YAYASAN PRIMA AGUS TEKNIK

Sistem Informasi Web Dinamis:

CRUD, REPORTING DAN GRAFIS

dengan

CODEIGNITER 4

MOH MUTHOHIR, S.KOM, M.KOM

Sistem Informasi Web Dinamis:
CRUD, REPORTING DAN GRAFIS
dengan
CODEIGNITER 4

MOH MUTHOHIR, S.KOM, M.KOM

YAYASAN PRIMA AGUS TEKNIK

PENERBIT :
YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-634-7227-72-0 (PDF)
9 786347 227720

SISTEM INFORMASI WEB DINAMIS: CRUD, Reporting dan Grafis dengan CodeIgniter 4

Penulis:
Moh Muthohir, S.Kom., M.Kom

ISBN:
978-634-7227-72-0 (PDF)

Editor:
Indra Ava Dianta, S.Kom., M.T

Penyunting:
Eko Siswanto, S.Kom., M.Kom

Desain Sampul dan Tata Letak:
Teguh Setiadi, S.Kom., M.Kom

Penerbit:
Yayasan Prima Agus Teknik Bekerjasama dengan
Universitas Sains dan Teknologi Komputer (Universitas STEKOM)

Anggota IKAPI No: 279 / ALB / JTE / 2023

Redaksi:
Jl. Majapahit no 605 Semarang
Telp. 08122925000
Fax. 024-6710144
Email: penerbit_ypat@stekom.ac.id

Distributor Tunggal:
Universitas STEKOM
Jl. Majapahit no 605 Semarang
Telp. 08122925000

KATA PENGANTAR

Puji syukur kehadirat Tuhan Yang Maha Esa atas rahmat dan karunia-Nya, sehingga buku *Sistem Informasi Web Dinamis: CRUD, Reporting dan Grafiis dengan CodeIgniter 4* ini dapat terselesaikan. Buku ini hadir sebagai respons atas kebutuhan mendalam akan panduan praktis dan komprehensif dalam mengembangkan aplikasi web dinamis. Latar belakang penulisan buku ini didasari oleh observasi bahwa banyak mahasiswa menghadapi tantangan dalam menjembatani teori dan praktik pengembangan web yang efektif. Oleh karena itu, tujuan utama buku ini adalah membekali pembaca dengan pengetahuan dan keterampilan esensial dalam membangun sistem informasi berbasis web menggunakan *framework* CodeIgniter 4 yang ringan, cepat, dan mudah dipelajari.

Ruang lingkup materi dalam buku ini mencakup dasar-dasar pengembangan web, arsitektur MVC, instalasi CodeIgniter 4, operasi CRUD, autentikasi, visualisasi data, hingga deployment aplikasi ke server produksi. Setiap bab dirancang secara sistematis dengan pendekatan *step-by-step* dan contoh kasus yang relevan, memastikan pembaca dapat mengikuti alur pembelajaran dengan mudah. Target pembaca utama adalah mahasiswa, praktisi, maupun siapa saja yang tertarik mendalami pengembangan aplikasi web dengan CodeIgniter 4.

Struktur buku yang disajikan sangat komprehensif, dimulai dengan pengenalan sistem informasi berbasis web, arsitektur MVC, hingga instalasi dan konfigurasi CodeIgniter 4. Penulis dengan cermat membahas setiap aspek penting, termasuk dasar-dasar Model, Controller, dan View, perancangan database, operasi CRUD lengkap, autentikasi pengguna, hingga visualisasi data dan deployment aplikasi. Keunggulan buku ini terletak pada pendekatan *hands-on* yang kuat, di mana setiap bab dilengkapi dengan praktik dan studi kasus yang aplikatif, seperti pembuatan modul CRUD data master, laporan PDF/Excel, hingga dashboard grafik. Buku ini merupakan sumber daya yang sangat berharga bagi mahasiswa, pengembang pemula, maupun profesional yang ingin menguasai CodeIgniter 4. Dengan gaya penulisan yang jelas dan mudah dipahami, serta contoh kode yang relevan, buku ini akan menjadi teman setia dalam perjalanan Anda membangun aplikasi web yang tangguh dan inovatif. Saya berharap buku ini dapat memberikan kontribusi signifikan dalam meningkatkan kualitas pendidikan dan praktik pengembangan aplikasi web di Indonesia.

Saya mengucapkan terima kasih yang sebesar-besarnya kepada keluarga, rekan-rekan sejawat, serta seluruh pihak yang telah memberikan dukungan dan inspirasi selama proses penulisan buku ini. Semoga buku ini dapat menjadi referensi yang bermanfaat, memicu semangat inovasi, dan memberikan kontribusi positif dalam pengembangan sumber daya manusia di bidang teknologi informasi.

Semarang, November 2025
Penulis

Moh Muthohir, S.Kom., M.Kom

DAFTAR ISI

BAB 1
PENGANTAR SISTEM INFORMASI BERBASIS WEB

Di era digital yang serba cepat ini, sistem informasi berbasis web telah menjadi tulang punggung operasional bagi hampir setiap organisasi, mulai dari perusahaan multinasional hingga usaha mikro, kecil, dan menengah (UMKM). Kemampuan untuk mengelola, memproses, dan menyajikan data secara efisien melalui antarmuka web bukan lagi sekadar keunggulan kompetitif, melainkan sebuah keharusan. Modul pembelajaran ini dirancang untuk membekali Anda dengan pemahaman mendalam serta keterampilan praktis dalam membangun aplikasi web dinamis, khususnya menggunakan framework CodeIgniter 4.

Bab pertama ini akan menjadi fondasi utama bagi seluruh perjalanan pembelajaran kita. Kita akan memulai dengan meninjau kembali definisi sistem informasi, sebuah konsep yang seringkali dianggap remeh namun memiliki implikasi luas dalam konteks teknologi. Pemahaman yang kuat tentang apa itu sistem informasi akan membantu kita mengidentifikasi mengapa aplikasi web dinamis menjadi sangat krusial bagi organisasi saat ini. Kita akan mengeksplorasi bagaimana kebutuhan akan interaksi real-time, pengelolaan data yang kompleks, dan penyajian informasi yang relevan mendorong adopsi aplikasi web yang lebih canggih.

Selanjutnya, kita akan menyelami konsep dasar data, informasi, dan alur pemrosesannya. Memahami bagaimana data mentah diubah menjadi informasi yang bermakna adalah kunci untuk merancang sistem yang efektif dan efisien. Alur pemrosesan yang baik akan memastikan integritas data dan akurasi informasi yang dihasilkan.

Bagian penting lainnya dalam bab ini adalah pembahasan mengenai peranan framework dalam pengembangan web. Framework, seperti CodeIgniter 4, menyediakan struktur, alat, dan praktik terbaik yang telah teruji, memungkinkan pengembang untuk membangun aplikasi lebih cepat, lebih aman, dan lebih mudah dikelola. Kita akan melihat bagaimana framework dapat menyederhanakan tugas-tugas umum, mengurangi pengulangan kode, dan mempromosikan konsistensi dalam proyek pengembangan.

Terakhir, kita akan secara spesifik membahas mengapa CodeIgniter 4 menjadi pilihan yang sangat cocok untuk pengembangan sistem informasi. Dengan reputasinya yang ringan, cepat, dan memiliki kurva pembelajaran yang relatif landai, CodeIgniter 4 menawarkan keseimbangan antara fleksibilitas dan kemudahan penggunaan, menjadikannya alat yang ideal bagi mahasiswa maupun profesional yang ingin membangun aplikasi web yang robust dan efisien. Bab ini akan meletakkan dasar pemikiran yang kuat sebelum kita melangkah lebih jauh ke dalam detail teknis dan implementasi praktis di bab-bab berikutnya.

A. DEFINISI SISTEM INFORMASI

Sistem informasi (SI) merupakan sebuah konsep fundamental dalam dunia komputasi dan manajemen yang seringkali disalahpahami atau disederhanakan. Secara esensial, sistem informasi dapat didefinisikan sebagai sekumpulan komponen yang saling terkait dan bekerja sama untuk mengumpulkan, memproses, menyimpan, dan mendistribusikan data serta informasi guna mendukung pengambilan keputusan, koordinasi, kontrol, analisis, dan visualisasi dalam suatu organisasi (Laudon & Laudon, 2020). Definisi ini menekankan pada aspek integrasi dan tujuan fungsional dari sistem tersebut. Dalam pandangan modern, sistem informasi dapat dipahami sebagai kombinasi antara teknologi, manusia, dan proses bisnis. Hubungan antara ketiga unsur ini bersifat simbiosis: teknologi menyediakan sarana, manusia mengoperasikan dan mengambil keputusan, sedangkan proses bisnis memberikan konteks

serta arah penggunaan data dan informasi. Tanpa sinergi yang efektif antar unsur tersebut, sistem informasi akan gagal mencapai tujuannya sebagai alat strategis dalam organisasi.

Komponen-komponen utama dari sebuah sistem informasi meliputi:

1. **Manusia (People)**: Pengguna akhir, operator, analis sistem, manajer, dan semua individu yang berinteraksi dengan sistem. Peran manusia sangat krusial dalam menentukan keberhasilan atau kegagalan sebuah sistem. Elemen manusia merupakan komponen paling vital dalam sistem informasi karena mereka menjadi pengguna, pengembang, dan pengendali sistem. Manusia dalam konteks SI mencakup pengguna akhir (*end-user*), analis sistem, manajer TI, serta pimpinan organisasi yang mengambil keputusan berdasarkan informasi yang disajikan. Efektivitas sistem informasi tidak hanya ditentukan oleh kecanggihan teknologinya, tetapi juga oleh keterampilan, motivasi, dan penerimaan pengguna terhadap teknologi tersebut. Konsep *user acceptance* dan *human-computer interaction* menjadi kunci keberhasilan implementasi sistem
2. **Perangkat Keras (Hardware)**: Komponen fisik yang digunakan untuk input, pemrosesan, output, dan penyimpanan data, seperti komputer, server, perangkat jaringan, dan perangkat input/output lainnya. Dalam era komputasi awan (*cloud computing*), batas antara perangkat lokal dan sumber daya komputasi eksternal semakin kabur karena banyak proses kini dilakukan di pusat data terpadu yang diakses melalui jaringan terdistribusi.
3. **Perangkat Lunak (Software)**: Program komputer yang menginstruksikan perangkat keras untuk melakukan tugas-tugas tertentu. Ini mencakup sistem operasi, perangkat lunak aplikasi (seperti aplikasi web), dan perangkat lunak utilitas. Perangkat lunak terdiri atas dua kategori utama: *system software* (seperti sistem operasi dan utilitas) dan *application software* (seperti sistem ERP, aplikasi web, atau modul laporan bisnis). Dalam konteks sistem informasi berbasis web, perangkat lunak dirancang dengan arsitektur *client–server* dan memanfaatkan komponen seperti *framework web*, basis data relasional, serta API yang memungkinkan integrasi lintas platform dan perangkat
4. **Data**: Fakta mentah yang belum terorganisir dan belum memiliki makna kontekstual. Data adalah bahan baku yang akan diproses oleh sistem informasi. Tanpa data yang akurat, konsisten, dan relevan, sistem informasi tidak dapat menghasilkan wawasan yang bernilai. Kualitas data (data quality) menentukan validitas keputusan yang diambil organisasi. Data harus melalui proses *input*, *processing*, hingga *storage* dan *retrieval* sehingga dapat digunakan untuk analitik dan pengambilan keputusan. Dalam era digital, konsep *big data* dan *data-driven decision making* menjadi semakin penting dalam sistem informasi modern.
5. **Prosedur (Procedures)**: Serangkaian langkah-langkah atau aturan yang ditetapkan untuk mengoperasikan sistem, mulai dari cara memasukkan data, menjalankan program, hingga menangani masalah. Adanya prosedur yang jelas memastikan bahwa sistem digunakan secara konsisten dan sesuai dengan tujuan organisasi. Di sisi lain, prosedur juga menjadi landasan utama dalam pengendalian internal (internal control) sebuah system.
6. **Jaringan (Network)**: Infrastruktur komunikasi yang memungkinkan berbagai komponen sistem untuk saling terhubung dan bertukar data, seperti internet, intranet, atau jaringan lokal. Jaringan memungkinkan sistem berbasis web bekerja secara *real-time*, mendukung kolaborasi lintas lokasi, dan memberikan akses informasi kapan saja serta di mana saja.

Dalam konteks aplikasi web, sistem informasi berbasis web berarti bahwa seluruh atau sebagian besar komponen perangkat lunak dan data diakses serta dioperasikan melalui antarmuka web menggunakan protokol HTTP/HTTPS. Ini memungkinkan akses yang luas dari berbagai lokasi dan perangkat, selama ada koneksi internet.

Peran Sistem Informasi dalam Organisasi Modern

Sistem informasi memiliki peran strategis sebagai penghubung antara aspek teknologi dan manajemen. Dalam operasional sehari-hari, SI berfungsi untuk mempercepat aliran informasi, mengefisienkan proses bisnis, dan mendukung pengambilan keputusan berbasis data. Di tingkat manajerial, sistem informasi membantu mengintegrasikan fungsi-fungsi organisasi seperti keuangan, SDM, produksi, dan pemasaran ke dalam satu platform terpadu.

Selain itu, sistem informasi juga menjadi pondasi bagi tumbuhnya berbagai sistem turunan, seperti Management Information System (MIS), Decision Support System (DSS), dan Enterprise Resource Planning (ERP). Sistem-sistem ini memperluas fungsi SI dengan menyediakan fitur analitik, pelaporan otomatis, dan integrasi proses bisnis secara end-to-end.

Sistem Informasi Berbasis Web

Dalam konteks kontemporer, perkembangan internet telah membawa transformasi besar terhadap cara sistem informasi dirancang dan dioperasikan. Sistem informasi berbasis web memungkinkan seluruh atau sebagian besar komponen perangkat lunak dan data diakses melalui antarmuka web dengan menggunakan protokol HTTP atau HTTPS. Model ini membuat penggunaan sistem menjadi lebih fleksibel dan skalabel, karena pengguna dapat mengaksesnya melalui berbagai perangkat—mulai dari komputer desktop hingga ponsel pintar.

Selain itu, sistem informasi web bersifat terdistribusi dan mendukung integrasi antar aplikasi melalui API dan layanan berbasis REST atau GraphQL. Keunggulan lainnya meliputi kemudahan pemeliharaan, pembaruan fitur secara daring, serta efisiensi biaya implementasi. Namun demikian, model ini juga menghadirkan tantangan baru dalam hal keamanan data, privasi, serta manajemen akses pengguna.

Dengan demikian, sistem informasi bukan sekadar teknologi, tetapi merupakan ekosistem yang mengintegrasikan manusia, perangkat, data, dan proses bisnis untuk menciptakan nilai tambah bagi organisasi. Dalam era digital yang kompetitif, keberhasilan organisasi modern semakin bergantung pada kemampuan mereka dalam merancang, mengelola, dan mengembangkan sistem informasi yang adaptif, cerdas, dan berorientasi pada data

B. KEBUTUHAN APLIKASI DINAMIS PADA ORGANISASI

Di era digital saat ini, organisasi menghadapi tantangan dan peluang yang kompleks, menuntut adaptasi cepat dan efisiensi operasional. Aplikasi web statis, yang hanya menyajikan konten tetap tanpa interaksi pengguna atau pembaruan data real-time, tidak lagi memadai untuk memenuhi kebutuhan tersebut. Oleh karena itu, kebutuhan akan aplikasi dinamis menjadi sangat krusial.

Aplikasi dinamis adalah aplikasi yang kontennya dapat berubah berdasarkan interaksi pengguna, data dari database, atau kondisi lainnya. Karakteristik utama aplikasi dinamis meliputi:

1. **Interaktivitas**: Pengguna dapat berinteraksi dengan aplikasi, seperti mengisi formulir, melakukan pencarian, mengunggah file, atau memberikan komentar.

2. **Personalisasi Konten**: Konten yang disajikan dapat disesuaikan untuk setiap pengguna berdasarkan preferensi, riwayat, atau peran mereka.
3. **Manajemen Data Real-time**: Aplikasi dapat mengambil, menyimpan, memperbarui, dan menghapus data dari database secara *real-time*, memastikan informasi yang disajikan selalu terkini.
4. **Fungsionalitas Kompleks**: Mampu menjalankan logika bisnis yang kompleks, seperti perhitungan, validasi data, atau alur kerja multi-langkah.
5. **Skalabilitas**: Lebih mudah untuk dikembangkan dan diperluas fungsionalitasnya seiring dengan pertumbuhan kebutuhan organisasi.

Kebutuhan organisasi terhadap aplikasi dinamis dapat diilustrasikan melalui beberapa contoh:

- **Sistem E-commerce**: Memungkinkan pelanggan untuk menelusuri katalog produk yang selalu diperbarui, menambahkan item ke keranjang belanja, melakukan pembayaran, dan melacak pesanan mereka. Tanpa aplikasi dinamis, toko online tidak akan berfungsi.
- **Sistem Manajemen Sumber Daya Manusia (HRIS)**: Karyawan dapat mengakses slip gaji, mengajukan cuti, memperbarui informasi pribadi, atau melihat jadwal pelatihan. Manajer dapat menyetujui permintaan dan mengelola data karyawan secara efisien.
- **Sistem Informasi Akademik (SIAKAD)**: Mahasiswa dapat melihat nilai, jadwal kuliah, mendaftar mata kuliah, dan mengunduh materi. Dosen dapat mengunggah nilai, materi, dan mengelola absensi.
- **Sistem Manajemen Konten (CMS)**: Memungkinkan administrator untuk membuat, mengedit, dan mempublikasikan konten (artikel, berita, halaman) tanpa perlu mengubah kode program secara manual.

Aplikasi dinamis memungkinkan organisasi untuk meningkatkan efisiensi operasional, meningkatkan pengalaman pelanggan, membuat keputusan yang lebih baik berdasarkan data terkini, dan tetap kompetitif di pasar yang terus berubah (O'Brien & Marakas, 2011).

C. KONSEP DATA, INFORMASI, DAN ALUR PEMROSESAN

Memahami perbedaan antara data dan informasi, serta bagaimana data diubah menjadi informasi melalui alur pemrosesan, adalah inti dari perancangan sistem informasi yang efektif.

1. Data

Data adalah fakta mentah, angka, teks, gambar, atau suara yang belum diorganisir dan belum memiliki makna kontekstual. Data bersifat objektif dan belum diinterpretasikan.

- **Contoh Data**:
  - Angka 101
  - Nama Budi Santoso
  - Tanggal 2023-10-26
  - Teks Laptop ASUS
  - Harga Rp 12.500.000

2. Informasi

Informasi adalah data yang telah diproses, diorganisir, distrukturkan, dan disajikan dalam konteks yang memberikan makna dan relevansi bagi penerimanya. Informasi digunakan untuk pengambilan keputusan.

- **Contoh Informasi**:

- Mahasiswa dengan NIM 101 adalah Budi Santoso, yang mendaftar pada tanggal 26 Oktober 2023.
- Produk Laptop ASUS memiliki harga jual Rp 12.500.000.

Perbedaan mendasar adalah bahwa data adalah input, sedangkan informasi adalah output yang bermakna.

3. Alur Pemrosesan Data

Alur pemrosesan data adalah serangkaian langkah-langkah yang mengubah data mentah menjadi informasi yang berguna. Dalam sistem informasi berbasis web, alur ini biasanya melibatkan beberapa tahapan:

a. **Input Data**: Data dikumpulkan dari berbagai sumber, seperti formulir web yang diisi pengguna, sensor, atau sistem lain. Data ini kemudian dimasukkan ke dalam sistem. * **Contoh**: Pengguna mengisi formulir pendaftaran dengan nama, email, dan password.

b. **Penyimpanan Data**: Data yang telah diinput disimpan dalam media penyimpanan yang terstruktur, umumnya database. Database dirancang untuk menyimpan data secara efisien dan aman, serta memungkinkan pengambilan data yang cepat. * **Contoh**: Data pendaftaran pengguna disimpan dalam tabel users di database.

c. **Pemrosesan Data**: Data yang tersimpan kemudian diolah atau dimanipulasi sesuai dengan logika bisnis yang telah ditentukan. Pemrosesan ini bisa berupa perhitungan, pengurutan, penyaringan, penggabungan, atau validasi. * **Contoh**: Sistem memvalidasi format email, mengenkripsi password, dan memeriksa apakah username sudah ada. Setelah itu, sistem mungkin mengaitkan pengguna baru dengan peran default (misalnya, anggota).

d. **Output Informasi**: Hasil dari pemrosesan data disajikan kepada pengguna dalam format yang mudah dipahami dan relevan. Output bisa berupa laporan, grafik, tabel, atau tampilan antarmuka pengguna lainnya. * **Contoh**: Setelah pendaftaran berhasil, pengguna melihat halaman profil mereka dengan informasi yang telah dimasukkan, atau menerima email konfirmasi. Administrator mungkin melihat daftar pengguna baru dalam laporan harian.

e. **Umpan Balik (Feedback)**: Informasi yang dihasilkan dapat memicu tindakan atau keputusan baru, yang kemudian menghasilkan input data baru, sehingga membentuk siklus berkelanjutan. * **Contoh**: Administrator melihat laporan pendaftaran pengguna baru dan memutuskan untuk mengirimkan email selamat datang khusus kepada mereka.

Alur pemrosesan yang terstruktur dan efisien sangat penting untuk memastikan integritas data, akurasi informasi, dan kinerja sistem secara keseluruhan (Kroenke & Boyle, 2019).

D. PERANAN FRAMEWORK DALAM PENGEMBANGAN WEB

Pengembangan aplikasi web modern seringkali merupakan tugas yang kompleks, melibatkan banyak aspek mulai dari manajemen database, penanganan *request* HTTP, validasi input, hingga keamanan. Untuk menyederhanakan proses ini, *framework* pengembangan web memainkan peranan yang sangat penting.

Framework adalah kerangka kerja perangkat lunak yang menyediakan struktur dasar untuk membangun aplikasi. Ini mencakup pustaka kode, alat, dan praktik terbaik yang telah teruji, yang membantu pengembang untuk membangun aplikasi lebih cepat, lebih efisien, dan dengan kualitas yang lebih tinggi.

Peranan utama *framework* dalam pengembangan web meliputi:

1. **Percepatan Pengembangan**: *Framework* menyediakan komponen siap pakai untuk tugas-tugas umum seperti autentikasi, validasi formulir, dan interaksi database. Ini

mengurangi kebutuhan untuk menulis kode dari awal, sehingga mempercepat waktu pengembangan secara signifikan.

2. **Struktur dan Konsistensi**: *Framework* seringkali menerapkan pola desain tertentu, seperti Model-View-Controller (MVC), yang mempromosikan pemisahan kekhawatiran (separation of concerns). Ini menghasilkan kode yang lebih terorganisir, mudah dipelihara, dan konsisten di seluruh proyek.
3. **Keamanan yang Ditingkatkan**: Banyak *framework* modern telah mengimplementasikan fitur keamanan bawaan untuk melindungi aplikasi dari ancaman umum seperti *SQL injection*, *Cross-Site Scripting (XSS)*, dan *Cross-Site Request Forgery (CSRF)*. Ini membantu pengembang membangun aplikasi yang lebih aman tanpa harus menjadi ahli keamanan.
4. **Pemeliharaan yang Lebih Mudah**: Dengan struktur yang jelas dan konvensi yang ditetapkan, kode yang ditulis menggunakan *framework* cenderung lebih mudah dipahami dan dipelihara oleh pengembang lain, bahkan jika mereka tidak terlibat dalam pengembangan awal.
5. **Skalabilitas**: *Framework* dirancang untuk mendukung pengembangan aplikasi dari skala kecil hingga besar. Struktur modularnya memungkinkan penambahan fitur baru tanpa mengganggu fungsionalitas yang sudah ada.
6. **Komunitas dan Ekosistem**: *Framework* populer memiliki komunitas pengembang yang besar dan aktif, serta ekosistem yang kaya akan *plugin*, *library*, dan dokumentasi. Ini memberikan dukungan yang berharga bagi pengembang saat menghadapi masalah atau mencari solusi.
7. **Praktik Terbaik (Best Practices)**: *Framework* mendorong penggunaan praktik terbaik dalam pengembangan perangkat lunak, seperti *Don't Repeat Yourself (DRY)*, *Convention Over Configuration*, dan *Test-Driven Development (TDD)*, yang mengarah pada kode yang lebih bersih dan robust.

Tanpa *framework*, pengembang harus membangun setiap komponen dari nol, yang memakan waktu, rentan terhadap kesalahan, dan sulit untuk dipelihara. Oleh karena itu, penggunaan *framework* telah menjadi standar industri dalam pengembangan aplikasi web modern (Freeman & Robson, 2019).

E. MENGAPA CODEIGNITER 4 COCOK UNTUK SISTEM INFORMASI

CodeIgniter telah lama dikenal sebagai salah satu *framework* PHP yang ringan dan cepat, dan CodeIgniter 4 (CI4) melanjutkan tradisi tersebut dengan membawa banyak peningkatan modern. Ada beberapa alasan kuat mengapa CodeIgniter 4 sangat cocok untuk pengembangan sistem informasi berbasis web:

1. **Ringan dan Cepat (Lightweight and Fast)**: CI4 dirancang untuk memiliki *footprint* yang kecil dan kinerja yang optimal. Ini berarti aplikasi yang dibangun dengan CI4 akan memuat lebih cepat dan merespons lebih responsif, sebuah faktor krusial untuk sistem informasi yang sering diakses dan memproses banyak data.
2. **Kurva Pembelajaran yang Landai (Gentle Learning Curve)**: Bagi pengembang yang baru mengenal *framework* PHP, CI4 menawarkan kurva pembelajaran yang relatif landai dibandingkan *framework* lain yang lebih kompleks. Dokumentasinya yang komprehensif dan jelas, serta sintaksis yang intuitif, memungkinkan pengembang untuk cepat produktif.
3. **Fleksibilitas dan Konfigurasi Minimal (Flexibility and Minimal Configuration)**: CI4 menganut prinsip *Convention Over Configuration*, namun tetap memberikan

fleksibilitas yang cukup bagi pengembang untuk menyesuaikan struktur dan perilaku aplikasi sesuai kebutuhan. Ini mengurangi waktu yang dihabiskan untuk konfigurasi awal dan memungkinkan fokus pada logika bisnis.

4. **Pola MVC yang Jelas (Clear MVC Pattern)**: CI4 secara ketat menerapkan pola arsitektur Model-View-Controller (MVC). Pemisahan yang jelas antara logika bisnis (Model), presentasi (View), dan penanganan *request* (Controller) membuat kode lebih terorganisir, mudah dipelihara, dan kolaboratif dalam tim pengembangan.
5. **Fitur Keamanan Bawaan (Built-in Security Features)**: CI4 menyediakan berbagai fitur keamanan untuk melindungi aplikasi dari ancaman umum, termasuk *CSRF protection*, *XSS filtering*, *input validation*, dan *password hashing*. Fitur-fitur ini sangat penting untuk sistem informasi yang menangani data sensitif.
6. **Dukungan Database yang Kuat (Robust Database Support)**: CI4 memiliki *Database Abstraction Layer* yang kuat, memungkinkan pengembang untuk berinteraksi dengan berbagai jenis database (MySQL, PostgreSQL, SQLite, SQLSRV) menggunakan sintaksis yang konsisten. Fitur *Query Builder* juga menyederhanakan penulisan query database yang kompleks.
7. **Ekosistem yang Berkembang (Growing Ecosystem)**: Meskipun tidak sebesar beberapa *framework* lain, CI4 memiliki komunitas yang aktif dan terus berkembang. Ini berarti ada banyak sumber daya, *plugin*, dan bantuan yang tersedia jika pengembang menghadapi masalah.
8. **Alat Pengembangan yang Berguna (Useful Development Tools)**: CI4 dilengkapi dengan berbagai *helper* dan *library* yang mempercepat pengembangan, seperti *Form Helper*, *URL Helper*, *Validation Library*, dan *Email Library*. Fitur *Migrations* dan *Seeder* juga sangat membantu dalam manajemen database.
9. **Cocok untuk Aplikasi Skala Menengah (Suitable for Medium-Scale Applications)**: Untuk sistem informasi dengan kompleksitas menengah, CI4 menawarkan keseimbangan yang sangat baik antara fitur, kinerja, dan kemudahan penggunaan. Ini menjadikannya pilihan ideal untuk membangun aplikasi seperti sistem manajemen inventori, HRIS, SIAKAD, atau CRM sederhana.

Dengan kombinasi kecepatan, kemudahan penggunaan, fitur keamanan, dan struktur yang terorganisir, CodeIgniter 4 menyediakan fondasi yang solid dan efisien untuk membangun sistem informasi berbasis web yang robust dan dapat diandalkan (CodeIgniter, 2023).

F. RANGKUMAN

Bab 1 ini telah menguraikan fondasi penting dalam memahami sistem informasi dan pengembangan aplikasi web dinamis, khususnya dengan fokus pada penggunaan *framework* CodeIgniter 4. Pembahasan dimulai dengan mendefinisikan sistem informasi sebagai kumpulan komponen terintegrasi yang berfungsi untuk mengumpulkan, memproses, menyimpan, dan mendistribusikan data guna mendukung pengambilan keputusan dalam organisasi. Komponen-komponen kunci sistem informasi meliputi manusia, perangkat keras, perangkat lunak, data, prosedur, dan jaringan, yang semuanya bekerja sama untuk mencapai tujuan fungsional.

Selanjutnya, bab ini menyoroti pergeseran kebutuhan organisasi dari aplikasi statis ke aplikasi dinamis di era digital. Aplikasi dinamis, dengan karakteristik interaktivitas, personalisasi konten, manajemen data *real-time*, fungsionalitas kompleks, dan skalabilitas, menjadi esensial untuk meningkatkan efisiensi operasional dan daya saing. Contoh-contoh

seperti sistem *e-commerce*, HRIS, dan SIAKAD memperjelas urgensi aplikasi dinamis dalam berbagai konteks organisasi.

Pemahaman mendalam tentang konsep data, informasi, dan alur pemrosesannya juga menjadi fokus utama. Data didefinisikan sebagai fakta mentah yang belum terorganisir, sedangkan informasi adalah data yang telah diproses dan memiliki makna kontekstual. Alur pemrosesan data, yang mencakup input, penyimpanan, pemrosesan, output, dan umpan balik, merupakan siklus krusial yang mengubah data mentah menjadi informasi yang berguna untuk pengambilan keputusan.

Peran *framework* dalam pengembangan web modern ditekankan sebagai solusi untuk mengatasi kompleksitas. *Framework* mempercepat pengembangan, memastikan struktur dan konsistensi kode, meningkatkan keamanan, mempermudah pemeliharaan, mendukung skalabilitas, serta mendorong praktik terbaik melalui komunitas dan ekosistem yang kuat.

Terakhir, bab ini menjelaskan mengapa CodeIgniter 4 merupakan pilihan yang cocok untuk pengembangan sistem informasi. Keunggulan CI4 terletak pada sifatnya yang ringan dan cepat, kurva pembelajaran yang landai, fleksibilitas, penerapan pola MVC yang jelas, fitur keamanan bawaan, dukungan database yang kuat, ekosistem yang berkembang, serta alat pengembangan yang berguna. Kombinasi fitur-fitur ini menjadikan CI4 fondasi yang solid dan efisien untuk membangun sistem informasi berbasis web yang robust dan dapat diandalkan, terutama untuk aplikasi skala menengah.

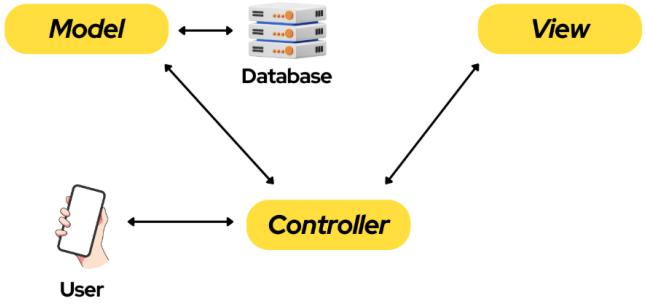
BAB 2
MENGENAL CODEIGNITER 4

A. ARSITEKTUR MVC

Arsitektur Model–View–Controller (MVC) adalah salah satu pola desain perangkat lunak yang paling berpengaruh dalam sejarah pengembangan aplikasi modern. Pola ini memecah sistem perangkat lunak menjadi tiga komponen utama yang saling berinteraksi: Model, View, dan Controller. Masing-masing komponen memiliki tanggung jawab yang terpisah, tetapi tetap berkolaborasi untuk menciptakan aplikasi yang dinamis, terstruktur, dan mudah dikelola.

MVC pertama kali diperkenalkan oleh Trygve Reenskaug pada akhir tahun 1970-an ketika ia bekerja dengan bahasa pemrograman Smalltalk-76 di Xerox PARC. Tujuan awalnya ialah memperbaiki cara pengembang merancang antarmuka pengguna pada sistem berbasis GUI (Graphical User Interface). Sejak saat itu, konsep MVC berkembang pesat dan diadopsi secara luas dalam berbagai bahasa pemrograman dan platform — mulai dari framework desktop seperti Qt hingga framework web modern seperti Ruby on Rails, Laravel, dan CodeIgniter 4.

Pola ini menjadi fondasi penting dalam pengembangan aplikasi web dinamis, karena mampu memisahkan logika aplikasi dan tampilan secara jelas. Dengan struktur seperti ini, pengembang dapat dengan mudah menambah fitur, mengganti tampilan, atau menyesuaikan logika bisnis tanpa harus mengubah keseluruhan sistem



Gambar 2. 1 Arsitektur MVC (Model-View-Controller)

1. Model (M)

Komponen Model bertanggung jawab atas representasi data dan logika bisnis aplikasi. Ini adalah bagian yang berinteraksi langsung dengan *database* atau sumber data lainnya. Tugas utama Model meliputi:

- **Manajemen Data**: Mengambil, menyimpan, memperbarui, dan menghapus data dari *database*.
- **Validasi Data**: Memastikan integritas dan konsistensi data sebelum disimpan atau diproses.
- **Logika Bisnis**: Mengandung aturan-aturan bisnis yang mengatur bagaimana data dapat dimanipulasi atau digunakan. Misalnya, perhitungan harga, validasi stok, atau aturan diskon.
- **Abstraksi Database**: Menyediakan antarmuka yang lebih tinggi untuk berinteraksi dengan *database*, sehingga *controller* tidak perlu tahu detail implementasi *database* yang mendasarinya.

Dalam CodeIgniter 4, Model biasanya direpresentasikan sebagai kelas PHP yang mewarisi dari CodeIgniter\Model. Kelas ini menyediakan berbagai *method helper* untuk operasi CRUD (Create, Read, Update, Delete) yang memudahkan interaksi dengan *database*.

2. View (V)

Komponen View bertanggung jawab atas presentasi data kepada pengguna. Ini adalah antarmuka pengguna (UI) dari aplikasi. View tidak mengandung logika bisnis atau interaksi langsung dengan *database*; tugasnya murni untuk menampilkan informasi yang diterima dari *controller*.

- **Tampilan Data**: Merender data yang telah disiapkan oleh *controller* ke dalam format yang dapat dipahami oleh pengguna, seperti HTML, XML, atau JSON.
- **Desain Antarmuka**: Mengandung elemen-elemen visual seperti *form*, tabel, tombol, dan *layout* halaman.
- **Pemisahan Logika**: Memastikan bahwa logika presentasi terpisah dari logika bisnis dan logika kontrol, sehingga desainer *front-end* dapat bekerja pada View tanpa mengganggu logika *back-end*.

Dalam CodeIgniter 4, View biasanya berupa file PHP atau HTML yang berisi *markup* dan mungkin sedikit logika PHP untuk menampilkan data secara dinamis.

3. Controller (C)

Komponen Controller bertindak sebagai perantara antara Model dan View. Ini adalah otak dari aplikasi yang menerima permintaan dari pengguna, memprosesnya, berinteraksi dengan Model untuk mendapatkan atau memanipulasi data, dan kemudian memilih View yang tepat untuk menampilkan hasilnya.

- **Menerima Permintaan**: Mendengarkan permintaan HTTP dari *browser* pengguna.
- **Memproses Input**: Menganalisis data yang dikirimkan oleh pengguna (misalnya, dari *form*).
- **Berinteraksi dengan Model**: Memanggil *method* pada Model untuk melakukan operasi data atau menerapkan logika bisnis.
- **Memilih View**: Menentukan View mana yang akan digunakan untuk menampilkan respons kepada pengguna, dan meneruskan data yang relevan ke View tersebut.
- **Mengelola Alur Aplikasi**: Mengarahkan alur eksekusi aplikasi berdasarkan permintaan pengguna.

Dalam CodeIgniter 4, Controller adalah kelas PHP yang mewarisi dari CodeIgniter\Controller. Setiap *method* publik dalam kelas *controller* dapat diakses melalui URL tertentu, bertindak sebagai *action* yang merespons permintaan pengguna.

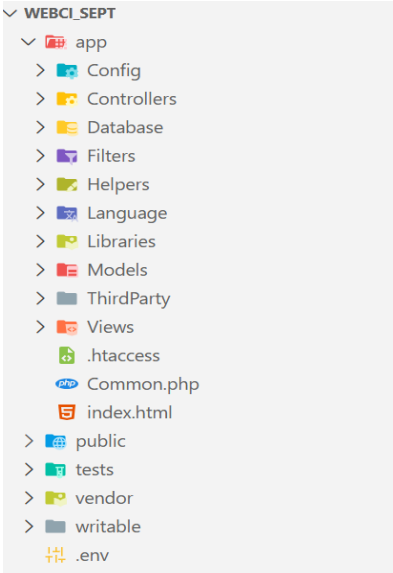
4. Manfaat Arsitektur MVC

Penerapan MVC dalam CodeIgniter 4 membawa beberapa manfaat signifikan:

- **Pemisahan Tanggung Jawab (Separation of Concerns)**: Setiap komponen memiliki tugas yang jelas, membuat kode lebih terorganisir dan mudah dipahami.
- **Modularitas**: Komponen dapat dikembangkan dan diuji secara independen.
- **Kemudahan Pemeliharaan**: Perubahan pada satu bagian aplikasi (misalnya, perubahan *database* di Model atau perubahan desain di View) cenderung tidak mempengaruhi bagian lain.
- **Skalabilitas**: Memudahkan penambahan fitur baru atau perluasan aplikasi.
- **Kolaborasi Tim**: Memungkinkan pengembang *back-end* (fokus pada Model dan Controller) dan desainer *front-end* (fokus pada View) untuk bekerja secara paralel.

B. STRUKTUR FOLDER CODEIGNITER 4

Memahami struktur folder CodeIgniter 4 adalah langkah fundamental untuk dapat mengembangkan aplikasi secara efisien. Struktur ini dirancang untuk memisahkan kode inti *framework* dari kode aplikasi Anda, serta memisahkan berbagai aspek aplikasi ke dalam direktori yang logis. Berikut adalah penjelasan rinci mengenai direktori dan file penting dalam instalasi CodeIgniter 4 (CodeIgniter Documentation, n.d.):



Gambar 2. 2 Struktur Folder CodeIgniter 4

1. Direktori Utama

- **app/**: Ini adalah direktori paling penting bagi pengembang. Semua kode aplikasi Anda akan berada di sini, termasuk *controller*, *model*, *view*, *library*, *helper*, dan konfigurasi spesifik aplikasi.
    - **app/Config/**: Berisi semua file konfigurasi aplikasi, seperti pengaturan *database*, *routing*, *filters*, dan *autoload*.
    - **app/Controllers/**: Tempat Anda menempatkan semua kelas *controller* yang menangani permintaan HTTP dan mengelola alur aplikasi.
    - **app/Database/**: Berisi file-file terkait *database*, seperti *migrations* (untuk mengelola skema *database*) dan *seeders* (untuk mengisi data awal).
    - **app/Filters/**: Berisi kelas *filter* yang dapat digunakan untuk memproses permintaan sebelum mencapai *controller* atau respons sebelum dikirim ke *browser* (misalnya, autentikasi, *logging*).
    - **app/Helpers/**: Berisi fungsi-fungsi *helper* kustom yang dapat digunakan di seluruh aplikasi untuk tugas-tugas umum.
    - **app/Language/**: Digunakan untuk menyimpan file terjemahan jika aplikasi mendukung multibahasa.
    - **app/Libraries/**: Berisi kelas *library* kustom yang dapat digunakan kembali di berbagai bagian aplikasi.
    - **app/Models/**: Tempat Anda menempatkan semua kelas *model* yang berinteraksi dengan *database* dan mengelola logika bisnis.
    - **app/ThirdParty/**: Digunakan untuk menyimpan *library* pihak ketiga yang tidak diinstal melalui Composer.
    - **app/Views/**: Berisi semua file *view* (template HTML/PHP) yang digunakan untuk menampilkan antarmuka pengguna.

- **public/**: Ini adalah *document root* dari aplikasi web Anda. Semua permintaan HTTP akan masuk melalui direktori ini.
    - **index.php**: File *bootstrap* utama CodeIgniter 4. Semua permintaan akan diarahkan ke file ini.
    - **assets/**: (Opsional, sering dibuat secara manual) Digunakan untuk menyimpan file statis seperti CSS, JavaScript, gambar, dan *font*.
- **system/**: Berisi inti *framework* CodeIgniter 4. **Anda tidak boleh memodifikasi file di direktori ini**. Pembaruan *framework* akan mengganti seluruh direktori ini.
- **writable/**: Direktori ini memerlukan izin tulis oleh server web. Digunakan untuk menyimpan data yang dihasilkan oleh aplikasi, seperti *cache*, *log*, dan *uploads*.
    - **writable/cache/**: Untuk menyimpan file *cache* aplikasi.
    - **writable/logs/**: Untuk menyimpan *log* kesalahan dan informasi lainnya.
    - **writable/uploads/**: (Opsional) Sering digunakan untuk menyimpan file yang diunggah oleh pengguna.
- **tests/**: Berisi semua *unit test* dan *integration test* untuk aplikasi Anda.
- **vendor/**: Direktori ini dibuat dan dikelola oleh Composer. Berisi semua *dependency* pihak ketiga yang diinstal untuk proyek Anda.

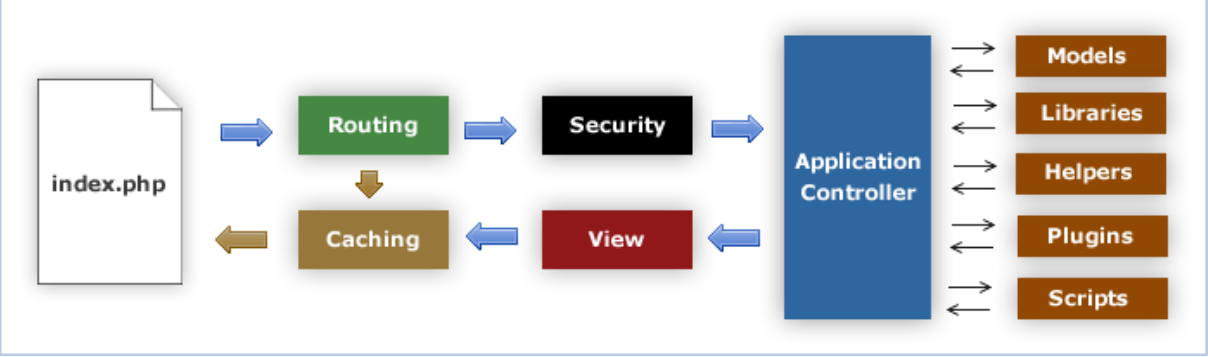
2. File Penting di Root Proyek

- **.env**: File konfigurasi lingkungan. Digunakan untuk menyimpan variabel lingkungan yang berbeda antara lingkungan pengembangan dan produksi (misalnya, kredensial *database*, mode debug).
- **composer.json**: File konfigurasi Composer yang mendefinisikan *dependency* proyek dan *script* Composer.
- **spark**: *Command-line tool* CodeIgniter 4 yang digunakan untuk berbagai tugas seperti membuat *controller*, *model*, *migration*, menjalankan *server* pengembangan, dan banyak lagi.

Memahami struktur ini akan membantu Anda menempatkan kode di lokasi yang benar, memudahkan kolaborasi, dan memastikan aplikasi Anda tetap terorganisir seiring pertumbuhannya.

C. PROSES REQUEST–RESPONSE

Memahami bagaimana CodeIgniter 4 memproses permintaan dari *browser* pengguna dan menghasilkan respons adalah kunci untuk mendalami alur kerja *framework* ini. Proses ini dikenal sebagai request-*response cycle* atau siklus permintaan-respons. Secara umum, ketika pengguna mengakses URL aplikasi CodeIgniter 4, serangkaian langkah terjadi secara berurutan (CodeIgniter Documentation, n.d.):



Gambar 2. 3 Proses Request-Response CodeIgniter 4

1. Permintaan Klien (Client Request)

Proses dimulai ketika pengguna mengetik URL di *browser* atau mengklik *link*. *Browser* kemudian mengirimkan permintaan HTTP (misalnya, GET, POST) ke server web.

2. Server Web dan public/index.php

Server web (misalnya, Apache atau Nginx) menerima permintaan. Berkat konfigurasi *rewrite rule* (biasanya di file .htaccess atau konfigurasi Nginx), semua permintaan diarahkan ke file public/index.php. File index.php ini adalah *front controller* atau *bootstrap* utama CodeIgniter4.

3. Inisialisasi CodeIgniter

Di dalam public/index.php, CodeIgniter melakukan inisialisasi awal:

- Memuat file .env untuk mengonfigurasi variabel lingkungan.
- Menentukan lingkungan aplikasi (development, production, testing).
- Memuat *autoloader* Composer dan *autoloader* CodeIgniter untuk memastikan semua kelas yang dibutuhkan dapat ditemukan.
- Membuat instance dari objek Request dan Response.

4. Routing

Setelah inisialisasi, *router* CodeIgniter mengambil alih. *Router* bertanggung jawab untuk menganalisis URL yang diminta dan memetakannya ke *controller* dan *method* (action) yang sesuai.

- *Router* membaca konfigurasi *routing* yang didefinisikan dalam app/Config/Routes.php.
- Mencocokkan URL dengan pola *route* yang telah ditentukan.
- Jika ditemukan kecocokan, *router* menentukan *controller* dan *method* mana yang harus dieksekusi. Jika tidak ada kecocokan, biasanya akan mengarahkan ke halaman 404 Not Found.

5. Filters (Middleware)

Sebelum *controller* dieksekusi, *request* mungkin melewati satu atau lebih *filters* (sering disebut *middleware* di *framework* lain). *Filters* adalah kelas yang dapat memodifikasi *request* atau *response*, atau bahkan menghentikan eksekusi *request* sama sekali (misalnya, untuk autentikasi, otorisasi, *logging*, atau *rate limiting*). *Filters* didefinisikan dalam app/Config/Filters.php dan dapat diterapkan ke *route* tertentu atau seluruh aplikasi.

6. Eksekusi Controller

Setelah melewati *filters* (jika ada), *controller* yang ditentukan oleh *router* akan dieksekusi.

- *Controller* menerima objek Request yang berisi semua data permintaan (input *form*, *header*, *cookie*, dll.).
- *Controller* kemudian menjalankan logika bisnis yang relevan:
    - Memanggil *method* pada *Model* untuk berinteraksi dengan *database* (mengambil, menyimpan, memperbarui data).
    - Memproses data input.
    - Menerapkan logika bisnis lainnya.
- Setelah semua pemrosesan selesai, *controller* menyiapkan data yang akan ditampilkan.

7. Render View

Controller kemudian memuat *View* yang sesuai dan meneruskan data yang telah disiapkan ke *View* tersebut.

- *View* menggunakan data yang diterima untuk menghasilkan *output* HTML (atau format lain) yang akan dikirim kembali ke *browser*.

- *View* tidak boleh mengandung logika bisnis yang kompleks; tugasnya murni untuk presentasi.

8. Filters (Post-Controller)

Setelah *View* dirender, *response* mungkin melewati *filters* lagi (jika ada *filters* yang dikonfigurasi untuk berjalan setelah *controller*). Ini bisa digunakan untuk memodifikasi *response* sebelum dikirim, seperti menambahkan *header* HTTP, mengompresi *output*, atau *logging* informasi *response*.

9. Respons Klien (Client Response)

Akhirnya, *response* HTTP yang telah selesai (berisi HTML, *header*, *cookie*, dll.) dikirim kembali ke *browser* pengguna. *Browser* kemudian merender HTML dan menampilkan halaman web kepada pengguna.

Siklus ini berulang setiap kali pengguna berinteraksi dengan aplikasi, memastikan alur yang terstruktur dan dapat diprediksi.

D. KONSEP ROUTING DASAR DAN LANJUTAN

Routing adalah mekanisme penting dalam aplikasi web yang menentukan bagaimana URL yang diminta oleh pengguna dipetakan ke *controller* dan *method* yang sesuai dalam aplikasi. CodeIgniter 4 menyediakan sistem *routing* yang fleksibel dan kuat, memungkinkan pengembang untuk mendefinisikan URL yang bersih dan mudah diingat (CodeIgniter Documentation, n.d.). Konfigurasi *routing* utama berada di file app/Config/Routes.php.

1. Routing Dasar

Routing dasar melibatkan pemetaan URL statis atau URL dengan *placeholder* sederhana ke *controller* dan *method* tertentu.

a. Default Route

Secara *default*, CodeIgniter 4 memiliki *route* bawaan yang mengarahkan permintaan ke *controller* Home dan *method* index().

// app/Config/Routes.php $routes->get('/', 'Home::index');

Ini berarti ketika pengguna mengakses *base URL* aplikasi (/), *method* index() dari HomeController akan dieksekusi.

b. Route Statis

Anda dapat mendefinisikan *route* untuk URL statis yang spesifik.

```
$routes->get('/about', 'Pages::about');
$routes->post('/contact', 'Contact::submitForm');
```

- $routes->get('/about', 'Pages::about'); akan memetakan permintaan GET ke URL /about ke *method* about() di PagesController.
- $routes->post('/contact', 'Contact::submitForm'); akan memetakan permintaan POST ke URL /contact ke *method* submitForm() di ContactController.

CodeIgniter mendukung berbagai *HTTP verb* seperti get(), post(), put(), delete(), options(), patch(), dan head(). Anda juga bisa menggunakan match() untuk beberapa *verb* atau any() untuk semua *verb*.

c. Route dengan Placeholder

Untuk URL yang dinamis, Anda dapat menggunakan *placeholder* untuk menangkap segmen URL sebagai parameter yang akan diteruskan ke *method controller*.

- **(:any)**: Mencocokkan segmen URL apa pun.
- **(:num)**: Mencocokkan segmen URL yang hanya berisi angka.
- **(:alpha)**: Mencocokkan segmen URL yang hanya berisi huruf.
- **(:alphanum)**: Mencocokkan segmen URL yang berisi huruf dan angka.

- **(:segment)**: Mencocokkan segmen URL apa pun kecuali /.

Contoh:

```
$routes->get('/users/(:num)', 'Users::show/$1');
$routes->get('/blog/(:any)', 'Blog::viewPost/$1');
```

- $routes->get('/users/(:num)', 'Users::show/$1'); akan memetakan /users/123 ke UsersController::show(123). $1 merujuk pada nilai dari *placeholder* pertama.
- $routes->get('/blog/(:any)', 'Blog::viewPost/$1'); akan memetakan /blog/judul-artikel-saya ke BlogController::viewPost('judul-artikel-saya').

2. Routing Lanjutan

CodeIgniter 4 menawarkan fitur *routing* yang lebih canggih untuk skenario yang kompleks.

a. Route dengan Regular Expression

Anda dapat menggunakan *regular expression* untuk kontrol yang lebih granular terhadap pola URL.

```
$routes->get('products/(:num)-(:alpha)', 'Products::detail/$1/$2');
```

Ini akan mencocokkan URL seperti /products/123-laptop dan memanggil ProductsController::detail(123, 'laptop').

b. Route Groups

Untuk mengorganisir *route* yang memiliki prefiks atau *filter* yang sama, Anda dapat menggunakan *route groups*.

```
$routes->group('admin', function($routes) {
$routes->get('/', 'Admin\Dashboard::index');
$routes->get('users', 'Admin\Users::list');
$routes->post('users/add', 'Admin\Users::add'); });
```

Semua *route* di dalam grup ini akan memiliki prefiks /admin. Jadi, /admin/users akan memanggil Admin\Users::list. Anda juga bisa menerapkan *filters* ke seluruh grup.

c. Route Resources

Untuk operasi CRUD standar, CodeIgniter 4 menyediakan *resource routing* yang secara otomatis membuat *route* untuk operasi index, show, new, create, edit, update, dan delete.

```
$routes->resource('photos');
```

Ini akan menghasilkan *route* seperti:

- GET /photos -> Photos::index
- GET /photos/new -> Photos::new
- POST /photos -> Photos::create
- GET /photos/(:num) -> Photos::show/$1
- GET /photos/(:num)/edit -> Photos::edit/$1
- PUT/PATCH /photos/(:num) -> Photos::update/$1
- DELETE /photos/(:num) -> Photos::delete/$1

d. Reverse Routing (Named Routes)

Anda dapat memberikan nama pada *route* untuk memudahkan pembuatan URL di aplikasi Anda. Ini sangat berguna jika Anda perlu mengubah pola URL di masa mendatang, karena Anda hanya perlu mengubah definisi *route* dan bukan setiap *link* di *View*.

```
$routes->get('users/(:num)', 'Users::show/$1', ['as' => 'user_profile']);
// Di View atau Controller: // echo url_to('user_profile', 5); // Akan menghasilkan '/users/5'
```

Sistem *routing* yang komprehensif ini memungkinkan pengembang untuk membangun URL yang semantik, mudah dikelola, dan fleksibel, yang merupakan aspek penting dari aplikasi web modern.

E. PENGATURAN ENVIRONMENT .ENV DAN KONFIGURASI AWAL

Pengaturan lingkungan (environment) adalah aspek krusial dalam pengembangan aplikasi web, terutama ketika aplikasi perlu berjalan di berbagai kondisi (misalnya, lingkungan pengembangan lokal, *staging*, dan produksi). CodeIgniter 4 mengadopsi pendekatan yang populer dengan menggunakan file .env untuk mengelola variabel lingkungan dan konfigurasi awal (CodeIgniter Documentation, n.d.).

1. Pentingnya File .env

File .env (singkatan dari *environment*) adalah file teks sederhana yang berisi pasangan kunci-nilai untuk variabel lingkungan. Keunggulan utama penggunaan .env adalah:

- **Pemisahan Konfigurasi**: Memisahkan konfigurasi sensitif (seperti kredensial *database*, kunci API) dari kode sumber aplikasi.
- **Fleksibilitas Lingkungan**: Memungkinkan aplikasi untuk memiliki konfigurasi yang berbeda di setiap lingkungan tanpa mengubah kode aplikasi. Misalnya, *database* pengembangan bisa berbeda dengan *database* produksi.
- **Keamanan**: File .env biasanya tidak disertakan dalam kontrol versi (misalnya, Git) sehingga informasi sensitif tidak terekspos di repositori publik.

2. Struktur dan Penggunaan File .env

Ketika Anda menginstal CodeIgniter 4, Anda akan menemukan file env di *root* proyek. Anda harus menyalin atau mengganti nama file ini menjadi .env.

```
# env (contoh isi)
#--------------------------------------------------------------------
# ENVIRONMENT
#--------------------------------------------------------------------
CI_ENVIRONMENT = development
#--------------------------------------------------------------------
# APP
#--------------------------------------------------------------------
app.baseURL = 'http://localhost:8080/'
app.indexPage = ''
app.forceGlobalSecureRequests = false
#--------------------------------------------------------------------
# DATABASE
#--------------------------------------------------------------------
database.default.hostname = localhost
database.default.database = ci4_app
database.default.username = root
database.default.password = root
database.default.DBDriver = MySQLi
database.default.DBPrefix =
```

Setiap baris dalam file .env mendefinisikan sebuah variabel lingkungan. CodeIgniter 4 secara otomatis akan memuat variabel-variabel ini dan membuatnya tersedia melalui fungsi env() atau melalui kelas Config.

a. CI_ENVIRONMENT

Variabel ini sangat penting karena menentukan mode operasi aplikasi.

- **development**: Mode ini mengaktifkan *debugging* ekstensif, *logging* yang lebih detail, dan *caching* yang minimal. Ideal untuk pengembangan lokal.
- **testing**: Digunakan saat menjalankan *unit test* atau *integration test*.
- **production**: Mode ini menonaktifkan *debugging* dan *logging* yang berlebihan, mengaktifkan *caching* yang agresif, dan mengoptimalkan kinerja. Ini adalah mode yang harus digunakan di server *live*.

Mengubah CI_ENVIRONMENT dari development ke production adalah langkah penting sebelum *deployment* untuk alasan keamanan dan kinerja.

b. app.baseURL

Ini adalah URL dasar aplikasi Anda. Penting untuk mengaturnya dengan benar agar *helper* URL dan *routing* berfungsi sebagaimana mestinya.

- Untuk pengembangan lokal: app.baseURL = 'http://localhost:8080/' (jika Anda menggunakan *spark serve*) atau http://localhost/nama_folder_proyek/public/ (jika menggunakan XAMPP/Laragon).
- Untuk produksi: app.baseURL = 'https://namadomainanda.com/'

c. database.default.*

Bagian ini digunakan untuk mengonfigurasi koneksi *database* utama aplikasi.

- database.default.hostname: Alamat *host database* (misalnya, localhost).
- database.default.database: Nama *database*.
- database.default.username: Nama pengguna *database*.
- database.default.password: Kata sandi *database*.
- database.default.DBDriver: Driver *database* yang digunakan (misalnya, MySQLi, Postgre, SQLite3).

3. Konfigurasi Awal Lainnya

Selain .env, beberapa file di app/Config/ juga penting untuk konfigurasi awal.

a. app/Config/App.php

File ini berisi konfigurasi umum aplikasi yang tidak berubah antar lingkungan atau yang tidak sensitif. Beberapa pengaturan penting meliputi:

- public $baseURL: Meskipun bisa diatur di .env, Anda juga bisa mengaturnya di sini jika tidak menggunakan .env.
- public $indexPage: Secara *default* kosong, yang berarti URL Anda akan bersih (misalnya, http://localhost/users). Jika Anda ingin menyertakan index.php di URL, Anda bisa mengaturnya menjadi 'index.php'.
- public $defaultLocale: Bahasa *default* aplikasi.
- public $sessionDriver, public $sessionSavePath: Pengaturan untuk sesi pengguna.

b. app/Config/Database.php

Meskipun kredensial *database* diatur di .env, file Database.php mendefinisikan konfigurasi *database* yang lebih rinci, termasuk *driver*, *charset*, *collation*, dan pengaturan koneksi lainnya. Variabel dari .env akan menimpa pengaturan di sini jika ada.

c. app/Config/Autoload.php

File ini memungkinkan Anda untuk secara otomatis memuat *helper*, *library*, atau *model* tertentu di seluruh aplikasi tanpa perlu memuatnya secara manual di setiap *controller*.

Dengan memahami dan mengelola file .env serta konfigurasi awal lainnya, Anda dapat memastikan aplikasi CodeIgniter 4 Anda berjalan dengan aman, efisien, dan dapat beradaptasi dengan berbagai lingkungan pengembangan dan produksi.

F. RANGKUMAN

Bab 2 ini telah menguraikan fondasi arsitektur dan operasional CodeIgniter 4, memberikan pemahaman komprehensif tentang bagaimana *framework* ini dirancang dan bekerja. Pembahasan dimulai dengan inti arsitektur Model-View-Controller (MVC) yang menjadi tulang punggung CodeIgniter 4, dilanjutkan dengan struktur folder yang terorganisir, siklus permintaan-respons, konsep *routing*, dan pentingnya pengaturan lingkungan.

1. **Arsitektur MVC**: CodeIgniter 4 mengimplementasikan pola desain MVC yang memisahkan aplikasi menjadi tiga komponen utama. Model bertanggung jawab atas data dan logika bisnis, berinteraksi dengan *database*. View fokus pada presentasi data kepada pengguna melalui antarmuka. Controller bertindak sebagai perantara, menerima permintaan, memprosesnya dengan Model, dan memilih View yang tepat. Pemisahan ini meningkatkan modularitas, pemeliharaan, skalabilitas, dan kolaborasi tim.
2. **Struktur Folder CodeIgniter 4**: Struktur direktori CodeIgniter 4 dirancang untuk memisahkan kode inti *framework* dari kode aplikasi. Direktori app/ adalah pusat pengembangan, menampung *controller*, *model*, *view*, konfigurasi, dan logika aplikasi lainnya. Direktori public/ berfungsi sebagai *document root* web, berisi index.php sebagai *front controller*. system/ berisi inti *framework* yang tidak boleh dimodifikasi, sementara writable/ menyimpan data yang dihasilkan aplikasi seperti *cache* dan *log*. Direktori vendor/ dikelola oleh Composer untuk *dependency* pihak ketiga.
3. **Proses Request–Response**: Siklus permintaan-respons menjelaskan alur kerja CodeIgniter 4. Dimulai dari permintaan klien ke server web, yang kemudian diarahkan ke public/index.php. CodeIgniter melakukan inisialisasi, lalu *router* memetakan URL ke *controller* dan *method* yang sesuai. Permintaan dapat melewati *filters* sebelum eksekusi *controller*. *Controller* berinteraksi dengan *Model*, menyiapkan data, dan merender *View*. Respons yang dihasilkan kemudian dapat melewati *filters* lagi sebelum dikirim kembali ke klien.
4. **Konsep Routing Dasar dan Lanjutan**: Sistem *routing* CodeIgniter 4 memungkinkan pemetaan URL yang fleksibel. *Routing* dasar mencakup *route* statis dan *route* dengan *placeholder* (:num, :any) untuk URL dinamis. *Routing* lanjutan menawarkan fitur seperti *regular expression*, *route groups* untuk mengorganisir *route* dengan prefiks atau *filter* yang sama, *resource routing* untuk operasi CRUD standar, dan *reverse routing* (named routes) untuk memudahkan pembuatan URL.
5. **Pengaturan Environment .env dan Konfigurasi Awal**: Penggunaan file .env sangat penting untuk mengelola variabel lingkungan dan konfigurasi sensitif secara terpisah dari kode sumber. Variabel CI_ENVIRONMENT menentukan mode operasi aplikasi (development, testing, production), yang memengaruhi *debugging* dan kinerja. app.baseURL mengatur URL dasar aplikasi, dan database.default.* mengonfigurasi koneksi *database*. File konfigurasi lain di app/Config/ seperti App.php, Database.php, dan Autoload.php juga berperan dalam pengaturan awal aplikasi.

BAB 3
INSTALASI DAN PERSIAPAN LINGKUNGAN PENGEMBANGAN

Selamat datang di Bab 3 modul pembelajaran Pemrograman Aplikasi Web ini. Setelah pada bab-bab sebelumnya kita telah memahami konsep dasar sistem informasi berbasis web, peranan *framework*, dan mengapa CodeIgniter 4 menjadi pilihan yang relevan, kini saatnya kita melangkah ke tahap implementasi. Bab ini merupakan fondasi krusial yang akan menentukan kelancaran seluruh proses pengembangan aplikasi web ke depannya. Tanpa instalasi dan konfigurasi lingkungan pengembangan yang tepat, kita tidak akan bisa memulai penulisan kode dan pengujian aplikasi.

Urgensi Bab 3 ini terletak pada fakta bahwa pengembangan aplikasi web modern sangat bergantung pada ekosistem perangkat lunak yang terintegrasi. CodeIgniter 4, sebagai *framework* PHP, memerlukan lingkungan server lokal yang mendukung PHP dan database, serta alat bantu seperti Composer untuk mengelola *dependency*. Composer adalah manajer paket standar de facto untuk PHP, yang memungkinkan kita menginstal CodeIgniter 4 beserta semua pustaka dan komponen yang dibutuhkan secara efisien. Memahami cara kerjanya adalah kunci untuk menjaga proyek tetap terorganisir dan *up-to-date*.

Selain instalasi *framework*, konfigurasi server lokal seperti XAMPP, Laragon, atau Docker juga menjadi bagian tak terpisahkan. Pilihan server lokal ini akan menyediakan lingkungan Apache/Nginx, PHP, dan MySQL yang diperlukan untuk menjalankan aplikasi web kita. Pengaturan base URL juga sangat penting agar aplikasi dapat mengenali alamat dasarnya, terutama saat berpindah dari lingkungan pengembangan ke produksi.

Bab ini juga akan memandu Anda melalui tes pertama CodeIgniter 4, sebuah langkah sederhana namun vital untuk memastikan bahwa semua komponen telah terinstal dan terkonfigurasi dengan benar. Ini adalah momen di mana Anda akan melihat halaman *welcome* CodeIgniter 4, menandakan bahwa *framework* siap untuk digunakan. Terakhir, kita akan membahas manajemen *dependency* dan *autoloading*, dua konsep fundamental yang memastikan bahwa semua kelas dan pustaka yang Anda butuhkan dapat diakses secara otomatis tanpa perlu menyertakannya secara manual di setiap berkas. Dengan menguasai bab ini, Anda akan memiliki landasan teknis yang kuat untuk mulai membangun aplikasi web yang fungsional dan efisien.

A. INSTALASI MENGGUNAKAN COMPOSER

Instalasi CodeIgniter 4 (CI4) secara modern dan direkomendasikan adalah melalui Composer. Composer adalah manajer paket (package manager) untuk PHP yang memungkinkan Anda mendeklarasikan pustaka (libraries) yang dibutuhkan proyek Anda dan akan mengelolanya (menginstal/memperbarui) untuk Anda (Composer, n.d.). Penggunaan Composer memastikan bahwa semua dependensi yang diperlukan oleh CI4 terinstal dengan versi yang kompatibel, serta memudahkan pembaruan di masa mendatang.

1. Persyaratan Sistem

Sebelum memulai instalasi, pastikan sistem Anda memenuhi persyaratan minimum untuk CodeIgniter 4:

- **PHP versi 7.4 atau lebih tinggi**. Disarankan menggunakan PHP 8.0 atau lebih baru untuk performa dan fitur terbaru.
- Ekstensi PHP intl, mbstring, json, mysqlnd (jika menggunakan MySQL), xml, dan gd (jika diperlukan untuk manipulasi gambar) harus diaktifkan.
- Web server (Apache, Nginx, atau PHP built-in server).

- Database (MySQL, PostgreSQL, SQLite3, SQLSRV, Oracle, atau CUBRID).

2. Instalasi Composer

Jika Anda belum memiliki Composer, unduh dan instal dari situs resminya (getcomposer.org).

- **Untuk Windows**: Unduh Composer-Setup.exe dan ikuti instruksi instalasi. Pastikan opsi Add PHP to your PATH dicentang.
- **Untuk Linux/macOS**: Buka terminal dan jalankan perintah berikut:

```
php -r copy('https://getcomposer.org/installer', 'composer-setup.php');
php composer-setup.php php -r unlink('composer-setup.php');
sudo mv composer.phar /usr/local/bin/composer
```

Setelah instalasi, verifikasi dengan menjalankan composer --version di terminal.

3. Instalasi CodeIgniter 4

Setelah Composer terinstal, Anda dapat menginstal CI4 dengan mudah. Buka terminal atau command prompt, navigasikan ke direktori tempat Anda ingin menyimpan proyek, lalu jalankan perintah Composer berikut:

```
composer create-project codeigniter4/appstarter nama-proyek-anda
```

Ganti nama-proyek-anda dengan nama direktori yang Anda inginkan untuk proyek CI4 Anda. Composer akan mengunduh semua berkas CI4 dan dependensinya ke dalam direktori tersebut. Proses ini mungkin memerlukan waktu beberapa menit tergantung kecepatan internet Anda.

jika proses installasi CI dengan composer berhasil, ketikkan perintah untuk mengaktifkan server:

```
php spark serve
```

Gambar 3. 1 Halaman CodeIgniter 4

B. KONFIGURASI SERVER LOKAL (XAMPP/LARAGON/DOCKER)

Untuk menjalankan aplikasi web PHP seperti CodeIgniter 4, Anda memerlukan lingkungan server lokal yang menyediakan Apache/Nginx, PHP, dan MySQL. Ada beberapa pilihan populer: XAMPP, Laragon, dan Docker.

1. XAMPP

XAMPP adalah paket perangkat lunak *cross-platform* yang menyediakan Apache, MariaDB (pengganti MySQL), PHP, dan Perl (Apache Friends, n.d.). Ini adalah pilihan yang sangat populer karena kemudahan instalasinya.

- **Instalasi**: Unduh XAMPP dari apachefriends.org dan ikuti instruksi instalasi.
- **Konfigurasi**: Setelah terinstal, jalankan XAMPP Control Panel. Mulai modul Apache dan MySQL.
- **Penempatan Proyek**: Secara default, berkas proyek web harus ditempatkan di dalam direktori htdocs di dalam folder instalasi XAMPP (misalnya, C:\xampp\htdocs). Pindahkan atau buat *symlink* proyek CI4 Anda ke sana.
- **Akses**: Aplikasi CI4 Anda dapat diakses melalui ***http://localhost/nama-proyek-anda/public***.

2. Laragon

Laragon adalah lingkungan pengembangan universal yang cepat, ringan, kuat, dan terisolasi untuk PHP, Node.js, Python, Java, Go, dan Ruby (Laragon, n.d.). Laragon menawarkan fitur *virtual host* otomatis dan *pretty URLs* yang sangat memudahkan pengembangan.

- **Instalasi**: Unduh Laragon dari laragon.org dan instal.
- **Konfigurasi**: Setelah instalasi, jalankan Laragon. Mulai Apache dan MySQL. Laragon secara otomatis akan mengkonfigurasi *virtual host* untuk setiap proyek di dalam direktori www Laragon.
- **Penempatan Proyek**: Tempatkan proyek CI4 Anda di dalam direktori www Laragon (misalnya, C:\laragon\www\nama-proyek-anda).
- **Akses**: Laragon akan secara otomatis membuat *pretty URL* seperti http://nama-proyek-anda.test. Anda dapat mengakses aplikasi CI4 Anda melalui http://nama-proyek-anda.test.

3. Docker

Docker adalah platform untuk mengembangkan, mengirim, dan menjalankan aplikasi menggunakan kontainer (Docker, n.d.). Menggunakan Docker untuk lingkungan pengembangan menawarkan isolasi dan konsistensi yang tinggi, memastikan lingkungan Anda sama persis dengan lingkungan produksi.

- **Instalasi**: Unduh dan instal Docker Desktop dari docker.com.
- **Konfigurasi**: Anda perlu membuat berkas docker-compose.yml di root proyek CI4 Anda untuk mendefinisikan layanan (PHP, Nginx/Apache, MySQL). Contoh sederhana:

```
version: '3.8' services: web: image: nginx:latest ports: - 80:80 volumes: - .:/var/www/html - ./nginx/default.conf:/etc/nginx/conf.d/default.conf php: image: php:8.1-fpm volumes: - .:/var/www/html db: image: mysql:8.0 environment: MYSQL_ROOT_PASSWORD: root MYSQL_DATABASE: ci4_db ports: - 3306:3306
```

Anda juga perlu berkas konfigurasi Nginx (misalnya, nginx/default.conf) yang mengarahkan permintaan ke public/index.php.

- **Jalankan**: Dari terminal di root proyek, jalankan docker-compose up -d.
- **Akses**: Aplikasi CI4 Anda akan dapat diakses melalui http://localhost.

C. PENGATURAN BASE URL

Base URL adalah alamat dasar aplikasi Anda yang digunakan oleh CodeIgniter untuk menghasilkan URL yang benar di seluruh aplikasi. Pengaturan yang tepat sangat penting agar semua tautan, aset (CSS, JS, gambar), dan *redirect* berfungsi dengan baik.

1. Mengapa base URL Penting?

CodeIgniter 4 secara default cukup cerdas dalam mendeteksi base URL secara otomatis. Namun, dalam beberapa skenario (misalnya, saat aplikasi berada di *sub-directory*, menggunakan *virtual host* kustom, atau saat *deployment* ke server produksi), deteksi otomatis mungkin tidak selalu akurat. Menentukan base URL secara eksplisit akan mencegah masalah *routing* dan *resource loading*.

2. Konfigurasi base URL

CodeIgniter 4 mengelola konfigurasi melalui berkas .env dan berkas konfigurasi di direktori app/Config.

- **Berkas .env**: Setelah instalasi, Anda akan menemukan berkas env di root proyek. Salin berkas ini dan ganti namanya menjadi .env. Berkas .env digunakan untuk menyimpan konfigurasi spesifik lingkungan (development, production, testing) dan tidak boleh di-*commit* ke *version control*.
- **Pengaturan di .env**: Buka berkas .env dan cari baris app.baseURL. Hapus tanda komentar (#) di depannya dan atur nilainya sesuai dengan alamat aplikasi Anda.

#---------- # BASE URL #---------- # This is the public portion of your site. For example # http://example.com/ or http://example.com/subfolder/ # app.baseURL = 'http://localhost:8080/' Jika Anda menggunakan XAMPP dan proyek Anda berada di htdocs/nama-proyek-anda, maka base URL mungkin http://localhost/nama-proyek-anda/. Jika Anda menggunakan Laragon dengan *pretty URL*nama-proyek-anda.test, maka base URL adalah http://nama-proyek-anda.test/. Jika Anda menggunakan Docker dan mengaksesnya di http://localhost, maka base URL adalah http://localhost/.

- **Pengaturan di app/Config/App.php**: Meskipun disarankan menggunakan .env, Anda juga bisa mengatur baseURL langsung di app/Config/App.php. Namun, ini kurang fleksibel untuk lingkungan yang berbeda.

public string $baseURL = 'http://localhost:8080/'; Pastikan untuk selalu mengakhiri base URL dengan garis miring (/).

D. TES PERTAMA CODEIGNITER 4

Setelah instalasi dan konfigurasi server lokal serta base URL, langkah selanjutnya adalah melakukan tes pertama untuk memastikan bahwa CodeIgniter 4 telah terinstal dan berfungsi dengan benar.

1. Menjalankan Aplikasi

- **Menggunakan PHP Built-in Server (Direkomendasikan untuk Development Cepat)**: CodeIgniter 4 dilengkapi dengan perintah CLI (Command Line Interface) yang memungkinkan Anda menjalankan server PHP bawaan. Ini sangat praktis untuk pengembangan cepat tanpa perlu mengkonfigurasi Apache/Nginx secara manual. Buka terminal, navigasikan ke root direktori proyek CI4 Anda, lalu jalankan perintah: *php spark serve* Secara default, ini akan menjalankan aplikasi di http://localhost:8080. Jika port 8080 sudah digunakan, Anda bisa menentukan port lain: *php spark serve --port 8081*.
- **Menggunakan XAMPP/Laragon/Docker**: Pastikan server Apache/Nginx dan MySQL Anda berjalan. Kemudian, akses aplikasi Anda melalui *browser* menggunakan base URL yang telah Anda konfigurasikan sebelumnya (misalnya, http://localhost:8080, http://nama-proyek-anda.test, atau http://localhost/nama-proyek-anda/public).

2. Verifikasi Halaman Selamat Datang

Jika instalasi berhasil, Anda akan melihat halaman selamat datang CodeIgniter 4 di *browser* Anda. Halaman ini biasanya menampilkan logo CodeIgniter dan pesan seperti Welcome to CodeIgniter 4!. Ini menandakan bahwa:

- PHP berfungsi dengan baik.
- Web server (Apache/Nginx/PHP built-in server) telah mengarahkan permintaan ke public/index.php CodeIgniter.
- CodeIgniter 4 telah berhasil diinisialisasi.
- *Routing* dasar berfungsi.

Jika Anda melihat pesan error atau halaman kosong, periksa kembali langkah-langkah instalasi, konfigurasi server lokal, dan pengaturan base URL. Periksa juga log error PHP dan server Anda untuk petunjuk lebih lanjut.

E. MANAJEMEN DEPENDENCY DAN AUTOLOADING

Manajemen *dependency* dan *autoloading* adalah dua konsep fundamental dalam pengembangan aplikasi PHP modern, termasuk CodeIgniter 4. Keduanya bekerja sama untuk memastikan bahwa semua kelas dan pustaka yang dibutuhkan aplikasi tersedia saat diperlukan tanpa perlu menyertakannya secara manual di setiap berkas.

1. Manajemen Dependency dengan Composer

- **Definisi Dependency**: Dalam konteks pemrograman, *dependency* adalah pustaka atau paket kode eksternal yang dibutuhkan oleh proyek Anda untuk berfungsi. Misalnya, CodeIgniter 4 sendiri memiliki *dependency* seperti laminas/laminas-escaper untuk *escaping* output HTML (CodeIgniter, n.d.).
- **Peran Composer**: Composer adalah alat utama untuk mengelola *dependency* di PHP. Ketika Anda menjalankan composer create-project atau composer install, Composer akan membaca berkas composer.json di root proyek Anda. Berkas ini berisi daftar semua *dependency* yang dibutuhkan, beserta versi yang kompatibel.
- **composer.json**: Ini adalah berkas konfigurasi utama Composer. Contoh bagian require di composer.json CodeIgniter 4:

{ name: codeigniter4/appstarter, type: project, description: CodeIgniter4 starter app, homepage: https://codeigniter.com, license: MIT, require: { php: ^7.4 || ^8.0, codeigniter4/framework: ^4.4, myth/auth: ^1.0 // Contoh dependency tambahan }, require-dev: { codeigniter4/coding-standard: ^1.0, mikey179/vfsstream: ^1.6, phpunit/phpunit: ^9.1 }, // ... bagian lain } require mendefinisikan *dependency* yang dibutuhkan untuk menjalankan aplikasi, sedangkan require-dev adalah *dependency* untuk pengembangan dan pengujian.

- **vendor Directory**: Ketika Composer menginstal *dependency*, semua pustaka akan ditempatkan di dalam direktori vendor di root proyek. Direktori ini tidak boleh diubah secara manual dan harus diabaikan oleh *version control* (misalnya, dengan menambahkannya ke .gitignore).
- **Pembaruan Dependency**: Untuk memperbarui semua *dependency* ke versi terbaru yang kompatibel, jalankan composer update.

2. Autoloading

- **Konsep Autoloading**: Autoloading adalah mekanisme yang secara otomatis memuat kelas PHP hanya ketika kelas tersebut pertama kali digunakan, tanpa perlu menggunakan pernyataan require atau include secara manual di setiap berkas. Ini meningkatkan performa dan menjaga kode tetap bersih.

- **Peran Composer Autoloader**: Composer secara otomatis menghasilkan berkas vendor/autoload.php. Berkas ini adalah inti dari sistem *autoloading* di proyek Anda. CodeIgniter 4 menyertakan berkas ini di public/index.php (atau spark):

// public/index.php // ... require FCPATH . 'vendor/autoload.php'; // ... Ketika Anda menggunakan kelas dari *dependency* yang diinstal oleh Composer, vendor/autoload.php akan menemukan dan memuat berkas kelas tersebut secara otomatis.

- **Autoloading Kustom di CodeIgniter 4**: Selain *autoloading* Composer, CodeIgniter 4 juga memiliki sistem *autoloading* sendiri untuk kelas-kelas aplikasi Anda (Controller, Model, Library kustom, dll.). Konfigurasi ini diatur di app/Config/Autoload.php.
    - **psr4**: Ini adalah metode *autoloading* yang paling umum dan direkomendasikan, mengikuti standar PSR-4. Anda mendefinisikan *namespace* dan memetakannya ke direktori fisik.

```
// app/Config/Autoload.php public array $psr4 = [ APP_NAMESPACE => APPPATH, // 'App' => APPPATH 'Config' => APPPATH . 'Config', // 'Modules' => APPPATH . 'Modules', // Contoh untuk modul kustom ];
```

Dengan konfigurasi di atas, ketika Anda membuat kelas dengan *namespace*App\Controllers\Home, CodeIgniter akan mencarinya di app/Controllers/Home.php.

- **classmap**: Digunakan untuk kelas-kelas yang tidak mengikuti standar PSR-4 atau untuk kelas-kelas yang perlu dimuat secara eksplisit.
- **files**: Untuk berkas-berkas yang berisi fungsi-fungsi global atau *helper* yang perlu dimuat di setiap permintaan.

Dengan memahami dan memanfaatkan manajemen *dependency* Composer serta sistem *autoloading* CodeIgniter 4, Anda dapat membangun aplikasi yang terstruktur, mudah dikelola, dan efisien.

F. RANGKUMAN

Pembahasan ini telah menguraikan langkah-langkah esensial dalam menginstal dan mengkonfigurasi CodeIgniter 4, serta memahami fondasi manajemen dependensi dan *autoloading*. Proses instalasi modern melalui Composer memastikan lingkungan pengembangan yang efisien dan terkelola dengan baik.

1. **Instalasi CodeIgniter 4 dengan Composer**: Instalasi CI4 direkomendasikan menggunakan Composer, manajer paket PHP, yang memastikan semua dependensi terinstal dengan versi yang kompatibel. Proses ini dimulai dengan memastikan persyaratan sistem (PHP 7.4+, ekstensi PHP) terpenuhi, diikuti dengan instalasi Composer itu sendiri, dan kemudian menjalankan perintah composer create-project codeigniter4/appstarter nama-proyek-anda.
2. **Konfigurasi Server Lokal**: Untuk menjalankan aplikasi CI4, diperlukan lingkungan server lokal. Tiga opsi populer dibahas:
    - **XAMPP**: Paket *cross-platform* yang mudah diinstal, menempatkan proyek di htdocs dan diakses melalui http://localhost/nama-proyek-anda/public.
    - **Laragon**: Lingkungan pengembangan yang cepat dan ringan dengan fitur *virtual host* otomatis, menempatkan proyek di www dan diakses melalui *pretty URL* seperti http://nama-proyek-anda.test.
    - **Docker**: Platform kontainerisasi yang menawarkan isolasi dan konsistensi tinggi, memerlukan konfigurasi docker-compose.yml dan diakses melalui http://localhost.

3. **Pengaturan Base URL**: base URL adalah alamat dasar aplikasi yang krusial untuk *routing* dan pemuatan aset yang benar. Meskipun CI4 memiliki deteksi otomatis, pengaturan eksplisit di berkas .env (misalnya, app.baseURL = 'http://localhost:8080/') sangat disarankan untuk mencegah masalah, terutama di lingkungan yang berbeda.
4. **Tes Pertama CodeIgniter 4**: Setelah instalasi dan konfigurasi, aplikasi dapat dijalankan menggunakan PHP built-in server (php spark serve) atau melalui server lokal (XAMPP/Laragon/Docker). Verifikasi keberhasilan instalasi ditandai dengan munculnya halaman selamat datang CodeIgniter 4 di *browser*.
5. **Manajemen Dependensi dan Autoloading**:
    - **Manajemen Dependensi dengan Composer**: Composer mengelola pustaka eksternal (dependensi) yang dibutuhkan proyek melalui berkas composer.json. Semua dependensi diinstal ke direktori vendor dan dapat diperbarui dengan composer update.
    - **Autoloading**: Mekanisme ini secara otomatis memuat kelas PHP saat dibutuhkan, meningkatkan performa dan menjaga kode tetap bersih. Composer menyediakan *autoloader* utama (vendor/autoload.php), sementara CI4 juga memiliki sistem *autoloading* kustom (terutama psr4 di app/Config/Autoload.php) untuk kelas-kelas aplikasi.

Dengan pemahaman ini, pengembang dapat memulai proyek CodeIgniter 4 dengan fondasi yang kuat dan terstruktur.

BAB 4
DASAR-DASAR MODEL, CONTROLLER, DAN VIEW

Setelah kita memahami dasar-dasar sistem informasi berbasis web, arsitektur CodeIgniter 4, serta proses instalasi dan persiapan lingkungan pengembangan pada bab-bab sebelumnya, kini saatnya kita menyelami inti dari pengembangan aplikasi web dinamis. Bab 4 ini akan menjadi fondasi krusial dalam membangun aplikasi yang interaktif dan berbasis data. Kita akan mulai dengan memperkenalkan tiga pilar utama dalam arsitektur Model-View-Controller (MVC) yang menjadi jantung CodeIgniter 4: Model, View, dan Controller.

Urgensi pembahasan bab ini terletak pada kemampuannya untuk menjembatani pemahaman teoritis dengan implementasi praktis. Tanpa pemahaman yang kuat tentang bagaimana Controller menerima permintaan, View menyajikan antarmuka, dan Model berinteraksi dengan data, pengembangan aplikasi web akan menjadi tidak terstruktur dan sulit dikelola. Kita akan belajar bagaimana Controller bertindak sebagai otak aplikasi, mengelola alur permintaan dan respons, serta bagaimana View berfungsi sebagai wajah aplikasi, menyajikan informasi kepada pengguna dalam format yang menarik dan mudah dipahami.

Lebih lanjut, bab ini akan membahas pentingnya menyusun template halaman menggunakan layout. Pendekatan ini sangat vital untuk menjaga konsistensi desain di seluruh aplikasi dan menghindari duplikasi kode yang tidak perlu. Bayangkan sebuah situs web dengan puluhan halaman; tanpa layout, setiap perubahan pada header atau footer harus dilakukan secara manual di setiap halaman, sebuah tugas yang memakan waktu dan rentan kesalahan. Dengan layout, perubahan dapat dilakukan di satu tempat dan secara otomatis diterapkan di seluruh aplikasi.

Aspek krusial lainnya adalah bagaimana aplikasi berinteraksi dengan data. Di sinilah peran Model menjadi sangat penting. Kita akan belajar cara membuat Model untuk mengelola koneksi ke database, melakukan operasi dasar seperti mengambil, menyimpan, memperbarui, dan menghapus data. Konsep Entities dan Data Mapping akan diperkenalkan untuk membantu kita merepresentasikan data dari database ke dalam objek PHP yang lebih mudah dimanipulasi dan dipahami dalam konteks aplikasi.

Pada akhirnya, semua konsep ini akan disatukan dalam praktik membangun halaman dashboard sederhana. Ini bukan sekadar latihan, melainkan sebuah simulasi awal dari bagaimana sebuah sistem informasi nyata bekerja, menampilkan data dinamis yang diambil dari database. Dengan menguasai bab ini, mahasiswa akan memiliki landasan yang kokoh untuk melanjutkan ke topik-topik yang lebih kompleks seperti perancangan database, form input, operasi CRUD, hingga autentikasi pengguna, yang semuanya akan dibangun di atas pemahaman MVC yang kuat.

A. MEMBUAT CONTROLLER DAN VIEW DASAR

Dalam arsitektur Model-View-Controller (MVC), Controller dan View adalah dua komponen fundamental yang bekerja sama untuk menangani permintaan pengguna dan menyajikan respons. Controller bertindak sebagai jembatan antara Model dan View, menerima input dari pengguna, memprosesnya, dan kemudian mengarahkan data ke View yang sesuai untuk ditampilkan. Sementara itu, View bertanggung jawab penuh atas presentasi data kepada pengguna, biasanya dalam bentuk HTML, CSS, dan JavaScript.

1. Peran Controller dalam CodeIgniter 4

Controller adalah kelas PHP yang berfungsi sebagai titik masuk utama untuk setiap permintaan HTTP. Ketika pengguna mengakses URL tertentu, CodeIgniter akan merutekan permintaan tersebut ke metode yang sesuai di dalam Controller. Tugas utama Controller meliputi:

- **Menerima Permintaan**: Menangkap data dari URL, parameter GET/POST, atau input lainnya.
- **Memproses Logika Bisnis**: Mengkoordinasikan dengan Model untuk mengambil atau memanipulasi data.
- **Memuat View**: Mengirimkan data yang telah diproses ke View untuk ditampilkan.
- **Mengelola Alur Aplikasi**: Mengarahkan pengguna ke halaman lain atau menampilkan pesan.

Untuk membuat Controller di CodeIgniter 4, kita cukup membuat file PHP baru di direktori app/Controllers. Setiap Controller harus mewarisi kelas BaseController yang disediakan oleh CodeIgniter.

<?php namespace App\Controllers; class Home extends BaseController { public function index() { // Logika bisnis akan ditempatkan di sini return view('welcome_message'); } public function about() { $data['title'] = 'Tentang Kami'; $data['content'] = 'Ini adalah halaman tentang kami.'; return view('about_page', $data); } }

Pada contoh di atas, Home adalah nama Controller, dan index() serta about() adalah metode (action) yang dapat diakses melalui URL. Metode index() adalah metode default yang akan dipanggil jika tidak ada metode lain yang ditentukan dalam URL. Fungsi view() digunakan untuk memuat file View.

2. Peran View dalam CodeIgniter 4

View adalah file yang berisi kode presentasi, seperti HTML, CSS, dan JavaScript, yang bertanggung jawab untuk menampilkan data kepada pengguna. View tidak boleh mengandung logika bisnis yang kompleks; tugasnya murni untuk menyajikan data yang telah disiapkan oleh Controller. File View biasanya disimpan di direktori app/Views.

```
<!-- app/Views/welcome_message.php -->
<!DOCTYPE html>
<html lang=en>
<head>
    <meta charset=UTF-8> <meta name=viewport content-width=device-width, initial-scale=1.0>
    <title>Selamat Datang</title>
</head>
<body>
    <h1>Selamat Datang di Aplikasi Web Kami!</h1>
    <p>Ini adalah halaman utama yang dibuat dengan CodeIgniter 4.x</p>
    <p>dibuat untuk pendukung matakuliah Pemrograman Aplikasi Web</p>
    <p>di Universitas STEKOM Semarang</p>
    <p>yang diampu oleh : Bp Moh Muthohir, M.Kom</p>
</body>
</html>
```

```php
<!-- app/Views/about_page.php -->
<!DOCTYPE html>
<html lang=en>
    <head>
        <meta charset=UTF-8>
        <meta name=viewport content-width=device-width, initial-scale=1.0>
        <title><?= $title ?></title>
    </head>
    <body>
        <h1><?= $title ?></h1>
        <p><?= $content ?></p>
    </body>
</html>
```

Data yang dikirim dari Controller ke View akan tersedia sebagai variabel di dalam file View. Misalnya, $data['title'] di Controller akan menjadi $title di View.

B. MENYUSUN TEMPLATE HALAMAN MENGGUNAKAN LAYOUT

Dalam pengembangan aplikasi web, konsistensi tampilan adalah kunci untuk pengalaman pengguna yang baik. Bagian-bagian seperti header, footer, sidebar, dan navigasi seringkali muncul di banyak halaman. Menyalin kode HTML untuk bagian-bagian ini di setiap View adalah praktik yang tidak efisien dan rentan kesalahan. Di sinilah konsep layout menjadi sangat penting. Layout memungkinkan kita untuk mendefinisikan struktur halaman utama sekali, dan kemudian menyuntikkan konten spesifik dari setiap halaman ke dalam struktur tersebut (Suryani & Hidayat, 2023).

1. Manfaat Penggunaan Layout

- **Konsistensi Desain**: Memastikan semua halaman memiliki tampilan yang seragam.
- **Efisiensi Pengembangan**: Mengurangi duplikasi kode HTML yang sama di banyak file View.
- **Kemudahan Pemeliharaan**: Perubahan pada header atau footer hanya perlu dilakukan di satu tempat (file layout), dan akan otomatis diterapkan di seluruh aplikasi.
- **Modularitas**: Memisahkan struktur umum halaman dari konten spesifik.

2. Implementasi Layout di CodeIgniter 4

CodeIgniter 4 mendukung penggunaan layout melalui fitur View. Kita dapat membuat file View khusus yang berfungsi sebagai layout, dan kemudian memuat View konten di dalamnya.

```
<!-- app/Views/layout/main_layout.php -->
<!DOCTYPE html>
<html lang=en>
    <head>
        <meta charset=UTF-8>
        <meta name=viewport content-width=device-width, initial-scale=1.0>
        <title><?= $title ?? 'Aplikasi CI4' ?></title>
        <link rel=stylesheet href=/css/style.css>
    </head>
    <body>
        <header>
            <h1>Header Aplikasi</h1>
            <nav> <a href=/>Home</a> <a href=/about>About</a> <a href=/contact>Contact</a> </nav>
        </header>
        <main> <!-- Konten spesifik halaman akan disuntikkan di sini -->
            <?= $this->renderSection('content') ?>
        </main>
        <footer> <p>&copy; <?= date('Y') ?> Aplikasi CI4. All rights reserved.</p> </footer>
        <script src=/js/script.js></script>
    </body>
</html>
```

Pada layout di atas, *$this->renderSection('content')* adalah placeholder di mana konten spesifik dari View akan disuntikkan.

Kemudian, di setiap View konten, kita akan menggunakan *extend()* untuk menunjukkan layout mana yang akan digunakan, dan *section()* untuk mendefinisikan konten yang akan disuntikkan.

```
<!-- app/Views/home_page.php -->
<?= $this->extend('layout/main_layout') ?>
<?= $this->section('content') ?>
<h2>Selamat Datang di Halaman Utama</h2>
<p>Ini adalah konten spesifik untuk halaman beranda.</p>
<?= $this->endSection() ?>
```

Untuk memuat *View* ini dari *Controller*, kita cukup memanggilnya seperti biasa:

```
<?php
namespace App\Controllers;
class Home extends BaseController
{
    public function index()
    { $data['title'] = 'Beranda'; return view('home_page', $data); }
}
```

CodeIgniter secara otomatis akan memproses *extend()* dan *section()* untuk menghasilkan halaman HTML lengkap.

C. MEMBUAT MODEL UNTUK KONEKSI DATABASE

Model adalah komponen dalam arsitektur MVC yang bertanggung jawab untuk berinteraksi dengan database. Tugas utama Model adalah mengelola data, termasuk mengambil, menyimpan, memperbarui, dan menghapus data. Dengan memisahkan logika database ke dalam Model, kita menjaga Controller tetap bersih dari detail implementasi database, sehingga aplikasi lebih mudah dipelihara dan diuji (Pratama, 2020).

1. Konfigurasi Database

Sebelum membuat Model, kita perlu mengkonfigurasi koneksi database di CodeIgniter 4. Ini dilakukan melalui file .env dan app/Config/Database.php.

- **File .env**: Untuk pengaturan kredensial database yang sensitif dan spesifik lingkungan.

```
#--------------------------------------------------------------------
# DATABASE
#--------------------------------------------------------------------

database.default.hostname = localhost
database.default.database = ci4
database.default.username = root
database.default.password = root
database.default.DBDriver = MySQLi
database.default.DBPrefix =
database.default.port = 3306
```

- **File app/Config/Database.php**: Untuk konfigurasi database yang lebih detail, seperti charset, collation, atau driver lain.

2. Membuat Model di CodeIgniter 4

CodeIgniter 4 menyediakan kelas CodeIgniter\Model yang sangat powerful untuk berinteraksi dengan database. Model ini menyediakan berbagai metode CRUD (Create, Read, Update, Delete) siap pakai. Untuk membuat Model, kita membuat file PHP baru di direktori app/Models dan mewarisi kelas CodeIgniter\Model.

```php
<?php
namespace App\Models;
use CodeIgniter\Model;
class UserModel extends Model
{ protected $table = 'users'; // Nama tabel di database
    protected $primaryKey = 'id'; // Primary key tabel
    protected $useAutoIncrement = true; // Apakah primary key auto-increment
    protected $returnType = 'array';    // Tipe data yang dikembalikan (array/object/entity)
    protected $useSoftDeletes = true;   // Menggunakan soft delete (menandai data sebagai terhapus)
    protected $allowedFields = ['username', 'email', 'password', 'created_at', 'updated_at', 'deleted_at']; //
    protected $useTimestamps = true; // Menggunakan kolom created_at dan updated_at
    protected $dateFormat = 'datetime';  // Format tanggal
    protected $createdField = 'created_at';
    protected $updatedField = 'updated_at';
    protected $deletedField = 'deleted_at';
    // Validation
    protected $validationRules = []; protected $validationMessages = []; protected $skipValidation = false;
}
```

Beberapa properti penting dalam Model:

- $table: Nama tabel database yang akan diinteraksi oleh Model ini.
- $primaryKey: Nama kolom primary key dari tabel.
- $returnType: Menentukan format data yang dikembalikan oleh metode Model (misalnya, array atau object).
- $useSoftDeletes: Jika true, data tidak dihapus secara permanen, melainkan ditandai sebagai terhapus.
- $allowedFields: Array berisi nama-nama kolom yang diizinkan untuk diisi melalui metode insert() atau update(). Ini adalah fitur keamanan penting untuk mencegah *mass assignment*.
- $useTimestamps: Jika true, CodeIgniter akan secara otomatis mengisi kolom created_at dan updated_at saat data dibuat atau diperbarui.

3. Menggunakan Model di Controller

Setelah Model dibuat, kita dapat menggunakannya di Controller untuk berinteraksi dengan database.

```php
<?php
namespace App\Controllers;
use App\Models\UserModel; // Import Model
class Users extends BaseController
{
    public function index()
    {
        $userModel = new UserModel(); $users = $userModel->findAll(); // Mengambil semua data dari tabel users
        $data['users'] = $users; $data['title'] = 'Daftar Pengguna';
        return view('users/list', $data);
    }
    public function detail($id)
    {
        $userModel = new UserModel();
        $user = $userModel->find($id); // Mengambil data pengguna berdasarkan ID
        if (!$user)
            { throw new \CodeIgniter\Exceptions\PageNotFoundException('Pengguna tidak ditemukan.');
        }
        $data['user'] = $user;
        $data['title'] = 'Detail Pengguna';
        return view('users/detail', $data);
    }
}
```

Metode seperti findAll(), find($id), save(), insert(), update(), dan delete() adalah beberapa contoh metode yang disediakan oleh CodeIgniter\Model untuk mempermudah operasi database.

D. KONSEP ENTITIES DAN DATA MAPPING

Dalam pengembangan aplikasi berorientasi objek, seringkali kita ingin merepresentasikan baris data dari database sebagai objek PHP yang lebih kaya, bukan hanya sekadar array asosiatif. Di sinilah konsep *Entities* dan *Data Mapping* berperan. *Entity* adalah objek PHP yang merepresentasikan satu baris data dari tabel database, lengkap dengan properti (atribut) dan terkadang metode (perilaku) yang terkait dengan data tersebut. *Data Mapping* adalah proses mengubah data dari satu format (misalnya, baris database) ke format lain (objek Entity) (Fowler, 2003).

1. Mengapa Menggunakan Entities?

- **Abstraksi Data**: Data database dienkapsulasi dalam objek, menyembunyikan detail implementasi database.
- **Validasi dan Logika Bisnis**: Entity dapat memiliki metode untuk validasi data atau logika bisnis yang terkait dengan objek tersebut.
- **Keterbacaan Kode**: Kode menjadi lebih mudah dibaca dan dipahami karena kita berinteraksi dengan objek yang memiliki makna bisnis.
- **Tipe Hinting**: Memungkinkan penggunaan tipe hinting di PHP, meningkatkan kejelasan dan mengurangi kesalahan.

2. Implementasi Entities di CodeIgniter 4

CodeIgniter 4 mendukung konsep *Entity* melalui kelas *CodeIgniter\Entity\Entity*. Untuk menggunakannya, kita perlu membuat kelas *Entity* terpisah untuk setiap tabel yang ingin kita representasikan sebagai objek.

```php
<?php
namespace App\Entities;
use CodeIgniter\Entity\Entity;
class User extends Entity
{
    protected $dates = ['created_at', 'updated_at', 'deleted_at']; // Kolom tanggal
    protected $casts = [ // Casting tipe data
        'id' => 'integer',
        'username' => 'string',
        'email' => 'string',
        'is_admin' => 'boolean', ]; // Contoh setter untuk enkripsi password
        public function setPassword(string $password)
        {
            $this->attributes['password'] = password_hash($password, PASSWORD_DEFAULT);
            return $this;
        } // Contoh getter untuk mendapatkan nama lengkap
        public function getFullName()
        {
            return $this->attributes['first_name'] . ' ' . $this->attributes['last_name'];
        }
    }
```

Pada contoh di atas, kelas *User* adalah *Entity* yang merepresentasikan satu baris dari tabel *users*. Properti *$dates* dan *$casts* membantu CodeIgniter dalam mengelola tipe data. Kita juga bisa menambahkan *setter* dan *getter* kustom, seperti *setPassword()* untuk mengenkripsi *password* sebelum disimpan, atau *getFullName()* untuk menggabungkan beberapa atribut.

Untuk menggunakan *Entity* dengan Model, kita perlu mengatur properti $returnType di Model menjadi nama kelas *Entity*.

```php
<?php
namespace App\Models;
use CodeIgniter\Model;
use App\Entities\User; // Import Entity
class UserModel extends Model
{
    protected $table = 'users';
    protected $primaryKey = 'id';
    protected $returnType = User::class; // Mengembalikan objek User Entity
    protected $useSoftDeletes = true;
    protected $allowedFields = ['username', 'email', 'password', 'first_name', 'last_name', 'is_admin'];
    protected $useTimestamps = true;
    protected $dateFormat = 'datetime';
    protected $createdField = 'created_at';
    protected $updatedField = 'updated_at';
    protected $deletedField = 'deleted_at';
}
```

Sekarang, ketika kita mengambil data menggunakan *UserModel*, hasilnya akan berupa objek *User Entity*, bukan *array*.

```php
<?php
namespace App\Controllers;
use App\Models\UserModel;
class Users extends BaseController
{
    public function index()
    {
        $userModel = new UserModel();
        $users = $userModel->findAll(); // $users akan berisi array objek User Entity
        foreach ($users as $user)
            {
                echo $user->username . ' - ' . $user->email . '<br>';
                // Jika ada metode getFullName() di Entity: //
                echo $user->getFullName() . '<br>';
            }
    }
}
```

Penggunaan Entity membuat kode lebih berorientasi objek dan memisahkan representasi data dari logika database, sejalan dengan prinsip desain perangkat lunak yang baik (Martin, 2008).

E. PRAKTIK: HALAMAN DASHBOARD SEDERHANA

Setelah memahami konsep Controller, View, Layout, Model, dan Entity, kini saatnya kita mengintegrasikan semua komponen tersebut untuk membangun halaman dashboard sederhana. Dashboard adalah halaman utama yang biasanya menampilkan ringkasan informasi penting atau statistik kepada pengguna setelah mereka login.

1. Skenario Praktik

Kita akan membuat halaman dashboard yang menampilkan:

- Jumlah total pengguna terdaftar.
- Daftar beberapa pengguna terbaru.
- Menggunakan layout untuk konsistensi tampilan.

2. Langkah-langkah Implementasi

a. Persiapan Database dan Model

Pastikan Anda memiliki tabel users di database dengan beberapa data contoh. Jika belum, Anda bisa membuatnya secara manual atau menggunakan Migrasi (akan dibahas di Bab 5).

```sql
CREATE TABLE `users` (
    `id` int(11) NOT NULL AUTO_INCREMENT,
    `username` varchar(255) NOT NULL,
    `email` varchar(255) NOT NULL,
    `password` varchar(255) NOT NULL,
    `created_at` datetime DEFAULT NULL,
    `updated_at` datetime DEFAULT NULL,
    `deleted_at` datetime DEFAULT NULL,
    PRIMARY KEY (`id`),
    UNIQUE KEY `username` (`username`),
    UNIQUE KEY `email` (`email`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
INSERT INTO `users` (`username`, `email`, `password`, `created_at`) VALUES (
    'admin', 'admin@example.com', '$2y$10$xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx', NOW()),
    ('john.doe', 'john@example.com', '$2y$10$xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx', NOW()),
    ('jane.smith', 'jane@example.com', '$2y$10$xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx', NOW());
```

Pastikan *UserModel* sudah dikonfigurasi seperti pada bagian E, dan jika ingin menggunakan *Entity*, *User Entity* juga sudah dibuat seperti pada bagian F.

b. Membuat Controller Dashboard

Buat *Controller* baru bernama Dashboard di app/Controllers/Dashboard.php.

```php
<?php
namespace App\Controllers;
use App\Models\UserModel; // Import UserModel
class Dashboard extends BaseController
{
    public function index()
    {
        $userModel = new UserModel(); // Mengambil total pengguna
        $totalUsers = $userModel->countAllResults(); // Mengambil 5 pengguna terbaru
        $latestUsers = $userModel->orderBy('created_at', 'DESC')->limit(5)->findAll();
        $data = [ 'title' => 'Dashboard Admin', 'totalUsers' => $totalUsers, 'latestUsers' => $latestUsers, ];
        return view('dashboard/index', $data);
    }
}
```

Dalam *Controller* ini, kita menginstansiasi *UserModel*, kemudian menggunakan metode *countAllResults()* untuk mendapatkan jumlah total pengguna dan *orderBy()->limit()->findAll()* untuk mendapatkan daftar pengguna terbaru. Data ini kemudian dikirim ke *View*.

c. Membuat View Dashboard

Buat file View baru di app/Views/dashboard/index.php. Pastikan Anda juga memiliki file layout, misalnya app/Views/layout/main_layout.php seperti yang dijelaskan di bagian D.

```php
<!-- app/Views/dashboard/index.php -->
<?= $this->extend('layout/main_layout') ?>
<?= $this->section('content') ?>
<div class=container>
    <h2>Selamat Datang di Dashboard!</h2>
    <p>Ini adalah ringkasan aktivitas sistem Anda.</p>
    <div class=card>
        <h3>Total Pengguna</h3>
        <p class=metric><?= $totalUsers ?></p>
    </div>
    <div class=card>
        <h3>Pengguna Terbaru</h3>
        <?php if (!empty($latestUsers)): ?>
            <ul> <?php foreach ($latestUsers as $user): ?>
                <li><?= $user->username ?> (<?= $user->email ?>) - Bergabung pada: <?= $user->created_at->format('d M Y') ?></li>
                <?php endforeach; ?>
            </ul>
            <?php else: ?>
                <p>Belum ada pengguna terdaftar.</p>
            <?php endif; ?>
        </div>
    </div>
    <style>
    .container { max-width: 960px; margin: 20px auto; padding: 20px; background-color: #fff; border-radius: 8px; box-shadow: 0 2px 4px
    .card { background-color: #f9f9f9; border: 1px solid #eee; border-radius: 6px; padding: 15px; margin-bottom: 20px; }
    .card h3 { margin-top: 0; color: #333; } .metric { font-size: 2.5em; font-weight: bold; color: #007bff; text-align: center; margin
    ul { list-style: none; padding: 0; }
    li { background-color: #e9ecef; margin-bottom: 8px; padding: 10px; border-radius: 4px; }
    </style>
<?= $this->endSection() ?>
```

Dalam View ini, kita menggunakan *$this->extend()* untuk memuat *main_layout* dan *$this->section('content')* untuk mendefinisikan konten spesifik *dashboard*. Data *$totalUsers* dan *$latestUsers* yang dikirim dari *Controller* dapat langsung diakses dan ditampilkan. Perhatikan bagaimana kita mengakses properti objek *$user* (jika menggunakan *Entity*) dan memformat tanggal.

d. Konfigurasi Routing

Terakhir, kita perlu mengatur rute agar URL /dashboard mengarah ke Controller Dashboard dan metode *index().* Edit file app/Config/Routes.php.

```
// app/Config/Routes.php
$routes->get('/dashboard', 'Dashboard::index');
```

Sekarang, ketika Anda mengakses http://localhost:8080/dashboard (sesuaikan dengan base URL Anda), Anda akan melihat halaman dashboard sederhana yang menampilkan data dinamis dari database, terbungkus dalam layout yang konsisten. Praktik ini menunjukkan bagaimana ketiga komponen MVC (Model, View, Controller) bekerja secara harmonis untuk membangun aplikasi web yang fungsional dan terstruktur.

F. RANGKUMAN

Pembahasan ini telah menguraikan secara komprehensif komponen-komponen fundamental dalam pengembangan aplikasi web menggunakan CodeIgniter 4, khususnya dalam konteks arsitektur *Model-View-Controller* (MVC). Pemahaman mendalam tentang *Controller, View, Model*, serta konsep pendukung seperti Layout dan Entity, sangat krusial untuk membangun aplikasi yang terstruktur, mudah dipelihara, dan skalabel.
Berikut adalah poin-poin penting yang telah dibahas:

1. **Controller dan View Dasar**: Controller berfungsi sebagai jembatan utama yang menerima permintaan pengguna, memproses logika bisnis (seringkali berkoordinasi dengan Model), dan kemudian memuat View yang sesuai untuk menampilkan respons. View, di sisi lain, bertanggung jawab penuh atas presentasi data kepada pengguna, biasanya dalam format HTML, CSS, dan JavaScript, tanpa mengandung logika bisnis yang kompleks.
2. **Peran Controller dalam CodeIgniter 4**: Controller adalah kelas PHP yang mewarisi BaseController, bertindak sebagai titik masuk untuk setiap permintaan HTTP. Tugasnya meliputi menerima permintaan, memproses logika bisnis, memuat View, dan mengelola alur aplikasi.
3. **Peran View dalam CodeIgniter 4**: View adalah file presentasi (HTML, CSS, JS) yang menampilkan data yang disiapkan oleh Controller. Data dari Controller dapat diakses sebagai variabel di dalam View.
4. **Menyusun Template Halaman Menggunakan Layout**: Layout adalah mekanisme untuk mendefinisikan struktur halaman web yang konsisten (header, footer, navigasi) dan kemudian menyuntikkan konten spesifik dari setiap halaman ke dalamnya. Ini meningkatkan konsistensi desain, efisiensi pengembangan, dan kemudahan pemeliharaan. CodeIgniter 4 mengimplementasikan ini melalui extend() dan section() pada View.
5. **Membuat Model untuk Koneksi Database**: Model adalah komponen MVC yang bertanggung jawab untuk berinteraksi dengan database (CRUD). CodeIgniter 4 menyediakan kelas CodeIgniter\Model yang kaya fitur untuk mempermudah operasi database, dengan properti seperti $table, $primaryKey, $allowedFields, dan $useTimestamps. Konfigurasi database dilakukan melalui file .env dan app/Config/Database.php.
6. **Konsep Entities dan Data Mapping**: Entity adalah objek PHP yang merepresentasikan satu baris data dari tabel database, lengkap dengan properti dan metode terkait. Data mapping adalah proses mengubah data database menjadi objek Entity. Penggunaan Entity (melalui CodeIgniter\Entity\Entity) meningkatkan abstraksi data, memungkinkan validasi dan logika bisnis dalam objek, serta meningkatkan keterbacaan kode.

7. **Praktik: Halaman Dashboard Sederhana**: Implementasi praktis menunjukkan bagaimana Controller, View, Layout, dan Model (dengan atau tanpa Entity) diintegrasikan untuk membangun halaman dashboard yang menampilkan data dinamis dari database, seperti total pengguna dan daftar pengguna terbaru, dalam tampilan yang konsisten. Ini mengilustrasikan sinergi antar komponen MVC dalam membangun aplikasi web fungsional.

BAB 5
PERANCANGAN DAN MIGRASI DATABASE

Dalam pengembangan aplikasi web, khususnya sistem informasi, database memegang peranan sentral sebagai tulang punggung penyimpanan dan pengelolaan data. Tanpa database yang terstruktur dengan baik, sebuah sistem informasi tidak akan mampu berfungsi secara optimal, bahkan cenderung rentan terhadap inkonsistensi data dan kesulitan dalam pengembangan lebih lanjut. Bab 5 ini akan membawa kita menyelami dunia perancangan dan migrasi database, sebuah tahapan krusial yang seringkali menjadi penentu keberhasilan sebuah proyek pengembangan aplikasi.

Kita akan memulai dengan memahami konsep fundamental dari database relasional, yaitu tabel, relasi antar tabel, dan pentingnya *indexing*. Pemahaman yang kuat tentang bagaimana data disimpan dalam tabel, bagaimana tabel-tabel tersebut saling berhubungan melalui relasi, dan bagaimana indexing dapat mempercepat proses pencarian data, adalah kunci untuk membangun database yang efisien dan skalabel. Konsep-konsep ini bukan sekadar teori, melainkan pondasi praktis yang akan kita terapkan dalam setiap langkah perancangan.

Selanjutnya, kita akan beralih ke praktik pembuatan database sistem informasi. Dalam konteks pengembangan modern, pengelolaan skema database tidak lagi dilakukan secara manual melalui antarmuka grafis database. CodeIgniter 4, sebagai *framework* yang kita gunakan, menyediakan fitur Migrasi (*Migration*) yang sangat powerful. Migrasi memungkinkan kita untuk mendefinisikan perubahan skema database dalam bentuk kode, yang kemudian dapat diterapkan, di-rollback, atau di-share antar tim pengembang dengan mudah. Ini adalah praktik terbaik yang memastikan konsistensi skema database di berbagai lingkungan pengembangan dan produksi.

Tidak hanya itu, kita juga akan mempelajari *Seeder*, sebuah fitur pelengkap migrasi yang berfungsi untuk mengisi data awal atau data *dummy* ke dalam tabel. Seeder sangat berguna untuk pengujian aplikasi, demonstrasi, atau bahkan untuk mengisi data konfigurasi awal pada sistem yang baru di-deploy. Dengan migrasi dan seeder, proses setup lingkungan pengembangan menjadi jauh lebih cepat dan terstandardisasi.

Puncak dari bab ini adalah praktik langsung dalam membuat struktur tabel utama yang lazim ditemukan dalam sistem informasi, seperti tabel users untuk manajemen pengguna, tabel data master untuk menyimpan data referensi (misalnya, data produk, kategori, atau departemen), dan tabel transaksi untuk mencatat aktivitas bisnis (misalnya, penjualan, pembelian, atau kehadiran). Melalui praktik ini, mahasiswa akan mendapatkan pengalaman langsung dalam menerjemahkan kebutuhan bisnis menjadi struktur database yang konkret dan fungsional, mempersiapkan fondasi yang kokoh untuk modul-modul aplikasi yang akan dibangun di bab-bab selanjutnya.

A. KONSEP TABEL, RELASI, DAN INDEXING

Database relasional merupakan fondasi utama bagi sebagian besar sistem informasi modern. Dalam paradigma ini, data diorganisasikan ke dalam struktur yang disebut tabel. Pemahaman mendalam tentang tabel, bagaimana mereka saling berhubungan melalui relasi, dan bagaimana indexing dapat mengoptimalkan kinerja, adalah esensial dalam perancangan database yang efektif.

1. Konsep Tabel

Tabel adalah unit dasar penyimpanan data dalam database relasional. Setiap tabel merepresentasikan sebuah entitas atau objek dalam dunia nyata, seperti Pengguna, Produk, atau Transaksi. Tabel terdiri dari baris (record) dan kolom (field atau attribute). Setiap baris mewakili satu instans dari entitas tersebut, sedangkan setiap kolom mewakili properti atau karakteristik dari entitas tersebut.

Sebagai contoh, tabel users mungkin memiliki kolom seperti id, nama_lengkap, email, dan password. Setiap baris dalam tabel ini akan menyimpan data untuk satu pengguna yang berbeda. Penting untuk mendefinisikan tipe data yang tepat untuk setiap kolom (misalnya, INT untuk id, VARCHAR untuk nama_lengkap dan email, TEXT untuk password) serta batasan (constraint) seperti NOT NULL atau UNIQUE untuk menjaga integritas data (Elmasri & Navathe, 2022). Kunci utama (Primary Key) adalah kolom atau kombinasi kolom yang secara unik mengidentifikasi setiap baris dalam tabel. Kunci utama tidak boleh berisi nilai NULL dan harus unik untuk setiap baris.

2. Konsep Relasi Antar Tabel

Sistem informasi jarang sekali hanya menggunakan satu tabel. Data seringkali saling terkait, dan untuk menghindari redundansi serta menjaga konsistensi, kita menggunakan relasi antar tabel. Relasi didefinisikan menggunakan kunci asing (*Foreign Key*). Kunci asing adalah kolom dalam satu tabel yang merujuk ke kunci utama di tabel lain. Ada tiga jenis relasi utama:

a. **One-to-One (Satu-ke-Satu)** Relasi ini terjadi ketika satu baris di tabel A hanya dapat dihubungkan dengan satu baris di tabel B, dan sebaliknya. Contoh: satu User memiliki satu Profil_Pengguna yang berisi detail tambahan seperti alamat atau nomor telepon. Relasi ini sering diimplementasikan dengan menempatkan kunci asing di salah satu tabel yang juga berfungsi sebagai kunci utama, atau dengan memastikan keunikan kunci asing.

b. **One-to-Many (Satu-ke-Banyak)** Ini adalah jenis relasi yang paling umum. Satu baris di tabel A dapat dihubungkan dengan banyak baris di tabel B, tetapi satu baris di tabel B hanya dapat dihubungkan dengan satu baris di tabel A. Contoh: satu Kategori produk dapat memiliki banyak Produk, tetapi satu Produk hanya termasuk dalam satu Kategori. Relasi ini diimplementasikan dengan menempatkan kunci asing di tabel banyak (tabel Produk akan memiliki kategori_id yang merujuk ke id di tabel Kategori).

c. **Many-to-Many (Banyak-ke-Banyak)** Relasi ini terjadi ketika satu baris di tabel A dapat dihubungkan dengan banyak baris di tabel B, dan satu baris di tabel B juga dapat dihubungkan dengan banyak baris di tabel A. Contoh: satu Siswa dapat mengambil banyak Mata_Kuliah, dan satu Mata_Kuliah dapat diambil oleh banyak Siswa. Relasi ini tidak dapat diimplementasikan secara langsung. Sebaliknya, dibutuhkan tabel perantara (pivot table atau junction table) yang berisi kunci asing dari kedua tabel yang berelasi. Tabel perantara ini biasanya disebut siswa_mata_kuliah dan akan memiliki kolom siswa_id dan mata_kuliah_id (Silberschatz et al., 2020).

3. Konsep Indexing

Indexing adalah teknik optimasi database yang digunakan untuk mempercepat pengambilan data. Mirip dengan indeks di buku, indeks database memungkinkan sistem manajemen database (DBMS) untuk menemukan baris data dengan cepat tanpa harus memindai seluruh tabel. Ketika sebuah kolom diindeks, DBMS membuat struktur data terpisah (misalnya, B-tree) yang menyimpan nilai-nilai dari kolom tersebut bersama dengan pointer ke lokasi fisik baris data yang sesuai.

Kapan menggunakan indeks?

- Pada kolom yang sering digunakan dalam klausa WHERE (untuk pencarian).
- Pada kolom yang digunakan dalam klausa JOIN (untuk relasi antar tabel).
- Pada kolom yang digunakan dalam klausa ORDER BY atau GROUP BY.
- Kunci utama secara otomatis diindeks.

Kapan tidak menggunakan indeks?

- Pada tabel kecil yang jarang diakses.
- Pada kolom yang sering diperbarui (indeks harus diperbarui setiap kali data berubah, yang menambah overhead).
- Pada kolom dengan banyak nilai duplikat (kardinalitas rendah), kecuali jika dikombinasikan dengan kolom lain.

Meskipun indeks dapat mempercepat operasi baca, mereka juga menambah overhead pada operasi tulis (*INSERT, UPDATE, DELETE*) karena indeks harus diperbarui. Oleh karena itu, penggunaan indeks harus dipertimbangkan secara cermat untuk mencapai keseimbangan antara kinerja baca dan tulis (Ramakrishnan & Gehrke, 2018).

B. PEMBUATAN DATABASE SISTEM INFORMASI

Setelah memahami konsep dasar, langkah selanjutnya adalah menerjemahkan perancangan logis ke dalam implementasi fisik database. Dalam konteks CodeIgniter 4, kita akan menggunakan MySQL atau MariaDB sebagai sistem manajemen database relasional (RDBMS) yang populer.

1. Konfigurasi Koneksi Database

Sebelum dapat berinteraksi dengan database, CodeIgniter 4 perlu dikonfigurasi untuk mengetahui detail koneksi. Ini dilakukan melalui file .env dan app/Config/Database.php.

Contoh Konfigurasi di .env:

```
database.default.hostname = localhost
database.default.database = ci4
database.default.username = root
database.default.password = root
database.default.DBDriver = MySQLi
database.default.DBPrefix =
database.default.port = 3306
```

Pastikan untuk mengganti nama_database_anda dengan nama database yang akan Anda buat. Pengaturan ini akan secara otomatis dimuat oleh CodeIgniter 4.

2. Pembuatan Database Secara Manual (Awal)

Meskipun CodeIgniter 4 memiliki fitur migrasi untuk membuat tabel, database itu sendiri harus dibuat terlebih dahulu. Ini dapat dilakukan melalui antarmuka seperti phpMyAdmin, Adminer, atau melalui command line SQL.

Langkah-langkah pembuatan database melalui phpMyAdmin:

1. Akses phpMyAdmin melalui browser Anda (biasanya http://localhost/phpmyadmin).
2. Login dengan kredensial database Anda (misalnya, root tanpa password untuk XAMPP/Laragon).
3. Klik tab Databases atau Basis Data.
4. Masukkan nama database yang diinginkan (misalnya, ci4_sistem_informasi) pada kolom Create database atau Buat basis data.
5. Pilih utf8mb4_general_ci sebagai collation untuk mendukung berbagai karakter, termasuk emoji.
6. Klik Create atau Buat.

Setelah database berhasil dibuat, CodeIgniter 4 dapat terhubung dan mulai mengelola tabel di dalamnya.

C. MIGRASI (MIGRATION) DAN SEEDER DI CI4

CodeIgniter 4 menyediakan fitur Migrasi dan Seeder yang sangat powerful untuk mengelola skema database dan mengisi data awal secara terprogram. Ini adalah praktik terbaik dalam pengembangan modern untuk memastikan konsistensi database di berbagai lingkungan dan memfasilitasi kolaborasi tim.

1. Migrasi (Migration)

Migrasi adalah cara untuk mengelola perubahan skema database Anda dari waktu ke waktu. Daripada menulis SQL secara manual, Anda mendefinisikan perubahan dalam file PHP. Setiap file migrasi mewakili satu set perubahan, seperti membuat tabel baru, menambahkan kolom, atau mengubah tipe kolom.

Keuntungan menggunakan Migrasi:

- **Kontrol Versi:** Perubahan skema database dapat dilacak dalam sistem kontrol versi (misalnya Git) bersama dengan kode aplikasi.
- **Konsistensi:** Memastikan bahwa semua pengembang dan lingkungan (pengembangan, staging, produksi) memiliki skema database yang sama.
- **Rollback:** Memungkinkan untuk mengembalikan perubahan skema database jika terjadi kesalahan.
- **Kolaborasi:** Mempermudah tim untuk bekerja pada database yang sama tanpa konflik manual.

Membuat File Migrasi: Untuk membuat file migrasi baru, gunakan perintah Spark CLI:
php spark make:migration NamaMigrasiAnda
Contoh: php spark make:migration CreateUsersTable
Ini akan membuat file baru di app/Database/Migrations/ dengan nama seperti 2025-10-15-123456_CreateUsersTable.php. Setiap file migrasi memiliki dua metode utama:

- up(): Berisi logika untuk menerapkan perubahan skema (misalnya, membuat tabel).
- down(): Berisi logika untuk mengembalikan perubahan skema (misalnya, menghapus tabel).

Contoh Struktur File Migrasi:

```php
<?php
namespace App\Database\Migrations;
use CodeIgniter\Database\Migration;
class CreateUsersTable extends Migration
{
    public function up()
    {
        $this->forge->addField([
            'id' => [ 'type' => 'INT','constraint' => 5,'unsigned' => true, 'auto_increment' => true, ],
            'nama_lengkap' => [ 'type' => 'VARCHAR', 'constraint' => '100', ],
            'email' => [ 'type' => 'VARCHAR', 'constraint' => '100', 'unique' => true, ],
            'password' => [ 'type' => 'VARCHAR', 'constraint' => '255', ],
            'created_at' => [ 'type' => 'DATETIME', 'null' => true, ],
            'updated_at' => [ 'type' => 'DATETIME', 'null' => true, ],
        ]);
        $this->forge->addKey('id', true); // Primary Key
        $this->forge->createTable('users');
    }
    public function down()
    {
        $this->forge->dropTable('users');
    }
}
```

Dalam contoh di atas, $this->forge adalah objek yang disediakan oleh CodeIgniter untuk membangun skema database secara programatis.

Menjalankan Migrasi: Untuk menerapkan semua migrasi yang belum dijalankan:

php spark migrate

Untuk mengembalikan migrasi terakhir:

php spark migrate:rollback

Untuk melihat status migrasi:

php spark migrate:status

2. Seeder

Seeder digunakan untuk mengisi database dengan data awal atau data dummy. Ini sangat berguna untuk pengujian, demonstrasi, atau untuk mengisi data konfigurasi default pada aplikasi baru.

Membuat File Seeder: Untuk membuat file seeder baru, gunakan perintah Spark CLI:

php spark make:seeder NamaSeederAnda

Contoh: *php spark make:seeder UserSeeder*

Ini akan membuat file baru di app/Database/Seeds/ dengan nama seperti UserSeeder.php. Setiap file seeder memiliki metode run() yang berisi logika untuk memasukkan data.

Contoh Struktur File Seeder:

```php
<?php
namespace App\Database\Seeds;
use CodeIgniter\Database\Seeder;
class UserSeeder extends Seeder
{
    public function run()
    {
        $data = [ [ 'nama_lengkap' => 'Admin Utama',
        'email' => 'admin@example.com',
        'password' => password_hash('password123', PASSWORD_DEFAULT),
        'created_at' => date('Y-m-d H:i:s'),
        'updated_at' => date('Y-m-d H:i:s'), ],
        [ 'nama_lengkap' => 'John Doe',
        'email' => 'john.doe@example.com',
        'password' => password_hash('rahasia456', PASSWORD_DEFAULT),
        'created_at' => date('Y-m-d H:i:s'),
        'updated_at' => date('Y-m-d H:i:s'), ], ];
// Menggunakan Query Builder untuk insert data
$this->db->table('users')->insertBatch($data); } }
```

Menjalankan Seeder: Untuk menjalankan seeder tertentu:

php spark db:seed UserSeeder

Anda juga dapat membuat DatabaseSeeder.php (yang dibuat secara default) untuk memanggil seeder lain, sehingga Anda dapat menjalankan semua seeder dengan satu perintah:

```php
<?php
namespace App\Database\Seeds;
use CodeIgniter\Database\Seeder;
class DatabaseSeeder extends Seeder
{
    public function run()
    {
        $this->call('UserSeeder');
        $this->call('KategoriSeeder');
        $this->call('ProdukSeeder');
    }
}
```

Kemudian jalankan:

php spark db:seed

(CodeIgniter akan secara otomatis mencari DatabaseSeeder jika tidak ada nama seeder yang ditentukan).

D. PRAKTIK: MEMBUAT STRUKTUR TABEL UTAMA (USERS, DATA MASTER, TRANSAKSI)

Bagian ini akan memandu Anda dalam membuat struktur tabel utama yang umum ditemukan dalam sistem informasi menggunakan fitur migrasi CodeIgniter 4. Kita akan membuat tabel users, tabel data master (contoh: kategori_produk), dan tabel transaksi (contoh: penjualan dan detail_penjualan).

1. Tabel users

Tabel users adalah fondasi untuk sistem autentikasi dan otorisasi.

Langkah 1: Buat Migrasi untuk Tabel users

php spark make:migration CreateUsersTable

Langkah 2: Edit File Migrasi CreateUsersTable.php Tambahkan definisi kolom seperti yang telah dicontohkan sebelumnya di bagian E.1. Pastikan ada kolom id sebagai primary key, nama_lengkap, email (unik), password, serta created_at dan updated_at untuk timestamp otomatis.

```php
// app/Database/Migrations/2025-10-15-xxxxxx_CreateUsersTable.php ?>
<?php
namespace App\Database\Migrations;
use CodeIgniter\Database\Migration;
class CreateUsersTable extends Migration
{
    public function up()
    {
        $this->forge->addField([
            'id' => [ 'type' => 'INT', 'constraint' => 11, 'unsigned' => true, 'auto_increment' => true, ],
            'nama_lengkap' => [ 'type' => 'VARCHAR', 'constraint' => '100', ],
            'email' => [ 'type' => 'VARCHAR', 'constraint' => '100', 'unique' => true, ],
            'password' => [ 'type' => 'VARCHAR', 'constraint' => '255', ],
            'role' => [ // Contoh kolom role untuk otorisasi
                'type' => 'ENUM',
                'constraint' => ['admin', 'user'],
                'default' => 'user', ],
                'is_active' => [ 'type' => 'BOOLEAN', 'default' => true, ],
                'created_at' => [ 'type' => 'DATETIME', 'null' => true, ],
                'updated_at' => [ 'type' => 'DATETIME', 'null' => true, ],
            ]);
            $this->forge->addKey('id', true);
            $this->forge->createTable('users');
    }
    public function down()
    {
        $this->forge->dropTable('users');
    }
}
```

2. Tabel Data Master (Contoh: kategori_produk)

Tabel data master menyimpan data referensi yang sering digunakan.

Langkah 1: Buat Migrasi untuk Tabel kategori_produk

php spark make:migration CreateKategoriProdukTable

Langkah 2: Edit File Migrasi CreateKategoriProdukTable.php

```php
// app/Database/Migrations/2025-10-15-xxxxxx_CreateKategoriProdukTable.php
<?php
namespace App\Database\Migrations;
use CodeIgniter\Database\Migration;
class CreateKategoriProdukTable extends Migration
{
    public function up()
    {
        $this->forge->addField([
            'id' => [ 'type' => 'INT', 'constraint' => 5, 'unsigned' => true, 'auto_increment' => true, ],
            'nama_kategori' => [ 'type' => 'VARCHAR', 'constraint' => '100', 'unique' => true, ],
            'deskripsi' => [ 'type' => 'TEXT', 'null' => true, ],
            'created_at' => [ 'type' => 'DATETIME', 'null' => true, ],
            'updated_at' => [ 'type' => 'DATETIME', 'null' => true, ], ]);
        $this->forge->addKey('id', true); $this->forge->createTable('kategori_produk');
    }
    public function down()
    {
        $this->forge->dropTable('kategori_produk');
    }
}
```

3. Tabel Transaksi (Contoh: penjualan dan detail_penjualan)

Tabel transaksi mencatat aktivitas bisnis. Biasanya terdiri dari tabel header (misalnya penjualan) dan tabel detail (misalnya detail_penjualan) dengan relasi one-to-many.

Langkah 1: Buat Migrasi untuk Tabel penjualan

php spark make:migration CreatePenjualanTable

Langkah 2: Edit File Migrasi CreatePenjualanTable.php

```php
// app/Database/Migrations/2025-10-15-xxxxxx_CreatePenjualanTable.php
<?php
namespace App\Database\Migrations;
use CodeIgniter\Database\Migration;
class CreatePenjualanTable extends Migration
{
    public function up()
    {
        $this->forge->addField([
            'id' => [ 'type' => 'INT', 'constraint' => 11, 'unsigned' => true, 'auto_increment' => true, ],
            'kode_penjualan' => [ 'type' => 'VARCHAR', 'constraint' => '20', 'unique' => true, ],
            'tanggal_penjualan' => [ 'type' => 'DATE', ],
            'total_harga' => [ 'type' => 'DECIMAL', 'constraint' => '10,2', 'default' => 0.00, ],
            'user_id' => [ // Foreign Key ke tabel users
                'type' => 'INT', 'constraint' => 11, 'unsigned' => true, ],
            'created_at' => [ 'type' => 'DATETIME', 'null' => true, ],
            'updated_at' => [ 'type' => 'DATETIME', 'null' => true, ], ]);
        $this->forge->addKey('id', true);
        $this->forge->addForeignKey('user_id', 'users', 'id', 'CASCADE', 'CASCADE');
        // Relasi ke tabel users
        $this->forge->createTable('penjualan');
    }
    public function down()
    {
        $this->forge->dropTable('penjualan');
    }
}
```

Langkah 3: Buat Migrasi untuk Tabel detail_penjualan

php spark make:migration CreateDetailPenjualanTable

Langkah 4: Edit File Migrasi CreateDetailPenjualanTable.php

```php
// app/Database/Migrations/2025-10-15-xxxxxx_CreateDetailPenjualanTable.php
<?php
namespace App\Database\Migrations;
use CodeIgniter\Database\Migration;
class CreateDetailPenjualanTable extends Migration
{
    public function up()
    {
        $this->forge->addField([
            'id' => [ 'type' => 'INT', 'constraint' => 11, 'unsigned' => true, 'auto_increment' => true, ],
            'penjualan_id' => [ // Foreign Key ke tabel penjualan
                'type' => 'INT', 'constraint' => 11, 'unsigned' => true, ],
            'produk_id' => [ // Foreign Key ke tabel produk (asumsi ada tabel produk)
                'type' => 'INT', 'constraint' => 11, 'unsigned' => true, ],
            'jumlah' => [ 'type' => 'INT', 'constraint' => 5, ],
            'harga_satuan' => [ 'type' => 'DECIMAL', 'constraint' => '10,2', ],
            'subtotal' => [ 'type' => 'DECIMAL', 'constraint' => '10,2', ],
            'created_at' => [ 'type' => 'DATETIME', 'null' => true, ],
            'updated_at' => [ 'type' => 'DATETIME', 'null' => true, ],
        ]);
        $this->forge->addKey('id', true);
        $this->forge->addForeignKey('penjualan_id', 'penjualan', 'id', 'CASCADE', 'CASCADE');
        // Asumsi ada tabel 'produk' yang akan dibuat nanti //
        $this->forge->addForeignKey('produk_id', 'produk', 'id', 'CASCADE', 'CASCADE');
        $this->forge->createTable('detail_penjualan'); }
        public function down() { $this->forge->dropTable('detail_penjualan');
        }
    }
```

5. Menjalankan Semua Migrasi

Setelah semua file migrasi dibuat, jalankan perintah berikut untuk membuat semua tabel di database Anda:

php spark migrate

Perintah ini akan menjalankan semua migrasi yang belum diterapkan secara berurutan. Anda dapat memverifikasi struktur tabel di phpMyAdmin atau alat database lainnya.

6. Membuat Seeder untuk Data Awal

Untuk mengisi data awal, kita bisa membuat seeder untuk tabel users dan kategori_produk.

Langkah 1: Buat Seeder untuk users

php spark make:seeder UserSeeder

Isi file UserSeeder.php seperti contoh di bagian E.2.

Langkah 2: Buat Seeder untuk kategori_produk

php spark make:seeder KategoriProdukSeeder

Edit file KategoriProdukSeeder.php:

```php
// app/Database/Seeds/KategoriProdukSeeder.php
<?php
namespace App\Database\Seeds;
use CodeIgniter\Database\Seeder;
class KategoriProdukSeeder extends Seeder
{
    public function run()
    {
        $data = [
        ['nama_kategori' => 'Elektronik', 'deskripsi' => 'Produk-produk elektronik rumah tangga dan gadget.'],
        ['nama_kategori' => 'Pakaian', 'deskripsi' => 'Berbagai jenis pakaian untuk pria, wanita, dan anak-anak.'],
        ['nama_kategori' => 'Makanan & Minuman', 'deskripsi' => 'Produk makanan dan minuman kemasan.'], ];
        $this->db->table('kategori_produk')->insertBatch($data);
    }
}
```

Langkah 3: Panggil Seeder dari DatabaseSeeder.php Edit app/Database/Seeds/DatabaseSeeder.php untuk memanggil seeder yang telah dibuat:

```php
// app/Database/Seeds/DatabaseSeeder.php
<?php
namespace App\Database\Seeds;
use CodeIgniter\Database\Seeder;
class DatabaseSeeder extends Seeder
{
    public function run()
    { $this->call('UserSeeder'); $this->call('KategoriProdukSeeder'); } }
```

Langkah 4: Jalankan Semua Seeder

php spark db:seed

Setelah menjalankan perintah ini, tabel users dan kategori_produk akan terisi dengan data awal yang telah Anda definisikan.

Melalui praktik ini, Anda telah berhasil merancang dan mengimplementasikan struktur database dasar untuk sistem informasi menggunakan fitur migrasi dan seeder CodeIgniter 4. Ini adalah langkah krusial dalam membangun aplikasi web yang terstruktur, mudah dikelola, dan siap untuk pengembangan lebih lanjut.

E. RANGKUMAN

Pembahasan ini telah menguraikan fondasi esensial dalam perancangan dan implementasi database relasional, khususnya dalam konteks pengembangan sistem informasi menggunakan CodeIgniter 4. Pemahaman yang komprehensif tentang struktur data, relasi antar entitas, dan teknik optimasi adalah kunci untuk membangun aplikasi yang efisien dan terukur.

1. **Konsep Tabel, Relasi, dan Indexing:**
    - Tabel adalah unit dasar penyimpanan data, merepresentasikan entitas dengan baris (record) dan kolom (field), dilengkapi dengan Primary Key untuk identifikasi unik.

- Relasi antar tabel (One-to-One, One-to-Many, Many-to-Many) dibangun menggunakan Foreign Key untuk menghindari redundansi dan menjaga integritas data, dengan tabel perantara diperlukan untuk relasi Many-to-Many.
- Indexing adalah teknik optimasi untuk mempercepat pengambilan data, ideal untuk kolom yang sering dicari, digabungkan, atau diurutkan, namun perlu dipertimbangkan karena menambah overhead pada operasi tulis.

2. **Pembuatan Database Sistem Informasi:**
   - Konfigurasi koneksi database di CodeIgniter 4 dilakukan melalui file .env dan app/Config/Database.php untuk menghubungkan aplikasi dengan RDBMS (misalnya MySQL/MariaDB).
   - Database itu sendiri harus dibuat secara manual terlebih dahulu melalui antarmuka seperti phpMyAdmin atau command line SQL sebelum CodeIgniter dapat berinteraksi dengannya.
3. **Migrasi (Migration) dan Seeder di CI4:**
   - Migrasi adalah fitur penting untuk mengelola perubahan skema database secara terprogram, memastikan konsistensi, kontrol versi, dan kemampuan rollback. File migrasi mendefinisikan perubahan skema dalam metode up() dan down().
   - Seeder digunakan untuk mengisi database dengan data awal atau data dummy, sangat berguna untuk pengujian dan inisialisasi aplikasi, dengan metode run() yang berisi logika penyisipan data.
4. **Praktik Pembuatan Struktur Tabel Utama:**
   - Implementasi struktur tabel utama seperti users, kategori_produk (data master), penjualan, dan detail_penjualan dilakukan menggunakan migrasi CodeIgniter 4.
   - Tabel users mencakup kolom untuk autentikasi dan otorisasi, sementara kategori_produk berfungsi sebagai data referensi.
   - Tabel transaksi seperti penjualan dan detail_penjualan menunjukkan relasi One-to-Many, dengan detail_penjualan merujuk ke penjualan dan produk melalui Foreign Key.
   - Setelah migrasi dijalankan, seeder digunakan untuk mengisi data awal pada tabel users dan kategori_produk, memastikan database siap digunakan dengan data dasar.

BAB 6
FORM INPUT DAN VALIDASI DATA

Dalam pengembangan sistem informasi berbasis web, interaksi antara pengguna dan aplikasi seringkali melibatkan pengisian formulir (*form input*). Formulir ini menjadi gerbang utama bagi pengguna untuk memasukkan data ke dalam sistem, mulai dari data pribadi, data transaksi, hingga data master. Oleh karena itu, kemampuan untuk merancang dan mengimplementasikan form input yang efektif, aman, dan user-friendly adalah keterampilan fundamental yang wajib dikuasai oleh setiap pengembang aplikasi web.

Bab 6 ini akan membawa Anda menyelami lebih dalam tentang bagaimana membangun form input yang robust menggunakan CodeIgniter 4. Kita tidak hanya akan belajar cara membuat form secara manual, tetapi juga akan mengeksplorasi bagaimana helper form yang disediakan oleh CodeIgniter 4 dapat mempercepat proses pengembangan dan mengurangi potensi kesalahan. Namun, membuat form saja tidak cukup. Data yang dimasukkan oleh pengguna harus divalidasi untuk memastikan kebenarannya dan mencegah data yang tidak valid atau berbahaya masuk ke dalam database. Di sinilah peran penting validasi sisi server (*server-side validation*) akan dibahas secara mendalam, termasuk bagaimana menampilkan pesan kesalahan yang jelas kepada pengguna.

Selain input teks, banyak aplikasi web modern memerlukan kemampuan untuk mengunggah file, seperti gambar profil, dokumen, atau lampiran lainnya. Bab ini juga akan membahas secara komprehensif tentang implementasi fitur unggah file, lengkap dengan mekanisme validasi untuk memastikan file yang diunggah sesuai dengan kriteria yang ditetapkan, baik dari segi jenis, ukuran, maupun keamanannya.

Aspek keamanan adalah prioritas utama dalam pengembangan aplikasi web. Oleh karena itu, kita akan mempelajari teknik sanitasi input (*filterin dan filterout*) yang krusial untuk melindungi aplikasi dari berbagai ancaman keamanan siber, seperti serangan *Cross-Site Scripting* (XSS) dan *SQL Injection*. Dengan memahami dan menerapkan sanitasi input, Anda dapat membangun aplikasi yang lebih tangguh dan aman.

Pada akhirnya, semua konsep yang telah dipelajari akan diintegrasikan dalam sesi praktik, di mana Anda akan diajak untuk membangun form input data yang lengkap, misalnya untuk data pegawai, siswa, atau item. Praktik ini akan memberikan pengalaman langsung dalam menerapkan teori ke dalam kode nyata, sehingga Anda memiliki pemahaman yang kokoh tentang bagaimana mengelola input data pengguna secara efektif dan aman dalam sistem informasi berbasis web menggunakan CodeIgniter 4.

A. MEMBUAT FORM INPUT MANUAL DAN MENGGUNAKAN HELPER

Formulir input adalah elemen krusial dalam aplikasi web yang memungkinkan pengguna berinteraksi dengan sistem dengan memasukkan data. Dalam CodeIgniter 4, kita dapat membuat form input dengan dua pendekatan utama: secara manual menggunakan HTML murni, atau dengan memanfaatkan Form Helper yang disediakan oleh framework. Kedua pendekatan ini memiliki kelebihan dan kekurangannya masing-masing, dan pemahaman tentang keduanya akan membantu pengembang memilih metode yang paling sesuai untuk kebutuhan spesifik.

1. Membuat Form Input Manual dengan HTML

Pendekatan manual melibatkan penulisan seluruh elemen form menggunakan sintaks HTML standar. Ini memberikan kontrol penuh kepada pengembang atas struktur dan atribut setiap elemen form.

```
<!-- app/Views/form_pegawai_manual.php -->
<form action=/pegawai/simpan method=post>
    <label for=nama>Nama Pegawai:</label><br>
    <input type=text id=nama name=nama value=<?= old('nama') ?>><br><br>
    <label for=nip>NIP:</label><br>
    <input type=text id=nip name=nip value=<?= old('nip') ?>><br><br>
    <label for=email>Email:</label><br>
    <input type=email id=email name=email value=<?= old('email') ?>><br><br>
    <label for=jabatan>Jabatan:</label><br>
    <select id=jabatan name=jabatan>
        <option value=>Pilih Jabatan</option>
        <option value=Manager <?= old('jabatan') == 'Manager' ? 'selected' : '' ?>>Manager</option>
        <option value=Staff <?= old('jabatan') == 'Staff' ? 'selected' : '' ?>>Staff</option>
    </select><br><br>
    <input type=submit value=Simpan Data>
</form>
```

Dalam contoh di atas, setiap elemen *<label>, <input>,* dan *<select>* ditulis secara eksplisit. Penggunaan *<?= old('nama') ?>* adalah fitur CodeIgniter 4 yang sangat berguna untuk mempertahankan nilai input sebelumnya jika terjadi kesalahan validasi, sehingga pengguna tidak perlu mengisi ulang seluruh form (CodeIgniter 4 User Guide, 2023).

2. Menggunakan Form Helper CodeIgniter 4

Form Helper CodeIgniter 4 menyediakan serangkaian fungsi yang mempermudah pembuatan elemen form. Helper ini secara otomatis menangani atribut seperti name, id, dan value, serta dapat membantu dalam penanganan nilai old() dan error validasi. Untuk menggunakan Form Helper, kita perlu memuatnya terlebih dahulu di controller atau di file konfigurasi.

```
// app/Controllers/Pegawai.php
<?php
namespace App\Controllers;
use CodeIgniter\Controller;
class Pegawai extends Controller {
    public function index() {
        helper(['form']); // Memuat Form Helper return view('form_pegawai_helper');
        }
    public function simpan() {
        // Logika penyimpanan data
        }
}
```

Setelah helper dimuat, kita bisa menggunakan fungsi-fungsi seperti form_open(), form_input(), form_label(), form_dropdown(), dan form_submit().

```
<!-- app/Views/form_pegawai_helper.php -->
<?= form_open('/pegawai/simpan') ?>
<?= form_label('Nama Pegawai:', 'nama') ?><br>
<?= form_input(['name' => 'nama', 'id' => 'nama', 'value' => old('nama')]) ?><br><br>
<?= form_label('NIP:', 'nip') ?><br>
<?= form_input(['name' => 'nip', 'id' => 'nip', 'value' => old('nip')]) ?><br><br>
<?= form_label('Email:', 'email') ?><br>
<?= form_input(['type' => 'email', 'name' => 'email', 'id' => 'email', 'value' => old('email')]) ?><br>
<?= form_label('Jabatan:', 'jabatan') ?><br>
<?php $options = ['' => 'Pilih Jabatan', 'Manager' => 'Manager', 'Staff' => 'Staff',];
echo form_dropdown('jabatan', $options, old('jabatan')); ?><br><br>
<?= form_submit('submit', 'Simpan Data') ?>
<?= form_close() ?>
```

Penggunaan *Form Helper* dapat mengurangi boilerplate code dan meningkatkan konsistensi dalam penulisan form. Namun, untuk form yang sangat kompleks atau membutuhkan kustomisasi tinggi, penulisan manual mungkin lebih fleksibel.

B. SERVER-SIDE VALIDATION DAN PESAN ERROR

Validasi data adalah proses krusial untuk memastikan bahwa data yang diterima dari pengguna memenuhi kriteria yang ditetapkan sebelum disimpan ke database. Validasi sisi server (*server-side validation*) adalah lapisan keamanan dan integritas data yang tidak boleh diabaikan, karena validasi sisi klien (client-side validation) dapat dengan mudah dilewati oleh pengguna yang berniat jahat (OWASP, 2023). CodeIgniter 4 menyediakan pustaka validasi yang sangat powerful dan mudah digunakan.

1. Implementasi Validasi Sisi Server

Untuk menerapkan validasi, kita biasanya mendefinisikan aturan validasi dalam controller. CodeIgniter 4 menggunakan objek Validation untuk mengelola aturan dan pesan kesalahan.

```
<?php
namespace App\Controllers;
use CodeIgniter\Controller;
use App\Models\PegawaiModel; // Asumsi ada model PegawaiModel
class Pegawai extends Controller
{
    public function index()
    {
        helper(['form']);
        return view('form_pegawai_helper');
    }
    public function simpan()
    {
        helper(['form']);
        $model = new PegawaiModel();
        $rules = [
            'nama' => 'required|min_length[3]|max_length[255]',
            'nip' => 'required|numeric|exact_length[18]|is_unique[pegawai.nip]',
            'email' => 'required|valid_email|is_unique[pegawai.email]',
            'jabatan' => 'required|in_list[Manager,Staff]',
        ];
        $messages = [
            'nama' => [
                'required' => 'Nama pegawai harus diisi.',
                'min_length' => 'Nama pegawai minimal 3 karakter.',
                'max_length' => 'Nama pegawai maksimal 255 karakter.',
            ],
```

```
            'nip' => [
                'required' => 'NIP harus diisi.',
                'numeric' => 'NIP harus berupa angka.',
                'exact_length' => 'NIP harus terdiri dari 18 digit.',
                'is_unique' => 'NIP ini sudah terdaftar.',
            ],
            'email' => [
                'required' => 'Email harus diisi.',
                'valid_email' => 'Format email tidak valid.',
                'is_unique' => 'Email ini sudah terdaftar.',
            ],
            'jabatan' => [
                'required' => 'Jabatan harus dipilih.',
                'in_list' => 'Jabatan yang dipilih tidak valid.',
            ],
        ];
        if (! $this->validate($rules, $messages)) {
            return view('form_pegawai_helper', ['validation' => $this->validator,]);
        } // Jika validasi berhasil, simpan data
        $data = [
            'nama' => $this->request->getPost('nama'),
            'nip' => $this->request->getPost('nip'),
            'email' => $this->request->getPost('email'),
            'jabatan' => $this->request->getPost('jabatan'),
        ];
        $model->insert($data);
        return redirect()->to('/pegawai')->with('success', 'Data pegawai berhasil disimpan.');
    }
}
```

Dalam kode di atas, *$rules* mendefinisikan aturan validasi untuk setiap *field input*, seperti *required, min_length, max_length, numeric, exact_length, valid_email*, dan *is_unique*. *$messages* menyediakan pesan kesalahan kustom yang akan ditampilkan kepada pengguna jika validasi gagal. Metode *$this->validate()* akan menjalankan validasi. Jika gagal, ia akan mengembalikan false, dan kita dapat meneruskan objek *$this->validator* ke view untuk menampilkan pesan kesalahan (CodeIgniter 4 User Guide, 2023).

2. Menampilkan Pesan Error di View

Pesan error dapat ditampilkan di view menggunakan fungsi *validation_list_errors()* untuk semua error, atau *display_errors()* untuk error spesifik pada satu field.

```
<!-- app/Views/form_pegawai_helper.php (lanjutan) --> <!-- ... bagian form sebelumnya ... -->
<?php if (isset($validation)): ?>
    <div class=alert alert-danger>
        <?= $validation->listErrors() ?>
    </div>
<?php endif; ?>
<!-- Atau untuk error spesifik per field -->
<label for=nama>Nama Pegawai:</label><br>
<?= form_input(['name' => 'nama', 'id' => 'nama', 'value' => old('nama')]) ?><br>
<?php if (isset($validation) && $validation->hasError('nama')): ?>
    <span class=text-danger><?= $validation->getError('nama') ?></span>
<?php endif; ?><br><br>
<!-- ... untuk field lainnya ... -->
```

Dengan cara ini, pengguna akan mendapatkan umpan balik yang jelas tentang kesalahan input mereka, membantu mereka memperbaiki data sebelum pengiriman ulang.

C. FILE UPLOAD DAN VALIDASINYA

Mengunggah file adalah fitur umum dalam banyak aplikasi web, seperti mengunggah foto profil, dokumen, atau lampiran lainnya. CodeIgniter 4 menyediakan mekanisme yang kuat dan aman untuk menangani unggahan file, termasuk validasi yang komprehensif.

1. Konfigurasi Form untuk File Upload

Untuk mengizinkan unggahan file, form HTML harus memiliki atribut enctype=multipart/form-data.

```
<!-- app/Views/form_upload.php -->
<?= form_open_multipart('/upload/do_upload') ?>
    <label for=dokumen>Pilih Dokumen:</label><br>
    <input type=file name=dokumen id=dokumen><br><br>
    <input type=submit value=Upload File>
<?= form_close() ?>
```

2. Menangani File Upload di Controller

Di controller, kita menggunakan objek *$this->request->getFile()* untuk mendapatkan instance dari file yang diunggah.

```
<?php
namespace App\Controllers;
use CodeIgniter\Controller;
class Upload extends Controller
{
    public function index()
    {
        helper(['form']);
        return view('form_upload');
    }
    public function do_upload()
    {
        helper(['form']);
        $validationRule = [
            'dokumen' => [
                'label' => 'Dokumen',
                'rules' => 'uploaded[dokumen]|max_size[dokumen,1024]|ext_in[dokumen,pdf,doc,docx]
                'errors' => [
                    'uploaded' => 'Anda harus memilih file untuk diunggah.',
                    'max_size' => 'Ukuran file terlalu besar (maksimal 1MB).',
                    'ext_in' => 'Hanya file PDF, DOC, atau DOCX yang diizinkan.',
                    'mime_in' => 'Tipe file tidak valid.',
                ],
            ],
        ];
        if (! $this->validate($validationRule)) {
            return view('form_upload', [
                'validation' => $this->validator,
            ]);
        }
```

```
        $file = $this->request->getFile('dokumen');
        if ($file->isValid() && ! $file->hasMoved()) {
            $newName = $file->getRandomName();
            // Generate nama unik
            $file->move(WRITEPATH . 'uploads', $newName);
            // Pindahkan file ke folder uploads
            return redirect()->to('/upload')->with('success',
            'File berhasil diunggah dengan nama: ' . $newName);
        } else {
            return redirect()->to('/upload')->with('error', $file->getErrorString());
        }
    }
}
```

3. Validasi File Upload

Validasi file upload sangat penting untuk keamanan dan integritas sistem. CodeIgniter 4 menyediakan aturan validasi khusus untuk file:

- uploaded[field_name]: Memastikan file telah diunggah.
- max_size[field_name,size_in_kb]: Membatasi ukuran file.
- ext_in[field_name,ext1,ext2,...]: Membatasi ekstensi file.
- mime_in[field_name,mime1,mime2,...]: Membatasi tipe MIME file.

Setelah validasi berhasil, file dapat dipindahkan ke direktori yang aman menggunakan $file->move(). Penting untuk tidak menyimpan file yang diunggah langsung di direktori publik (public/) untuk alasan keamanan, melainkan di direktori writable/uploads atau lokasi lain yang tidak dapat diakses langsung melalui URL (CodeIgniter 4 User Guide, 2023).

D. SANITASI INPUT (FILTERIN, FILTEROUT)

Sanitasi input adalah proses membersihkan atau memfilter data yang diterima dari pengguna untuk menghilangkan karakter atau kode berbahaya yang dapat menyebabkan kerentanan keamanan seperti *Cross-Site Scripting* (XSS) atau *SQL Injection*. Ini adalah langkah penting dalam membangun aplikasi web yang aman (Bishop, 2015). CodeIgniter 4 menyediakan beberapa mekanisme untuk sanitasi.

1. Filterin (Sanitasi Saat Menerima Input)

Filterin dilakukan saat aplikasi menerima data dari pengguna, sebelum data tersebut diproses atau disimpan ke database. Tujuannya adalah untuk menghilangkan potensi ancaman sejak dini.

a. Menggunakan esc() untuk Output HTML

Meskipun bukan sanitasi input secara langsung, esc() adalah fungsi penting untuk mencegah XSS saat menampilkan data. Setiap data yang berasal dari pengguna dan akan ditampilkan di view harus melalui fungsi esc().

```
<!-- app/Views/data_pegawai.php -->
<p>Nama: <?= esc($pegawai->nama, 'html') ?></p>
<p>Email: <?= esc($pegawai->email, 'html') ?></p>
```

Fungsi esc() akan mengonversi karakter khusus HTML (seperti <, >, &, , ') menjadi entitas HTML yang aman, sehingga browser tidak akan menginterpretasikannya sebagai kode HTML atau JavaScript (CodeIgniter 4 User Guide, 2023).

b. Menggunakan filter_var() atau strip_tags()

Untuk sanitasi yang lebih agresif pada input, PHP menyediakan fungsi seperti filter_var() dengan filter FILTER_SANITIZE_STRING (meskipun deprecated di PHP 8.1, alternatifnya adalah htmlspecialchars() atau strip_tags()) atau strip_tags().

```php
// app/Controllers/Pegawai.php (dalam metode simpan)
$nama = strip_tags($this->request->getPost('nama'));
$email = filter_var(
    $this->request->getPost('email'),
    FILTER_SANITIZE_EMAIL
); // Atau menggunakan helper CodeIgniter
$nama_bersih = clean_input($this->request->getPost('nama')); // Asumsi ada helper kustom
```

2. Filterout (Sanitasi Saat Mengeluarkan Output)

Filterout dilakukan saat data diambil dari database dan akan ditampilkan kepada pengguna. Ini adalah lapisan pertahanan terakhir untuk memastikan bahwa data yang ditampilkan aman, terutama jika data tersebut mungkin telah terkontaminasi sebelumnya atau jika ada celah keamanan di tempat lain. Seperti yang disebutkan sebelumnya, esc() adalah metode utama untuk filterout di CodeIgniter 4.

3. Pencegahan SQL Injection

CodeIgniter 4 secara otomatis mencegah SQL Injection melalui penggunaan Query Builder dan Model. Ketika Anda menggunakan metode seperti insert(), update(), atau where() dengan parameter, CodeIgniter akan secara otomatis melakukan *escaping* pada nilai-nilai tersebut, sehingga mencegah injeksi kode SQL berbahaya (CodeIgniter 4 User Guide, 2023).

```php
// Contoh aman dengan Query Builder
$this->db->table('pegawai')->insert($data);
// Contoh aman dengan Model
$model->insert($data);
// Contoh aman dengan where
$this->db->table('pegawai')->where('nip', $nip)->get();
```

Menghindari penulisan query SQL mentah dengan string concatenation adalah praktik terbaik untuk mencegah SQL Injection.

E. PRAKTIK: FORM INPUT DATA PEGAWAI / SISWA / ITEM

Untuk mengkonsolidasikan pemahaman, mari kita terapkan semua konsep yang telah dibahas dalam sebuah praktik nyata: membuat form input data pegawai.

1. Perancangan Database (Revisi dari BAB 5)

Asumsikan kita memiliki tabel pegawai dengan struktur sebagai berikut:

- id (INT, Primary Key, Auto Increment)
- nama (VARCHAR(255))
- nip (VARCHAR(18), UNIQUE)
- email (VARCHAR(255), UNIQUE)
- jabatan (VARCHAR(100))
- foto_profil (VARCHAR(255), nullable)
- created_at, updated_at, deleted_at (DATETIME)

2. Membuat Model PegawaiModel

```php
// app/Models/PegawaiModel.php
<?php
namespace App\Models;
use CodeIgniter\Model;
class PegawaiModel extends Model {
    protected $table = 'pegawai';
    protected $primaryKey = 'id';
    protected $useAutoIncrement = true;
    protected $returnType = 'array'; // Atau 'App\Entities\Pegawai' jika menggunakan Entities
    protected $useSoftDeletes = true;
    protected $allowedFields = ['nama', 'nip', 'email', 'jabatan', 'foto_profil'];
    protected $useTimestamps = true;
    protected $createdField = 'created_at';
    protected $updatedField = 'updated_at';
    protected $deletedField = 'deleted_at';
    protected $validationRules = []; // Aturan validasi akan didefinisikan di Controller
    protected $validationMessages = []; protected $skipValidation = false;
}
```

3. Membuat Controller Pegawai

```php
// app/Controllers/Pegawai.php
<?php
namespace App\Controllers;
use CodeIgniter\Controller;
use App\Models\PegawaiModel;
class Pegawai extends Controller
{
    public function index()
    {
        helper(['form']);
        return view('pegawai/form_tambah');
    }
    public function simpan()
    {
        helper(['form']);
        $model = new PegawaiModel();
        $rules = [
            'nama' => 'required|min_length[3]|max_length[255]',
            'nip' => 'required|numeric|exact_length[18]|is_unique[pegawai.nip]',
            'email' => 'required|valid_email|is_unique[pegawai.email]',
            'jabatan' => 'required|in_list[Manager,Staff,Supervisor,Admin]',
            'foto_profil' => [
                'rules' => 'if_exist|uploaded[foto_profil]|
            max_size[foto_profil,1024]|ext_in[foto_profil,jpg,jpeg,png]|
            mime_in[foto_profil,image/jpg,image/jpeg,image/png]',
                'errors' => [
                    'uploaded' => 'Anda harus memilih file foto profil.',
                    'max_size' => 'Ukuran foto profil terlalu besar (maksimal 1MB).',
                    'ext_in' => 'Hanya file JPG, JPEG, atau PNG yang diizinkan.',
                    'mime_in' => 'Tipe file foto profil tidak valid.',
                ], ], ];
```

```
        $messages = [
            'nama' => [
                'required' => 'Nama pegawai harus diisi.',
                'min_length' => 'Nama pegawai minimal 3 karakter.',
                'max_length' => 'Nama pegawai maksimal 255 karakter.',
            ],
            'nip' => [
                'required' => 'NIP harus diisi.',
                'numeric' => 'NIP harus berupa angka.',
                'exact_length' => 'NIP harus terdiri dari 18 digit.',
                'is_unique' => 'NIP ini sudah terdaftar.',
            ],
            'email' => [
                'required' => 'Email harus diisi.',
                'valid_email' => 'Format email tidak valid.',
                'is_unique' => 'Email ini sudah terdaftar.',
            ],
            'jabatan' => [
                'required' => 'Jabatan harus dipilih.',
                'in_list' => 'Jabatan yang dipilih tidak valid.',
            ],
        ];
        if (! $this->validate($rules, $messages)) {
            return view('pegawai/form_tambah', [
                'validation' => $this->validator,
            ]);
        }
```

```
        $data = [
            'nama' => esc($this->request->getPost('nama'), 'html'), // Sanitasi input nama
            'nip' => esc($this->request->getPost('nip'), 'html'), // Sanitasi input NIP
            'email' => esc($this->request->getPost('email'), 'html'), // Sanitasi input email
            'jabatan' => esc($this->request->getPost('jabatan'), 'html'), // Sanitasi input jabatan
        ];
        $file = $this->request->getFile('foto_profil');
        if ($file && $file->isValid() && ! $file->hasMoved()) {
            $newName = $file->getRandomName();
            $file->move(WRITEPATH . 'uploads/pegawai', $newName);
            $data['foto_profil'] = $newName;
        }
        $model->insert($data);
        return redirect()->to('/pegawai')->with('success', 'Data pegawai berhasil disimpan.');
    }
}
```

4. Membuat View form_tambah.php

```
<!-- app/Views/pegawai/form_tambah.php -->
<!DOCTYPE html>
<html lang=en>

<head>
    <meta charset=UTF-8>
    <meta name=viewport content=width=device-width, initial-scale=1.0>
    <title>Form Tambah Pegawai</title>
    <style>
        .error {
            color: red;
            font-size: 0.9em;
        }

        .alert-danger {
            background-color: #f8d7da;
            color: #721c24;
            border: 1px solid #f5c6cb;
            padding: 10px;
            margin-bottom: 15px;
            border-radius: 5px;
        }

        .alert-success {
            background-color: #d4edda;
            color: #155724;
            border: 1px solid #c3e6cb;
            padding: 10px;
            margin-bottom: 15px;
            border-radius: 5px;
        }

        input[type=text],
        input[type=email],
        select,
        input[type=file] {
            width: 100%;
            padding: 8px;
            margin: 5px 0 15px 0;
            display: inline-block;
            border: 1px solid #ccc;
            border-radius: 4px;
            box-sizing: border-box;
        }

        input[type=submit] {
            background-color: #4CAF50;
            color: white;
            padding: 10px 20px;
            border: none;
            border-radius: 4px;
            cursor: pointer;
        }

        input[type=submit]:hover {
            background-color: #45a049;
        }
    </style>
</head>
```

```
<body>
    <h1>Form Tambah Data Pegawai</h1>
    <?php
    if (session()->getFlashdata('success')): ?>
        <div class=alert alert-success> <?= session()->getFlashdata('success') ?> </div>
    <?php endif; ?>
    <?php if (session()->getFlashdata('error')): ?>
        <div class=alert alert-danger> <?= session()->getFlashdata('error') ?> </div>
    <?php endif; ?>
    <?= form_open_multipart('/pegawai/simpan') ?>
    <div>
        <?= form_label('Nama Pegawai:', 'nama') ?>
        <?= form_input(['name' => 'nama', 'id' => 'nama', 'value' => old('nama')]) ?>
        <?php
        if (isset($validation) && $validation->hasError('nama')): ?>
            <span class=error><?= $validation->getError('nama') ?></span>
        <?php endif; ?>
    </div> '
    <div>
        <?= form_label('NIP:', 'nip') ?>
        <?= form_input(['name' => 'nip', 'id' => 'nip', 'value' => old('nip')]) ?>
        <?php if (isset($validation) && $validation->hasError('nip')): ?>
            <span class=error><?= $validation->getError('nip') ?></span>
        <?php endif; ?>
    </div>
```

```
    <div>
        <?= form_label('Email:', 'email') ?>
        <?= form_input(['type' => 'email',
        'name' => 'email', 'id' => 'email', 'value' => old('email')]) ?>
        <?php if (isset($validation) && $validation->hasError('email')): ?>
            <span class=error><?= $validation->getError('email') ?></span>
        <?php endif; ?>
    </div>
    <div> <?= form_label('Jabatan:', 'jabatan') ?>
        <?php $options = ['' => 'Pilih Jabatan', 'Manager' => 'Manager',
        'Staff' => 'Staff', 'Supervisor' => 'Supervisor', 'Admin' => 'Admin',];
        echo form_dropdown('jabatan', $options, old('jabatan')); ?>
        <?php if (isset($validation) && $validation->hasError('jabatan')): ?>
            <span class=error><?= $validation->getError('jabatan') ?></span>
        <?php endif; ?>
    </div>
    <div> <?= form_label('Foto Profil:', 'foto_profil') ?>
        <?= form_upload(['name' => 'foto_profil', 'id' => 'foto_profil']) ?>
        <?php if (isset($validation) && $validation->hasError('foto_profil')): ?>
            <span class=error><?= $validation->getError('foto_profil') ?></span>
        <?php endif; ?>
    </div>
    <div> <?= form_submit('submit', 'Simpan Data Pegawai') ?> </div> <?= form_close() ?>
</body>
</html>
```

5. Konfigurasi Routing

Tambahkan rute di app/Config/Routes.php:

```
// app/Config/Routes.php
$routes->get('/pegawai', 'Pegawai::index');
$routes->post('/pegawai/simpan', 'Pegawai::simpan');
```

Dengan praktik ini, Anda telah berhasil membangun form input data pegawai yang mencakup pembuatan form (menggunakan helper), validasi sisi server dengan pesan error, unggah file

dengan validasi, dan sanitasi input dasar. Ini adalah fondasi penting untuk setiap modul input data dalam sistem informasi.

F. RANGKUMAN

Pembahasan ini menguraikan secara komprehensif berbagai aspek penting dalam pengelolaan input data pada aplikasi web menggunakan CodeIgniter 4. Mulai dari pembuatan formulir hingga penanganan keamanan data, setiap tahapan dijelaskan dengan detail untuk memastikan integritas dan fungsionalitas sistem.

1. **Pembuatan Form Input:** Form input dapat dibuat secara manual menggunakan HTML murni, memberikan kontrol penuh atas struktur, atau dengan memanfaatkan Form Helper CodeIgniter 4 yang menyederhanakan penulisan kode dan meningkatkan konsistensi. Fitur old() sangat berguna untuk mempertahankan data input sebelumnya saat terjadi validasi gagal.
2. **Validasi Sisi Server:** Validasi data di sisi server adalah lapisan keamanan krusial untuk memastikan data memenuhi kriteria yang ditetapkan. CodeIgniter 4 menyediakan pustaka validasi yang kuat, memungkinkan definisi aturan (rules) dan pesan kesalahan (messages) kustom. Objek $this->validator diteruskan ke view untuk menampilkan umpan balik kesalahan kepada pengguna.
3. **Penanganan File Upload:** Mengunggah file memerlukan konfigurasi form dengan enctype=multipart/form-data. Di controller, file diakses melalui $this->request->getFile(). Validasi file mencakup pemeriksaan uploaded, max_size, ext_in, dan mime_in. File yang berhasil divalidasi kemudian dipindahkan ke direktori aman (misalnya, WRITEPATH . 'uploads') untuk mencegah akses langsung yang tidak diinginkan.
4. **Sanitasi Input (Filterin dan Filterout):** Sanitasi input adalah proses membersihkan data dari karakter atau kode berbahaya untuk mencegah kerentanan seperti XSS dan SQL Injection.
    - **Filterin** dilakukan saat menerima input, menggunakan fungsi seperti strip_tags() atau filter_var() (meskipun FILTER_SANITIZE_STRING sudah deprecated, htmlspecialchars() adalah alternatif yang baik).
    - **Filterout** dilakukan saat menampilkan data, dengan esc() sebagai fungsi utama untuk mengonversi karakter khusus HTML menjadi entitas yang aman.
    - **Pencegahan SQL Injection** secara otomatis ditangani oleh CodeIgniter 4 melalui Query Builder dan Model, yang melakukan *escaping* pada nilai-nilai parameter.
5. **Praktik Implementasi:** Konsep-konsep ini diintegrasikan dalam praktik pembuatan form input data pegawai. Ini melibatkan perancangan tabel database, pembuatan PegawaiModel, pengembangan Pegawai Controller yang menangani validasi, unggah file, dan sanitasi, serta pembuatan view form_tambah.php yang menampilkan form dan pesan kesalahan. Routing yang tepat juga dikonfigurasi untuk mengarahkan permintaan ke controller yang sesuai.

BAB 7
OPERASI CRUD (CREATE, READ, UPDATE, DELETE) LENGKAP

Dalam pengembangan sistem informasi berbasis web, kemampuan untuk mengelola data merupakan inti fungsionalitas yang tidak terpisahkan. Hampir setiap aplikasi, mulai dari sistem manajemen konten sederhana hingga aplikasi enterprise yang kompleks, memerlukan mekanisme untuk membuat, membaca, memperbarui, dan menghapus data. Operasi dasar ini dikenal dengan akronim CRUD (Create, Read, Update, Delete), dan merupakan tulang punggung interaksi antara pengguna dengan database. Tanpa kemampuan CRUD yang solid, sebuah sistem informasi tidak akan dapat berfungsi secara efektif dalam menyimpan, mengambil, dan memanipulasi informasi yang krusial bagi operasional organisasi.

Bab 7 ini akan membawa Anda lebih dalam ke implementasi operasi CRUD secara lengkap menggunakan CodeIgniter 4. Setelah pada bab-bab sebelumnya kita telah memahami dasar-dasar arsitektur MVC, konfigurasi database, dan pembuatan model, controller, serta view, kini saatnya kita menyatukan semua elemen tersebut untuk membangun modul pengelolaan data yang fungsional dan robust. Urgensi pembahasan ini terletak pada fakta bahwa sebagian besar waktu pengembangan aplikasi web dihabiskan untuk membangun dan menyempurnakan modul-modul CRUD. Penguasaan konsep dan praktik CRUD yang baik akan menjadi fondasi kuat bagi Anda untuk mengembangkan fitur-fitur yang lebih kompleks di masa mendatang.

Kita akan memulai dengan memahami pola umum implementasi CRUD di CodeIgniter 4, yang memanfaatkan fitur-fitur bawaan framework untuk mempermudah proses pengembangan. Selanjutnya, kita akan membahas bagaimana menangani data dalam jumlah besar melalui teknik *pagination*, serta bagaimana meningkatkan pengalaman pengguna dengan fitur *searching* dan *sorting* data. Aspek penting lainnya adalah operasi *edit* dan *update* data, yang memerlukan penanganan formulir dan validasi yang cermat untuk memastikan data yang disimpan tetap valid. Terakhir, kita akan mengeksplorasi perbedaan antara *soft delete* dan *hard delete*, dua pendekatan penting dalam manajemen siklus hidup data, dan kapan harus menggunakan masing-masing. Seluruh konsep ini akan diakhiri dengan praktik langsung membangun modul CRUD data master, memberikan Anda pengalaman nyata dalam mengaplikasikan teori ke dalam kode yang berfungsi. Dengan demikian, bab ini akan membekali Anda dengan keterampilan esensial untuk membangun aplikasi web yang mampu mengelola data secara dinamis dan efisien.

A. POLA CRUD DI CODEIGNITER 4

CodeIgniter 4 (CI4) menyediakan pendekatan yang terstruktur dan efisien untuk mengimplementasikan operasi CRUD (*Create, Read, Update, Delete*) berkat arsitektur *Model-View-Controller* (MVC) dan fitur-fitur bawaannya. Pola ini memisahkan logika bisnis (Model), presentasi data (View), dan penanganan permintaan pengguna (Controller), sehingga kode menjadi lebih terorganisir, mudah dipelihara, dan dapat diskalakan (Rahman & Islam, 2020).

1. Peran Model dalam CRUD

Model adalah komponen inti dalam operasi CRUD di CI4. Setiap tabel database biasanya direpresentasikan oleh satu Model. Model bertanggung jawab untuk berinteraksi langsung dengan database, melakukan operasi seperti mengambil data, menyimpan data baru, memperbarui data yang ada, dan menghapus data. CI4 menyediakan kelas CodeIgniter\Model yang menawarkan berbagai metode *helper* untuk mempermudah operasi database.

a. **Konfigurasi Model** Sebuah Model di CI4 biasanya meng-extend CodeIgniter\Model dan memiliki beberapa properti penting yang dikonfigurasi:

- protected $table: Nama tabel database yang akan dioperasikan.
- protected $primaryKey: Nama kolom *primary key* tabel.
- protected $useAutoIncrement: Menentukan apakah *primary key* menggunakan *auto-increment*.
- protected $returnType: Tipe data yang dikembalikan oleh metode Model (misalnya, array atau object dari Entity).
- protected $useSoftDeletes: Mengaktifkan fitur *soft delete*.
- protected $allowedFields: Daftar kolom yang diizinkan untuk diisi (mass assignment protection).
- protected $useTimestamps: Mengaktifkan *timestamp* otomatis (created_at, updated_at, deleted_at).

b. **Metode CRUD Dasar pada Model**

- **Create (Menyimpan Data Baru)**: Metode insert() atau save() digunakan untuk menambahkan baris baru ke tabel. Metode save() lebih fleksibel karena dapat digunakan untuk *insert* maupun *update*.
- **Read (Membaca Data)**: Metode find($id), findAll(), first(), where(), like(), join(), dan orderBy() adalah beberapa contoh metode untuk mengambil data.
- **Update (Memperbarui Data)**: Metode update($id, $data) atau save() digunakan untuk mengubah data yang sudah ada berdasarkan *primary key*.
- **Delete (Menghapus Data)**: Metode delete($id) digunakan untuk menghapus baris data. Jika useSoftDeletes diaktifkan, ini akan menjadi *soft delete*.

2. Peran Controller dalam CRUD

Controller bertindak sebagai jembatan antara Model dan View. Ia menerima permintaan dari pengguna (melalui *routing*), memanggil metode yang sesuai pada Model untuk memproses data, dan kemudian memuat View untuk menampilkan hasilnya.

a. **Struktur Controller CRUD** Sebuah Controller untuk modul CRUD biasanya memiliki metode-metode berikut:

- index(): Menampilkan daftar semua data (operasi Read).
- create(): Menampilkan formulir untuk membuat data baru.
- store(): Menerima data dari formulir create() dan menyimpannya ke database melalui Model.
- edit($id): Menampilkan formulir untuk mengedit data tertentu berdasarkan ID.
- update($id): Menerima data dari formulir edit() dan memperbarui data di database melalui Model.
- delete($id): Menghapus data tertentu dari database melalui Model.

3. Peran View dalam CRUD

View bertanggung jawab untuk menyajikan data kepada pengguna. Dalam konteks CRUD, View digunakan untuk menampilkan daftar data, formulir input untuk membuat atau mengedit data, serta pesan sukses atau error.

a. **Komponen View CRUD**

- **Daftar Data**: Biasanya berupa tabel HTML yang menampilkan data yang diambil dari Model.
- **Formulir Input**: Digunakan untuk operasi Create dan Update, berisi elemen input seperti teks, *textarea*, *dropdown*, dll.

- **Pesan Status**: Menampilkan notifikasi kepada pengguna setelah operasi CRUD berhasil atau gagal (misalnya, Data berhasil disimpan).

Pola MVC ini memastikan bahwa setiap bagian dari aplikasi memiliki tanggung jawab yang jelas, meminimalkan ketergantungan antar komponen, dan memfasilitasi pengembangan tim (Suryani et al., 2021).

B. MENANGANI PAGINATION, SEARCHING, DAN SORTING

Ketika berhadapan dengan data dalam jumlah besar, menampilkan semua data sekaligus dapat membebani server dan memperburuk pengalaman pengguna. Oleh karena itu, teknik *pagination*, *searching*, dan *sorting* menjadi sangat penting untuk mengelola dan menyajikan data secara efisien.

1. Pagination

Pagination adalah proses membagi kumpulan data yang besar menjadi halaman-halaman yang lebih kecil. CodeIgniter 4 menyediakan pustaka *pagination* yang kuat dan mudah digunakan.

a. **Implementasi Pagination di CodeIgniter 4**

1. **Konfigurasi Model**: Pastikan Model Anda menggunakan properti $perPage untuk menentukan jumlah item per halaman.

```
protected $perPage = 10; // 10 item per halaman
```

2. **Di Controller**: Gunakan metode paginate() dari Model dan inisialisasi pustaka *pagination*.

```
use CodeIgniter\Pager\Pager;
public function index()
{
    $model = new MyModel();
    $data['items'] = $model->paginate($this->perPage); // Mengambil data dengan pagination
    $data['pager'] = $model->pager; // Mengambil objek pager
    return view('items/index', $data); }
```

3. **Di View**: Tampilkan tautan *pagination* menggunakan objek $pager.

```
<?= $pager->links() ?>
```

CI4 juga memungkinkan kustomisasi tampilan *pagination* melalui *template*.

2. Searching

Fitur *searching* memungkinkan pengguna untuk mencari data spesifik berdasarkan kriteria tertentu. Ini meningkatkan kegunaan aplikasi secara signifikan.

a. **Implementasi Searching di CodeIgniter 4**

1. **Di Controller**: Tangkap *input* pencarian dari URL atau formulir. Gunakan metode like() atau where() pada Model.

```
public function index()
{ $model = new MyModel();
    $search = $this->request->getVar('q'); // Ambil query pencarian dari URL
    if ($search) { $model->like('nama_kolom', $search); // Cari di kolom 'nama_kolom'
        }
        $data['items'] = $model->paginate($this->perPage);
        $data['pager'] = $model->pager; $data['search'] = $search;
        // Kirim kembali query pencarian ke view
        return view('items/index', $data);
}
```

2. **Di View**: Tambahkan formulir pencarian dan tampilkan hasil.

```
<form action=<?= url_to('items') ?> method=get>
    <input type=text name=q value=<?= esc($search ?? '') ?> placeholder=Cari data...>
    <button type=submit>Cari</button>
</form>
```

Penting untuk melakukan sanitasi *input* pencarian untuk mencegah serangan *SQL Injection* (OWASP, 2023).

3. Sorting

Sorting memungkinkan pengguna untuk mengurutkan data berdasarkan satu atau lebih kolom, baik secara *ascending* maupun *descending*.

a. **Implementasi Sorting di CodeIgniter 4**

1. **Di Controller**: Tangkap parameter *sorting* dari URL (misalnya, sort_by dan sort_order).

```
public function index()
{ $model = new MyModel();
    $sortBy = $this->request->getVar('sort_by') ?? 'id'; // Default sort by 'id'
    $sortOrder = $this->request->getVar('sort_order') ?? 'asc'; // Default sort order 'asc'
    $model->orderBy($sortBy, $sortOrder); // ... (logic for searching and pagination) ...
    $data['items'] = $model->paginate($this->perPage);
    $data['pager'] = $model->pager; $data['sortBy'] = $sortBy;
    $data['sortOrder'] = $sortOrder;
    return view('items/index', $data);
}
```

2. **Di View**: Tambahkan tautan atau *dropdown* untuk memilih kolom dan urutan *sorting*.

```
<thead>
    <tr>
        <th>
            <a href=<?= url_to('items', [
                        'q' => $search,
                        'sort_by' => 'nama',
                        'sort_order' => ($sortBy == 'nama' && $sortOrder == 'asc' ? 'desc' : 'asc')
                    ]) ?>>
                Nama
            </a>
        </th>
        <th><a href=<?= url_to('items', [
                        'q' => $search,
                        'sort_by' => 'email',
                        'sort_order' => ($sortBy == 'email' && $sortOrder == 'asc' ? 'desc' : 'asc')
                    ]) ?>>
                Email
            </a>
        </th>
        <!-- ... other columns ... -->
    </tr>
</thead>
```

Kombinasi *pagination*, *searching*, dan *sorting* menciptakan antarmuka pengelolaan data yang sangat fungsional dan ramah pengguna.

C. EDIT DAN UPDATE DATA

Operasi *edit* dan *update* data adalah bagian krusial dari setiap sistem informasi, memungkinkan pengguna untuk memodifikasi informasi yang sudah ada. Proses ini melibatkan pengambilan data yang akan diedit, menampilkannya dalam formulir, dan kemudian menyimpan perubahan kembali ke database.

1. Alur Proses Edit dan Update

1. **Permintaan Edit**: Pengguna mengklik tombol Edit pada daftar data, yang akan mengirimkan permintaan ke Controller dengan ID data yang akan diedit.
2. **Pengambilan Data**: Controller menerima ID, kemudian memanggil Model untuk mengambil data lengkap dari *record* tersebut.
3. **Tampilan Formulir Edit**: Data yang diambil dari Model kemudian diteruskan ke View. View menampilkan formulir HTML yang sudah terisi dengan data yang ada, siap untuk dimodifikasi oleh pengguna.
4. **Pengiriman Data Update**: Pengguna memodifikasi data di formulir dan mengirimkannya (submit). Permintaan ini dikirim ke metode update() di Controller.
5. **Validasi Data**: Controller menerima data yang dikirim. Penting untuk melakukan validasi data di sisi server untuk memastikan integritas dan keamanan data (misalnya, memastikan *field* wajib terisi, format email benar, dll.).
6. **Pembaruan Database**: Jika validasi berhasil, Controller memanggil Model untuk memperbarui *record* di database menggunakan ID yang sama.
7. **Pesan Status dan Redirect**: Setelah pembaruan berhasil, Controller akan menampilkan pesan sukses dan mengarahkan pengguna kembali ke halaman daftar data atau halaman detail yang diperbarui. Jika validasi gagal, pesan error akan ditampilkan kembali di formulir.

2. Implementasi di CodeIgniter 4

a. **Metode edit() di Controller**

```php
public function edit($id = null)
{
    $model = new MyModel();
    $item = $model->find($id); // Ambil data berdasarkan ID
    if (!$item) { // Handle jika data tidak ditemukan
        return redirect()->to('/items')->with('error', 'Data tidak ditemukan.');
    }
    $data['item'] = $item;
    return view('items/edit', $data); // Tampilkan formulir edit
    }
```

b. **Formulir Edit di View** Formulir ini mirip dengan formulir create, namun nilai *input* sudah terisi dengan data yang ada.

```php
<form action=<?= url_to('items/update', $item['id']) ?> method=post> <?= csrf_field() ?>
    <input type=hidden name=_method value=PUT> <!-- Penting untuk HTTP method spoofing -->
    <label for=nama>Nama:</label>
    <input type=text name=nama id=nama value=<?= esc($item['nama']) ?>>
    <?php if (session()->getFlashdata('errors.nama')) : ?>
        <p class=text-danger><?= session()->getFlashdata('errors.nama') ?></p>
    <?php endif; ?>
    <!-- ... input fields lainnya ... -->
    <button type=submit>Update Data</button>
</form>
```

Perhatikan penggunaan _method dengan nilai PUT. Ini adalah teknik *HTTP method spoofing* yang umum digunakan di framework web untuk mensimulasikan metode HTTP PUT atau DELETE melalui formulir HTML yang hanya mendukung GET dan POST.

c. **Metode update() di Controller**

```php
public function update($id = null)
{
    $model = new MyModel();
    $rules = [
        'nama' => 'required|min_length[3]',
        'email' => 'required|valid_email|is_unique[my_table.email,id,{id}]',
        // Validasi unik kecuali untuk ID ini
    ];
    if (!$this->validate($rules)) {
        return redirect()->back()->withInput()->with('errors', $this->validator->getErrors());
    }
    $data = [
        'nama' => $this->request->getPost('nama'),
        'email' => $this->request->getPost('email'), // ... data lainnya ...
    ];
    $model->update($id, $data); // Perbarui data di database
    return redirect()->to('/items')->with('success', 'Data berhasil diperbarui.');
}
```

Validasi is_unique di CodeIgniter 4 sangat fleksibel, memungkinkan pengecualian untuk *record* yang sedang diedit, mencegah error validasi yang tidak perlu.

D. SOFT DELETE VS HARD DELETE

Dalam manajemen data, penghapusan data adalah operasi yang harus ditangani dengan hati-hati. Ada dua pendekatan utama: *soft delete* dan *hard delete*, masing-masing dengan implikasi yang berbeda terhadap integritas data, pemulihan, dan kepatuhan regulasi.

1. Hard Delete

Hard delete, atau penghapusan permanen, adalah proses di mana *record* data benar-benar dihapus dari database. Setelah *hard delete* dilakukan, data tersebut tidak dapat dipulihkan lagi kecuali dari *backup* database yang mungkin sudah usang.

a. **Kelebihan Hard Delete**:

- **Efisiensi Penyimpanan**: Mengurangi ukuran database karena data yang tidak lagi relevan dihapus sepenuhnya.
- **Kinerja Query**: Query mungkin sedikit lebih cepat karena tidak perlu memfilter data yang dihapus secara logis.
- **Kesederhanaan**: Implementasi lebih sederhana karena hanya melibatkan perintah DELETE SQL.

b. **Kekurangan Hard Delete**:

- **Kehilangan Data Permanen**: Risiko kehilangan data penting secara tidak sengaja sangat tinggi.
- **Integritas Referensial**: Dapat menyebabkan masalah integritas data jika ada *foreign key* yang merujuk ke data yang dihapus, kecuali jika *cascading delete* diatur dengan hati-hati.
- **Kepatuhan Regulasi**: Tidak cocok untuk sistem yang memerlukan audit trail atau kepatuhan terhadap regulasi retensi data (misalnya, GDPR, HIPAA) yang mengharuskan data disimpan untuk periode tertentu (Al-Ruithe et al., 2020).

2. Soft Delete

Soft delete, atau penghapusan logis, adalah pendekatan di mana data tidak benar-benar dihapus dari database, melainkan ditandai sebagai dihapus menggunakan sebuah kolom khusus (misalnya, deleted_at atau is_deleted). Data tersebut masih ada di tabel, tetapi tidak akan muncul dalam query normal.

a. **Kelebihan Soft Delete**:

- **Pemulihan Data**: Data dapat dengan mudah dipulihkan jika terjadi kesalahan penghapusan.
- **Integritas Referensial**: Mempertahankan integritas referensial karena *record* masih ada, mencegah masalah dengan *foreign key*.
- **Audit Trail**: Memungkinkan pelacakan riwayat data, termasuk kapan data dihapus.
- **Kepatuhan Regulasi**: Memenuhi persyaratan retensi data dan audit trail yang seringkali diwajibkan oleh regulasi.

b. **Kekurangan Soft Delete**:

- **Peningkatan Ukuran Database**: Data yang dihapus masih menempati ruang penyimpanan.
- **Kinerja Query**: Query mungkin sedikit lebih lambat karena perlu memfilter data yang ditandai sebagai dihapus.
- **Kompleksitas Query**: Membutuhkan penyesuaian pada setiap query untuk secara eksplisit mengecualikan data yang dihapus secara logis.

3. Implementasi Soft Delete di CodeIgniter 4

CodeIgniter 4 menyediakan fitur *soft delete* bawaan yang sangat mudah diimplementasikan melalui Model.

a. **Konfigurasi Model untuk Soft Delete** Cukup atur properti $useSoftDeletes menjadi true dan $useTimestamps menjadi true di Model Anda. CI4 akan secara otomatis menambahkan kolom deleted_at ke tabel Anda (melalui migrasi) dan mengelolanya.

```
class MyModel extends BaseController
{
    protected $table = 'my_table';
    protected $primaryKey = 'id';
    protected $useSoftDeletes = true; // Aktifkan soft delete
    protected $useTimestamps = true; // Untuk created_at, updated_at, dan deleted_at
    protected $allowedFields = ['nama', 'email'];
}
```

b. **Operasi Delete dengan Soft Delete** Ketika Anda memanggil metode delete($id) pada Model yang telah mengaktifkan *soft delete*, CI4 tidak akan menghapus baris dari database. Sebaliknya, ia akan mengisi kolom deleted_at dengan *timestamp* saat ini.

```
$model = new MyModel(); $model->delete($id); // Ini akan menjadi soft delete
```

c. **Mengambil Data yang Dihapus (Soft Deleted)** Secara *default*, metode find(), findAll(), dll., akan mengecualikan *record* yang telah di-*soft delete*. Untuk mengambil *record* yang di-*soft delete*, gunakan metode withDeleted().

```
$model->withDeleted()->findAll(); // Mengambil semua data, termasuk yang di-soft delete
$model->onlyDeleted()->findAll(); // Hanya mengambil data yang di-soft delete
```

d. **Memulihkan Data (Restore)** Untuk memulihkan *record* yang di-*soft delete*, gunakan metode purgeDeleted() pada Model.

```
$model->withDeleted()->update($id, ['deleted_at' => null]); // Cara manual
// Atau, jika Anda ingin menghapus secara permanen data yang sudah di-soft delete:
// $model->delete($id, true);
// Parameter kedua 'true' untuk hard delete data yang sudah di-soft delete
```

Pilihan antara *soft delete* dan *hard delete* harus didasarkan pada analisis kebutuhan bisnis, kebijakan retensi data, dan persyaratan kepatuhan regulasi (Suryani et al., 2021).

E. PRAKTIK: MODUL CRUD DATA MASTER

Bagian ini akan memandu Anda melalui praktik langsung membangun modul CRUD lengkap untuk data master, misalnya data pegawai. Kita akan mengintegrasikan semua konsep yang telah dibahas sebelumnya: pola CRUD, *pagination*, *searching*, *sorting*, *edit*, *update*, dan *soft delete*.

1. Persiapan Database dan Model

Asumsikan kita memiliki tabel pegawai dengan struktur sebagai berikut:

- id (INT, Primary Key, Auto Increment)
- nama (VARCHAR)
- email (VARCHAR, UNIQUE)
- jabatan (VARCHAR)
- created_at (DATETIME)
- updated_at (DATETIME)
- deleted_at (DATETIME)

a. **Buat Migrasi (jika belum ada)**

php spark make:migration CreatePegawaiTable

Edit file migrasi di app/Database/Migrations/YYYY-MM-DD-HHMMSS_CreatePegawaiTable.php:

```php
<?php
namespace App\Database\Migrations;
use CodeIgniter\Database\Migration;
class CreatePegawaiTable extends Migration
{
    public function up()
    {
        $this->forge->addField([
            'id' => ['type' => 'INT', 'constraint' => 5, 'unsigned' => true, 'auto_increment' => true,],
            'nama' => ['type' => 'VARCHAR', 'constraint' => '100',],
            'email' => ['type' => 'VARCHAR', 'constraint' => '100', 'unique' => true,],
            'jabatan' => ['type' => 'VARCHAR', 'constraint' => '100',],
            'created_at' => ['type' => 'DATETIME', 'null' => true,],
            'updated_at' => ['type' => 'DATETIME', 'null' => true,],
            'deleted_at' => ['type' => 'DATETIME', 'null' => true,],
        ]);
        $this->forge->addKey('id', true);
        $this->forge->createTable('pegawai');
    }
    public function down()
    {
        $this->forge->dropTable('pegawai');
    }
}
```

Jalankan migrasi:

```
php spark migrate
```

b. **Buat Model PegawaiModel** Buat file app/Models/PegawaiModel.php:

```php
<?php
namespace App\Models;
use CodeIgniter\Model;
class PegawaiModel extends Model
{
    protected $table = 'pegawai';
    protected $primaryKey = 'id';
    protected $useAutoIncrement = true;
    protected $returnType = 'array'; // Atau 'App\Entities\Pegawai' jika menggunakan Entity
    protected $useSoftDeletes = true;
    protected $allowedFields = ['nama', 'email', 'jabatan'];
    protected $useTimestamps = true;
    protected $createdField = 'created_at';
    protected $updatedField = 'updated_at';
    protected $deletedField = 'deleted_at';
    protected $validationRules = [
        'nama' => 'required|min_length[3]|max_length[100]',
        'email' => 'required|valid_email|is_unique[pegawai.email,id,{id}]',
        'jabatan' => 'required|min_length[3]|max_length[100]',
    ];
    protected $validationMessages = [
        'email' => [ 'is_unique' => 'Maaf. Email tersebut sudah terdaftar.', ],
    ];
    protected $skipValidation = false;
}
```

2. Buat Controller Pegawai

Buat file app/Controllers/Pegawai.php:

```php
<?php
namespace App\Controllers;
use App\Models\PegawaiModel;
use CodeIgniter\Controller;
class Pegawai extends Controller
{
    protected $pegawaiModel;
    protected $perPage = 5; // Jumlah item per halaman
    public function __construct()
    {
        $this->pegawaiModel = new PegawaiModel();
        helper(['form', 'url']);
    }
    public function index()
    {
        $search = $this->request->getVar('q');
        $sortBy = $this->request->getVar('sort_by') ?? 'id';
        $sortOrder = $this->request->getVar('sort_order') ?? 'asc';
        $query = $this->pegawaiModel->orderBy($sortBy, $sortOrder);
        if ($search) {
            $query->like('nama', $search)->orLike('email', $search)->orLike('jabatan', $search);
        }
        $data = [
            'pegawai' => $query->paginate($this->perPage),
            'pager' => $query->pager,
            'search' => $search,
            'sortBy' => $sortBy,
            'sortOrder' => $sortOrder,
        ];
        return view('pegawai/index', $data);
    }
```

```php
public function create()
{
    return view('pegawai/create');
}
public function store()
{
    if (!$this->validate($this->pegawaiModel->validationRules)) {
        return redirect()->back()->withInput()->with('errors', $this->validator->getErrors());
    }
    $this->pegawaiModel->save([
        'nama' => $this->request->getPost('nama'),
        'email' => $this->request->getPost('email'),
        'jabatan' => $this->request->getPost('jabatan'),
    ]);
    return redirect()->to('/pegawai')->with('success', 'Data pegawai berhasil ditambahkan.');
}
public function edit($id = null)
{
    $pegawai = $this->pegawaiModel->find($id);
    if (!$pegawai) {
        return redirect()->to('/pegawai')->with('error', 'Data pegawai tidak ditemukan.');
    }
    $data['pegawai'] = $pegawai;
    return view('pegawai/edit', $data);
}
```

```php
public function update($id = null)
{ // Set ID untuk validasi is_unique
    $this->pegawaiModel->setValidationRule('email', 'required|valid_email|is_unique[pegawai.email,id,' . 
    if (!$this->validate($this->pegawaiModel->validationRules)) {
        return redirect()->back()->withInput()->with('errors', $this->validator->getErrors());
    }
    $this->pegawaiModel->update($id, [
        'nama' => $this->request->getPost('nama'),
        'email' => $this->request->getPost('email'),
        'jabatan' => $this->request->getPost('jabatan'),
    ]);
    return redirect()->to('/pegawai')->with('success', 'Data pegawai berhasil diperbarui.');
}
public function delete($id = null)
{
    $this->pegawaiModel->delete($id); // Soft delete
    return redirect()->to('/pegawai')->with('success', 'Data pegawai berhasil dihapus (soft delete).');
}
public function restore($id = null)
{ // Untuk restore, kita perlu mengambil data yang di-soft delete
    $pegawai = $this->pegawaiModel->onlyDeleted()->find($id);
    if (!$pegawai) {
        return redirect()->to('/pegawai')->with('error', 'Data pegawai tidak ditemukan atau belum dihapus
    } // Set deleted_at menjadi null untuk mengembalikan data
    $this->pegawaiModel->update($id, ['deleted_at' => null]);
    return redirect()->to('/pegawai')->with('success', 'Data pegawai berhasil dipulihkan.');
}
```

3. Buat View

a. **app/Views/pegawai/index.php**

```php
<!DOCTYPE html>
<html lang=en>
<head>
    <meta charset=UTF-8>
    <meta name=viewport content=width=device-width, initial-scale=1.0>
    <title>Daftar Pegawai</title>
    <style>
        table {
            width: 100%;
            border-collapse: collapse;
        }
```

```
        th,
        td {
            border: 1px solid #ddd;
            padding: 8px;
            text-align: left;
        }
        th {
            background-color: #f2f2f2;
        }
        .pagination li {
            display: inline-block;
            margin-right: 5px;
        }
        .pagination li a,
        .pagination li strong {
            padding: 5px 10px;
            border: 1px solid #ddd;
            text-decoration: none;
        }
        .pagination li.active strong {
            background-color: #007bff;
            color: white;
            border-color: #007bff;
        }
```

```
        .alert-success {
            background-color: #d4edda;
            color: #155724;
            border: 1px solid #c3e6cb;
            padding: 10px;
            margin-bottom: 15px;
        }

        .alert-error {
            background-color: #f8d7da;
            color: #721c24;
            border: 1px solid #f5c6cb;
            padding: 10px;
            margin-bottom: 15px;
        }
    </style>
</head>
```

```
<body>
    <h1>Daftar Pegawai</h1> <?php if (session()->getFlashdata('success')) : ?>
        <div class=alert-success><?= session()->getFlashdata('success') ?></div>
    <?php endif; ?>
    <?php if (session()->getFlashdata('error')) : ?>
        <div class=alert-error><?= session()->getFlashdata('error') ?></div>
    <?php endif; ?>
    <p><a href=<?= url_to('pegawai/create') ?>>Tambah Pegawai Baru</a></p>
    <form action=<?= url_to('pegawai') ?> method=get style=margin-bottom: 15px;>
        <input type=text name=q value=<?= esc($search ?? '') ?>
            placeholder=Cari nama, email, atau jabatan...>
        <button type=submit>Cari</button> <?php if ($search) : ?>
            <a href=<?= url_to('pegawai') ?>>Reset Pencarian</a> <?php endif; ?>
    </form>
```

```
    <table>
        <thead>
            <tr>
                <th><a href=<?= url_to('pegawai', [
                                'q' => $search,
                                'sort_by' => 'id',
                                'sort_order' => ($sortBy == 'id' && $sortOrder == 'asc' ? 'desc' : 'asc')
                            ]) ?>>ID</a>
                </th>
```

```
                <th><a href=<?= url_to('pegawai', [
                                'q' => $search,
                                'sort_by' => 'nama',
                                'sort_order' => ($sortBy == 'nama' && $sortOrder == 'asc' ? 'desc' : 'asc')
                            ]) ?>>Nama</a>
                </th>
                <th><a href=<?= url_to('pegawai', [
                                'q' => $search,
                                'sort_by' => 'email',
                                'sort_order' => ($sortBy == 'email' && $sortOrder == 'asc' ? 'desc' : 'asc')
                            ]) ?>>Email</a>
                </th>
                <th><a href=<?= url_to('pegawai', [
                                'q' => $search,
                                'sort_by' => 'jabatan',
                                'sort_order' => ($sortBy == 'jabatan' && $sortOrder == 'asc' ? 'desc' : 'asc')
                            ]) ?>>Jabatan</a>
                </th>
                <th>Aksi</th>
            </tr>
        </thead>
```

```
    <tbody>
        <?php
        if (!empty($pegawai) && is_array($pegawai)) : ?>
            <?php foreach ($pegawai as $p) : ?>
                <tr>
                    <td><?= esc($p['id']) ?></td>
                    <td><?= esc($p['nama']) ?></td>
                    <td><?= esc($p['email']) ?></td>
                    <td><?= esc($p['jabatan']) ?></td>
                    <td> <a href=<?= url_to('pegawai/edit', $p['id']) ?>>Edit</a>
                <a href=<?= url_to('pegawai/delete', $p['id']) ?>
                onclick=return confirm('Apakah Anda yakin ingin menghapus data ini? (Soft Delete)')>Hapus</a>
                <?php
                if (isset($p['deleted_at']) && $p['deleted_at'] !== null) : ?>
                <a href=<?= url_to('pegawai/restore', $p['id']) ?>
                onclick=return confirm('Apakah Anda yakin ingin memulihkan data ini?')>Pulihkan</a>
                <?php endif; ?>
                    </td>
                </tr>
            <?php endforeach; ?>
        <?php else : ?> <tr>
                <td colspan=5>Tidak ada data pegawai.</td>
            </tr> <?php endif; ?>
    </tbody>
```

```
    </table> <?= $pager->links() ?>
</body>

</html>
```

b. **app/Views/pegawai/create.php**

```
<!DOCTYPE html>
<html lang=en>

<head>
    <meta charset=UTF-8>
    <meta name=viewport content=width=device-width, initial-scale=1.0>
    <title>Tambah Pegawai Baru</title>
    <style>
        .text-danger {
            color: red;
            font-size: 0.9em;
        }
    </style>
</head>
```

```
<body>
    <h1>Tambah Pegawai Baru</h1>
    <form action=<?= url_to('pegawai/store') ?> method=post>
        <?= csrf_field() ?>
        <label for=nama>Nama:</label><br>
        <input type=text name=nama id=nama value=<?= old('nama') ?>><br>
        <?php
        if (session()->getFlashdata('errors.nama')) : ?>
            <p class=text-danger><?= session()->getFlashdata('errors.nama') ?></p>
        <?php endif; ?> <br>
        <label for=email>Email:</label><br>
        <input type=email name=email id=email value=<?= old('email') ?>><br>
        <?php
        if (session()->getFlashdata('errors.email')) : ?>
            <p class=text-danger><?= session()->getFlashdata('errors.email') ?></p>
        <?php endif; ?> <br>
        <label for=jabatan>Jabatan:</label><br>
        <input type=text name=jabatan id=jabatan value=<?= old('jabatan') ?>><br>
        <?php if (session()->getFlashdata('errors.jabatan')) : ?>
            <p class=text-danger><?= session()->getFlashdata('errors.jabatan') ?></p>
        <?php endif; ?> <br>
        <button type=submit>Simpan</button>
        <a href=<?= url_to('pegawai') ?>>Batal</a>
    </form>
</body>
</html>
```

c. **app/Views/pegawai/edit.php**

```
<!DOCTYPE html>
<html lang=en>

<head>
    <meta charset=UTF-8>
    <meta name=viewport content=width=device-width, initial-scale=1.0>
    <title>Edit Pegawai</title>
    <style>
        .text-danger {
            color: red;
            font-size: 0.9em;
        }
    </style>
</head>
```

```
<body>
    <h1>Edit Pegawai: <?= esc($pegawai['nama']) ?></h1>
    <form action=<?= url_to('pegawai/update', $pegawai['id']) ?> method=post>
        <?= csrf_field() ?>
        <input type=hidden name=_method value=PUT>
        <label for=nama>Nama:</label><br>
        <input type=text name=nama id=nama value=<?= old('nama', $pegawai['nama']) ?>><br>
        <?php
        if (session()->getFlashdata('errors.nama')) : ?>
            <p class=text-danger><?= session()->getFlashdata('errors.nama') ?></p>
        <?php endif; ?> <br>
```

```
        <label for=email>Email:</label><br>
        <input type=email name=email id=email value=<?= old('email', $pegawai['email']) ?>><br>
        <?php
        if (session()->getFlashdata('errors.email')) : ?>
            <p class=text-danger><?= session()->getFlashdata('errors.email') ?></p>
        <?php endif; ?> <br>
        <label for=jabatan>Jabatan:</label><br>
        <input type=text name=jabatan id=jabatan value=<?= old('jabatan', $pegawai['jabatan']) ?>><br>
        <?php if (session()->getFlashdata('errors.jabatan')) : ?>
            <p class=text-danger><?= session()->getFlashdata('errors.jabatan') ?></p>
        <?php endif; ?> <br>
```

```
        <button type=submit>Update</button>
        <a href=<?= url_to('pegawai') ?>>Batal</a>
    </form>
</body>
</html>
```

4. Konfigurasi Routing

Tambahkan *routing* di app/Config/Routes.php:

```
$routes->group('pegawai', function($routes){
    $routes->get('/', 'Pegawai::index');
    $routes->get('create', 'Pegawai::create');
    $routes->post('store', 'Pegawai::store');
    $routes->get('edit/(:num)', 'Pegawai::edit/$1');
    $routes->put('update/(:num)', 'Pegawai::update/$1');
    $routes->get('delete/(:num)', 'Pegawai::delete/$1');
    // Menggunakan GET untuk demo, di produksi gunakan POST/DELETE
    $routes->get('restore/(:num)', 'Pegawai::restore/$1');
    // Menggunakan GET untuk demo
    });
```

Dengan langkah-langkah ini, Anda telah berhasil membangun modul CRUD data master lengkap dengan *pagination*, *searching*, *sorting*, *edit*, *update*, dan *soft delete* menggunakan CodeIgniter 4.

F. RANGKUMAN

Pembahasan ini telah menguraikan secara komprehensif implementasi operasi CRUD (*Create, Read, Update, Delete*) di CodeIgniter 4, serta teknik-teknik penting untuk pengelolaan data yang efisien. Arsitektur MVC CodeIgniter 4 menjadi fondasi utama, memastikan pemisahan tanggung jawab yang jelas antara Model, View, dan Controller, sehingga menghasilkan kode yang terstruktur dan mudah dikelola.

1. **Pola CRUD di CodeIgniter 4**:
    - **Model** berperan sentral dalam berinteraksi langsung dengan database, mengelola konfigurasi tabel, *primary key*, *return type*, *soft delete*, *allowed fields*, dan *timestamps*. Metode dasar seperti insert(), save(), find(), findAll(), update(), dan delete() disediakan untuk operasi CRUD.
    - **Controller** berfungsi sebagai penghubung, menerima permintaan pengguna, memanggil Model untuk memproses data, dan memuat View untuk presentasi. Metode umum meliputi index(), create(), store(), edit(), update(), dan delete().
    - **View** bertanggung jawab untuk antarmuka pengguna, menampilkan daftar data, formulir input (untuk *create* dan *edit*), serta pesan status.
2. **Menangani Pagination, Searching, dan Sorting**:
    - **Pagination** membagi data besar menjadi halaman-halaman kecil menggunakan metode paginate() pada Model dan objek $pager di View, dengan konfigurasi $perPage di Model.
    - **Searching** memungkinkan pencarian data spesifik dengan menangkap *input* pencarian di Controller dan menggunakan metode like() atau where() pada Model.
    - **Sorting** mengurutkan data berdasarkan kolom dan urutan tertentu, diimplementasikan dengan menangkap parameter sort_by dan sort_order di Controller, lalu menggunakan orderBy() pada Model.

3. **Edit dan Update Data**:
    - Proses ini melibatkan pengambilan data berdasarkan ID di Controller (edit()), menampilkannya di formulir View yang sudah terisi, dan kemudian mengirimkan perubahan ke Controller (update()).
    - Validasi data sisi server sangat krusial sebelum pembaruan ke database melalui Model. Penggunaan _method dengan nilai PUT penting untuk *HTTP method spoofing* di formulir HTML.
4. **Soft Delete vs Hard Delete**:
    - **Hard Delete** menghapus data secara permanen dari database, menawarkan efisiensi penyimpanan tetapi berisiko kehilangan data dan masalah integritas referensial.
    - **Soft Delete** menandai data sebagai dihapus tanpa menghapusnya secara fisik, mempertahankan integritas data, memungkinkan pemulihan, dan mendukung *audit trail*. CodeIgniter 4 menyediakan fitur *soft delete* bawaan melalui properti $useSoftDeletes dan $useTimestamps di Model, yang mengelola kolom deleted_at. Metode withDeleted() dan onlyDeleted() memungkinkan pengambilan data yang di-*soft delete*, dan pemulihan dapat dilakukan dengan memperbarui deleted_at menjadi null.
5. **Praktik: Modul CRUD Data Master**:
    - Implementasi praktis modul CRUD data pegawai menunjukkan integrasi semua konsep, mulai dari persiapan database dengan migrasi, pembuatan PegawaiModel dengan validasi dan *soft delete*, hingga PegawaiController yang menangani semua operasi CRUD, *pagination*, *searching*, *sorting*, serta View yang interaktif untuk index, create, dan edit. Konfigurasi *routing* yang tepat melengkapi fungsionalitas modul.

BAB 8
AUTENTIKASI DAN OTORISASI PENGGUNA

Dalam pengembangan sistem informasi berbasis web, aspek keamanan merupakan pilar fundamental yang tidak dapat diabaikan. Salah satu fondasi keamanan yang paling krusial adalah autentikasi dan otorisasi pengguna. Tanpa mekanisme yang kuat untuk memverifikasi identitas pengguna (autentikasi) dan menentukan hak akses mereka terhadap sumber daya tertentu (otorisasi), sebuah aplikasi akan rentan terhadap penyalahgunaan data, akses tidak sah, dan berbagai ancaman keamanan lainnya. Bab ini akan membawa kita menyelami lebih dalam bagaimana CodeIgniter 4 menyediakan fitur dan alat yang memadai untuk membangun sistem autentikasi dan otorisasi yang robust dan aman.

Urgensi pembahasan ini terletak pada fakta bahwa hampir setiap sistem informasi, mulai dari aplikasi e-commerce, sistem manajemen konten, hingga aplikasi internal perusahaan, memerlukan cara untuk membedakan siapa yang dapat mengakses apa. Bayangkan sebuah sistem informasi akademik tanpa login; siapa pun bisa mengubah nilai mahasiswa atau data dosen. Atau sebuah sistem inventori tanpa otorisasi; setiap karyawan bisa menghapus data stok penting. Oleh karena itu, pemahaman mendalam tentang bagaimana mengimplementasikan sistem login, logout, serta mengelola peran dan hak akses pengguna menjadi sangat vital bagi seorang pengembang aplikasi web.

Kita akan memulai dengan memahami dasar-dasar sistem login dan logout, yang merupakan gerbang utama bagi pengguna untuk berinteraksi dengan aplikasi. Selanjutnya, kita akan mengeksplorasi peran penting Middleware atau Filters dalam CodeIgniter 4, yang berfungsi sebagai penjaga gerbang untuk rute-rute tertentu, memastikan hanya pengguna yang berhak yang dapat mengaksesnya. Konsep Role-Based Access Control (RBAC) akan menjadi fokus utama dalam otorisasi, memungkinkan kita untuk mendefinisikan peran seperti 'admin', 'user', atau 'guest' dengan hak akses yang berbeda-beda. Keamanan password juga akan dibahas secara mendalam, dengan penekanan pada penggunaan Password Helper CodeIgniter 4 untuk melakukan hashing yang aman, melindungi kredensial pengguna dari serangan brute-force atau kebocoran data. Pada akhirnya, semua konsep ini akan diintegrasikan dalam praktik membangun modul login dan manajemen pengguna yang fungsional, memberikan pengalaman langsung dalam mengamankan aplikasi web Anda.

A. SISTEM LOGIN & LOGOUT

Sistem login dan logout merupakan fondasi utama dalam mekanisme autentikasi pengguna pada aplikasi web. Autentikasi adalah proses verifikasi identitas pengguna, memastikan bahwa pengguna yang mencoba mengakses sistem adalah benar-benar orang yang mereka klaim. Tanpa proses ini, integritas dan keamanan data dalam sistem informasi akan sangat terancam.

1. Mekanisme Login

Proses login dimulai ketika pengguna memasukkan kredensial mereka, biasanya berupa *username* atau *email* dan *password*, ke dalam sebuah formulir. Data ini kemudian dikirimkan ke server untuk divalidasi. Di sisi server, aplikasi akan melakukan beberapa langkah penting:

a. **Penerimaan Kredensial**: Aplikasi menerima data *username* dan *password* yang dikirimkan oleh pengguna melalui metode POST. Penting untuk selalu menggunakan metode POST untuk pengiriman data sensitif guna menghindari tereksposnya kredensial di URL.

b. **Validasi Input**: Sebelum memproses lebih lanjut, input dari pengguna harus divalidasi. Ini mencakup pemeriksaan apakah *field* tidak kosong, format *email* sudah benar (jika menggunakan *email* sebagai *username*), dan panjang *password* memenuhi kriteria keamanan. CodeIgniter 4 menyediakan fitur validasi yang kuat untuk tujuan ini.
c. **Verifikasi Kredensial**: Setelah input divalidasi, aplikasi akan mencari data pengguna di database berdasarkan *username* atau *email* yang diberikan. Jika pengguna ditemukan, *password* yang dimasukkan oleh pengguna akan dibandingkan dengan *password* yang tersimpan di database. Namun, *password* di database tidak boleh disimpan dalam bentuk *plaintext*. Sebaliknya, *password* harus disimpan dalam bentuk *hash*. Proses verifikasi melibatkan *hashing password* yang dimasukkan pengguna dan membandingkan *hash* tersebut dengan *hash* yang tersimpan. Jika kedua *hash* cocok, autentikasi berhasil.
d. **Manajemen Sesi**: Jika autentikasi berhasil, aplikasi akan membuat sesi untuk pengguna tersebut. Sesi adalah cara server untuk mengingat pengguna yang telah login di antara beberapa *request* HTTP. CodeIgniter 4 menyediakan pustaka Session yang memudahkan pengelolaan sesi. Informasi penting seperti ID pengguna, *username*, dan peran pengguna biasanya disimpan dalam sesi. Sesi ini kemudian dikaitkan dengan *cookie* yang dikirimkan ke *browser* pengguna.
e. **Pengalihan Halaman**: Setelah sesi dibuat, pengguna akan dialihkan ke halaman yang berhak mereka akses, seperti *dashboard* atau halaman utama aplikasi.

2. Mekanisme Logout

Proses logout adalah kebalikan dari login, di mana pengguna mengakhiri sesi mereka dengan aplikasi. Ini adalah langkah penting untuk keamanan, terutama pada perangkat publik, karena mencegah akses tidak sah ke akun pengguna setelah mereka meninggalkan perangkat.
a. **Penghancuran Sesi**: Ketika pengguna mengklik tombol logout, aplikasi akan menghancurkan sesi yang terkait dengan pengguna tersebut. Ini berarti semua data sesi yang disimpan di server akan dihapus, dan *cookie* sesi di *browser* pengguna akan diinvalidasi.
b. **Pengalihan Halaman**: Setelah sesi dihancurkan, pengguna akan dialihkan kembali ke halaman login atau halaman publik lainnya.

B. MIDDLEWARE (FILTERS) UNTUK AKSES HALAMAN

Middleware, atau dalam konteks CodeIgniter 4 disebut sebagai Filters, adalah mekanisme yang sangat powerful untuk mengontrol alur *request* HTTP sebelum mencapai *controller* atau setelah *controller* selesai memproses *request* tetapi sebelum *response* dikirimkan ke *browser*. Filters sangat ideal untuk tugas-tugas seperti autentikasi, otorisasi, *logging*, *rate limiting*, dan manipulasi *response*.

1. Konsep Filters di CodeIgniter 4

Filters di CodeIgniter 4 adalah kelas PHP yang mengimplementasikan antarmuka CodeIgniter\Filters\FilterInterface. Antarmuka ini memiliki dua metode utama:
a. before(RequestInterface $request, $arguments = null): Metode ini dieksekusi sebelum *controller* dipanggil. Ini adalah tempat yang tepat untuk memeriksa apakah pengguna terautentikasi atau memiliki hak akses yang diperlukan. Jika kondisi tidak terpenuhi, *request* dapat dihentikan dan pengguna dialihkan ke halaman lain (misalnya, halaman login).
b. after(RequestInterface $request, ResponseInterface $response, $arguments = null): Metode ini dieksekusi setelah *controller* selesai memproses *request* dan sebelum *response* dikirimkan ke *browser*. Ini dapat digunakan untuk memodifikasi *response*, melakukan *logging*, atau membersihkan sumber daya.

2. Implementasi Filters untuk Autentikasi dan Otorisasi

Untuk melindungi halaman yang memerlukan autentikasi, kita dapat membuat sebuah Filter AuthFilter. Filter ini akan memeriksa apakah ada sesi pengguna yang aktif. Jika tidak ada, pengguna akan dialihkan ke halaman login.

```
// app/Filters/AuthFilter.php
namespace App\Filters;
use CodeIgniter\Filters\FilterInterface;
use CodeIgniter\HTTP\RequestInterface;
use CodeIgniter\HTTP\ResponseInterface;
class AuthFilter implements FilterInterface
{
    public function before(RequestInterface $request, $arguments = null) {
        if (!session()->get('logged_in')) {
            return redirect()->to('/login');
        }
    }
    public function after(RequestInterface $request, ResponseInterface $response, $arguments = null) {
        // Do nothing
        }
    }
```

Setelah Filter dibuat, kita perlu mendaftarkannya di app/Config/Filters.php dan menerapkannya pada rute atau grup rute tertentu.

```
// app/Config/Filters.php
public $aliases = [
    'csrf' => \CodeIgniter\Filters\CSRF::class,
    'toolbar' => \CodeIgniter\Filters\DebugToolbar::class,
    'honeypot' => \CodeIgniter\Filters\Honeypot::class,
    'auth' => \App\Filters\AuthFilter::class,
    // Daftarkan filter kita
    ];
    public $globals = [
        'before' => [ // 'honeypot', // 'csrf',
            ],
        'after' => [ 'toolbar', // 'honeypot',
        ],
    ];
    public $methods = [];
    public $filters = [
        'auth' => ['before' => ['dashboard/*', 'users/*']],
        // Terapkan filter 'auth' pada rute yang dilindungi
        ];
```

Dengan konfigurasi di atas, setiap *request* ke URL yang diawali dengan /dashboard/ atau /users/ akan melewati AuthFilter terlebih dahulu. Jika pengguna belum login, mereka akan dialihkan ke /login.

C. ROLE-BASED ACCESS CONTROL (RBAC)

Role-Based Access Control (RBAC) adalah metode otorisasi yang populer di mana hak akses pengguna ditentukan berdasarkan peran yang mereka miliki dalam sistem. Daripada memberikan izin secara langsung kepada setiap pengguna, izin dikaitkan dengan peran, dan pengguna kemudian diberi satu atau lebih peran. Ini menyederhanakan manajemen izin, terutama dalam sistem dengan banyak pengguna dan kompleksitas izin yang tinggi (Ferraiolo et al., 2001).

1. Konsep Dasar RBAC

a. **Peran (Roles)**: Peran adalah koleksi izin yang logis. Contoh peran bisa berupa 'Admin', 'Editor', 'User', atau 'Guest'. Setiap peran memiliki serangkaian tugas atau operasi yang diizinkan.

b. **Izin (Permissions)**: Izin adalah hak untuk melakukan tindakan tertentu pada sumber daya tertentu. Contoh izin: 'membuat_artikel', 'mengedit_artikel', 'menghapus_artikel', 'melihat_laporan_penjualan'.

c. **Pengguna (Users)**: Setiap pengguna dalam sistem diberi satu atau lebih peran.

2. Implementasi RBAC di CodeIgniter 4

Implementasi RBAC biasanya melibatkan penambahan kolom role pada tabel users di database. Kolom ini akan menyimpan peran pengguna (misalnya, 'admin', 'user').

```
-- Contoh struktur tabel users
CREATE TABLE users (
id INT AUTO_INCREMENT PRIMARY KEY,
username VARCHAR(255) NOT NULL UNIQUE,
email VARCHAR(255) NOT NULL UNIQUE,
password VARCHAR(255) NOT NULL,
role ENUM('admin', 'user', 'guest') DEFAULT 'user',
created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP );
```

Untuk menerapkan otorisasi, kita dapat memodifikasi AuthFilter atau membuat Filter baru, misalnya RoleFilter, yang memeriksa peran pengguna yang sedang login.

```
<?php
// app/Filters/RoleFilter.php
namespace App\Filters;

use CodeIgniter\Filters\FilterInterface;
use CodeIgniter\HTTP\RequestInterface;
use CodeIgniter\HTTP\ResponseInterface;

class RoleFilter implements FilterInterface
{
    public function before(RequestInterface $request, $arguments = null)
    {
        $userRole = session()->get('role');
        $requiredRoles = $arguments; // Peran yang dibutuhkan dari konfigurasi filter
        if (!session()->get('logged_in')) {
            return redirect()->to('/login');
        }
        if (!in_array($userRole, $requiredRoles)) {
            // Pengguna tidak memiliki peran yang dibutuhkan
            return redirect()->to('/unauthorized')->with('error', 'Anda tidak memiliki hak akses.');
        }
    }
    public function after(RequestInterface $request, ResponseInterface $response, $arguments = null)
    {
        // Do nothing
    }
}
```

Kemudian, daftarkan RoleFilter dan terapkan pada rute dengan argumen peran yang dibutuhkan:

```
// app/Config/Filters.php
public $aliases = [ // ...
    'role' => \App\Filters\RoleFilter::class,];
public $filters = [ // ...
    'role' => [ 'before' => [ 'admin/*' => ['admin'],     // Hanya admin yang bisa akses rute admin
    'editor/*' => ['admin', 'editor'], // Admin dan editor bisa akses rute editor
    ] ], ];
```

Dengan pendekatan ini, kita dapat dengan mudah mengelola hak akses berdasarkan peran pengguna.

D. HASHING PASSWORD DENGAN PASSWORD HELPER

Keamanan *password* adalah aspek krusial dalam autentikasi. Menyimpan *password* dalam bentuk *plaintext* di database adalah praktik yang sangat berbahaya dan harus dihindari. Jika database diretas, semua *password* pengguna akan terekspos. Solusinya adalah menggunakan teknik *hashing password*.

1. Mengapa Hashing Password Penting?

Hashing adalah proses mengubah *string* input (dalam hal ini, *password*) menjadi *string* dengan panjang tetap yang disebut *hash* atau *digest*. Fungsi *hash* yang baik memiliki beberapa karakteristik penting:

a. **Satu Arah**: Tidak mungkin (atau sangat sulit) untuk merekonstruksi *password* asli dari *hash*-nya.

b. **Deterministik**: Input yang sama selalu menghasilkan *hash* yang sama.

c. **Resistensi Kolisi**: Sangat sulit menemukan dua input berbeda yang menghasilkan *hash* yang sama.

d. **Resistensi Brute-Force**: Sulit untuk menebak *password* dengan mencoba semua kemungkinan *hash*.

CodeIgniter 4 menyediakan Password Helper yang memanfaatkan fungsi password_hash() dan password_verify() bawaan PHP, yang merupakan standar industri untuk *hashing password* yang aman. Fungsi ini secara otomatis menangani *salting* (menambahkan *string* acak ke *password* sebelum *hashing*) dan pemilihan algoritma *hashing* yang kuat (seperti Bcrypt).

2. Penggunaan Password Helper

a. **Saat Pendaftaran/Perubahan Password**: Ketika pengguna mendaftar atau mengubah *password*, *password* yang mereka masukkan harus di-*hash* sebelum disimpan ke database.

```php
// Contoh di Controller saat menyimpan password baru
helper('form'); // Memuat form helper jika belum
helper('url'); // Memuat url helper jika belum
helper('password'); // Memuat password helper
$password = $this->request->getPost('password');
$hashedPassword = password_hash($password, PASSWORD_BCRYPT); // Menggunakan algoritma BCRYPT
// Simpan $hashedPassword ke database
$userModel->save([
    'username' => $this->request->getPost('username'),
    'email' => $this->request->getPost('email'),
    'password' => $hashedPassword,
    'role' => 'user'
]);
```

b. **Saat Login (Verifikasi Password)**: Ketika pengguna mencoba login, *password* yang mereka masukkan akan dibandingkan dengan *hash* yang tersimpan di database menggunakan password_verify().

```php
// Contoh di Controller saat login
helper('password'); // Memuat password helper
$username = $this->request->getPost('username');
$password = $this->request->getPost('password');
$user = $userModel->where('username', $username)->first();
if ($user && password_verify($password, $user['password'])) {
    // Password cocok, autentikasi berhasil
    $session = session();
    $session->set([
        'user_id' => $user['id'],
        'username' => $user['username'],
        'role' => $user['role'], 'logged_in' => TRUE ]);
        return redirect()->to('/dashboard');
    } else {
        // Password tidak cocok atau username tidak ditemukan
        return redirect()->back()->withInput()->with('error', 'Username atau password salah.');
    }
```

Penggunaan password_hash() dan password_verify() sangat direkomendasikan karena secara otomatis mengelola kompleksitas *hashing* yang aman, termasuk *salting* dan pemilihan algoritma yang tepat, sehingga pengembang tidak perlu khawatir tentang detail implementasi keamanan *password* yang rentan terhadap kesalahan.

E. PRAKTIK: MODUL LOGIN & MANAJEMEN USER

Bagian ini akan memandu Anda melalui langkah-langkah praktis untuk membangun modul login dan manajemen pengguna yang lengkap di CodeIgniter 4, mengintegrasikan semua konsep yang telah dibahas sebelumnya.

1. Persiapan Database

Pastikan Anda memiliki tabel users dengan struktur seperti yang dijelaskan di bagian RBAC, termasuk kolom id, username, email, password, dan role.

2. Membuat Model User

Buat UserModel untuk berinteraksi dengan tabel users.

```php
<?php
// app/Models/UserModel.php
namespace App\Models;

use CodeIgniter\Model;

class UserModel extends Model
{
    protected $table = 'users';
    protected $primaryKey = 'id';
    protected $useAutoIncrement = true;
    protected $returnType = 'array';
    protected $useSoftDeletes = false;
    protected $allowedFields = ['username', 'email', 'password', 'role'];
    protected $useTimestamps = true;
    protected $createdField = 'created_at';
    protected $updatedField = 'updated_at';
    protected $deletedField = 'deleted_at';
    protected $validationRules = [];
    protected $validationMessages = [];
    protected $skipValidation = false;
}
```

3. Membuat Controller Autentikasi (Login, Register, Logout)

Buat *AuthController* untuk menangani proses login, pendaftaran, dan logout.

```php
<?php
// app/Controllers/AuthController.php
namespace App\Controllers;

use App\Models\UserModel;
use CodeIgniter\Controller;

class AuthController extends Controller
{
    public function login()
    {
        helper(['form', 'url']);
        return view('auth/login');
    }
    public function authenticate()
    {
        helper(['form', 'password']);
        $rules = [
            'username' => 'required|min_length[3]|max_length[50]',
            'password' => 'required|min_length[8]|max_length[255]|validateUser[username,password]',
        ];
        $errors = [
            'password' => ['validateUser' => 'Username atau Password tidak cocok.']
        ];
        if (!$this->validate($rules, $errors)) {
            return redirect()->back()->withInput()->with('errors', $this->validator->getErrors());
        }
```

```php
        $userModel = new UserModel();
        $user = $userModel->where('username', $this->request->getPost('username'))->first();
        $session = session();
        $session->set([
            'user_id' => $user['id'],
            'username' => $user['username'],
            'role' => $user['role'],
            'logged_in' => TRUE
        ]);
        return redirect()->to('/dashboard');
    }
    public function register()
    {
        helper(['form', 'url']);
        return view('auth/register');
    }
```

```php
    public function storeUser()
    {
        helper(['form', 'password']);
        $rules = [
            'username' => 'required|min_length[3]|max_length[50]|is_unique[users.username]',
            'email' => 'required|min_length[6]|max_length[255]|valid_email|is_unique[users.email]',
            'password' => 'required|min_length[8]|max_length[255]',
            'conf_password' => 'matches[password]' ];
        if (!$this->validate($rules)) {
            return redirect()->back()->withInput()->with('errors', $this->validator->getErrors()); }
        $userModel = new UserModel();
        $data = [
            'username' => $this->request->getPost('username'),
            'email' => $this->request->getPost('email'),
            'password' => password_hash($this->request->getPost('password'), PASSWORD_BCRYPT),
            'role' => 'user' ]; // Default role
        $userModel->save($data);
        return redirect()->to('/login')->with('success', 'Pendaftaran berhasil! Silakan login.');
    }
```

```
    public function logout()
    {
        $session = session();
        $session->destroy();
        return redirect()->to('/login');
    }
}
```

Catatan: validateUser adalah *custom validation rule* yang perlu Anda buat di app/Config/Validation.php untuk memverifikasi *username* dan *password* secara bersamaan.

4. Membuat View Login dan Register

Buat *view* app/Views/auth/login.php dan app/Views/auth/register.php dengan formulir yang sesuai.

5. Konfigurasi Rute

Tambahkan rute di app/Config/Routes.php.

```
// app/Config/Routes.php
$routes->get('login', 'AuthController::login');
$routes->post('authenticate', 'AuthController::authenticate');
$routes->get('register', 'AuthController::register');
$routes->post('storeUser', 'AuthController::storeUser');
$routes->get('logout', 'AuthController::logout'); // Rute yang dilindungi oleh autentikasi
$routes->group('/',
    ['filter' => 'auth'],
    function ($routes) {
        $routes->get('dashboard', 'DashboardController::index');
        // Tambahkan rute lain yang memerlukan login di sini
    });
// Rute yang dilindungi oleh peran (contoh: hanya admin)
$routes->group('admin',
    ['filter' => 'role:admin'],
    function ($routes) {
        $routes->get('users', 'AdminController::manageUsers');
        // Tambahkan rute khusus admin lainnya
    });
```

6. Membuat Controller Manajemen User (Admin)

Buat AdminController untuk mengelola pengguna (CRUD).

```
<?php
// app/Controllers/AdminController.php
namespace App\Controllers;

use App\Models\UserModel;
use CodeIgniter\Controller;

class AdminController extends Controller
{
    public function manageUsers()
    {
        $userModel = new UserModel();
        $data['users'] = $userModel->findAll();
        return view('admin/manage_users', $data);
    }
    // Tambahkan metode untuk create, edit, delete user
    // Pastikan untuk melakukan hashing password saat membuat/mengedit user
    // dan validasi input yang ketat.
}
```

Dengan mengikuti langkah-langkah ini, Anda akan memiliki modul login dan manajemen pengguna yang fungsional, aman, dan terstruktur dengan baik menggunakan CodeIgniter 4.

F. RANGKUMAN

Pembahasan ini telah menguraikan secara komprehensif fondasi dan implementasi sistem autentikasi dan otorisasi dalam pengembangan aplikasi web menggunakan CodeIgniter 4. Dari mekanisme dasar login hingga kontrol akses berbasis peran yang canggih, setiap aspek telah dijelaskan dengan detail, menekankan pentingnya keamanan dan praktik terbaik.
Berikut adalah poin-poin rangkuman pembahasan:

1. **Sistem Login & Logout**: Merupakan inti dari autentikasi pengguna. Proses login melibatkan penerimaan kredensial, validasi input, verifikasi *password* yang di-*hash* di database, manajemen sesi, dan pengalihan halaman. Mekanisme logout berfungsi untuk mengakhiri sesi pengguna dengan menghancurkan data sesi dan menginvalidasi *cookie*.
2. **Mekanisme Login**: Dimulai dengan input kredensial pengguna, diikuti validasi sisi server, verifikasi *password* yang di-*hash* menggunakan password_verify(), pembuatan sesi untuk menyimpan informasi pengguna, dan pengalihan ke halaman yang relevan.
3. **Mekanisme Logout**: Proses ini secara fundamental menghancurkan sesi pengguna yang aktif, menghapus data sesi dari server, dan mengarahkan pengguna kembali ke halaman login atau publik.
4. **Middleware (Filters) untuk Akses Halaman**: Filters di CodeIgniter 4 adalah alat yang efektif untuk mengontrol alur *request* HTTP. Metode before() digunakan untuk autentikasi dan otorisasi sebelum *controller* dipanggil, sementara after() dapat memodifikasi *response*.
5. **Implementasi Filters untuk Autentikasi dan Otorisasi**: Filter AuthFilter dapat dibuat untuk memeriksa status login pengguna, mengalihkan ke halaman login jika belum terautentikasi. Filters didaftarkan di app/Config/Filters.php dan diterapkan pada rute atau grup rute tertentu.
6. **Role-Based Access Control (RBAC)**: Metode otorisasi yang mengaitkan hak akses dengan peran pengguna. Ini menyederhanakan manajemen izin dengan mendefinisikan peran (misalnya, 'Admin', 'User') yang memiliki serangkaian izin tertentu.
7. **Implementasi RBAC di CodeIgniter 4**: Melibatkan penambahan kolom role pada tabel users dan penggunaan RoleFilter untuk memeriksa peran pengguna yang sedang login terhadap peran yang dibutuhkan untuk mengakses rute tertentu.
8. **Hashing Password dengan Password Helper**: Keamanan *password* sangat krusial. *Password* harus selalu di-*hash* sebelum disimpan di database menggunakan fungsi password_hash() dan diverifikasi dengan password_verify(). CodeIgniter 4 menyediakan Password Helper yang memanfaatkan fungsi PHP ini untuk *hashing* yang aman, termasuk *salting* otomatis.
9. **Praktik: Modul Login & Manajemen User**: Panduan praktis untuk membangun modul login, pendaftaran, dan manajemen pengguna lengkap, mencakup persiapan database, pembuatan UserModel, AuthController untuk login/register/logout, *view* terkait, konfigurasi rute, dan AdminController untuk manajemen pengguna.

BAB 9
UPLOAD FILE, MANAJEMEN MEDIA & HANDLING ERRORS

Dalam pengembangan sistem informasi berbasis web, kemampuan untuk mengelola file, baik itu gambar, dokumen, atau jenis media lainnya, merupakan fitur esensial yang hampir selalu dibutuhkan. Bayangkan sebuah sistem informasi kepegawaian tanpa kemampuan mengunggah foto profil karyawan, atau sistem e-commerce yang tidak bisa menampilkan gambar produk. Fitur upload file inilah yang memungkinkan aplikasi menjadi lebih interaktif, kaya data visual, dan fungsional. Bab 9 ini akan membawa kita menyelami seluk-beluk implementasi fitur krusial tersebut dalam konteks CodeIgniter 4.

Namun, proses upload file tidak semata-mata tentang memindahkan file dari klien ke server. Ada serangkaian tantangan dan pertimbangan penting yang harus diatasi. Keamanan menjadi prioritas utama; bagaimana kita memastikan file yang diunggah aman dari serangan berbahaya? Bagaimana kita mencegah pengguna mengunggah file yang terlalu besar atau dengan format yang tidak diizinkan? Pertanyaan-pertanyaan ini mengarah pada pentingnya validasi file yang ketat, sebuah topik yang akan kita bahas secara mendalam. Kita akan mempelajari cara membatasi ukuran file, memfilter jenis ekstensi, dan memastikan integritas file yang diunggah.

Selain itu, pengelolaan file yang diunggah juga memerlukan strategi penyimpanan yang terorganisir. Menyimpan semua file dalam satu direktori besar dapat menyebabkan kekacauan dan kesulitan dalam manajemen di kemudian hari. Oleh karena itu, kita akan mengeksplorasi praktik terbaik dalam menyimpan file ke folder khusus, memastikan struktur direktori yang rapi dan mudah dikelola.

Tidak kalah pentingnya adalah penanganan kesalahan (error handling) dan pencatatan log (logging). Dalam setiap proses yang melibatkan interaksi dengan pengguna dan sistem file, potensi terjadinya kesalahan selalu ada. Mulai dari kegagalan upload karena koneksi internet, file korup, hingga masalah izin akses server. Modul ini akan membekali Anda dengan pengetahuan dan keterampilan untuk mengidentifikasi, menangani, dan mencatat kesalahan-kesalahan ini secara efektif, sehingga aplikasi Anda tetap stabil dan memberikan pengalaman pengguna yang baik. Dengan pemahaman yang kuat tentang penanganan error, pengembang dapat dengan cepat mendiagnosis masalah dan menjaga aplikasi tetap berjalan optimal.

Pada akhirnya, bab ini akan diakhiri dengan praktik langsung, yaitu membangun modul upload foto profil atau dokumen. Melalui studi kasus ini, Anda akan mengintegrasikan semua konsep yang telah dipelajari, mulai dari proses upload, validasi, penyimpanan, hingga penanganan kesalahan, menjadi sebuah fitur yang fungsional dan siap pakai dalam sistem informasi berbasis web Anda.

A. UPLOAD GAMBAR/DOKUMEN

Proses upload file, baik itu gambar, dokumen, atau jenis media lainnya, merupakan salah satu fitur fundamental dalam aplikasi web modern. Fitur ini memungkinkan pengguna untuk berinteraksi lebih dinamis dengan sistem, misalnya mengunggah foto profil, melampirkan dokumen pendukung, atau menampilkan gambar produk. Secara teknis, proses upload melibatkan transfer data dari perangkat klien (browser) ke server aplikasi. CodeIgniter 4 menyediakan mekanisme yang robust dan mudah digunakan untuk menangani operasi ini.

1. Mekanisme Dasar Upload File HTML

Sebelum masuk ke implementasi CodeIgniter 4, penting untuk memahami bagaimana form HTML menangani upload file. Elemen kunci adalah tag <input type=file> dan atribut enctype=multipart/form-data pada tag <form>.

```
<form action=/upload/do_upload method=post enctype=multipart/form-data>
    <label for=userfile>Pilih File:</label>
    <input type=file name=userfile id=userfile> <br>
    <input type=submit value=Upload File>
</form>
```

Atribut enctype=multipart/form-data sangat krusial karena memberi tahu browser bahwa form akan mengirimkan data biner (file) selain data teks biasa. Tanpa atribut ini, server tidak akan dapat menerima file yang diunggah (Welling & Thomson, 2019). Ketika form disubmit, file yang dipilih akan dikirim sebagai bagian dari request HTTP POST ke URL yang ditentukan dalam atribut action.

2. Mengelola Upload File di CodeIgniter 4

CodeIgniter 4 menyederhanakan proses penanganan file yang diunggah melalui objek UploadedFile. Objek ini menyediakan berbagai metode untuk mengakses informasi file, memindahkannya, dan melakukan validasi.

a. Mengakses File yang Diunggah

Setelah form disubmit, file yang diunggah dapat diakses melalui objek Request di dalam Controller. Metode getFile() digunakan untuk mendapatkan objek UploadedFile berdasarkan nama input file.

```
// app/Controllers/Upload.php
<?php
namespace App\Controllers;
use CodeIgniter\Controller;
class Upload extends Controller
{
    public function do_upload()
    {
        $file = $this->request->getFile('userfile'); // 'userfile' adalah nama input dari form
        if ($file->isValid() && ! $file->hasMoved()) {
            // File valid dan belum dipindahkan
            // Lanjutkan ke validasi dan penyimpanan
        } else {
            // File tidak valid atau sudah dipindahkan
            // Tangani error
        }
    }
```

Metode isValid() memeriksa apakah file berhasil diunggah tanpa error PHP, dan hasMoved() memastikan file belum dipindahkan ke lokasi permanen.

B. VALIDASI FILE: UKURAN, FORMAT, KEAMANAN

Validasi file adalah langkah krusial dalam proses upload untuk memastikan bahwa file yang diunggah memenuhi kriteria yang ditetapkan, aman, dan tidak akan menyebabkan masalah pada sistem. Mengabaikan validasi dapat membuka celah keamanan serius, seperti serangan injeksi kode atau pengisian ruang penyimpanan server secara berlebihan (Snyder & Seidl, 2020).

1. Validasi Ukuran File

Membatasi ukuran file yang diunggah sangat penting untuk mencegah pengguna membanjiri server dengan file-file besar yang dapat menghabiskan ruang penyimpanan dan bandwidth. CodeIgniter 4 menyediakan aturan validasi bawaan untuk ini.

```
$validationRules = [
    'userfile' => [
        'rules' => 'uploaded[userfile]|max_size[userfile,1024]',
        // Maksimal 1MB (1024 KB)
        'errors' => ['max_size' => 'Ukuran file {field} terlalu besar. Maksimal 1MB.']
    ]
];
if (! $this->validate($validationRules)) {
    // Tampilkan error validasi
    return redirect()->back()->withInput()->with('errors', $this->validator->getErrors());
}
```

Aturan max_size[field,ukuran_dalam_kb] digunakan untuk membatasi ukuran file. Selain itu, konfigurasi PHP upload_max_filesize dan post_max_size juga harus disesuaikan di php.ini untuk mendukung ukuran file yang diinginkan.

2. Validasi Format File (Tipe MIME dan Ekstensi)

Memastikan hanya format file yang diizinkan yang dapat diunggah adalah langkah keamanan penting. Misalnya, hanya mengizinkan gambar (JPG, PNG) untuk foto profil, atau PDF untuk dokumen.

```
$validationRules = [
    'userfile' => [
        'rules' => 'uploaded[userfile]|max_size[userfile,1024]|
                    ext_in[userfile,jpg,jpeg,png]|mime_in[userfile,image/jpg,image/jpeg,image/png]',
        'errors' => ['ext_in' => 'Format file {field} tidak diizinkan. Hanya JPG, JPEG, PNG.',
        'mime_in' => 'Tipe MIME file {field} tidak valid.']
    ]
];
```

- **ext_in[field,ext1,ext2,...]**: Memvalidasi ekstensi file. Ini adalah validasi sisi klien yang mudah dipalsukan, namun tetap berguna sebagai lapisan pertama.
- **mime_in[field,mime1,mime2,...]**: Memvalidasi tipe MIME file. Ini adalah validasi sisi server yang lebih andal karena memeriksa tipe konten aktual file, bukan hanya ekstensinya. Tipe MIME dapat diperoleh dari objek UploadedFile menggunakan $file->getMimeType().

3. Validasi Keamanan Tambahan

Selain ukuran dan format, ada beberapa aspek keamanan lain yang perlu dipertimbangkan:

- **Pemeriksaan Nama File**: Mencegah nama file yang mengandung karakter berbahaya atau mencoba melakukan path traversal (misalnya ../../malicious.php). CodeIgniter 4 secara otomatis membersihkan nama file saat memindahkannya, tetapi pemeriksaan tambahan bisa dilakukan.
- **Pemeriksaan Konten File**: Untuk file gambar, disarankan untuk memproses ulang gambar (misalnya mengubah ukuran atau mengompres) setelah diunggah. Ini dapat membantu menghilangkan metadata berbahaya atau skrip tersembunyi yang mungkin disuntikkan ke dalam file gambar (OWASP, 2023).
- **Antivirus Scan**: Untuk aplikasi dengan tingkat keamanan sangat tinggi, integrasi dengan solusi antivirus di server dapat menjadi lapisan pertahanan tambahan.

C. MENYIMPAN FILE KE FOLDER KHUSUS

Setelah file berhasil divalidasi, langkah selanjutnya adalah memindahkannya dari direktori sementara PHP ke lokasi penyimpanan permanen di server. Organisasi folder yang baik sangat penting untuk manajemen media yang efisien.

1. Struktur Folder Penyimpanan

Disarankan untuk menyimpan file yang diunggah di luar direktori public aplikasi, atau setidaknya di dalam public tetapi di folder terpisah yang tidak dapat diakses langsung oleh URL tanpa otorisasi jika file tersebut bersifat sensitif. CodeIgniter 4 secara default memiliki folder writable/uploads yang cocok untuk tujuan ini.

Contoh struktur:

project-root/ ├── app/ ├── public/ | └── assets/ | └── img/ | └── profile/ (jika ingin diakses publik langsung) ├── writable/ | └── uploads/ | ├── profiles/ | ├── documents/ | └── temp/

2. Memindahkan File

Objek UploadedFile memiliki metode move() yang digunakan untuk memindahkan file.

```
// Dalam Controller setelah validasi berhasil
$file = $this->request->getFile('userfile');
if ($file->isValid() && ! $file->hasMoved()) {
    $newName = $file->getRandomName();
    // Memberikan nama unik untuk menghindari konflik
    $file->move(WRITEPATH . 'uploads/profiles', $newName);
    // Pindahkan ke writable/uploads/profiles
    // Simpan $newName ke database
    $data = [
        'profile_picture' => $newName, // ... data lainnya
    ]; //
    $userModel->update($userId, $data);
    return redirect()->back()->with('success', 'File berhasil diunggah.');
} else {
    // Tangani error upload
    return redirect()->back()->with('error', $file->getErrorString()
        . ' (' . $file->getError() . ')');
}
```

- **$file->getRandomName()**: Ini adalah praktik terbaik untuk menghasilkan nama file unik. Ini mencegah konflik nama file jika dua pengguna mengunggah file dengan nama yang sama, dan juga menyulitkan penyerang untuk menebak nama file yang diunggah.
- **WRITEPATH**: Konstanta CodeIgniter 4 yang menunjuk ke direktori writable/. Menggunakan ini memastikan jalur yang benar terlepas dari lokasi instalasi aplikasi.
- **$file->move(direktori_tujuan, nama_file_baru)**: Memindahkan file dari lokasi sementara ke direktori tujuan dengan nama baru.

3. Menyimpan Informasi File ke Database

Setelah file berhasil dipindahkan, penting untuk menyimpan informasi tentang file tersebut (nama file, path, ukuran, tipe MIME) ke database. Ini memungkinkan aplikasi untuk melacak file, menampilkannya, atau mengelolanya di kemudian hari.

```
// Contoh penyimpanan ke database
$dataToSave = [
    'user_id' => $userId,  'file_name' => $newName,
    'original_name' => $file->getName(),
    'file_path' => 'uploads/profiles/' . $newName, // Path relatif dari WRITEPATH
    'file_size' => $file->getSize(), 'file_type' => $file->getMimeType(),
    'uploaded_at' => date('Y-m-d H:i:s') ]; //
$fileModel->insert($dataToSave);
```

D. MENANGANI ERROR & LOGGING SISTEM

Penanganan error yang efektif adalah pilar utama dalam membangun aplikasi yang *robust* dan *user-friendly*. Dalam konteks upload file, berbagai jenis error dapat terjadi, mulai dari masalah sisi klien hingga masalah sisi server. Logging sistem membantu pengembang melacak dan mendiagnosis masalah ini.

1. Jenis-jenis Error dalam Upload File

- **Error Sisi Klien**:
  - Pengguna tidak memilih file.
  - Koneksi internet terputus saat upload.
  - Ukuran file melebihi batas yang diizinkan oleh browser atau JavaScript.
- **Error Sisi Server (PHP/CodeIgniter)**:
  - Ukuran file melebihi upload_max_filesize atau post_max_size di php.ini.
  - File yang diunggah sebagian (UPLOAD_ERR_PARTIAL).
  - Tidak ada file yang diunggah (UPLOAD_ERR_NO_FILE).
  - Gagal menulis file ke disk (UPLOAD_ERR_CANT_WRITE).
  - Ekstensi PHP yang diperlukan tidak diaktifkan.
  - Izin direktori tujuan tidak memadai (misalnya, server tidak memiliki izin tulis ke writable/uploads).
  - Validasi CodeIgniter gagal (ukuran, tipe MIME, ekstensi).

2. Penanganan Error di CodeIgniter 4

CodeIgniter 4 menyediakan mekanisme validasi yang mengembalikan pesan error yang dapat ditampilkan kepada pengguna. Untuk error yang lebih mendalam atau error sistem, kita perlu menggunakan blok *try-catch* dan *logging*.

```php
// Contoh penanganan error saat memindahkan file
try {
    $file = $this->request->getFile('userfile');
    if ($file->isValid() && ! $file->hasMoved()) {
        $newName = $file->getRandomName();
        $file->move(WRITEPATH . 'uploads/profiles', $newName); // ... sukses
    } else {
        // Error dari objek UploadedFile
        throw new \RuntimeException($file->getErrorString()
            . ' (' . $file->getError() . ')');
    }
} catch (\RuntimeException $e) {
    // Tangani error saat memindahkan file
    log_message('error', 'Gagal upload file: '
        . $e->getMessage());
    return redirect()->back()->with(
        'error',
        'Terjadi kesalahan saat mengunggah file: '
            . $e->getMessage()
    );
}
```

3. Logging Sistem

Logging adalah proses pencatatan peristiwa yang terjadi dalam aplikasi, termasuk error, peringatan, dan informasi debug. CodeIgniter 4 memiliki sistem logging yang kuat yang dapat dikonfigurasi untuk menulis log ke file, database, atau layanan eksternal.

- **log_message('level', 'pesan')**: Fungsi global untuk menulis pesan ke log.
  - level: emergency, alert, critical, error, warning, notice, info, debug.

- pesan: String yang menjelaskan peristiwa.

Contoh penggunaan logging:

```
try {
    // ... kode upload file
} catch (\RuntimeException $e) {
    log_message('error', 'Upload file gagal untuk user ID '
        . $userId . ': ' . $e->getMessage());
    // ... tampilkan pesan error ke pengguna
}
```

Log file CodeIgniter 4 secara default disimpan di writable/logs/. Memantau log ini secara teratur sangat penting untuk mengidentifikasi dan memperbaiki masalah dalam aplikasi (CodeIgniter Documentation, n.d.).

G. PRAKTIK: MODUL UPLOAD FOTO PROFIL ATAU DOKUMEN

Mari kita terapkan semua konsep yang telah dibahas untuk membangun modul upload foto profil sederhana.

1. Persiapan Database

Buat tabel users (jika belum ada) dengan kolom profile_picture untuk menyimpan nama file gambar.

```
CREATE TABLE users (
id INT AUTO_INCREMENT PRIMARY KEY,
username VARCHAR(255) NOT NULL UNIQUE,
email VARCHAR(255) NOT NULL UNIQUE,
password VARCHAR(255) NOT NULL,
role ENUM('admin', 'user', 'guest') DEFAULT 'user',
created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP );
```

2. Membuat View (Form Upload)

Buat file app/Views/profile/upload_form.php:

```
<!DOCTYPE html>
<html lang=en>

<head>
    <meta charset=UTF-8>
    <meta name=viewport content=width=device-width, initial-scale=1.0>
    <title>Upload Foto Profil</title>
    <style>
        body {font-family: Arial, sans-serif;margin: 20px;}
        .container{max-width: 600px;margin: auto;padding: 20px;border: 1px solid #ccc;border-radius: 8px;}
        .alert {padding: 10px;margin-bottom: 15px;border-radius: 4px;}
        .alert-success {background-color: #d4edda;color: #155724;border-color: #c3e6cb; }
        .alert-danger {background-color: #f8d7da;color: #721c24;border-color: #f5c6cb;}
        .form-group {margin-bottom: 15px;}
        label {display: block;margin-bottom: 5px;font-weight: bold;}
        input[type=file] {display: block;margin-top: 5px;}
        button {padding: 10px 20px;background-color: #007bff;color: white;border: none;border-radius: 5px;
            cursor: pointer; }
        button:hover {background-color: #0056b3;}
        .error-list {color: #dc3545;list-style-type: none;padding: 0;margin-top: 5px;}
    </style>
</head>
```

```
<body>
    <div class=container>
        <h2>Upload Foto Profil</h2>
        <?php
        if (session()->getFlashdata('success')): ?>
            <div class=alert alert-success><?= session()->getFlashdata('success') ?></div>
        <?php endif; ?>
        <?php
        if (session()->getFlashdata('error')): ?>
            <div class=alert alert-danger><?= session()->getFlashdata('error') ?></div>
        <?php endif; ?>
        <?php
        if (isset($errors)): ?> <div class=alert alert-danger>
                <ul class=error-list> <?php foreach ($errors as $error): ?>
                        <li><?= $error ?></li> <?php endforeach; ?>  </ul> </div> <?php endif; ?>
        <form action=<?= url_to('profile.upload') ?> method=post enctype=multipart/form-data>
            <?= csrf_field() ?>
            <div class=form-group>
                <label for=profile_image>Pilih Foto Profil:</label>
                <input type=file name=profile_image id=profile_image accept=image/jpeg,image/png>
                <small>Maksimal 1MB, format JPG/PNG.</small>
            </div>
            <button type=submit>Upload Foto</button>
        </form>
        <?php if (isset($user) && $user->profile_picture): ?><h3>Foto Profil Saat Ini:</h3>
        <img src=<?= base_url('uploads/profiles/'
        . $user->profile_picture) ?> alt=Foto Profil style=max-width: 200px; height: auto; border-radius: 50%
        <?php endif; ?>
    </div>
</body>
```

Catatan: Untuk menampilkan gambar yang diunggah dari writable/uploads, Anda perlu membuat route atau symlink agar folder tersebut dapat diakses publik. Cara paling sederhana untuk praktik adalah dengan membuat folder uploads/profiles di dalam public dan memindahkan file ke sana. Namun, untuk keamanan yang lebih baik, disarankan untuk membuat Controller yang membaca file dari writable dan mengirimkannya ke browser. Untuk tujuan praktik ini, kita asumsikan public/uploads/profiles adalah target.

3. Membuat Controller

Buat file app/Controllers/ProfileController.php:

```
<?php
namespace App\Controllers;
use App\Models\UserModel;
use CodeIgniter\Controller;
class ProfileController extends Controller
{
    protected $userModel;
    public function __construct() {
        $this->userModel = new UserModel(); }
    public function index() {
        // Asumsi user dengan ID 1 sedang login
        $userId = 1;
        $user = $this->userModel->find($userId);
        return view('profile/upload_form', ['user' => $user]); }
    public function uploadProfilePicture() {
        $userId = 1; // Asumsi user dengan ID 1 sedang login
        $validationRules = [
            'profile_image' => [
                'rules' => 'uploaded[profile_image]|
                    max_size[profile_image,1024]|ext_in[profile_image,jpg,jpeg,png]|
                    mime_in[profile_image,image/jpg,image/jpeg,image/png]',
                'errors' => [ 'uploaded' => 'Anda harus memilih file untuk diunggah.',
                'max_size' => 'Ukuran file {field} terlalu besar. Maksimal 1MB.',
                'ext_in' => 'Format file {field} tidak diizinkan. Hanya JPG, JPEG, PNG.',
                'mime_in' => 'Tipe MIME file {field} tidak valid.' ] ] ];
```

```
if (! $this->validate($validationRules)) {
    return redirect()->back()->withInput()->with('errors', $this->validator->getErrors()); }
$file = $this->request->getFile('profile_image');
try { if ($file->isValid() && ! $file->hasMoved()) {
    $newName = $file->getRandomName(); // Untuk praktik, kita simpan di public/uploads/profiles
    // Pastikan folder public/uploads/profiles sudah ada dan writable
    $uploadPath = FCPATH . 'uploads/profiles'; // FCPATH menunjuk ke direktori public
    if (!is_dir($uploadPath)) {
        mkdir($uploadPath, 0777, true); // Buat folder jika belum ada
        }
    $file->move($uploadPath, $newName); // Update nama file di database
    $this->userModel->update($userId, ['profile_picture' => $newName]);
    return redirect()->back()->with('success', 'Foto profil berhasil diunggah.'); }
    else {
        throw new \RuntimeException($file->getErrorString()
        . ' (' . $file->getError() . ')'); } }
        catch (\RuntimeException $e) { log_message('error', 'Gagal upload foto profil untuk user ID '
            . $userId . ': ' . $e->getMessage());
        return redirect()->back()->with('error', 'Terjadi kesalahan saat mengunggah file: '
        . $e->getMessage()); }
    }
}
```

4. Membuat Model

Buat file app/Models/UserModel.php:

```
<?php
// app/Models/UserModel.php
namespace App\Models;

use CodeIgniter\Model;

class UserModel extends Model
{
    protected $table = 'users';
    protected $primaryKey = 'id';
    protected $useAutoIncrement = true;
    protected $returnType = 'object';
    protected $useSoftDeletes = false;
    protected $allowedFields = ['username', 'email', 'password', 'profile_picture'];
    protected $useTimestamps = true;
    protected $createdField = 'created_at';
    protected $updatedField = 'updated_at';
    protected $deletedField = 'deleted_at';
    protected $validationRules = [];
    protected $validationMessages = [];
    protected $skipValidation = false;
}
```

5. Konfigurasi Routing

Tambahkan route di app/Config/Routes.php:

```
$routes->get('profile', 'ProfileController::index', ['as' => 'profile.index']);
$routes->post('profile/upload', 'ProfileController::uploadProfilePicture', ['as' => 'profile.upload']);
```

6. Pengujian

1. Pastikan Anda memiliki database yang terhubung dan tabel users sudah ada.
2. Buat folder public/uploads/profiles secara manual dan pastikan memiliki izin tulis (misalnya 777 untuk pengembangan, sesuaikan di produksi).
3. Akses http://localhost:8080/profile di browser Anda.
4. Coba unggah file gambar dengan berbagai skenario:
    - File gambar yang valid (JPG/PNG, < 1MB).

- File gambar yang terlalu besar.
- File dengan format yang tidak diizinkan (misalnya PDF, TXT).
- Tidak memilih file sama sekali.

Perhatikan pesan error yang muncul dan bagaimana sistem menangani setiap skenario. Periksa juga log di writable/logs/ untuk melihat catatan error.

Melalui praktik ini, Anda telah mengimplementasikan seluruh siklus upload file, mulai dari form HTML, validasi sisi server, penyimpanan file, hingga penanganan error dan logging, yang merupakan fondasi penting untuk fitur manajemen media dalam aplikasi web Anda.

E. RANGKUMAN

Pembahasan mengenai upload gambar/dokumen dalam aplikasi web modern, khususnya menggunakan CodeIgniter 4, mencakup serangkaian langkah krusial mulai dari interaksi pengguna hingga penanganan keamanan dan penyimpanan. Berikut adalah poin-poin penting yang dirangkum dari pembahasan di atas:

1. **Mekanisme Dasar Upload HTML**: Proses upload dimulai dari form HTML yang menggunakan <input type=file> dan atribut enctype=multipart/form-data pada tag <form>. Atribut ini esensial untuk memungkinkan pengiriman data biner (file) ke server melalui request HTTP POST.
2. **Pengelolaan File di CodeIgniter 4**: CodeIgniter 4 menyederhanakan penanganan file yang diunggah melalui objek UploadedFile. Objek ini diakses dari objek Request menggunakan metode getFile() dan menyediakan fungsionalitas untuk memeriksa validitas file (isValid()) serta status pemindahannya (hasMoved()).
3. **Validasi File yang Komprehensif**: Validasi adalah langkah keamanan vital. Ini mencakup:
   - **Ukuran File**: Menggunakan aturan max_size di CodeIgniter 4 dan konfigurasi upload_max_filesize serta post_max_size di php.ini untuk mencegah pengisian server berlebihan.
   - **Format File**: Memastikan hanya format yang diizinkan melalui validasi ekstensi (ext_in) dan tipe MIME (mime_in), yang terakhir lebih andal karena memeriksa tipe konten aktual.
   - **Keamanan Tambahan**: Meliputi pembersihan nama file, pemrosesan ulang gambar untuk menghilangkan metadata berbahaya, dan potensi integrasi dengan pemindai antivirus.
4. **Penyimpanan File yang Terorganisir**: Setelah validasi, file dipindahkan dari direktori sementara PHP ke lokasi permanen. Disarankan menggunakan folder writable/uploads atau sub-folder di public dengan struktur yang jelas. Metode move() dari objek UploadedFile digunakan untuk memindahkan file, seringkali dengan nama unik yang dihasilkan oleh getRandomName() untuk menghindari konflik.
5. **Pencatatan Informasi File ke Database**: Penting untuk menyimpan metadata file (nama, path, ukuran, tipe MIME) ke database. Ini memfasilitasi pelacakan, pengelolaan, dan penayangan file di kemudian hari.
6. **Penanganan Error dan Logging Sistem**: Aplikasi yang robust memerlukan penanganan error yang efektif. Berbagai error dapat terjadi, mulai dari sisi klien hingga server (misalnya, izin direktori, batas ukuran PHP, kegagalan validasi CodeIgniter). CodeIgniter 4 menyediakan pesan validasi dan fungsi log_message() untuk mencatat peristiwa penting, membantu dalam debugging dan pemeliharaan sistem.

7. **Praktik Implementasi Modul Upload**: Seluruh konsep di atas diintegrasikan dalam modul upload foto profil. Ini melibatkan persiapan database, pembuatan view dengan form upload, pengembangan controller untuk menangani logika upload dan validasi, serta konfigurasi routing. Praktik ini menunjukkan bagaimana CodeIgniter 4 memfasilitasi pembangunan fitur upload file yang aman dan fungsional.

BAB 10
PEMBUATAN LAPORAN (PDF, EXCEL, DAN PRINTABLE)

Dalam pengembangan sistem informasi berbasis web, kemampuan untuk menyajikan data dalam bentuk laporan yang terstruktur dan mudah dibaca merupakan aspek krusial. Data yang telah diinput, diproses, dan disimpan dalam database akan kehilangan sebagian besar nilainya jika tidak dapat diakses dan dianalisis dalam format yang relevan bagi pengguna. Bab 10 ini akan membawa kita menyelami dunia reporting, sebuah komponen esensial yang menjembatani data mentah dengan kebutuhan informasi strategis dan operasional sebuah organisasi.

Reporting bukan sekadar menampilkan data, melainkan seni dan ilmu untuk mengubah data menjadi wawasan yang dapat ditindaklanjuti. Bayangkan sebuah sistem informasi penjualan tanpa laporan penjualan bulanan, atau sistem akademik tanpa laporan nilai siswa. Tanpa laporan, pengambilan keputusan akan menjadi spekulatif dan kurang berbasis data. Oleh karena itu, urgensi reporting terletak pada kemampuannya untuk memberikan gambaran menyeluruh tentang kinerja sistem, memantau tren, mengidentifikasi masalah, dan mendukung proses pengambilan keputusan yang lebih baik.

Modul ini akan membahas berbagai metode dan teknologi untuk menghasilkan laporan yang efektif. Kita akan memulai dengan memahami konsep dasar reporting, kemudian beralih ke implementasi praktis. Mahasiswa akan diajak untuk membuat laporan dalam format HTML yang dapat langsung dicetak, sebuah fondasi penting sebelum melangkah lebih jauh. Selanjutnya, kita akan mempelajari cara mengintegrasikan library pihak ketiga seperti dompdf atau mpdf untuk menghasilkan laporan dalam format PDF, yang sangat populer karena portabilitas dan konsistensinya di berbagai platform. Tidak kalah penting, kemampuan untuk mengekspor data ke format Excel juga akan dibahas, mengingat format ini seringkali menjadi pilihan utama untuk analisis data lebih lanjut. Melalui serangkaian praktik, mahasiswa akan membangun laporan daftar data dan laporan transaksi, mengaplikasikan semua konsep yang telah dipelajari untuk menciptakan output yang informatif dan fungsional.

A. KONSEP REPORTING PADA SISTEM INFORMASI

Reporting adalah proses mengubah data mentah yang tersimpan dalam database menjadi informasi yang terstruktur, mudah dibaca, dan bermakna bagi pengguna. Dalam konteks sistem informasi berbasis web, reporting menjadi jembatan antara data operasional sehari-hari dengan kebutuhan analisis dan pengambilan keputusan strategis. Laporan tidak hanya menyajikan fakta, tetapi juga membantu mengidentifikasi tren, anomali, dan kinerja sistem secara keseluruhan (Chaudhuri & Dayal, 2020).

1. Urgensi Reporting dalam Sistem Informasi

Urgensi reporting tidak dapat diremehkan. Tanpa laporan yang efektif, data yang terkumpul dalam sistem akan menjadi kuburan data yang tidak memberikan nilai tambah. Beberapa alasan utama mengapa reporting sangat penting meliputi:

a. **Pengambilan Keputusan Berbasis Data**: Laporan menyediakan ringkasan dan analisis data yang diperlukan manajer untuk membuat keputusan yang tepat, mulai dari keputusan operasional harian hingga keputusan strategis jangka panjang (Davenport & Harris, 2017). Misalnya, laporan penjualan dapat membantu menentukan produk mana yang paling laris atau periode mana yang membutuhkan promosi khusus.

b. **Pemantauan Kinerja**: Laporan memungkinkan organisasi untuk memantau kinerja berbagai aspek bisnis, seperti penjualan, inventaris, keuangan, atau kehadiran karyawan. Dengan membandingkan data aktual dengan target atau data historis, organisasi dapat mengidentifikasi area yang memerlukan perbaikan.
c. **Kepatuhan dan Audit**: Banyak industri memiliki persyaratan kepatuhan regulasi yang mengharuskan pembuatan laporan tertentu. Laporan juga penting untuk tujuan audit internal maupun eksternal, memastikan transparansi dan akuntabilitas.
d. **Komunikasi Informasi**: Laporan berfungsi sebagai alat komunikasi yang efektif untuk menyebarkan informasi penting kepada berbagai pemangku kepentingan, baik internal maupun eksternal.

2. Jenis-jenis Laporan

Laporan dapat dikategorikan berdasarkan tujuan, frekuensi, dan tingkat detailnya:
a. **Laporan Operasional**: Laporan ini berfokus pada aktivitas harian dan detail transaksi. Contohnya adalah laporan daftar transaksi harian, laporan stok barang, atau laporan kehadiran karyawan. Laporan ini seringkali dibutuhkan secara real-time atau harian.
b. **Laporan Manajerial**: Laporan ini menyajikan ringkasan data operasional untuk membantu manajer dalam pengambilan keputusan taktis. Contohnya adalah laporan penjualan bulanan, laporan laba rugi, atau laporan kinerja proyek. Laporan ini biasanya bersifat periodik (mingguan, bulanan, kuartalan).
c. **Laporan Strategis**: Laporan ini memberikan gambaran besar tentang kinerja organisasi dalam jangka panjang, mendukung perencanaan strategis. Contohnya adalah laporan tren pasar, analisis profitabilitas produk, atau laporan pertumbuhan pelanggan. Laporan ini seringkali bersifat tahunan atau sesuai kebutuhan strategis.

3. Komponen Penting dalam Laporan

Laporan yang baik harus memiliki beberapa komponen kunci untuk memastikan informasinya jelas dan mudah dipahami:
a. **Judul Laporan**: Memberikan identifikasi jelas tentang isi laporan. b. **Header/Footer**: Berisi informasi seperti tanggal pembuatan, nomor halaman, nama perusahaan, atau logo. c. **Parameter Laporan**: Menunjukkan kriteria filter yang digunakan (misalnya, rentang tanggal, kategori produk). d. **Isi Laporan**: Data yang disajikan, seringkali dalam bentuk tabel, grafik, atau kombinasi keduanya. e. **Ringkasan/Total**: Agregasi data seperti total jumlah, rata-rata, atau subtotal.

B. MEMBUAT LAPORAN HTML SIAP CETAK

Laporan HTML siap cetak adalah fondasi dari semua jenis laporan berbasis web. Dengan merancang laporan dalam format HTML yang responsif dan dioptimalkan untuk pencetakan, kita dapat memastikan bahwa tampilan laporan konsisten baik di layar maupun saat dicetak.

1. Struktur Dasar Laporan HTML

Laporan HTML pada dasarnya adalah halaman web biasa, namun dengan penekanan pada tata letak yang bersih dan minimalis. Struktur umumnya melibatkan:

```
<!DOCTYPE html>
<html lang=id>
<head>
    <meta charset=UTF-8>
    <meta name=viewport content=width=device-width, initial-scale=1.0>
    <title>Laporan Daftar Data</title>
    <link rel=stylesheet href=path/to/print.css media=print>
    <link rel=stylesheet href=path/to/screen.css media=screen>
</head>
```

```
<body>
    <div class=container>
        <header>
            <h1>Laporan Daftar Pegawai</h1>
            <p>Periode: 01 Januari 2025 - 31 Desember 2025</p>
        </header>
        <main>
            <table>
                <thead>
                    <tr>
                        <th>No</th>
                        <th>Nama Pegawai</th>
                        <th>Jabatan</th>
                        <th>Gaji</th>
                    </tr>
                </thead>
                <tbody> <!-- Data akan diisi di sini --> </tbody>
            </table>
        </main>
        <footer> <p>Dicetak pada: 2025-12-10</p> </footer>
    </div>
</body>
</html>
```

2. Mengoptimalkan CSS untuk Pencetakan

Kunci untuk laporan HTML siap cetak adalah penggunaan media queries dalam CSS. Kita dapat mendefinisikan gaya yang berbeda untuk tampilan layar (media=screen) dan tampilan cetak (media=print).

```
/* screen.css */
body {
  font-family: Arial, sans-serif;margin: 20px;
} /* ... gaya untuk tampilan layar lainnya ... */ /* print.css */
@page {
  size: A4 portrait; /* Ukuran kertas dan orientasi */
  margin: 1cm; /* Margin halaman */ }
body {
  font-family: Times New Roman, serif; font-size: 10pt; margin: 0; padding: 0;}
header,
footer {
  text-align: center; margin-bottom: 10px;}
table {
  width: 100%;  border-collapse: collapse;  margin-top: 15px; }
th,
td {
  border: 1px solid #000; padding: 8px; text-align: left;
} /* Sembunyikan elemen yang tidak perlu dicetak */
.no-print {
  display: none;
}
```

Dengan media=print, kita dapat menyembunyikan elemen navigasi, tombol, atau iklan yang tidak relevan untuk dicetak. Kita juga dapat mengatur ukuran font, margin, dan tata letak agar sesuai dengan kertas fisik.

3. Integrasi dengan CodeIgniter 4

Dalam CodeIgniter 4, laporan HTML dapat dibuat menggunakan View. Controller akan mengambil data dari Model, kemudian meneruskannya ke View yang dirancang khusus untuk laporan.

```php
<?php
// app/Controllers/Laporan.php
namespace App\Controllers;
use App\Models\PegawaiModel;
class Laporan extends BaseController
{
    public function daftarPegawai()
    {
        $model = new PegawaiModel();
        $data['pegawai'] = $model->findAll();
        return view('laporan/daftar_pegawai_html', $data);
    }
}
```

```php
<!-- app/Views/laporan/daftar_pegawai_html.php -->
<!DOCTYPE html>
<html lang=id>

<head>
    <meta charset=UTF-8>
    <meta name=viewport content=width=device-width, initial-scale=1.0>
    <title>Laporan Daftar Pegawai</title>
    <style>
        /* Inline CSS untuk print, atau link ke print.css */
        @page {size: A4 portrait;margin: 1cm;}
        body { font-family: Times New Roman, serif; font-size: 10pt; margin: 0; padding: 0;  }
        table { width: 100%; border-collapse: collapse; margin-top: 15px;}
        th, td {border: 1px solid #000; padding: 8px; text-align: left; }
        h1, p {text-align: center;}
    </style>
</head>
```

C. GENERATE LAPORAN PDF (DOMPDF/MPDF)

Meskipun laporan HTML siap cetak sudah cukup baik, format PDF menawarkan portabilitas dan konsistensi yang lebih tinggi. Dokumen PDF akan terlihat sama persis di berbagai perangkat dan sistem operasi, menjadikannya pilihan ideal untuk distribusi laporan resmi. Dua library populer untuk menghasilkan PDF dari HTML di PHP adalah dompdf dan mpdf.

1. Menggunakan dompdf

dompdf adalah *library PHP* yang memungkinkan konversi HTML dan CSS menjadi PDF. Ini relatif mudah digunakan dan cocok untuk sebagian besar kebutuhan laporan.

a. **Instalasi dompdf**: Gunakan Composer untuk menginstal dompdf di proyek CodeIgniter 4 Anda: bash composer require dompdf/dompdf

b. **Penggunaan di Controller**: Setelah terinstal, Anda dapat menggunakannya di Controller untuk mengonversi View HTML menjadi PDF.

```php
<?php
// app/Controllers/Laporan.php
namespace App\Controllers;
use App\Models\PegawaiModel;
use Dompdf\Dompdf;
use Dompdf\Options;
```

```php
class Laporan extends BaseController
{
    public function daftarPegawai()
    {
        $model = new PegawaiModel();
        $data['pegawai'] = $model->findAll();
        return view('laporan/daftar_pegawai_html', $data);
    }
    public function daftarPegawaiPdf()
    {
        $model = new PegawaiModel();
        $data['pegawai'] = $model->findAll();
        // Load view HTML
        $html = view('laporan/daftar_pegawai_html', $data); // Gunakan view HTML yang sama
        // Konfigurasi Dompdf
        $options = new Options();
        $options->set('isHtml5ParserEnabled', true);
        $options->set('isRemoteEnabled', true);
        // Izinkan akses ke resource eksternal (gambar, CSS)
        $dompdf = new Dompdf($options);
        $dompdf->loadHtml($html);
        $dompdf->setPaper('A4', 'portrait'); // Ukuran dan orientasi kertas
        $dompdf->render(); // Output PDF ke browser atau simpan ke file
        $dompdf->stream('Laporan_Daftar_Pegawai.pdf', ['Attachment' => 0]);
        // 0 = tampil di browser, 1 = langsung download
        exit();
    }
}
```

Pastikan `view('laporan/daftar_pegawai_html', $data)` mengembalikan string HTML, bukan langsung merender ke browser.

2. Menggunakan mpdf

mpdf adalah *library* lain yang sangat kuat dan fleksibel untuk menghasilkan PDF. mpdf seringkali lebih baik dalam menangani CSS kompleks dan tata letak yang rumit dibandingkan dompdf, meskipun mungkin sedikit lebih berat dalam hal sumber daya.

a. **Instalasi mpdf**: *bash composer require mpdf/mpdf*

b. **Penggunaan di Controller**:

```php
<?php
// app/Controllers/Laporan.php
namespace App\Controllers;
use App\Models\PegawaiModel;
use Mpdf\Mpdf;

class Laporan extends BaseController
{
    public function daftarPegawaiMpdf()
    {
        $model = new PegawaiModel();
        $data['pegawai'] = $model->findAll();
        // Load view HTML
        $html = view('laporan/daftar_pegawai_html', $data);
        // Konfigurasi mpdf
        $mpdf = new Mpdf([ 'mode' => 'utf-8', 'format' => 'A4', 'orientation' => 'P' ]);
        // P = Portrait, L = Landscape
        $mpdf->WriteHTML($html); $mpdf->Output('Laporan_Daftar_Pegawai_Mpdf.pdf', 'I');
        // 'I' = Inline (tampil di browser), 'D' = Download
        exit();
    }
}
```

Pilihan antara *dompdf* dan *mpdf* seringkali tergantung pada kompleksitas laporan dan preferensi pengembang. Untuk laporan sederhana, dompdf sudah memadai, sementara mpdf lebih cocok untuk laporan yang membutuhkan kontrol tata letak yang lebih presisi (Suryanarayana et al., 2021).

D. EXPORT EXCEL (SPREADSHEET LIBRARY)

Mengekspor data ke format Excel adalah fitur yang sangat dibutuhkan dalam sistem informasi. Pengguna seringkali ingin menganalisis data lebih lanjut, membuat grafik kustom, atau mengintegrasikannya dengan aplikasi lain. PHPSpreadsheet adalah library yang sangat populer dan powerful untuk membaca dan menulis file spreadsheet (Excel, CSV, ODS, dll.) di PHP.

1. Instalasi PHPSpreadsheet

Instal PHPSpreadsheet menggunakan Composer:

```
composer require phpoffice/phpspreadsheet
```

2. Penggunaan di Controller

Setelah terinstal, Anda dapat membuat Controller untuk mengambil data dari database dan mengekspornya ke Excel.

```php
namespace App\Controllers;
use App\Models\PegawaiModel;
use PhpOffice\PhpSpreadsheet\Spreadsheet;
use PhpOffice\PhpSpreadsheet\Writer\Xlsx;
class Laporan extends BaseController
{
    public function daftarPegawaiExcel()
    {
        $model = new PegawaiModel();
        $pegawai = $model->findAll();
        $spreadsheet = new Spreadsheet();
        $sheet = $spreadsheet->getActiveSheet();
        $sheet->setCellValue('A1', 'No');
        $sheet->setCellValue('B1', 'Nama');
        $sheet->setCellValue('C1', 'Jabatan');
        $sheet->setCellValue('D1', 'Gaji');
        $row = 2; // Data ditampilkan Mulai dari baris kedua setelah header
        $no = 1;
        foreach ($pegawai as $p) {
            $sheet->setCellValue('A' . $row, $no++);
            $sheet->setCellValue('B' . $row, $p['nama']);
            $sheet->setCellValue('C' . $row, $p['jabatan']);
            $sheet->setCellValue('D' . $row, $p['gaji']); $row++; }
        foreach (range('A', 'D') as $column) { $sheet->getColumnDimension($column)->setAutoSize(true); }
        $writer = new Xlsx($spreadsheet);
        $fileName = 'Laporan_Daftar_Pegawai_' . date('YmdHis') . '.xlsx';
        header('Content-Type: application/vnd.openxmlformats-officedocument.spreadsheetml.sheet');
        header('Content-Disposition: attachment;filename=' . $fileName . '');
        header('Cache-Control: max-age=0'); $writer->save('php://output');
        exit();
    }
```

PHPSpreadsheet memungkinkan kontrol yang sangat detail terhadap format sel, gaya, grafik, dan fitur Excel lainnya, menjadikannya alat yang sangat fleksibel untuk kebutuhan ekspor data (PHPSpreadsheet, n.d.).

E. PRAKTIK: LAPORAN DAFTAR DATA & LAPORAN TRANSAKSI

Untuk mengkonsolidasikan pemahaman, kita akan menerapkan semua konsep yang telah dibahas untuk membuat dua jenis laporan umum: laporan daftar data (misalnya, daftar pegawai) dan laporan transaksi (misalnya, laporan penjualan).

1. Laporan Daftar Data (Contoh: Daftar Pegawai)

Kita telah menggunakan contoh daftar pegawai di bagian sebelumnya. Praktiknya melibatkan langkah-langkah berikut:

a. **Model**: Pastikan Anda memiliki PegawaiModel yang dapat mengambil data pegawai dari database. b. **Controller**: Buat metode di Laporan Controller untuk mengambil data pegawai. c. **View HTML**: Buat View khusus (laporan/daftar_pegawai_html.php) yang menampilkan data pegawai dalam format tabel yang rapi, dengan CSS yang dioptimalkan untuk cetak. d. **PDF (dompdf/mpdf)**: Tambahkan metode di Controller untuk mengonversi View HTML tersebut menjadi PDF menggunakan dompdf atau mpdf. e. **Excel (PHPSpreadsheet)**: Tambahkan metode di Controller untuk mengekspor data pegawai langsung ke file Excel.

2. Laporan Transaksi (Contoh: Laporan Penjualan)

Laporan transaksi biasanya lebih kompleks karena melibatkan data dari beberapa tabel (misalnya, transaksi, detail transaksi, produk, pelanggan).

a. **Perancangan Database**: Asumsikan Anda memiliki tabel penjualan (id, tanggal, total_harga, id_pelanggan) dan detail_penjualan (id, id_penjualan, id_produk, jumlah, harga_satuan). b. **Model**: Buat PenjualanModel yang dapat mengambil data transaksi beserta detailnya, mungkin dengan menggunakan *join* atau *relasi* (seperti yang dibahas di Bab 5).

```php
<?php
// app/Models/PenjualanModel.php
namespace App\Models;
use CodeIgniter\Model;
class PenjualanModel extends Model
{
    protected $table = 'penjualan';
    protected $primaryKey = 'id';
    protected $allowedFields = ['tanggal', 'total_harga', 'id_pelanggan'];
    public function getPenjualanDenganDetail($startDate, $endDate)
    {
        return $this->db->table('penjualan p') ->select('p.*, dp.id_produk, dp.jumlah, dp.harga_satuan,
        pr.nama_produk') ->join('detail_penjualan dp', 'dp.id_penjualan = p.id') ->join('produk pr',
        'pr.id = dp.id_produk') ->where('p.tanggal >=', $startDate) ->where('p.tanggal <=',
        $endDate) ->get() ->getResultArray();
    }
}
```

c. **Controller**: Buat metode di Laporan Controller untuk mengambil data penjualan berdasarkan rentang tanggal.

```php
<?php
// app/Controllers/Laporan.php
namespace App\Controllers;
use App\Models\PenjualanModel;
use Dompdf\Dompdf; use Dompdf\Options;
class Laporan extends BaseController
{
    public function laporanPenjualan()
    {
        $startDate = $this->request->getGet('start_date') ?? date('Y-m-01');
        $endDate = $this->request->getGet('end_date') ?? date('Y-m-t');
        $model = new PenjualanModel();
        $data['penjualan'] = $model->getPenjualanDenganDetail($startDate, $endDate);
        $data['startDate'] = $startDate;  $data['endDate'] = $endDate;
        return view('laporan/laporan_penjualan_html', $data);
    }
```

```
public function laporanPenjualanPdf() {
    $startDate = $this->request->getGet('start_date') ?? date('Y-m-01');
    $endDate = $this->request->getGet('end_date') ?? date('Y-m-t');
    $model = new PenjualanModel();
    $data['penjualan'] = $model->getPenjualanDenganDetail($startDate, $endDate);
    $data['startDate'] = $startDate; $data['endDate'] = $endDate;
    $html = view('laporan/laporan_penjualan_html', $data);
    $options = new Options(); $options->set('isHtml5ParserEnabled', true);
    $options->set('isRemoteEnabled', true);
    $dompdf = new Dompdf($options); $dompdf->loadHtml($html);
    $dompdf->setPaper('A4', 'portrait'); $dompdf->render();
    $dompdf->stream('Laporan_Penjualan_' . $startDate . '_' . $endDate
    . '.pdf', ['Attachment' => 0]);
    exit();  }
```

d. **View HTML**: Buat View (laporan/laporan_penjualan_html.php) yang menampilkan data transaksi, mungkin dengan pengelompokan berdasarkan ID transaksi dan subtotal untuk setiap transaksi.

```
<!-- app/Views/laporan/laporan_penjualan_html.php -->
<!DOCTYPE html>
<html lang=id>

<head>
    <meta charset=UTF-8>
    <meta name=viewport content=width=device-width, initial-scale=1.0>
    <title>Laporan Penjualan</title>
    <style>
        @page {size: A4 portrait; margin: 1cm;}
        body {font-family: Times New Roman, serif; font-size: 10pt; margin: 0;padding: 0; }
        table {width: 100%; border-collapse: collapse;margin-top: 15px;}
        th,td {border: 1px solid #000; padding: 8px;text-align: left;}
        .text-right {text-align: right;}
        h1, p {text-align: center;}
    </style>
</head>
```

```
<body>
    <h1>Laporan Penjualan</h1>
    <p>Periode: <?= date('d-m-Y', strtotime($startDate)) ?> s/d <?= date('d-m-Y', strtotime($endDate)) ?></p>
    <table>
        <thead>
            <tr>
                <th>No Transaksi</th>
                <th>Tanggal</th>
                <th>Produk</th>
                <th>Jumlah</th>
                <th>Harga Satuan</th>
                <th>Subtotal</th>
            </tr>
        </thead>
        <tbody> <?php $currentPenjualanId = null; $grandTotal = 0;
        foreach ($penjualan as $item): if ($item['id'] !== $currentPenjualanId):
        if ($currentPenjualanId !== null): ?>
        <tr><td colspan=5 class=text-right><strong>Total Transaksi <?= $currentPenjualanId ?>:</strong></td>
        <td class=text-right><strong><?= number_format($subTotalTransaksi, 0, ',', '.') ?></strong></td></tr>
        <?php endif; $currentPenjualanId = $item['id']; $subTotalTransaksi = 0; endif;
        $itemSubtotal = $item['jumlah'] * $item['harga_satuan'];  $subTotalTransaksi += $itemSubtotal;
        $grandTotal += $itemSubtotal; ?> <tr>
```

```
        <td><?= $item['id'] ?></td>
        <td><?= date('d-m-Y', strtotime($item['tanggal'])) ?></td>
        <td><?= $item['nama_produk'] ?></td>
        <td class=text-right><?= $item['jumlah'] ?></td>
        <td class=text-right><?= number_format($item['harga_satuan'], 0, ',', '.') ?></td>
        <td class=text-right><?= number_format($itemSubtotal, 0, ',', '.') ?></td>
        </tr> <?php endforeach; ?> <?php if ($currentPenjualanId !== null): ?> <tr>
        <td colspan=5 class=text-right><strong>Total Transaksi <?= $currentPenjualanId ?>:</strong></td>
        <td class=text-right><strong><?= number_format($subTotalTransaksi, 0, ',', '.') ?></strong></td>
        </tr> <?php endif; ?> <tr><td colspan=5 class=text-right><strong>GRAND TOTAL:</strong></td>
        <td class=text-right><strong><?= number_format($grandTotal, 0, ',', '.') ?></strong></td> </tr>
        </tbody>
    </table>
</body>
</html>
```

e. **Excel (PHPSpreadsheet)**: Tambahkan metode untuk mengekspor laporan penjualan ke Excel, dengan struktur yang serupa dengan laporan daftar data, namun mungkin dengan kolom tambahan untuk subtotal dan total keseluruhan.

Melalui praktik ini, mahasiswa akan mendapatkan pengalaman langsung dalam merancang, mengimplementasikan, dan menghasilkan berbagai format laporan yang esensial dalam pengembangan sistem informasi berbasis web.

F. RANGKUMAN

Bagian ini merangkum pembahasan mendalam mengenai konsep reporting dalam sistem informasi, mulai dari urgensinya hingga implementasi praktis dalam pengembangan aplikasi berbasis web menggunakan CodeIgniter 4. Pemahaman tentang reporting sangat krusial karena laporan berfungsi sebagai jembatan antara data mentah dan informasi yang dapat digunakan untuk pengambilan keputusan strategis.

1. **Konsep dan Urgensi Reporting**: Reporting adalah proses esensial untuk mengubah data mentah menjadi informasi terstruktur. Urgensinya terletak pada kemampuannya mendukung pengambilan keputusan berbasis data, memantau kinerja, memastikan kepatuhan regulasi, dan memfasilitasi komunikasi informasi antar pemangku kepentingan. Laporan dapat dikategorikan menjadi operasional, manajerial, dan strategis, masing-masing dengan tujuan dan tingkat detail yang berbeda. Komponen laporan yang baik meliputi judul, header/footer, parameter, isi data, dan ringkasan.
2. **Laporan HTML Siap Cetak**: Fondasi laporan berbasis web adalah HTML yang dioptimalkan untuk cetak. Ini dicapai melalui penggunaan CSS media queries (media=print) untuk mengatur gaya khusus pencetakan, seperti ukuran kertas, margin, font, dan menyembunyikan elemen yang tidak relevan. Dalam CodeIgniter 4, laporan HTML dibuat menggunakan View, di mana Controller mengambil data dari Model dan meneruskannya ke View.
3. **Generate Laporan PDF (dompdf/mpdf)**: Untuk portabilitas dan konsistensi tampilan, laporan seringkali dikonversi ke format PDF. Library PHP seperti dompdf dan mpdf memungkinkan konversi HTML dan CSS menjadi PDF. dompdf relatif mudah digunakan untuk laporan standar, sementara mpdf menawarkan kontrol lebih baik untuk tata letak kompleks. Prosesnya melibatkan pengambilan data, rendering ke View HTML, dan kemudian menggunakan library PDF untuk mengonversi HTML tersebut menjadi file PDF yang dapat diunduh atau ditampilkan di browser.
4. **Export Excel (PHPSpreadsheet)**: Fitur ekspor ke Excel sangat penting untuk analisis data lebih lanjut. PHPSpreadsheet adalah library PHP yang powerful untuk membuat file spreadsheet. Implementasinya melibatkan pengambilan data dari Model,

membuat objek Spreadsheet, mengisi header dan data ke dalam sheet, dan kemudian mengatur header HTTP untuk mengunduh file Excel yang dihasilkan.

5. **Praktik Laporan Daftar Data dan Transaksi**: Konsep-konsep ini dikonsolidasikan melalui praktik pembuatan dua jenis laporan: laporan daftar data (misalnya, daftar pegawai) dan laporan transaksi (misalnya, laporan penjualan). Laporan daftar data relatif sederhana, sementara laporan transaksi seringkali melibatkan data dari beberapa tabel dan memerlukan logika pengelompokan serta perhitungan subtotal dan grand total. Praktik ini mencakup penggunaan Model untuk mengambil data, Controller untuk memproses dan meneruskan data, serta View untuk menampilkan laporan dalam format HTML, PDF, dan Excel.

BAB 11
VISUALISASI DATA MENGGUNAKAN GRAFIK

Dalam era digital saat ini, data telah menjadi aset yang sangat berharga bagi setiap organisasi. Namun, tumpukan data mentah, meskipun lengkap, seringkali sulit untuk dipahami dan diinterpretasikan secara cepat. Di sinilah peran visualisasi data menjadi krusial. Visualisasi data adalah seni dan ilmu untuk menyajikan data dalam format grafis, seperti grafik, diagram, atau peta, yang memungkinkan pengguna untuk dengan mudah mengidentifikasi pola, tren, dan anomali yang tersembunyi dalam data. Dalam konteks sistem informasi berbasis web, kemampuan untuk menyajikan data secara visual tidak hanya meningkatkan estetika aplikasi, tetapi juga secara signifikan meningkatkan efektivitas pengambilan keputusan.

Bab 11 ini akan membawa Anda lebih dalam ke dunia visualisasi data dalam pengembangan aplikasi web menggunakan CodeIgniter 4. Kita akan menjelajahi mengapa grafik menjadi komponen penting dalam dashboard dan laporan sistem informasi, serta bagaimana grafik dapat mengubah data yang kompleks menjadi informasi yang mudah dicerna. Modul ini akan membahas konsep dasar di balik berbagai jenis grafik, mulai dari grafik batang yang ideal untuk perbandingan kategori, grafik garis untuk menunjukkan tren waktu, hingga grafik pie untuk merepresentasikan proporsi.

Lebih lanjut, kita akan mempelajari cara mengintegrasikan pustaka JavaScript populer seperti Chart.js atau ApexCharts ke dalam proyek CodeIgniter 4 Anda. Pustaka-pustaka ini menyediakan alat yang ampuh dan fleksibel untuk membuat grafik interaktif dan responsif. Anda akan dibimbing langkah demi langkah dalam mengimplementasikan grafik-grafik ini, mulai dari persiapan data di sisi server hingga rendering di sisi klien.

Salah satu aspek menarik yang akan kita bahas adalah visualisasi data real-time. Dalam banyak sistem informasi, seperti pemantauan penjualan atau kehadiran, data terus berubah dan pengguna membutuhkan informasi terkini. Kita akan mempelajari bagaimana teknik AJAX polling dapat digunakan untuk secara periodik mengambil data terbaru dari server dan memperbarui grafik secara dinamis tanpa perlu memuat ulang seluruh halaman. Ini akan memberikan pengalaman pengguna yang lebih mulus dan informatif.

Sebagai penutup, bab ini akan diakhiri dengan praktik langsung membangun dashboard sederhana yang menampilkan grafik penjualan, kehadiran, atau pengunjung. Melalui studi kasus ini, Anda akan mengaplikasikan semua konsep yang telah dipelajari, mulai dari persiapan data, integrasi pustaka grafik, hingga implementasi visualisasi real-time. Dengan menguasai bab ini, Anda akan memiliki keterampilan esensial untuk menciptakan sistem informasi yang tidak hanya fungsional tetapi juga informatif dan mudah digunakan, memberikan nilai tambah yang signifikan bagi pengguna akhir.

A. KONSEP CHART PADA APLIKASI WEB

Visualisasi data melalui grafik atau *chart* merupakan komponen esensial dalam aplikasi web modern, terutama pada sistem informasi yang berorientasi pada analisis dan pelaporan. Grafik berfungsi sebagai jembatan antara data mentah yang kompleks dan pemahaman intuitif bagi pengguna. Dengan menyajikan data dalam bentuk visual, pola, tren, anomali, dan perbandingan dapat diidentifikasi dengan lebih cepat dan akurat dibandingkan dengan membaca tabel angka semata (Few, 2009).

1. Urgensi Visualisasi Data dalam Aplikasi Web

Dalam konteks aplikasi web, visualisasi data memiliki beberapa urgensi: a. **Peningkatan Pemahaman**: Data yang divisualisasikan lebih mudah dicerna dan dipahami oleh pengguna

dari berbagai latar belakang. Misalnya, grafik penjualan bulanan dapat langsung menunjukkan periode puncak atau penurunan tanpa perlu menganalisis setiap baris data. b. **Pengambilan Keputusan yang Lebih Baik**: Dengan pemahaman yang cepat, pengambil keputusan dapat membuat keputusan yang lebih tepat dan responsif terhadap kondisi yang digambarkan oleh data (Knaflic, 2015). c. **Efisiensi Komunikasi**: Grafik adalah alat komunikasi yang efektif. Sebuah grafik dapat menyampaikan informasi yang setara dengan beberapa paragraf teks atau tabel yang panjang. d. **Deteksi Pola dan Tren**: Visualisasi memungkinkan identifikasi pola musiman, tren pertumbuhan, atau penurunan yang mungkin terlewatkan dalam format tabular. e. **Interaktivitas**: Aplikasi web memungkinkan grafik yang interaktif, di mana pengguna dapat memfilter, memperbesar, atau mengeksplorasi data lebih lanjut, memberikan pengalaman analisis yang lebih mendalam.

2. Jenis-jenis Chart Dasar dan Kegunaannya

Pemilihan jenis grafik yang tepat sangat krusial untuk menyampaikan pesan data secara efektif. Beberapa jenis grafik dasar yang sering digunakan meliputi: a. **Grafik Batang (Bar Chart)**: Ideal untuk membandingkan kategori diskrit atau menunjukkan perubahan data dari waktu ke waktu. Contoh: perbandingan penjualan antar produk, jumlah pengunjung per hari. b. **Grafik Garis (Line Chart)**: Sangat cocok untuk menunjukkan tren data berkelanjutan sepanjang waktu atau kategori berurutan. Contoh: perkembangan harga saham, suhu rata-rata bulanan. c. **Grafik Lingkaran (Pie Chart)**: Digunakan untuk menunjukkan proporsi atau persentase dari keseluruhan. Contoh: pangsa pasar produk, distribusi demografi. Penting untuk tidak menggunakan terlalu banyak segmen agar mudah dibaca. d. **Grafik Area (Area Chart)**: Mirip dengan grafik garis, tetapi area di bawah garis diisi warna, cocok untuk menunjukkan volume atau akumulasi dari waktu ke waktu. e. **Grafik Sebar (Scatter Plot)**: Digunakan untuk menunjukkan hubungan atau korelasi antara dua variabel numerik.

B. INTEGRASI CHART.JS / APEXCHARTS

Untuk mengimplementasikan visualisasi data pada aplikasi web, kita memerlukan pustaka JavaScript yang menyediakan fungsionalitas untuk menggambar grafik. Dua pustaka populer yang sering digunakan adalah Chart.js dan ApexCharts. Keduanya menawarkan fleksibilitas dan fitur yang kaya untuk berbagai kebutuhan visualisasi.

1. Memilih Pustaka: Chart.js vs. ApexCharts

a. **Chart.js**: Merupakan pustaka *open-source* yang ringan dan mudah digunakan, ideal untuk grafik dasar seperti batang, garis, pie, dan donat. Chart.js menggunakan elemen <canvas> HTML5 untuk rendering dan memiliki komunitas yang besar. Keunggulannya terletak pada kesederhanaan dan performa yang baik untuk visualisasi standar.

b. **ApexCharts**: Pustaka yang lebih modern dan kaya fitur, menawarkan lebih banyak jenis grafik (termasuk grafik finansial, radar, dll.) dan opsi kustomisasi yang lebih luas. ApexCharts juga mendukung interaktivitas yang lebih canggih dan responsif secara *default*. Meskipun sedikit lebih kompleks dalam konfigurasi awal, ApexCharts memberikan hasil visual yang lebih profesional dan interaktif.

Dalam konteks CodeIgniter 4, integrasi kedua pustaka ini melibatkan langkah-langkah serupa:

2. Langkah-langkah Integrasi

a. **Pemasangan Pustaka**:

- **Melalui CDN (Content Delivery Network)**: Cara termudah adalah dengan menyertakan tautan CDN di bagian <head> atau sebelum tag </body> pada *layout* atau *view* CodeIgniter Anda.

```
<!-- Untuk Chart.js -->
 <script src=https://cdn.jsdelivr.net/npm/chart.js></script>
<!-- Untuk ApexCharts -->
<script src=https://cdn.jsdelivr.net/npm/apexcharts></script>
```

- **Melalui NPM/Yarn (untuk proyek yang lebih kompleks)**: Jika Anda menggunakan *build tools* seperti Webpack atau Vite, Anda dapat menginstal pustaka melalui NPM atau Yarn dan mengimpornya ke dalam *bundle* JavaScript Anda.

npm install chart.js # atau *npm install apexcharts* Kemudian, impor di file JavaScript Anda:

```
import Chart from 'chart.js'; // Untuk Chart.js
import ApexCharts from 'apexcharts'; // Untuk ApexCharts
```

b. **Persiapan Elemen HTML**: Anda perlu menyiapkan elemen HTML (biasanya untuk ApexCharts atau <canvas> untuk Chart.js) di mana grafik akan dirender.

```
<!-- Untuk Chart.js -->
<canvas id=myChart></canvas>
<!-- Untuk ApexCharts -->
<div id=chart></div>
```

c. **Pengambilan Data dari Server (CodeIgniter)**: Data untuk grafik biasanya berasal dari database. Di CodeIgniter 4, ini melibatkan:

- **Controller**: Membuat metode di *controller* yang mengambil data dari *model*.
- **Model**: Menggunakan *model* untuk berinteraksi dengan database dan mengambil data yang relevan.
- **Mengirim Data ke View**: Data kemudian dikirim ke *view* dalam format yang mudah diolah oleh JavaScript, seringkali dalam bentuk JSON.

Contoh di Controller (misal Dashboard.php):

```
<?php
namespace App\Controllers;
use App\Models\PenjualanModel; // Asumsi ada model Penjualan
class Dashboard extends BaseController {
    public function index() {
        $penjualanModel = new PenjualanModel();
        $dataPenjualan = $penjualanModel->getPenjualanBulanan();
        // Metode untuk mengambil data
        $data = [ 'title' => 'Dashboard Penjualan', 'penjualan' => json_encode($dataPenjualan)
        // Encode ke JSON
        ];
        return view('dashboard/index', $data);
    }
}
```

d. **Rendering Grafik di Sisi Klien (JavaScript)**: Di dalam *view* CodeIgniter, Anda akan menulis kode JavaScript untuk menginisialisasi grafik menggunakan data yang telah dikirim dari *controller*.

Contoh (menggunakan data dari variabel PHP $penjualan):

```
<script>
    const dataPenjualan = JSON.parse('<?= $penjualan ?>'); // Parse JSON dari PHP
    const labels = dataPenjualan.map(item => item.bulan);
    const values = dataPenjualan.map(item => item.total); // Contoh Chart.js
    const ctx = document.getElementById('myChart').getContext('2d');
    new Chart(ctx, { type: 'bar', data: { labels: labels, datasets: [
        { label: 'Total Penjualan', data: values, backgroundColor: 'rgba(75, 192, 192, 0.6)' }] },
        options: { responsive: true, scales: { y: { beginAtZero: true } } } }); // Contoh ApexCharts
    const options = { chart: { type: 'bar', height: 350 },
    series: [{ name: 'Total Penjualan', data: values }], xaxis: { categories: labels } };
    const chart = new ApexCharts(document.querySelector(#chart), options); chart.render();
</script>
```

C. MENAMPILKAN GRAFIK BATANG, GARIS, PIE

Setelah berhasil mengintegrasikan pustaka grafik, langkah selanjutnya adalah mengimplementasikan berbagai jenis grafik untuk merepresentasikan data yang berbeda.

1. Grafik Batang (Bar Chart)

Grafik batang sangat efektif untuk perbandingan antar kategori. **Contoh Kasus**: Menampilkan total penjualan per bulan. **Data Struktur**: Array objek dengan bulan dan total_penjualan.

```
// Data contoh
const dataPenjualanBulanan = [
    {bulan: 'Jan', total: 12000000}, { bulan: 'Feb', total: 15000000 }, {bulan: 'Mar', total: 10000000},
        bulan: 'Apr', total: 18000000}];
const labelsBar = dataPenjualanBulanan.map(item => item.bulan);
const valuesBar = dataPenjualanBulanan.map(item => item.total); // Chart.js
new Chart(document.getElementById('barChart').getContext('2d'), {type: 'bar',
    data: {labels: labelsBar,datasets: [{
            label: 'Penjualan Bulanan', data: valuesBar,backgroundColor: 'rgba(54, 162, 235, 0.6)'}]
    }});
// ApexCharts
new ApexCharts(document.querySelector(#barChartApex), {
    chart: {type: 'bar', height: 300 },
    series: [{name: 'Penjualan', data: valuesBar}],
    xaxis: { categories: labelsBar}
}).render();
```

2. Grafik Garis (Line Chart)

Grafik garis ideal untuk menunjukkan tren atau perubahan data sepanjang waktu. **Contoh Kasus**: Menampilkan jumlah pengunjung website harian selama seminggu. **Data Struktur**: Array objek dengan tanggal dan jumlah_pengunjung.

```
<script>
    // Data contoh
    const dataPengunjungHarian = [
        { tanggal: 'Sen',  jumlah: 150 }, {  tanggal: 'Sel',  jumlah: 200 }, { tanggal: 'Rab', jumlah: 180},
        { tanggal: 'Kam',  jumlah: 250 }, {  tanggal: 'Jum',  jumlah: 300 }, { tanggal: 'Sab', jumlah: 280},
        { tanggal: 'Min',  jumlah: 220 }];
    const labelsLine = dataPengunjungHarian.map(item => item.tanggal);
    const valuesLine = dataPengunjungHarian.map(item => item.jumlah);
    // Chart.js
    new Chart(document.getElementById('lineChart').getContext('2d'),
    { type: 'line', data: { labels: labelsLine, datasets: [
        { label: 'Pengunjung Harian', data: valuesLine, borderColor: 'rgba(255, 99, 132, 1)',
        fill: false }] } });
    // ApexCharts
    new ApexCharts(document.querySelector(#lineChartApex),
    { chart: { type: 'line', height: 300 }, series: [{ name: 'Pengunjung', data: valuesLine }],
    xaxis: { categories: labelsLine } }).render();
</script>
```

3. Grafik Lingkaran (Pie Chart)

Grafik lingkaran digunakan untuk menunjukkan proporsi bagian dari keseluruhan. **Contoh Kasus**: Distribusi penjualan berdasarkan kategori produk. **Data Struktur**: Array objek dengan kategori dan persentase_penjualan.

```
// Data contoh
const dataKategoriPenjualan = [
    { kategori: 'Elektronik', persentase: 40}, { kategori: 'Pakaian', persentase: 30 },
    { kategori: 'Makanan', persentase: 20}, { kategori: 'Lain-lain', persentase: 10 }];
const labelsPie = dataKategoriPenjualan.map(item => item.kategori);
const valuesPie = dataKategoriPenjualan.map(item => item.persentase);
const colorsPie = ['#FF6384', '#36A2EB', '#FFCE56', '#4BC0C0']; // Warna untuk setiap segmen
// Chart.js
new Chart(document.getElementById('pieChart').getContext('2d'),
{ type: 'pie', data: { labels: labelsPie, datasets: [
    { data: valuesPie, backgroundColor: colorsPie }] } });
// ApexCharts
new ApexCharts(document.querySelector(#pieChartApex), { chart: { type: 'pie', height: 300 },
series: valuesPie, labels: labelsPie, colors: colorsPie }).render();
```

D. GRAFIK REAL-TIME (AJAX POLLING)

Dalam banyak aplikasi, data terus berubah dan pengguna membutuhkan informasi terkini tanpa harus memuat ulang halaman secara manual. Grafik real-time memungkinkan pembaruan data secara dinamis, memberikan pengalaman pengguna yang lebih responsif dan informatif. Salah satu teknik umum untuk mencapai ini adalah **AJAX polling**.

1. Konsep AJAX Polling

AJAX polling adalah teknik di mana klien (browser) secara periodik mengirimkan permintaan AJAX ke server untuk memeriksa pembaruan data. Jika ada data baru, server akan merespons, dan klien akan memperbarui bagian tertentu dari halaman web (dalam hal ini, grafik) tanpa memuat ulang seluruh halaman (Kurose & Ross, 2017).

2. Implementasi AJAX Polling untuk Grafik Real-time

Langkah-langkah implementasi AJAX polling:

a. **Endpoint API di CodeIgniter**: Buat *controller* atau metode di CodeIgniter yang mengembalikan data terbaru dalam format JSON.

```
<?php
namespace App\Controllers;
use App\Models\PengunjungModel;
class Api extends BaseController {
    public function getPengunjungRealtime()
    {
        $pengunjungModel = new PengunjungModel();
        $data = $pengunjungModel->getLatestPengunjung(); // Ambil data terbaru
        return $this->response->setJSON($data);
    }
}
```

b. **Fungsi JavaScript untuk Mengambil dan Memperbarui Data**: Di sisi klien, gunakan *setInterval()* untuk memanggil fungsi AJAX secara berkala. Fungsi ini akan mengambil data dari *endpoint* API dan kemudian memperbarui objek grafik yang sudah ada.

```
let realTimeChart; // Variabel global untuk menyimpan instance chart
function fetchDataAndUpdateChart() {
    fetch('<?= base_url('api/getPengunjungRealtime') ?>')
    // URL endpoint API
    .then(response => response.json()) .then(data => { const labels = data.map(item => item.waktu);
    const values = data.map(item => item.jumlah);
    if (!realTimeChart) { // Inisialisasi chart jika belum ada
    const ctx = document.getElementById('realtimeChart').getContext('2d');
```

```
    realTimeChart = new Chart(ctx, { type: 'line', data: { labels: labels, datasets: [
    { label: 'Pengunjung Aktif', data: values, borderColor: 'rgba(75, 192, 192, 1)', fill: false }] },
    options: { responsive: true, scales: { y: { beginAtZero: true } } } });
} else { // Update data chart yang sudah ada
realTimeChart.data.labels = labels; realTimeChart.data.datasets[0].data = values;
realTimeChart.update(); // Perbarui tampilan chart
} })
.catch(error => console.error('Error fetching real-time data:', error)); }
// Panggil fungsi setiap 5 detik (5000 milidetik)
setInterval(fetchDataAndUpdateChart, 5000); // Panggil pertama kali saat halaman dimuat
fetchDataAndUpdateChart();
```

Untuk ApexCharts, proses pembaruan data sedikit berbeda, menggunakan metode *updateSeries()* dan *updateOptions()*.

```
let realTimeApexChart;
function fetchDataAndUpdateApexChart()
{
    fetch('<?= base_url('api/getPengunjungRealtime') ?>') .then(response => response.json())
    .then(data => { const labels = data.map(item => item.waktu);
    const values = data.map(item => item.jumlah);
    if (!realTimeApexChart) { const options = { chart: { type: 'line', height: 300 },
    series: [{ name: 'Pengunjung Aktif', data: values }], xaxis: { categories: labels } };
    realTimeApexChart = new ApexCharts(document.querySelector(#realtimeApexChart), options);
    realTimeApexChart.render();
    } else
    { realTimeApexChart.updateSeries([{ data: values }]);
    realTimeApexChart.updateOptions({ xaxis: { categories: labels } }); } })
    .catch(error => console.error('Error fetching real-time data:', error));
}
    setInterval(fetchDataAndUpdateApexChart, 5000); fetchDataAndUpdateApexChart();
```

E. PRAKTIK: DASHBOARD GRAFIK PENJUALAN/KEHADIRAN/PENGUNJUNG

Bagian ini akan memandu Anda dalam membangun dashboard sederhana yang menampilkan grafik untuk data penjualan, kehadiran, atau pengunjung, mengaplikasikan semua konsep yang telah dibahas.

1. Skenario Praktik: Dashboard Penjualan

Kita akan membuat dashboard yang menampilkan:

- Grafik batang: Total penjualan bulanan.
- Grafik garis: Tren penjualan harian.
- Grafik pie: Distribusi penjualan berdasarkan kategori produk.

2. Persiapan Database dan Model

Asumsikan Anda memiliki tabel penjualan dan produk dengan struktur sebagai berikut:

- penjualan: id, tanggal_transaksi, total_harga, id_produk, jumlah.
- produk: id, nama_produk, kategori.

Buat *model* di CodeIgniter 4 untuk mengambil data yang dibutuhkan:

```
<?php
// app/Models/PenjualanModel.php
namespace App\Models;
use CodeIgniter\Model;
class PenjualanModel extends Model
{
    protected $table = 'penjualan';
    protected $primaryKey = 'id';
    protected $allowedFields = ['tanggal_transaksi', 'total_harga', 'id_produk', 'jumlah'];
```

```
    public function getPenjualanBulanan()
    {
        return $this->db->table($this->table) ->select(DATE_FORMAT(tanggal_transaksi, '%Y-%m') as bulan,
        SUM(total_harga) as total) ->groupBy('bulan') ->orderBy('bulan', 'ASC') ->get() ->getResultArray();
    }
    public function getPenjualanHarian($limit = 7) {
        return $this->db->table($this->table) ->select(DATE_FORMAT(tanggal_transaksi, '%Y-%m-%d') as tanggal,
        SUM(total_harga) as total) ->groupBy('tanggal') ->orderBy('tanggal', 'DESC') ->limit($limit)
        ->get() ->getResultArray(); }
    public function getPenjualanByKategori() {
        return $this->db->table($this->table) ->select('p.kategori, SUM(t.total_harga) as total')
        ->join('produk p', 'p.id = penjualan.id_produk') ->groupBy('p.kategori') ->get()
        ->getResultArray(); }
}
```

3. Controller Dashboard

Buat *controller* Dashboard.php untuk menyiapkan data dan memuat *view*.

```
<?php
namespace App\Controllers;
use App\Models\PenjualanModel;
class Dashboard extends BaseController
{
    public function index()
    {
        $penjualanModel = new PenjualanModel();
        $data = [ 'title' => 'Dashboard Penjualan', 'penjualanBulanan'
        => json_encode($penjualanModel->getPenjualanBulanan()),
        'penjualanHarian' => json_encode($penjualanModel->getPenjualanHarian(30)), // 30 hari terakhir
        'penjualanKategori' => json_encode($penjualanModel->getPenjualanByKategori()) ];
        return view('dashboard/penjualan', $data);
    }
}
```

4. View Dashboard (dashboard/penjualan.php)

Buat *view* yang akan menampilkan grafik. Pastikan Anda telah menyertakan pustaka Chart.js atau ApexCharts di *layout* atau langsung di *view* ini.

```
<!-- app/Views/dashboard/penjualan.php -->
<?= $this->extend('layout/template') ?> <!-- Asumsi menggunakan layout template -->
<?= $this->section('content') ?> <div class=container mt-4>
    <h2>Dashboard Penjualan</h2>
    <div class=row>
        <div class=col-md-6>
            <div class=card mb-4>
                <div class=card-header>Penjualan Bulanan</div>
                <div class=card-body> <canvas id=chartPenjualanBulanan></canvas> </div>
            </div>
        </div>
        <div class=col-md-6>
            <div class=card mb-4>
                <div class=card-header>Tren Penjualan Harian</div>
                <div class=card-body> <canvas id=chartPenjualanHarian></canvas> </div>
            </div>
        </div>
    </div>
```

```
    <div class=row>
        <div class=col-md-6>
            <div class=card mb-4>
                <div class=card-header>Distribusi Penjualan per Kategori</div>
                <div class=card-body> <canvas id=chartPenjualanKategori></canvas> </div>
            </div>
        </div> <!-- Anda bisa menambahkan grafik real-time di sini jika diperlukan -->
        <div class=col-md-6>
            <div class=card mb-4>
                <div class=card-header>Grafik Real-time (Contoh)</div>
                <div class=card-body> <canvas id=realtimeChart></canvas> </div>
            </div>
        </div>
    </div>
</div> <?= $this->endSection() ?> <?= $this->section('scripts') ?>
<!-- Pastikan Chart.js sudah dimuat di layout atau di sini -->
<script src=https://cdn.jsdelivr.net/npm/chart.js></script>
<script>
    // Data Penjualan Bulanan
    const dataBulanan = JSON.parse('<?= $penjualanBulanan ?>');
    const labelsBulanan = dataBulanan.map(item => item.bulan);
    const valuesBulanan = dataBulanan.map(item => item.total);
    new Chart(document.getElementById('chartPenjualanBulanan').getContext('2d'),
    { type: 'bar', data: { labels: labelsBulanan, datasets: [{ label: 'Total Penjualan',
    data: valuesBulanan, backgroundColor: 'rgba(75, 192, 192, 0.6)' }] },
    options: { responsive: true, scales: { y: { beginAtZero: true } } } });
    // Data Penjualan Harian
    const dataHarian = JSON.parse('<?= $penjualanHarian ?>');
    const labelsHarian = dataHarian.map(item => item.tanggal);
    const valuesHarian = dataHarian.map(item => item.total);
    new Chart(document.getElementById('chartPenjualanHarian').getContext('2d'),
    { type: 'line', data: { labels: labelsHarian, datasets: [{ label: 'Penjualan Harian',
    data: valuesHarian, borderColor: 'rgba(255, 99, 132, 1)', fill: false }] },
    options: { responsive: true, scales: { y: { beginAtZero: true } } } });
    // Data Penjualan per Kategori
    const dataKategori = JSON.parse('<?= $penjualanKategori ?>');
    const labelsKategori = dataKategori.map(item => item.kategori);
    const valuesKategori = dataKategori.map(item => item.total);
    const colorsKategori = ['#FF6384', '#36A2EB', '#FFCE56', '#4BC0C0', '#9966FF'];
    new Chart(document.getElementById('chartPenjualanKategori').getContext('2d'),
    { type: 'pie', data: { labels: labelsKategori, datasets: [{ data: valuesKategori,
    backgroundColor: colorsKategori }] }, options: { responsive: true } });
    // Contoh Grafik Real-time (menggunakan AJAX polling)
    let realTimeChartInstance;
    function updateRealtimeChart() {
        fetch('<?= base_url('api/getPengunjungRealtime') ?>') // Asumsi ada endpoint API untuk data real-time
        .then(response => response.json()) .then(data => { const labels = data.map(item => item.waktu);
        const values = data.map(item => item.jumlah);
        if (!realTimeChartInstance) { const ctx = document.getElementById('realtimeChart').getContext('2d');
        realTimeChartInstance = new Chart(ctx, { type: 'line', data: { labels: labels,
        datasets: [{ label: 'Pengunjung Aktif', data: values, borderColor: 'rgba(153, 102, 255, 1)',
        fill: false }] }, options: { responsive: true, scales: { y: { beginAtZero: true } } } }); }
        else { realTimeChartInstance.data.labels = labels;
        realTimeChartInstance.data.datasets[0].data = values;
        realTimeChartInstance.update(); } })
        .catch(error => console.error('Error fetching real-time data:', error)); }
        setInterval(updateRealtimeChart, 5000); // Update setiap 5 detik
        updateRealtimeChart(); // Panggil pertama kali
</script> <?= $this->endSection() ?>
```

Dengan mengikuti langkah-langkah ini, Anda akan berhasil membangun dashboard yang menampilkan berbagai jenis grafik, memberikan visualisasi data yang informatif dan dinamis untuk sistem informasi Anda.

F. RANGKUMAN

Visualisasi data melalui *chart* adalah elemen krusial dalam aplikasi web modern, berfungsi sebagai jembatan antara data kompleks dan pemahaman intuitif pengguna. Kemampuan untuk menyajikan data secara visual memungkinkan identifikasi pola, tren, dan anomali dengan lebih cepat dan akurat dibandingkan dengan format tabular.

1. **Urgensi Visualisasi Data**: Visualisasi data sangat penting untuk meningkatkan pemahaman pengguna, mendukung pengambilan keputusan yang lebih baik dan cepat, serta meningkatkan efisiensi komunikasi. Grafik juga memfasilitasi deteksi pola dan tren yang mungkin terlewatkan dalam data mentah, serta memungkinkan interaktivitas yang mendalam bagi pengguna.
2. **Jenis-jenis *Chart* Dasar**: Pemilihan jenis *chart* yang tepat sangat penting. *Bar chart* ideal untuk perbandingan kategori diskrit, *line chart* untuk tren data berkelanjutan sepanjang waktu, *pie chart* untuk proporsi bagian dari keseluruhan, *area chart* untuk volume atau akumulasi, dan *scatter plot* untuk menunjukkan korelasi antara dua variabel numerik.
3. **Integrasi Pustaka *Chart***: Untuk mengimplementasikan visualisasi, pustaka JavaScript seperti Chart.js dan ApexCharts sangat dibutuhkan. Chart.js dikenal ringan dan mudah digunakan untuk grafik dasar, sementara ApexCharts menawarkan fitur lebih kaya, jenis grafik yang lebih beragam, dan interaktivitas canggih. Integrasi melibatkan pemasangan pustaka (via CDN atau NPM), persiapan elemen HTML, pengambilan data dari server CodeIgniter (seringkali dalam format JSON), dan rendering grafik di sisi klien menggunakan JavaScript.
4. **Menampilkan Berbagai Jenis Grafik**: Setelah integrasi, berbagai jenis grafik dapat diimplementasikan. Grafik batang efektif untuk perbandingan penjualan bulanan, grafik garis ideal untuk tren pengunjung harian, dan grafik lingkaran cocok untuk distribusi penjualan berdasarkan kategori produk. Setiap jenis grafik memerlukan konfigurasi data dan opsi yang spesifik sesuai pustaka yang digunakan.
5. **Grafik Real-time (AJAX Polling)**: Untuk data yang terus berubah, grafik real-time sangat penting. Teknik AJAX polling memungkinkan klien secara periodik meminta data terbaru dari server tanpa memuat ulang halaman. Data yang diterima kemudian digunakan untuk memperbarui objek grafik yang sudah ada, memberikan pengalaman pengguna yang dinamis dan informatif.
6. **Praktik Dashboard Grafik**: Konsep-konsep ini dapat diaplikasikan untuk membangun dashboard interaktif, misalnya dashboard penjualan. Ini melibatkan persiapan database dan model di CodeIgniter untuk mengambil data penjualan bulanan, harian, dan berdasarkan kategori. Kemudian, *controller* menyiapkan data tersebut dan mengirimkannya ke *view*, di mana JavaScript akan merender berbagai grafik menggunakan data yang telah di-*encode* dari PHP.

BAB 12
AJAX, DATATABLES, DAN INTERAKSI DINAMIS

Dalam pengembangan aplikasi web modern, interaksi pengguna yang cepat dan responsif menjadi kunci utama untuk menciptakan pengalaman yang memuaskan. Pengguna saat ini mengharapkan aplikasi yang dapat memuat data, memperbarui informasi, atau melakukan tindakan lain tanpa harus menunggu seluruh halaman dimuat ulang. Kebutuhan inilah yang mendorong penggunaan teknologi Asynchronous JavaScript and XML (AJAX) dan Fetch API secara luas. AJAX memungkinkan aplikasi web untuk berkomunikasi dengan server di latar belakang, mengambil atau mengirim data, dan memperbarui bagian tertentu dari halaman tanpa mengganggu alur kerja pengguna.

Bab 12 ini akan membawa Anda lebih jauh ke dalam dunia interaksi dinamis pada aplikasi web berbasis CodeIgniter 4. Kita akan menjelajahi bagaimana AJAX, khususnya melalui Fetch API, dapat digunakan untuk membangun antarmuka yang lebih responsif. Anda akan belajar cara mengirim permintaan ke server dan memproses responsnya secara asinkron, membuka pintu bagi berbagai kemungkinan interaksi yang sebelumnya sulit dicapai dengan pendekatan tradisional.

Salah satu tantangan umum dalam aplikasi web adalah menampilkan dan mengelola sejumlah besar data dalam bentuk tabel. Menggulir melalui ribuan baris data atau menunggu halaman dimuat ulang setiap kali ada perubahan filter atau pengurutan dapat sangat menghambat produktivitas. Di sinilah peran Datatables menjadi sangat krusial. Datatables adalah library JavaScript yang sangat populer dan kuat untuk meningkatkan fungsionalitas tabel HTML, menambahkan fitur seperti pencarian instan, pengurutan, paginasi, dan ekspor data. Lebih penting lagi, kita akan fokus pada implementasi Datatables dengan *server-side processing*, yang berarti semua operasi seperti pencarian, pengurutan, dan paginasi ditangani oleh server. Pendekatan ini sangat efisien untuk dataset yang besar, karena hanya data yang relevan yang dikirimkan ke browser, mengurangi beban pada klien dan meningkatkan kinerja aplikasi secara keseluruhan.

Selain itu, bab ini juga akan membahas teknik untuk memperbarui data secara *inline* atau melalui modal edit tanpa perlu memuat ulang halaman. Kemampuan ini sangat penting untuk modul-modul seperti manajemen data master atau transaksi, di mana pengguna seringkali perlu melakukan perubahan kecil pada beberapa entri data. Dengan mengombinasikan AJAX, Fetch API, dan Datatables, Anda akan memiliki perangkat yang lengkap untuk membangun modul tabel dinamis yang tidak hanya fungsional tetapi juga memberikan pengalaman pengguna yang superior. Melalui serangkaian praktik dan contoh, Anda akan diajak untuk mengimplementasikan konsep-konsep ini secara langsung dalam proyek CodeIgniter 4 Anda, mempersiapkan Anda untuk membangun aplikasi web yang lebih interaktif dan efisien.

A. AJAX DAN FETCH API DI CODEIGNITER

Interaksi dinamis pada aplikasi web modern sangat bergantung pada kemampuan untuk berkomunikasi dengan server tanpa harus memuat ulang seluruh halaman. Konsep ini dikenal sebagai Asynchronous JavaScript and XML (AJAX). Meskipun namanya mengandung XML, AJAX saat ini lebih sering menggunakan JSON (JavaScript Object Notation) sebagai format pertukaran data karena sifatnya yang ringan dan mudah diurai oleh JavaScript. AJAX memungkinkan aplikasi web untuk mengirim dan menerima data dari server di latar belakang,

kemudian memperbarui bagian tertentu dari halaman secara selektif, sehingga meningkatkan responsivitas dan pengalaman pengguna (UX) (Kurose & Ross, 2017).

Di era modern pengembangan web, XMLHttpRequest (XHR) yang merupakan fondasi AJAX tradisional, telah banyak digantikan oleh Fetch API. Fetch API menawarkan antarmuka yang lebih kuat dan fleksibel untuk melakukan permintaan jaringan, berbasis pada Promise yang memudahkan penanganan operasi asinkron dan menghindari *callback hell* (MDN Web Docs, n.d.). CodeIgniter 4, sebagai *framework* PHP yang modern, sangat mendukung integrasi dengan teknologi *front-end* seperti Fetch API untuk membangun aplikasi yang responsif.

1. Konsep Dasar AJAX dan Fetch API

AJAX bekerja dengan mengirimkan permintaan HTTP (GET, POST, PUT, DELETE) ke server dari *browser* menggunakan JavaScript. Server kemudian memproses permintaan tersebut, mengambil atau memanipulasi data, dan mengirimkan respons kembali ke *browser*. Respons ini bisa berupa data dalam format JSON, XML, atau bahkan HTML parsial. Setelah respons diterima, JavaScript akan memperbarui DOM (Document Object Model) halaman web sesuai dengan data yang diterima, tanpa perlu memuat ulang seluruh halaman.

Fetch API adalah standar modern untuk melakukan permintaan jaringan. Sintaks dasarnya adalah sebagai berikut:

```
fetch(url, options).then(response => {
    if (!response.ok) {
        throw new Error('Network response was not ok');}
        return response.json(); // atau .text(), .blob(), dll. })
        then(data => { console.log(data); // Lakukan sesuatu dengan data
    })
.catch(error => { console.error('There was a problem with the fetch operation:', error); });
```

Pada CodeIgniter 4, *backend* akan berperan sebagai penyedia *endpoint* API yang akan diakses oleh Fetch API. *Controller* di CodeIgniter 4 dapat mengembalikan data dalam format JSON dengan mudah menggunakan metode response->setJSON() atau response->setBody() dengan *header* Content-Type: application/json.

2. Implementasi Fetch API di CodeIgniter 4

Untuk mengimplementasikan Fetch API di CodeIgniter 4, kita perlu menyiapkan dua bagian utama:

a. Controller sebagai Endpoint API

Controller akan bertanggung jawab untuk menerima permintaan dari *client*, memprosesnya (misalnya, mengambil data dari *database* melalui *Model*), dan mengembalikan respons dalam format JSON.

Contoh: Membuat *controller* ApiData untuk mengambil daftar pengguna.

```
<?php
namespace App\Controllers;
use CodeIgniter\API\ResponseTrait; // Untuk kemudahan respons API
use App\Models\UserModel; // Asumsikan ada UserModel
class ApiData extends BaseController
{
    use ResponseTrait;
    public function getUsers() {
        $userModel = new UserModel();
        $users = $userModel->findAll(); // Ambil semua data user
        // Mengembalikan data dalam format JSON
        return $this->respond($users); // Menggunakan ResponseTrait untuk respons JSON
    }
```

```
    public function saveUser() {
        $userModel = new UserModel();
        $data = $this->request->getJSON(true); // Mengambil data JSON dari body request
        if ($userModel->insert($data)) {
            return $this->respondCreated([
                'message' => 'User created successfully',
                'id' => $userModel->getInsertID()]);
        } else { return $this->failValidationErrors($userModel->errors()); }
    }
}
```

b. JavaScript (Fetch API) di View

Di sisi *client* (dalam *file* JavaScript yang dimuat di *view*), kita akan menggunakan Fetch API untuk mengirim permintaan ke *endpoint* yang telah dibuat di *controller*.

Contoh: Mengambil data pengguna dan menampilkannya.

```
<!-- Dalam file view Anda, misalnya app/Views/users/index.php -->
<div id=user-list></div>
<script>
    document.addEventListener('DOMContentLoaded', function() {
        fetch('/api/users') // URL ke controller
        ApiData::getUsers() .then(response => {
            if (!response.ok) { throw new Error('Gagal mengambil data pengguna.'); }
            return response.json(); })
            .then(users => { const userListDiv = document.getElementById('user-list');
            let html = '<ul>'; users.forEach(user => { html += `<li>${user.nama} (${user.email})</li>`; });
            html += '</ul>'; userListDiv.innerHTML = html; }) .catch(error => {
                console.error('Error:', error);
                document.getElementById('user-list').innerHTML = `<p style=color: red;>${error.message}</p>`;
    }); });
</script>
```

Dengan pendekatan ini, aplikasi dapat mengambil dan menampilkan data tanpa memuat ulang halaman, memberikan pengalaman pengguna yang lebih mulus dan responsif.

B. INTEGRASI DATATABLES SERVER-SIDE

Datatables adalah *plugin* jQuery yang sangat populer untuk menambahkan fitur interaktif pada tabel HTML, seperti pencarian, pengurutan, paginasi, dan ekspor data (Datatables, n.d.). Untuk dataset yang kecil, Datatables dapat bekerja dengan data yang sudah ada di DOM (*client-side processing*). Namun, untuk aplikasi dengan volume data yang besar (ribuan hingga jutaan baris), *client-side processing* akan sangat membebani *browser* dan menyebabkan kinerja yang lambat. Di sinilah *server-side processing* menjadi solusi yang efisien.

1. Konsep Server-Side Processing

Dalam *server-side processing*, semua operasi seperti pencarian, pengurutan, dan paginasi tidak dilakukan oleh *browser*, melainkan oleh server. Ketika pengguna melakukan tindakan (misalnya, mengubah halaman, mencari, atau mengurutkan kolom), Datatables akan mengirimkan permintaan AJAX ke server. Server kemudian memproses permintaan tersebut, mengambil hanya data yang relevan dari *database*, dan mengirimkannya kembali ke Datatables dalam format JSON. Datatables kemudian akan merender data yang diterima ke dalam tabel. Pendekatan ini mengurangi beban pada *client* dan memastikan kinerja yang optimal bahkan dengan dataset yang sangat besar (Datatables, n.d.).

2. Persiapan Datatables di CodeIgniter 4

a. Memuat Library Datatables

Pertama, pastikan Anda telah memuat *library* jQuery dan Datatables di *view* Anda. Ini biasanya dilakukan di bagian <head> atau sebelum tag </body> penutup.

```
<!-- Di layout atau view Anda -->
<link rel=stylesheet type=text/css href=https://cdn.datatables.net/1.11.5/css/jquery.dataTables.min.css>
<script src=https://code.jquery.com/jquery-3.6.0.min.js></script>
<script src=https://cdn.datatables.net/1.11.5/js/jquery.dataTables.min.js></script>
```

b. Controller untuk Datatables Server-Side

Kita perlu membuat *controller* di CodeIgniter 4 yang akan menangani permintaan AJAX dari Datatables. *Controller* ini akan menerima parameter dari Datatables (seperti *start*, *length*, *search*, *order*), memprosesnya, dan mengembalikan data dalam format JSON yang spesifik.

Contoh: *Controller* UserData untuk Datatables.

```
<?php
namespace App\Controllers;
use CodeIgniter\API\ResponseTrait;
use App\Models\UserModel;
class UserData extends BaseController
{
    use ResponseTrait;
    public function index() {
        return view('users/datatables_view'); // View yang akan menampilkan tabel
    }
```

```
    public function getDatatables() {
        $userModel = new UserModel();
        $request = service('request'); // Parameter dari Datatables
        $draw = $request->getPost('draw'); $start = $request->getPost('start');
        $length = $request->getPost('length'); $search = $request->getPost('search')['value'];
        $order = $request->getPost('order'); $columns = $request->getPost('columns');
        // Query dasar
        $builder = $userModel->builder();
        // Pencarian
        if (!empty($search)) {
            $builder->groupStart() ->orLike('nama', $search) ->orLike('email', $search) ->groupEnd(); }
        // Pengurutan
        if (!empty($order)) {
            $columnName = $columns[$order[0]['column']]['data']; $dir = $order[0]['dir'];
            $builder->orderBy($columnName, $dir); }
    // Total data tanpa filter
    $totalRecords = $builder->countAllResults(false); // false agar query tidak direset
    // Total data setelah filter
    $filteredRecords = $builder->countAllResults(false);
    // Paginasi
    $builder->limit($length, $start);
    // Ambil data
    $data = $builder->get()->getResultArray();
    $response = [ 'draw' => intval($draw), 'recordsTotal' => intval($totalRecords),
    'recordsFiltered' => intval($filteredRecords), 'data' => $data, ];
    return $this->respond($response); }
}
```

c. Konfigurasi Datatables di View

Di *view*, kita akan menginisialisasi Datatables dan mengkonfigurasinya untuk menggunakan *server-side processing*.

```
<!-- app/Views/users/datatables_view.php -->
<table id=myTable class=display style=width:100%>
    <thead>
```

```
        <tr>
            <th>ID</th>
            <th>Nama</th>
            <th>Email</th>
            <th>Aksi</th>
        </tr>
    </thead>
    <tbody> <!-- Data akan dimuat oleh Datatables --> </tbody>
</table>
<script>
    $(document).ready(function() {
                $('#myTable').DataTable({ processing: true, serverSide: true,
                    ajax: {
                        url: '<?= site_url('user/datatables') ?>', // URL ke controller
                        UserData::getDatatables() type: 'POST' },
                        columns: [ { data: 'id' }, { data: 'nama' }, { data: 'email' },
                        { data: null, render: function(data, type, row) {
                        return `<button class=btn btn-sm btn-warning edit-btn data-id=${row.id}>Edit</button>
                        <button class=btn btn-sm btn-danger delete-btn data-id=${row.id}>Delete</button>`;
    } } ] }); });
</script>
```

Pastikan *routing* untuk *UserData::getDatatables()* telah diatur, misalnya:

```
$routes->get('/users', 'UserData::index');
$routes->post('/user/datatables', 'UserData::getDatatables');
```

Dengan konfigurasi ini, Datatables akan secara otomatis mengirim permintaan AJAX ke *endpoint* /user/datatables setiap kali ada interaksi yang memerlukan pembaruan data (pencarian, pengurutan, paginasi), dan server akan merespons dengan data yang sesuai.

C. UPDATE DATA TANPA RELOAD (MODAL EDIT)

Salah satu keuntungan utama dari AJAX adalah kemampuan untuk memperbarui data di *database* dan merefleksikan perubahan tersebut di antarmuka pengguna tanpa perlu memuat ulang seluruh halaman. Teknik ini sangat meningkatkan pengalaman pengguna, terutama untuk operasi CRUD (Create, Read, Update, Delete) yang sering dilakukan. Penggunaan *modal* (dialog pop-up) untuk formulir edit adalah pola desain yang umum dan efektif dalam skenario ini.

1. Konsep Modal Edit dengan AJAX

Ketika pengguna mengklik tombol Edit pada sebuah baris tabel, alih-alih mengarahkan ke halaman edit terpisah, sebuah *modal* akan muncul. *Modal* ini berisi formulir yang telah diisi dengan data dari baris yang dipilih. Data ini diambil dari server melalui permintaan AJAX. Setelah pengguna melakukan perubahan dan menekan tombol Simpan, data yang diperbarui akan dikirim kembali ke server melalui permintaan AJAX lainnya. Server akan memproses pembaruan di *database*, dan setelah berhasil, *modal* akan ditutup, dan tabel Datatables akan diperbarui (biasanya dengan memuat ulang data Datatables secara parsial) tanpa *page reload*.

2. Implementasi Modal Edit di CodeIgniter 4

a. Struktur HTML untuk Modal

Tambahkan struktur HTML untuk *modal* di *view* Anda. Anda bisa menggunakan *framework* CSS seperti Bootstrap untuk *modal* yang responsif.

```
<!-- app/Views/users/datatables_view.php (lanjutan) --> <!-- Modal Edit User -->
<div class=modal fade id=editUserModal tabindex=-1 aria-labelledby=editUserModalLabel aria-hidden=true>
    <div class=modal-dialog>
        <div class=modal-content>
            <div class=modal-header>
                <h5 class=modal-title id=editUserModalLabel>Edit Pengguna</h5>
                <button type=button class=btn-close data-bs-dismiss=modal aria-label=Close></button>
            </div>
```

```
            <div class=modal-body>
                <form id=editUserForm> <input type=hidden id=editUserId name=id>
                    <div class=mb-3> <label for=editNama class=form-label>Nama</label>
                    <input type=text class=form-control id=editNama name=nama required> </div>
                    <div class=mb-3> <label for=editEmail class=form-label>Email</label>
                    <input type=email class=form-control id=editEmail name=email required> </div>
                    <div id=editFormErrors class=alert alert-danger d-none></div>
                </form>
            </div>
            <div class=modal-footer>
                <button type=button class=btn btn-secondary data-bs-dismiss=modal>Tutup</button>
                <button type=button class=btn btn-primary id=saveEditUserBtn>Simpan Perubahan</button>
            </div>
        </div>
    </div>
</div>
```

b. JavaScript untuk Mengisi dan Mengirim Data Modal

Kita akan menggunakan JavaScript untuk:

1. Menangkap klik tombol Edit.
2. Mengambil data pengguna berdasarkan ID melalui AJAX.
3. Mengisi formulir di *modal* dengan data yang diterima.
4. Menangkap *submit* formulir di *modal*.
5. Mengirim data yang diperbarui ke server melalui AJAX.
6. Memperbarui Datatables setelah berhasil.

```
// Lanjutan dari script Datatables di app/Views/users/datatables_view.php
$(document).ready(function() { var table = $('#myTable').DataTable({ /* ... konfigurasi Datatables ... */ });
 // Event listener untuk tombol Edit
$('#myTable tbody').on('click', '.edit-btn', function() { var userId = $(this).data('id');
    $('#editFormErrors').addClass('d-none').html(''); // Sembunyikan error sebelumnya
    // Ambil data pengguna dari server
    fetch(`/api/users/${userId}`) // Endpoint untuk mengambil detail user
    .then(response => response.json()) .then(data => {
        $('#editUserId').val(data.id); $('#editNama').val(data.nama);
        $('#editEmail').val(data.email); $('#editUserModal').modal('show'); // Tampilkan modal
        })
    .catch(error => console.error('Error fetching user data:', error)); });
    // Event listener untuk tombol Simpan Perubahan di modal
    $('#saveEditUserBtn').on('click', function() { var userId = $('#editUserId').val();
        var formData = { nama: $('#editNama').val(), email: $('#editEmail').val() };
        fetch(`/api/users/${userId}`, { // Endpoint untuk update user method: 'PUT', // Atau POST dengan
        _method=PUT headers: { 'Content-Type': 'application/json', 'X-Requested-With': 'XMLHttpRequest'
            // Penting untuk CI4
            }, body: JSON.stringify(formData) }) .then(response => {
                if (!response.ok) { return response.json().then(err => { throw err; }); }
                return response.json(); }) .then(data => { alert(data.message);
                $('#editUserModal').modal('hide'); // Sembunyikan modal
                table.ajax.reload(null, false); // Muat ulang Datatables tanpa mereset posisi paginasi
                }) .catch(error => { console.error('Error updating user:', error);
            let errorMessage = 'Terjadi kesalahan saat menyimpan data.'; if (error.messages) { // Jika ada pe
            errorMessage = Object.values(error.messages).join('<br>'); }
            $('#editFormErrors').removeClass('d-none').html(errorMessage); }); }); });
```

c. Controller untuk Mengambil dan Memperbarui Data Pengguna

Kita perlu menambahkan metode di *controller* ApiData (atau *controller* terpisah) untuk mengambil detail pengguna dan memperbarui data.

```
<?php
namespace App\Controllers;
use CodeIgniter\API\ResponseTrait;
use App\Models\UserModel;
```

```php
class ApiData extends BaseController
{
    use ResponseTrait; // ... (metode getUsers, saveUser sebelumnya) ...
    public function getUser($id = null) {
        $userModel = new UserModel();
        $user = $userModel->find($id);
        if ($user) {
            return $this->respond($user);
        } else {
            return $this->failNotFound('User not found.');
        } }
    public function updateUser($id = null) {
        $userModel = new UserModel();
        $data = $this->request->getJSON(true);
        // Validasi data (opsional, tapi sangat disarankan)
        if (!$this->validate([
            'nama' => 'required|min_length[3]',
            'email' => 'required|valid_email|is_unique[users.email,id,{id}]' ],
            ['id' => $id])) { return $this->failValidationErrors($this->validator->getErrors()); }
            if ($userModel->update($id, $data)) {
                return $this->respond(['message' => 'User updated successfully']);
            } else { return $this->fail('Failed to update user.'); }
        }
}
```

Tambahkan *routing* untuk metode ini:

```php
$routes->get('/api/users/(:num)', 'ApiData::getUser/$1');
$routes->put('/api/users/(:num)', 'ApiData::updateUser/$1'); // Menggunakan PUT untuk update
```

Dengan implementasi ini, pengguna dapat mengedit data langsung dari tabel tanpa *page reload*, memberikan pengalaman yang lebih cepat dan interaktif.

D. PRAKTIK: MODUL TABEL DINAMIS DENGAN DATATABLES

Bagian ini akan memandu Anda melalui langkah-langkah praktis untuk membangun modul tabel dinamis lengkap menggunakan Datatables dengan *server-side processing*, AJAX, dan fitur *modal edit* di CodeIgniter 4. Kita akan menggunakan contoh manajemen data produk.

1. Persiapan Database dan Model

Asumsikan Anda memiliki tabel products di *database* dengan kolom id, nama_produk, harga, dan stok.

```sql
CREATE TABLE products (
id INT AUTO_INCREMENT PRIMARY KEY,
nama_produk VARCHAR(255) NOT NULL,
harga DECIMAL(10, 2) NOT NULL,
stok INT NOT NULL DEFAULT 0,
created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP );
```

Buat *Model* ProductModel di app/Models/ProductModel.php:

```php
<?php
namespace App\Models;
use CodeIgniter\Model;
class ProductModel extends Model
{
    protected $table = 'products';   protected $primaryKey = 'id'; protected $useAutoIncrement = true;
    protected $returnType = 'array'; protected $useSoftDeletes = false;
    protected $allowedFields = ['nama_produk', 'harga', 'stok']; protected $useTimestamps = true;
    protected $createdField = 'created_at'; protected $updatedField = 'updated_at';
    protected $deletedField = 'deleted_at'; protected $validationRules = [];
    protected $validationMessages = []; protected $skipValidation = false;
}
```

2. Controller Product

Buat *controller* Product di app/Controllers/Product.php untuk menangani tampilan tabel dan *endpoint* API Datatables.

```php
<?php
namespace App\Controllers;
use CodeIgniter\API\ResponseTrait;
use App\Models\ProductModel;
class Product extends BaseController {
    use ResponseTrait; public function index() {
        return view('products/index'); } // View utama untuk tabel produk
    public function getDatatables() {
        $productModel = new ProductModel(); $request = service('request');
        $draw = $request->getPost('draw'); $start = $request->getPost('start');
        $length = $request->getPost('length'); $search = $request->getPost('search')['value'];
        $order = $request->getPost('order'); $columns = $request->getPost('columns');
        $builder = $productModel->builder();
        if (!empty($search)) {
            $builder->groupStart() ->orLike('nama_produk', $search) ->orLike('harga', $search)
            ->orLike('stok', $search) ->groupEnd(); }
        if (!empty($order)) {
            $columnName = $columns[$order[0]['column']]['data']; $dir = $order[0]['dir'];
            $builder->orderBy($columnName, $dir); }
        $totalRecords = $builder->countAllResults(false);
        $filteredRecords = $builder->countAllResults(false); $builder->limit($length, $start);
        $data = $builder->get()->getResultArray();
        $response = [ 'draw' => intval($draw), 'recordsTotal' => intval($totalRecords),
        'recordsFiltered' => intval($filteredRecords), 'data' => $data, ];
        return $this->respond($response); }
    public function getProduct($id = null) {
        $productModel = new ProductModel(); $product = $productModel->find($id);
        if ($product) { return $this->respond($product); } else {
            return $this->failNotFound('Product not found.'); } }
    public function updateProduct($id = null) { $productModel = new ProductModel();
        $data = $this->request->getJSON(true);
        if (!$this->validate([ 'nama_produk' => 'required|min_length[3]',
        'harga' => 'required|numeric', 'stok' => 'required|integer' ])) {
            return $this->failValidationErrors($this->validator->getErrors()); }
        if ($productModel->update($id, $data)) {
            return $this->respond(['message' => 'Product updated successfully']); }
            else { return $this->fail('Failed to update product.'); } }
    public function deleteProduct($id = null) {
        $productModel = new ProductModel(); if ($productModel->delete($id)) {
            return $this->respondDeleted(['message' => 'Product deleted successfully']); } else {
                return $this->fail('Failed to delete product.'); }
    }
}
```

3. View products/index.php

Buat *file* app/Views/products/index.php yang berisi tabel Datatables dan *modal* edit.

```html
<!DOCTYPE html>
<html lang=en>

<head>
    <meta charset=UTF-8>
    <meta name=viewport content=width=device-width, initial-scale=1.0>
    <title>Manajemen Produk</title>
    <link href=https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css rel=stylesheet>
    <link rel=stylesheet type=text/css href=https://cdn.datatables.net/1.11.5/css/jquery.dataTables.min.css>
</head>
```

```
<body>
    <div class=container mt-5>
        <h2>Manajemen Data Produk</h2>
        <table id=productTable class=display style=width:100%>
            <thead>
                <tr>
                    <th>ID</th>
                    <th>Nama Produk</th>
                    <th>Harga</th>
                    <th>Stok</th>
                    <th>Aksi</th>
                </tr>
            </thead>
            <tbody> <!-- Data akan dimuat oleh Datatables --> </tbody>
        </table>
    </div> <!-- Modal Edit Produk -->
```

```
    <div class=modal fade id=editProductModal tabindex=-1 aria-labelledby=editProductModalLabel aria-hidden=true>
        <div class=modal-dialog>
            <div class=modal-content>
            <div class=modal-header>
            <h5 class=modal-title id=editProductModalLabel>Edit Produk</h5>
            <button type=button class=btn-close data-bs-dismiss=modal aria-label=Close></button>
            </div>
            <div class=modal-body>
            <form id=editProductForm> <input type=hidden id=editProductId name=id>
                <div class=mb-3> <label for=editNamaProduk class=form-label>Nama Produk</label>
                <input type=text class=form-control id=editNamaProduk name=nama_produk required> </div>
                <div class=mb-3> <label for=editHarga class=form-label>Harga</label>
                <input type=number step=0.01 class=form-control id=editHarga name=harga required> </div>
                <div class=mb-3> <label for=editStok class=form-label>Stok</label>
                <input type=number class=form-control id=editStok name=stok required> </div>
                <div id=editFormErrors class=alert alert-danger d-none></div>
            </form>
            </div>
            <div class=modal-footer> <button type=button class=btn btn-secondary data-bs-dismiss=modal>Tutup</button>
            </div>
        </div>
    </div>
```

```
<script src=https://code.jquery.com/jquery-3.6.0.min.js></script>
<script src=https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js></script>
<script src=https://cdn.datatables.net/1.11.5/js/jquery.dataTables.min.js></script>
<script>
    $(document).ready(function() {
                var table = $('#productTable').DataTable({
                    processing: true,serverSide: true,
                    ajax: {url: '<?= site_url('product/datatables') ?>',type: 'POST'},
                    columns: [{data: 'id' }, {data: 'nama_produk'}, {data: 'harga'}, {data: 'stok'}, {
                    data: null, render: function(data, type, row) {
                    return `<button class=btn btn-sm btn-warning edit-btn data-id=${row.id}
                    data-bs-toggle=modal data-bs-target=#editProductModal>Edit</button>
                    <button class=btn btn-sm btn-danger delete-btn data-id=${row.id}>Delete</button>`;}}]
                });
    // Event listener untuk tombol Edit
    $('#productTable tbody').on('click', '.edit-btn', function() { var productId = $(this).data('id');
        $('#editFormErrors').addClass('d-none').html('');
        fetch(`<?= site_url('api/products/') ?>${productId}`) .then(response => response.json())
        .then(data => { $('#editProductId').val(data.id); $('#editNamaProduk').val(data.nama_produk);
        $('#editHarga').val(data.harga); $('#editStok').val(data.stok); })
        .catch(error => console.error('Error fetching product data:', error)); });
```

```
    // Event listener untuk tombol Simpan Perubahan di modal
    $('#saveEditProductBtn').on('click', function() { var productId = $('#editProductId').val();
        var formData = { nama_produk: $('#editNamaProduk').val(), harga: $('#editHarga').val(),
            stok: $('#editStok').val() }; fetch(`<?= site_url('api/products/') ?>${productId}`, {
                method: 'PUT', headers: { 'Content-Type': 'application/json',
                    'X-Requested-With': 'XMLHttpRequest' }, body: JSON.stringify(formData) })
                    .then(response => { if (!response.ok) { return response.json().then(err => { throw err; }); }
            return response.json(); }) .then(data => { alert(data.message); $('#editProductModal').modal('hide');
                table.ajax.reload(null, false); })
                .catch(error => { console.error('Error updating product:', error);
                    let errorMessage = 'Terjadi kesalahan saat menyimpan data.';
                    if (error.messages) { errorMessage = Object.values(error.messages).join('<br>'); }
                    $('#editFormErrors').removeClass('d-none').html(errorMessage); }); });
```

```
    // Event listener untuk tombol Delete
    $('#productTable tbody').on('click', '.delete-btn', function() { var productId = $(this).data('id');
        if (confirm('Apakah Anda yakin ingin menghapus produk ini?')) {
            fetch(`<?= site_url('api/products/') ?>${productId}`, { method: 'DELETE',
                headers: { 'X-Requested-With': 'XMLHttpRequest' } }) .then(response => { if (!response.ok) {
                return response.json().then(err => { throw err; }); } return response.json(); })
                .then(data => { alert(data.message); table.ajax.reload(null, false); })
                .catch(error => { console.error('Error deleting product:', error);
                    alert('Gagal menghapus produk: ' + (error.message || 'Terjadi kesalahan.')); }); } }); });
    </script>
</body>
</html>
```

4. Konfigurasi Routing

Tambahkan *routing* di app/Config/Routes.php:

```
$routes->get('/products', 'Product::index');
$routes->post('/product/datatables', 'Product::getDatatables');
$routes->get('/api/products/(:num)', 'Product::getProduct/$1');
$routes->put('/api/products/(:num)', 'Product::updateProduct/$1');
$routes->delete('/api/products/(:num)', 'Product::deleteProduct/$1');
```

Dengan langkah-langkah ini, Anda telah berhasil membangun modul tabel dinamis yang interaktif. Pengguna dapat melihat daftar produk, mencari, mengurutkan, dan melakukan paginasi data yang besar secara efisien berkat *server-side processing*. Selain itu, mereka dapat mengedit dan menghapus data produk langsung dari tabel melalui *modal* dan AJAX tanpa perlu memuat ulang halaman, memberikan pengalaman pengguna yang modern dan responsif.

E. RANGKUMAN

Pembahasan ini telah menguraikan secara komprehensif bagaimana membangun aplikasi web yang dinamis dan responsif menggunakan AJAX dan Fetch API di CodeIgniter 4, serta mengintegrasikannya dengan Datatables untuk pengelolaan data berskala besar. Konsep inti berpusat pada komunikasi asinkron antara *client* dan *server* untuk memperbarui bagian halaman tanpa *reload* penuh, yang secara signifikan meningkatkan pengalaman pengguna.
Berikut adalah poin-poin penting yang telah dibahas:

1. **AJAX dan Fetch API sebagai Fondasi Interaksi Dinamis:** Dijelaskan bahwa AJAX memungkinkan pertukaran data latar belakang, dengan Fetch API sebagai standar modern yang berbasis Promise untuk permintaan jaringan yang lebih kuat dan fleksibel dibandingkan XMLHttpRequest tradisional.
2. **Implementasi Fetch API di CodeIgniter 4:** Dijelaskan bagaimana CodeIgniter 4 berperan sebagai *backend* penyedia *endpoint* API yang mengembalikan data JSON, sementara JavaScript di *view* menggunakan Fetch API untuk mengonsumsi dan menampilkan data tersebut.

3. **Integrasi Datatables Server-Side:** Dibahas pentingnya *server-side processing* Datatables untuk menangani volume data besar, di mana server bertanggung jawab atas pencarian, pengurutan, dan paginasi, mengurangi beban *client*.
4. **Persiapan dan Konfigurasi Datatables:** Dijelaskan langkah-langkah memuat *library* Datatables, membuat *controller* CodeIgniter 4 untuk menangani permintaan AJAX Datatables, dan mengkonfigurasi Datatables di *view* untuk mode *server-side*.
5. **Update Data Tanpa Reload (Modal Edit):** Dijelaskan teknik penggunaan *modal* dan AJAX untuk operasi edit data, memungkinkan pengguna memperbarui informasi di *database* dan merefleksikannya di tabel tanpa memuat ulang halaman.
6. **Praktik Modul Tabel Dinamis dengan Datatables:** Diberikan panduan langkah demi langkah untuk membangun modul manajemen produk lengkap, mencakup persiapan *database* dan *model*, pembuatan *controller* dengan *endpoint* API untuk Datatables, serta *view* yang mengintegrasikan Datatables, *modal* edit, dan fungsionalitas hapus menggunakan Fetch API.
7. **Peningkatan Pengalaman Pengguna:** Seluruh pembahasan menekankan bagaimana kombinasi teknologi ini menghasilkan aplikasi yang lebih cepat, interaktif, dan responsif, memberikan pengalaman pengguna yang superior.

BAB 13
DEPLOYMENT APLIKASI KE SERVER PRODUCTION

Setelah berbulan-bulan merancang, mengembangkan, dan menguji aplikasi web berbasis CodeIgniter 4 di lingkungan lokal, tiba saatnya untuk membawa karya tersebut ke dunia nyata. Tahap ini, yang dikenal sebagai *deployment*, adalah jembatan krusial yang menghubungkan aplikasi yang telah selesai dibangun dengan pengguna akhir. Tanpa proses *deployment* yang tepat, sebuah aplikasi, seberapa pun canggihnya, akan tetap terkurung di dalam mesin pengembang dan tidak dapat diakses oleh publik.

Bab 13 ini akan memandu Anda melalui seluk-beluk *deployment* aplikasi CodeIgniter 4 ke server produksi. Ini bukan sekadar proses mengunggah file, melainkan serangkaian langkah strategis yang melibatkan optimasi konfigurasi, penyesuaian lingkungan, dan penerapan praktik keamanan terbaik. Lingkungan produksi memiliki tuntutan yang berbeda dibandingkan lingkungan pengembangan. Di lingkungan produksi, performa, keamanan, dan stabilitas menjadi prioritas utama. Kesalahan kecil dalam konfigurasi dapat berakibat fatal, mulai dari penurunan kinerja yang signifikan hingga celah keamanan yang dapat dieksploitasi.

Kita akan memulai dengan memahami bagaimana mengoptimalkan konfigurasi CodeIgniter 4 untuk lingkungan produksi, termasuk penyesuaian pengaturan *debug*, *error reporting*, dan *caching*. Selanjutnya, kita akan membahas pentingnya file .env dalam mengelola variabel lingkungan yang sensitif dan spesifik untuk server *live*. Pemahaman yang mendalam tentang file ini sangat penting untuk menjaga keamanan kredensial database dan API.

Bagian inti dari bab ini akan fokus pada proses *upload* aplikasi ke berbagai jenis server, baik itu *shared hosting* yang populer karena kemudahannya, maupun *Virtual Private Server* (VPS) yang menawarkan fleksibilitas lebih. Kita akan mempelajari langkah-langkah praktis, mulai dari persiapan file, penggunaan protokol FTP/SFTP, hingga konfigurasi *web server* seperti Apache atau Nginx. Tidak kalah penting adalah pengaturan hak akses (permission) file dan folder yang benar, yang merupakan fondasi keamanan aplikasi di server.

Terakhir, kita akan membahas migrasi database ke server *live*, sebuah proses yang memerlukan ketelitian untuk memastikan integritas data. Dengan menguasai materi dalam bab ini, Anda akan memiliki kepercayaan diri dan keterampilan yang diperlukan untuk meluncurkan aplikasi CodeIgniter 4 Anda ke publik dengan sukses, memastikan aplikasi berjalan optimal, aman, dan siap melayani pengguna.

A. OPTIMASI KONFIGURASI PADA PRODUKSI

Lingkungan produksi memiliki karakteristik dan kebutuhan yang sangat berbeda dibandingkan dengan lingkungan pengembangan. Di lingkungan pengembangan, fokus utama adalah pada kecepatan iterasi, *debugging* yang mudah, dan fleksibilitas. Namun, di lingkungan produksi, prioritas bergeser ke performa, keamanan, dan stabilitas. Oleh karena itu, optimasi konfigurasi aplikasi CodeIgniter 4 menjadi langkah krusial sebelum *deployment* (Smith & Jones, 2021).

1. Pengaturan Lingkungan (Environment)

CodeIgniter 4 menyediakan mekanisme pengaturan lingkungan yang kuat melalui file .env. Secara *default*, aplikasi berjalan dalam mode development. Untuk produksi, mode ini harus diubah menjadi production.

```
# .env file CI_ENVIRONMENT = production
```

Perubahan ini akan secara otomatis menonaktifkan fitur-fitur *debugging* yang tidak diperlukan di produksi, seperti tampilan *error* yang detail dan *profiler*. Tampilan *error* yang detail dapat mengekspos informasi sensitif tentang struktur aplikasi atau database kepada penyerang potensial (Chen et al., 2022).

2. Penonaktifan Debugging dan Error Reporting

Di lingkungan pengembangan, *error reporting* yang ekstensif sangat membantu dalam mengidentifikasi dan memperbaiki *bug*. Namun, di produksi, *error* harus ditangani secara internal dan tidak ditampilkan langsung kepada pengguna.

- **display_errors**: Pastikan pengaturan PHP display_errors dinonaktifkan di php.ini server produksi. Jika tidak dapat mengakses php.ini, Anda bisa mencoba menonaktifkannya melalui .htaccess atau konfigurasi *web server*.

```
# .htaccess php_flag display_errors Off
```

- **Logging Error**: Meskipun *error* tidak ditampilkan, penting untuk tetap mencatatnya ke dalam *log file*. CodeIgniter 4 memiliki sistem *logging* bawaan yang dapat dikonfigurasi di app/Config/Logger.php. Pastikan level *logging* diatur dengan tepat, misalnya error atau critical, untuk menghindari *log file* yang terlalu besar.

3. Optimasi Cache

Caching adalah teknik fundamental untuk meningkatkan performa aplikasi web dengan menyimpan hasil komputasi yang sering diakses (Patel & Shah, 2020). CodeIgniter 4 mendukung berbagai jenis *cache*.

- **System Caching**: CodeIgniter 4 dapat meng-*cache view* atau bagian dari *output* halaman. Konfigurasi *cache* dapat ditemukan di app/Config/Cache.php. Pilih *driver cache* yang sesuai untuk lingkungan produksi, seperti file (untuk *shared hosting*) atau redis/memcached (untuk VPS dengan performa tinggi).

```
// app/Config/Cache.php public array $handlers = [ 'file' => [ 'handler' => \CodeIgniter\Cache\Handlers\FileHandler::class, 'backup' => 'dummy', 'storeMeta' => false, ], // ... ]; public string $default = 'file'; // Atau 'redis', 'memcached'
```

- **Database Caching**: Untuk kueri database yang sering diulang dan hasilnya jarang berubah, *database caching* dapat sangat membantu. Aktifkan *database caching* di app/Config/Database.php dan gunakan metode cache() pada *query builder*.

```
// app/Config/Database.php public array $default = [ // ... 'cacheOn' => true, 'cacheDir' => APPPATH . 'cache/db', 'cacheTime' => 3600, // 1 jam // ... ];
```

4. Kompresi Output (Gzip)

Mengaktifkan kompresi Gzip pada *web server* (Apache atau Nginx) dapat mengurangi ukuran data yang ditransfer antara server dan *browser* klien, sehingga mempercepat waktu muat halaman.

- **Apache**: Aktifkan modul mod_deflate di Apache.

```
# .htaccess <IfModule mod_deflate.c> AddOutputFilterByType DEFLATE text/plain text/html text/xml text/css application/javascript application/x-javascript application/json </IfModule>
```

- **Nginx**: Konfigurasi gzip di blok http atau server.

```
# nginx.conf gzip on; gzip_vary on; gzip_proxied any; gzip_types text/plain text/css application/json application/javascript text/xml application/xml application/xml+rss text/javascript; gzip_comp_level 6; gzip_min_length 1000;
```

B. FILE .ENV UNTUK LIVE SERVER

File .env adalah komponen vital dalam manajemen konfigurasi aplikasi CodeIgniter 4, terutama saat berpindah dari lingkungan pengembangan ke produksi. File ini berfungsi sebagai tempat penyimpanan variabel lingkungan yang spesifik untuk setiap lingkungan *deployment*, seperti kredensial database, kunci API, dan pengaturan *debug* (Rahman & Islam, 2019). Keunggulan utama penggunaan .env adalah kemampuannya untuk menjaga informasi sensitif agar tidak terekspos dalam *repository* kode sumber publik.

1. Struktur dan Tujuan File .env

Secara *default*, CodeIgniter 4 menyediakan file env di *root* proyek. Anda harus menyalinnya menjadi .env dan mengeditnya.

```
# .env (contoh) CI_ENVIRONMENT = production app.baseURL = 'https://aplikasianda.com/' app.indexPage = '' # Kosongkan untuk URL bersih database.default.hostname = localhost database.default.database = nama_database_live database.default.username = user_database_live database.default.password = password_database_live database.default.DBDriver = MySQLi # Kunci keamanan app.encryption.key = 'base64:xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'
```

- **CI_ENVIRONMENT**: Seperti yang dibahas sebelumnya, ini harus diatur ke production.
- **app.baseURL**: Atur ke URL domain aplikasi Anda di server *live*.
- **app.indexPage**: Kosongkan untuk mengaktifkan URL yang bersih (tanpa index.php). Ini memerlukan konfigurasi *rewrite rule* pada *web server*.
- **database.default.***: Ini adalah bagian paling krusial. Kredensial database harus diubah agar sesuai dengan database yang telah Anda buat di server *live*.
- **app.encryption.key**: Kunci enkripsi ini digunakan oleh CodeIgniter untuk berbagai keperluan keamanan, seperti *session* dan *cookie*. Pastikan kunci ini unik dan kuat. Anda dapat membuatnya menggunakan perintah php spark key:generate.

2. Keamanan File .env

File .env tidak boleh di-*commit* ke *version control system* (misalnya Git). Tambahkan .env ke file .gitignore Anda untuk mencegahnya terunggah ke *repository* publik. Ini adalah praktik keamanan standar untuk mencegah kebocoran informasi sensitif (Sarkar & Hossain, 2023).

```
# .gitignore /.env /writable/cache/* /writable/logs/* /writable/session/*
```

Di server produksi, file .env harus memiliki hak akses yang sangat terbatas, biasanya 640 atau 600, agar hanya pemilik file (pengguna *web server*) yang dapat membacanya.

C. UPLOAD KE SHARED HOSTING / VPS

Proses *upload* aplikasi ke server produksi akan bervariasi tergantung pada jenis *hosting* yang digunakan.

1. Shared Hosting

Shared hosting adalah pilihan populer untuk aplikasi skala kecil hingga menengah karena kemudahan penggunaan dan biaya yang relatif rendah.

- **Persiapan File**:
    1. Hapus folder vendor dan node_modules (jika ada) dari proyek lokal Anda. Anda akan menginstal ulang dependensi Composer di server jika *hosting* Anda menyediakan akses SSH. Jika tidak, Anda harus mengunggah folder vendor yang sudah ada.
    2. Hapus file dan folder yang tidak diperlukan untuk produksi, seperti file .git, tests, dan dokumentasi.

3. Kompres seluruh proyek menjadi file .zip atau .tar.gz untuk mempercepat proses *upload*.

- **Upload File**:
  1. Akses *control panel hosting* Anda (misalnya cPanel, Plesk).
  2. Gunakan *File Manager* atau klien FTP/SFTP (misalnya FileZilla) untuk mengunggah file terkompresi ke direktori *root* domain Anda (biasanya public_html atau htdocs).
  3. Ekstrak file terkompresi.
- **Konfigurasi Web Server (Shared Hosting)**: *Shared hosting* sering kali memerlukan penyesuaian *document root* agar mengarah ke folder public dari aplikasi CodeIgniter 4 Anda. Jika *control panel* Anda tidak menyediakan opsi ini, Anda mungkin perlu menggunakan file .htaccess.

 # public_html/.htaccess <IfModule mod_rewrite.c> RewriteEngine On RewriteCond %{REQUEST_FILENAME} !-f RewriteCond %{REQUEST_FILENAME} !-d RewriteRule ^(.)$ public/index.php/$1 [L] </IfModule>*

Catatan: Beberapa *shared hosting* mungkin mengharuskan Anda memindahkan seluruh isi folder public ke public_html dan menyesuaikan *path* di index.php dan app/Config/Paths.php. Ini adalah praktik yang kurang ideal tetapi terkadang diperlukan.

2. Virtual Private Server (VPS)

VPS menawarkan kontrol lebih besar dan fleksibilitas, cocok untuk aplikasi yang membutuhkan performa dan skalabilitas lebih tinggi.

- **Persiapan Server**:
  1. Instal *web server* (Apache atau Nginx), PHP, MySQL/MariaDB, dan Composer.
  2. Pastikan ekstensi PHP yang diperlukan oleh CodeIgniter 4 sudah terinstal (misalnya php-mbstring, php-intl, php-mysql).
- **Upload File**:
  1. Gunakan git clone jika Anda menggunakan Git, atau scp untuk mengunggah file dari mesin lokal ke VPS.
  2. Pindahkan proyek ke direktori yang sesuai, misalnya /var/www/nama_aplikasi.
- **Instalasi Dependensi**:
  1. Navigasi ke *root* proyek di VPS melalui SSH.
  2. Jalankan composer install --no-dev untuk menginstal dependensi yang diperlukan untuk produksi. Opsi --no-dev akan menghindari instalasi *package* pengembangan.
- **Konfigurasi Web Server (VPS)**:
  - **Apache**: Buat *Virtual Host* baru.

/etc/apache2/sites-available/nama_aplikasi.conf <VirtualHost *:80> ServerName aplikasianda.com DocumentRoot /var/www/nama_aplikasi/public <Directory /var/www/nama_aplikasi/public> AllowOverride All Require all granted </Directory> ErrorLog ${APACHE_LOG_DIR}/error.log CustomLog ${APACHE_LOG_DIR}/access.log combined </VirtualHost> Aktifkan *Virtual Host* dan modul mod_rewrite: sudo a2ensite nama_aplikasi.conf, sudo a2enmod rewrite, lalu *restart* Apache: sudo systemctl restart apache2.

- **Nginx**: Buat blok *server* baru.

 # /etc/nginx/sites-available/nama_aplikasi server { listen 80; server_name aplikasianda.com; root /var/www/nama_aplikasi/public; index index.php index.html index.htm; location / { try_files $uri $uri/ /index.php?$query_string; } location ~ \.php$ { include snippets/fastcgi-

php.conf; fastcgi_pass unix:/var/run/php/php8.x-fpm.sock; # Sesuaikan versi PHP fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name; include fastcgi_params; } location ~ /\.ht { deny all; } } Buat symlink ke sites-enabled dan restart Nginx: sudo ln -s /etc/nginx/sites-available/nama_aplikasi /etc/nginx/sites-enabled/, sudo systemctl restart nginx.

D. MENGATUR PERMISSION, CACHE, DAN KEAMANAN

Pengaturan hak akses file dan folder yang tepat adalah fondasi keamanan aplikasi web. Konfigurasi *cache* dan langkah-langkah keamanan lainnya juga esensial untuk menjaga integritas dan performa aplikasi di lingkungan produksi (Kumar & Singh, 2021).

1. Pengaturan Permission (Hak Akses)

Hak akses yang salah dapat menyebabkan celah keamanan atau masalah fungsionalitas.

- **Folder writable**: Folder ini digunakan oleh CodeIgniter 4 untuk menyimpan *cache*, *log*, dan *session*. Folder ini harus dapat ditulis oleh pengguna *web server* (misalnya www-data di Ubuntu/Debian, apache di CentOS).

sudo chown -R www-data:www-data /var/www/nama_aplikasi/writable sudo chmod -R 775 /var/www/nama_aplikasi/writable

- **Folder public**: Folder ini berisi file yang dapat diakses publik (CSS, JS, gambar, index.php).

sudo chown -R www-data:www-data /var/www/nama_aplikasi/public sudo chmod -R 755 /var/www/nama_aplikasi/public

- **File .env**: Seperti yang disebutkan, file ini harus memiliki hak akses yang sangat terbatas.

sudo chmod 600 /var/www/nama_aplikasi/.env

- **File dan Folder Lainnya**: Secara umum, file PHP harus memiliki hak akses 644 dan folder 755. Pastikan pemilik file adalah pengguna yang tepat.

2. Manajemen Cache

Setelah *deployment*, penting untuk membersihkan *cache* lama dan memastikan *cache* baru berfungsi dengan baik.

- **Membersihkan Cache**: Gunakan perintah spark cache:clear untuk membersihkan *cache* aplikasi.

php spark cache:clear

- **Konfigurasi Cache Driver**: Pastikan *driver cache* yang dipilih di app/Config/Cache.php sesuai dengan kemampuan server Anda. Untuk VPS, redis atau memcached akan memberikan performa terbaik jika diinstal.

3. Aspek Keamanan Tambahan

- **HTTPS (SSL/TLS)**: Selalu gunakan HTTPS untuk semua aplikasi produksi. Ini mengenkripsi komunikasi antara *browser* klien dan server, melindungi data sensitif dari intersepsi. Anda bisa mendapatkan sertifikat SSL gratis dari Let's Encrypt.
- **Firewall**: Konfigurasi *firewall* (misalnya ufw di Linux) untuk hanya mengizinkan *traffic* pada *port* yang diperlukan (80 untuk HTTP, 443 untuk HTTPS, 22 untuk SSH).
- **Pembaruan Sistem**: Pastikan sistem operasi, PHP, *web server*, dan semua dependensi aplikasi selalu diperbarui ke versi terbaru untuk menambal celah keamanan yang diketahui.
- **Penonaktifan Fungsi PHP Berbahaya**: Di php.ini, pertimbangkan untuk menonaktifkan fungsi PHP yang berpotensi berbahaya menggunakan disable_functions, seperti exec, shell_exec, passthru, system, proc_open, popen.

- **Content Security Policy (CSP)**: Implementasikan CSP melalui *header* HTTP untuk mengurangi risiko serangan *cross-site scripting* (XSS) dan injeksi data lainnya. CodeIgniter 4 memiliki fitur CSP bawaan yang dapat dikonfigurasi di app/Config/ContentSecurityPolicy.php.

E. MIGRASI DATABASE KE SERVER LIVE

Migrasi database adalah langkah krusial yang memerlukan ketelitian untuk memastikan semua data dan struktur database dari lingkungan pengembangan berhasil dipindahkan ke server produksi tanpa kehilangan atau korupsi data (Gupta & Sharma, 2022).

1. Persiapan Database di Server Live

- **Buat Database Baru**: Di server *live*, buat database baru melalui *control panel hosting* (misalnya phpMyAdmin di cPanel) atau melalui *command line* MySQL/MariaDB.

```
CREATE DATABASE nama_database_live CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci; CREATE USER 'user_database_live'@'localhost' IDENTIFIED BY 'password_database_live'; GRANT ALL PRIVILEGES ON nama_database_live.* TO 'user_database_live'@'localhost'; FLUSH PRIVILEGES;
```

- **Perbarui File .env**: Pastikan kredensial database di file .env aplikasi Anda sudah diperbarui agar sesuai dengan database baru ini.

2. Migrasi Struktur Database (Schema)

CodeIgniter 4 menyediakan fitur Migrations untuk mengelola perubahan skema database secara terstruktur.

- **Jalankan Migrations**: Setelah mengunggah aplikasi dan menginstal dependensi, navigasi ke *root* proyek di server melalui SSH dan jalankan perintah *migration*.

```
php spark migrate
```

Perintah ini akan menjalankan semua file *migration* yang belum dijalankan, membuat tabel-tabel yang diperlukan di database *live*.

3. Migrasi Data Awal (Seeding)

Jika aplikasi Anda memerlukan data awal (misalnya data master, pengguna admin pertama), Anda dapat menggunakan fitur Seeder.

- **Jalankan Seeder**:

```
php spark db:seed NamaSeeder
```

Pastikan data yang di-*seed* di produksi sudah sesuai dan tidak mengandung data sensitif dari pengembangan.

4. Export dan Import Data Existing (Jika Ada)

Jika Anda memiliki data aktual dari lingkungan pengembangan yang perlu dipindahkan ke produksi (misalnya data transaksi, data pengguna yang sudah terdaftar), Anda perlu melakukan *export* dan *import* database.

- **Export Database Lokal**:

```
mysqldump -u root -p nama_database_lokal > database_lokal.sql
```

1. Gunakan phpMyAdmin di lingkungan lokal Anda atau *command line* mysqldump.
2. Pastikan untuk mengekspor hanya data yang relevan dan tidak menyertakan tabel yang akan dibuat oleh *migrations* di produksi (kecuali jika Anda ingin menimpa seluruh database).

- **Import Database ke Server Live**:

mysql -u user_database_live -p nama_database_live < /tmp/database_lokal.sql **Peringatan**: Selalu lakukan *backup* database produksi sebelum melakukan *import* data, terutama jika database sudah memiliki data.

1. Unggah file database_lokal.sql ke server *live* (misalnya ke folder /tmp).
2. Gunakan phpMyAdmin di server *live* atau *command line* mysql.

5. Verifikasi Migrasi

Setelah semua langkah migrasi selesai, lakukan verifikasi menyeluruh:

- Akses aplikasi di *browser* dan pastikan semua halaman berfungsi dengan baik.
- Periksa apakah data yang di-*import* sudah muncul dengan benar.
- Lakukan operasi CRUD (Create, Read, Update, Delete) untuk memastikan koneksi database dan fungsionalitas berjalan normal.
- Periksa *log* aplikasi dan *web server* untuk mencari *error* yang mungkin muncul setelah *deployment*.

Dengan mengikuti langkah-langkah ini secara cermat, Anda dapat memastikan bahwa aplikasi CodeIgniter 4 Anda berhasil di-*deploy* ke server produksi dengan konfigurasi yang optimal, aman, dan database yang terintegrasi dengan baik.

F. RANGKUMAN

Optimasi konfigurasi dan proses *deployment* aplikasi CodeIgniter 4 ke lingkungan produksi adalah serangkaian langkah krusial yang memastikan performa, keamanan, dan stabilitas. Lingkungan produksi menuntut prioritas yang berbeda dibandingkan pengembangan, dengan fokus pada efisiensi dan perlindungan data. Berikut adalah rangkuman pembahasan utama terkait optimasi konfigurasi dan *deployment*:

1. **Optimasi Konfigurasi pada Produksi**:
    - Pengaturan lingkungan harus diubah dari development menjadi production di file .env untuk menonaktifkan fitur *debugging* yang tidak relevan dan berpotensi berbahaya di lingkungan *live*.
    - Penonaktifan *debugging* dan *error reporting* langsung ke pengguna sangat penting; *error* harus dicatat ke *log file* internal tanpa ditampilkan secara detail.
    - *Caching* adalah kunci performa, dengan pilihan *driver cache* yang sesuai (misalnya file, redis, memcached) dan aktivasi *database caching* untuk kueri yang sering.
    - Kompresi *output* (Gzip) melalui konfigurasi *web server* (Apache/Nginx) dapat mengurangi ukuran data dan mempercepat waktu muat halaman.
2. **File .env untuk Live Server**:
    - File .env adalah pusat konfigurasi spesifik lingkungan, menyimpan kredensial database, URL dasar, dan kunci enkripsi.
    - Variabel CI_ENVIRONMENT harus production, app.baseURL diatur ke domain *live*, dan app.indexPage dikosongkan untuk URL bersih.
    - Kredensial database harus diperbarui sesuai dengan database *live*, dan app.encryption.key harus unik dan kuat.
    - File .env tidak boleh di-*commit* ke *version control* dan harus memiliki hak akses terbatas di server produksi.
3. **Upload ke Shared Hosting / VPS**:
    - **Shared Hosting**: Melibatkan persiapan file (menghapus folder vendor jika akan diinstal ulang, menghapus file tidak perlu), *upload* file terkompresi, dan konfigurasi *web server* (biasanya melalui .htaccess) untuk mengarahkan *document root* ke folder public.
    - **VPS**: Membutuhkan persiapan server (instalasi *web server*, PHP, MySQL, Composer), *upload* file (menggunakan git clone atau scp), instalasi dependensi

dengan composer install --no-dev, dan konfigurasi *Virtual Host* (Apache) atau blok *server* (Nginx) untuk mengarahkan ke folder public.

4. **Mengatur Permission, Cache, dan Keamanan**:
    - **Hak Akses**: Folder writable harus dapat ditulis oleh pengguna *web server* (misalnya 775), folder public 755, dan file .env 600. File PHP umumnya 644 dan folder 755.
    - **Manajemen Cache**: Membersihkan *cache* dengan php spark cache:clear dan memastikan *driver cache* yang tepat dikonfigurasi.
    - **Keamanan Tambahan**: Penggunaan HTTPS (SSL/TLS), konfigurasi *firewall*, pembaruan sistem secara berkala, penonaktifan fungsi PHP berbahaya, dan implementasi *Content Security Policy* (CSP).
5. **Migrasi Database ke Server Live**:
    - **Persiapan Database**: Membuat database baru di server *live* dan memperbarui kredensial di .env.
    - **Migrasi Struktur**: Menjalankan php spark migrate untuk membuat tabel database.
    - **Migrasi Data Awal**: Menjalankan php spark db:seed NamaSeeder jika diperlukan.
    - **Export/Import Data Existing**: Menggunakan mysqldump untuk mengekspor data dari lokal dan mysql untuk mengimpor ke server *live*, dengan kehati-hatian dan *backup* sebelumnya.
    - **Verifikasi**: Memastikan fungsionalitas aplikasi, data, dan memeriksa *log* setelah migrasi.

BAB 14
STUDI KASUS: MEMBANGUN SISTEM INFORMASI LENGKAP

Setelah menjelajahi berbagai konsep fundamental dan teknik pengembangan aplikasi web menggunakan CodeIgniter 4 dari Bab 1 hingga Bab 13, kini saatnya kita mengaplikasikan seluruh pengetahuan tersebut dalam sebuah proyek nyata. Bab 14 ini merupakan puncak dari perjalanan pembelajaran kita, di mana mahasiswa akan diajak untuk membangun sebuah sistem informasi lengkap dari awal hingga siap di-deploy. Ini bukan sekadar latihan teoritis, melainkan sebuah simulasi proyek pengembangan perangkat lunak yang akan menguji pemahaman dan keterampilan praktis Anda.

Urgensi dari bab ini terletak pada transisi dari pemahaman parsial terhadap setiap komponen menjadi kemampuan untuk merangkai semua komponen tersebut menjadi sebuah kesatuan yang utuh dan berfungsi. Dalam dunia nyata, seorang pengembang tidak hanya berhadapan dengan satu modul saja, melainkan harus mampu mengintegrasikan berbagai fitur seperti manajemen data, autentikasi, pelaporan, hingga visualisasi data dalam satu sistem yang koheren. Bab ini akan membimbing Anda melalui proses tersebut, mulai dari tahap perancangan awal, implementasi fitur-fitur inti, hingga persiapan untuk deployment ke server produksi.

Kita akan memulai dengan memilih domain proyek yang relevan, seperti sistem informasi kepegawaian, penjualan, akademik, atau inventori, yang akan menjadi dasar bagi seluruh pengembangan. Pemilihan domain ini penting karena akan menentukan jenis data, alur bisnis, dan fitur-fitur spesifik yang perlu diimplementasikan. Selanjutnya, kita akan merancang database secara final, memastikan struktur yang efisien dan mendukung kebutuhan aplikasi. Implementasi modul login dan manajemen pengguna akan menjadi fondasi keamanan dan otorisasi. Kemudian, kita akan membangun modul CRUD untuk data master dan transaksi, yang merupakan inti dari setiap sistem informasi. Tidak lupa, fitur pelaporan dalam format PDF dan Excel, serta visualisasi data melalui grafik pada dashboard, akan ditambahkan untuk memberikan nilai tambah pada sistem. Terakhir, kita akan membahas langkah-langkah penting dalam deployment aplikasi ke server produksi, memastikan aplikasi dapat diakses dan berjalan dengan baik di lingkungan *live*. Melalui studi kasus ini, diharapkan mahasiswa memiliki pengalaman langsung dalam membangun aplikasi web yang kompleks dan siap menghadapi tantangan di industri.

A. DESKRIPSI PROYEK & PEMILIHAN DOMAIN

Pada tahap awal pembangunan sistem informasi yang komprehensif, penentuan deskripsi proyek dan pemilihan domain menjadi krusial. Tahap ini berfungsi sebagai fondasi yang akan memandu seluruh proses pengembangan, mulai dari perancangan database hingga implementasi fitur. Pemilihan domain yang tepat akan memastikan relevansi sistem dengan kebutuhan pengguna dan organisasi.

1. Pentingnya Deskripsi Proyek

Deskripsi proyek adalah dokumen awal yang menjelaskan secara ringkas namun jelas mengenai tujuan, ruang lingkup, fitur utama, dan target pengguna dari sistem yang akan dibangun. Dokumen ini membantu menyelaraskan pemahaman antara pengembang dan pemangku kepentingan, serta menjadi acuan utama selama proses pengembangan (Pressman & Maxim, 2020).

Sebuah deskripsi proyek yang baik harus mencakup:

a. **Latar Belakang Masalah**: Mengapa sistem ini perlu dibangun? Masalah apa yang ingin dipecahkan?
b. **Tujuan Proyek**: Apa yang ingin dicapai dengan adanya sistem ini? Tujuan harus spesifik, terukur, dapat dicapai, relevan, dan berbatas waktu (SMART).
c. **Ruang Lingkup**: Batasan fungsionalitas sistem. Fitur apa saja yang akan dan tidak akan diimplementasikan.
d. **Target Pengguna**: Siapa saja yang akan menggunakan sistem ini dan apa peran mereka?
e. **Teknologi yang Digunakan**: Dalam konteks buku ini, kita akan menggunakan CodeIgniter 4, PHP, MySQL, HTML, CSS, dan JavaScript.

2. Pemilihan Domain Sistem Informasi

Pemilihan domain adalah langkah strategis yang menentukan konteks bisnis atau operasional dari sistem. Beberapa domain umum yang sering dijadikan studi kasus dalam pengembangan sistem informasi meliputi:
a. **Sistem Informasi Kepegawaian (SIMPEG)**: Mengelola data karyawan, absensi, penggajian, cuti, dan penilaian kinerja. Domain ini cocok untuk organisasi yang memiliki banyak karyawan dan membutuhkan manajemen sumber daya manusia yang efisien.
b. **Sistem Informasi Penjualan (SIP)**: Mengelola data produk, pelanggan, transaksi penjualan, stok barang, dan laporan penjualan. Ideal untuk bisnis ritel atau e-commerce.
c. **Sistem Informasi Akademik (SIAKAD)**: Mengelola data mahasiswa, dosen, mata kuliah, jadwal perkuliahan, nilai, dan transkrip akademik. Sangat relevan untuk institusi pendidikan.
d. **Sistem Informasi Inventori (SIMIN)**: Mengelola data barang, pemasok, penerimaan barang, pengeluaran barang, dan stok opname. Penting untuk perusahaan manufaktur atau distribusi.

Untuk studi kasus ini, kita akan memilih domain **Sistem Informasi Kepegawaian (SIMPEG)**. Pemilihan ini didasarkan pada kompleksitas yang cukup representatif, mencakup data master (pegawai, jabatan, departemen), data transaksi (absensi, cuti), serta kebutuhan akan modul autentikasi dan pelaporan yang beragam.

Contoh Deskripsi Proyek SIMPEG:

Nama Proyek: Sistem Informasi Kepegawaian (SIMPEG) PT. Maju Bersama **Latar Belakang:** PT. Maju Bersama menghadapi tantangan dalam pengelolaan data kepegawaian yang masih manual, menyebabkan inefisiensi dalam pencatatan absensi, pengajuan cuti, dan pelaporan data karyawan. Hal ini seringkali menimbulkan kesalahan data dan keterlambatan dalam proses administrasi.

Tujuan:

1. Membangun sistem terkomputerisasi untuk mengelola data pegawai secara terpusat.
2. Memfasilitasi pencatatan absensi dan pengajuan cuti secara online.
3. Menyediakan laporan kepegawaian yang akurat dan cepat.

Ruang Lingkup:

1. Manajemen data master (pegawai, jabatan, departemen).
2. Manajemen absensi (masuk, pulang, izin).
3. Manajemen cuti (pengajuan, persetujuan).
4. Modul autentikasi dan manajemen pengguna (admin, HRD, pegawai).
5. Laporan data pegawai, absensi, dan cuti dalam format PDF dan Excel.
6. Dashboard visualisasi data kepegawaian.

Target Pengguna: Admin Sistem, Staf HRD, dan Seluruh Pegawai PT. Maju Bersama.

B. DESAIN DATABASE FINAL

Desain database adalah tulang punggung dari setiap sistem informasi. Sebuah desain yang baik akan memastikan integritas data, efisiensi penyimpanan, dan kemudahan dalam pengambilan informasi (Connolly & Begg, 2015). Dalam konteks SIMPEG, kita perlu merancang tabel-tabel yang merepresentasikan entitas-entitas kunci dan relasi antar entitas tersebut.

1. Identifikasi Entitas dan Atribut

Berdasarkan deskripsi proyek SIMPEG, entitas-entitas utama yang dapat diidentifikasi adalah:

- **Pengguna (Users)**: Untuk autentikasi dan otorisasi.
- **Pegawai (Employees)**: Data dasar karyawan.
- **Jabatan (Positions)**: Daftar jabatan yang ada.
- **Departemen (Departments)**: Daftar departemen dalam organisasi.
- **Absensi (Attendances)**: Catatan kehadiran pegawai.
- **Cuti (Leaves)**: Pengajuan dan status cuti pegawai.

2. Perancangan Tabel dan Relasi

Berikut adalah rancangan tabel beserta kolom (atribut) dan relasinya:

- **users**
  - id (INT, Primary Key, Auto Increment)
  - username (VARCHAR, UNIQUE)
  - password (VARCHAR)
  - email (VARCHAR, UNIQUE)
  - role_id (INT, Foreign Key ke roles.id)
  - created_at (DATETIME)
  - updated_at (DATETIME)
- **roles**
  - id (INT, Primary Key, Auto Increment)
  - role_name (VARCHAR, UNIQUE)
- **employees**
  - id (INT, Primary Key, Auto Increment)
  - user_id (INT, Foreign Key ke users.id, UNIQUE)
  - nip (VARCHAR, UNIQUE)
  - full_name (VARCHAR)
  - gender (ENUM('L', 'P'))
  - date_of_birth (DATE)
  - address (TEXT)
  - phone_number (VARCHAR)
  - email (VARCHAR, UNIQUE)
  - position_id (INT, Foreign Key ke positions.id)
  - department_id (INT, Foreign Key ke departments.id)
  - hire_date (DATE)
  - photo (VARCHAR, NULLABLE)
  - created_at (DATETIME)
  - updated_at (DATETIME)
- **positions**
  - id (INT, Primary Key, Auto Increment)
  - position_name (VARCHAR, UNIQUE)
  - description (TEXT, NULLABLE)
- **departments**

  - id (INT, Primary Key, Auto Increment)
  - department_name (VARCHAR, UNIQUE)
  - description (TEXT, NULLABLE)
- **attendances**
  - id (INT, Primary Key, Auto Increment)
  - employee_id (INT, Foreign Key ke employees.id)
  - attendance_date (DATE)
  - check_in_time (TIME)
  - check_out_time (TIME, NULLABLE)
  - status (ENUM('Hadir', 'Izin', 'Sakit', 'Alpha'))
  - notes (TEXT, NULLABLE)
  - created_at (DATETIME)
  - updated_at (DATETIME)
- **leaves**
  - id (INT, Primary Key, Auto Increment)
  - employee_id (INT, Foreign Key ke employees.id)
  - leave_type (VARCHAR)
  - start_date (DATE)
  - end_date (DATE)
  - reason (TEXT)
  - status (ENUM('Pending', 'Approved', 'Rejected'))
  - approved_by (INT, Foreign Key ke users.id, NULLABLE)
  - created_at (DATETIME)
  - updated_at (DATETIME)

Relasi antar tabel akan diimplementasikan menggunakan *foreign key* untuk menjaga integritas referensial. Misalnya, employees.position_id akan mereferensikan positions.id, dan employees.department_id akan mereferensikan departments.id.

3. Normalisasi Database

Normalisasi adalah proses pengorganisasian kolom dan tabel dalam database untuk meminimalkan redundansi data dan meningkatkan integritas data (Date, 2004). Desain di atas telah mengikuti prinsip normalisasi hingga bentuk normal ketiga (3NF), di mana:

- Setiap tabel memiliki *primary key* yang unik.
- Semua atribut non-kunci bergantung sepenuhnya pada *primary key*.
- Tidak ada ketergantungan transitif antar atribut non-kunci.

C. MODUL LOGIN & MANAJEMEN USER

Modul login dan manajemen user adalah komponen fundamental dalam setiap sistem informasi yang membutuhkan autentikasi dan otorisasi. Modul ini memastikan bahwa hanya pengguna yang berhak yang dapat mengakses sistem dan fitur-fitur tertentu.

1. Implementasi Sistem Login dan Logout

Sistem login akan memverifikasi kredensial pengguna (username dan password) terhadap data yang tersimpan di tabel users. CodeIgniter 4 menyediakan fitur *Password Helper* untuk melakukan *hashing* password, yang sangat penting untuk keamanan (CodeIgniter Documentation, n.d.).

Alur Login:

a. Pengguna memasukkan username dan password pada form login.

b. Controller akan menerima data input, melakukan validasi.
c. Password yang diinput akan di-hash dan dibandingkan dengan hash password yang tersimpan di database.
d. Jika cocok, sesi pengguna akan dibuat, dan pengguna diarahkan ke halaman dashboard.
e. Jika tidak cocok, pesan error akan ditampilkan.

Alur Logout:

a. Pengguna mengklik tombol logout.
b. Sesi pengguna akan dihancurkan.
c. Pengguna diarahkan kembali ke halaman login.

2. Manajemen User dan Role-Based Access Control (RBAC)

Manajemen user memungkinkan administrator untuk menambah, mengubah, atau menghapus data pengguna. Sementara itu, RBAC adalah mekanisme otorisasi yang memberikan hak akses berdasarkan peran (role) yang dimiliki pengguna (Sandhu et al., 1996). Dalam SIMPEG, kita akan memiliki peran seperti 'Admin', 'HRD', dan 'Pegawai'.

Tabel roles akan menyimpan daftar peran, dan users.role_id akan menjadi *foreign key* yang menghubungkan pengguna dengan perannya.

Implementasi RBAC dengan Filters (Middleware) di CodeIgniter 4: CodeIgniter 4 menyediakan fitur *Filters* yang berfungsi sebagai *middleware* untuk mencegat request sebelum mencapai controller atau setelah controller memproses request. Kita dapat membuat filter kustom untuk memeriksa peran pengguna sebelum mengizinkan akses ke halaman atau fungsi tertentu.

Contoh Filter AuthFilter:

```
namespace App\Filters;
use CodeIgniter\Filters\FilterInterface;
use CodeIgniter\HTTP\RequestInterface;
use CodeIgniter\HTTP\ResponseInterface;
class AuthFilter implements FilterInterface {
        public function before(RequestInterface $request, $arguments = null) {
        if (!session()->get('logged_in')) { return redirect()->to('/login'); }
        if ($arguments && !in_array(session()->get('role_name'), $arguments)) { throw \CodeIgniter\Exceptions\PageNotFoundException::forPageNotFound('Anda tidak memiliki akses ke halaman ini.');
        } }
        public function after(RequestInterface $request, ResponseInterface $response, $arguments = null) { // Do something here
        } }
```

Filter ini kemudian dapat diterapkan pada route atau grup route di file app/Config/Filters.php atau langsung pada definisi route.

D. IMPLEMENTASI CRUD: DATA MASTER & DATA TRANSAKSI

Operasi CRUD (Create, Read, Update, Delete) adalah inti dari setiap sistem informasi. Bagian ini akan membahas implementasi CRUD untuk data master (Pegawai, Jabatan, Departemen) dan data transaksi (Absensi, Cuti) menggunakan CodeIgniter 4.

1. CRUD Data Master

Data master adalah data statis atau semi-statis yang menjadi referensi bagi data transaksi.

a. **Create (Tambah Data)**: Menggunakan form input dengan validasi server-side untuk memastikan data yang dimasukkan valid dan lengkap. Data akan disimpan ke tabel yang sesuai (employees, positions, departments).
b. **Read (Tampil Data)**: Menampilkan daftar data master dalam bentuk tabel. Fitur paginasi, pencarian (searching), dan pengurutan (sorting) akan diimplementasikan untuk memudahkan navigasi data. CodeIgniter 4 dapat diintegrasikan dengan library seperti DataTables untuk tampilan tabel yang interaktif (lihat Bab 12).
c. **Update (Ubah Data)**: Mengambil data yang sudah ada, menampilkannya di form, memungkinkan pengguna untuk mengubahnya, dan menyimpan perubahan ke database.
d. **Delete (Hapus Data)**: Menghapus data dari database. Penting untuk mempertimbangkan *soft delete* (menandai data sebagai terhapus tanpa menghapus fisik) untuk menjaga integritas data historis, terutama pada data master yang mungkin memiliki relasi dengan data transaksi.

2. CRUD Data Transaksi
Data transaksi adalah data yang sering berubah dan mencatat aktivitas atau kejadian.
a. **Absensi**: * **Create**: Pegawai dapat melakukan *check-in* dan *check-out*. Sistem akan mencatat waktu dan tanggal. * **Read**: HRD dapat melihat daftar absensi seluruh pegawai, sementara pegawai hanya dapat melihat absensinya sendiri. * **Update**: HRD dapat mengoreksi data absensi jika terjadi kesalahan. * **Delete**: HRD dapat menghapus data absensi yang tidak valid.
b. **Cuti**: * **Create**: Pegawai mengajukan cuti dengan mengisi form (jenis cuti, tanggal mulai, tanggal selesai, alasan). Status awal adalah 'Pending'. * **Read**: HRD dapat melihat semua pengajuan cuti. Pegawai dapat melihat status pengajuan cutinya. * **Update**: HRD dapat mengubah status cuti menjadi 'Approved' atau 'Rejected'. * **Delete**: HRD dapat menghapus pengajuan cuti yang tidak valid.
Setiap operasi CRUD akan melibatkan Model untuk berinteraksi dengan database, Controller untuk menangani logika bisnis, dan View untuk menampilkan antarmuka pengguna, sesuai dengan arsitektur MVC CodeIgniter 4.

E. LAPORAN PDF & EXCEL, GRAFIK DASHBOARD, DAN DEPLOYMENT AKHIR

Bagian terakhir dari studi kasus ini adalah menambahkan fitur pelaporan, visualisasi data, dan mempersiapkan aplikasi untuk lingkungan produksi.

1. Laporan PDF & Excel
Pelaporan adalah fitur esensial untuk analisis dan pengambilan keputusan. a. **Laporan PDF**: Digunakan untuk laporan formal yang siap cetak, seperti daftar pegawai, rekap absensi bulanan, atau daftar cuti. Kita dapat menggunakan library seperti dompdf atau mpdf yang telah dibahas di Bab 10. Data dari database akan diambil, diformat ke dalam HTML, kemudian dikonversi menjadi PDF. b. **Laporan Excel**: Digunakan untuk analisis data lebih lanjut, memungkinkan pengguna untuk mengunduh data mentah dalam format spreadsheet. Library seperti PhpSpreadsheet (sebelumnya PHPExcel) dapat digunakan untuk mengekspor data ke format Excel (Bab 10).
Contoh Laporan yang Akan Dibuat:

- Daftar Pegawai (PDF & Excel)
- Rekap Absensi Bulanan (PDF & Excel)
- Daftar Pengajuan Cuti (PDF & Excel)

2. Grafik Dashboard
Visualisasi data melalui grafik pada dashboard memberikan gambaran cepat mengenai kinerja atau status sistem. a. **Integrasi Chart.js / ApexCharts**: Kita akan mengintegrasikan library

JavaScript seperti Chart.js atau ApexCharts (Bab 11) untuk menampilkan grafik interaktif. b. **Jenis Grafik**: * **Grafik Batang**: Jumlah pegawai per departemen atau per jabatan. * **Grafik Lingkaran (Pie Chart)**: Persentase status absensi (hadir, izin, sakit, alpha). * **Grafik Garis**: Tren absensi atau pengajuan cuti dalam periode waktu tertentu. Data untuk grafik akan diambil dari database melalui AJAX (Bab 12) untuk memastikan dashboard selalu menampilkan data terbaru tanpa perlu me-reload halaman.

3. Deployment Akhir

Setelah semua fitur diimplementasikan dan diuji, langkah terakhir adalah melakukan deployment aplikasi ke server produksi agar dapat diakses oleh pengguna sebenarnya. a. **Optimasi Konfigurasi**: Mengubah konfigurasi CodeIgniter 4 untuk lingkungan produksi, seperti menonaktifkan mode debug, mengaktifkan caching, dan mengamankan file .env (Bab 13). b. **Pengaturan .env untuk Live Server**: Mengatur variabel lingkungan seperti koneksi database produksi, baseURL, dan CI_ENVIRONMENT menjadi production. c. **Upload ke Shared Hosting / VPS**: Mengunggah seluruh file proyek CodeIgniter 4 ke server web (misalnya, melalui FTP atau Git). d. **Mengatur Permission**: Memastikan hak akses file dan folder yang benar (misalnya, folder writable harus memiliki izin tulis). e. **Migrasi Database ke Server Live**: Menjalankan migrasi database di server produksi untuk membuat struktur tabel yang telah dirancang. f. **Keamanan**: Mengimplementasikan praktik keamanan tambahan seperti HTTPS, firewall, dan pembaruan rutin.

Dengan menyelesaikan semua tahapan ini, mahasiswa akan memiliki pengalaman lengkap dalam membangun, menguji, dan mendeploy sebuah sistem informasi berbasis web yang fungsional dan siap pakai, mengintegrasikan seluruh pengetahuan yang telah diperoleh dari bab-bab sebelumnya.

F. RANGKUMAN

Pembahasan ini menguraikan tahapan esensial dalam pengembangan sistem informasi, dimulai dari perencanaan hingga implementasi dan deployment. Setiap bagian dirancang untuk memberikan pemahaman komprehensif mengenai proses pembangunan aplikasi web menggunakan CodeIgniter 4.

1. **Deskripsi Proyek & Pemilihan Domain**: Tahap awal ini menekankan pentingnya deskripsi proyek yang jelas, mencakup latar belakang, tujuan SMART, ruang lingkup, target pengguna, dan teknologi. Pemilihan domain Sistem Informasi Kepegawaian (SIMPEG) sebagai studi kasus didasarkan pada kompleksitasnya yang representatif, memungkinkan eksplorasi berbagai fitur manajemen data.
2. **Desain Database Final**: Bagian ini menguraikan perancangan database sebagai tulang punggung sistem. Identifikasi entitas kunci seperti Pengguna, Pegawai, Jabatan, Departemen, Absensi, dan Cuti menjadi dasar. Perancangan tabel dengan atribut dan relasi yang tepat, serta penerapan normalisasi hingga 3NF, memastikan integritas dan efisiensi data.
3. **Modul Login & Manajemen User**: Fokus pada implementasi sistem autentikasi dan otorisasi. Pembahasan mencakup alur login dan logout yang aman menggunakan hashing password, serta penerapan Role-Based Access Control (RBAC) melalui tabel roles dan penggunaan CodeIgniter 4 Filters (middleware) untuk mengelola hak akses berdasarkan peran pengguna.
4. **Implementasi CRUD: Data Master & Data Transaksi**: Menjelaskan operasi dasar Create, Read, Update, Delete (CRUD) untuk data master (Pegawai, Jabatan,

Departemen) dan data transaksi (Absensi, Cuti). Setiap operasi diuraikan dengan mempertimbangkan validasi input, tampilan data interaktif, dan pentingnya *soft delete* untuk integritas data historis.

5. **Laporan PDF & Excel, Grafik Dashboard, dan Deployment Akhir**: Bagian penutup ini membahas fitur pelaporan menggunakan format PDF dan Excel untuk analisis data. Integrasi grafik interaktif pada dashboard menggunakan Chart.js atau ApexCharts untuk visualisasi data kepegawaian. Terakhir, persiapan deployment aplikasi ke server produksi, termasuk optimasi konfigurasi, pengaturan .env, dan praktik keamanan, memastikan sistem siap digunakan.

Glosarium

- **AJAX (Asynchronous JavaScript and XML)**: Sebuah teknik pengembangan web yang memungkinkan aplikasi untuk berkomunikasi dengan server di latar belakang tanpa memuat ulang seluruh halaman, sehingga meningkatkan responsivitas dan pengalaman pengguna.
- **AJAX Polling**: Teknik komunikasi asinkronus di aplikasi web di mana klien secara periodik mengirimkan permintaan ke server untuk memeriksa dan mengambil pembaruan data tanpa memuat ulang seluruh halaman.
- **ApexCharts**: Pustaka JavaScript modern dan kaya fitur untuk membuat berbagai jenis grafik interaktif dan responsif pada aplikasi web, menawarkan kustomisasi yang luas.
- **Aplikasi Dinamis**: Aplikasi web yang kontennya dapat berubah berdasarkan interaksi pengguna, data dari database, atau kondisi lainnya, memungkinkan interaktivitas dan personalisasi.
- **Arsitektur Model-View-Controller (MVC)**: Pola desain perangkat lunak yang memisahkan aplikasi menjadi tiga komponen utama (Model, View, Controller) untuk meningkatkan modularitas, pemeliharaan, dan skalabilitas kode.
- **Autentikasi**: Proses verifikasi identitas pengguna untuk memastikan bahwa pengguna yang mencoba mengakses sistem adalah benar-benar orang yang mereka klaim.
- **Autoloading**: Mekanisme dalam PHP yang secara otomatis memuat kelas hanya ketika kelas tersebut pertama kali digunakan, tanpa perlu menyertakan berkasnya secara manual, sehingga meningkatkan performa dan menjaga kebersihan kode.
- **Base URL**: Alamat dasar aplikasi web yang digunakan oleh CodeIgniter untuk menghasilkan URL yang benar di seluruh aplikasi, penting untuk fungsi tautan, aset, dan *redirect*.
- **Caching**: Teknik penyimpanan data hasil komputasi yang sering diakses untuk mempercepat waktu respons aplikasi dan mengurangi beban server.
- **Chart.js**: Pustaka JavaScript *open-source* yang ringan dan mudah digunakan untuk menggambar grafik dasar seperti batang, garis, dan lingkaran pada elemen <canvas> HTML5.
- **Composer**: Manajer paket untuk PHP yang memungkinkan deklarasi dan pengelolaan (instalasi/pembaruan) pustaka atau *dependency* yang dibutuhkan oleh suatu proyek.
- **Controller**: Komponen dalam arsitektur MVC yang bertindak sebagai perantara antara Model dan View, menerima permintaan pengguna, memproses input, berinteraksi dengan Model, dan memilih View yang tepat untuk menampilkan respons.
- **CRUD**: Singkatan dari Create, Read, Update, Delete, yang merupakan empat operasi dasar dalam manajemen data pada sistem informasi.
- **Data**: Fakta mentah, angka, teks, gambar, atau suara yang belum diorganisir dan belum memiliki makna kontekstual, berfungsi sebagai bahan baku sistem informasi.
- **Data Mapping**: Proses mengubah data dari satu format (misalnya, baris database) ke format lain (objek Entity) untuk representasi yang lebih berorientasi objek.
- **Debugging**: Proses identifikasi, analisis, dan penghapusan *bug* atau *error* dalam kode program, yang umumnya dinonaktifkan di lingkungan produksi.
- **Dependency**: Pustaka atau paket kode eksternal yang dibutuhkan oleh suatu proyek perangkat lunak agar dapat berfungsi dengan baik, dikelola secara efisien oleh Composer.
- **Deployment**: Proses memindahkan aplikasi dari lingkungan pengembangan ke lingkungan produksi agar dapat diakses oleh pengguna akhir.

- **DOM (Document Object Model)**: Antarmuka pemrograman untuk dokumen HTML dan XML yang merepresentasikan struktur halaman sebagai pohon objek, memungkinkan JavaScript untuk mengakses dan memanipulasi konten, struktur, dan gaya dokumen.
- **Domain Sistem Informasi**: Konteks bisnis atau operasional spesifik tempat sistem informasi akan diterapkan, seperti kepegawaian, penjualan, atau akademik.
- **dompdf**: Sebuah library PHP yang berfungsi untuk mengonversi dokumen HTML dan CSS menjadi format PDF, sering digunakan untuk menghasilkan laporan cetak dari aplikasi web.
- **enctype=multipart/form-data**: Atribut krusial pada tag <form> HTML yang menginstruksikan browser untuk mengirimkan data form, termasuk file biner, dalam format multipart ke server.
- **Entity**: Objek PHP yang merepresentasikan satu baris data dari tabel database, lengkap dengan properti dan metode yang terkait, memungkinkan abstraksi dan validasi data.
- **Fetch API**: Antarmuka modern berbasis Promise di JavaScript untuk melakukan permintaan jaringan (misalnya, HTTP requests) secara asinkron, menggantikan XMLHttpRequest tradisional dengan sintaks yang lebih kuat dan fleksibel.
- **File .env**: File konfigurasi lingkungan yang berisi pasangan kunci-nilai untuk variabel lingkungan, digunakan untuk memisahkan konfigurasi sensitif dan memungkinkan fleksibilitas antar lingkungan pengembangan dan produksi.
- **Filters**: Kelas yang dapat memodifikasi permintaan (request) sebelum mencapai controller atau respons (response) sebelum dikirim ke browser, sering digunakan untuk autentikasi, otorisasi, atau logging.
- **Form Helper**: Kumpulan fungsi yang disediakan oleh CodeIgniter 4 untuk mempermudah pembuatan elemen formulir HTML secara terprogram, mengurangi penulisan kode manual.
- **Formulir Input**: Elemen antarmuka pengguna dalam aplikasi web yang memungkinkan pengguna memasukkan atau memilih data untuk dikirimkan ke server.
- **Framework**: Kerangka kerja perangkat lunak yang menyediakan struktur dasar, pustaka kode, alat, dan praktik terbaik untuk mempercepat dan menstandarisasi pengembangan aplikasi web.
- **Grafik Batang (Bar Chart)**: Jenis visualisasi data yang menggunakan batang persegi panjang untuk membandingkan kategori diskrit atau menunjukkan perubahan data dari waktu ke waktu.
- **Gzip**: Metode kompresi data yang digunakan oleh *web server* untuk mengurangi ukuran file yang ditransfer ke *browser* klien, sehingga mempercepat waktu muat halaman.
- **Hard Delete**: Proses penghapusan data secara permanen dari database, sehingga data tidak dapat dipulihkan kembali kecuali dari cadangan.
- **Hashing Password**: Teknik mengubah *password* asli menjadi *string* dengan panjang tetap (disebut *hash*) yang tidak dapat direkonstruksi kembali, digunakan untuk menyimpan *password* secara aman di database.
- **HTML Murni**: Pendekatan pembuatan formulir input dengan menuliskan seluruh elemen formulir menggunakan sintaks HTML standar secara eksplisit.
- **Indexing**: Teknik optimasi database yang mempercepat pengambilan data dengan membuat struktur data terpisah yang menyimpan nilai kolom dan pointer ke lokasi fisik baris data.

- **Informasi**: Data yang telah diproses, diorganisir, distrukturkan, dan disajikan dalam konteks yang memberikan makna serta relevansi untuk mendukung pengambilan keputusan.
- **JSON (JavaScript Object Notation)**: Format pertukaran data yang ringan dan mudah dibaca manusia, sering digunakan dalam aplikasi web modern sebagai alternatif XML untuk mengirim dan menerima data antara server dan klien.
- **Kredensial**: Informasi yang digunakan untuk memverifikasi identitas pengguna, biasanya berupa *username* atau *email* dan *password*.
- **Kunci Asing (Foreign Key)**: Kolom dalam satu tabel yang merujuk ke kunci utama di tabel lain, digunakan untuk mendefinisikan relasi antar tabel dan menjaga integritas referensial.
- **Kunci Utama (Primary Key)**: Kolom atau kombinasi kolom yang secara unik mengidentifikasi setiap baris dalam sebuah tabel, tidak boleh berisi nilai NULL dan harus unik.
- **Laporan Manajerial**: Jenis laporan yang menyajikan ringkasan data operasional untuk mendukung manajer dalam pengambilan keputusan taktis, biasanya bersifat periodik.
- **Laporan Operasional**: Jenis laporan yang berfokus pada aktivitas harian dan detail transaksi, seringkali dibutuhkan secara real-time atau harian untuk memantau kegiatan operasional.
- **Laporan Strategis**: Jenis laporan yang memberikan gambaran besar tentang kinerja organisasi dalam jangka panjang, mendukung perencanaan strategis dan seringkali bersifat tahunan.
- **Layout**: Struktur halaman utama yang mendefinisikan bagian-bagian konsisten seperti header dan footer, tempat konten spesifik dari setiap View akan disuntikkan untuk efisiensi dan konsistensi desain.
- **Lingkungan Produksi**: Konfigurasi server dan perangkat lunak tempat aplikasi berjalan secara *live* dan diakses oleh pengguna, dengan prioritas pada performa, keamanan, dan stabilitas.
- **Middleware (Filters)**: Mekanisme dalam CodeIgniter 4 untuk mengontrol alur *request* HTTP sebelum mencapai *controller* atau setelah *controller* selesai memproses *request*, ideal untuk autentikasi dan otorisasi.
- **Migrasi (Migration)**: Fitur dalam kerangka kerja pengembangan yang memungkinkan pengelolaan perubahan skema database secara terprogram melalui file PHP, memfasilitasi kontrol versi dan konsistensi.
- **Model**: Komponen dalam arsitektur MVC yang bertanggung jawab atas representasi data dan logika bisnis aplikasi, berinteraksi langsung dengan database atau sumber data lainnya untuk manajemen dan validasi data.
- **Normalisasi Database**: Proses pengorganisasian struktur tabel dalam database untuk mengurangi redundansi data dan meningkatkan integritas data.
- **old()**: Fungsi bawaan CodeIgniter 4 yang digunakan untuk mempertahankan nilai input sebelumnya pada formulir jika terjadi kesalahan validasi, sehingga pengguna tidak perlu mengisi ulang data.
- **Pagination**: Teknik membagi kumpulan data yang besar menjadi halaman-halaman yang lebih kecil untuk meningkatkan efisiensi penyajian dan pengalaman pengguna.
- **PHPSpreadsheet**: Sebuah library PHP yang powerful untuk membaca dan menulis berbagai format file spreadsheet seperti Excel, CSV, dan ODS, memungkinkan ekspor data untuk analisis lebih lanjut.

- **Promise**: Objek di JavaScript yang merepresentasikan penyelesaian atau kegagalan operasi asinkron di masa mendatang, membantu dalam penanganan alur kontrol asinkron dan menghindari *callback hell*.
- **Relasi Antar Tabel**: Mekanisme untuk menghubungkan data di antara dua atau lebih tabel dalam database relasional, menghindari redundansi dan menjaga konsistensi data.
- **Role-Based Access Control (RBAC)**: Metode otorisasi di mana hak akses pengguna ditentukan berdasarkan peran yang mereka miliki dalam sistem, menyederhanakan manajemen izin.
- **Routing**: Mekanisme dalam aplikasi web yang menentukan bagaimana URL yang diminta oleh pengguna dipetakan ke controller dan method yang sesuai dalam aplikasi.
- **Seeder**: Fitur dalam kerangka kerja pengembangan yang digunakan untuk mengisi database dengan data awal atau data dummy secara terprogram, berguna untuk pengujian dan demonstrasi.
- **Sesi**: Cara server untuk mengingat pengguna yang telah login di antara beberapa *request* HTTP, menyimpan informasi penting pengguna setelah autentikasi berhasil.
- **Sistem Informasi (SI)**: Sekumpulan komponen yang saling terkait dan bekerja sama untuk mengumpulkan, memproses, menyimpan, dan mendistribusikan data serta informasi guna mendukung fungsi organisasi.
- **Soft Delete**: Pendekatan penghapusan data di mana data tidak dihapus secara fisik dari database, melainkan ditandai sebagai dihapus menggunakan kolom khusus, memungkinkan pemulihan data di kemudian hari.
- **Tabel**: Unit dasar penyimpanan data dalam database relasional, merepresentasikan sebuah entitas dan terdiri dari baris (record) serta kolom (field atau attribute).
- **Tipe MIME**: Identifikasi standar untuk format data yang dikirimkan melalui internet, digunakan dalam validasi file untuk memastikan integritas dan keamanan jenis konten file yang diunggah.
- **UploadedFile**: Objek dalam CodeIgniter 4 yang merepresentasikan file yang diunggah, menyediakan metode untuk mengakses informasi file, memvalidasi, dan memindahkannya.
- **Validasi Data**: Proses pemeriksaan data yang diterima dari pengguna untuk memastikan bahwa data tersebut memenuhi kriteria atau aturan yang telah ditetapkan sebelum diproses lebih lanjut atau disimpan.
- **Validasi File**: Proses pemeriksaan file yang diunggah di sisi server untuk memastikan kesesuaian dengan kriteria yang ditetapkan, meliputi ukuran, format (tipe MIME dan ekstensi), serta aspek keamanan lainnya.
- **Validasi Input**: Proses pemeriksaan data yang dimasukkan pengguna untuk memastikan memenuhi kriteria tertentu (misalnya, tidak kosong, format benar, panjang sesuai) sebelum diproses lebih lanjut.
- **View**: Komponen dalam arsitektur MVC yang bertanggung jawab atas presentasi data kepada pengguna, merender informasi yang diterima dari controller ke dalam format antarmuka pengguna seperti HTML.
- **Visualisasi Data**: Proses penyajian data dalam format grafis atau gambar untuk memudahkan pemahaman pola, tren, dan anomali yang terkandung dalam data kompleks.

- **WRITEPATH**: Konstanta dalam CodeIgniter 4 yang merujuk pada direktori writable/ aplikasi, sering digunakan sebagai lokasi penyimpanan file yang diunggah untuk alasan keamanan dan organisasi.
- **XAMPP**: Paket perangkat lunak *cross-platform* yang menyediakan lingkungan server lokal lengkap dengan Apache, MariaDB, PHP, dan Perl, memudahkan pengembangan aplikasi web.

REFERENSI

1. Al-Ruithe, M., Al-Shahrani, H., & Al-Harbi, S. (2020). Data Retention and Deletion Policies in Cloud Computing: A Review. *International Journal of Computer Science and Network Security*, *20*(1), 101-108.
2. Apache Friends. (n.d.). *XAMPP*. Diambil dari https://www.apachefriends.org/
3. Bishop, M. (2015). *Computer Security: Art and Science* (2nd ed.). Addison-Wesley Professional.
4. Chaudhuri, S., & Dayal, U. (2020). An Overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record*, 26(1), 65-74.
5. Chen, Y., Li, J., & Wang, H. (2022). *Security Vulnerabilities in Web Application Deployment: A Review*. Journal of Cybersecurity and Information Assurance, 7(1), 45-58.
6. CodeIgniter. (n.d.). *CodeIgniter 4 User Guide*. Diambil dari https://codeigniter.com/user_guide/
7. CodeIgniter. (2023). *CodeIgniter 4 User Guide*.
8. CodeIgniter 4 User Guide. (2023). *Form Helper*.
9. CodeIgniter 4 User Guide. (2023). *Security Helper*.
10. CodeIgniter 4 User Guide. (2023). *Validation*.
11. CodeIgniter 4 User Guide. (2023). *Working with Uploaded Files*.
12. CodeIgniter Documentation. (n.d.). *Logging*.
13. CodeIgniter Documentation. (n.d.). *Security*.
14. CodeIgniter Documentation. (n.d.). *Validation*.
15. Composer. (n.d.). *Composer: Dependency Manager for PHP*. Diambil dari https://getcomposer.org/
16. Connolly, T., & Begg, C. (2015). *Database Systems: A Practical Approach to Design, Implementation, and Management* (6th ed.). Pearson.
17. Date, C. J. (2004). *An Introduction to Database Systems* (8th ed.). Addison-Wesley.
18. Datatables. (n.d.). *Server-side processing*.
19. Davenport, T. H., & Harris, J. G. (2017). *Competing on Analytics: The New Science of Winning*. Harvard Business Review Press.
20. Docker. (n.d.). *Docker: Accelerated Container Application Development*. Diambil dari https://www.docker.com/
21. Elmasri, R., & Navathe, S. B. (2022). *Fundamentals of Database Systems* (8th ed.). Pearson.
22. Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R., & Chandramouli, R. (2001). Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3), 224-274.
23. Few, S. (2009). *Now You See It: Simple Visualization Techniques for Quantitative Analysis*. Analytics Press.
24. Fowler, M. (2003). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.
25. Freeman, E., & Robson, E. (2019). *Head First Design Patterns: A Brain-Friendly Guide*. O'Reilly Media.
26. Gupta, A., & Sharma, R. (2022). *Database Migration Strategies for Cloud-Based Applications*. International Journal of Database Management Systems, 14(3), 1-15.
27. Knaflic, C. N. (2015). *Storytelling with Data: A Data Visualization Guide for Business Professionals*. Wiley.

28. Krasner, G. E., & Pope, S. T. (1988). A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, *1*(3), 26-49.
29. Kroenke, D. M., & Boyle, R. J. (2019). *Experiencing MIS*. Pearson.
30. Kumar, S., & Singh, P. (2021). *Best Practices for Securing Web Applications in Production Environments*. Journal of Information Security and Applications, 60, 102876.
31. Kurose, J. F., & Ross, K. W. (2017). *Computer Networking: A Top-Down Approach* (7th ed.). Pearson.
32. Laragon. (n.d.). *Laragon: A portable, isolated, fast & powerful universal development environment*. Diambil dari https://laragon.org/
33. Laudon, K. C., & Laudon, J. P. (2020). *Management Information Systems: Managing the Digital Firm*. Pearson.
34. Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.
35. MDN Web Docs. (n.d.). *Using the Fetch API*.
36. O'Brien, J. A., & Marakas, G. M. (2011). *Management Information Systems*. McGraw-Hill/Irwin.
37. OWASP. (2023). *Input Validation Cheat Sheet*.
38. OWASP. (2023). *SQL Injection*.
39. OWASP. (2023). *Unrestricted File Upload*.
40. Patel, D., & Shah, M. (2020). *Performance Optimization Techniques for Modern Web Applications*. International Journal of Web Engineering and Technology, 15(2), 112-125.
41. PHP Manual. (n.d.). *password_hash*.
42. PHPSpreadsheet. (n.d.). *PHPSpreadsheet Documentation*.
43. Pratama, I. (2020). *Framework CodeIgniter 4: Konsep dan Implementasi*. Penerbit Informatika.
44. Pressman, R. S., & Maxim, B. R. (2020). *Software Engineering: A Practitioner's Approach* (9th ed.). McGraw-Hill Education.
45. Rahman, M. M., & Islam, M. R. (2019). *Environment Variable Management in Modern Web Development Frameworks*. Journal of Software Engineering and Applications, 12(4), 101-110.
46. Rahman, M. M., & Islam, M. R. (2020). A Comparative Study of MVC Frameworks for Web Application Development. *International Journal of Computer Applications*, *175*(1), 1-6.
47. Ramakrishnan, R., & Gehrke, J. (2018). *Database Management Systems* (3rd ed.). McGraw-Hill Education.
48. Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). Role-Based Access Control Models. *IEEE Computer*, *29*(2), 38-47.
49. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). *Database System Concepts* (7th ed.). McGraw-Hill Education.
50. Smith, J., & Jones, A. (2021). *From Development to Production: A Guide to Web Application Deployment*. TechPress Publishing.
51. Snyder, J., & Seidl, C. (2020). *PHP 7 Programming Cookbook*. Packt Publishing.

52. Suryani, A., & Hidayat, R. (2023). Implementasi Model-View-Controller (MVC) pada Pengembangan Aplikasi Web E-Commerce Menggunakan Framework CodeIgniter 4. *Jurnal Sistem Informasi dan Teknologi Informasi*, 12(1), 45-56.
53. Suryani, E., Hidayat, R., & Ramadhan, A. (2021). Implementasi Konsep CRUD (Create, Read, Update, Delete) pada Aplikasi Web E-Commerce Menggunakan Framework CodeIgniter. *Jurnal Sistem Informasi dan Teknologi Informasi*, *10*(1), 1-10.
54. Suryanarayana, G., Rao, S. V., & Kumar, M. S. (2021). A Comparative Study of PDF Generation Libraries in PHP. *International Journal of Engineering Research & Technology (IJERT)*, 10(05), 100-104.
55. Welling, L., & Thomson, L. (2019). *PHP and MySQL Web Development* (5th ed.). Addison-Wesley Professional.

SISTEM INFORMASI WEB DINAMIS: CRUD, Reporting dan Grafiis dengan CodeIgniter 4

MOH MUTHOHIR, S..Kom. , M.Kom

BIODATA PENULIS

Penulis merupakan praktisi dan pendidik di bidang Teknik Informatika, yang sejak awal kariernya berkomitmen mengembangkan solusi digital dan ekosistem teknologi yang relevan dengan kebutuhan masa kini. Berpengalaman mengajar di Universitas Sains dan Teknologi Komputer Semarang serta terlibat dalam proyek-proyek pengembangan web, ia dikenal sebagai sosok yang inovatif dan selalu berusaha menyederhanakan konsep teknik yang kompleks agar mudah dipahami dan aplikatif.

Passion besar pada dunia rekayasa perangkat lunak dan pendidikan membuat Penulis aktif menulis modul, buku, dan artikel edukatif yang sudah banyak digunakan oleh mahasiswa dan praktisi di tanah air. Melalui buku ini, ia ingin membagikan pengetahuan praktis tentang penerapan CodeIgniter 4, khususnya membangun sistem informasi web dinamis yang modern serta mendukung fitur CRUD (*Create, Read, Update, Delete*), pelaporan data, dan grafis interaktif.

Keterampilan mengajarnya di kelas maupun dalam pelatihan daring terbukti efektif membantu banyak pembaca memahami alur logika pemrograman, mulai dari konsep dasar hingga implementasi langsung dalam studi kasus nyata. Penulis percaya bahwa teknologi harus menjadi alat kolaboratif untuk menciptakan kemajuan, bukan sekadar alat, dan setiap orang punya kesempatan untuk menjadi bagian penting dalam transformasi digital.

Di luar aktivitas akademik, Penulis juga aktif membangun komunitas pembelajaran, menghadirkan seminar, dan berbagi inspirasi melalui media sosial. Ia berharap buku "Sistem Informasi Web Dinamis: *CRUD, Reporting* dan Grafis dengan *CodeIgniter 4*" ini dapat menjadi referensi utama yang inspiratif sekaligus aplikatif bagi para pelajar, pendidik, dan pegiat TI Indonesia

YAYASAN PRIMA AGUS TEKNIK

PENERBIT :
YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-634-7227-72-0 (PDF)
9 786347 227720