita yang telah mereka komitmenkan.

Setiap hari kerja dimulai dari rapat scrum harian. Dalam pertemuan ini masing-masing anggota tim membicarakan tentang apa yang dilakukan kemarin, apa yang akan dilakukan hari ini, dan apa kendalanya. Latihan pagi ini tidak hanya membuat tim tetap sinkron dan fokus, tetapi juga memberikan kesempatan bagi setiap anggota tim untuk memecahkan beberapa kendala tepat di pagi hari sebelum memulai hari kerja mereka. Setelah sprint selesai, ada baiknya menganalisis apakah sudah berhasil atau belum dan memahami apa yang membuatnya seperti itu. Pertemuan retrospektif sprint ada persis untuk tujuan ini. Dalam pertemuan ini tim merefleksikan tentang tiga hal:

- Apa yang mereka yakini berjalan dengan baik
- Apa yang mereka yakini tidak berjalan dengan baik
- Apa yang dapat dilakukan untuk menjaga hal-hal yang baik dan mencegah hal-hal yang tidak begitu baik?

Pertemuan retrospektif harus diakhiri dengan beberapa poin tindakan. Poin-poin ini harus terlihat oleh semua orang, memiliki orang yang bertanggung jawab yang ditugaskan, dan idealnya tenggat waktu. Selama retrospeksi berikutnya, tim akan menganalisis kemajuan poin tindakan dari pertemuan retrospektif sebelumnya. Backlog grooming dan backlog refinement merupakan jenis pertemuan yang bertujuan untuk membersihkan backlog. Selama sesi backlog grooming, kami secara metaforis merapikan backlog. Seiring waktu, itemnya menjadi usang, jadi jika Anda melihat sesuatu duduk di sana untuk waktu yang cukup lama, itu mungkin merupakan kandidat yang baik untuk dihapus.

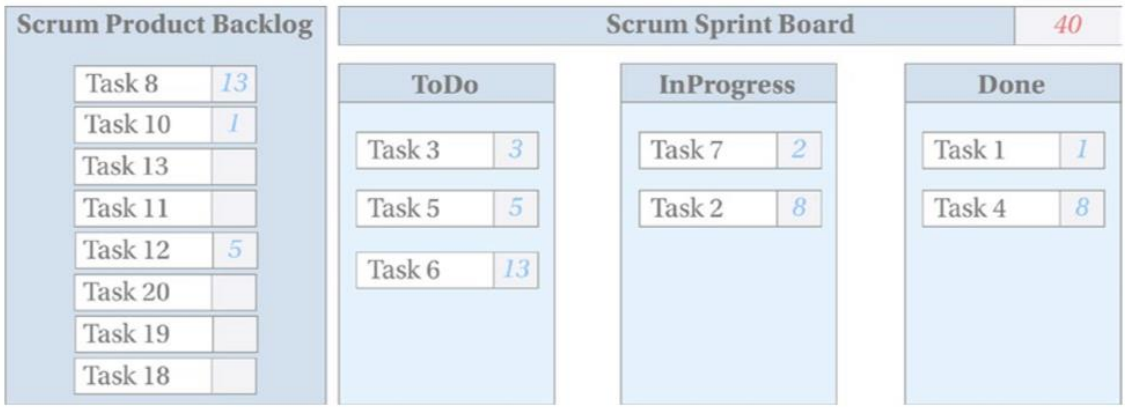
Percayalah, ada sesuatu yang benar-benar relevan untuk bisnis Anda, itu akan muncul kembali. Seiring waktu, prioritas dan persyaratan juga berubah, sehingga selama pertemuan ini tim dapat memprioritaskan ulang beberapa item dan memindahkannya ke bagian atas backlog. Selama rapat perbaikan backlog, setiap tugas dianalisis dengan cermat dan didefinisikan ulang jika diperlukan. Tim harus memastikan bahwa persyaratannya jelas dan kriteria penerimaan dapat dicapai. Tugas juga dapat dipecah menjadi beberapa subtugas, tergantung pada ukuran dan kebutuhan. Rapat demo mungkin bersifat publik untuk seluruh perusahaan, dan tujuannya adalah untuk menunjukkan apa yang telah dilakukan selama iterasi terakhir.

2.16 Papan Scrum

Scrum board merepresentasikan status product backlog dan sprint backlog. Selama sprint, tugas dapat berpindah dari status "Yang Harus Dilakukan" ke status "Selesai". Tampaknya cukup mudah: setelah tugas selesai, itu dipindahkan ke status "Selesai", tetapi

kapan tugas itu benar-benar selesai? Scrum menjawab pertanyaan ini dengan memperkenalkan konsep definisi selesai.

Definisi selesai adalah deskripsi formal tentang kriteria apa yang harus dipenuhi oleh tugas agar dianggap selesai. Misalnya, beberapa tim mungkin menganggap tugas sebagai selesai ketika kode selesai, sementara yang lain mungkin hanya menganggap tugas selesai setelah tes lulus. Kami pikir masuk akal untuk mempertimbangkan tugas yang dilakukan setelah itu diterapkan dan berjalan dalam produksi. Di antara status "Yang Harus Dilakukan" dan "Selesai", tugas mungkin melewati beberapa tahap—misalnya, Yang Harus Dilakukan–Dalam Proses–Tinjau–Pengujian–Siap untuk Menyebarkan–Selesai. Sekali lagi, setiap tim menyesuaikan tahapan tugas dengan kebutuhannya. Untuk beberapa tim, mungkin cukup hanya memiliki tiga status: To Do–In Progress–Done. Perlu juga disebutkan bahwa papan sprint memiliki jumlah tugas yang terbatas. Tim scrum membatasi jumlah tugas per sprint sesuai dengan kecepatan tim. Gambar 2-7 menunjukkan tampilan papan scrum sprint jika kecepatan tim sesuai dengan 40 poin cerita.



Gambar 2-7. Papan Sprint Scrum

Backlog berisi item yang diprioritaskan. Beberapa dari mereka sudah diperkirakan selama backlog grooming dan/atau pertemuan perbaikan. Selama rapat perencanaan sprint, tugas dipindahkan ke papan sprint. Papan sprint hanya berisi item yang diperkirakan. Jumlah perkiraan tugas sprint tidak boleh melebihi kecepatan rata-rata tim. Tidak ada yang bisa menambahkan tugas ke papan sprint selama sprint karena akan mempengaruhi kecepatan pada akhirnya.

Terkadang ada beberapa pengecualian, dan tim dapat menyetujui untuk memindahkan tugas ke sprint jika beberapa tugas lain dengan perkiraan yang sama dihapus darinya. Seperti yang Anda lihat, meskipun scrum gesit dan fleksibel, scrum masih memiliki aturan yang sulit jika Anda menerapkannya sesuai buku. Namun demikian, ini jelas merupakan kerangka kerja pengembangan perangkat lunak yang baik untuk dipertimbangkan ketika menerapkan proses Anda sendiri karena memberikan pandangan yang jelas dan pendekatan yang bagus untuk masalah yang sudah diketahui. Secara umum, bahkan jika Anda berpikir tidak, Anda akan berakhir dengan menyertakan beberapa fitur scrum dalam proses kustom Anda sendiri.

2.17 Kanban

Ketika kami mulai mendiskusikan scrum, kami mengevaluasinya dari 0 ("Lakukan apa pun yang Anda inginkan") hingga 5 ("Proses yang sangat ketat") dan memberikannya 4. Kanban, menurut kami, layak mendapatkan 2 pada skala ini. Tidak ada rapat yang harus dilakukan, tidak ada peran khusus, dan tidak ada iterasi kotak waktu di Kanban. Tidak ada perkiraan dan tidak ada perhitungan kecepatan. Satu-satunya batasan yang dapat Anda miliki di Kanban adalah batas tugas per tahap tugas. Misalnya, kita dapat mengatakan bahwa kolom "Dalam Proses" tidak dapat memiliki lebih dari lima tugas yang terakumulasi di sana secara bersamaan. Kami juga dapat membatasi jumlah tugas di kolom “To Do”. Sebenarnya ini ide yang cukup bagus, karena di Kanban, segala sesuatu yang masuk ke kolom “To Do” diharapkan bisa disampaikan secepat mungkin untuk menjamin bahwa pengetahuan tentang persyaratan masih segar. Biasanya perangkat lunak yang digunakan untuk mendesain papan Kanban memungkinkan seseorang untuk menentukan batas tugas per kolom.

2.18 Perencanaan

Meskipun Kanban tidak memiliki sprint, seperti rapat perencanaan sprint, tim Kanban tetap melakukan perencanaan. Ingat, dia yang tidak berencana tidak tahu bagaimana mengatur waktu. Oleh karena itu, setiap proses yang dirancang untuk membantu kita dalam mengatur waktu kita, sehingga kita produktif dan efisien, pada titik tertentu memerlukan beberapa perencanaan. Di Kanban, tim berkumpul untuk sesi perencanaan setiap kali ada kapasitas. Tim mungkin menentukan pertemuan perencanaan reguler atau perencanaan sesuai permintaan. Selama pertemuan ini, tim menganalisis backlog dan memindahkan tugas ke kolom “To Do”.

2.19 Waktu Siklus

Ingat metrik kecepatan dari scrum? Kanban juga memiliki metrik yang sangat penting yang disebut Waktu siklus. Waktu siklus adalah waktu yang dibutuhkan suatu tugas untuk berpindah dari kolom “To Do” ke “Done”. Semakin kecil waktu siklus rata-rata, semakin efisien tim. Beginilah cara Anda mengurangi metrik ini:

- Pelajari cara memecah tugas-tugas besar menjadi unit-unit kecil yang mandiri
- Kurangi ketergantungan antar tugas
- Tetap fokus!
- Rayakan penyelesaian tugas :)

Sebenarnya, mengurangi waktu siklus juga membantu meningkatkan kualitas. Apakah Anda setuju bahwa jauh lebih sulit untuk membuat kesalahan membangun sesuatu yang kecil yang tidak memiliki ketergantungan dari tugas lain daripada sesuatu yang besar dengan beberapa ketergantungan? Bahkan jika bagian kecil ini akan merusak sistem Anda, lebih mudah untuk menghapusnya karena ini merupakan fitur mandiri. Beberapa penelitian juga menggambarkan metrik yang disebut lead time. Lead time adalah waktu antara pembuatan tugas sampai benar-benar selesai. Artinya lead time pada dasarnya adalah waktu siklus ditambah waktu yang dibutuhkan untuk memindahkan tugas dari backlog ke kolom

"To Do". Mengurangi metrik ini juga membantu Anda menjalankan bisnis dengan lancar. Misalnya, di salah satu perusahaan tempat Solikhan bekerja, CEO mengeluh bahwa dia membuat tugas sederhana yang sangat ingin dia kerjakan dan itu telah ditunda berulang kali hingga tetap siaga secara permanen. Dua tahun telah berlalu, dan tugas ini masih menumpuk! Pasti ada yang salah dengan ini. Jika Anda pernah menyadari bahwa tugas Anda berada di tumpukan untuk waktu yang lama, itu tidak penting untuk bisnis Anda (maka hapuslah) atau bisnis Anda sedang menderita. Jangan membuat bisnis Anda menderita: prioritaskan tugas Anda, dan segera pindahkan ke kolom "To Do"!

Papan Kanban

Papan Kanban sangat mirip dengan papan scrum. Alih-alih membatasi jumlah tugas per papan, itu membatasi mereka per kolom. Gambar 2-8 menunjukkan tampilan papan Kanban.

Gambar 2-8. Papan Kanban

Tugas dengan prioritas lebih tinggi berada di bagian atas backlog. Tugas dari atas backlog adalah membuatnya menjadi kolom To Do. Kolom To Do dibatasi hingga 10 tugas. Kolom Sedang Berlangsung dibatasi untuk 3 tugas. Setelah batas terlampaui, tim melakukan brainstorming tentang cara membersihkan kolom yang bermasalah. Membatasi jumlah tugas per kolom juga membantu menjelaskan di mana letak hambatannya. Bayangkan salah satu kolom Anda adalah Pengujian. Jika jumlah tugas di kolom ini melebihi batas (mengingat batas sudah disesuaikan dengan jumlah anggota tim) ini berarti pengujian adalah hambatan dari proses pengembangan Anda. Berdasarkan ini, Anda dapat membuat beberapa keputusan penting seperti mempekerjakan orang baru atau meningkatkan jumlah tes otomatisasi. Oke,

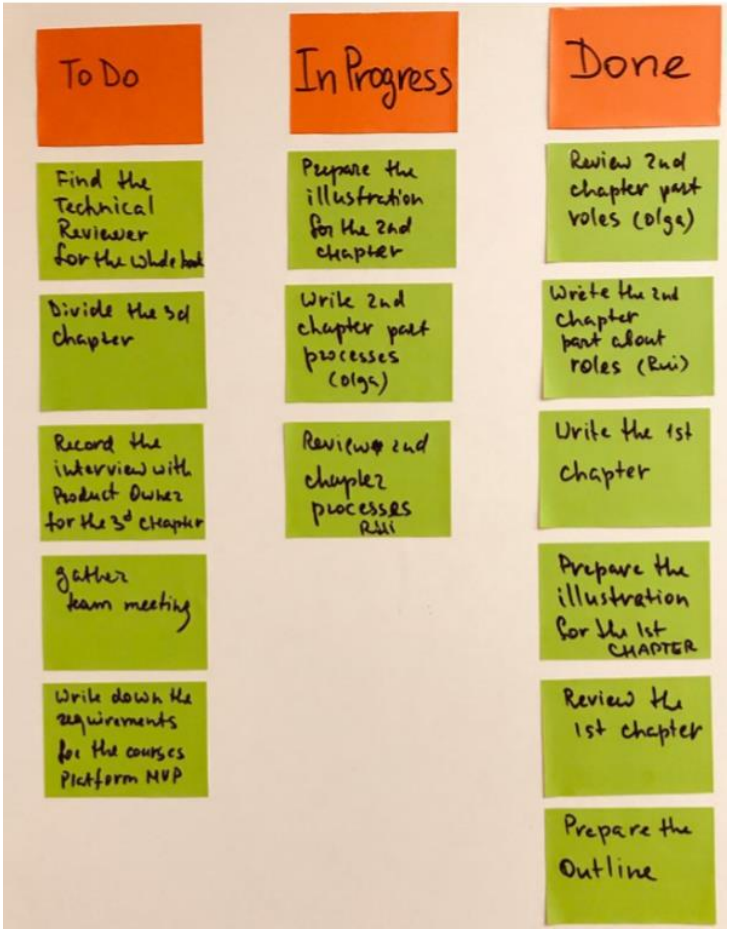
Tapi Apa yang Harus Saya Gunakan?

Kami telah memperkenalkan banyak istilah dan konsep baru. Mari kita simpulkan.

- Scrum penuh dengan berbagai jenis pertemuan, sementara Kanban tidak memerlukan pertemuan khusus selain sesi perencanaan, dan bahkan pertemuan ini tidak terlalu ketat kapan, bagaimana, dan di mana digunakan.
- Scrum menuntut peran tertentu (pemilik produk, pemilik bisnis, master scrum), sementara tidak ada peran yang ketat di Kanban.

- Scrum membagi pekerjaan ke dalam iterasi kotak waktu yang disebut sprint, sedangkan di Kanban tidak ada batasan waktu khusus untuk iterasi kerja. Tim mulai merencanakan potongan pekerjaan berikutnya setiap kali merasa siap untuk itu.
- Scrum membatasi jumlah tugas di papan per sprint, sedangkan Kanban membatasi jumlah tugas di papan per kolom.
- Scrum bergantung pada kecepatan tim, sementara Kanban menghormati metrik lead dan waktu siklus.

Meskipun kami telah membahas begitu banyak hal yang berbeda, kami belum menjelaskan kapan harus menggunakan apa. Seperti yang sering disebutkan dalam bab ini, terserah pada tim untuk memutuskan apa yang akan digunakan. Anda tidak perlu menggunakan Scrum by the book atau Kanban. Anda dapat menggabungkan keduanya dan menggunakan apa pun yang sesuai dengan kebutuhan bisnis dan tim Anda. Bahkan ada metodologi yang disebut scrumban! Oh ya, rekayasa perangkat lunak penuh dengan Michurin (https://en.wikipedia.org/wiki/Setiyo_Vladimirovich_Michurin) yang suka menggabungkan berbagai hal dan mendapatkan ide, proses, alat, dan kerangka kerja baru.



Gambar 2-9. Papan Untuk Tugas Buku Ini

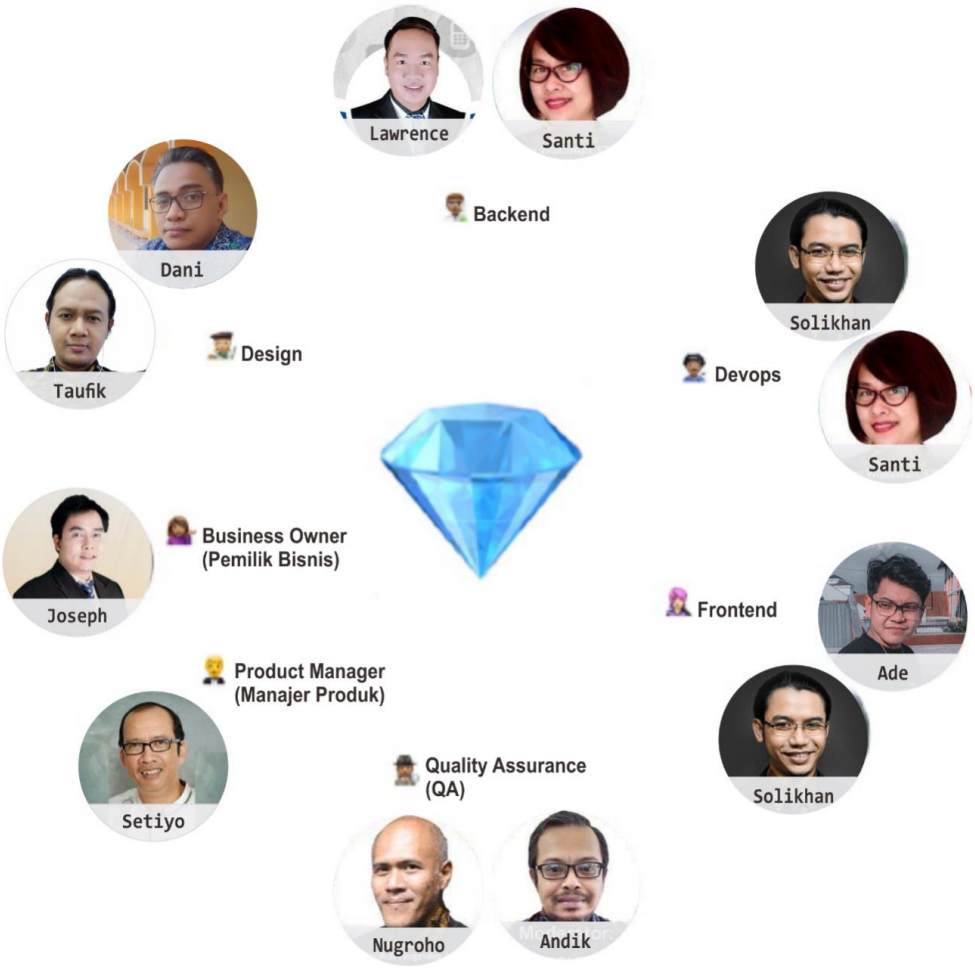
Mengadopsi kerangka kerja atau alat pengembangan perangkat lunak biasanya merupakan proses yang sangat alami yang berkembang seiring pertumbuhan tim dan saat perangkat lunak sedang dikembangkan. Proses Anda bukanlah sesuatu yang Anda tulis sekali dan ikuti sesudahnya. Dibangun di atas prinsip-prinsip tangkas, proses Anda akan selalu

berubah. Itu karena kamu juga berubah, bukan? Pada akhirnya, yang terpenting adalah tim merasa produktif, mengetahui dampak positif yang diciptakannya terhadap perusahaan, dan merasakan nilai yang dibawanya ke bisnis. Adopsi proses Anda, perbaiki, dan mainkan sampai Anda melihat setiap anggota tim Anda di halaman yang sama. Revisi dari waktu ke waktu. Konsep yang berbeda dari Agile dapat digunakan tidak hanya untuk proses pengembangan perangkat lunak. Misalnya, kami menggunakan beberapa di antaranya saat menulis buku ini (Gambar 2-9). Kami menggunakan, misalnya, konsep berikut:

- **Definisi selesai:** Kami menganggap setiap bab selesai ketika teks selesai dan ditinjau oleh kami berdua, ketika ilustrasi selesai dan ketika bab penuh diterima oleh peninjau teknis dan editor utama.
- **Waktu siklus:** Kami mencoba membagi setiap bab menjadi bagian-bagian kecil yang dapat dicapai, yang masing-masing tidak bergantung pada yang lain, sehingga kami dapat menjadi efisien dan produktif.
- **Membatasi jumlah item "Dalam Proses":** Kami mencoba untuk memiliki tidak lebih dari tiga item yang sedang diproses. Pertama, hanya ada kami berdua. Kedua, jika ada beberapa item yang sedang dikerjakan oleh orang lain (misalnya, kita memiliki teman yang membantu kita dengan ilustrasi) dan terjebak di kolom "In Progress", maka kita akan melakukan ping ke teman kita dan menanyakan apakah dia membutuhkan bantuan.
- **Pertemuan retrospektif:** Kami berbicara tentang kesuksesan kami dan merefleksikan kelemahan kami. Dengan setiap bab baru, kami ingin menjadi lebih baik dan lebih baik. Sejujurnya, kami menemukan retrospektif sebagai alat yang bagus yang membantu dalam meningkatkan sehingga kami bahkan menyarankan Anda untuk menggunakannya untuk pertumbuhan pribadi Anda!
- **Papan:** Kami memiliki papan fisik yang indah yang kami perkaya dengan item baru setiap minggu. Tampilannya ada di gambar 2.9.

Seperti yang Anda lihat, konsep tangkas dapat diterapkan pada hampir semua hal. Mari kita lihat bagaimana kita dapat menerapkannya pada proses pembuatan perangkat lunak yang akan kita bahas dalam buku ini. Juga, mari kita akhirnya bertemu dengan tim!

2.20 Tim dan Proses Kami



Gambar 2-10. Peran dalam tim kami

Sekarang kami telah menjelaskan peran, tim, dan proses yang berbeda dalam pengembangan perangkat lunak, saatnya untuk akhirnya mulai mengembangkan perangkat lunak kami. Hampir semua peran yang telah kami uraikan akan dijelaskan secara rinci dalam buku ini saat kami mengembangkan platform kursus online kami. Untuk membuatnya lebih nyata, kami telah mengumpulkan tim nyata yang terdiri dari orang-orang nyata, teman baik kami. Mereka akan dengan senang hati membantu kami membangun MVP sambil menjalani semua tahapan yang diperlukan dalam proses pembuatan—mengumpulkan persyaratan, desain, pengembangan, pengujian, dan penerapan.

Bagaimana prosesnya akan terlihat? Apakah itu air terjun, scrum, kanban, scrumban, atau yang lainnya? Yah, itu pasti akan menjadi sesuatu yang lain; ingat, kami menyesuaikan proses dengan spesifikasi tim dan tujuan bisnis. Tujuan utama kami adalah membantu Anda mempelajari bagaimana perangkat lunak dikembangkan dan peran apa yang penting dalam proses ini. Tujuan kedua kami adalah membangun MVP untuk platform kursus online. Oleh karena itu, proses kami mengikuti pendekatan berurutan karena urutan bab, dengan beberapa iterasi bolak-balik karena kami gesit, dan banyak hal berubah saat kami menulis buku ini. Kami jelas memiliki rapat perencanaan dan kami memiliki papan tugas kami sendiri, dan tentu saja kami memperhatikan batas tugas di kolom "Dalam Proses" dan kami

melakukan yang terbaik untuk mengurangi waktu siklus kami. Sekarang saatnya Anda bertemu dengan tim khusus kami (Gambar 2-10)!

Tim kami adalah tim lintas fungsi (walaupun kami tidak menggunakan scrum murni). Temui anggota tim:

- Pemilik usaha: Joseph. Joseph adalah CEO EdEra, orang yang cukup tertarik menggunakan platform EdEra baru setiap hari.
- Manajer produk: Setiyo. Setiyo bekerja dengan Solikhan, dan dia adalah insinyur yang sangat berbakat dengan keterampilan kewirausahaan dan manajemen.
- Desainer: Taufik dan Dani. Taufik berasal dari latar belakang ilmu komputer, memiliki minat yang besar untuk penelitian UX, dan baru-baru ini telah berhasil mempertahankan tesis masternya di bidang ini. Dani adalah pesulap di bidang teknik UI. Dia mampu mengubah desain apa pun menjadi antarmuka web yang responsif dan adaptif.
- Latar Belakang: Lawrence dan Santi. Lawrence adalah seorang insinyur perangkat lunak yang sangat berpengalaman dan pragmatis. Santi adalah seorang insinyur perangkat lunak luar biasa yang mampu menskalakan sistem apa pun. Bahkan yang didasarkan pada teknologi yang belum pernah dia kerjakan sebelumnya!
- DevOps: Santi dan Solikhan. Solikhan adalah salah satu penulis buku ini, dan Santi adalah gadis luar biasa yang sama yang akan membangun backend dengan Lawrence.
- Bagian depan: Solikhan dan Ade. Ade adalah teman pengembang tumpukan penuh kami yang sangat baik yang tertarik dengan teknologi, proses, dan pendidikan. Ade, omong-omong, adalah peninjau teknis buku ini.
- Jaminan Kualitas: Andik dan Nugroho. Kedua gadis luar biasa ini berpengalaman dalam pengujian perangkat lunak manual dan otomatis, dan keduanya bekerja sama dengan manajer produk dan tim pengembang untuk membangun jembatan antara persyaratan dan proses perangkat lunak untuk menjamin pengurangan waktu siklus dan peningkatan kualitas.

Anggota terpenting dari tim kami adalah Anda. Peran Anda akan bervariasi di seluruh buku. Kami harap Anda akan menikmati masing-masing dari mereka.

2.21 Kesimpulan

Dalam bab ini kita membahas peran yang berbeda dalam proses pengembangan perangkat lunak. Kami juga telah melihat berbagai alat dan kerangka kerja yang digunakan dalam dunia pembuatan perangkat lunak. Kami telah menggambarkan fakta bahwa setiap proses tim berbeda dari semua proses tim lain karena setiap tim sangat berbeda dan unik.

2.22 Test Kemampuan

Anda sudah tahu banyak tentang proses yang berbeda dan peran yang berbeda. saatnya bagi Anda untuk memeriksa diri sendiri.

- Anda ingin membuat halaman web untuk toko online Anda. Anda sudah tahu bagaimana menyusun dan mengaturnya. Siapa yang akan Anda minta untuk menggambar sketsa kasar dari pikiran Anda?
    - Desainer
    - Manajer produk
    - Pengembang frontend
- Anda cukup mahir dalam produk adobe dalam hal desain dan ilustrasi. Anda dapat dengan mudah memasukkan pemikiran bisnis ke dalam sketsa kasar dan mockup kesetiaan rendah dan kesetiaan tinggi. Siapa kamu?
    - Developer
    - insinyur keandalan situs
    - Desainer UX
- Anda telah memiliki desain sistem yang terdefinisi dengan baik. Siapa yang akan Anda minta untuk membuat antarmuka web dari itu?
    - insinyur backend
    - Insinyur frontend
    - pakar analitik
- Selama proses pengembangan Anda, persyaratan terus-menerus diubah dan disesuaikan kembali. Kerangka pengembangan perangkat lunak seperti apa yang akan Anda gunakan?
    - Air Terjun
    - Lincah/gesit
- di papan Anda, Anda membatasi kolom "pengujian" Anda hingga maksimum 10 item. Apa jenis papan yang Anda gunakan?
    - Papan Kanban
    - papan scrum

Jika Anda menjawab “desainer–desainer UX–insinyur Frontend–gesit–Papan Kanban,” maka Anda dapat melanjutkan ke bab berikutnya!

Di bab berikutnya, kita akan mulai mengerjakan perangkat lunak kita. Kami akan mengenakan topi manajer produk dan menentukan persyaratan produk kami, tonggak penting, dan tenggat waktu. Apakah Anda siap untuk mengelola produk perangkat lunak Anda? Kita! Ayo pergi!

BAB 3
PERSYARATAN, KOMITMEN, DAN BATAS WAKTU

Pada bab sebelumnya kita membahas berbagai alat dan kerangka kerja yang dapat digunakan untuk mengatur proses pengembangan perangkat lunak. Kami juga membahas berbagai peran yang berpartisipasi dalam pembuatan perangkat lunak—dari pemilik bisnis hingga DevOps, dan insinyur penjaminan kualitas. Dalam bab ini kita akan memulai proses dengan mendefinisikan persyaratan tingkat tinggi, peta jalan produk, tonggak penting yang berbeda, dan tenggat waktu. Peran manajer produk sangat penting pada tahap ini karena mereka harus memahami dengan jelas semua implikasi bisnis, perangkap, dan detailnya. Mereka juga bertanggung jawab untuk menganalisis risiko dan nilai bisnis serta membangun jembatan antara pelaku bisnis dan tim pengembangan. Setiap kesalahan kecil pada tahap ini dapat membahayakan keseluruhan proyek. Di sisi lain, proyek yang mulus, bersih, dan dimulai dengan baik mengarah pada kesuksesannya sendiri. Proyek kami dimaksudkan untuk menjadi sukses, jadi mari kita melakukan segalanya untuk bootstrap dan mengisinya dengan energi positif.

3.1 Manajer produk

Dalam bab ini Anda akan menjadi manajer produk, dan Anda akan memulai proyek. Apa artinya menjadi manajer produk? Mengapa sangat penting untuk memiliki seseorang dalam tim yang bertanggung jawab untuk mengelola suatu produk? Apa sebenarnya yang harus dilakukan manajer produk? Kami dapat berbicara banyak teori seputar posisi ini dan mengarahkan Anda ke ratusan artikel di Internet, tetapi apakah ada yang lebih baik daripada berbicara dengan orang sungguhan?

3.2 Wawancara Dengan Manajer Produk

Keuntungan besar bekerja di perusahaan perangkat lunak adalah kita tidak perlu menghabiskan waktu mencari orang sungguhan untuk diajak bicara. Mereka mengelilingi kita setiap hari. Untuk buku ini, Solikhan mewawancarai Lawrence, manajer produk yang saat ini bekerja dengannya di OptioPay. Wawancara ini berlangsung antara Maret dan April 2018.

3.3 Bagaimana Orang Menjadi Manajer Produk dan Apa Yang Mereka Tangani

Solikhan: Apa sebenarnya arti peran Anda?

Lawrence: Saat ini di OptioPay, peran saya adalah campuran antara manajer produk dan pemilik produk. Sebagai manajer produk, saya bertanggung jawab untuk mencapai tujuan bisnis produk saya, yang mencakup penetapan strategi untuk mencapai tujuan “indikator produk utama” (KPI). Sebagai pemilik produk, saya bertanggung jawab untuk mengelola Product Backlog, yang mencakup pengungkapan item backlog produk dengan jelas,

memesan item dalam product backlog, sambil mengoptimalkan nilai pekerjaan yang dilakukan dalam tim.

Solikhan: Sudah berapa lama Anda bekerja sebagai manajer produk?

Lawrence: Saya telah bekerja sebagai manajer produk sejak tahun 2012. Sebelumnya, gelar saya adalah desainer game, yang memiliki beberapa elemen peran produksi. Sebagai seorang desainer game, saya akan merancang konsep dan menghidupkannya untuk membentuk pengalaman yang dapat dimainkan. Ini terletak di landasan proses produksi, jadi secara alami memaparkan saya pada beberapa tugas manajemen produk.

Solikhan: Apa yang kamu pelajari?

Lawrence: Saya memiliki gelar sarjana dalam studi manajemen dan gelar master dalam media digital. Tujuan dari kurikulum gelar master saya adalah untuk membuat lulusan siap kerja dengan keterampilan yang fleksibel, daripada spesialisasi tunggal yang memberi saya paparan berbagai jalur karir. Yang paling menonjol adalah peran produksi. Selama master saya, saya menemukan mentor yang mengajari saya dan membimbing saya di jalan yang sekarang dikenal sebagai manajemen produk.

Solikhan: Bagaimana Anda memulai karir Anda sebagai manajer produk?

Lawrence: Transisinya tidak sesederhana jawaban ini. Rekan kerja saya memutuskan untuk pindah dari peran manajer produk. Saya secara alami mengisi celah dan mengambil alih peran.

Solikhan: Apa yang paling Anda nikmati di hari Anda?

Lawrence: Menghadapi tantangan dan mengatasinya bersama dengan tim pengembangan. Tidak ada yang lebih baik daripada melihat semua kerja keras terwujud dan memiliki bukti bahwa itu memiliki dampak positif!

Solikhan: Apakah Anda harus menghadapi stres? Dari mana asalnya?

Lawrence: Kadang ya, tapi hampir selalu bisa diatur. Sebagian besar waktu stres ini diseduh sendiri dalam pikiran saya dari situasi yang telah disaksikan di masa lalu. Ini mulai memperkuat pikiran negatif. Mengatasinya sederhana—terimalah bahwa itu adalah bagian normal dari kehidupan dan itu dapat dan akan diatasi. Tetap tenang dalam situasi ini adalah yang paling penting.

Perbedaan Antara Pemilik Produk dan Pengelola Produk

Solikhan: Kami tahu tentang peran yang disebut pemilik produk. Apa perbedaan antara pemilik produk dan manajer produk?

Lawrence: Bagi saya perbedaan utama antara pemilik produk dan manajer produk adalah peran apa yang dimainkan orang tersebut dalam tim scrum versus peran yang bertanggung jawab untuk mencapai tujuan bisnis produk. Pemilik produk menurut panduan scrum harus:

- Mengekspresikan item Product Backlog dengan jelas;
- Memesan item dalam Product Backlog untuk mencapai tujuan dan misi terbaik;
- Mengoptimalkan nilai pekerjaan yang dilakukan tim pengembangan;
- Memastikan bahwa Product Backlog terlihat, transparan, dan jelas bagi semua orang dan menunjukkan apa yang akan dikerjakan tim scrum selanjutnya; dan,
- Memastikan tim pengembang memahami item-item di Product Backlog hingga level yang dibutuhkan.

Sedangkan manajer produk bagi saya:

- Membuat strategi untuk mencapai tujuan bisnis;
- Mewakili suara pasar dan pelanggan;
- Menghasilkan peta jalan jangka panjang;
- Melakukan analisis harga dan pesaing; dan
- Memiliki visi produk dan bertanggung jawab penuh untuk mendorong visi ini dengan pemangku kepentingan lainnya.

3.4 Keterlibatan dalam Jalur Dari Ide ke Produk

Solikhan: Dalam perjalanan dari ide ke produk yang diterapkan, seberapa banyak Anda terlibat dalam prosesnya?

Lawrence: Saya terlibat dalam hampir setiap langkah proses. Jika sebuah ide baru sesuai dengan tujuan dan visi bisnis, maka ide tersebut melalui proses yang melibatkan saya melakukan hal-hal berikut:

- Yang paling penting adalah memvalidasi ide! Ini adalah waktu yang paling murah untuk memikirkan apakah ide Anda akan berhasil atau tidak. Buat prototipe secepat mungkin, uji, dan, lebih baik lagi, jual!
- Jika ide tersebut terbukti berhasil, tentukan apa MVP Anda; pastikan pemangku kepentingan Anda mengetahui apa yang dimaksud dengan MVP;
- Selanjutnya adalah mengambil ini dengan mengetahui apa yang Anda dan tim Anda setujui tentang apa yang harus dibangun terlebih dahulu dalam MVP Anda;
- Selama pengembangan jika ada alasan seperti wawasan pasar baru, atau pesaing baru yang membuat alasan kuat untuk berporos, maka bersedia untuk mewujudkannya; dan
- Pada akhirnya, memastikan bahwa produk dikirim ke pasar yang tepat pada waktu yang tepat untuk akhirnya memenuhi tujuan bisnis.

Solikhan: Bagaimana Anda mendapatkan ide untuk dimasukkan ke dalam peta jalan? Siapa pemangku kepentingan?

Lawrence: Ide datang dari banyak tempat. Dari setiap departemen dan setiap fungsi. Sebagian besar dari klien, pelanggan, orang-orang yang bekerja erat pada produk dan produk serupa lainnya. Saya mencoba mengingatkan orang-orang yang terlibat bahwa ide itu murah, tetapi apa yang terjadi setelahnya bisa menjadi sangat mahal. Oleh karena itu, ide-ide dikumpulkan dan dicocokkan dengan tujuan bisnis dan visi produk. Jika ada kecocokan, mereka masuk ke daftar ide tertentu, yang dinilai berdasarkan KPI yang dapat mereka pengaruhi. Jika mereka berdampak pada KPI yang sedang kami kerjakan secara positif, dalam kuartal ini, maka mereka akan ditempatkan di jalur cepat dan dibuat prototipe dengan cara secepat mungkin untuk diuji. Sering gagal dan gagal cepat telah bekerja yang terbaik bagi saya dalam banyak kasus.

Solikhan: Apakah penentuan prioritas berdasarkan kebutuhan bisnis? Apakah ini berarti Anda, manajer produk, harus memiliki pemahaman bisnis yang baik dan kuat?

Lawrence: Dalam jawaban saya atas pertanyaan sebelumnya, saya menyebutkan kriteria yang saya gunakan untuk memetakan ide ke prioritas mana. Pemetaan ini juga didasarkan pada KPI, yang diturunkan dengan mempertimbangkan tujuan dan kebutuhan bisnis. Memiliki pemahaman tentang mengapa? sangat penting. Sebagai manajer produk, Anda harus memiliki dukungan yang sama atau bahkan lebih tinggi terhadap KPI yang ditargetkan. Keberhasilan produk Anda bergantung pada bisnis yang dihasilkannya (hampir semua kasus komersial). Untuk alasan ini, beberapa perusahaan memiliki pemilik bisnis untuk meningkatkan dampak bisnis selama pengembangan produk.

3.5 Mengelola Produk dari Area Bisnis yang Berbeda

Solikhan: Apa bedanya menjadi manajer produk produk di area bisnis yang berbeda?

Lawrence: Saya dapat berbicara dari pengalaman saya datang dari bisnis game ke fintech. Inti dari hari-hari dan teknik Anda dapat tetap sama di seluruh industri, tetapi Anda perlu beradaptasi dengan persyaratan bisnis untuk produk yang sukses. Mempelajari ini lebih mudah jika Anda terbuka untuk berubah dan siap membuat kesalahan.

3.6 Kekacauan Terbesar

Solikhan: Ceritakan tentang kesalahan terbesar dalam karir Anda sebagai manajer produk.

Lawrence: Saat saya mengerjakan game free-to-play, tujuan kami adalah meningkatkan KPI retensi harian. Untuk meningkatkan retensi, kami memperkenalkan mini-game, yang memiliki hadiah (termasuk yang premium), yang Anda dapatkan lebih banyak jika Anda kembali setiap hari. Setelah memperkenalkan ini, saya perhatikan bahwa itu bekerja dengan sangat baik; oleh karena itu untuk mendapatkan lebih banyak dari mini game (yang dinikmati pemain kami), saya memperkenalkan fitur monetisasi di bagian atas. Sekarang Anda dapat membayar dalam mata uang game untuk melewati mini game lebih cepat tanpa harus menunggu. Saya menguji[1] A/B ini untuk waktu yang singkat dan memutuskan untuk menjadi agresif dan menghidupkannya untuk semua pemain. Tiga hingga empat bulan setelah rilis, kami mulai melihat penurunan pendapatan yang stabil tanpa mengetahui apa penyebabnya. Setelah banyak menggali data selama sebulan dan beberapa pengujian A/B, saya menyadari itu disebabkan karena rilis fitur monetisasi yang saya perkenalkan di atas. Fitur ini memungkinkan pengguna membayar untuk mendapatkan item premium lebih cepat, membuat pembelian drop-in item premium menggunakan toko dalam game dari waktu ke waktu.

3.7 Sukses Terbesar

Solikhan: Ceritakan tentang kesuksesan terbesar dalam karir Anda sebagai manajer produk.

Lawrence: Bersama dengan tim kecil yang sangat berbakat, saya dapat mengambil permainan yang sedang saya kerjakan, mulai dari menghasilkan angka pendapatan 4 digit awal dalam sehari hingga pendapatan bernilai 5 digit yang layak. Saya mampu

[1] Pengujian A/B, dengan kata yang sangat sederhana, adalah pengujian dua versi dari variabel yang sama (misalnya, halaman web) untuk memeriksa mana yang terjual lebih banyak atau berkonversi lebih baik. Biasanya itu adalah sedikit perubahan sehingga mudah untuk dimanipulasi dan dikembalikan jika diperlukan.

menumbuhkan pendapatan 8 kali lipat dalam 2 tahun saat saya mengerjakan game itu. Game ini akan dihentikan karena statistik awal, yang tidak terlihat menjanjikan, tetapi berkat tim yang berdedikasi, kami dapat membalikkannya. Seperti yang Anda lihat, kehidupan seorang manajer produk adalah tantangan besar di mana tim memainkan peran besar. Sekarang mari kita lihat lebih dekat tantangan ini dan selesaikan!

3.8 Persiapan

Meskipun kami hanya ingin mengotori tangan kami dan mulai mengerjakan produk kami, kami tidak mampu melakukannya tanpa persiapan apa pun. Kita harus memastikan bahwa semua *pemangku kepentingan* selaras dengan persyaratan awal, tenggat waktu, pencapaian, dan risiko. Kita harus memastikan bahwa *tanggung jawab* dan *kepemilikan* didistribusikan dengan benar di antara anggota tim dan bahwa setiap orang dalam tim memahami dampaknya dalam proyek. Jika ada klien eksternal, kami harus menjamin bahwa kami membangun proses yang lancar dan *saluran komunikasi* yang benar dengan mereka. Kami mungkin membuat perjanjian kerja yang terdokumentasi agar semuanya terlihat dan jelas. Kata kunci utama Anda di sini adalah ***visibilitas*** dan ***transparansi***. Jangan biarkan siapa pun merasa bahwa keputusan penting sedang dibuat tanpa mempertimbangkan pendapat mereka. Jangan tinggalkan siapa pun. Setiap orang harus merasakan dampak dan pentingnya mereka dalam proyek. Setiap orang harus merasa bahwa mereka mendorong proyek untuk sangat sukses. Maka pasti akan ada satu.

3.9 Persyaratan dan Peta Jalan

Sebelum mengumpulkan seluruh tim, Anda harus memikirkan persyaratan *tingkat tinggi* dan membuat *peta jalan* sederhana. "Ya Tuhan, sendirian?" Anda mungkin berseru. Jika ini adalah proyek kecil dan sederhana, maka Anda dapat mempertimbangkan untuk melakukan latihan otak ini sendiri dan kemudian mempresentasikannya kepada tim dan meminta pendapat mereka. Untuk proyek yang lebih kompleks dengan lebih banyak pemangku kepentingan yang terlibat, Anda pasti tidak ingin menjadi satu-satunya yang bertanggung jawab atas artefak bootstrap penting ini. Tentukan orang-orang kunci dan bertukar pikiran dengan mereka.

Ambil seseorang yang memiliki *pemahaman bisnis*, beberapa *orang teknis*, dan, jika berlaku, seseorang yang bertanggung jawab dari *sisi klien*. Pergilah dengan semua orang ini ke suatu tempat di mana Anda dapat mengadakan sesi strategi tanpa terganggu. Tempat di luar kantor bekerja lebih baik! Jika memiliki beberapa alam dan kegiatan di luar ruangan, maka lebih baik lagi! Pernahkah Anda mendengar tentang cara berpikir yang terfokus dan tersebar yang otak Anda beralih antara saat Anda belajar? Ternyata, tidak melupakan mode ini juga berguna untuk sesi brainstorming apa pun. Anda duduk mengelilingi meja, Anda menyajikan beberapa masalah. Untuk beberapa waktu, Anda benar-benar fokus pada masalah ini dan mencoba mencari solusinya. Setelah beberapa waktu, ada baiknya untuk berjalan-jalan di mana pikiran Anda dapat berjalan dengan bebas seperti alam sekitarnya. Ini membantu untuk membuka batas di sekitar otak kita dan sampai pada beberapa solusi yang tidak standar dan sederhana.

Kami memiliki seorang guru ketika kami belajar di Portugal yang juga merupakan pendiri salah satu perusahaan terbesar yang menyediakan solusi perangkat lunak seluler. Tuan yang sangat cerdas ini akan mengadakan pertemuan bisnisnya di …sebuah lapangan golf! Dia akan menjelaskan bahwa mode pemikiran terfokus dan menyebar bekerja cukup bagus di sana. Ternyata alam di sekitar perkemahan membantu pikiran menyebar, dan saat Anda berkonsentrasi pada gerakan Anda, Anda menjadi sangat fokus. Tidak hanya ide dan solusi suksesnya yang dihasilkan di lapangan golf, tetapi semua kontrak dan perjanjian terbaiknya juga ditandatangani di sana! Bermain golf, menandatangani kontrak, menghasilkan ide.

Hitung pendapatan Anda. Begitu banyak keuntungan, bukan begitu? Sepertinya kita lari dari topik utama bagian ini. Persyaratan dan peta jalan, ingat? Jadi, Anda telah mengumpulkan kelompok kecil untuk menentukan persyaratan tingkat tinggi. Apa berikutnya? Bagaimana cara melakukannya? Apa yang harus dibahas dalam pertemuan ini? Pikirkan tentang versi awal produk dan tuliskan karakteristik apa yang dimilikinya. Apakah Anda ingat gambar tentang MVP? Ini dimulai dengan skateboard. Jika kita memikirkannya, kita akan segera menentukan persyaratan yang diperlukan:

- Itu harus memiliki roda.
- Harus memiliki papan.
- Harus memiliki semacam mekanisme yang menyatukan papan dan roda.
- Itu harus bergerak menggunakan roda yang menempel pada papan.
- Harus memiliki mekanisme rem.

Ini adalah persyaratan yang sangat diperlukan. Tanpa ini skateboard kami bukanlah skateboard. Kemudian Anda dapat menentukan beberapa fitur yang bagus untuk dimiliki, seperti desain papan yang indah, ergonomis, kapasitas lompatan, motor listrik, apa pun. Itu sebabnya Anda perlu memiliki seseorang dengan pemahaman bisnis dan juga seseorang yang teknis. Orang dengan pemahaman bisnis akan membantu Anda untuk menentukan apakah fitur ini atau yang lain sangat diperlukan untuk kesuksesan bisnis dan KPI bisnis, sedangkan orang teknis mungkin menunjukkan beberapa jebakan implementasi di sisi teknis. Semua ini adalah proses rekayasa. Rekayasa adalah tentang kreativitas, inovasi, teknologi, dan proses. Bahkan Wikipedia tahu ini:

> *Teknik adalah aplikasi kreatif sains, metode matematika, dan bukti empiris untuk inovasi, desain, konstruksi, dan pemeliharaan struktur, mesin, material, perangkat, sistem, proses, dan organisasi.* *Wikipedia: https://en.wikipedia.org/wiki/Engineering*

Setelah Anda menentukan persyaratan tingkat tinggi, Anda perlu menentukan peta jalan tingkat tinggi Anda. Jika ada uang yang terlibat, siapa pun yang membayar uang ini akan tertarik untuk mengetahui tonggak penting pengiriman proyek. Jika tidak ada uang yang terlibat, tim harus mengetahui hari-hari penting pengiriman potongan produk mereka. Mengapa tanggal penting? Pertama-tama, dari pengalaman kami, tenggat waktu sebenarnya mendorong proses dan kemajuan. Tanpa merencanakan beberapa tonggak dan tanggal pengiriman, Anda dapat yakin bahwa tidak ada yang akan dikirimkan.

Tidak ada batasan untuk perfeksionisme. Hal-hal dapat dipoles, disempurnakan, dan ditingkatkan selamanya. Tanggal pengiriman membantu menghentikan proses perbaikan

yang tidak pernah berakhir ini. Kedua, dan mungkin terdengar sangat kontroversial, tenggat waktu cukup memotivasi. Bagaimana sesuatu yang disebut “tenggat waktu”, sebuah istilah yang biasanya berkaitan dengan stres dan tekanan, bisa menjadi motivasi?

Nah, ini karena setiap pencapaian harus dirayakan! Tonggak pencapaian yang dicapai harus dirayakan bersama seluruh tim. Tim biasanya bekerja keras untuk mencapai tonggak dan menjadi bahagia ketika tujuan tercapai. Jadikan perayaan sebagai bagian dari budaya tim Anda. Bekerja keras, bermain lebih keras! Itulah mengapa sangat penting untuk memiliki orang teknis dengan Anda. Orang teknis ini akan membantu Anda menjalankan estimasi yang sangat kasar untuk membuat peta jalan Anda. Mari, misalnya, pikirkan peta jalan untuk skateboard MVP.

- Membangun roda: 2 minggu
- Membangun papan: 1 minggu
- Pasang roda ke papan: 3 minggu
- Pasang mekanisme pengereman: 2 minggu
- Tes: 2 minggu

kebijakan gal, dan bahkan musim tahun ini! Beberapa orang, misalnya, merasa sedikit tertekan dan kurang termotivasi selama musim dingin, yang mempengaruhi produktivitas. Di sisi lain, orang cenderung berpesta lebih banyak selama musim panas, yang mungkin juga memengaruhi mereka yang menderita mabuk. Namun jangan khawatir, pencapaian Anda tidak perlu dinyatakan dalam tanggal yang akurat. Mereka dapat mewakili minggu atau bahkan bulan untuk proyek yang panjang dan kompleks. Tanggal pasti dapat disesuaikan selama proyek bergerak maju. Selain itu, Anda memiliki tim yang luar biasa yang akan membantu Anda dengan penyesuaian peta jalan selama pertemuan awal.

3.10 Kick-Off

Sekarang setelah Anda memiliki persyaratan dan peta jalan yang dijelaskan secara kasar, Anda dapat mengumpulkan tim Anda untuk rapat awal. Buatlah acara yang serius namun menyenangkan. Serius karena kami memulai proyek baru, dan semua orang harus selaras dengan tanggung jawab besar yang ada di pundak kami. Menyenangkan karena proyek baru pasti menyenangkan! Karena tim akan berpartisipasi dalam petualangan baru. Karena akan ada perayaan. Karena kesuksesan besar sedang menunggu kita dan kita harus siap untuk itu. Ketika orang-orang meninggalkan rapat awal, mereka harus benar-benar selaras dengan hal-hal berikut:

- Tentang apa proyek itu?
- Mengapa proyek ini penting
- Apa tanggung jawab setiap orang
- Apa hasil dan pencapaian penting
- Bagaimana kita akan mencapai hasil yang penting

Setiap pertemuan dimulai dengan agenda. Kick-off meeting tidak terkecuali. Persiapkan agenda Anda dengan sangat jelas dan kirimkan ke semua orang yang akan

berpartisipasi. Jangan menyimpan detail kecil apa pun untuk diri Anda sendiri. Agenda rapat kick-off bisa seperti ini:

- Deskripsi proyek (10 menit)
- Mengapa kita melakukan ini? (5 menit)
- Pembahasan persyaratan (5 menit)
- Diskusi dependensi (misalnya, kebutuhan teknologi atau dependensi pihak ketiga) (10 menit)
- Garis waktu dan peta jalan (15 menit)
- Peran dan tanggung jawab (10 menit)
- Tanya jawab (10 menit)

Opsional, Anda dapat menyertakan deskripsi pelanggan (jika ada), definisi proses (jika tidak ada sebelumnya), saluran komunikasi, diskusi teknis, perkiraan anggaran, dll. Ingat, setiap proyek sangat unik dan membutuhkan pendekatan yang berbeda untuk memulai. Misalnya, untuk menulis buku ini (karena ini juga merupakan proyek) proses awal kami cukup sederhana. Kami mendefinisikan garis besar dan tenggat waktu kasar untuk setiap bab, dan penerbit membuat kontrak berdasarkan data itu dan mengirimkannya kembali kepada kami. Setelah itu kami membaca kontrak, menandatanganinya, dan mengirimkannya kembali. Sebelum setiap bab, kami duduk bersama dan melakukan sesi curah pendapat kecil tentang konten bab dan menentukan siapa yang menulis apa untuk setiap bagian bab. Dengan demikian, proses kami lancar dan lugas, tetapi ingat, kami adalah tim yang terdiri dari dua orang yang sudah berjalan lama. :)

Untuk proyek platform pembelajaran kami, prosesnya benar-benar berbeda. Kami mengirim tiga bab pertama dari buku ini ke tim kami, sehingga mereka selaras dengan proyek dan persyaratannya, dan kami mengadakan pertemuan dengan mereka untuk bertukar pikiran tentang persyaratan dan peta jalan. Garis waktu roadmap disesuaikan dengan tenggat waktu bab-bab buku sejak produk dibangun selama penulisan buku ini. Juga, tim kami tidak hanya terdiri dari 10 orang yang kami presentasikan di bab sebelumnya. Tim kami adalah Anda. Dan fakta ini membuat proyek ini semakin fantastis. Apakah kamu siap? Anda sebaiknya melakukannya karena Anda sudah melakukannya!

3.11 Komitmen dan Batas Waktu

Kata komitmen sangat penting untuk setiap proyek dan setiap detik proyek. Mari kita lihat lebih dekat definisi kata ini (cukup Google saja, seperti yang ditunjukkan pada Gambar 3-1):

komitmen

bahasa Indonesia [sunting]

Nomina

komitmen (posesif ku, mu, nya; partikel: kah, lah)

komitmen

1. perjanjian (keterikatan) untuk melakukan sesuatu; kontrak:
   Perkumpulan mahasiswa seharusnya mempunyai komitmen terhadap perjuangan reformasi

Etimologi

Kata turunan

Sinonim

Frasa dan kata majemuk

Terjemahan[?]

Terjemahan

Lihat pula

- Semua halaman dengan kata "komitmen"
- Semua halaman dengan judul mengandung kata "komitmen"
- Lema yang terhubung ke "komitmen"

Gambar 3-1. Detail arti kata "komitmen" menurut wikitionary

Lihat sinonimnya: dedikasi, pengabdian, loyalitas, tanggung jawab, tanggung jawab… Komitmen berarti dedikasi terhadap proyek dan produk, dan itu adalah salah satu fondasi kesuksesan mereka. Itu berarti kesetiaan kepada anggota tim. Selama pelaksanaan proyek Anda berkomitmen untuk tenggat waktu tertentu. Kemungkinan akan ada tekanan dan sejumlah tekanan, yang tidak selalu buruk jika didistribusikan secara merata di antara semua anggota tim. Sangat penting untuk proyek bahwa setiap anggota tim berkomitmen untuk itu dan mengambil kepemilikan penuh dari apa yang mereka lakukan.

Jika setidaknya satu anggota tim memiliki perasaan, "Saya hanya melakukan tugas ini dan pulang," alih-alih, "Ini adalah proyek saya dan saya bertanggung jawab sepenuhnya," maka proyek tersebut tidak akan sepenuhnya berhasil. Kami telah berada di proyek-proyek di mana segala sesuatunya tidak berjalan seperti yang diharapkan. Tenggat waktu yang sulit akan datang, dan masih banyak pekerjaan yang harus diselesaikan. Beberapa tim menghadapinya dengan menjadi organisme yang unik, sesuatu yang besar, koheren, kuat, dan teguh. Mereka akan mencoba untuk mencapai tonggak komitmen apa pun yang terjadi. Mereka akan saling membantu tidak hanya dengan tugas-tugas yang berhubungan dengan proyek tetapi juga dengan membuat kopi, membawa makanan, dan mengurus beberapa masalah pribadi. Tim-tim ini akan menjadi teman baik selama cuaca badai dan selama-lamanya.

Di tim lain, para anggota akan berperilaku seolah-olah mereka bukan tim sama sekali! Masing-masing akan terus melakukan pekerjaan mereka, tidak khawatir tentang tenggat

waktu yang mendekat, dan tidak peduli dengan anggota tim lainnya. Tak perlu dikatakan, dalam kasus pertama tim memenuhi tenggat waktu untuk memberikan set fitur berkomitmen penuh, sementara dalam kasus kedua beberapa fitur harus dijatuhkan dan beberapa percakapan yang sulit harus diadakan dengan klien.

Ketika kami bertanya kepada tim kedua mengapa mereka tidak merasa khawatir dengan tenggat waktu yang semakin dekat dan tidak memberikan fitur yang telah disepakati sebelumnya, mereka menjawab bahwa ini bukan kesalahan mereka. Kesalahan ada pada manajemen. Menurut pendapat mereka, manajer produk bertanggung jawab atas tenggat waktu dan komitmen. Jangan biarkan ini terjadi pada tim Anda. Seluruh tim bertanggung jawab. Setiap anggota tim memegang kepemilikan proyek. Sangat penting untuk menyampaikan ide ini selama pertemuan pertama.

Buatlah agar semua orang bertanggung jawab, buat semua orang merasa penting, dan buat semua orang menjadi bagian tak terpisahkan dari mekanisme kesuksesan. Cara yang sangat sederhana untuk membuat setiap orang merasa bertanggung jawab adalah dengan banyak bertanya tentang pendapat mereka dan menggunakan kata “kita” atau “kami” ketika berbicara tentang kesuksesan dan kata “saya” atau “aku” ketika berbicara tentang kegagalan.

Bukan manajer produk yang membangun proyek; itu adalah tim yang membawa proyek melalui jalan menuju sukses. Bukan manajer produk yang menentukan tanggal pengiriman; itu adalah sebuah tim, memperkirakan tanggal-tanggal itu dan menyetujuinya. Bukan manajer produk yang merayakan pencapaian tujuan; itu adalah tim yang berpesta keras sama sekali. Semua ini tidak berarti bahwa ketika ada yang salah, manajer produk dapat lari dari tanggung jawab dengan mengatakan, "Kami sebagai tim gagal menyelesaikan proyek." Dalam hal ini, Anda, sebagai manajer produk, harus berdiri dan menanggung kegagalan Anda. Dan, percayalah, jika tim Anda menyerap budaya kepemilikan dan tanggung jawab, setiap anggota akan melakukan hal yang sama.

3.12 Persyaratan untuk MVP Kami

Sekarang saatnya untuk mulai mengumpulkan persyaratan untuk MVP kita. Seperti yang dinyatakan sebelumnya dalam buku ini, MVP adalah singkatan dari produk yang layak minimum, tetapi apa sebenarnya artinya? Sebagai permulaan, tidak ada jawaban yang jelas untuk itu—itu hanya tergantung… tetapi pada apa sebenarnya? Saat memikirkan MVP, kita perlu memahami lingkungan dan target. Untuk membuatnya sedikit lebih jelas, mari lakukan latihan kecil ini:

a. Apakah ada produk serupa lainnya di pasar?
b. Apa target audiensnya? Apakah untuk ceruk pasar tertentu?
c. Apakah saya perlu memiliki semacam fitur terobosan untuk meyakinkan orang agar memilih produk saya?
d. Berapa lama saya diperbolehkan untuk mengembangkan produk sampai keluar?
e. Apakah saya bergantung pada pihak eksternal? Misalnya, apakah saya memerlukan semacam sertifikasi atau lisensi untuk mulai menjual produk saya?

Daftarnya bisa bertambah tak terbatas, dan itu bisa menjadi masalah karena bisa menurunkan motivasi Anda. Jadi, aturan praktis pertama ketika memikirkan MVP adalah: tetap sederhana dan ringkas. Pertanyaan penting yang perlu dijawab akan selalu bergantung pada jenis produk yang akan Anda kirim. Beberapa contoh berikut:

a. Anda sedang membangun jaringan sosial baru. Jika jejaring sosial Anda tidak memiliki nilai jual unik (USP) untuk membuat orang beralih dari orang lain atau bergabung dengan Anda sejak awal, Anda tidak dapat memikirkan untuk mengirimkannya tanpa semua fitur dasar yang ditawarkan orang lain: menyukai dan berbagi pos, mencari konten, pengorganisasian acara, dll.
b. Anda sedang membuat produk X, dan baru-baru ini mengumumkan bahwa perusahaan Alpha mengirimkan produk Y yang serupa. Kedua produk tersebut akan menjadi produk besar berikutnya di pasar, karena tidak ada produk lain di luar sana. Sangat penting bahwa Anda adalah yang pertama atau setidaknya tidak menunggu terlalu lama setelah perusahaan lain meluncurkan produknya. Dapat diterima bahwa Anda mengurangi cakupan MVP menjadi yang pertama di pasar dan kemudian dengan cepat menutupi semua fitur yang Anda jatuhkan.
c. Ada beberapa aplikasi yang sudah ada di pasaran yang mengajarkan Anda cara memasak. Mereka memiliki beberapa fitur seperti menyukai resep, mengomentari video, membuat daftar belanja dari resep, apa saja. Di sisi lain, semua konten dalam bahasa Inggris. Anda ingin membuat produk yang sama persis tetapi untuk kelompok sasaran tertentu: orang-orang yang berbicara bahasa Portugis—tidak hanya mencakup Portugal tetapi juga Brasil. Karena Anda memiliki USP, tidak apa-apa untuk membuang beberapa fitur yang ditawarkan semua orang lain. Sepertinya mereka tidak dapat bersaing dengan Anda dalam waktu sesingkat itu karena mereka harus mengubah semua konten mereka ke dalam bahasa Portugis, dan itu menimbulkan banyak pekerjaan dan alokasi sumber daya. Karena orang lain mungkin masih berpikir atau benar-benar mengonversi semua konten mereka ke bahasa Portugis, Anda dapat menyelesaikan fitur yang Anda hapus pada awalnya dan tetap menjadi yang pertama di pasar dengan implementasi penuh dan dalam bahasa yang tidak dimiliki pesaing lain. Tidak meluncurkan semua fitur dan mengirimkannya dari waktu ke waktu juga bisa menjadi strategi yang baik sehingga Anda memberi pelanggan perasaan bahwa produk Anda sangat dinamis, dan Anda sering berupaya menawarkan fungsionalitas baru kepada mereka.

Latihan yang baru saja kita lakukan di sini cukup sederhana tetapi juga terlalu disederhanakan. Dalam kehidupan nyata dan ketika ada uang yang terlibat, riset pasar penuh, penilaian risiko, dan laporan harus dibuat. Keputusan akan dibuat dengan mempertimbangkan hal-hal tersebut. Bayangkan Anda memiliki uang selama 6 bulan untuk pengembangan dan setelah itu Anda harus segera mulai menjual produk Anda, jadi 6 bulan setelah peluncuran Anda sudah mencapai tujuan yang Anda dan investor Anda sepakati. Pembatasan ini, suka atau tidak suka, akan memengaruhi MVP Anda.

Apa yang perlu Anda pertahankan dari semua hal di atas adalah penting untuk memikirkan MVP sebelum Anda mulai dan bahwa Anda tidak selalu dapat sepenuhnya

mengontrol keputusan karena Anda mungkin bergantung pada faktor eksternal. Kembali ke kasus khusus kami, ini cukup menarik, dan kami akan memberi tahu Anda alasannya: kami tidak hanya bersaing dengan perusahaan lain tetapi juga bersaing dengan diri kami sendiri.

Sepertinya lucu, kan? Tapi kalau dipikir-pikir, memang benar. Kami sudah memiliki platform yang tersedia, memperlihatkan banyak fitur dan kami ingin membangun yang baru di mana versi pertama tidak akan menawarkan jumlah fitur yang sama. Kabar baiknya adalah kami memiliki USP yang cukup keren: konten kami. USP ini akan memberi kita kelonggaran tidak hanya terhadap pesaing eksternal kita, tetapi juga terhadap diri kita sendiri, karena isinya tidak akan berubah. Dengan cara ini kami dapat memikirkan MVP kami dengan cara yang berbeda dari yang kami pikirkan jika kami baru saja bergabung dengan pasar dari awal.

Di EdEra kami memiliki tiga set pengguna: siswa, guru, dan administrator. Jadi inilah keputusan pertama kami: MVP hanya akan mencakup aplikasi siswa. Keputusan ini tidak dibuat tiba-tiba; kami sadar bahwa kami tidak punya banyak waktu untuk menghasilkan MVP yang bagus, jadi tentu saja kami akan fokus pada pengguna platform yang paling penting: pelanggan kami yang sebenarnya. Selain itu, kami tidak meluncurkan atau mengelola kursus setiap hari, sehingga kami dapat hidup dengan sempurna di hari-hari awal tanpa sistem manajemen konten dan mendelegasikan semua itu kepada tim pengembangan, sehingga mereka melakukan ini secara manual di database, misalnya. Setelah menetapkan fokus, mari buat daftar (Tabel 3-1) fitur tingkat tinggi untuk platform kita. Kami membagi fitur menjadi blok logis yang akan cocok dengan bagian aplikasi seperti:

a. Pra-login: login dan registrasi
b. Tampilan kursus: daftar kursus yang dapat didaftarkan pengguna
c. Tampilan dasbor: daftar kursus yang dilakukan pengguna
d. Tampilan kursus: sebelum mendaftar
e. Tampilan kursus: setelah mendaftar
f. Tampilan profil: edit informasi pengguna

Ini adalah bagian dari aplikasi awal kami. Kami sekarang akan menjelaskannya secara lebih rinci dan apa yang harus disertakan di antaranya. Untuk itu kami akan membuat tabel dan menetapkan setiap fitur yang kami inginkan ke bagian aplikasi yang menjadi miliknya. Semua fitur dijelaskan dalam perspektif pengguna (siswa).

Tabel 3-1. Fitur untuk MVP Platform Pendidikan

Fitur	Seksi	Deskripsi
Formulir Registrasi	A	formulir sederhana dengan email dan kata sandi. pengguna dapat menyelesaikan pendaftaran setelah di bagian profil.
Formulir Login	A	formulir sederhana dengan email dan kata sandi dan tombol login.
Pemulihan Password	A	formulir di mana pengguna dapat mengetikkan alamat email mereka, sehingga tautan reset dikirim.

Bar Navigasi	ALL	bilah navigasi dengan bagian aplikasi. selalu terlihat seperti header.
Footer	ALL	bilah navigasi dengan bagian aplikasi. selalu terlihat seperti header.
List semua kursus	B	daftar sederhana dengan semua kursus yang tersedia (mungkin harus diberi halaman di masa mendatang seiring bertambahnya daftar ini); orang dapat mengklik kursus untuk mengetahui lebih banyak tentangnya
Daftar semua kursus	C	daftar sederhana (diberi halaman di masa depan karena dapat bertambah) dengan semua kursus yang didaftarkan pengguna dan statusnya. pengguna dapat mengklik kursus untuk melanjutkan atau memeriksa penilaian jika sudah selesai.
Daftar semua kursus yang terdaftar	D	pengguna akan merasakan apa yang akan mereka lakukan dalam kursus apa yang akan mereka butuhkan dan tujuan kursus
Daftar semua informasi tentang kursus	D	pengguna dapat mendaftar di kursus. ini hanya bisa dilakukan sekali seumur hidup.
Lakukan kursus	E	pengguna dapat melihat konten kursus dan berinteraksi dengannya.
Render dan putar Video	E	pengguna dapat memutar video yang terikat pada kursus.
Render Teks	E	penjelasan misalnya untuk tujuan pengajaran diberikan kepada pengguna.
Render dan jawab kuis pilihan	E	pengguna dapat menjawab kuis milik kursus. hasilnya disimpan dalam penyimpanan persistensi, sehingga tidak pernah hilang.
Menyediakan umpan balik	E	pengguna harus mendapatkan umpan balik jika tebakannya benar atau salah.
Render jawaban yang benar	E	mengungkapkan jawaban jika pengguna ingin atau menebaknya salah pertama kali.
Melengkapi kursus	E	pengguna menerima penilaian akhir setelah menyelesaikan kursus. tampilan ini adalah satu-satunya tampilan yang dapat diakses pengguna setelah kursus selesai.
Melihat prifl pengguna	D	pengguna dapat memeriksa data pribadi mereka.
Perubahan Password	D	pengguna dapat mengubah kata sandi mereka jika yang sekarang benar.
Perubahan data pribadi	D	pengguna dapat mengubah atau menambahkan nama, jenis kelamin, dan tanggal lahir mereka.

Ketika kami mulai meletakkan fitur yang ingin kami lakukan untuk MVP, itu juga mulai menjadi lebih jelas bagaimana aplikasi akan terstruktur. Sebagai contoh, sekarang kita dapat melihat dengan jelas bahwa setiap kursus memiliki semacam "modul", seperti video atau hanya teks dan bahwa setiap modul ini dapat memiliki kuis yang terkait dengannya. Ini akan menjadi lebih jelas ketika kita mulai merancang, menyempurnakan, dan memecah setiap fitur menjadi cerita pengguna yang lebih kecil dan mandiri, yang persis seperti yang akan kita lakukan di bab berikutnya.

3.13 Kesimpulan

Dalam bab ini kami memulai proyek kami. Kami memakai topi manajer proyek untuk melakukannya dengan benar. Kami telah melakukan wawancara dengan Lawrence, seorang manajer produk dengan pengalaman beberapa tahun, jadi kami bisa mendapatkan gambaran nyata tentang apa peran ini. Kami menjadi akrab dengan istilah-istilah seperti peta jalan, estimasi, persyaratan, pertemuan awal, dan prioritas. Kami membahas betapa pentingnya mem-bootstrap proyek dengan benar dan membuat semua orang merasa penting sejak awal. Kami membahas akuntabilitas anggota tim, kepemilikan, dan tanggung jawab. Kami juga membahas tentang pengumpulan persyaratan untuk MVP.

Kami membahas tidak hanya bagaimana menganalisis persyaratan, tetapi juga bagaimana mengumpulkan persyaratan awal untuk proyek yang akan kami bangun di sepanjang buku ini. Dalam bab berikutnya kita akan melanjutkan ke tahap perancangan dari proses pengembangan kita. Kami akan memakai topi seorang desainer dan mendiskusikan bagaimana desainer melakukan pekerjaan mereka. Cerita pengguna, maket, gambar rangka... Apakah Anda siap untuk persyaratan ini? Kalau begitu mari kita mulai, tetapi pertama-tama mari kita lakukan pemeriksaan cepat.

3.14 Tes Kemampuan

- Apa singkatan dari MVP?
    - Money Value Project
    - minimum Viable product
    - maximum Viable product
    - minimum Vulnerable product
- Tim Anda mengikuti kerangka scrum, Anda banyak berurusan dengan backlog, memprioritaskan tugas, dan mendistribusikannya ke seluruh tim. Anda berpartisipasi dalam kegiatan seperti pertemuan harian, retrospektif, dan perencanaan. Siapa kamu?
    - manajer proyek
    - insinyur keandalan situs
    - manajer produk
    - pemilik produk
- Anda telah menentukan persyaratan awal dan menyetujui tujuan bisnis untuk mVp dengan pemilik bisnis. tim masih tidak menyadari apa yang sedang terjadi. Pertemuan seperti apa yang harus Anda kumpulkan secepatnya?

    - stand up harian
    - pertemuan retrospektif
    - acara di luar kantor dengan bir dan balon
    - rapat awal proyek
- Kata ganti seperti apa yang harus digunakan untuk menciptakan rasa tanggung jawab dan tanggung jawab setiap orang?
    - saya atau Aku
    - Anda
    - Kami atau Kita
    - dia

jika Anda menjawab “produk yang layak minimum—pemilik produk—pertemuan awal proyek—Kami atau Kita,” maka Anda dapat melanjutkan ke bab berikutnya!

BAB 4
DESAIN YANG BERPUSAT PADA PENGGUNA

Dalam bab sebelumnya, kami menganalisis persyaratan, berkomitmen pada strategi pengembangan, dan mendefinisikan peta jalan sederhana. Dalam bab ini kita akan berbicara tentang salah satu tahap pengembangan produk yang paling sulit: desainnya. Seorang desainer adalah orang yang menghubungkan semua titik. Desainer menghubungkan kebutuhan bisnis dan persyaratan produk dengan tantangan teknis dan, yang paling penting, kepada pengguna akhir. Kata kunci dalam cerita ini adalah pengguna. Kami sedang membangun produk kami serta meningkatkan dan menyempurnakannya sehingga dapat digunakan. Kami berharap pengguna kami senang dengan produk kami. Kami berharap produk dan layanan kami memiliki lebih banyak pengguna.

Dalam bab ini Anda akan mengenakan topi seorang desainer dan Anda akan melihat bahwa desain yang bagus bukanlah tentang kemampuan menggambar. Sebuah desain yang baik adalah tentang orang-orang, kebutuhan dan masalah mereka, dan cara mengatasi kebutuhan dan memecahkan masalah. Desain yang tepat menempatkan penggunanya sebagai pusat perhatian. Itulah mengapa dalam bab ini kita akan berbicara tentang desain yang berpusat pada pengguna. Bab ini bukan tentang desain produk Anda, bab ini tentang pengguna produk Anda. Ini tidak mengajarkan tentang bagaimana menjadi seorang desainer melainkan memberikan beberapa wawasan tentang mengarahkan proses desain dengan pengguna sebagai pusat perhatian.

Catatan : Jika Anda berpikir bahwa Anda harus melewatkan bab ini karena Anda tidak terlalu kreatif atau karena Anda tidak akan pernah menjadi seorang desainer, atau karena "semua hal indah ini bukan milik saya", mohon pertimbangkan kembali.

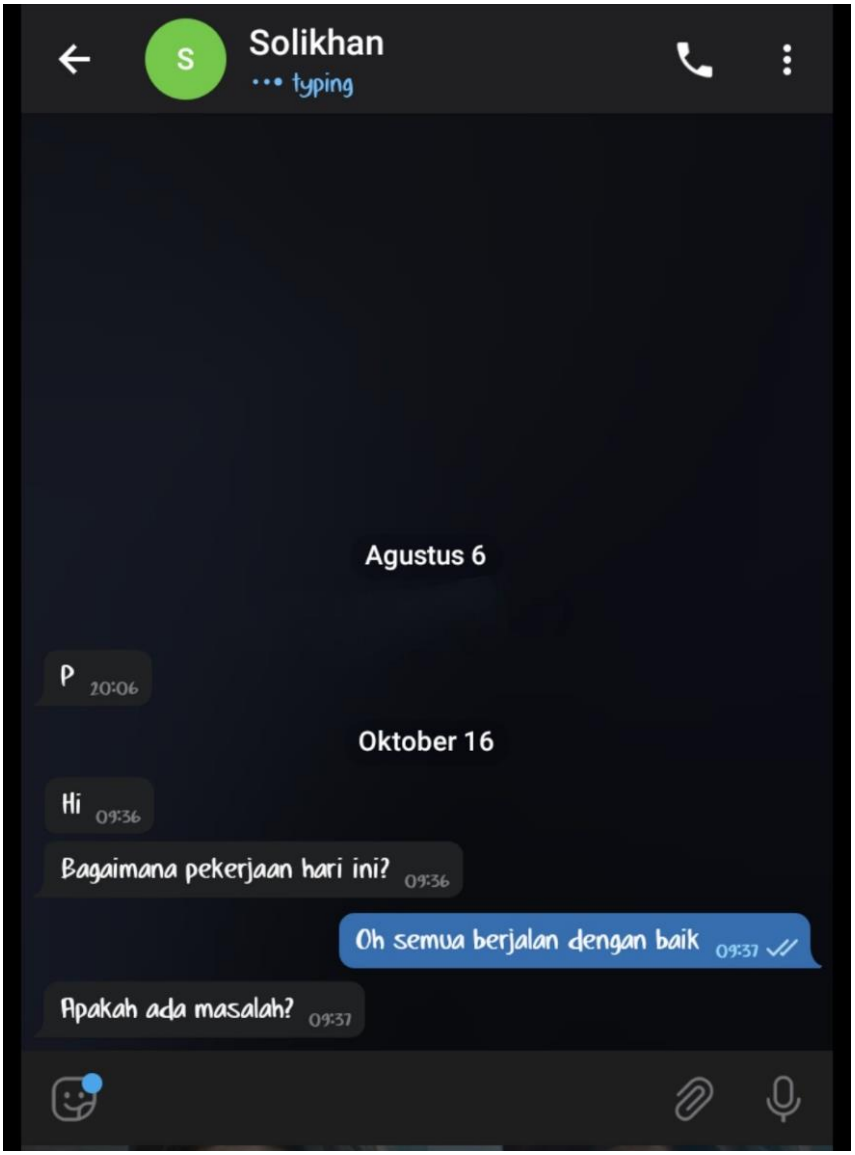
Apakah Anda ingat bagaimana Anda bisa melakukan hampir semua hal ketika Anda masih kecil? Anda bisa menyanyi, menggambar, menari... Anda bisa membayangkan diri Anda menjadi seorang seniman, kosmonot, aktor... Hanya pada titik tertentu dalam hidup Anda, seseorang mengatakan kepada Anda, Hei, berhenti bernyanyi, itu mengerikan! Atau mungkin sesuatu seperti, Mengapa Anda menggambar jika Anda tidak tahu bagaimana melakukannya dengan benar? Atau mungkin sesuatu seperti, Anda harus menjadi dokter, meninggalkan karir akting bagi mereka yang memiliki bakat untuk itu.

Sepanjang hidup kita, kita mengembangkan kemampuan luar biasa untuk tidak percaya pada diri kita sendiri. Buku ini bukan tentang memberi Anda dorongan motivasi atau menjelaskan betapa singkatnya hidup Anda dan bagaimana tidak ada gunanya menghabiskan waktu dengan tidak memercayai diri sendiri. Ada banyak sumber daya web dan pembicaraan TED tentang topik ini. Kami hanya ingin mengingatkan Anda bahwa Anda kreatif. Jika Anda tidak mempercayai kami, periksa buku berjudul Creative Confidence (https://www.creativeconfidence.com/). Buku ini ditulis oleh pencipta d.school (https://dschool.stanford.edu/) menjelaskan mengapa Anda kreatif dan bagaimana percaya diri dalam kreativitas Anda

4.1 Perjalanan Desain—Awal dan Akhirnya

Jadi, persyaratan bisnis Anda ditulis, peta jalan Anda ditentukan, kiriman ditentukan, dan peran dihapus. Sekarang, akhirnya, saatnya untuk beberapa desain, bukan? Tidak, itu tidak benar. Waktu untuk mendesain dimulai pada saat ide muncul di kepala Anda… atau bahkan sebelumnya. Pikirkan tentang masalah yang Anda coba selesaikan. Tanyakan kepada banyak orang apakah mereka memiliki masalah yang sama dan bagaimana mereka ingin menyelesaikannya. Berbicara dengan orang, mengidentifikasi masalah potensial mereka, dan memikirkan solusi yang mungkin adalah bagian intrinsik dari proses desain. Perjalanan desain Anda dimulai pada saat mengidentifikasi masalah dan menjadi intens setelah Anda memvisualisasikan solusinya. Perjalanan ini adalah perjalanan tanpa tujuan, karena idealnya tidak pernah berakhir! Setelah penerapan yang sebenarnya, Anda ingin membuat pengguna tetap terlibat, Anda ingin menerima umpan balik terus-menerus dari mereka, dan Anda ingin mereka menyukai produk Anda. Untuk itu Anda harus terus-menerus memberikan cinta ini kembali kepada mereka.

Pikirkan tentang aplikasi perpesanan Anda. Pernahkah Anda mendapati diri Anda menatapnya? Jawaban langsung Anda adalah "tidak", tetapi saya akan meyakinkan Anda bahwa Anda telah menatap aplikasi perpesanan Anda cukup sering. Tiga titik: apakah itu berarti bagi Anda? Tidak juga? Kemudian lihat Gambar 4-1.



Gambar 4-1. Umpan Balik Tiga Titik Pesan Ketika Seseorang Mengetik Pesan Kepada Anda

Apakah Anda ingat titik-titik ini? Beberapa aplikasi akan secara eksplisit memberi tahu Anda bahwa responden Anda sedang mengetik (Gambar 4-2):

Tolong, jangan bilang Anda tidak menatap aplikasi perpesanan Anda sementara tiga titik ini secara ajaib berkedip di wajah Anda. Lihat betapa pintarnya itu! Aplikasi perpesanan sebenarnya akan berfungsi sempurna tanpa detail kecil ini, tetapi bagian kecil dari desain yang sangat sederhana ini membuat kami, penggunanya, menghabiskan lebih banyak waktu untuk melihatnya daripada jika umpan balik ini tidak ada. Anda dapat menemukan hal-hal kecil ini di mana-mana, di hampir semua hal. Hal-hal semacam ini tidak mungkin dipikirkan sejak awal ketika Anda hanya memiliki sedikit gambaran tentang produk Anda. Hal-hal ini dihasilkan dari umpan balik konstan yang Anda dapatkan dari pengguna Anda, belum lagi tren yang berubah. Dunia bergerak dengan kecepatan gila total. Tidak mungkin hal-hal yang menarik kemarin akan tetap seperti itu besok. Jika Anda ingin produk Anda kompatibel dengan dunia dan kebutuhan pengguna, selalu lihat sekeliling dan selalu dengarkan pengguna Anda.

Gambar 4-2. Umpan balik tiga titik Telegram ketika responden Anda mengetik pesan

4.2 Persona dan Kisah Pengguna

Pada bagian sebelumnya kita telah membahas cukup banyak tentang pengguna dan kebutuhannya. Pengguna. Bisakah Anda membangun ikatan khusus, koneksi dengan sesuatu yang disebut "pengguna?" Ketika saya memikirkan istilah abstrak "pengguna", hal pertama yang muncul di benak saya adalah robot Sophia. Saya tidak tahu mengapa. Untuk membuat keterikatan khusus dengan pengguna, desainer datang dengan konsep "persona", Persona adalah pengguna yang dipersonifikasikan. Anda harus bisa membayangkan orang, atau sekelompok orang, yang akan menggunakan produk Anda.

Produk Anda akan selalu menyelesaikan beberapa masalah, tetapi masalah ini tidak umum bagi semua orang di dunia. Ini akan memecahkan beberapa masalah khusus untuk beberapa kelompok orang tertentu. Misalnya, jika Anda membuat produk yang membantu wanita hamil untuk melacak berat badan mereka, maka Anda dapat segera mempersempit basis pengguna Anda untuk wanita dalam kondisi yang sangat khusus yang sedang hamil. Anda juga harus mengingat cara persona Anda menggunakan perangkat tempat produk perangkat lunak Anda seharusnya dijalankan. Misalnya, orang-orang dari rentang usia yang berbeda menggunakan teknologi dengan cara yang berbeda. Hal yang sama berlaku untuk profesi yang berbeda.

Kita semua memiliki cerita kita sendiri; pengguna kami tidak terkecuali. Apa itu cerita pengguna? Cerita pengguna adalah cerita yang membantu desainer untuk mengidentifikasi persona mereka dan menciptakan ikatan khusus dengan mereka. Anda hanya menutup mata dan membuat cerita yang melibatkan bagaimana produk Anda digunakan. Misalnya, bayangkan seorang pengusaha menggunakan platform pembelajaran kita.

Alex, seorang pengusaha sukses berusia 47 tahun, pulang dari konferensi di mana ia ditanyai beberapa pertanyaan sulit tentang blockchain. Dia membuka laptopnya, membuka situs web platform pembelajaran favoritnya, dan mengetik "blockchain" di kotak pencarian. Dua kursus muncul sebagai hasilnya: salah satunya adalah kursus dalam 2 bulan dan yang lain adalah kursus pengantar yang memakan waktu 3 jam. Karena Mr. Alex berjanji kepada hadirin bahwa dia akan memberikan jawaban keesokan harinya, dia mendaftar di kursus kedua. Dia berpikir untuk mendalami subjek setelah konferensi, jadi dia menambahkan kursus pertama ke daftar "ToStudy" -nya. Saya cukup yakin Anda bisa membuat banyak cerita seperti ini. Apa yang kita miliki sebagai hasil dari cerita ini? Kami sebenarnya memiliki banyak masukan mengenai platform pembelajaran kami, bukan begitu? Jadi, platform kami harus memiliki semacam mekanisme pencarian yang memungkinkan pengguna untuk mencari kursus. Daftar kursus yang dihasilkan harus menampilkan panjang kursus untuk masing-masingnya. Juga harus ada kemungkinan untuk memindahkan kursus ke daftar "ToStudy". Bisakah Anda melihat berapa banyak wawasan yang bisa Anda dapatkan hanya dari satu cerita pengguna?

Kami juga mendapatkan Alex kami, seorang pria serius, yang menggunakan laptop dan telepon dalam kehidupan sehari-harinya. Dia aktif, dan dia ingin belajar tentang teknologi blockchain. Tabel 4-1 memberikan karakteristiknya.

Tabel 4-1. Persona Alex

Nama	Alex
Usia	37 Tahun
Pendidikan	M.Kom
Pekerjaan/Jabatan	Bisnis di Industri Keuangan
Pendapatan	Rp. 120 juta/Tahun
Gaya Hidup	Konferensi, Perjalanan
Personalitas	Kejujuran, Terbuka, Kreatif
Penggunaan Pola	Profesional penggunaan PC
Etika dan penggunaan Teknologi	Big-Cloud-Based Computer system, Online magazine, Online education.

Tidakkah menyenangkan mengenal pengguna Anda, meskipun Anda baru saja membuatnya? Apakah Anda memiliki teman imajiner ketika Anda masih kecil? Menjadi kreatif adalah menjadi seorang anak—hilangkan semua batasan dan biarkan imajinasi Anda mengalir. Semakin banyak cerita yang Anda ceritakan, semakin banyak pengguna berbeda yang akan Anda temukan dan semakin banyak pola penggunaan yang akan Anda identifikasi. Memikirkan pengguna Anda dan kisah mereka akan membantu Anda membangun

keterikatan khusus dengan mereka. Membuat cerita terasa menyenangkan dan merupakan praktik curah pendapat umum di dunia pembuatan perangkat lunak, tetapi tentu saja itu tidak cukup. Anda juga harus keluar dan bertemu pengguna Anda, tetapi itu adalah cerita lain. Untuk saat ini, perlu diingat bahwa ketika kita berbicara tentang pengguna, kita tidak berbicara tentang beberapa entitas abstrak, kita berbicara tentang persona kita, dan kita sudah memiliki keterikatan emosional khusus dengan mereka.

4.3 Jenis Desain

Karena buku kami dikhususkan untuk pengembangan perangkat lunak—khususnya untuk pengembangan perangkat lunak aplikasi web—kami akan berfokus pada desain untuk web. Berbicara tentang desain untuk web cukup sederhana beberapa tahun yang lalu. Setiap desain untuk web dapat disebut desain web, dan setiap orang yang melakukan beberapa desain untuk aplikasi web dapat disebut desainer web. Selalu ada perbedaan yang jelas antara web dan desain grafis atau antara desain cetak dan digital, tetapi desain untuk web disebut desain web. Namun, segala sesuatunya berkembang, web berubah, begitu pula desain untuk web. Itu menjadi lebih kompleks; bukan hanya tentang bentuk, warna, tipografi, dan positioning. Ini tentang pengguna dan tentang pengalaman pengguna. Ini tentang kecepatan memberikan informasi yang mereka butuhkan kepada pengguna. Ini tentang jumlah data yang mengelilingi kita.

Kita hidup di era data besar dan seni mengarahkan pengguna Anda ke arah yang benar adalah tantangan desain besar saat ini. Istilah sederhana "desain web" telah dipecah menjadi kumpulan istilah dan konsep tag cloud yang besar. Anda mungkin pernah mendengar beberapa dari mereka: desain UX, desain UI, desain interaksi (IxD), desain animasi, desain arsitektur informasi, dll. Ini bukan hanya beberapa kata mewah untuk memperkenalkan lebih banyak kompleksitas ke bidang yang sudah cukup kompleks, ini adalah muncul tren didikte oleh dunia kita hidup di hari ini. Menyempurnakan dan memisahkan bidang-bidang ini juga membantu desainer dan insinyur untuk berspesialisasi dalam bidang tertentu dan menguasainya.

Ada begitu banyak informasi yang dapat berisi layanan atau produk Anda. Tentu saja, Anda ingin terlihat jelas bagi pengguna Anda. Anda ingin pengguna Anda menemukan apa pun yang mereka butuhkan dan menemukannya dengan cara yang sederhana dan cepat. Untuk itu, Anda perlu menyusun informasi Anda dengan cara sebaik mungkin. Jadi di sini kita berbicara tentang desain arsitektur informasi. Bergantung pada jenis aplikasi yang Anda buat, akan ada status dan transisi yang berbeda di antara mereka. Misalnya, ketika pengguna login, ada transisi antara halaman login dan halaman aplikasi yang sebenarnya setelah pengguna menekan tombol login. Saat Anda mencari sesuatu di halaman, Anda memerlukan transisi antara status saat Anda menekan tombol Cari dan status tempat hasil muncul di halaman.

Dalam kasus ini, Anda dapat menggunakan sedikit desain animasi untuk menghibur pengguna Anda saat status aplikasi berubah. Cara pengguna merasa tentang aplikasi Anda ditentukan oleh desain pengalaman pengguna, dan cara pengguna berinteraksi dengan aplikasi Anda ditentukan oleh desain interaksi. Cara antarmuka dibangun ditentukan oleh desain antarmuka pengguna. Tentu saja, semua area ini bersinggungan, tumpang tindih,

bercampur, dan menggabungkan semuanya dengan menawarkan cita rasa unik dari perjalanan pengguna yang mulus dan menyenangkan. Anda jarang akan menemukan tim yang memiliki orang-orang dalam semua peran yang baru saja kami jelaskan.

Anda akan menemukan orang-orang yang baik di beberapa bidang. Terkadang satu orang melakukan semua pekerjaan. Misalnya, di Feedzai, perusahaan tempat kami berdua bekerja, ada orang brilian yang akan mendorong seluruh proses desain mulai dari mockup yang digambar tangan hingga implementasi aktual dalam HTML dan CSS. Selama tahap desain ulang utama, perusahaan eksternal yang berspesialisasi dalam desain UX dipekerjakan untuk menjalankan penelitian pengguna dan membantu menentukan antarmuka yang sempurna untuk perjalanan pengguna.

Perusahaan lain yang bekerja sama dengan Solikhan telah mendelegasikan seluruh desain ke agen eksternal. Apakah Anda ingin pendapat jujur kami tentang hal itu? Jangan lakukan itu kecuali anggaran Anda benar-benar terbatas. Desain menjadi benar-benar terlepas dari bisnis, misi, dan visi perusahaan, dan sangat sulit untuk mendorong perubahan. Jika memungkinkan, cobalah untuk memiliki desainer Anda sendiri—seseorang yang menghayati produk Anda dengan cara yang sama seperti Anda.

Di OptioPay kami memiliki departemen kreatif yang dipimpin oleh manajer produk. Departemen ini terdiri dari seorang direktur seni—seorang gadis luar biasa yang dapat dengan mudah membuat grafik yang sederhana dan tampak hebat serta merancang seluruh sistem yang kompleks. Ada juga seorang insinyur UI — seorang pria dengan pikiran desainer-insinyur campuran. Dia mampu mendorong seluruh proses desain, dan kekuatan sebenarnya terletak pada implementasi hal-hal. Dia dapat mengambil desain apa pun dan mengubahnya menjadi antarmuka yang siap produksi, sepenuhnya responsif, dan adaptif. Apa yang tampak seperti sihir hitam total bagi kami sangat jelas baginya. Ada juga UX designer yang menghubungkan user experience dengan kebutuhan bisnis. Dia mendorong penelitian pengguna, menggambar maket, mengulanginya sampai benar-benar jelas bagi pengguna, dan menyerahkannya kepada insinyur UI.

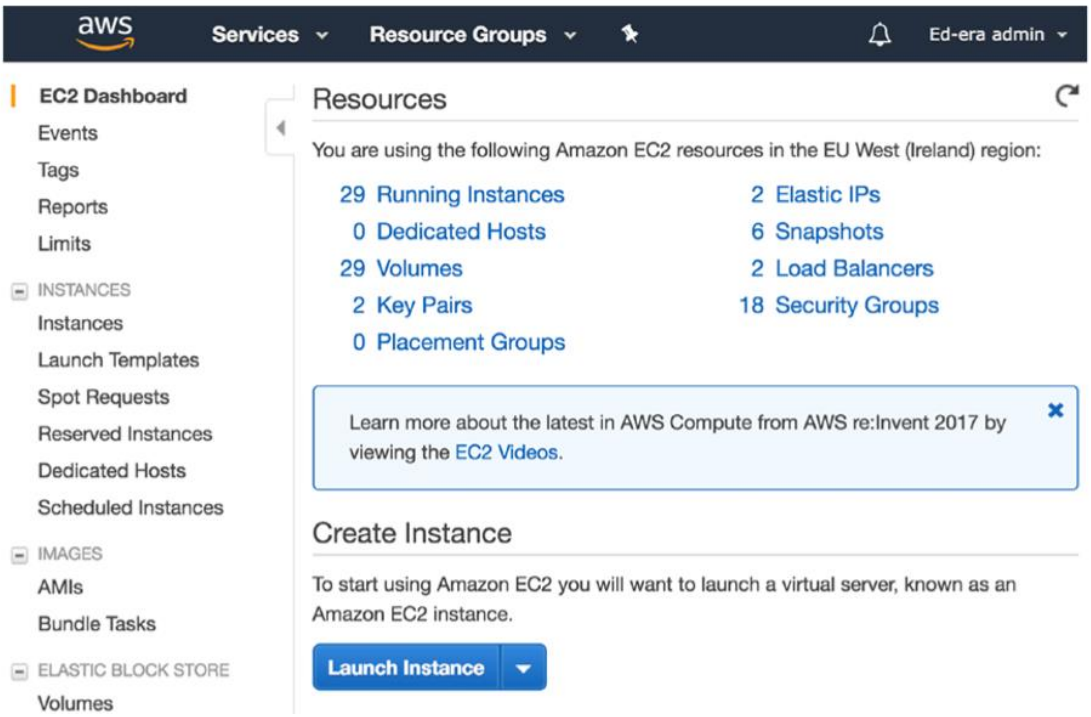
Di perusahaan tempat Lawrence bekerja, hanya ada dua desainer yang melakukan semua pekerjaan—mereka mendorong seluruh proses desain, mulai dari brainstorming bisnis dan wireframe dengan fidelitas rendah hingga mockup fidelitas tinggi dan serah terima ke pengembang frontend. Kebutuhan yang berbeda memerlukan pendekatan yang berbeda, tetapi satu hal yang pasti benar—jika produk Anda memiliki antarmuka pengguna, produk Anda secara otomatis menjadi berorientasi pengguna; oleh karena itu, jangan pernah meremehkan kekuatan desain. Pengguna Anda harus berada di tengah seluruh desain dan konsep pengalaman.

4.4 Antarmuka Pengguna dan Pengalaman Pengguna

Di bagian sebelumnya kita telah membahas berbagai jenis desain dan perbedaan utama di antara mereka. Sumber utama perdebatan, pembicaraan kontroversial, dan artikel blog adalah perbedaan antara antarmuka pengguna (UI) dan pengalaman pengguna (UX). Kedua bidang ini memiliki batas paling kabur di antara keduanya. Pengalaman pengguna selalu membutuhkan antarmuka pengguna. Sebuah antarmuka bisa hampir segalanya dan

belum tentu sesuatu yang terlihat. Misalnya, Alexa tidak memiliki antarmuka pengguna yang terlihat, tetapi memiliki pengalaman pengguna yang luar biasa!

Terkadang Anda dapat menemukan produk yang memiliki antarmuka yang buruk namun pengalaman pengguna yang sangat baik. Misalnya, konsol administrasi Amazon Web Services (AWS) tidak memiliki UI yang menakjubkan, tetapi itu tidak masalah karena memiliki pengalaman pengguna yang luar biasa. Setiap kali kami membuka dasbor AWS untuk memantau beberapa masalah atau menambahkan beberapa layanan, kami tahu persis di mana harus mengklik, di mana harus mencari, dan apa yang harus diketik untuk menemukan informasi yang dibutuhkan, seperti yang ditunjukkan pada Gambar 4-3.

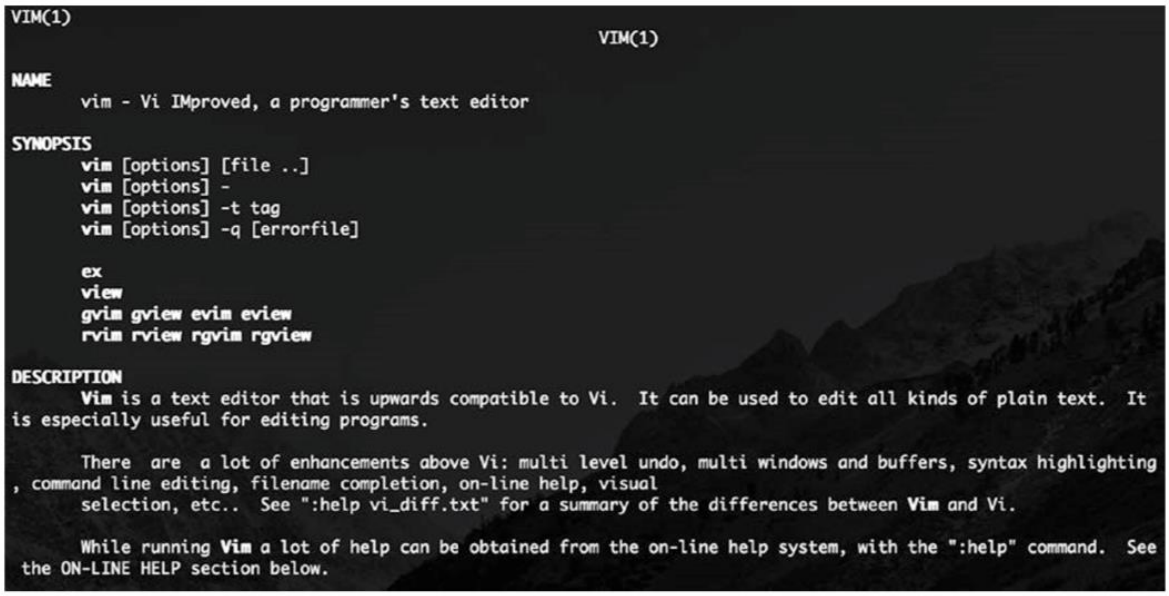


Gambar 4-3. Konsol AWS: Hanya UI Biasa, Namun Merupakan UX Yang Bagus Dan Kuat

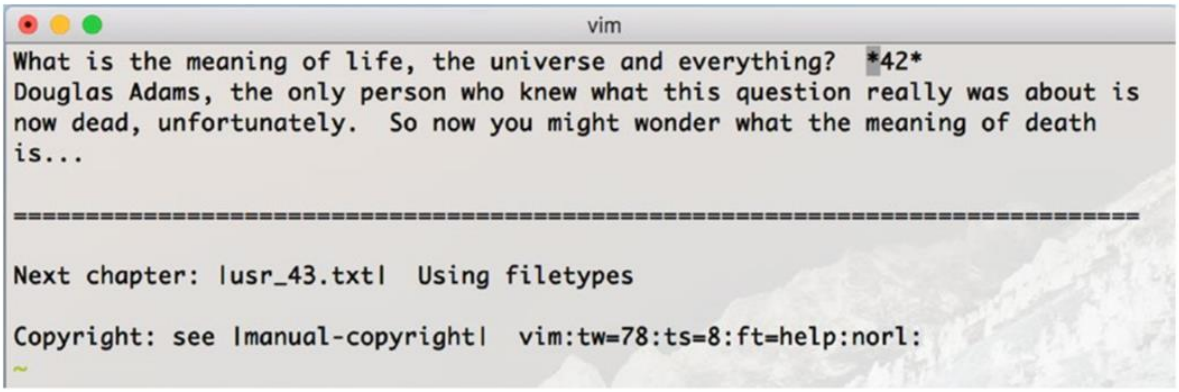
Contoh lain dari UI yang jelek tapi UX yang bagus adalah konsol Linux. Konsol memungkinkan Anda untuk mengoperasikan sistem Anda melalui perintah yang Anda ketik di antarmuka baris perintah. Tidak ada tombol mewah, tidak ada warna yang indah. Hanya teks dan panah. Tapi inilah yang membuat konsol begitu kuat. Mereka tidak memiliki gangguan; mereka langsung ke intinya. Itulah mengapa pengembang dari seluruh dunia menyukai Linux—antarmuka Linux jelek, tetapi ia memiliki konsol yang kuat yang memungkinkan melakukan segalanya tanpa harus meninggalkannya! Mari kita ambil Vim sebagai contoh. Vim adalah editor teks yang dapat Anda jalankan di konsol. Sangat mudah untuk mengetahui lebih lanjut tentang Vim jika Anda menggunakan konsol. Cukup ketik "man vim." Ini akan memberi Anda panduan pengguna lengkap editor teks Vim (Gambar 4-4).

Vim adalah editor teks yang sangat bagus untuk programmer. Antarmukanya terdiri dari perintah teks. Itu tidak memiliki tombol, animasi, gradien, atau transisi mewah apa pun. Namun itu sangat kuat, dan pengembang menyukainya. Ini juga menyenangkan dan memiliki

beberapa telur Paskah. Misalnya, coba ketik ":help 42" di antarmuka Vim. Anda akan melihat sesuatu seperti yang ditunjukkan pada Gambar 4-5.



Gambar 4-4. Unix shell: perintah man vim

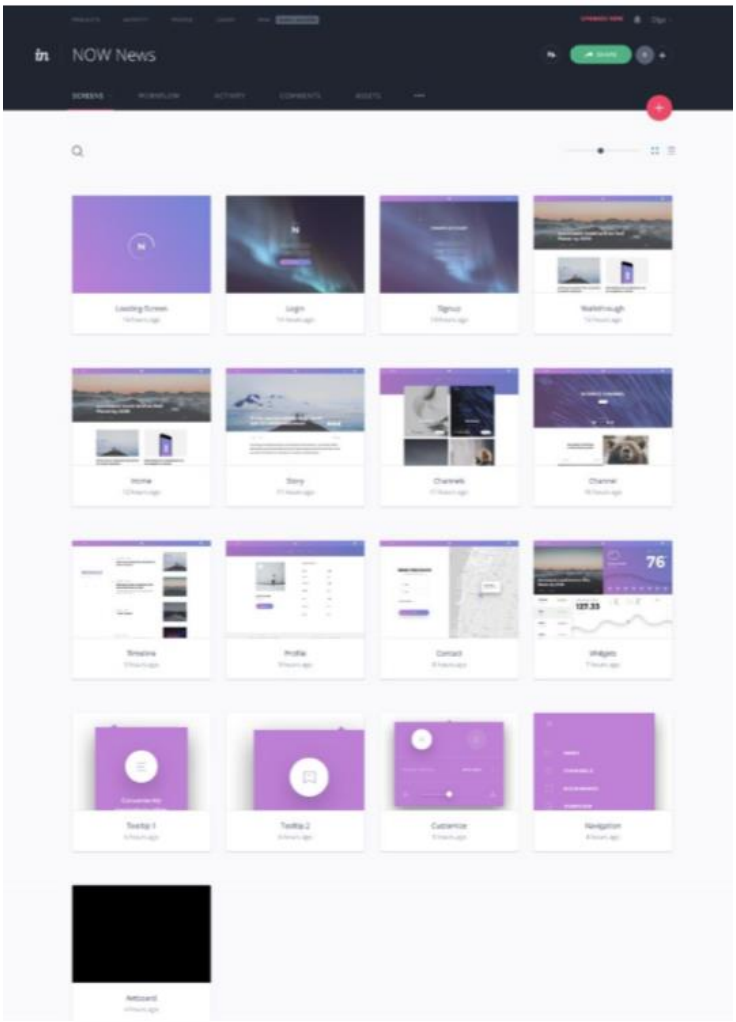


Gambar 4-5. Telur Paskah Vim - :help 42

Ada banyak contoh lain dari situs web dengan UI yang buruk namun UX yang bagus. Misalnya, Hackernews (https://news.ycombinator.com/), Reddit (https://www.reddit.com/), Craigslist (https://craigslist.org/) —ini adalah contoh-contoh yang jelek belum situs web yang sangat populer. Oleh karena itu, Anda dapat melihat bahwa Anda dapat memiliki pengalaman pengguna yang hebat tanpa memiliki antarmuka pengguna yang menakjubkan. Bisakah sebaliknya? Bisakah Anda memiliki antarmuka pengguna yang indah namun pengalaman pengguna yang buruk?

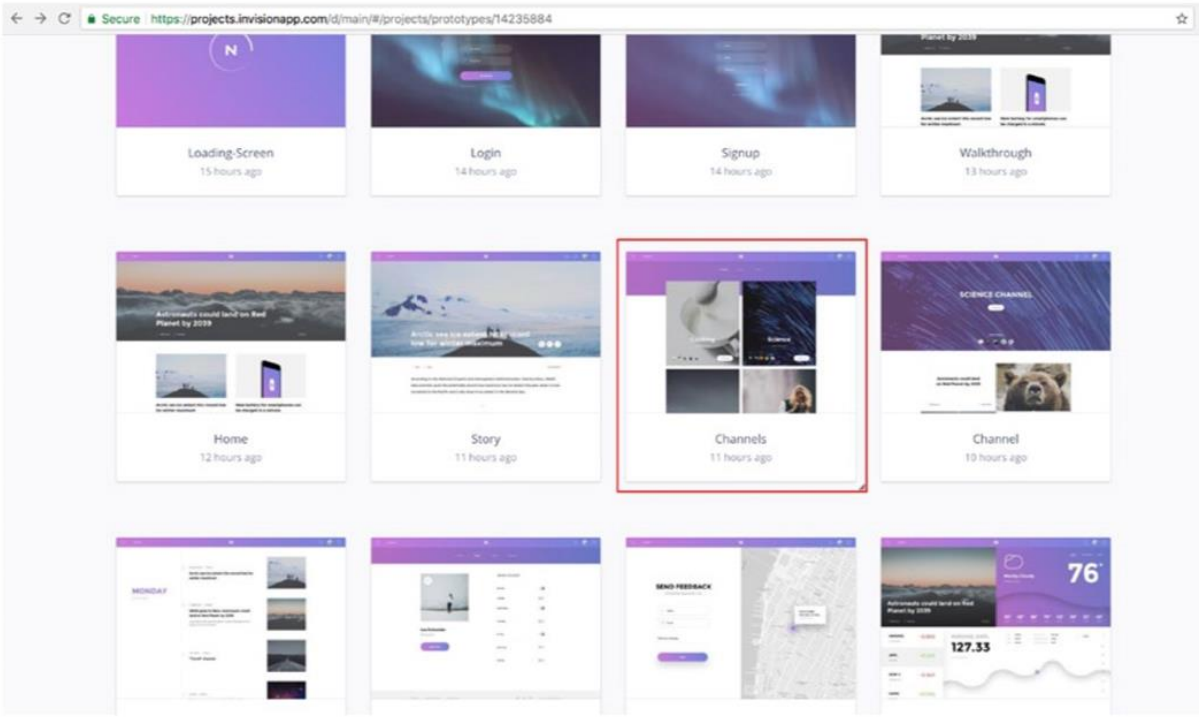
Ambil Windows 10 sebagai contoh. Ini sebenarnya memiliki antarmuka yang cukup bagus, terlihat sangat modern dan mewah, tetapi cobalah, misalnya, untuk mengelola pengaturan nirkabel Anda. Anda akan merasa sedikit rumit dan tidak intuitif sama sekali. Contoh lain dari layanan yang memiliki beberapa masalah UX di UI yang sangat menarik adalah Invision (https://www.invisionapp.com/). Invision adalah produk hebat yang memungkinkan Anda membuat dan menguji prototipe desain Anda. Kami menggunakannya

cukup banyak. Ini memiliki antarmuka yang sangat menarik, dengan transisi yang mulus, warna yang bagus, dan font yang bagus… Sebenarnya, ini juga bekerja dengan cukup bagus. Namun, ia memiliki beberapa masalah UX. Misalnya, bayangkan Anda berada di dasbor prototipe di mana Anda memiliki banyak prototipe; karenanya, Anda perlu menggulir. Misalnya, ada halaman prototipe besar, seperti yang ditunjukkan pada Gambar 4-6.



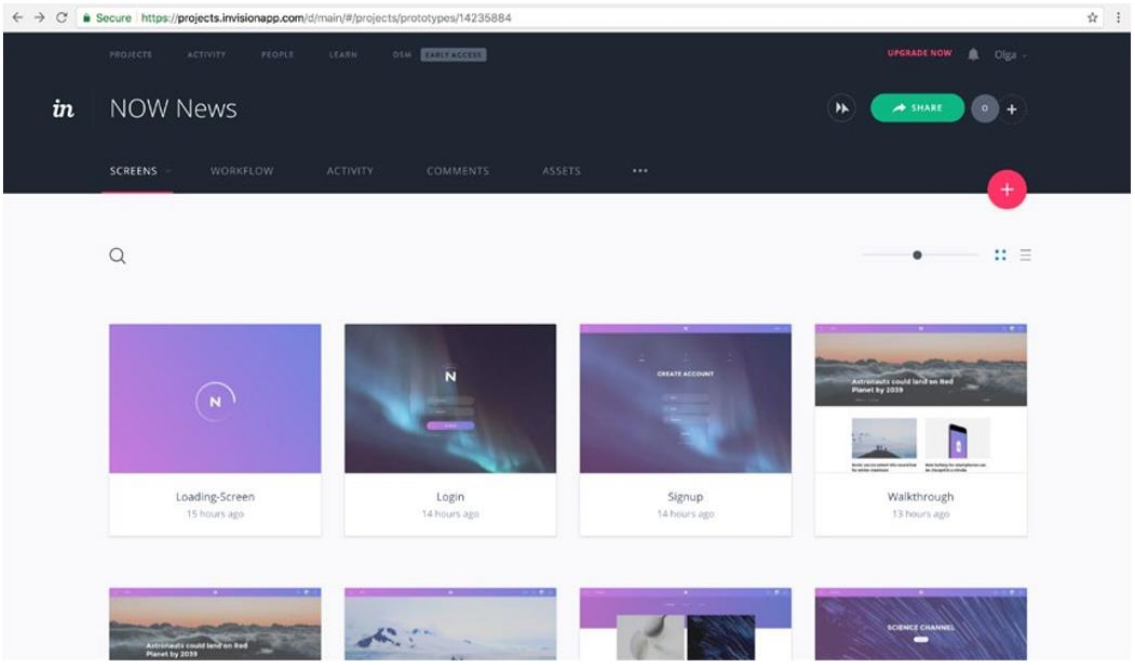
Gambar 4-6. Halaman Prototipe Besar Yang Dapat Digulir Di Antarmuka Invision

Bayangkan Anda menggulir sekitar setengah halaman dan ingin memeriksa beberapa prototipe tertentu. Jadi, Anda mengkliknya (Gambar 4-7).

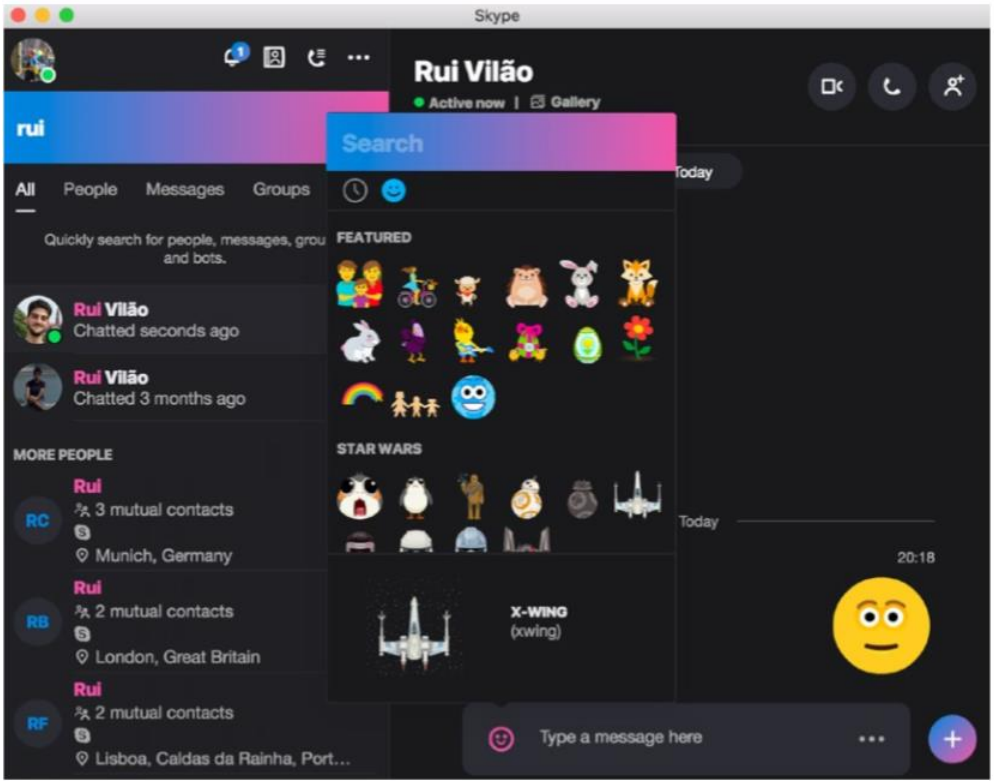


Gambar 4-7. Pilih Salah Satu Prototipe Setelah Menggulir

Prototipe dibuka pada halaman yang sama, Anda melakukan apa yang perlu Anda lakukan di dalamnya dan kemudian Anda mengklik kembali pada panel navigasi browser. Apa yang kamu harapkan? Anda berharap untuk kembali ke tempat yang sama persis sebelum Anda mengklik prototipe—di tengah halaman. Apa yang terjadi? Skenario yang ditunjukkan pada Gambar 4-8 terjadi.



Gambar 4-8. Setelah Anda Mengklik Kembali Halaman Prototipe, Anda Kembali Ke Bagian Atas Dasbor Alih-Alih Tempat Sebenarnya Yang Pernah Anda Kunjungi Sebelumnya.



Gambar 4-9. Skype: Contoh UI Buruk Dan UX Buruk Secara Bersamaan

Ini adalah pengalaman pengguna yang mengerikan. Kemungkinan Anda ingin melihat lebih dekat prototipe di sebelah yang baru saja Anda periksa. Jadi setiap kali Anda kembali, Anda dipaksa untuk menggulir lagi dan mencari tempat yang pernah Anda kunjungi sebelumnya. Anda dapat menemukan banyak produk di luar sana yang memiliki UI hebat dan UX hebat secara bersamaan. Google Drive, misalnya, terlihat bagus, terasa enak, dan berfungsi dengan baik. Twitter adalah contoh lain yang terlihat bagus dan cukup intuitif untuk digunakan. Facebook dan Instagram adalah contoh bagus dari sesuatu yang tidak hanya terlihat bagus dan mudah digunakan, tetapi juga sangat menular. Mereka membuat UX mereka untuk menyerap jutaan orang dan menghabiskan waktu mereka dalam gulungan tak terbatas yang tidak berguna. Itu menakjubkan dan menakutkan pada saat bersamaan. Tentu saja, Anda dapat menemukan contoh UI buruk dan UX buruk di produk yang sama. Misalnya, setelah Microsoft membeli Skype, Skype menjadi jelek dan sulit digunakan (Gambar 4-9).

Jadi, apa yang dilakukan desainer UI dan UX? Begitu mereka mulai melakukan pekerjaan mereka, mereka harus mengajukan beberapa pertanyaan untuk menyelaraskannya dengan kebutuhan pengguna dan produk serta tujuan bisnis. Pertanyaannya berbeda sifatnya. Jadi, desainer UI akan peduli dengan pertanyaan seperti:

- Warna apa yang harus saya gunakan?
- Font apa yang harus saya gunakan?
- Apa identitas perusahaan?

Seperti yang Anda lihat, perancang antarmuka pengguna khawatir tentang hal-hal yang dapat Anda lihat. Sedangkan desainer pengalaman pengguna akan mengajukan pertanyaan seperti:

- Siapa pengguna kami?
- Apa yang seharusnya dicapai pengguna saat menggunakan produk?
- Apa tujuan bisnis dari produk ini?

Dengan demikian, perancang pengalaman pengguna khawatir tentang hal-hal yang dapat Anda rasakan. Desainer UX lebih berorientasi bisnis. Baik desainer UI dan UX mengkhawatirkan pengguna. Yang pertama prihatin tentang apa yang dilihat pengguna, dan yang kedua khawatir tentang bagaimana perasaan pengguna. Bekerja sama, desainer UI dan UX dapat menghadirkan produk hebat dan membuat kami, pengguna, bahagia.

4.5 Proses Desain—Bagaimana Desainer Menjalankannya

Bagaimana proses desain terstruktur? Apakah Anda hanya pergi ke desainer Anda dan memberi tahu mereka, "Hei, saya ingin membangun beberapa toko online!" dan kemudian desainer hanya menggambarnya? Nah, alangkah baiknya jika semuanya bisa semudah itu. Bahkan, proses desainnya cukup melelahkan. Kami meminta seorang teman kami, seorang desainer, untuk menjelaskan bagaimana dia menjalankan prosesnya. Ini dia biodata singkatnya:

nama: Yujia Ma Deskripsi: Seorang desainer UX/UI yang berbasis di Berlin. Suka memecahkan masalah desain, membuat antarmuka pengguna yang cerdas, dan membayangkan interaksi yang bermanfaat, dengan fokus pada desain yang berpusat pada pengguna. Linkedin: https://www.linkedin.com/in/mayujia/

Kami memanggilnya Malu, singkatnya. Inilah cerita Malu tentang proses desainnya.

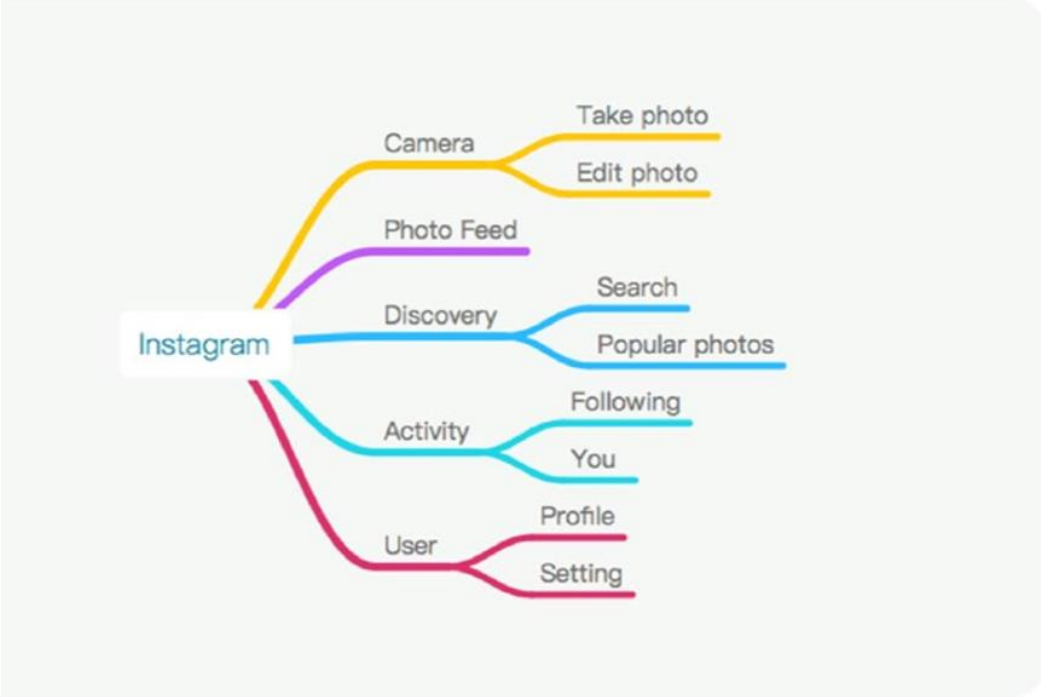
1. ***Cerita Persona dan Pengguna***

Malu biasanya dimulai dengan persona dan cerita pengguna. Dia merekomendasikan untuk membuat hingga tiga persona dan menulis cerita pengguna yang paling penting di atas kertas.

2. ***Arsitektur informasi***

Setelah menuliskan cerita pengguna dan membuat persona, Malu melanjutkan dengan arsitektur informasi. Dia mendefinisikan arsitektur informasi sebagai struktur dan urutan produk. Dia merekomendasikan untuk memikirkan tugas apa yang dapat diselesaikan pengguna dalam produk Anda, dan aliran pengguna di seluruh produk Anda.

“Arsitektur informasi yang baik memastikan bahwa desain Anda dikelompokkan secara logis dan saling terkait.” Lihat Gambar 4-10.



Gambar 4-10. Contoh Arsitektur Informasi Aplikasi Instagram Oleh Malu

3. Membuat sketsa

Setelah memikirkan arsitektur informasi, Malu menggambar beberapa bentuk di atas kertas (seperti yang ditunjukkan pada Gambar 4-11) untuk mendapatkan gambaran tentang tampilan produk.

Gambar 4-11. Sketsa Yang Digambar Tangan Oleh Malu

Begitu dia merasa bahwa sketsanya memenuhi dan mewakili cerita pengguna penuh, dia melanjutkan ke pengujian. Malu menggunakan aplikasi POP (https://marvelapp.com/pop), yang memungkinkannya menguji tiruan kertas apa pun di berbagai simulator perangkat.

4. Gambar rangka

"Seperti membangun rumah, Anda sudah memiliki kerangka aplikasi Anda (arsitektur informasi) dan Anda juga mencoba beberapa sketsa di atas kertas. Sekarang Anda perlu menggambar tanaman lantai untuk memastikan bahwa ruang tamu dan dapur berada di tempat yang benar. Untuk menetapkan tujuan setiap halaman, interaksi yang mereka miliki, dll. Gambar rangka memberi Anda gambaran tentang penempatan elemen pada halaman dan hubungannya satu sama lain, tetapi tanpa warna, grafik, spasi, atau tipografi. Jangan lupa untuk membuat catatan untuk menjelaskan interaksi selain layar Anda."
Jika Anda mendesain aplikasi iOS atau Android, Malu merekomendasikan untuk mempelajari lebih lanjut tentang pedoman antarmuka pengguna grafis (GUI) terlebih dahulu.

- iOs: https://developer.apple.com/ios/human-interface-guidelines/
- Android: https://material.google.com/

Malu sangat menyarankan untuk mempertimbangkan topik berikut saat merancang gambar rangka:

- Kebutuhan apa yang paling penting?
- Konten dan fungsi apa yang harus ada di halaman?
- Apakah elemennya masuk akal?
- Apakah ada hal penting yang hilang pada halaman tersebut?
- Apakah ada yang membuat pengguna bingung?
- Bagaimana kita bisa membangun hubungan antar halaman?

Anda dapat mendesain gambar rangka dengan fidelitas rendah menggunakan area abu-abu untuk menggantikan konten sebenarnya, seperti yang ditunjukkan pada Gambar 4-12.

Gambar 4-12. Gambar Rangka Dengan Fidelitas Rendah Oleh Malu

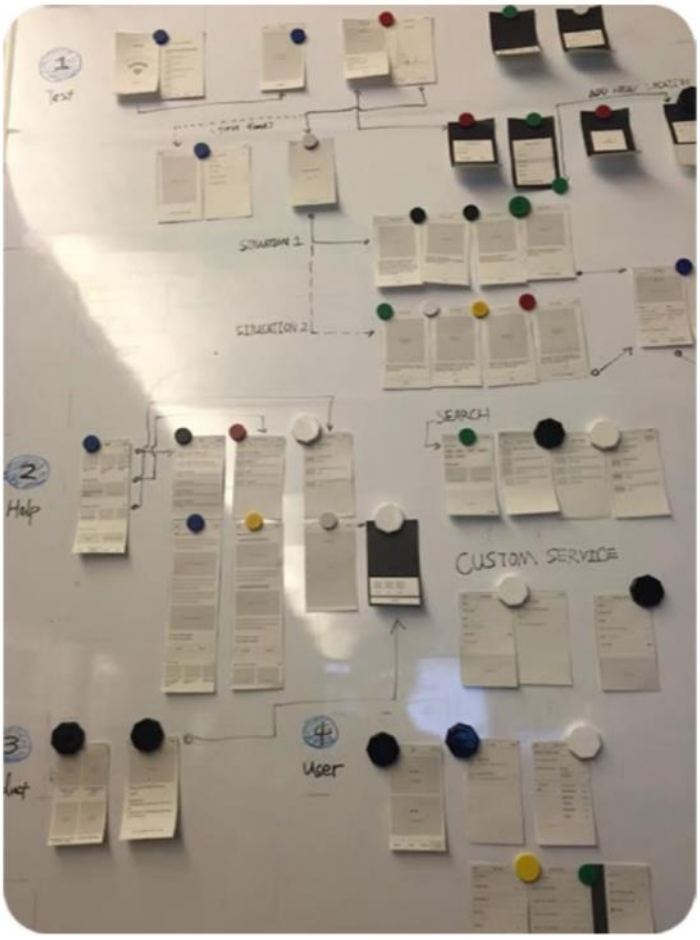
Anda juga dapat mendesain gambar rangka dengan fidelitas yang lebih tinggi. Ini lebih dekat dengan desain visual dengan konten nyata. Desainer memiliki banyak cara untuk melakukan wireframe. Malu mendesainnya menggunakan Sketch. Dia memiliki kit gambar rangka sendiri untuk Android, iOS, dan web untuk membuat prosesnya cepat dan efisien. Anda juga dapat menemukan banyak sumber daya kit gambar rangka gratis. Pada awalnya,

Anda dapat mempelajari berapa banyak elemen dasar yang harus dimiliki sebuah aplikasi. Anda juga dapat menggunakan alat seperti Omnigraffle (https://www.omnigroup.com/omnigraffle), Axure (http://www.axure.com/), dan Balsamiq (https://balsamiq.com/) untuk membuat gambar rangka.

5. Prototipe

Setelah wireframe, kita perlu menguji apakah desain kita berhasil. Proses Malu adalah mengunggah file sketsanya ke Invision dan membuat prototipe yang dapat diklik dengan cepat.

Selama pengujian, Anda perlu meninjau apakah aliran produk lancar dan konsisten. Terkadang Malu lebih suka mencetak semua layar untuk membuat papan alur kerja yang besar. Saat Anda menghubungkan semua layar dengan pena, sangat mudah untuk menemukan apa yang hilang dan di mana ada kesalahan logis. Malu juga merekomendasikan untuk berbicara dengan tim pengembang Anda saat ini. Hasil brainstorming tim ditunjukkan pada Gambar 4-13.

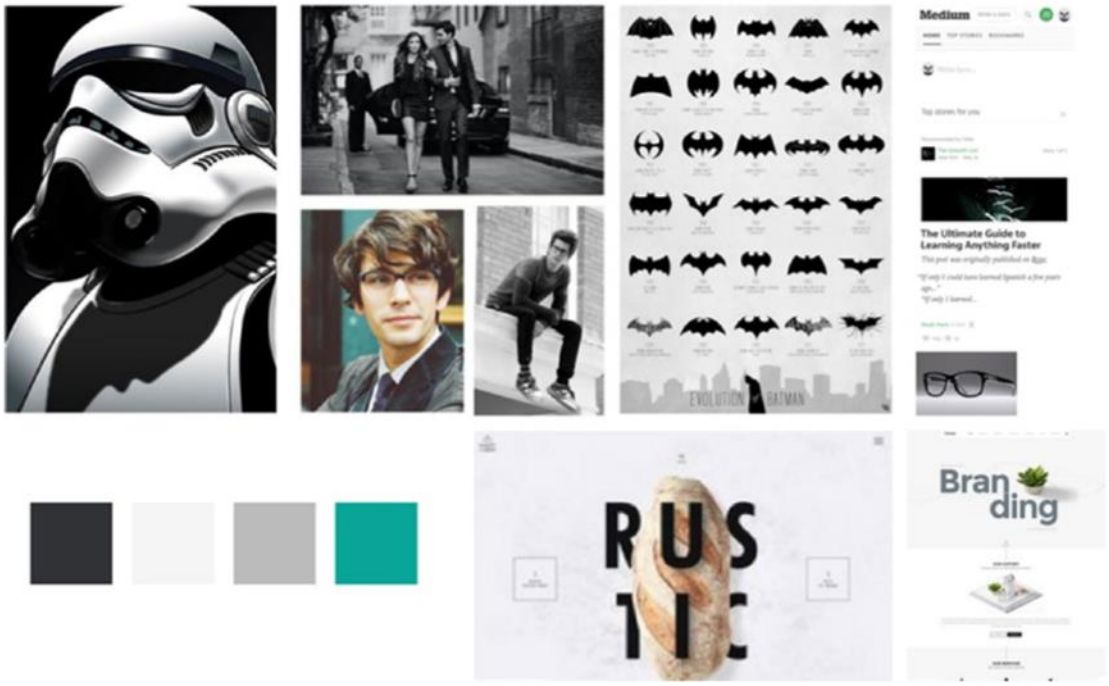


Gambar 4.13 Prototipe Kertas

6. Desain visual

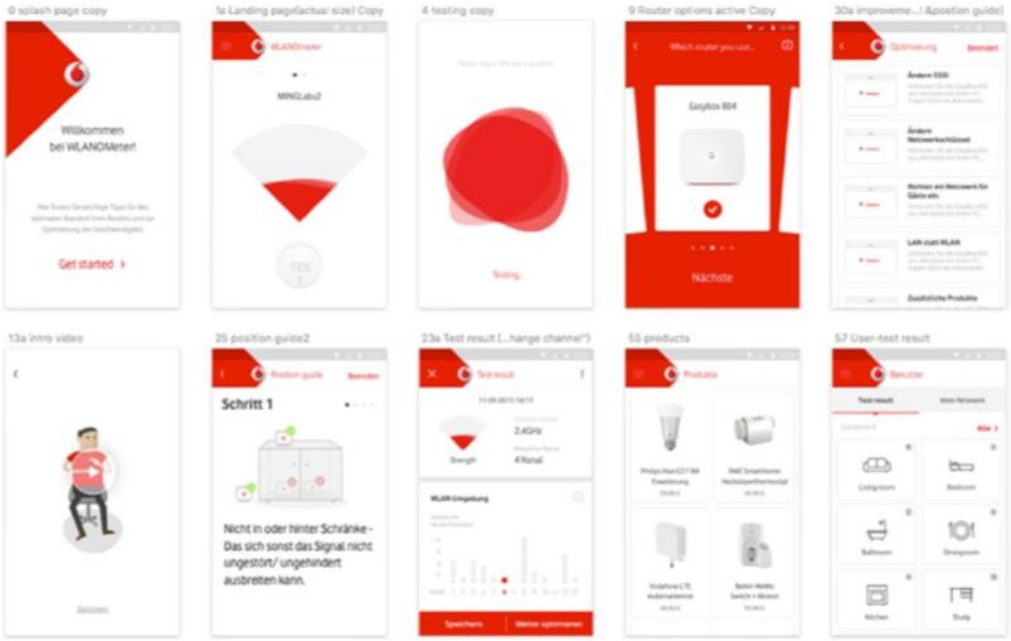
"Wireframe disetujui. Ya! Kita bisa mulai dengan desain visual! Tunggu sebentar sebelum membuat desain visual. Saya akan menyarankan Anda untuk mendapatkan lebih banyak inspirasi. Sangat membantu untuk memahami gaya produk Anda. Buka Pinterest

(mis., seperti yang ditunjukkan pada Gambar 4-14), masukkan beberapa kata kunci untuk membuat moodboard."—Malu



Gambar 4-14. Pinterest Moodboard Oleh Malu

Buka sketsa dan lakukan keajaiban (Gambar 4-15):



Gambar 4-15. Desain visual di Sketch oleh Malu

7. Prototipe kesetiaan yang lebih tinggi

Pada titik ini Anda mungkin sudah lapar untuk mulai menerapkan produk Anda. Sebelum melakukan itu, Malu menguji produknya lagi. Dia menggunakan Invision untuk membuat prototipe fidelitas yang lebih tinggi. Dia menemukan itu sebagai cara yang baik untuk menunjukkan desain kepada orang lain untuk brainstorming.

"Tidak cukup? Saya tahu, semua orang menyukai desain gerak dan animasi… Oke, Anda dapat menggunakan Principle, Justinmind, Pixate, After effect, atau bahkan hanya Keynote (logo aplikasi ini ditampilkan pada Gambar 4-16) untuk membuat beberapa animasi mewah untuk menunjukkan interaksi yang ingin Anda lakukan."—Malu

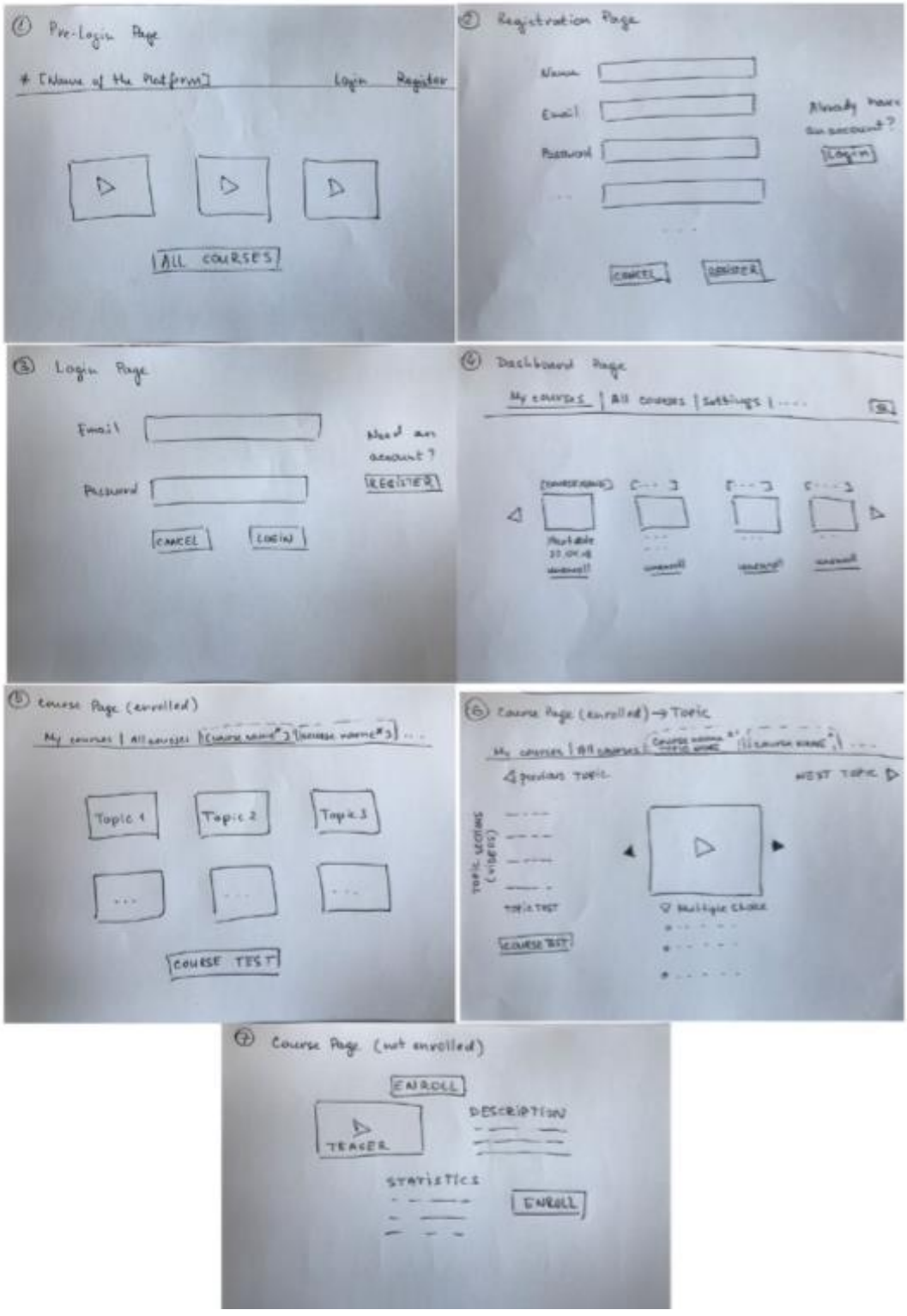
Gambar 4-16. Alat Yang Dapat Digunakan Untuk Merancang Prototipe Dan Animasi Fidelitas Tinggi

8. Serahkan

Setelah semua pengujian dan brainstorming, Malu mengatur file sketsanya serta file desain lainnya dan menyerahkannya kepada tim pengembang. Ayo kode!

4.6 Merancang Platform Pendidikan Online Kami

Sekarang setelah kita mempelajari bagaimana desainer lain menyusun proses desain mereka, kita dapat berpura-pura menjadi desainer dan mulai membuat sketsa platform pendidikan online kita. Beberapa tahapan dan prinsip proses Malu kami terapkan. Pada bagian ini, kami akan menjelaskan apa yang kami lakukan untuk mempersiapkan segala sesuatunya untuk implementasi yang sebenarnya.



Gambar 4-17. Maket brainstorming awal untuk platform pendidikan online

4.7 Brainstorming Awal

Jadi, kami mulai dari brainstorming. Lawrence, Taufik, dan Solikhan duduk bersama, dan Solikhan menceritakan tentang ekspektasi dari MVP kami. Kami berbicara tentang Persona dan menghasilkan beberapa cerita pengguna selain cerita Mr. Alex kami. Taufik

mengajukan banyak pertanyaan. Dia memiliki kemampuan khusus untuk membombardir Anda dengan semua pertanyaan dari beberapa sudut dan perspektif yang berbeda. Biasanya ketika kami melakukan sesi brainstorming dengannya, kami merasa seperti lemon yang diperas, tetapi sebenarnya terasa enak! Jika Anda akan membangun produk Anda sendiri, pastikan Anda memiliki orang seperti Taufik. Orang ini harus memiliki pikiran yang kritis dan tajam, tidak malu-malu, dan ingin tahu dan bersemangat. Jadi, setelah brainstorming yang intens, kami telah menggambar sketsa ini (Gambar 4-17) di atas kertas.

Gambar 4-18. Gambar Rangka Platform Kursus Online

Selain itu, selama brainstorming, kami menemukan beberapa ide untuk implementasi di masa mendatang, seperti yang ditunjukkan pada Tabel 4-2.

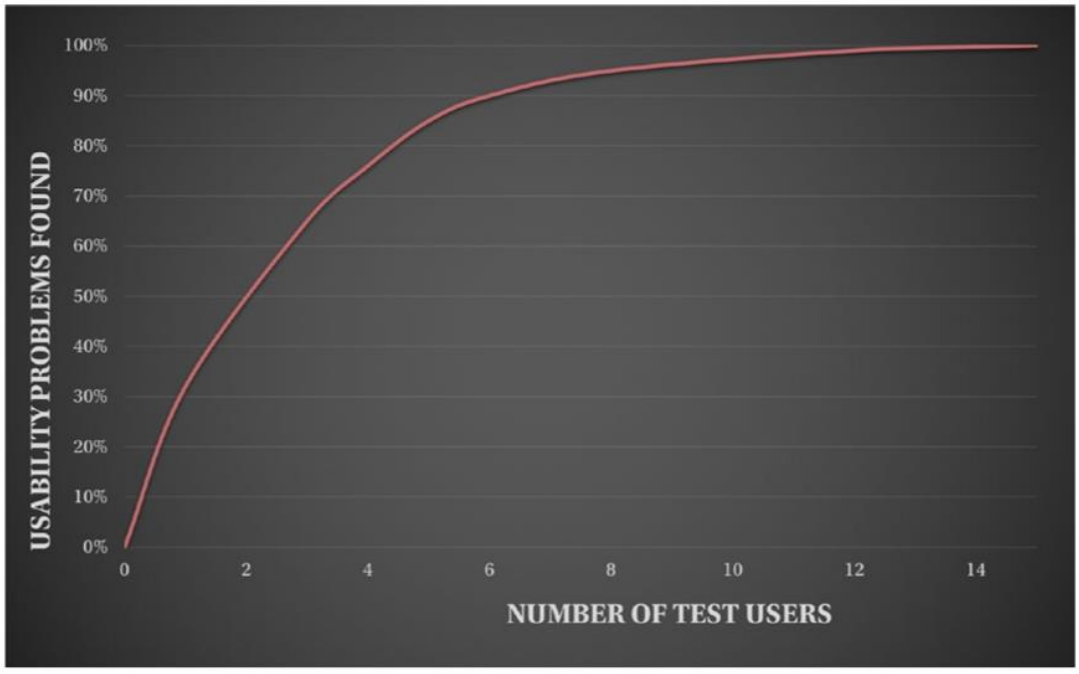
Tabel 4-2. Ide Untuk Implementasi Masa Depan

Kategori Kursus	Untuk navigasi yang mudah, tambahkan kategori ke kursus dan izinkan pencarian dan penyaringan oleh mereka.
Rekomendasi	siswa menerima saran kursus berdasarkan kursus tempat mereka mendaftar sebelumnya, dan pencarian mereka (kami mungkin memerlukan persetujuan khusus untuk itu karena undang-undang privasi data yang berlaku).
Sertifikasi	Harus ada tombol yang dapat diklik oleh siswa untuk membuat sertifikat mereka sendiri (kapan pun mereka merasa siap).

Kami juga menemukan ide menarik tentang navigasi tab—periksa mockup nomor 6. Saat pengguna membuka kursus mereka, kursus dibuka di tab, sehingga memungkinkan kami membuka lebih dari satu kursus pada saat itu. Anda bahkan dapat membuka kursus yang sama lebih dari sekali di tab yang berbeda. Bayangkan, misalnya, Anda masih bisa menonton video kursus di satu tab dan melakukan tes di tab lain.

Gambar rangka

Setelah brainstorming awal dengan Taufik, kami menyerahkan semua hasilnya ke Dani dan dia menggunakan Sketch untuk menggambar wireframe. Gambar 4-18 adalah apa yang keluar darinya. Anda dapat melihat betapa lambatnya produk kami mulai mendapatkan bentuknya. Bukankah itu indah?



Gambar 4-19. Rasio Antara Jumlah Pengguna Uji Dan Masalah Kegunaan Yang Ditemukan

4.8 Pengujian Kegunaan

Apakah Anda ingat bahwa pengguna kami adalah pusat dari desain kami? Kami merancang produk kami untuk pengguna kami, oleh karena itu kami harus mengujinya pada pengguna kami! Biasanya, setelah prototipe pertama siap, mereka harus diuji dengan

pengguna nyata. Bagaimana Anda menjalankan tes pengguna? Anda memberikan prototipe Anda kepada pengguna, memberi tahu mereka apa yang perlu mereka capai, dan melihat bagaimana mereka bisa mencapainya. Anda dapat memfilmkan ekspresi wajah mereka, menuliskan apa yang mereka katakan, dan menemukan semua masalah dan solusi yang mungkin. Bagaimana Anda tahu berapa banyak pengguna yang harus Anda ajak melakukan penelitian? Jawaban logisnya adalah "semakin banyak pengguna semakin baik!" Ternyata cukup untuk memiliki lima tes pengguna! Lima! Mengapa demikian? Grup Nielsen Norman telah menemukan rasio berikut (Gambar 4-19) antara jumlah pengguna dan jumlah masalah kegunaan yang dapat ditemukan dalam produk tertentu.

Seperti yang Anda lihat, mulai dari lima pengguna, jumlah masalah tidak bertambah banyak. Dengan demikian, upaya melakukan pengujian pengguna lebih lanjut sebenarnya tidak sepadan. Hal lain yang sangat penting yang jelas dari bagan ini adalah lebih baik menguji dengan satu pengguna daripada tanpa pengguna sama sekali. Dengan satu pengguna uji, Anda dapat menemukan hingga 31% masalah kegunaan! Ini adalah sepertiga dari masalah produk Anda.

Jadi, kami memutuskan untuk melakukan pengujian pengguna dengan satu pengguna. Kami memberikan maket kepada pengguna dan menjelaskan bahwa kami mengharapkan dia untuk berinteraksi dengan platform pendidikan online.

Pengguna kami adalah seorang pria berusia 26 tahun yang pekerjaannya di bidang IT. Dapat dikatakan bahwa peserta tes adalah pengguna kursus online yang berat. Di satu sisi, itu berarti dia sudah memiliki gagasan bagaimana platform pendidikan seperti itu seharusnya terlihat dan mengetahui kebutuhan dan harapannya. Di sisi lain, itu berarti harapannya sudah bias. Karena penggunaan intensif situs web pendidikan serupa, ia mungkin telah terbiasa dengan antarmuka pengguna yang berbeda dan mencarinya, meskipun itu tidak optimal. Tabel 4-3 menggambarkan hasil yang kami rekam selama pengujian.

Tabel 4-3. Pengujian Kegunaan Halaman yang Berbeda

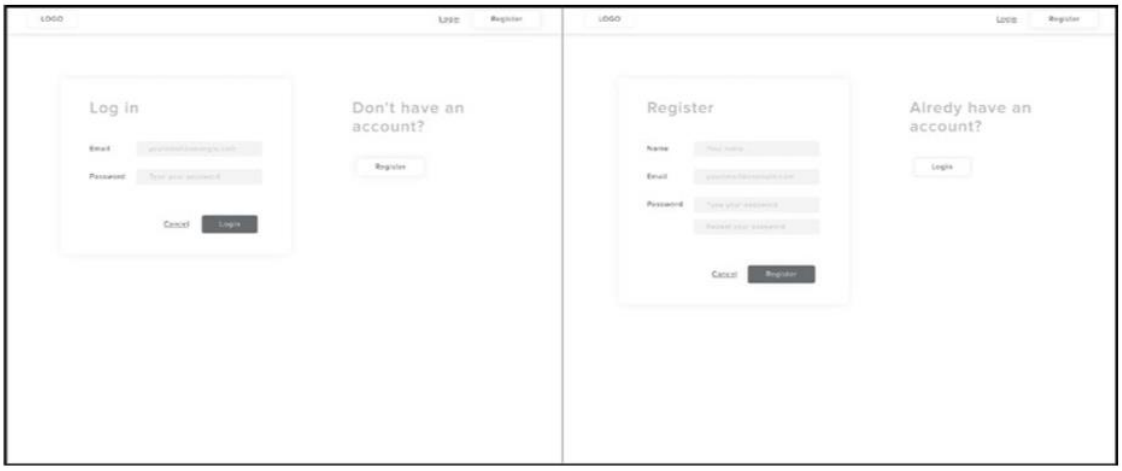
Halaman	**Hasil**
1. Halaman Pre-Login	Halaman ini terstruktur dengan jelas untuk pengguna dan sangat intuitif. Dua harapan utama pengguna adalah: • Klasifikasi video di berbagai bidang • Kemungkinan penjelasan sertifikasi, atau testimoni pengguna. Pengguna ingin tahu persis MENGAPA platform ini bisa lebih baik dari yang lain.
2. Halaman Registrasi	Pengguna bingung bahwa formulir pendaftaran dibiarkan rata. dia berharap itu ditempatkan di tengah. • Tombol login harus ditempatkan di atas formulir pendaftaran.

	• Bidang "nama" berlebihan. Pengguna ingin memiliki sesedikit mungkin bidang untuk diisi. • Selain itu, pengguna mengharapkan (dari platform pendidikan online lainnya) bahwa ia dapat mendaftar melalui Facebook atau google apI.
3. Halaman Login	Komentar yang sama tentang posisi bidang seperti untuk 2.
4. Halaman Dashbor (Kursus Saya)	Pengguna mengharapkan penekanan pada kursus terbarunya. • Pengguna ingin melihat rekomendasi untuk kursus serupa lebih lanjut. • Tombol logout terlalu ringkas; seharusnya tidak ditampilkan sama sekali. Pengguna tidak selalu ingin logout. dia akan mengharapkan bagian pengaturan berada di tempat tombol logout. • Dengan mengklik bidang itu, dia dapat memilih antara menyesuaikan profilnya, preferensi kursus, dan sebagainya untuk membantu meningkatkan algoritme rekomendasi. dan di bagian paling bawah kemungkinan logout. • Fungsi pencarian tidak ada. Pengguna mengatakan dalam banyak kasus dia tahu persis apa yang dia cari dan ingin menemukannya dengan cepat.
5. Halaman Kursus dengan Topik	Hal pertama yang membingungkan pengguna adalah tombol "Tes Kursus". Tujuan dari tombol tersebut masih belum jelas. Tata letak tampaknya tidak biasa bagi pengguna. Untuk ikhtisar kursus yang lebih baik, pengguna lebih suka mengharapkan topik dicantumkan di sisi kiri di bawah satu sama lain. Di sisi kanan, masing-masing deskripsi topik singkat dan durasi. Bidang pencarian juga hilang di sini.
6. Halaman Topik Kursus	Tata letak terlihat sangat intuitif dan mudah dimengerti, tetapi pengguna diharapkan untuk melihat lebih detail tentang konten topik (misalnya, sebagai deskripsi singkat di bawah video), durasi, guru, komentar dari pengguna lain, dan sebagainya. Pertanyaan pilihan ganda agak tidak terduga pada saat ini.

Seperti yang Anda lihat, kami dapat menemukan beberapa masalah kegunaan menarik yang pasti akan membantu kami dalam membangun platform kami. Idealnya, kami akan mengatasi semua masalah dan menjalankan kembali pengujian. Namun, sebagian besar masalah yang terlihat dapat diatasi selama implementasi sebagai fitur masa depan; dengan demikian, kita akan meninggalkan maket seperti apa adanya untuk saat ini.

Desain visual

Setelah maket siap dan pengujian pengguna selesai, saatnya kita mulai menggambar desain visual. Sekarang kita berbicara tentang UI dengan warna, batas, ukuran tombol, dan elemen visual lainnya. Pada titik ini, kami tidak akan memperkenalkan warna, font, ukuran tombol, atau elemen lain dari sistem desain, karena ini adalah bagian dari identitas perusahaan, dan kami tidak memiliki perusahaan apa pun; kami baru saja membangun MVP kami. Oleh karena itu, kami akan membiarkannya dalam skala abu-abu. Desain halaman login dan registrasi akan terlihat, misalnya, seperti yang ditunjukkan pada Gambar 4-21.



Gambar 4-21. Desain visual untuk halaman Login dan Registrasi

Desain ini dirancang oleh Alexandra Siryk, desainer kami dari EdEra. Dia sebenarnya adalah orang yang bertanggung jawab untuk mendesain sebagian besar ilustrasi buku ini. Tidak akan ada cukup bir untuk membayarnya atas kerja keras dan menakjubkannya!

4.9 Kesimpulan

Dalam bab ini kita berbicara tentang desain, berbagai jenisnya, serta berbagai jenis orang yang mengerjakannya. Kami membahas perbedaan utama antara UI dan UX, dan kami menjelajahi contoh yang memiliki UI dan/atau UX yang baik atau buruk. Kami berbicara tentang pengujian kegunaan, dan kami telah membahas berapa banyak pengguna yang perlu Anda miliki untuk menguji produk Anda dengan benar. Kami menjalankan pengujian kegunaan dengan platform pendidikan kami. Kami melakukan brainstorming tentang fitur-fitur yang harus dimiliki platform kami dan mengembangkan mockup yang digambar tangan dan wireframes dengan fidelitas rendah dan fidelitas tinggi untuk itu. Periksa seberapa baik Anda memahaminya di bawah ini.

4.10 Tes Kemampuan

- Apa singkatan dari UI?
    - Internet Berguna
    - Pendapatan Pengguna
    - Antarmuka pengguna
    - Antarmuka Unik
- Anda membuka beberapa situs web dan mengalami orgasme visual. Anda melihat warna yang paling indah, bentuk yang sangat bagus, font yang menakjubkan, dan kaligrafi yang paling indah. Sangat indah sehingga Anda ingin segera menggunakan layanan bahkan tanpa mengetahui apa yang ditawarkan layanan tersebut. setelah beberapa pengguliran Anda menemukan tombol "bergabung dengan kami", Anda mengkliknya, dan Anda diarahkan ke halaman lain yang indah dengan beberapa tombol lagi. Pada akhirnya Anda harus mengklik tombol yang berbeda lima kali sampai benar-benar masuk ke formulir pendaftaran. Ini adalah contoh yang jelas dari:
    - implementasi yang baik dengan desain yang buruk
    - UI yang sangat bagus tetapi UX yang membingungkan
    - manajemen proyek yang mengerikan tetapi manajemen produk yang bagus
    - frontend bagus tapi backend jelek
- menurut kelompok nielsen dan norman, berapa banyak pengguna yang cukup untuk menemukan sebagian besar masalah kegunaan?
    - sebanyak mungkin!
    - Satu pengguna sudah cukup—lebih baik menggunakannya dengan satu orang daripada tanpa siapa pun.
    - Lima pengguna sudah cukup karena pengujian dengan lima pengguna akan mengungkap sekitar 85% masalah kegunaan.
    - Tes kegunaan tidak diperlukan, manajer produk dapat menemukan semua masalah dalam produk.
- menurut bab ini, apa atau siapa yang harus menjadi pusat desain kita?
    - produk
    - Pengguna
    - Bisnis
    - Pengalaman

Jika Anda menjawab “Antarmuka Pengguna—UI yang sangat bagus tetapi UX—5 pengguna—Pengguna yang membingungkan”, maka Anda dapat melanjutkan ke bab berikutnya!

Sekarang kami siap untuk implementasi MVP kami yang sebenarnya! Di bab berikutnya Anda akan menjadi pengembang backend dan melakukan bagian yang tak terlihat namun kuat—berurusan dengan database, server, dan API untuk menyiapkan data aplikasi kami untuk dibuat, digunakan, dan ditampilkan kepada pengguna kami. Apakah Anda siap untuk hardcore? Ayo pergi!

BAB 5
PENGEMBANGAN BACKEND

Kami telah datang jauh! Kami harap Anda sama bersemangatnya dengan kami karena ini adalah bab pertama di mana kami akan menulis kode. Pada bab sebelumnya kami membahas implementasi desain produk kami. Anda melihat bagaimana semuanya berkembang dari fase konsep, melewati alur, maket, dan gambar rangka hingga tahap visual terakhirnya: piksel sempurna. Dalam bab ini kita akan memulai proyek kita dengan membangun aplikasi backend kita yang nantinya akan mendukung semua yang dibutuhkan oleh aplikasi frontend. Kami akan mulai dengan mem-bootstrap proyek, sehingga dapat dibangun dengan cara yang nantinya dapat kami terapkan.

Kami akan berbicara tentang penyimpanan data dan cara menyusun aplikasi dan menghasilkan arsitektur yang baik untuk meminimalkan kemungkinan ketidaknyamanan di masa depan. Ingatlah bahwa bab ini akan memperkenalkan banyak istilah baru, dan kami akan memberikan beberapa definisi dan penjelasan singkat, tetapi tidak akan membahas detailnya. Tujuannya adalah pada akhirnya Anda mendapatkan ide tentang apa yang Anda butuhkan untuk membangun aplikasi backend dan bagaimana bagian-bagian ini bekerja bersama. Ini akan memiliki beberapa bagian praktis, tetapi seperti yang dapat Anda bayangkan proyeknya cukup besar, jadi kami hanya akan memandu Anda melalui contoh singkat tentang cara mem-bootstrap proyek dan beberapa detail implementasi kecil, sehingga Anda bisa merasakan bagaimana aplikasi backend dibuat.

Catatan Jika Anda berpikir untuk melewatkan bab ini karena Anda lebih tertarik pada topik lain, seperti frontend, kami mengundang Anda untuk berpikir dua kali. Kami percaya bahwa meskipun Anda cenderung lebih condong ke satu sisi alur pengembangan, penting untuk memiliki pengetahuan tentang cara kerja bersama secara umum. Jangan khawatir, karena kita tidak akan membahas banyak detail, tetapi hanya semua langkah yang diperlukan untuk mem-bootstrap proyek backend dan juga pertimbangan yang perlu kita lakukan saat merancang aplikasi backend. Kami harap Anda yakin sekarang, dan Anda akan terus mengikuti perkembangannya!

5.1 Tentang Tumpukan...

Kami memutuskan tumpukan berikut karena kami merasa itulah yang dapat kami jelaskan dengan lebih baik dan lebih bersemangat. Kami tidak mengatakan bahwa ini adalah tumpukan terbaik—sebenarnya, kami tidak percaya bahwa ada "tumpukan terbaik"; kami percaya ada sekumpulan tumpukan yang bagus untuk apa yang ingin Anda lakukan—beberapa lebih baik daripada yang lain tergantung pada sifat proyek. Ada beberapa hal yang dapat membantu Anda memutuskan apa yang harus dipilih, baik karena Anda lebih nyaman dengan bahasa pemrograman tertentu, atau karena ada satu yang Anda rasa lebih baik untuk memecahkan masalah tertentu yang Anda miliki atau hanya karena Anda terpaksa

melakukannya. gunakan tumpukan tertentu karena perusahaan Anda sudah berinvestasi sebelumnya.

Kami biasanya tidak terlibat dalam diskusi semacam ini, karena mereka cenderung tidak membawa kami ke mana-mana. Kami memutuskan untuk menulis aplikasi backend di Java 8. Mengapa? Pada dasarnya, karena Lawrence telah bekerja dengan Java selama hampir 15 tahun dan dia percaya bahwa itu adalah bahasa yang sangat bagus tidak hanya untuk dipelajari tetapi juga untuk skenario produksi. Sekali lagi, tidak ada diskusi, ini adalah pendapat pribadi Lawrence dan bukan kebenaran universal! Semua contoh seperti instalasi, perintah, dll, akan dilakukan menggunakan Ubuntu Linux versi 16.04 LTS. Jika Anda memiliki sistem lain, Anda harus mencocokkan semua yang kami lakukan di sini dengan sistem Anda, tetapi sekali lagi kami yakin tidak akan sulit untuk melakukannya. Materi untuk bab ini disebut “kursus-tidak lengkap.” Ini berisi proyek lengkap, tetapi ada beberapa bagian yang perlu Anda selesaikan untuk membuatnya berfungsi. Untuk beberapa latihan, seperti latihan otentikasi, Anda perlu menggunakan proyek yang diimplementasikan sepenuhnya.

5.2 Mendefinisikan Aplikasi Backend

Pertama-tama, mari kita definisikan apa yang sebenarnya kita bicarakan. Ketika kita berbicara tentang frontend dan backend, kita mengacu pada model client-server. Jadi, backend adalah bagian yang berjalan di server di suatu tempat di cloud dan, di antara banyak hal, bertanggung jawab untuk menangani data secara umum. Ini berarti bahwa ia bertanggung jawab untuk menerima permintaan dari klien untuk menyediakan data, sehingga mereka dapat ditampilkan. Ini tampaknya sederhana, tetapi ada banyak topik tentang backend. Di mana kita menyimpan data? Bagaimana kita mengaksesnya? Siapa yang dapat melihat data apa? Semua topik ini ditangani oleh aplikasi backend. Kami akan mencoba menguraikannya untuk Anda agar Anda dapat memahami semua aktor yang berperan dalam apa yang baru saja kami jelaskan.

5.3 Bootstrap Proyek

Sekarang kami memutuskan untuk melakukan implementasi kami di Jawa, kami perlu menemukan cara untuk membangun proyek. Membangun proyek berisi semua langkah yang diperlukan untuk membuat proyek dapat dieksekusi pada akhirnya. Dalam kasus khusus kami, itu berarti mengkompilasi semua dependensi modul dan membuat bundel yang dapat berjalan di server aplikasi. Untuk itu, kita dapat menulis skrip atau hanya berasumsi bahwa kita dapat melakukannya secara manual, tetapi percayalah, saat ini proyek menjadi besar dengan sangat cepat, dan hampir tidak mungkin untuk membuatnya tanpa Alat Otomasi Bangun yang tepat. Dalam kasus kami, kami akan menggunakan Maven karena ini adalah alat standar de facto yang diterima secara umum untuk membangun proyek Java. Maven berjalan di Java jadi mari kita instal Java Development Kit dari Oracle karena kita juga membutuhkannya untuk mengkompilasi dan menjalankan proyek kita. Untuk itu kami akan menggunakan repositori pihak ketiga. Cukup buka terminal dan mulailah mengetik yang berikut:

```
sudo add-apt-repository -y ppa:webupd8team/java
```

```
sudo apt-get update
sudo apt-get install oracle-java8-installer
sudo apt-get install oracle-java8-set-default
```

Jika semuanya berjalan dengan baik, Anda akan melihat sesuatu yang mirip dengan itu ketika Anda menjalankan "java -version".

```
java version "1.8.0_102"
Java(TM) SE Runtime Environment (build 1.8.0_102-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.102-b14, mixed mode)
```

Abaikan "_102" asalkan dimulai dengan "1,8" kita akan pergi. Sekarang mari kita instal Maven. Mari kita juga menggunakan repositori pihak ketiga:

```
sudo add-apt-repository -y ppa:natecarlson/maven3
sudo apt-get update
sudo apt-get --assume-yes install maven3
sudo ln -sf /usr/bin/mvn3 /usr/bin/mvn
```

Siap? Jalankan saja "mvn -version" dan Anda akan melihat sesuatu seperti berikut:

```
Apache Maven 3.2.1 (ea8b2b07643dbb1b84b6d16e1f08391b666bc1e9; 2014-0214T18:37:52+01:00)
Maven home: /usr/share/maven3
Java version: 1.8.0_102, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-8-oracle/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "4.12.2-041202-generic", arch: "amd64",
family: "unix"
```

Dan itu saja. Kami memiliki lingkungan yang ditetapkan untuk pengembangan, tetapi kami juga membutuhkan satu hal yang sangat penting: editor. Sekali lagi, diskusi ini bisa memakan waktu lama dan masih belum mengarah ke mana-mana, jadi saya akan meninggalkan yang ingin kita gunakan. Ini disebut IntelliJ IDEA, dan Anda dapat mengunduh versi Komunitas secara gratis di https://www.jetbrains.com/.

5.4 Bangun Alat Otomasi: Maven

Apa yang menurut Lawrence keren tentang Maven adalah mudah digunakan, mudah dibaca, dan memiliki banyak plugin yang dapat mempermudah hidup Anda. Untuk membuat proyek kami, kami perlu menuliskan sejumlah besar konfigurasi, karena bergantung pada beberapa dependensi. Kami tidak akan melakukannya di sini. Kami merasa jauh lebih menarik bagi Anda untuk melihat bagaimana segala sesuatunya dimulai, karena setelah itu lebih sama. Maven menawarkan cara untuk mem-bootstrap proyek atau membuat kerangka menggunakan arketipe. Dalam kasus kami, kami tidak akan menggunakan arketipe yang ada, kami akan membuat proyek sendiri dengan menulisnya dari awal. Untuk mendefinisikan proyek, kami menggunakan file pom.xml (Project Object Model). Masing-masing file ini mendefinisikan modul Maven. Mari kita mulai dengan root pom.xml, titik masuk proyek kita. Pertama-tama kita buat folder untuk menampung project, sebut saja course, misalnya. Cukup beralih ke folder tempat Anda ingin menyimpan semuanya, buat folder kursus, dan pom.xml berikut di dalamnya:

```
mkdir projects
cd projects
mkdir courses
cd courses
cat << EOF > pom.xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.
w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>com.example</groupId>
<artifactId>courses</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>pom</packaging>
</project>
EOF
```

Kami telah menggunakan beberapa perintah shell (mis., mkdir untuk membuat folder). Jika Anda tidak terbiasa dengan perintah ini, silakan google mereka (http://lmgtfy.com/?q=shell+commands). Ini adalah modul root dari proyek. Anda sudah dapat membangunnya dengan memanggil "mvn install". Itu belum akan membangun apa pun, karena kosong. Kami menyarankan bahwa pada titik ini Anda membuka Lingkungan Pengembangan Terintegrasi (IDE) pilihan Anda (kami akan menggunakan IntelliJ IDEA, seperti yang kami nyatakan sebelumnya). Hal pertama yang Anda tentukan adalah koordinat proyek, artinya nilai yang secara unik mengidentifikasi proyek Anda:

1. **groupId**: Harus unik, biasanya sebagian besar proyek menggunakan nama domain perusahaan atau proyek jika demikian. Dalam kasus kami, kami akan menggunakan com.example.
2. **artifactId**: Mencocokkan folder yang menampung modul. Dalam kursus kasus kami.
3. **versi**: Versi proyek kami saat ini.
4. **kemasan**: Jenis modul. Dalam hal ini, kami mengatakan itu adalah modul pom, yang berarti bahwa itu adalah modul yang akan menampung modul lain.

Karena proyek kami masih dalam pengembangan dan belum ada rilis, kami menggunakan akhiran “-SNAPSHOT”. Jangan khawatir tentang ini, Anda dapat mempelajarinya lebih lanjut jika Anda mau dengan memeriksa dokumentasi; tetapi agar Anda mengetahui secara kasar apa artinya, itu hanya memberi tahu Maven bahwa proyek Anda masih dalam pengembangan dan dapat berubah. Karena proyek kami adalah proyek Java, kami perlu mengompilasinya, dan untuk itu Maven sudah memiliki plugin untuk melakukan ini untuk Anda. Mari kita sertakan di root pom.xml kita:

```
(...)
<properties>
<java.version>1.8</java.version>
<maven.compiler.plugin.version>3.7.0</maven.compiler.plugin.version>
```

```
</properties>
<build>
<pluginManagement>
<plugins>
<plugin>
<artifactId>maven-compiler-plugin</artifactId>
<version>${maven.compiler.plugin.version}</version>
<configuration>
<source>${java.version}</source>
<target>${java.version}</target>
</configuration>
</plugin>
</plugins>
</pluginManagement>
<plugins>
<plugin>
<artifactId>maven-compiler-plugin</artifactId>
</plugin>
</plugins>
</build>
(...)
```

Itu selalu merupakan ide yang baik untuk menyimpan versi dependensi dan plugin Anda di bagian properti POM root Anda sehingga dibagikan di antara semua sub-modul yang Anda buat. Ini memudahkan di masa mendatang untuk meningkatkan dan memastikan semua modul menggunakan versi yang sama. Ya... Anda mungkin pada suatu saat perlu menggunakan versi yang berbeda untuk modul tertentu, tetapi Anda selalu dapat mengganti definisi itu nanti, jadi jangan khawatir tentang itu. Apa yang baru saja kami tentukan adalah bahwa kami akan menulis proyek kami di Java 1.8 dan kami ingin memproduksi paket Java 1.8.

Bagian manajemen plugin adalah tempat Anda menentukan plugin, versi, dan konfigurasi yang akan digunakan. Bagian build berarti Anda ingin proyek ini menggunakan konfigurasi plugin yang baru saja Anda tentukan di bagian manajemen. Dengan melakukannya seperti ini, setiap modul dalam proyek akan menggunakan konfigurasi ini, artinya Anda tidak perlu menulisnya untuk setiap modul. Kemungkinan Anda akan menggunakan fungsionalitas umum di seluruh proyek, jadi sebaiknya buat modul yang menampung fungsi umum Anda. Biasanya modul ini adalah modul sederhana dengan ketergantungan sesedikit mungkin, jadi cobalah untuk tidak menyertakan kerangka kerja yang berat. Dalam kasus kami, kami akan membuat modul yang disebut commons:

```
mkdir commons
cd commons
cat << EOF > pom.xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.
```

```
w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<artifactId>commons</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
<parent>
<groupId>com.example</groupId>
<artifactId>courses</artifactId>
<version>1.0-SNAPSHOT</version>
<relativePath>../pom.xml</relativePath>
</parent>
</project>
EOF
```

Jadi, kami baru saja membuat modul bernama commons yang mewarisi semua konfigurasi dari modul root utama, seperti yang kami nyatakan di bagian induk. Seperti yang mungkin Anda perhatikan, modul baru ini berbeda dari yang lain, karena ini bukan modul yang menampung modul lain tetapi menghasilkan artefak JAR yang sebenarnya. JAR tidak lain adalah file zip dengan kode yang dapat dijalankan. Kita hanya perlu melakukan hal lain untuk melengkapi hubungan antara orang tua dan anak “umum” yang baru lahir. Kita perlu memberi tahu orang tua bahwa modul ini miliknya. Untuk itu kami membuat bagian yang disebut modul dan menyertakan modul commons di pom induk:

```
(...)
<modules>
<module>commons</module>
</modules>
(...)
```

Seperti yang dikatakan sebelumnya, konfigurasi untuk proyek menjadi cukup besar, jadi kami mengundang Anda untuk melihat versi final dan menjelajahinya, berkenalan dengan strukturnya, tetapi sekali lagi jangan terlalu mengkhawatirkannya. Tujuan dari latihan kecil ini hanya agar Anda mempelajari istilah dan cara membuat proyek dari awal menggunakan Maven. Jika sekarang Anda menjalankan "maven install" di root proyek Anda, Anda akan melihat bahwa model "commons" juga dibuat:

```
(...)
[INFO] Reactor Summary:
[INFO]
[INFO] course ..................................... SUCCESS [  0.479 s]
[INFO] commons .................................... SUCCESS [  1.314 s]
[INFO] ------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------
[INFO] Total time: 1.947 s
[INFO] Finished at: 2018-04-23T22:22:46+02:00
```

```
[INFO] Final Memory: 10M/193M
[INFO] ------------------------------------------------------------------------
```

Apa yang baru saja kami lakukan adalah membuat struktur proyek menggunakan Maven. Meskipun tidak ada kode atau tidak ada yang perlu dibuat, beginilah cara Anda membangun proyek dari awal. Struktur yang baru saja kita buat ini siap untuk berkembang, jadi kita bisa mulai menambahkan lebih banyak dependensi, plugin, atau bahkan membuat modul baru! Anda dapat melihat proyek lengkapnya jika Anda tertarik dengan tampilannya saat selesai.

5.5 Database

Seperti yang dinyatakan sebelumnya, backend bertanggung jawab untuk menangani data. Ini berarti bahwa ia bertanggung jawab untuk mengambil dan menyimpan data ini, dan untuk itu ia menggunakan basis data. Dalam rekayasa perangkat lunak, sebagian besar waktu ketika kita merujuk ke database, kita mengacu pada database relasional (RDBMS), tetapi segala bentuk data terorganisir dapat disebut database. Ada beberapa jenis database: persisten, dalam memori, relasional, penyimpanan kolom, penyimpanan dokumen, atau bahkan file teks biasa. Daftar ini bisa bertambah besar seiring berjalannya waktu dan yang baru ikut bermain, tetapi dalam kasus kami, kami akan berbicara tentang database relasional. Meskipun database non-relasional telah mendapatkan banyak popularitas dalam beberapa tahun terakhir, database relasional masih banyak digunakan. Mari kita jelaskan beberapa pro dan kontra dari database relasional sehingga Anda memahami dengan tepat apa yang sedang kita bicarakan.

5.6 Pro

1. Data terstruktur sehingga tidak ada duplikasi
2. Kendala pada data, yang mengarah ke lebih sedikit data yang salah dari aplikasi
3. Hubungan antar data dapat memiliki kendala, sehingga aman untuk membangun hubungan dengan memastikan mereka benar-benar ada.
4. Dukungan transaksional, artinya kita dapat membuat sesi, melakukan serangkaian operasi, dan jika terjadi kesalahan, semuanya dikembalikan, dan kita berakhir di keadaan sebelumnya.
5. Isolasi dari sesi lain; tidak ada perubahan yang terlihat oleh mereka sampai sesi yang memperkenalkan perubahan ditutup oleh komit (ini sebenarnya tergantung pada tingkat isolasi, yang berada di luar cakupan penjelasan ini).

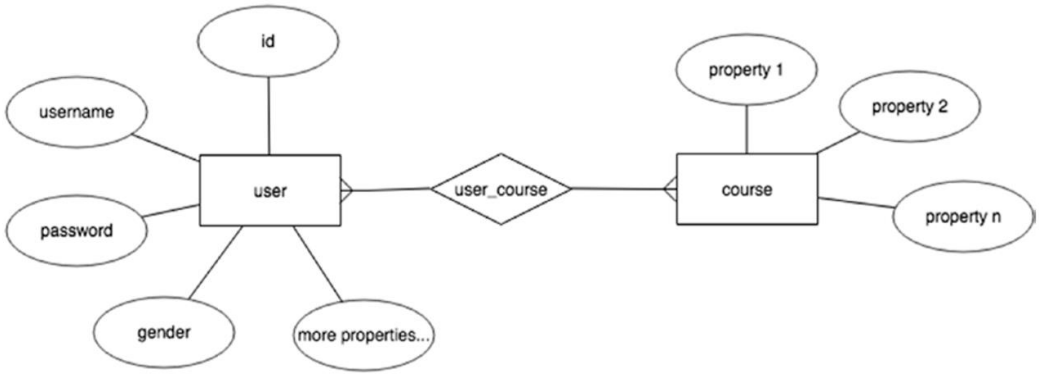
5.7 Kontra

1. Tidak mudah untuk menskalakan secara horizontal (dengan menambahkan lebih banyak instance dan tidak hanya memutakhirkan perangkat keras). Sekadar catatan, pada umumnya sebagian besar vendor bahkan tidak menyediakan fitur tersebut. Itulah mengapa database NoSQL dikembangkan—untuk mengatasi masalah itu. Aurora RDBMS dari Amazon AWS menyediakan solusi di antara, di mana dimungkinkan untuk membangun klaster tempat beban kerja baca didistribusikan. Saya harus mengatakan, lebih baik daripada tidak sama sekali!

2. Data dinormalisasi, yang berarti bahwa terkadang untuk mengumpulkan beberapa informasi sederhana, kita perlu membuat kueri atau gabungan kompleks yang menimbulkan kelemahan dalam kinerja
3. Tidak ada objek kompleks, karena sebagian besar bekerja dengan tipe sederhana seperti angka, string, tanggal, dll.

Apakah itu berarti kita harus selalu menggunakan RDBMS? Tentu saja tidak! Kita harus menggunakan apa pun yang sesuai dengan kebutuhan kita dengan lebih baik. Secara umum, Lawrence selalu suka memikirkan pendekatan hybrid, menggunakan RDBMS untuk menyusun data dan menggunakan penyimpanan dokumen dengan kemampuan pencarian saat kita membutuhkan objek yang kompleks, misalnya. Dalam kasus kami dan untuk tujuan pengembangan, kami akan menggunakan database yang tidak perlu kami instal dan konfigurasikan. Ini disebut H2 dan kita dapat menggunakannya sebagai tertanam, artinya itu akan berjalan di dalam aplikasi kita. Tentu saja, untuk skenario produksi, ini bukan opsi, tetapi ini adalah solusi yang cukup baik untuk pengembangan, jadi kami tidak peduli dengan sisanya. Jika nanti kita memutuskan untuk mengubah database, tidak ada masalah karena kita akan menggunakan library *Object Relational Mapping* (ORM) yang mengabstraksi dan menerjemahkan semua instruksi ke vendor database yang kita inginkan. Hal ini dimungkinkan, terutama karena SQL (Structured Query Language, bahasa khusus yang digunakan untuk "berbicara" dengan database untuk menyimpan, memodifikasi, dan mengambil data) menjadi standar dan sebagian besar vendor mengikuti standar itu.

Saat mendesain skema database, kita sering menggunakan diagram *Entity Relation* (ER) di mana kita menyatakan entitas (tabel) dan relasinya. Untuk tujuan contoh yang sedang kita buat sekarang, kita hanya akan bekerja dengan dua entitas: pengguna dan kursus, seperti yang ditunjukkan pada Gambar 5-1.



Gambar 5-1. Diagram Relasi Entitas dari tabel pengguna dan kursus

Dalam diagram kami membangun hubungan antara pengguna dan entitas kursus. Relasi ini merupakan relasi N-ke-N, artinya seorang pengguna dapat mendaftar pada beberapa mata kuliah dan suatu mata kuliah dapat didaftarkan oleh beberapa pengguna; sebagai hasilnya, entitas baru yang disebut user_course ikut bermain. Entitas baru ini akan berisi kunci utama dari entitas pengguna dan kursus. Kunci ini dikenal di entitas baru ini sebagai kunci asing.

Sebelum kita melanjutkan, mari kita definisikan beberapa istilah yang baru saja kita sebutkan. Kami mengundang Anda untuk mempelajari lebih lanjut tentang mereka; Wikipedia dan Google dapat membantu Anda dengan itu.

1. **Tabel kolom dan baris**: Tabel adalah struktur yang menyimpan data dalam bentuk kolom dan baris. Data masuk ke tabel menggunakan baris, sedangkan kolom menentukan data apa yang ada (sangat mirip dengan lembar excel).
2. **Kunci utama**: Kunci yang secara unik mengidentifikasi satu baris dalam tabel. Dalam skema modern, kunci utama sering kali berupa angka yang ditetapkan ke baris saat dibuat.
3. **Kunci asing**: Kunci yang mengidentifikasi baris pada tabel lain. Dalam hal ini jika Anda menggabungkan kunci pengguna dengan kunci kursus, Anda dapat mengatakan bahwa pengguna X melakukan kursus Y.
4. **Relasi**: Mendefinisikan bagaimana tabel berhubungan satu sama lain. Hubungan N-ke-N menghasilkan tabel lain dengan kunci utama dari kedua tabel sebagai kunci asing; Relasi 1-ke-N tidak menghasilkan lebih banyak tabel tetapi akan menghasilkan kunci asing di sisi N relasi, mengacu pada kunci utama sisi "1".

Ini adalah hal yang sangat mendasar yang perlu Anda ketahui tentang database relasional. Jika Anda merasa suka berpetualang, Anda masih dapat mencari di Google untuk "bentuk normal pertama, kedua, dan ketiga", tetapi konsep-konsep ini tidak terlalu mudah untuk dicerna seperti yang biasanya dijelaskan. Jadi, pada tahap ini kita memiliki tiga tabel yang perlu diperhatikan: pengguna, kursus, dan relasinya user_course. Ada dua jenis utama pernyataan SQL: *Data Definition Language* (DDL) dan *Data Modification Language* (DML). Mereka cukup jelas; yang pertama digunakan untuk membuat atau mendefinisikan (skema) bagaimana data kita disimpan dan diatur. Yang terakhir digunakan untuk berinteraksi dengan skema yang kita definisikan—menyisipkan, memperbarui, dan menghapus data. Kami tidak akan membahasnya secara detail, karena kami memiliki alat yang bagus yang akan melakukan ini untuk kami. Ini disebut liquibase, dan ini adalah pustaka sumber terbuka yang membantu kami mendefinisikan skema basis data dan melacak perubahannya. Kami mengundang Anda untuk memeriksa proyek liquibase—khususnya file changelog.xml—untuk keseluruhan struktur database. Terlalu rumit? Mari kita coba mencocokkan hanya yang kita definisikan sebelumnya di bab ini:

```
<changeSet
 id="20180225-1700"
author="Lawrence Vilao">
<createTable
tableName="user">
<column
name="id"
type="BIGINT(19)"
autoIncrement="true">
<constraints
primaryKey="true" />
</column>
```

```
<column
name="create_time"
type="datetime"
defaultValueComputed="CURRENT_TIMESTAMP" />
<column
name="update_time"
type="datetime" />
<column
name="username"
type="VARCHAR(256)">
 <constraints
 nullable="false" />
</column>
<column
name="password"
type="VARCHAR(128)" />
<column      name="salt"
type="VARCHAR(128)" />
<column
name="name"
type="VARCHAR(256)" />
</createTable>
</changeSet>
<changeSet
id="20180424-2230"
author="Lawrence Vilao">
<createTable
tableName="course">
<column      name="id"
type="BIGINT(19)"
autoIncrement="true">
<constraints
primaryKey="true" />
</column>
<column
name="create_time"
type="datetime"
defaultValueComputed="CURRENT_TIMESTAMP" />
<column
name="update_time"
type="datetime" />
</createTable>
</changeSet>
<changeSet
```

```
id="20180424-2231"
author="Lawrence Vilao">
<createTable
tableName="user_course">
<column
name="id"
type="BIGINT(19)"
autoIncrement="true">
<constraints
primaryKey="true" />
</column>
<column
name="create_time"
type="datetime"
defaultValueComputed="CURRENT_TIMESTAMP" />
<column      name="update_time"      type="datetime" />
<column
name="user_id"
type="BIGINT(19)">
<constraints
nullable="false" />
</column>
<column
name="course_id"
type="BIGINT(19)">
<constraints
nullable="false" />
</column>
</createTable>
<addForeignKeyConstraint
baseTableName="user_course"
baseColumnNames="user_id"
constraintName="fk__user_course__user_id"
referencedTableName="user"
referencedColumnNames="id" />
<addForeignKeyConstraint
baseTableName="user_course"
baseColumnNames="course_id"
constraintName="fk__user_course__course_id"
referencedTableName="course"
referencedColumnNames="id" />
</changeSet>
```

Kami mulai dengan mendefinisikan tabel bernama pengguna dengan id, nama pengguna, kata sandi, dll. Kemudian kami mendefinisikan tabel, kursus, yang untuk saat ini hanya

menyimpan id dan beberapa metadata lain seperti waktu pembuatan dan waktu pembaruan (terutama berguna untuk melacak dari perubahan). Kami kemudian mendefinisikan tabel relasi yang disebut user_course, di mana kami mengatakan bahwa itu terkait dengan tabel pengguna dan tabel kursus.

Perhatikan kunci utama—di sini, kami dapat dengan sempurna mengatakan bahwa kunci utama dapat berupa user_id dan course_id bersama-sama, tetapi kami ingin agar orang-orang dapat mendaftar lebih dari satu kali dalam kursus yang sama, tetapi tidak pada waktu yang sama. Tampaknya sah-sah saja jika Anda mendaftar di satu kursus, menyelesaikannya, dan kemudian Anda ingin memulainya lagi—contoh kursus yang benar-benar baru dengan semuanya kosong (kuis, jawaban, dll.). Sekarang Anda penasaran bagaimana semua ini bekerja, buka proyek kami dan kompilasi secara keseluruhan (folder root proyek):

```
mvn clean install -DskipTests
```

Setelah Anda selesai dengan itu, jalankan yang berikut ini untuk membuat database:

```
cd liquibase
mvn liquibase:update
```

Ini akan membuat beberapa file database di direktori /tmp Anda. Jika Anda menggunakan Windows, Anda mungkin harus mengubah folder sementara di parameter url dari pom.xml proyek liquibase dan juga di file ketekunan-layer.properties (di dalam modul lapisan ketekunan). Pada tahap ini Anda memiliki database sederhana yang dibuat di sistem Anda. Jika Anda menjalankan perintah ini lagi, Anda akan melihat bahwa tidak ada yang terjadi, karena tidak ada log perubahan baru yang akan diterapkan. Sekarang jika Anda ingin menambahkan kolom, misalnya, Anda dapat membuat set perubahan baru, mengkompilasi proyek, dan menjalankan perintah lagi, dan hanya perubahan baru yang akan dijalankan. Cukup keren, bukan? Jika karena alasan tertentu Anda berakhir dengan masalah pada database Anda saat mencobanya atau Anda ingin menghapusnya, hapus saja semua file, dimulai dengan kursus, di folder tmp Anda:

```
rm /tmp/courses.*
```

Dan kita sudah selesai dengan database. Jika Anda tertarik dengan topik ini dan ingin mempelajarinya lebih dalam, ada banyak materi di Internet. Kami mengundang Anda untuk memeriksanya!

5.8 Autentikasi

Otentikasi adalah cara untuk membuktikan kepada beberapa entitas bahwa Anda sebenarnya adalah siapa yang Anda katakan. Agar ini mungkin, otoritas ikut bermain. Contoh yang baik adalah sebagai berikut: ketika pergi ke negara lain, Anda membawa paspor Anda. Setelah Anda tiba di tempat tujuan, Anda harus menyerahkan paspor Anda kepada petugas yang bertanggung jawab. Dalam contoh ini, Anda adalah entitas yang mencoba mengautentikasi, paspor adalah tanda autentikasi Anda, dan petugas adalah autentikator, artinya orang yang akan atau tidak akan memberi Anda akses berdasarkan fakta bahwa ia memercayai otoritas yang mengeluarkan Anda paspor.

Dalam rekayasa perangkat lunak tidak ada bedanya. Ada beberapa cara untuk menerapkan otentikasi, beberapa metode dan teknik, dan coba tebak: ya, tidak ada cara terbaik yang bekerja untuk semuanya; setiap kasus adalah kasus. Bahkan untuk HTTP Basic Auth, yang merupakan salah satu cara autentikasi yang paling tidak disarankan karena pengguna dan kata sandi berjalan berdampingan dalam permintaan yang sama dan harus disertakan untuk setiap permintaan, ada tempatnya. Protokol OAuth2, yang akan kami gunakan untuk proyek kami, juga menggunakan HTTP Basic Auth saat meminta untuk mengautentikasi pengguna, dan penyedia layanan pembayaran (PSP) juga menggunakan Basic Auth untuk mendorong notifikasi ke server merchant, di antara contoh lainnya adalah di luar cakupan bagian ini. OAuth2 adalah salah satu protokol otentikasi paling populer yang digunakan saat ini.

Kami percaya alasan mengapa itu menjadi sangat populer adalah karena fitur yang Anda dapat mendelegasikan otentikasi ke otoritas tepercaya. Bayangkan Anda ingin membangun situs web Anda sendiri, tetapi Anda tidak ingin menerapkan mekanisme otentikasi Anda. Anda cukup mendelegasikan bagian ini ke Google atau Facebook, misalnya. Tapi ini hanyalah salah satu mode yang disediakan OAuth2 untuk otentikasi. Ini disebut Hibah Kode Otorisasi. Dalam contoh kami, kami akan menggunakan Hibah Kredensial Kata Sandi Pemilik Sumber Daya, yang berarti bahwa pengguna akan memberikan nama pengguna dan kata sandi untuk diautentikasi dengan otoritas kami. Siap? Ayo mulai! Untuk latihan ini gunakan "kursus" proyek lengkap. Beralih di dalam proyek kami dan kompilasi:

```
mvn clean install -DskipTests
```

Kemudian beralih di dalam modul yang disebut rest-api dan jalankan (jika Anda tidak mengatur database, jalankan "mvn liquibase: update" di dalam modul liquibase).

```
mvn tomcat7:run
```

Kami mengonfigurasi plugin Tomcat7 Maven dengan modul kami, sehingga kami dapat menjalankan server dengan aplikasi kami. Tomcat adalah salah satu server de facto untuk menjalankan aplikasi Java, tetapi tentu saja jumlahnya banyak. Ini akan membuka port yang mendengarkan pada 8080. Sekarang mari kita coba mengautentikasi pengguna:

```
curl -u webapp:test -X POST localhost:8080/oauth/token -H "Content-type: application/x-www-form-urlencoded" -d "grant_type=password&username=test@example.com&password=123456a$"
```

Anda akan melihat bahwa jawabannya adalah

```
{"error":"invalid_grant","error_description":"Bad credentials"}
```

Faktanya, tidak apa-apa... pengguna itu belum ada di database kami! Ayo daftarkan:

```
curl -v -X POST localhost:8080/api/v1/public/users -H "Content-type: application/json" -d '{"username":"test@example.com","password": "123456a$"}'
curl -u webapp:test -X POST localhost:8080/oauth/token -H "Content-type: application/x-www-form-urlencoded" -d "grant_type=password&username=test@example.com&password=123456a$"
```

Sekarang responsnya terdiri dari dua token:

```
{"access_token":"7977c2f9-6664-4bdf-9bab-8d8ce6f0bc44",
"token_type":"bearer","refresh_token":"587d6ee2-5cb2-479d-
b64f9bd8724f56e1","expires_in":86399,"scope":"read write"}
```

Mari kita jelaskan apa yang baru saja kita lakukan di sini. Untuk mengautentikasi pengguna di server autentikasi, kita perlu mengautentikasi panggilan ke server autentikasi menggunakan HTTP Basic Auth; maka kami menggunakan kredensial "webapp" (pengguna) dan "test" (kata sandi). Kemudian kami meneruskan ke badan permintaan kredensial pengguna kami (yang baru saja kami buat). Hasilnya, kami menerima dua token:

1. **Token akses**: Token yang akan kita gunakan untuk membuat permintaan selanjutnya. Token ini digunakan untuk mengakses sumber daya server dan biasanya token berumur pendek. Ini meningkatkan keamanan, karena jika karena alasan tertentu seseorang memegang token ini, itu tidak akan memiliki akses lama.
2. **Refresh token:** Token yang akan kita gunakan untuk mendapatkan token akses baru setelah token yang kita gunakan kedaluwarsa.

Jika token akses yang baru saja kami terima kedaluwarsa, kami dapat memperbaruinya dengan tidak mengirimkan kredensial pengguna lagi (inilah yang terjadi ketika, misalnya, Anda membuka aplikasi di ponsel setelah 1 atau 2 hari—Anda tidak perlu mengetik di lagi kredensial Anda). Untuk memperbarui token, kita harus melakukan hal berikut:

```
curl -u webapp:test -X POST localhost:8080/oauth/token -H "Content-type: application/x-www-form-urlencoded" -d "grant_type=refresh_token&refresh_ token=587d6ee2-5cb2-479d-b64f-9bd8724f56e1"
```

dan Anda menerima sepasang token akses dan penyegaran baru. Untuk kesederhanaan pengaturan kami, kami menyimpan semua token ini di dalam memori. Ini berarti bahwa ketika kami me-restart server atau jika kami nanti dalam produksi memutuskan untuk menambahkan server baru ke server farm kami, token ini tidak akan berlaku lagi. Ini dapat dengan mudah diperbaiki dengan menyimpannya di penyimpanan persisten (seperti database) atau menggunakan *JSON Web Tokens* (JWT), yang merupakan token otomatis di mana token itu sendiri menggambarkan pengguna dan aspek keamanannya. Sekarang Anda dapat mulai menggunakan token untuk membuat permintaan seperti berikut (ingat bahwa untuk Anda, token berbeda!)

```
curl -v -X GET localhost:8080/api/v1/secured/users/me -H "Content-type: application/json" -H "Authorization: bearer 7977c2f9-6664-4bdf-9bab8d8ce6f0bc44"
```

Di mana Anda akan mendapatkan sesuatu yang mirip dengan

```
{"id":1,"password":null,"name":null,"age":null,"gender":null,"username":
"test@example.com"}
```

Sebagai penutup, kami ingin memperkenalkan Anda pada istilah baru yang sejalan dengan Otentikasi. Istilah ini adalah Otorisasi. Orang sering menyalahgunakannya, tetapi sebenarnya sangat jelas. Kami telah mendefinisikan otentikasi sebagai mekanisme yang memverifikasi seseorang sebenarnya siapa atau apa yang mereka katakan, sementara otorisasi pada dasarnya berarti apa yang mereka otorisasi untuk akses. Misalnya, saat ini platform pendidikan online kami hanya akan memberikan akses kepada siswa, tetapi di masa depan akan memberikan akses kepada administrator dan guru. Otorisasi sering dilakukan dengan menetapkan peran saat pengguna mengautentikasi. Dalam kasus kami, kami

menetapkan peran SISWA. Di masa mendatang, kami mungkin perlu memiliki beberapa metadata dalam database, sehingga kami dapat memasukkan peran yang tepat saat pengguna login.

5.9 Perkembangan

Mari mulai melakukan beberapa pengkodean! Untuk bagian ini Anda akan menggunakan proyek "kursus-tidak lengkap". Sebelum kita mulai, mari kita jelaskan bagaimana aplikasi kita terstruktur. Kami membagi aplikasi kami menjadi modul logis di mana beberapa dari mereka juga menerjemahkan ke modul fisik. Tumpukannya adalah sebagai berikut (Gambar 5-2).

Hirarki bekerja dengan blok mandiri di mana modul di atas dapat melihat semua modul di bawahnya, tetapi tidak sebaliknya. Ini adalah praktik yang baik untuk melakukannya, karena jika di masa mendatang Anda membuat proyek baru, akan lebih mudah dan lebih bersih untuk menggunakan kembali modul ini. Mari kita berikan contoh kehidupan nyata; sangat mungkin bahwa pada suatu saat kita membutuhkan Sistem Manajemen Pelanggan (CMS). Karena ini akan beroperasi dalam database yang sama, kita dapat dengan mudah berbagi lapisan persistensi di mana kita telah memiliki semua objek yang dipetakan ke entitas dan relasi database. Sekarang mari kita lakukan pengenalan singkat setiap modul dan perannya dalam proyek.

Gambar 5-2. Tumpukan aplikasi

5.10 Database

Modul database adalah mesin database yang kami pilih untuk proyek tersebut. Dalam kasus kami, kami memilih untuk menggunakan H2. Jembatan antara ranah kami dan modul ini dilakukan dengan bantuan Spring Data, Hibernate, dan driver H2. Lapisan abstraksi ini membantu kita untuk dapat mengubah mesin database di masa mendatang tanpa terlalu mengkhawatirkan interaksi dengan database itu sendiri. Secara umum, kita harus dapat

hidup dengan abstraksi ini untuk sebagian besar proyek kita, tetapi terkadang kita perlu melewatinya untuk menggunakan beberapa fitur yang tidak didukung oleh kerangka kerja. Entah bagaimana ini baik-baik saja, tetapi Anda perlu tahu bahwa setelah Anda melakukannya, Anda mungkin memperkenalkan beberapa ketergantungan yang di masa depan mungkin menjadi rumit untuk ditangani. Jadi, jika memungkinkan, kita harus menghindarinya.

5.11 Persistence Layer/Lapisan Presistensi

Lapisan persistensi adalah kumpulan fitur yang kita buat untuk berinteraksi dengan database kita. Di dalamnya kita mendefinisikan, melalui objek Java, bagaimana database kita mencari aplikasi kita. Kami juga mendefinisikan satu set repositori, yang tidak kurang dari satu set kueri yang akan digunakan lapisan atas kami untuk mengambil dan menyimpan data. Repositori ini menggunakan entitas yang kami definisikan untuk memetakan tabel kami, kolomnya, dan hubungan antara tabel tersebut.

5.12 Service Layer

Di sinilah sebagian besar logika berada. Modul ini bertanggung jawab untuk berinteraksi dengan lapisan persistensi untuk menyimpan, mengubah, dan mengambil data. Dalam modul ini kami juga memvalidasi data kami sebelum disimpan, dan kami juga menentukan bagaimana operasi yang salah harus ditangani. Misalnya, jika modul atas mencoba mengambil sesuatu yang tidak ada, kami membuat kesalahan. Jika mereka mencoba untuk menyimpan sesuatu yang sudah ada, kami melempar kesalahan lain. Kesalahan tersebut kemudian ditangani oleh modul atas dengan cara yang menurut mereka terbaik. Itu dapat ditekan atau disebarkan dengan kode kesalahan yang tepat, misalnya, klien yang melakukan panggilan.

5.13 Service API

API Layanan adalah seperangkat antarmuka yang diimplementasikan oleh lapisan layanan sehingga modul atas tidak harus berurusan dengan spesifikasi implementasi aktual dari lapisan layanan. Ini tidak begitu jelas, tetapi ini lebih merupakan cara untuk mencegah kemungkinan pertumbuhan pada kompleksitas proyek. Bayangkan bahwa di masa depan kami ingin membuat modul pembayaran. Sangat mungkin bahwa modul pembayaran ini perlu menggunakan fitur dari lapisan layanan juga, tetapi kami ingin memisahkannya. Dengan cara ini kita dapat mengesampingkan modul ini dan tetap dapat berkomunikasi dengan lapisan layanan karena API tersebut kita ketahui.

5.14 REST API dan Lapisan Transformasi

REST API memaparkan aplikasi kita ke dunia luar melalui titik akhir REST. Ini bertanggung jawab untuk menerjemahkan cara kami berbicara dengan klien kami dengan cara kami berbicara dengan aplikasi backend kami. Misalnya, objek pengguna yang kita kerjakan di aplikasi frontend mungkin tidak memiliki struktur atau data yang sama dengan yang kita gunakan secara internal. REST API mengetahui hal ini dan juga dapat meminta

untuk mengubah data bolak-balik dengan bantuan lapisan terjemahan sehingga komunikasi ini dilakukan dengan benar. Selain itu, dan Anda mungkin telah memperhatikan hal ini, kami memperkenalkan pembuatan versi di API kami. Ini adalah salah satu hal terpenting yang harus dilakukan saat ini karena setiap produk yang dikenal sekarang tersedia juga di beberapa platform, terutama aplikasi seluler.

Jika Anda hanya memikirkan aplikasi frontend yang dikelola oleh Anda dan orang-orang mengambil kodenya setiap kali mereka ingin menggunakannya, kita tidak perlu membuat versi, bukan? Kami mengontrol segalanya, sehingga kami dapat membuat perubahan dan merilis semuanya, tapi... bagaimana dengan aplikasi seluler? Orang-orang tidak segera memperbarui aplikasi seluler mereka setelah dirilis—kadang-kadang mereka bahkan terus menggunakan versi lama selama berbulan-bulan! Jika, untuk beberapa alasan, Anda perlu memperkenalkan perubahan yang melanggar di API Anda, Anda dapat terus mendukung versi sebelumnya dan membuat perubahan ini hanya untuk versi 2 (yang baru akan Anda bagi hanya untuk menjaga yang lain tetap kompatibel). Pembuatan versi adalah sesuatu yang harus selalu Anda ingat saat mendesain API. Anda tidak perlu segera mengetahui apa yang harus dilakukan dan bagaimana melakukannya, tetapi Anda harus siap; dalam kasus kami, persiapan kami adalah memiliki URL khusus untuk versi satu dan format khusus untuk versi itu.

5.15 Menerapkan Pendaftaran

Untuk latihan ini gunakan proyek "kursus-tidak lengkap". Apakah Anda ingat fitur “Formulir Pendaftaran” dari Bab 3? Mari implementasikan bagian backend dari awal sekarang sehingga Anda memiliki perasaan tentang bagaimana menerapkan fitur ujung ke ujung di backend. Jadi, kami diminta untuk memungkinkan pengguna mendaftarkan diri di platform. Anda mungkin sudah memperhatikan bahwa kami memiliki dua ranah berbeda di API: aman dan publik. Mereka cukup jelas; publik tidak memerlukan otentikasi dan yang aman melakukannya. Karena tentu saja pengguna tidak dapat mengautentikasi sebelum didaftarkan, titik akhir ini harus bersifat publik. Buka file PublicUserController.java. Sekarang kami perlu memperkenalkan Anda lagi pada beberapa informasi baru. Saat merancang REST API, kami mengandalkan metode HTTP untuk jenis operasi tertentu:

1. **GET**: Digunakan untuk mengambil data. Bagaimanapun, Anda tidak boleh mengubah data saat memanggil permintaan tersebut, kecuali jika Anda ingin melakukan semacam pelacakan, tetapi tidak pernah terkait dengan data itu sendiri. Klien yang membuat permintaan *GET TIDAK AKAN PERNAH* berharap bahwa data dapat berubah. Titik akhir GET adalah nullipoten, artinya memanggilnya tidak akan pernah memiliki efek samping.
2. **POST**: Kami menggunakan yang ini untuk membuat entitas baru. Artinya, jika Anda mencoba membuat entitas lain yang sama—misalnya, dua pengguna dengan nama pengguna yang sama—harus gagal. Lebih lanjut tentang itu nanti dalam contoh.
3. **PUT**: Digunakan untuk memperbarui entitas penuh. Ini berarti bahwa ketika pengguna melakukan panggilan seperti itu, diharapkan seluruh objek diganti dengan yang baru yang dikirim.

4. **PATCH**: Mengganti elemen non-null dalam entitas. Itu harus digunakan, misalnya, jika kita hanya ingin memperbarui nama pengguna, tetapi tidak seluruh pengguna. Ini adalah cara yang paling disukai untuk memperbarui data. Secara umum, kami tidak menyarankan orang untuk menggunakan PUT dan menggunakan PATCH untuk kepentingannya.
5. **DELETE**: Gunakan metode ini untuk menghapus entitas. Terkadang penghapusan sebenarnya bukan penghapusan fisik, tetapi hanya pembatalan yang membuatnya tidak terlihat lagi untuk didaftar atau diambil (GET); mungkin berfungsi sebagai semacam bendera untuk kotor.

Oke, kembali ke contoh kita! Kami ingin menerapkan metode buat pengguna baru, jadi kami perlu mengekspos metode POST.

```
@RequestMapping(method = RequestMethod.POST)
@ResponseStatus(HttpStatus.CREATED)
public UserV1Dto create(@RequestBody UserV1Dto userV1Dto) {
return transformationsV1.user2Dto(userService.create(userV1Dto.toUser()));
}
```

Apa yang kami lakukan di sini adalah sebagai berikut: kami mengubah UserV1Dto menjadi objek pengguna yang dipahami oleh lapisan layanan kami (dan yang di bawah)—userV1Dto.toUser(). Kemudian kami mengirim objek pengguna ini ke lapisan layanan kami melalui api layanan kami. Kita akan membahasnya sebentar lagi; mari kita fokus pada apa yang terjadi di sini sekarang. Kemudian layanan ini merespons dengan objek yang sama yang diperkaya dengan lebih banyak informasi, seperti id pengguna (karena id pengguna dibuat hanya ketika kami menyimpan pengguna di database).

Kami mengambil hasil ini dan mengubahnya lagi menjadi UserV1Dto dan mengirimkannya kembali ke pengguna dengan status respons default CREATED (201). Bila memungkinkan penggunaan dan penyalahgunaan kode HTTP yang tepat; klien frontend akan berterima kasih untuk itu. Anda mungkin telah memperhatikan bahwa kami tidak menentukan titik akhir, tetapi jangan khawatir—sebenarnya, kami menetapkannya pada tingkat kelas "/api/v1/public/users". Yang terjadi dalam hal ini adalah endpoint untuk membuat entitas sama dengan root dari endpoint, tetapi tentunya dengan metode POST.

Sekarang mari kita lihat implementasi lapisan layanan. Apa yang perlu kita lakukan di sini adalah sebagai berikut: periksa apakah pengguna belum ada dan gagal jika ada; memvalidasi input—terutama nama pengguna dan kata sandi; dan menyimpan informasi di database mendapatkan kembali dengan id sehingga pengguna dapat direferensikan nanti. Untuk memvalidasi alamat email pengguna, mari gunakan ekspresi reguler (regex). Untungnya, kita tidak perlu memikirkan ini, karena banyak orang sudah melakukannya, dan jujur saja, hal semacam ini adalah tempat Anda biasanya memperkenalkan bug... jadi mari kita gunakan sesuatu yang sudah dibuat (dari http://emailregex.com/).

```
(?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\x0e-
\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-
9])?\.)+[a-z0-9](?:[a-z0-9]*[a-z0-9])?|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-
```

```
9]?)\.){3}(?:25[05]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-
\x1f\x21-\x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])
```

Cukup besar bukan? Mungkin bukan ide yang baik secara umum untuk merangkul petualangan membuat regex ini sendiri. Kedua regex sudah ada dalam kode sehingga Anda tidak perlu melakukan apa pun. Mari kita buat kode logika untuk membuat pengguna.

```
@Transactional
@Override
public User create(User user) throws AlreadyExistsServiceException {
validateUserCreation(user);
if (userRepository.findByUsernameIgnoreCase(user.getUsername()) != null) {throw
new AlreadyExistsServiceException("Username already exists.");
}
 final String salt = UUID.randomUUID().toString();
user.setPassword(passwordEncoder.encode(user.getPassword()+salt));
user.setSalt(salt);
return userRepository.save(user);
}
```

Garam? Apa-apaan itu? Hal pertama yang pertama! Kami memanggil validasi—kami akan membahas lebih detail setelahnya—dan kemudian hal pertama yang perlu kami ketahui setelah validasi selesai adalah apakah kami sudah memiliki pengguna dengan nama pengguna yang sama, dalam kasus kami, alamat email. Jika itu masalahnya, kami memberikan pengecualian. Jenis pengecualian yang kami berikan di sini penting karena kami telah menyiapkan pengontrol terjemahan pengecualian pada lapisan rest-api yang akan menerjemahkan pengecualian ini ke kode kesalahan HTTP yang benar (Anda dapat memeriksanya di DefaultExceptionHandler.java). Setelah kami baik-baik saja dengan pemeriksaan itu, kami akan mengerjakan kata sandi pengguna.

Teknik garam adalah teknik yang baik untuk mengacak sedikit lebih banyak kata sandi di database kami. Ini berguna karena ketika kita meng-hash password pada database, input akan selalu menghasilkan output yang sama, itulah mengapa hal ini bisa dilakukan seperti itu. Jadi, bayangkan seseorang entah bagaimana mencuri data database kita. Hal pertama yang akan mereka coba adalah mengelompokkan semua kata sandi hash yang sama. Kemudian mereka akan mencoba untuk mencari tahu yang paling populer, yang merupakan kandidat yang paling rentan karena itu adalah hal yang umum. Setelah peretas kami yang baik mengetahui kata sandi populer itu, itu berarti semua pengguna lain dengan hash yang sama akan memiliki kata sandi itu!

Dengan menambahkan beberapa informasi acak ke kata sandi, kata sandi yang di-hash akan selalu berbeda, jadi jika karena alasan tertentu peretas ini tiba-tiba mendapatkan satu kata sandi, bahkan jika pengguna lain memiliki kata sandi yang sama, itu akan membuat hidup mereka lebih sulit. Keren, bukan? Namun begitu sederhana. Ini tentu saja berarti bahwa ketika kita mengautentikasi pengguna, kita perlu mempertimbangkan hal ini dan melakukan hal yang sama persis: tambahkan garam ke input pengguna dan bandingkan dengan kata sandi hash yang kita miliki di database. Setelah selesai, kami mengatur semua

data yang kami buat, kata sandi hash, dan garam bersama-sama dan menyimpan pengguna. Adapun validasinya, jika Anda masih tertarik, sesederhana itu:

```
protected void validateUserCreation(User user) {
Preconditions.checkNotNull(user, "User cannot be null.");
Preconditions.checkNotNull(user.getUsername(), "Username cannot be null.");
Preconditions.checkNotNull(user.getPassword(), "Password cannot be null.");
if (!EMAIL_REGEX.matcher(user.getUsername()).matches()) { throw new
PreconditionFailedServiceException("Username is not valid.");
}
if (!PASSWORD_REGEX.matcher(user.getPassword()).matches()) { throw new
PreconditionFailedServiceException("Password is not valid.");
}
}
```

Mari kita coba! Kompilasi semuanya (mvn install -DskipTests di root proyek). Kemudian mari kita lakukan beberapa uji coba:

```
curl -v -X POST localhost:8080/api/v1/public/users -H "Content-type: application/json" -d
'{"username":"test@example.","password":"secret"}'
```

Responnya:

```
{"reason":"Precondition failed: Username is not valid."}
```

Yang lainnya:

```
curl -v -X POST localhost:8080/api/v1/public/users -H "Content-type: application/json" -d
'{"username":"test@example.com","password":"secret"}'
{"reason":"Precondition failed: Password is not valid."}
curl -v -X POST localhost:8080/api/v1/public/users -H "Content- type:  application/json" -d
'{"username":"test@example.com","password":"w1secret$"}'
{"id":1,"password":null,"name":null,"age":null,"gender":null,"username":
"test1@example.com"}
```

Dan sekarang jika Anda mencoba lagi dengan pengguna yang sama:

```
curl -v -X POST localhost:8080/api/v1/public/users -H "Content- type:  application/json" -d
'{"username":"test@example.com","password":"w1secret$"}'
{"reason":"That resource already exists: Username already exists."}
```

Sebelum melanjutkan dengan lebih banyak contoh, izinkan kami memberi tahu Anda cara kerja struktur URL saat Anda menggunakan REST. Kami sudah membahas metodenya. Ini penting karena metode bersama dengan URL menentukan kumpulan operasi. Secara umum, ini adalah praktik yang baik bahwa URL Anda adalah kata benda dalam bentuk jamak ketika mereka menunjuk ke sumber daya yang sebenarnya, seperti pengguna, kursus, program, dll. Tidak ada konsensus dalam hal tindakan; beberapa menggunakan kata kerja, yang lain menggunakan garis bawah dan kata kerja (mis., _search, _start, _stop). Secara umum, kami menyukai pendekatan dengan garis bawah, karena dengan jelas menyatakan bahwa Anda melakukan beberapa operasi dan tidak berurusan dengan sumber daya secara umum. Mari kita tentukan tabel dengan operasi dan struktur URL dan hasilnya. Contohnya adalah dengan sumber daya pengguna, seperti yang ditunjukkan pada Tabel 5-1.

Tabel 5-1. Metode REST pada Sumber Daya pengguna

URL	GET	POST	PUT	PATCH	DELETE
../users	mengambil daftar pengguna (dilarang)	Membuat pengguna baru	Mengganti daftar pengguna dengan yang baru (tidak digunakan dan agak berbahaya)	Tidak digunakan	Menghapus koleksi pengguna
../users/1	Dapatkan pengguna dengan id 1	Tidak digunakan	mengganti data pengguna (dengan id 1) dengan yang baru	Mengubah data spesifik untuk pengguna dengan id 1	Menghapus user dengan id 1

Jangan terlalu stres sekarang! Anda akan mempelajarinya seiring kebutuhan Anda berkembang. Mari sertakan fungsionalitas PATCH. Apakah Anda ingat tentang hal itu? Ini adalah salah satu yang digunakan untuk memperbarui informasi entitas, tetapi hanya elemen yang kita inginkan (yang bukan nol). Jadi, panggilan ini harus dilakukan dengan pengguna yang sudah diautentikasi sehingga kita perlu melakukannya di cabang "aman" dari pohon URL kita. Buka file SecuredUserController.java dan mulai coding PATCH.

```
@RequestMapping(value= "/{id}", method = RequestMethod.PATCH)
public UserV1Dto patch(@AuthenticationPrincipal UserDetailsImpl user,
@PathVariable("id") Long id,
@RequestBody UserV1Dto userV1Dto) {
if (!id.equals(user.getId())) {
throw new ForbiddenServiceException("Not your user.");
}
 return transformationsV1.user2Dto(userService.patch(id,  userV1Dto. toUser()));
}
```

Mungkin Anda tidak menyadarinya, tetapi ada dua hal yang perlu diperhatikan dalam implementasi ini. Yang pertama adalah jika pengguna dalam sesi, mengapa kita harus melewati id? Sebenarnya kami tidak perlu melewatinya, tetapi dengan cara ini, Anda memenuhi standar REST. Jadi, kami sangat menganjurkan Anda untuk melakukannya seperti itu meskipun tidak diperlukan. Namun demikian, jika Anda benar-benar melakukannya, berhati-hatilah. Dalam kasus kami, kami menambahkan validasi untuk memeriksa apakah pengguna yang mencoba untuk diedit adalah pemilik sebenarnya dari sumber daya dan jika bukan itu masalahnya, kami hanya membuang pengecualian terlarang yang akan dipetakan ke kode kesalahan 403 (Terlarang). Dengan cara ini Anda membuat API Anda siap untuk masa depan. Ingat bahwa untuk saat ini kami hanya akan mengizinkan siswa, tetapi di masa mendatang, kami mungkin ingin mengaktifkan layanan pelanggan untuk mengedit beberapa informasi pengguna, dan dengan cara ini sudah setengah jadi!

Adapun sisa kode, tampaknya cukup jelas; kami hanya akan mengatakan bahwa Anda menentukan variabel (id dalam kasus ini) di dalam tanda kurung dan Anda menggunakan @PathVariable untuk menyuntikkan nilainya ke dalam variabel id Panjang. Jadi, bagaimana

dengan lapisan layanan? Untuk saat ini, kami hanya mengimplementasikan PATCH untuk nama pengguna, tetapi kami dapat memperluasnya ke semua atribut yang dapat diedit (id tentu saja tidak termasuk!). Berikan perhatian khusus pada kata sandi, yang membutuhkan pemrosesan tambahan dan bukan hanya penggantian belaka.

```
@Override @Transactional public User patch(Long id, User user) {
// Remember that if the user doesn't exist, this throws not found!
final User storedUser = get(id);
if (user.getName() != null) {
storedUser.setName(user.getName());
}
// Add more in the future!
return userRepository.save(storedUser);
}
```

Itu mudah, bukan? Mari kita coba!

```
curl -v -X PATCH localhost:8080/api/v1/secured/users/2 -H "Content-type: application/json" -H
"Authorization: bearer 35d8cda1-0f54-411e-a2a1b57c8cf42b75" -d '{"name": "test"}'
```

Kami baru saja mencoba memodifikasi pengguna yang bukan milik kami, jadi kami mendapatkan:

```
{"reason":"Forbidden: Not your user."}
```

Sekarang mari kita coba dengan pengguna kami:

```
curl -v -X PATCH localhost:8080/api/v1/secured/users/1 -H "Content-type: application/json" -H
"Authorization: bearer 35d8cda1-0f54-411e-a2a1b57c8cf42b75" -d '{"name": "test"}'
```

Kami mendapatkan pembaruan!

```
{"id":1,"password":null,"name":"test","age":null,"gender":null,"username":"
test@example.com"}
```

Seperti yang Anda lihat, namanya sekarang ''test''. Mari kita verifikasi apakah itu benar-benar bertahan.

```
curl -v -X GET localhost:8080/api/v1/secured/users/me -H "Content-type: application/json" -H
"Authorization: bearer 35d8cda1-0f54-411e-a2a1b57c8cf42b75"
```

Dan memang itu!

```
{"id":1,"password":null,"name":"test","age":null,"gender":null,"username":"
test@example.com"}
```

Kami harap Anda mendapatkan pemahaman tentang cara kerja berbagai hal. Sebagian besar kasusnya hampir sama. Kami bukan penggemar berat PUT; kami lebih suka mengkodekan semuanya dengan PATCH, tetapi mungkin ada beberapa kasus penggunaan. Bagaimanapun, metode PUT dapat menyebabkan bug jika Anda lupa mengimplementasikan sesuatu; semuanya akan ditimpa. Adapun DELETE, pada dasarnya sama — kami menerima id sebagai variabel jalur dan kami melakukan operasi, yang biasanya tidak berarti penghapusan aktual tetapi hanya penonaktifan. Kami telah menyiapkan rangkaian pengujian bagi Anda untuk mulai mengkodekan beberapa pengujian. Mari kita beralih ke topik itu sekarang!

5.16 Pengujian

Kami akan memiliki bab khusus yang terkait dengan pengujian di mana kami akan menjelaskan jenis pengujian yang paling umum dan kapan harus diterapkan. Namun demikian, masuk akal jika kita membahas beberapa pengujian pada kode yang baru saja kita tulis. Kami akan melakukan beberapa unit atau tes integrasi yang lebih baik, karena kami akan menggunakan sistem lain seperti database atau sistem otentikasi. Tes semacam ini biasanya dilakukan dengan memanggil serangkaian operasi dan menyatakan hasilnya dengan apa yang kita harapkan. Mari kita lihat contoh /me. Apa yang kami lakukan adalah mengautentikasi pengguna dan memanggil titik akhir /me, dan kami berharap pengguna yang dikembalikan adalah pengguna yang sama dalam sesi (yang kami autentikasi). Jadi, kami menulis yang berikut:

```
@Test
public void getUserInTheSessionTest() throws Exception {
final MockHttpServletResponse result = mockMvc.perform(get("/api/v1/
secured/users/me")
.header("Authorization",
"bearer" + getAccessToken())
.contentType(MediaType.APPLICATION_JSON))
.andReturn()
.getResponse();
assertThat(result.getStatus()).isEqualTo(HttpStatus.OK.value());
final UserV1Dto user = readJson(result.getContentAsString(), UserV1Dto.
class);
assertThat(user.getUsername()).isEqualTo(testUser.getUsername());
}
```

Pengguna uji diatur untuk setiap pengujian oleh kelas AbstractRestApiTest.java. Apa yang kami periksa adalah bahwa kode hasil HTTP bertipe OK, dan pengguna yang dikembalikan adalah pengguna sebenarnya yang kami gunakan dalam otentikasi. Bagaimana kami bisa menguji kreasi pengguna? Kita perlu mencakup beberapa hal, seperti membuat pengguna dengan semua data yang valid, data yang tidak valid seperti nama pengguna atau kata sandi yang tidak valid, dan bahkan kasus di mana kita mencoba mendaftarkan pengguna yang sudah ada. Berikut ini adalah kasus terakhir.

```
@Test
public void createAUserTwiceTest() throws Exception {
final MockHttpServletResponse result = mockMvc.perform(post("/api/v1/
public/users")
.contentType(MediaType.APPLICATION_JSON)
.content(writeJson(UserV1Dto.builder().withUsername("ex1@example.com")
.withPassword("123456a$").build())))
.andReturn()
.getResponse();
assertThat(result.getStatus()).isEqualTo(HttpStatus.CREATED.value());
```

```
final MockHttpServletResponse result2 = mockMvc.perform(post("/api/v1/
public/users")
.contentType(MediaType.APPLICATION_JSON)
.content(writeJson(UserV1Dto.builder().withUsername("ex1@example.com")
.withPassword("123456a$").build())))
.andReturn()
.getResponse();
assertThat(result2.getStatus()).isEqualTo(HttpStatus.CONFLICT.value());
}
```

Untuk panggilan pertama, kami mengharapkan status HTTP CREATED, tetapi untuk panggilan kedua kami mengharapkan kode kesalahan CONFLICT. Jangan ragu untuk menambahkan lebih banyak pengujian—kenalan dengan kerangka pengujian! Tes menulis itu menyenangkan dan akan mengamankan kode Anda untuk perubahan di masa mendatang. Ini adalah garis pertahanan pertama untuk kode Anda, jadi manfaatkanlah! Untuk menjalankan pengujian, Anda memerlukan database pengujian, tetapi jangan khawatir, kami telah menyiapkannya untuk Anda! Beralih di dalam modul liquibase dan jalankan

```
mvn liquibase:update -Ptest
```

Ini akan membuat database pengujian di folder /tmp Anda (sekali lagi, pengguna Windows, Anda mungkin perlu mengubahnya). Sekarang beralih kembali ke modul rest-api. Jika Anda menjalankan mvn test, ini akan menjalankan semua tes yang ada di modul, tetapi Anda juga dapat menjalankan tes dari satu kelas saja dengan menentukan nama kelas seperti berikut:

```
mvn test -Dtest=RestApiUserTest
```

Dan Anda bahkan dapat menentukan satu pengujian dengan menambahkan nama pengujian ke variabel pengujian

```
mvn test -Dtest=RestApiUserTest#getUserInTheSessionTest
```

5.17 Kesimpulan

Dalam bab ini kita membahas bagaimana aplikasi backend dapat terstruktur dan beberapa teknologi yang biasanya terlibat. Kami mulai dengan mendefinisikan lapisan basis data kami, kemudian lapisan layanan kami, hingga kami mencapai puncak, yang merupakan titik masuk ke aplikasi kami, rest-api. Kami memberikan beberapa contoh implementasi, sehingga Anda dapat berhubungan dengan pengembangan backend menggunakan kerangka kerja Spring untuk Java. Pada akhirnya, kami membuat beberapa tes integrasi untuk mencakup beberapa basis kode kami. Di bab berikutnya, kita akan mulai mengimplementasikan frontend aplikasi kita. Kami akan menghubungkannya dengan apa yang baru saja kami lakukan di backend, otentikasi, dan manipulasi sumber daya. Pantau terus!

BAB 6
PENGEMBANGAN FRONTEND

Dalam bab ini kita akan memakai topi pengembang frontend dan menghubungkan apa pun yang dilakukan perancang ke backend yang dikembangkan di bab sebelumnya. Pernahkah Anda menyadari bahwa setiap peran yang kita bicarakan sebenarnya menghubungkan sesuatu dengan sesuatu? Manajer produk menghubungkan kebutuhan dan pengembangan bisnis, perancang menghubungkan produk dan tujuan bisnis dengan kebutuhan pengguna, pengembang backend menghubungkan data ke produk, dan, akhirnya (sebenarnya, belum), pengembang frontend kami akan menghubungkan backend dan desain . Ya, pengembangan perangkat lunak adalah tentang menghubungkan beberapa blok.

Kami, insinyur perangkat lunak, bertujuan untuk menghubungkan orang dan teknologi. Beberapa orang melihat teknologi dan kemajuannya sebagai masalah, dan kami ingin menunjukkan bahwa itu sebenarnya adalah pemecah masalah. Mari kita lakukan bersama, langkah demi langkah. Di bab sebelumnya, kami menyiapkan di balik layar untuk data kami. Dalam bab ini, kita akan menggunakan titik akhir yang telah disiapkan untuk membawa data ke pengguna dan menampilkannya di antarmuka pengguna. Akan ada pemrograman di sini juga, jadi terimalah kenyataan bahwa tidak ada pembuatan perangkat lunak tanpa pengkodean dan bersiaplah untuk menulis beberapa kode. “Tapi saya menulis beberapa kode di bab sebelumnya,” beberapa orang mungkin berkata. Ya itu benar. Ini sedikit berbeda, namun. Perbedaan utama antara kode backend dan frontend terletak pada kenyataan bahwa Anda dapat langsung melihat hasil kode frontend langsung di browser Anda.

Catatan Jika Anda benar-benar yakin bahwa frontend bukan untuk Anda, Anda mungkin ingin melewati bab ini dan melanjutkan ke pengujian dan DevOps. Atau, mungkin Anda sudah menjadi pengembang frontend berpengalaman dan bab ini mungkin tampak agak mendasar bagi Anda, tetapi jika Anda berpikir bahwa pengkodean itu sulit, dan Anda tidak dibuat untuk ini dan bab sebelumnya sudah cukup pengkodean untuk Anda, silakan mengambil satu menit, mengambil napas dalam-dalam, dan mempertimbangkan kembali.
Dalam bab ini kita akan melalui proses yang menyenangkan untuk menunjukkan betapa mudahnya memprogram antarmuka dan membuatnya terlihat seperti yang Anda inginkan.
Penafian : Semua contoh kode yang muncul dalam bab ini telah diuji di browser Chrome, Safari, dan Firefox. Kami tidak menjamin sampel ini berfungsi di Internet Explorer.

6.1 Ayo Kode!

Buka editor teks favorit Anda dan ketik, “Halo teman saya! &hati;” Persis seperti ini. Sekarang simpan file ini sebagai hello.html.

Jika Anda menggunakan TextEditor di macOS, pilih Format -> buat teks biasa

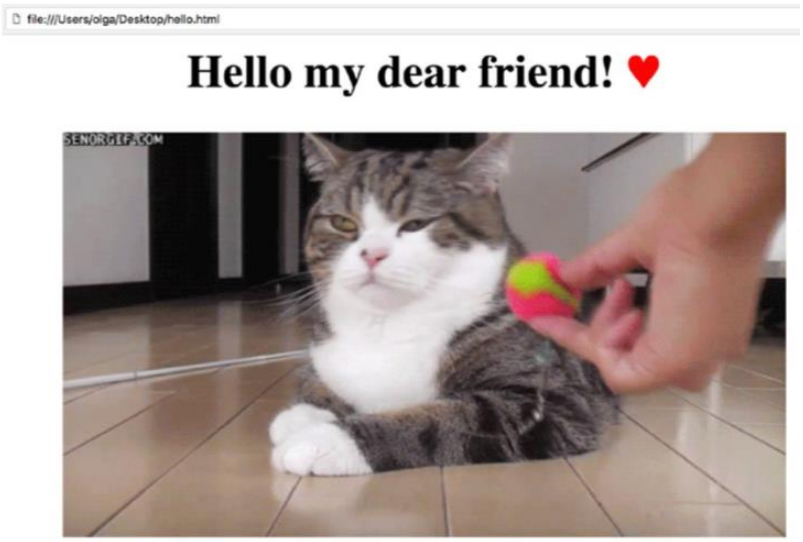
Setelah menyimpan file Anda, buka dengan klik dua kali. Anda akan melihat bagaimana jendela browser Anda terbuka, seperti yang ditunjukkan pada Gambar 6-1.

Gambar 6-1. File HTML pertama Anda dirender di browser

Apakah Anda ingin menemukan lebih banyak? Atau belum cukup meyakinkan? Buka file di editor lagi dan tambahkan beberapa hal:

```
<div style='text-align:center'>
<h1>Hello my dear friend!
<span style='color:red'>&hearts;</span>
</h1>
<img src=http://thecatapi.com/api/images/get?format=src&type=gif />
</div>
```

Simpan file, buka di browser Anda, dan Anda akan melihat sesuatu seperti Gambar 6-2 ini.



Gambar 6-2. Menambahkan warna pada teks, bermain dengan gambar, dan memusatkan konten

Setiap kali Anda me-refresh halaman, gambar kucing akan berubah. Inilah sebenarnya cara kerja internet—penuh dengan kucing, selebihnya hanyalah kebisingan. Agak frustasi harus me-refresh halaman setiap kali Anda ingin mengubah gambar kucing. Mari kita tambahkan baris kode terakhir. Buka file di editor teks Anda, dan tambahkan bagian berikut:

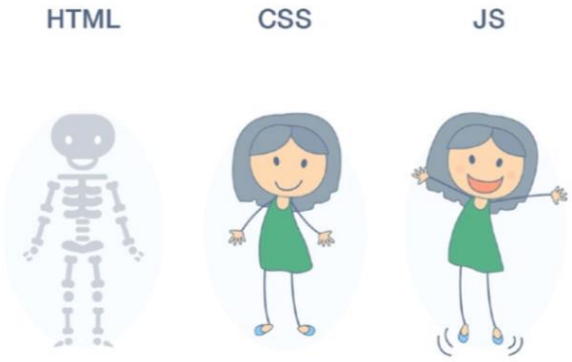
```
<div style='text-align:center'>
<h1>Hello my dear friend!
<span style='color:red'>&hearts;</span></h1>
<img src='http://thecatapi.com/api/images/get?format=src&type=gif' />
</div>
<script>
```

```
const src = 'http://thecatapi.com/api/images/get?format=src&type=gif'  setInterval(() => {
document.getElementsByTagName('img')[0].src = src + '&ts=' + Date.now()
}, 3000)
</script>
```

Anda dapat menyalin kode ini dari file beforeyouskip.html. Segarkan halaman di browser dan lihat. Anda akan melihat bahwa setiap 3 detik gambar kucing berubah. Apakah Anda masih bersama kami? Kami tahu kucing itu luar biasa, tetapi kami memiliki beberapa pekerjaan yang harus dilakukan. Hanya dengan beberapa baris kode, Anda dapat memusatkan hal-hal di halaman, menambahkan gambar dan beberapa warna ke teks, dan membuat hal-hal berubah di halaman tanpa harus menyegarkannya. Betapa menakjubkannya itu? Dan bahkan jika proses coding tidak cukup meyakinkan untuk membuat Anda terus membaca bab ini, saya yakin kucingnya. Jadi, mari kita coba memahami apa itu frontend dan di mana itu dimulai.

Di mana Frontend Mulai?

Kami telah membahas desain, kami juga telah menerapkan beberapa bagian backend di belakang layar. Di mana frontend dimulai? Sebenarnya, itu sudah dimulai. Sambil memikirkan desain, kami prihatin dengan bagaimana pengembang frontend akan mengimplementasikannya. Saat mengerjakan backend kami, kami sudah memikirkan bagaimana kami akan meneruskan data ke frontend. Jadi, apa itu frontend? Frontend adalah bagian yang terlihat dari aplikasi kita dan koneksi antara itu dan backend. Jadi, selain sebagai wajah aplikasi, frontend memiliki beberapa bagian tak kasat mata yang menghubungkan wajah ini dengan otak, yang terletak di backend. Selain itu, aplikasi frontend harus mendapatkan data dan menampilkannya dengan cara yang menarik dan dinamis. Apakah Anda ingat gambar yang kami buat untuk menjelaskan bagian-bagian frontend, seperti yang ditunjukkan pada Gambar 6-3?



Gambar 6-3. HTML, CSS, dan JS: Tiga pilar pengembangan frontend

Anda lihat bahwa pada dasarnya terdiri dari tiga bagian: HTML (hypertext markup language), CSS (cascading style sheets), dan JS (JavaScript). Apa semua ini? Kode HTML mendefinisikan struktur antarmuka kita; CSS mendefinisikan gayanya; dan JS mendefinisikan bagaimana perubahannya dari waktu ke waktu, dan JS bertanggung jawab untuk berkomunikasi dengan backend sambil menjamin bahwa semua perubahan yang terjadi pada data disebarkan ke antarmuka kita. Ini adalah tiga pilar pengembangan frontend. Tentu

saja, Anda mungkin pernah mendengar banyak hal seputar pengembangan web—banyak istilah seperti less, sass, react, redux, vue.js, bootstrap, desain material, tanpa server (uhh)—tetapi ingat, pada akhirnya browser benar-benar memahami dan dapat menginterpretasikan tiga hal ini: HTML, CSS, dan JavaScript. Pada dasarnya, browser mem-parsing kode HTML kita, menatanya dengan kode CSS kita, dan menafsirkan kode JavaScript kita untuk menerapkan apa pun yang kita perintahkan. Sebenarnya, Anda sudah menggunakan ketiganya di bagian sebelumnya! Lihat—struktur dokumen kita ditentukan oleh HTML:

```
<div>
<h1>Hello my dear friend! <span>&hearts;</span><h1>
<img src="...">
</div>
```

Penataan (warna, perataan, dll.) ditentukan oleh CSS:

```
<div style='text-align:center'>
<h1>Hello my dear friend! <span style='color:red'>&hearts;</span><h1>
<img src="...">
</div>
```

Dan, akhirnya, bagian dinamis halaman (mengubah gambar kucing) ditentukan oleh JavaScript:

```
<script> const src = 'http://thecatapi.com/api/images/get?format=src&type=gif'
setInterval(() => {
document.getElementsByTagName('img')[0].src=src + '&ts=' + Date.now()
}, 3000)
</script>
```

Jadi, Anda sudah mencoba semuanya. Selain itu, kami menemukan bahwa frontend dimulai pada tahap awal proses pengembangan. Anda mungkin bertanya pada diri sendiri sekarang: di mana dan kapan frontend berakhir? Yah, itu sebenarnya tidak pernah berakhir. Setelah aplikasi Anda dikirim ke produksi, Anda akan menemukan diri Anda dalam proses perubahan dan peningkatan yang konstan. Anda harus menjamin bahwa produk Anda sesuai dengan standar web modern, kebutuhan pengguna baru, dan tujuan bisnis baru. Keinginan utama Anda adalah untuk selalu berada di atas. Itulah mengapa proses pengembangan frontend untuk aplikasi Anda, serta proses apa pun yang terkait dengannya, tidak akan pernah berhenti kecuali Anda memutuskan untuk keluar dari produk Anda dan membiarkannya sendiri di dunia web yang liar. Mari kita bahas lebih detail masing-masing dari ketiga pilar tersebut.

6.2 Markup dan DOM

Markup adalah kerangka aplikasi kita. Markup mendefinisikan struktur halaman Anda dan placeholder untuk kontennya. Selama lebih dari beberapa dekade (dikembangkan pada tahun 1993), HTML telah digunakan untuk membuat halaman web dan aplikasi web. Singkatnya, dokumen HTML terdiri dari satu set tag, atributnya, dan konten untuk tag ini.

- **Tag HTML:** dapat dilihat sebagai wadah yang mendefinisikan semantik dari konten tertutup. Setiap tag ditulis dalam tanda kurung siku (mis., <p> - tag paragraf). Tag

dapat membuka dan menutup—misalnya, mendefinisikan awal paragraf sementara mendefinisikan akhir.

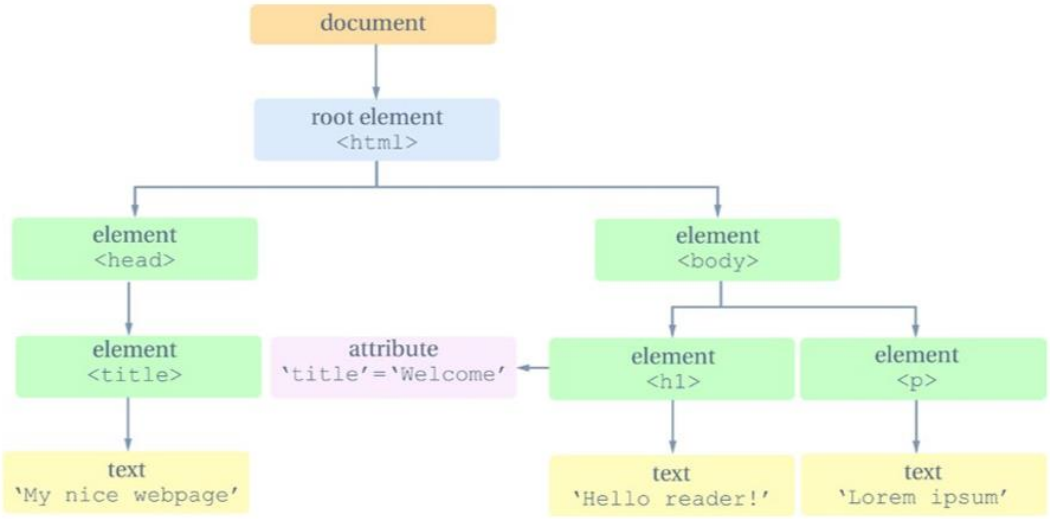
- **Atribut HTML:** beberapa tag dapat memiliki beberapa atribut yang memberi tahu dan mendefinisikan sedikit lebih banyak tentang wadah tag selain semantiknya. Atribut dapat memiliki nilai yang berbeda. Atribut ditulis di dalam tag pembuka, dan nilainya ditulis di dalam tanda kutip setelah tanda sama dengan. Misalnya, atribut judul dapat diterapkan ke tag apa pun dan akan membuat efek teks yang muncul saat Anda mengarahkan mouse ke elemen yang berisi judul ini (mis.,).
- **Konten tag HTML:** teks apa pun yang muncul di dalam tag adalah konten halaman Anda. Ini akan dirender dan ditampilkan oleh browser sebagai teks dengan semantik yang diterapkan dan gaya dasar dari tag pembungkus dan atributnya. Misalnya, Gambar 6-4 akan ditampilkan sebagai paragraf sederhana dengan judul yang muncul setelah Anda mengarahkan mouse ke atasnya: Hello reader!

Gambar 6-4. Bagaimana browser merender paragraf dengan atribut judul yang diterapkan

Ada banyak tag berbeda yang dapat digunakan untuk menyusun dokumen Anda. Kami telah berbicara tentang tag paragraf , Anda telah melihat tag untuk gambar <img>, tetapi sebelum kita melanjutkan ke beberapa tag lainnya, mari kita bicara tentang DOM.

6.3 Model Objek Dokumen

DOM (model objek dokumen) pada dasarnya adalah struktur yang dibuat oleh browser setelah membuka halaman web apa pun. Ini direpresentasikan sebagai pohon karena mesin render browser membaca kode markup dari atas ke bawah (Gambar 6-5).



Gambar 6-5. representasi pohon DOM

Dalam diagram ini Anda sudah dapat melihat bahwa ada beberapa elemen root yang disebut html dan ada elemen yang disebut head dan body. Mari kita bicarakan tentang mereka sebelum kita melanjutkan ke tag HTML lainnya. Jadi, elemen <html> membungkus seluruh halaman web Anda. Ini memberitahu browser bahwa dokumen ini adalah dokumen html, sehingga browser tahu persis bagaimana membaca dan menafsirkannya. Di dalam tag html ada dua wadah utama: <head> dan <body>. Tag <head> berisi semua meta data tentang halaman web Anda—judul, deskripsi, bahasa, data terkait pengoptimalan mesin telusur (SEO), jalur favicon, dll. Semua konten yang akan muncul di halaman Anda harus ada di dalam tag tubuh. Masuk akal, sebenarnya, kepala adalah untuk informasi deskriptif singkat, dan tubuh untuk yang lainnya. Dengan demikian, struktur dasar untuk file html tanpa konten akan terlihat sebagai berikut:

```
<html>
<head></head>
<body></body>
<html>
```

Perhatikan bahwa tag seperti kotak dengan konten di dalamnya. Kotak bisa berisi kotak lain, dan kotak itu juga bisa berisi kotak lain. Oleh karena itu, tag html berisi tag kepala dan tubuh, dan, pada gilirannya, tag tubuh akan berisi banyak tag lain yang mendefinisikan struktur dan konten halaman web.

6.4 Heading/Judul

Mari kita bicara tentang beberapa tag. Kami telah berbicara tentang tag paragraf: <p>. Konten di dalam tag p akan muncul sebagai paragraf baru. Jika Anda ingin menyorot judul halaman Anda, Anda dapat menggunakan tag heading. Tag heading dimulai dari heading terbesar, <h1>, dan berlanjut hingga heading terkecil, <h6>. Jadi, di antara Anda memiliki <h2>, <h3>, <h4>, dan <h5>. Coba buat konten dalam tag heading yang berbeda di file html Anda, dan periksa tampilannya di browser Anda:

```
<h1>Hello reader!</h1>
<h2>Hello reader!</h2>
<h3>Hello reader!</h3>
<h4>Hello reader!</h4>
<h5>Hello reader!</h5>
<h6>Hello reader!</h6>
```

Ini akan terlihat seperti yang ditunjukkan pada Gambar 6-6.

Gambar 6-6. Tag heading HTML dari h1 hingga h6

6.5 Hyperlink

Tentu saja, tidak ada internet tanpa hyperlink. Apakah Anda menggunakan Wikipedia? Pernahkah Anda terjebak dalam situasi di mana Anda mencari, katakanlah, kucing siam dan berakhir setelah 5,5 jam mengetahui segala sesuatu tentang teknologi blockchain dan dampak legalisasi ganja? Hyperlink Wikipedia itu jahat. Mari kita lihat tag HTML apa yang digunakan untuk menghasilkan kejahatan ini. Tag tersebut disebut anchor dan ditulis sebagai <a>. Apakah Anda ingat tentang atribut? Tag ini hanya berfungsi jika Anda memberikan atribut yang disebut href padanya (href berarti referensi hypertext). Masuk akal, bukan? Jika kita ingin mengirim orang itu ke suatu tempat, kita harus memberi mereka referensi ke mana harus pergi. Isi sebenarnya dari tag adalah teks yang ingin Anda tampilkan di halaman sebagai tautan. Katakanlah kita ingin mengirim orang tersebut ke situs web Google dan kita ingin kata "google" muncul sebagai hyperlink di halaman tersebut. Maka kode markup akan terlihat seperti berikut:

```
<a href="https://www.google.com">Google</a>
```

Anda juga dapat mereferensikan halaman situs web Anda sendiri, memberikan jalur relatif ke halaman tersebut.

Jalurnya bisa agak relatif atau absolut. Jalur absolut berarti Anda menyediakan seluruh jalur mulai dari akar server Anda (atau lainnya), dan jalur relatif berarti Anda menyediakan jalur relatif ke dokumen saat ini.

Jadi, misalnya, jika kita memiliki halaman bernama hyperlink1.html dan ingin merujuk seseorang ke halaman lain bernama hyperlink2.html yang terletak di jalur yang sama, kita akan melakukan hal berikut:

```
<a href="hyperlink2.html">Go to the second page</a>
```

Periksa halaman html hyperlink1.html dan hyperlink2.html di dalam folder kode untuk bab ini. Secara default, tautan dibuka di tab yang sama dengan tempat Anda berada. Jika Anda ingin tautan terbuka di tab baru, Anda dapat memberikan atribut lain yang disebut target dengan nilai _blank:

```
<a href="https://www.google.com" target="_blank">Google in new tab</a>
```

6.6 Gambar

Sama seperti internet tidak dapat eksis tanpa hyperlink, internet tidak dapat eksis tanpa gambar. Anda telah menggunakan tag gambar; itu ditulis sebagai <img>. Itu harus memiliki atribut yang disebut src (sumber gambar), tetapi tidak mengandung konten apa pun, karena kontennya sebenarnya ditentukan oleh sumber gambar. Karena semua tag html harus dibuka dan ditutup, ia akan menutup sendiri. Tag semacam itu disebut tag "menutup sendiri". Jadi, ditulis sebagai:

```
<img src="http://via.placeholder.com/350x150" />
```

Seperti atribut href dari tag jangkar, atribut src juga dapat memiliki nilai absolut atau relatif. Jadi, Anda dapat meletakkan gambar di dekat file html Anda dan memberikan jalur relatif ke gambar tersebut di atribut src Anda. Misalnya, jika kita meletakkan gambar bernama cat.png di dalam folder img, kita dapat menampilkan gambar ini menggunakan tag img seperti ini:

```
<img src="img/cat.png" />
```

Secara default, gambar akan muncul dalam ukuran aslinya. Terkadang itu tidak terlalu tepat karena ukuran gambar mungkin tidak sesuai dengan tata letak halaman kita. Untuk mengatasi masalah ini, spesifikasi html memungkinkan Anda menggunakan atribut lebar dan tinggi yang diterapkan pada tag img. Jika Anda hanya menentukan salah satunya, yang lain akan menyesuaikan skalanya. Ukuran dapat ditentukan dalam nilai absolut (misalnya, piksel):

```
<img src="img/cat.png" width="500px" />
```

atau dalam nilai relatif, dalam persentase (ini akan menjadi persentase relatif terhadap elemen induk). Jika gambar kami adalah anak langsung dari tag tubuh maka lebar relatif akan relatif terhadap tubuh dan menempati bagian yang tepat dari layar yang kami tunjukkan:

```
<img src="img/cat.png" width="40%" />
```

Periksa kode dalam file bernama image.html. Cobalah untuk mengubah ukuran halaman browser. Anda akan melihat bahwa sementara gambar yang lebarnya ditentukan dalam piksel selalu tetap sama, yang lain, yang ukurannya ditentukan dalam persentase akan menyesuaikan sesuai dengan ukuran browser.

Ketika elemen halaman web dengan mudah menyesuaikan dengan ukuran perangkat tempat mereka dibuka, itu berarti mereka responsif dan adaptif.

Responsif dan adaptif adalah konsep yang sangat penting dari web saat ini. Ada begitu banyak perangkat, dan Anda ingin produk Anda berjalan di semuanya. Atau mungkin tidak? Itulah mengapa sangat penting untuk memahami pengguna Anda. Anda harus mengetahui dengan baik perangkat apa yang paling sering digunakan dan memastikan bahwa produk Anda sempurna di perangkat tersebut. Saat ini, perangkat lunak sedang dibangun berdasarkan paradigma “mobilefirst”, yang berarti bahwa prioritas nomor satu adalah memastikan bahwa produk akan muncul dengan baik di perangkat seluler. Hal ini dapat dimengerti—orang-orang saat ini lebih banyak menggunakan perangkat seluler daripada perangkat desktop.

6.7 Formulir

Sejak kita mulai berbicara tentang pengguna, mari kita pikirkan tentang interaksi dan komunikasi dengan mereka. Sering kali selain menampilkan beberapa konten kepada pengguna Anda, Anda ingin menerima beberapa data dari pengguna atau konfirmasi. Saat saya menulis kata-kata ini, sekarang tanggal 25 Mei 2018, yang berarti bahwa IDN-GDPR (Peraturan Perlindungan Data Umum Indonesia) telah dimulai hari ini. Jangan meremehkan kekuatan undang-undang ini saat meminta data dari pengguna Anda! Tapi sebelum Anda berpikir tentang undang-undang, Anda ingin tahu bagaimana Anda menggunakan HTML untuk meminta data dari pengguna Anda, bukan? Spesifikasi HTML memiliki cara untuk melakukannya dengan menggunakan tag <form>. Di dalam tag formulir Anda dapat memiliki banyak input yang meminta data pengguna. Mari kita bicara tentang bidang input ini. Tag input (<input>) juga merupakan tag penutup sendiri yang dapat memiliki beberapa atribut yang menentukan jenis bidang input yang tepat. Secara default, dan Anda bahkan tidak perlu memberikan atribut apa pun untuk ini, inputnya akan berupa teks. Ini adalah input umum di mana Anda dapat menulis teks apa pun:

```
<input/>
```

Ini akan dirender di halaman web seperti yang ditunjukkan pada Gambar 6-7.

Gambar 6-7. Tag masukan aktif dan tidak aktif

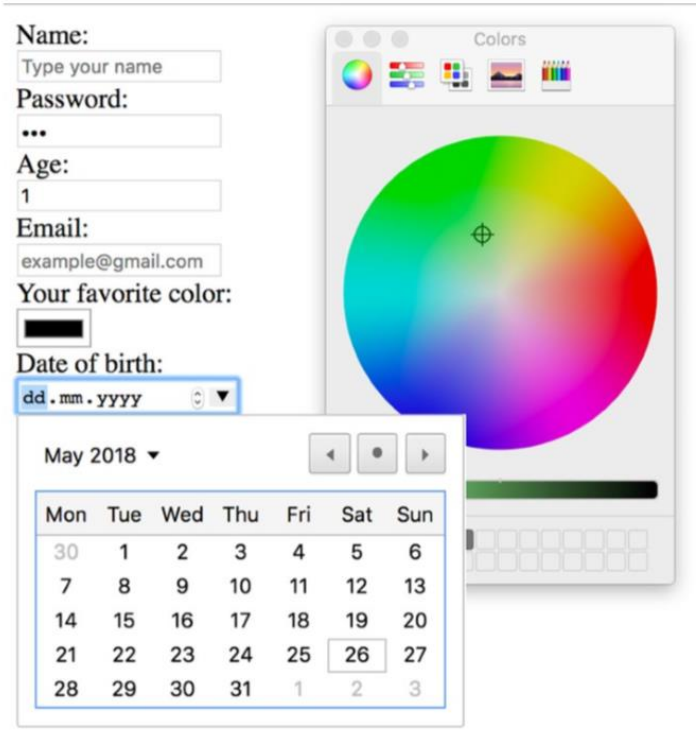
Dimungkinkan untuk memiliki tipe input yang berbeda dengan memberikan atribut yang disebut tipe ke tag input. Nilainya bisa sangat berbeda, misalnya:

- kata sandi
- nomor
- tanggal
- surel
- kotak centang
- jangkauan
- Cari

Misalnya, formulir ini:

```
<form action="">
Name: <input type="text" placeholder="Type your name" />
Password: <input type="password" />
Age: <input type="number" />
Email: <input type="email" placeholder="example@gmail.com" />
Your favorite color: <input type="color" placeholder="#eee" />
Date of birth: <input type="date" />
</form>
```

akan dirender seperti yang ditunjukkan pada Gambar 6-8.



Gambar 6-8. Formulir dengan bidang input dari berbagai jenis

Periksa kode dengan berbagai jenis bidang input dalam file form.html. Tentu saja, sekarang Anda bertanya pada diri sendiri, “Oke, dan apa yang harus saya lakukan dengan semua data ini?” Itu adalah pertanyaan yang wajar, dan pada titik ini HTML biasa saja tidak cukup. Entah bagaimana, kami perlu mengumpulkan data ini dan memberikannya ke backend kami untuk diproses, disimpan, atau mengambil beberapa tindakan. Mari kita kembali ke ini nanti. Selain memberikan beberapa semantik ke elemen halaman Anda, Anda ingin memberi halaman Anda beberapa struktur dan menentukan beberapa divisi penting. Itulah mengapa spesifikasi HTML menawarkan elemen yang sangat penting yang disebut . Tag div berarti pembagian. Tag ini tidak memiliki gaya apa pun yang melekat padanya, tetapi sangat berguna untuk memisahkan potongan konten satu sama lain dan menatanya secara berbeda.

Elemen Sebaris dan Blokir

Anda mungkin telah memperhatikan bahwa beberapa elemen muncul di baris yang sama, sementara beberapa di antaranya muncul di baris berikutnya. Karakteristik yang sangat penting yang dimiliki semua elemen berkaitan dengan posisi relatif terhadap elemen sebelumnya. Itu bisa salah satu dari dua ini:

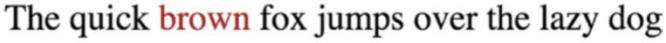
- Elemen sebaris: muncul di baris yang sama tempat mereka dipanggil. Misalnya, <a>, <img />, adalah elemen sebaris.
- Elemen blok: menempati baris penuh dan elemen berikutnya akan dirender dari baris baru. , <p>, elemen heading (<h1>–<h6>) adalah elemen blok.

Anda mungkin telah memperhatikan bahwa kita berbicara tentang elemen span sebagai contoh elemen inline. Elemen span adalah tag kecil dan tidak terlihat yang tidak memberikan konteks atau gaya tertentu. Namun demikian, ini adalah tag yang sangat

penting, seperti div, karena dapat digunakan sebaris untuk menata konten yang dibungkus secara berbeda dari baris lainnya. Sebagai contoh,

```
<div>The quick <span style="color: brown">brown</span> fox jumps over the lazy dog</div>
```

Bagian kode ini akan dirender seperti yang ditunjukkan pada Gambar 6-9.



Gambar 6-9. Elemen rentang digunakan untuk menata satu kata di dalam frasa lengkap

Kita dapat berbicara banyak tentang HTML, spesifikasinya, berbagai tag (yang belum kita bahas, misalnya, tag daftar <ul> dan <ol> atau tag media seperti <video> atau <audio>), atributnya, dan berbagai jenis sihir yang mungkin dicapai dengan menggunakan bahasa markup yang sederhana namun kuat ini, tetapi kami percaya bahwa kami menyediakan informasi yang cukup bagi Anda untuk dapat mencari informasi yang Anda minati mengenai topik ini. Mari kita periksa dengan latihan sederhana ini.

6.8 Uji Diri Anda

Buat halaman html sederhana yang berisi berikut ini:

- tajuk yang menyebutkan nama Anda
- Gambar Anda
- Beberapa paragraf tentang diri Anda
- tautan ke profil sosial Anda (mis., linkedin, github) atau hanya tautan ke https://www.apress.com/ jika Anda tidak memiliki akun sosial
- Tambahkan formulir sederhana yang menanyakan nama dan email orang yang membaca profil Anda dan tombol kirim

Pada akhirnya, halaman Anda akan terlihat mirip dengan Gambar 6-10 ini.

Gambar 6-10. Buat markup untuk menghasilkan hasil yang diberikan serupa

Anda dapat menemukan kode untuk halaman ini di dalam file latihan.html.
Sekarang setelah Anda ahli dalam membuat struktur HTML untuk halaman web Anda, mari mulai menatanya!

6.9 Gaya

Tentu saja, kami tidak dapat menikmati halaman web kami jika itu hanya teks hitam biasa, meskipun itu terstruktur dengan baik ke dalam paragraf, divisi, jangkar, dan komponen bagus lainnya. Kita perlu menyesuaikan warna, font, ukuran, perataan, pemosisian, latar belakang, dan banyak hal lainnya untuk membuat halaman kita menarik untuk dilihat dan dapat dijalankan di browser apa pun dari perangkat apa pun. Di sinilah CSS berguna, dan inilah yang akan kita *bicarakan di bagian ini selain berbagai jenis tata letak dan sistem desain siap pakai.*

CSS adalah bahasa yang digunakan untuk menjelaskan aturan gaya untuk markup hTml. Istilah cascading hanya mendefinisikan skema prioritas ketika lebih dari satu aturan gaya cocok dengan elemen yang sama.

Aturan penataan gaya ditulis dalam format sederhana—nama aturan diikuti oleh titik dua dan nilainya. Jadi, misalnya, aturan untuk warna teks merah akan terlihat sebagai berikut:

```
Color: red;
```

Tentu saja, aturan harus diterapkan pada sesuatu; mereka tidak bisa begitu saja terbang di udara. Aturan dapat diterapkan pada beberapa elemen tertentu atau pada

kelompok elemen. Ini tergantung pada cara Anda memutuskan untuk menentukan aturan tersebut. Pada dasarnya ada tiga cara untuk menentukan aturan CSS:

- Inline: aturan CSS dilampirkan ke elemen tertentu di dalam atribut tag yang disebut gaya: Hello! Here's the red paragraph
- Lembar gaya internal: Aturan CSS ditulis di dalam tag gaya di dalam elemen kepala file html Anda. Aturan harus ditulis di dalam blok yang ditentukan oleh pemilih. Misalnya, untuk menerapkan aturan warna: merah ke semua paragraf, Anda harus menulis yang berikut:

```
<style>
p {
color: red;
}
</style>
```

- Stylesheet eksternal: aturan CSS ditulis dengan cara yang sama seperti yang ditulis di dalam tag gaya, hanya di file eksternal yang memiliki ekstensi .css dan diimpor ke file html menggunakan tag khusus <link> yang berisi jalur ke CSS file di dalam atribut href: <link rel="stylesheet" href="style.css">

Apa kelebihan dari masing-masingnya? Jika Anda hanya perlu mengubah satu elemen tertentu, Anda dapat menggunakan gaya inline. Jika Anda perlu menentukan gaya yang serupa untuk grup elemen yang berbeda, tetapi tidak terlalu banyak gaya, Anda dapat membuatnya secara internal dalam file yang sama di dalam tag <style>, tetapi jika Anda memiliki lembar gaya kompleks besar yang menjelaskan gaya atau animasi yang berbeda untuk kelompok elemen yang berbeda, Anda harus memilih lembar gaya eksternal. Periksa file inline-style.html, internal-style.html, dan external-style.html untuk masing-masing dari tiga jenis penggunaan stylesheet. Anda mungkin telah memperhatikan bahwa ketika kami menentukan aturan gaya di dalam tag <style> atau di file eksternal, kami membungkus aturan di blok elemen terkait yang harus diterapkan gaya:

```
p {
color: red;
font-size: larger;
}
```

Apakah ini berarti kita hanya dapat menentukan gaya untuk tag? Bagaimana jika kita tidak ingin semua paragraf terlihat sama? Sebenarnya, blok yang dapat kita tentukan aturan gayanya bisa sangat rumit. Blok ini disebut pemilih, dan ini adalah salah satu konsep terpenting dari pengembangan web.

Selector mendefinisikan grup elemen yang ingin kita terapkan beberapa gaya tertentu. Hal ini dapat dijelaskan dengan tag, kelas, id, atau campuran dari mereka.

Ternyata elemen html dapat memiliki id dan kelas yang dilampirkan, dan ini dapat digunakan serta pemilih CSS. Mari kita bicara tentang mereka.

- Id elemen adalah atribut (id), yang nilainya harus mengidentifikasi elemen secara unik. Sebagai contoh : This is my first paragraph

- Kelas elemen adalah atribut yang dapat mengidentifikasi lebih dari satu elemen, biasanya digunakan ketika Anda perlu menata atau memilih lebih dari satu elemen dengan cara yang sama. Sebagai contoh : This paragraph contains some important information

Cara kita menulis penyeleksi untuk kelas dan id berbeda dengan cara kita menulis penyeleksi untuk tag. Jadi:

- Pemilih untuk tag hanyalah nama tag:
  - p—memilih tag paragraf
  - img—memilih tag img
  - div img—memilih tag img di dalam blok div
- Selektor untuk id elemen adalah nilai id yang diawali dengan #:
  - #first—memilih elemen dengan id “pertama”
- Selector untuk kelas ditulis sebagai nama kelas yang diawali dengan titik:
  - .important—memilih semua elemen yang memiliki kelas “penting”
  - p.important—memilih semua paragraf yang memiliki kelas “penting”
  - div img.important—memilih semua gambar yang ada di dalam div dan memiliki kelas penting
  - #first .important—memilih semua elemen di dalam elemen dengan id “first” yang memiliki kelas “important”

Periksa lebih lanjut tentang penyeleksi di halaman w3c: https://www.w3schools.com/cssref/css_selectors.asp.

Cek juga kodenya di file selectors.html. Sekarang setelah Anda mengetahui cara memilih elemen dan menulis blok yang akan berisi aturan CSS, mari kita bicara tentang aturan itu sendiri. Aturan apa yang bisa kita terapkan? Apa yang bisa kita lakukan dengan CSS? Saya pikir saat ini Anda hanya bisa bertanya, “Apa yang tidak bisa kita lakukan dengan CSS?” karena CSS sangat kuat saat ini! Jika Anda google dengan "proyek yang dibangun hanya dengan CSS," Anda akan benar-benar kagum. Tentu saja, dalam buku ini kita tidak akan menggambar potret Mona Lisa dengan CSS. Kami hanya akan berbicara tentang aturan umum. Anda akan melakukan semua sisanya, oke?

Mari kita mulai dari dasar. Warna. Untuk mengubah warna teks, Anda menerapkan aturan "warna" dengan nilai apa pun yang Anda butuhkan untuk itu. Nilai untuk warna dapat ditentukan dalam format yang berbeda:

- Nama warna itu sendiri (mis., black, red, yellow, dll.)
- Kode warna heksadesimal (mis., #000000, #ff0000, dll.)
- Spesifikasi rgb (mis., rgb(0, 0, 0), rgb(255, 0, 0))

Dengan cara yang sama, Anda dapat mengubah warna latar belakang. Anda lebih suka menggunakan aturan css "warna latar" atau aturan "latar belakang". Perbedaan antara keduanya adalah bahwa sementara aturan warna latar digunakan untuk mendefinisikan warna latar belakang secara eksplisit, aturan latar belakang dapat berisi lebih banyak properti—misalnya, gambar latar belakang. Setiap bagian yang dapat ditentukan dalam aturan latar belakang juga dapat ditentukan di properti individu, seperti "gambar latar", "posisi latar belakang", dll. Periksa lebih lanjut tentang aturan CSS latar belakang di halaman

w3schools: https://www.w3schools.com/cssref/css3_pr_background.asp. Dimungkinkan juga untuk menentukan beberapa gradien untuk latar belakang Anda. Sebagai contoh,

```
#gradient {
background: linear-gradient(blue, yellow);
}
```

Mari kita bicara tentang ukuran. Anda dapat menentukan ukuran untuk elemen Anda menggunakan CSS dan properti lebar dan tingginya. Nilai untuk properti ini bisa absolut atau relatif. Sebagai contoh,

```
div {
height: 200px;
width: 20%;
}
```

Mari kita bicara tentang font. Font adalah bagian khusus dari pengembangan web di mana dimungkinkan untuk berbicara berjam-jam karena tipografi sebenarnya adalah sesuatu yang cukup banyak berkorelasi dengan seluruh konsep desain dan bahkan arsitektur. Berbagai jenis tipografi digunakan untuk menyampaikan berbagai jenis pesan, dan ada beberapa font khusus yang kami kaitkan dengan beberapa hal tertentu. Misalnya, Anda tidak akan menggunakan font yang sama untuk membuat poster punk seperti yang Anda gunakan untuk menulis makalah ilmiah. Jarak yang berbeda antara paragraf dan huruf mengirimkan pesan yang berbeda. Lebar garis yang berbeda digunakan untuk kebutuhan yang berbeda—misalnya, artikel penting di surat kabar akan menggunakan lebar garis yang berbeda dari, katakanlah, dokumen CV online. CSS menyediakan banyak properti untuk dimainkan dengan font, jenis, dan karakteristiknya. Properti terkait font adalah sebagai berikut:

- ***font-family:*** tentukan font family, berikan beberapa di antaranya untuk fallback. Sebagai contoh,

```
p {
font-family: "Times New Roman", Times, serif;
}
```

- ***font-size:*** menentukan ukuran font. Sekali lagi, Anda dapat menentukan ukuran font dalam satuan absolut (misalnya, piksel) atau dalam satuan relatif (misalnya, dalam em). 1em setara dengan ukuran font saat ini yang biasanya 16px.

```
p {
font-size: 1.5em;
}
```

- ***font-style***: bisa agak miring, miring, atau normal:

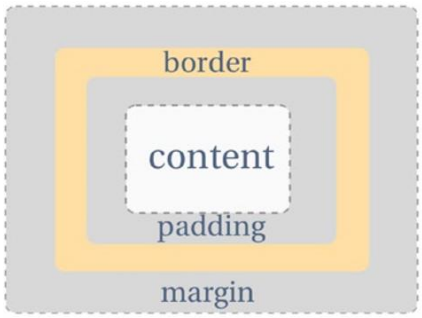
```
p {
font-style: italic;
}
```

- ***font-weight:*** tentukan berat font Anda:

```
p {
font-weight: bold;
}
```

Selain pemformatan font untuk mengubah tampilan teks, CSS memungkinkan penerapan banyak aturan pada teks itu sendiri. Properti seperti penspasian huruf, perataan vertikal, arah, tinggi garis, identitas teks, dan transformasi teks hanyalah beberapa properti yang dapat diterapkan pada teks. Periksa lebih detail tentang mereka di halaman w3schools: https://www.w3schools.com/css/css_text.asp.

Hal lain yang sangat penting untuk dijelaskan tentang CSS adalah model kotaknya, yang ditunjukkan pada Gambar 6-11. Setiap elemen seperti kotak, dan ukurannya terdiri dari ukuran konten, margin, padding, dan batasnya.



Gambar 6-11. model kotak CSS

Jadi, jika Anda menentukan lebar div Anda sebagai 200px dan menambahkan properti margin, padding, atau border, Anda harus menjumlahkannya untuk mendapatkan lebar akhir elemen. Mari kita bicara tentang tiga properti ini:

- Margin: mendefinisikan ruang di luar elemen. Properti ini terdiri dari nilai margin atas, kanan, bawah, dan kiri:

  margin: 20px 10px 10px 20px;

  Jika Anda hanya menentukan satu nilai, elemen akan memiliki margin yang sama untuk keempat sisinya:

  margin: 10 px;

  Anda juga dapat menentukan setiap sisi secara terpisah, menggunakan properti margin-top, margin-right, margin-bottom, dan margin-left.
- Border: mendefinisikan perbatasan di sekitar elemen. Itu bisa putus-putus, putus-putus, padat, bulat. Garis perbatasan dapat memiliki ketebalan yang berbeda. Anda juga dapat menentukan batas yang berbeda untuk sisi elemen yang berbeda, dan, tentu saja, Anda dapat menentukan warna untuk batas tersebut.

  Border: dashed gray 2px;
- Padding: mendefinisikan ruang interior antara batas elemen dan isinya. Cara Anda mendefinisikan padding sangat mirip dengan margin:

  Padding: 20px 10px 10px 20px

  Atau

  Padding: 10 px;

Centang file box-model.html untuk bermain dengan padding, margin, dan border. Di bagian ini, kita telah membahas dasar-dasar CSS, seperti model kotak dan gaya teks dan warna. Masih banyak lagi yang dapat Anda lakukan dengan CSS—misalnya, animasi. Oh ya,

Anda bisa melakukan hal-hal gila hanya dengan menggunakan aturan CSS. Kami akan meninggalkannya untuk pembelajaran pribadi Anda; Anda akan menemukan seluruh sumber daya internet tentang animasi CSS. Mari kita bicara tentang topik yang sangat penting dalam hal mengimplementasikan antarmuka pengguna: tata letak.

6.10 Tata Letak

Salah satu tantangan terpenting CSS adalah menyediakan cara mudah bagi pengembang dan desainer untuk mendistribusikan konten mereka di halaman dengan cara yang terlihat bagus dan tidak merusak ukuran layar yang berbeda. Sangat jelas bahwa Anda tidak ingin konten Anda didistribusikan secara berurutan di atas halaman dari atas ke bawah. Anda ingin memiliki beberapa kisi, beberapa kotak, beberapa struktur yang menarik. Sebelum CSS yang bagus dan mewah saat ini, orang akan menyusun konten mereka menggunakan tabel (neraka programmer!). Setelah itu, orang akan membuat tata letak dan tata letak yang benar-benar diposisikan berdasarkan properti float CSS; "float: left" akan memberi tahu elemen untuk menempel di sisi kiri elemen induk, sementara "float: right" akan memberi tahu elemen untuk menempel di sisi kanan. Dikombinasikan dengan lebar relatif, itu akan memungkinkan membangun tata letak yang kurang lebih fleksibel. Sekarang kita berbicara tentang tata letak fleksibel, saatnya berbicara tentang flexbox—alat CSS populer yang memungkinkan penyusunan tata letak responsif fleksibel menggunakan properti yang mudah dipahami.

> ***Flexbox (kotak fleksibel) adalah cara memposisikan elemen di dalam wadah secara berurutan atau dalam kolom dalam urutan tertentu dan dengan cara yang sesuai. Flexbox menjamin bahwa meskipun ukuran elemennya dinamis, tata letaknya tetap fleksibel untuk perangkat apa pun. Untuk menggunakan flexbox, terapkan properti display: flexbox ke container: .container {display: flex;}***

Di dalam wadah flexbox, elemen dapat ditampilkan dalam kolom atau baris. Anda dapat menentukannya dengan menerapkan properti "flex-direction" dengan kolom atau baris nilai. Anda dapat menyelaraskan item dengan cara apa pun yang Anda suka (menggunakan properti align-items), dan Anda dapat memberi tahu elemen untuk dibungkus saat diperlukan; ini berguna untuk tata letak responsif. Ada banyak lagi tentang flexbox, coba mainkan sedikit menggunakan panduan ini: https://css-tricks.com/snippets/css/a-guide-to-flexbox/. Belum lama ini, nilai lain untuk properti tampilan datang untuk membantu flexbox: grid.

Tata letak kotak adalah alat yang ampuh untuk meletakkan konten halaman web yang beroperasi dalam ruang 2 dimensi. Anda menentukan aturan untuk kolom dan baris untuk tata letak Anda. Untuk memberi tahu wadah bahwa itu akan menjadi kisi, gunakan yang sama seperti yang Anda gunakan untuk flexbox, cukup ganti nilai flex untuk properti display dengan grid:.container {display: grid;}

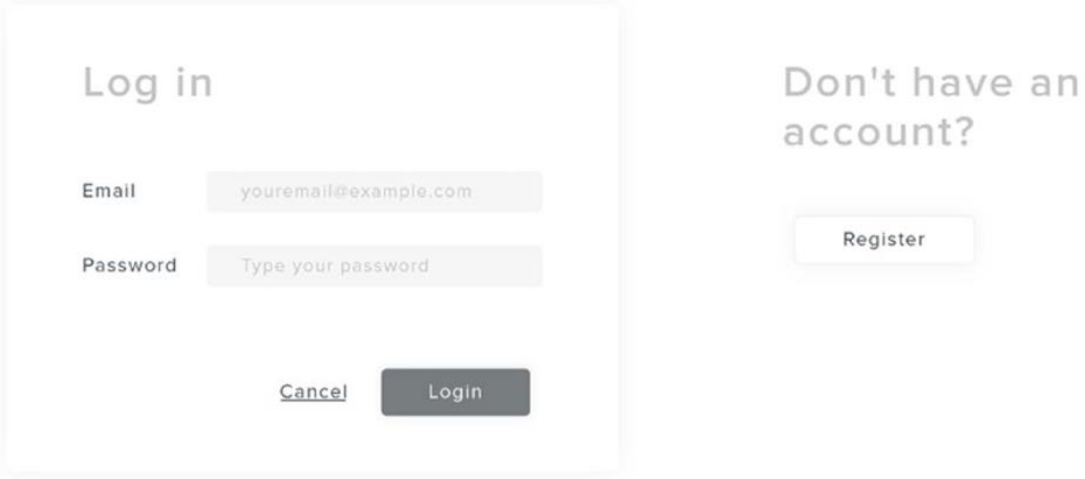
Perhatikan bahwa kami mengatakan bahwa grid datang untuk membantu flexbox, bukan untuk menggantinya. Kedua alat ini dapat saling melengkapi untuk mencapai tata letak responsif fleksibel yang kuat. Perbedaan utama di antara mereka adalah bahwa grid

beroperasi dalam ruang 2 dimensi, sedangkan flexbox sangat bagus untuk ruang unidimensional di mana kita membutuhkan kolom atau baris untuk memiliki perilaku yang diperlukan. Grid adalah sesuatu yang lebih banyak beroperasi pada tingkat tata letak, sedangkan flexbox lebih banyak beroperasi pada tingkat konten.

Flexbox mendengarkan konten dan menyesuaikannya, sementara grid pada dasarnya menentukan strukturnya. Oleh karena itu, grid harus digunakan untuk mendefinisikan tata letak skala besar, sedangkan flexbox dapat digunakan pada tingkat skala yang lebih kecil untuk elemen fleksibel. Bermain dengan keduanya. Panduan untuk kisi dapat ditemukan di sini: https://css-tricks.com/snippets/css/complete-guide-grid/. Sekarang Anda tahu banyak tentang menggunakan CSS untuk tidak hanya menentukan warna, ukuran, dan font, tetapi juga untuk memposisikan elemen pada halaman. Saya rasa inilah saatnya untuk mengkonsolidasikan pengetahuan Anda dan menguji diri Anda dengan latihan sederhana ini.

6.11 Latihan CSS

Apakah Anda ingat maket platform pembelajaran kami? Apakah Anda ingat halaman untuk login dan mendaftar? Gunakan apa yang baru saja kita pelajari untuk mengubah html di dalam latihan2.html menjadi seperti yang ditunjukkan pada Gambar 6-12.



Gambar 6-12. Layar masuk dengan gaya yang diterapkan

- Terapkan pengetahuan Anda tentang flexbox dan sistem grid untuk membangun tata letak dua kolom untuk konten utama dan untuk menampilkan label dan elemen input seperti yang ditampilkan pada mockup.
- Terapkan pengetahuan Anda tentang margin, padding, dan border untuk membuat wadah login, elemen input, dan tombol terlihat seperti pada mockup.
- Terapkan pengetahuan Anda tentang gaya warna dan latar belakang untuk mewarnai semua teks dan latar belakang persis seperti tampilannya di mockup.

Periksa diri Anda di latihan2-solusi.html.

Saya harap Anda menikmati menjadi insinyur UI dan membuat antarmuka pengguna sesuai dengan mockup yang dibuat oleh desainer kami. Sekarang mari kita lihat bagaimana teknologi saat ini dapat membantu kita dalam mengungguli tugas-tugas ini.

6.12 Sistem Desain

Singkatnya, sistem desain adalah seperangkat komponen HTML yang telah ditentukan sebelumnya dengan aturan CSS yang sudah diterapkan yang mendefinisikan semua desain Anda, mulai dari skema warna produk Anda dan menentukan kelas untuk meletakkan kontennya. Setelah sistem desain ditentukan, Anda hanya menikmati proses penyusunan perangkat lunak Anda tanpa khawatir tentang menentukan gaya, elemen pemosisian, menulis aturan CSS yang rumit, dan membuat animasi untuk transisi. Jika Anda ingin membangun sistem desain Anda sendiri untuk produk Anda, pujian untuk Anda dan semoga berhasil. Jika Anda, seperti kami, tidak memiliki cukup sumber daya dan kesabaran untuk itu, maka Anda dapat menggunakan salah satu dari yang sudah ada.

Saya akan memberikan beberapa contoh dari mereka. Bootstrap adalah kerangka kerja hebat yang dilengkapi dengan semua yang Anda butuhkan untuk membangun antarmuka. Lihat di https://getbootstrap.com/. Ini open-source dan benar-benar gratis. Periksa file bootstsap.html, di mana kami menunjukkan bagaimana menerapkan formulir login kami menggunakan sistem grid Bootstrap dan kelas untuk menata komponen. Kerangka kerja populer lainnya adalah Foundation (https://foundation.zurb.com/). Satu lagi yang dikembangkan oleh orang-orang dari Google disebut Desain material dan dapat ditemukan di https://material.io/design/. Ide di balik semua sistem ini sangat sederhana: pengembang harus fokus pada pengembangan produk, dan alat ini akan membantu mereka mencapai efek visual yang dibutuhkan dengan cara yang cepat dan ramah. Namun, jika Anda menemukan diri Anda berjuang dengan alat untuk waktu yang cukup lama untuk membuat segala sesuatunya bekerja seperti yang Anda inginkan, dan Anda tahu bahwa Anda dapat mencapai efek ini dengan menulis beberapa baris kode CSS, maka lepaskan saja alat dan tulis kode CSS Anda sendiri. Ingat: alat ada di sini untuk mengurangi waktu pengembangan Anda dan bukan untuk menambahnya!

6.13 Pra-Prosesor dan Mesin Template

Seperti yang Anda lihat, menulis kode HTML dan bahkan CSS bukanlah ilmu roket. Ini membutuhkan beberapa disiplin, karena Anda dapat dengan mudah berakhir dengan basis kode besar yang sulit dipelihara, tetapi tidak sulit sama sekali. Namun, pengembang adalah orang paling malas di dunia, jadi mereka menilai secara kritis setiap simbol yang harus mereka tulis dan dapat dengan mudah mengatakan, "Ini terlalu bertele-tele!" atau, “Ada banyak kode yang berulang, ayo lakukan sesuatu!” Jadi, misalnya, ketika datang ke kode HTML, mereka memutuskan bahwa simbol tag pembuka dan penutup (< dan >) tidak berkontribusi pada tujuan keseluruhan sehingga mereka datang dengan bahasa baru di mana mereka menghapusnya. Atau, misalnya, mengapa kita harus mengulang kode untuk hal-hal seperti footer di setiap halaman situs web kita? Pengembang sangat menyukai prinsip KERING dan KISS—itulah sebabnya mereka terus-menerus menghadirkan bahasa baru untuk membuat hidup mereka lebih mudah.

Prinsip KERING berarti "Jangan ulangi dirimu sendiri" dan KISS berarti "Tetap Sederhana Bodoh!"

Bahasa ini disebut pra-prosesor atau mesin templating untuk HTML dan pada dasarnya adalah bahasa pemrograman yang mengurangi waktu pengembangan dengan menyediakan sarana penggunaan kembali kode dan mengurangi jumlah kode itu sendiri. Setelah menulis dalam bahasa ini, Anda harus menerapkan alat khusus yang disebut transpiler yang akan mengubah kode ini menjadi HTML dan CSS lagi, sehingga browser dapat memprosesnya. Ada banyak dari mereka. Jadi, untuk HTML Anda memiliki, misalnya, sebagai berikut:

- HAML: bahasa markup abstraksi HTML
- Giok: mesin templat
- Kumis
- HandlebarsJS

Mari, misalnya, membandingkan kode yang ditulis dalam HTML dan Jade. Pertimbangkan potongan kode HTML ini:

```
<div class="container">
<div id="main" class="row">
<h2>Log in</h2>
<form>
<div class="form-group">
<label for="email">Email</label>
< input type="email" class="form-control" id="email"
aria-describedby="emailHelp" placeholder="youremail@example.com">
</div>
</form>
</div>
</div>
```

Di Jade, Anda akan melakukan sesuatu seperti ini:

```
.container  #main.row
h2 Log in
form
.form-group
label(for='email') Email
input#email.form-control(type='email',
placeholder='youremail@example.com')
```

Ini akan sangat mirip di HAML:

```
.container
 #main.row
%h2 Log in
%form
.form-group
%label{:for => "email"} Email
%input#email.form-control{:placeholder => "youremail@example.com",
:type => "email"}/
```

Seperti yang Anda lihat, mesin templat untuk HTML menghapus kode yang tidak perlu dan bahkan menyediakan sarana untuk menggunakan kembali templat di dalam templat; jadi, misalnya, Anda dapat memiliki template untuk footer dan hanya memasukkannya ke dalam template lain, sehingga Anda tidak perlu mengulang kode. Itu cukup berguna.

Ada sesuatu yang mirip untuk CSS juga. Dua alat paling populer untuk CSS adalah Sass (https://sass-lang.com/) dan KURANG (http://lesscss.org/). Sass adalah bahasa ekstensi CSS yang berarti "Sintactically Awesome Style Sheets". less adalah pra-praprosesor lain untuk CSS, yang terinspirasi oleh Sass, yang ditulis dalam JavaScript, dan tagline-nya adalah, "Ini CSS, dengan sedikit lebih banyak."

Satu hal yang sangat berguna yang ditawarkan oleh Sass dan Less adalah variabel. Bayangkan Anda memiliki beberapa warna tertentu yang Anda gunakan di seluruh halaman dalam elemen yang berbeda. Katakanlah, merah muda, #f28fe9. Anda harus mengulangi kode warna ini di semua tempat di mana Anda ingin menggunakannya. Bayangkan bahwa untuk beberapa alasan Anda harus mengubah kode ini menjadi merah muda yang sedikit lebih terang, #ffb6c1. Tentu saja, ada fitur bagus yang disebut temukan dan ganti, tetapi tetap saja itu membosankan. Apa yang dapat Anda lakukan di Sass or Less adalah dengan mendeklarasikan variabel bernama pink dan menggunakan serta menggunakannya kembali di seluruh kode:

```
/ Variables
@pink:    #f28fe9;
@light-pink:  lighten(@pink, 10%);
// Usage
a,
.link {
color: @pink;
}
a:hover {
color: @light-pink;
}
```

Sekarang jika Anda perlu mengubah warna merah muda Anda menjadi sesuatu yang lebih gelap atau lebih terang, Anda hanya dapat mengubahnya di satu tempat! Seberapa mengagumkan itu? Selain menawarkan dukungan bahasa untuk membuat kode Anda dapat digunakan kembali dan ramah, baik Sass dan Less menawarkan beberapa pustaka komponen siap pakai, dukungan untuk sistem Grid, tema yang berbeda, dll. Jika Anda tertarik dengan subjek ini untuk membuat web lebih cantik menggunakan CSS , kami sangat menyarankan Anda untuk melihat dan bermain dengan semua alat ini.

6.14 Konten Dinamis

Sekarang kita tahu bagaimana mengimplementasikan antarmuka pengguna kita menggunakan HTML dan CSS, sekarang saatnya untuk membuatnya hidup! Di bagian ini,

seperti yang mungkin sudah Anda duga, kita akan berbicara tentang JavaScript! JavaScript adalah bahasa pemrograman yang ditafsirkan tingkat tinggi.

Menjadi bahasa pemrograman tingkat tinggi berarti kode lebih dekat dengan manusia daripada ke mesin, sehingga mudah dibaca oleh manusia. Diinterpretasikan berarti tidak perlu dikompilasi sebelum dieksekusi (seperti, misalnya, Java). Ini ditafsirkan oleh browser selama runtime.

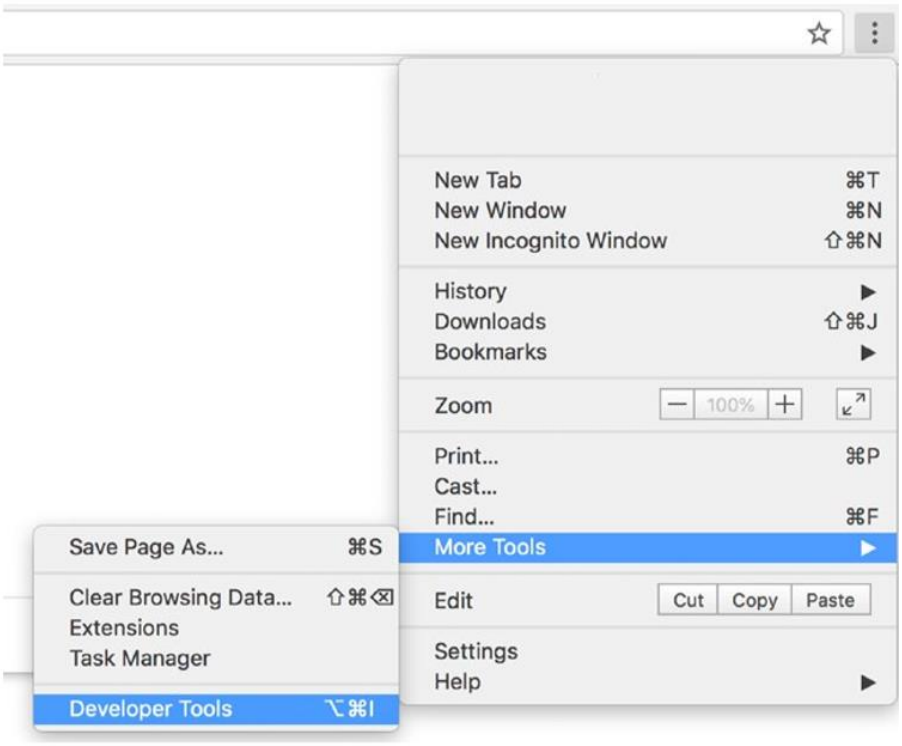
Menjadi bahasa yang ditafsirkan dan tidak dikompilasi di satu sisi mengurangi waktu pengembangan, karena Anda tidak perlu mengkompilasi kode setiap kali Anda mengubahnya; di sisi lain, ini lebih rawan kesalahan karena Anda mungkin menemukan kesalahan hanya selama runtime.

harap dicatat, meskipun setengah dari nama bahasa JavaScript adalah Java, itu tidak ada hubungannya dengan bahasa pemrograman Java! Beberapa orang bahkan mengatakan, "JavaScript ke Java seperti Mobil ke Karpet."

Karena JavaScript berjalan di sisi klien, ia memiliki akses ke properti browser klien; sekali lagi, berhati-hatilah dengan informasi yang Anda akses dan gunakan—hukum perlindungan data tidak tidur!

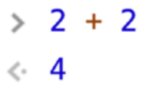
6.15 Konsol Alat Pengembangan

Cara termudah untuk mencoba JavaScript adalah menggunakan konsol alat pengembangan. Itu ada di sebagian besar browser. Kami akan menunjukkan alat pengembang Chrome di sini. Gambar 6-13 menunjukkan cara Anda membukanya.



Gambar 6-13. Membuka konsol alat pengembang di Chrome

Di beberapa sistem, Anda cukup menekan F12. Di macOS, Anda harus menekan commandoption-I (harus berbeda, kan?). Dengan alat pengembang dibuka, klik pada tab Konsol. Sekarang Anda dapat mengetik beberapa kode JavaScript, dan itu akan segera mengeksekusinya. Ketik, misalnya, 2 + 2 diikuti dengan Enter. Anda akan melihat sesuatu seperti yang ditunjukkan pada Gambar 6-14.



Gambar 6-14. JavaScript dieksekusi di chrome devtools console

Sekarang ketik new Date() dan klik Enter. Date() adalah fungsi JavaScript bawaan yang memungkinkan operasi dengan tanggal dan waktu. Ketik alert(new Date()), dan Anda akan melihat popup dengan Date baru. Alert adalah fungsi JavaScript bawaan lainnya yang menampilkan popup jelek ini. Tolong jangan gunakan itu.

6.16 Variabel

JavaScript juga memiliki variabel. Untuk mendeklarasikan variabel dalam JavaScript, gunakan kata kunci var diikuti dengan nama variabel. Misalnya, ketik var myVar = 10; Klik enter dan sekarang ketik myVar. Angka 10 akan ditampilkan. Ketik var a = 10; var b = 20;. Sekarang ketik a + b. Mainkan sedikit dengan variabel menggunakan konsol.

6.17 JavaScript

Tentu saja, menulis JavaScript di konsol browser adalah proses yang menyenangkan, tetapi Anda mungkin bertanya-tanya, “Oke, dan bagaimana cara menggunakannya di aplikasi? Saya tidak akan pergi ke konsol browser pengguna mengetik beberapa hal di sana.” Tentu saja, Anda tidak! Anda memasukkan kode JavaScript Anda di halaman Anda dengan cara yang sama seperti Anda memasukkan CSS Anda. Anda dapat, misalnya, menjadikannya internal untuk setiap file html menggunakan tag <script> dan menulis beberapa JavaScript di sana:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>JavaScript Alert</title>
</head>
<body>
<script>
alert(new Date())
</script>
</body>
</html>
```

Jika Anda membuka halaman ini (alert.html) di browser, popup dengan tanggal saat ini akan muncul. Itu juga bisa eksternal dan diimpor ke halaman html Anda menggunakan tag skrip kosong dengan atribut src:

```
<script src="script.js"></script>
```

Mari kita lihat bagaimana kita dapat menampilkan tanggal di halaman web menggunakan JavaScript. JavaScript memungkinkan penggantian konten elemen HTML. Anda dapat memperoleh referensi ke elemen menggunakan salah satu fungsi "document.getElementBy...". Sebagai contoh:

```
document.getElementById(container)
```

Sekarang Anda dapat mengakses berbagai atribut elemen ini, termasuk 'innerHTML', yang mendefinisikan apa pun yang tertulis di dalam elemen ini. Jika Anda mengubah nilai ini, itu akan secara otomatis berubah di halaman. Misalnya, jika kita membuat div dengan id 'date', mendapatkan referensinya, dan menetapkan nilai Date() baru ke innerHTML-nya, kita akan berakhir dengan tanggal yang muncul di dalam div: File 'date.html '

```
<body>
<div id="date"></div>
<script>
document.getElementById('date').innerHTML = new Date()
</script>
</body>
```

Itu terlihat agak jelek, bukan begitu? Bagaimana jika kita ingin menampilkan tanggal seperti: “hari ini tanggal 28 Mei”. Bagaimana kita melakukannya? Objek tanggal yang dibuat oleh fungsi Date() baru memiliki banyak metode yang dapat kita gunakan untuk mengakses properti tanggal yang berbeda. Misalnya, getMonth() memberi Anda bulan saat ini, getDate() akan memberi Anda hari dalam sebulan, dan seterusnya. Jika Anda ingin menggunakan nama bulan, Anda dapat menggunakan API internasionalisasi: new Date().toLocaleString('en-us', {month: 'long'})

6.18 Latihan Tentang Obyek Tanggal Di Javascript

Gunakan pengetahuan tentang fungsi Date() untuk menampilkan tanggal dalam bentuk “Hari ini 28 Mei” di browser. Gunakan file exercise3.html sebagai basis.

- Terapkan pengetahuan Anda tentang variabel.
- Gunakan fungsi getElementById() dan properti innerHTML-nya.
- Terapkan pengetahuan Anda tentang mengakses properti yang berbeda dari objek Date.

Periksa diri Anda di latihan3-solusi.html

Bagaimana jika kita ingin menampilkan waktu saat ini alih-alih tanggal saat ini? Kita bisa melakukan sesuatu seperti: File time.html

```
<body>
<div id="time"></div>
<script>    var date = new Date()
```

```
document.getElementById('time').innerHTML = `Now it is ${date.
getHours()}:${date.getMinutes()}`
</script>
</body>
```

Perhatikan bahwa alih-alih menambahkan string dengan tanda plus seperti yang kita lakukan di latihan sebelumnya, saya menggunakan literal template JavaScript.

Literal template dalam JavaScript dibungkus dengan karakter back-tick (` `) alih-alih tanda kutip ganda atau tunggal. Literal ini dapat berisi ekspresi JavaScript di dalamnya yang ditunjukkan oleh tanda dolar dan kurung kurawal: ${expression}.

Jika Anda membuka file ini di browser, Anda akan melihat waktu saat ini. Bagaimana jika Anda ingin memperbaruinya setiap kali Anda mengklik beberapa tombol? Objek DOM JavaScript (yang Anda peroleh menggunakan fungsi document.getElementBy...) memiliki beberapa properti yang terkait dengan peristiwa yang dapat diterapkan padanya—misalnya, onclick, onblur, onchange, ondblclick, dll. Kemudian Anda dapat memanggil tindakan apa pun pada acara ini. Sebagai contoh:

```
button.onclick = function () {
alert('Button clicked!')
}
```

Mari kita lakukan untuk pembaruan waktu kita. Kita harus membuat elemen tombol pada halaman, mendapatkan referensinya, dan memberitahu bahwa elemen tersebut harus mengisi bagian dalam HTML dari div waktu dengan waktu baru:

File time-with-button.html

```
<body>
<div id="time"></div>
<button id="update">Update time!</button>
<script>
var button = document.getElementById('update')
button.onclick = function () {
var date = new Date()
document.getElementById('time').innerHTML = `Now it is ${date.
getHours()}:${date.getMinutes()}:${date.getSeconds()}`
}
</script>
</body>
```

Buka file ini di browser Anda dan klik tombol.

Fungsi

Apakah Anda ingat bahwa pengembang berusaha untuk dapat digunakan kembali kode? Itu sebabnya JavaScript memiliki fungsi. Kita dapat membungkus beberapa pernyataan JavaScript ke dalamnya, daripada mengetik pernyataan tersebut hanya menggunakan fungsi.

Untuk mendeklarasikan fungsi dalam JavaScript, aktifkan pernyataan fungsi diikuti dengan nama fungsi, tanda kurung, dan kurung kurawal untuk badan fungsi:

```
function myFunction () {
```

```
// JavaScript statements
}
myFunction()
```

Misalnya, kita dapat meletakkan kode yang membuat string waktu dan menulisnya dalam div waktu ke dalam fungsi dan memanggil fungsi ini di dalam panggilan balik onclick:

```
function updateTime () {
var date = new Date()
document.getElementById('time').innerHTML = `Now it is ${date.
getHours()}:${date.getMinutes()}:${date.getSeconds()}`
}
var button = document.getElementById('update')
button.onclick = updateTime
```

Bukankah agak membosankan harus mengklik tombol setiap kali kita ingin melihat waktu yang diperbarui? Bukankah lebih baik jika itu hanya memperbarui dirinya sendiri secara berkala? Sebenarnya, JavaScript menyediakan fungsi ini yang disebut "setInterval". Fungsi ini menerima dua argumen: fungsi panggilan balik, untuk dipanggil secara berkala, dan interval waktu, dari mana panggilan balik harus dipanggil. Misalnya, jika kita ingin menambahkan "halo" ke beberapa div setiap detik, kita bisa melakukan sesuatu seperti:

```
setInterval(() => {
textDiv.innerHTML += 'hello'
}, 1000)
```

Perhatikan bahwa satuan waktu interval diberikan dalam milidetik—dengan demikian, 1000 sama dengan 1 detik. Kami juga dapat mencetak beberapa penghitung dan memperbaruinya di dalam fungsi setInterval: File set-interval.html

```
var counter = 1
var textDiv = document.getElementById('text')
setInterval(() => {
textDiv.innerHTML = `Hello for already ${counter} times!`
counter++
}, 1000)
```

Penghitung notasi++ hanya berarti bahwa kami meningkatkan nilai penghitung satu per satu. Ini adalah jalan pintas untuk pernyataan counter = counter + 1

Kemungkinan JavaScript tidak terbatas, dan kita bisa menghabiskan halaman ${counter} lagi hanya untuk membicarakannya. Jika Anda merasa antusias dengan topik ini, ikuti beberapa kursus online, pergi ke codecamp, pergi ke pertemuan, dan mulai membangun sesuatu. Tidak ada yang lebih efisien untuk belajar daripada mengotori tangan Anda.

LATIHAN TENTANG SETINTERVAL DI JAVASCRIPT

Gunakan fungsi setInterval dan fungsi updateTime yang telah kami terapkan sebelumnya untuk memperbarui waktu secara otomatis di halaman.

Periksa diri Anda di latihan4-solusi.html.

Anda telah menyadari, bukan, bahwa para pengembang adalah makhluk yang malas dan kreatif, dan mereka tidak pernah puas dengan cara kerja segala sesuatunya. Mereka

selalu berusaha untuk meningkatkan hal-hal sehingga mereka harus mengetikkan lebih sedikit kode dan bisa lebih efisien dan produktif. Itulah sebabnya mereka datang dengan ratusan kerangka kerja yang berbeda untuk membuat hidup mereka lebih mudah dan bekerja lebih sedikit. Mari kita bicara tentang kerangka kerja.

6.19 Kerangka kerja

Tentu saja, Anda dapat menulis seluruh perangkat lunak hanya menggunakan JavaScript biasa, tetapi selama ini pengembang tidak tidur dan mengembangkan banyak kerangka kerja untuk membantu Anda menyusun, membuat, dan memelihara semua jenis aplikasi JavaScript—dari skala kecil hingga skala besar , mulai dari jQuery, Backbone, Ampersand, React, Angular, Meteor, Ember, Vuejs yang terkenal... Daftarnya tidak ada habisnya. Beberapa dari mereka datang dengan tema yang sudah dibuat sebelumnya, dukungan Sass and Less, komponen yang dapat digunakan kembali, fungsi utilitas, plugin, add-on, dll.

Bagaimana seseorang memilih apa yang akan digunakan? Saran saya adalah: jangan gunakan kerangka kerja hanya demi menggunakan kerangka kerja. Jika Anda memiliki halaman web sederhana yang dapat Anda buat dan kelola sendiri menggunakan tumpukan HTML+CSS+JS biasa, lakukanlah! Pada akhirnya, semua kode yang dibangun dengan kerangka kerja akan ditranspilasikan ke dalam trio ini. Tentu saja, jika Anda memiliki arsitektur skala besar, banyak data yang harus ditangani, transisi yang berbeda, kondisi, dan kompleksitas yang sangat besar, maka benar-benar ya, periksa kerangka kerja.

Perlu diingat bahwa kerangka kerja memerlukan beberapa kurva pembelajaran. Periksa tabel perbandingan dan putuskan apa dan bagaimana Anda membutuhkannya. Terkadang saat Anda bekerja dalam tim dan memulai proyek baru, mungkin ada diskusi tentang apa yang harus digunakan. Apa yang biasanya Solikhan lakukan dalam kasus ini adalah mendistribusikan kerangka kerja yang berbeda di dalam anggota tim, menetapkan beberapa tenggat waktu (misalnya, akhir minggu), dan menjadwalkan presentasi 20 menit. Setiap orang harus menyiapkan POC (bukti konsep) menggunakan kerangka kerja yang dipilih dan mempresentasikannya kepada kelompok. Kemudian kelompok memutuskan kerangka mana yang paling menjawab kebutuhan proyek. Solikhan secara pribadi menyukai Vue.js karena dapat digunakan untuk kebutuhan yang sangat mendasar tanpa harus menginstal banyak alat dan juga dapat diskalakan untuk proyek besar dengan arsitektur yang kompleks. Jika Anda tertarik mempelajari Vue.js sambil membangun proyek menarik dengannya, lihat buku-buku yang ditulis Solikhan ini:

- Mempelajari Vue.js 2 https://www.packtpub.com/web-development/learning-vuejs-2
- Pengembangan Web Vue.js dan Bootstrap 4 https://www.packtpub.com/web-development/vuejs-2-and-bootstrap-4-web-development.

Belum lama ini, banyak pengembang web menggunakan jQuery (https://jquery.com/) untuk memodifikasi struktur DOM dan berkomunikasi dengan server (di antara ribuan hal lain yang ditawarkan oleh kerangka kerja ini). jQuery, pada kenyataannya, mudah dimengerti

dan digunakan. Keindahannya terletak pada seberapa baik ia menyederhanakan semua operasi terkait DOM. Misalnya, baris kode ini:

```
document.getElementById('container').innerHTML = 'hello'
```

hanya bisa diganti dengan

```
$('#container').html('hello')
```

jika Anda menggunakan jQuery.

jQuery menyediakan banyak fungsi praktis untuk menangani elemen web Anda. .hide(), .show(), .toggle(), .fadeOut(), .animate()—ini hanya beberapa .hide(), .show(), .toggle(), .fadeOut(), .animate()—ini hanya beberapa operasi yang dapat Anda jalankan pada elemen Anda menggunakan jQuery.

jQuery juga menyediakan cara mudah untuk berkomunikasi dengan server. Apakah Anda ingat di bab sebelumnya kita berbicara tentang metode REST (GET, POST, PUT, DELETE, PATCH)—metode yang digunakan untuk mengoperasikan dan mengambil dari server? jQuery menyediakan cara mudah untuk memanggil metode ini pada titik akhir yang diberikan dan mengambil data atau meneruskannya ke server. Metode jQuery disebut ajax(), dan menerima URL sebagai argumen dan, bila diperlukan, data untuk diteruskan ke server. Ajax berarti JavaScript dan Xml asinkron. Ini sebenarnya adalah salah satu singkatan yang paling menyesatkan dalam pengembangan web. Pertama, menggunakan ajax, Anda dapat menjalankan panggilan sinkron dan asinkron. Kedua, tidak ada lagi yang menggunakan Xml untuk mengangkut data melalui jaringan.

Untuk menggunakan metode ajax(), Anda cukup memanggil $.ajax(URL, [data]). Periksa dokumentasi lengkap di halaman jQuery resmi: http://api.jquery.com/jquery.ajax/. Selain metode ajax, jQuery menyediakan pintasan seperti .get() dan .post(). Dengan cara ini Anda tidak perlu meneruskan objek konfigurasi sebagai parameter yang menentukan metode HTTP mana yang ingin Anda gunakan dalam beberapa kasus tertentu. Mari kita gunakan jQuery untuk login di aplikasi kita. Kita tahu bahwa titik akhir untuk login adalah oauth/token, kita harus menggunakan metode POST, dan kita harus memasukkan nama pengguna dan kata sandi ke sana. Pertama, mari kita jalankan aplikasi server kita. Masuk ke dalam folder integrasi server, pastikan Anda telah menginstal Java 8, dan jalankan:

```
mvn clean install -DskipTests
cd liquibase
mvn liquibase:update
cd ..
mvn tomcat7:run
```

Sekarang jika Anda membuka halaman Anda di http://localhost:8080/oauth/token, itu akan menimbulkan kesalahan. Tentu saja, ini harus disebut sebagai metode POST dan meneruskan nama pengguna dan kata sandi ke server. Apakah Anda ingat halaman login kami yang harus Anda tata? Untuk contoh ini, kami telah memasukkannya ke dalam folder rest-api/src/main/webapp sehingga berjalan di server kami. Untuk memeriksa apakah itu benar-benar berjalan, buka localhost:8080/login.html. Jika Anda mengisi formulir dengan beberapa data dan mengklik tombol login, tidak akan terjadi apa-apa. Apa yang kita inginkan

terjadi adalah bahwa dengan mengklik tombol, metode posting jQuery dipanggil pada titik akhir oauth/token dengan nama pengguna dan kata sandi. Kami juga harus memberikan properti grant_type untuk menunjukkan bahwa itu adalah kata sandi. Jadi, panggilan kita akan terlihat seperti berikut:

```
$.post('http://localhost:8080/oauth/token', {
grant_type: 'password',
username: <EMAIL>,
password: <PASSWORD>
})
```

Bagaimana kita mendapatkan nilai untuk email dan kata sandi? Apakah Anda ingat betapa mudahnya dengan jQuery untuk memanipulasi DOM dan mengambil sesuatu darinya? Untuk mengambil nilai dari beberapa input dengan jQuery, Anda dapat menggunakan metode .val(). Jadi, misalnya, untuk mengambil nilai dari input email, Anda harus memanggil $('#email').val(). Hal penting lainnya untuk dipahami tentang metode ajax jQuery adalah bahwa mereka adalah janji.

janji adalah fungsi asinkron yang sangat khusus dalam JavaScript yang mengembalikan nilai di masa mendatang. Kami tidak menunggu mereka menyelesaikan eksekusi mereka; sebagai gantinya kami memanggil metode .then() pada mereka dengan panggilan balik yang disediakan, dan panggilan balik ini akan dieksekusi setelah metode janji menyelesaikan eksekusinya, atau, secara ilmiah, diselesaikan. Tentu saja, janji bisa gagal juga, panggilan balik untuk kegagalan dieksekusi di dalam metode janji .catch().

Jadi, postingan kita harus memiliki metode .then dan .catch yang terpasang sehingga kita dapat mengeksekusi beberapa kode saat login berhasil atau gagal. Ini akan terlihat seperti berikut:

```
$.post('http://localhost:8080/oauth/token', {
<...>
})
.then(result => {    // Do something
})
.catch( error => {
// do something
})
```

Apa yang harus kita lakukan jika login berhasil? Nah, kita bisa mengarahkan pengguna ke halaman kursus. Anda cukup melakukannya dengan menetapkan properti window.location ke yang Anda butuhkan:

```
window.location = 'http://localhost:8080/courses.html'
```

MENGINTEGRASI HALAMAN LOGIN DENGAN BACKEND

Gunakan kode folder integrasi server sebagai basis. jalankan server seperti yang dijelaskan dalam file README.md. Buat pengguna baru menggunakan skrip createuser.sh. Dalam latihan ini, Anda harus dapat masuk dengan pengguna dengan kredensial test@example.com/w1secret$. Anda hanya perlu memodifikasi hal-hal di dalam folder rest-api/src/main/webapp. Di sana Anda dapat melihat tiga folder file html, css, dan js. Kami telah

mengekstrak CSS yang diperlukan ke file style.css di dalam folder css, dan kami menambahkan kode jQuery ke folder js dan mengimpornya ke file login.html.

- Gunakan metode $.post di dalam fungsi login dengan url 'oauth/token'.
- mengarahkan pengguna ke halaman course.html jika login berhasil. Anda perlu memodifikasi kode di dalam tag <script> di dalam file login.html untuk menyelesaikan implementasi fungsi login.

Periksa diri Anda di folder solusi integrasi server.

Dalam contoh ini, entah bagaimana kita tahu bahwa titik akhir untuk login adalah “oauth/token” (ha! “entah bagaimana” berarti saya membaca bab sebelumnya di mana Lawrence mengimplementasikan titik akhir tersebut). Di sisi lain, bukan kasus biasa bahwa orang-orang frontend dan backend menulis buku bersama, oleh karena itu mengetahui sebelumnya apa yang akan dilakukan setiap orang selanjutnya. Jadi, orang-orang frontend perlu memahami cara menggunakan API yang disediakan oleh backend. Untuk ini, kita harus berbicara tentang kontrak. Ya, ketika kami (pengembang) bekerja sama, terkadang kami harus menandatangani semacam perjanjian agar pekerjaan kami bagus dan lancar.

6.20 Kontrak Antara Frontend dan Backend

Kami telah menyatakan bahwa JavaScript bertanggung jawab untuk mengambil data dari backend dan menampilkannya di frontend; itu juga bertanggung jawab untuk mengumpulkan data di sisi klien dan meneruskannya ke backend. Tentu saja, semua data ini harus mengikuti beberapa format; kita tidak bisa begitu saja meneruskan apa pun ke backend karena tidak akan tahu bagaimana menafsirkannya. Selain itu, kami tidak bisa mendapatkan data dari backend dan menampilkannya di frontend tanpa mengetahui format data mana yang masuk. Itulah sebabnya pengembang banyak berkomunikasi untuk membuat semacam kontrak. Anda tidak akan pernah menemukan departemen frontend dan backend yang hanya bekerja secara terpisah. Anda telah melihat bahwa pada setiap tahap proses, ada banyak komunikasi bolak-balik antara orang-orang dengan peran yang berbeda: pemilik produk banyak berbicara kepada pebisnis dan kemudian kepada pengembang dan desainer; desainer terus-menerus terlibat dalam brainstorming dengan pengembang; dan pengembang backend harus dilibatkan pada tahap awal dalam diskusi untuk memahami jenis data apa yang ada dan kapan serta bagaimana data itu harus disimpan, diambil, dan ditampilkan.

Pengembang frontend banyak berkomunikasi dengan desainer untuk mengetahui cara menampilkan data dan dengan pengembang backend untuk mengetahui data mana yang akan disediakan dan dalam format apa. Ada berbagai cara untuk membuat perjanjian semacam ini. Yang biasanya kita lakukan adalah mendesain file JSON dengan deskripsi API—dengan cara ini frontend tidak perlu menunggu implementasi siap, kita cukup mengejek beberapa respons server dan bekerja dengan data palsu hingga sisi server siap . Ini banyak terjadi. Jadi, karena kami (Lawrence dan Solikhan) sedang mengerjakan backend dan frontend dari platform kursus online kami, kami harus membuat semacam kontrak. Lawrence membuat file JSON untuk Solikhan yang menjelaskan API. Perhatikan, ini masih

merupakan API yang sangat mendasar—ini belum lengkap, tetapi ini menunjukkan cukup banyak apa yang kami maksud dengan kontrak.

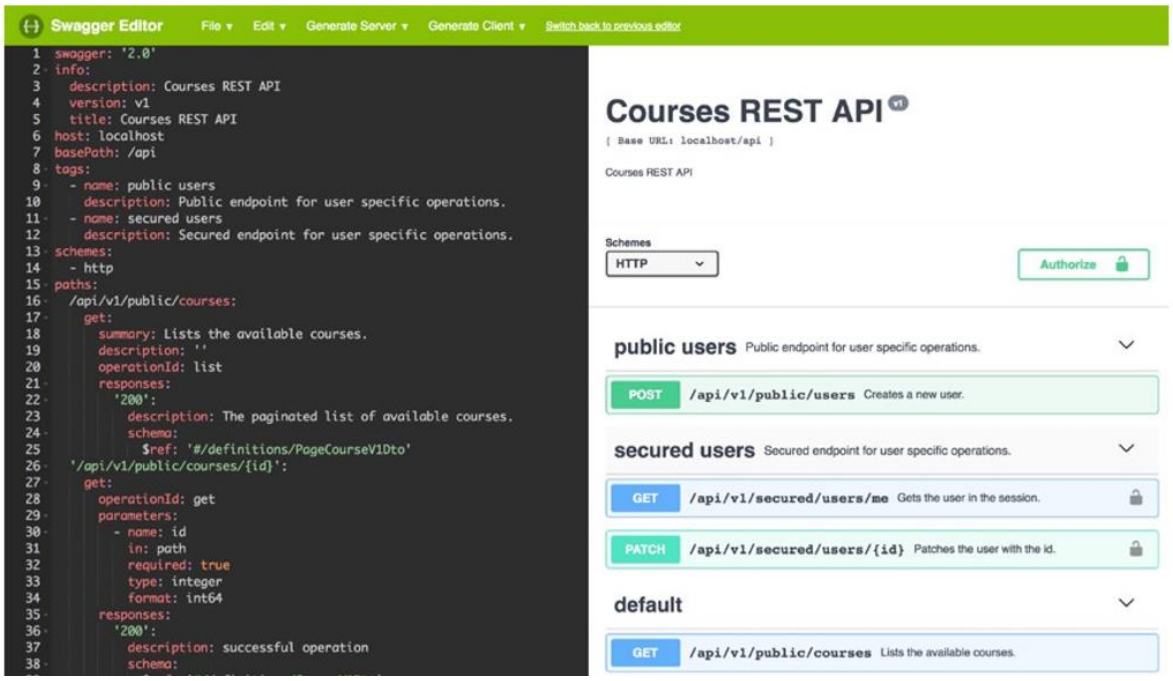
JSOn (Notasi Objek JavaScript) adalah format umum yang digunakan dalam pengembangan web. Format ini mudah dibaca oleh manusia, mudah diinterpretasikan oleh browser, dapat dengan mudah diulang oleh JavaScript, dan mudah dibuat oleh bahasa backend apa pun.

Jadi, bagian dari JSON ini yang terkait dengan daftar kursus terlihat seperti berikut (kami membutuhkan informasi mengenai jalur itu sendiri dan definisi tentang objek kursus):

```
"paths" : {
"/api/v1/public/courses" : {
"get" : {
"summary" : "Lists the available courses.",
"operationId" : "list",
"responses" : {
"200" : {
"description" : "The paginated list of available courses.",
"schema" : {
"$ref" : "#/definitions/PageCourseV1Dto"
}
}
}
}
}
},
"definitions" : {
"CourseV1Dto" : {
"type" : "object",
"properties" : {
"id" : {
"type" : "integer",
"format" : "int64"
},
"name" : {
"type" : "string"
},
"description" : {
"type" : "string"
},
"durationHours" : {
"type" : "integer",
"format" : "int32"
},
"active" : {
"type" : "boolean"
```

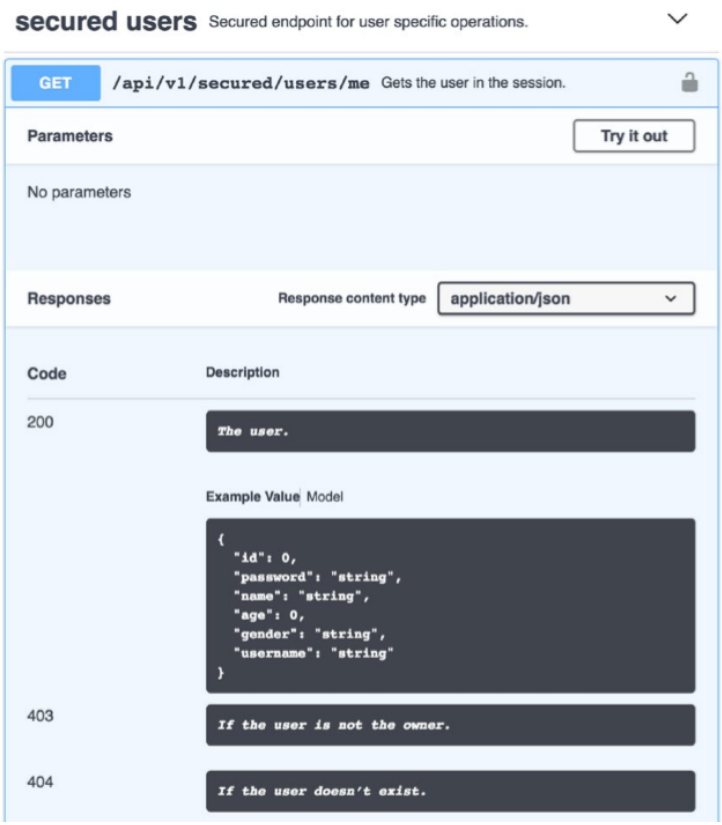
```
}
}
}
}
```

Ini hanyalah contoh kecil dari apa yang mungkin terlihat seperti objek JSON. Tentu saja, ini masih jauh dari lengkap, tetapi penting bahwa itu ada dan up to date, sehingga semua orang berada di halaman yang sama tentang bagaimana API dapat terlihat. Pada titik tertentu, “kontrak” ini menjadi sangat besar dan sulit untuk diikuti oleh manusia. Ada banyak alat di luar sana yang tidak hanya memungkinkan visualisasi yang bagus dari file API Anda, tetapi juga pengujiannya. Salah satu alat ini disebut Swagger (https://swagger.io/). Ini memiliki editor online tempat Anda dapat menempelkan JSON atau YAML Anda (yaml adalah format lain yang banyak digunakan—terutama untuk mendeskripsikan API) dan mendapatkan visualisasi API yang bagus. Cukup buka editor swagger (https://editor.swagger.io/) dan rekatkan swagger. json, yang dapat Anda temukan di dalam direktori rest-api/target. Anda akan melihat sesuatu yang ditunjukkan pada Gambar 6-15.



Gambar 6-15. Visualisasi API oleh Swagger

Jika Anda mengklik metode, Anda akan melihat deskripsi objek yang mendasarinya serta deskripsi kode kesalahan dan keberhasilan (Gambar 6-16).



Gambar 6-16. Deskripsi terperinci dari objek pengguna yang mendasarinya

Sekarang setelah Anda mengetahui bagaimana frontend berkomunikasi dengan backend, cara kerja API, cara mengimplementasikan desain, dan cara membuatnya bekerja, Anda siap untuk memilih kerangka kerja favorit Anda dan mulai mengimplementasikan frontend untuk aplikasi kita.

6.21 Membuat Aplikasi Frontend untuk Platform Kami

Kami telah menerapkan halaman HTML untuk halaman login untuk platform kami dan menatanya menggunakan CSS. Kami bahkan telah menjalin komunikasi dengan server menggunakan jQuery. Mari kita mendapatkan beberapa rasa untuk pengembangan dengan kerangka kerja dan menggunakan sistem desain Vue.js dan Material untuk mengembangkan frontend kita. Pertama, pastikan Anda telah menginstal npm di sistem Anda (https://docs.npmjs.com/cli/install). Untuk kompatibilitas versi, saya menggunakan versi 8.11.1. Sebagai saran pribadi, instal nvm (pengelola versi node), karena ini akan memungkinkan Anda untuk menginstal versi yang berbeda dan menggantinya sesuai kebutuhan. Sekarang mari kita instal antarmuka baris perintah vue:

```
npm install vue-cli -g a
```

Sekarang kita ingin membuat proyek Vue. Ada berbagai cara untuk melakukan itu; kami ingin menggunakan template yang menggunakan desain material. Ini disebut vuetifyjs (https://vuetifyjs.com/), dan untuk mem-bootstrap proyek baru menggunakan template ini, gunakan template vuetifyjs/nuxt. Vuetify, seperti yang telah kami tunjukkan, adalah kerangka kerja desain material, dan nuxt (https://nuxtjs.org/) adalah alat yang dirancang

untuk vue yang membuat kerangka kerja Vue yang sudah mudah digunakan menjadi lebih mudah. Ini pada dasarnya mengabstraksi semua komunikasi dan perutean klien-server sementara kami fokus pada bagian UI. Jadi, mari buat proyek dan beri nama frontend:

```
vue init vuetifyjs/nuxt frontend
```

Sekarang beralih di dalam folder frontend dan jalankan yarn install dan yarn run dev:

```
yarn install yarn run dev
```

Jika Anda belum memasang yarn, instal secara global: npm install yarn -g. Jika Anda membuka browser Anda di localhost:3000, Anda akan melihat halaman indeks vuetify default. Keindahan nuxt.js yang mendasarinya adalah ia memberi Anda implementasi perutean dan transisi antar halaman yang siap pakai. Segala sesuatu yang terletak di dalam folder halaman menjadi rute. Lanjutkan dan buat halaman login.vue, register.vue, dashboard.vue, dan kursus.vue di dalam folder halaman. Kita akan menggunakan komponen file tunggal.

Komponen file tunggal di vue adalah file yang memiliki template, gaya, dan skrip semua dalam file yang sama. Idealnya ini adalah komponen kecil yang dapat digunakan kembali.

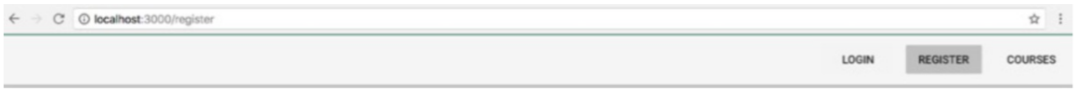
Buat bagian template di setiap halaman yang dibuat dan tulis saja nama bagiannya. Sebagai contoh,

```
// pages/login.vue
<template>
<div>Login</div>
</template>
// pages/register.vue
<template>
<div>Register</div>
</template>
```

Ubah tata letak default sehingga bilah navigasinya berisi tautan ke halaman ini:

```
// layouts/default.vue
<template>
<...>
<v-toolbar fixed app :clipped-left="clipped">
<v-spacer></v-spacer>
<v-btn flat router :to="'login'" exact>
Login
</v-btn>
<v-btn flat router :to="'register'" exact>
Register
</v-btn>
<v-btn flat router :to="'courses'" exact>
Courses
</v-btn>
</v-toolbar>
<...>
</template>
```

Klik tombol ini dan periksa betapa indahnya transisi antar halaman (Gambar 6-17).



Gambar 6-17. Transisi antar halaman

Perhatikan bagaimana kami tidak melakukan apa pun agar ini terjadi! Kami telah menerapkan halaman login dan register. Ada juga kode dasar untuk halaman kursus. Periksa kode di folder frontend. Anda harus menjalankan yarn install dan yarn run dev untuk menjalankan kode frontend dalam mode pengembang. Perhatikan bahwa Anda harus mengaktifkan dan menjalankan server agar dapat berfungsi. Beberapa penjelasan:

- Kami telah menggunakan plugin @nuxtjs/proxy untuk mengarahkan permintaan ke server kami yang terletak di bawah localhost:8080. Anda dapat menemukannya di file nuxt.config.js
- Kami telah menggunakan plugin vue-resource untuk komunikasi dengan server. Anda dapat menemukan kode yang sesuai di dalam file api/index.js
- Kami menggunakan Vuex untuk manajemen toko terpusat. Anda dapat menemukan status, tindakan, mutasi, dan getter yang sesuai di dalam folder toko.
- Perhatikan bahwa kami menggunakan titik akhir kursus untuk mendapatkan daftar kursus dan menampilkannya di halaman kursus. Namun, tampilannya sendiri tidak sepenuhnya dilaksanakan—kami hanya mencetak nama-nama mata kuliah saja. Gunakan apa yang telah Anda pelajari sejauh ini untuk membuatnya dengan baik!

Mainkan kode ini, ubah, tambahkan komponen, gunakan, gunakan kembali, menyalahgunakannya. Ini adalah proses yang menyenangkan dan gila!

6.22 Ide

Tentu saja, Anda dapat menulis kode Anda menggunakan notepad sederhana atau bahkan editor vim, tetapi pasti Anda telah bertanya pada diri sendiri lingkungan pengkodean apa yang paling cocok untuk pengembangan web. Sekali lagi, ada banyak dari mereka. Solikhan menggunakan Webstorm—ini adalah IDE keluarga IntelliJ, sama seperti yang digunakan Lawrence untuk pengembangan backend. Tapi ada banyak yang lain: sublime, atom, vim (ya, vim), tanda kurung... Kode Visual Studio IDE yang dikembangkan oleh Microsoft telah mendapatkan banyak popularitas akhir-akhir ini. Ya, orang-orang itu akhirnya mulai melakukan sesuatu untuk komunitas open source, dan mereka telah mencapai beberapa hasil yang luar biasa. Saya sarankan Anda mencoba beberapa editor ini sebelum memilih salah satu favorit Anda.

6.23 Kesimpulan

Dalam bab ini kami mengenakan topi seorang pengembang frontend untuk mengimplementasikan antarmuka yang kami rancang sebelumnya dan menghubungkannya ke backend yang diimplementasikan pada bab sebelumnya. Kami sebenarnya telah menyelesaikan perkenalan kami dengan implementasi untuk MVP kami. Saatnya untuk

mengujinya! Mari beralih ke bab berikutnya, di mana kita akan mencoba topi seorang insinyur penjaminan mutu dan menerapkan beberapa tes yang tidak hanya akan menilai beberapa fungsi sistem kita, tetapi juga memastikan kualitasnya dari ujung ke ujung. Jika, sampai sekarang, kita menikmati proses pembuatan perangkat lunak, di bab berikutnya kita harus melakukan sesuatu yang sangat berlawanan—menikmati proses menghancurkannya. Ketika Anda menguji perangkat lunak Anda, Anda harus melupakan bagian betapa sulitnya membangunnya. Lebih baik Anda memecahkan sesuatu sekarang dan memperbaikinya daripada jika pengguna Anda menemukan bug dan menjadi kecewa. Sekarang mari kita beralih ke bab berikutnya dan memecahkan masalah!

BAB 7
MENGUJI PRODUK KITA

Pada bab sebelumnya, kita telah menyelesaikan pengembangan frontend. Apakah itu berarti kita sudah selesai dengan pengembangan? Tidak! Tes menulis juga merupakan bagian dari pengembangan. Dalam bab ini kita akan membahas jenis pengujian apa yang ada di luar sana, bagaimana melakukannya, dan kapan menggunakannya. Dalam bab ini kami akan menjelaskan perbedaan antara penguji manual dan penguji otomatisasi dan membahas beberapa platform dan kerangka kerja pengujian. Kami harap Anda menikmatinya dan pada akhir bab ini Anda tahu betapa pentingnya topik itu.

7.1 Berbagai Jenis Pengujian

Berbagai jenis pengujian melayani kebutuhan yang berbeda. Ini tidak berarti bahwa satu dapat digunakan untuk mendukung yang lain atau bahwa beberapa jenis lebih baik daripada yang lain, semuanya memiliki tujuan mereka sendiri, dan, tergantung pada sifat proyek, beberapa mungkin lebih penting daripada yang lain. Bagian ini membahas beberapa jenis pengujian, yang kami yakini paling penting untuk kasus penggunaan kami, tetapi ada lebih banyak lagi, dan kami mengundang Anda untuk melihatnya jika Anda tertarik.

7.2 Pengujian Unit

Seperti namanya, pengujian unit berarti menguji sebagian kecil (unit) kode. Pengujian semacam ini dilakukan secara individual mungkin, artinya kode yang diuji sebaiknya dijalankan secara terpisah dari bagian kode lainnya. Dalam sebagian besar kasus, yang sedang diuji adalah metode, kelas, atau bahkan modul, dan kasus uji harus independen satu sama lain (misalnya, uji B tidak boleh mengharapkan sesuatu dibuat oleh uji A untuk lulus). Bergantung pada sifat aplikasinya, pengaturannya bisa lebih atau kurang rumit. Dalam kasus kita, misalnya, kita perlu menangani panggilan database; ini berarti bahwa kami harus mengatur semuanya sedemikian rupa sehingga semua perubahan basis data hanya valid selama pengujian dan kemudian dibuang.

Pengujian semacam ini adalah garis pertahanan pertama untuk kode Anda. Anda mungkin bertanya, “Mengapa saya harus menulis tes ini jika saya hanya dapat memverifikasi jika secara manual?” Ini adalah poin yang baik dan valid, tetapi hanya jika Anda berasumsi bahwa kode Anda tidak akan pernah berubah, dan kita semua tahu bahwa itu tidak benar di sebagian besar (jika tidak semua) kasus. Menulis pengujian unit akan mencegah perubahan lebih lanjut untuk memecahkan basis kode Anda di masa mendatang, karena Anda menyatakan harapan dari beberapa tindakan. Jika besok rekan baru Anda melakukan tugas untuk mengubah sesuatu dan menghapus beberapa parameter yang Anda gunakan sebelumnya, pengujian akan gagal. Kadang-kadang umum untuk menggunakan Pengujian Unit Parameterized, artinya input dihasilkan oleh kerangka pengujian dan bukan oleh orang

yang menulis pengujian. Dalam kebanyakan kasus, sulit untuk menyiapkan pengujian semacam ini karena modul yang diuji terlalu rumit.

Bayangkan kita menggunakan pengujian parameter untuk menguji kreasi pengguna. Pembatasan kami dengan kata sandi dan email akan mempersulit penyediaan benih yang baik untuk menghasilkan data. Di sisi lain, jika Anda baru saja membuat struktur memori, seperti daftar atau peta, yang bersifat generik, pengujian berparameter akan sangat cocok. Sekali lagi, alat yang tepat untuk pekerjaan yang tepat! Tes unit sering ditulis selama pengembangan karena mereka menyediakan cara untuk menguji kode segera tanpa harus memiliki semua bagian lain yang terpasang. Ini juga merupakan praktik yang baik untuk menyediakan beberapa tes yang menjalankan bagian kode yang berbeda, seperti cabang "jika" atau "loop".

Dalam kasus kami, ini akan mencerminkan, misalnya, menguji kreasi pengguna. Jika Anda hanya menyediakan kasus uji di mana pengguna baru, Anda tidak pernah menguji kasus di mana kami perlu mengeluarkan pengecualian karena Anda mendaftarkan pengguna yang sama, dan cabang kode itu akan dibiarkan tanpa pengujian apa pun. Keuntungan dari pengujian unit sangat besar, dan kami menyarankan agar semua proyek memilikinya. Pertama-tama, ini membantu kita menemukan inkonsistensi atau kesalahan pada tahap awal pengembangan. Kedua, seperti yang disebutkan sebelumnya, ini melindungi kemungkinan perubahan di masa mendatang atau refactor besar dari kerusakan kode Anda, terutama jika ditulis dalam bahasa dengan waktu kompilasi seperti Java.

Kelemahan dari pengujian unit adalah bahwa biasanya orang yang menulis tes sama dengan yang menulis kode, yang membuat tes menjadi bias. Ini juga hampir tidak mungkin untuk memberikan tes untuk semua kemungkinan kombinasi, yang mungkin membuat Anda berpikir bahwa hanya karena tes lulus, modul Anda adalah bukti peluru. Orang sering berdebat tentang seberapa dekat dengan kenyataan tes tersebut, karena terkadang tidak mudah untuk memberikan contoh kehidupan nyata untuk sebagian besar tes. Ini adalah poin yang valid, tetapi kami tetap percaya bahwa itu membawa manfaat.

Contoh pengujian unit untuk proyek kami adalah pendaftaran pengguna. Mendaftarkan pengguna adalah peristiwa terisolasi yang menggunakan sebagian besar satu modul di mana tidak ada interaksi dengan kode lain selain kode kerangka kerja dan database. Apa yang terutama kami verifikasi dengan pengujian ini adalah bahwa pengguna berhasil terdaftar, bahwa nama pengguna sama dengan yang kami berikan, dan bahwa id tidak nol (mengembalikan id saat membuat entitas sangat penting, karena kami mungkin perlu segera referensi entitas yang baru dibuat). Beberapa tes lain juga berlaku sebagai tes unit mengenai pengguna yang terdaftar. Tes semacam itu akan memverifikasi cabang kode lainnya, seperti kata sandi yang tidak valid atau nama pengguna yang sudah ada.

```
@Test
public void createAUserTest() throws Exception {
final MockHttpServletResponse result = mockMvc.perform(post
("/api/v1/public/users").contentType(MediaType.APPLICATION_JSON).
content(writeJson(UserV1Dto.builder().withUsername("ex1@example.
com").withPassword("123456a$").build()))).andReturn().getResponse();
```

```
assertThat(result.getStatus()).isEqualTo(HttpStatus.CREATED.value());
final UserV1Dto user = readJson(result.getContentAsString(),
UserV1Dto.class);
assertThat(user.getUsername()).isEqualTo("ex1@example.com");
assertThat(user.getId()).isNotNull();
}
```

7.3 Tes integrasi

Pengujian integrasi dalam pengembangan perangkat lunak digunakan untuk menegaskan bahwa modul yang berbeda dapat bekerja sama. Selama fase pengujian ini kontrak antarmuka diperiksa, artinya akan diverifikasi bahwa modul A dan modul B dapat bekerja sama dengan antarmuka yang telah disepakati sebelumnya. Mari kita definisikan bahwa dalam kasus kita, kita akan mempertimbangkan pengujian integrasi sebagai serangkaian pengujian antara modul dari aplikasi atau proyek yang sama, artinya modul di dalam aplikasi backend dan modul di dalam aplikasi frontend, tetapi tidak pernah di antara keduanya. Contoh yang baik dari kemungkinan pengujian integrasi adalah dengan memasukkan pengguna dan mencoba melakukan panggilan aman. Contoh sebelumnya menegaskan integrasi modul login dan titik akhir, di mana yang terakhir hanya akan berfungsi jika yang pertama juga berfungsi. Karena ada banyak modul yang terlibat, beberapa teknik dapat digunakan untuk pengujian semacam ini. Sekali lagi, ini tergantung pada kompleksitas dan seberapa cepat modul dapat diselesaikan dari pengembangan.

Pengujian big bang, seperti namanya, menunggu "badai sempurna"—yaitu, semua modul siap untuk mulai menulis atau melakukan pengujian. Dalam pengujian big bang, semua modul terintegrasi sekaligus, sehingga menghilangkan kebutuhan untuk mematikan atau mengejek modul lain untuk mulai membangun pengujian. Kelemahannya adalah seseorang harus menunggu semua modul siap untuk mengujinya sama sekali. Ini mungkin tampak dapat diterima, tetapi jika beberapa modul membutuhkan waktu berbulan-bulan untuk diselesaikan, maka tampaknya tidak tepat. Pengujian inkremental mengatasi masalah ini dengan membuat tiruan atau rintisan untuk modul yang belum siap. Pengujian semacam ini tentu saja tidak 100% menegaskan integrasi kedua modul, tetapi akan memverifikasi bahwa transfer data antara kedua modul berfungsi dengan baik dan bahwa kontrak telah dipenuhi.

Untuk pendekatan inkremental dan bergantung pada sifat proyek, Anda dapat menggunakan pendekatan bottom-up di mana modul tingkat rendah dan biasanya modul dengan dependensi lebih sedikit diuji terlebih dahulu. Untuk ini, Anda memerlukan semacam driver atau pembuat lalu lintas/konten untuk menjalankan modul itu. Untuk pendekatan top-down, kebanyakan sebaliknya, jadi kita mulai dengan menguji modul tingkat tinggi—biasanya yang memiliki beberapa dependensi—dan untuk itu kita perlu membuat stub atau tiruan. Di lingkungan yang tangkas, di mana siklus rilis biasanya sekitar 2 minggu, big bang banyak digunakan. Juga tidak umum bahwa Anda akan membuat dua atau tiga modul dari awal dalam satu iterasi, artinya Anda akan membangun aplikasi Anda dalam blok dan menambahkan modul baru secara iteratif. Ini mengarah pada fakta bahwa ketika kita benar-

benar perlu melakukan pengujian integrasi, sebagian besar modul sudah ada, dan kita dapat menulis pengujian integrasi tanpa harus menunggu modul lain atau membuat stub/ mock atau driver untuk data palsu.

Kami dapat memberikan beberapa contoh pengujian yang dapat kami lakukan, tetapi kami akan memberikan satu yang kami yakini penting karena melibatkan beberapa modul bahkan beberapa disediakan oleh kerangka kerja yang kami gunakan di backend kami. Kami akan menguji apakah token penyegaran berfungsi dengan baik dalam kasus ini. Agar token penyegaran berfungsi dengan baik, kami tidak hanya mencoba untuk mendapatkan akses baru dan token penyegaran menggunakan token penyegaran sebelumnya, tetapi kami juga memverifikasi bahwa token akses baru ini valid dan kami dapat menggunakannya untuk mengakses titik akhir dan sumber daya yang dilindungi.

```
@Test
public void refreshTest() throws Exception {
final String accessTokenAfterRefresh = readJson(mockMvc.perform(post
("/oauth/token")   .contentType(MediaType.APPLICATION_FORM_URLENCODED)
.header("Authorization", "Basic d2ViYXBwOnRlc3Q=")
.content(String.format("grant_type=refresh_token&refresh_token=%s",
getTokens().get("refresh_token"))))
.andReturn()
.getResponse().getContentAsString(), new TypeReference < Map < String,
String >> () {}).get("access_token")
final MockHttpServletResponse result = mockMvc.perform(get("/api/v1/
secured/users/me")
.header("Authorization", "bearer " + accessTokenAfterRefresh)
.contentType(MediaType.APPLICATION_JSON))
.andReturn()
.getResponse(); assertThat(result.getStatus()).isEqualTo(HttpStatus.OK.value());
final UserV1Dto user = readJson(result.getContentAsString(), UserV1Dto.
class);
assertThat(user.getUsername()).isEqualTo(testUser.getUsername());
}
```

Anda dapat menemukan seluruh contoh dan dependensi di RestApiUserTest.java dan AbstractRestApiTest.java. Penerimaan tes ini tidak hanya kita dapat melakukan panggilan dengan token akses baru, tetapi juga bahwa token yang kita hasilkan sebenarnya untuk pengguna yang sama.

7.4 Pengujian Sistem

Pengujian sistem adalah istilah yang menggabungkan beberapa jenis pengujian. Biasanya tahap pengujian ini terjadi setelah pengujian integrasi dan bertujuan untuk memverifikasi bahwa sistem dapat bekerja dengan baik secara keseluruhan. Dalam kasus kami, misalnya, ini akan menguji bahwa frontend dan backend bekerja sama dengan baik. Untuk pengujian semacam ini, tidak diperlukan pengetahuan tentang internal produk, kode dan logika membuatnya menjadi fase pengujian kotak hitam.

Pengujian kotak hitam dapat berupa fungsional atau non-fungsional. Pengujian fungsional berfokus pada memeriksa apakah sistem memenuhi persyaratan yang ditentukan lebih awal, sedangkan pengujian non-fungsional berfokus pada memeriksa bagaimana kinerja sistem di bawah beban berat dan situasi stres. Beberapa contoh pengujian fungsional termasuk kesalahan antarmuka atau penanganan yang tidak tepat; kesalahan dalam struktur atau perilaku basis data; dan implementasi yang salah atau fungsionalitas yang hilang.

Keuntungan dari teknik pengujian ini adalah karena penguji tidak memiliki informasi apa pun tentang internal aplikasi, mereka dapat membantu menunjukkan perbedaan antara spesifikasi dan implementasi sebenarnya lebih baik daripada seseorang yang memiliki pengetahuan karena mereka tidak bias. Juga, penguji ini tidak perlu memiliki keterampilan pemrograman untuk melakukan tugas ini. Menjadi hampir tidak mungkin untuk menguji semua kasus, sehingga meninggalkan beberapa jalur yang belum diuji, adalah salah satu kelemahan utama dari teknik pengujian semacam ini. Kami tidak akan membahas semua kemungkinan tes yang dilakukan selama fase ini, karena ada banyak tes, dan tergantung pada sifat proyek, beberapa mungkin dapat diterapkan sementara yang lain tidak, tetapi jika Anda tertarik dengan topik ini dan ingin tahu lebih banyak, jangan ragu untuk menyelam lebih jauh—Google akan menjadi teman Anda!

7.5 Ujian Penerimaan

Pengujian penerimaan adalah salah satu tingkat pengujian terakhir, terutama saat mengirimkan produk baru. Selama fase pengujian ini, aplikasi perangkat lunak akan diuji kesesuaiannya antara spesifikasi dan implementasi, dengan mempertimbangkan bahwa persyaratan bisnis atau kontrak juga dipenuhi. Seperti pada tahap pengujian sebelumnya, pengujian ini juga dilakukan dengan menggunakan pengujian black-box. Tes ini dilakukan pada abstraksi tingkat tinggi dari implementasi, dengan fokus pada bagaimana klien akhir akan menggunakan produk. Tergantung pada sifat produk, dua tahap mungkin berlaku selama pengujian.

Tahap pertama adalah tahap di mana penerimaan dilakukan secara internal oleh tim dari perusahaan yang sama yang mengimplementasikan produk, tetapi biasanya bukan seseorang yang terlibat dalam pengembangan itu sendiri. Tim tersebut termasuk Manajemen Produk, Layanan Pelanggan, Pengembangan Bisnis, atau bahkan departemen Penjualan. Yang lainnya dilakukan secara eksternal oleh orang atau perusahaan yang tidak terlibat dalam pembangunan. Dalam hal ini dapat dilakukan Customer Acceptance Testing, dimana perusahaan yang meminta layanan bertanggung jawab untuk memverifikasi bahwa semuanya sesuai dengan kontrak yang ditetapkan; atau, dalam kasus lain, itu mungkin Pengujian Penerimaan Pengguna di mana satu set pengguna akhir bertanggung jawab untuk menguji dan memberikan umpan balik. Terkadang itu disebut juga sebagai pengujian Beta.

7.6 Pengujian Regresi

Pengujian regresi adalah salah satu fase pengujian yang paling penting ketika memelihara dan mengembangkan suatu produk. Tujuannya adalah untuk memastikan bahwa perubahan kode baru tidak mempengaruhi versi perangkat lunak yang telah diuji

sebelumnya dan stabil. Meskipun Anda merasa bahwa perubahan kecil dan terisolasi tidak mungkin berdampak pada sistem secara keseluruhan, kemungkinannya selalu ada. Percayalah pada kami ketika kami mengatakan itu karena terkadang Anda tidak melihatnya datang dan bahkan jika Anda mengetahui seluruh basis kode dari titik A ke titik B, pada titik tertentu Anda akan memecahkan sesuatu. Terkadang itu bahkan bukan kode Anda, tetapi beberapa kerangka kerja yang Anda gunakan.

Sebagai contoh, hanya dengan menerima daftar kosong pada fungsi yang menggunakan kueri Hibernasi dengan klausa IN dapat menyebabkan bug dalam kode Anda. Apakah itu berarti mereka salah? Yah itu rumit. Jika Anda mencoba menulis kueri dalam SQL menggunakan "... WHERE kolom IN ()..:" itu akan gagal, jadi siapa yang harus mengurusnya? Perpustakaan pemetaan atau programmer? Ini mungkin menghasilkan diskusi tanpa akhir, jadi mari kita tinggalkan di sini; yang penting adalah Anda mengerti bahwa bahkan hal kecil yang tampaknya tidak berbahaya dapat merusak kode Anda, dan Anda mungkin menyadarinya hanya ketika sudah dalam produksi.

Dalam kasus ini, jika sebelumnya Anda memiliki pengujian yang memeriksa daftar kosong dan menerima kegagalan dengan anggun, dengan mengubah kode untuk mengizinkan daftar kosong, pengujian tersebut akan gagal—tidak hanya dari sudut pandang penerimaan, tetapi juga akan menghasilkan dalam sebuah kesalahan. Jadi seperti yang Anda ketahui, selama regresi Anda tidak menulis tes baru; Anda hanya menjalankan yang sebelumnya terhadap versi baru Anda. Inilah alasan mengapa sangat penting bagi Anda untuk menutupi kode Anda dengan sebanyak mungkin pengujian yang bermakna. Tes regresi dapat dilakukan pada salah satu fase sebelumnya yang dibahas dalam bagian ini (Unit, Integrasi, Sistem, atau Penerimaan); tetapi secara umum, menurut kami mereka sebagian besar berguna untuk Unit, Integrasi, dan Pengujian Sistem.

Terkadang pengujian semacam ini bisa memakan waktu dan sumber daya, jadi pastikan Anda membidik target yang tepat—sedikit bertentangan dengan apa yang telah kami katakan sebelumnya, karena setiap perubahan kecil mungkin merusak kode Anda, namun terkadang Anda akan mengalaminya. untuk mengambil beberapa risiko jika Anda ingin mengirimkan perubahan dengan cepat. Salah satu cara untuk mengatasi masalah ini adalah dengan berinvestasi dalam otomatisasi. Otomatiskan semuanya! Pada setiap push cabang, jalankan pengujian unit dan integrasi, jalankan pengujian sistem setiap malam, dan tinggalkan hanya apa yang tidak cocok untuk otomatisasi ke pengujian manual manusia—misalnya, pengujian penerimaan.

Siapa Menguji?

Sebanyak ada berbagai jenis pengujian, ada berbagai jenis orang yang menguji. Pada titik ini kita harus menjawab pertanyaan: kapan pengujian dimulai? Seperti yang mungkin sudah Anda duga, mengingat Anda telah membaca bab-bab sebelumnya, pengujian suatu produk dimulai dari tahap yang sangat awal, kadang-kadang bahkan sebelum produk itu ada. Ketika kami mulai mendefinisikan persyaratan kami dan memikirkan tujuan dan kebutuhan bisnis, kami sudah tahu apa yang kami harapkan dari fungsionalitas produk; oleh karena itu, kita dapat membayangkan dan mendefinisikan bagaimana hal itu dapat diuji. Idealnya skenario pengujian didefinisikan sebelum implementasi. Ketika pemilik produk menentukan

fitur dan meneruskannya ke pengembang, mereka dapat dan seharusnya sudah menentukan kriteria penerimaan untuk setiap fitur dan skenario uji kasus. Sehingga pada akhir implementasi mudah untuk menguji apa yang telah dilakukan. Selama dan segera setelah implementasi, garis pertahanan pertama dibangun oleh pengembang. Pengembang harus menjamin bahwa kode dicakup oleh pengujian unit dan bahwa fitur berfungsi seperti yang dijelaskan oleh kriteria penerimaannya. Idealnya, mereka juga harus menerapkan beberapa alat otomatis yang memeriksa bahwa tidak ada fungsi sebelumnya yang rusak; tes unit sudah melakukan pekerjaan itu, tetapi penting untuk memeriksa setidaknya bahwa jalur bahagia tidak rusak.

Jalur bahagia adalah perjalanan penuh pengguna melalui produk Anda, membayangkan bahwa tidak ada masalah yang terjadi. jadi, misalnya, jika itu adalah platform e-niaga, jalur bahagia dimulai di halaman pendaftaran dan berakhir di halaman pembayaran yang berhasil.

Beberapa perusahaan sangat bergantung pada pengembang untuk melakukan pengujian mereka dan benar-benar percaya bahwa mereka dapat menjamin kualitas produk karena mereka dapat mengembangkan solusi otomatis apa pun untuk mengujinya. Dengan demikian, perusahaan-perusahaan ini tidak memiliki siapa pun selain pengembang yang menguji kode mereka. Pengembang menulis tes unit (mungkin beberapa tes otomatis), manajer produk menguji produk untuk beberapa jalur bahagia (mungkin bahkan di browser yang berbeda), dan pengujian selesai. Namun, kami percaya bahwa pengujian juga harus dilakukan oleh orang lain selain pengembang. Bahkan jika pengembangnya adalah insinyur yang brilian, pola pikir untuk menguji suatu produk bergantung pada keinginan untuk memecahkannya.

Ingat tes mobil itu—mereka sedang mengalami kecelakaan nyata. Ini karena ketika mobil berada di luar sana, apa pun bisa terjadi padanya; oleh karena itu penting untuk menguji bagaimana reaksinya terhadap crash dan kondisi tak terduga lainnya. Tidak ada perbedaan antara mobil dan perangkat lunak yang kami buat. Setelah kami mengirimkannya, itu akan berjalan di jalan yang sama sekali tidak dikenal yang penuh dengan tindakan tak terduga yang datang dari pengguna kami. Itulah mengapa untuk mengujinya dengan benar, kita perlu berusaha keras untuk memecahkannya. Lebih baik kami menemukan pelanggaran sendiri dan memperbaikinya segera daripada pengguna kami menjalankan perangkat lunak yang rusak. Nah, dengan itu, izinkan saya mengajukan pertanyaan.

Ketika Anda membangun sesuatu, seberapa besar kemungkinan Anda akan bersedia untuk menghancurkannya? Bayangkan furnitur IKEA dan bayangkan diri Anda menghabiskan akhir pekan penuh untuk merakitnya sepotong demi sepotong, bagian demi bagian. Apakah Anda bersedia untuk memecahkannya? Apakah Anda bersedia memukulnya dengan palu untuk melihat seberapa andalnya itu? Tentu saja, Anda tidak akan melakukannya! Bahkan jika Anda diminta untuk melakukan sesuatu yang berbahaya pada furnitur Anda untuk mengeksplorasi keandalannya, Anda akan melakukannya dengan penuh perhatian dan cinta. Pengembang melakukan hal yang sama dengan kode mereka. Sangat sulit bagi pengembang untuk menerapkan tindakan berbahaya apa pun pada kode yang telah mereka bangun dengan sangat hati-hati dan penuh cinta. Itulah mengapa kami percaya bahwa

mengandalkan pengembang untuk menguji pekerjaan mereka sendiri adalah salah. Itulah sebabnya kami termasuk orang-orang sekolah tua yang percaya bahwa setiap tim harus memiliki Insinyur Penjaminan Mutu, atau penguji yang berdedikasi. Ada berbagai jenis penguji. Salah satu bentuk pengujian produk yang paling sederhana adalah memeriksanya secara manual. Itu sebabnya kami memiliki penguji manual.

7.7 Penguji QA Manual

Penguji manual adalah mereka yang akan menghabiskan waktu berinteraksi secara manual dengan produk, mencoba membuat skenario di mana fitur dapat rusak. Misalnya, pertimbangkan formulir login. Pengembang mungkin akan mengujinya dengan memasukkan email dan kata sandi yang benar dan mengklik tombol "Masuk", sedangkan penguji yang baik akan mengetikkan omong kosong apa pun di bidang input. Apakah Anda tahu lelucon terkenal tentang seorang insinyur QA ini?

seorang insinyur Qa masuk ke sebuah bar. Memesan bir. Memesan 0 bir. Memesan 9999999 bir. Memesan -1 bir. Memesan bir asdqweqads.

Oh ya, penguji yang baik luar biasa kreatif. Ada orang yang beruntung bisa bekerja sama dengan kami yang akan menghasilkan skenario yang benar-benar gila. Kadang-kadang dia akan datang kepada kami dan mengatakan sesuatu seperti: Jadi, saya telah mengetik ini di kolom input "QWADASDasodjaioqwehyq2o239190q283qasdasd", menyegarkan halaman, mengklik tombol "kembali", dan menghentikan server pada saat yang sama, dan kemudian ini kesalahan terjadi.

Dan Anda hanya akan menatapnya dengan mata terbuka lebar dan berpikir: bung, apa yang baru saja Anda hisap? Bisakah saya juga memilikinya? Omong-omong, orang ini sedang mengelola tim yang terdiri dari 10 insinyur QA di salah satu perusahaan rintisan terbesar di Berlin saat ini. Rasanya menyenangkan memiliki penguji manual di tim Anda, dan itu berkontribusi pada kualitas keseluruhan bukan hanya karena mereka menguji perangkat lunak Anda, tetapi juga karena pengembang mulai menulis kode yang lebih baik. Mengapa demikian? Kami akan menceritakan sebuah cerita.

Suatu kali, Solikhan bekerja sebagai pengembang frontend di tim yang memiliki insinyur QA manual. Setiap kali insinyur akan pergi ke Solikhan, mengatakan: "Solikhan, saya punya pertanyaan ..." Solikhan akan mulai gemetar karena biasanya pernyataan yang mengikuti pertanyaan akan berisi bug yang diperkenalkan oleh kode Solikhan. Solikhan sangat ingin memberikan kualitas terbaik hanya untuk tidak mendengar pertanyaan ini lagi dan lagi, jadi dia tidak hanya akan meningkatkan teknik pengkodeannya tetapi juga proses keseluruhan dalam tim, seperti tinjauan kode dan pemrograman berpasangan.

7.8 Penguji QA Otomatisasi

Tentu saja, pemeriksaan manual itu penting; namun, terkadang tugas untuk penguji manual menjadi sangat berulang. Ini menjadi sangat jelas dan mengkhawatirkan dalam hal pengujian regresi.

Sekedar mengingatkan, pengujian regresi berkaitan dengan memeriksa fungsionalitas penting yang harus dijalankan setiap kali fitur baru diperkenalkan untuk memastikan

bahwa fitur ini tidak merusak apa pun. tes untuk fitur ini ditambahkan ke rangkaian tes regresi.

Bayangkan Anda memiliki 10 tes regresi, yang masing-masing membutuhkan waktu 5 menit untuk diperiksa. Bayangkan Anda harus mendukung dua browser: IE dan Chrome. Mari kita bahkan tidak membahas versi browser ini, dengan fokus hanya pada versi terbaru. Jadi, untuk sepenuhnya menguji fungsionalitas aplikasi, Anda memerlukan lebih dari 2,5 jam pengujian manual. Jika itu adalah pekerjaan satu kali saja, itu dapat diterima, tetapi produk Anda tidak membeku dalam waktu—ini sedang dikembangkan dan ditingkatkan, dan fitur-fitur baru sedang diperkenalkan. Oleh karena itu, regresi harus dijalankan cukup sering, dan seiring waktu akan menjadi pemborosan sumber daya yang besar.

Ada banyak hal yang dapat diotomatisasi dalam proses ini. Misalnya, jika kita berurusan dengan aplikasi web, kita dapat menulis pengujian otomatis menggunakan kerangka kerja yang mensimulasikan interaksi browser dan memerintahkan mereka untuk memeriksa bahwa sebagai hasil dari beberapa interaksi tertentu, hasil tertentu harus muncul di halaman. Penguji otomatisasi adalah mereka yang menentukan skenario dan menulis tes ini. Penguji otomatisasi tahu cara menulis kode, menggunakan kerangka kerja, dan memanfaatkan infrastruktur untuk mensimulasikan lingkungan langsung. Misalnya, pengujian otomatisasi untuk menguji fungsionalitas login dapat ditulis seperti berikut:

```
open("/login");
$("input#login")).setValue("test@test.com");
$("button#login").click();
$("#username").shouldHave(text("Hello, test!")); //  Waits until element gets text
```

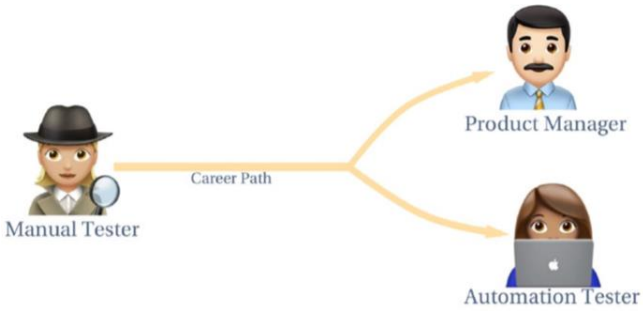
Bergantung pada kerangka kerja yang digunakan untuk pengujian otomatisasi, mereka dapat ditulis dalam beberapa bahasa pemrograman. Selenium biasanya digunakan (https://www.seleniumhq.org/) sebagai kerangka kerja untuk pengujian otomatis yang terutama ditulis dalam Java. Oleh karena itu, ada banyak penguji otomatisasi yang mengkhususkan diri dalam menulis tes Selenium otomatis di Jawa. Namun, ada pembungkus untuk Selenium dalam bahasa pemrograman yang berbeda, termasuk JavaScript. Terkadang mereka bahkan lebih mudah digunakan. Cukup mudah untuk memilih apa yang akan digunakan—jika Anda memiliki penguji otomatisasi khusus, beri mereka kebebasan untuk memilih bahasa dan kerangka kerja yang nyaman bagi mereka. Jika pengujian otomatisasi Anda harus ditulis oleh pengembang, maka mereka akan menulisnya dalam bahasa pilihan mereka. Kami telah bekerja di perusahaan yang memiliki insinyur QA khusus yang menulis seluruh rangkaian pengujian di Java, sedangkan kami juga bekerja di perusahaan di mana pengembang frontend bertanggung jawab untuk menulis pengujian otomatis, atau yang disebut end-to-end, dalam JavaScript. Fleksibilitas untuk menang!

Dari Manual hingga Penguji Otomatis

Bagaimana penguji QA manual menjadi penguji otomatisasi? Nah, belajarlah banyak. Penting bagi Anda untuk mengetahui cara menulis kode. Penting bagi Anda untuk memahami dasar-dasar tentang cara kerja protokol web (mis., HTTP). Penting juga untuk memiliki pemahaman dasar tentang alat infrastruktur, skrip, server, dan basis data. Idealnya, pertama-tama Anda menyelami mentalitas pengujian secara mendalam dan kemudian

selangkah demi selangkah Anda mempelajari dasar-dasar ilmu komputer dan mengkhususkan diri Anda pada otomatisasi.

Namun, ada jalur karir lain untuk penguji manual, dan kami sudah sering melihatnya terjadi (Gambar 7-1). Perhatikan bahwa penguji bekerja sama dengan manajer produk untuk memahami persyaratan, menentukan kriteria penerimaan, menentukan berbagai cara produk harus diuji untuk menjamin bahwa semua browser yang didukung tercakup, dll. Dengan demikian, seiring waktu, mereka mengembangkan sangat tajam dan baik pemahaman tentang produk, peringatannya, strategi pertumbuhan, tujuan, dan kebutuhan. Pada titik waktu tertentu, orang-orang ini menemukan diri mereka di posisi manajer produk atau pemilik produk. Jadi, jika Anda seorang penguji manual, pada titik tertentu dalam karier Anda, Anda mungkin akan dihadapkan pada dua kemungkinan arah: pengujian otomatisasi atau manajemen produk. Keputusan akan terjadi secara alami, dan Anda bahkan tidak akan menyadarinya—jika Anda menyukai alat, infrastruktur, skrip, dan teknologi, Anda sebagian besar akan memilih jalur penguji otomatisasi; jika Anda menikmati proses, pengembangan dan peningkatan produk, definisi fitur, dan strategi produk, maka Anda harus mengikuti jalur manajer produk.



Gambar 7-1. Jalur karir seorang penguji manual

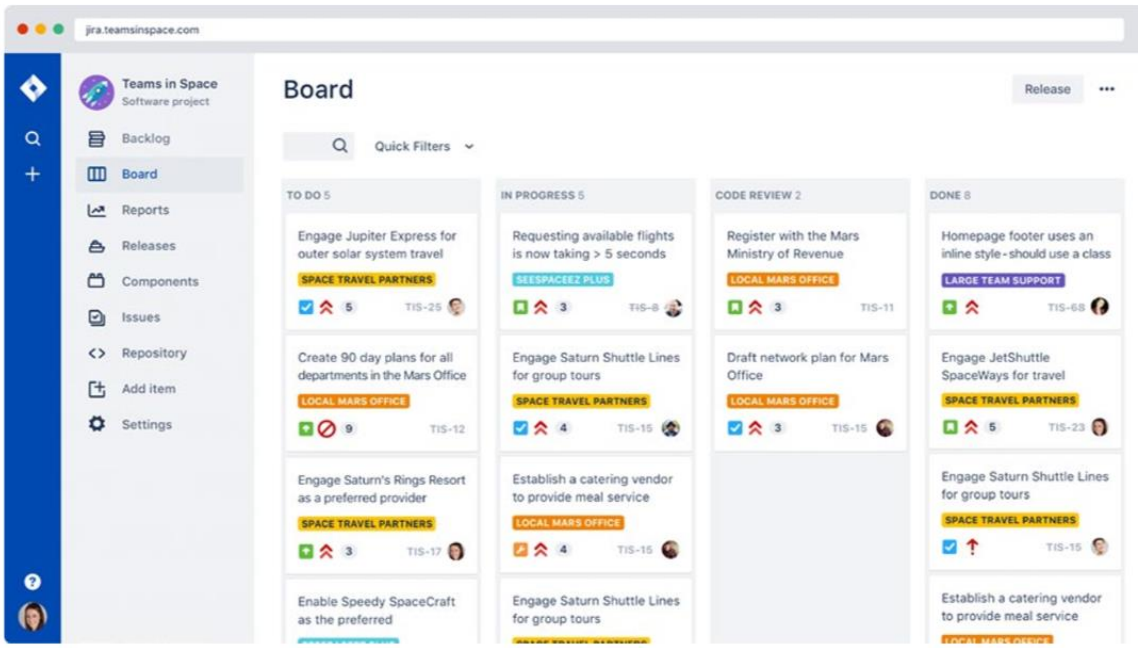
Jika Anda memutuskan untuk mengikuti jalur penguji otomatisasi, mulailah menggali bahasa pemrograman dengan fokus pada kerangka pengujian. Pelajari konsep dasar arsitektur client-server, protokol komunikasi, dan alat web dan seluler yang memungkinkan aplikasi debugging dan pemantauan. Pada titik tertentu Anda mungkin memutuskan untuk mendapatkan beberapa sertifikasi. Yang paling terkenal untuk penguji disebut ISTQB (International Software Testing Qualifications Board; https://www.istqb.org/). Sertifikasi ini memiliki level dan sublevel yang berbeda. Pusat sertifikasi ISTQB ada di seluruh dunia dan ujiannya harus dibayar.

Anda mungkin menemukan banyak sumber daya persiapan online. Jika Anda merasa topik ini menarik bagi Anda, kami sarankan Anda menggunakan alat persiapan tersebut. Bahkan jika Anda memutuskan untuk tidak mengikuti sertifikasi, ini adalah kesempatan yang sangat baik untuk belajar, tidak hanya dari perspektif pengujian itu sendiri tetapi juga untuk memahami proses dan standar yang digunakan dalam dunia pengembangan perangkat lunak.

7.9 Alat, Platform, dan Framework

Ada banyak alat yang dapat mempermudah kehidupan penguji QA. Ini berkisar dari alat yang memungkinkan penulisan spesifikasi pengujian manual hingga kerangka kerja

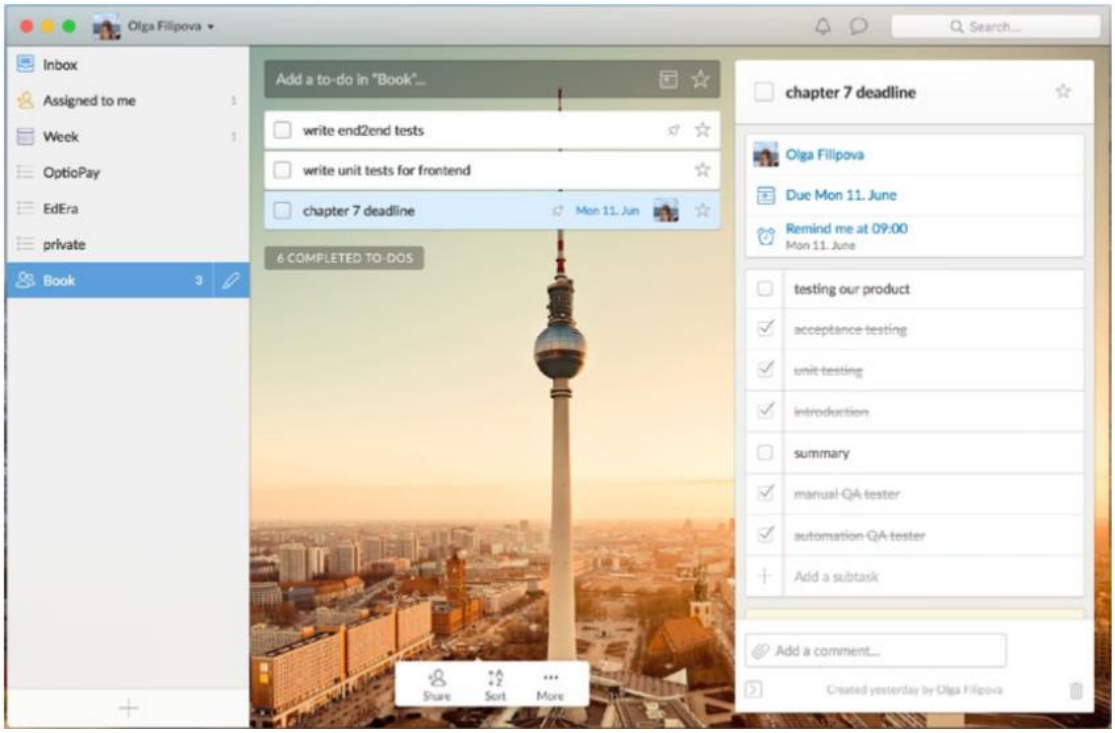
kompleks yang digunakan untuk menulis kode untuk pengujian otomatis. Sebelum kita mulai membahas alat untuk manajemen pengujian, mari kita bahas alat untuk manajemen tugas karena tes terkait dengan beberapa tugas pengembangan. Salah satu platform manajemen tugas paling terkenal untuk tim perangkat lunak tangkas adalah JIRA (https://www.atlassian.com/software/jira). Ini dikembangkan oleh Atlassian, dan sangat kuat sehingga kita bahkan tidak bisa membayangkan apa yang tidak bisa dilakukan. Gambar 7-2 menunjukkan tampilan papan JIRA.



Gambar 7-2. Papan JIRA untuk manajemen tugas

Pengembang dari seluruh dunia menyukai dan membenci JIRA. Ini adalah cinta yang besar karena, seperti yang telah kami sebutkan, itu sangat kuat dan tidak ada yang tidak bisa dilakukan. Pada saat yang sama, itu mengerikan dan cukup rumit untuk mengonfigurasinya dengan benar sesuai kebutuhan. Terkadang Anda berpikir bahwa Anda memperkenalkan perubahan kecil yang akan memengaruhi bagian yang sangat spesifik dari konfigurasi proyek, dan tiba-tiba, setiap pengguna terpengaruh oleh perubahan Anda! Atau, Anda mengubah filter untuk papan, lupa mengubah privasi filter (tidak jelas bahwa itu harus diubah), dan tim tidak dapat melihat papan lagi. Kami punya banyak cerita menarik dan lucu terkait pengelolaan JIRA.

Beberapa perusahaan bahkan mengundang agen eksternal untuk mengadakan lokakarya bagi karyawan tentang cara menggunakan JIRA. Namun, ini terintegrasi dengan hampir semua hal, dan banyak alat terintegrasi dengannya, menjadikannya kandidat yang jelas. Terkadang menggunakan JIRA untuk proyek kecil mungkin tampak seperti kita mencoba membunuh lalat dengan meriam, dan untuk itu kita bisa menggunakan solusi lain yang lebih sederhana. Wunderlist dan Trello, misalnya, adalah alat yang sangat ringan dan mudah digunakan. Anda dapat menggunakannya sebagai alat berbasis web serta aplikasi mandiri. Gambar 7-3 menunjukkan seperti apa Wunderlist.



Gambar 7-3. Daftar Wunder

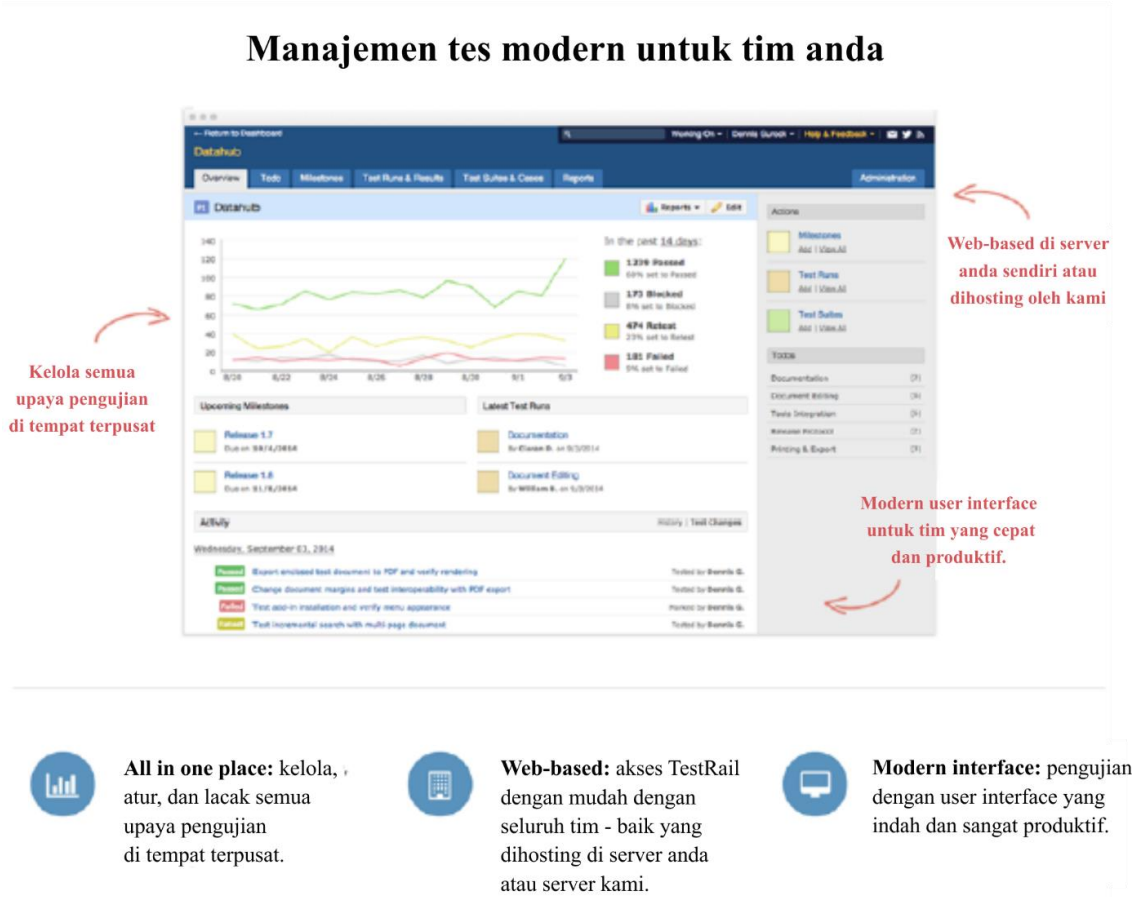
Ya, inilah latar belakang menara TV terkenal di Berlin. Soalnya, aplikasi tersebut dikembangkan oleh startup asal Berlin bernama 6Wunderkinder. Itu diakuisisi oleh Microsoft, tetapi masih terus menjadi produk yang hebat dan mudah digunakan. Perangkat lunak apa yang digunakan untuk mengelola tugas Anda terserah Anda. Yang dapat kami sampaikan kepada Anda adalah bahwa alat seperti Wunderlist dan Trello dapat digunakan untuk tugas sederhana dan tidak hanya terkait perangkat lunak, seperti "beli roti". Trello memungkinkan memiliki banyak papan dan tim; oleh karena itu, dapat digunakan untuk proyek yang lebih kompleks tetapi masih dapat digunakan untuk penggunaan pribadi. Kami tidak mengenal siapa pun yang menggunakan JIRA untuk tugas pribadi kecil.

JIRA datang ke panggung ketika Anda berurusan dengan proyek perangkat lunak besar yang dikembangkan dalam tim yang gesit. Sekarang setelah kita memiliki gambaran kasar tentang alat apa yang dapat digunakan untuk mendefinisikan tugas dan fitur, mari kita bicara tentang alat yang dapat digunakan untuk mendefinisikan kasus uji. Tentu saja, Anda dapat menulis kasus pengujian Anda dalam file teks sederhana. Anda juga dapat menggunakan beberapa editor yang lebih terstruktur seperti Excel. Atau Anda dapat menggunakan beberapa perangkat lunak yang dirancang dan dibuat khusus untuk tugas tersebut.

Alat-alat ini membantu kami menyusun kasus uji, langkah, dan hasil. Anda dapat menentukan siklus rilis dan mengikat alat tersebut ke continuous integration dan continuous delivery pipeline. Saat Anda mengerjakan spesifikasi kasus uji, Anda bekerja sama dengan manajer produk dan tim pengembangan karena kasus uji harus mencerminkan nilai bisnis dan risiko fitur yang diuji. Pengembang juga harus terlibat, karena mereka dapat memberikan masukan berharga tentang jumlah upaya yang diperlukan untuk

mengimplementasikan beberapa hasil yang diharapkan atau dampak pengembangan jika fitur tidak berfungsi seperti yang diharapkan.

Ingat, cukup penting bahwa apa pun yang Anda gunakan untuk spesifikasi pengujian, Anda entah bagaimana mengintegrasikannya dengan alat manajemen tugas Anda; jika tidak, akan sangat sulit untuk mengelola dan mengaitkan tugas Anda dengan kasus uji yang sesuai. Ada kemungkinan untuk menentukan kasus uji sebagai bagian dari tugas, tetapi kemudian, ketika tugas SELESAI, pengujian juga akan dianggap selesai, dan Anda harus menjalankannya berulang kali sebagai bagian dari rangkaian pengujian regresi. Untuk JIRA ada plugin bernama Zephyr (https://bit.ly/2mpiThF). Alat ini memungkinkan pengujian yang ditentukan, melampirkannya ke tugas yang sesuai, dan menjalankannya kapan pun diperlukan. Alat serupa lainnya disebut testrail (http://www.gurock.com/testrail/). Ini adalah solusi berbasis web yang terintegrasi mulus dengan JIRA dan menyediakan segala macam metode untuk menangani skenario pengujian Anda, termasuk semua kemungkinan visual seperti pelaporan dan diagram (Gambar 7–4).



Gambar 7-4. Testrail (https://bit.ly/1TrocHO)

Menentukan kasus pengujian saja tidak cukup—seseorang perlu menjalankan pengujian tersebut dan memberi tahu sistem saat pengujian gagal atau lulus. Pengujian ini dapat ditentukan dan dijalankan secara manual, bukan oleh penguji QA manual atau pemilik produk. Beberapa tes dapat diotomatisasi. Tentu saja, ada banyak alat dan kerangka kerja untuk mengotomatiskannya, mulai dari yang membutuhkan keterampilan pengkodean hingga yang dapat diterapkan oleh manajer produk. Salah satu kerangka kerja paling populer

untuk pengujian UI dan mungkin yang pertama akan Anda dengar jika Anda masuk ke bidang ini adalah Selenium.

Selenium adalah seperangkat alat yang dikembangkan di Java yang memungkinkan mengeksekusi semua jenis interaksi web menggunakan bahasa pemrograman Java. Salah satu alat ini adalah server Java mandiri yang memungkinkan koneksi ke browser apa pun melalui API sederhana. Alat lain adalah kerangka itu sendiri—pustaka Java yang berisi berbagai macam perintah untuk mengeksekusi apa pun yang dapat Anda bayangkan di browser, mulai dari membuka browser di URL tertentu dan melangkah lebih jauh dengan mengklik, mengarahkan kursor, mengetik, menyeret elemen, dll. Karena interaksi sederhana dengan browser saja tidak cukup, Selenium juga menyediakan beberapa cara bagi kita untuk memeriksa kondisi tertentu untuk menegaskan pengujian. Anda dapat menggunakan pustaka pernyataan apa pun bersama dengan Selenium untuk menegaskan apa pun yang perlu Anda tegaskan. Mari kita pertimbangkan, misalnya fungsi login. Pertimbangkan bahwa kami ingin menegaskan bahwa setelah login berhasil, nama pengguna muncul di beberapa kotak tertentu di header situs web. Untuk tes ini, kita perlu memasukkan nama pengguna dan kata sandi, klik tombol, dan periksa teksnya. Kode Java untuk pengujian ini menggunakan perpustakaan Selenium akan terlihat seperti berikut:

```
webDriver.findElement(By.cssSelector(("input[name=username]"))).
sendKeys(username);
webDriver.findElement(By.cssSelector(("input[name=password]"))).
sendKeys(password); webDriver.findElement(By.cssSelector(".loginBtn")).click();
assertTrue(webDriver.findElement(By.cssSelector(".userbox")).getText().
contains(username));
```

Agak ribet, tapi masih bisa dibaca. Ada banyak kerangka kerja dan pembungkus yang membuat hidup Anda lebih mudah. Misalnya, ada pembungkus untuk Selenium yang disebut Selenide (http://selenide.org/). Itu membungkus rantai besar pernyataan Selenium yang jelek menjadi yang elegan dan ramah manusia. Tes yang sama menggunakan Selenide dapat ditulis sebagai berikut:

```
$("input[name=username]").setValue(username);
$("input[name=password]").setValue(password);
$(".loginBtn").click();
$(".userbox").shouldHave(text(username));
```

Bukankah itu luar biasa? Anda mungkin lebih suka bahasa lain untuk pengkodean tes UI. Ada pembungkus selenium untuk banyak dari mereka. Misalnya, untuk JavaScript ada solusi yang sangat bagus dan elegan yang disebut Nightwatch (http://nightwatchjs.org/). Jika Anda seorang pengembang JavaScript, Anda tidak perlu keluar dari zona nyaman Anda dan menggunakan bahasa pemrograman lain untuk menguji UI Anda. Anda dapat, dan harus, menggunakan bahasa yang sama untuk menulis kode dan pengujian Anda. Sintaks Nightwatch sangat mirip dengan Selenide.

Untuk menulis tes untuk fungsionalitas login menggunakan Nightwatch, Anda akan melakukan sesuatu seperti berikut:

```
.setValue('input[type=email]', username)
.setValue('input[type=password]',password)
```

```
.click('.loginBtn')
.assert('.userbox').toHaveText(username)
```

Siapa yang menulis tes otomatis? Seperti yang telah kami tunjukkan, terkadang ada insinyur QA otomatis khusus yang menerapkan otomatisasi, terkadang itu harus dilakukan oleh pengembang. Beberapa perusahaan berinvestasi di departemen QA, perusahaan lain memupuk budaya "pengembang bertanggung jawab atas kualitas." Kami telah melihat departemen QA terdiri dari satu anggota yang akan melakukan segalanya—mulai dari pengujian manual hingga mendefinisikan infrastruktur kompleks untuk menjalankan pengujian otomatis yang ditulis oleh mereka setiap kali rilis baru terjadi.

Kami juga telah melihat departemen QA yang cukup besar yang terdiri dari penguji manual, penguji otomatis, manajer rilis, dan personel infrastruktur yang berpengalaman. Kami juga tidak melihat departemen QA sama sekali—pengembang akan menulis semua pengujian, dan personel infrastruktur akan membantu menyiapkan alat yang benar untuk menjalankannya setiap kali mereka perlu menjalankannya. Jika tidak ada departemen QA dan tim tidak memiliki sumber daya yang cukup untuk menulis pengujian otomatis, manajer produk dapat menggunakan beberapa alat visual yang menggunakan Selenium (atau kerangka kerja lainnya) di belakang layar untuk pengujian UI. Misalnya, Katalon studio (https://www.katalon.com/) adalah kerangka kerja yang kuat yang memungkinkan perekaman interaksi browser yang dapat diputar ulang setiap kali Anda membutuhkan, menegaskan bahwa hasil dari proses tersebut sama seperti di awal. waktu perekaman. Contoh lain adalah Pingdom (https://www.pingdom.com/). Alat ini dirancang untuk pemantauan situs web, dan salah satu fiturnya disebut pemantauan transaksi (https://www.pingdom.com/product/transaction-monitoring/). Alat ini tidak gratis, tetapi mudah digunakan. Kelemahannya, bagaimanapun, adalah hanya menjalankan tes di Chrome, dan jika Anda benar-benar perlu menjalankan pemeriksaan aplikasi Anda di browser lain, Anda harus melakukannya sendiri.

Karena kami menyebutkan pengujian lintas-browser, kami harus berbicara tentang Browserstack (https://www.browserstack.com/). Ini adalah alat berbasis web yang kuat untuk pengujian lintas-browser dan lintas-perangkat. Ini tidak gratis, tetapi menawarkan berbagai kemungkinan, mulai dari pengujian manual menggunakan browser jarak jauh hingga pengujian otomatis. Pada dasarnya, Anda dapat menulis pengujian UI otomatis menggunakan kerangka Selenium apa pun dan menghubungkannya ke akun Browserstack Anda, menentukan platform dan browser untuk menjalankan pengujian Anda. Alat lain yang mirip dengan sumber daya Browserstack yang sangat populer saat ini adalah Sauce Labs (https://saucelabs.com/).

Jika Anda tidak ingin menghabiskan uang dan berinvestasi di Browserstack atau alat serupa untuk pengujian lintas-browser, tetapi Anda masih memerlukan pengujian untuk dijalankan di beberapa browser, Anda dapat mengatur sendiri kisi Selenium. Anda harus menyiapkan mesin (fisik atau virtual) dengan sistem operasi dan browser yang Anda perlukan untuk menguji aplikasi Anda. Kemudian Anda harus mengatur instance server Selenium Anda untuk berfungsi sebagai hub yang terhubung ke mesin yang berbeda ini. Idealnya Anda harus mengonfigurasi beberapa infrastruktur yang memungkinkan Anda

menjalankan semua pengujian sesuai permintaan. Konfigurasi ini memerlukan beberapa pengalaman pemrograman dan DevOps. Biasanya dilakukan oleh insinyur QA atau pengembang / personel DevOps yang berpengalaman. Gambar 7-5 adalah tangkapan layar perbandingan beberapa alat dari presentasi yang pernah dilakukan Solikhan untuk tim frontendnya sehingga mereka dapat memutuskan alat apa yang dapat mereka gunakan untuk pengujian UI.

	Bisa dilakukan oleh siapa saja	Mudah diinstall & dikonfigurasi	Dapat diinte-grasikan dalam pipa CI/CD	Fleksibel	Mudah digunakan	x-browser, x-device
Nightwatch + Selenium Grid	✗	✗	✓	✓	✓	✓
Nightwatch + browserstack	✗	✓	✓	✓	✓	✓
Pingdom	✓	✓	✗	✗	✓	✗
Katalon	✓	✗	✓	✓	✓	✓

Gambar 7-5. Perbandingan berbagai alat untuk menjalankan pengujian otomatis

Jika Anda tidak ingin mempekerjakan penguji QA atau mengalokasikan sumber daya untuk pengujian, Anda selalu dapat memutuskan untuk menggunakan platform "pengujian orang banyak", yang sangat populer saat ini. Pada dasarnya, platform ini menghubungkan aplikasi Anda ke ribuan penguji di seluruh dunia. Anda dapat mendaftar sebagai penguji dan menghabiskan hari Anda dengan mengklik tombol proyek yang berbeda dan mengisi laporan bug, atau Anda dapat mendaftar sebagai pemilik bisnis dan memberi tahu apa, bagaimana, dan kapan Anda memerlukan pengujian untuk dilakukan. Beberapa contoh platform tersebut termasuk test.io (https://test.io/) dan hutan hujan QA (https://www.rainforestqa.com/). Seperti yang Anda lihat, ada banyak pilihan di luar sana. Tidak masalah apa yang Anda pilih — yang terpenting adalah Anda menemukan keseimbangan yang tepat antara menghabiskan waktu dan menghabiskan uang untuk menguji apa yang benar-benar penting untuk diuji. Penting bagi Anda untuk menemukan hambatan dan menentukan dengan tepat apa yang perlu diotomatisasi dan apa yang dapat diuji secara manual. Penting bagi Anda untuk menilai semua risiko dan dampak dari pendekatan pengujian yang berbeda. Juga sangat penting bahwa setidaknya jalur bahagia produk Anda tidak pernah gagal. Inilah yang membuat pengguna Anda loyal terhadap produk Anda.

7.10 Menguji Produk Kita

Kami telah membahas banyak tentang berbagai jenis pengujian dan orang yang berbeda yang berpartisipasi dalam berbagai jenis pengujian. Cukup berdiskusi, kami memiliki beberapa pekerjaan yang harus dilakukan. Kami harus menguji platform kami. Di bagian

sebelumnya, Anda telah memeriksa cara membuat pengujian unit dan integrasi untuk modul backend. Sekarang mari kita bahas beberapa kode frontend dengan pengujian unit, menjalankan beberapa pengujian manual, dan menulis beberapa pengujian otomatis untuk antarmuka pengguna kita. Ingat, sekarang Anda akan mengenakan topi perusak. Jangan bersikap lembut terhadap produk Anda—tujuan utama Anda adalah menghancurkannya!

7.11 Pengujian Manual

Pengujian perangkat lunak secara manual cukup sering terjadi pada pengembang frontend. Saat mereka mengembangkan fitur baru, mereka selalu memeriksa halaman di browser untuk melihat hasil pekerjaan mereka. Namun, ini bukan pengujian yang adil—seperti yang telah kami sebutkan, kami pengembang menguji hasil positif dan tidak mencoba untuk benar-benar merusak apa yang baru saja kami lakukan. Jadi, jika kami menerapkan, katakanlah halaman login, kami pasti akan menguji apakah kami dapat mengautentikasi dengan kredensial yang benar, tetapi kecil kemungkinan kami mulai menguji skenario negatif seperti "memperkenalkan email tidak valid" atau "mengklik login tombol dengan bidang kata sandi kosong." Itulah mengapa sangat berguna untuk mendelegasikan pengujian manual kami kepada orang lain, seperti pemilik produk atau penguji QA khusus.

Dalam kasus kami, kami cukup beruntung karena seorang teman kami saat ini sedang belajar bagaimana menjadi seorang penguji, dan dia dengan senang hati setuju untuk menguji platform kami. Dia adalah orang yang nyata bernama Natalia, dan dia saat ini bekerja sebagai pegawai bank dan ingin beralih ke IT. Dia memutuskan untuk merangkul perjalanan ini dengan menjadi penguji manual. Dia menghabiskan beberapa minggu membaca beberapa artikel, berbicara dengan orang-orang yang saat ini bekerja sebagai insinyur QA, dan mempelajari dinamika pekerjaan ini, dan sekarang dia siap untuk mengotori tangannya. Jadi, kami memberinya tautan yang menunjuk ke implementasi yang kami miliki pada tahap ini dan memintanya untuk menguji fungsionalitas pendaftaran dan masuk. Hasil pengujiannya adalah lembar spesifikasi kasus uji dan laporan bug. Mari kita lihat mereka.

7.12 Kasus Uji

Untuk fungsi login dan registrasi, Natalia memutuskan untuk menguji skenario yang menyenangkan— kasus ketika semua kolom disetel dengan benar—dan beberapa skenario negatif—kasus di mana beberapa kolom tidak valid (misalnya, format email tidak valid) atau bahkan kosong. Dia menuliskan judul tes, langkah-langkah tes, dan perilaku yang diharapkan. Juga, dan sangat penting, dia menuliskan sistem operasi dan browser yang dia gunakan untuk pengujian. Menentukan hal-hal ini memudahkan pengembang untuk mereproduksi bug di kemudian hari, karena mungkin hanya terjadi dalam kondisi tertentu. Singkatnya, Gambar 7-6 menunjukkan tampilan tabel pengujiannya.

Test Case # dan Judul	Langkah Tes	Perilaku yang Diharapkan	Browser	Sistem Operasi

TC01 – Register Sukses	Klik pada tombol "Register"diatas halaman	Pengguna diarahkan ke halaman dasbor	Google Crhome 66.0.3359.139	Windows 10
	Lengkapo kolom yang wajib diisi		Internet explorer version 1111.431.16299.0	
	Klik tombol "Join Us"			
TC02 – Preview Password	Lengkapi kolom Password Klik tombol "Show"	Pengguna dapat melihat string pengantar	Google Crhome 66.0.3359.139	Windows 10
TC03 – Invalid email	Isi email yang tidak valid		Internet explorer version 1111.431.16299.0	Windows 10
	Klik tombol "Masuk"	Pesan kesalahan ditampilkan dan pengguna tidak masuk	Google Crhome 66.0.3359.139	
TC04-Login dengan email kosong	Klik tombol "Login" di header halaman	Pesan kesalahan ditampilkan dan pengguna tidak masuk	Internet explorer version 1111.431.16299.0 Google Crhome 66.0.3359.139 Internet explorer version 1111.431.16299.0	Windows 10
	Kosongkan bidang email			
	Setel bidang kata sandi			
	Klik tombol "Masuk"			
TC05-Login dengan password kosong	Klik tombol "Login" di header halaman	Pengguna tidak dapat masuk	Google Crhome 66.0.3359.139 Internet explorer version 1111.431.16299.0	Windows 10
	Kosongkan bidang kata sandi			
	Isi kolom email			
	Klik tombol "Masuk"			
TC06-Login Sukses	Klik tombol "Masuk" di tajuk halaman		Google Crhome 66.0.3359.139	Windows 10

	Isi semua bidang wajib dalam formulir yang valid Klik tombol "Masuk"	Pengguna berhasil diautentikasi	Internet explorer version 1111.431.16299.0	

Gambar 7-6. Uji kasus untuk fungsi login dan register

Tentu saja, Anda dapat membuat beberapa skenario lain untuk diuji. Terserah Anda seberapa dalam Anda ingin pergi dengan pengujian. Itu juga tergantung pada sifat bisnis Anda. Jika kita berurusan dengan aplikasi perbankan yang sangat sensitif yang dapat menarik penipu dan peretas, maka Anda perlu memikirkan semua kemungkinan skenario dan kombinasinya. Jika Anda membuat aplikasi untuk teman Anda yang tidak berurusan dengan data yang masuk akal, maka Anda bisa lebih santai.

7.13 Laporan Bug

Saat mencoba mengikuti langkah-langkah yang dijelaskan dalam skenarionya, Natalia menemukan bahwa beberapa kasus tidak berperilaku seperti yang diharapkan. Khususnya, dia menyadari bahwa tidak ada umpan balik ketika data yang diberikan salah. Dia menandai kasus pengujian tersebut dengan warna merah di lembar kasus pengujiannya dan membuat laporan bug sederhana (Gambar 7-7).

Bug # dan Judul	Uji Kasus#	Langkah Produksi Ulang	Perilaku yang Diharapkan	Perilaku aktual	Sistem Operasi dan Browser
B01-Tidak ada pesan eror pada email yang tidak valid	TC03	Klik tombol login di bagian atas halaman Isi bagian kolom email dengan format email yang tidak valid (contoh : Test.com) Klik tombol “Login”	Error ditampilkan kepada User dan User tidak dapat login	Tidak ada feedbacnk, User tetap berada di halaman yang sama	Windows 10 Internet Explorer 11 dan Chrome 66
B2-Tidak ada pesan eror padaemail kosong	TC04	Klik tombol login di bagian atas halaman	Error ditampilkan kepada User dan User	Tidak ada feedbacnk, User tetap berada di	Windows 10 Internet Explorer 11

		Lewati kolom email Isi kolom password Klik tombol “Login”	tidak dapat login	halaman yang sama	dan Chrome 66

Gambar 7-7. Laporan bug sederhana untuk fungsi login

Tentu saja, jika Anda menggunakan perangkat lunak khusus yang dirancang untuk membuat kasus uji dan laporan bug, semuanya akan dihubungkan bersama, dan akan jauh lebih mudah bagi pengembang dan manajer produk untuk menindaklanjuti bug dan mengubahnya menjadi tugas. Bug dapat dan harus diprioritaskan dan prioritasnya untuk diperbaiki juga bergantung pada tingkat keparahan dan dampaknya.

Beberapa bug dapat dianggap sebagai showstoppers—tanpa memperbaikinya, sistem tidak dapat berfungsi. Beberapa bug dianggap sebagai prioritas rendah sehingga hanya berakhir di bagian bawah backlog selamanya, backlog purgatory. Beberapa bug dianggap… fitur! Ini tidak terduga, bukan? Sebenarnya, ini adalah salah satu lelucon favorit para programmer. Ketika mereka didekati oleh pemilik produk atau penguji yang menunjukkan beberapa bug, mereka berkata, "Ini bukan bug, itu adalah fitur!" Terkadang ternyata benar. Beberapa fungsionalitas yang diperkenalkan secara tidak sengaja ternyata sangat mengagumkan sehingga menjadi fitur baru. Namun, ini adalah kasus yang langka dan membahagiakan; dalam kebanyakan kasus, bug entah bagaimana berbahaya bagi sistem.

Saat memikirkan tingkat keparahan bug, selalu pikirkan pengguna Anda. Seberapa besar kemungkinan mereka akan menderita jika bug ini tidak diperbaiki? Seberapa besar kemungkinan mereka akan berhenti menggunakan layanan Anda? Misalnya, jika saya memiliki aplikasi jam alarm dan fungsi snooze rusak, saya mungkin akan bertahan tanpanya (sebenarnya, dalam beberapa kasus, kami dapat menganggap bug semacam ini sebagai fitur aktual karena akan membuat kami tepat waktu untuk janji temu kami.). Namun, jika fungsi dering akan rusak sama sekali, maka saya akan berhenti menggunakan aplikasi ini.

7.14 Pengujian Unit untuk Frontend

Anda sudah familiar dengan konsep unit test. Sebenarnya, Anda sudah melihatnya—ketika Anda membaca bab pengembangan backend. Sekarang kita akan mengambil kode di folder “before_frontend_tests” sebagai basis dan mengimplementasikan beberapa unit test pada aplikasi frontend. Kami akan menggunakan moka sebagai uji coba dan berharap sebagai pustaka pernyataan. Yang harus Anda lakukan adalah menginstalnya menggunakan npm atau benang:

```
cd courses/frontend
npm install --save-dev mocha-webpack mocha expect
// or
yarn add --dev mocha-webpack mocha expect
```

Langkah selanjutnya adalah menambahkan skrip pengujian ke package.json kami:

```
// package.json
{
<...>
"scripts": {
"dev": "nuxt",
"build": "nuxt build",
"start": "nuxt start",
"generate": "nuxt generate",
"test": "mocha-webpack --require tests/setup.js tests/unit/**/*.spec.js"
},
<...>
}
```

Sekarang mari tambahkan pengaturan dasar untuk pengujian dan mari mulai menambahkannya. Buat folder bernama tes dan buat file setup.js:

```
// tests/setup.js
global.expect = require('expect')
```

Mari kita buat contoh pengujian. Buat file bernama test.spec.js di folder tes/unit. Tambahkan tes palsu di sana:

```
describe('test', () => {
it('should run test', () => {
expect(1).toEqual(1)
})
})
```

Banyak kerangka pengujian dalam JavaScript menawarkan sintaks yang serupa. Gunakan fungsi yang dijelaskan untuk menentukan rangkaian pengujian besar untuk beberapa unit kode. Kemudian gunakan fungsi yang menerima spesifikasi pengujian kecil dan fungsi pengujian itu sendiri. Fungsi tes memanggil beberapa bagian dari unit yang sedang diuji dan menegaskan beberapa hasil. Mari, misalnya, tutup dengan unit test metode modul getter dari Vuex store:

```
// store/getters.js
import { find } from 'lodash'
export default {
courses: state => state.courses,
isAuthenticated: state => !!state.token,
auth: state => state.auth,
user: state => state.user,
course: state => find(state.courses, { id: state.courseId }) || {},
userCourse: state => state.userCourse,
userCourses: state => state.courses.filter(course => course.enrolled ===
true)
}
```

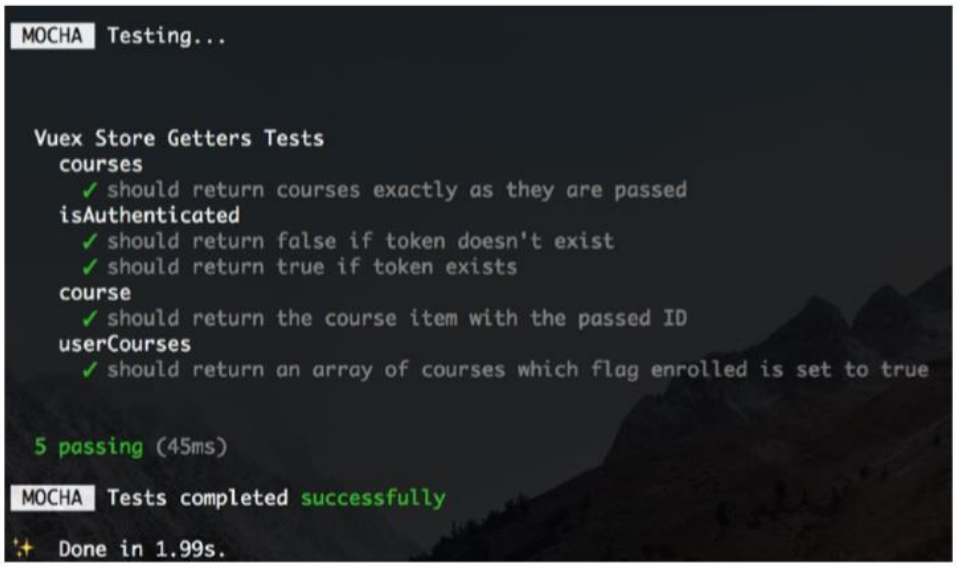
Metode-metode ini mudah untuk diuji karena kita dapat melewati objek statusnya, dan menurut apa yang kita lewati, kita dapat mengharapkan beberapa hasil yang spesifik.

Misalnya, untuk metode isAuthenticated, jika kita melewatkan objek status tanpa token, metode tersebut harus mengembalikan false, dan jika kita melewati objek status dengan token maka metode tersebut harus mengembalikan nilai true:

```
// tests/unit/store/getters.spec.js
describe('isAuthenticated', () => {
it('should return false if token doesnt exist', () => {
expect(getters.isAuthenticated({})).toBeFalsy()
})
it('should return true if token exists', () => {
expect(getters.isAuthenticated({token: 'asd'})).toBeTruthy()
})
})
```

Periksa kode di folder after_frontend_tests. Kami telah membahas hampir semua metode getter Vuex. Jika kita menjalankan pengujian menggunakan npm test atau yarn test, kita akan mendapatkan output di console seperti pada Gambar 7-8.

Bukankah menyenangkan melihat tanda centang hijau itu? Solikhan mencintai mereka! Idealnya, kami seharusnya menulis beberapa pengujian ujung ke ujung, tetapi kami akan mengerjakannya setelah kami memiliki UI yang lebih stabil. Namun, Anda dipersilakan untuk menggunakan Nightwatch atau alat lain apa pun untuk menentukan beberapa pengujian UI.



Gambar 7-8. Menjalankan tes unit pengambil toko Vuex

7.15 Kesimpulan

Dalam bab ini kita telah membahas betapa pentingnya menguji perangkat lunak kita, sehingga pengguna kita tidak pernah frustrasi dengan apa pun yang kita tawarkan kepada mereka. Kami membahas berbagai jenis penguji di luar sana dan berbagai jenis tes. Kami membahas tentang bug, tingkat keparahannya, dan prioritasnya. Kami harap Anda menyadari saat ini bahwa tidak ada perangkat lunak bebas bug di dunia. Semua aplikasi, sistem, dan produk memiliki beberapa bug. Beberapa produk memiliki bug yang jelas yang mencegah kami menggunakannya, beberapa memiliki bug kecil yang tersembunyi yang

bahkan tidak kami sadari. Proses berburu serangga bisa jadi menantang sekaligus menyenangkan. Proses menemukan penyebab bug bisa jadi membosankan dan membuat frustrasi. Terkadang kami menghabiskan beberapa hari mencoba mencari tahu mengapa beberapa bug terjadi, dan kemudian kami menyadari bahwa itu hanya kesalahan ketik bodoh yang dapat diperbaiki dalam hitungan detik! Terkadang kita tahu persis mengapa bug itu terjadi, tetapi untuk memperbaikinya kita perlu satu minggu penuh. Setiap kali kami menulis kode, kami berpotensi memperkenalkan bug baru. Itulah mengapa sangat penting untuk memiliki proses pengujian dan membuatnya terdefinisi dengan baik dan otomatis mungkin.

7.16 Tes Kemampuan

- Apa itu bug di dunia pengembangan perangkat lunak?
    - serangga yang menggigit kabel komputer
    - lalat kecil yang menggigit programmer dan mencegah mereka bekerja
    - kegagalan fungsi dalam sistem yang mencegahnya bekerja seperti yang diharapkan
    - perilaku sebenarnya dari fitur tersebut terhadap perilaku yang diharapkan
- Anda membuka skenario uji kasus dan mengikuti langkah-langkah yang telah Anda jelaskan sebelumnya: Anda membuka halaman web, klik tombol yang bertuliskan, “join us!,” isi formulir, klik tombol lain, dan periksa apakah yang berikutnya halaman menampilkan pesan sukses. Kamu adalah:
    - pemilik produk
    - penguji otomatisasi
    - penguji manual
    - seorang pengembang
    - siapa saja dari atas
    - siapa pun dari atas, tetapi penguji otomatisasi yang lebih kecil kemungkinannya
- Kode Anda memiliki fungsi yang meringkas dua angka. fungsi ini adalah kandidat yang sempurna untuk menjadi:
    - diuji terhadap injeksi sQL
    - Dicakup oleh unit test
    - diuji terhadap beberapa tolok ukur kinerja
    - Diuji secara manual di browser
- hari ini adalah hari besar! Kami merilis versi final dari produk kami. rilis telah diumumkan, dan semua pelanggan menantikannya. Anda menjalankan tahap akhir dari rangkaian tes regresi Anda. semua orang di tim Anda—terutama pengembang dan manajer produk—sangat senang menunggu lampu hijau Anda, dan tiba-tiba Anda menyadari bahwa pengujian di internet explorer gagal. setelah menggali sedikit, Anda menyimpulkan bahwa fungsi login rusak pada mis. Tindakan Anda:
    - Buat laporan bug, hentikan proses rilis, beri tahu manajer produk bahwa rilis tidak dapat dikirimkan sampai bug diperbaiki.

- Siapa yang peduli dengan penjelajah internet? ini bahkan bukan peramban. Mari bergerak maju dengan rilis!
- Periksa pangsa pengguna yaitu; jika tidak terlalu tinggi, mari kita lepaskan. jika lebih dari 10%, mari kita perbaiki bugnya terlebih dahulu.
- lepaskan, tetapi tulis di catatan rilis bahwa ada bug yang diketahui di mis. Perbaiki sesegera mungkin dan rilis versi minor.

Jawaban yang benar untuk pertanyaan pertama adalah “kerusakan pada sistem,” tetapi sebenarnya kata bug muncul karena serangga yang mengacaukan beberapa sirkuit, oleh karena itu memperkenalkan malfungsi ke dalam sistem. Tidak ada jawaban yang benar untuk pertanyaan kedua. Tentu saja, tindakan yang dijelaskan biasanya dilakukan oleh penguji manual, tetapi jika tidak ada penguji manual dalam tim, itu bisa siapa saja! bahkan penguji otomatisasi.

Satu-satunya jawaban yang benar untuk pertanyaan ketiga adalah bahwa fungsi tersebut harus dicakup oleh unit test. Pertanyaan keempat rumit, dan itu sebenarnya tergantung pada banyak faktor. terkadang Anda harus benar-benar kuat untuk mengatakan kepada semua orang, “itu tidak akan berlalu!” Karena jika sistem berjalan dengan beberapa bug kritis, pada akhirnya Anda akan bertanggung jawab untuk itu, karena Anda membiarkannya berlalu. namun, terkadang karena beberapa tujuan bisnis, bug sebenarnya dapat masuk ke produksi dan pengguna harus menyadarinya. jadi, semua jawaban kecuali yang mengatakan, "Siapa yang peduli dengan internet explorer?" dapat diaplikasikan.

Jika Anda memiliki pemikiran yang sama selama tes ini, maka selamat! Anda telah berhasil mengenakan topi penguji perangkat lunak! Sangat bagus bahwa sekarang kita tahu cara menguji perangkat lunak kita, tetapi akan lebih baik lagi jika kita bisa santai dan bahkan tidak memikirkannya. Alangkah baiknya jika setiap kali kami memperbarui produk kami, tes secara otomatis hanya berjalan sendiri. Akan sangat bagus jika alih-alih menjalankan platform kami secara lokal di mesin kami, kami dapat melihatnya beraksi dengan membuka beberapa URL yang bagus. Dan bagaimana jika pada setiap perubahan kita cukup menekan sebuah tombol dan kemudian halaman akan diperbarui, ditayangkan, dan pengujian akan dijalankan? Apakah itu sihir? Sebenarnya, itu akan menjadi kenyataan. Kita akan belajar di bab berikutnya bagaimana membangun realitas ini. Bab 8 dikhususkan untuk DevOps, infrastruktur, integrasi berkelanjutan, dan pengiriman berkelanjutan. Anda akan belajar cara menayangkan produk Anda dan cara mengotomatiskan proses penayangannya. Bukankah itu mengasyikkan? Ayo pergi!

BAB 8
AYO LIVE!

Pada bab sebelumnya kita membahas teknik pengujian yang berbeda untuk produk perangkat lunak kita. Kami menggunakan pendekatan yang berbeda untuk menguji platform pembelajaran kami dan kami dapat mengetahuinya sekarang: ok, produk kami diimplementasikan dan diuji, kami selesai. Hanya bercanda. Pertama, produk kita belum siap, masih membutuhkan banyak perhatian dan kasih sayang, dan kedua, belum bisa dikatakan "siap" jika kita tidak bisa menunjukkannya kepada siapa pun. Bagaimana Anda menunjukkannya?

Saat ini, satu-satunya cara Anda dapat menunjukkan pekerjaan Anda kepada teman-teman Anda adalah dengan mengundang mereka ke tempat Anda, membuka halaman web Anda langsung di browser komputer Anda, dan membuat presentasi kecil tentang apa yang telah Anda lakukan. Pasti teman-teman Anda akan menyukainya! Tapi… bagaimana jika Anda ingin membuat pekerjaan Anda tersedia untuk lebih dari teman Anda yang bisa datang ke tempat Anda? Bagaimana semua situs web ini berakhir di jaringan global, dan bagaimana kita dapat mengaksesnya hanya dengan mengetikkan alamat yang bagus di bilah peramban seperti "www.google.com"? Jika kami berhasil menerbitkan produk kami dan kemudian mengubah sesuatu, bagaimana perubahan ini disebarkan? Jika kami menemukan beberapa bug dan memperbaikinya, bagaimana kami membuat perbaikan segera tersedia secara online? Bagaimana kami menjamin bahwa pengujian kami lulus sebelum memublikasikan perubahan ke dunia? Bagaimana kita tahu jika produk kita sedang digunakan? Bagaimana kami tahu jika produk kami aktif dan berjalan? Jika ada lebih dari satu orang yang mengerjakan kode sumber perangkat lunak kami, bagaimana sinkronisasinya dijamin? Ini dan banyak pertanyaan lainnya akan dijawab dalam bab ini, yang dikhususkan untuk infrastruktur dan alat yang diperlukan untuk menjaga agar kode tetap sinkron, diuji, diterbitkan, dan mutakhir. Kami juga akan berbicara tentang para penyihir yang berdiri di pucuk pimpinan tanggung jawab ini. Anda akan memakai topi dari masing-masing karakter ini dan Anda akan merasa luar biasa dan kuat.

8.1 Bagaimana Cara Menerbitkan Proyek Perangkat Lunak Anda?

Sekarang setelah Anda mengimplementasikan proyek web Anda, lebih dari yang diharapkan Anda ingin mempublikasikannya. Untuk membuatnya tersedia bagi dunia, Anda harus menjalankannya di suatu tempat dan membuatnya dapat ditemukan. Anda sudah tahu bagaimana menjalankan proyek, bukan? Anda telah menerapkan server sederhana dan Anda dapat melayani halaman html dengannya. Apa artinya membuatnya dapat ditemukan?

Untuk dapat ditemukan berarti hanya dapat diakses dari komputer mana pun yang terhubung ke jaringan global.

Bagaimana kami membuat halaman kami dapat diakses di jaringan? Mesin tempat Anda menjalankan proyek memiliki alamat, seperti apartemen atau rumah Anda. Alamat ini

disebut alamat IP, dan begitulah cara mesin lain yang terhubung ke jaringan dapat menemukan alamat Anda. Biasanya ketika Anda menjalankan aplikasi Anda, Anda harus menentukan port untuk menjalankannya. Pelabuhan itu seperti nomor rumah di gedung. Dalam kasus platform kami, portnya adalah 8080. Sekarang alamat lengkap aplikasi Anda akan menjadi alamat IP mesin Anda diikuti oleh port—misalnya, 172.20.10.4:8080. Jika Anda memberikan alamat ini kepada teman Anda, mereka akan dapat melihat aplikasi Anda dari luar. Ya, itu indah, tetapi Anda tidak ingin komputer Anda menyala 24/7 hanya agar teman Anda dapat mengakses aplikasi Anda, bukan? Juga, bagaimana dengan URL? Apakah Anda dapat mengingat dan mengingat alamat IP situs web favorit Anda? Tentu saja tidak, dan selain itu alamat IP berubah dari waktu ke waktu untuk kontrak pribadi! Anda ingin memberikan nama yang berarti kepada teman Anda dan Anda ingin menjalankan aplikasi Anda di mesin lain yang bukan milik Anda! Mari kita mulai dengan menghosting layanan Anda di suatu tempat yang bukan komputer pribadi Anda. Pertama, Anda dapat membeli mesin khusus dan memasangnya di rumah Anda. Ini akan menjadi server bare-metal Anda sendiri.

Server bare-metal adalah mesin fisik khusus yang melayani kebutuhan satu penyewa saja. Istilah tersebut telah diperkenalkan untuk membedakan mesin fisik yang digunakan untuk meng-hosting layanan internet dari teman virtual dan cloud mereka.

Ingatlah bahwa jika Anda ingin menjalankan sesuatu di server Anda sendiri, Anda harus memiliki koneksi internet yang sangat kuat yang tidak pernah gagal. Anda harus yakin bahwa alamat IP mesin sudah diperbaiki, dan Anda harus yakin bahwa Anda melakukan investasi dengan mempertimbangkan kebutuhan Anda. Ingatlah bahwa Anda mungkin memerlukan layanan untuk ditingkatkan—dengan waktu Anda mungkin perlu memiliki lebih banyak ruang, lebih banyak memori, atau lebih banyak daya. Anda mungkin juga meng-host layanan Anda di layanan cloud. Anda memiliki banyak dari mereka; salah satu yang paling sering digunakan adalah AWS (Amazon Web Services) —di sana Anda dapat membayar server virtual, menyambungkannya, dan menjalankan layanan Anda di sana.

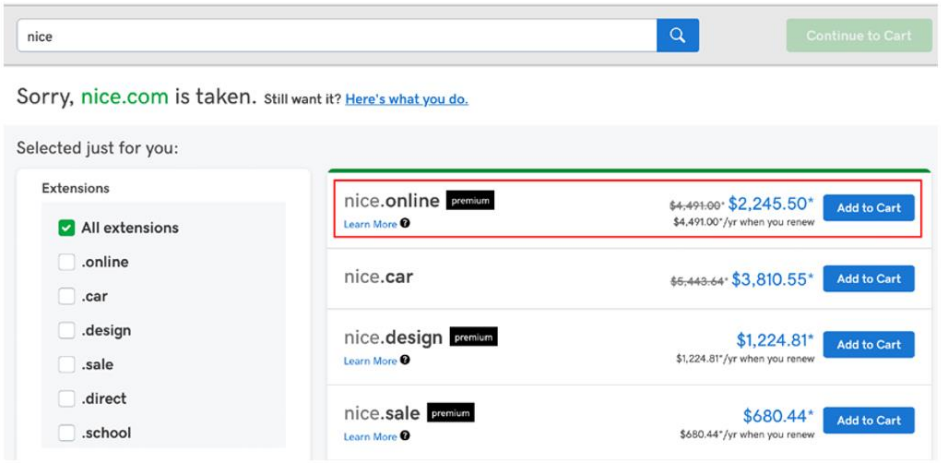
Anda memiliki Digital Ocean, Microsoft Azure, layanan Google Cloud, solusi cloud IBM, dll. Hanya perusahaan malas yang tidak menawarkan solusi cloud dan virtual saat ini. Setelah Anda memutuskan tempat untuk meng-host layanan Anda, Anda dapat memutuskan bagaimana menamainya. Nama tingkat atas yang akan digunakan di internet harus dibayar dan setelah Anda membelinya, Anda dapat dengan mudah menghubungkannya ke alamat IP tempat aplikasi Anda berjalan. Layanan yang menjual nama untuk layanan web agar dapat ditemukan di jaringan global disebut penyedia DNS.

DNS berarti sistem Nama Domain, dan pada dasarnya adalah sistem yang memetakan nama yang bermakna ke alamat ip, membuatnya dapat ditemukan dan diakses di jaringan global.

Penyedia DNS memungkinkan orang untuk membeli nama yang mereka inginkan dan memetakannya ke layanan yang mereka butuhkan untuk tujuan bisnis maupun pribadi. Biaya bervariasi sesuai dengan kesederhanaan nama, kemudahan menghafal, dan jenis domain tingkat atas.

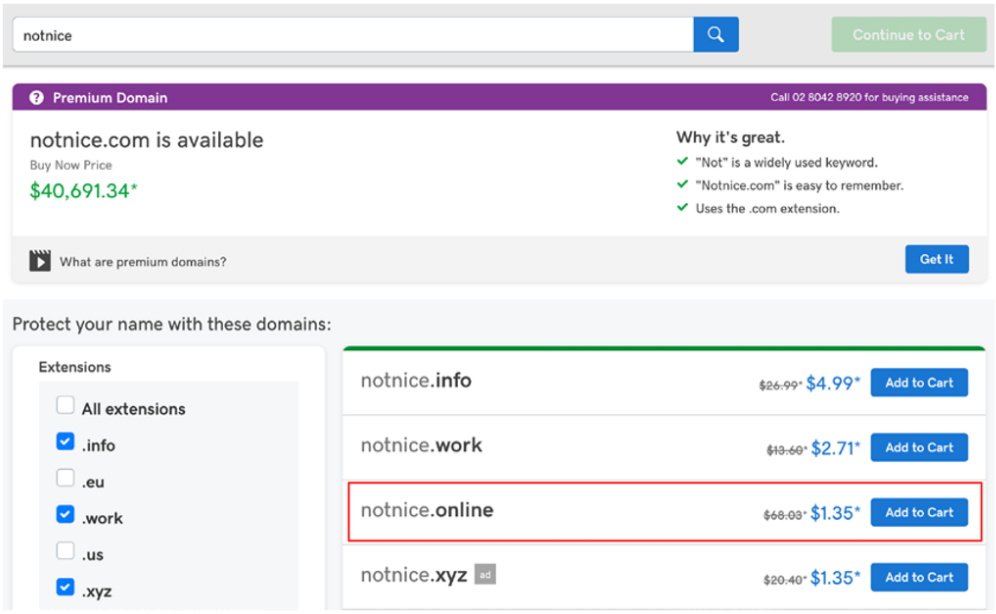
Domain tingkat atas adalah segmen terakhir dari nama domain (sesuatu yang muncul setelah nama: .com, .edu, .org, dll.)

Ada penyedia DNS yang berbeda di luar sana. Salah satu yang kami gunakan disebut GoDaddy (https://www.godaddy.com). Ada beberapa opsi lain: cloudflare (https://www.cloudflare.com/dns), ClouDNS (https://www.cloudns.net/), freeDNS (https://freedns.afraid.org/), namecheap (https://www.namecheap.com/domains/freedns.aspx), penyedia DNS gratis bernama … 1984 (https://www.1984hosting.com/product/freedns/), dll. Nama terakhir adalah agak menakutkan harus kita akui, tetapi di zaman kita hidup dengan akses data yang mudah—Facebook, Google, dan lainnya—setidaknya nama ini tampak akurat. Seperti yang sudah kami tunjukkan, harga nama domain dapat bervariasi. Mari kita telusuri, misalnya, untuk nama domain "bagus" di GoDaddy (Gambar 8-1).



Gambar 8-1. Harga untuk domain nice.online

Seperti yang Anda lihat, itu tidak terlalu murah. Mari kita lihat apa yang terjadi jika kita mencari domain "notnice" (Gambar 8-2).



Gambar 8-2. Harga untuk domain notnice.online

Wow, sedikit lebih murah, bukan? Jika Anda memutuskan untuk membeli domain untuk layanan web Anda, sebelum membahas nama apa yang terdengar bagus, periksa ketersediaan dan harga di situs web penyedia DNS. Setelah Anda membeli nama domain Anda, Anda harus menghubungkannya ke alamat IP dari host tempat Anda menjalankan layanan Anda. Baik penyedia host dan domain memberikan instruksi yang jelas tentang cara melakukannya. Kemudian ketikkan saja namanya di browser dan akses layanan Anda! Jika semua ini terdengar agak rumit bagi Anda, jangan khawatir, ada banyak layanan yang dapat membuat hidup Anda lebih mudah. Misalnya, wix (https://www.wix.com/) menawarkan perjalanan dari pembuatan situs web drag-and-drop sederhana ke pendaftaran domain. Anda membangun situs web Anda dengan cepat, Anda mendaftarkannya, dan segera tersedia di jaringan.

Layanan serupa lainnya adalah squarespace (https://www.squarespace.com/). Ini juga memungkinkan pembuatan situs web tempat Anda dapat memilih dari ratusan templat yang menakjubkan dan segera mendaftarkan domain Anda. Jika Anda masih ingin membangun layanan Anda sendiri tetapi ingin mengotomatiskan bagian hosting, sehingga Anda tidak perlu menyalin dan menempelkan file Anda secara manual ke server setiap kali Anda mengubahnya, Anda dapat menggunakan salah satu solusi otomatis seperti AWS (https://aws.amazon.com/), heroku (https://www.heroku.com/), Google cloud (https://cloud.google.com/), Google firebase (https://firebase.google.com/), dll. Layanan ini memberi Anda antarmuka baris perintah sederhana yang memungkinkan cara mudah untuk menerapkan layanan Anda hanya dengan beberapa perintah.

8.2 Kapan Kita Harus Mulai Memikirkan Penerapan?

Jika Anda cukup memperhatikan bab-bab sebelumnya dan jika Anda pandai menggambar paralel, Anda mungkin sudah tahu jawaban untuk pertanyaan ini. Tentu saja, kita harus mulai memikirkan penerapan di awal implementasi. Anda perlu tahu bagaimana layanan Anda akan terintegrasi dengan alat penerapan apa pun yang akan Anda gunakan. Artinya, Anda harus mempersiapkan infrastruktur yang dibutuhkan terlebih dahulu, sehingga kapan pun kode Anda siap dikirim, Anda dapat mengirimkannya dengan mengklik tombol. Tentu saja, saat Anda mulai menulis kode, tidak ada yang perlu diterapkan. Versi deployable pertama mungkin akan membutuhkan waktu untuk siap.

Jadi bagaimana Anda bisa mempersiapkan semuanya terlebih dahulu? Simulasikan lingkungan; berpura-pura semuanya sudah siap. Anda memiliki gagasan tentang arsitektur Anda dan bagaimana berbagai hal akan berinteraksi satu sama lain (mis., basis data, backend, dan frontend yang mendasarinya). Buat tiruan dari hal-hal ini dan letakkan secara online. Pastikan itu berhasil. Tingkatkan proses penerapan hingga 100% atau mendekati 100% otomatis. Ada banyak alat yang dapat membantu Anda dalam hal ini. Setelah Anda memiliki proses penerapan otomatis, Anda adalah raja kode Anda. Anda tahu pasti bahwa setelah Anda mengubah kode Anda, Anda dapat dengan mudah membuatnya hidup. Namun, Anda harus yakin bahwa semua tes lulus sebelum menerapkannya. Anda juga harus yakin bahwa

jika Anda bukan satu-satunya yang mengerjakan kode, kode dari beberapa pengembang mudah diintegrasikan.

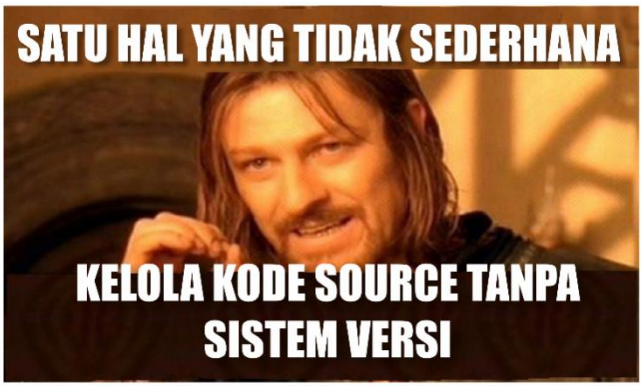
Seluruh proses ini juga harus otomatis. Proses otomatis untuk mengintegrasikan beberapa kode pengguna dalam basis kode yang sama dan menjalankan pengujian serta pemeriksaan kode lainnya sebelum penerapan disebut Integrasi Berkelanjutan. Selangkah lebih maju: selain integrasi dan pengujian otomatis, siapkan kode Anda untuk dikirim ke produksi, dan Anda akan memiliki Pengiriman Berkelanjutan. Deploy hasil Continuous Delivery Anda secara otomatis ke platform hosting Anda— voila, Anda juga memiliki Continuous Deployment!

Jika Anda ingin menjadi mewah, Anda dapat menyingkat semua integrasi berkelanjutan dan proses penyebaran berkelanjutan sebagai CI/CD (jika saya memiliki band rock, saya mungkin akan menyebutnya demikian). Ada banyak alat yang siap membantu Anda untuk mengatur proses sulap berkelanjutan Anda, tetapi sebelum membahasnya, pertama-tama mari kita pikirkan tentang kode itu sendiri. Di mana kita harus meletakkannya, sehingga dapat dengan mudah diperiksa, diuji, disebarkan, dan dikirim? Cukup jelas bahwa folder di komputer kita saja tidak cukup untuk pekerjaan itu.

8.3 Di Mana Saya Menempatkan Kode Saya?

Di bagian sebelumnya, kami menyimpulkan bahwa agar dapat terus menjalankan proses otomatis integrasi, pembuatan, dan penerapan perangkat lunak, kode kami tidak cukup untuk hidup di mesin kami. Harus ada tempat terpusat di mana setiap orang yang terlibat dalam pengembangan bisa mendapatkan kode kapan saja dan di mana kode dapat diperbarui setelah diubah. Mari kita berpikir. Bagaimana seharusnya tempat ini terlihat? Mungkin Dropbox atau Google Drive? Atau beberapa layanan cloud lainnya, seperti iCloud? Bisa jadi. Itu terpusat, Anda dapat berbagi akses dengan orang yang relevan, dan Anda dapat mengakses file, mengubahnya, menghapusnya, dan menambahkan yang baru. Itu benar.

Coba bayangkan skenario berikut ini: Anda dan teman Anda mulai mengerjakan beberapa file secara bersamaan. Anda berdua mengakses cloud Anda, mengunduh file, dan mulai bekerja. Anda selesai dengan perubahan Anda terlebih dahulu, jadi Anda mengunggah file. Berkasnya diganti. Setelah beberapa saat sobat juga selesai dan juga mengupload file tersebut, dan file tersebut diganti lagi. Versi baru file ini tidak akan berisi perubahan Anda. Ups! Bayangkan skenario lain—Anda mengerjakan file Anda sendiri. Anda memperkenalkan beberapa perubahan, mengganti file, memperkenalkan lebih banyak perubahan, mengganti file lagi, memperkenalkan lebih banyak perubahan, dan mengganti file lagi. Tiba-tiba Anda menyadari bahwa sistemnya rusak. Apa sebenarnya yang merusaknya? Bagaimana Anda mengembalikan file? Bagaimana Anda memeriksa mana dari tiga versi terbaru yang berfungsi? Boromir memiliki jawaban untuk pertanyaan ini (Gambar 8-3).



Gambar 8-3. Seseorang tidak hanya mengelola kode sumber tanpa sistem versi

Boromir benar, seperti biasa. Untuk dapat mengontrol versi kode, kita membutuhkan sesuatu yang lebih dari sekedar sistem cloud terpusat. Kami membutuhkan sistem kontrol versi. Dan mereka ada! Mereka disebut SCM (Alat Manajemen Kode Sumber) atau VCS (Sistem Kontrol Versi).

Ada banyak dari mereka: SVN (Apache Subversion; https://subversion.Apache.org/), GIT (https://git-scm.com/), lincah (https://www.mercurial-scm.org/), dll. Di antara pengembang, GIT adalah yang paling populer. Ini memungkinkan pengelolaan kode sumber menggunakan serangkaian perintah sederhana namun kuat. Misalnya, untuk membuat repositori git pada folder tertentu, Anda akan menjalankan "git init". Untuk menarik perubahan terbaru, Anda menjalankan perintah "git pull". Untuk menambahkan file baru, Anda akan menjalankan "git add". Setelah Anda mengubah beberapa file dan Anda puas dengan perubahan itu, Anda harus mengkomitnya; kemudian Anda menjalankan "git commit", memberikan pesan komit yang bermakna yang menyatakan apa yang telah Anda ubah. Setelah Anda siap untuk membagikan perubahan yang Anda lakukan dengan anggota tim lainnya, Anda mendorong mereka dengan menjalankan perintah "git push".

Setiap komit memiliki hash terkait, dan Anda dapat mengembalikan kode Anda ke komit apa pun kapan saja. Jika kDania Anda mengerjakan file yang sama, git akan memastikan bahwa semua perubahan berakhir di repositori. Jika perubahannya bertentangan—bayangkan bahwa Anda telah mengubah baris kode yang sama dengan orang lain—git akan memperingatkan Anda dan meminta Anda untuk menyelesaikan konflik tersebut sebelum repositori dapat berada dalam keadaan yang konsisten lagi. Anda dapat menggunakan alat visual ini untuk melatih praktik git Anda: http://git-school.github.io/visualizing-git/.

Tips: setiap kali Anda mengerjakan sepotong kode, pastikan Anda selalu mengerjakan versi terbaru dan Anda melakukan dan mendorong perubahan Anda cukup sering. Ini akan memungkinkan Anda untuk menghindari konflik.

Oke, menjalankan perintah git terdengar keren, tetapi di mana tepatnya kode itu berada? Masih harus ada beberapa layanan cloud tempat kode berada sehingga semua orang dapat mengaksesnya. Itu benar! Ada banyak solusi untuk memelihara repositori git Anda. Bitbucket (https://bitbucket.org/), gitlab (https://about.gitlab.com/), dan github (https://github.com/) adalah yang paling populer. Kami menggunakan github karena

menyediakan banyak alat dan integrasi yang berguna untuk CI/CD. GitLab sekarang mendapatkan popularitas juga, karena memungkinkan serangkaian fitur dengan pola pikir di sisi DevOps. Semua pipeline CI/CD sudah terintegrasi dalam alat ini, jadi Anda tidak perlu mengintegrasikannya dengan hal lain, dan frontend-nya ditulis dalam Vue.js, yang merupakan penggemar berat Solikhan!

8.4 Integrasi Berkelanjutan dan Pengujian Otomatis

Kami telah membahas cara terus mengintegrasikan kode kami dengan Git sehingga Anda dapat yakin bahwa semua perubahan Anda disimpan dan diversi dengan hati-hati! Sekarang mari kita bicara tentang memeriksa kode Anda setiap kali Anda mendorongnya. Di bab sebelumnya, Anda telah mempelajari cara menulis berbagai jenis tes. Mari kita lihat sekarang bagaimana kita menjalankan tes tersebut untuk berjalan secara otomatis ketika kita mendorong kode kita ke VSC kita.

Mari kita ingat satu hal: tes biasanya berjalan karena kita menjalankan beberapa perintah, bukan? Misalnya, di bab sebelumnya kita menulis beberapa unit test untuk backend, dan untuk menjalankannya kita akan menjalankan perintah mvn clean test -Ptest. Di aplikasi frontend, kami memiliki pengujian unit yang akan dijalankan dengan perintah npm test dan pengujian ujung ke ujung yang akan dijalankan dengan perintah npm run E2E. Anda dapat menyertakan pemeriksaan lain—misalnya, pemeriksaan kualitas kode, cakupan pengujian, pedoman kode spesifik seperti jumlah baris kode maksimum per file, dll. Anda dapat mengonfigurasi ambang batas untuk pemeriksaan tersebut dan menentukan perintah untuk memverifikasi semua itu (Gambar 8- 4).

Gambar 8-4. Ada ratusan metrik kualitas kode yang dapat diperiksa!

Periksa semua hal, tetapi pertama-tama kenali hal-hal yang penting untuk jenis bisnis Anda! "Ok, jadi saya mendefinisikan beberapa perintah untuk memeriksa hal-hal penting saya, sekarang apa?" Anda mungkin bertanya. Sekarang entah bagaimana Anda harus memberi tahu sistem Anda untuk menjalankan perintah ini. Kapan Anda menjalankannya? Terserah Anda untuk memutuskan. Anda dapat, misalnya, membuat kait pra-komit yang akan menjalankan semua pemeriksaan sebelum Anda melakukan perubahan. Atau, itu bisa menjadi kait pra-dorong. Namun, hook pre-commit dan pre-push akan berjalan di mesin lokal Anda, yang akan memperlambat proses pengembangan Anda jika ada banyak pemeriksaan.

Biasanya, Anda menjalankan pemeriksaan paling kritis secara lokal—misalnya, pengujian unit—dan membiarkan pemeriksaan kualitas lainnya untuk alat integrasi berkelanjutan Anda. Ada banyak alat di luar sana. Mereka dengan mudah berintegrasi dengan GitHub atau platform hosting Git lainnya, mendeteksi komit baru, menjalankan pemeriksaan pada mereka dan mendapatkan pemberitahuan jika terjadi kegagalan. Salah satu alatnya adalah Travis CI (https://travis-ci.org/). Ini adalah yang kami gunakan untuk platform kursus kami.

Pada dasarnya, Anda terhubung ke platform TravisCI, menggunakan antarmukanya untuk menghubungkannya ke repositori GitHub Anda, dan membuat file konfigurasi untuk travis untuk mengetahui perintah mana yang harus dijalankan untuk memeriksa kewarasan kode Anda. Alat bagus lainnya yang mudah diintegrasikan dengan GitHub adalah CircleCi (https://circleci.com/). Ini sangat mirip dalam pengaturan dan konfigurasinya dengan TravisCI. Platform lama yang sangat bagus yang telah digunakan selama berabad-abad oleh generasi pengembang adalah Jenkins (https://jenkins.io/). Ini agak jelek dan rumit tetapi sangat kuat, dan tidak ada yang tidak bisa Anda lakukan dengannya. Jika Anda ingin merasa seperti seorang geek, Anda dapat menggunakan ConcourseCI (https://concourse-ci.org/). Alat ini juga dikonfigurasi menggunakan file konfigurasi yaml dan memberi Anda visualisasi yang bagus dari pipeline build. Terserah Anda mau menggunakan apa. Kami lebih memilih TravisCI atau CircleCI karena mereka mudah diintegrasikan dengan GitHub dan memiliki antarmuka yang ramah pengguna. Kurva pembelajaran untuk menggunakan alat-alat ini minimal yang memungkinkan Anda memiliki pengaturan dasar untuk integrasi berkelanjutan Anda dalam hitungan menit.

8.5 Pengiriman dan Penerapan Berkelanjutan

Kami telah membahas bahwa semua pemeriksaan yang Anda jalankan pada setiap push bersama dengan build dan pengemasan untuk penerapan menentukan pengiriman berkelanjutan Anda. Di bagian sebelumnya, kami membahas cara menyiapkan alat CI Anda untuk menjalankan semua pemeriksaan; satu-satunya hal yang hilang untuk memiliki pengiriman berkelanjutan Anda adalah memberi tahu alat CI untuk menyiapkan kode yang akan dikirim ke produksi. Mempersiapkan kode yang akan dikirim berarti hal yang berbeda, tergantung pada kasusnya. Misalnya, jika hanya sekumpulan file HTML, JavaScript, dan CSS yang harus disalin ke beberapa server web, maka file tersebut mungkin sudah siap produksi.

Namun, jika Anda menggunakan beberapa kerangka kerja atau praprosesor, Anda mungkin ingin menjalankan beberapa perintah untuk mengubah kode Anda menjadi HTML+CSS+JS biasa. Misalnya, dalam kasus kami, kami menggunakan Vue.js. Agar siap untuk produksi, kita harus mengubah semua file Vue menjadi file JavaScript biasa. Karena aplikasi kita dibuat menggunakan boilerplate webpack Vue, yang dilengkapi dengan serangkaian perintah yang telah ditentukan, kita dapat menggunakan perintah “npm run dist” untuk menyiapkan kode untuk distribusi. Oleh karena itu, kami harus memberi tahu alat CI/CD kami untuk menjalankan perintah ini.

Dalam kasus proyek Java, kita tahu bahwa kita harus mengkompilasi server kita menggunakan perintah mvn clean install. Jadi, untuk menutupi pengiriman berkelanjutan,

kita perlu menentukan dalam file konfigurasi kita semua perintah yang diperlukan yang akan mengubah kode kita menjadi artefak yang dapat dijalankan. Satu-satunya hal yang hilang sekarang adalah penyebaran. Setelah Anda membangun semua file aplikasi, Anda dapat menerapkannya. "Untuk menyebarkan" juga dapat berarti hal yang berbeda tergantung pada bentuk hosting yang digunakan aplikasi Anda. Jika ini hanya tentang menyalin file Anda ke beberapa server jarak jauh, maka Anda dapat memberi tahu alat CI/CD Anda untuk menyalin melalui SCP (protokol penyalinan aman) file yang dihasilkan oleh build.

Dalam kasus kami, karena kami menggunakan Heroku, kami tidak perlu menyalin apa pun, karena Heroku juga menyediakan fungsionalitas bangunan. Jadi, dalam kasus kami, setelah Travis berhasil memverifikasi pengujian sehingga kami dapat memberi tahu Heroku bahwa kode tersebut dapat dibuat dan digunakan. Ada banyak kemungkinan skenario. Anda dapat mendelegasikan proses pembangunan ke mesin atau layanan hosting menggunakan alat CI/CD hanya untuk memicu proses. Misalnya, adalah umum untuk menggunakan Git untuk tujuan ini—mesin hosting memiliki akses ke repositori Git dan berisi semua skrip yang diperlukan untuk membuat dan menyalin file ke tempat seharusnya agar dapat dijalankan. Setelah proses dipicu, mesin hosting akan menarik perubahan terbaru, membuat file dan menyalinnya ke folder server. Ini memerlukan beberapa pekerjaan manual dari Anda untuk mengatur semuanya dengan benar, tetapi itu berfungsi jika Anda tidak ingin mempercayai layanan pihak ketiga untuk melakukan pekerjaan ini untuk Anda. Ada banyak perselisihan tentang apakah akan menyebarkan secara otomatis atau tidak setelah semua pemeriksaan lulus. Beberapa orang lebih suka memiliki tombol untuk menyetujui dorongan ke produksi. Kami cenderung setuju dengan pendekatan ini. Terkadang terlalu berisiko untuk menerapkan penerapan otomatis pada setiap push, terutama bila Anda memiliki banyak pengguna yang mengandalkan layanan Anda. Bahkan dengan segala sesuatu yang otomatis dan diperiksa, kita masih manusia dan kita membuat kesalahan! Kami tidak ingin secara tidak sengaja menggulung sesuatu ke produksi seperti stiker telegram kumbang kotoran yang bagus ini (Gambar 8-5).

Gambar 8-5. Stiker Telegram yang bertuliskan "Diluncurkan ke produksi"

Apa yang biasanya dilakukan adalah Anda memiliki lingkungan pementasan yang sangat mirip dengan lingkungan produksi. Kemudian Anda memberi tahu alat CI/CD Anda

untuk terus menyebarkan ke lingkungan ini, dan setelah Anda memeriksa secara manual bahwa semuanya bagus dan bagus, Anda menekan tombol yang akan digunakan untuk produksi. Semua alat CI/CD memungkinkan pendekatan ini. Ini disebut "mempromosikan ke produksi." Dengan cara ini tidak sepenuhnya otomatis, tetapi upaya mendorong ke produksi sama dengan mengklik tombol, yang sangat dapat diterima.

8.6 Siapa Melakukan Apa dan Bagaimana?

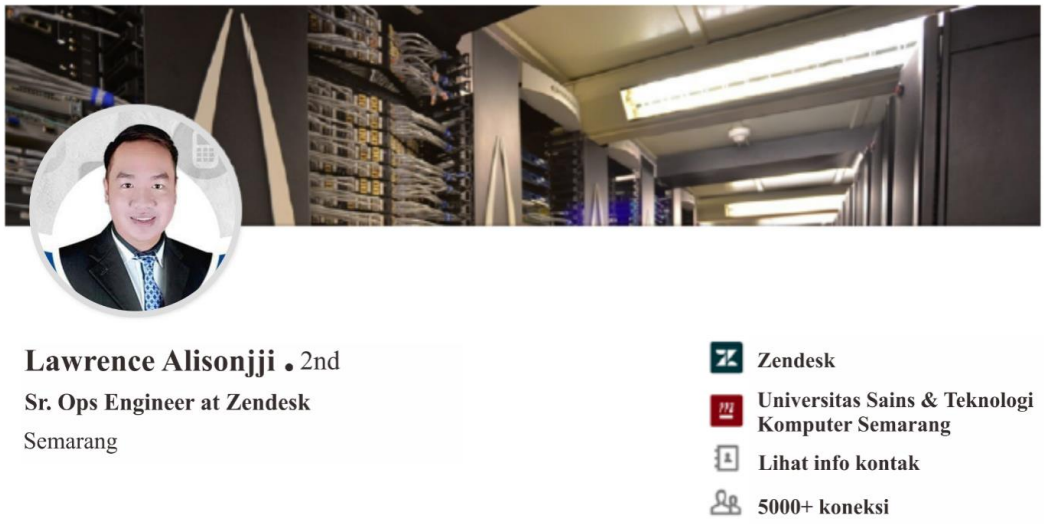
Kami membahas banyak hal berbeda di bagian sebelumnya—mengatur proses CI/CD, menyiapkan infrastruktur untuk memiliki banyak lingkungan, menulis skrip untuk menjalankan semua pemeriksaan, menekan tombol… Siapa yang bertanggung jawab untuk apa? Siapa yang memutuskan kapan harus menekan tombol mana? Siapa yang bertanggung jawab untuk mendefinisikan suatu proses? Sebenarnya, semua orang bertanggung jawab. Dalam tim yang baik, semua orang merasa bertanggung jawab dan merasa memiliki produk dan proses yang membantu memastikan kualitas produk ini.

Saat Anda memulai, kemungkinan besar Anda tidak mampu memiliki orang khusus untuk menangani infrastruktur dan proses CI/CD, sehingga pengembang harus menerapkan langkah-langkah yang diperlukan untuk menjamin otomatisasi langkah pengiriman. Setelah produk Anda solid dan ukuran tim pengembangan cukup besar, Anda dapat memikirkan untuk memiliki orang-orang yang berdedikasi untuk menangani semua itu. Orang-orang ini dapat memiliki nama yang berbeda—DevOps (insinyur pengembangan dan operasi), tim infrastruktur, insinyur keamanan, SRE (insinyur keandalan situs) —semua orang ini mengkhususkan diri dalam menghubungkan pengembangan dengan operasi yang diperlukan untuk menjaga produk Anda dalam keadaan sehat dan mengotomatisasi proses ini semaksimal mungkin.

Kami melihat banyak skenario berbeda. Misalnya, ketika kami mulai bekerja di Feedzai, kami adalah tim yang cukup kecil. Kami tidak memiliki CI/CD dan produk kami adalah paket mandiri yang akan kami kirimkan ke klien kami di bawah lisensi berbayar. Oleh karena itu, kami memiliki siklus rilis 2 bulan, dan di akhir setiap rilis kami akan mengirimkan executable untuk platform yang berbeda bersama dengan catatan rilis. Di Gymondo, tempat Lawrence bekerja, kami memiliki CI dan CD, dengan penerapan otomatis untuk lingkungan pengujian dan panggung dan siap dirilis ke produksi dengan serangkaian langkah manual sederhana untuk mencapai waktu henti nol. Di mana Solikhan bekerja (OptioPay), kami cukup sering mengirim; setelah kode fitur ditinjau, disetujui, dan pengujian dijalankan, kami mengklik tombol untuk mempromosikannya ke produksi. Oleh karena itu, kami tidak memiliki tanggal rilis khusus, yang terkadang kami sesali karena kami pikir merayakan rilis itu sehat untuk semangat tim. Kami berpikir bahwa skenario yang sempurna adalah mengirimkan dengan cepat dan sering tetapi memiliki beberapa tanggal "rilis" khusus ketika Anda dapat mengatakan, "Hei, kami telah merilis ini, ini, dan itu. Mari kita rayakan!” Dan saat Anda menerapkan fitur baru ke produksi, jangan lupa untuk memutar lagu ini dengan keras: https://soundcloud.com/the-avid-capitalz/the-avid-capitals-we-are. Ini akan memberi Anda suasana hati bahagia yang gila.

8.7 Wawancara Dengan DevOps

Tentu saja, berbicara tentang orang-orang DevOps tanpa berbicara dengan salah satu dari mereka tidak masuk akal. Itu sebabnya kami memutuskan untuk mengobrol dengan teman kami yang bernama Lawrence (Gambar 8-6). Dia bekerja untuk Zendesk, dan dia adalah pesulap yang cukup bagus di bidangnya. Kami melakukan percakapan Skype dengannya dan menyalinnya untuk Anda. Menikmati!



Gambar 8-6. Profil Linkedin Lawrence (jadi Anda tahu dia adalah orang yang nyata!)

Solikhan: Hei Lawrence , jadi saya diberitahu bahwa Anda adalah seorang profesional DevOps yang hebat. Bisakah Anda memberi tahu kami secara sederhana apa yang Anda lakukan dan apa artinya menjadi seorang DevOps?

Lawrence : Ada banyak perjuangan di masa lalu antara pengembang dan orang-orang operasi. Mereka tidak dapat berbicara satu sama lain dengan jelas dan pada dasarnya untuk menemukan titik temu. Ada banyak budaya tentang bagaimana kami dapat mengembangkan perangkat lunak secara efektif dan menjalankan perangkat lunak ini dalam produksi dengan efisiensi tinggi, dan itu semua tentang DevOps— menyatukan pengembang dan operasi dan memastikan kami dapat membuat saluran yang melakukan pengembangan sampai kami mendapatkan perangkat lunak dalam produksi.

Solikhan: Oke, jadi ini pada dasarnya untuk membuat jalur dari pengembangan ke produksi lebih mudah. Ini berarti bahwa itu memiliki dampak nyata pada keseluruhan bisnis, jika terjadi kesalahan, seluruh bisnis mungkin menderita...

Lawrence : Ya, Anda benar. Jika ada yang tidak beres di antaranya... Bayangkan sebuah pabrik dan Anda sedang membangun mobil dan Anda memiliki semua tahapan ini untuk membuat mobil. Anda mendapatkan bahannya, Anda mulai membuat mobil, dan pada akhirnya, Anda memiliki mobil.

Jadi, dengan perangkat lunak saya pikir itu sama, tetapi itu adalah teks yang Anda tulis, jadi pengembang menulis sepotong teks yang harus dijalankan di komputer atau sekumpulan komputer. Jadi, saat Anda mengembangkan perangkat lunak itu, ada banyak tahapan yang perlu dibuat dan memiliki gerbang kualitas untuk memastikan bahwa perangkat lunak yang masuk ke produksi memenuhi persyaratan kualitas. Kami tahu

perangkat lunak yang sempurna tidak ada di dunia kami, sebagian besar perangkat lunak memiliki ribuan bug, tetapi setidaknya Anda tahu bahwa sebagian besar bug yang Anda hadapi tidak akan berdampak pada pengguna kami. Jadi, budaya DevOps adalah memikirkan bagaimana kami dapat mengalihkan loop umpan balik ke pengembang secepat mungkin, karena jika ada sesuatu yang salah dan Anda dapat langsung memperbaikinya, itu adalah cara termudah dan dengan biaya terendah juga. Menemukan bug dalam produksi menimbulkan biaya bagi organisasi mana pun karena akan berdampak pada pengguna kami. ***Jadi, kita berbicara tentang pengiriman perangkat lunak, memberikan perangkat lunak lebih cepat, dengan umpan balik yang lebih cepat untuk pengembang, penguji, atau operasi dan dengan demikian mengurangi jumlah bug dalam produksi.***

Solikhan: Kedengarannya cukup penting bagi perusahaan untuk memiliki orang-orang profesional yang melakukan ini. Bisakah Anda menjelaskan kepada kami bagaimana Anda menjadi seorang DevOps? Saya tidak yakin apakah ada beberapa kursus yang dirancang khusus untuk menjadi insinyur DevOps, jadi entah bagaimana orang melakukan beberapa jalur karier sampai mereka mencapai peran DevOps... Bagaimana dengan Anda?

Lawrence : Saat ini ada beberapa sertifikasi untuk DevOps—misalnya, Amazon AWS memiliki sertifikasi DevOps sehingga Anda bahkan dapat mempelajari cara menjadi insinyur DevOps. Menjawab pertanyaan Anda, Anda mencampur sedikit pengembangan dan operasi, jadi saya lulus dalam ilmu komputer, jadi saya memiliki semua latar belakang rekayasa perangkat lunak tetapi sebagian besar pengalaman saya didasarkan pada administrasi sistem, tetapi saya juga punya teman yang datang langsung dari rekayasa perangkat lunak.

Jadi, Anda adalah seorang pengembang perangkat lunak dan Anda mulai menjalankan perangkat lunak di komputer Anda, tetapi Anda menyadari bahwa komputer Anda tidak serumit lingkungan produksi Anda, sehingga orang-orang tertarik pada bagaimana mereka dapat menerapkan perangkat lunak tersebut dalam produksi. Karir saya dimulai sebagai administrator sistem sejak lama. Sama seperti orang lain, saya adalah administrator Linux, saya tahu segalanya tentang sistem operasi, jaringan komputer, mengelola semua server, firewall, proxy, server web, server aplikasi seperti Apache nginx... Jadi, banyak teknologi yang terlibat. Setelah beberapa saat saya mulai menyadari bahwa operator seperti saya tidak menggunakan kode untuk mengelola infrastruktur, jadi biasanya semuanya manual atau melalui banyak skrip ad-hoc, jadi jika Anda perlu menyiapkan server web, Anda perlu SSH ke mesin dan ketik perintah apa pun yang diperlukan dan pastikan itu berfungsi.

Masalahnya adalah bahwa itu tidak berskala, jadi saya mulai belajar lebih banyak tentang rekayasa perangkat lunak, sehingga saya dapat membuat alat saya sendiri untuk mendukung penerapan dan strategi untuk organisasi. Jadi, ini tentang mengotomatisasi seluruh proses penyediaan infrastruktur dan penerapan aplikasi yang akan berjalan di infrastruktur itu. Jadi itulah jalur karir saya, tapi sejujurnya Anda bisa menjadi insinyur DevOps jika Anda seorang insinyur perangkat lunak atau jika Anda seorang administrator sistem. Tentu saja, Anda mungkin memiliki lebih banyak keterampilan di satu sisi tertentu dari dev atau ops sehingga Anda secara alami menjadi devop dengan mempelajari dan

melengkapi sisi yang kurang Anda ketahui, jadi sangat penting untuk memiliki kedua sisi di atas meja.

Solikhan: Bayangkan beberapa pembaca atau siswa kami memutuskan untuk menjadi insinyur DevOps tetapi tidak memiliki latar belakang ilmu komputer atau tidak pernah melakukan apa pun yang berhubungan dengan administrasi sistem. Apakah ada yang bisa mereka lakukan untuk menjadi insinyur DevOps, seperti kursus dan sebagainya, atau menurut Anda mereka harus pergi dan belajar ilmu komputer hardcore?

Lawrence : Anda bisa menjadi insinyur DevOps tanpa gelar ilmu komputer— misalnya, jika Anda ingin mengikuti jalur sertifikasi, yang disarankan Amazon adalah Anda memiliki tiga sertifikasi dasar yang mencakup pengembangan, operasi, dan arsitektur. Jadi, setelah Anda memilikinya, Anda bisa mendapatkan sertifikasi untuk menjadi profesional DevOps. Meskipun sertifikasi tersebut didasarkan pada layanan AWS mereka sendiri, jika Anda mempelajari prinsip-prinsip dari segala hal yang terkait dengan DevOps, saya yakin Anda dapat mencapainya. AWS bukan satu-satunya pilihan di pasar, tetapi ini jelas merupakan awal yang baik.

Solikhan: Dari pengalaman pribadi Anda, dapatkah Anda memberi tahu kami beberapa masalah besar yang harus Anda tangani, sesuatu yang berdampak besar pada produk atau perusahaan dan bagaimana Anda berhasil menyelesaikannya?

Lawrence : Ini sebenarnya yang sedang ditangani DevOps... Beberapa tahun yang lalu saya mengubah mesin dalam produksi, mesin tersebut bertanggung jawab atas seluruh sistem otentikasi universitas, jadi setiap guru atau profesor mana pun yang harus mengakses sumber daya apa pun yang mereka miliki untuk mengotentikasi, dan server otentikasi, dikelola oleh saya.

Jadi, saya dengan senang hati meningkatkan sistem otentikasi ke mesin baru; semuanya berjalan baik-baik saja, tetapi tanpa otomatisasi sama sekali, mengubah semua skrip dan file konfigurasi secara manual di mesin menggunakan terminal, dan tiba-tiba saya menyadari bahwa saya ada rapat dengan bos saya dan saya sudah terlambat untuk rapat, jadi saya bergegas untuk me-restart server, dan mematikan server lama, tetapi saya mematikan yang baru. Jadi, sistemnya mati, dan dengan itu seluruh universitas selama sekitar satu jam. Semua orang menelepon menanyakan apa yang sedang terjadi, mengapa sistem mati, dan seterusnya, dan saya sudah berada di rapat ketika profesor lain masuk dan bertanya apa yang terjadi; apakah kami melakukan beberapa perubahan, mungkin dalam sistem otentikasi? Saya seperti, "Ups..." Jadi, pesan moral dari cerita ini adalah jika Anda tidak memiliki otomatisasi, kemungkinan besar Anda akan membuat kesalahan.

Saya saat itu mungkin memiliki empat terminal terbuka dengan konfigurasi, dan meskipun saya tahu apa yang saya lakukan, jika Anda mengetikkan perintah di jendela yang salah atau sesuatu, Anda dapat mengalami masalah. Jadi, dengan memiliki mekanisme otomatisasi yang baik, Anda tidak menyentuh server mana pun, Anda memiliki saluran pipa, proses otomatis yang akan melalui semua langkah ini dan akan menjalankan semua tugas yang perlu dijalankan dan akan memastikan bahwa pada akhirnya Anda akan mendapatkan hasil yang diharapkan. Semua proses ini mengikuti alur DevOps, prinsip yang sama seperti mekanisme pengembangan perangkat lunak lainnya, Anda harus memeriksa kode Anda

terhadap sistem pengujian yang serupa dengan produksi, Anda memiliki tinjauan kode dengan kDania Anda, Anda harus mendorong kode ini ke mesin pengujian terlebih dahulu , dan hanya dengan begitu Anda dapat mendorong kode Anda ke produksi.

Saya percaya perubahan terbesar dari sebelumnya datang ketika mereka mulai mengembangkan konsep yang disebut "Infrastruktur yang Ditentukan Perangkat Lunak", jadi pada dasarnya bagaimana kita sebagai orang-orang infrastruktur mendefinisikan infrastruktur kita tidak hanya melakukan perintah manual di dalam router atau mesin Linux tetapi menggunakan kode perangkat lunak. Anda menulis perangkat lunak untuk menggambarkan seluruh keadaan infrastruktur Anda dan keadaan itu akan didorong ke infrastruktur yang Anda coba konfigurasikan. Dan untuk itu ada banyak alat di pasaran.

Solikhan: Oke, jadi sekarang kita berbicara tentang beberapa peristiwa frustasi yang terjadi pada Anda, dapatkah Anda menggambarkan kesuksesan besar yang terjadi pada Anda saat berada di peran DevOps, sesuatu yang sangat Anda banggakan misalnya?

Lawrence : Jadi, saya bertanggung jawab untuk menyediakan seluruh infrastruktur cloud untuk pemerintah Brasil beberapa tahun yang lalu. Kami dulu memiliki server bare-metal yang benar-benar tua dengan cara penyediaan yang lama, jadi kami menyiapkan seluruh infrastruktur cloud baru dan tidak hanya itu—kami sebenarnya dapat menyediakan layanan infrastruktur untuk seluruh pemerintah Brasil.

Jadi, saya bukan orang teknis pada waktu itu, tetapi saya mengelola proyek itu dan saya sangat bangga akan hal itu. Saat ini perusahaan masih menjual solusi itu ke agensi di Brasil, dan departemen administrasi, jadi mereka semua menggunakan solusi itu dan itu sangat keren.

Solikhan: Itu cukup mengesankan...

Lawrence : Ya, dan itu mengurangi waktu penyediaan. Biasanya di organisasi kami, kami membutuhkan waktu sekitar 3 bulan untuk menyediakan infrastruktur yang kompleks, dan dengan lingkungan cloud itu kami dapat menyediakan hampir semua hal dalam waktu sekitar 5 menit.

Solikhan: Wow... itu perbedaan yang sangat besar! Mungkin kalian juga punya cerita lucu untuk dibagikan...

Lawrence : Saya bertanggung jawab atas semua pusat data di Brasil dan sekali pada jam 3 pagi, telepon saya berdering. Jadi, jam 3 pagi saya bangun, katakanlah tidak terlalu senang karena saya tahu ada sesuatu yang terjadi dan sangat buruk... Jadi, manajer saya di sana menelepon saya dan berkata, "Ok, hei Lawrence , kami punya masalah di sini.. . alarm kebakaran dipicu di salah satu pusat data kami." Jadi, kami tidak tahu apa yang sedang terjadi, dan karena saya dulu tinggal dua blok jauhnya dari pusat data itu, saya membangunkan teman saya, yang saat itu tinggal bersama saya, dan saya berkata kepadanya: "Hei man , bangun kita harus pergi ke kantor...," dan dia menjawab, "Kamu gila? Saya tidak pergi ... ini jam 3 pagi ..." kata saya, "Tidak man, bangun mulai menelepon pers karena sesuatu yang sangat buruk akan terjadi..." Pada akhirnya kami pergi ke sana, dan pada akhirnya itu adalah alarm palsu; sistem itu sendiri gagal, tetapi ketika alarm disetel di pusat data, semua mesin dimatikan secara otomatis, jadi seperti yang dapat Anda bayangkan, sekitar tiga hingga empat ratus server, sistem penyimpanan, mainframe... semuanya mati. Jadi, kami memiliki

4 jam antara jam 3 pagi dan 7 pagi untuk menghidupkan kembali seluruh infrastruktur, dan beberapa sistem penyimpanan membutuhkan waktu sekitar 1,5 jam untuk melakukan semua pemeriksaan untuk memastikan data tidak rusak, tetapi pada akhirnya kami bisa melakukannya. Jadi, pada akhirnya itu menyenangkan, saya harus memanggil setiap insinyur; mereka tidak terlalu senang tentang itu, tetapi pada akhirnya, kami mendapat beberapa pizza dan banyak tertawa dan pada hari berikutnya kami pergi untuk merayakannya.

Solikhan: Itu sangat bagus. Apa yang saya rasakan adalah banyak gairah ketika Anda berbicara tentang apa yang Anda lakukan. Jadi, rasanya Anda sangat mencintai pekerjaan Anda, dan saya yakin ini sangat penting, bukan?

Lawrence : Ya, saya pikir ini fundamental... Jika Anda tidak menyukai apa yang Anda lakukan, maka saya tidak tahu mengapa Anda melakukannya. Jadi, ini adalah filosofi dasar saya untuk hidup saya. Saya hanya mencoba melakukan hal-hal yang saya sukai. Tentu saja, akan ada saat-saat Anda akan mengambil satu tugas sederhana yang mungkin bukan yang Anda inginkan, tetapi tidak apa-apa! Kadang-kadang Anda perlu melakukan tugas-tugas itu juga, tetapi secara umum saya bersemangat tentang teknologi dan saya tidak bisa memikirkan saya melakukan hal lain. Saya pergi bekerja dengan bahagia setiap hari; bagi saya itu seperti video game, saya pergi ke sana setiap hari dan bermain, tetapi bagi saya video game adalah tentang pengkodean dan pengaturan infrastruktur, layanan baru. Inilah yang saya suka lakukan.

Solikhan: Itu bagus. Saya percaya ini bisa menjadi nasihat yang sangat baik untuk siswa dan pembaca kami: Cintai apa yang Anda lakukan, dan Anda akan bahagia dan sukses.

Lawrence : Saya cukup yakin karena jika Anda tidak menyukainya, ubah saja. Jangan biarkan hidupmu seperti itu, hidup ini terlalu singkat untuk tidak dinikmati.

8.8 Pemantauan dan Pemberitahuan

Baiklah, kita tahu sekarang bagaimana cara memeriksa apakah kode kita siap produksi, bagaimana menjalankannya, dan yang paling penting, bagaimana mengotomatisasi seluruh proses ini. Apa itu cukup? Bayangkan Anda punya anak. Anda membesarkan anak ini, Anda mempersiapkan mereka untuk pergi ke sekolah, dan suatu hari, voila—anak Anda ada di sekolah! Bisakah Anda melonggarkan kendali dan memastikan semuanya baik-baik saja? Tentu saja tidak, Anda harus mengontrol nilai anak Anda, Anda harus tetap waspada jika terjadi sesuatu. Anda merasa perlu diberi tahu jika anak Anda mendapat masalah di sekolah atau jika mereka bolos sekolah. Itu benar, kita berbicara tentang mekanisme pemantauan dan peringatan. Perangkat lunak Anda seperti anak Anda—Anda menciptakannya, mengembangkannya, Anda menyukainya, Anda membuka jalan baginya untuk menghadapi dunia nyata, dan tentu saja Anda tidak dapat meninggalkannya tanpa diawasi. Bagaimana jika sistem mati di malam hari saat Anda tidur? Bagaimana jika banyak pengguna mulai menggunakan sistem Anda dan tidak siap untuk itu dan sistem itu mati? Ini adalah skenario nyata, omong-omong. Di EdEra kami menjalankan kursus online. Sistem kami selalu menggunakan hanya satu instans AWS, dan itu selalu cukup. Kami, sebenarnya, tidak pernah memikirkan pemantauan atau pengendalian yang ekstensif. Selama 3 tahun, itu hanya bekerja. Suatu hari kami meluncurkan kursus untuk guru Indonesia.

Kami memberi tahu mereka bahwa materi kursus akan dibuka pada tanggal 1 Desember pukul 3 sore. Itu adalah kursus online, jadi itu akan tersedia 24/7 tetapi untuk guru Ukraina yang dididik oleh sistem Soviet yang ketat, kata-kata "1 Desember pukul 3 sore"—ini seperti pesanan! Jadi 100.000 guru sekaligus mencoba mendaftar di platform. Jadi, pada tanggal 1 Desember tepatnya jam 3 sore, sistem kami baru saja down. Itu seperti serangan DDoS. Setelah situs web mati, kami mendapat serangan email dan panggilan telepon. Semua guru ini mulai menelepon dan mengirim email. Selama malam itu kami menjawab sekitar 1500 e-mail. Kami belum siap untuk itu, tetapi jika kami memiliki sistem pemantauan dan peringatan, kami setidaknya akan diperingatkan sebelumnya bahwa sumber daya server akan turun.

Bayangkan Anda membuka alat pengelola tugas dan mulai menatapnya mendeteksi beberapa anomali dan aktivitas aneh. Inilah yang dimaksud dengan alat pemantauan dan peringatan. Pada dasarnya, Anda mengonfigurasi beberapa tolok ukur dan mekanisme peringatan (mis., "Kirim saya email jika CPU berjalan di atas 80%") dan alat ini akan melakukannya untuk Anda. Biasanya penyedia hosting juga memberikan monitoring dan alerting. Anda juga dapat menginstal dan menyiapkan beberapa alat khusus untuk itu—misalnya, pagerduty (https://www.pagerduty.com/). Alat ini akan memantau waktu aktif layanan Anda dan memberi peringatan melalui saluran tertentu (bisa berupa apa saja mulai dari panggilan telepon hingga pesan kendur) jika terjadi kesalahan. Pingdom (https://www.pingdom.com/) memungkinkan pemantauan ketersediaan dan kinerja situs web Anda dan memberi tahu Anda jika ada sesuatu yang melampaui atau di bawah tolok ukur yang ditentukan. Untuk platform pembelajaran kami, kami menggunakan Heroku, yang menyediakan alat pemantauan dan peringatan (Anda hanya perlu membayarnya). Jika Anda menggunakan penyedia hosting awan dan sangat penting bagi Anda untuk tetap hidup, berinvestasilah dalam langganan berbayar untuk menggunakan alat yang ditawarkan oleh penyedia ini untuk mengawasi kesehatan anak Anda!

8.9 Analitik

Di bagian sebelumnya, kami membahas betapa pentingnya untuk tetap diberi tahu jika sistem Anda mati. Bagaimana jika tidak turun bukan karena sangat bagus tetapi hanya karena tidak ada yang menggunakannya? Bagaimana kami tahu jika perangkat lunak Anda sedang digunakan sama sekali? Bagaimana kita tahu apakah orang menyukainya atau tidak? Bagaimana kita bisa tahu siapa pengguna kita? Ternyata, menggunakan layanan analitik di sistem Anda sama pentingnya dengan memantaunya. Anda pasti pernah mendengar tentang google analytics. Alat ini hadir hampir di setiap situs web yang Anda gunakan saat ini. Ini memungkinkan pemilik layanan web untuk mengetahui kapan dan bagaimana layanan mereka digunakan. Ini memberi Anda dasbor ikhtisar real-time yang bagus yang memberi tahu Anda berapa banyak pengguna aktif di situs web, wilayah asal pengguna, halaman apa yang paling banyak dikunjungi, berapa lama rata-rata sesi berlangsung, dll. Ini sangat penting angka, dan ada baiknya memiliki analis data untuk menganalisis data tersebut dan bekerja sama dengan tim untuk menemukan titik terlemah dari sistem. Misalnya, hanya dengan menganalisis data semacam ini dengan cermat di OptioPay, kami dapat menyadari bahwa dalam perjalanan pengguna kami, kami kehilangan banyak konversi di halaman arahan kami.

Mendesain ulang dan membuatnya lebih menarik memberi kami kemungkinan untuk secara signifikan meningkatkan tingkat konversi secara keseluruhan.

Ada banyak alat selain analitik Google yang memberi Anda sarana untuk mendapatkan data penting dari penggunaan layanan Anda. Mixpanel (https://mixpanel.com/), misalnya, adalah platform yang bagus dan mudah digunakan. Ini juga menyediakan API untuk membuat peristiwa khusus, sehingga dapat dianalisis setelahnya. Itu membutuhkan pengetahuan pemrograman, tetapi untuk itulah kami memiliki pengembang kami, bukan? Beberapa layanan analitik bahkan memberi Anda peta panas, artinya Anda benar-benar dapat melihat bagian mana dari layanan web Anda yang paling sering digunakan. Ini akan memberi Anda wawasan tentang cara meningkatkan UI/UX Anda untuk lebih melibatkan pengguna. Apa pun yang Anda putuskan untuk digunakan, jangan lupakan masalah privasi. Saat ini teknologi memungkinkan pengumpulan begitu banyak data dari pengguna, tetapi jangan terlalu bersemangat dengan itu; jika tidak, hal itu mungkin berdampak negatif pada bisnis Anda. Bahkan jika Anda merasa tahu apa yang Anda lakukan, masalah privasi selalu dapat memengaruhi Anda (jangan lupa, misalnya, tentang kasus ini: https://en.wikipedia.org/wiki/Facebook%E2%80%93Cambridge_Analytica_data_scandal).

Hosting dan Membuat CI/CD untuk Platform Kami

Sekarang setelah kita mendapatkan semua teori itu, saatnya kita mengotori tangan kita. Ada beberapa pilihan di pasar untuk CI/CD dan aplikasi hosting. Ada yang berbayar, ada yang gratis, dan ada yang hybrid. Ketika kami mulai menulis buku ini, kami secara internal memutuskan bahwa kami akan menerapkan aplikasi kami di Heroku. Heroku menawarkan cara yang cukup bagus dan mudah untuk menyebarkan aplikasi yang ditulis dalam beberapa bahasa pemrograman. Selain itu, ia juga menawarkan layanan terintegrasi seperti database, sehingga memudahkan integrasi. Layanan yang akan kami gunakan untuk menyebarkan aplikasi kami di Heroku semuanya gratis, tetapi terlepas dari itu, Heroku meminta detail pembayaran Anda. Jika Anda takut untuk memberikan detail kartu kredit Anda ke Heroku, kami menyarankan Anda untuk setidaknya melanjutkan dengan kami karena sebagian besar langkahnya cukup mudah dipahami bahkan jika Anda tidak benar-benar melakukannya.

8.10 Hosting

Urutan bisnis pertama adalah membuat akun Heroku jika Anda belum memilikinya. Buka https://www.heroku.com/ dan buat akun. Download juga Heroku CLI (https://devcenter.heroku.com/articles/heroku-cli#download-and-install) dan dapatkan kode aplikasinya dari folder “courses”. Unduh dan atur git, buat akun github jika Anda belum memilikinya, dan buat repositori untuk kode sumber yang baru saja Anda unduh dan salin URL repositori. Kemudian hanya mengkloning repositori. Bagi kami itu seperti berikut:

```
$ git clone git@github.com:rpvilao/courses-test.git
```

Ini akan mengkloning repositori kosong yang baru saja Anda buat. Ekstrak kode sumber ke direktori baru ini (tes-kursus dalam kasus kami; milik Anda mungkin berbeda tergantung pada nama yang Anda berikan ke repositori). Apa yang akan kita lakukan sekarang adalah menambahkan file ke repositori

```
$ git add .gitignore .travis.yml *
$ git commit -a -m "Init"
$ git push origin master
```

Pada titik ini kode kami sudah ada di github dan dapat diakses oleh semua orang (jika Anda membuat repositori publik). Seperti yang Anda ingat, Git adalah sistem kontrol versi, artinya Anda membuat perubahan pada file dan menavigasi dalam riwayat dan bekerja dengan baik dengan kolaborator proyek lainnya. Selamat, karena Anda baru saja menyelesaikan langkah pertama, namun penting, untuk mencapai integrasi berkelanjutan. Tanpa sistem kontrol versi, integrasi berkelanjutan tidak akan mungkin… Ya, tetapi Anda tidak dapat membayangkan betapa sulit dan rawan kesalahannya, jadi mari kita asumsikan bahwa memang tidak mungkin untuk standar saat ini. Sekarang saatnya untuk kembali ke Heroku. Hal pertama yang perlu kita lakukan adalah login dan membuat aplikasi.

```
$ heroku login
$ heroku create
Creating app... done, ⬢ stark-headland-67097
https://stark-headland-67097.herokuapp.com/ | https://git.heroku.com/stark- headland-67097.git
```

Karena aplikasi kita membutuhkan database, mari kita buat. Kami memutuskan untuk menggunakan postgresql. Aplikasi sudah siap untuk menggunakannya, jadi jalankan saja yang berikut ini:

```
$ heroku addons:create heroku-postgresql:hobby-dev
creating heroku-postgresql:hobby-dev on ⬢ stark-headland-67097... free
Database has been created and is available
! This database is empty. If upgrading, you can transfer
! data from another database with pg:copy
Created postgresql-regular-72889 as DATABASE_URL
Use heroku addons:docs heroku-postgresql to view documentation
```

Dengan menjalankan "heroku config" Anda akan dapat melihat cara terhubung ke database ini. Karena aplikasi kita adalah aplikasi Java, kita perlu meletakkan URL koneksi dalam bentuk JDBC. Langkah ini tidak terlalu jelas, tapi jangan khawatir, sudah bagus kita membuat aplikasi dan database hanya dengan beberapa perintah! Jika Anda melihat string koneksi, Anda akan melihat Anda memiliki sesuatu yang mirip dengan

```
postgres://USERNAME:PASSWORD@ec2-XX-21-YYY-ZZZ.compute-1.amazonaws.com:5432/DB_NAME
```

Untuk membuat string JDBC ini, Anda harus mencocokkan dengan yang berikut:

```
jdbc:postgresql://ec2-XX-21-YYY-
ZZZ.compute-.amazonaws.com:5432/DB_NAME?ss
lmode=require&user=USERNAME&password=PASSWORD&currentSchema=public
```

Ini adalah string yang perlu Anda gunakan untuk membuat bootstrap database kami. Mari kita lakukan sekarang menggunakan liquibase (kami menyiapkan profil, sehingga kami dapat membuat database produksi kami).

```
$DATABASE_URL_JDBC="jdbc:postgresql://ec2-XX-21-YYY-ZZZ.compute-1.
```

```
amazonaws.com:5432/DB_NAME?sslmode=require&user=USERNAME&password=
PASSWORD& currentSchema=public" mvn liquibase:update -pl liquibase -Pheroku
```

Kami hampir siap, mari kita gunakan aplikasinya

```
$git push heroku master
```

Serius, keren banget kan? Hal-hal yang mungkin Anda tanyakan saat ini: Bagaimana Heroku tahu cara menjalankan aplikasi saya? Jawabannya ada di Procfile di root proyek. Bagaimana aplikasi kita tahu cara terhubung ke database? Heroku mengekspos URL database dalam variabel lingkungan yang disebut DATABASE_URL dan aplikasi kita sudah mengharapkan itu. Anda ingat sebelumnya, variabel ini tidak dalam bentuk JDBC, jadi kami membacanya, mengubahnya, dan membuat string terakhir dalam kode. Lihatlah file PersistenceLayerConfig di modul lapisan persistensi. Setelah build selesai dan berhasil, buka saja aplikasi menggunakan

```
$heroku open
```

Anda mungkin mendapatkan kesalahan pada awalnya karena aplikasi membutuhkan waktu untuk memuat; tekan saja refresh jika ini terjadi.

Meskipun ini tampak cukup keren, bagaimana kita memastikan aplikasi dalam keadaan deployable? Selanjutnya, bayangkan Anda memiliki tim yang terdiri dari enam orang yang menulis kode pada aplikasi… Sangat mungkin bahwa pada suatu saat kode tersebut mungkin tidak dapat digunakan, bukan? Kita hanyalah manusia… Di sinilah Integrasi Berkelanjutan berperan. Kami telah menyatakan bahwa memiliki sistem kontrol versi adalah langkah pertama untuk mencapai CI, tetapi sekarang kami akan menyelami topik dan menyiapkan serangkaian verifikasi di mana kami berasumsi bahwa jika mereka lulus, kode kami dalam keadaan dapat digunakan. Ayo pergi!

8.11 Integrasi dan Penerapan Berkelanjutan

Sekadar rekap, Continuous Integration (CI) adalah serangkaian proses yang memungkinkan orang untuk bekerja sama dan berkontribusi untuk proyek yang sama tanpa membuatnya sulit untuk diintegrasikan. Saat ini secara harfiah berarti memiliki kontrol versi, build otomatis, dan serangkaian pengujian otomatis yang berjalan setiap kali Anda membuat perubahan pada build. Kasus kami tidak berbeda—kami telah membuat repositori kontrol versi di latihan sebelumnya dan Anda melihat bahwa build kami berjalan secara otomatis hanya dengan memanggil perintah. Sekarang saatnya untuk menangani bagian di mana kita menjalankan serangkaian tes setelah kita melakukan beberapa perubahan pada cabang tertentu dari repositori—dalam hal ini cabang master, yang menunjuk ke versi produksi kita saat ini. Meskipun Anda dapat dan harus menjalankan pengujian otomatis untuk setiap perubahan pada cabang apa pun, kami melakukannya hanya pada master untuk kesederhanaan dan untuk memberikan contoh.

Kami juga dapat menggunakan Heroku untuk membangun sistem CI kami; masalahnya kita harus membayar untuk menggunakan fitur ini. Untungnya, ada lebih banyak alternatif dan bahkan alternatif yang dapat diintegrasikan dengan Heroku. Dalam contoh ini, kita akan menggunakan TravisCI dan berintegrasi dengan GitHub dan Heroku. Alur untuk CI kami adalah sebagai berikut: untuk setiap push di cabang master, Travis akan menerima

pesan dari github yang mengatakan bahwa ada perubahan dalam repositori. Travis kemudian akan membaca konfigurasi dan memeriksa apa yang harus dilakukan untuk melakukan build. Setelah build selesai, jika perubahan yang baru saja kita lakukan berlaku untuk apa yang ditentukan dalam file konfigurasi dan jika memang berlaku, Travis akan menerapkan perubahan ke lingkungan produksi kita di Heroku hanya jika tes lulus dan build berhasil. Mulailah dengan membuat akun Travis (https://travis-ci.org/). Setelah Anda masuk, hubungkan akun Travis yang baru Anda buat ke akun github Anda dan pilih repositori untuk proyek kami. Sekarang kita perlu mengizinkan Travis untuk dapat menerapkan perubahan di Heroku jika build dan tes lulus. Untuk itu kita perlu membuat token di sisi Heroku, menandatanganinya dengan Travis dan meletakkannya di file konfigurasi Travis kita.

```
$ heroku auth:token
› Warning: token will expire 06/18/2019
› Use heroku authorizations:create to generate a long-term token 258cfb90-XXXX-
4720-XXX-9bfba5332254
```

Jadi sekarang kita perlu mengatur token ini di file konfigurasi Travis kita, sehingga Travis dapat membacanya dan melakukan panggilan ke Heroku. Hum… ada yang aneh… bukankah itu pelanggaran keamanan besar, memperlihatkan token ke dunia—terutama di repositori publik? Benar, itulah sebabnya Travis memungkinkan cara untuk mengenkripsi bagian teks ini dengan kunci publik yang hanya dapat didekripsi dengan kunci pribadi (hanya tersedia di Travis dan di akun Anda); ini disebut enkripsi asimetris—teknik yang sama yang digunakan browser Anda untuk melakukan panggilan aman melalui HTTP. Kembali ke topik, ada skrip kecil di root proyek yang disebut travis-encrypt.sh (https://Gist.github.com/openscript) untuk menyelesaikan pekerjaan ini. Ganti "pengguna/repo" dengan nama pengguna dan repositori Anda di Travis dan ingat bahwa tokennya berbeda untuk Anda!

```
$ ./travis-encrypt.sh -r user/repo -e 258cfb90-XXXX-4720-XXX-9bfba5332254
```

Ini akan memberi Anda string terenkripsi yang perlu Anda tempatkan dalam file bernama ".travis.yml". File akan terlihat mirip dengan

```
language: java
jdk:
- openjdk8
script: mvn clean install -Ptest
cache:
directories:
- $HOME/.m2/
deploy:
provider: heroku
app: HEROKU-APP-NAME  api_key:
secure: YOUR-API-KEY
run: "DATABASE_URL_JDBC=$DB_PROD mvn liquibase:update
-pl liquibase
-Pheroku"
on: master
```

Mari kita pergi melalui file. Kami mendefinisikan bahwa proyek ditulis dalam Java dan kami ingin menggunakan openjdk8 untuk menjalankan build. Kemudian kita tentukan bagaimana kita ingin menjalankan build dengan memanggil "maven install" dengan profil bernama "test". Untuk membuat pembangunan lebih cepat, kami melakukan caching semua artefak maven yang digunakan proyek kami; ini berarti mereka hanya akan diunduh sekali dan tidak setiap kali build berjalan (ini mempercepat proses sekitar 75% untuk panggilan berikutnya tergantung pada jaringan tentunya). Bagian terakhir adalah Continuous Deployment kami, di mana kami menentukan bahwa jika build berhasil dan cabangnya adalah "master", kami ingin menerapkannya di Heroku bersama dengan menjalankan migrasi database yang diperlukan (bagian "run"). Kami hampir selesai. Agar ini berfungsi, kita hanya perlu membuat variabel lingkungan dengan koneksi JDBC di Travis. Buka Travis, lebih banyak opsi, pengaturan, dan cukup atur variabel lingkungan yang disebut DB_PROD dengan string koneksi JDBC (jdbc:postgresql…) yang telah kita definisikan sebelumnya di bagian CI. Siap untuk mengujinya? Lakukan saja perubahan Anda

```
$ git commit -a -m "Adding Travis' configuration"
```

dan dorong mereka!

```
$ git push origin master
```

Segarkan dasbor Travis dan periksa apakah build Anda berjalan. Setelah selesai, Anda harus menerapkan perubahan di URL Heroku aplikasi Anda (jika Anda tidak mengingatnya, ketik saja "heroku open").

Dan hanya dengan itu kami membuat pipa CI/CD sederhana untuk proyek kami. Apakah itu layak? Ya itu! Kami memiliki aplikasi kami dan berjalan di alamat ini: https://eleplatform.herokuapp.com. Tentu saja, dalam skenario nyata, hal-hal tidak sesederhana itu, tetapi prinsip yang sama berlaku. Sangat umum bahwa, misalnya, Anda memiliki versi pementasan yang pada dasarnya sama dengan produksi tetapi untuk tujuan pengujian sebelum Anda ditayangkan. Ini dapat dicapai dengan membuat aplikasi dan database lain di Heroku dan mengonfigurasi pipeline penerapan menggunakan Travis, di mana Anda menyatakan bahwa jika cabang sedang staging, maka Anda menerapkan ke aplikasi stage dan bukan yang produksi. Hal lain yang perlu Anda pertimbangkan adalah bahwa dalam skenario nyata, sangat kecil kemungkinannya Anda akan menemukan aplikasi backend untuk diterapkan secara otomatis ke produksi secara push. Tidak hanya itu terlalu berisiko, tetapi juga terkadang mereka perlu memiliki urutan eksekusi tertentu untuk mencapai penerapan zero-downtime dengan, misalnya, penerapan hijau biru (juga dikenal sebagai canary) di mana perubahan diluncurkan pada pendekatan bertahap. Bagaimanapun, ini adalah contoh akademis yang sangat baik untuk tujuan buku ini, dan sekarang setelah Anda mengetahui cara melakukannya, Anda dapat membangun aplikasi staging dan saluran CI/ CD untuk aplikasi kursus kami. Apakah Anda siap untuk tantangan itu?

8.12 Ringkasan

Dalam bab ini kami berhasil mengenakan topi seorang profesional DevOps. Atau mungkin Anda ingin menyebutnya sebagai insinyur infrastruktur, atau Insinyur Keandalan Situs... Sebut saja apa pun yang Anda inginkan, yang penting sekarang Anda tahu apa artinya

"ditayangkan". Ini sama dengan "menyebarkan ke produksi." Anda tahu cara mengotomatiskan proses ini. Anda juga mempelajari betapa pentingnya memastikan bahwa pemeriksaan dasar lulus sebelum penerapan. Kami telah melihat betapa pentingnya memantau perangkat lunak Anda dalam produksi dan menyiapkan alat peringatan bagi Anda untuk mendapatkan pemberitahuan jika terjadi kesalahan. Kami juga telah membahas bahwa penting untuk memiliki beberapa alat analitik, jadi kami tahu bagaimana sistem kami digunakan dan dapat meningkatkannya berdasarkan informasi ini. Sepertinya ... kita sudah selesai? Sama sekali tidak! Dengan penerapan pertama, perjalanan kami baru saja dimulai. Aplikasi ini akan dikembangkan dan ditingkatkan dari waktu ke waktu. Saat ini kami baru saja menerapkan MVP (Minimum Viable Product) dan masih ada pekerjaan yang menunggu kami. Di bab berikutnya, kami akan menjelaskan apa yang biasanya terjadi setelah penerapan pertama— selain perayaan, pekerjaan berlanjut, proses sedang berlangsung, produk harus tetap dipertahankan, dikembangkan, mungkin difaktorkan ulang pada suatu saat, dan bahkan suatu hari nanti. didesain ulang, karena waktu terus berjalan dan tren selalu berubah. Jadi, mari kita lanjutkan ke bab berikutnya dan temukan apa yang terjadi pada perangkat lunak setelah di-deploy!!

BAB 9
MEMELIHARA DAN MENINGKATKAN PERANGKAT LUNAK ANDA

Pada bab sebelumnya, kami akhirnya menerapkan platform e-learning kami ke produksi. Sekarang kami telah menjalankannya di Heroku (http://eleplatform.herokuapp.com/). Namun, proses pembangunan belum selesai. Sebenarnya, perjalanan kami baru saja dimulai. Ini seperti bayi. Anda merencanakannya (atau tidak), mempersiapkannya, kemudian Anda membayangkannya, kemudian ada masa kehamilan yang panjang, di mana Anda mempersiapkan pikiran Anda dan membeli semua hal yang Anda butuhkan. Anda mempersiapkan kamar bayi, berbicara dengan dokter, memilih rumah sakit yang tepat, dan memastikan dunia dan lingkungan di sekitar Anda siap untuk menerima manusia baru. Kemudian bayi lahir. Ya! Kegembiraan! Pada titik tertentu Anda menggendong bayi Anda dan Anda perlahan-lahan menyadari bahwa semua yang telah Anda lakukan selama 9 bulan terakhir sebenarnya tidak ada apa-apanya dibandingkan dengan apa yang sebenarnya menunggu Anda. Anda baru saja melahirkan manusia baru ke dunia. Ini bukan hanya bayi, ini adalah kehidupan baru, perjalanan baru, jalan baru, inspirasi baru, dan dunia baru. Dan tanggung jawab untuk semua ini ada pada Anda. Perangkat lunak yang Anda siapkan untuk dikirimkan ke dunia tidak berbeda. Anda juga menaruh banyak cinta dalam penciptaannya. Anda memastikan bahwa itu sehat dan dekat dengan bebas bug. Setelah online, Anda ingin membagikannya ke seluruh dunia karena Anda sangat senang akhirnya berhasil! Setelah aktif dan berjalan dan dapat diakses oleh seluruh dunia, Anda tahu bahwa hidupnya baru saja dimulai dan Anda bertanggung jawab atas kualitas dan kesuksesannya. Tidak ada waktu untuk bersantai. Sekarang saatnya untuk pekerjaan yang sebenarnya; semua yang telah kami lakukan sebelumnya hanyalah sedikit persiapan. Mari kita lihat apa yang menanti kita dalam perjalanan ini. Dalam perjalanan menuju kesempurnaannya, Anda akan terus-menerus terlibat dalam dua kegiatan: memelihara dan meningkatkan.

9.1 Mempertahankan

Anda pasti pernah mendengar istilah “memelihara”. Ini adalah sesuatu yang harus Anda lakukan dengan bayi Anda—memeliharanya agar tetap hidup. Pastikan infrastruktur Anda memungkinkan Anda untuk menjaga perangkat lunak Anda tetap solid. Pastikan bahwa jika Anda membayar untuk server Anda bahwa semua pembayaran dilakukan tepat waktu. Jika tidak, Anda mungkin berakhir dalam situasi yang memalukan di mana layanan Anda turun karena beberapa pembayaran yang gagal. Pastikan tidak ada kegagalan yang dapat menghancurkan bisnis Anda.

9.2 Cadangan

Salah satu pilar dari maintainability adalah backup. Sesuatu terjadi. Data Anda dapat secara tidak sengaja dihancurkan karena beberapa alasan. Basis data dapat disusupi, peretas

dapat menyerang Anda, server virtual Anda mungkin turun dan tidak akan pernah bisa naik lagi. Beberapa insinyur mungkin secara tidak sengaja menjalankan "rm -rf" dan semuanya hilang.

Perintah rm menghapus file. Bendera -r melakukannya secara rekursif di folder saat ini dan bendera -f memaksa penghapusan, tidak pernah meminta pengguna tentang penghapusan. Perintah ini cukup berbahaya, karena akan menghapus semuanya secara rekursif tanpa mengajukan pertanyaan apa pun kepada Anda!

Suatu kali saya (Solikhan) secara tidak sengaja membuat folder bernama ~ di mesin Ubuntu saya. Setelah saya menyadari bahwa saya memutuskan untuk menghapusnya. rm -rf~. Saya menonton dalam gerakan lambat bagaimana semua barang saya menghilang, dan sistem saya hilang begitu saja (~ di sistem unix menunjuk ke direktori home pengguna). Tidak super pintar. Jika Anda berpikir bahwa hal bodoh semacam ini tidak akan pernah terjadi pada Anda, tunggulah sampai pertama kali. Insiden "rm -rf" adalah alasan yang cukup populer di antara beberapa yang secara tidak sengaja menghapus data. Itu bahkan terjadi di Pixar dengan seluruh film Toy Story 2! Itulah mengapa sangat penting untuk membuat cadangan data penting. Ketika kita berbicara tentang sistem yang kompleks, Anda mungkin mencadangkan seluruh sistem atau hanya penyimpanan datanya. Mencadangkan berarti sesekali semua data penting disalin ke tempat lain, sehingga jika terjadi sesuatu sistem dapat dipulihkan. Anda harus memastikan bahwa file dapat dengan mudah diambil dari cadangan, dan setelah disalin kembali ke sistem, sistem akan bekerja seperti biasa.

Pixar sebenarnya memiliki salinan cadangan dari film Toy Story, tetapi rusak (!!!); oleh karena itu tidak mungkin untuk memulihkan file darinya. Jika Anda menggunakan beberapa penyedia layanan hosting awan—misalnya, AWS—Anda mungkin ingin menggunakan layanan pencadangan yang ditawarkan oleh mereka. Mereka dapat diandalkan dan dapat dikonfigurasi. Jika sistem Anda tidak sering berubah, Anda mungkin memilih jeda waktu yang lebih lama untuk mencadangkan data Anda. Jika data di sistem Anda terus berubah dan perubahan ini sangat penting untuk fungsi yang benar, maka Anda mungkin ingin memilih interval pencadangan yang lebih sering (misalnya, setiap hari setiap kali aplikasi jarang digunakan). Omong-omong, jika Anda penasaran dengan film Toy Story 2, ceritanya berakhir bahagia. Salah satu karyawan yang harus bekerja dari rumah karena memiliki bayi kecil memiliki salinan lengkap film tersebut di komputer pribadinya. Fakta ini memungkinkan Pixar untuk mengambil file dan berhasil mengirimkan film ke penonton. Jangan mengandalkan kebetulan seperti itu untuk bisnis Anda!

Replikasi

Terkadang data sangat penting sehingga setiap detik menjadi penting. Dalam hal ini, cadangan sederhana tidak cukup. Untuk kasus ini, data harus direplikasi dan disimpan di beberapa perangkat penyimpanan sehingga setiap kali terjadi perubahan, data tersebut disebarkan di antara semua replika. Cluster ketersediaan tinggi biasanya mengikuti skema master-slave untuk mencapai efek ini.

Master-slave adalah skema di mana ada perangkat atau layanan utama yang melakukan semua pekerjaan tetapi juga mereplikasi semua operasi ke node lain yang hanya menunggu untuk diambil jika, karena alasan tertentu, yang utama gagal.

Saat beberapa perangkat mati, permintaan dikirim ke perangkat lain. Seluruh proses harus benar-benar transparan bagi pengguna. Proses beralih ke perangkat yang berlebihan disebut failover. Sistem cloud seperti AWS, MS Azure, Google Cloud, dll., menawarkan mekanisme replikasi dan failover di antara layanan mereka.

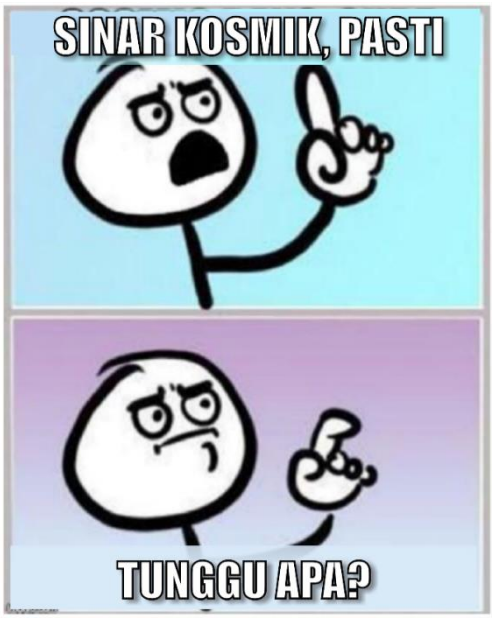
9.3 Bencana Alam dan Sinar Kosmik

Telah terbukti selama bertahun-tahun bahwa kesalahan manusia adalah alasan paling umum kegagalan sistem. Namun, ada kasus langka yang tidak dapat dikendalikan oleh manusia. Ya, kita berbicara tentang alam dan fenomenanya. Anda dapat memanaskan rumah atau membuatnya lebih dingin menggunakan perangkat AC, tetapi Anda tidak dapat mengontrol suhu di luar. Setiap tahun bencana alam merenggut ratusan nyawa manusia, dan kita tidak bisa berbuat banyak. Bayangkan sistem Anda berjalan di server bare metal di seberang jalan. Bayangkan ada gempa bumi dan seluruh area hancur. Anda kehilangan semua yang telah Anda kerjakan selama beberapa tahun terakhir dan bisnis Anda berakhir. Anda tidak dapat berbuat banyak tentang hal itu. Hanya bercanda. Tentu saja Anda bisa. Bahkan ada istilah dalam ilmu komputer yang disebut pemulihan bencana.

Pemulihan bencana (DR) melibatkan serangkaian kebijakan, alat, dan prosedur untuk memungkinkan pemulihan atau kelanjutan infrastruktur dan sistem teknologi penting setelah bencana alam atau yang disebabkan oleh manusia. <…>. Pemulihan bencana dapat dianggap sebagai bagian dari kelangsungan bisnis.

—Wikipedia (https://en.wikipedia.org/wiki/Disaster_recovery)

Tentu saja, jika kita berbicara tentang proyek kecil dan kumpulan data kecil, pemulihan bencana Anda dapat berupa hard drive eksternal yang disembunyikan di tempat teman Anda. Tetapi jika itu adalah sejumlah besar data penting, maka Anda harus hati-hati memikirkan rencana pemulihan bencana. Biasanya layanan cloud besar menyediakan paket tersebut untuk Anda, dan Anda dapat tidur nyenyak. Bagaimanapun, penting untuk menyadari keberadaan hal-hal ini dan bersiap untuk yang terburuk dan berharap untuk yang terbaik. Karena kita berbicara tentang penyebab alami dan dampaknya pada perangkat lunak, kami pikir cukup menarik dan menyenangkan untuk berbicara tentang sinar kosmik. Apa? (Gambar 9-1).



Gambar 9-1. Tunggu apa? Sinar kosmik?

Ya! Sinar kosmik dapat mengakibatkan kesalahan pada RAM (Random Access Memory). Singkat cerita, sinar kosmik dapat menghasilkan efek di atmosfer yang dapat menyebabkan perubahan potensial pada memori perangkat yang berjalan, sehingga menyebabkan kesalahan perangkat lunak. Berapa probabilitas kesalahan seperti itu terjadi? Wikipedia mengutip sebuah studi yang dilakukan oleh IBM:

Studi oleh IBM pada 1990-an menunjukkan bahwa komputer biasanya mengalami sekitar satu kesalahan yang disebabkan oleh sinar kosmik per 256 megabyte RAM per bulan.

—Wikipedia: (https://en.wikipedia.org/wiki/Cosmic_ ray#Effect_on_electronics

Apa yang harus Anda lakukan untuk melindungi diri dari sinar kosmik? Mungkin tidak ada. Kami baru saja menambahkan penyebab alami ini untuk meningkatkan rasa ingin tahu Anda. Chip modern sebagian besar dilindungi dari penyebab ini dengan menerapkan checksum, jadi sangat kecil kemungkinannya bahwa hal itu dapat memengaruhi sistem, tetapi masih mungkin. Hal yang sama terjadi dengan radioaktivitas misalnya. Sistem yang terpapar pada kondisi yang tidak optimal—misalnya, satelit—perlu mempertimbangkan hal ini.

9.4 Meningkatkan

Produk Anda, bahkan terpelihara dengan baik, perlu terus ditingkatkan. Jika tidak, tren dan persaingan baru akan menghancurkannya di lautan teknologi yang gila ini. Bayangkan bahwa sistem operasi Windows akan terlihat sama seperti pada versi pertamanya. Apakah ada yang akan menggunakannya? Pertimbangkan perangkat lunak atau aplikasi seluler apa pun yang Anda gunakan—jika tidak terus-menerus memanaskan Anda dengan pembaruan dan fitur baru, berapa lama Anda akan menggunakannya?

Kita semua adalah manusia—sangat mudah bagi kita untuk terganggu; sangat mudah bagi kita untuk menjadi malas; sangat sulit untuk setia pada satu merek dagang atau produk tertentu. Kami ingin mencoba hal yang berbeda, dan dunia terus bergerak maju di sekitar

kami. Pada saat yang sama, kami sangat curiga terhadap hal-hal baru, dan kami lebih suka berpegang pada teknologi lama daripada mencoba yang baru (karena membutuhkan waktu dan usaha), kecuali semua orang mulai membicarakannya dan rasa ingin tahu alami kami menang. Jika Anda dengan benar dan terus-menerus menggunakan karakteristik manusia yang kontroversial ini dalam pengembangan perangkat lunak Anda, Anda mungkin memenangkan pasar. Dengan demikian, Anda membutuhkan orang-orang yang pandai dalam pemasaran, Anda membutuhkan orang-orang yang baik dalam penjualan, Anda membutuhkan departemen R&D (Penelitian dan Pengembangan) dan layanan pelanggan yang kuat, dan Anda perlu membangun getaran yang kuat terhadap empati pengguna di tim Anda. Ini sangat penting. Tidak masalah jika Anda B2B atau B2C atau B2B2C atau apa pun.

Selalu ada beberapa pengguna akhir produk Anda, meskipun dalam beberapa kasus tertentu sulit untuk menjangkau pengguna ini karena rantai besar B2B sebelum pelanggan akhir. Pikirkan tentang pengguna ini, jadilah pengguna ini, buat semua orang di tim merasakan pengguna ini, dan terbukalah terhadap ide-ide baru bahkan jika mereka tampaknya benar-benar bertentangan dengan arah awal produk.

Jika semuanya benar, basis pelanggan Anda akan tumbuh. Pastikan Anda memiliki rencana penskalaan Anda. Pengguna akan memiliki pertanyaan; mereka akan membutuhkan cara untuk menghubungi Anda dan menanyakan pertanyaan-pertanyaan itu. Pastikan Anda memiliki cara mudah untuk mengumpulkan umpan balik pengguna dan memberi mereka layanan pelanggan yang efisien. Basis kode akan tumbuh, dan pada titik tertentu, itu akan membutuhkan beberapa refactoring atau bahkan penulisan ulang. Pastikan Anda mengalokasikan cukup waktu untuk hal-hal seperti itu—itu penting!

Karena dunia akan terus bergerak maju dan tren desain baru akan muncul setiap hari, desain produk Anda juga harus beradaptasi dengan perubahan tersebut. Terkadang Anda perlu melakukan proses desain ulang untuk produk Anda. Bersiaplah untuk itu juga. Secara umum, selalu siap untuk perubahan dan untuk menerima perubahan itu dengan senyum dan energi positif. Jika Anda telah membaca buku Creativity, Inc.: Overcoming the Unseen Forces That Stand in the Way of True Inspiration (http://a.co/8kdojHA), Anda tahu bagaimana seluruh konsep kartun animasi Pixar bisa berubah setelah 1 tahun mengerjakannya. Karena mereka bertaruh pada kualitas dan kreativitas, dan suara semua orang diperhitungkan.

9.5 Penskalaan

Di EdEra, seperti yang telah kami tunjukkan, kami menjalankan platform kursus online. Sejak awal, telah menggunakan satu instans EC2 di AWS.

AWS EC2 adalah Layanan web amazon yang memungkinkan Anda untuk menjalankan instans tujuan umum yang dapat dengan mudah sesuai dengan kebutuhan komputasi Anda.

Memiliki satu contoh untuk itu benar-benar baik-baik saja selama 3 tahun. Pada bulan Desember 2017, kami memutuskan untuk meluncurkan kursus untuk guru. Beruntung bagi kami, kursus ini dianggap oleh Kementerian Pendidikan sebagai kursus wajib bagi para guru yang akan mengajar di sekolah dasar selama tahun ajaran 2018 hingga 2019. Kami kira-kira berbicara tentang 20.000 guru. Solikhan ingat saat kakaknya Joseph menelepon dengan

panik. Dia telah menghadiri pertemuan di Kementerian dan orang-orang skeptis; mereka bertanya-tanya apakah server kami akan bertahan dengan 20.000 pengguna. Joseph memberi tahu mereka bahwa EdEra memiliki insinyur terbaik di dunia, dan tidak ada yang salah. Tapi dia panik. Dia bertanya kepada Solikhan apakah mungkin jumlah guru itu akan mematikan server EdEra atau tidak. Solikhan berpikir, "Yah, kursus tersedia 24/7, tidak akan pernah ada 20.000 pengguna secara bersamaan di platform." Seperti yang dikatakan Joseph di rapat, tidak ada yang salah (Gambar 9-2)!

Ternyata kami tidak memikirkan karakteristik yang sangat spesifik dari target audiens kami. Kita berbicara tentang guru yang dididik selama rezim Soviet. Kami sudah membicarakan kasus ini sebelumnya dalam buku ini, tetapi izinkan kami mengingatkan Anda. Apa yang terjadi adalah bahwa kata "wajib" dalam kombinasi dengan tanggal dan waktu mulai spesifik kursus menghasilkan ajakan bertindak. Semua guru Ukraina berpikir bahwa kursus itu wajib, bukan hanya guru yang memenuhi syarat untuk mengajar sekolah dasar di tahun berikutnya. Jadi, 1 Desember, pada waktu tertentu, hampir 100.000 guru mencoba mendaftar di platform kami.

Gambar 9-2. Gambar ini diambil sekitar 5 detik sebelum anak kucing itu jatuh

Tentu saja Kami tidak siap untuk itu. Kami harus menemukan solusi penskalaan dengan cepat. Kami beruntung, karena cukup mudah untuk dicapai jika Anda menggunakan AWS. Selama kursus itu, kami mengalami pelambatan besar pada server kami selama hari-hari ketika modul baru diterbitkan dan ketika tes terakhir keluar. Kami memiliki beberapa puncak ketika kami memiliki 50 instans yang secara bersamaan melayani permintaan pengguna. Dari 1 server menjadi 50 tanpa persiapan. Tentu saja, itu menyebabkan banyak stres, saraf, dan kepanikan. Tidak hanya kami tidak siap untuk menangani beban semacam itu, kami juga tidak siap dalam hal penyimpanan. Awalnya kami hanya bertahan dengan penyimpanan 50 gigabyte! Bisakah Anda bayangkan itu? Ponsel kami memiliki lima kali lebih banyak dari itu!

Solikhan ingat berada di acara bowling perusahaan, dan tiba-tiba, server EdEra kehabisan ruang dan server mati. Beruntung bagi Solikhan, salah satu rekannya membawa

laptopnya. Dia terhubung ke server kami dan selama 10 menit, semua rekannya membantu mengosongkan beberapa ruang di server—membersihkan log, file cadangan lama, dll. Tak perlu dikatakan, kami harus meningkatkan jumlah ini dan memasang alat pemantauan dan peringatan di tempat untuk kejadian semacam ini. Kami menulis semua ini bukan untuk Anda tertawakan dan memikirkan betapa bodohnya kami saat itu; kami menulis ini agar Anda menyadari pentingnya bersiap untuk skala. Perlu disebutkan bahwa bagaimana Anda mendesain arsitektur Anda adalah yang paling penting.

Di EdEra kami menggunakan sistem sumber terbuka pihak ketiga yang sulit untuk diukur. Kami harus menempatkan 50 server untuk dapat menangani beban. Beban permintaan yang sama di Gymondo (tempat Lawrence bekerja) hanya ditangani oleh dua contoh, karena aplikasi dirancang sedemikian rupa sehingga tidak macet dengan beban. Itulah mengapa penting untuk menjalankan semua jenis pengujian pada sistem Anda, termasuk pengujian beban berat. Jika Anda menggunakan platform cloud hosting, biasanya mereka menyediakan alat yang tepat agar Anda dapat menskalakan sesuai beban Anda (penskalaan otomatis). Lihat Google Kubernetes. Ini adalah sistem sumber terbuka yang dikembangkan oleh Google yang memungkinkan Anda menyimpan perangkat lunak dan mengonfigurasinya untuk penerapan otomatis, penskalaan otomatis, dan pengelolaan. Kembali ke platform pembelajaran baru kami, ketika kami mulai mengembangkannya, kami menyederhanakan beberapa aspeknya. Implementasi saat ini tidak dapat menskala secara horizontal. Ini sebagian benar; sebenarnya itu bisa berskala tetapi tidak seperti yang kita inginkan.

Masalahnya adalah mekanisme otentikasi kami menggunakan database dalam memori untuk menyimpan informasi sesi (token akses dan penyegaran). Ini berarti bahwa jika kita menyeimbangkan beban antara beberapa node aplikasi, pengguna yang menyentuh node otentikasi A dan membuat permintaan berikutnya ke node B tidak akan memenuhi permintaan tersebut, karena node B tidak tahu tentang otentikasi yang telah dilakukan. dilakukan oleh node A. Hal ini dapat diselesaikan dengan mudah dengan menyimpan informasi autentikasi di tempat yang dapat diakses oleh semua node yang melayani permintaan. Ini dapat berupa, misalnya, dalam database relasional, elasticsearch, atau jenis penyimpanan lainnya, seperti memcached.

Jika strategi kami adalah untuk terus melayani situs web dari node server juga, tidak masuk akal jika kami selalu menyajikan konten statis yang sama dan membuang waktu dan sumber daya komputasi. Dalam hal ini, sebaiknya kita meletakkan beberapa mekanisme caching di depan aplikasi untuk menyajikan konten semacam ini. Di AWS, ini akan dilakukan dengan membuat distribusi cloudfront dan menentukan perilaku cache berdasarkan jalur permintaan—misalnya, - men-cache semuanya kecuali /oauth* dan /api*. Dengan cara ini kami akan membebaskan server dari menghitung permintaan yang tidak perlu. Namun, jangan terlalu antusias dengan penskalaan.

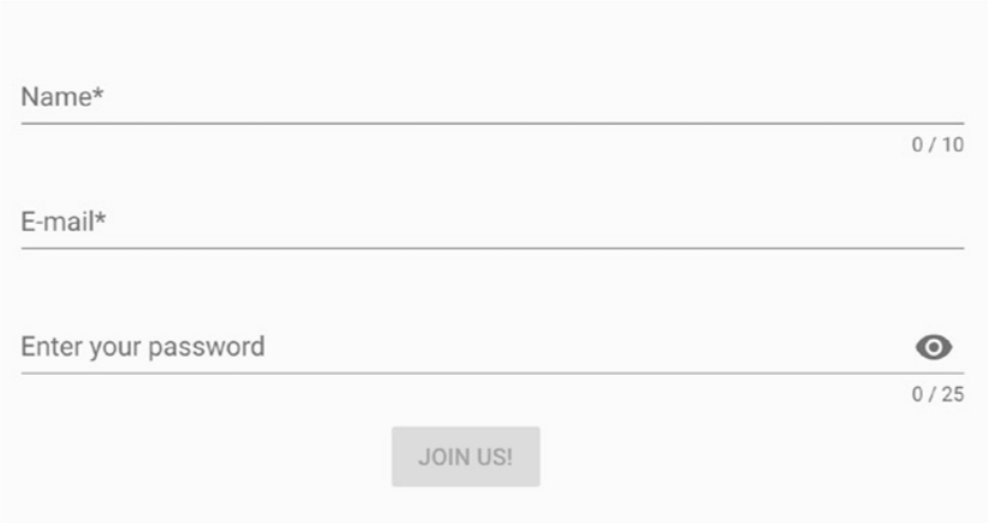
Baru-baru ini Solikhan berbincang dengan seorang pria yang bekerja sebagai konsultan CTO. Dia diundang oleh perusahaan yang terjebak dalam beberapa masalah arsitektur atau manajemen teknik. Solikhan bertanya kepadanya apa masalah teknis paling umum di perusahaan rintisan. Dan ternyata over-engineering terkadang menjadi masalah.

Insinyur menjadi terlalu bersemangat dengan alat-alat seperti Kubernetes dan containerization dan solusi cloud lainnya serta merancang arsitektur yang akan menskalakan untuk miliaran pengguna. Mempertahankan dan mengembangkan arsitektur semacam itu menjadi proses yang rumit dan rumit yang memperlambat pengembangan dan pengiriman. Dan, yang mengejutkan, perusahaan-perusahaan ini tidak pernah mencapai jumlah pengguna yang telah mereka rancang arsitekturnya yang kompleks. Artinya, bersiaplah untuk penskalaan, gunakan alat pemantauan dan analitik untuk mengawasi pengguna Anda, tetapi jangan menjadikannya satu-satunya tujuan. Tujuan perangkat lunak Anda adalah untuk memecahkan masalah pengguna akhir Anda dan membuat hidup mereka lebih mudah.

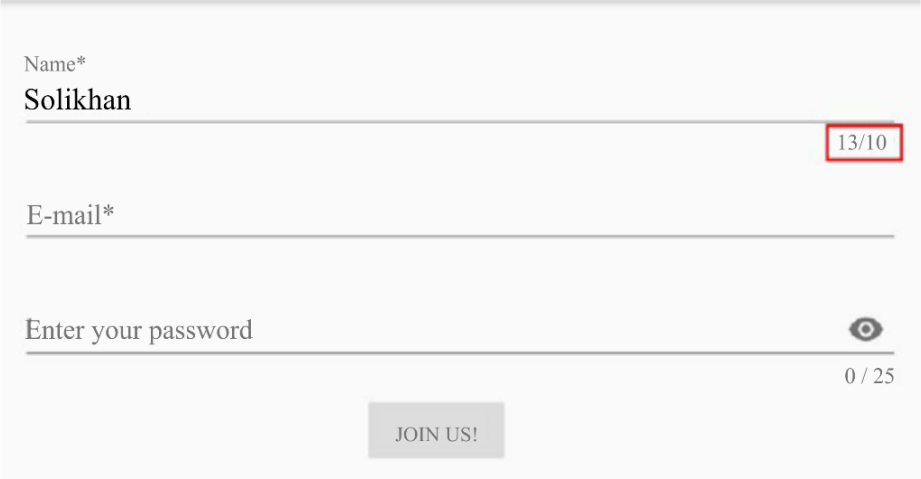
9.6 Menangani Umpan Balik

Seperti yang telah kami sebutkan, pengguna Anda akan bersedia memberikan umpan balik tentang produk Anda dan mengajukan pertanyaan. Pastikan Anda memberi mereka alat yang tepat untuk melakukannya. Letakkan formulir umpan balik di situs web Anda, berikan mereka cara untuk menghubungi Anda, dan buat alat tersebut mudah ditemukan. Yang terpenting, minta orang memantau alat kontak dan menjawab pengguna Anda. Tidak ada gunanya jika Anda memiliki email, nomor telepon, atau alat komunikasi lainnya jika Anda tidak merawatnya dan membalas pengguna Anda. Siapkan email otomatis yang akan langsung menjawab seperti, “Kami telah menerima email Anda; tim kami akan menghubungi Anda dalam beberapa hari ke depan.” Pasti akan ada permintaan yang sangat mirip. Beberapa di antaranya akan diubah menjadi fitur baru; beberapa dari mereka Anda hanya perlu menjawab. Kembangkan template dengan pertanyaan/jawaban paling umum untuk membuat hidup Anda lebih mudah.

Kumpulkan pertemuan dan pertemuan yang membahas tentang produk Anda dan fitur-fiturnya, dan atur pengujian langsung dengan audiens target potensial Anda. Bahkan jika produk Anda tampak sangat jelas dan lugas, Anda akan kagum dengan banyaknya masalah yang dapat Anda temukan dalam pertemuan semacam itu. Pertanyaan pengguna akan membawa Anda beberapa masalah UI/UX—misalnya, jika pengguna tidak dapat mendaftar karena mereka tidak dapat menemukan tombol pendaftaran, itu pasti menjadi masalah dalam perjalanan pengguna Anda. Mengatasi mereka akan meningkatkan perangkat lunak Anda. Jika Anda adalah tim kecil dan semua orang melakukan segalanya, jangan lupa untuk menambahkan masalah semacam ini ke dalam alur tugas Anda. Jika Anda memiliki tim produk dan tim layanan pelanggan, pastikan proses dalam tim diatur sedemikian rupa sehingga tim layanan pelanggan bekerja sama dengan tim produk untuk memasukkan masalah pengguna ke dalam backlog. Jika perangkat lunak Anda adalah open source, lebih baik lagi—pengguna dapat mendaftarkan masalah secara langsung di repositori GitHub Anda. Ingatlah, apa pun sarana untuk umpan balik yang Anda berikan, pastikan Anda memiliki sarana untuk mengatasinya!



Gambar 9-3. Formulir pendaftaran



Gambar 9-4. Formulir memungkinkan untuk mengetik nama lebih dari 10 karakter

9.7 Perbaikan Bug

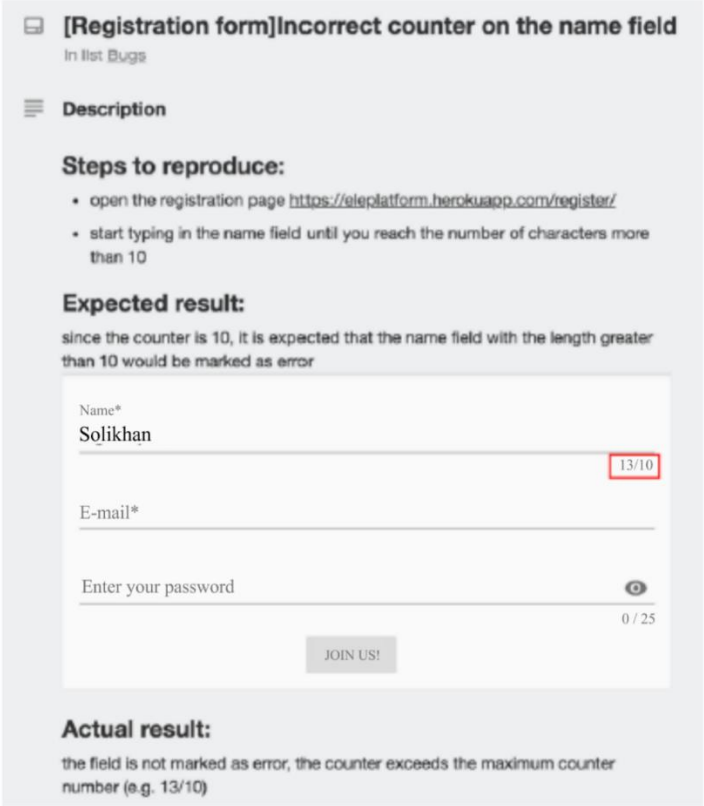
Di bagian sebelumnya, kita telah berbicara tentang menangani umpan balik pengguna. Seperti yang dapat Anda bayangkan, umpan baliknya bukan tentang meningkatkan sesuatu yang sempurna. Faktanya, tidak ada perangkat lunak yang sempurna di dunia. Perangkat lunak apa pun mengandung bug. Secara harfiah, setiap perangkat lunak di dunia memiliki bug. Jika seseorang memberi tahu Anda bahwa mereka dapat menulis perangkat lunak bebas bug, Anda dapat tertawa terbahak-bahak di depan mereka! Setiap baris kode berpotensi mengandung bug. Jika Anda ingin menjadi programmer tanpa bug, jangan membuat kode sama sekali. Memiliki bug di perangkat lunak Anda berarti Anda harus memperbaikinya. Miliki proses untuk mengatasi masalah semacam ini. Itu tidak harus menjadi proses yang rumit, tetapi setidaknya Anda harus memiliki alat pelacak bug dan cara untuk memprioritaskan bug dan mengkategorikan tingkat keparahannya. Misalnya, jika bug mencegah pengguna Anda mendaftar di platform Anda, maka itu adalah bug kritis dan harus segera diatasi. Di sisi lain, jika itu adalah kesalahan ketik kecil di carousel testimonial, maka

itu adalah bug sepele yang dapat diatasi kapan saja, dan seharusnya tidak mempengaruhi perkembangan saat ini.

Mari kita lihat platform pembelajaran kami, khususnya di halaman pendaftarannya. Gambar 9-3 menunjukkan tampilan formulir pendaftaran.

Apa artinya 0/10 pada kolom input nama dan 0/25 pada kolom input kata sandi? Tampaknya jelas bahwa itu berarti jumlah maksimum karakter yang diizinkan dalam nama dan kata sandi, masing-masing. Oke, mari kita coba menulis sesuatu di sini (Gambar 9-4).

Hm, apa? Nama saya mengandung 13 dari 10 simbol? Itu aneh. Dan ini, jelas, bug. Nomor yang digunakan untuk penghitung harus sama dengan panjang nama maksimum yang diizinkan yang kami gunakan dalam validasi formulir. Jika saya bekerja sendiri, saya hanya akan memperbaiki bug ini, tetapi jika Anda memiliki tim dan proses, seseorang harus mengajukan bug ini. Gambar 9-5 menunjukkan tampilan laporan bug untuk masalah khusus ini.



Gambar 9-5. Bug terdaftar tentang penghitung nama bidang

Mari kita lihat kode untuk memahami dari mana bug itu berasal. Berikut bagian dari formulir untuk bidang nama:

```
<v-text-field
v-model="name"
:rules="nameRules"
:counter="10"
label="Name"
required
autocomplete="name"
```

```
></v-text-field>
And here is the validation code:
data: () => ({
valid: false,
name: “,
nameRules: [
v => !!v || 'Name is required',
v => v.length <= 32 || 'Name must be less than 32 characters'
],
}),
```

Jelas bahwa kita hanya perlu mengatur penghitung ke 32l, tetapi sebelum melakukan apa pun dalam kode, pertama-tama kita harus menulis tes! Ingat aturan bagus ini: jika Anda memiliki bug, pertama tulis tes untuk menutupi bug ini, pastikan tes ini gagal, perbaiki bug, dan pastikan tes ini lulus. Saat kami bekerja sama di Feedzai, bahkan ada meme tentang Lawrence. Terlepas dari seberapa mudah untuk memecahkan bug itu dia harus menutupinya dengan tes. Terkadang dia akan memperbaiki bug sebelum makan siang dan mengumumkannya kepada tim. Kami akan berdiri dan mulai bergerak ke pintu untuk pergi makan siang. Tapi Lawrence akan tetap duduk di mejanya dengan konsentrasi penuh. Dan kemudian semua orang akan berkata, "Ah tentu, dia menjalankan tes untuk memeriksa apakah bug tersebut benar-benar diperbaiki!" Kembali ke bug dengan penghitung karakter. Pengujian kami harus memeriksa bahwa setelah halaman pendaftaran terbuka, elemen bidang nama berisi teks "0/32". Jika kami memperkenalkan sesuatu, penghitung harus memperbarui. Kami akan menggunakan Nightwatch (http:// nightwatchjs.org/) untuk pengujian. Pengujian kami akan terlihat sebagai berikut:

```
module.exports = {
'Test Registration page': function (browser) {
browser
.url('http://localhost:3000/register')
.waitForElementVisible('#app', 5000)
.assert.containsText('.input-group__counter', '0 / 32')
.setValue('input[autocomplete="name"]', '1234567890123')
.assert.containsText('.input-group__counter', '13 / 32')
.end()
}
}
```

Jika saya menjalankan tes ini sekarang akan gagal: Pengujian jika elemen <.input-group__counter> berisi teks: "0/32". - diharapkan "0 / 32" tetapi mendapat: "0 / 10". Sekarang mari kita perbaiki bug. Seperti yang telah kami sebutkan, kami harus mengubah nomor penghitung menjadi 32. Atau, lebih baik lagi, untuk menghindari jumlah "angka ajaib" melalui kode, mari tambahkan nilai konstan untuk panjang nama maksimum dan gunakan itu.

Angka ajaib adalah angka-angka yang muncul di tempat kode yang berbeda dan maknanya sulit dipahami. hindari menggunakan angka ajaib; buat konstanta dengan nama yang bermakna sebagai gantinya.

Jadi, setelah diperbaiki kodenya akan terlihat seperti ini:

```
<script>
import { mapActions, mapGetters } from 'vuex'
const MAX_NAME_LENGTH = 32
export default {
middleware: 'notauthenticated',
data: () => ({
MAX_NAME_LENGTH,
valid: false,
name: “,
nameRules: [
v => !!v || 'Name is required',
v  => v.length <= MAX_NAME_LENGTH || `Name must be less than
${MAX_NAME_LENGTH} characters`
]
})
}
</script>
```

Markup untuk bidang nama akan terlihat sebagai berikut:

```
<v-text-field
<...>
:counter="MAX_NAME_LENGTH"
<...>
></v-text-field>
```

Dan ya! Tes berlalu! Kami dapat menandai masalah sebagai terselesaikan. Kode untuk bagian ini dapat ditemukan di folder BugFixing di dua subfolder: sebelum dan sesudah. Hanya kode frontend yang diubah. Lihat file package.json untuk dependensi tambahan, pages/register.vue untuk perbaikan sebenarnya, dan folder tes/e2e untuk tes tambahan. File README.md berisi langkah-langkah untuk menjalankan pengujian ujung ke ujung. Jadi, ingatlah: Anda tidak dapat menulis kode bebas bug, tetapi Anda dapat—dan Anda harus—menulis kode yang bersih, dapat dipelihara, dan telah teruji dengan baik. Maka bug apa pun yang muncul di jalan Anda akan dibersihkan dalam hitungan detik!

9.8 Pemfaktoran Ulang, Penulisan Ulang, dan Utang Teknis

Tidak peduli seberapa bersih basis kode Anda dan seberapa baik cakupan pengujiannya, pada titik tertentu Anda akan melihat kode dan Anda akan merasa perlu untuk mengaturnya kembali dengan cara yang berbeda. Beberapa kelas mungkin tidak masuk akal lagi, beberapa file dapat dipecah menjadi modul yang berbeda, beberapa fungsi dapat dibagi menjadi subfungsi yang lebih kecil, beberapa pengujian dapat ditingkatkan, dan beberapa dokumentasi dapat ditambahkan. Ketika Anda merasakan dorongan untuk meningkatkan

kode Anda, untuk mengatur ulang dan membuatnya lebih baik, itu berarti sudah waktunya untuk refactoring!

Memfaktorkan ulang kode berarti kode direstrukturisasi, tetapi perilaku dan fungsinya tetap sama. Semakin banyak kode Anda dicakup oleh pengujian unit, semakin Anda dapat yakin bahwa tidak ada refactoring yang merusak fungsinya.

Misalnya, dalam kasus frontend kami, kami memiliki folder "komponen", yang saat ini berisi tiga komponen:

- CourseItem.vue
- CourseModule.vue
- UserCourseItem.vue

Meskipun hanya tiga komponen, cukup mudah untuk memahami tujuan masing-masing komponen. Di sisi lain, dengan pertumbuhan basis kode, jumlah komponen juga akan bertambah, dan ada kemungkinan besar bahwa kita harus menyusun ulang folder ini menjadi subfolder yang berarti dan bahkan mungkin berpikir untuk mengganti nama beberapa file. Mengenai arsitektur secara umum, seperti yang dikatakan sebelumnya, kami mendistribusikan situs web statis juga dari node rest-api.

Dalam arsitektur modern, ini adalah larangan yang kuat, terutama karena saat ini semuanya dihosting di penyedia cloud yang memungkinkan Anda membangun banyak arsitektur menggunakan layanan mereka. Satu hal yang perlu diperbaiki tentang arsitektur kita adalah memisahkan sepenuhnya backend dari frontend. Frontend dapat dilayani oleh node server web khusus, atau—bahkan lebih baik—jika cukup sederhana dan tidak diperlukan aturan gila, itu dapat dilayani oleh solusi S3 + Cloudfront di Amazon AWS.

AWS memungkinkan Anda untuk meng-host situs web di penyimpanan cloud (S3) mereka dan menempatkan distribusi Cloudfront di depannya dan menyimpan semua file dalam cache untuk menghemat uang dan membuatnya lebih cepat. Lawrence adalah penggemar solusi ini, tetapi ini hanya berfungsi jika Anda hanya menyajikan file dengan sederhana dan sederhana. Dimungkinkan juga untuk mengaktifkan beberapa aturan penulisan ulang, tetapi itu tidak akan pernah sebagus server web tertentu lainnya di pasar. Contoh kami adalah pasangan yang cocok untuk dilayani oleh solusi ini, setidaknya untuk saat ini! Ini tidak berarti bahwa dalam waktu dekat, produk berkembang sedemikian rupa sehingga ini tidak berlaku lagi. Jika atau ketika saatnya tiba, migrasi akan lancar, jadi kita tidak perlu terlalu khawatir tentang itu.

Terkadang kita tidak pernah berhenti untuk merenungkan kualitas dan struktur kode kita. Itu terjadi ketika tim penjualan secara agresif menjual fitur yang tidak ada dan kemudian tim pengembangan harus mengimplementasikannya dengan tenggat waktu yang cukup keras. Itu terjadi ketika tim produk terus mendorong fitur-fitur baru ke backlog, ketika manajemen ingin melihat hal-hal baru di jam-demo mingguan dan tim pengembangan merasa berkewajiban untuk menambahkan hal-hal itu apa pun yang terjadi. Itu mungkin terjadi karena banyak alasan. Ke mana arahnya? Ini mengarah pada fakta bahwa 1 tahun kemudian, kami menyadari bahwa basis kode kami sangat berantakan. Kami menyadari bahwa kode kami penuh dengan pernyataan TODO.

Sebenarnya, ketika kami meninggalkan pernyataan TODO dalam kode, kami benar-benar percaya bahwa kami akan segera melakukannya! dan kami benar-benar percaya bahwa pada saat kami akan melakukannya, kami akan benar-benar mengingat konteks dan alasan TODO ini. Percayalah, tidak pernah terjadi!

Kami merasa takut dengan kekacauan dalam struktur kode, dengan jumlah modul yang sangat banyak, oleh perbedaan dalam gaya kode melalui basis kode, dan oleh fakta sederhana bahwa kami melihat kode dan kami tidak menyukainya. Dalam hal ini, kami pikir akan lebih mudah untuk menulis ulang semuanya dari awal. Kami secara naif percaya bahwa jika kami mulai menulis ulang seluruh basis kode dari nol, itu akan menjadi bersih, bagus, dan sempurna. Kasus lain ketika kami merasa seperti kami akan menulis ulang seluruh basis kode dengan cara yang bagus dan sempurna adalah ketika kami menemukan beberapa bahasa pemrograman baru atau kerangka kerja baru. Kami merasakan dorongan untuk mencoba hal-hal baru yang hebat ini dan kami benar-benar yakin bahwa itu lebih berlaku untuk kasus kami daripada apa pun. Kami ingin menulis ulang semuanya (Gambar 9-6)!

Gambar 9-6. Kami benar-benar percaya bahwa jika kami menulis ulang semuanya dari awal, itu akan menjadi sempurna

Tolong jangan jatuh dalam ilusi bahwa itu akan terjadi dengan kode Anda. Anda mungkin pernah mendengar tentang beberapa kasus sukses penulisan ulang kode, tetapi biasanya itu tidak akan menghasilkan apa yang Anda harapkan. Netscape melakukannya, dan mereka membutuhkan waktu 3 tahun, dan mereka harus melewati satu versi, langsung dari versi 4 ke 6. Contoh ini diberikan dalam artikel oleh Joel Spolsky, pendiri stackoverflow (https://bit.ly/ 2iF64Qm). Kode yang ditulis ulang juga akan mengalami masalah; setelah beberapa waktu Anda juga akan benci membacanya, dan pada akhirnya, Anda akan berakhir dengan dua basis kode yang tidak efisien.

Namun, jika Anda memutuskan bahwa tidak ada pilihan lain selain menulis ulang penuh, buatlah rencana yang bagus untuk itu, perkirakan dalam waktu berapa lama waktu yang dibutuhkan. Kembangkan strategi yang memungkinkan Anda untuk mempertahankan kode yang ada sambil perlahan tapi pasti membuat yang baru. Bagilah kode menjadi potongan-potongan otonom yang strategis; terus ganti setelah penulisan ulang sehingga Anda dengan lancar mengganti seluruh basis kode. Pastikan kode baru bersih, teruji, dan dapat dibaca. Namun demikian, perlu diingat refactoring kecil selalu lebih baik daripada

penulisan ulang besar. Jadi, pastikan Anda selalu memiliki ruang untuk mengatasi utang teknis.

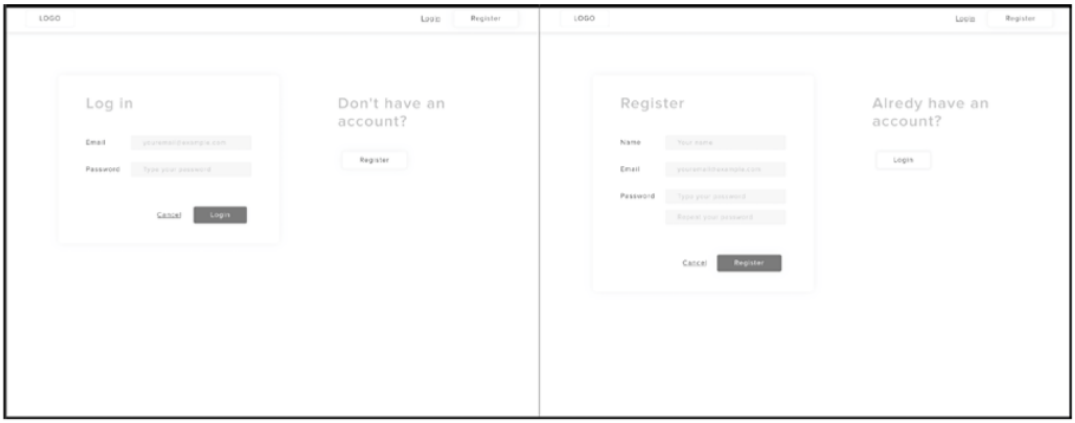
Utang teknis terdiri dari masalah dengan kode Anda yang mungkin menyebabkan refactoring. cakupan pengujian yang tidak efisien, keputusan arsitektur yang buruk, kurangnya struktur, gaya kode yang tidak koheren—semua ini dapat dianggap sebagai hutang teknis. tidak mungkin untuk tidak memiliki hutang teknis, tetapi dimungkinkan untuk mengatur proses sedemikian rupa sehingga tidak terakumulasi.

Bicaralah dengan tim Anda, pastikan semua orang berada di halaman yang sama mengenai bahaya menumpuk hutang teknis. Pastikan bahwa siapa pun yang bertanggung jawab atas backlog memasukkan tugas utang teknis ke dalamnya. Pastikan bahwa setiap sprint (atau unit kerja lain yang Anda gunakan) mengatasi beberapa masalah ini. Pastikan kode Anda membuat Anda merasa bangga dan bahagia.

9.9 Mendesain ulang dan Merebranding

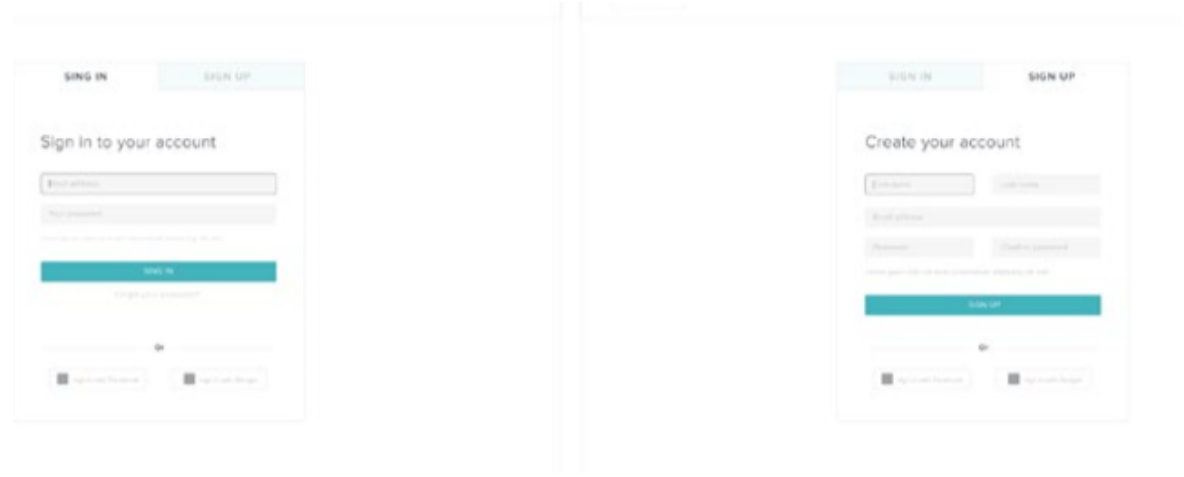
Kami telah menyebutkan dalam bab ini bahwa sementara produk Anda berhasil digunakan oleh pengguna Anda yang bahagia, tren di dunia desain berubah dan hal-hal baru menjadi populer. Tombol datar datang untuk menggantikan tombol tiga dimensi, gradien tiba-tiba menjadi sesuatu, pola abu-abu bersih secara alami menggantikan ornamen yang berkedip dan berwarna-warni, dll. Belum lagi invasi perangkat seluler dan kebutuhan untuk mendesain aplikasi dengan pikiran mobile-first di tempat. Dengan demikian, tim desain Anda akan terus-menerus mengusulkan solusi baru dan tim pengembangan Anda harus menerapkannya.

Penting untuk menjaga keseimbangan yang baik saat mendesain ulang. Jangan mengejar semua tren yang muncul—beberapa di antaranya bersifat sementara dan mungkin akan mati besok. Mengawasi mereka yang mungkin bermanfaat bagi produk Anda, mengingat kebutuhan pengguna Anda. Kadang-kadang desainer mengusulkan mockup yang didesain ulang karena mereka membuat penelitian pengguna dari desain saat ini dan menemukan bahwa itu perlu ditingkatkan dalam beberapa cara. Misalnya, ketika kami sedang sibuk menulis buku ini, desainer kami yang luar biasa, Alexandra Syrik, datang dengan mendesain ulang halaman login dan registrasi. Sekedar mengingatkan, Gambar 9-7 menunjukkan keadaannya.



Gambar 9-7. Desain halaman login dan registrasi sebelumnya

Dan Gambar 9-8 menunjukkan tampilannya sekarang.



Gambar 9-8. Versi halaman masuk dan pendaftaran yang didesain ulang

Alexandra juga telah mendesain halaman arahan dan sepenuhnya mendesain ulang halaman dasbor kursus. Kami sangat menyukai desain baru, dan ini berarti kami akan memiliki beberapa pekerjaan yang harus dilakukan. Itu selalu baik untuk sesekali mendesain ulang produk Anda atau bagian-bagiannya. Terkadang redesign berubah menjadi full rebranding, di mana identitas lengkapnya diubah (logo, font, skema warna, dll). Jangan terlalu sering mengubah citra produk Anda, jika tidak, pengguna Anda akan berhenti mengenalinya. Namun, terkadang terasa enak dan menyegarkan, terutama ketika bisnis Anda sedikit atau sama sekali berubah arah. Branding berkaitan dengan visi, misi, dan nilai-nilai Anda. Pastikan mereka semua bergabung.

9.10 Kesimpulan

Dalam bab ini kita telah membahas banyak hal yang terjadi dengan produk perangkat lunak selama masa pakainya. Kami membahas banyak proses berbeda mulai dari umpan balik pengguna, melalui pemfaktoran ulang kode dan penulisan ulang dan berakhir dengan desain ulang. Anda sekarang tahu betapa pentingnya menjaga produk Anda dalam keadaan sehat, betapa pentingnya untuk terus meningkatkannya, dan betapa pentingnya untuk mengawasinya agar tidak mengecewakan pengguna Anda. Tampaknya Anda tahu apa yang harus dilakukan dengan perangkat lunak Anda berada di luar sana di dunia liar. Kamu siap! Pada prinsipnya, perjalanan kita akan segera berakhir. Pada bab berikutnya, kami akan memberikan beberapa tip dan mendiskusikan beberapa trik dari pengalaman kami sendiri sebagai insinyur dan manajer perangkat lunak selama lebih dari 10 tahun.

BAB 10
MENGAKHIRI DENGAN BEBERAPA TIPS DAN TRIK

Dalam bab sebelumnya kita telah membahas segala sesuatu yang terjadi dengan perangkat lunak setelah rilis pertama. Kami menyimpulkan bahwa rilis pertama hanyalah awal dari perjalanan besar yang terdiri dari banyak langkah terkait dengan pemeliharaan dan peningkatan produk Anda. Seperti perjalanan lainnya, perjalanan ini dapat mengubah jalannya. Seperti perjalanan lainnya, mungkin berakhir di sudut tanpa jalan keluar yang jelas dan Anda harus mundur selangkah. Itu juga dapat membawa Anda ke terowongan di mana Anda akan dipaksa untuk bergerak maju sampai Anda mulai melihat cahaya. Namun, ini adalah perjalanan yang mengasyikkan; Anda akan menemukan tempat-tempat wisata yang indah dan kecelakaan yang menakutkan, Anda pasti akan bertemu orang-orang baru dan menemukan diri Anda dalam situasi yang aneh dan petualangan yang menakjubkan. Kami telah melalui perjalanan yang berbeda, bekerja dengan perusahaan yang berbeda, orang yang berbeda, produk yang berbeda, dan proyek. Kami percaya kami dapat berbagi beberapa tips dan trik yang berguna dengan Anda.

10.1 Kiat Pengembangan

Dalam buku ini kita telah membahas pengembangan frontend dan backend. Pengembangan terdiri dari keputusan arsitektur, pengkodean, peninjauan kode, pengujian, mendorong kode Anda ke produksi, bekerja dengan tim, dan terus meningkatkan basis kode sehingga perangkat lunak Anda mudah dipelihara dan bersih.

10.2 Memilih Bahasa atau Kerangka Pemrograman

Ketika Anda memulai proses pengembangan, pertanyaan tentang bahasa apa yang akan digunakan pasti akan muncul. Pengembang mungkin akan mulai berkelahi dan menyajikan argumen untuk atau melawan Java atau Go, Python atau Ruby, atau bahkan PHP. Juga akan ada pertanyaan mengenai penyimpanan dan mesin telusur—SQL versus NoSQL, basis data relasional versus penyimpanan dokumen, teknologi hipster versus solusi klasik… Standar apa yang harus Anda gunakan? Teman lama REST atau GraphQL modern? Kerangka kerja frontend apa yang harus Anda pilih? Bereaksi dan Redux, Vue.js? Atau mungkin, Elm? Atau mungkin, tidak ada kerangka sama sekali? Jangan biarkan keputusan ini berkembang menjadi perang suci. Pada akhirnya, bukan teknologi yang mendorong produk—tetapi orang-orang yang membuatnya luar biasa. Saat memilih teknologi yang tepat untuk produk Anda, ingatlah, Anda tidak boleh menjadi budak dari teknologi itu, sebaliknya—teknologi harus melayani kebutuhan Anda dan kebutuhan produk Anda.

Buat tabel perbandingan—tulis persyaratan teknis perangkat lunak Anda dan gambarkan teknologi yang sesuai dengannya. Apakah produk Anda harus melakukan kueri gabungan kompleks yang harus memberikan hasil dengan cepat? Pertimbangkan sesuatu seperti elasticsearch. Apakah Anda memerlukan sinkronisasi waktu-nyata antara

penyimpanan data dan lapisan presentasi Anda? Silahkan lihat di database realtime. Haruskah itu menyelesaikan tugas-tugas kompleks dan memiliki kinerja tinggi? Jalankan beberapa benchmark pada bahasa pemrograman yang berbeda. Jika ada diskusi tanpa akhir di antara anggota tim mengenai teknologi ini atau itu, mintalah masing-masing untuk menyiapkan pitch deck dan mempresentasikannya kepada tim dan membuat polling sesudahnya.

Tetapkan tenggat waktu yang kuat untuk keputusan itu. Jam tic tac terus berdetak, dan Anda tidak dapat menghabiskan seluruh waktu Anda untuk diskusi semacam ini atau pesaing Anda mungkin akan menyusul Anda. Pada akhirnya yang terpenting adalah semua orang merasa nyaman dengan teknologi yang dipilih. Jika ada kekurangan pengetahuan, sediakan sarana bagi orang untuk belajar. Bangun perangkat lunak Anda dengan cara modular yang bagus sehingga jika teknologi terbukti menjadi pilihan yang buruk, itu dapat dengan mudah diganti.

10.3 Pedoman Gaya Kode

Selain perang suci untuk memilih bahasa atau kerangka kerja pemrograman, ada diskusi lain di antara para pengembang — IDE apa yang lebih baik, spasi di atas tab, tanda kutip ganda versus tanda kutip tunggal. Tahukah Anda bahwa Anda dapat mengalihkan perhatian teknisi dari hampir semua percakapan dengan menanyakan apakah mereka lebih suka vim atau emacs? Suatu ketika Solikhan sangat bosan dengan diskusi "ski VS snowboard" tanpa akhir sehingga dia mengalihkan perhatian semua orang hanya dengan mengatakan bahwa itu seperti diskusi vim versus emacs. Tentu saja, kemudian dia bosan dengan diskusi vim versus emacs, tetapi Anda mengerti maksudnya. Pengembang JavaScript mungkin akan mendiskusikan apakah mereka harus menempatkan titik koma (;) setelah pernyataan—tren terbaru bergerak menuju penyederhanaan segalanya, jadi, tidak ada titik koma. Secara keseluruhan orang mungkin akan berdebat mengenai lebar garis maksimum. Ini mungkin tampak tidak penting sama sekali, dan setiap orang mungkin harus membuat kode dengan caranya sendiri, tetapi ada beberapa masalah utama dengan ini:

- Kode dengan gaya campuran (mis., tanda kutip ganda dan tunggal dalam string dalam file yang sama) terlihat tidak koheren dan jelek.
- Segera setelah Anda mulai dengan tinjauan kode, Anda akan berakhir di neraka "permintaan perubahan". Bayangkan seseorang yang telah coding menggunakan 2-spasi, dan kemudian orang lain membuat perubahan dalam file yang sama dan editor mereka menerapkan aturan 4-spasi. Tinjauan kode akan penuh dengan perubahan ini, dan perubahan penting akan hilang dalam kekacauan ini.

Penting bahwa orang memiliki kebebasan untuk melakukan sesuatu dengan cara mereka sendiri, tetapi juga penting bahwa perangkat lunak Anda tidak menderita karenanya. Oleh karena itu, kami menyarankan Anda untuk membuat pedoman gaya kode sejak awal. Panduan gaya ini harus mendefinisikan semua aturan penting, mengenai lebar garis, konvensi penamaan, titik dua, titik koma, tanda kutip, aturan kompleksitas (misalnya, tidak lebih dari tiga tingkat bersarang dalam siklus). Setiap orang harus membaca dan setuju dan kemudian hanya mematuhi aturan yang ditetapkan. Saat ini ada banyak alat yang

memformat kode Anda sesuai dengan aturan yang diberikan, jadi Anda bisa memindahkannya ke IDE favorit Anda dan membiarkannya menangani aturan untuk Anda.

10.4 Ulasan Kode dan Pemrograman Penyandingan

Kami telah menyebutkan tinjauan kode di bagian sebelumnya, dan kami menyebutkan bagaimana tinjauan kode mungkin menderita ketika orang menggunakan gaya pengkodean yang berbeda. Sebenarnya, ini bekerja dua arah—proses peninjauan kode dapat membantu mencapai keseragaman dan homogenitas dalam basis kode. Ini terjadi secara alami karena orang mulai membaca kode satu sama lain dengan cermat, menyarankan peningkatan, dan mempelajari pola baru satu sama lain. Dengan demikian, seiring waktu kode mulai terlihat seperti satu bagian yang solid dan indah. Kami merasakan ini ketika kami bekerja di Feedzai ketika kami memperkenalkan ulasan kode. Awalnya cukup membosankan, dan kami banyak berdiskusi. Beberapa ulasan kode memiliki hingga 20 iterasi!

Namun, dalam beberapa bulan kami tidak dapat mengenali kode kami. Itu adalah sebuah karya seni—indah, mudah dibaca, terdokumentasi dengan baik, teruji, dan koheren. Penting untuk menetapkan beberapa aturan agar kode siap untuk tinjauan kode—yang disebut definisi siap. Misalnya, kode yang baru diperkenalkan harus dicakup oleh pengujian unit, kode harus mengikuti panduan gaya, tidak boleh ada lebih dari N file yang diubah per tinjauan kode, dll. Perhatikan bahwa menetapkan jumlah maksimum baris dan/atau file diubah per tinjauan kode sangat penting. Seperti yang dapat Anda bayangkan, meninjau sebagian kecil kode logis jauh lebih mudah daripada meninjau sebagian besar file yang tidak terkait yang telah diubah.

Beberapa pengembang antusias dengan perubahan dan mulai memperkenalkan refactor kode kecil saat mengerjakan fitur kecil. Kemudian tinjauan kode mereka mungkin mencakup lebih dari 40 file yang diubah atau beberapa ribu baris kode yang diubah. Ini membuat para pengulas frustrasi, dan itu sebenarnya bisa dianggap tidak sopan. Untuk menghindari frustrasi semacam ini, jelaskan dalam definisi Anda tentang siap untuk tinjauan kode bahwa tidak boleh lebih dari, katakanlah, 1000 baris kode yang diubah. Percayalah, semuanya mungkin untuk dipecah menjadi potongan logis yang lebih kecil dari pekerjaan. Teknik lain yang banyak digunakan di kalangan pengembang dan yang membantu mencapai homogenitas dalam basis kode adalah pemrograman berpasangan. Pemrograman berpasangan persis seperti yang Anda pikirkan—dua orang duduk bersama dan membuat kode. Sebenarnya, hanya satu kode pengembang pada satu waktu dan yang lain hanya melihat dan berkomentar. Sepasang pengembang dapat dan juga harus bertukar pikiran dari waktu ke waktu dan mendiskusikan beberapa solusi. Setelah beberapa waktu, peran berubah—"pengamat" pindah ke pengkodean dan pembuat kode duduk di samping mereka dan membantu mereka berpikir. Teknik ini membantu untuk mencapai keseragaman dalam kode untuk alasan yang jelas; orang belajar dari satu sama lain dan saling membantu untuk meningkatkan gaya pengkodean mereka.

Ada keuntungan lain: tingkat perhatian dan konsentrasi meningkat banyak. Pikirkan aktivitas apa pun yang Anda lakukan sendiri atau ketika seseorang memperhatikan Anda. Ketika Anda sendirian, Anda mungkin kadang-kadang membiarkan diri Anda ceroboh, Anda

mungkin membiarkan diri Anda menunda-nunda, dan Anda dengan mudah memaafkan diri sendiri untuk itu. Ketika seseorang memperhatikan Anda, Anda ingin menunjukkan betapa hebatnya Anda, seberapa dalam konsentrasi Anda pada pekerjaan, dan hasil luar biasa yang bisa keluar dari kerja keras Anda. Anda pasti akan mengabaikan email baru atau pemberitahuan jejaring sosial.

Sebenarnya, produktivitas yang dicapai oleh pemrograman berpasangan, meskipun satu orang tampaknya menganggur, terkadang mengungguli produktivitas dua pemrogram yang mengkodekan pada saat yang sama.

10.5 Tip Jaminan Kualitas

Pada bagian sebelumnya kita telah membahas beberapa teknik yang dapat digunakan untuk meningkatkan keseragaman kode. Seperti yang Anda bayangkan, semakin homogen, terdokumentasi dengan baik, dan tercakup dengan pengujian kode, semakin tinggi kualitasnya, dan itulah mengapa kami dapat mengatakan bahwa tinjauan kode dan pemrograman pasangan adalah alat yang membantu meningkatkan kualitas produk kami. Selain itu, ada teknik dan proses QA lain yang telah kami persembahkan untuk seluruh bab buku ini. Bahkan ada orang yang berdedikasi melakukannya.

Ketika Anda mulai mengembangkan produk Anda, Anda mungkin tidak memiliki sumber daya yang cukup untuk mengalokasikan orang-orang khusus untuk mengamankan kualitas produk, tetapi jangan meremehkan pekerjaan ini. Suatu ketika Solikhan mencoba meyakinkan seseorang untuk mempekerjakan seorang insinyur QA dan orang ini menjawabnya sebagai berikut, “Jangan khawatir tentang kualitas untuk saat ini. Mari kita khawatir tentang memiliki lebih banyak pengguna. Ketika kami memiliki banyak pengguna, kami dapat khawatir tentang kualitas.” Kamu tahu apa? Ini adalah hal terburuk yang pernah Anda pikirkan atau katakan kepada seseorang. Dengan premis seperti ini, produk Anda tidak akan pernah menang! Dan pastikan, basis pengguna Anda tidak akan pernah bertambah karena tidak ada yang menyukai produk yang tidak stabil dan buggy.

Kualitas produk Anda mendefinisikan Anda, nilai-nilai perusahaan Anda, cara Anda berpikir. Tentu saja, kesalahan terjadi dan bahkan pemain besar pun punya masalah. Terkadang layanan AWS down, menurunkan setengah dari internet. Terkadang server Telegram berada di bawah beban besar dan penggunanya tidak dapat berkomunikasi. Tidak ada yang kebal! Tetapi beberapa akan terus mengalami masalah ini dan masih belum belajar darinya, kehilangan kepercayaan pelanggan mereka sementara yang lain akan memiliki proses yang terdefinisi dengan baik untuk membantu mencegah kesalahan yang sama terjadi dua kali. Itulah sebabnya kami menyarankan Anda, bahkan jika Anda tidak memiliki orang yang berdedikasi, sejak awal menetapkan beberapa proses dan ambang kendali kualitas. Misalnya, cakupan kode pengujian unit harus tidak kurang dari 80%. Memiliki tes otomatis di tempat.

Siapkan daftar periksa fitur paling penting untuk memeriksa setiap kali fitur baru didorong ke produksi, jadi Anda menjamin bahwa barang baru tidak merusak yang sudah ada. Buat daftar browser dan/atau perangkat yang HARUS menjalankan perangkat lunak Anda tanpa masalah. Periksa mereka setiap kali barang baru dikirim. Bahkan jika Anda

melakukannya secara manual, jika Anda memiliki daftar periksa kasus uji regresi dan browser atau perangkat yang terdefinisi dengan baik, proses pengujian yang didistribusikan di antara semua orang dalam tim tidak akan memakan waktu terlalu lama.

Berinvestasi dalam otomatisasi. Anda harus memiliki setidaknya jalur bahagia Anda diuji menggunakan mekanisme otomatisasi.

Sekedar pengingat, jalur bahagia adalah serangkaian aktivitas pengguna yang ditetapkan yang mengarahkan mereka dari awal hingga akhir yang bermakna. Misalnya, dalam kasus situs web e-niaga, jalur bahagia adalah dari pendaftaran hingga pembelian yang sebenarnya dilakukan.

Tip kecil tapi penting untuk otomatisasi: jangan terlalu bersemangat dengannya. Tentu saja, ada baiknya bila antarmuka Anda dan segala sesuatu yang mungkin dilakukan dengannya diotomatisasi, tetapi terkadang pengujiannya begitu rumit, tidak dapat diprediksi, dan sulit diterapkan sehingga ternyata lebih mudah dan sebenarnya lebih cepat untuk memeriksanya secara manual sebelum setiap melepaskan. Mengotomatiskan hanya apa yang mudah diprediksi. Jangan mencoba mengotomatiskan hal-hal seperti, misalnya, gambar acak yang muncul di korsel. Periksa secara manual. Selalu berusaha untuk menjaga keseimbangan yang baik antara otomatisasi dan pemeriksaan manual.

Tip lain tentang otomatisasi: pastikan pengembang frontend Anda dan siapa pun yang melakukan otomatisasi benar-benar tersinkronisasi. Saat kami memperkenalkan pengujian otomatis di Feedzai, awalnya agak membuat frustrasi karena pengembang frontend akan selalu mengubah kelas elemen CSS, sehingga merusak otomatisasi yang mengandalkan kelas tersebut. Setelah sedikit berjuang dan menyalahkan satu sama lain, kami mencapai kesepakatan bahwa kami akan menggunakan kelas awalan untuk otomatisasi dan kelas lain untuk gaya yang sebenarnya. Keputusan ini sangat membantu kami untuk menghilangkan ketergantungan antara pengujian dan pengembangan otomatis serta mengurangi ketegangan antara pengembang dan penguji. Dan yang tak kalah pentingnya, segera setelah Anda memiliki sumber daya untuk mengalokasikan orang-orang yang berdedikasi untuk pengujian, lakukanlah! Tidak hanya itu akan meningkatkan kualitas produk, tetapi juga akan membantu dalam mencapai keseimbangan keseluruhan antara proses antara pemangku kepentingan yang berbeda.

10.6 Kiat DevOps

Di bagian sebelumnya kita telah membahas pengujian otomatis. Anda mungkin telah mengetahui bahwa untuk memiliki otomatisasi yang bagus, Anda memerlukan infrastruktur bagus yang memungkinkan memiliki banyak lingkungan dan proses otomatis integrasi dan pengiriman berkelanjutan, sehingga memungkinkan proses menjalankan pengujian yang lancar dan transparan. Nah, sekarang kita berbicara tentang proses DevOps yang harus ada jika kita ingin mencapai harmoni integrasi berkelanjutan, pengujian, penerapan, dan pengiriman.

Sekali lagi, seperti dalam hal jaminan kualitas, jangan meremehkan pentingnya infrastruktur Anda. Anda mungkin berpikir bahwa pada awalnya Anda dapat membuat file secara lokal dan menyalinnya secara manual ke server produksi dan suatu hari nanti Anda

akan meningkatkan proses ini. Tidak bekerja seperti itu. Yah, itu mungkin berhasil, karena cepat atau lambat Anda akan takut dengan ketergantungan dan jumlah pekerjaan manual yang perlu Anda lakukan untuk melakukan penerapan. Semakin banyak basis kode dan jumlah dependensi Anda bertambah, semakin sulit untuk membangun saluran yang tepat dan lancar untuk mendorong semua itu ke dalam pengujian atau produksi. Jauh lebih mudah untuk mendefinisikan pipeline CI/CD Anda dari awal, membuatnya sangat sederhana dan kecil, dan kemudian dengan waktu menyesuaikannya sesuai dengan basis kode dan pertumbuhan dependensi Anda.

Suatu ketika Solikhan bekerja di sebuah perusahaan pengukuran iklan online. Skrip mereka berjalan di halaman web dan mengukur keberhasilan kampanye iklan klien mereka. Untuk membuatnya berfungsi, semua kampanye memiliki ID, dan ID ini harus cocok dengan ID yang ditentukan dalam skrip. Ada proses semi-otomatis yang akan membuat skrip bersama dengan file konfigurasi per kampanye untuk menyematkan ID ke dalam skrip. Setelah Solikhan memperbaiki beberapa bug, dan untuk membuat pengujiannya lebih mudah, dia membuat hardcode ID ini dalam skrip, sehingga mengesampingkan kemungkinan konfigurasi apa pun. Kemudian dia secara manual mendorong skrip ini ke server pementasan. Setidaknya dia mengira itu adalah server pementasan. Faktanya, dia melakukan kesalahan besar dan mendorongnya ke server produksi. Selama setengah hari mereka menjalankan skrip dengan ID hardcode untuk setiap klien mereka.

Bayangkan apa artinya bagi industri periklanan, di mana setiap menit mungkin menghabiskan biaya jutaan, untuk tidak mengukur dampak iklan Anda selama setengah hari. Setelah epik gagal, Solikhan menyiapkan proses CI yang mulai menjalankan semua pemeriksaan yang diperlukan untuk proyek dan ID kampanye yang menegakkan bahwa tidak ada yang akan mendorong ke produksi secara manual. Kami tidak ingin Anda membayar untuk pembelajaran semacam ini. Seseorang telah belajar, dan Anda bisa menggunakan pengetahuan ini. Siapkan proses otomatis Anda dan investasikan pada orang yang tepat yang senang bekerja dengan infrastruktur, server, dan skrip sejak awal!

10.7 Bagaimana dengan Ide Saya?

Anda mungkin mulai bertanya, “Mengapa saya membaca tips tentang pengembangan dan jaminan kualitas jika saya hampir tidak tahu apa yang akan saya lakukan? Bagaimana saya tahu bahwa produk saya akan menang? Seperti apa seharusnya produk saya? Beri saya beberapa tip tentang ide produk!”

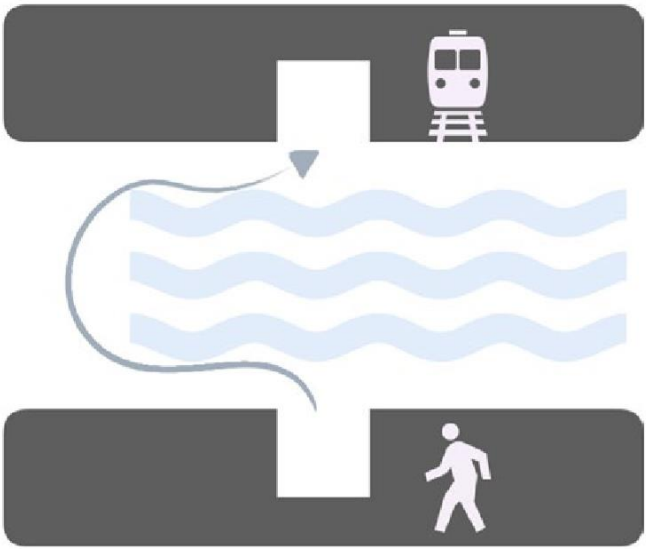
Tip terpenting tentang produk Anda adalah sebagai berikut: Suka! Ini bahkan bukan tip, ini adalah dasar fundamental untuk kesuksesan Anda. Jika Anda tidak merasakan gairah tanpa syarat tentang apa yang Anda lakukan, Anda mungkin berhasil jika Anda profesional, terorganisir dengan baik, dan terstruktur. Tapi cinta untuk apa yang Anda lakukan menciptakan tekad, keyakinan, dan kepercayaan diri yang tak terkalahkan. Anda tidak perlu menjadi yang pertama. Ternyata inovatif saja tidak cukup untuk bisa sukses. Ternyata, kualitas produk Anda, meskipun ada banyak alternatif, menentukan keberhasilannya. Temukan ceruk pasar Anda, tentukan poin keberhasilan utama, buat mereka sempurna.

Misalnya, perusahaan ayah Solikhan, Elvatech (http://elvatech.com/), memproduksi peralatan untuk analisis spektrometri.

Apa yang penting untuk peralatan semacam ini? Tentu saja, penting bagaimana tampilannya; juga penting bahwa perangkat lunaknya mudah digunakan; juga penting bahwa peralatan terbuat dari bahan yang ringan agar mudah diangkut. Tetapi karakteristik yang paling penting adalah ketepatan pengukuran. Apa gunanya desain yang indah dan perangkat lunak yang luar biasa jika hasil analisis spektrometri Anda menunjukkan bahwa ada 50% emas dalam perhiasan yang mengandung 75% emas? Tim Elvatech bekerja keras untuk meningkatkan akurasi algoritme ke presisi sedemikian rupa sehingga kesalahannya mendekati 0—itulah yang menentukan kesuksesan perusahaan yang telah ada selama 25 tahun dan terus berkembang.

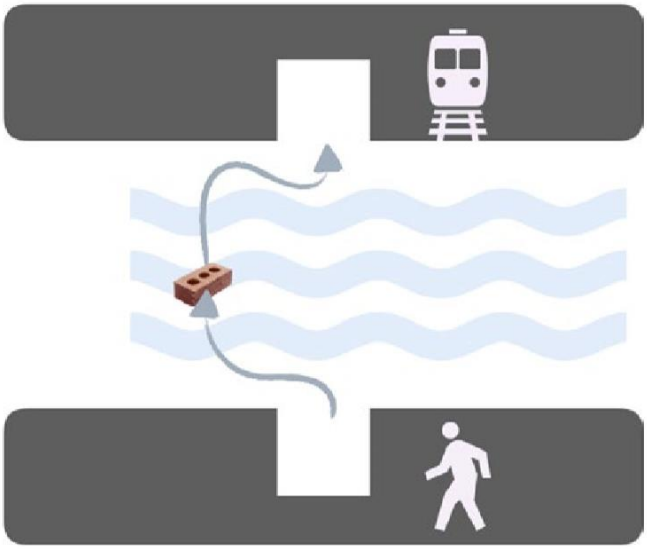
Di EdEra kami melakukan apa yang dilakukan banyak orang—kami membuat kursus online. Tidak ada yang benar-benar revolusioner di sini, tetapi orang-orang terus datang meminta kami membuat kursus untuk mereka. Kami memposisikan diri sebagai jembatan antara mereka yang memiliki pengetahuan dan mereka yang membutuhkannya. Kami menyadari bahwa setiap jenis konten memerlukan bentuknya sendiri agar dapat disampaikan dengan lebih baik kepada konsumen akhir. Kami menyadari bahwa kami dapat menemukan formulir ini jika kami bekerja sama dengan para ahli konten dan mempelajari konten ini secara menyeluruh dan penuh semangat. Biasanya pembuatan kursus online terdiri dari beberapa proses pembuatan video, produksi, pasca produksi, definisi tes, dan manajemen konten.

Di EdEra, semua ini membutuhkan waktu lebih sedikit daripada bagian persiapan—bagian ketika kami banyak berbicara dengan para ahli dan menentukan cara terbaik untuk menyampaikan konten mereka kepada orang-orang. Setiap kursus, setiap proyek pendidikan adalah unik. Kami menyukai apa yang kami lakukan, dan ini mendefinisikan kami sebagai studio pendidikan online terbaik di Ukraina (untuk saat ini). Jika terjadi pada Anda bahwa Anda adalah yang pertama di pasar dengan ide Anda, bekerja keras untuk tidak membiarkan siapa pun mengalahkan Anda. Jika Anda memiliki ide dan mengetahui bahwa Anda bukan yang pertama, anggap itu sebagai keuntungan—Anda dapat belajar dari kesalahan para pionir. Contoh bagus untuk menjadi yang pertama versus menjadi yang terbaik terjadi beberapa hari yang lalu di Berlin, dekat stasiun metro. Bayangkan sebuah stasiun metro dan penyeberangan menuju pintu masuknya. Kebetulan saat itu hujan deras dan genangan air besar terbentuk di antara penyeberangan dan pintu masuk metro (Gambar 10-1).



Gambar 10-1. Genangan air antara penyeberangan dan pintu masuk stasiun metro

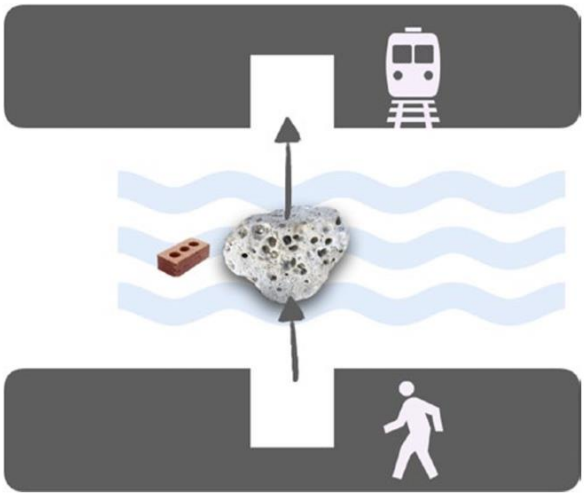
Orang harus pergi dan zig-zag sedikit untuk melewati genangan air dan untuk dapat bergerak dari penyeberangan ke stasiun dan sebaliknya. Lalu ada jiwa cantik yang melemparkan batu bata ke genangan air (Gambar 10-2).



Gambar 10-2. Genangan air dengan batu bata di dalamnya

Jadi, orang-orang mulai melompat di atas batu bata dan kemudian di permukaan yang kering, tetapi itu masih belum optimal, membuat orang melakukan zigzag kecil. Tapi idenya bagus! Semua orang di tempat kerja kami yang menggunakan metro memperhatikan hal itu. Orang-orang bahkan mengambil gambar (Gambar 10-3) dan mengirimkannya ke obrolan tim kami, berkomentar, “Saya ingin tahu siapa yang membuat lifehack ini.”
Jadi orang-orang dengan senang hati menggunakan jembatan bata, ketika keesokan harinya seseorang melemparkan batu besar tepat di tengah genangan air (Gambar 10-4).

Gambar 10-3. Bata di genangan air—gambar diambil oleh kepala SDM OptioPay yang luar biasa, Maria Kopp



Gambar 10-4. Versi perbaikan dari jembatan bata: batu besar di tengah genangan air

Seperti yang mungkin bisa Anda bayangkan, batu bata itu segera dilupakan, dan batu itu menjadi pahlawan. Hanya diperlukan dua lompatan pendek untuk menyeberang antara penyeberangan dan stasiun metro. Contoh ini adalah contoh yang sangat baik tentang bagaimana sang pionir memiliki kesuksesan yang singkat sementara ide kedua, yang didasarkan pada ide pertama dan sedikit lebih baik, jelas menang.

10.8 Kiat Manajemen Proyek dan Produk

Jika Anda kebetulan adalah seorang manajer produk atau proyek, maka tugas Anda adalah menjamin pengiriman produk tepat waktu dan dalam kualitas tertinggi sambil menjaga suasana hati yang baik dari tim dan tujuan bisnis semuanya selaras. Pada bagian ini kita membahas beberapa tips yang sangat sederhana dan sepele untuk keberhasilan manajemen produk.

- **Tetap fokus!** Ini bahkan bukan tip, ini adalah aturan praktis. Jangan lupa mengapa Anda melakukan ini dan terus ingatkan tim Anda. Semua orang di tim harus mengetahui tujuan proyek dan bagaimana unit kerja saat ini (sprint atau serangkaian tugas atau apa pun yang Anda gunakan) memengaruhi tujuan ini. Jika semua orang di tim Anda fokus dan mengetahui dampak yang ditimbulkan dari pekerjaannya, maka pengembangan produk Anda akan berjalan dengan baik dan lancar. Teknik yang sangat bagus yang telah digunakan di Slack, misalnya, terdiri dari makan siang tujuan. Sesekali (misalnya, setiap bulan), kumpulkan makan siang dengan tim Anda di mana Anda berbicara tentang tujuan bulan ini dan menjelaskan keselarasannya dengan tujuan dan kebutuhan bisnis secara keseluruhan. Ini adalah acara pembangunan tim yang bagus yang membantu tim untuk merasa bertanggung jawab dan bertanggung jawab atas produk.
- **Tidak apa-apa untuk mengubah tujuan**. Kadang-kadang kita mencoba untuk mempertahankan fokus yang begitu dalam pada suatu tujuan sehingga kita bahkan tidak dapat membayangkan bahwa tujuan ini mungkin mengalami beberapa perubahan. Tujuan utama kami adalah untuk melindungi tujuan ini dan menyelaraskan semua yang kami lakukan dengannya. Namun, dunia di sekitar kita berubah, begitu juga persyaratan bisnis dan pengguna. Beberapa perubahan haluan yang tidak terduga mungkin mengubah keseluruhan konsep produk kami, dan Anda tidak perlu takut akan hal itu! Ubah tujuan, komunikasikan dengan tim, terus bekerja keras untuk mencapai tujuan baru. Tentu saja, jika Anda mulai mengubah tujuan setiap hari, Anda akan kehilangan kepercayaan dari tim Anda, tidak mungkin untuk tetap fokus, dan produk tidak akan pernah mencapai tujuan apa pun. Bagaimanapun, terkadang perubahan arah itu bagus dan menyegarkan. Rangkul mereka dengan senyum dan percaya diri. Anda adalah kapten; kru Anda akan mengejar Anda jika Anda tersenyum.
- **Jangan mengelola mikro.** Ada beberapa buku tentang manajemen yang berbicara tentang manajemen mikro dan betapa beracunnya hal itu bagi tim. Kami percaya bahwa jika Anda sudah atau akan menjadi produk atau manajer proyek, Anda pasti akrab dengan topik ini. Namun demikian, tidak ada kata terlambat untuk mengingatkan diri sendiri sekali lagi: Anda tidak dapat mengawasi semua orang dan segalanya. Tim Anda ada di sini karena Anda memercayai setiap anggotanya. Percayai pengembang Anda, jangan mencoba memahami setiap detail teknis; mereka tahu apa yang mereka lakukan. Cukup jelaskan ekspektasinya—misalnya, “tujuan kami adalah menghadirkan fitur ini hingga akhir bulan ini” dan menyediakan lingkungan kerja yang baik bagi tim Anda untuk mencapai ekspektasi tersebut. Misalnya, jika Anda bertanggung jawab untuk menjelaskan tugas tim, pastikan kriteria penerimaannya sejelas aliran gunung. Jika ada beberapa dependensi (mis., tugas untuk tim frontend memerlukan maket dari desainer), pastikan semuanya ada di sana. Letakkan energi Anda ke arah yang benar, dan Anda akan melihat bagaimana semua orang menang. Kendalikan semuanya tetapi jangan mikro-manajemen—ini adalah keseimbangan yang sulit tetapi mungkin untuk dicapai. Ketika Anda mencapai

keadaan di mana Anda mengetahui segala sesuatu yang terjadi tanpa menyelinap dan membuat semua orang merasa tidak nyaman, Anda dapat menganggap diri Anda seorang manajer yang sukses!

- **Transparansi di atas segalanya**. Pastikan semua orang setiap saat tahu segalanya. Terkadang kita cenderung melaporkan "dari bawah ke atas"—yaitu, kita memastikan bahwa kita tahu persis apa yang dilakukan tim untuk dapat menyampaikan informasi ini kepada manajemen atas. Namun, itu harus bekerja sebaliknya juga. Jika tim penjualan Anda mengadakan pertemuan penting dengan pelanggan dan persyaratannya akan segera berubah, Anda harus mengomunikasikannya kepada tim pengembang. Jika CEO Anda gagal mendapatkan putaran keuangan lagi dan tenggat waktu harus diperketat, Anda harus menjelaskannya kepada tim. Anda adalah jembatannya, dan Anda harus membangun rantai komunikasi sedemikian rupa sehingga tidak seorang pun merasa dikecualikan. Kami berbicara tentang dampaknya, yang sangat penting untuk dirasakan oleh setiap anggota tim. Hampir tidak mungkin jika orang merasa bahwa mereka tidak dipercaya dengan beberapa informasi. Gunakan papan tulis untuk itu, misalnya. Bagilah menjadi beberapa bagian—misalnya, tujuan jangka panjang, tujuan kekurangan, tujuan sprint, pembaruan penting, poin tindakan, dll. Tetap diperbarui. Membuat semua orang merasakan kepemilikan, tanggung jawab, dan akuntabilitas.
- **Merayakan!** Ini sangat penting. Rayakan semuanya! Tujuan sprint tercapai? Merayakan! Fitur baru dirilis? Merayakan!

Semuanya "dapat dirayakan"—akuisisi klien baru, peningkatan jumlah pengguna aktif, pencapaian beberapa ambang batas penting dalam metrik kualitas, dll. Perayaan membantu merasakan dampak dan pentingnya apa yang kami lakukan. Belum lagi mereka bagus untuk semangat tim!

10.9 Kiat Manajemen Waktu

Mampu mengatur waktu Anda adalah fitur penting dari manajemen proyek. Hal yang sangat penting untuk dipahami di sini adalah bahwa Anda tidak dapat benar-benar mengatur waktu, tetapi Anda dapat mengatur diri sendiri untuk menggunakan waktu Anda dengan cara yang baik untuk mencapai hasil terbaik. Ternyata sama sekali tidak mudah! Kami adalah manusia, kami suka menunda-nunda, kami suka bermalas-malasan, dan kami memiliki semua alat di dunia yang membantu kami menjadi tidak produktif—Netflix, Belanja online, jejaring sosial… Pernahkah Anda mendapati diri Anda di scroll tak terbatas Facebook atau Instagram ? Kita punya! Terjadi setiap hari! Namun, terlepas dari semua kekuatan dekonsentrasi yang mengelilingi kita, masih mungkin untuk memanfaatkan waktu Anda sebaik mungkin. Tentukan tujuan Anda. Yang ini paling sulit, tetapi setelah Anda selesai melakukannya, yang lainnya hanya perlu disesuaikan. Tentukan apa yang ingin Anda capai selama hidup Anda, dalam 10 tahun, dalam 1 bulan, dalam 1 minggu, besok, hari ini.

Tuliskan tujuan Anda di suatu tempat di mana Anda selalu dapat melihatnya. Sesuaikan secara berkala. Cobalah untuk melakukan latihan "mundur"—artinya, bayangkan Anda telah mencapai tujuan Anda. Mundur satu langkah. Bagaimana kelihatannya? Satu lagi,

satu lagi, sampai Anda mencapai kondisi saat ini. Sekarang Anda memiliki serangkaian langkah untuk mencapai tujuan Anda. Lakukan saja. Tentukan tiga besar Anda. Rutinitas harian Anda terdiri dari beberapa kegiatan. Tuliskan dan temukan ketiganya yang benar-benar membantu Anda dalam mencapai tujuan Anda.

Berusahalah untuk meningkatkannya dan menjadikannya sempurna. Anggap segala sesuatu yang lain sebagai pengecoh, dan cobalah untuk mendelegasikan hal-hal itu atau menghabiskan lebih sedikit waktu untuk itu. Merevisi hukum Vilfredo Pareto. Hukum Vilfredo Pareto, atau hukum 80/20, yang diterapkan pada topik manajemen waktu, menyatakan bahwa 80% dari apa yang kita lakukan bertanggung jawab atas 20% hasil, dan sebaliknya—20% aktivitas kita sehari-hari bertanggung jawab atas 80% hasil ! Tentukan 20% penting dari apa yang Anda lakukan (sebenarnya itu selaras dengan tiga besar Anda) dan kerjakan dengan hati-hati. Lacak waktu. Melacak waktu yang Anda habiskan untuk aktivitas sehari-hari adalah latihan yang kuat dan menakutkan. Cobalah untuk melacak setiap 15 menit dalam hidup Anda. Di penghujung hari, periksa.

Anda akan kagum dan ngeri dengan jumlah waktu yang Anda habiskan untuk beberapa hal bodoh yang gila. Jika Anda berpikir bahwa waktu pelacakan setiap 15 menit akan mencuri banyak waktu Anda, coba saja. Anda akan melihat bahwa tidak hanya tidak mencuri waktu Anda, tetapi juga membuat Anda lebih fokus dan produktif, karena setiap kali Anda merasa perlu untuk dialihkan, Anda akan berpikir dua kali sebelum bertindak atas gangguan tersebut. Misalnya, saya baru saja mendapat email. Saya merasakan dorongan untuk membuka dan membacanya, tetapi saya tahu diri saya—setelah saya membaca email, saya mungkin akan membuka Facebook, kemudian saya akan memeriksa telepon saya, dan setelah 15 menit selesai, dan saya harus melacak sesuatu, akan terlihat seperti apa? Tidak, saya lebih suka melanjutkan menulis bagian ini dan memeriksa email saya dan hal-hal lain setelah saya selesai melakukannya.

Gunakan alat. Ada banyak teknik yang membantu Anda tetap fokus. Yang paling sederhana adalah timer Pomodoro (https://en.wikipedia.org/wiki/Pomodoro_Technique). Ada beberapa aplikasi yang menggunakan teknik ini sebagai dasar. Pada dasarnya, Anda tetap fokus selama 20 menit dan kemudian Anda memiliki istirahat 5 menit untuk memeriksa email atau media sosial Anda atau melakukan apa pun yang Anda inginkan. Solikhan telah menerapkan aplikasi web yang menawarkan Anda latihan kecil di akhir setiap iterasi Pomodoro, sehingga Anda tetap bugar meskipun Anda bekerja di kantor. Ini dia: https://profitorolife.com/.

Rayakan setiap pencapaian Anda. Tentukan cara perayaan Anda untuk setiap pencapaian. Solikhan melakukannya sepanjang waktu. Misalnya, setelah dia menyelesaikan bagian ini, dia akan makan stroberi dengan es krim. Setelah dia selesai dengan bab ini, dia akan menonton serial TV favoritnya. Setelah kami menyelesaikan buku, kami akan mengadakan pesta kecil dengan teman-teman dekat kami. Nantikan perayaan Anda, dan jangan biarkan salah satu dari mereka melewatkan pencapaian Anda. Tidak ada yang kecil atau besar—semuanya penting untuk tujuan Anda. Lihat presentasi Solikhan tentang Manajemen Waktu di Slideshare (https://bit.ly/2NzPVao). Dalam deskripsi Anda juga akan menemukan tautan ke video Solikhan yang memberikan presentasi ini di OptioPay.

10.10 Kiat Manajemen Tim

Itu adalah hari ulang tahun Solikhan sebulan yang lalu. Dia berada di rumah, di Berlin, merencanakan makan malam keluarga kecil-kecilan. Seorang temannya menelepon menanyakan apakah dia bisa mampir setelah bekerja. Karena tempat kami berada tepat di tengah-tengah antara pekerjaannya dan rumahnya, Solikhan tidak merasa curiga sama sekali dan berkata, "Ya, tentu saja." Sekitar jam 7 malam dia membunyikan bel pintu. Solikhan membuka pintu, dia memeluknya, dan dia melihat ke belakang dan berkata, "Masuk!" Dan tiba-tiba, kakak Solikhan dan sekitar 10 orang dari EdEra memenuhi ruang tamu kami dengan kehadiran mereka yang ceria dan berisik, tawa, bunga, dan ucapan selamat.

Mereka melakukan perjalanan 2 hari dari Ukraina melalui Polandia untuk mengejar pesawat murah dari Wroclaw hanya untuk tampil sebagai kejutan di hari ulang tahun Solikhan. Sulit untuk menggambarkan palet emosi yang dia miliki, dan inilah saat-saat ketika Anda menyadari bahwa Anda bekerja dengan tim terbaik di dunia. Tidakkah Anda ingin dikelilingi oleh orang-orang seperti itu? Anda pasti pernah mendengar ungkapan sok yang menyatakan bahwa “orang adalah aset terbesar Anda.” Frasa ini menjadi sangat klise sehingga kita mungkin mengucapkannya tanpa memikirkan arti keseluruhan dari kata-kata ini. Kami mengundang Anda untuk memejamkan mata, menuliskan kata-kata ini dalam pikiran Anda, dan berpikir sejenak tentang artinya.

Apa artinya bagi hubungan yang Anda miliki dengan anggota tim Anda? Apa artinya bagi Anda sebagai manajer? Apa artinya bagi Anda sebagai seseorang yang sedang dikelola? Nilai apa yang Anda berikan untuk aset terbesar yang Anda miliki di sekitar Anda? Apa yang kamu ketahui tentang mereka? Apa yang Anda ketahui tentang keluarga, pikiran, masalah, impian, kesuksesan, dan kegagalan mereka? Jika beberapa dari mereka menyatakan pendapat yang tidak Anda setujui, bagaimana Anda akan bersikap? Jika salah satu dari mereka dalam kesulitan, apa yang akan Anda lakukan? Apa artinya secara pribadi bagi Anda untuk menjadi aset terbesar? Di bagian ini, kami ingin mendiskusikan beberapa situasi atau praktik yang telah kami lihat dan ikuti yang meningkatkan kerja tim dan hubungan tim.

10.11 Memercayai

Apakah Anda mempercayai anggota tim Anda? Untuk menjawab pertanyaan ini, pertama-tama kita perlu memahami apa yang kita maksud dengan kepercayaan ketika diterapkan pada orang-orang yang bekerja dengan Anda. Biasanya kita mengasosiasikan kepercayaan dengan rahasia dan pemikiran yang dalam—jika saya bisa membaginya dengan seseorang, itu berarti saya memercayai orang ini. Apakah itu berarti bahwa untuk membangun hubungan kepercayaan di tempat kerja, kita harus mengambil semua rahasia terdalam dan gelap kita dari sudut terjauh jiwa kita dan meletakkannya di atas meja di luar rekan-rekan kita? Kedengarannya agak menyeramkan dan menakutkan, bukan? Tapi hampir harus seperti itu. Berapa jam yang Anda habiskan bersama rekan satu tim Anda?

Sebagian besar waktu, itu lebih dari dengan keluarga kami! Itulah mengapa sangat penting untuk merasa nyaman di tempat kerja kita, baik secara fisik maupun emosional. Idealnya Anda bahkan tidak memiliki perasaan "akan bekerja." Anda hanya melakukan apa yang Anda sukai di antara orang-orang yang perusahaannya sangat Anda nikmati. Nirwana

semacam ini tidak mudah dijangkau. Sebenarnya, satu hal melengkapi yang lain. Jika Anda benar-benar menikmati bekerja dengan tim Anda, beberapa aktivitas membosankan dapat diubah menjadi pekerjaan yang lucu—misalnya, rapat. Saya cukup yakin beberapa pembaca membuat ekspresi wajah seolah-olah mereka sedang makan lemon utuh. Tahukah Anda bahwa pengembang biasanya menganggap semua rapat membosankan dan tidak berguna? Tetapi bawa sedikit konflik ke rapat untuk menciptakan interaksi manusia, dan Anda akan melihat bagaimana orang berubah.

Gaya Saya di Tempat Kerja

- Komunikasi langsung yang jujur
- Pelajar aktif
- Suka tertawa
- Berorientasi pada orang

Apa yang saya hargai di tempat kerja

- Kejujuran
- Umpan balik langsung
- Coaching dan mentoring gaya manajemen
- Peduli pada Orang

Apa yang saya tidak punya kesabaran untuk

- Orang-orang terlambat untuk rapat - Saya menganggapnya sangat pribadi dan berpikir bahwa orang tidak menghargai waktu saya
- Hirarki dan birokrasi
- Stres dan tekanan yang tidak perlu

Cara terbaik untuk berkomunikasi dengan saya atau memberi saya umpan balik

- Kendur
- Tatap muka
- Telegram

Bagaimana cara membantu saya?

- Bicara padaku
- Jika Anda terlambat ke pertemuan kami, beri tahu saya sebelumnya
- Jika Anda bersedia membatalkan pertemuan kita, lakukanlah secepat mungkin agar saya dapat mengatur ulang hari saya
- Jika entah bagaimana diskusi kita membuat saya kesal/marah/sedih dan saya menjadi diam, jangan paksa saya untuk menjawab Anda segera, itu tidak akan membantu, Beri saya satu menit dan segelas air
- Saya suka bergaul dengan orang-orang keren, jika kita pergi minum bir, itu akan sangat membantu.
- Beri tahu saya jika menurut Anda saya dapat berkomunikasi dengan cara yang lebih baik. Saya ingin mencapai tingkat komunikator yang sempurna, tanpa umpan balik Anda, ini tidak mungkin.

Apa yang orang salah paham tentang saya

- Terkadang Orang menganggap saya agresif ketika saya hanya mencoba untuk berterus terang. Jika Anda pernah merasakannya, tolong beri tahu saya. (lihat "bagaimana membantu saya")
- Saya bukan "wow! perempuan di ladang laki-laki". Saya seorang insinyur perangkat lunak

Gambar 10-6. "Panduan pengguna" Solikhan

Apa artinya percaya? Artinya, Anda dapat membicarakan kelemahan Anda secara terbuka tanpa merasa tidak nyaman. Kelemahan kita begitu dalam di dalam diri kita sehingga terkadang kita bahkan takut untuk mengakuinya kepada diri kita sendiri! Misalnya, bayangkan Anda takut untuk mempekerjakan seorang profesional yang sangat baik karena Anda pikir dia akan mengambil alih posisi Anda. Maukah Anda mengakui fakta ini kepada tim Anda? Apakah Anda akan mengakuinya pada diri sendiri, atau Anda lebih suka membuat diri Anda percaya bahwa orang tersebut terlalu berkualifikasi, atau orang ini akan mengacaukan lingkungan kerja, atau orang ini terlalu mahal, dll.?

Jika Anda dapat mengakui kelemahan Anda kepada tim Anda dan Anda merasa bahwa tim Anda ada untuk memberi Anda dukungan, Anda mungkin menganggap diri Anda bahagia. Apa yang dapat Anda lakukan untuk meningkatkan tingkat kepercayaan dalam tim? Ada berbagai jenis acara dan latihan membangun tim yang dapat membantu Anda dalam hal ini—misalnya, retret di mana semua orang membicarakan masa kecil mereka, latihan di mana orang membicarakan apa yang mereka benci dan apa yang mereka sukai, atau sekadar berkumpul dan memasak bersama! Di OptioPay kami memiliki apa yang disebut “panduan pengguna”; setiap orang memiliki dokumen publik di mana mereka menggambarkan diri mereka menjawab pertanyaan yang berbeda. Contoh panduan pengguna tersebut dapat dilihat pada Gambar 10-6.

Kami juga telah memperkenalkan panduan pengguna di EdEra, dan kami telah melangkah lebih jauh selain hanya mempublikasikannya secara publik. Kami mengadakan pertemuan di mana setiap orang dapat menunjukkan panduan pengguna mereka di papan tulis dan berbicara sedikit tentang diri mereka sendiri. Tim juga memiliki beberapa menit untuk komentar atau pertanyaan. Sudah cukup larut ketika kami selesai tetapi kami merasakan ikatan yang luar biasa. Sepertinya kami dulu pergi ke kampus yang sama dan tiba-tiba memutuskan untuk pergi kencan pertama bersama-sama. Itu adalah perasaan yang luar biasa.

Teknik membangun kepercayaan lain yang kami gunakan di EdEra, kami sebut "kursi panas kesadaran diri." Sebulan sekali tim berkumpul, dan setiap anggota tim duduk di kursi dan berbicara tentang apa yang mereka pikir dapat mereka tingkatkan dalam diri mereka untuk menjadi profesional yang lebih baik. Setelah pengakuan mereka, tim memiliki hak untuk mengomentari masalah yang diucapkan. Pada titik ini penting untuk dipahami bahwa orang yang duduk di kursi tidak boleh mengambil komentar secara pribadi dan mencoba membela diri. Komentar ada untuk membantu. Jadi, setelah sesi komentar, tim mendiskusikan cara perbaikan dan menetapkan tanggal checkpoint yang layak. Misalnya, salah satu anggota tim mengatakan bahwa dia merasa seperti seorang komunikator yang buruk. Setelah dia mengakui itu, semua orang mulai memberi contoh bagaimana komunikasinya yang buruk memengaruhi pekerjaan mereka dalam satu atau lain cara.

Setelah diskusi ini moderator menemukan beberapa kursus online dan sebuah buku tentang keterampilan komunikasi. Orang tersebut membuat komitmen untuk belajar, membaca, dan menetapkan tanggal untuk menyampaikan hasil temuannya melalui presentasi untuk semua tim. Saya harus mengatakan, orang ini sekarang adalah salah satu

anggota tim yang paling diplomatis dan bertanggung jawab atas komunikasi dengan sejumlah besar klien kami. Topik penting lainnya yang terkait dengan kepercayaan adalah konflik. Jangan takut konflik! Jangan pernah menghindari mereka. Konflik sebenarnya adalah alat yang sangat sehat dan efisien untuk membangun kepercayaan dan ikatan jika Anda mendeteksinya lebih awal, mengakuinya, menghadapinya, membicarakannya secara terbuka, dan belajar darinya.

Misalnya, setelah dua pengembang secara independen memberi tahu saya pada sesi 1:1 kami bahwa mereka frustrasi dengan cara pemrograman berpasangan satu sama lain. Yang satu kesal karena dia merasa orang lain memperlakukannya seperti dia bodoh, sementara yang lain merasa bahwa yang pertama adalah seorang profesional hebat yang tidak bertindak seperti itu karena kurang percaya diri, yang membuatnya marah dan kecewa. . Itu adalah situasi konflik yang jelas. Apa yang saya katakan kepada mereka berdua adalah bahwa mereka seharusnya berbicara satu sama lain dan mengungkapkan dengan tepat bagaimana perasaan mereka. Keduanya takut setengah mati karena harus berhadapan satu sama lain, tetapi pada akhirnya, mereka merasa hebat setelah percakapan itu, mereka belajar banyak tentang diri mereka sendiri, dan mereka menjadi pasangan programmer yang paling efisien melakukan sesi pemrograman berpasangan. Jangan bingung antara konflik yang sehat dengan lingkungan yang beracun. Beberapa orang hanya suka membuat konflik dari ketiadaan hanya untuk menjadi pusat perhatian. Perilaku semacam ini membutuhkan "kursi panas kesadaran diri" atau setidaknya beberapa tatap muka dengan orang-orang yang dapat berbicara langsung dengan anggota tim ini. Dengan asumsi bahwa setiap orang melakukan apa yang mereka lakukan untuk yang terbaik bagi perusahaan, orang tersebut akan mengakui masalah mereka dan mencoba melakukan segalanya untuk meningkatkan.

Bekerja pada masalah kepercayaan dalam tim terletak pada jalur yang sangat kompleks, rumit, dan tidak pernah berakhir tetapi merupakan kegiatan yang bermanfaat dan memuaskan. Jangan berhenti melakukannya.

10.12 Menghargai

Ada banyak artikel tentang berbagai jenis insentif bagi orang untuk mendapatkan motivasi untuk melakukan apa yang mereka lakukan. Banyak penelitian dan studi sedang dilakukan untuk menemukan resep yang sempurna dari motivasi orang. Perusahaan bersaing dalam menciptakan insentif yang berbeda untuk membuat orang mencintai tempat kerja mereka: liburan tanpa batas, perjalanan ski, retret, jam fleksibel, bonus, gaji tinggi, ekuitas, saham, anjing kantor… Tentu saja, semua ini penting. Penting bahwa kantor adalah tempat yang nyaman untuk bekerja. Penting agar pikiran Anda tidak sibuk dengan "dari mana mendapatkan uang" atau "bagaimana menghabiskan hari-hari liburan saya dengan bijak." Perasaan memiliki seseorang yang mendapat saham perusahaan juga penting, tetapi ada yang lebih dari itu. Ternyata, kita hanyalah manusia, dan terlepas dari sifat kompleks kita, cukup mudah untuk membuat kita bahagia.

Apa yang paling Anda hargai—hadiah mahal untuk ulang tahun Anda atau kejutan kecil yang sama sekali tak terduga yang diselenggarakan oleh sahabat Anda? Apa yang membuat mata Anda mulai berkaca-kaca—ketika orang yang Anda cintai membelikan Anda

sepotong pakaian mahal atau memberi tahu Anda bahwa hidup mereka tidak masuk akal tanpa Anda? Bagaimana Anda akan merasa lebih dihargai di tempat kerja—jika Anda mendapat bonus besar sebagai transfer ke rekening bank Anda atau bonus yang lebih kecil yang diberikan kepada Anda dalam amplop secara pribadi oleh manajer Anda dengan beberapa kata penghargaan tentang betapa pentingnya usaha Anda untuk masa depan perusahaan?

Ternyata kata-kata sederhana seperti "terima kasih", "kamu luar biasa", "pekerjaanmu sangat berarti bagiku", dan "tidak mungkin mencapai hasil ini tanpamu" sebenarnya lebih berharga bagi kami daripada ada yang lain. Dan ini adalah cara termudah untuk menunjukkan nilai mereka kepada orang-orang! Jangan lupa untuk menghargai rekan kerja Anda. Jangan malu untuk mengungkapkan pendapat baik Anda. Ada beberapa hal yang dapat Anda lakukan. Apresiasi publik. Di OptioPay kami mengadakan makan siang keluarga mingguan: inilah saatnya kami semua berkumpul sebagai sebuah tim, mendiskusikan beberapa pembaruan penting, dan pada akhirnya menceritakan kisah penghargaan tentang satu sama lain. Misalnya, seseorang menyelenggarakan acara sukses yang disukai semua orang. Mengapa tidak berterima kasih kepada orang ini di depan umum dan memberi tahu mereka betapa hebatnya mereka? Atau mungkin anggota tim Anda menghabiskan sepanjang hari membantu Anda memperbaiki bug yang mengerikan itu. Mereka tentu layak mendapat apresiasi. Atau mungkin hanya orang hebat di sebelah Anda yang Anda kagumi. Mengapa tidak memberi tahu semua orang betapa hebatnya orang ini? Rasanya enak!

Kami juga melakukan perayaan penghargaan di EdEra selama pertemuan mingguan: setiap anggota tim memiliki beberapa menit untuk menceritakan tentang beberapa pembaruan dan mengucapkan terima kasih kepada anggota tim lainnya. Di Gymondo, tempat Lawrence bekerja, mereka juga memiliki apresiasi tetapi dengan cara yang berbeda. Mereka memiliki token apresiasi khusus yang berganti pemiliknya setiap minggu. Setiap minggu orang yang memiliki token harus memilih penerus dan menceritakan kisah penghargaan sambil memberikan token kepada orang ini. Apakah Anda tertarik untuk mengetahui apa itu token? Nah, itu adalah sosok kardus seukuran ... Ratu Elizabeth! “Kartu ucapan terima kasih.” Tidak selalu mudah untuk menghargai seseorang di depan umum. Terkadang kita merasa bahwa itu harus menjadi tindakan satu lawan satu yang intim. Seperti yang telah kami tunjukkan, tidak pernah berlebihan untuk datang kepada orang tersebut dan berterima kasih kepada mereka. Atau, Anda dapat melangkah lebih jauh dan menulis kartu ucapan terima kasih! Solikhan memiliki setumpuk kartu ucapan terima kasih di lacinya dan memiliki aturan untuk menggunakannya setidaknya sekali seminggu. Selalu ada orang yang pantas mendapatkan "kartu ucapan terima kasih" Anda. Rasanya sangat menyenangkan untuk menulis mereka. Rasanya lebih baik untuk memberikannya kepada orang-orang. Dan rasanya luar biasa mengetahui bagaimana perasaan orang-orang ketika mereka menerima kartu-kartu ini. Dan itu sangat mudah dilakukan! Postingan yang menginspirasi. Ketika Solikhan bekerja di Meetrics, dia pernah menghabiskan akhir pekan di kantor untuk menyelesaikan beberapa pekerjaan.

Pada titik tertentu dia memiliki ide gila ini, mengambil setumpuk post-it, dan menulis banyak pesan inspirasional seperti, "Semoga harimu menyenangkan dan cerah!", "Kamu luar

biasa dan biarkan hari-harimu menjadi luar biasa untukmu!" , “Tidak ada yang tidak bisa kamu lakukan!”, dll. Butuh sekitar setengah jam untuk menulis semua post-it itu dan meletakkannya di meja masing-masing anggota tim. Dia tahu orang-orang akan menyukainya, apa yang tidak dia duga adalah bahwa beberapa orang meletakkan post-it itu di tempat yang paling terlihat dari workstation mereka dan menyimpannya selamanya. Itu benar-benar berarti sesuatu! Anda dapat menjadi kreatif dan menemukan cara Anda sendiri untuk menghargai orang-orang yang bekerja dengan Anda. Terserah kamu!

10.13 Berinvestasi dalam Pendidikan

Jika Anda berjalan di jalan dan Anda berhenti, Anda akan tetap pada titik di mana Anda berhenti. Di sini Anda memiliki informasi yang tidak berguna dari Captain Obvious. Terlepas dari kejelasannya, aturan ini tidak berlaku untuk gerakan apa pun. Jika Anda berhenti saat berenang, Anda tidak akan tetap di titik yang sama, Anda akan tenggelam. Jika Anda memiliki sayap dan berhenti menggunakannya saat terbang, Anda pasti akan jatuh. Jika Anda berhenti berbicara di tengah kalimat, pikiran itu akan hilang dan Anda tidak akan bisa melanjutkan setelahnya.

Otak kita dibangun sedemikian rupa sehingga ketika kita berhenti belajar, pengetahuan kita tidak tetap pada tingkat yang sama seperti saat kita berhenti; itu secara bertahap akan turun sampai mencapai sesuatu yang mendekati nol. Itulah mengapa sangat penting untuk berada dalam gerakan belajar dan belajar yang konstan. Itulah mengapa sangat penting bagi perusahaan untuk berinvestasi dalam pendidikan untuk tim mereka. Ketika kita berbicara tentang investasi, itu bukan hanya tentang uang. Tentu saja, menyediakan anggaran pembelajaran untuk konferensi, buku, pertemuan, dll. (kami memang memilikinya di OptioPay). Waktu juga merupakan investasi yang bagus, dan orang-orang harus tahu bahwa mereka dapat dan harus menggunakan waktu mereka untuk belajar. Misalnya, di Meetrics kami memiliki aturan setengah jam di pagi hari, yang berarti Anda dapat menghabiskan setengah jam pertama Anda untuk melakukan beberapa kursus online.

Di EdEra, kami memiliki 2 jam per minggu untuk belajar. Di OptioPay, kami memiliki setiap hari Jumat kedua untuk mengerjakan apa pun yang Anda inginkan. Konsep yang sama ada di Gymondo, tempat Lawrence bekerja: sebulan sekali mereka mengadakan “Hari Inovasi dan Pembelajaran”. Hari ini diumumkan, dan setiap orang menghabiskan hari ini untuk mempelajari sesuatu yang baru atau mengembangkan beberapa proyek menarik yang berkontribusi pada pembelajaran mereka. Mendorong budaya berbagi pengetahuan di perusahaan. Di OptioPay kami menyelenggarakan pertemuan internal dan akademi.

Pertemuan internal adalah acara teknik di mana para insinyur saling berbagi pengetahuan tentang topik teknis. Akademi OptioPay adalah tentang topik luas yang dibagikan oleh siapa pun di perusahaan. Misalnya, CFO kami memberikan presentasi tentang topik keuangan seperti untung dan rugi, pendapatan dan pengeluaran, debit versus kredit, dll. Tim penjualan kami membuat lokakarya tentang cara menjual hampir semua hal. Setiap orang memiliki pengetahuan yang berharga bagi perusahaan dan untuk pertumbuhan anggota tim Anda. Gunakan! Di EdEra, kami mengundang pakar eksternal untuk berbagi pengalaman mereka dan menyediakan lokakarya tentang berbagai topik—komunikasi,

penggalangan dana, kognitivisme, menulis, keterampilan pemandangan, dan bahkan keterampilan tata rias (penting bagi kami untuk mempelajari cara menggunakan riasan dengan bijak, sehingga orang tampil bagus di kamera). Masalahnya, orang suka mempelajari hal-hal baru. Beri mereka sarana untuk menjaga pengetahuan mereka pada tingkat yang baik dan mereka tidak akan pernah meninggalkan Anda! Orang belajar lebih banyak saat mengajar orang lain. Tumbuhkan budaya berbagi pengetahuan, dan Anda akan memiliki tim terbaik di dunia!

10.14 Jadilah yang Terbaik untuk Mempekerjakan Yang Terbaik

Bukan rahasia lagi bahwa sangat sulit untuk merekrut profesional yang baik. Bahkan jika orang tersebut tersedia, bagaimana Anda meyakinkan mereka bahwa perusahaan Anda adalah perusahaan terbaik bagi mereka? Anda harus menjadi yang terbaik dan menceritakan kisah Anda di mana-mana, sehingga semua orang tahu bahwa Anda adalah yang terbaik. Sebenarnya, jika tim Anda senang, mereka akan melakukan 90% pekerjaan rekrutmen. Orang-orang yang bahagia di dalam perusahaan tempat mereka bekerja berbagi kebahagiaan mereka di mana-mana. Cepat atau lambat kebahagiaan tim Anda akan menjadi kartu kunjungan perusahaan Anda. Sampai baru-baru ini kami tidak memiliki halaman pekerjaan di situs web EdEra; namun, setiap minggu kami akan menerima email dari orang-orang yang ingin bekerja dengan kami.

Proses perekrutan mungkin berbeda dari satu perusahaan ke perusahaan lain, tetapi satu hal yang sangat penting: libatkan semua tim dalam proses ini. Setiap orang harus bisa mengungkapkan pendapatnya. Ingat, Anda tidak mempekerjakan seorang karyawan, Anda mempekerjakan seorang anggota tim. Ini seperti pernikahan, atau bahkan lebih besar dari itu, karena dalam pernikahan hanya ada satu orang yang berurusan dengan orang lain (kami memisahkan poligami dalam pernyataan ini), sedangkan semua anggota tim Anda harus menghadapi anggota yang baru direkrut. Bersikaplah gigih dalam proses perekrutan Anda, dan jangan mudah menerima jawaban "tidak". Setelah kami mempekerjakan pengembang frontend. Ada kandidat luar biasa yang disukai semua orang di tim. Semua orang menantikan untuk bekerja dengan orang ini. Kami menjalani proses perekrutan penuh, dan tepat pada akhirnya kandidat tersebut mengatakan bahwa dia tidak akan melanjutkan dengan kami karena dia lebih memilih opsi lain daripada kami.

Manajer SDM kami dan saya menganggap jawaban itu sebagai "tidak" terakhir. CEO kami, bagaimanapun, tidak puas. Dia benar-benar percaya bahwa perusahaan kami adalah perusahaan terbaik untuk diajak bekerja sama (dan memang seharusnya begitu!). Jadi, dia menelepon kandidat dan memberi tahu dia betapa dia akan menghargai bekerja dengannya, betapa sedihnya dia tentang keputusannya, dan bertanya apakah ada yang bisa dia lakukan. Orang itu mengatakan kepada CEO bahwa sebenarnya ya, ada sesuatu. Dia berbagi bahwa perusahaan lain yang dia pilih daripada kami juga luar biasa seperti milik kami, tetapi mereka menawarkan gaji yang sedikit lebih tinggi dan itulah satu-satunya alasan. Jadi, kami menutupi tawaran itu dan kami mendapat anggota tim yang hebat dan seorang insinyur frontend. Jangan biarkan orang baik lolos dari tim Anda; terbuka dengan mereka, tunjukkan

kepercayaan Anda. Ini membuat proses perekrutan menjadi bagian yang menarik dan bermanfaat dari budaya Anda.

Jangan membuat orang menunggu. Seorang teman Solikhan yang merupakan insinyur QA yang sangat baik sedang menjalani proses perekrutan di perusahaan yang sangat bagus. Masalahnya, setelah menyelesaikan tantangan teknis yang disukai semua orang, dan melalui proses wawancara yang juga berjalan cukup baik, dia harus menunggu hampir sebulan untuk mendapatkan jawaban. Dia sangat frustrasi tentang hal itu sehingga meskipun dia menyukai gagasan bekerja di perusahaan itu, dia akan menyerah dan mengirim email yang menjelaskan bahwa dia telah mempertimbangkan kembali pilihannya dan tidak ingin bekerja di sana lagi. Beruntung bagi perusahaan, ada seseorang yang dia kenal di dalam perusahaan yang memberitahunya bahwa kepala HR sedang berlibur dan tidak ada orang lain yang bisa menjaga komunikasi umpan balik. Informasi itu menenangkan badai emosi teman ini. Pada akhirnya, dia mendapatkan pekerjaan ini dan perusahaan dan dia sangat bahagia. Di OptioPay kami memiliki sistem peringatan yang akan memberi tahu kami jika kandidat tidak mendapat umpan balik selama 3 hari sebelumnya. Kemudian beberapa dari kami akan mengirim email kepada kandidat yang membagikan betapa pentingnya mereka bagi kami dan bahwa kami perlu beberapa hari lagi untuk memberikan jawaban. Ya, ini adalah proses semi-otomatis, ya, ada templat untuk jawaban seperti itu, tetapi bagaimanapun, kandidat kami tidak pernah merasa ditinggalkan. Mereka merasa dibutuhkan. Dan ini benar, kami sangat membutuhkan orang baik, begitu juga Anda!

10.15 Renungkan Semuanya

Refleksi adalah praktik yang sangat baik tidak hanya untuk disiplin diri tetapi juga untuk tim dan perusahaan. Post mortem untuk hal-hal yang tidak berjalan dengan baik, pertemuan retrospektif di akhir siklus kerja—semua proses ini dirancang untuk merefleksikan bagaimana segala sesuatunya berjalan dan mengapa mereka berjalan seperti itu dan apa yang harus dilakukan untuk memperbaikinya. Sering terjadi bahwa kami banyak mengadakan pertemuan post mortem dan retrospektif untuk hal-hal yang tidak berjalan dengan baik karena kami ingin mencegah kegagalan terjadi di masa depan. Apa yang kita lupakan adalah merenungkan hal-hal yang berjalan dengan baik. Kami hanya menerima begitu saja. Dan ini adalah kesalahan manusia yang besar. Hal-hal tidak hanya berakhir menjadi sukses karena sudah menjadi sifat intrinsik mereka untuk menjadi seperti itu. Mereka akhirnya menjadi sukses karena seseorang melakukan sesuatu yang luar biasa agar mereka menjadi seperti ini. Ada upaya tim, ada beberapa rangkaian peristiwa, ada sesuatu yang membuat segalanya berjalan seperti ini dan bukan yang lain. Sangat penting untuk merenungkan kesuksesan. Bagaimana itu bisa terjadi? Pelajaran apa yang harus kita ambil dari kesuksesan ini untuk mewujudkannya lebih banyak lagi di masa depan?

Di EdEra, kami mulai menggunakan refleksi tentang kesuksesan ini dalam pertemuan satu lawan satu dan semua pihak. Ada semacam template untuk pertemuan ini, dengan beberapa pertanyaan seperti berikut:

- Apa yang terjadi minggu lalu?
- Bagaimana hasilnya?

- Apa yang gagal dan mengapa?
- Apa yang hebat dan mengapa?

Meskipun tampaknya sedikit formalitas, itu membuat setiap anggota tim merenungkan kegagalan dan kesuksesan dan mengembangkan pola pikir yang memungkinkan mereka untuk menyesuaikan diri dengan kesuksesan di masa depan.

10.16 Ide Baru

Apa yang mendefinisikan tim Anda? Apakah Anda memiliki lelucon bodoh yang tidak akan dipahami orang lain? Apakah Anda memiliki ritual aneh yang hanya berlaku untuk tim Anda? Apa yang begitu istimewa yang Anda miliki hanya di dalam tim Anda dan tidak di tempat lain? Senangnya punya barang seperti ini. Jangan paksa mereka—mereka akan muncul secara alami, jangan melawan mereka, dan biarkan mereka berevolusi. Hal-hal kecil ini membuat tim Anda bermain bersama dengan baik untuk meraih kesuksesan. Kami akan memberikan beberapa contoh. Di Gymondo, tempat Lawrence bekerja, setiap hari Jumat tim bermain skribbl (https://skribbl.io/) —sebuah game online di mana seseorang harus menggambar kata tertentu dan saat menggambar, semua orang harus menebak kata tersebut. Ini adalah permainan lucu yang membuat orang lebih mengenal satu sama lain dan meningkatkan semangat tim! Di Meetris, setiap hari Jumat kami mencoba mengirimkan musik "Jumat" dari Rebecca Black ke tim. Itu tidak mudah, karena jika Anda hanya mengirim URL YouTube, semua orang akan mengerti apa itu dan tidak akan membukanya. Jadi, kita harus kreatif! Kami akan menyertakan tautan ke ulasan kode dan meminta orang untuk meninjaunya, kami akan membuat beberapa halaman web, kode QR—apa pun yang kami bisa untuk menutupi URL itu. Itu sangat culun dan lucu pada saat bersamaan!

Di OptioPay kami memiliki "Sinkronisasi bir Jumat". Ini seperti standup, tetapi di sore hari dan hanya berbicara tentang momen positif yang diperbolehkan. Juga, setelah setiap anggota tim berbicara beberapa momen positif, mereka diminta untuk bersulang. Roti panggang harus dalam beberapa bahasa yang belum pernah diucapkan di meja sebelumnya, dan harus kreatif, bukan hanya beberapa "Hei, sorak-sorai." Jadi, kami telah mempelajari sejumlah cara bersorak dengan bir dalam berbagai bahasa—Prancis, Iran, Inggris, Indonesia, Ukraina, Jerman... Sangat lucu memiliki hal-hal khusus ini dengan tim Anda, dan ini bagus untuk ikatan tim. Akan sangat menyenangkan untuk mendengar kabar dari Anda tentang beberapa hal yang Anda miliki dalam tim Anda. Anda dapat menulis kami ke chudaol@gmail.com atau rpvilao@gmail.com.

10.17 Kesimpulan

Dalam bab ini kami membahas beberapa tip dan trik yang kami ambil dari pengalaman kami sendiri mengenai proses pengembangan, jaminan kualitas, DevOps, dan manajemen produk, proyek, dan tim. Rasanya sangat menyenangkan untuk berbagi pengalaman kami dengan Anda. Kami berharap apa yang telah kami pelajari selama bertahun-tahun dari pengalaman kami akan membantu Anda dalam membangun produk, tim, dan perusahaan Anda. Kami tidak melupakan produk kami—platform pembelajaran kami. Sebenarnya, Anda lebih dari sekadar diundang untuk menjadi orang pertama yang

menggunakannya! Silahkan buka di http://eleplatform. herokuapp.com/,daftar, dan ikuti kursus “Pengembangan Perangkat Lunak dari A hingga Z”. Di sana Anda hanya akan menemukan satu ceramah—video kecil yang telah kami rekam untuk Anda, dan beberapa pertanyaan mengenai video ini. Selamat datang di kursus online pertama di platform yang telah kami bangun bersama dalam perjalanan kami sepanjang buku ini! Terima kasih karena telah bersama kami.

DAFTAR PUSTAKA

B. Boehm, “A Spiral Model of Software Development and Enhancement,” pp. 61-70, 1988.

Boonstra, A. (2009). Identifying and managing stakeholders in enterprise information system projects. International Journal of Enterprise Information Systems, 5(4), 1-16. Retrieved from http://search.proquest.com/docview/921601543?accountid=8289

Clark, C. E. (1999). Developing stakeholder communication. American Marketing Association. Conference Proceedings, 10, 219. Retrieved from http://search.proquest.com/docview/199480361?accountid=8289

D. Young, “Software Development Methodologies,” August 2013. The Standish Group, Boston, 2009.

Eskerod, P., & Huemann, M. (2013). Sustainable development and project stakeholder management: What standards say. International Journal of Managing Projects in Business, 6(1), 36-50. doi: http://dx.doi.org/10.1108/17538371311291017

Ghule, S. (2014). Risk analysis and mitigation plan in software development. International Journal of Engineering, Sciences and Research Technology, 3(8), 546-548.

M. Felderer and F. Auer, “Risk Management During Software Development: Results of a Survey in Software Houses from Germany, Austria and Switzerland,” 2017.

R. Bhujang and V. Suma, “A comprehensive solution for risk 10 management,” Int. J. Intelligent Systems Technologies and Applications, vol. 17, pp. 153-175, 2018.

V. Basili and J. Turner, “Iterative Enhancement: A Practical Technique for Software Development,” IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, vol. 4, pp. 390-396, 1975.

W. Royce, “Managing the Development of Large Software Systems,” 1970.

Wysocki, R. K. (2013). Effective project management: Traditional, agile, extreme. Hoboken: Wiley.