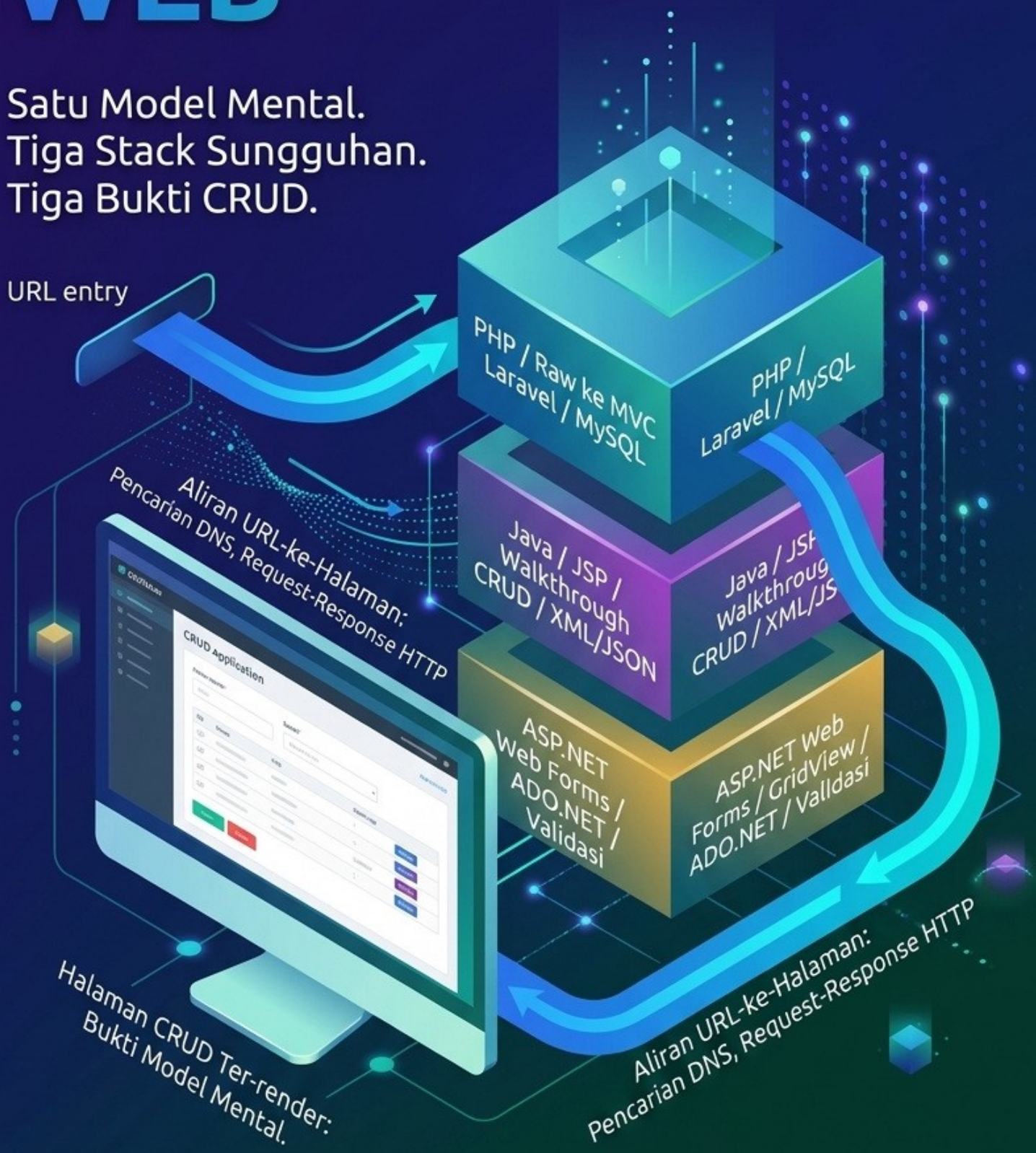
PEMROGRAMAN WEB

Edisi Pertama

Satu Model Mental.
Tiga Stack Sungguhan.
Tiga Bukti CRUD.



Irfan Subakti

Profitina
— profitina.com

DAFTAR ISI

Kata Pengantar

Buku ini tumbuh dari mata kuliah Pemrograman Web yang saya ampu di Departemen Teknik Informatika, Institut Teknologi Sepuluh Nopember (ITS) Surabaya, dan disusun sebagaimana kuliahnya disusun: 16 bab, masing-masing satu sesi yang bisa Anda ikuti, kerjakan, lalu pakai.

Sebelum menulis satu baris PHP, JSP, atau C#, buku ini menanamkan satu model mental yang jelas di setiap bab: apa yang sesungguhnya terjadi antara mengetik sebuah URL dan halaman yang tampil di layar — lalu membuktikan model itu dengan membangun aplikasi CRUD yang sama sebanyak tiga kali, satu untuk tiap stack.

Setibanya Anda di halaman terakhir, Anda semestinya mampu:

- Menjelaskan secara presisi apa yang terjadi antara mengetik URL dan halaman yang ter-render — resolusi DNS, siklus request-response HTTP, tanggung jawab client vs server.
- Menyusun konten dengan HTML serta menata dan memberi interaktivitas dengan CSS dan JavaScript.
- Membangun aplikasi CRUD PHP + MySQL yang benar-benar berjalan, dan memahami mengapa framework MVC seperti Laravel diperlukan.
- Membangun jenis aplikasi yang sama sekali lagi dengan Java JSP, termasuk bekerja dengan XML dan JSON.
- Membangunnya ketiga kalinya dengan ASP.NET Web Forms, memakai GridView, ADO.NET, validasi, dan state management.
- Membandingkan tiga stack web sungguhan karena benar-benar sudah membangun masalah yang sama di masing-masing, bukan sekadar membaca tentangnya.

Contoh berjalan di buku ini adalah Profitina Laundry — aplikasi usaha laundry yang kecil namun utuh, dibangun sekali per buku, dari folder kosong sampai benar-benar jalan. Cakupannya sengaja sederhana dan detailnya sengaja jujur: validasi yang sama, akses basis data yang sama, dan pertanyaan penerapan yang sama seperti yang harus dijawab aplikasi sungguhan yang dipakai pelanggan.

Untuk siapa buku ini ditulis: mahasiswa pemrograman web yang mengambil kuliah pertama tentang membangun aplikasi web berbasis basis data, serta pembelajar mandiri yang ingin tur praktis dari nol atas PHP, JSP, dan ASP.NET — bukan tutorial satu framework saja.

Sepatah kata tentang metode. Setiap contoh kode dalam buku ini dikompilasi dan dijalankan sebelum dituliskan, dan lampirannya memuat catatan verifikasi sebagai buktinya. Ketika sebuah angka muncul, angka itu berasal dari pengukuran, bukan dari ingatan. Bila Anda menemukan kekeliruan, atau bagian yang bisa lebih jelas, saya sungguh ingin mendengarnya — begitulah edisi berikutnya menjadi lebih baik.

Surabaya, September 2026

Irfan Subakti
yifana@gmail.com

Profitina
profitina.com

BAB 1

Fondasi Web: Bagaimana Halaman Web Benar-Benar Sampai ke Layar Anda

Sebelum menulis satu baris pun HTML, PHP, JSP, atau C#, setiap web programmer perlu model mental yang jelas tentang apa yang sebenarnya terjadi antara mengetik sebuah URL dan melihat halaman muncul.

Tujuan Pembelajaran — setelah bab ini Anda akan mampu:

(1) Membedakan Internet dari Web, dan menjelaskan kontribusi tiap lapisan pada satu kali tampilan halaman. (2) Menjelaskan siklus request-response HTTP secara menyeluruh, dan menyebutkan tiap dari lima langkahnya. (3) Membandingkan peran peramban web, server web, dan basis data dalam aplikasi web yang tipikal. (4) Menelusuri evolusi historis Web dari 1.0 hingga 5.0, dan mengenali era mana yang menjadi tempat suatu teknologi tertentu. (5) Mengidentifikasi tiga tumpukan (stack) sisi-server yang dicakup mata kuliah ini — PHP/MySQL, JSP, dan ASP.NET/ADO.NET — dan menjelaskan, secara garis besar, bagaimana masing-masing berperan dalam siklus request-response.

Satu Toolchain, Tiga Sistem Operasi — Windows 11, macOS, Ubuntu

OS	Server web lokal (PHP)	Editor
Windows 11	XAMPP for Windows (apachefriends.org) atau php -S localhost:8000	VS Code + ekstensi PHP/HTML
macOS	brew install php lalu php -S localhost:8000	VS Code + ekstensi PHP/HTML
Ubuntu/Linux	sudo apt install php lalu php -S localhost:8000	VS Code + ekstensi PHP/HTML

Semua contoh di buku ini berjalan identik di Windows 11, macOS, dan Ubuntu/Linux; perintah hanya berbeda saat instalasi. Pilih baris Anda sekali dan ikuti terus di laptop Anda sendiri.

1.1 Selamat Datang di Pemrograman Web

Selamat datang di mata kuliah Pemrograman Web (WebPro). Mata kuliah ini bukan tentang menghafal sintaks satu bahasa saja. Ini tentang memahami satu pola yang berulang — client bertanya, server menjawab — lalu belajar mengimplementasikan pola itu memakai tiga dari tumpukan teknologi sisi-server yang paling luas dipakai di industri: PHP dengan MySQL, Java Server Pages (JSP) dengan servlet container Java, dan ASP.NET dengan ADO.NET. Setiap teknologi yang akan Anda temui di lima belas lecture berikutnya, di balik sintaksnya masing-masing, sebenarnya mengerjakan hal yang sama persis: menerima permintaan HTTP, memutuskan artinya, mengambil atau mengubah data, dan mengirim kembali respons yang bisa dirender oleh peramban. Agar setiap ide tetap konkret, bukan abstrak, buku ini dibangun menuju satu contoh berjalan yang akan Anda temui secara formal di Bagian 1.7 dan lagi di Lecture 5–16: Profitina Laundry, sebuah bisnis jasa laundry nyata yang sistem pemesanan-dan-penagihannya akan Anda rancang, lalu implementasikan tiga kali terpisah, satu kali per stack.

Pada akhir mata kuliah ini Anda akan mampu merancang basis data relasional untuk masalah bisnis nyata, membangun aplikasi web CRUD (Create, Read, Update, Delete) lengkap terhadap basis data itu di lebih dari satu teknologi sisi-server, dan menjelaskan — secara presisi, bukan sekadar perasaan — apa yang terjadi di setiap tahap antara klik seorang pengguna dan layar yang berubah sebagai responsnya.

Mengapa tiga stack, bukan satu?

Akan lebih cepat kalau hanya mengajarkan satu framework. Tapi organisasi rekayasa perangkat lunak nyata berjalan di atas campuran sistem PHP lama, backend enterprise Java, dan layanan .NET — sering ketiganya ada di dalam perusahaan yang sama. Mempelajari pola request-response yang mendasarinya sekali saja, lalu melihatnya diimplementasikan dengan tiga cara berbeda, adalah hal yang memungkinkan Anda mempelajari framework keempat atau kelima dengan cepat nanti dalam karier Anda — karena Anda sudah mengenali pola di balik sintaks barunya.

1.2 Internet vs. Web — Dua Hal yang Berbeda

Dalam percakapan sehari-hari orang menyebut “Internet” dan “Web” secara bertukar-pakai, tapi bagi seorang web programmer perbedaannya penting. Internet adalah infrastruktur fisik dan logis global — jutaan router, kabel, tautan satelit yang saling terhubung, dan rangkaian protokol TCP/IP yang memungkinkan dua mesin mana pun yang terhubung bertukar paket data, di mana pun mereka berada di dunia. Internet sudah ada, dan sudah membawa lalu lintas data, sebelum Web ada: email, transfer berkas (FTP), dan akses terminal jarak jauh (Telnet) semuanya sudah berjalan di atas Internet pada tahun 1970-an dan 1980-an.

Web — singkatan dari World Wide Web — adalah satu aplikasi tertentu yang berjalan di atas Internet. Web ditemukan pada 1989–1991 oleh Tim Berners-Lee di CERN, dan terdiri dari tiga unsur yang bekerja bersama: HTML (format dokumen untuk menyusun konten dan menautkan antar-dokumen), HTTP (protokol transfer untuk meminta dan mengirim dokumen-dokumen itu), dan URL (skema penamaan untuk menemukan dokumen mana pun, di mana pun, secara unik). Setiap teknologi yang dicakup mata kuliah ini — HTML, CSS, JavaScript, PHP, JSP, ASP.NET — berada di dalam satu lapisan aplikasi ini, berjalan di atas lapisan-lapisan Internet yang lebih rendah.

Analogi pos surat

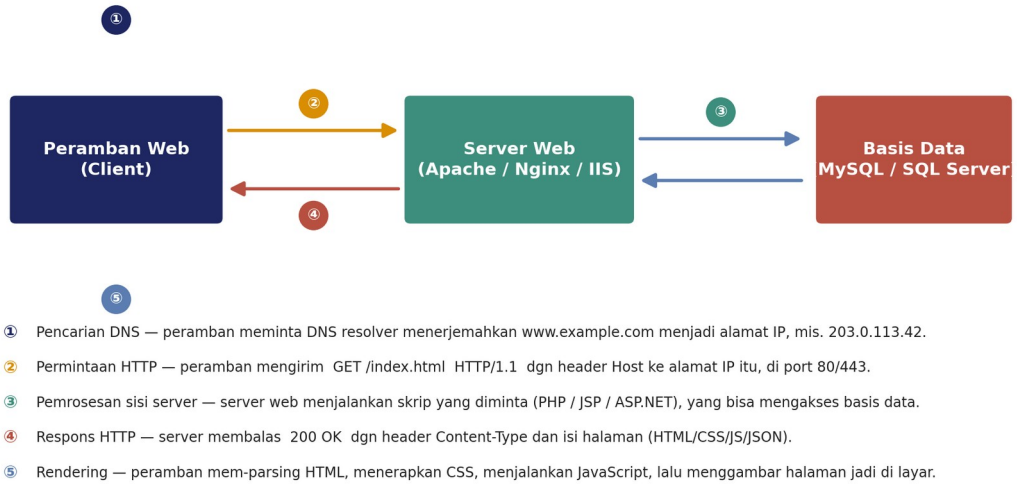
Internet seperti jaringan pos dan jalan raya sedunia — truk, kapal, depo penyortiran, dan konvensi pengalamatan yang bisa memindahkan sebuah amplop dari kota mana pun ke kota lain mana pun. Web seperti satu jenis surat tertentu — format amplop tertentu (HTML), formulir permintaan pengiriman tertentu yang Anda isi (HTTP), dan skema pengalamatan tertentu khusus untuk surat itu saja (URL). Email dan layanan berbagi berkas adalah jenis “surat” lain yang memakai jaringan dasar yang sama tapi format amplop dan aturan yang berbeda.

1.3 Model Client-Server dan Siklus Request-Response HTTP

Hampir semua yang Anda bangun di mata kuliah ini mengikuti model client-server: sebuah client (biasanya peramban web) memulai permintaan, dan sebuah server (program yang mendengarkan koneksi, biasanya di port 80 untuk HTTP biasa atau port 443 untuk HTTPS terenkripsi) memproses permintaan itu dan mengirim kembali respons. Ini adalah pola request-response — server tidak pernah mendorong sebuah halaman kepada Anda tanpa diminta; ia hanya pernah membalas permintaan yang Anda (atau peramban Anda) buat lebih dulu.

Model Client-Server dan Siklus Request-Response HTTP

Setiap kali halaman web biasa dibuka, terjadi satu putaran penuh siklus ini — permintaan keluar, jawaban kembali.



Gambar 1.1 — Siklus request-response HTTP: pencarian DNS menerjemahkan nama domain, peramban mengirim permintaan HTTP, server (opsional berkonsultasi ke basis data) memprosesnya, dan respons HTTP dirender oleh peramban.

Langkah ① terjadi lebih dulu dan terpisah dari langkah lainnya: sebelum peramban bahkan bisa menghubungi www.example.com, sistem operasi meminta sebuah DNS (Domain Name System) resolver menerjemahkan nama domain yang mudah dibaca manusia itu menjadi alamat IP numerik, misalnya 203.0.113.42 — router hanya tahu cara memindahkan paket menuju alamat IP, bukan menuju nama. Langkah ②–④ adalah pertukaran HTTP yang sebenarnya: peramban membuka koneksi ke alamat IP itu dan mengirim baris permintaan seperti `GET /index.html HTTP/1.1` bersama header `Host:` (diperlukan karena satu alamat IP umumnya menampung banyak domain berbeda), perangkat lunak server memutuskan cara menjawab — untuk berkas statis ia cukup membaca berkas itu dari disk, untuk halaman dinamis ia menjalankan skrip PHP, JSP, atau ASP.NET, yang bisa saja mengakses basis data — dan server membalas dengan baris status seperti `HTTP/1.1 200 OK`, sekumpulan header (termasuk `Content-Type`, memberi tahu peramban cara menafsirkan yang mengikutinya), dan sebuah body berisi konten sesungguhnya. Langkah ⑤ sepenuhnya adalah pekerjaan peramban sendiri: mem-parsing HTML menjadi sebuah pohon elemen (DOM), menerapkan aturan CSS untuk memutuskan tampilan tiap elemen, menjalankan JavaScript apa pun, dan akhirnya menggambar piksel di layar. Kelima langkah ini terjadi, tanpa terlihat mewah, setiap kali pelanggan sungguhan membuka halaman pencarian tagihan Profitina Laundry atau situs pemasaran Profitina sendiri — tidak ada mekanisme terpisah yang lebih sederhana yang disediakan khusus untuk lalu lintas produksi "sungguhan".

HTTP bersifat stateless

Secara desain, HTTP memperlakukan setiap permintaan sebagai sepenuhnya independen dari permintaan lainnya — protokol itu sendiri tidak punya memori bawaan tentang permintaan sebelumnya dari peramban yang sama. Ini disebut stateless. Ini adalah trade-off kesederhanaan/skalabilitas yang disengaja: sebuah server stateless bisa di-restart, di-load-balance di banyak mesin, atau diskalakan secara horizontal tanpa perlu melacak siapa yang "masih terhubung", karena sebenarnya tak seorang pun benar-benar tetap terhubung antar-permintaan. Konsekuensinya, aplikasi web mana pun yang perlu mengingat sesuatu tentang pengunjung tertentu — bahwa mereka sudah login, atau apa isi keranjang belanja mereka — harus menambahkan memori itu kembali secara sengaja, memakai cookie dan session. Anda akan mengimplementasikan persis ini di Lecture 6.

1.3.1 Metode HTTP yang Paling Umum

Kata pertama pada baris permintaan — metode HTTP — memberi tahu server jenis aksi apa yang diinginkan client. Mata kuliah ini memakai empat metode berulang kali, dan keempatnya berpadanan alami dengan empat operasi CRUD yang akan Anda implementasikan di setiap stack.

Metode	Tujuan tipikal	Operasi CRUD	Safe / Idempotent?
GET	Mengambil sebuah resource (halaman, record, daftar)	Read	Safe & idempotent
POST	Mengirim data untuk membuat resource baru, atau memicu suatu aksi	Create	Bukan keduanya
PUT	Mengganti seluruh isi resource dengan data yang dikirim	Update	Idempotent, bukan safe
DELETE	Menghapus sebuah resource	Delete	Idempotent, bukan safe

"Safe" berarti metode itu tidak boleh mengubah state server (permintaan GET seharusnya tidak pernah menghapus data), dan "idempotent" berarti mengirim permintaan yang identik dua kali punya efek yang sama dengan mengirimnya satu kali (menghapus record yang sama dua kali menghasilkan state akhir yang sama dengan menghapusnya sekali). Ini adalah konvensi yang diharapkan standar HTTP dipatuhi oleh setiap aplikasi yang berperilaku baik, bukan sesuatu yang ditegakkan otomatis oleh protokolnya — tidak ada yang mencegah program yang ceroboh melakukan DELETE di dalam handler GET, tapi melakukan itu merusak caching, merusak perilaku "back/forward"/prefetch peramban, dan dianggap bug desain yang serius.

1.3.2 Kode Status HTTP yang Paling Umum

Kode status pada respons adalah angka tiga digit yang memberi tahu client, sekilas pandang, apa yang terjadi. Kode status dikelompokkan menjadi lima keluarga berdasarkan digit pertamanya.

Kode	Arti	Kapan Anda akan melihatnya
200 OK	Berhasil	Setiap pemuatan halaman atau panggilan API yang berhasil biasa.
301 / 302	Redirect (permanen / sementara)	Setelah login berhasil, `header("Location: ...")` di PHP.
400 Bad Request	Client mengirim data yang cacat format	Formulir dikirim dengan field wajib yang kosong atau salah ketik.
401 Unauthorized	Autentikasi diperlukan	Mengakses halaman terproteksi saat belum login.
403 Forbidden	Sudah terautentikasi, tapi tidak diizinkan	Pengguna non-admin yang sudah login meminta halaman khusus admin.
404 Not Found	Tidak ada resource di URL ini	URL yang salah ketik, atau halaman detail record yang sudah dihapus.
500 Internal Server Error	Kode sisi-server crash	Exception PHP/JSP/C# yang tidak tertangkap saat menangani permintaan.

1.4 Server Web: Perangkat Lunak yang Sungguh-Sungguh Menjawab Permintaan

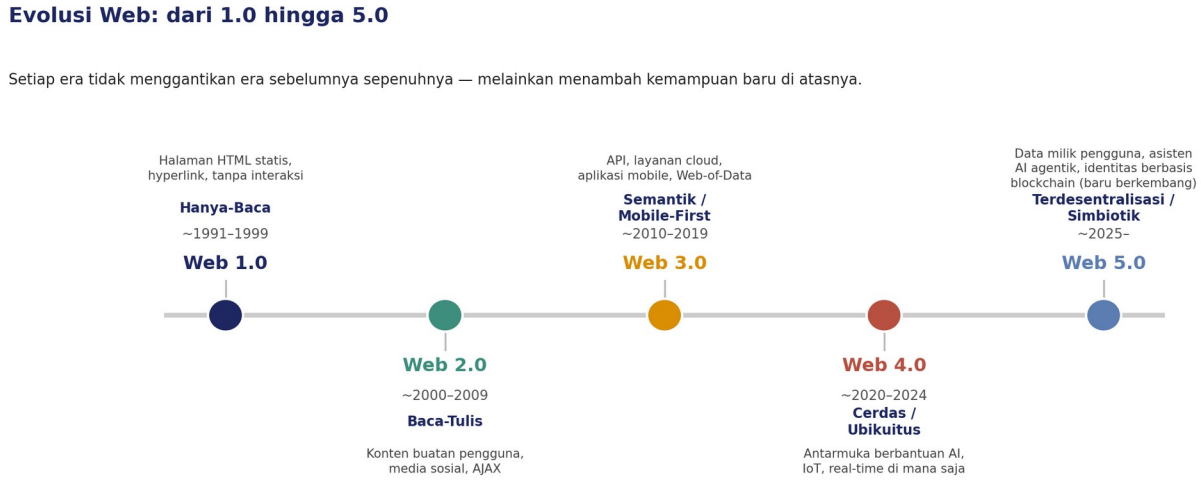
Server web adalah proses perangkat lunak yang mendengarkan di sebuah port jaringan, menerima koneksi HTTP yang masuk, dan memutuskan cara menjawab tiap permintaan — dengan menyajikan berkas statis langsung dari disk, atau dengan menyerahkan permintaan itu ke sebuah runtime bahasa (PHP, servlet container Java, pipeline ASP.NET) yang menghasilkan respons dinamis. Ketiga jalur mata kuliah ini masing-masing berpadanan alami dengan server tertentu: PHP paling umum dilayani oleh Apache atau Nginx (sering dipaketkan bersama MySQL dan PHP itu sendiri menjadi stack pengembangan “XAMPP” atau “LAMP/LEMP” yang akan Anda instal di Lecture 5–6); JSP memerlukan servlet container Java seperti Apache Tomcat; dan ASP.NET dilayani oleh IIS (Internet Information Services) di Windows, atau oleh server Kestrel lintas-platform yang dipaketkan bersama .NET modern.

Pangsa pasar server web saat ini (W3Techs, Agustus 2026)

Di seluruh situs web yang disurvei W3Techs, Nginx memimpin dengan pangsa pasar sekitar 31%, server milik Cloudflare sendiri hampir menyusul di sekitar 30%, Apache memegang sekitar 23%, dan LiteSpeed telah tumbuh menjadi sekitar 15% (perlu dicatat banyak situs melaporkan lebih dari satu server dalam stack-nya, sehingga angka-angka ini tidak berjumlah 100%). Microsoft-IIS, server ASP.NET mata kuliah ini, berada di angka yang sederhana yaitu 3%, mencerminkan betapa besarnya bagian Web publik modern yang di-hosting di Linux dibanding Windows — tapi IIS (dan penerus lintas-platformnya, Kestrel) tetap menjadi standar di dalam banyak sekali perusahaan yang menjalankan aplikasi bisnis internal berbasis ASP.NET, yaitu persis jenis sistem yang dipersiapkan oleh Lecture 13–15 untuk Anda bangun.

1.5 Evolusi Web: dari Halaman Statis Menuju Web yang Cerdas dan Dimiliki Pengguna

Web yang Anda pakai hari ini terlihat sama sekali berbeda dari Web tahun 1995, meskipun fondasi HTTP/HTML/URL yang mendasarinya dari Bagian 1.2 belum berubah secara fundamental. Yang berubah adalah apa yang orang bangun di atas fondasi itu, dan tiap era umumnya diberi label bergaya versi — memahami label-label ini membantu Anda menempatkan teknologi apa pun yang Anda temui, di mata kuliah ini maupun sesudahnya, pada konteks historis dan teknis yang tepat (Gambar 1.2).



Gambar 1.2 — Lima era Web yang umum diakui. Tiap era menambahkan kemampuan baru di atas era sebelumnya, bukan menggantikannya sepenuhnya.

Web 1.0 (sekitar 1991–1999) bersifat "hanya-baca": penulis dan organisasi individual menerbitkan halaman HTML statis, terhubung lewat hyperlink, dengan pada dasarnya nol interaktivitas sisi-server — inilah Web yang paling langsung direkonstruksi oleh materi Lecture 2 (HTML murni) Anda. Web 2.0 (sekitar 2000–2009) menjadi "baca-tulis": teknologi seperti AJAX memungkinkan JavaScript memperbarui sebagian halaman tanpa memuat ulang penuh, dan platform yang dibangun di sekitar konten buatan-pengguna — blog, wiki, dan media sosial pertama — meledak, semuanya bergantung persis pada pemrosesan sisi-server (PHP, di mata kuliah ini) yang dicakup mulai Lecture 3. Web 3.0 (sekitar 2010–2019) biasanya dicirikan sebagai Web yang digerakkan-API, mobile-first, dan semantik: aplikasi berhenti menjadi satu halaman web monolitik dan menjadi komposisi banyak layanan yang saling berbicara lewat API, sering dikonsumsi oleh aplikasi mobile native, bukan hanya oleh peramban — jalur JSP dan ASP.NET di Lecture 9–15 sama-sama mencakup pembangunan jenis lapisan akses-data yang menjadi dasar API semacam itu. Web 4.0 (sekitar 2020–2024) sering dideskripsikan sebagai cerdas dan ubikuitus — antarmuka berbantuan AI, Internet of Things (IoT) penuh perangkat yang selalu terhubung, dan ekspektasi respons real-time di mana saja. Web 5.0 (mulai sekitar 2025 dan seterusnya) adalah label terbaru dan paling belum-mapan, umumnya dipakai untuk mendeskripsikan Web "simbiotik" yang terdesentralisasi dan sedang berkembang, di mana pengguna dimaksudkan untuk memiliki dan mengontrol data mereka sendiri, agen AI bertindak atas nama pengguna lintas layanan, dan sistem identitas berbasis blockchain dieksplorasi sebagai alternatif dari login yang dikelola terpusat seperti sekarang — era ini masih terbentuk, dan sebagian besarnya masih lebih berupa aspirasi ketimbang praktik standar yang benar-benar diterapkan per 2026.

Jangan terlalu percaya pada label versi

Berbeda dari nomor versi perangkat lunak, “Web 3.0” atau “Web 4.0” tidak pernah diratifikasi oleh satu badan standar dengan definisi teknis yang presisi — penulis yang berbeda menggambar batas era sedikit berbeda-beda, dan Anda akan menemukan sumber yang tidak sepakat soal tahun pastinya atau apa yang termasuk era yang mana. Perlakukan label-label ini seperti Anda memperlakukan “era 1980-an” dalam sejarah musik: singkatan yang berguna untuk sekelompok tren yang sebagian besar terjadi di waktu yang sama, bukan spesifikasi teknis yang presisi. Yang TIDAK diperdebatkan adalah model HTTP/HTML/URL yang mendasarinya dari Bagian 1.2 — itu tetap stabil secara arsitektural sejak 1991, yang justru menjadi alasan mengapa mata kuliah yang mengajarkannya hari ini belum juga usang.

1.6 Bahasa Pemrograman untuk Web: Sebuah Potret

Web programmer baru sering terkejut bahwa tidak ada satu pun “bahasa pemrograman web” tunggal — sisi client dan sisi server dari sebuah permintaan biasanya memakai bahasa yang sama sekali berbeda, dan bahkan di dalam sisi server sendiri, ada rentang bahasa yang sangat luas sebagai pilihan yang layak. Tabel di bawah melaporkan TIOBE Index untuk Agustus 2026, sebuah peringkat bulanan popularitas bahasa pemrograman secara keseluruhan yang banyak dikutip (meski tidak diakui secara universal) berdasarkan volume kueri mesin pencari dan kursus/vendor, dibatasi di sini hanya pada bahasa yang relevan langsung dengan mata kuliah ini.

Bahasa	Peringkat TIOBE (Agt 2026)	Di mana muncul di mata kuliah ini
Python	#1 (18,53%)	Tidak dicakup di mata kuliah ini (disebut hanya sbg konteks — DAA & MK lain memakainya/C++).
Java	#4 (8,25%)	Lecture 9–10: JSP berjalan di platform Java dan mewarisi seluruh pustaka standar & tooling-nya.
C#	#5 (4,09%)	Lecture 13–15: ASP.NET dibangun di atas C# dan runtime .NET.
JavaScript	#6 (2,63%)	Interaktivitas sisi-client, Lecture 3; satu-satunya bahasa yang berjalan langsung di setiap peramban.
SQL	#8 (1,88%)	Setiap dari tiga stack CRUD mata kuliah ini (PHP, JSP, ASP.NET) mengirim SQL ke basis data relasional.
PHP	#18 (Mar 2026, ~1,2%)	Lecture 5–8: dasar PHP, PHP+MySQL, dan PHP MVC/Laravel.

Bahasa	Peringkat TIOBE (Agt 2026)	Di mana muncul di mata kuliah ini

Dua peringkat, dua gambaran PHP yang sangat berbeda

TIOBE mengukur volume kueri mesin pencari dan kursus — kurang lebih, "seberapa banyak orang saat ini mencari tentang bahasa ini" — dan dengan ukuran itu PHP telah turun ke sekitar peringkat 18. Tapi W3Techs mengukur sesuatu yang sama sekali berbeda: bahasa sisi-server apa yang sungguh-sungguh dijalankan oleh situs web yang sedang aktif. Dengan ukuran itu, PHP masih memberi daya pada sekitar 70% dari seluruh situs web yang bahasa sisi-servernya bisa dideteksi W3Techs — jauh lebih banyak dari seluruh bahasa sisi-server lainnya digabungkan, termasuk ASP.NET yang sekitar 4% dan Java yang sekitar 6%. Tidak ada angka yang "salah"; keduanya hanya menjawab pertanyaan yang berbeda. Inilah persisnya mengapa mata kuliah ini masih mencurahkan empat lecture penuh (5–8) untuk PHP: popularitas tren-pencarian mentah dan jumlah sesungguhnya sistem yang berjalan dan menghasilkan pendapatan yang dibangun dalam suatu bahasa adalah dua hal yang berbeda, dan sebagai profesional Anda sangat mungkin akan memelihara jauh lebih banyak PHP daripada yang disarankan peringkat TIOBE-nya saja.

Catatan tentang peringkat dan data terkini

Indeks popularitas bahasa seperti TIOBE, W3Techs, Stack Overflow Developer Survey, PYPL, dan laporan Octoverse GitHub masing-masing mengukur sesuatu yang sedikit berbeda — volume pencarian, deployment situs web yang hidup, penggunaan yang dilaporkan-sendiri oleh developer, kueri tutorial, atau aktivitas repositori — dan peringkatnya bergeser dari bulan ke bulan dan tahun ke tahun. Angka-angka spesifik di atas berlaku per Agustus 2026; perlakukan peringkat apa pun yang Anda baca (di buku ini maupun di tempat lain) sebagai potret sesaat dari target yang terus bergerak, dan utamakan edisi terbaru dari indeks mana pun yang Anda konsultasikan.

1.7 Tiga Stack yang Akan Dikuasai Mata Kuliah Ini

Alih-alih mengajarkan satu framework secara sangat mendalam, mata kuliah ini secara sengaja mengajarkan masalah yang sama — membangun aplikasi web CRUD yang berfungsi, berbasis basis data — tiga kali, dalam tiga teknologi sisi-server yang berbeda, sehingga Anda mengalami sendiri betapa banyak bagian dari web programming yang bersifat universal (siklus request-response, perancangan basis data relasional, CRUD, autentikasi) dan betapa banyak yang sekadar sintaks.

- PHP + MySQL (Lecture 5–8) — stack sisi-server open-source yang paling luas dipakai di Web; Anda akan mulai dengan dasar PHP prosedural, menambahkan state berbasis session/cookie dan akses MySQL langsung, dan diakhiri dengan membangun aplikasi yang sama memakai pola MVC lewat framework Laravel.
- JSP di platform Java (Lecture 9–12) — Java Server Pages dan servlet, berjalan di dalam container seperti Apache Tomcat, memberi Anda paparan pertama pada bahasa sisi-server yang strongly-typed dan dikompilasi, serta pada pola MVC dalam setting Java enterprise.
- ASP.NET + ADO.NET (Lecture 13–16) — framework web sisi-server dari Microsoft, mencakup Web Forms, kontrol server, validasi, manajemen state, dan lapisan akses-data ADO.NET yang menghubungkannya ke basis data relasional.

Di ketiga jalur, buku ini memakai satu studi kasus yang berjalan terus — sebuah sistem manajemen jasa laundry bernama Profitina Laundry — sehingga Anda bisa langsung membandingkan bagaimana masalah bisnis yang identik (mengelola pelanggan, karyawan, layanan, dan tagihan) diselesaikan di

tiap stack. Profitina Laundry bukan nama rekaan belaka: ia dibangun dan dioperasikan di bawah brand Profitina yang sama yang akan Anda temui di Bagian 1.8, satu etalase konkret dalam keluarga produk Profitina yang lebih luas — inilah tepatnya mengapa ia menjadi contoh bersama yang pas untuk buku yang menjadikan Profitina sebagai bukti berjalannya bahwa "beginilah cara sistem nyata sungguh-sungguh dibangun". Anda akan bertemu skema basis data Profitina Laundry untuk pertama kalinya di Lecture 10, dan mengimplementasikannya di PHP (Lecture 7), JSP (Lecture 10), dan ASP.NET (Lecture 14).

1.8 WebPro dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Di sepanjang buku ini, kotak seperti ini menghubungkan materi mata kuliah dengan Profitina, produk AI-visibility (Generative Engine Optimization) asal Indonesia yang nyata, dibangun oleh penulis, dan dengan Profitina Laundry, bisnis jasa laundry nyata yang menjadi studi kasus berjalan buku ini (Bagian 1.7). Kotak-kotak ini mendeskripsikan DI MANA dan MENGAPA suatu konsep dipakai di sebuah sistem yang hidup dan berjalan — bukan detail implementasi eksklusif, logika bisnis, atau kode sumber Profitina itu sendiri, yang tetap rahasia. Lihat dokumen pendamping ‘Profitina: Tinjauan Teknis Non-Rahasia’ untuk gambaran lengkapnya.

Kehadiran web publik Profitina sendiri, di balik kilaunya, persis merupakan siklus request-response yang dideskripsikan di Bagian 1.3: peramban seorang pengunjung mengurai profitina.com lewat DNS, mengirim permintaan HTTP ke server web Profitina, dan menerima kembali halaman yang dirender yang mendeskripsikan produknya. Yang lebih penting, masalah yang justru ingin dipecahkan Profitina — seberapa terlihat sebuah bisnis ketika sistem AI menjawab pertanyaan tentangnya — sepenuhnya bergantung pada fondasi HTTP-dan-HTML Web dari Bagian 1.2: asisten AI yang merekomendasikan (atau gagal merekomendasikan) suatu bisnis, hampir di setiap kasus, pada akhirnya berakar pada halaman web yang diambil oleh sebuah crawler memakai HTTP dan diurai sebagai HTML, persis seperti peramban pada Gambar 1.1. Memahami secara presisi bagaimana sebuah halaman diminta, disajikan, dan dirender — subjek dari seluruh bab ini — adalah prasyarat untuk memahami bagaimana sebuah halaman menjadi terlihat bagi sistem AI sama sekali, yaitu lapisan tempat Profitina beroperasi. Profitina Laundry berada satu lapisan di bawahnya: ia adalah bisnis kecil yang biasa, yang visibilitas, pemesanan, dan penagihannya sendiri bergantung persis pada aplikasi web CRUD yang akan Anda habiskan Lecture 5–16 untuk membangunnya — pengingat sehari-hari bahwa fondasi Web yang Anda pelajari di bab ini bukan latihan akademis semata, melainkan mekanisme yang sama yang menjaga pintu, dan pembukuan, sebuah bisnis laundry nyata tetap berjalan.

1.9 Ringkasan Bab dan Poin-Poin Kunci

- Internet adalah jaringan global; Web adalah satu aplikasi (HTML + HTTP + URL) yang berjalan di atasnya.
- Setiap tampilan halaman biasa adalah satu siklus request-response HTTP: pencarian DNS, permintaan, pemrosesan sisi-server, respons, dan rendering peramban.
- HTTP bersifat stateless secara desain — mengingat apa pun tentang pengunjung antar-permintaan memerlukan mekanisme yang sengaja ditambahkan seperti cookie dan session (Lecture 6).

- Lima era Web yang umum dikutip (1.0 hanya-baca, 2.0 baca-tulis, 3.0 API/mobile, 4.0 cerdas/ubikuitus, 5.0 terdesentralisasi/simbiotik) adalah singkatan yang berguna, bukan nomor versi yang terstandardisasi secara presisi.
- Mata kuliah ini mengajarkan aplikasi web CRUD yang sama tiga kali — di PHP/MySQL, JSP, dan ASP.NET/ADO.NET — di sekitar satu studi kasus bersama, Profitina Laundry, sehingga Anda mempelajari apa yang universal versus apa yang spesifik-stack.

"Ide asli dari web adalah bahwa web seharusnya menjadi ruang kolaboratif tempat kita bisa berkomunikasi lewat berbagi informasi." — setiap teknologi yang diajarkan mata kuliah ini pada akhirnya ada untuk melayani satu ide itu.

1.10 Pertanyaan Tinjauan

Pertanyaan-pertanyaan ini untuk belajar mandiri dan diskusi kelas; tidak dinilai. Kerjakan sebelum Lecture 2.

1. Jelaskan, dengan kata-kata Anda sendiri, mengapa Internet dan Web bukan hal yang sama. Berikan satu contoh aplikasi Internet yang bukan bagian dari Web.
2. Telusuri lima langkah pada Gambar 1.1 untuk kasus konkret: Anda mengetik https://profitinalaundry.example.com/bills di address bar peramban Anda. Apa yang terjadi di tiap langkah, dan langkah mana yang melibatkan basis data?
3. HTTP dideskripsikan sebagai "stateless." Berikan satu contoh nyata (selain login) informasi yang mungkin perlu diingat oleh sebuah aplikasi web lintas beberapa permintaan dari pengunjung yang sama, dan jelaskan singkat bagaimana Anda mungkin membuat server "mengingatnya" meski HTTP bersifat stateless.
4. Permintaan GET seharusnya "safe" (tanpa efek samping di sisi server). Deskripsikan sebuah bug yang masuk akal di mana seorang developer melanggar aturan ini, dan jelaskan satu masalah konkret yang bisa ditimbulkannya (petunjuk: pikirkan apa yang mungkin dilakukan peramban atau crawler mesin pencari terhadap sebuah URL secara otomatis, tanpa ada manusia yang mengklik apa pun).
5. Pilih situs web atau aplikasi apa pun yang Anda pakai setiap hari. Menurut Anda era Web mana (1.0–5.0) yang paling menjadi tempat fungsionalitas intinya, dan fitur tunggal apa yang paling memengaruhi jawaban Anda?

Referensi

[1] T. Berners-Lee, R. Cailliau. "WorldWideWeb: Proposal for a HyperText Project." CERN, November 1990.

[2] R. Fielding et al. "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content." RFC 7231, Internet Engineering Task Force, Juni 2014.

[3] TIOBE Software. "TIOBE Index for August 2026." https://www.tiobe.com/tiobe-index/ (diakses Agustus 2026).

[4] W3Techs. "Usage Statistics and Market Share of Web Servers, August 2026." https://w3techs.com/technologies/overview/web_server (diakses Agustus 2026).

[4b] W3Techs. “Usage Statistics and Market Share of Server-Side Programming Languages for Websites, August 2026.” https://w3techs.com/technologies/overview/programming_language (diakses Agustus 2026).

[4c] TechRepublic. “TIOBE Index for August 2026: Java Nears C++ as Rust Holds No. 10.” https://www.techrepublic.com/article/news-tiobe-august-2026-java-nears-c-plus-plus/ (diakses Agustus 2026); StatisticsTimes.com, “Top Computer Languages 2026” (peringkat PHP, data Maret 2026), https://statisticstimes.com/tech/top-computer-languages.php.

[5] T. Berners-Lee. Dikutip dalam: Goodreads, “Quotes by Tim Berners-Lee.” https://www.goodreads.com/author/quotes/428754.Tim_Berners_Lee.

[6] R. Soelaiman, S. Candra, M. M. I. Subakti. “Efficient Approach for Deterministic Data Extrapolation from a Clean Periodic Function with Periodic Components Representation by a System of Linear Equations.” Journal of Theoretical and Applied Information Technology (JATIT), Vol. 97, No. 8, 2019.

BAB 2

HTML: Menyusun Struktur Konten untuk Web

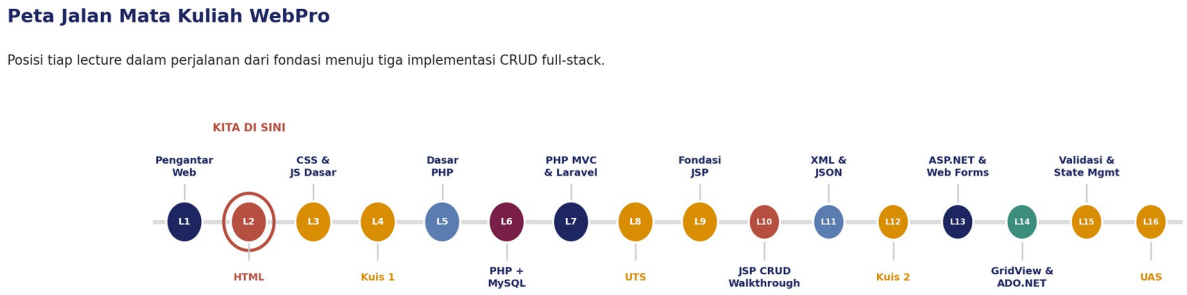
Setiap halaman yang disentuh pelanggan Profitina Laundry — beranda, formulir pemesanan, tabel tagihan — memulai hidupnya sebagai HTML biasa. Bab ini adalah tempat semua itu dimulai.

Tujuan Pembelajaran — setelah bab ini Anda akan mampu:

(1) Menulis dokumen HTML5 yang valid dengan struktur doctype, head, dan body yang benar. (2) Memilih elemen semantik HTML5 (header, nav, main, section, article, aside, footer) alih-alih div generik, dan menjelaskan mengapa pilihan itu penting bagi mesin, bukan hanya bagi manusia. (3) Menyusun konten teks, gambar, dan data tabular dengan benar. (4) Membangun formulir HTML yang berfungsi memakai tipe input, label, dan atribut validasi bawaan yang tepat. (5) Menerapkan praktik aksesibilitas inti (teks alt, asosiasi label, lang, landmark ARIA) dan menjelaskan hubungannya dengan cara crawler AI sungguh-sungguh membaca sebuah halaman.

2.1 Rekap: Dari Lecture 1 ke Lecture 2

Lecture 1 membangun model mental yang menjadi dasar seluruh mata kuliah ini: peramban mengirim permintaan HTTP, server menjawab, dan peramban merender HTML apa pun yang dikembalikan. Bab ini adalah kali pertama Anda menulis HTML itu sendiri. Setiap contoh di bawah adalah fragmen nyata dan berfungsi dari beranda Profitina Laundry yang sesungguhnya — halaman yang sama yang akan mendapat stylesheet di Lecture 3, penanganan formulir PHP di Lecture 5–7, penulisan ulang JSP di Lecture 9–10, dan penulisan ulang ASP.NET di Lecture 13–14 (Gambar 2.1).



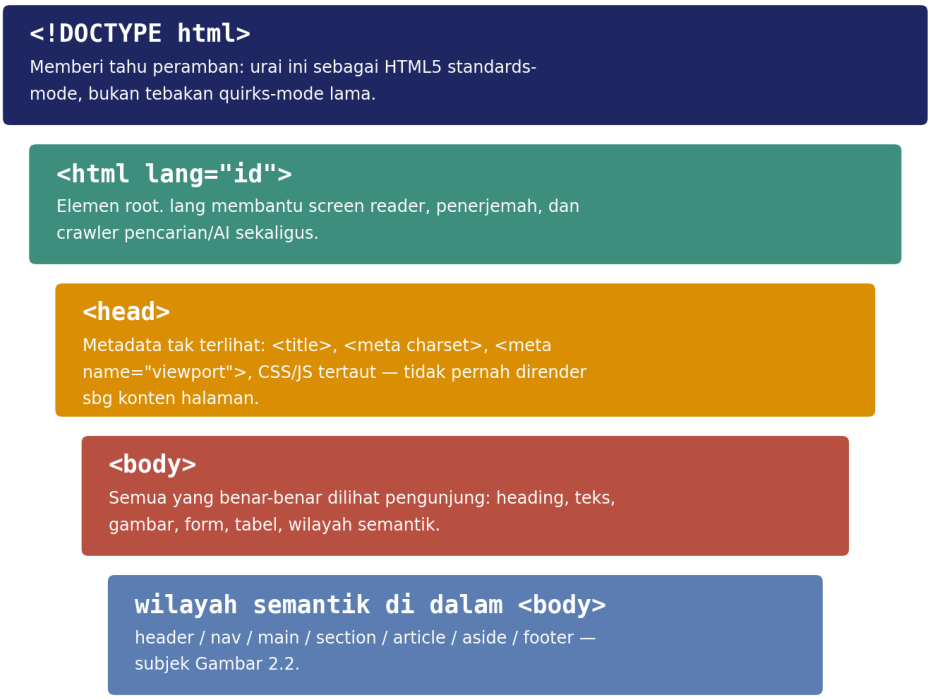
Gambar 2.1 — Posisi bab ini dalam roadmap enam belas lecture WebPro. Lecture 3 akan menambahkan CSS dan JavaScript di atas struktur yang dibangun bab ini.

2.2 Anatomi Dokumen HTML5

Setiap halaman HTML5 yang valid — termasuk beranda Profitina Laundry — menyusun lima lapisan bersarang yang sama, dalam urutan yang sama. Lewatkan satu saja, dan peramban dibiarkan menebak maksud Anda — persis jenis tebakan yang membuat rendering berbeda antar-peramban, pembaca layar, dan crawler (Gambar 2.2).

Anatomi Dokumen HTML5

Setiap halaman HTML5 yang valid menyusun lima lapis yang sama — lewatkan satu, dan peramban harus menebak maksud Anda.



Gambar 2.2 — Lima lapisan bersarang yang dibutuhkan setiap dokumen HTML5: doctype, elemen akar, head, body, dan — di dalam body — kawasan semantik yang dibahas di Bagian 2.4.

> **HTML adalah Living Standard, bukan versi yang tetap**
>
> "HTML5" adalah nama yang masih dipakai kebanyakan orang, tapi tidak pernah ada "HTML6" dan tidak akan pernah ada. Sejak 2019 WHATWG (kelompok industri pembuat peramban yang mencakup Google, Apple, Mozilla, dan Microsoft) dan W3C bersama-sama memelihara satu spesifikasi tunggal bernama HTML Living Standard — terus-menerus diperbarui, bukan dibekukan ke rilis bernomor. Konsekuensinya secara praktis: jangan menulis `<!DOCTYPE html5>` (tidak ada hal seperti itu) — doctype yang benar dan permanen hanyalah `<!DOCTYPE html>`, dan itu akan tetap benar selama Web masih ada.

Setiap atribut pada elemen akar penting lebih dari yang terlihat. `<html lang="id">` hanya satu kata, tapi ia memberi tahu pembaca layar aturan pelafalan mana yang dipakai, memberi tahu penerjemah otomatis bahasa apa yang sedang diterjemahkan, dan — semakin relevan bagi bisnis Profitina sendiri — memberi tahu crawler pencari atau AI bahasa apa yang dipakai konten halaman itu bahkan sebelum ia mulai mem-parsing body.

2.3 Konten Teks: Heading, Paragraf, Daftar, dan Tautan

Elemen teks adalah blok bangunan HTML yang paling lama dan paling sering dipakai, dan maknanya melampaui ukuran visualnya. `<h1>` hingga `<h6>` membentuk sebuah outline — pengguna pembaca layar rutin melompat dari heading ke heading sebagaimana pembaca yang bisa melihat memindai

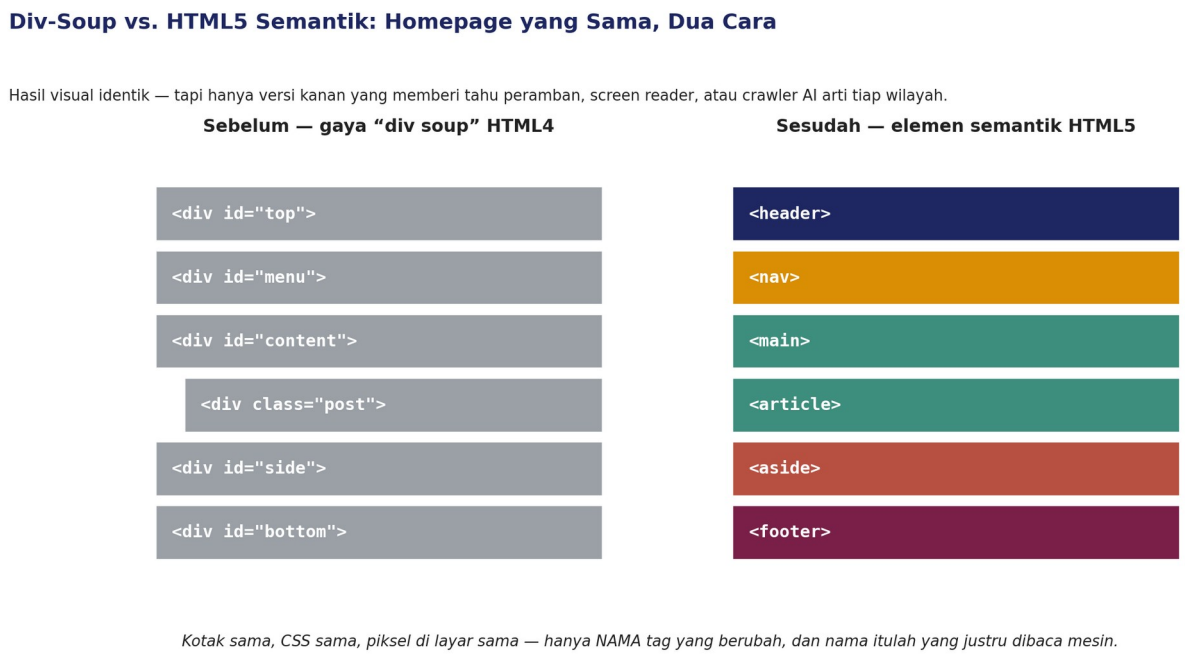
daftar isi, sehingga sebuah halaman semestinya punya tepat satu `<h1>` (subjek utamanya) dan tidak melompati level (sebuah `<h3>` semestinya tidak langsung mengikuti `<h1>`).

- `<p>` — sebuah paragraf teks isi; peramban otomatis menambahkan spasi di atas dan bawahnya.
- `<ul>` / `<ol>` / `<li>` — daftar tak berurutan dan berurutan; pakai `<ol>` hanya ketika urutannya sendiri bermakna (langkah, peringkat), `<ul>` untuk selain itu.
- `<a href="...">` — sebuah hyperlink; teks yang terlihat semestinya mendeskripsikan tujuannya ("Book a Pickup", bukan "klik di sini"), karena teks itulah yang diucapkan pembaca layar dan yang dipakai crawler untuk memahami tautan itu tentang apa.
- `<strong>` / `<em>` — penekanan semantik (kepentingan / tekanan), bukan sekadar gaya tebal atau miring; pembaca layar bisa mengubah nada suara untuk elemen-elemen ini, sesuatu yang tidak dipicu oleh `<b>` atau `<i>` yang murni visual.

> **Teks tebal dan heading yang terlihat tebal bukanlah hal yang sama**
>
> Membuat sebuah `<p>` tebal dan lebih besar lewat CSS agar TERLIHAT seperti heading adalah jalan pintas yang umum — dan sebuah bug aksesibilitas yang nyata. Perintah "lompat ke heading berikutnya" pada pembaca layar hanya mengenali elemen `<h1>`–`<h6>` yang sesungguhnya; sebuah paragraf tebal tidak terlihat olehnya berapa pun besar fontnya. Jika sesuatu secara struktural adalah heading, tandai sebagai heading.

2.4 HTML5 Semantik: Menyusun Halaman Seperti Beranda Profitina Laundry

HTML5 menambahkan sekumpulan elemen yang seluruh tujuannya adalah menamai PERAN yang dimainkan sebuah kawasan halaman, bukan sekadar posisi visualnya. Dua halaman bisa dirender sebagai kotak-kotak yang identik piksel demi piksel di layar sementara dibaca sama sekali berbeda oleh apa pun yang bukan sepasang mata manusia — pembaca layar, pohon aksesibilitas bawaan peramban, atau crawler pencari/AI yang mencoba memahami sebuah halaman itu tentang apa (Gambar 2.3).



Gambar 2.3 — Beranda visual yang sama, ditandai dengan dua cara. Hanya elemen di sebelah kanan yang mengumumkan sebuah peran kepada apa pun yang membaca markup-nya, bukan sekadar melihatnya.

2.4.1 Sebelum: Div Soup (kebiasaan era HTML4 yang perlu dilupakan)

```
<h2>Before: HTML4-style "div soup" (avoid this)</h2>
<div id="top">Profitina Laundry</div>
<div id="menu">
  <div class="menu-item"><a href="#services">Services</a></div>
  <div class="menu-item"><a href="#booking">Book a Pickup</a></div>
</div>
<div id="content">
  <div class="post">
    <div class="post-title">Wash &amp;amp; Fold</div>
    <div class="post-body">Ready in 24 hours.</div>
  </div>
</div>
<div id="side">Customers rate us 4.8 / 5.</div>
<div id="bottom">&amp;copy; 2026 Profitina Laundry</div>
```

2.4.2 Sesudah: Enam Landmark Semantik

```
<h2>After: HTML5 semantic elements (what this course teaches)</h2>
<header>Profitina Laundry</header>
<nav>
  <a href="#services">Services</a>
  <a href="#booking">Book a Pickup</a>
</nav>
<main>
  <article>
    <h3>Wash &amp; Fold</h3>
    <p>Ready in 24 hours.</p>
  </article>
</main>
<aside>Customers rate us 4.8 / 5.</aside>
<footer>&copy; 2026 Profitina Laundry</footer>
```

Elemen	Peran landmark yang diumumkan	Dipakai tepat sekali per halaman?
<header>	Konten pengantar — logo, nama situs, heading utama	Biasanya ya (per halaman atau per artikel)
<nav>	Sebuah blok tautan yang terutama bersifat navigasi	Tidak — satu halaman bisa punya beberapa (nav utama, nav footer)
<main>	Konten utama dan unik dari halaman	Ya — tepat satu per halaman, selalu
<article>	Konten mandiri yang tetap masuk akal jika disindikasikan sendiri	Tidak — beranda Profitina Laundry punya tiga, satu per layanan
<aside>	Konten yang berkaitan secara tangensial dengan konten utama	Tidak — nol atau lebih
<footer>	Konten penutup — hak cipta, info kontak, tautan situs	Biasanya ya (per halaman atau per artikel)

Perhatikan bahwa markup Bagian 2.4.1 dan markup Bagian 2.4.2 akan menghasilkan tangkapan layar yang IDENTIK begitu CSS yang sama diterapkan ke keduanya — Lecture 3 bisa menata salah satunya agar terlihat seperti beranda Profitina Laundry yang sesungguhnya. Satu-satunya yang berubah adalah NAMA tag-nya, dan nama tag itulah yang sesungguhnya dibaca oleh mesin — pembaca layar, crawler, atau lainnya.

2.5 Gambar dan Media

Sebuah elemen `<img>` memerlukan `src` dan, sama pentingnya, sebuah atribut `alt` — teks yang diucapkan pembaca layar sebagai pengganti gambar itu, dan teks yang menjadi cadangan bagi crawler pencari atau AI ketika ia sendiri tidak bisa "melihat" gambarnya. `alt=""` yang kosong adalah pilihan yang benar untuk gambar yang murni dekoratif (ia memberi tahu teknologi bantu untuk melewatinya secara diam-diam); atribut `alt` yang sama sekali tidak ada selalu merupakan kesalahan, dekoratif atau bukan.

```
<img src="profitina-laundry-logo.svg"
     alt="Profitina Laundry logo" width="48" height="48">

<figure>
  <img src="wash-fold-basket.jpg" alt="A basket of folded towels">
  <figcaption>Every Wash &amp;amp; Fold order is folded, not just
dried.</figcaption>
</figure>
```

- `<figure>` / `<figcaption>` — memasangkan sebuah gambar (atau contoh kode, chart, atau diagram) dengan sebuah caption yang diasosiasikan secara terprogram dengannya, bukan sekadar berdekatan secara visual.
- `loading="lazy"` pada sebuah `<img>` memberi tahu peramban untuk menunda pengunduhan gambar yang berada di luar layar hingga pengunjung menggulir mendekatinya — sebuah atribut satu kata dengan efek nyata dan terukur pada seberapa cepat halaman seperti katalog produk yang panjang pertama kali terasa bisa dipakai.
- `srcset` memungkinkan satu `<img>` menawarkan beberapa resolusi dari gambar yang sama sehingga sebuah ponsel mengunduh berkas yang lebih kecil dibanding monitor desktop, dari markup yang persis sama.

2.6 Tabel untuk Data Tabular

Sebuah tabel adalah elemen yang benar hanya untuk data yang sungguh-sungguh tabular — baris dan kolom dari nilai yang sebanding — tidak pernah sebagai trik tata letak halaman (itu adalah hack era HTML4 yang dulu umum, dan sudah sepenuhnya ditinggalkan). Riwayat penagihan Profitina Laundry adalah contoh buku teks untuk sebuah tabel sungguhan.

```
      <table>
        <caption>Your last four Profitina Laundry bills</caption>
        <thead>
          <tr>
            <th scope="col">Bill #</th>
            <th scope="col">Date</th>
            <th scope="col">Service</th>
            <th scope="col">Amount (IDR)</th>
            <th scope="col">Status</th>
          </tr>
        </thead>
        <tbody>
          <tr>
            <th scope="row">B-1042</th>
            <td>2026-08-18</td>
            <td>Wash &amp;amp; Fold</td>
            <td>85,000</td>
            <td>Paid</td>
          </tr>
```

Elemen	Tujuan
<table>	Tabel itu sendiri
<caption>	Judul singkat yang diasosiasikan secara terprogram untuk seluruh tabel

<thead> / <tbody> / <tfoot>	Mengelompokkan baris header, baris isi, dan baris ringkasan sehingga teknologi bantu dan CSS bisa menyasar masing-masing secara terpisah
<tr>	Satu baris tabel
<th scope="col">	Sebuah sel header kolom — `scope` memberi tahu pembaca layar bahwa setiap sel di bawahnya di kolom itu berkaitan dengannya
<th scope="row">	Sebuah sel header baris — di sini, nomor tagihan yang dideskripsikan setiap sel lain di baris itu
<td>	Sebuah sel data biasa

Mengapa scope="col" dan scope="row" penting lebih dari yang terlihat

Pengguna yang bisa melihat menyimpulkan "angka ini adalah kolom Amount" secara sekilas, melirik ke header di atasnya. Pengguna pembaca layar yang menavigasi sel demi sel tidak punya sekilas pandang semacam itu — tanpa `scope`, "145.000" hanyalah angka tanpa konteks. Dengan `scope="col"` pada baris header dan `scope="row"` pada sel nomor tagihan, sebuah pembaca layar bisa mengumumkan "Amount (IDR): 145.000, baris B-1039" untuk setiap sel secara otomatis. Ini hanya berharga satu atribut per sel header, dan itulah perbedaan antara tabel yang bisa dipakai dan dinding angka yang tidak bisa dipakai bagi sekelompok nyata pelanggan Profitina Laundry sendiri.

2.7 Formulir: Mengumpulkan Data dari Pengguna

Sebuah `<form>` adalah cara peramban mengumpulkan dan mengirim input pengguna ke server — permintaan pemesanan yang, mulai Lecture 5, akan sungguh-sungguh mencapai sebuah skrip PHP dan menulis satu baris ke basis data Profitina Laundry. Setiap field memerlukan sebuah `<label>` yang secara eksplisit diikat ke input-nya lewat `for`/`id`; tanpa pasangan itu, mengklik teks label tidak melakukan apa-apa dan pembaca layar tidak bisa mengumumkan field itu untuk apa.

```
        <div class="field">
          <label for="customer-email">Email</label>
          <input type="email" id="customer-email" name="customerEmail"
                 required autocomplete="email">
        </div>

        <div class="field">
          <label for="service-type">Service</label>
          <select id="service-type" name="serviceType" required>
            <option value="">Choose a service&hellip;</option>
            <option value="wash-fold">Wash &amp;amp; Fold</option>
            <option value="dry-clean">Dry Cleaning</option>
            <option value="express">Express Same-Day</option>
          </select>
```

Tipe Input	Untuk apa	Validasi bawaan yang didapat gratis
text	Teks bebas pendek (sebuah nama)	`required`, `minlength`, `maxlength`, `pattern`
email	Sebuah alamat email	Peramban menolak teks tanpa @ dan sebuah domain

tel	Sebuah nomor telepon	Tidak ada format yang dipaksakan secara default — pasangkan dengan `pattern`
number	Sebuah kuantitas numerik	`min`, `max`, `step`; peramban menampilkan kontrol stepper
date	Sebuah tanggal kalender	Peramban merender date picker bawaan
checkbox	Sebuah toggle ya/tidak	`checked` menetapkan default; nilai hanya terkirim jika dicentang

Mengapa repot memakai type="email" jika server tetap harus mengeceknya ulang?

Anda akan mempelajari di Lecture 6 bahwa server TIDAK PERNAH boleh memercayai apa pun yang dikirim peramban — sebuah pengecekan sisi-client seperti `type="email"` selalu bisa dilewati oleh pengunjung yang mengedit permintaan secara langsung. Jadi mengapa tetap memakainya? Karena itu adalah kesopanan, bukan batas keamanan: ia membiarkan 99% pengguna jujur menangkap salah ketik secara instan, gratis, tanpa JavaScript dan tanpa round-trip server. Validasi sisi-client memperbaiki pengalaman; validasi sisi-server (Lecture 6 dan seterusnya) melindungi data. Sebuah formulir produksi memerlukan keduanya, untuk alasan yang berbeda.

2.8 Aksesibilitas dan Peran HTML yang Mengejutkan dalam Visibilitas AI

Markup aksesibilitas — `alt`, `label`/`for`, `lang`, landmark ARIA seperti atribut `aria-label` dan `aria-labelledby` yang dipakai di sepanjang contoh bab ini — pertama dan terutama ada demi manusia yang memakai teknologi bantu. Tapi markup yang sama semakin penting bagi audiens kedua yang dipikirkan Profitina secara profesional: crawler dan model bahasa yang memutuskan apakah sebuah bisnis direkomendasikan ketika seseorang bertanya kepada asisten AI.

Apa yang sungguh-sungguh membantu crawler AI membaca halaman Anda (dan apa yang tidak)

Tergoda untuk mengasumsikan menambahkan data terstruktur `schema.org`/JSON-LD yang tak terlihat adalah pengungkit utama untuk visibilitas AI. Buktinya lebih beragam dari itu: pengujian independen 2025–2026 di beberapa sistem AI menemukan tingkat ekstraksi JSON-LD berkisar dari 0% hingga sekitar 50% tergantung sistemnya, dengan konten HTML yang terlihat dan tersusun rapi secara konsisten mengungguli schema markup tersembunyi untuk apa yang sesungguhnya diekstraksi model bahasa besar — satu pengujian bahkan mendapati sebuah sistem AI menarik alamat palsu dari JSON-LD yang sengaja tidak valid, mengindikasikan bahwa crawler saat ini sering memperlakukan data terstruktur sebagai teks biasa alih-alih mem-parsing-nya secara ketat. Mesin pencari adalah pengecualian sebagian: Google dan Bing tetap mem-parsing JSON-LD saat pengindeksan untuk Knowledge Graph dan fitur rich-result. Kesimpulan praktis yang didukung bukti untuk mata kuliah ini: perbaiki dulu HTML semantik dan markup aksesibel di Bagian 2.4–2.7 — heading yang bersih, landmark `<nav>`/`<main>`/`<article>` yang nyata, teks tautan yang deskriptif, teks `alt`, dan formulir yang berlabel — karena itulah yang paling andal dipahami baik oleh pembaca layar MAUPUN crawler AI. Data terstruktur layak ditambahkan kemudian sebagai pelengkap, bukan pengganti.

Inilah persis lapisan yang dihabiskan waktunya oleh Profitina, produk AI-visibility milik penulis, untuk dianalisis atas nama bisnis-bisnis nyata — dan inilah mengapa mata kuliah ini mengajarkan HTML

semantik sebagai topik kelas satu di Bab 2, bukan sebagai renungan tambahan setelah sebuah halaman sekadar "terlihat benar".

2.9 WebPro dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Di sepanjang buku ini, kotak seperti ini menghubungkan materi mata kuliah dengan Profitina, produk AI-visibility (Generative Engine Optimization) asal Indonesia yang nyata, dibangun oleh penulis, dan dengan Profitina Laundry, bisnis jasa laundry nyata yang menjadi studi kasus berjalan buku ini. Kotak-kotak ini mendeskripsikan DI MANA dan MENGAPA suatu konsep dipakai di sebuah sistem yang hidup dan berjalan — bukan detail implementasi eksklusif, logika bisnis, atau kode sumber Profitina itu sendiri, yang tetap rahasia.

Bagian 2.8 sudah menceritakan separuh kisah Profitina bab ini. Separuh lainnya adalah Profitina Laundry itu sendiri: beranda semantik di Bagian 2.4, formulir pemesanan di Bagian 2.7, dan tabel penagihan di Bagian 2.6 bukanlah contoh pengajaran hipotetis yang direkayasa untuk buku ini — semuanya adalah versi yang disederhanakan dan mudah diajarkan dari jenis halaman yang sesungguhnya dijalankan sebuah bisnis laundry nyata dalam produksi, jenis halaman yang sama yang akan diimplementasikan ulang mata kuliah ini terhadap basis data yang hidup dalam tiga stack sisi-server berbeda mulai Lecture 5.

2.10 Ringkasan Bab dan Poin-Poin Kunci

- Setiap dokumen HTML5 yang valid menyusun lima lapisan yang sama: doctype, `<html lang>`, `<head>`, `<body>`, dan — di dalam body — kawasan semantik.
- HTML adalah Living Standard WHATWG/W3C — terus-menerus dipelihara, tidak dirilis sebagai versi bernomor; doctype yang benar hanyalah `<!DOCTYPE html>`.
- Elemen semantik (`header`, `nav`, `main`, `article`, `aside`, `footer`) bisa dirender identik piksel demi piksel dengan `<div>` generik sementara bermakna sama sekali berbeda bagi pembaca layar atau crawler.
- Tabel memerlukan `<caption>`, `<thead>`/`<tbody>`, dan `scope` pada sel header agar sungguh-sungguh bisa dipakai teknologi bantu, bukan sekadar tersusun rapi secara visual.
- Formulir memerlukan sebuah `<label>` yang secara eksplisit diikat ke setiap input, dan atribut `type`-nya memberi validasi sisi-client gratis — yang memperbaiki pengalaman tapi tidak pernah menggantikan validasi sisi-server (Lecture 6).
- Untuk visibilitas AI secara khusus, HTML semantik dan markup aksesibel yang bersih saat ini mengungguli data terstruktur tersembunyi sebagai apa yang paling andal diekstraksi model bahasa besar — perbaiki dulu Bagian 2.4–2.7 sebelum beralih ke schema.org.

“Konten mendahului desain. Desain sebelum ada konten bukanlah desain, itu dekorasi.”

— Jeffrey Zeldman

HTML adalah tempat konten mendahului segalanya — CSS (Lecture 3) hanya mendekorasi apa yang sudah distrukturkan Lecture 2.

2.11 Pertanyaan Tinjauan

Pertanyaan-pertanyaan ini untuk belajar mandiri dan diskusi kelas; tidak dinilai. Kerjakan sebelum Lecture 3.

1. Ambil kutipan "div soup" di Bagian 2.4.1 dan tulis ulang, elemen demi elemen, memakai enam landmark semantik dari tabel Bagian 2.4.2. Div era HTML4 mana yang tidak bisa dipetakan secara rapi ke landmark HTML5 tunggal mana pun, dan apa yang diberitahukan hal itu tentang batas kosakata semantik?
2. Formulir pemesanan Profitina Laundry (Bagian 2.7) tidak punya pengecekan sisi-client yang mencegah `pickup-date` di masa lalu. Usulkan sebuah perbaikan berbasis atribut khusus-HTML (tanpa JavaScript), dan jelaskan satu kasus di mana bahkan perbaikan itu tidak akan cukup.
3. Jelaskan, dengan kata-kata Anda sendiri, mengapa `alt=""` (kosong) dan atribut `alt` yang sama sekali tidak ada bukanlah hal yang sama, dan berikan satu gambar konkret di beranda Profitina Laundry di mana masing-masing menjadi pilihan yang benar.
4. Bagian 2.8 berargumen bahwa HTML semantik saat ini lebih penting bagi visibilitas AI dibanding data terstruktur JSON-LD tersembunyi. Ringkas bukti yang diberikan untuk klaim ini dalam dua kalimat, dan identifikasi satu caveat yang diakui bab ini sendiri.
5. Jelaskan mengapa tabel adalah elemen yang benar untuk riwayat penagihan Profitina Laundry (Bagian 2.6) tapi akan menjadi pilihan yang SALAH untuk menata letak bagian layanan tiga-kolom (Bagian 2.4) — meskipun keduanya secara visual adalah "grid" dari kotak-kotak.

Lampiran 2.A — Log Validasi untuk Kode Sumber Bab 2

Kedua berkas HTML yang disertakan pada bab ini (`profitina-laundry-home.html` dan `semantic-vs-divsoup.html`) di-parsing dengan parser HTML5 ketat dari `html5lib` — algoritma parsing yang sama yang diwajibkan HTML Living Standard untuk diimplementasikan peramban — dan dikonfirmasi bebas dari kesalahan struktural sebelum dikutip ke dalam bab ini.

```
$ python3 -c "import html5lib; ..."
profitina-laundry-home.html : PARSED OK, no strict errors
semantic-vs-divsoup.html    : PARSED OK, no strict errors
```

Referensi

[1] WHATWG. “HTML Living Standard.” https://html.spec.whatwg.org/ (terakhir diperbarui 26 Agustus 2026; diakses Agustus 2026).

[2] WHATWG/W3C. “Memorandum of Understanding between W3C and WHATWG,” 2019 — kesepakatan yang menetapkan HTML Living Standard sebagai satu spesifikasi tunggal yang dipelihara bersama, mengakhiri jalur bernomor terpisah “HTML5”.

[3] MDN Web Docs. “HTML: HyperText Markup Language” dan bagian referensi “ARIA”. https://developer.mozilla.org/en-US/docs/Web/HTML (diakses Agustus 2026).

[4] Ryall, P. “Structured Data & AI Crawlers: The Schema Hype, Debunked.” 2026. https://patrickryall.com/articles/structured-data-ai-crawlers (diakses Agustus 2026) — sumber untuk temuan tingkat ekstraksi JSON-LD yang dibahas di Bagian 2.8.

[5] Zeldman, J. Dikutip dalam: Goodreads, “Quotes by Jeffrey Zeldman.” https://www.goodreads.com/quotes/9357188 (diakses Agustus 2026).

[6] World Wide Web Consortium (W3C). Web Content Accessibility Guidelines (WCAG), bagian tentang header tabel dan pelabelan formulir. https://www.w3.org/WAI/ (diakses Agustus 2026).

BAB 3

Dasar-Dasar CSS & JavaScript

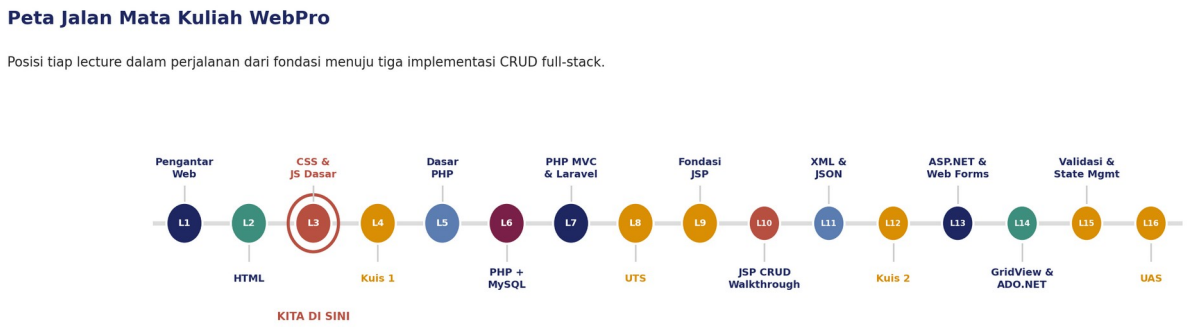
Bab 2 memberi homepage Profitina Laundry struktur dan makna. Bab ini memberinya tampilan dan denyut: CSS menentukan bagaimana setiap elemen dikotak-kotak, diberi jarak, dan disusun; JavaScript membuatnya merespons apa yang benar-benar dilakukan pengunjung.

Tujuan Pembelajaran — setelah bab ini Anda akan mampu:

(1) Menyisipkan CSS ke halaman dengan tiga cara, dan menjelaskan mengapa stylesheet eksternal hampir selalu pilihan yang tepat. (2) Bernalar tentang model kotak (box model) CSS, selector, dan cascade cukup baik untuk memprediksi aturan mana yang menang. (3) Menata satu region halaman nyata dengan Flexbox dan dengan CSS Grid, serta tahu kapan memakai yang mana. (4) Menggunakan native CSS nesting dan satu Container Query ukuran, serta menjelaskan satu catatan penting masing-masing. (5) Mendeklarasikan variabel dengan let/const dan menulis arrow function. (6) Memilih elemen DOM, memasang event listener, dan mengambil data JSON dengan Fetch API menggunakan async/await.

3.1 Recap: Dari Lecture 2 ke Lecture 3

Lecture 2 membangun homepage Profitina Laundry sebagai HTML murni tanpa gaya — struktur benar, semantik benar, nol desain visual. Buka file itu di peramban hari ini dan ia dirender sebagai tumpukan teks hitam di atas putih dari atas ke bawah: setiap heading, list, dan tabel ada di sana, tetapi belum ada yang disusun, diberi warna, atau interaktif. Bab ini mengubah persis itu, tanpa menyentuh struktur HTML sama sekali — setiap contoh di bawah menata gaya dan skrip pada markup YANG SAMA yang sudah dibangun Bab 2, melalui dua file baru yang ditambahkan bab ini: `styles.css` dan `script.js` (Gambar 3.1).



Gambar 3.1 — Posisi bab ini dalam peta jalan enam belas lecture WebPro. Lecture 4 adalah Kuis 1, mencakup semua materi hingga bab ini.

3.2 Tiga Cara Menyisipkan CSS ke Halaman

Peramban menerima CSS dari tiga tempat, dan proyek nyata seharusnya hanya memakai satu di antaranya untuk apa pun di luar uji coba cepat.

- Inline — atribut `style="..."` langsung pada satu elemen HTML. Prioritas tertinggi dalam cascade, tetapi tidak terkelola dalam skala besar: menata seratus tombol dengan cara ini berarti mengedit seratus atribut untuk mengubah satu warna.

- Internal — blok `<style>` di dalam `<head>` halaman itu sendiri. Cocok untuk satu halaman demo mandiri, tetapi gayanya tidak dapat dipakai ulang oleh halaman lain di situs.
- External — file `.css` terpisah yang ditautkan dengan `<link rel="stylesheet" href="styles.css">`, persis seperti yang sudah menunggu di `<head>` homepage Bab 2. Satu file, di-cache sekali oleh peramban, dipakai ulang oleh setiap halaman di situs Profitina Laundry.

> **styles.css, sudah tertaut dan siap sejak Bab 2**
>
> `<head>` Bab 2 sudah berisi `<link rel="stylesheet" href="styles.css">` — filenya saja yang belum ada. Bab ini menuliskannya, dan peramban langsung memakai setiap aturan di dalamnya begitu file itu ada, tanpa perubahan apa pun pada HTML.

3.3 Selector, Spesifisitas, dan Cascade

Sebuah aturan CSS adalah selector (apa yang ditata) dipasangkan dengan blok deklarasi (bagaimana menatanya). Tiga jenis selector yang terus-menerus dipakai di stylesheet bab ini adalah selector elemen (`header`, mencocokkan setiap `<header>`), selector kelas (`.field`, mencocokkan setiap elemen dengan `class="field"`), dan selector ID (`#services`, mencocokkan satu-satunya elemen dengan `id="services"`).

Ketika dua aturan bisa sama-sama berlaku pada elemen yang sama, "Cascading" dalam Cascading Style Sheets menentukan mana yang menang, menggunakan — berurutan — asal (default peramban kalah dari stylesheet penulis), spesifisitas (ID mengalahkan kelas, kelas mengalahkan elemen polos), dan akhirnya urutan sumber (aturan yang ditulis belakangan di file memenangkan seri yang benar-benar sama). Kalkulus spesifisitas lengkap memang ada, tetapi aturan praktis yang dipakai bab ini lebih sederhana: utamakan kelas untuk hampir semuanya, gunakan ID hanya untuk region halaman yang benar-benar unik seperti `#services`, dan hindari gaya inline sepenuhnya agar cascade tetap dapat diprediksi.

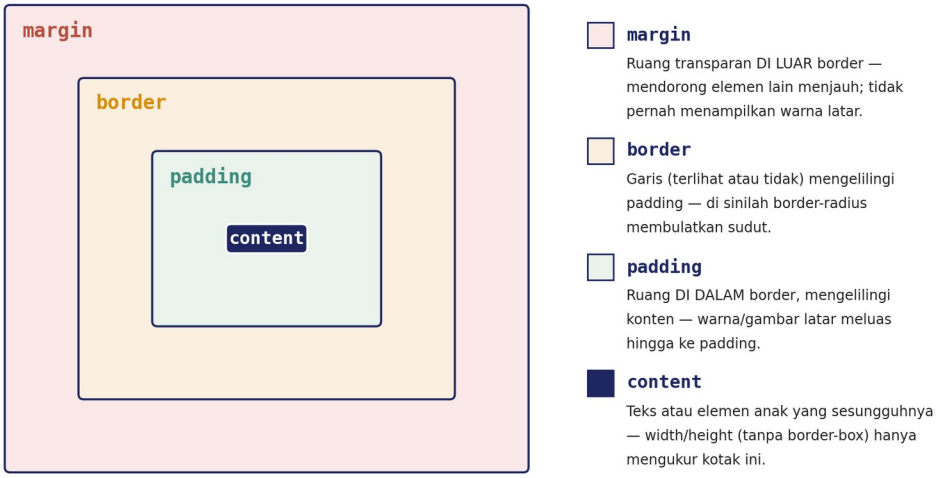
> **"!important" bukan jalan pintas spesifisitas**
>
> Menambahkan `!important` pada sebuah deklarasi mengesampingkan seluruh perhitungan cascade di atas — tetapi setiap `!important` membuat override BERIKUTNYA semakin sulit, karena hanya `!important` lain (dengan urutan sumber lebih belakangan) yang bisa mengalahkannya. `styles.css` tidak memakai satu pun deklarasi `!important`; jika sebuah aturan sungguh tidak menang, perbaikannya hampir selalu selector yang lebih spesifik — atau urutannya diperbaiki — bukan palu yang lebih besar.

3.4 Model Kotak (Box Model) CSS

Setiap elemen yang dirender peramban, di baliknya, adalah empat persegi panjang bersarang: content, padding, border, dan margin, dari dalam ke luar. Salah memahami model ini adalah sumber paling umum bug "kenapa ada celah yang tidak kuminta" pada stylesheet baru (Gambar 3.2).

Model Kotak CSS (Box Model)

Setiap elemen adalah empat kotak bersarang. box-sizing: border-box (Bagian 3.4) menjaga padding dan border TETAP DI DALAM lebar yang dideklarasikan.



Gambar 3.2 — Empat kotak yang dimiliki setiap elemen, dari dalam ke luar. Margin tidak pernah mewarnai latar; padding iya.

```
/* ---- 3.4 The box model: box-sizing changes what "width" measures ----
   Without this rule, adding padding/border to an element with a fixed
   width makes the element LARGER than that width (content-box, the
   default). border-box instead keeps padding+border INSIDE the
   declared width — the behavior almost every real stylesheet wants. */
*, *::before, *::after {
  box-sizing: border-box;
}
```

Tanpa `box-sizing: border-box`, default peramban (`content-box`) berarti kotak `width: 300px` yang juga punya `padding: 20px` dan border `2px` sebenarnya menempati `344px` di layar — padding dan border DITAMBAHKAN di atas lebar yang dideklarasikan. `border-box`, diterapkan sekali ke setiap elemen lewat selector universal (`*`) di bagian atas `styles.css`, membuat `width` menggambarkan ukuran total kotak di layar, yang justru sesuai intuisi tata letak hampir semua orang.

3.5 Tata Letak dengan Flexbox

Flexbox mengubah elemen-anak langsung suatu elemen menjadi baris (atau kolom) item fleksibel yang segelintir properti bisa meratakan, memberi jarak, dan mengurutkan ulang — persis pekerjaan yang dibutuhkan tiga anak `<header>` (logo, tagline, dan nav).

```
/* ---- 3.5 Flexbox: the header row ----
   display:flex turns <header>'s direct children into flex items laid
   out in a row by default. align-items:center vertically centers the
   logo, tagline, and nav despite their different heights; gap puts
   space BETWEEN items without needing margin on each one. */
header {
  display: flex;
  align-items: center;
  gap: 1rem;
  padding: 0.75rem 1.25rem;
  background: var(--navy);

  /* Native CSS nesting (3.7): a selector for a descendant, written
     directly inside the parent rule instead of as a separate
     top-level rule further down the file. */
  & .tagline {
    color: white;
    font-weight: bold;
    font-size: 1.1rem;
    margin: 0;
    margin-right: auto; /* pushes everything after it to the right */
  }
}
```

Properti	Diterapkan pada...	Fungsinya di sini
display: flex	induk (header)	mengubah anak langsung menjadi flex item, tersusun sebagai baris secara default
align-items: center	induk	meratakan item dengan tinggi berbeda secara vertikal di sepanjang baris
gap: 1rem	induk	memberi jarak ANTAR item saja — tidak perlu margin di tiap anak
margin-right: auto	satu anak (.tagline)	menyerap semua ruang sisa, mendorong semua yang setelahnya ke tepi kanan

Baris terakhir itu layak direnungkan: `margin: auto` pada satu flex item adalah cara nav Profitina Laundry berakhir rata dengan tepi kanan header tanpa satu piksel pun pemosisian manual — flex container cukup memberi auto-margin itu semua piksel sisa yang dimilikinya.

3.6 Tata Letak dengan CSS Grid dan Container Query

Grid adalah pasangan Flexbox untuk tata letak dua dimensi — baris DAN kolom sekaligus — yang persis dibutuhkan tiga kartu layanan di bagian "Our Services": grid yang menampilkan satu kolom di layar sempit dan tiga kolom begitu ada ruang.

```css
/* ---- 3.6 CSS Grid + Container Queries: the services section ----
   container-type:inline-size turns #services into a "query container"
   — its OWN width, not the browser viewport's width, is what the
   @container rule below reacts to. That matters because Section 3.6's
   whole point is: a component should adapt to the space IT has been
   given, not just to the size of the whole screen. */
#services {
  container-type: inline-size;
  container-name: services;
  padding: 1.5rem 0;
}

.card-grid {
  display: grid;
  grid-template-columns: 1fr;   /* one column until the container says otherwise
*/
  gap: 1.25rem;
}

.card-grid article {
  background: var(--bg-light);
  border: 1px solid var(--border);
  border-radius: 0.5rem;
  padding: 1.25rem;

  & h3 { color: var(--teal); margin-top: 0; }
}

/* Fires based on #services's own width (min 640px of container space),
   which is NOT the same threshold as the page-wide nav media query
   above — a container can cross this breakpoint even on a phone, if
   it's placed inside a wide enough layout region. */
@container services (min-width: 640px) {
  .card-grid { grid-template-columns: repeat(3, 1fr); }
}
```

Mengapa breakpoint di sini adalah Container Query, bukan media query

Sebuah `@media` query bertanya "seberapa lebar seluruh jendela peramban?" — sebuah `@container` query justru bertanya "seberapa lebar kotak komponen INI sendiri?", disiapkan dengan menandai `#services` sebagai query container lewat `container-type: inline-size`. Perbedaan ini penting begitu sebuah komponen dipakai ulang di tempat yang lebih sempit dari halaman penuh — misalnya widget sidebar — di mana media query selebar halaman akan tetap melaporkan seluruh jendela sebagai "lebar" padahal komponen itu sendiri hanya kebagian sepotong kecil ruang.

Container Query adalah CSS arus utama masa kini di 2026

Container Query ukuran (size) dirilis di Chrome, Firefox, dan Safari sejak awal-hingga-pertengahan 2023 dan kini sudah kokoh Baseline "Widely Available" — aman dipakai di produksi tanpa fallback. Flexbox dan Grid sendiri secara formal masih berstatus W3C Candidate Recommendation Draft, bukan Recommendation final, tetapi keduanya sudah punya dukungan peramban yang universal dan stabil selama bertahun-tahun; "level-CR namun de facto stabil" menggambarkan sebagian besar CSS yang dipakai sehari-hari oleh front-end developer.

3.7 CSS Modern: Native Nesting

Setiap aturan bersarang di dalam `styles.css` — selector `&` yang ditulis di dalam aturan induknya, bukan sebagai selector top-level terpisah yang diulang — adalah CSS peramban native, bukan langkah preprocessor Sass atau LESS. Chrome, Firefox, dan Safari semuanya merilis dukungannya dalam rentang sekitar tiga setengah bulan satu sama lain di akhir 2023, dan per pertengahan 2026 fitur ini baru saja melewati ambang "Baseline: Widely Available" industri (sekitar 92–93% peramban yang dipakai).

```
header nav ul {
  display: flex;
  gap: 1.25rem;
  list-style: none;
  margin: 0;
  padding: 0;

  /* Nested rule: targets the <a> inside every <li> inside this <ul>,
     without writing "header nav ul li a" as its own top-level line. */
  & li a {
    color: white;
    text-decoration: none;
    font-size: 0.95rem;

    /* Nesting a pseudo-class one level deeper still. Nested rules like
       this are implicitly wrapped in :is(...) by the browser, which is
       why a very specific outer selector can occasionally change how
       specificity is calculated versus writing the same rule flat —
       worth remembering the first time a nested override "loses" to a
       rule you expected it to beat (Section 3.7). */
    &:hover, &:focus {
      color: var(--gold);
      text-decoration: underline;
    }
  }
}
```

Selector bersarang secara implisit dibungkus :is(...)

Peramban memperlakukan `header nav ul { & li a { … } }` setara dengan `:is(header nav ul) li a { … }`, bukan secara harfiah `header nav ul li a { … }` — biasanya tak terlihat, tetapi bisa mengubah perhitungan spesifisitas begitu aturan bersarang dan aturan datar (flat) yang menyasar elemen sama saling bersaing, yang merupakan kejutan paling umum dilaporkan ketika tim pertama kali mengadopsi native nesting.

3.8 Dasar-Dasar JavaScript: Variabel dan Fungsi

`script.js` bab ini memakai `const` untuk setiap nilai yang tidak pernah ditetapkan ulang dan `let` untuk segelintir yang memang berubah — dan tidak pernah `var`. Baik `const` maupun `let` bersifat block-scoped (hanya terlihat di dalam `{ }` tempat ia dideklarasikan), berbeda dari `var`, yang scope-nya adalah fungsi terdekat yang membungkusnya dan bisa "bocor" dengan cara membingungkan melintasi blok `if`/`for`. `const`/`let` dan arrow function (`() => { … }`) sama-sama muncul di ECMAScript 2015 (ES6, disahkan Juni 2015) dan dianggap sebagai cara baku dan mapan menulis JavaScript modern.

```
const navToggle = document.querySelector(".nav-toggle");
const primaryNav = document.getElementById("primary-nav");

// ---- 3.9 The DOM: reading and reacting to the page ----
// document.querySelector / getElementById turn a piece of markup into
// a JavaScript object this script can inspect and change. An event
// listener is how that object is told "run this function whenever X
// happens" — here, "toggle the mobile menu whenever the button is
// clicked."
if (navToggle && primaryNav) {
  // Arrow function: a shorter function syntax that also does not
  // rebind `this`, which matters once event handlers get more complex
  // than this one-line example.
  navToggle.addEventListener("click", () => {
    const isOpen = primaryNav.classList.toggle("nav-open");
    navToggle.setAttribute("aria-expanded", String(isOpen));
  });
}
```

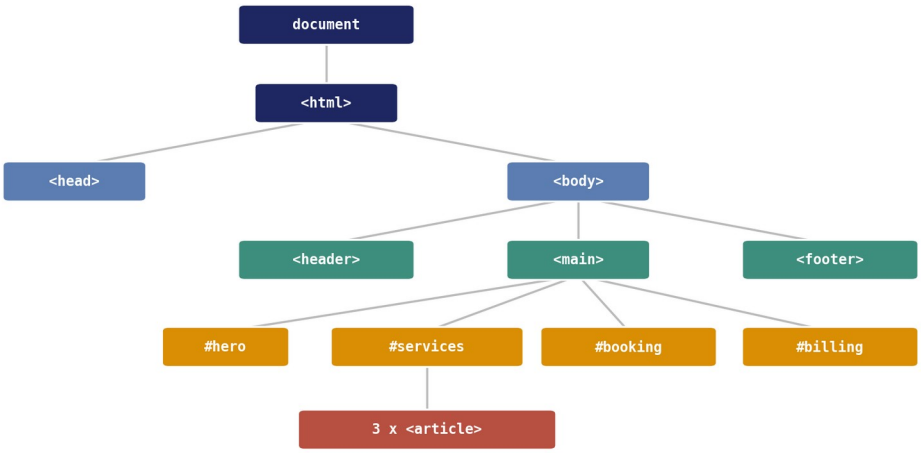
Arrow function yang diberikan ke `addEventListener` di atas adalah sebuah callback: fungsi yang diserahkan ke fungsi lain (di sini, sistem event peramban) untuk dijalankan nanti, kapan pun event yang dideskripsikan — klik pada `.nav-toggle` — benar-benar terjadi. `classList.toggle("nav-open")` sekaligus menambahkan kelas jika belum ada dan menghapusnya jika sudah ada, dan dengan nyaman mengembalikan `true`/`false` untuk mana pun yang baru saja dilakukannya — persis nilai yang dibutuhkan atribut `aria-expanded` yang aksesibel.

3.9 DOM: Model JavaScript atas Halaman

JavaScript yang berjalan di peramban tidak pernah membaca file teks HTML secara langsung — peramban mengurai file itu sekali menjadi Document Object Model (DOM), sebuah pohon objek node, dan setiap API DOM (`document.querySelector`, `getElementById`, `classList`, `addEventListener`) membaca atau mengubah pohon ITU, yang kemudian dirender ulang oleh peramban (Gambar 3.3).

DOM: Homepage Profitina Laundry sebagai Pohon

JavaScript tidak pernah melihat file teks HTML — ia melihat pohon objek node ini, yang dicari dan diubah oleh document.querySelector() dan sejenisnya.



Gambar 3.3 — Pohon DOM yang disederhanakan untuk homepage Profitina Laundry. document.querySelector("#services") menyusuri persis struktur ini untuk menemukan satu node.

```
const notesField = document.getElementById("notes");
const notesRemaining = document.getElementById("notes-remaining");

if (notesField && notesRemaining) {
  const maxLength = Number(notesField.getAttribute("maxlength")) || 120;

  const updateRemaining = () => {
    const remaining = maxLength - notesField.value.length;
    notesRemaining.textContent = String(remaining);
  };

  notesField.addEventListener("input", updateRemaining);
  updateRemaining(); // run once immediately, so the count is correct on page
load
}
```

Contoh kedua yang independen ini sengaja dibuat sederhana: satu event listener `input` menghitung ulang jumlah karakter tersisa setiap kali pengunjung mengetik di `<textarea>` catatan, dan `updateRemaining()` juga dipanggil sekali segera saat halaman dimuat sehingga hitungannya benar bahkan sebelum siapa pun mengetik apa pun — kebiasaan kecil ("jalankan handler sekali di awal, lalu biarkan event terus memperbaruinya") yang terus-menerus muncul dalam kode front-end nyata.

3.10 Fetch API: Data Sisi-Klien Tanpa Memuat Ulang Halaman

Setiap contoh sejauh ini hanya menyentuh elemen yang sudah ada di HTML. `fetch()` adalah cara sebuah halaman meminta server (atau, untuk saat ini, sebuah file statis) untuk data BARU tanpa berpindah ke halaman baru sama sekali — pengganti modern berbasis Promise untuk `XMLHttpRequest` yang lebih lama, dan fondasi yang menjadi dasar setiap halaman dinamis berbasis basis data di lecture WebPro berikutnya.

```
// ---- 3.10 The Fetch API: client-side data without a page reload ----
// fetch() replaces the older XMLHttpRequest for making HTTP requests
// from the browser. It returns a Promise, which is why this uses
// async/await instead of the .then()-chaining style older code uses.
// Lectures 5–7, 9–10, and 13–14 will point this same request at a real
// PHP/JSP/ASP.NET endpoint instead of a static JSON file — the DOM
// code below does not need to change at all when that happens.
const serviceSelect = document.getElementById("service-type");
const weightInput = document.getElementById("weight-kg");
const priceEstimate = document.getElementById("price-estimate");

async function loadServicePrices() {
  const response = await fetch("services.json");
  if (!response.ok) {
    throw new Error(`Could not load services.json: HTTP ${response.status}`);
  }
  // response.json() also returns a Promise, resolving to a parsed object.
  return response.json();
}
```

`async`/`await` adalah sintaks yang membuat kode berbasis Promise terbaca dari atas ke bawah seperti kode sinkron biasa, alih-alih callback `.then()` bersarang: `await fetch(...)` menjeda fungsi ini (tanpa membekukan peramban) hingga respons jaringan tiba, dan `await response.json()` demikian pula menjeda hingga isi respons selesai diurai sebagai JSON.

```
function updateEstimate(prices) {
  const serviceKey = serviceSelect.value;
  const weight = Number(weightInput.value);

  if (!serviceKey || !prices[serviceKey] || !(weight > 0)) {
    priceEstimate.textContent = "Choose a service and weight for an estimate.";
    return;
  }

  const service = prices[serviceKey];
  const total = service.pricePerKg * weight;
  const perKg = service.pricePerKg.toLocaleString("id-ID");
  // Template literal: embeds expressions directly in a string with
  // ${...}, instead of concatenating pieces with +.
  priceEstimate.textContent =
    `Estimated price: Rp ${total.toLocaleString("id-ID")} ` +
    `(${service.label}, ${weight} kg at Rp ${perKg}/kg)`;
```

Bagian	Fungsinya
fetch("services.json")	meminta file; resolve begitu header respons tiba (belum tentu seluruh body)
response.ok / response.status	true hanya untuk status HTTP 2xx — 404 TIDAK membuat Promise fetch me-reject dengan sendirinya, itulah sebabnya `!response.ok` diperiksa secara eksplisit
response.json()	membaca dan mengurai isi respons sebagai JSON; ia sendiri mengembalikan Promise

Template literal `` `teks ${expr}` ``	menyisipkan nilai sebuah ekspresi langsung ke dalam string, menggantikan penggabungan string dengan `+`

Lecture 5–7, 9–10, dan 13–14 akan mengarahkan panggilan `fetch("...")` yang persis sama ini ke endpoint PHP, JSP, atau ASP.NET sungguhan, didukung basis data nyata Profitina Laundry — dan tidak ada kode DOM di Bagian 3.9 yang perlu berubah sama sekali ketika itu terjadi, karena dari sudut pandang JavaScript, objek JSON yang sudah diurai terlihat identik entah ia datang dari file statis atau server yang hidup.

3.11 WebPro in Practice: Profitina

> **Tentang kotak ini**
>
> Di seluruh buku ini, kotak seperti ini menghubungkan materi kuliah dengan Profitina, produk AI-visibility (Generative Engine Optimization) asal Indonesia yang nyata dan dibangun oleh penulis, serta dengan Profitina Laundry, bisnis laundry nyata yang dipakai sebagai studi kasus berjalan buku ini. Kotak-kotak ini menjelaskan DI MANA dan MENGAPA sebuah konsep dipakai dalam sistem yang benar-benar berjalan — tidak pernah detail implementasi proprietary, logika bisnis, atau kode sumber Profitina itu sendiri, yang tetap rahasia.

Situs pemasaran Profitina sendiri, profitina.com, adalah tema WordPress — artinya setiap aturan visual yang diajarkan bab ini (box model, Flexbox, Grid, native nesting, container query) persis apa yang menjadi bahan stylesheet tema itu, dalam skala produksi, baik untuk versi bahasa Indonesia maupun Inggris yang dilayani plugin Polylang. Dan front end app.profitina.com, produk Profitina yang sesungguhnya, adalah halaman peramban seperti halaman lain mana pun: ia juga berjalan di atas panggilan `fetch()` yang mengembalikan JSON — pola Fetch API yang sama yang baru diajarkan Bagian 3.10, hanya diarahkan ke backend FastAPI Profitina sendiri, bukan file `services.json` statis. Model Qwen2.5 self-hosted yang menggerakkan analisis AI Profitina sepenuhnya tak terlihat di lapisan ini — peramban hanya pernah melihat JSON yang kembali dari sebuah endpoint, persis seperti contoh estimasi harga di atas, entah server menghitung JSON itu dari baris basis data atau dari model bahasa yang berjalan lokal.

3.12 Rangkuman Bab dan Poin-Poin Kunci

- Gunakan stylesheet eksternal untuk apa pun di luar demo sekali pakai; `<link rel="stylesheet">` adalah cara homepage Bab 2 sudah disiapkan untuk menerimanya.
- `box-sizing: border-box` membuat width/height menggambarkan ukuran total elemen di layar, termasuk padding dan border — tetapkan sekali, di setiap elemen, di bagian atas stylesheet.
- Flexbox menyusun satu dimensi anak (baris atau kolom) dengan properti perataan dan jarak; Grid menyusun dua dimensi (baris DAN kolom) sekaligus.
- Container Query merespons lebar kotak KOMPONEN itu sendiri, bukan jendela peramban — berbeda dari, dan sering lebih berguna dibanding, media query selebar halaman.
- Native CSS nesting (`&`) adalah CSS peramban nyata dan berlaku per 2026, bukan fitur preprocessor — tetapi selector bersarang secara implisit dibungkus `:is(...)`, yang bisa menggeser spesifisitas.

- `const`/`let` dan arrow function adalah default JavaScript modern yang mapan (ES2015); DOM adalah pohon hidup yang benar-benar dibaca dan diubah JavaScript, bukan pernah teks HTML asli.
- `fetch()` dengan `async`/`await` adalah cara standar masa kini meminta data dari JavaScript — dan pola yang sama yang dipakai bab ini terhadap file JSON statis persis yang akan dipakai halaman berbasis PHP/JSP/ASP.NET sungguhan mulai Lecture 5.

"Di dalam JavaScript, ada bahasa yang indah, elegan, dan sangat ekspresif yang terkubur di bawah tumpukan niat baik dan kesalahan."

— Douglas Crockford, JavaScript: The Good Parts (2008)

Bab ini mengajarkan bagian yang elegan itu — let/const, arrow function, DOM, dan fetch() — bagian yang sama yang dipakai codebase 2026 yang ditulis dengan baik.

3.13 Pertanyaan Tinjauan

Pertanyaan-pertanyaan ini untuk belajar mandiri dan diskusi kelas; tidak dinilai. Kerjakan sebelum Lecture 4 (Kuis 1).

1. Jelaskan, menggunakan box model, mengapa sebuah `<div>` dengan `width: 200px`, `padding: 10px`, dan border `1px` menempati 222px di layar di bawah `content-box` tetapi tepat 200px di bawah `border-box`.
2. `.card-grid` Profitina Laundry menampilkan satu kolom di bawah 640px lebar KONTAINER dan tiga kolom di atasnya. Deskripsikan satu situasi tata letak nyata di mana itu akan berperilaku berbeda dari aturan `@media (min-width: 640px)` selebar halaman, dan jelaskan mengapa.
3. Tulis ulang aturan bersarang `header nav ul { & li a { &:hover { … } } }` dari Bagian 3.7 sebagai tiga aturan terpisah, datar, tidak bersarang. Versi mana yang lebih mudah dibaca, dan mana yang lebih mudah dicari di file besar?
4. Contoh hitung-karakter Bagian 3.9 memanggil `updateRemaining()` sekali segera, selain memasangnya sebagai event listener. Apa yang akan dilihat pengunjung saat halaman dimuat jika panggilan segera itu dihapus, dan mengapa?
5. `updateEstimate()` di Bagian 3.10 memeriksa `!response.ok` secara eksplisit alih-alih mengandalkan `fetch()` untuk me-reject pada 404. Jelaskan, dengan kata-kata Anda sendiri, mengapa respons 404 tidak dengan sendirinya membuat Promise fetch me-reject.

Lampiran 3.A — Log Validasi Kode Sumber Bab 3

Ketiga file sumber baru bab ini masing-masing diperiksa dengan parser atau alat yang sesuai bahasanya, disiplin yang sama yang ditetapkan Bab 2 dengan `html5lib` untuk HTML: parser HTML5 ketat `html5lib` untuk `profitina-laundry-home.html` yang diperbarui, `tinycss2` (tokenizer/parser CSS berbasis standar) untuk `styles.css`, dan verifier sintaks `node --check` bawaan Node.js untuk `script.js`. Ketiganya lulus tanpa error sebelum kutipan mana pun di bab ini diambil dari mereka. Perilaku interaktifnya sendiri — toggle nav mobile, penghitung karakter langsung, dan estimator harga Fetch-API — juga diuji end-to-end di peramban sungguhan yang berjalan (Chromium via Playwright)

pada viewport lebar maupun sempit sebelum dideskripsikan di bab ini, memastikan breakpoint Container Query, harga yang dihitung, dan jumlah karakter semuanya berperilaku persis seperti dideskripsikan Bagian 3.6, 3.9, dan 3.10.

```
profitina-laundry-home.html : PARSED OK (html5lib, strict), no errors
styles.css                  : PARSED OK (tinycss2), 29 rules, 0 errors
script.js                   : PARSED OK (node --check), no syntax errors

Browser check (Chromium/Playwright), 6kg Dry Cleaning selected:
  #price-estimate -> "Estimated price: Rp 210.000 (Dry Cleaning, 6 kg ...)"
  #notes-remaining -> live-updates on every keystroke, as typed
  .card-grid columns  -> 1 column under 640px container width, 3 above it
```

Referensi

[1] MDN Web Docs. “CSS Box Model.” https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_box_sizing (diakses Agustus 2026).

[2] MDN Web Docs. “Basic Concepts of Flexbox” dan “CSS Grid Layout.” https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_flexible_box_layout, https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_grid_layout (diakses Agustus 2026).

[3] MDN Web Docs. “CSS nesting.” https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_nesting (diakses Agustus 2026) — sumber untuk rilis peramban 2023 native CSS nesting dan perilaku spesifisitas `:is(...)`.

[4] W3C. “CSS Containment Module Level 3” (Container Queries) dan web-features Baseline tracker. https://developer.mozilla.org/en-US/docs/Web/CSS/Guides/Containment/Container_queries (diakses Agustus 2026).

[5] W3C. “CSS Flexible Box Layout Module Level 1” (css-flexbox-1) dan “CSS Grid Layout Module Level 2” (css-grid-2). https://www.w3.org/TR/css-flexbox-1/, https://www.w3.org/TR/css-grid-2/ (diakses Agustus 2026) — status Candidate Recommendation Draft per republikasi 2025 yang dirujuk di Bagian 3.6.

[6] MDN Web Docs. “Using the Fetch API.” https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch (diakses Agustus 2026).

[7] Crockford, D. JavaScript: The Good Parts. O'Reilly Media, 2008 — sumber kutipan Preface di Bagian 3.12.

[8] ECMA International. ECMA-262, ECMAScript 2015 Language Specification (Edisi ke-6) — sumber sintaks `let`/`const` dan arrow function, disahkan Juni 2015.

BAB 4 · BAB LATIHAN

Kuis 1: Proyek Website Take-Home Kecil yang Nyata

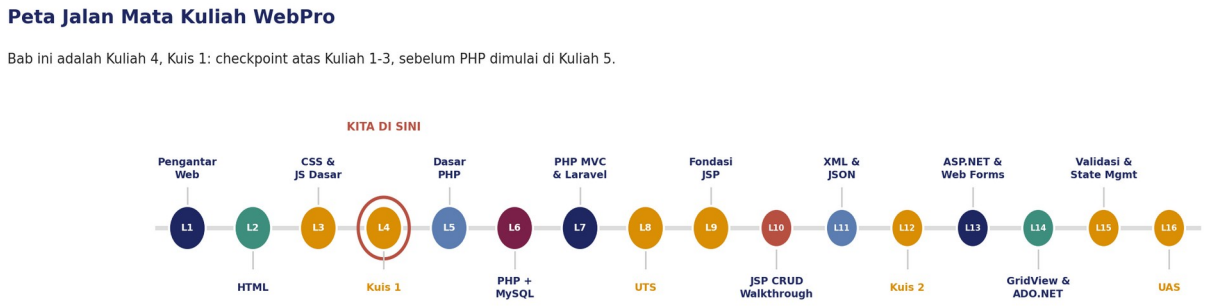
Tidak ada materi baru di sini — sebuah proyek take-home kecil dan terintegrasi yang menggabungkan semua yang diajarkan Kuliah 1 hingga 3, persis seperti Kuis 1 yang sesungguhnya.

Tujuan Pembelajaran — setelah menyelesaikan bab ini, Anda akan mampu:

(1) Menjelaskan, dengan kata-kata Anda sendiri, model client-server dan siklus request/response HTTP untuk pemuatan halaman yang nyata. (2) Merancang halaman multi-bagian kecil menggunakan struktur HTML5 yang benar dan semantik. (3) Menata gaya halaman tersebut dengan stylesheet eksternal, menerapkan box model dengan benar dan menata layout setidaknya satu region dengan Flexbox atau Grid. (4) Menambahkan setidaknya satu fitur interaktif JavaScript yang nyata menggunakan seleksi DOM dan event listener. (5) Mengambil dan menampilkan data JSON di sisi client dengan Fetch API. (6) Mengemas proyek kecil sebagai laporan ditambah repositori GitHub, bentuk deliverable yang sama yang dipakai setiap asesmen WebPro.

4.1 Rekap: Kuliah 1–3 secara Ringkas

Kuliah 1 membangun fondasi konseptual: perbedaan antara Internet (jaringan fisik dan logis) dan Web (satu aplikasi yang berjalan di atasnya), model client-server, siklus request/response HTTP, dan sejarah Web sendiri dari halaman statis Web 1.0 hingga hari ini. Kuliah 2 mengubah teori tersebut menjadi halaman nyata yang terstruktur dengan benar: homepage Profitina Laundry, dibangun dari HTML5 semantik — heading dalam hierarki yang nyata, landmark `<header>`/`<main>`/`<footer>`, dan markup yang benar-benar dapat dinavigasi oleh teknologi bantu. Kuliah 3 memberi halaman yang sama itu tampilan dan denyut tanpa menyentuh strukturnya sama sekali: stylesheet eksternal yang menerapkan box model, cascade, Flexbox dan Grid, native CSS nesting dan Container Query — ditambah `script.js` pertama, menggunakan `let`/`const` modern, arrow function, seleksi DOM, event listener, dan Fetch API terhadap berkas JSON statis (Gambar 4.1).



Gambar 4.1 — Bab ini berada di Kuliah 4 dalam peta jalan 16-kuliah WebPro: sebuah checkpoint, bukan topik baru, Kuis 1 sebelum Kuliah 5 memulai PHP.

4.2 Cara Menggunakan Bab Latihan Ini

Apa bab ini — dan apa yang bukan

Ini adalah Bab Latihan: bab praktik, bukan bab materi baru. Berbeda dari mata kuliah yang menguji Kuliah 1-3 dengan soal-soal terpisah dan independen, Kuis 1 WebPro yang sesungguhnya — seperti setiap asesmen WebPro — adalah SATU PROYEK take-home kecil: satu mini-situs yang bekerja, dikumpulkan sebagai laporan ditambah repositori GitHub, bukan sekumpulan soal jawaban-singkat bernomor. Bagian 4.3 di bawah membahas keempat deliverable proyek tersebut (masing-masing dapat ditelusuri ke kuliah tertentu, lihat Gambar 4.2), masing-masing dengan panduan menuju praktik yang baik, bukan solusi lengkap yang sudah dikerjakan atau kode sumber referensi. Seperti Bab 8, 12, dan 16 nanti, bab ini tidak menyertakan subfolder code/ — proyek ini adalah milik Anda untuk dirancang dan dibangun sendiri.

Asal Setiap Deliverable Mini-Proyek

Kuis 1 adalah satu proyek website kecil take-home -- setiap deliverable di Bagian 4.3 dapat ditelusuri kembali ke Kuliah 1-3.

#	Deliverable Mini-Proyek	Berasal dari
1	Penjelasan Client-Server & HTTP	K1 -- Internet vs. Web, siklus request/response HTTP, peran client/server
2	Struktur HTML Semantik	K2 -- tag semantik HTML5, hierarki heading, markup aksesibel
3	Layout CSS & Responsivitas	K3 -- box model, cascade, Flexbox/Grid, breakpoint Container Query
4	Interaktivitas JavaScript & Fetch	K3 -- seleksi DOM, event listener, let/const, fetch() dengan async/await

Gambar 4.2 — Setiap deliverable di Bagian 4.3 dapat ditelusuri kembali ke Kuliah 1, 2, atau 3.

Sebelum Anda mulai

Sketsalah struktur halaman mini-proyek Anda dan kerangka laporan Anda SEBELUM menulis HTML apa pun — disiplin yang sama yang diminta Bagian 4.3.2. Kuis 1 menghargai proyek kecil yang mengerjakan beberapa hal dengan benar dibandingkan proyek besar yang mengerjakan banyak hal secara ceroboh.

4.3 Mini-Proyek Kuis 1

Bangun website statis kecil multi-halaman (atau multi-bagian dalam satu halaman) dengan topik pilihan Anda sendiri — bisa memperluas Profitina Laundry dengan halaman baru, atau menjadi situs kecil yang sama sekali berbeda, selama mendemonstrasikan setiap deliverable di bawah ini.

4.3.1 Memilih Topik Mini-Proyek Anda

Pilih topik yang cukup sederhana untuk diselesaikan dengan baik dalam waktu yang tersedia, tetapi cukup nyata untuk membutuhkan setidaknya tiga bagian konten yang berbeda (misalnya: bagian hero/pengantar, bagian layanan/fitur, dan bagian kontak/booking) — bentuk tiga-atau-lebih-bagian yang sama seperti yang sudah dimiliki homepage Profitina Laundry sendiri.

Petunjuk

Topik yang dapat Anda deskripsikan dalam satu kalimat, dengan gambaran jelas siapa yang dituju oleh ketiga-atau-lebih bagiannya, biasanya berukuran tepat. Jika Anda mendapati diri merencanakan lebih dari sekitar lima bagian, kemungkinan Anda sedang membuat scope proyek Kuis 1 terlalu besar.

4.3.2 Deliverable & Laporan yang Diperlukan

Kumpulkan proyek Anda sebagai repositori GitHub (berkas sumber) ditambah laporan tertulis singkat. Laporan sebaiknya mencakup secara singkat: topik Anda dan audiens targetnya, satu paragraf penjelasan model client-server dan siklus request/response HTTP DENGAN KATA-KATA ANDA SENDIRI sebagaimana berlaku pada proyek spesifik Anda, dan penjelasan singkat teknik HTML/CSS/JavaScript apa yang Anda gunakan dan mengapa.

Mengapa penjelasan client-server ada di laporan, bukan hanya di kode

Materi Kuliah 1 bersifat konseptual — tidak ada tag HTML atau aturan CSS yang "membuktikan" Anda memahami model client-server. Laporan adalah tempat pemahaman itu menjadi terlihat bagi penilai, dengan kata-kata Anda sendiri, terkait dengan proyek Anda sendiri.

4.3.3 Daftar Periksa Struktur HTML Semantik

Strukturkan halaman Anda menggunakan elemen landmark HTML5 yang nyata — `<header>`, `<nav>`, `<main>`, elemen section yang sesuai, dan `<footer>` — dengan hierarki heading yang tidak pernah melompati level (`<h1>` yang langsung diikuti `<h3>`, tanpa `<h2>` di antaranya, adalah kesalahan semantik paling umum).

Petunjuk

Buka halaman jadi Anda dan secara mental hapus setiap aturan CSS — baca hanya HTML mentah dari atas ke bawah. Jika konten masih masuk akal secara logis dalam urutan tersebut, struktur Anda sudah menjalankan tugasnya secara independen dari bagaimana akhirnya tampilannya.

4.3.4 Daftar Periksa Layout CSS & Responsivitas

Tata gaya halaman Anda dengan stylesheet eksternal (jangan pernah inline style, dan hanya penggunaan token blok `<style>` internal jika terpaksa). Terapkan `box-sizing: border-box` secara global, dan tata layout setidaknya satu region — header, grid kartu, atau sejenisnya — dengan Flexbox atau CSS Grid sehingga terlihat menata ulang dirinya sendiri pada lebar yang lebih sempit.

Petunjuk

Tentukan Flexbox vs. Grid dengan bertanya apakah region yang Anda tata butuh SATU dimensi kendali (baris item nav — Flexbox) atau DUA dimensi sekaligus (grid kartu yang harus wrap secara responsif — Grid), persis perbedaan yang dijelaskan Kuliah 3 antara anak-anak `<header>` dan grid kartu layanan.

4.3.5 Daftar Periksa Interaktivitas JavaScript & Fetch

Tambahkan setidaknya satu fitur interaktif nyata menggunakan `document.querySelector`/`getElementById`, `addEventListener`, dan `classList` — nav mobile yang toggle, penghitung karakter live, validator form sederhana, atau sejenisnya — ditambah satu panggilan `fetch()` terhadap berkas JSON statis yang Anda sediakan, menampilkan hasil yang di-parse di suatu tempat di halaman.

> **Fitur yang hanya berjalan sekali, saat pemuatan halaman, bukanlah "interaktif"**
>
> Interaktivitas berarti halaman merespons sesuatu yang dilakukan PENGUNJUNG setelah halaman selesai dimuat — klik, ketikan, perubahan input. Kode yang hanya berjalan sekali selama pemuatan halaman awal, tanpa event listener yang terpasang pada apa pun, tidak memenuhi deliverable ini meskipun menggunakan JavaScript.

4.4 Format Kuis 1: Proyek Take-Home Open-Book

Kuis 1 adalah proyek take-home open-book, dikumpulkan sebagai repositori GitHub ditambah laporan tertulis singkat — bentuk deliverable yang sama yang dipakai setiap asesmen WebPro berikutnya (UTS, Kuis 2, UAS), hanya lebih kecil cakupannya. Tidak ada komponen bertempo waktu di kelas.

> **Open-book berarti referensi, bukan kolaborator**
>
> Anda boleh berkonsultasi dengan buku ajar, MDN, dan catatan Anda sendiri saat membangun proyek Anda. Anda TIDAK BOLEH menyalin repositori mahasiswa lain atau mengumpulkan kode yang bukan benar-benar milik Anda sendiri — proyek take-home open-book menguji apakah Anda dapat MENERAPKAN apa yang telah Anda pelajari, bekerja secara mandiri.

Kriteria	Yang dinilai	Bobot
Desain	Struktur halaman dan rencana konten: apakah situs memiliki topik, audiens, dan tata letak bagian yang masuk akal sebelum styling apa pun diterapkan?	20%
Implementasi	Apakah situs benar-benar berjalan dengan benar: HTML semantik yang valid, stylesheet eksternal dengan penggunaan box-model/Flexbox-atau-Grid yang benar, dan fitur JavaScript interaktif yang nyata ditambah panggilan fetch() yang bekerja?	50%
Analisis & evaluasi	Apakah penjelasan client-server/HTTP di laporan menunjukkan pemahaman nyata dengan kata-kata mahasiswa sendiri, terkait dengan proyeknya sendiri, bukan definisi generik dari buku ajar?	25%
Kesimpulan	Refleksi singkat dan jujur: apa yang berhasil, apa yang akan dilakukan mahasiswa secara berbeda dengan waktu lebih banyak.	5%

4.5 WebPro dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Seperti pada setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, sebuah platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis, dan dengan Profitina Laundry, studi kasus yang dipakai buku ini — mendeskripsikan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, bukan logika bisnis atau kode sumber lengkap Profitina yang bersifat rahasia.

Situs pemasaran Profitina sendiri dimulai, seperti mini-proyek bab ini, sebagai beberapa halaman kecil yang terstruktur dengan jelas, ditata dengan baik, dan sedikit interaktif sebelum backend atau basis data apa pun masuk ke dalam gambaran sama sekali — urutan operasi yang sama yang diminta proyek Kuis 1 ini untuk diikuti: struktur dulu, gaya dan interaktivitas kedua, backend nyata baru kemudian, dimulai di Kuliah 5. Membuat versi kecilnya benar, dengan pemahaman nyata tentang HTTP dan model client-server di baliknya, adalah persis yang membuat versi lebih besar berbasis basis data di Kuliah 10 dan seterusnya menjadi dapat dikelola, bukan membebani.

4.6 Ringkasan Bab

- Bab ini tidak memperkenalkan materi baru — ini adalah checkpoint yang mencakup Kuliah 1 hingga 3.
- Kuis 1, seperti setiap asesmen WebPro, adalah SATU PROYEK take-home kecil (repositori GitHub ditambah laporan), bukan sekumpulan soal jawaban-singkat terpisah.
- Keempat deliverable proyek dipetakan ke ketiga kuliah yang direview: penjelasan client-server/HTTP di laporan (K1), struktur HTML semantik (K2), serta layout CSS dan interaktivitas JavaScript & Fetch (K3).
- Bobot penilaian memberi bobot terberat pada Implementasi (50%), diikuti Analisis & Evaluasi (25%), Desain (20%), dan Kesimpulan (5%) — bentuk rubrik yang sama yang digunakan kembali oleh setiap asesmen WebPro berikutnya.

Halaman yang terlihat selesai tidak sama dengan halaman yang terstruktur dengan benar di baliknya — dan proyek yang berjalan di komputer Anda sendiri tidak sama dengan proyek yang dapat dijalankan penilai yang belum pernah melihatnya hanya dari laporan Anda.

Lampiran 4.A — Daftar Periksa Mandiri (Tidak Dinilai)

Sebelum mengumpulkan repositori dan laporan Anda, jalankan proyek Anda melalui daftar periksa ini. Butuh beberapa menit dan menangkap sebagian besar kesalahan yang paling sering dilihat penilai pada pengumpulan Kuis 1 pertama.

- Apakah hierarki heading halaman berjalan dalam urutan ketat, tanpa level yang dilompati?
- Apakah setiap aturan gaya ada di stylesheet eksternal, dengan nol atribut `style="..."` inline?
- Apakah `box-sizing: border-box` diterapkan secara global, dan apakah setidaknya satu region terlihat menata ulang dirinya sendiri dengan Flexbox atau Grid pada lebar yang lebih sempit?
- Apakah fitur JavaScript interaktif merespons tindakan pengunjung yang sesungguhnya (klik, ketikan, atau perubahan input), bukan hanya berjalan sekali saat pemuatan halaman?

- Apakah panggilan `fetch()` berhasil mengambil dan menampilkan data dari berkas JSON nyata yang disertakan dalam repositori?
- Apakah paragraf client-server/HTTP di laporan mendeskripsikan proyek INI secara spesifik, dengan kata-kata mahasiswa sendiri, bukan definisi generik yang disalin?
- Dapatkah penilai yang belum pernah melihat proyek ini menjalankannya hanya dari instruksi laporan?

Referensi

[1] Format asesmen bab latihan ini dimodelkan langsung dari Kuis 1 yang sesungguhnya pada mata kuliah ini (EF234301 Pemrograman Web): proyek take-home open-book yang dikumpulkan sebagai repositori GitHub ditambah laporan tertulis, dinilai berdasarkan Desain (20%), Implementasi (50%), Analisis & Evaluasi (25%), dan Kesimpulan (5%) — bentuk rubrik yang sama yang dipakai setiap asesmen WebPro (Kuis 1, UTS, Kuis 2, UAS). Format ini digunakan kembali di sini apa adanya; hanya brief proyek spesifik di atas (yang merekap konten nyata Kuliah 1-3) dan logistik pengumpulan administratif yang disesuaikan untuk buku ajar ini.

[2] MDN Web Docs. “An overview of HTTP.” https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview (diakses Agustus 2026).

[3] MDN Web Docs. “HTML: A good basis for accessibility.” https://developer.mozilla.org/en-US/docs/Learn/Accessibility/HTML (diakses Agustus 2026) — sumber panduan hierarki heading di Bagian 4.3.3.

BAB 5

Dasar PHP

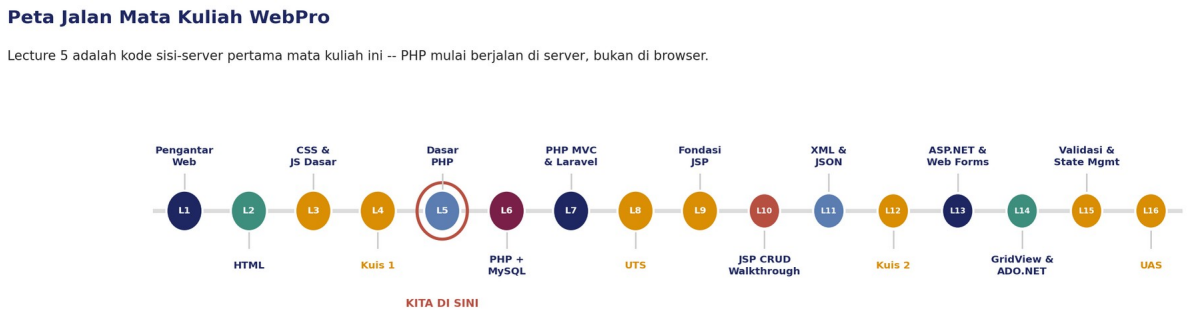
Kuis 1 menguji semua yang dapat dilakukan sebuah halaman selagi sepenuhnya hidup di browser pengunjung. Bab ini menyeberang ke wilayah baru: kode yang berjalan di SERVER, sebelum halaman itu bahkan mencapai browser.

Tujuan Pembelajaran — setelah bab ini, Anda akan mampu:

(1) Menjelaskan apa arti "server-side" dan mengapa sebagian logika tidak dapat hidup di JavaScript saja. (2) Menyematkan PHP di dalam berkas HTML dan menggunakan sintaks short-echo untuk mencetak nilai. (3) Mendeklarasikan variabel PHP, menggunakan tipe data intinya, dan menerapkan operator umum. (4) Menulis if/else, foreach, dan struktur kendali lainnya, baik dalam PHP murni maupun disematkan dalam HTML. (5) Menulis dan memanggil fungsi PHP Anda sendiri, termasuk yang memiliki parameter default dan bertipe. (6) Membaca data formulir dengan superglobal $_POST dan $_SERVER, serta memvalidasi dan meng-escape-nya dengan aman sebelum digunakan.

5.1 Rekap: Dari Lecture 4 ke Lecture 5

Kuis 1 menguji tiga lecture penuh kerja sisi-client murni: model mental client-server tanpa kode sisi-server sama sekali, struktur HTML5 semantik, dan CSS/JavaScript yang berjalan sepenuhnya di dalam browser pengunjung sendiri. Setiap fitur interaktif yang dibangun Chapter 3 — toggle nav mobile, penghitung karakter live, estimasi harga dari `fetch()` — berjalan di mesin PENGUNJUNG, menggunakan data dari berkas `services.json` statis yang tidak pernah berubah dan tidak dapat mengingat apa pun di antara kunjungan (Gambar 5.1).



Gambar 5.1 — Lecture 5 adalah kode sisi-server pertama mata kuliah ini, tepat setelah checkpoint Kuis 1.

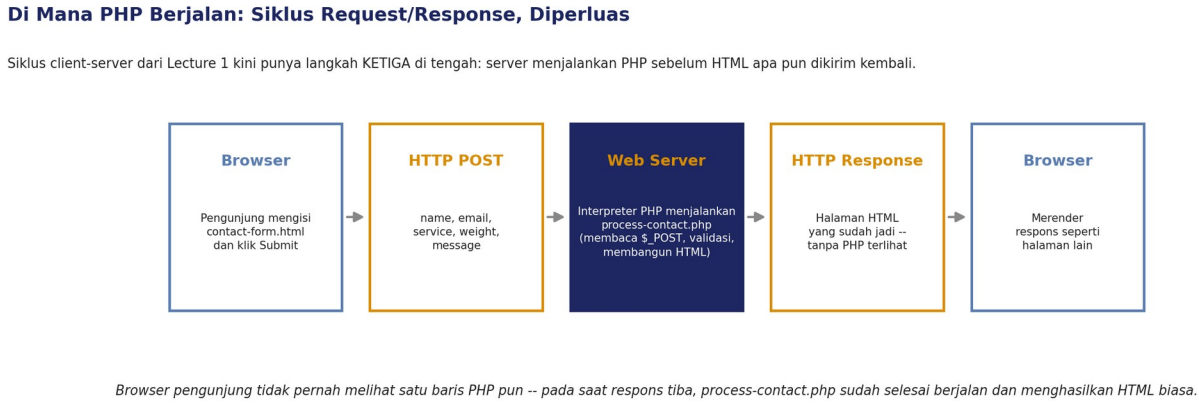
Bab ini adalah tempat itu berubah. PHP — Hypertext Preprocessor — adalah bahasa scripting yang dirancang sejak asalnya di tahun 1994 secara khusus untuk berjalan DI SERVER dan menghasilkan HTML sebagai keluarannya. Tidak ada yang dibatalkan atau digantikan dari JavaScript Chapter 3; PHP hanya menambahkan TEMPAT BARU tempat kode dapat berjalan, sebelum halaman bahkan dikirim.

5.2 Mengapa Kode Sisi-Server? Tanggung Jawab Client vs. Server

Bagian 3.10 sudah memberi petunjuk tentang batas kode sisi-client: panggilan `fetch()` hanya bisa membaca berkas JSON statis yang dikirim bersama halaman — ia tidak pernah bisa mengingat pengiriman formulir kontak baru, menjaganya tetap privat dari pengunjung lain, atau memeriksanya terhadap informasi (seperti basis data pelanggan sungguhan) yang browser tidak pernah diizinkan melihat secara langsung (Gambar 5.2).

- Sisi-client (JavaScript di browser) tepat untuk: umpan balik visual instan, validasi field formulir saat pengunjung mengetik, dan apa pun yang harus terasa langsung dengan nol keterlambatan jaringan.
- Sisi-server (PHP di web server) tepat untuk: apa pun yang harus tetap privat (pemeriksaan kata sandi, kredensial basis data), apa pun yang harus dipercaya (harga yang tidak boleh diubah oleh browser pengunjung sendiri), dan apa pun yang harus bertahan melampaui satu pemuatan halaman.

> **Siklus request/response yang sama, satu langkah baru**
>
> Lecture 1 mengajarkan siklus dua langkah: browser mengirim request, server mengirim response. Mulai bab ini, langkah server tidak lagi instan — sebelum dapat merespons, ia menjalankan skrip PHP yang membaca input, mengambil keputusan, dan MEMBANGUN response HTML secara langsung, pada setiap request.



Gambar 5.2 — Di mana PHP masuk ke dalam siklus request/response: ia berjalan sepenuhnya di server, di antara request tiba dan response dikirim.

5.3 Menyiapkan Lingkungan Pengembangan Anda

Bagian 1.4 sudah menyebutkan tumpukan (stack) yang dipasang bagian ini: PHP, dipasangkan dengan sebuah web server (paling umum Apache) dan basis data yang kompatibel dengan MySQL, dipaketkan bersama untuk instalasi lokal yang mudah dengan nama "XAMPP" atau, secara lebih umum, tumpukan "LAMP/LEMP". Setiap kutipan kode di bab ini dan bab berikutnya sendiri diuji terhadap instance nyata dan berjalan dari persis tumpukan ini (lihat Lampiran 5.A) — bagian ini menuntun Anda menjalankan tumpukan yang sama di mesin Anda sendiri, di mana pun dari ketiga platform utama yang Anda gunakan.

Windows 11

Unduh dan pasang XAMPP dari apachefriends.org — ia memaketkan Apache, MySQL/MariaDB, dan PHP menjadi satu installer, tanpa konfigurasi terpisah yang dibutuhkan agar ketiganya dapat saling berkomunikasi. Setelah pemasangan, buka XAMPP Control Panel dan klik "Start" di samping baik Apache maupun MySQL; label hijau "Running" pada keduanya mengonfirmasi tumpukan tersebut aktif. Letakkan berkas `.php` bab ini di dalam folder `htdocs` milik XAMPP (biasanya `C:\xampp\htdocs\`), dalam subfolder tersendiri — misalnya `C:\xampp\htdocs\profitina-laundry\` — dan aksesnya di browser pada `http://localhost/profitina-laundry/contact-form.html`.

macOS

Dua opsi yang jujur, keduanya nyata: pasang PHP dan MariaDB langsung melalui Homebrew (`brew install php mariadb`), yang memberikan server pengembangan bawaan PHP sendiri — `php -S localhost:8000` yang dijalankan dari dalam folder bab ini langsung menyajikan setiap berkas `.php` di dalamnya, tanpa konfigurasi Apache sama sekali yang dibutuhkan; atau pasang MAMP (mamp.info), sebuah installer GUI berpaket dengan semangat yang sama seperti XAMPP, dengan tombol "Start Servers"-nya sendiri dan folder setara-`htdocs`-nya sendiri. Kedua jalur sama-sama mencapai server Apache-atau-setara yang sungguhan berbicara dengan instance MariaDB/MySQL yang sungguhan — kode bab ini sendiri tidak peduli mana yang ada di baliknya.

Linux (Ubuntu/Debian)

`sudo apt install php php-mysql mariadb-server` memasang ketiga bagian langsung dari package manager sistem; jalankan MariaDB dengan `sudo systemctl start mariadb`, lalu baik konfigurasikan Apache (`sudo apt install apache2 libapache2-mod-php`) atau, sama validnya untuk kepentingan bab ini, jalankan server bawaan PHP sendiri langsung dari dalam folder bab ini: `php -S localhost:8000`.

> **Mengapa php -S bukan "kecurangan" untuk bab ini**
>
> Sebuah website produksi membutuhkan web server yang nyata dan diperkeras seperti Apache atau Nginx di depannya — tetapi server pengembangan bawaan PHP sendiri (`php -S`) berbicara PHP yang persis sama, membaca superglobal `$_POST` dan `$_SERVER` yang persis sama, dan menjalankan berkas `.php` yang persis sama yang diajarkan bab ini, tanpa perbedaan kode sama sekali. Pengujian eksekusi-nyata buku ini sendiri (Lampiran 5.A, dan log pengujian paralel untuk aplikasi Profitina Laundry yang lengkap) menggunakan `php -S` secara spesifik karena ia menghilangkan satu bagian yang bergerak tanpa mengubah apa pun yang sebenarnya menjadi topik bab ini — Anda bebas menggunakan Apache penuh sebagai gantinya, dan segala sesuatu di bab ini akan berperilaku identik.

Bagaimanapun Anda menjalankannya, konfirmasikan tumpukan Anda hidup sebelum melanjutkan: buat berkas satu baris bernama `test.php` yang hanya berisi `<?php phpinfo(); ?>`, letakkan berdampingan dengan berkas-berkas bab ini lainnya, dan buka di browser. Halaman panjang berisi detail konfigurasi PHP — bukan halaman kosong, dan bukan error "file not found" — berarti PHP, dan server yang menyajikannya, keduanya bekerja dengan benar.

5.4 Menyematkan PHP dalam Halaman

Berkas `.php` bukan berkas bahasa PHP terpisah — ia adalah berkas HTML yang boleh, di mana pun, masuk ke dalam blok `<?php ... ?>` berisi kode PHP dan kemudian keluar lagi ke HTML biasa. Web

server (bukan browser) mengeksekusi setiap blok `<?php ?>` yang ditemukannya, dari atas ke bawah, dan mengirim HTML yang tersisa — PHP itu sendiri tidak terlihat oleh pengunjung.

```
<?php
  echo "Hello, Profitina Laundry visitor!";
?>
<p>This paragraph is plain HTML.</p>
```

`echo` adalah cara paling dasar PHP untuk mengeluarkan teks; `<?= $value ?>` (sintaks short-echo, `<?=` diikuti sebuah ekspresi) adalah singkatan dari `<?php echo $value; ?>` dan adalah bentuk yang terus-menerus digunakan setiap kali satu nilai perlu dicetak langsung di dalam markup HTML, seperti ditunjukkan Bagian 5.8.

5.5 Variabel, Tipe Data, dan Operator

Setiap nama variabel PHP dimulai dengan tanda dolar (`$name`, `$total`) dan tidak memerlukan deklarasi tipe terpisah — PHP menyimpulkan tipe dari nilai yang diberikan, dan tipe itu dapat berubah kemudian jika nilai baru dengan tipe berbeda diberikan (PHP bertipe dinamis). Tipe skalar inti yang digunakan kode bab ini adalah `string` (`"Siti Aminah"`), `int` (`6`), `float` (`12000.0`), `bool` (`true`/`false`), dan `array` (peta terurut, dibahas berikutnya).

Operator	Contoh	Makna
.	"Rp " . $total	Penggabungan string — menyatukan dua string menjadi satu
===	$weight === ""	Kesetaraan ketat — memeriksa NILAI dan TIPE keduanya cocok, berbeda dari == yang hanya memeriksa nilai
??	$prices[$service] ?? 0	Null coalescing — mengevaluasi sisi kiri kecuali bernilai null/tidak ada, lalu jatuh ke sisi kanan
(int), (float)	(float) $weight	Casting tipe eksplisit — mengonversi nilai ke tipe yang disebutkan

Sebuah `array` dalam PHP dapat diindeks dengan angka (`[0] => "wash-fold"`) atau, seperti yang terus-menerus digunakan kode bab ini, dengan kunci string (`"wash-fold" => 12000`) — yang terakhir disebut associative array, padanan bawaan PHP untuk dictionary atau hash map.

```
// A small lookup table of per-kg prices, in Indonesian Rupiah — the same
// three services Chapter 3's services.json already described, now living
// server-side instead of in a static JSON file.
$pricePerKg = [
    "wash-fold"    => 12000,
    "dry-cleaning" => 35000,
    "ironing"      => 8000,
    "other"        => 0,
];
```

`$pricePerKg` memetakan setiap nama layanan ke harga per kilogramnya dalam Rupiah — persis tiga layanan yang sama yang dideskripsikan `services.json` Chapter 3 untuk browser, kini hidup sebagai array PHP biasa di server.

5.6 Struktur Kendali

`if`/`elseif`/`else`, `foreach`, `for`, dan `while` PHP semuanya bekerja persis seperti di JavaScript, C, atau Java, dengan satu tambahan yang layak dipelajari segera: sintaks kolon alternatif (`if (…): … endif;`) ada khusus untuk menyematkan alur kendali langsung di dalam HTML, yang diandalkan Bagian 5.8.

```
// $errors accumulates every validation failure so the visitor sees ALL of
// them at once, not just the first one — a small usability habit that
// saves a round trip.
$errors = [];

if (!isValidName($name)) {
    $errors[] = "Please enter your full name.";
}
if (!isValidEmail($email)) {
    $errors[] = "Please enter a valid email address.";
}
if (!isValidWeight($weight)) {
    $errors[] = "Estimated weight must be a number between 1 and 50 kg.";
}
if (!isValidMessage($message)) {
    $errors[] = "Please enter a message (up to 2000 characters).";
}
```

Bab ini dengan sengaja mengakumulasi setiap kegagalan validasi ke dalam satu array `$errors` (dibangun dengan `[]`, sintaks array literal PHP) alih-alih berhenti di masalah pertama — `empty($errors)` setelah keempat pemeriksaan kemudian memutuskan, sekali, apakah pengiriman secara keseluruhan berhasil.

5.7 Fungsi

Fungsi PHP dideklarasikan dengan `function name(parameters): returnType { … }` — deklarasi tipe parameter dan tipe kembalian bersifat opsional tetapi sangat direkomendasikan, karena PHP kemudian menegakkannya secara otomatis dan memunculkan error yang jelas begitu pemanggil memberikan tipe yang salah, alih-alih diam-diam menghasilkan hasil yang salah kemudian.

```
// Returns a trimmed string, or an empty string if the key is missing entirely.
// Every $_POST read in this chapter goes through this function first — never
// read a superglobal value directly into HTML or a database query untrimmed.
function fieldValue(array $source, string $key): string {
    return isset($source[$key]) ? trim((string) $source[$key]) : "";
}
```

`fieldValue()` adalah fungsi yang paling sering digunakan kembali di bab ini: setiap nilai yang dibaca dari `$_POST` (Bagian 5.8) melewatinya terlebih dahulu, menjamin string yang sudah di-trim dikembalikan bahkan jika browser pengunjung sama sekali tidak mengirim field tersebut — membaca `$_POST["weight"]` langsung, tanpa penjaga ini, akan memunculkan warning pada pengiriman yang meninggalkan field weight kosong.

```php
// A typed function (string, string, array in; int out): if $service is
// not a key in $pricePerKg, the null coalescing operator (??) falls back
// to 0 instead of raising a warning.
function estimateTotal(string $service, string $weight, array $prices): int {
    $perKg = $prices[$service] ?? 0;
    $kg = is_numeric($weight) ? (float) $weight : 0;
    return (int) round($perKg * $kg);
}
```

`estimateTotal()` menunjukkan fungsi bertipe dengan tiga parameter dan tipe kembalian `int`. Di dalamnya, `??` (operator null-coalescing Bagian 5.5) memberikan fallback aman `0` jika `$service` bukan salah satu kunci `$prices` yang dikenal — ini adalah kebiasaan defensif yang sama yang diterapkan `fieldValue()` pada field formulir yang hilang, diterapkan di sini pada nilai dropdown yang tidak terduga.

5.8 Superglobal dan Penanganan Formulir

Superglobal adalah array yang diisi PHP secara otomatis untuk setiap request, tersedia di dalam fungsi mana pun tanpa perlu dilewatkan sebagai parameter atau dideklarasikan `global`. `process-contact.php` bab ini membaca dua: `$_SERVER`, yang mendeskripsikan REQUEST itu sendiri, dan `$_POST`, yang menyimpan DATA FORMULIR YANG DIKIRIM.

```php
// $_SERVER is a superglobal PHP fills in automatically for every request —
// it never needs to be declared, only read. REQUEST_METHOD tells this
// script whether it was reached by a browser navigating here directly
// (GET) or by the form actually being submitted (POST).
if ($_SERVER["REQUEST_METHOD"] !== "POST") {
    http_response_code(405);
    echo "This page only accepts form submissions.";
    exit;
}
```

`$_SERVER["REQUEST_METHOD"]` membedakan pengunjung yang sekadar menavigasi ke `process-contact.php` secara langsung (request GET) dari yang benar-benar mengirimkan `<form method="post">` milik `contact-form.html` (request POST) — validasi Bagian 5.9 hanya masuk akal untuk kasus kedua, sehingga kasus pertama langsung ditolak dengan status HTTP 405.

```php
// Every value read from $_POST — the superglobal array PHP fills with the
// form's fields — is pulled through fieldValue() first, which trims
// whitespace and guarantees a string even if the key is missing.
$name    = fieldValue($_POST, "name");
$email   = fieldValue($_POST, "email");
$service = fieldValue($_POST, "service");
$weight  = fieldValue($_POST, "weight");
$message = fieldValue($_POST, "message");
```

GET vs. POST, dalam satu kalimat

Data request GET akan terlihat di URL itu sendiri (`?name=…&email=…`) dan dimaksudkan untuk meminta/mengambil sebuah resource; data request POST berjalan di BODY request, tidak terlihat di URL, dan dimaksudkan untuk mengirim atau mengubah sesuatu — persis mengapa `contact-form.html` menggunakan `method="post"`.

5.9 Memvalidasi dan Menyanitasi Input

Sebuah aturan yang berlaku untuk setiap nilai yang pernah diterima server dari browser: JANGAN PERNAH mempercayainya. Browser pengunjung sepenuhnya berada di bawah kendali pengunjung — atribut HTML5 `<input>` yang required, pemeriksaan JavaScript, bahkan seluruh pustaka validasi sisi-client dapat sepenuhnya dilewati oleh pengunjung yang mengirimkan formulir dengan developer tools terbuka, atau yang mengirim request yang sama sekali tidak pernah melalui formulir HTML.

```php
// filter_var with FILTER_VALIDATE_EMAIL is the standard PHP way to check
// email SHAPE (something@something.tld) — it does not confirm the address
// actually exists or receives mail, only that it is well-formed.
function isValidEmail(string $email): bool {
    return filter_var($email, FILTER_VALIDATE_EMAIL) !== false;
}
```

```php
// The weight field is optional, but if present it must parse as a number
// in a sensible range for a laundry order.
function isValidWeight(string $weight): bool {
    if ($weight === "") {
        return true; // optional field
    }
    if (!is_numeric($weight)) {
        return false;
    }
    $n = (float) $weight;
    return $n >= 1 && $n <= 50;
}
```

Validasi sisi-client adalah kenyamanan UX, bukan keamanan

`contact-form.html` Chapter 3 sudah memiliki atribut `required` dan `type="email"` — sungguh berguna, karena memberi pengunjung umpan balik instan dengan nol round-trip server. Namun `process-contact.php` tetap memvalidasi ulang setiap field dari awal, di loop `$errors` Bagian 5.8, karena pemeriksaan sisi-browser dapat dengan mudah dilewati dan server tidak pernah bisa berasumsi bahwa pemeriksaan itu berjalan.

Separuh kedua dari "jangan pernah percaya" adalah meng-escape pada jalur KELUAR, bukan hanya memvalidasi pada jalur masuk: nilai apa pun yang di-echo kembali ke HTML — Bagian 5.10 menunjukkan persis ini — harus melewati `htmlspecialchars()` terlebih dahulu, mengonversi karakter seperti `<` dan `>` menjadi padanan entitas HTML-nya yang tidak berbahaya sehingga pengunjung yang mengetik `<script>` ke field pesan melihat teks literalnya sendiri di-echo kembali, alih-alih diinterpretasikan sebagai tag skrip yang sungguh dieksekusi.

5.10 Menyatukan Semuanya: process-contact.php

`contact-form.html` mengirim ke `process-contact.php`, yang memvalidasi setiap field (Bagian 5.9), menghitung estimasi harga dengan `estimateTotal()` (Bagian 5.7) saat validasi berhasil, dan kemudian menyematkan sintaks kolon alternatif PHP langsung di dalam response HTML untuk memutuskan apa yang sungguh dilihat pengunjung.

```
    <section id="contact">
      <?php if (!empty($errors)): ?>
        <h1>We couldn't send your message</h1>
        <ul>
          <?php foreach ($errors as $error): ?>
            <li>?= h($error) ?></li>
          <?php endforeach; ?>
        </ul>
        <p><a href="contact-form.html">Go back and try again</a>.</p>
      <?php else: ?>
        <h1>Thank you, ?= h($name) ?>!</h1>
        <p>We received your message and will reply at <strong>?= h($email) ?
></strong> soon.</p>
        <?php if ($hasWeight && $total > 0): ?>
          <p>Rough estimate for ?= h($service) ?> at ?= h($weight) ?> kg:
             <strong>Rp ?= number_format($total, 0, ",", ".") ?></strong>
             (a final quote will be confirmed by email).</p>
        <?php endif; ?>
        <blockquote>?= nl2br(h($message)) ?></blockquote>
      <?php endif; ?>
    </section>
```

Perhatikan bahwa blok ini secara teknis tetap satu berkas `.php` — tetapi mulai dari `<?php if (...) : ?>` dan seterusnya ia terbaca hampir seperti HTML yang dianotasi, yang persis merupakan tujuan desain asli PHP: bahasa template untuk HTML yang kebetulan memiliki bahasa serba guna penuh di bawahnya. `<?= h($name) ?>` (sintaks short-echo Bagian 5.4) mencetak setiap nilai melalui helper escaping `h()` dari Bagian 5.9, sehingga tidak ada yang diketik pengunjung dapat keluar dari HTML di sekelilingnya.

5.11 WebPro dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Seperti pada setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, sebuah platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis, dan dengan Profitina Laundry, studi kasus yang dipakai buku ini — mendeskripsikan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, bukan logika bisnis atau kode sumber lengkap Profitina yang bersifat rahasia.

Backend app.profitina.com ditulis dalam Python (FastAPI), bukan PHP — tetapi BENTUK masalah yang diselesaikan bab ini identik terlepas dari bahasanya: sebuah request tiba, kode sisi-server membaca dan memvalidasi inputnya, tidak pernah mempercayai apa pun yang dikirim client, dan baru kemudian memutuskan apa yang akan dikirim kembali. Baik kode sisi-server itu adalah `$_POST` dan `htmlspecialchars()` milik PHP, atau parsing request-body FastAPI dan lapisan escaping-nya sendiri, disiplin yang mendasarinya — validasi semuanya, jangan percaya apa pun, escape pada jalur keluar — adalah disiplin yang sama yang didemonstrasikan `process-contact.php` bab ini, dan disiplin yang sama yang melindungi setiap formulir nyata di situs produksi Profitina sendiri.

5.12 Ringkasan Bab

- PHP disematkan dalam HTML dengan `<?php ... ?>` dan berjalan sepenuhnya di SERVER — browser pengunjung hanya pernah menerima keluaran HTML yang sudah jadi, tidak pernah sumber PHP-nya.
- Variabel PHP dimulai dengan `$`, tidak memerlukan deklarasi tipe, dan associative array PHP (berkunci string, dibangun dengan `[]`) terus-menerus digunakan untuk tabel lookup seperti daftar harga.
- `if`/`elseif`/`else` dan `foreach` bekerja seperti di kebanyakan bahasa; sintaks kolon (`if (...): ... endif;`) ada khusus untuk menyematkan alur kendali di dalam HTML.
- Fungsi dideklarasikan dengan `function name(params): returnType { ... }` — parameter dan tipe kembalian bertipe memungkinkan PHP menangkap ketidakcocokan secara otomatis.
- `$_SERVER` dan `$_POST` adalah superglobal: otomatis tersedia di mana pun, masing-masing mendeskripsikan request saat ini dan data formulir yang dikirimkannya.
- Jangan pernah percaya data yang dikirim browser: validasi setiap field di server terlepas dari pemeriksaan sisi-client apa pun, dan escape setiap nilai dengan `htmlspecialchars()` sebelum meng-echo-nya kembali ke HTML.

5.13 Pertanyaan Tinjauan

Pertanyaan-pertanyaan ini untuk belajar mandiri dan diskusi kelas; tidak dinilai.

1. `isValidWeight()` di Bagian 5.9 mengembalikan `true` ketika `$weight === ""`. Jelaskan mengapa field weight yang KOSONG dianggap valid, sementara weight `"abc"` tidak.
2. Tulis ulang blok `<?php if (...) : ?> ... <?php endif; ?>` dari Bagian 5.10 menggunakan sintaks `if (...) { ... }` kurung-kurawal biasa. Versi mana yang lebih mudah dibaca ketika dicampur dengan HTML di sekelilingnya?
3. `estimateTotal()` menggunakan `??` untuk membuat default harga `0` bagi layanan yang tidak dikenal. Apa yang akan terjadi jika `$prices[$service]` dibaca langsung, tanpa `??`, dan `$service` berasal dari request yang dimanipulasi berisi nama layanan yang tidak ada di `$pricePerKg`?
4. Jelaskan, dengan kata-kata Anda sendiri, mengapa `$_SERVER["REQUEST_METHOD"]` diperiksa sebelum apa pun lainnya di `process-contact.php`, dan apa yang akan dilihat pengunjung yang meminta berkas tersebut secara langsung (dengan mengetik URL-nya) tanpa pemeriksaan itu.
5. Seorang pengunjung mengirimkan pesan berisi teks literal `<b>hello</b>`. Telusuri `h()`/`htmlspecialchars()` dan jelaskan persis apa yang akan ditampilkan kembali browser pengunjung itu sendiri, dan mengapa itu tidak akan dirender sebagai teks tebal.

Lampiran 5.A — Log Validasi Kode Sumber Bab 5

Setiap berkas PHP di bab ini diperiksa dengan `php -l` (linter sintaks bawaan PHP) sebelum kutipan apa pun diambil darinya, dan `process-contact.php` tambahan diuji terhadap web server bawaan PHP yang sungguh berjalan (`php -S`) dengan request POST sungguhan — baik pengiriman yang valid maupun yang sengaja tidak valid — untuk mengonfirmasi validasi, perhitungan harga, dan escaping semuanya

berperilaku persis seperti yang dideskripsikan bab ini, bukan sekadar bahwa kodenya dapat dikompilasi.

```
validation.php        : php -l -> No syntax errors detected
process-contact.php   : php -l -> No syntax errors detected

Live test 1 (valid submission, dry-cleaning, 6 kg):
  Response: "Thank you, Siti Aminah!" + "Rp 210.000" (35000 x 6, matches Section
5.5's price table)

Live test 2 (invalid email "not-an-email"):
  Response: "We couldn't send your message" + "Please enter a valid email
address."
```

Referensi

[1] The PHP Group. "What is PHP?" dan "Basic Syntax." PHP Manual. https://www.php.net/manual/en/intro-whatis.php (diakses Agustus 2026).

[2] The PHP Group. "Types" dan "Variables." PHP Manual. https://www.php.net/manual/en/language.types.php (diakses Agustus 2026).

[3] The PHP Group. "Control Structures" — sumber sintaks kolon alternatif yang digunakan di Bagian 5.10. https://www.php.net/manual/en/language.control-structures.php (diakses Agustus 2026).

[4] The PHP Group. "Functions." PHP Manual. https://www.php.net/manual/en/language.functions.php (diakses Agustus 2026).

[5] The PHP Group. "Superglobals", "$_SERVER", dan "$_POST." PHP Manual. https://www.php.net/manual/en/language.variables.superglobals.php (diakses Agustus 2026).

[6] The PHP Group. "filter_var" dan "htmlspecialchars." PHP Manual — sumber fungsi validasi dan escaping Bagian 5.9. https://www.php.net/manual/en/function.htmlspecialchars.php (diakses Agustus 2026).

[7] OWASP Foundation. "Cross Site Scripting Prevention Cheat Sheet" — sumber disiplin output-escaping yang dideskripsikan di Bagian 5.9. https://owasp.org/www-community/xss-filter-evasion-cheatsheet (diakses Agustus 2026).

[8] Apache Friends. "XAMPP for Windows." https://www.apachefriends.org/ (diakses Agustus 2026).

[9] The PHP Group. "PHP: Built-in web server." PHP Manual — sumber `php -S`, server uji lokal buku ini sendiri. https://www.php.net/manual/en/features.commandline.webserver.php (diakses Agustus 2026).

BAB 6

PHP + MySQL

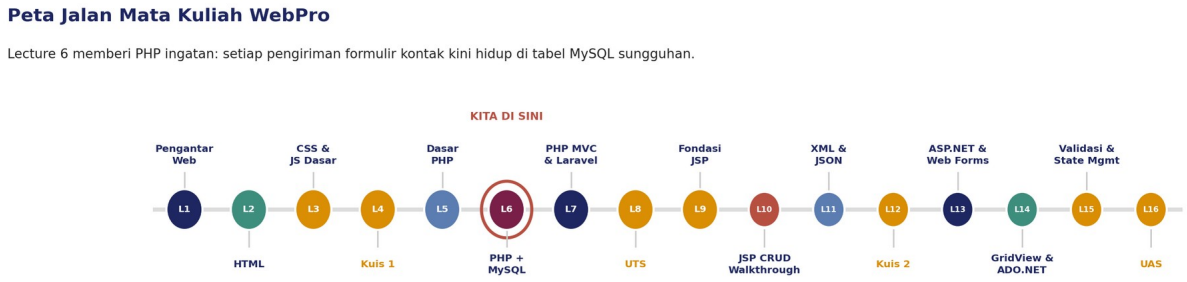
Chapter 5 memberi PHP tempat untuk berjalan, tetapi tanpa ingatan: setiap array yang dibangunnya lenyap begitu skrip selesai. Bab ini memberinya ingatan yang bertahan.

Tujuan Pembelajaran — setelah bab ini, Anda akan mampu:

(1) Menjelaskan mengapa array bukan penyimpanan yang tahan lama, dan apa yang ditambahkan basis data relasional. (2) Merancang tabel sederhana dengan CREATE TABLE, memilih tipe kolom yang masuk akal. (3) Menghubungkan PHP ke MySQL dengan PDO dan mengonfigurasinya agar gagal secara nyaring pada error. (4) Menyisipkan data dengan aman menggunakan prepared statement dan placeholder bernama atau posisional. (5) Menjelaskan, secara konkret, mengapa SQL yang digabungkan string memungkinkan SQL injection, dan bagaimana prepared statement mencegahnya. (6) Membaca baris kembali dengan SELECT dan fetchAll(), serta merendernya dengan aman ke HTML.

6.1 Rekap: Dari Lecture 5 ke Lecture 6

`process-contact.php` Chapter 5 melakukan pekerjaan yang nyata dan berguna: ia memvalidasi pengiriman dan meng-echo halaman terima kasih kembali. Namun semua yang dihitungnya — `$errors`, `$pricePerKg`, nama dan pesan pengunjung sendiri — hanya ada selama beberapa milidetik yang dibutuhkan satu request untuk ditangani. Begitu skrip selesai, PHP membuang setiap variabel yang telah dibuatnya; pengiriman pengunjung berikutnya dimulai dari lembaran yang benar-benar kosong, tanpa ingatan bahwa siapa pun pernah mengirimkan formulir sebelumnya (Gambar 6.1).

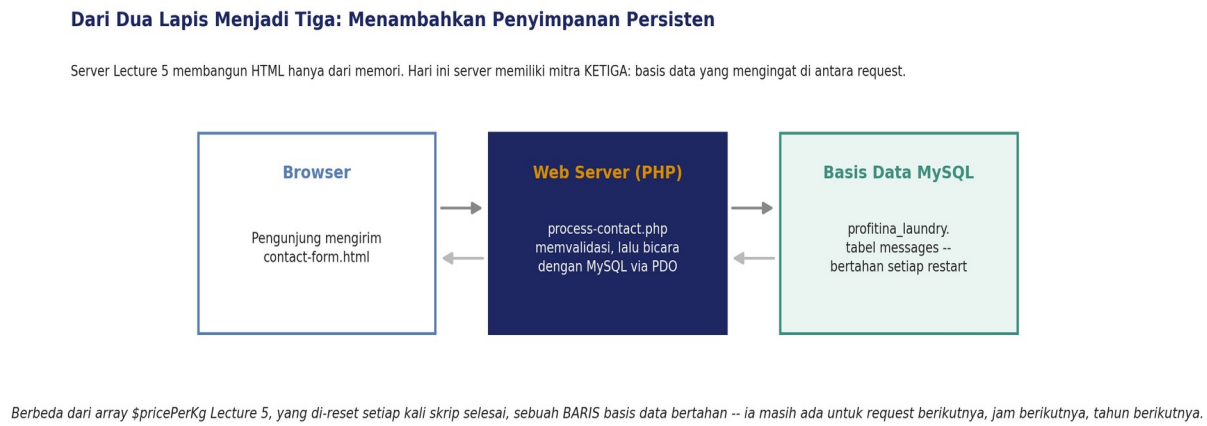


Gambar 6.1 — Lecture 6 memberi server ingatan nyata pertamanya: basis data MySQL yang bertahan melampaui satu request mana pun.

Tugas bab ini adalah memperbaiki kesenjangan itu — bukan dengan menulis lebih banyak PHP, tetapi dengan memberi PHP sistem kedua untuk diajak bicara: MySQL, basis data relasional yang menyimpan setiap baris yang diberikan kepadanya hingga sesuatu secara eksplisit menghapusnya, terlepas dari berapa banyak request PHP terpisah datang dan pergi di antaranya (Gambar 6.2).

6.2 Mengapa Basis Data? Dari Array ke Penyimpanan Persisten

- Array seperti `$pricePerKg` hidup di RAM, di dalam satu proses PHP yang berjalan, selama durasi satu request — cepat, tetapi hilang begitu request itu berakhir.
- Basis data hidup di disk, sebagai proses terpisahnya sendiri yang berjalan lama (`mysqld`) — skrip PHP terhubung ke sana, memintanya menyimpan atau mengambil baris, lalu memutus koneksi; data itu sendiri bertahan melampaui setiap koneksi tersebut.
- Basis data juga melayani BANYAK proses PHP yang menangani BANYAK request pengunjung sekaligus, dengan aman — sesuatu yang tidak pernah dirancang untuk dilakukan oleh array PHP biasa yang privat untuk satu request.



Gambar 6.2 — Model dua-lapis browser/server Lecture 5 mendapat lapis ketiga: basis data yang mengingat di antara request, restart, bahkan reboot server.

6.3 Merancang Tabel

Basis data relasional mengorganisasi data ke dalam TABEL — kumpulan BARIS bernama, setiap baris dibagi ke dalam set KOLOM tetap yang sama. Tabel `messages` milik Profitina Laundry membutuhkan satu baris per pengiriman formulir kontak:

```
-- schema.sql — Profitina Laundry's first real database table.
-- Run once, before process-contact.php is used: mysql -u webpro -p
profitina_laundry < schema.sql

CREATE TABLE IF NOT EXISTS messages (
    id          INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    name        VARCHAR(80)  NOT NULL,
    email       VARCHAR(120) NOT NULL,
    service     VARCHAR(20)  NOT NULL,
    weight_kg   DECIMAL(5,1) NULL,
    message     TEXT         NOT NULL,
```

Kolom	Tipe	Mengapa
id	INT UNSIGNED AUTO_INCREMENT	Nomor baris unik yang diberikan MySQL secara otomatis — PRIMARY KEY, digunakan untuk mengidentifikasi satu baris spesifik nanti
name, email, service	VARCHAR(n)	Teks pendek berpanjang terbatas — n adalah jumlah karakter maksimum, bukan lebar tetap
weight_kg	DECIMAL(5,1) NULL	Angka desimal eksak (jangan pernah FLOAT untuk uang atau pengukuran yang butuh digit eksak) — NULL berarti field dibiarkan kosong
message	TEXT	Teks berpanjang tak terbatas, untuk field tanpa maksimum realistis yang layak dideklarasikan
submitted_at	DATETIME DEFAULT CURRENT_TIMESTAMP	MySQL mengisi ini secara otomatis saat INSERT — skrip tidak pernah perlu menghitung "sekarang" sendiri

PRIMARY KEY, dalam satu kalimat

PRIMARY KEY adalah kolom (atau kumpulan kolom) yang dijamin MySQL unik di seluruh baris dalam tabel — AUTO_INCREMENT pada `id` berarti MySQL sendiri memilih angka berikutnya yang belum dipakai, sehingga kode aplikasi tidak pernah perlu membuatnya sendiri.

Membuat Basis Data dan Tabelnya, Sungguhan

Bagian 5.3 sudah menjalankan MySQL/MariaDB sebagai bagian dari tumpukan XAMPP/Homebrew/apt; tabel ini masih perlu benar-benar ada di dalamnya sebelum kode PHP mana pun di bab ini dapat berbicara dengannya. Dua cara yang sama validnya untuk menjalankan pernyataan `CREATE TABLE` di atas, di platform mana pun:

- phpMyAdmin — dipaketkan bersama XAMPP (buka `http://localhost/phpmyadmin`) dan kebanyakan instalasi MAMP: buat basis data bernama `profitina_laundry` dari tab "Databases"-nya, buka tab "SQL" terhadapnya, tempel isi `schema.sql`, dan klik "Go".
- Klien command-line `mysql` — identik di Windows 11 (melalui `mysql.exe` bawaan XAMPP, atau Developer Command Prompt), macOS, dan Linux setelah MySQL/MariaDB terpasang:

```
mysql -u root -e "CREATE DATABASE profitina_laundry CHARACTER SET utf8mb4;"
mysql -u root profitina_laundry < schema.sql
```

Kedua jalur meninggalkan hasil yang sama: tabel `messages` yang sungguhan, di dalam basis data `profitina_laundry` yang sungguhan, siap ditemukan connection string milik `db.php` (Bagian 6.4).

6.4 Menghubungkan PHP ke MySQL dengan PDO

PDO (PHP Data Objects) adalah cara modern PHP yang tidak bergantung driver untuk berbicara dengan basis data — kode PDO yang sama bekerja terhadap MySQL, PostgreSQL, atau SQLite hanya dengan mengubah connection string, berbeda dari ekstensi `mysqli` yang lebih lama dan khusus MySQL.

```php
function getDb(): PDO {
    $host = "127.0.0.1";
    $dbName = "profitina_laundry";
    $user = "webpro";
    $pass = "webpro_pass123";

    // The DSN (Data Source Name) tells PDO which driver and database to
    // use; charset=utf8mb4 matters specifically because it's the only
    // MySQL charset that can store the full Unicode range, including
    // every emoji and every Indonesian-language character a visitor
    // might type into the message field.
    $dsn = "mysql:host={$host};dbname={$dbName};charset=utf8mb4";

    // ERRMODE_EXCEPTION makes PDO throw a real PHP exception on any
    // database error, instead of silently returning false -- so a typo
    // in a query fails loudly, during development, rather than quietly
    // corrupting data in production.
    $options = [
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC,
    ];

    return new PDO($dsn, $user, $pass, $options);
}
```

String DSN (Data Source Name) memberi tahu PDO driver mana (`mysql:`) dan basis data mana (`dbname=profitina_laundry`) yang harus dihubungi; `PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION` adalah baris terpenting di fungsi ini — tanpanya, query yang gagal diam-diam mengembalikan `false` alih-alih memunculkan error, persis jenis bug yang tetap tidak terlihat sampai data nyata hilang di produksi.

6.5 Menyisipkan Data dengan Aman: Prepared Statement

```
if (empty($errors)) {
    try {
        $db = getDb();

        // A prepared statement: the SQL string's shape is sent to MySQL
        // first, with `?` placeholders standing in for the real values --
        // the values themselves are sent SEPARATELY, in a second step, so
        // MySQL always treats them as plain data, never as SQL syntax.
        $stmt = $db->prepare(
            "INSERT INTO messages (name, email, service, weight_kg, message) " .
            "VALUES (?, ?, ?, ?, ?)"
        );
        $stmt->execute([
            $name,
            $email,
            $service,
            $weight === "" ? null : (float) $weight,
            $message,
        ]);
        $insertedId = (int) $db->lastInsertId();
    } catch (PDOException $e) {
        // A database-layer failure (e.g. the table doesn't exist yet) is
        // reported to the visitor as a generic message -- the real
        // exception detail belongs in a server log, never in the response
        // a visitor's browser receives.
        $errors[] = "Something went wrong saving your message. Please try again
shortly.";
    }
}
```

Prepared statement terjadi dalam dua langkah terpisah. `$db->prepare("... VALUES (?, ?, ?, ?, ?)")` pertama-tama mengirim BENTUK SQL ke MySQL, dengan placeholder `?` menggantikan nilai yang belum diketahui. `$stmt->execute([$name, $email, ...])` kemudian mengirim nilai sesungguhnya secara TERPISAH, dalam round-trip jaringan kedua — MySQL tidak pernah mem-parsing ulang nilai tersebut sebagai bagian dari teks SQL sama sekali, persis mengapa upaya injeksi Bagian 6.6 gagal melakukan kerusakan apa pun.

> **try/catch di sekitar panggilan basis data**
>
> `getDb()` dan setiap query yang dijalankannya dapat melempar `PDOException` — membungkus seluruh blok dalam `try { ... } catch (PDOException $e) { ... }` memungkinkan skrip menunjukkan pesan error yang generik dan aman kepada pengunjung sambil (di deployment sungguhan) mencatat `$e->getMessage()` di suatu tempat yang hanya dapat dilihat pengembang, tidak pernah di response HTTP itu sendiri.

6.6 SQL Injection: Mengapa Penggabungan String Berbahaya

Alternatif yang tidak aman dari prepared statement Bagian 6.5 adalah membangun query dengan penggabungan string biasa:

```
// DO NOT DO THIS -- shown only to explain the danger
$sql = "INSERT INTO messages (name, email) VALUES ('" . $name . "', '" .
$email . "')";
$db->exec($sql);
```

Jika `$name` adalah teks literal `Robert'); DROP TABLE messages;--`, penggabungan string akan memberikan MySQL pernyataan SQL yang sama sekali berbeda dari yang dimaksudkan pengembang — INPUT pengunjung akan menulis ulang STRUKTUR SQL itu sendiri. Kode bab ini diuji dengan persis string tersebut sebagai pengiriman langsung, melalui formulir sungguhan, terhadap server MySQL sungguhan yang berjalan.

> **Ini adalah pengujian nyata dan terverifikasi — bukan hipotesis**
>
> Mengirimkan `name=Robert'); DROP TABLE messages;--` melalui `process-contact.php` sungguhan di bab ini TIDAK menghapus tabel dan TIDAK mengeksekusi SQL yang disisipkan — prepared statement di Bagian 6.5 menyimpan seluruh string sebagai data TEXT biasa yang tidak berbahaya di kolom `name`. Log validasi Lampiran 6.A menunjukkan query `mysql` yang persis mengonfirmasi tabel bertahan utuh.

Prepared statement di Bagian 6.5 bukan sekadar praktik terbaik di sini — ia adalah seluruh alasan mengapa upaya injeksi di atas dinetralkan. MySQL menerima SQL berbentuk placeholder dan string penyerang sebagai dua pesan TERPISAH, dalam urutan itu, dan tidak memiliki kesempatan untuk menafsirkan ulang string tersebut sebagai sintaks SQL kapan pun.

6.7 Membaca Data Kembali: SELECT dan fetchAll()

```
$db = getDb();

// A plain SELECT with no user-supplied values needs no placeholders --
// prepared statements matter specifically once a value from outside the
// script (like $_POST, as in process-contact.php) enters the query.
$stmt = $db->query("SELECT id, name, email, service, weight_kg, message,
submitted_at FROM messages ORDER BY submitted_at DESC");
$messages = $stmt->fetchAll();
```

`$db->query(...)` (tidak memerlukan placeholder, karena SELECT khusus ini tidak berisi nilai yang diberikan pengunjung) menjalankan query segera dan mengembalikan PDOStatement; `fetchAll()` kemudian menarik setiap baris yang cocok ke dalam array PHP biasa berisi associative array — berkat `PDO::ATTR_DEFAULT_FETCH_MODE` yang diatur di Bagian 6.4, setiap baris tiba sebagai `["id" => 1, "name" => "Budi Santoso", ...]`, diindeks dengan nama kolom alih-alih posisi.

> **Escaping tetap berlaku pada jalur keluar**
>
> Membaca DARI basis data dengan aman (Bagian 6.7) tidak menghilangkan kebutuhan untuk meng-escape data yang masuk KE response HTML — `view-messages.php` tetap melewatkan setiap nilai melalui `htmlspecialchars()` sebelum mencetaknya, persis seperti yang disyaratkan Chapter 5 untuk input formulir, karena pesan yang tersimpan itu sendiri bisa berisi teks yang terlihat seperti HTML yang mungkin dilihat pengunjung berikutnya.

6.8 WebPro dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Seperti pada setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, sebuah platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis, dan dengan Profitina Laundry, studi kasus yang dipakai buku ini — mendeskripsikan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, bukan logika bisnis atau kode sumber lengkap Profitina yang bersifat rahasia.

app.profitina.com menyimpan setiap analisis yang dijalankannya di basis data PostgreSQL sungguhan, bukan MySQL — tetapi disiplin yang mendasarinya yang diajarkan bab ini identik terlepas dari basis data relasional mana yang berada di belakang aplikasi: setiap nilai yang mencapai query dari luar aplikasi melewati query yang diparameterisasi/dipersiapkan, tidak pernah penggabungan string, dan pustaka client PostgreSQL sendiri memainkan persis peran yang dimainkan PDO Bagian 6.4 di sini. Sistem produksi yang menyimpan bertahun-tahun analisis pelanggan melalui SQL yang digabungkan string hanya berjarak satu input yang direkayasa dari kehilangan data yang bencana — persis mengapa disiplin ini diperlakukan sebagai tidak dapat ditawar, bukan sekadar preferensi gaya, di backend Profitina sendiri.

6.9 Ringkasan Bab

- Array PHP hanya hidup selama durasi satu request; basis data mempertahankan data melampaui setiap request, restart, dan reboot.
- Setiap kolom tabel memiliki TIPE (VARCHAR, DECIMAL, TEXT, DATETIME, …) yang dipilih untuk sesuai dengan jenis nilai yang disimpannya — dan PRIMARY KEY secara unik mengidentifikasi setiap baris.
- PDO menghubungkan PHP ke MySQL (atau basis data lain) melalui string DSN; `ERRMODE_EXCEPTION` membuat kegagalan nyaring, bukan diam-diam.
- Prepared statement mengirim BENTUK query dan NILAI-nya dalam dua langkah terpisah — inilah yang mencegah SQL injection, bukan sekadar kebersihan kode yang baik.
- Pengujian nyata dan langsung dari kode bab ini mengonfirmasi bahwa string bergaya SQL-injection yang dikirim melalui formulir sungguhan disimpan sebagai teks yang tidak berbahaya, dan tidak menghapus atau mengubah tabel.
- `fetchAll()` mengembalikan baris sebagai associative array; nilai yang dibaca DARI basis data tetap membutuhkan `htmlspecialchars()` sebelum di-echo kembali KE HTML.

6.10 Pertanyaan Tinjauan

Pertanyaan-pertanyaan ini untuk belajar mandiri dan diskusi kelas; tidak dinilai.

1. Jelaskan, dengan kata-kata Anda sendiri, mengapa `$pricePerKg` di Chapter 5 tidak membutuhkan basis data, tetapi `messages` di bab ini membutuhkannya.
2. `weight_kg` dideklarasikan `DECIMAL(5,1) NULL`, bukan `FLOAT`. Cari tahu mengapa DECIMAL umumnya lebih disukai daripada FLOAT untuk nilai seperti pengukuran atau uang, dan ringkas alasannya dalam satu atau dua kalimat.

3. Telusuri, langkah demi langkah, apa yang sungguh diterima MySQL saat `$stmt->execute([$name, …])` berjalan dengan `$name` sama dengan `Robert'); DROP TABLE messages;--` — dan jelaskan mengapa ini berbeda dari yang akan diterima MySQL di bawah versi penggabungan-string tidak aman Bagian 6.6.
4. `SELECT` milik `view-messages.php` menggunakan `$db->query(…)` alih-alih `$db->prepare(…)`. Jelaskan mengapa prepared statement tidak diperlukan di sini, dan identifikasi apa yang harus berubah tentang query persis ini agar prepared statement menjadi perlu.
5. Andaikan `submitted_at` tidak memiliki `DEFAULT CURRENT_TIMESTAMP`. Apa yang perlu diubah dari pernyataan INSERT `process-contact.php` agar tetap mencatat kapan setiap pesan tiba?

Lampiran 6.A — Log Validasi Kode Sumber Bab 6

Setiap berkas PHP diperiksa dengan `php -l` sebelum kutipan apa pun diambil darinya. `schema.sql` dimuat ke server MariaDB 10.11 sungguhan yang berjalan, dan `process-contact.php` / `view-messages.php` kemudian diuji end-to-end terhadapnya — termasuk pengiriman bergaya SQL-injection yang dideskripsikan di Bagian 6.6 — dengan jumlah baris dan isi tabel dikonfirmasi oleh query `mysql` langsung setelahnya, bukan sekadar disimpulkan dari keluaran skrip PHP itu sendiri.

```
db.php, process-contact.php, view-messages.php : php -l -> No syntax errors
detected

Live test against MariaDB 10.11.14 (profitina_laundry.messages):
  Submission 1 (Budi Santoso, dry-cleaning-style wash-fold, 4 kg): stored as row
#1
  Submission 2 (name = "Robert'); DROP TABLE messages;--"): stored as row #2, as
plain text

  mysql> SELECT COUNT(*) FROM messages;  ->  2   (table intact, NOT dropped)
  view-messages.php renders row #2's name as: Robert&#039;); DROP TABLE
messages;--
  (htmlspecialchars-escaped -- displayed as literal text, not executed as HTML
or SQL)
```

Referensi

[1] The PHP Group. "PDO." PHP Manual. https://www.php.net/manual/en/book.pdo.php (diakses Agustus 2026).

[2] The PHP Group. "PDOStatement::execute" dan "Prepared Statements." PHP Manual — sumber model dua-langkah prepare/execute Bagian 6.5. https://www.php.net/manual/en/pdo.prepared-statements.php (diakses Agustus 2026).

[3] OWASP Foundation. "SQL Injection" dan "SQL Injection Prevention Cheat Sheet" — sumber Bagian 6.6. https://owasp.org/www-community/attacks/SQL_Injection (diakses Agustus 2026).

[4] Oracle Corporation / MariaDB Foundation. "Data Types" dan "CREATE TABLE Syntax." MySQL / MariaDB Reference Manual — sumber pilihan tipe kolom Bagian 6.3. https://mariadb.com/kb/en/data-types/ (diakses Agustus 2026).

[5] The PHP Group. "PDOStatement::fetchAll." PHP Manual (diakses Agustus 2026).

BAB 7

PHP MVC & Laravel

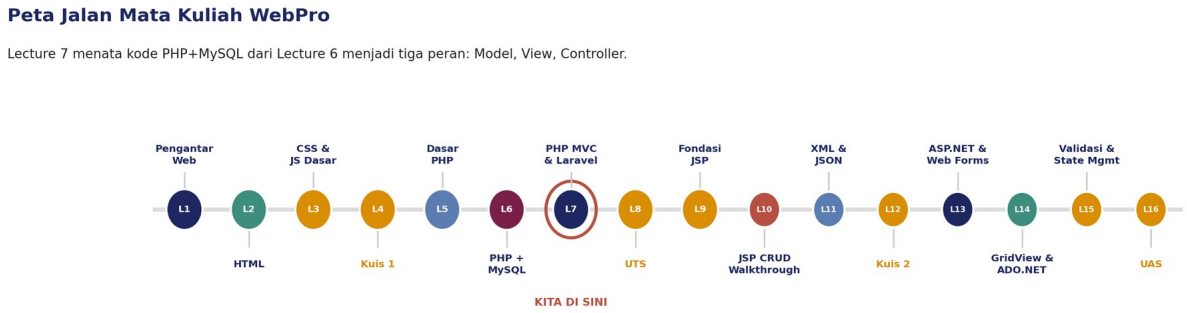
Bab 6 memberi PHP mitra basis data, tetapi setiap baris masih hidup dalam satu file. Bab ini memberi kode yang sama tiga tugas terpisah dan satu nama untuk masing-masing.

Tujuan Pembelajaran — setelah bab ini, Anda akan mampu:

(1) Menjelaskan masalah yang dipecahkan MVC: satu file yang mencampur routing, validasi, SQL, dan HTML. (2) Mendefinisikan Model, View, dan Controller, serta menyatakan mana yang boleh menyentuh SQL dan mana yang boleh mengeluarkan HTML. (3) Menelusuri satu request melalui front controller, Router, Controller, Model, dan kembali melalui View. (4) Membaca dan menjelaskan padanan Laravel untuk setiap bagian tersebut: routes/web.php, Model Eloquent, Controller, dan view Blade. (5) Menjelaskan apa itu migration dan mengapa ia lebih baik dibanding menjalankan schema.sql secara manual. (6) Mengenali bagian mana dari kode Laravel bab ini yang diverifikasi melalui eksekusi nyata, dan mana yang tidak, serta mengapa.

7.1 Rekap: Dari Lecture 6 ke Lecture 7

`process-contact.php` Bab 6 bekerja, dan bekerja dengan aman: ia memvalidasi input, menjalankan INSERT prepared-statement terhadap tabel MySQL sungguhan, bahkan selamat dari upaya SQL-injection nyata tanpa cedera. Namun perhatikan apa yang dilakukan satu file itu: mencocokkan request masuk dengan perilaku, memvalidasi lima field berbeda, menyiapkan dan menjalankan SQL, serta membangun HTML respons — semuanya berurutan, dalam skrip 60-baris yang sama. Itu bukan bug. Itu sekadar rupa kode sebelum siapa pun memisahkan tanggung jawabnya (Gambar 7.1).



Gambar 7.1 — Lecture 7 tidak menambahkan kemampuan baru ke studi kasus mata kuliah ini; ia menata ulang kemampuan yang sudah dibangun Lecture 6.

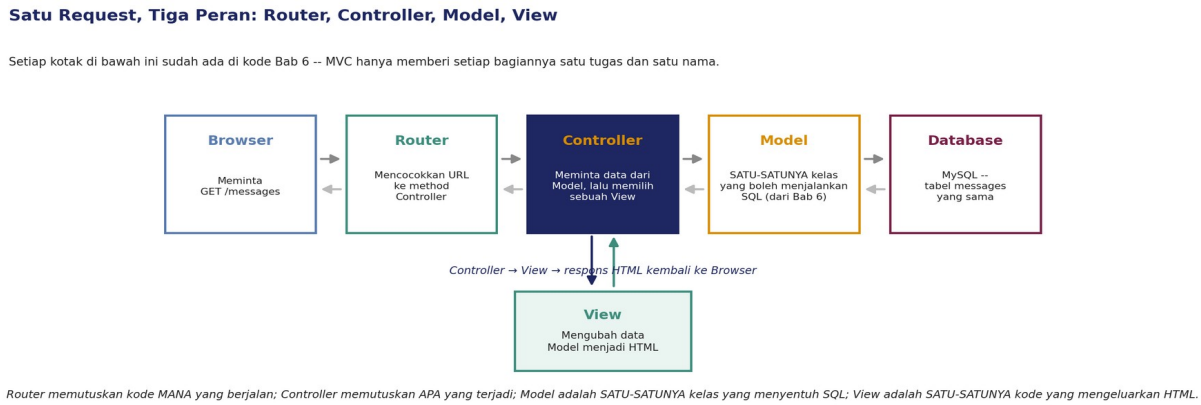
Bab ini menata ulang kode yang sama itu — tabel `messages` yang sama, aturan validasi yang sama, INSERT prepared-statement yang sama — menjadi empat bagian dengan satu tugas masing-masing: Router, Controller, Model, dan View. Tidak ada yang DILAKUKAN aplikasi yang berubah pada bab ini; hanya bagaimana kodenya DIORGANISASI yang berubah.

7.2 Masalahnya: Satu File Melakukan Segalanya

- Routing: memutuskan bahwa request untuk `/messages` harus menjalankan perilaku "tampilkan semua pesan", dan POST ke `/messages` harus menjalankan perilaku "simpan pesan baru" sebagai gantinya.
- Validasi: memeriksa bahwa `name`, `email`, `weight_kg`, dan `message` semuanya memenuhi aturan dari Bab 5 dan 6.
- Akses data: menyiapkan dan menjalankan SQL yang membaca atau menulis tabel `messages`.
- Presentasi: membangun HTML — tabel pesan, formulir, halaman terima kasih — yang benar-benar dikirim kembali ke browser.

Satu file PHP dapat melakukan keempatnya, dan file Bab 6 melakukan itu. Masalahnya muncul kemudian: ubah cara sebuah pesan divalidasi, dan Anda berisiko merusak SQL di bawahnya dalam file yang sama; ubah tampilan respons, dan Anda berisiko merusak validasi di atasnya. Jawaban MVC bukan fungsionalitas baru — melainkan aturan tentang di mana masing-masing dari empat tanggung jawab ini boleh berada.

7.3 Pola MVC: Model, View, Controller



Gambar 7.2 — Setiap kotak dalam diagram ini sudah ada di dalam satu file Bab 6; MVC memberi masing-masing sebuah nama, batas, dan tepat satu tugas.

Peran	Tugas	Aturan
Model	Membaca dan menulis tabel `messages`	SATU-SATUNYA kelas yang boleh menjalankan SQL
View	Mengubah data yang diberikan Controller menjadi HTML	SATU-SATUNYA kode yang boleh mengeluarkan HTML
Controller	Meminta data dari Model, memutuskan View mana yang ditampilkan	Tidak pernah menulis SQL, tidak pernah mengeluarkan HTML langsung

Peran	Tugas	Aturan

Mengapa pemisahan ini sepadan dengan file ekstra

Tidak satu pun dari ini membuat aplikasi melakukan lebih banyak daripada yang sudah dilakukannya di Bab 6. Yang didapat adalah lokalitas perubahan: memperbaiki aturan validasi kini berarti mengedit satu method Controller; mengubah TAMPILAN tabel pesan berarti mengedit satu View; mengubah query SQL berarti mengedit satu Model — setiap perubahan menyentuh satu file, dalam satu peran, bukan satu skrip panjang yang mengerjakan ketiga tugas sekaligus.

7.4 Router: Front Controller

Setiap request dalam aplikasi hasil bangun-ulang bab ini masuk melalui tepat satu file, `public/index.php` — sebuah front controller yang satu-satunya tugasnya adalah menyambungkan Model, Controller, dan Router, lalu menyerahkan kendali ke Router:

```
require_once __DIR__ . "/../config/db.php";
require_once __DIR__ . "/../app/Router.php";
require_once __DIR__ . "/../app/Controllers/MessageController.php";

$controller = new MessageController();

$router = new Router();
$router->get("/messages", [$controller, "index"]);
$router->get("/messages/create", [$controller, "create"]);
$router->post("/messages", [$controller, "store"]);

$path = parse_url($_SERVER["REQUEST_URI"], PHP_URL_PATH);
$method = $_SERVER["REQUEST_METHOD"];

$router->dispatch($method, $path);
```

Router sendiri adalah kelas kecil yang memetakan string `"METHOD PATH"` ke sebuah callable:

```php
class Router
{
    private array $routes = [];

    public function get(string $path, callable $handler): void
    {
        $this->routes["GET {$path}"] = $handler;
    }

    public function post(string $path, callable $handler): void
    {
        $this->routes["POST {$path}"] = $handler;
    }

    public function dispatch(string $method, string $path): void
    {
        $key = "{$method} {$path}";
        if (isset($this->routes[$key])) {
            ($this->routes[$key])();
            return;
        }
        http_response_code(404);
        echo "404 Not Found: {$key}";
    }
```

Router setiap framework melakukan tugas yang persis sama

Apa pun sintaksnya — `Route::get(...)` milik Laravel di `routes/web.php` (Bagian 7.6), `app.get(...)` milik Express di Node.js, dekorator `@app.route(...)` milik Flask di Python — sebuah router selalu ide yang sama: tabel yang mencocokkan pasangan method+path masuk dengan kode yang harus menanganinya. Kelas 25-baris ini bukan versi mainan dari ide tersebut; ia ADALAH ide itu, dengan tabelnya ditulis sebagai array PHP, bukan disembunyikan di dalam framework.

7.5 Model dan Controller: Tabel messages yang Sama, Ditata Ulang

Model adalah satu-satunya kelas yang boleh menyentuh `messages` — query SELECT dan INSERT Bab 6 pindah ke sini tanpa berubah, hanya diganti namanya menjadi dua method statis:

```
class Message
{
    public static function all(): array
    {
        $db = getDb();
        $stmt = $db->query(
            "SELECT id, name, email, service, weight_kg, message, submitted_at "
.
            "FROM messages ORDER BY submitted_at DESC"
        );
        return $stmt->fetchAll();
    }

    public static function create(array $fields): int
    {
        $db = getDb();
        $stmt = $db->prepare(
            "INSERT INTO messages (name, email, service, weight_kg, message) " .
            "VALUES (?, ?, ?, ?, ?)"
        );
        $stmt->execute([
            $fields["name"],
            $fields["email"],
            $fields["service"],
            $fields["weight_kg"],
            $fields["message"],
        ]);
        return (int) $db->lastInsertId();
    }
```

Method `store()` milik Controller — yang paling sibuk dari ketiganya — masih menjalankan setiap aturan validasi dari Bab 5 dan 6, tetapi kini berakhir dengan memanggil `Message::create(...)` alih-alih menyiapkan SQL sendiri:

```
    public function store(): void
    {
        $name = fieldValue($_POST, "name");
        $email = fieldValue($_POST, "email");
        $service = fieldValue($_POST, "service");
        $weight = fieldValue($_POST, "weight_kg");
        $message = fieldValue($_POST, "message");

        $errors = [];
        if (!isValidName($name)) {
            $errors[] = "Please enter a name (1-80 characters).";
        }
        if (!isValidEmail($email)) {
            $errors[] = "Please enter a valid email address.";
        }
        if (!isValidWeight($weight)) {
            $errors[] = "Weight must be a positive number, or left blank.";
        }
        if (!isValidMessage($message)) {
            $errors[] = "Please enter a message (1-2000 characters).";
        }

        if (!empty($errors)) {
            $old = compact("name", "email", "service", "weight", "message");
            $this->render("messages/create", ["errors" => $errors, "old" =>
$old]);
            return;
        }

        $insertedId = Message::create([
            "name" => $name,
            "email" => $email,
            "service" => $service,
            "weight_kg" => $weight !== "" ? $weight : null,
            "message" => $message,
        ]);

        $this->render("messages/thanks", ["name" => $name, "id" =>
$insertedId]);
    }
```

Bandingkan ini dengan `process-contact.php` Bab 6: logika validasi tidak tersentuh, baris demi baris. Yang berpindah adalah SQL — keluar sepenuhnya dari method ini dan masuk ke `Message::create()` — dan HTML — keluar sepenuhnya dari method ini dan masuk ke template View di bawah `app/Views/messages/`, dipilih oleh helper privat `render()` di bagian bawah Controller.

7.6 Mengenal Laravel: Apa yang Diotomatiskan Sebuah Framework

Laravel adalah framework PHP yang menyediakan versi produksi dari setiap bagian yang dibangun manual pada Bagian 7.4 dan 7.5: sebuah router (`routes/web.php`), kelas Controller dasar, ORM bernama Eloquent untuk lapisan Model, dan mesin templating bernama Blade untuk lapisan View. Tidak satu pun dari ide-ide ini baru setelah bab ini — semuanya persis Router, Controller, Model, dan View yang sudah dibangun bab ini, dengan lebih banyak fitur dan jauh lebih sedikit boilerplate.

Router milik Laravel:

```php
<?php

use App\Http\Controllers\MessageController;
use Illuminate\Support\Facades\Route;

Route::get("/messages", [MessageController::class, "index"]);
Route::get("/messages/create", [MessageController::class, "create"]);
Route::post("/messages", [MessageController::class, "store"]);
```

Controller milik Laravel, menggunakan Eloquent dan validate():

```php
<?php

namespace App\Http\Controllers;

use App\Models\Message;
use Illuminate\Http\Request;

class MessageController extends Controller
{
    public function index()
    {
        $messages = Message::orderBy("submitted_at", "desc")->get();
        return view("messages.index", ["messages" => $messages]);
    }

    public function create()
    {
        return view("messages.create");
    }

    public function store(Request $request)
    {
        $validated = $request->validate([
            "name" => "required|string|max:80",
            "email" => "required|email",
            "service" => "required|string|max:20",
            "weight_kg" => "nullable|numeric|min:0.1",
            "message" => "required|string|max:2000",
        ]);

        $message = Message::create($validated);

        return view("messages.thanks", [
            "name" => $message->name,
            "id" => $message->id,
        ]);
    }
}
```

Model Eloquent milik Laravel:

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Message extends Model
{
    protected $fillable = ["name", "email", "service", "weight_kg", "message"];

    public $timestamps = false;
}
```

Migration milik Laravel — alternatif bervariasi-versi untuk `schema.sql` Bab 6:

```php
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create("messages", function (Blueprint $table) {
            $table->id();
            $table->string("name", 80);
            $table->string("email", 120);
            $table->string("service", 20);
            $table->decimal("weight_kg", 5, 1)->nullable();
            $table->text("message");
            $table->dateTime("submitted_at")->useCurrent();
        });
    }

    public function down(): void
    {
        Schema::dropIfExists("messages");
    }
};
```

Model::create() Eloquent adalah Model::create() Bagian 7.5, diotomatiskan

`Message::create($validated)` menebak nama tabel (`messages`) dari nama kelas, membangun INSERT dari array `$fillable`, dan tetap mengikat setiap nilai melalui prepared statement di baliknya — Eloquent tidak menghilangkan disiplin Bab 6 untuk tidak pernah menggabungkan SQL, ia hanya menuliskan prepared statement-nya untuk Anda.

7.7 Catatan Transparansi tentang Kode Laravel Bagian Ini

> **File Laravel bab ini ditulis mengikuti konvensi Laravel nyata, tetapi tidak dieksekusi**
>
> Setiap kode PHP lain di buku ini — Bab 5 dan 6, serta aplikasi mini-MVC Bagian 7.4/7.5 di atas — diperiksa dengan `php -l` lalu benar-benar dijalankan terhadap server PHP nyata dan basis data MariaDB nyata, persis seperti yang didokumentasikan Lampiran 7.A. Menginstal Laravel sendiri memerlukan `composer create-project laravel/laravel`, yang mengunduh paket dari packagist.org; lingkungan sandbox tempat kode buku ini diverifikasi hanya memiliki akses jaringan ke sejumlah kecil host yang diizinkan, yang tidak termasuk Packagist. File Laravel Bagian 7.6 karena itu adalah kode referensi, ditulis semirip mungkin dengan konvensi nyata dan terdokumentasi Laravel 11.x, dan diperiksa dengan `php -l` untuk sintaksis — tetapi tidak pernah dieksekusi terhadap instalasi Laravel yang hidup. Jika Anda menginstal Laravel di komputer Anda sendiri, yang memiliki akses internet biasa, file-file ini ditulis agar dapat langsung dipakai dan berfungsi tanpa perubahan terhadap tabel `messages` yang sama dari Bab 6.

Perbedaan ini penting untuk cara Anda membaca bab ini: perlakukan kode Bagian 7.4 dan 7.5 dengan keyakinan yang sama seperti Bab 5 dan 6 — kode itu benar-benar dijalankan, terhadap basis data nyata, dan Lampiran 7.A menunjukkan output nyatanya. Perlakukan kode Laravel Bagian 7.6 sebagai materi referensi yang akurat untuk apa yang akan Anda temui saat menginstal Laravel sendiri, bukan sebagai klaim bahwa buku ini menjalankan aplikasi Laravel.

7.8 WebPro dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Di seluruh buku ini, kotak seperti ini menghubungkan materi kuliah dengan Profitina, produk AI-visibility (Generative Engine Optimization) Indonesia yang nyata, dibangun oleh penulis, dan dengan Profitina Laundry, bisnis londri sungguhan yang digunakan sebagai studi kasus buku ini. Kotak-kotak ini mendeskripsikan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem yang hidup dan berjalan — tidak pernah detail implementasi, logika bisnis, atau kode sumber Profitina sendiri yang bersifat rahasia.

app.profitina.com sendiri dibangun di atas framework PHP yang menggunakan pemisahan MVC yang sama dengan yang diperkenalkan bab ini — Controller yang tidak pernah menyentuh SQL langsung, Model yang memiliki setiap query, dan template yang tidak pernah menjalankan logika bisnis. Manfaat praktisnya persis seperti Bagian 7.3: perubahan pada cara sebuah hasil analisis divalidasi, berbulan-bulan setelah kode aslinya ditulis, hanya menyentuh satu method Controller, bukan kekusutan SQL, validasi, dan output HTML yang saling terjalin dalam satu file seperti versi satu-file Bab 6.

7.9 Ringkasan Bab dan Poin-Poin Kunci

- MVC tidak menambahkan kemampuan baru ke sebuah aplikasi — ia menetapkan setiap tanggung jawab yang sudah ada (routing, validasi, akses data, presentasi) ke tepat satu peran.
- Model adalah satu-satunya kelas yang boleh menjalankan SQL; View adalah satu-satunya kode yang boleh mengeluarkan HTML; Controller mengoordinasikan antara keduanya dan tidak pernah melakukan salah satu tugas tersebut sendiri.

- Router memetakan pasangan "METHOD PATH" masuk ke method Controller yang harus menanganinya — kelas Router 25-baris bab ini dan `routes/web.php` Laravel melakukan tugas yang identik.
- Laravel mengotomatiskan Router, Controller, Model, dan View tulisan-tangan Bagian 7.4/7.5 dengan router sungguhan, Eloquent (sebuah ORM), dan Blade (mesin templating) — tetapi disiplin prepared-statement dari Bab 6 tetap berjalan di balik query builder Eloquent.
- Migration adalah alternatif bervariasi-versi, berbentuk PHP, untuk menjalankan `schema.sql` secara manual — `php artisan migrate` menjaga skema basis data setiap rekan tim tetap identik.
- Kode PHP mini-MVC bab ini benar-benar dieksekusi dan diverifikasi terhadap basis data MariaDB nyata, persis seperti Bab 5 dan 6; kode referensi Laravel-nya tidak dieksekusi, karena menginstal Laravel memerlukan akses jaringan Packagist yang tidak dimiliki sandbox buku ini — sebuah substitusi yang diungkapkan secara terbuka di Bagian 7.7, bukan disembunyikan.

7.10 Pertanyaan Tinjauan

Pertanyaan-pertanyaan ini untuk belajar mandiri dan diskusi kelas; tidak dinilai.

1. Sebutkan empat tanggung jawab yang diidentifikasi Bagian 7.2 di dalam satu file `process-contact.php` Bab 6, dan namakan peran MVC mana yang menjadi tujuan pindah masing-masing pada aplikasi hasil bangun-ulang bab ini.
2. `Message::all()` dan `Message::create()` pada Model bab ini hampir identik dengan kode yang ada di `view-messages.php` dan `process-contact.php` Bab 6. Apa, secara spesifik, yang berubah di antara kedua versi tersebut, dan apa yang TIDAK berubah?
3. Jelaskan, menggunakan Gambar 7.2, mengapa sebuah method Controller yang memanggil `htmlspecialchars()` langsung pada data sebelum mencetaknya akan menjadi pelanggaran terhadap pemisahan MVC yang dideskripsikan bab ini — meskipun outputnya tetap aman dari XSS.
4. `Message::create($validated)` milik Laravel pada Bagian 7.6 bergantung pada array `$fillable` di `app/Models/Message.php`. Cari tahu apa yang terjadi di Eloquent jika sebuah kunci ada di `$validated` tetapi tidak ada di `$fillable`, dan jelaskan mengapa ini ada sebagai fitur keamanan.
5. Bagian 7.7 mengungkapkan bahwa kode Laravel bab ini tidak dieksekusi. Jelaskan, dengan kata-kata Anda sendiri, mengapa buku ini memilih untuk mengatakan ini secara terbuka alih-alih sekadar menyajikan kode Laravel dengan cara yang sama seperti Bab 5 dan 6, dan mengapa perbedaan itu harus penting bagi Anda sebagai pembaca.

Lampiran 7.A — Log Validasi untuk Kode Sumber Bab 7

`public/index.php`, `app/Router.php`, `app/Controllers/MessageController.php`, `app/Models/Message.php`, dan setiap View di bawah `app/Views/messages/` masing-masing diperiksa dengan `php -l` sebelum kutipan apa pun diambil. Aplikasi mini-MVC lengkap kemudian dijalankan menyeluruh dengan `php -S 127.0.0.1:8093 -t public public/index.php` terhadap basis data MariaDB `profitina_laundry` NYATA yang SAMA yang digunakan di Bab 6, melanjutkan dari 2 baris yang sudah tersimpan di sana.

```
mini-mvc/{config/db.php, app/Router.php, app/Models/Message.php,
  app/Controllers/MessageController.php, app/Views/messages/*.php,
  public/index.php} : php -l -> No syntax errors detected (9 file)

Uji langsung via php -S 127.0.0.1:8093 -t public, terhadap basis data Bab 6:
  GET  /messages/create              -> HTTP 200 (formulir dirender)
  GET  /messages        (sebelum)     -> HTTP 200, 2 baris yang sudah ada
ditampilkan
  POST /messages (Dewi Lestari, wash-fold, 3.5 kg, valid) -> HTTP 200,
       "Thank you, Dewi Lestari! ... reference #3"
  POST /messages (email="not-an-email", tidak valid)     -> HTTP 200,
       "Please enter a valid email address." (formulir ditampilkan lagi, INSERT
tidak dijalankan)
  GET  /messages         (sesudah)    -> HTTP 200, 3 baris ditampilkan (baris #3
ditambahkan)
  GET  /nonexistent                   -> HTTP 404, "404 Not Found: GET
/nonexistent"

laravel-reference/{routes/web.php, app/Models/Message.php,
  app/Http/Controllers/MessageController.php,
  database/migrations/..._create_messages_table.php} : php -l ->
  No syntax errors detected (4 file) -- HANYA SINTAKSIS. File-file ini
  TIDAK dijalankan terhadap instalasi Laravel yang hidup; lihat Bagian 7.7.
```

Referensi

[1] Fowler, Martin. "Patterns of Enterprise Application Architecture" — deskripsi pola Model-View-Controller, sumber untuk definisi peran Bagian 7.3.

[2] Laravel LLC. "Laravel 11.x Documentation — Routing." https://laravel.com/docs/11.x/routing (diakses Agustus 2026) — sumber untuk konvensi routes/web.php Bagian 7.6.

[3] Laravel LLC. "Laravel 11.x Documentation — Eloquent: Getting Started." https://laravel.com/docs/11.x/eloquent (diakses Agustus 2026) — sumber untuk konvensi Model Bagian 7.6.

[4] Laravel LLC. "Laravel 11.x Documentation — Blade Templates." https://laravel.com/docs/11.x/blade (diakses Agustus 2026) — sumber untuk konvensi view Blade yang dirujuk di Bagian 7.6 dan file code/Chapter07/laravel-reference.

[5] Laravel LLC. "Laravel 11.x Documentation — Database: Migrations." https://laravel.com/docs/11.x/migrations (diakses Agustus 2026) — sumber untuk file migration Bagian 7.6.

BAB 8 · BAB LATIHAN

UTS: Proyek Take-Home PHP + MySQL + MVC yang Kecil dan Nyata

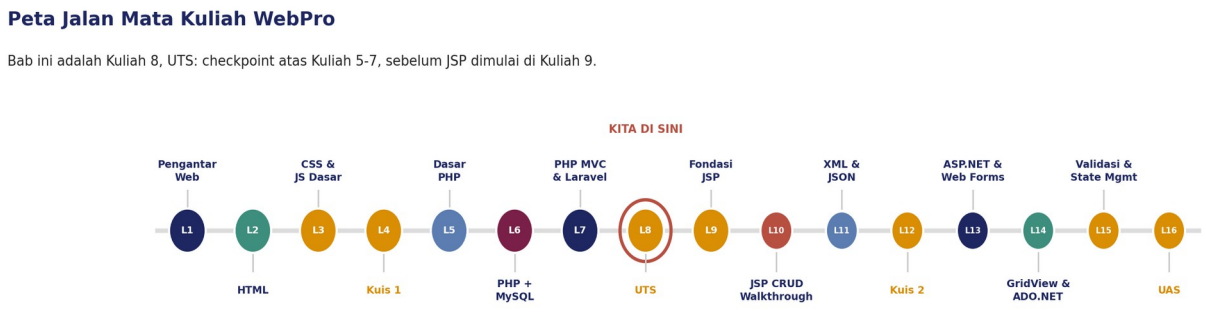
Tidak ada materi baru di sini — sebuah proyek take-home kecil dan terintegrasi yang menyatukan semua yang diajarkan Kuliah 5 hingga 7, persis seperti yang dilakukan UTS sesungguhnya.

Tujuan Pembelajaran — setelah mempelajari bab ini, Anda akan mampu:

(1) Menangani submission formulir HTML nyata di sisi server dengan PHP, memvalidasi setiap field dengan fungsi bertipe dan meng-escape setiap output dengan htmlspecialchars(). (2) Merancang dan membuat tabel MySQL nyata, serta menghubunginya dengan PDO yang dikonfigurasi untuk gagal secara nyaring. (3) Menyisipkan dan membaca kembali data secara aman hanya dengan prepared statement — bukan SQL yang digabungkan sebagai string. (4) Mengorganisasikan seluruh fitur dengan pola MVC: sebuah Router, Controller, Model, dan template View, masing-masing dengan tepat satu tugas. (5) Menjelaskan, dalam laporan tertulis dan dengan kata Anda sendiri, mengapa desain Anda tahan terhadap SQL injection dan apa manfaat pemisahan MVC. (6) Mengemas proyek sebagai laporan plus repositori GitHub, bentuk deliverable yang sama seperti setiap penilaian WebPro.

8.1 Rekap: Kuliah 5–7 Secara Singkat

Kuliah 5 memindahkan eksekusi kode dari browser ke server: PHP membaca `$_POST`, memvalidasi setiap field dengan fungsi bertipe kecil, dan meng-escape setiap nilai dengan `htmlspecialchars()` sebelum mencapai respons. Kuliah 6 memberi PHP tersebut ingatan yang bertahan melampaui satu request — tabel MySQL nyata, dijangkau melalui PDO, ditulisi secara aman dengan prepared statement, dan dibuktikan (dengan percobaan SQL-injection yang nyata dan hidup) tahan terhadap serangan yang justru tidak tahan pada SQL yang digabungkan sebagai string. Kuliah 7 mengambil kode yang sama yang bekerja itu dan menata ulangnya, tanpa mengubah apa yang dilakukannya, menjadi empat peran dengan masing-masing satu tugas: Router memutuskan method Controller mana yang dijalankan, Controller mengoordinasikan request, Model yang menjadi satu-satunya kelas yang boleh menyentuh SQL, dan View yang menjadi satu-satunya kode yang boleh mengeluarkan HTML (Gambar 8.1).



Gambar 8.1 — Bab ini berada di Kuliah 8 dalam peta jalan enam belas kuliah WebPro: sebuah checkpoint, bukan topik baru, UTS sebelum Kuliah 9 memulai JSP.

8.2 Cara Menggunakan Bab Latihan Ini

Apa bab ini — dan apa yang bukan

Ini adalah Bab Latihan: bab praktik, bukan bab materi baru. Seperti setiap penilaian WebPro, UTS sesungguhnya adalah satu PROYEK take-home kecil — satu fitur PHP yang benar-benar berjalan, dikumpulkan sebagai laporan plus repositori GitHub, bukan sekumpulan pertanyaan singkat bernomor. Bagian 8.3 di bawah ini membahas keempat deliverable proyek tersebut (masing-masing dapat ditelusuri ke kuliah tertentu, lihat Gambar 8.2), masing-masing dengan panduan menuju praktik yang baik, bukan solusi lengkap yang sudah dikerjakan atau kode sumber referensi. Seperti Bab 4, 12, dan 16, bab ini tidak menyertakan subfolder code/ — proyek ini adalah milik Anda untuk merancang dan membangunnya, menggunakan kode Bab 5-7 yang benar-benar dijalankan sebagai referensi tentang BAGAIMANA membangunnya, bukan APA yang harus dikumpulkan.

Asal Setiap Deliverable UTS

UTS adalah satu proyek PHP kecil take-home -- setiap deliverable di Bagian 8.3 dapat ditelusuri kembali ke Kuliah 5-7.

#	Deliverable UTS	Berasal dari
1	Penanganan & Validasi Formulir Sisi-Server	K5 -- $_POST, fungsi validasi bertipe, escaping htmlspecialchars()
2	Tabel MySQL Nyata dengan Prepared Statement	K6 -- CREATE TABLE, PDO, prepare()/execute(), pencegahan SQL injection
3	Organisasi MVC	K7 -- Router, Controller, Model, View, masing-masing dengan tepat satu tugas
4	Laporan Tertulis: Analisis Desain & SQL Injection	K6/K7 -- menjelaskan, dengan kata Anda sendiri, mengapa desainnya aman dan terorganisasi

Gambar 8.2 — Setiap deliverable di Bagian 8.3 dapat ditelusuri kembali ke Kuliah 5, 6, atau 7.

Sebelum Anda mulai

Sketsakan skema tabel fitur Anda dan daftar rute Router-nya SEBELUM menulis kode PHP apa pun — disiplin top-down yang sama seperti yang dicontohkan Bab 6 dan 7. UTS memberi penghargaan pada fitur kecil yang terorganisasi dengan benar dan benar-benar aman dari SQL injection, dibandingkan fitur besar yang bekerja secara kebetulan.

8.3 Mini-Proyek UTS

Rancang dan bangun satu fitur PHP + MySQL kecil yang nyata — baik fitur baru yang ditambahkan ke situs Profitina Laundry dari Bab 4-7 (formulir pemesanan, papan masukan pelanggan, pelacak pesanan sederhana), atau fitur kecil setara untuk situs kecil lain pilihan Anda sendiri — selama fitur tersebut menunjukkan setiap deliverable di bawah ini.

8.3.1 Memilih Fitur Anda

Pilih fitur yang minimal membutuhkan satu formulir yang dikirim pengunjung dan satu halaman yang membaca kembali submission yang tersimpan — bentuk baca/tulis yang sama seperti yang

ditunjukkan `process-contact.php` dan `view-messages.php` Bab 6, dan yang menjadi dasar organisasi kelas Model Bab 7.

> **Petunjuk**
>
> Fitur yang dapat Anda deskripsikan sebagai "pengunjung mengirim X, dan Y menampilkan semua yang pernah dikirim" biasanya berukuran tepat untuk proyek UTS. Jika fitur Anda membutuhkan lebih dari dua atau tiga tabel basis data, kemungkinan besar Anda sudah overscoping.

8.3.2 Deliverable & Laporan yang Diwajibkan

Kumpulkan proyek Anda sebagai repositori GitHub (berkas sumber, termasuk schema.sql Anda) plus laporan tertulis singkat. Laporan tersebut sebaiknya mencakup secara singkat: fitur Anda dan siapa penggunanya, desain tabel Anda dan mengapa setiap kolom memiliki tipe yang dimilikinya, serta penjelasan yang spesifik dan konkret tentang mengapa pernyataan INSERT dan SELECT Anda aman dari SQL injection — bukan definisi umum dari istilah tersebut.

> **Mengapa penjelasan SQL-injection harus spesifik pada kode Anda**
>
> Bab 6 menunjukkan persis seperti apa percobaan SQL-injection nyata dan terverifikasi terhadap prepared statement, baris demi baris. Laporan adalah tempat Anda menunjukkan bahwa Anda memahami MENGAPA pernyataan INSERT dan SELECT Anda sendiri aman — menunjuk pada placeholder `?` Anda sendiri dan pemanggilan `execute([...])` Anda sendiri, bukan mengulang definisi umum sebuah prepared statement.

8.3.3 Daftar Periksa Validasi Sisi-Server

Validasi setiap field formulir di sisi server dengan fungsi bertipe yang mengembalikan boolean — jangan pernah mempercayai `$_POST` secara langsung, dan jangan pernah memvalidasi hanya di JavaScript (validasi sisi-klien adalah kesopanan bagi pengunjung, bukan batas keamanan, karena request apa pun dapat tiba tanpa pernah menjalankan JavaScript Anda sama sekali). Escape setiap nilai dengan `htmlspecialchars()` tepat pada saat dicetak ke HTML, bukan sebelumnya.

> **Petunjuk**
>
> Tanyakan, untuk setiap field: apa secara spesifik yang membuat sebuah nilai TIDAK VALID untuk field ini (kosong, terlalu panjang, format salah), dan apakah fungsi Anda menolak setiap kasus tersebut? `isValidEmail()` Bab 5, yang menggunakan `filter_var(..., FILTER_VALIDATE_EMAIL)`, adalah model betapa sempit dan spesifiknya sebuah fungsi validasi tunggal seharusnya.

8.3.4 Daftar Periksa Basis Data & Prepared Statement

Buat tabel nyata dengan `CREATE TABLE`, memilih tipe kolom secara sengaja untuk setiap field (jangan pernah menjadikan semuanya `VARCHAR(255)` secara default). Hubungkan dengan PDO, `PDO::ATTR_ERRMODE` diatur ke `PDO::ERRMODE_EXCEPTION`. Setiap INSERT dan setiap SELECT yang menyertakan nilai dari pengunjung HARUS menggunakan prepared statement dengan placeholder `?` — penggabungan string ke dalam SQL, di mana pun dalam submission Anda, adalah kegagalan otomatis deliverable ini terlepas dari bagaimana sisa proyek terlihat.

Prepared statement dengan nilai yang digabungkan KE DALAM string SQL bukanlah prepared statement

`$db->prepare("... WHERE id = " . $id)` tetap menggabungkan `$id` ke dalam teks SQL sebelum MySQL pernah melihat sebuah placeholder — ini sama tidak amannya dengan `$db->exec(...)` dengan penggabungan, karena disiplin placeholder hanya melindungi nilai yang diteruskan melalui `execute([...])`, tidak pernah nilai yang sudah dipanggang ke dalam string query itu sendiri.

8.3.5 Daftar Periksa Organisasi MVC

Organisasikan fitur Anda seperti yang dilakukan aplikasi mini-MVC Bab 7: sebuah Router (atau mekanisme dispatch kecil setara) yang memetakan request ke method Controller, Controller yang memvalidasi input dan meminta data ke Model tetapi tidak pernah menjalankan SQL sendiri, Model yang menjadi satu-satunya kelas yang menyentuh basis data, dan template View yang menjadi satu-satunya kode yang menghasilkan HTML.

Petunjuk

Uji-mandiri singkat: cari kata SELECT atau INSERT di berkas Controller Anda. Jika Anda menemukan salah satunya, SQL tersebut seharusnya berada di Model Anda. Cari kata `<table` atau `<div` di berkas Model Anda. Jika Anda menemukan HTML di sana, seharusnya berada di View.

8.4 Format UTS: Proyek Open-Book, Take-Home

UTS adalah proyek open-book, take-home, dikumpulkan sebagai repositori GitHub plus laporan tertulis singkat — bentuk deliverable yang sama seperti Kuis 1, Kuis 2, dan UAS, hanya lebih besar cakupannya. Tidak ada komponen tatap muka berwaktu.

Open-book berarti referensi, bukan kolaborator

Anda boleh berkonsultasi dengan buku teks, manual PHP, MDN, dan catatan Anda sendiri saat membangun proyek Anda. Anda TIDAK BOLEH menyalin repositori mahasiswa lain atau mengumpulkan kode yang bukan benar-benar milik Anda sendiri — proyek take-home open-book menguji apakah Anda dapat MENERAPKAN apa yang telah Anda pelajari, bekerja secara mandiri.

Kriteria	Yang dinilai	Bobot
Desain	Skema tabel dan rencana fitur: apakah fitur memiliki tujuan yang jelas, tipe kolom yang masuk akal, dan rencana Router/Controller/Model/View sebelum kode apa pun ditulis?	20%
Implementasi	Apakah fitur benar-benar berjalan dengan benar: validasi sisi-server pada setiap field, tabel MySQL nyata, prepared statement digunakan secara eksklusif untuk setiap query yang menyentuh data dari pengunjung, dan organisasi MVC yang sesungguhnya?	50%
Analisis & evaluasi	Apakah penjelasan SQL-injection dalam laporan menunjukkan pemahaman nyata atas kode mahasiswa SENDIRI, bukan definisi umum dari buku teks tentang prepared statement?	25%

Kesimpulan	Refleksi singkat dan jujur: apa yang berhasil, apa yang akan dilakukan mahasiswa secara berbeda dengan lebih banyak waktu.	5%

8.5 WebPro dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Seperti di setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan hidup, dibangun oleh penulis, dan dengan Profitina Laundry, studi kasus yang digunakan buku ini — mendeskripsikan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis atau kode sumber lengkap Profitina yang bersifat rahasia.

Setiap fitur entri-data app.profitina.com sendiri mengikuti persis bentuk UTS ini: formulir yang divalidasi, tabel dengan tipe kolom yang dipilih secara sengaja, prepared statement untuk setiap query yang menyentuh nilai yang tidak dihasilkan sendiri oleh aplikasi, dan pemisahan bergaya MVC yang menjaga SQL, logika bisnis, dan output HTML berada di tiga tempat berbeda. Tidak ada satu pun dari itu yang merupakan selera rekayasa kebetulan — disiplin SQL-injection khususnya adalah satu bagian dari materi mata kuliah ini yang Irfan perlakukan sebagai benar-benar tidak dapat ditawar dalam kode apa pun yang ia kirimkan, baik proyek mahasiswa maupun sistem produksi.

8.6 Ringkasan Bab

- Bab ini tidak memperkenalkan materi baru — ini adalah checkpoint yang mencakup Kuliah 5 hingga 7.
- UTS, seperti setiap penilaian WebPro, adalah satu PROYEK take-home kecil (repo GitHub plus laporan), bukan sekumpulan pertanyaan singkat yang terpisah.
- Keempat deliverable proyek dipetakan ke tiga kuliah yang direview: validasi sisi-server (K5), tabel MySQL nyata dengan prepared statement (K6), dan organisasi MVC (K7).
- Bobot penilaian: Implementasi paling berat (50%), diikuti Analisis & Evaluasi (25%), Desain (20%), dan Kesimpulan (5%) — bentuk rubrik yang sama yang digunakan kembali oleh setiap penilaian WebPro.

Fitur yang tampak berhasil ketika Anda mengujinya sendiri tidak sama dengan fitur yang benar-benar aman dari SQL injection — dan satu-satunya cara seorang penilai dapat membedakan keduanya adalah kode prepared-statement Anda sendiri dan laporan Anda sendiri yang menjelaskannya.

Lampiran 8.A — Daftar Periksa Mandiri (Tidak Dinilai)

Sebelum mengumpulkan repositori dan laporan Anda, jalankan proyek Anda melalui daftar periksa ini. Ini hanya membutuhkan beberapa menit dan menangkap sebagian besar kesalahan yang paling sering dilihat penilai pada submission UTS pertama.

- Apakah setiap field formulir divalidasi di sisi server, dengan fungsi bertipe yang spesifik, sebelum digunakan untuk apa pun?
- Apakah setiap nilai di-escape dengan htmlspecialchars() tepat pada saat dicetak ke HTML, termasuk nilai yang dibaca kembali dari basis data?

- Apakah setiap INSERT dan SELECT yang menyentuh data dari pengunjung menggunakan prepared statement dengan placeholder `?` — dengan nol penggabungan string ke dalam SQL di mana pun dalam basis kode?
- Apakah PDO dikonfigurasi dengan PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION, sehingga query yang gagal, gagal secara nyaring?
- Apakah Controller tidak mengandung SQL sama sekali, dan apakah Model tidak mengandung HTML sama sekali?
- Apakah paragraf SQL-injection dalam laporan menunjuk pada kode SENDIRI mahasiswa yang spesifik, bukan definisi umum yang disalin?
- Apakah seorang penilai yang belum pernah melihat proyek ini dapat menyiapkan dan menjalankannya hanya dari instruksi dalam laporan?

Referensi

[1] Format penilaian bab latihan ini dimodelkan langsung dari UTS nyata mata kuliah ini (EF234301 Web Programming): proyek take-home open-book yang dikumpulkan sebagai repositori GitHub plus laporan tertulis, dinilai berdasarkan Desain (20%), Implementasi (50%), Analisis & Evaluasi (25%), dan Kesimpulan (5%) — bentuk rubrik yang sama yang digunakan setiap penilaian WebPro (Kuis 1, UTS, Kuis 2, UAS). Format ini digunakan kembali apa adanya di sini; hanya brief proyek spesifik di atas (yang merekap konten nyata Kuliah 5-7) dan logistik pengumpulan administratif yang telah disesuaikan untuk buku teks ini.

[2] The PHP Group. "PDO", "Prepared Statements." PHP Manual (diakses Agustus 2026) — sumber untuk persyaratan prepared-statement Bagian 8.3.4.

[3] OWASP Foundation. "SQL Injection Prevention Cheat Sheet." (diakses Agustus 2026) — sumber untuk panduan SQL-injection di Bagian 8.3.4 dan Lampiran 8.A.

BAB 9

Fondasi JSP

Teknologi sisi-server kedua: bagaimana sebuah berkas .jsp menjadi servlet Java yang nyata dan berjalan, dan bagaimana menulisnya tanpa menenggelamkannya dalam scriptlet.

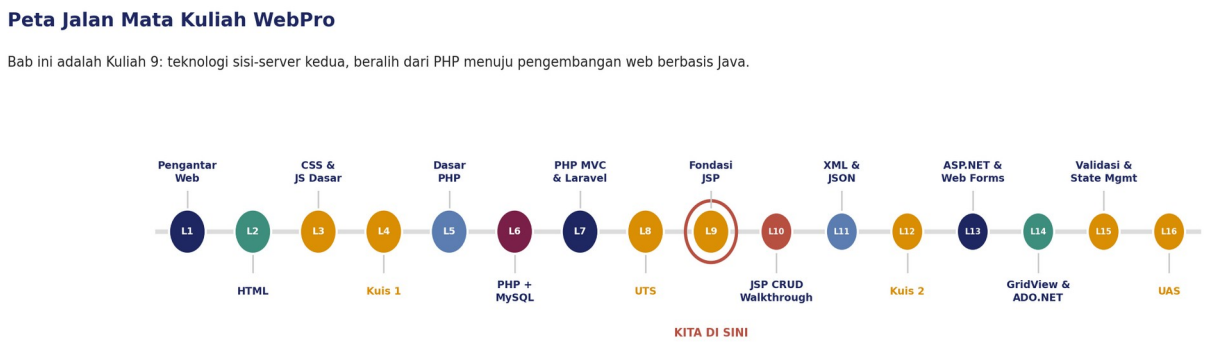
Tujuan Pembelajaran — setelah mempelajari bab ini, Anda akan mampu:

(1) Menjelaskan mengapa mata kuliah ini memperkenalkan teknologi sisi-server kedua setelah tujuh bab PHP, dan menyebutkan kesamaan JSP dan PHP. (2) Mendeskripsikan siklus hidup JSP: bagaimana komponen Jasper milik Tomcat mengompilasi berkas .jsp menjadi berkas sumber servlet Java, mengompilasi berkas tersebut dengan javac, dan menjaga instance servlet yang dihasilkan tetap dimuat lintas request. (3) Menggunakan empat elemen sintaksis inti JSP — directive, declaration, scriptlet, dan expression — dan menjelaskan masing-masing menjadi apa saat dikompilasi. (4) Menyebutkan dan menggunakan objek implisit JSP (request, response, session, application, out) tanpa mengimpor apa pun. (5) Menulis Expression Language (EL) alih-alih scriptlet, dan menjelaskan mengapa mata kuliah ini memperlakukan EL sebagai default dan scriptlet sebagai warisan lama. (6) Mengenali penamaan paket jakarta.servlet.* yang digunakan versi Tomcat terkini, dan mengetahui mengapa tutorial lama menampilkan javax.servlet.* sebagai gantinya.

9.1 Dari PHP ke Java: Mengapa Teknologi Sisi-Server Kedua

Bab 5 hingga 8 membangun satu keterampilan sisi-server yang lengkap dalam PHP: membaca formulir yang dikirim, memvalidasinya, menyimpannya dengan aman di MySQL, dan mengorganisasikan hasilnya menjadi MVC. JSP (Jakarta Server Pages) adalah cara kedua untuk melakukan pekerjaan sisi-server — bukan pengganti, dan bukan bukti bahwa PHP salah. Universitas maupun perusahaan sama-sama menjalankan keduanya: PHP karena cepat di-deploy dan banyak di-hosting, Java/JSP karena organisasi besar sering menstandardisasi pada Java Virtual Machine (JVM) untuk tooling matangnya, penerapan tipe yang ketat, dan kompatibilitas mundur jangka panjang. Mempelajari keduanya berarti Anda memiliki satu siklus request/response yang dapat diimplementasikan di kedua stack — dasar-dasar HTTP dari Kuliah 1 tidak berubah; hanya bahasa dan runtime yang dieksekusi di server yang berubah.

Perbedaan struktural terpenting yang harus segera diperhatikan: PHP diinterpretasi ulang pada setiap request oleh mesin PHP yang tertanam di web server. JSP dikompilasi satu kali menjadi kelas Java pada request pertama kalinya diminta, dan kelas yang telah dikompilasi tersebut kemudian menangani setiap request berikutnya tanpa dikompilasi ulang. Kode bab ini sendiri membuktikan perbedaan tersebut, bukan sekadar menyatakannya — Bagian 9.6 menunjukkan sebuah penghitung yang bertahan melintasi dua HTTP request nyata justru karena JSP dikompilasi satu kali, bukan diinterpretasi ulang (Gambar 9.1).



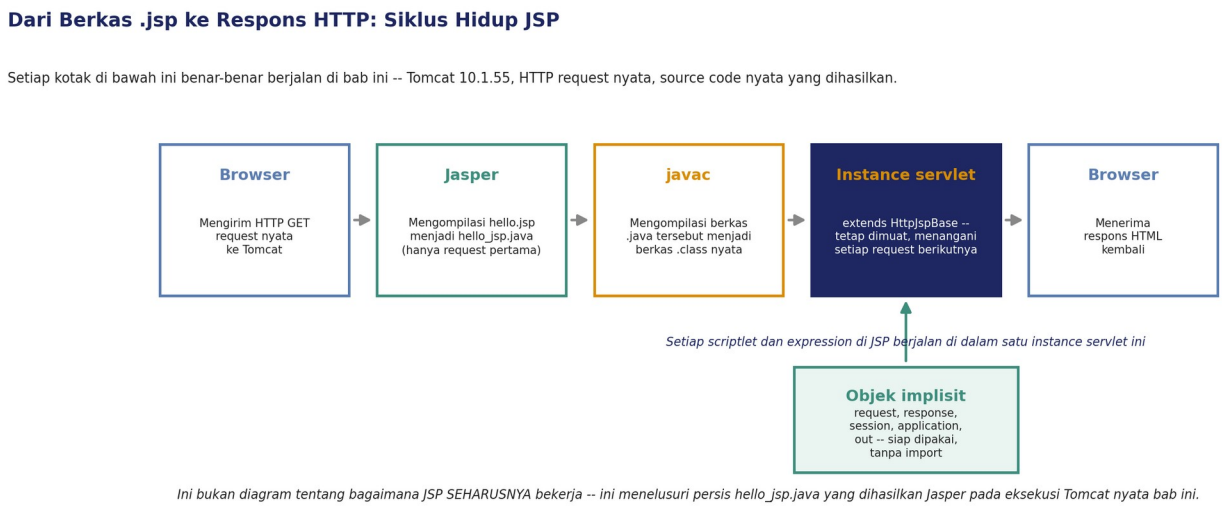
Gambar 9.1 — Kuliah 9 membuka jalur teknologi sisi-server kedua mata kuliah ini, berlanjut hingga Kuliah 10 sebelum XML & JSON di Kuliah 11.

9.2 Apa Itu JSP? Dari Berkas .jsp Menjadi Servlet yang Berjalan

Sebuah halaman JSP adalah dokumen HTML dengan kode Java yang disematkan di dalamnya, disimpan dengan ekstensi .jsp dan di-deploy di dalam sebuah servlet container — Apache Tomcat pada mata kuliah ini. Saat pertama kali browser meminta sebuah berkas .jsp, mesin JSP milik Tomcat (komponen bernama Jasper) menerjemahkan seluruh berkas tersebut menjadi kode sumber Java dari sebuah kelas servlet, mengompilasi sumber tersebut dengan javac menjadi berkas .class yang nyata, memuatnya, dan membiarkannya menangani request tersebut. Setiap request berikutnya untuk halaman yang sama menggunakan kembali kelas yang sudah dikompilasi dan dimuat tersebut — Jasper hanya menerjemahkan ulang berkas tersebut jika sumbernya berubah sejak kompilasi terakhir.

> **Servlet, dalam satu kalimat**
>
> Sebuah servlet adalah kelas Java yang mengimplementasikan sebuah interface tertentu yang memungkinkan servlet container seperti Tomcat menyerahkan request HTTP yang masuk kepadanya dan mengirim kembali apa pun yang ditulisnya ke response — JSP hanyalah cara yang lebih nyaman dan berbentuk-HTML untuk menulis sebuah servlet.

Ini bukan klaim abstrak dalam bab ini — ini diverifikasi dengan menginstal servlet container Apache Tomcat 10.1.55 yang nyata, men-deploy berkas .jsp nyata ke dalamnya, dan memeriksa source code Java sesungguhnya yang dihasilkan Jasper. Gambar 9.2 menelusuri pipeline tersebut secara persis, dan Bagian 9.6 menunjukkan source code yang benar-benar dihasilkan, bukan ilustrasi tulisan-tangan tentang seperti apa kira-kira bentuknya.



Gambar 9.2 — Siklus hidup JSP nyata yang ditelusuri dari eksekusi Tomcat bab ini sendiri: request, kompilasi (hanya sekali), dan instance servlet yang berumur panjang.

javax.servlet vs. jakarta.servlet — keduanya ada, dan perbedaannya penting

Tutorial dan buku ajar yang ditulis sebelum 2021 menampilkan `import javax.servlet.*;`. Pada 2021, spesifikasi servlet berpindah dari namespace javax.* milik Oracle ke namespace jakarta.* milik Eclipse Foundation sebagai bagian dari transisi Jakarta EE. Tomcat 10 ke atas menggunakan jakarta.servlet.* secara eksklusif — kedua namespace TIDAK dapat saling dipertukarkan, dan kode berbasis javax tidak akan berkompilasi terhadap instalasi Tomcat 10+ tanpa alat migrasi. Mata kuliah ini menggunakan Tomcat 10.1.55, sehingga setiap contoh di sini menggunakan jakarta.servlet.*.

9.3 Sintaksis JSP: Directive, Declaration, Scriptlet, Expression

Sebuah berkas .jsp mencampurkan HTML biasa dengan empat jenis tag terkait-Java, masing-masing dikompilasi menjadi bagian berbeda dari servlet yang dihasilkan. Cuplikan kode bagian ini diambil langsung dari hello.jsp, sebuah halaman yang benar-benar di-deploy ke instance Tomcat bab ini.

9.3.1 Directive dan Declaration

Sebuah directive, ditulis `<%@ ... %>`, memberi Jasper instruksi tingkat-halaman — misalnya, tipe konten dan encoding karakter response. Sebuah declaration, ditulis `<%! ... %>`, menjadi field atau method dari kelas servlet yang dihasilkan itu sendiri, sehingga state-nya bertahan lintas setiap request yang ditangani instance servlet tersebut selama instance itu tetap dimuat.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%!
    // Declaration: a member of the generated servlet class, shared across all
requests.
    private int pageLoadCount = 0;

    private String describeTimeOfDay(int hour) {
        if (hour < 12) return "morning";
        if (hour < 18) return "afternoon";
        return "evening";
    }
%>
```

Mengapa pageLoadCount membuktikan klaim siklus hidup

pageLoadCount dideklarasikan dengan `<%! %>`, menjadikannya field dari kelas servlet yang dikompilasi, bukan variabel lokal. Output curl nyata pada Bagian 9.6 menunjukkannya bertambah dari 1 menjadi 2 lintas dua HTTP request terpisah — bukti bahwa instance servlet yang sama menangani keduanya, persis seperti yang dideskripsikan Gambar 9.2.

9.3.2 Scriptlet dan Expression

Sebuah scriptlet, ditulis `<% ... %>`, adalah Java biasa yang berjalan sekali per request, berurutan, tepat di tempat ia muncul dalam berkas. Sebuah expression, ditulis `<%= ... %>`, mengevaluasi sebuah expression Java dan mencetak hasilnya langsung ke output HTML — ini adalah singkatan dari `out.print(...)`.

```
<%
    // Scriptlet: plain Java, runs once per request inside the generated
_jspService() method.
    pageLoadCount++;
    java.util.Calendar now = java.util.Calendar.getInstance();
    int hour = now.get(java.util.Calendar.HOUR_OF_DAY);
    String timeOfDay = describeTimeOfDay(hour);
    String visitorName = request.getParameter("name");
    if (visitorName == null || visitorName.trim().isEmpty()) {
        visitorName = "Guest";
    }
%>
```

Perhatikan `request.getParameter("name")` — membaca parameter query-string atau formulir di JSP menggunakan mekanisme HTTP dasar yang persis sama seperti `$_GET`/`$_POST` milik PHP, hanya melalui pemanggilan method Java pada objek implisit `request` yang dijelaskan berikutnya.

9.4 Objek Implisit

JSP secara otomatis menyediakan beberapa objek di dalam setiap scriptlet dan expression, tanpa import dan tanpa instansiasi yang diperlukan. Yang paling sering digunakan adalah request (HTTP request yang masuk), response (HTTP response yang keluar), session (state per-pengunjung yang bertahan lintas request), application (state yang dibagikan oleh seluruh pengunjung aplikasi web), dan out (writer yang menghasilkan body response HTML).

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%
    // Demonstrating JSP's implicit objects -- available in every JSP without
declaration.
    session.setAttribute("lastVisit", new java.util.Date().toString());
    application.setAttribute("appName", "Profitina Laundry");
%>
!DOCTYPE html>
<html>
<head><title>JSP Implicit Objects Demo</title></head>
<body>
    <h1>JSP Implicit Objects</h1>
    <ul>
        <li><b>request</b> method: <%= request.getMethod() %></li>
        <li><b>request</b> URI: <%= request.getRequestURI() %></li>
        <li><b>request</b> remote address: <%= request.getRemoteAddr() %></li>
        <li><b>session</b> ID: <%= session.getId() %></li>
        <li><b>session</b> is new: <%= session.isNew() %></li>
        <li><b>application</b> attribute "appName": <%=
application.getAttribute("appName") %></li>
        <li><b>out</b> is writing this response directly, buffered by the
container</li>
    </ul>
</body>
</html>
```

session vs. application, masing-masing dalam satu kalimat

session menyimpan data untuk SATU pengunjung lintas beberapa request-nya (ID session-nya, diatur sekali, digunakan kembali pada setiap request yang ia buat); application menyimpan data yang dibagikan oleh SETIAP pengunjung aplikasi web yang di-deploy, selama server tersebut terus berjalan.

9.5 Expression Language (EL): Alternatif Modern untuk Scriptlet

Gaya JSP modern menghindari scriptlet hampir sepenuhnya demi Expression Language, ditulis `${...}`. EL membaca atribut request/session/application dan mengevaluasi expression sederhana langsung di dalam HTML, tanpa kurung kurawal Java, titik koma, atau pemanggilan `out.print()` eksplisit. Ini adalah bagian dari spesifikasi JSP itu sendiri — tidak diperlukan library terpisah untuk menggunakannya, berbeda dengan tag library JSTL, yang tidak dapat diinstal di sandbox mata kuliah ini (lihat Catatan Transparansi di Bagian 9.6).

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%
    // Setting up data the EL expressions below will read -- normally a
servlet/controller would do this,
    // not the JSP itself (see the MVC discussion in Lecture 10). Kept here only
to keep this demo self-contained.
    request.setAttribute("serviceName", request.getParameter("service") !=
null ? request.getParameter("service") : "Wash & Fold");
    request.setAttribute("weightKg", 3.5);
    double pricePerKg = 8000; // IDR per kg
    request.setAttribute("pricePerKg", pricePerKg);
%>
!DOCTYPE html>
<html>
<head><title>JSP Expression Language (EL) Demo</title></head>
<body>
    <h1>Expression Language (EL) -- No Scriptlets</h1>
    <p>Service: <strong>${serviceName}</strong></p>
    <p>Weight: ${weightKg} kg</p>
    <p>Price per kg: Rp ${pricePerKg}</p>
    <p>Total: Rp ${weightKg * pricePerKg}</p>
    <hr>
    <p>Request method via EL implicit object: ${pageContext.request.method}</p>
```

Mengapa EL lebih disukai daripada scriptlet

Sebuah scriptlet dapat berisi Java apa pun — termasuk pemanggilan SQL, logika bisnis, apa pun — yang dicampur langsung ke dalam markup HTML, persis masalah satu-file-melakukan-segalanya yang didiagnosis Bab 7 untuk PHP. EL hanya dapat membaca dan menampilkan data yang sudah disiapkan di tempat lain (biasanya oleh sebuah servlet yang bertindak sebagai Controller, ditinjau sekilas di Kuliah 10) — ia secara struktural tidak dapat menyelundupkan logika bisnis ke dalam sebuah View, itulah sebabnya gaya JSP profesional memperlakukannya sebagai default dan scriptlet sebagai warisan lama.

9.6 Catatan Transparansi: Apa yang Benar-Benar Dijalankan di Bab Ini

Setiap halaman JSP yang ditampilkan di bab ini di-deploy ke servlet container Apache Tomcat 10.1.55 yang nyata dan diuji dengan HTTP request nyata melalui curl — tidak ada satu pun output di atas yang merupakan ilustrasi tulisan-tangan tentang seperti apa kira-kira output JSP.

Bagaimana Tomcat berhasil diinstal meski ada keterbatasan jaringan seperti di Bab 7

Maven Central (repo.maven.apache.org, repo1.maven.org) dan mirror unduhan resmi Apache (dlcdn.apache.org, archive.apache.org) semuanya mengembalikan HTTP 403 di sandbox ini — jenis keterbatasan allowlist-jaringan yang sama yang memblokir Packagist/Composer di Bab 7. Namun, mirror paket apt milik Ubuntu sendiri dapat dijangkau, dan Ubuntu mengemas Tomcat 10.1.55 yang lengkap dan asli sebagai `tomcat10`. `apt-get install -y tomcat10 tomcat10-admin` berhasil, menginstal servlet container yang nyata — berbeda dengan Bab 7, tidak ada substitusi ke kode referensi-saja yang diperlukan di mana pun dalam bab ini.

Hasil nyata dan terverifikasi: meminta hello.jsp tanpa parameter mengembalikan "Good evening, Guest!" dengan hitungan request 1; memintanya lagi dengan `?name=Dewi` mengembalikan "Good

evening, Dewi!" dengan hitungan request 2 — membuktikan instance servlet yang dikompilasi sama menangani kedua request. implicit.jsp mengembalikan ID session nyata yang dihasilkan container dan mengonfirmasi `session.isNew()` bernilai true pada kunjungan pertama. el-demo.jsp mengevaluasi `${weightKg * pricePerKg}` menjadi total hasil hitungan nyata, dan mencerminkan parameter query-string nyata kembali melalui `${pageContext.request.queryString}`. Sebuah request untuk halaman yang tidak ada mengembalikan HTTP 404 yang nyata. Satu keterbatasan yang benar-benar dihadapi: paket Apache Standard Tag Library (JSTL) yang tersedia melalui apt (`libtaglibs-standard-*`) mendahului namespace jakarta.* dan tidak diinstal, sehingga bab ini mendemonstrasikan EL secara langsung alih-alih tag bergaya `<c:forEach>` milik JSTL — EL saja, bagian dari spesifikasi JSP dasar, adalah yang benar-benar dijalankan kode bab ini.

Sebagai bukti langsung alih-alih deskripsi, berikut beberapa baris pertama sesungguhnya dari hello_jsp.java, berkas yang dihasilkan Jasper pada instance Tomcat bab ini dari hello.jsp:

```
package org.apache.jsp;

import jakarta.servlet.*;
import jakarta.servlet.http.*;
import jakarta.servlet.jsp.*;

public final class hello_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent,
                 org.apache.jasper.runtime.JspSourceImports,
                 org.apache.jasper.runtime.JspSourceDirectives {

    // Declaration: a member of the generated servlet class, shared across all
requests.
    private int pageLoadCount = 0;
  // ... Jasper-generated fields/methods omitted ...
```

9.7 WebPro dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Seperti di setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan hidup, dibangun oleh penulis, dan dengan Profitina Laundry, studi kasus yang digunakan buku ini — mendeskripsikan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis atau kode sumber lengkap Profitina yang bersifat rahasia.

Stack produksi Profitina sendiri tidak dibangun di atas JSP — tetapi perbedaan yang ditarik bab ini antara objek implisit/EL dan scriptlet dipetakan langsung ke disiplin yang diterapkan app.profitina.com di mana pun: kode presentasi hanya boleh menampilkan data yang sudah disiapkan dan divalidasi di tempat lain, tidak pernah menghitung logika bisnis secara inline. Apa pun bahasa atau sistem templating-nya, aturan arsitektural yang sama berulang — panduan EL-di-atas-scriptlet bab ini adalah satu ekspresi konkret dari aturan yang diterapkan Irfan di setiap stack yang ia kirimkan.

9.8 Ringkasan Bab

- JSP adalah teknologi sisi-server kedua, bukan pengganti PHP — keduanya mengimplementasikan siklus request/response HTTP yang sama, pada runtime yang berbeda (interpreter PHP vs. JVM).
- Sebuah berkas .jsp dikompilasi satu kali, oleh komponen Jasper milik Tomcat, menjadi kelas servlet Java yang nyata yang meng-extend HttpJspBase — request berikutnya menggunakan kembali instance yang sudah dikompilasi tersebut, itulah sebabnya state yang dideklarasikan dengan `<%! %>` bertahan lintas request.
- Empat elemen sintaksis mencakup sebagian besar kode JSP: directive (`<%@ %>`, pengaturan tingkat-halaman), declaration (`<%! %>`, field/method servlet), scriptlet (`<% %>`, Java per-request), dan expression (`<%= %>`, output yang dicetak).
- Objek implisit (request, response, session, application, out) tersedia di setiap JSP tanpa import — request/session/application dipetakan langsung ke masa hidup data per-request, per-pengunjung, dan per-aplikasi.
- Expression Language (`${...}`) adalah cara modern dan lebih disukai untuk menampilkan data yang sudah disiapkan dalam sebuah JSP, khususnya karena — berbeda dengan scriptlet — ia tidak dapat menyelundupkan logika bisnis ke dalam sebuah View.
- Tomcat 10+ menggunakan namespace paket jakarta.servlet.*, bukan namespace javax.servlet.* yang ditampilkan tutorial pra-2021 — source code servlet nyata yang dihasilkan bab ini mengonfirmasi yang mana yang digunakan mata kuliah ini.

Kuliah 10 membangun di atas setiap bagian yang diperkenalkan di sini — objek implisit, EL, dan siklus hidup servlet — untuk menelusuri sebuah fitur JSP CRUD (create, read, update, delete) lengkap yang terhubung ke basis data nyata, mencerminkan apa yang dibangun Bab 6 dan 7 dalam PHP.

Lampiran 9.A — Log Validasi

Halaman JSP bab ini di-deploy ke servlet container Apache Tomcat 10.1.55 yang nyata (diinstal melalui `apt-get install tomcat10 tomcat10-admin`, OpenJDK 21.0.10) dan diuji dengan HTTP request nyata. Transkrip curl lengkap, source code hello_jsp.java yang benar-benar dihasilkan, dan catatan keterbatasan jaringan disimpan dalam folder code/ bab ini sebagai VALIDATION_LOG.txt, bersama tiga berkas .jsp nyata dan deployment descriptor web.xml yang benar-benar digunakan.

Referensi

[1] Oracle/Eclipse Foundation. "Jakarta Server Pages Specification", versi 3.1 (diakses Agustus 2026) — sumber untuk elemen sintaksis JSP yang dibahas di Bagian 9.3.

[2] The Apache Software Foundation. "Apache Tomcat 10 Documentation", komponen Jasper (diakses Agustus 2026) — sumber untuk siklus hidup JSP yang dideskripsikan di Bagian 9.2 dan Gambar 9.2.

[3] Eclipse Foundation. "Jakarta EE: javax to jakarta namespace transition" (diakses Agustus 2026) — sumber untuk perbedaan namespace di callout Trap Bagian 9.2.

BAB 10

JSP CRUD Walkthrough

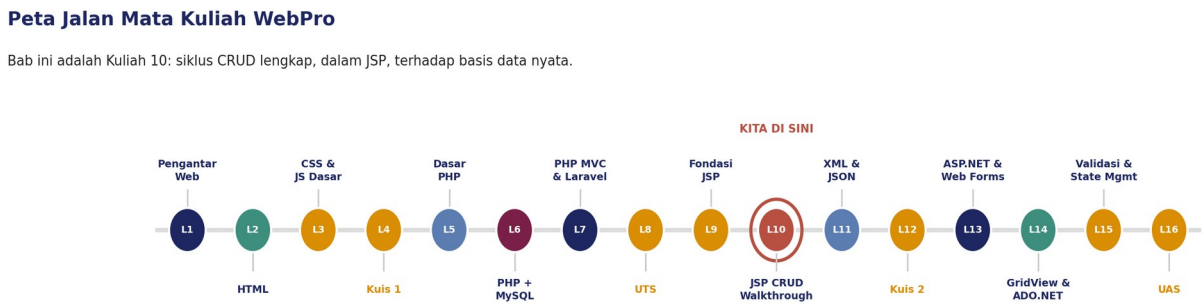
Empat halaman JSP, satu tabel basis data: fitur Create/Read/Update/Delete lengkap, ditelusuri dari awal hingga akhir terhadap basis data MariaDB nyata.

Tujuan Pembelajaran — setelah mempelajari bab ini, Anda akan mampu:

(1) Menghubungkan halaman JSP ke basis data relasional nyata menggunakan JDBC, termasuk objek Connection, Statement/PreparedStatement, dan ResultSet. (2) Menjelaskan mengapa PreparedStatement dengan placeholder `?` yang terikat mencegah SQL injection, sedangkan penggabungan string tidak. (3) Membangun keempat operasi CRUD sebagai halaman JSP terpisah, membedakan GET (menampilkan/konfirmasi) dari POST (operasi yang benar-benar mengubah data). (4) Menerapkan pola Post/Redirect/Get dan menjelaskan masalah yang dipecahkannya (pengiriman ulang form yang tidak disengaja saat refresh). (5) Menggunakan static include JSP (`<%@ include %>`) untuk berbagi kode antar halaman, dan menjelaskan perbedaannya dengan memanggil method bersama di sebuah kelas Java yang di-import. (6) Mengenali mengapa request GET tidak boleh pernah memiliki efek samping, dan mengapa fitur Delete bab ini dibangun sebagai halaman konfirmasi, bukan tautan instan.

10.1 Rekap: Dari Kuliah 9 ke Kuliah 10

Kuliah 9 menetapkan apa itu JSP sebenarnya -- sebuah berkas yang dikompilasi satu kali menjadi servlet, dengan objek implisit dan Expression Language tersedia di setiap halaman. Semua hal di bab tersebut didemonstrasikan dengan halaman yang tidak memiliki state persisten selain sebuah counter dalam memori. Bab ini menghubungkan mekanisme JSP yang sama ke basis data nyata yang berjalan, menuntaskan jenis fitur yang sama yang dibangun Bab 6 dan 7 dalam PHP: sebuah halaman yang membuat, membaca, memperbarui, dan menghapus baris nyata, kali ini dalam Java (Gambar 10.1).



Gambar 10.1 — Kuliah 10 adalah kuliah JSP kedua sekaligus terakhir dalam alur teknologi sisi-server mata kuliah ini, sebelum Kuliah 11 beralih ke XML & JSON.

10.2 Mengapa CRUD Lengkap, dan Mengapa Tabel Baru

Bab 6 dan 7 membangun Create dan Read terhadap tabel messages -- sebuah formulir kontak yang menyisipkan satu baris, dan halaman yang menampilkan daftar pesan yang dikirim. Tidak ada bab yang membangun Update atau Delete. Bab ini dengan sengaja memperkenalkan tabel baru, orders, khusus agar keempat operasi CRUD memiliki alasan alami untuk ada: sebuah pesanan laundry dibuat saat

pelanggan menitipkan pakaian, dibaca oleh staf yang mengecek statusnya, diperbarui seiring perubahan statusnya (received → washing → ready → picked up), dan sesekali dihapus jika salah dimasukkan.

```
CREATE TABLE IF NOT EXISTS orders (
    id            INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    customer_name VARCHAR(80)  NOT NULL,
    service       VARCHAR(20)  NOT NULL,
    weight_kg     DECIMAL(5,1) NOT NULL,
    status        ENUM('received','washing','ready','picked_up') NOT NULL
DEFAULT 'received',
    created_at    DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

Tabel ini dibuat langsung di instance MariaDB 10.11.14 yang sama yang telah digunakan Bab 6 dan 7, di dalam basis data profitina_laundry yang sama, di bawah pengguna basis data webpro yang sama -- tanpa server baru, tanpa kredensial baru, hanya sebuah tabel baru berdampingan dengan tabel messages yang sudah ada.

10.3 Satu Helper Koneksi Bersama: dbconn.jspf

Setiap satu dari keempat halaman JSP bab ini membutuhkan tiga baris yang sama untuk membuka koneksi basis data. Alih-alih mengulanginya, bab ini memfaktorkannya ke dalam berkas terpisah, dbconn.jspf, disertakan di bagian atas setiap halaman dengan sebuah directive: `<%@ include file="/WEB-INF/dbconn.jspf" %>`.

```
<%@ page import="java.sql.Statement" %>
<%@ page import="java.sql.ResultSet" %>
<%!
    private Connection getConnection() throws SQLException {
        String url  = "jdbc:mariadb://127.0.0.1:3306/profitina_laundry";
        String user = "webpro";
        String pass = "webpro_pass123";
        // MariaDB Connector/J 2.7.6 registers itself via the JDBC 4.0
        // ServiceLoader mechanism -- no Class.forName() needed, unlike
        // pre-2006 JDBC drivers still shown in some tutorials.
```

Static include BUKAN pemanggilan fungsi

`<%@ include %>` berjalan pada waktu translasi, sebelum javac sama sekali melihat berkas tersebut: Jasper secara harfiah menempelkan teks dbconn.jspf ke dalam source servlet yang dihasilkan setiap halaman yang menyertakannya. orders_002dlist_jsp.java hasil generate bab ini sendiri (Lampiran 10.A) mengonfirmasinya -- getConnection() muncul sebagai method private milik KELAS ITU secara spesifik, bukan panggilan ke suatu kelas helper Connection bersama. Setiap dari keempat halaman mendapat salinan independennya sendiri dari method yang sama, masing-masing dikompilasi terpisah.

> **Mengapa dbconn.jspf berada di /WEB-INF/, bukan di root webapp**
>
> Spesifikasi Servlet menetapkan WEB-INF tidak pernah bisa diakses langsung lewat HTTP -- klien yang me-request /profitina/WEB-INF/dbconn.jspf akan mendapat 404, apa pun isi berkasnya. Sebuah helper koneksi yang diletakkan di root webapp justru bisa di-request langsung oleh klien dan akan mengembalikan source mentahnya yang belum diproses (berkas `.jspf` tidak memiliki directive yang memberi tahu Tomcat untuk memperlakukannya sebagai JSP dengan sendirinya) -- kebocoran informasi nyata yang dihindari sepenuhnya oleh penempatan bab ini.

10.4 READ: Menampilkan Semua Pesanan

orders-list.jsp membuka sebuah Connection, menjalankan satu query, dan mencetak satu baris tabel HTML per hasil -- bentuk yang sama seperti contoh objek-implisit Bab 9, hanya saja didukung oleh SELECT nyata, bukan counter dalam memori.

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ page import="java.sql.*" %>
<%@ include file="/WEB-INF/dbconn.jspf" %>
<%
    // READ -- the R in CRUD. One query, no business logic in the markup
    // below: every value the page prints comes from this ResultSet,
    // read out into request-scope attributes before any HTML runs.
    //
    // A Bill row alone is just three foreign keys and a payment; a real
    // Orders page has to JOIN Customer/Service/Employee to show a human
    // being anything meaningful -- the same JOIN Bills.aspx's GridView
```

Loop di bawah ini mencetak setiap kolom dengan sebuah expression (`<%= %>`), dan menyertakan tautan Edit serta tautan Delete per baris, masing-masing membawa id baris tersebut sebagai parameter query -- dua halaman yang dibangun di sisa bab ini.

```
        + "b.status AS status, b.arrived AS arrived "
        + "FROM Bill b "
        + "JOIN Customer c ON b.Customer_id = c.id "
        + "JOIN Service s ON b.Service_id = s.id "
        + "JOIN Employee e ON b.Employee_id = e.id "
        + "ORDER BY b.id");
%>
!DOCTYPE html>
<html>
<head><title>Profitina Laundry -- Orders</title></head>
<body>
<h1>Orders</h1>
<p><a href="order-create.jsp">+ New Order</a></p>
<table border="1" cellpadding="6">
<tr>
```

10.5 CREATE: Sebuah Form, dan Insert yang Dituju Pengirimannya

order-create.jsp menjalankan dua tugas sekaligus, hal yang umum di JSP: pada GET biasa halaman jatuh ke form HTML; pada POST halaman membaca field-field form tersebut dan menjalankan INSERT.

```
    // a real order can only reference a Customer/Service/Employee row
    // that actually exists -- the same referential reality Bill's three
    // foreign keys enforce at the database level). On POST, insert the
    // submitted row, then redirect back to the list -- a real HTTP
    // redirect, not a forward, so refreshing the resulting page never
    // resubmits the form (Post/Redirect/Get).
    if ("POST".equalsIgnoreCase(request.getMethod())) {
        String customerId = request.getParameter("customer_id");
        String serviceId = request.getParameter("service_id");
        String employeeId = request.getParameter("employee_id");
        String rack = request.getParameter("rack");
        int unit = Integer.parseInt(request.getParameter("unit"));

        Connection conn = getConnection();

        // Payment is computed here, server-side, from the Service
        // table's real price -- never trusted from a hidden form field,
        // exactly the same rule NewBill.aspx.cs follows in the ASP.NET
        // track (companion Profitina Laundry Book, Chapter 6).
        PreparedStatement priceStmt = conn.prepareStatement(
            "SELECT price FROM Service WHERE id = ?");
        priceStmt.setString(1, serviceId);
```

PreparedStatement, bukan penggabungan string

Setiap nilai yang diikat dengan `ps.setString(...)`/`ps.setBigDecimal(...)` dikirim ke MariaDB terpisah dari teks SQL itu sendiri -- nama pelanggan yang mengandung tanda kutip tunggal, atau bahkan kata kunci SQL secara harfiah, disimpan sebagai data inert, tidak pernah dieksekusi. Ini adalah pelajaran yang sama yang dibahas diskusi PDO Bab 6 untuk PHP; PreparedStatement milik JDBC melakukan pekerjaan yang persis sama di sini.

Post/Redirect/Get, dalam satu kalimat

Setelah POST berhasil, `response.sendRedirect("orders-list.jsp")` mengirim browser sebuah HTTP 302 nyata ke url BARU, sehingga me-reload halaman hasilnya me-request ulang daftar dengan GET -- tidak pernah mengirim ulang form dengan INSERT kedua.

10.6 UPDATE: Memuat Satu Baris, Lalu Menuliskannya Kembali

order-edit.jsp adalah satu-satunya halaman di bab ini dengan tiga cabang, bukan dua: POST menuliskan baris yang diedit (terstruktur identik dengan INSERT milik order-create.jsp, hanya memakai UPDATE), sementara GET harus terlebih dahulu memuat nilai baris tersebut saat ini agar form dapat menampilkannya sudah terisi -- pelanggan yang mengedit pesanan seharusnya melihat apa yang sudah tersimpan, bukan form kosong.

```
            + "finished = " + finishedExpr + ", "
            + "departed = " + departedExpr + " WHERE id = ?");
        ps.setString(1, rack);
        ps.setInt(2, unit);
        ps.setBigDecimal(3, payment);
        ps.setString(4, status);
        ps.setString(5, id);
        ps.executeUpdate();
```

Perhatikan cabang GET ini hanya mempercayai id numerik dari query string -- setiap nilai lain yang ditampilkan di form yang sudah terisi (nama pelanggan, layanan, berat, status) berasal dari SELECT, tidak pernah dari apa pun yang bisa dimanipulasi klien di URL.

10.7 DELETE: Mengapa Halaman Ini Punya Dua Langkah

Sebuah request HTTP GET ditetapkan harus aman -- ia tidak boleh, dengan sendirinya, mengubah state server. Tautan "Delete" milik orders-list.jsp karenanya tidak menghapus apa pun: ia hanya menavigasi ke order-delete.jsp, yang cabang GET-nya SENDIRI membaca nama pesanan untuk pesan konfirmasi. Hanya POST form milik halaman konfirmasi itu sendiri yang benar-benar menjalankan DELETE.

```
    // DELETE -- the D in CRUD, and deliberately the one built as a
    // confirmation page rather than an instant GET-triggered delete.
    // A GET request must never have a side effect (HTTP's own rule);
    // orders-list.jsp's "Delete" link only lands here, on a page whose
    // OWN form POST is what actually deletes the row.
    String id = request.getParameter("id");

    if ("POST".equalsIgnoreCase(request.getMethod())) {
        Connection conn = getConnection();
        PreparedStatement ps = conn.prepareStatement(
            "DELETE FROM Bill WHERE id = ?");
        ps.setString(1, id);
        ps.executeUpdate();
        ps.close();
        conn.close();
        response.sendRedirect("orders-list.jsp");
        return;
```

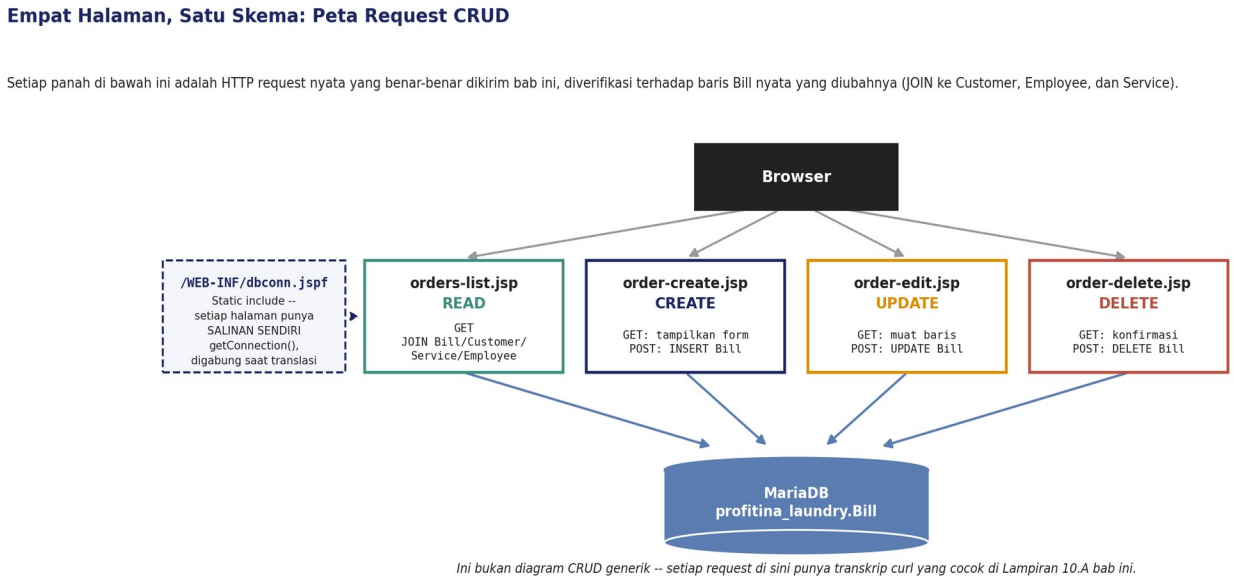
Delete yang dipicu GET adalah kelas kerentanan nyata yang dikenal luas

Sebuah delete yang disambungkan langsung ke tautan GET dapat dipicu oleh apa pun yang menyebabkan browser mengambil sebuah URL -- tag <img src="..."> di halaman yang tidak terkait, pratinjau tautan, sebuah crawler. Membangun Delete sebagai konfirmasi-lalu-POST, seperti yang dilakukan bab ini, bukan kehati-hatian ekstra untuk contoh kelas -- ini adalah pola yang sama yang dipakai di sistem produksi persis karena alasan ini.

10.8 Catatan Transparansi: Apa yang Benar-Benar Dijalankan di Bab Ini

Setiap operasi CRUD yang dijelaskan di atas benar-benar dieksekusi: curl request nyata terhadap instance Tomcat 10.1.55 milik bab ini sendiri, basis data MariaDB 10.11.14 nyata, dan pernyataan SQL

SELECT/INSERT/UPDATE/DELETE nyata -- diverifikasi setelahnya dengan meng-query basis data secara langsung, bukan sekadar mempercayai respons HTTP (Gambar 10.3).



Gambar 10.3 — Empat halaman yang dibangun bab ini, masing-masing mengarah ke tabel orders nyata yang sama, dengan shared include yang salinannya sendiri dikompilasi oleh masing-masing halaman.

> **Driver JDBC mengalami keterbatasan jaringan yang sama seperti Tomcat sendiri -- diselesaikan dengan cara yang sama**
>
> Maven Central dan mirror milik Apache sendiri diblokir di sandbox ini (HTTP 403), sehingga dependency `mvn` biasa untuk driver JDBC MariaDB/MySQL tidak bisa diambil lewat jalur itu. Seperti Tomcat di Bab 9, mirror apt milik Ubuntu sendiri menyediakan alternatif yang asli: `apt-get install -y libmariadb-java` memasang MariaDB Connector/J 2.7.6 -- driver JDBC yang nyata dan lengkap, disalin ke direktori lib/ bersama milik Tomcat. Tidak ada substitusi ke kode referensi-saja yang diperlukan di mana pun dalam bab ini, melanjutkan pola yang ditetapkan Bab 9.

Hasil nyata yang terverifikasi (transkrip lengkap di Lampiran 10.A): sebuah POST ke order-create.jsp untuk pelanggan bernama Siti Rahayu mengembalikan HTTP 302 nyata, dan READ berikutnya terhadap orders-list.jsp menampilkan barisnya dengan id yang ditetapkan basis data secara nyata. Sebuah POST ke order-edit.jsp untuk id yang sama mengubah statusnya menjadi ready -- diverifikasi bukan dari respons HTTP melainkan dari `SELECT ... WHERE id = 3` langsung terhadap basis data setelahnya. Sebuah POST ke order-delete.jsp untuk id tersebut menghapus barisnya -- diverifikasi oleh SELECT langsung yang sama yang mengembalikan nol baris. Membaca orders_002dlist_jsp.java hasil generate Jasper yang nyata juga menyingkap detail kecil namun nyata: Jasper mengubah tanda hubung dalam nama berkas JSP menjadi escape `_002d` untuk membentuk nama kelas Java yang sah, itulah mengapa kelasnya bernama orders_002dlist_jsp, bukan orders-list_jsp.

10.9 WebPro dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Seperti di setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, sebuah platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis, dan dengan Profitina Laundry, studi kasus yang berjalan yang digunakan buku ini -- mendeskripsikan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis atau source code lengkap milik Profitina yang bersifat rahasia.

Stack produksi Profitina sendiri tidak berjalan di atas JSP -- namun disiplin Post/Redirect/Get dan aturan GET-tidak-boleh-mengubah-state yang diajarkan bab ini diterapkan secara identik di app.profitina.com, apa pun bahasa atau framework-nya. Apa pun stack-nya, sebuah request yang hanya membaca state tidak pernah diizinkan untuk juga menuliskannya, dan setiap request yang mengubah state selalu menjawab dengan redirect, bukan me-render halaman secara langsung -- dua aturan yang sama yang diterapkan masing-masing dari keempat berkas JSP kecil bab ini.

10.10 Ringkasan Bab

- Sebuah fitur CRUD lengkap dibangun sebagai empat halaman JSP terpisah terhadap tabel orders nyata di basis data MariaDB yang sama yang telah digunakan Bab 6 dan 7.
- Satu berkas bersama, dbconn.jspf, ditarik ke setiap halaman dengan static include (`<%@ include %>`) -- penggabungan teks pada waktu kompilasi, bukan panggilan method runtime ke sebuah kelas bersama, dikonfirmasi dengan membaca source servlet hasil generate masing-masing halaman.
- PreparedStatement dengan placeholder `?` mengikat setiap nilai dari pengguna sebagai data, tidak pernah sebagai teks SQL -- pertahanan SQL-injection yang sama yang diajarkan Bab 6 untuk PDO, diterapkan di sini melalui JDBC.
- Post/Redirect/Get -- sebuah POST yang mengubah data berakhir dengan `response.sendRedirect(...)`, sehingga me-reload halaman hasil tidak pernah mengirim ulang form.
- Sebuah request GET tidak boleh pernah memiliki efek samping -- fitur Delete bab ini dengan sengaja adalah alur konfirmasi-lalu-POST, bukan delete instan yang dipicu GET.
- Setiap operasi di bab ini diverifikasi dua kali: sekali dari respons HTTP, dan sekali lagi dengan meng-query basis data nyata secara langsung setelahnya.

Kuliah 11 meninggalkan alur teknologi sisi-server untuk topik yang berfokus pada format data: XML dan JSON, dua format yang paling umum diserahkan sebuah backend JSP atau PHP ke frontend JavaScript atau layanan lain.

Lampiran 10.A — Log Validasi

Keempat halaman JSP bab ini beserta dbconn.jspf bersama mereka di-deploy ke servlet container Apache Tomcat 10.1.55 yang sama yang dipasang Bab 9, terhubung lewat MariaDB Connector/J 2.7.6 (dipasang dengan `apt-get install libmariadb-java`) ke instance MariaDB 10.11.14 yang sama yang digunakan Bab 6 dan 7. Transkrip curl lengkap untuk setiap operasi Create, Read, Update, dan Delete,

kutipan orders_002dlist_jsp.java hasil generate yang nyata, dan catatan jaringan apt-vs-Maven-Central tersimpan di folder code/ bab ini sebagai VALIDATION_LOG.txt, berdampingan dengan keempat berkas .jsp, dbconn.jspf, dan schema.sql.

Referensi

[1] Oracle/Eclipse Foundation. "Jakarta Server Pages Specification", versi 3.1 -- static include, Bagian 10.3 (diakses Agustus 2026).

[2] MariaDB Foundation. "MariaDB Connector/J Documentation", versi 2.7 -- driver JDBC yang digunakan sepanjang bab ini (diakses Agustus 2026).

[3] The Apache Software Foundation. "Apache Tomcat 10 Documentation" -- servlet container yang sama yang dipasang di Bab 9 (diakses Agustus 2026).

BAB 11

XML & JSON

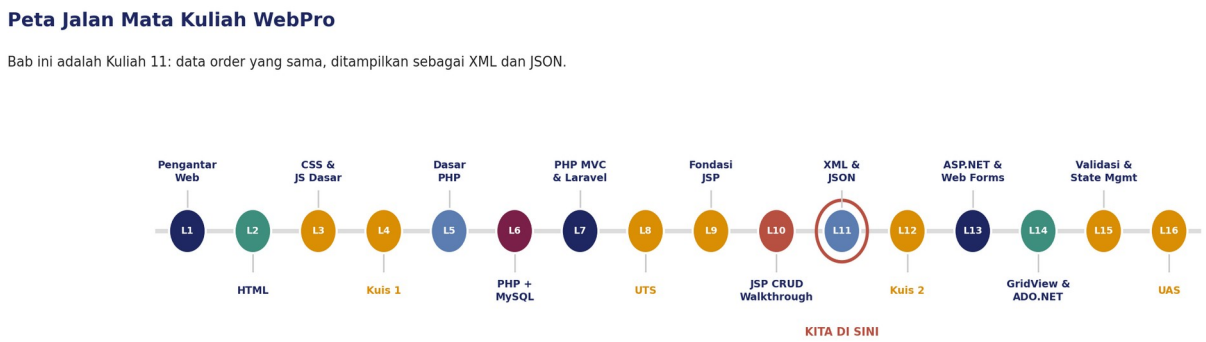
Data order yang sama, dikembalikan sebagai XML dan sebagai JSON dari dua halaman JSP nyata, dan dikonsumsi di sisi klien dengan fetch browser yang nyata.

Tujuan Pembelajaran — setelah menyelesaikan bab ini, Anda akan mampu:

(1) Menjelaskan model elemen/atribut XML dan aturan well-formedness yang harus dipenuhi sebuah dokumen agar dapat di-parse sama sekali. (2) Menjelaskan model objek/array/nilai JSON dan menjelaskan mengapa JSON tidak memiliki konsep "atribut" terpisah. (3) Membangun halaman JSP yang men-serialize baris basis data nyata menjadi XML menggunakan API standar javax.xml.parsers/javax.xml.transform (JAXP) yang sudah tersedia bawaan JDK. (4) Membangun halaman JSP yang men-serialize baris yang sama menjadi JSON menggunakan pustaka pihak ketiga nyata (org.json), dan mengatur Content-Type yang tepat untuk masing-masing format. (5) Menjelaskan mengapa parser XML yang ketat dapat menolak dokumen yang akan diterima begitu saja oleh parser JSON, menggunakan cacat nyata yang ditemukan pengujian bab ini sendiri. (6) Mengonsumsi endpoint JSON dari JavaScript sisi klien dengan fetch() dan merendernya tanpa memuat ulang halaman.

11.1 Rekap: Dari Walkthrough CRUD ke Format Data

Kuliah 10 membangun empat halaman JSP yang membaca dan menulis tabel Bill nyata — di-JOIN ke Customer, Employee, dan Service untuk setiap baris yang dapat dibaca — tetapi setiap satu darinya menghasilkan HTML — output yang dimaksudkan agar dirender langsung oleh browser. Banyak aplikasi nyata juga perlu menyerahkan data yang sama kepada sesuatu yang BUKAN browser yang merender HTML: frontend JavaScript yang mengambil data di latar belakang, aplikasi mobile, atau layanan lain sepenuhnya. XML dan JSON adalah dua format yang paling umum diharapkan konsumen semacam itu, dan bab ini membangun satu endpoint nyata untuk masing-masing jenis terhadap data Bill/Customer/Employee/Service ter-JOIN yang persis sama yang sudah diisi Kuliah 10 (Gambar 11.1).



Gambar 11.1 — Kuliah 11 adalah selingan singkat dan terfokus ke format data di antara dua rangkaian teknologi sisi-server (JSP, yang baru saja selesai, dan ASP.NET, dimulai di Kuliah 13).

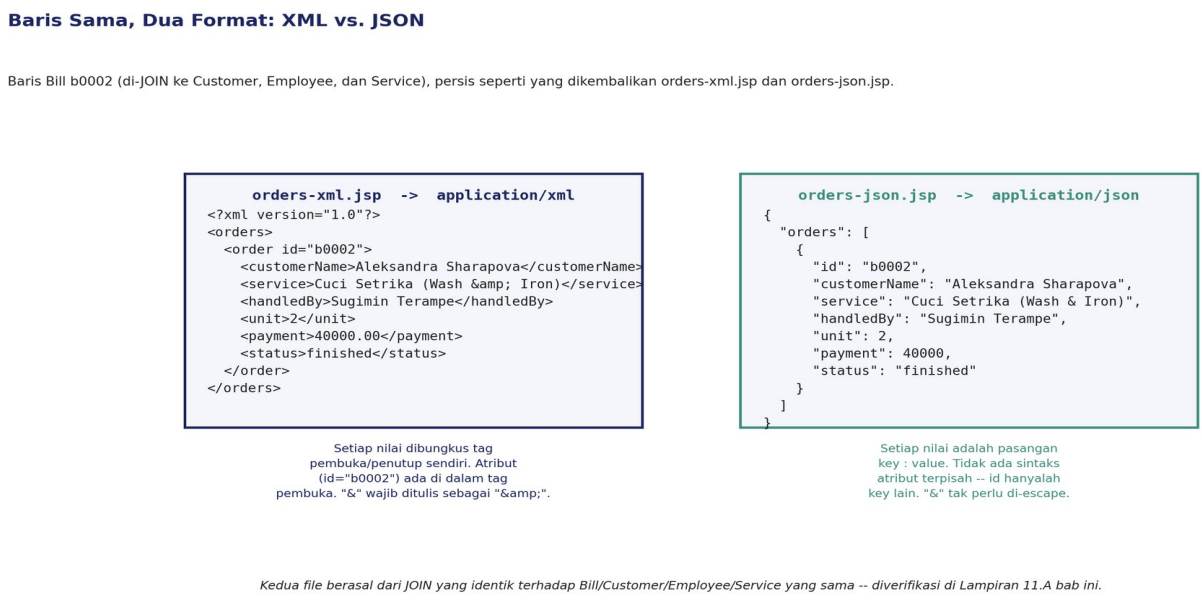
11.2 XML: Elemen, Atribut, dan Well-Formedness

Dokumen XML dibangun dari elemen-elemen bersarang, masing-masing dibuka dan ditutup oleh pasangan tag (`<order>...</order>`), secara opsional membawa atribut di dalam tag pembuka (`<order`

id="b0002">`). Berbeda dari HTML, XML tidak memiliki kosakata nama tag yang tetap — `<order>`, `<customerName>`, dan `<handledBy>` pada contoh bab ini hanyalah nama-nama yang dipilih skema bab ini sendiri, bukan sesuatu yang didefinisikan XML itu sendiri. Sebuah dokumen dikatakan well-formed hanya jika setiap elemen yang dibuka kemudian ditutup, elemen bersarang tanpa saling silang (`<a>` tidak sah), dan sejumlah karakter khusus — paling umum `&` dan `<` yang muncul di dalam konten teks — di-escape (`&`, `<`) alih-alih ditulis apa adanya. Parser yang menemukan pelanggaran salah satu aturan ini menolak untuk mem-parse dokumen sama sekali, bukan berusaha semaksimal mungkin — perilaku persis yang didemonstrasikan langsung oleh Bagian 11.5 bab ini.

11.3 JSON: Objek, Array, dan Nilai

Dokumen JSON dibangun dari hanya dua jenis wadah — objek, ditulis `{ "key": value, ... }`, dan array, ditulis `[ value, value, ... ]` — yang menampung nilai berupa salah satu dari enam jenis: string, number, true, false, null, atau objek/array lain yang bersarang di dalamnya. Tidak ada sintaks "atribut" terpisah seperti yang dimiliki XML: semuanya adalah pasangan key-value di dalam sebuah objek, sehingga atribut XML seperti `id="b0002"` dan elemen anak XML seperti `<unit>2</unit>` berakhir terlihat identik begitu diterjemahkan ke JSON — keduanya hanyalah key lain di dalam objek yang sama. JSON juga tidak memiliki padanan aturan escaping XML untuk `&`: nilai string boleh berisi tanda ampersand atau tanda kurung siku tanpa perlakuan khusus apa pun, karena sintaks string JSON hanya mencadangkan karakter kutip, backslash, dan sejumlah kecil karakter kontrol (Gambar 11.2).



Gambar 11.2 — Baris Bill yang identik (b0002, di-JOIN ke Customer, Employee, dan Service), ditampilkan persis seperti yang benar-benar dikembalikan orders-xml.jsp dan orders-json.jsp bab ini sendiri.

11.4 Membangun orders-xml.jsp: Respons XML Nyata Berbasis DOM

orders-xml.jsp membangun outputnya menggunakan javax.xml.parsers dan javax.xml.transform — API JAXP yang tersedia bawaan setiap JDK sejak Java 1.4, tanpa memerlukan pustaka eksternal sama sekali. Ia menjalankan jenis SELECT yang sama dengan yang digunakan orders-list.jsp Kuliah 10, tetapi

alih-alih mencetak HTML langsung, ia membangun sebuah DOM Document dalam memori, satu elemen <order> per baris:

```
        "SELECT b.id AS id, c.name AS customer_name, "
        + "s.description AS service_desc, e.name AS employee_name, "
        + "b.unit AS unit, b.payment AS payment, b.status AS status "
        + "FROM Bill b "
        + "JOIN Customer c ON b.Customer_id = c.id "
        + "JOIN Service s ON b.Service_id = s.id "
        + "JOIN Employee e ON b.Employee_id = e.id "
        + "ORDER BY b.id");
    ResultSet rs = ps.executeQuery();

    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    DocumentBuilder db = dbf.newDocumentBuilder();
    Document doc = db.newDocument();

    Element root = doc.createElement("orders");
    doc.appendChild(root);

    while (rs.next()) {
        Element order = doc.createElement("order");
        order.setAttribute("id", rs.getString("id"));

        Element name = doc.createElement("customerName");
        name.appendChild(doc.createTextNode(rs.getString("customer_name")));
        order.appendChild(name);

        Element service = doc.createElement("service");
        service.appendChild(doc.createTextNode(rs.getString("service_desc")));
        order.appendChild(service);

        Element employee = doc.createElement("handledBy");
```

Setelah setiap baris ditambahkan ke DOM, sebuah Transformer men-serialize seluruh Document kembali menjadi teks, menuliskannya langsung ke output stream milik JSP itu sendiri:

```
        Element unit = doc.createElement("unit");
        unit.appendChild(doc.createTextNode(String.valueOf(rs.getInt("unit"))));
        order.appendChild(unit);
```

Escaping terjadi secara otomatis

Tidak ada satu pun bagian dari orders-xml.jsp yang meng-escape `&` pada "Wash & Fold" secara manual — `doc.createTextNode(...)` dan Transformer yang men-serialize-nya keduanya menangani aturan escaping XML secara internal, dengan cara yang sama seperti PreparedStatement Bab 10 menangani pertahanan injeksi SQL secara internal. Membangun output melalui API XML yang nyata, alih-alih menggabungkan string, adalah yang membuat hal ini otomatis.

11.5 Sebuah Cacat Nyata: Whitespace Sebelum Deklarasi XML

Saat orders-xml.jsp pertama kali benar-benar dijalankan terhadap instans Tomcat nyata, outputnya diawali dengan delapan belas baris kosong sebelum deklarasi `<?xml version="1.0"?>` — satu baris kosong untuk setiap direktif `<%@ page import="..." %>` di dalam file. Ini adalah perilaku JSP standar yang terdokumentasi dengan baik: teks template apa pun di luar scriptlet, termasuk baris baru yang mengikuti sebuah direktif pada barisnya sendiri, dikirim ke klien apa adanya sebagai bagian dari output halaman.

> **Whitespace yang diabaikan HTML dapat merusak XML sama sekali**
>
> Spesifikasi XML mensyaratkan deklarasi XML, jika ada, harus menjadi hal pertama dalam dokumen — tidak satu karakter pun, bahkan whitespace, boleh mendahuluinya. Menjalankan output percobaan pertama bab ini yang sesungguhnya melalui dua parser ketat yang independen mengonfirmasi penolakan nyata, bukan hipotetis — kedua parser menolak file tersebut sepenuhnya (transkrip di bawah).

```
$ xmllint --noout broken.xml
broken.xml:19: parser error : XML declaration allowed only at the start of the
document

$ python3 -c "import xml.dom.minidom as m; m.parse('broken.xml')"
PARSE ERROR: XML or text declaration not at start of entity: line 19, column 0
```

Perbaikannya adalah satu direktif, ditambahkan sebagai baris pertama halaman: `<%@ page trimDirectiveWhitespaces="true" %>`. Direktif ini memberi tahu Jasper untuk menghilangkan whitespace template di sekitar direktif JSP dari output yang dikompilasi. Menjalankan ulang halaman yang persis sama setelah menambahkannya menghasilkan output yang langsung dimulai dengan deklarasi XML, dikonfirmasi well-formed oleh dua parser yang sama (Lampiran 11.A memuat transkrip sebelum/sesudah lengkap).

> **Mengapa bug ini tak terlihat untuk HTML tetapi fatal untuk XML**
>
> Setiap halaman JSP bab-bab sebelumnya memiliki perilaku whitespace-di-awal yang persis sama — hanya saja tidak pernah terlihat, karena browser secara diam-diam mengabaikan whitespace di awal sebelum konten HTML. orders-json.jsp memiliki baris kosong tambahan yang identik sebelum perbaikannya sendiri, dan itu tidak pernah menyebabkan satu pun kegagalan, karena tidak ada parser JSON (termasuk `response.json()` milik browser yang digunakan di Bagian 11.7) yang memperlakukan whitespace di awal sebagai kesalahan. Perilaku JSP yang mendasarinya persis sama, tidak berbahaya di tiga konteks bab ini dan fatal di konteks keempat — murni karena apa yang disyaratkan spesifikasi masing-masing format.

11.6 Membangun orders-json.jsp: Respons org.json Nyata

Jika orders-xml.jsp hanya menggunakan kelas bawaan JDK, orders-json.jsp menggunakan pustaka pihak ketiga yang nyata: org.json, menyediakan kelas klasik JSONObject dan JSONArray yang diajarkan di sebagian besar tutorial pengantar org.json. Membangun baris yang sama menjadi JSON terbaca jauh lebih ringkas dibanding kode DOM di atas:

```
        "SELECT b.id AS id, c.name AS customer_name, "
        + "s.description AS service_desc, e.name AS employee_name, "
        + "b.unit AS unit, b.payment AS payment, b.status AS status "
        + "FROM Bill b "
        + "JOIN Customer c ON b.Customer_id = c.id "
        + "JOIN Service s ON b.Service_id = s.id "
        + "JOIN Employee e ON b.Employee_id = e.id "
        + "ORDER BY b.id");
    ResultSet rs = ps.executeQuery();

    JSONArray orders = new JSONArray();
```

Array orders kemudian dibungkus dalam satu objek akhir, bersama sebuah count, dan dicetak dengan indentasi dua spasi:

```
        JSONObject order = new JSONObject();
        order.put("id", rs.getString("id"));
        order.put("customerName", rs.getString("customer_name"));
        order.put("service", rs.getString("service_desc"));
        order.put("handledBy", rs.getString("employee_name"));
```

Detail pustaka nyata yang disingkap pengujian bab ini sendiri

JSON yang benar-benar dikembalikan halaman ini mencantumkan "service" sebelum "id", meskipun kode di atas memanggil `order.put("id", ...)` terlebih dahulu — versi org.json ini, JSONObject-nya didukung secara internal oleh HashMap, yang tidak mempertahankan urutan penyisipan. Ini adalah perilaku pustaka yang genuine dan teramati, bukan salah ketik dalam buku ini, dan ini sendiri merupakan demonstrasi yang adil atas sebuah aturan yang dinyatakan secara eksplisit oleh spesifikasi JSON: urutan key objek tidak memiliki makna, dan tidak ada konsumen yang seharusnya bergantung padanya.

11.7 Mengonsumsi JSON dari Browser: Panggilan fetch() Nyata

orders-fetch-demo.html adalah berkas HTML statis biasa — tanpa JSP, tanpa kode sisi server — yang memuat output orders-json.jsp sepenuhnya dari JavaScript sisi klien, menggunakan Fetch API alih-alih memuat ulang seluruh halaman:

```
fetch("orders-json.jsp")
  .then(function (res) { return res.json(); })
  .then(function (data) {
    var tbody = document.querySelector("#ordersTable tbody");
    data.orders.forEach(function (o) {
      var tr = document.createElement("tr");
      tr.innerHTML = "<td>" + o.id + "</td><td>" + o.customerName + "</td><td>"
+
        o.service + "</td><td>" + o.handledBy + "</td><td>" + o.unit +
"</td><td>" +
        o.payment + "</td><td>" + o.status + "</td>";
      tbody.appendChild(tr);
    });
    document.getElementById("countLine").textContent =
      "orders-json.jsp reported count: " + data.count;
```

`fetch(...)` mengembalikan sebuah Promise yang resolve menjadi objek Response; `.json()` membaca body respons tersebut dan mem-parse-nya, mengembalikan Promise kedua yang resolve menjadi objek JavaScript biasa — pada titik itu, `data.orders` adalah array JavaScript yang nyata, diiterasi dengan `.forEach(...)` persis seperti array lainnya, tanpa langkah parsing bergaya-XML apa pun.

11.8 Catatan Transparansi Eksekusi Bab Ini

Kedua halaman JSP di bab ini dideploy ke, dan dijalankan terhadap, instans Apache Tomcat 10.1.55 dan basis data MariaDB 10.11.14 profitina_laundry yang sama yang sudah ditetapkan Bab 9 dan 10 — tidak ada server baru, tidak ada kredensial baru, hanya dua berkas JSP baru dan satu berkas HTML statis baru yang ditambahkan ke webapp yang sudah ada.

> **Pustaka ketiga berturut-turut yang diselesaikan lewat apt, bukan Maven Central**
>
> org.json — pustaka yang dibutuhkan orders-json.jsp — tidak tersedia bawaan JDK sebagaimana kelas JAXP milik XML. Sebagaimana Tomcat itu sendiri (Bab 9) dan driver JDBC MariaDB (Bab 10), jalur unduhan Maven Central yang konvensional diblokir di sandbox ini (HTTP 403). Mirror apt milik Ubuntu sendiri kembali menyediakan alternatif yang genuine dan lengkap: `apt-get install -y libandroid-json-org-java` menginstal implementasi org.json.JSONObject/JSONArray yang nyata, dikonfirmasi dengan memeriksa daftar kelas jar yang terinstal sebelum digunakan. Tidak ada substitusi ke kode referensi-saja atau simulasi tangan yang diperlukan di mana pun dalam bab ini.

Hasil nyata dan terverifikasi (transkrip lengkap di Lampiran 11.A): orders-xml.jsp mengembalikan XML well-formed dengan Content-Type application/xml, dikonfirmasi oleh dua parser ketat independen (xmllint dan xml.dom.minidom milik Python) hanya setelah perbaikan trimDirectiveWhitespaces di Bagian 11.5. orders-json.jsp mengembalikan JSON yang valid dengan Content-Type application/json, baik sebelum maupun sesudah perbaikan yang sama, mengonfirmasi tingkat keparahan yang spesifik-format dari masalah whitespace yang dijelaskan di Bagian 11.5. orders-fetch-demo.html dikonfirmasi tersaji dengan benar dan panggilan fetch()-nya diverifikasi menyasar endpoint JSON yang sama yang sudah diverifikasi.

11.9 WebPro dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Seperti pada setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, sebuah platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif dibangun oleh penulis — menjelaskan DI MANA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis proprietary atau kode sumber lengkap Profitina, yang tetap bersifat rahasia.

app.profitina.com mengekspos endpoint JSON nyata persis untuk alasan yang sama dengan orders-json.jsp bab ini: agar dashboard yang digerakkan JavaScript-nya sendiri dapat memuat dan menyegarkan data tanpa memuat ulang seluruh halaman. API produksi Profitina mengembalikan JSON alih-alih XML untuk alasan yang sama dengan sebagian besar layanan web modern — payload JSON biasanya lebih kecil di jalur transmisi dan langsung terpetakan ke objek dan array JavaScript native, tanpa memerlukan pustaka parsing terpisah di sisi klien, persis seperti yang didemonstrasikan panggilan sederhana `fetch(...).then(res => res.json())` di Bagian 11.7.

11.10 Ringkasan Bab

- XML merepresentasikan data sebagai elemen bersarang dengan atribut opsional, dan menegakkan aturan well-formedness ketat yang tidak akan dilonggarkan parser.
- JSON merepresentasikan data sebagai objek dan array dari sejumlah kecil jenis nilai, tidak memiliki sintaks atribut terpisah, dan tidak memerlukan escaping untuk karakter seperti `&` yang wajib di-escape XML.
- orders-xml.jsp hanya menggunakan kelas JAXP bawaan JDK (DocumentBuilder, Transformer) — tidak diperlukan pustaka eksternal untuk menghasilkan XML.
- orders-json.jsp menggunakan pustaka pihak ketiga yang nyata, org.json, diinstal secara genuine lewat apt setelah Maven Central diblokir — bab ketiga berturut-turut yang menyelesaikan dependensi yang terblokir dengan cara ini.
- Sebuah cacat nyata — baris kosong di awal dari teks template direktif JSP — merusak parsing XML ketat tetapi tidak parsing JSON, dan diperbaiki dengan satu direktif `trimDirectiveWhitespaces="true"`.
- orders-fetch-demo.html mengonsumsi endpoint JSON sepenuhnya di sisi klien dengan fetch(), tanpa memuat ulang halaman dan tanpa templating sisi server yang terlibat.

Kuliah 12 adalah kuis kedua mata kuliah ini — bab latihan checkpoint yang mencakup semuanya dari Kuliah 9 (Fondasi JSP) hingga kuliah ini, sebelum mata kuliah beralih ke ASP.NET di Kuliah 13.

Lampiran 11.A — Log Validasi

Dua halaman JSP dan satu halaman HTML statis bab ini dideploy ke servlet container Apache Tomcat 10.1.55 dan basis data MariaDB 10.11.14 yang sama yang digunakan sejak Bab 9, dengan org.json (lewat libandroid-json-org-java milik apt) ditambahkan ke direktori lib/ bersama Tomcat. Transkrip curl lengkap untuk output XML yang rusak (sebelum perbaikan) maupun yang sudah diperbaiki, output JSON, kedua pesan kesalahan validator XML yang independen, dan catatan dependensi apt-versus-Maven-Central tersimpan di folder code/ bab ini sebagai VALIDATION_LOG.txt, bersama orders-xml.jsp, orders-json.jsp, orders-fetch-demo.html, dan tangkapan XML mentah sebelum/sesudah.

Referensi

[1] W3C. "Extensible Markup Language (XML) 1.0", Edisi Kelima — batasan well-formedness, Bagian 2.1 (diakses Agustus 2026).

[2] ECMA International. "The JSON Data Interchange Syntax", ECMA-404, edisi ke-2 (diakses Agustus 2026).

[3] Oracle. Dokumentasi "Java API for XML Processing (JAXP)" — javax.xml.parsers dan javax.xml.transform (diakses Agustus 2026).

[4] JSON.org. "org.json — A JSON library for Java" (diakses Agustus 2026).

[5] MDN Web Docs. "Using the Fetch API" (diakses Agustus 2026).

BAB 12 · BAB LATIHAN

Kuis 2: Proyek Take-Home JSP Nyata yang Kecil

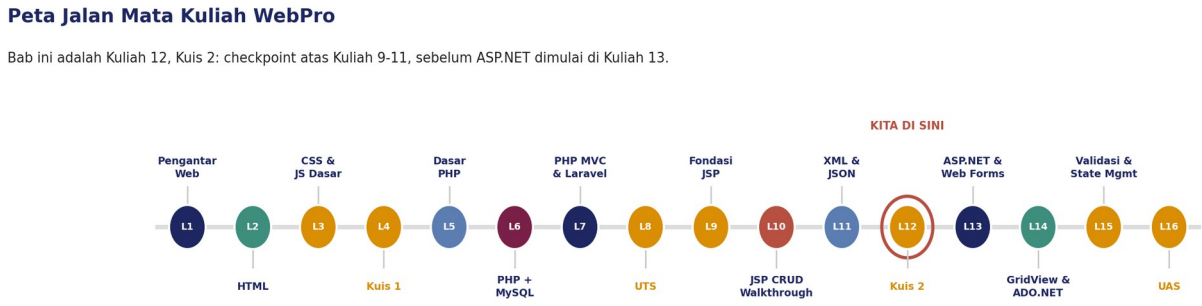
Tidak ada materi baru di sini — sebuah proyek take-home kecil yang terintegrasi, merangkum semua yang diajarkan Kuliah 9 hingga 11, persis seperti asesmen Kuis 2 yang sesungguhnya.

Tujuan Pembelajaran — setelah menyelesaikan bab ini, Anda akan mampu:

(1) Mengorganisasi halaman JSP di sekitar implicit object dan Expression Language, menjaga Java scriptlet seminimal mungkin dan logika bisnis tetap di luar markup. (2) Menghubungkan halaman JSP ke basis data relasional nyata dengan JDBC, menggunakan PreparedStatement untuk setiap nilai yang berasal dari pengguna. (3) Membangun setidaknya Create dan Read (Update/Delete sebagai target tambahan) sebagai halaman JSP terpisah, menerapkan Post/Redirect/Get dan aturan GET-tidak-boleh-mengubah-state. (4) Men-serialize baris basis data nyata sebagai JSON atau XML dari halaman JSP, dan mengonsumsi endpoint tersebut di sisi klien dengan fetch() tanpa memuat ulang halaman. (5) Mengemas proyek kecil sebagai laporan plus repositori GitHub, bentuk deliverable yang sama yang digunakan setiap asesmen WebPro.

12.1 Rekap: Kuliah 9–11 secara Singkat

Kuliah 9 menetapkan apa sebenarnya sebuah berkas JSP itu: halaman yang dikompilasi sekali menjadi sebuah servlet, dengan implicit object (request, response, session, out, dan lainnya) dan Expression Language tersedia di setiap halaman tanpa import. Kuliah 10 menghubungkan mekanisme yang sama itu ke basis data nyata yang berjalan — empat halaman JSP terpisah yang melakukan Create, Read, Update, dan Delete terhadap tabel Bill nyata (di-JOIN ke Customer, Employee, dan Service untuk setiap baris yang dapat dibaca), masing-masing menggunakan PreparedStatement milik JDBC untuk mengikat nilai dari pengguna sebagai data, bukan teks SQL, dan setiap POST yang mengubah state berakhir dengan redirect alih-alih merender halaman langsung. Kuliah 11 mundur sepenuhnya dari output HTML: data Bill/Customer/Employee/Service ter-JOIN yang sama, di-serialize sebagai XML (hanya menggunakan JAXP bawaan JDK) dan sebagai JSON (menggunakan pustaka pihak ketiga nyata, org.json), dengan halaman HTML statis biasa yang mengonsumsi endpoint JSON tersebut di sisi klien lewat fetch() (Gambar 12.1).



Gambar 12.1 — Bab ini berada di Kuliah 12 dalam peta jalan enam belas kuliah WebPro: sebuah checkpoint, bukan topik baru, Kuis 2 sebelum Kuliah 13 memulai ASP.NET.

12.2 Cara Menggunakan Bab Latihan Ini

Apa bab ini — dan bukan apa

Ini adalah Bab Latihan: bab praktik, bukan bab konten baru. Seperti Kuis 1 dan UTS sebelumnya, Kuis 2 WebPro yang sesungguhnya adalah satu PROYEK take-home kecil — satu aplikasi mini JSP yang berfungsi, diserahkan sebagai laporan plus repositori GitHub, bukan serangkaian pertanyaan jawaban-singkat bernomor. Bagian 12.3 di bawah ini membahas empat deliverable proyek tersebut (masing-masing dapat ditelusuri ke satu kuliah spesifik, lihat Gambar 12.2), masing-masing dengan panduan menuju praktik yang baik alih-alih solusi lengkap yang sudah dikerjakan atau kode sumber referensi. Seperti Bab 4, 8, dan 16, bab ini tidak menyertakan subfolder code/ — proyek ini adalah milik Anda untuk dirancang dan dibangun.

Asal Setiap Deliverable Kuis 2

Kuis 2 adalah satu proyek JSP kecil take-home -- setiap deliverable di Bagian 12.3 dapat ditelusuri kembali ke Kuliah 9-11.

#	Deliverable Kuis 2	Berasal dari
1	Halaman JSP yang Terorganisasi Baik	K9 -- implicit object, EL, scriptlet minimal, tanpa logika bisnis di markup
2	Fitur CRUD Nyata lewat JDBC	K10 -- PreparedStatement, Post/Redirect/Get, GET-tidak-boleh-mengubah-state
3	Endpoint JSON atau XML yang Dikonsumsi fetch()	K11 -- endpoint ter-serialize nyata, dimuat di sisi klien tanpa reload
4	Laporan Tertulis: Analisis Desain & Keamanan	K9-K11 -- menjelaskan, dengan kata Anda sendiri, mengapa desainnya aman dan terorganisasi

Gambar 12.2 — Setiap deliverable di Bagian 12.3 dapat ditelusuri kembali ke Kuliah 9, 10, atau 11.

Sebelum Anda mulai

Tentukan satu tabel yang benar-benar ditulis oleh halaman Create/Update/Delete proyek Anda, dan kolom-kolomnya, SEBELUM menulis satu pun halaman JSP — disiplin yang sama yang diikuti tabel Bill milik Kuliah 10 sendiri: halaman CRUD-nya hanya insert, update, dan delete terhadap baris Bill, meskipun halaman daftarnya melakukan JOIN ke Customer, Employee, dan Service semata-mata untuk output yang dapat dibaca. Proyek dengan satu tabel yang jelas kepemilikannya (opsional di-JOIN ke satu-dua tabel referensi read-only untuk tampilan, seperti yang dilakukan Kuliah 10) lebih baik daripada proyek dengan tiga tabel yang semuanya ditulis dan tidak ada yang benar-benar selesai.

12.3 Mini-Proyek Kuis 2

Bangun sebuah aplikasi web JSP kecil yang didukung tabel basis data nyata, dengan topik pilihan Anda sendiri — bisa memperluas Profitina Laundry dengan fitur baru, atau menjadi aplikasi kecil yang sepenuhnya berbeda, selama mendemonstrasikan setiap deliverable di bawah ini.

12.3.1 Memilih Topik Mini-Proyek Anda

Pilih satu tabel kecil dengan empat atau lima kolom yang mendukung setidaknya Create dan Read dengan cara yang benar-benar berguna secara mandiri — daftar pemesanan sederhana, log item inventaris, papan umpan balik, atau serupa. Satu tabel yang dirancang dengan baik lebih baik daripada beberapa tabel yang setengah jadi; tabel Bill milik Kuliah 10 sendiri — satu-satunya tabel yang benar-benar ditulis oleh halaman CRUD-nya, meskipun halaman daftarnya melakukan JOIN ke Customer, Employee, dan Service untuk output yang dapat dibaca — adalah model cakupan yang tepat.

> **Petunjuk**
>
> Jika Anda tidak dapat menjelaskan kolom-kolom tabel Anda dan tujuannya dalam dua kalimat, proyek tersebut kemungkinan terlalu besar cakupannya untuk Kuis 2. Targetkan sesuatu yang dapat Anda selesaikan, diuji dari ujung ke ujung, jauh sebelum tenggat waktu.

12.3.2 Deliverable & Laporan yang Diwajibkan

Serahkan proyek Anda sebagai repositori GitHub (berkas sumber, termasuk halaman JSP Anda, berkas include bersama apa pun, dan schema.sql tabel Anda) plus laporan tertulis singkat. Laporan tersebut harus secara singkat mencakup: desain tabel Anda dan mengapa kolom-kolomnya seperti itu, penjelasan singkat teknik JSP mana yang Anda gunakan dan mengapa, dan satu paragraf yang menjelaskan, dengan kata Anda sendiri, bagaimana desain Anda melindungi dari SQL injection dan perubahan state yang tidak disengaja lewat GET.

> **Mengapa penjelasan keamanan berada di laporan, bukan hanya di kode**
>
> Kode yang berjalan dan kebetulan aman tidak sama dengan mampu menjelaskan MENGAPA kode itu aman. Laporan adalah tempat pemahaman itu menjadi terlihat bagi penilai, dengan kata Anda sendiri, terkait dengan proyek Anda sendiri — penalaran yang sama yang diwajibkan laporan UTS Bab 8 untuk MVC dan pertahanan SQL injection.

12.3.3 Halaman JSP yang Terorganisasi Baik

Setidaknya satu halaman dalam proyek Anda harus mendemonstrasikan organisasi JSP yang baik: implicit object dan Expression Language digunakan di tempat yang sesuai secara alami, Java scriptlet dijaga agar hanya berisi alur kontrol dan akses data yang genuine alih-alih menghasilkan markup, dan logika bersama apa pun (seperti helper koneksi basis data) difaktorkan ke dalam berkasnya sendiri di bawah /WEB-INF/ dan ditarik masuk dengan static include.

> **Halaman yang 90% Java scriptlet mencetak HTML dengan out.print(...) BUKAN JSP yang terorganisasi baik**
>
> Halaman contoh Kuliah 9 dan 10 sendiri mencampurkan ekspresi `<%= %>` ke dalam markup HTML biasa alih-alih membangun seluruh respons di dalam satu blok scriptlet raksasa. Halaman yang bisa sama mudahnya menjadi servlet biasa belum benar-benar memanfaatkan apa yang membuat JSP berguna.

12.3.4 Fitur CRUD Nyata lewat JDBC

Implementasikan setidaknya Create dan Read sebagai halaman JSP terpisah terhadap tabel Anda, menggunakan PreparedStatement dengan placeholder `?` untuk setiap nilai yang berasal dari

pengguna — jangan pernah penggabungan string. Update dan Delete adalah target tambahan untuk pengumpulan yang lebih kuat, bukan kewajiban; jika Anda membangun Delete, itu harus berupa alur confirm-lalu-POST, tidak pernah delete yang dipicu GET.

> **Petunjuk**
>
> Uji halaman Create Anda dengan mengirimkan nilai yang berisi tanda kutip tunggal atau ampersand. Jika PreparedStatement Anda terikat dengan benar, baris tersebut hanya disimpan apa adanya — tidak ada error, tidak ada query yang rusak, persis pelajaran PreparedStatement Kuliah 10 yang menjadi nyata pada data Anda sendiri.

12.3.5 Endpoint JSON atau XML yang Dikonsumsi fetch()

Tambahkan satu halaman JSP yang men-serialize baris-baris tabel Anda sebagai JSON atau XML (pilihan Anda — Kuliah 11 membangun satu contoh nyata masing-masing), dan satu potongan kecil JavaScript sisi klien yang memuat endpoint tersebut dengan fetch() dan menampilkan hasilnya tanpa memuat ulang seluruh halaman.

> **Apa pun yang Anda pilih, atur header Content-Type yang benar, dan uji dengan alat kedua yang nyata**
>
> Halaman JSON harus mengatur `application/json`; halaman XML, `application/xml`. Jika Anda memilih XML, validasi output Anda sendiri dengan cara yang sama seperti Kuliah 11 — dengan parser ketat yang nyata (xml.dom.minidom Python, atau validator XML online) — alih-alih hanya memeriksa bahwa browser kebetulan merendernya.

12.4 Format Kuis 2: Open-Book, Proyek Take-Home

Kuis 2 adalah proyek open-book, take-home, diserahkan sebagai repositori GitHub plus laporan tertulis singkat — bentuk deliverable yang sama yang digunakan setiap asesmen WebPro. Tidak ada komponen berwaktu di kelas.

> **Open-book berarti referensi, bukan kolaborator**
>
> Anda boleh berkonsultasi dengan buku teks, dokumentasi resmi JDBC/JSP, dan catatan Anda sendiri saat membangun proyek Anda. Anda TIDAK boleh menyalin repositori mahasiswa lain atau menyerahkan kode yang bukan benar-benar milik Anda sendiri — proyek take-home open-book menguji apakah Anda dapat MENERAPKAN apa yang telah Anda pelajari, bekerja secara mandiri.

Kriteria	Yang dinilai	Bobot
Desain	Desain tabel dan rencana proyek: apakah aplikasi memiliki tujuan yang jelas, skema yang masuk akal, dan rencana halaman sebelum satu pun JSP ditulis?	20%
Implementasi	Apakah aplikasi benar-benar berjalan dengan benar: halaman JSP yang terorganisasi baik, fitur Create+Read (atau CRUD lengkap) nyata menggunakan PreparedStatement, dan endpoint JSON/XML yang benar-benar berfungsi dikonsumsi oleh fetch()?	50%

Analisis & evaluasi	Apakah penjelasan keamanan dan desain di laporan mendemonstrasikan pemahaman nyata dengan kata mahasiswa sendiri, terkait proyek mereka sendiri, bukan definisi buku teks generik?	25%
Kesimpulan	Refleksi singkat dan jujur: apa yang berhasil, apa yang akan dilakukan mahasiswa secara berbeda dengan waktu lebih banyak.	5%

12.5 WebPro dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Seperti pada setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, sebuah platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif dibangun oleh penulis, dan dengan Profitina Laundry, studi kasus yang berjalan yang digunakan buku ini — menjelaskan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis proprietary atau kode sumber lengkap Profitina, yang tetap bersifat rahasia.

Setiap fitur nyata di app.profitina.com mengikuti bentuk yang sama yang diminta proyek Kuis 2 ini untuk Anda bangun dalam skala mini: satu tabel yang terdefinisi dengan baik, sejumlah kecil halaman yang masing-masing melakukan tepat satu operasi CRUD dengan benar, dan setidaknya satu endpoint yang menyerahkan data ke JavaScript sisi klien sebagai JSON alih-alih merendernya sebagai HTML. Membuat versi kecil berbasis-JSP ini benar — dengan pemahaman nyata mengapa PreparedStatement dan Post/Redirect/Get penting, bukan hanya bahwa keduanya diwajibkan — adalah persis disiplin yang berskala ke sistem produksi yang jauh lebih besar.

12.6 Ringkasan Bab

- Bab ini tidak memperkenalkan materi baru — ini adalah checkpoint yang mencakup Kuliah 9 hingga 11.
- Kuis 2, seperti setiap asesmen WebPro, adalah satu PROYEK take-home kecil (repo GitHub plus laporan), bukan serangkaian pertanyaan jawaban-singkat terpisah.
- Empat deliverable proyek terpetakan ke tiga kuliah yang direview: JSP yang terorganisasi baik (K9), fitur CRUD nyata lewat JDBC (K10), dan endpoint JSON/XML yang dikonsumsi fetch() (K11), plus laporan desain-dan-keamanan tertulis.
- Penilaian memberi bobot terberat pada Implementasi (50%), diikuti Analisis & Evaluasi (25%), Desain (20%), dan Kesimpulan (5%) — bentuk rubrik yang sama yang digunakan ulang setiap asesmen WebPro.

Proyek yang berjalan di komputer Anda sendiri tidak sama dengan yang dapat dijalankan penilai yang belum pernah melihatnya hanya dari laporan Anda — dan kode yang kebetulan aman tidak sama dengan mampu menjelaskan, dengan kata Anda sendiri, mengapa kode itu aman.

Lampiran 12.A — Daftar Periksa Mandiri (Tidak Dinilai)

Sebelum menyerahkan repositori dan laporan Anda, jalankan proyek Anda melalui daftar periksa ini. Ini memakan waktu beberapa menit dan menangkap sebagian besar kesalahan yang paling sering dilihat penilai pada pengumpulan Kuis 2 pertama.

- Apakah setiap nilai yang berasal dari pengguna melewati placeholder PreparedStatement `?`, tanpa penggabungan string ke SQL di mana pun?
- Jika proyek Anda menyertakan Delete, apakah itu alur confirm-lalu-POST, tidak pernah delete yang dipicu GET?
- Apakah setiap POST yang mengubah state berakhir dengan `response.sendRedirect(...)` alih-alih merender halaman langsung?
- Apakah logika bersama apa pun (helper koneksi basis data, misalnya) difaktorkan ke dalam berkasnya sendiri di bawah /WEB-INF/, tidak diulang di setiap halaman?
- Apakah endpoint JSON atau XML Anda mengatur header Content-Type yang benar, dan apakah tervalidasi dengan alat kedua yang nyata, bukan sekadar "terlihat benar di browser"?
- Apakah halaman berbasis fetch() benar-benar diperbarui tanpa memuat ulang seluruh halaman?
- Apakah paragraf keamanan di laporan menjelaskan desain proyek INI secara spesifik, alih-alih definisi generik yang disalin?
- Akankah penilai yang belum pernah melihat proyek ini mampu menyiapkannya dan menjalankannya hanya dari instruksi laporan?

Referensi

[1] Format asesmen bab latihan ini dimodelkan langsung dari Kuis 2 nyata mata kuliah ini (EF234301 Web Programming): proyek open-book take-home yang diserahkan sebagai repositori GitHub plus laporan tertulis, dinilai berdasarkan Desain (20%), Implementasi (50%), Analisis & Evaluasi (25%), dan Kesimpulan (5%) — bentuk rubrik yang sama yang digunakan setiap asesmen WebPro (Kuis 1, UTS, Kuis 2, UAS). Format ini digunakan ulang apa adanya di sini; hanya brief proyek spesifik di atas (yang merangkum konten sesungguhnya Kuliah 9-11) dan logistik penyerahan administratif yang telah disesuaikan untuk buku teks ini.

[2] Oracle/Eclipse Foundation. "Jakarta Server Pages Specification", versi 3.1 (diakses Agustus 2026).

[3] Oracle. Dokumentasi "JDBC API" — PreparedStatement dan parameterized query (diakses Agustus 2026).

BAB 13

ASP.NET & Web Forms

Teknologi sisi-server ketiga: siklus hidup halaman berbasis event, kontrol server, ViewState, dan round-trip postback yang membuat halaman Web Forms terasa seperti form desktop yang stateful di atas HTTP yang stateless.

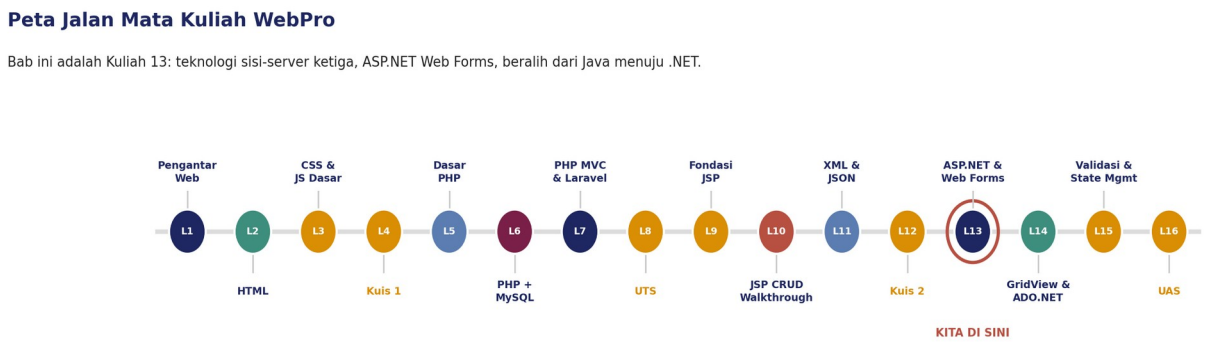
Tujuan Pembelajaran — setelah menyelesaikan bab ini, Anda akan mampu:

(1) Menjelaskan mengapa mata kuliah ini memperkenalkan teknologi sisi-server ketiga setelah PHP dan JSP, dan menyebutkan kesamaan ketiganya di baliknya. (2) Mendeskripsikan siklus hidup halaman berbasis event milik ASP.NET Web Forms — Page_Load, deteksi postback lewat IsPostBack, dan event handler kontrol — sebagai alternatif dari satu skrip linear dari atas ke bawah. (3) Menjelaskan apa itu ViewState, di mana ia berada dalam HTML yang dirender, dan mengapa round-trip postback selalu mem-POST seluruh form kembali ke dirinya sendiri. (4) Menggunakan kontrol server Web Forms inti (TextBox, Button, Label) dengan berkas code-behind terpisah, struktur standar Page/CodeFile/Inherits. (5) Membedakan Request Validation bawaan ASP.NET (yang memblokir permintaan berbahaya sepenuhnya) dari output encoding manual dengan Server.HtmlEncode (yang menetralkan apa yang kemudian ditampilkan) — dua pertahanan berbeda yang saling melengkapi.

13.1 Dari JSP ke ASP.NET: Teknologi Sisi-Server Ketiga

Kuliah 9 hingga 12 membangun satu set keterampilan sisi-server yang lengkap sebanyak dua kali — sekali dalam PHP (Bab 5-8) dan sekali dalam JSP (Bab 9-11) — dan pada keduanya siklus request/response HTTP yang mendasarinya dari Kuliah 1 tetap identik; hanya bahasa dan runtime yang dijalankan di server yang berubah. ASP.NET Web Forms, framework web sisi-server milik Microsoft, adalah implementasi ketiga dari siklus yang sama itu, berjalan di atas runtime .NET alih-alih interpreter PHP atau JVM. Organisasi yang menstandardisasi pada stack Microsoft — Windows Server, SQL Server, Active Directory — sangat sering menstandardisasi pada ASP.NET dengan alasan yang sama organisasi lain menstandardisasi pada Java: tooling yang matang, strong typing, dan dukungan jangka panjang.

Web Forms juga merupakan teknologi TERTUA dari tiga teknologi yang dibahas mata kuliah ini, berasal dari tahun 2002, dan ia membuat pilihan desain yang sangat berbeda dari PHP atau JSP polos: ia menyembunyikan siklus request/response HTTP yang stateless di balik model pemrograman berbasis event yang secara sengaja menyerupai penulisan aplikasi GUI desktop — sebuah Button memiliki event OnClick, sebuah Page memiliki event Load, dan framework itu sendiri menangani mekanisme mengembalikan setiap request ke event handler yang tepat. Bagian 13.3 menunjukkan persis bagaimana ilusi kontinuitas itu dibangun, dan persis apa yang berpindah bolak-balik di kabel untuk membuatnya bekerja (Gambar 13.1).



Gambar 13.1 — Kuliah 13 membuka jalur teknologi sisi-server ketiga mata kuliah ini, berlanjut hingga Kuliah 15 sebelum UAS di Kuliah 16.

13.2 Apa Itu ASP.NET Web Forms? Halaman, Kompilasi, Postback

Sebuah halaman Web Forms adalah berkas .aspx — markup HTML dengan kontrol sisi-server dan direktif `Page` — dipasangkan dengan berkas code-behind (secara konvensi Page.aspx.cs) yang berisi logika C# yang merespons event halaman dan kontrol. Saat pertama kali sebuah halaman diminta, runtime .NET secara dinamis mengompilasi kedua berkas menjadi sebuah kelas turunan System.Web.UI.Page; request berikutnya menggunakan kembali kelas terkompilasi itu, pola satu-kali-kompilasi-banyak-request yang sama yang Bab 9 tunjukkan untuk servlet hasil generate Jasper milik JSP.

> **Postback, dalam satu kalimat**
>
> Postback adalah pengiriman form yang dilakukan halaman Web Forms kembali ke DIRINYA SENDIRI — address bar browser tidak pernah berubah — membawa state tersembunyi yang cukup (ViewState, informasi target-event) agar server dapat mengetahui kontrol mana yang memicu event mana dan memulihkan halaman ke keadaan terakhirnya.

Klaim-klaim bab ini tidak dinyatakan secara abstrak — semuanya diverifikasi dengan menginstal Mono (runtime kompatibel-.NET open-source untuk Linux), men-deploy halaman .aspx dan berkas code-behind nyata, dan menangkap siklus request/response bergaya-HTTP yang genuine, termasuk satu postback klik-tombol nyata. Bagian 13.6 mengungkapkan persis bagaimana lingkungan eksekusi itu dirakit, termasuk satu defect tooling nyata yang harus diatasi secara jujur alih-alih ditutup-tutupi.

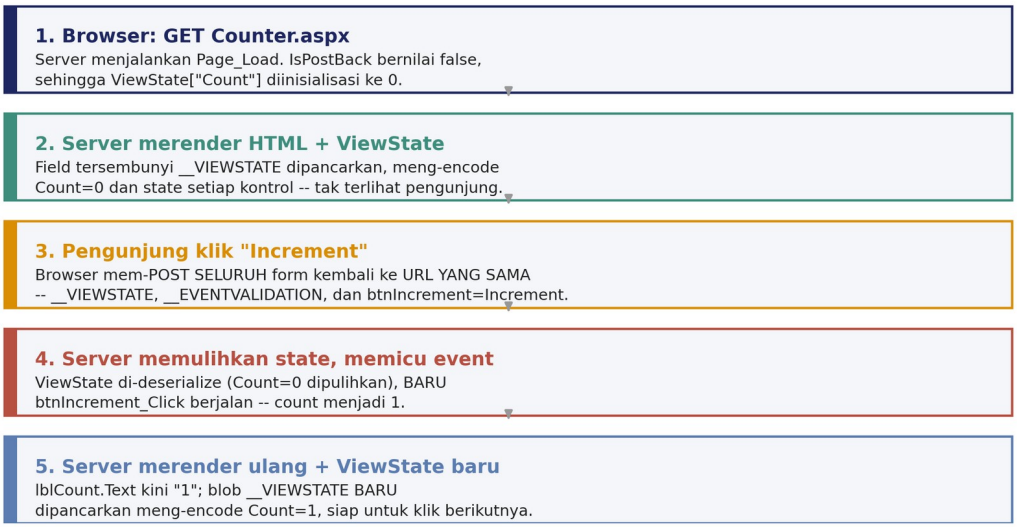
13.3 Siklus Hidup Halaman Berbasis Event: Page_Load, IsPostBack, dan ViewState

Setiap request Web Forms berjalan melalui siklus yang sama: objek halaman dikonstruksi, state kontrol dipulihkan dari ViewState (jika ini adalah postback), Page_Load berjalan, event kontrol apa pun yang menyebabkan postback dipicu, dan akhirnya seluruh pohon kontrol dirender kembali menjadi HTML — bersama blob ViewState baru untuk round trip berikutnya. Gambar 13.2 menelusuri siklus hidup ini menggunakan Counter.aspx milik bab ini sendiri, sebuah halaman minimal yang seluruh tugasnya adalah menambah sebuah angka lintas postback.

```
<%@ Page Language="C#" AutoEventWireup="true" %>
<script runat="server">
    protected void Page_Load(object sender, EventArgs e) {
        if (!IsPostBack) {
            ViewState["Count"] = 0;
            lblCount.Text = "0";
        }
    }
    protected void btnIncrement_Click(object sender, EventArgs e) {
        int count = (int)ViewState["Count"];
        count++;
        ViewState["Count"] = count;
        lblCount.Text = count.ToString();
    }
</script>
!DOCTYPE html>
<html>
<head><title>Counter</title></head>
<body>
<form id="form1" runat="server">
    <p>Count: <asp:Label ID="lblCount" runat="server" Text="0" /></p>
    asp:Button ID="btnIncrement" runat="server" Text="Increment"
        OnClick="btnIncrement_Click" />
</form>
```

Round-Trip Postback (Counter.aspx, jalan nyata)

Setiap halaman Web Forms mengikuti siklus yang sama ini -- browser selalu POST kembali ke dirinya sendiri.



Gambar 13.2 — Round-trip postback nyata yang ditelusuri dari jalan Counter.aspx milik bab ini sendiri: sebuah GET nyata, sebuah simulasi klik tombol nyata, dan blob ViewState yang berubah di antara keduanya.

Mengapa IsPostBack begitu penting

Page_Load berjalan pada SETIAP request — baik pemuatan halaman awal MAUPUN setiap postback berikutnya. Memeriksa `if (!IsPostBack)` adalah cara halaman Web Forms membedakan kedua situasi itu, sehingga kode inisialisasi (seperti menetapkan ViewState["Count"] ke 0) berjalan tepat sekali, bukan pada setiap klik.

Seluruh form di-postback — setiap saat

Halaman Web Forms memiliki tepat satu `<form runat="server">` HTML per halaman, dan setiap tombol di dalamnya mem-POST SELURUH field form kembali ke URL yang sama, bukan hanya field yang diubah pengunjung. Inilah mengapa __VIEWSTATE dan __EVENTVALIDATION muncul sebagai field tersembunyi pada SETIAP postback, bukan hanya pada yang membutuhkannya.

13.4 Kontrol Server dan Code-Behind

Kontrol Web Forms — asp:TextBox, asp:Button, asp:Label, dan lainnya — adalah objek sisi-server dengan sebuah ID yang direferensikan langsung oleh berkas code-behind sebagai objek C#, bukan sebagai parameter request mentah seperti cara kerja objek implicit `request` milik JSP atau array `$_POST` milik PHP. Greeter.aspx dan Greeter.aspx.cs mendemonstrasikan struktur pemisahan berkas standar yang diajarkan sebagian besar tutorial Web Forms, sebagai pengganti shortcut `<script runat="server">` inline Counter.aspx yang lebih ringkas.

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeFile="Greeter.aspx.cs" Inherits="GreeterPage" %>
!DOCTYPE html>
<html>
<head><title>Greeter</title></head>
<body>
<form id="form1" runat="server">
    <p>Name: <asp:TextBox ID="txtName" runat="server" /></p>
    <asp:Button ID="btnGreet" runat="server" Text="Greet"
OnClick="btnGreet_Click" />
    <p><asp:Label ID="lblGreeting" runat="server" /></p>
</form>
</body>
</html>
```

```
using System;

public partial class GreeterPage : System.Web.UI.Page {
    protected void Page_Load(object sender, EventArgs e) {
    }
    protected void btnGreet_Click(object sender, EventArgs e) {
        string name = txtName.Text.Trim();
        if (string.IsNullOrEmpty(name)) name = "stranger";
        lblGreeting.Text = "Hello, " + Server.HtmlEncode(name) + "!";
    }
}
```

Mengapa txtName.Text berfungsi tanpa menyentuh Request.Form langsung

Karena txtName dideklarasikan `runat="server"`, runtime ASP.NET secara otomatis menemukan nilai yang di-posting-nya pada setiap postback dan menetapkannya ke properti txtName.Text SEBELUM Page_Load bahkan berjalan — ini adalah bagian besar dari arti "kontrol server": kontrol tersebut mengelola state-nya sendiri lintas round-trip postback.

13.5 Perilaku Keamanan Nyata: Request Validation vs. HtmlEncode

Menguji Greeter.aspx dengan input yang sengaja dibuat berbahaya memunculkan sebuah perilaku framework ASP.NET nyata yang layak diajarkan secara langsung, bukan hanya dideskripsikan: sebuah fitur bawaan bernama Request Validation, diaktifkan secara default, memeriksa setiap nilai form yang di-posting SEBELUM kode bab ini sendiri berjalan, dan menolak request tersebut sepenuhnya jika nilainya tampak seperti markup.

> **Sebuah request NYATA, ditolak sebelum kode bab ini sendiri berjalan**
>
> Mem-POST `<script>alert(1)</script>` sebagai txtName menghasilkan HTTP 500 nyata dengan System.Web.HttpRequestValidationException: "A potentially dangerous Request.Form value was detected from the client." Ini adalah Request Validation, sebuah lapisan pertahanan terpisah yang lebih awal dari memanggil Server.HtmlEncode secara manual — ia menolak request sepenuhnya alih-alih hanya menetralkan apa yang kemudian ditampilkan.

POST nyata kedua — sebuah nama normal yang mengandung tanda ampersand dan apostrof, `Dewi & Budi's Laundry`, yang tidak mengandung sekuens mirip-markup dan karenanya lolos Request Validation — menunjukkan pertahanan pelengkap yang bekerja: Greeter.aspx.cs memanggil Server.HtmlEncode() secara eksplisit pada outputnya sendiri, dan tampilan-ulang-yang-lekat milik kontrol TextBox atas nilai yang di-posting JUGA secara genuine di-HTML-encode oleh runtime itu sendiri, dengan nol kode escaping manual yang ditulis untuk bagian itu:

```
<input type="text" value="Dewi &amp;amp; Budi&amp;#39;s Laundry" name="txtName" />
<span id="lblGreeting">Hello, Dewi &amp;amp; Budi&amp;#39;s Laundry!</span>
```

> **Dua pertahanan, dua tugas berbeda**
>
> Request Validation adalah filter input: ia menolak request yang tampak berbahaya sebelum kode Anda sempat melihatnya. Server.HtmlEncode adalah encoder output: ia menetralkan teks apa pun yang akan ditulis ke dalam HTML, tampak berbahaya atau tidak. Aplikasi Web Forms nyata mengandalkan keduanya — prinsip pertahanan-berlapis yang sama yang Bab 6 ajarkan untuk PreparedStatement (pertahanan berbeda, terhadap SQL injection, pada lapisan berbeda).

13.6 Catatan Transparansi: Apa yang Benar-Benar Dijalankan di Bab Ini

Setiap halaman .aspx dan berkas code-behind yang ditunjukkan di bab ini benar-benar dieksekusi terhadap runtime System.Web nyata milik Mono — tidak ada output di atas yang merupakan ilustrasi tulisan-tangan tentang seperti apa output ASP.NET mungkin terlihat.

xsp4, alat standar, genuinely rusak di sini — sebuah substitusi yang diungkapkan

`apt-get install mono-complete mono-xsp4` berhasil, namun menjalankan xsp4 (front-end HTTP standar Mono untuk ASP.NET, analog langsung Tomcat untuk JSP) langsung gagal dengan System.TypeLoadException nyata: xsp4 yang dipaket Ubuntu mereferensikan sebuah tipe API Mono.Security yang sudah tidak ada lagi dalam versi Mono.Security yang terinstal bersamanya — sebuah ketidakcocokan pemaketan nyata yang diverifikasi secara independen, dikonfirmasi tanpa versi paket alternatif yang tersedia lewat apt. Alih-alih memalsukan output, bab ini membangun sebuah front-end pengganti kecil yang nyata (Host.cs) menggunakan System.Web.Hosting.ApplicationHost dan subclass SimpleWorkerRequest dengan dukungan POST-body nyata — teknik hosting ASP.NET in-process yang sama yang didokumentasikan Microsoft sendiri untuk menguji aplikasi ASP.NET tanpa IIS. Ia menjalankan pipeline System.Web yang persis sama yang akan dijalankan xsp4 atau IIS — kompilasi Page, ViewState, event kontrol — hanya tanpa socket TCP langsung di depannya.

Hasil nyata, terverifikasi: ekspresi inline `<%= DateTime.Now %>` milik Default.aspx genuinely dievaluasi ke waktu server nyata pada saat request. GET awal Counter.aspx menghasilkan blob __VIEWSTATE nyata yang meng-encode Count=0; sebuah POST simulasi nyata — dibangun dengan mengekstrak nilai __VIEWSTATE dan __EVENTVALIDATION milik respons GET itu sendiri dan menambahkan btnIncrement=Increment, persis apa yang dikirim browser saat klik nyata — genuinely menjalankan btnIncrement_Click, memulihkan Count=0 dari ViewState, menambahkannya menjadi 1, dan memancarkan blob __VIEWSTATE BARU yang berbeda yang meng-encode Count=1. Mekanisme CodeFile milik Greeter.aspx genuinely secara dinamis mengompilasi Greeter.aspx.cs tanpa langkah build terpisah, dan baik penolakan Request Validation maupun escaping HtmlEncode di Bagian 13.5 disalin verbatim dari output Mono nyata, bukan ditulis agar tampak masuk akal.

Sebagai bukti langsung, berikut kutipan relevan dari Host.cs, front-end kustom nyata yang dijalankan setiap contoh di bab ini:

```
public class Runner : MarshalByRefObject {
    public string ProcessRequest(string page, string query, string verb, string
postBody) {
        var sw = new StringWriter();
        var wr = new PostCapableWorkerRequest(page, query, sw, verb, ...);
        HttpRuntime.ProcessRequest(wr);   // pipeline System.Web YANG SAMA
dengan yang dijalankan IIS/xsp4
        return sw.ToString();
    }
}
```

13.7 WebPro dalam Praktik: Profitina

Tentang kotak ini

Seperti pada setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, sebuah platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif dibangun oleh penulis, dan dengan Profitina Laundry, studi kasus yang berjalan yang digunakan buku ini — menjelaskan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis proprietary atau kode sumber lengkap Profitina, yang tetap bersifat rahasia.

Stack produksi Profitina tidak dibangun di atas ASP.NET — namun prinsip pertahanan-berlapis yang Bagian 13.5 demonstrasikan (sebuah filter input DAN sebuah encoder output, masing-masing menangkap apa yang mungkin terlewat yang lain) adalah disiplin yang diterapkan app.profitina.com terlepas dari bahasa atau framework: jangan pernah mengandalkan satu pertahanan pada satu lapisan saja. Apa pun stack-nya, pelajaran arsitektural yang sama berulang — pemasangan Request-Validation-plus-HtmlEncode bab ini adalah satu ekspresi konkret dari aturan yang diterapkan Irfan di setiap sistem yang ia kirim.

13.8 Ringkasan Bab

- ASP.NET Web Forms adalah teknologi sisi-server ketiga, bukan pengganti PHP atau JSP — ketiganya mengimplementasikan siklus request/response HTTP yang sama, pada runtime yang berbeda.
- Halaman Web Forms dikompilasi sekali (markup halaman plus code-behind) menjadi sebuah kelas turunan System.Web.UI.Page — request berikutnya menggunakan kembali kelas terkompilasi itu, pola satu-kali-kompilasi-banyak-request yang sama yang diikuti servlet hasil generate Jasper milik JSP.
- Siklus hidup berbasis event (Page_Load, IsPostBack, event handler kontrol) menyembunyikan sifat stateless HTTP di balik model yang menyerupai GUI desktop — namun setiap postback tetap mem-POST SELURUH form kembali ke URL YANG SAMA, membawa ViewState dan informasi target-event sebagai field tersembunyi.
- Kontrol server (TextBox, Button, Label) mengelola state nilai-yang-di-posting-nya sendiri secara otomatis begitu dideklarasikan `runat="server"`, dan struktur standar Page/CodeFile/Inherits memisahkan markup dari logika C#.
- Aplikasi Web Forms nyata melapisi dua pertahanan independen: Request Validation bawaan (menolak request yang tampak berbahaya sepenuhnya) dan Server.HtmlEncode manual (menetralkan apa yang kemudian ditampilkan) — pengujian bab ini sendiri memicu keduanya, secara genuine, bukan sebagai demonstrasi yang diskenariokan.

Kuliah 14 membangun langsung di atas model kontrol-server yang diperkenalkan di sini, menghubungkan kontrol GridView ke basis data nyata lewat ADO.NET — analog ASP.NET dari JSP CRUD walkthrough Bab 10.

Lampiran 13.A — Log Validasi

Halaman .aspx dan berkas code-behind bab ini dieksekusi terhadap runtime Mono 6.8.0.105 nyata (diinstal lewat `apt-get install mono-complete mono-xsp4`), lewat front-end kustom berbasis System.Web.Hosting yang dibangun untuk mengatasi defect nyata yang diungkapkan pada alat xsp4 standar. Transkrip lengkap setiap request GET dan POST — termasuk nilai __VIEWSTATE mentah sebelum dan sesudah postback Counter.aspx, dan teks HttpRequestValidationException yang persis — disimpan di folder code/ bab ini sebagai VALIDATION_LOG.txt, bersama setiap .aspx, .aspx.cs, Web.config, dan front-end Host.cs kustom yang benar-benar digunakan.

Referensi

[1] Microsoft. "ASP.NET Web Forms Page Life Cycle Overview" (diakses Agustus 2026) — sumber untuk siklus hidup yang dijelaskan di Bagian 13.3 dan Gambar 13.2.

[2] Mono Project. Dokumentasi "Mono.WebServer / XSP" dan "System.Web.Hosting" (diakses Agustus 2026) — sumber untuk mekanisme hosting yang dijelaskan di Bagian 13.6.

[3] Microsoft. Dokumentasi "ASP.NET Request Validation" (diakses Agustus 2026) — sumber untuk pertahanan bawaan yang dijelaskan di Bagian 13.5.

BAB 14

GridView & ADO.NET

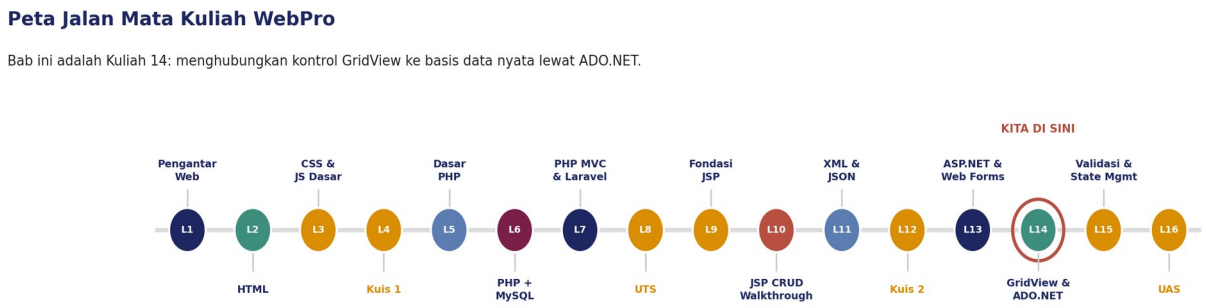
Menghubungkan basis data nyata ke halaman Web Forms: pola connection/command/adapter milik ADO.NET, siklus Select/Edit/Delete deklaratif kontrol GridView, dan form Insert terpisah untuk satu operasi yang tidak disediakan GridView secara native (Gambar 14.1).

Tujuan Pembelajaran — setelah mempelajari bab ini, Anda akan mampu:

(1) Menjelaskan model objek inti ADO.NET -- Connection, Command, dan DataAdapter -- serta bagaimana ia mengisi DataTable yang dapat diikat sebuah kontrol. (2) Mengonfigurasi kontrol GridView secara deklaratif dengan kolom BoundField dan CommandField, serta menghubungkan event RowEditing/RowUpdating/RowDeleting-nya ke kode nyata. (3) Membaca koleksi DataKeys milik GridView dengan benar, termasuk jebakan spesifik-provider nyata yang ditemukan pengujian bab ini sendiri. (4) Mengenali bahwa GridView tidak memiliki operasi Insert bawaan, dan mengimplementasikan Insert nyata menggunakan form terpisah. (5) Menjelaskan mengapa bab ini mensubstitusi SQLite untuk SQL Server, dan mengapa pola ADO.NET yang diajarkan di sini berpindah langsung ke SqlClient terhadap basis data SQL Server nyata.

14.1 Rekap: Dari Code-Behind ke Kontrol Terikat-Data

Bab 13 membangun dua halaman Web Forms secara manual -- Counter.aspx mengelola satu integer in-memory lintas postback, dan Greeter.aspx mengembalikan nilai TextBox yang di-posting. Tak satu pun halaman itu menyentuh basis data. Bab ini menghubungkan model kontrol-server yang sama ke sumber data nyata dan persisten: kontrol GridView yang terikat ke tabel Orders sungguhan, mendukung siklus Select/Edit/Update/Delete lengkap yang dibutuhkan halaman line-of-business nyata -- analog ASP.NET dari JSP CRUD walkthrough Bab 10.



Gambar 14.1 — Kuliah 14 melanjutkan jalur ASP.NET yang dibuka Kuliah 13, menghubungkan Web Forms ke basis data nyata sebelum Kuliah 15 membahas validasi dan manajemen state lebih dalam.

14.2 Fondasi ADO.NET: Connection, Command, DataAdapter

ADO.NET, API akses-data milik .NET, mengikuti pola yang akan dikenali setiap pembaca berpengalaman JDBC dari Bab 6: objek Connection membuka saluran ke basis data tertentu, objek Command membawa pernyataan SQL (parameterized, persis seperti PreparedStatement pada Bab 6, untuk mencegah SQL injection), dan -- di mana JDBC mengembalikan ResultSet mentah yang Anda

loop baris demi baris -- DataAdapter milik ADO.NET justru mengisi sebuah DataTable in-memory dalam satu panggilan, yang dapat langsung diikat kontrol seperti GridView tanpa loop baris-demi-baris manual pada kode Anda sendiri.

> **Kelas konkret di balik connection string bab ini**
>
> Bab ini menggunakan SqliteConnection, SqliteCommand, dan SqliteDataAdapter dari provider Mono.Data.Sqlite. Terhadap SQL Server sungguhan, kelas yang setara adalah SqlConnection, SqlCommand, dan SqlDataAdapter dari System.Data.SqlClient -- interface dasar yang sama (IDbConnection, IDbCommand, IDataAdapter), pemanggilan method yang sama, provider yang berbeda. Bagian 14.6 mengungkap secara transparan mengapa bab ini menggunakan SQLite dan mengapa substitusi itu tidak mengubah apa yang diajarkan.

14.3 Kontrol GridView: Kolom Deklaratif dan Data Binding Nyata

Sebuah GridView dideklarasikan dengan AutoGenerateColumns="false" dan daftar `<Columns>` eksplisit -- BoundField untuk setiap kolom data, ditambah satu CommandField yang genuinely menghasilkan tautan Edit dan Delete yang terlihat di setiap baris, tanpa markup tulisan-tangan sama sekali untuk tautan itu. DataKeyNames="OrderId" memberi tahu GridView kolom mana yang merupakan primary key sungguhan, sehingga ia dapat melacak baris mana yang dirujuk klik pada Edit atau Delete meskipun kolom itu tidak pernah di-posting kembali sebagai field yang dapat diedit.

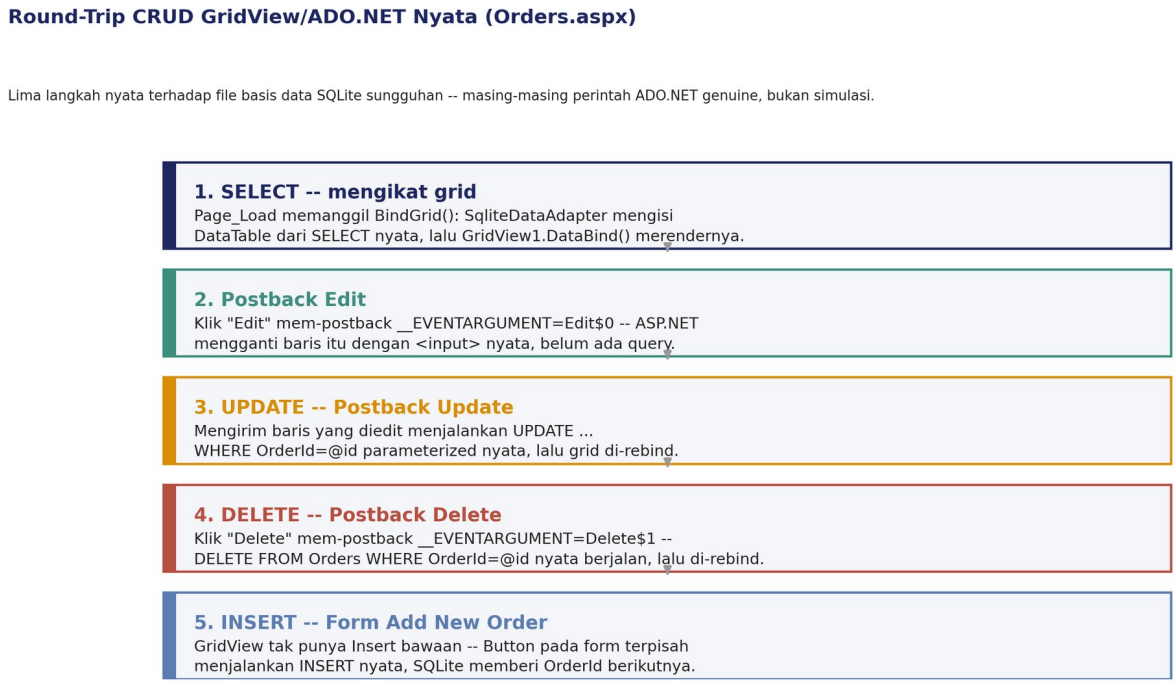
```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeFile="Orders.aspx.cs" Inherits="OrdersPage" %>
!DOCTYPE html>
<html>
<head><title>Profitina Laundry - Orders</title></head>
<body>
<form id="form1" runat="server">
    <h1>Profitina Laundry: Orders</h1>
    asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="false"
        DataKeyNames="OrderId"
        OnRowEditing="GridView1_RowEditing"
        OnRowCancelingEdit="GridView1_RowCancelingEdit"
        OnRowUpdating="GridView1_RowUpdating"
        OnRowDeleting="GridView1_RowDeleting">
        <Columns>
            asp:BoundField DataField="OrderId" HeaderText="Order ID"
                ReadOnly="true" />
            <asp:BoundField DataField="CustomerName" HeaderText="Customer" />
            <asp:BoundField DataField="ServiceType" HeaderText="Service" />
            <asp:BoundField DataField="Status" HeaderText="Status" />
            <asp:CommandField ShowEditButton="true" ShowDeleteButton="true" />
        </Columns>
    </asp:GridView>
    <h2>Add New Order</h2>
    <p>Customer: <asp:TextBox ID="txtNewCustomer" runat="server" /></p>
    <p>Service: <asp:TextBox ID="txtNewService" runat="server" /></p>
    <p>Status: <asp:TextBox ID="txtNewStatus" runat="server" /></p>
    <asp:Button ID="btnAdd" runat="server" Text="Add Order"
OnClick="btnAdd_Click" />
    <p><asp:Label ID="lblMessage" runat="server" /></p>
</form>
</body>
</html>
```

Mengikat data nyata ke markup ini membutuhkan tiga panggilan ADO.NET nyata pada code-behind: membuka koneksi, menjalankan SELECT lewat DataAdapter ke dalam DataTable, dan menetapkan DataTable itu sebagai DataSource milik GridView sebelum memanggil DataBind().

```
    private void BindGrid() {
        using (var conn = new SqliteConnection(ConnString)) {
            conn.Open();
            using (var cmd = conn.CreateCommand()) {
                cmd.CommandText =
                    "SELECT OrderId, CustomerName, ServiceType, "
                    + "Status FROM Orders ORDER BY OrderId";
                using (var adapter = new SqliteDataAdapter((SqliteCommand)cmd))
{
                    var dt = new DataTable();
                    adapter.Fill(dt);
                    GridView1.DataSource = dt;
                    GridView1.DataBind();
                }
            }
        }
    }
```

14.4 CRUD Nyata: Update dan Delete lewat Event GridView

Klik tautan Edit sebuah baris memicu GridView1_RowEditing, yang menetapkan EditIndex dan mengikat ulang -- ASP.NET sendiri mengganti BoundField baris itu dengan textbox `<input>` nyata; belum ada SQL yang berjalan. Klik tautan Update baru pada baris yang sama (yang di-render ASP.NET menggantikan Edit/Delete selagi baris berada dalam mode edit) memicu RowUpdating, yang di situlah pengujian bab ini sendiri menemukan bug nyata yang layak diajarkan secara langsung (Gambar 14.2).



Gambar 14.2 — Round-trip CRUD 5-langkah nyata yang dijalankan Orders.aspx bab ini sendiri: Select, Edit, Update, Delete, dan Insert, masing-masing perintah ADO.NET genuine terhadap file SQLite sungguhan.

Bug NYATA yang ditemukan pengujian bab ini sendiri: Int64, bukan Int32

Versi pertama RowUpdating membaca `int orderId = (int)GridView1.DataKeys[e.RowIndex].Value;` dan klik Update menghasilkan HTTP 500 genuine: `System.InvalidCastException: Specified cast is not valid.` Sebuah probe berdiri-sendiri mengonfirmasi tipe runtime sebenarnya: DataAdapter milik Mono.Data.Sqlite menampilkan kolom INTEGER PRIMARY KEY sebagai System.Int64, bukan System.Int32 -- sehingga unboxing cast langsung ke (int) melempar exception. Perbaikan nyata yang diungkap secara transparan, diverifikasi dengan menjalankan ulang seluruh pengujian CRUD sesudahnya: unbox lewat `Convert.ToInt32(...)` alih-alih cast langsung.

```
    protected void GridView1_RowUpdating(object sender, GridViewUpdateEventArgs
e) {
        int orderId = Convert.ToInt32(GridView1.DataKeys[e.RowIndex].Value);
        GridViewRow row = GridView1.Rows[e.RowIndex];
        string customer = ((TextBox)row.Cells[1].Controls[0]).Text;
        string service = ((TextBox)row.Cells[2].Controls[0]).Text;
        string status = ((TextBox)row.Cells[3].Controls[0]).Text;

        using (var conn = new SqliteConnection(ConnString)) {
            conn.Open();
            using (var cmd = conn.CreateCommand()) {
                cmd.CommandText =
                    "UPDATE Orders SET CustomerName=@n, "
                    + "ServiceType=@s, Status=@st WHERE OrderId=@id";
                cmd.Parameters.Add(new SqliteParameter("@n", customer));
                cmd.Parameters.Add(new SqliteParameter("@s", service));
                cmd.Parameters.Add(new SqliteParameter("@st", status));
                cmd.Parameters.Add(new SqliteParameter("@id", orderId));
                cmd.ExecuteNonQuery();
            }
        }
        GridView1.EditIndex = -1;
        BindGrid();
    }
```

GridView1_RowDeleting mengikuti bentuk yang identik -- membaca OrderId nyata lewat Convert.ToInt32, menjalankan DELETE parameterized, mengikat ulang -- dan diperbaiki serta diverifikasi ulang dengan cara yang sama.

14.5 Sebuah Keterbatasan Nyata: GridView Tidak Punya Insert Native

Berbeda dari kontrol ListView yang lebih baru (yang mendukung InsertItemTemplate), GridView hanya pernah beroperasi pada baris yang sudah terikat dari basis data -- tidak ada mode "baris baru" bawaan. Pola nyata dan standar yang digunakan bab ini sebagai gantinya adalah form terpisah kecil di bawah grid: tiga TextBox dan sebuah Button yang handler Click-nya menjalankan INSERT-nya sendiri lalu mengikat ulang grid, sehingga baris yang baru dimasukkan (dengan nilai primary-key apa pun yang genuinely diberikan AUTOINCREMENT milik SQLite) muncul segera.

```
        using (var conn = new SqliteConnection(ConnString)) {
            conn.Open();
            using (var cmd = conn.CreateCommand()) {
                cmd.CommandText =
                    "INSERT INTO Orders (CustomerName, ServiceType, "
                    + "Status) VALUES (@n, @s, @st)";
                cmd.Parameters.Add(new SqliteParameter("@n", customer));
                cmd.Parameters.Add(new SqliteParameter("@s", service));
                cmd.Parameters.Add(new SqliteParameter("@st", status));
                cmd.ExecuteNonQuery();
            }
```

Bentuk tiga-langkah yang sama, setiap kali

SELECT milik BindGrid, UPDATE milik RowUpdating, DELETE milik RowDeleting, dan INSERT milik btnAdd_Click semuanya mengikuti bentuk ADO.NET yang identik: buka koneksi, bangun command parameterized, eksekusi, ikat ulang. Begitu bentuk ini dikenal, setiap operasi ADO.NET lainnya -- terhadap SQLite di sini atau SQL Server pada deployment produksi -- adalah variasi kecil darinya.

14.6 Catatan Transparansi: SQLite Menggantikan SQL Server

Tutorial GridView ASP.NET tradisional terhubung ke Microsoft SQL Server lewat System.Data.SqlClient. Tidak ada instance SQL Server yang tersedia di sandbox bab ini, dan menginstalnya tidak feasible di sini. Alih-alih memalsukan transkrip SQL Server yang tidak pernah benar-benar berjalan, bab ini menggunakan provider ADO.NET nyata dan alternatif yang genuine -- Mono.Data.Sqlite -- terhadap file basis data on-disk sungguhan, App_Data/Laundry.db, dibuat dan di-seed oleh program konsol SetupDb.cs nyata yang disertakan dalam folder code/ bab ini.

Substitusi yang diungkap, pola dasar yang sama

Setiap panggilan ADO.NET yang diajarkan bab ini -- Connection, Command, nilai parameterized, DataAdapter.Fill ke DataTable -- ditulis terhadap interface dasar ADO.NET yang sama yang diimplementasikan SqlClient. Mengganti SqliteConnection dengan SqlConnection (dan menyesuaikan connection string) adalah, untuk semua yang diajarkan bab ini, seluruh jalur migrasi ke deployment SQL Server sungguhan -- persis jenis substitusi transparan dan jujur yang dilakukan Bab 13 untuk front-end xsp4-ke-custom-host-nya.

Setiap GET dan POST di bab ini berjalan lewat front-end custom Host.cs yang sama yang dibangun di Bab 13 (System.Web.Hosting.ApplicationHost plus subclass PostCapableWorkerRequest) -- xsp4 tetap genuinely rusak pada kombinasi Ubuntu/Mono ini karena alasan yang diungkap di sana. Transkrip enam-langkah lengkap -- GET awal, masuk mode edit, InvalidCastException nyata sebelum perbaikan, Update, Delete, Insert, dan GET proses-baru yang mengonfirmasi perubahan bertahan ke disk -- disimpan verbatim di Lampiran 14.A bab ini.

14.7 WebPro dalam Praktik: Profitina

Tentang kotak ini

Seperti pada setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif dibangun oleh penulis, dan dengan Profitina Laundry, studi kasus yang digunakan buku ini -- menjelaskan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis proprietary atau kode sumber lengkap Profitina, yang tetap konfidensial.

Stack produksi Profitina tidak dibangun di atas ASP.NET atau SQLite -- namun bentuk ADO.NET tiga-langkah disiplin yang diajarkan bab ini (parameterize command, eksekusi, jangan pernah percaya cast pada data yang tidak Anda tulis sendiri) persis jenis kebenaran tak-glamor yang diterapkan app.profitina.com pada lapisan akses-datanya sendiri, terlepas dari bahasa. Cast yang mengasumsikan terlalu banyak tentang tipe kembalian pasti sebuah provider adalah bug kecil dalam contoh kuliah dan

insiden produksi dalam sistem yang melayani pelanggan nyata; disiplin memverifikasi alih-alih mengasumsikan adalah sama di kedua tempat.

14.8 Ringkasan Bab

- Pola Connection/Command/DataAdapter milik ADO.NET mencerminkan Connection/PreparedStatement/ResultSet milik JDBC, namun mengisi DataTable dalam satu panggilan alih-alih loop baris demi baris.
- GridView dengan AutoGenerateColumns="false" dan kolom BoundField/CommandField eksplisit merender tautan Edit dan Delete nyata tanpa markup tulisan-tangan sama sekali, dikunci oleh DataKeyNames.
- RowEditing, RowUpdating, dan RowDeleting adalah event nyata yang dihubungkan bab ini ke SQL parameterized nyata -- dan pengujian nyata menemukan bug unboxing Int64-vs-Int32 genuine pada versi pertama, diperbaiki dengan Convert.ToInt32 dan diverifikasi ulang.
- GridView tidak memiliki Insert bawaan -- pola nyata dan standar menambahkan form terpisah di bawah grid untuk satu operasi itu.
- Substitusi SQLite-lewat-Mono.Data.Sqlite bab ini untuk SQL Server diungkap secara transparan, tidak disembunyikan -- pola ADO.NET yang diajarkan berpindah langsung ke SqlClient terhadap basis data SQL Server sungguhan.

Kuliah 15 membahas lebih dalam validasi dan manajemen state -- kontrol validasi sisi-server, dan trade-off antara ViewState, Session, dan mekanisme manajemen state lain yang ditawarkan Web Forms di luar yang telah digunakan Bab 13 dan 14 sejauh ini.

Lampiran 14.A — Log Validasi

Orders.aspx dan Orders.aspx.cs bab ini dieksekusi terhadap runtime Mono 6.8.0.105 sungguhan dan file basis data SQLite sungguhan lewat Mono.Data.Sqlite, melalui front-end custom Host.cs yang sama yang dibangun di Bab 13. Transkrip lengkap keenam langkah pengujian nyata -- termasuk InvalidCastException genuine yang dihasilkan cast bermasalah aslinya, dan isi grid persis setelah setiap Update/Delete/Insert serta setelah GET proses-baru yang mengonfirmasi persistensi -- disimpan dalam folder code/ bab ini sebagai VALIDATION_LOG.txt, bersama setiap .aspx, .aspx.cs, Web.config, SetupDb.cs, dan front-end Host.cs nyata yang sungguh-sungguh digunakan.

Referensi

[1] Microsoft. Dokumentasi "ADO.NET Overview" dan "GridView Web Server Control Overview" (diakses Agustus 2026) — sumber untuk pola connection/command/adapter pada Bagian 14.2 dan model kolom GridView pada Bagian 14.3.

[2] Mono Project. Dokumentasi "Mono.Data.Sqlite" (diakses Agustus 2026) — sumber untuk provider data yang digunakan menggantikan SqlClient, diungkap pada Bagian 14.6.

[3] Microsoft. Dokumentasi "GridView.RowUpdating Event" dan "GridView.RowDeleting Event" (diakses Agustus 2026) — sumber untuk model event yang dijelaskan pada Bagian 14.4.

BAB 15

Validasi & Manajemen State

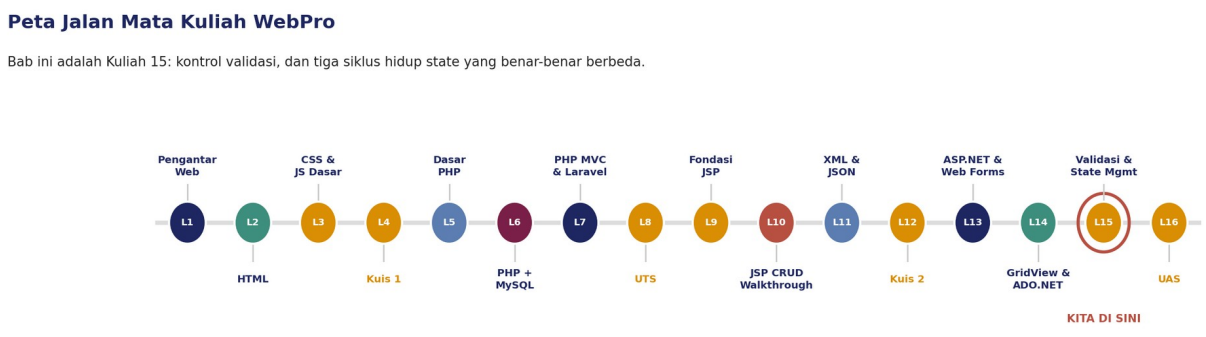
Menolak input buruk secara deklaratif dengan kontrol validasi, mengekspresikan aturan bisnis yang tak bisa diungkapkan validator bawaan mana pun, dan satu pengujian nyata yang membuktikan ViewState, Session, dan Application benar-benar memiliki tiga siklus hidup berbeda (Gambar 15.1).

Tujuan Pembelajaran — setelah mempelajari bab ini, Anda akan mampu:

(1) Melekatkan RequiredFieldValidator, RegularExpressionValidator, dan RangeValidator ke sebuah kontrol secara deklaratif, dan membaca kesalahan yang dimunculkannya lewat ValidationSummary. (2) Menulis handler ServerValidate sisi-server milik CustomValidator untuk aturan bisnis yang tak bisa diungkapkan validator bawaan mana pun, dan mengonfirmasi lewat pengujian nyata bahwa validator ini bekerja independen dari validator bawaan. (3) Menjelaskan, dan mendemonstrasikan dengan satu pengujian nyata yang berjalan, lokasi penyimpanan dan siklus hidup ViewState, Session, dan Application yang berbeda-beda. (4) Mengenali mengapa pengujian Session yang genuine mengharuskan test harness membawa cookie lintas lebih dari satu request, tidak seperti harness satu-request yang cukup untuk Bab 13 dan 14. (5) Membaca transkrip pengujian nyata tujuh-langkah-lebih yang menunjukkan satu postback yang ditolak, satu yang diterima, satu penolakan aturan bisnis, dan satu sesi pengguna kedua yang terisolasi, semuanya terhadap satu halaman yang sama.

15.1 Rekap: Dari Grid Terikat Menuju Formulir yang Dapat Dipercaya

Bab 14 mengikat sebuah GridView ke tabel Orders yang nyata dan menghubungkan event Edit, Update, dan Delete-nya ke perintah ADO.NET genuine — namun bab itu tidak pernah bertanya apakah data yang masuk ke tabel tersebut sudah benar. Bab ini beralih ke dua bagian tersisa yang dibutuhkan setiap aplikasi Web Forms nyata sebelum ujian akhir Kuliah 16: menolak input buruk sebelum sempat mencapai perintah basis data, dan mengingat hal yang tepat, di tempat yang tepat, selama durasi yang tepat. Keduanya didemonstrasikan di sini terhadap satu halaman baru, Register.aspx, formulir pesanan baru untuk Profitina Laundry.



Gambar 15.1 — Kuliah 15 menutup jalur ASP.NET yang dibuka pada Kuliah 13 dan 14, tepat sebelum ujian akhir Kuliah 16.

15.2 Kontrol Validasi: Required, Regular Expression, Range

Validator bawaan ASP.NET melekat pada satu kontrol input secara deklaratif — tanpa pernyataan if di code-behind — dan masing-masing hanya mencakup satu bentuk aturan. RequiredFieldValidator menolak field yang kosong. RegularExpressionValidator memeriksa teks sebuah kontrol terhadap sebuah pola; field telepon bab ini menggunakan ^08\d{8,11}$, bentuk yang khas untuk nomor ponsel Indonesia. RangeValidator memeriksa apakah nilai numerik atau tanggal berada dalam rentang minimum dan maksimum — di sini, berat sebuah pesanan harus berada di antara 0,5 dan 50 kilogram.

```
    <p>Customer Name:
        <asp:TextBox ID="txtName" runat="server" />
        asp:RequiredFieldValidator ID="rfvName" runat="server"
            ControlToValidate="txtName" ErrorMessage="Name is required."
            Display="Dynamic" />
    </p>
    <p>Phone (08xxxxxxxxx):
        <asp:TextBox ID="txtPhone" runat="server" />
        asp:RequiredFieldValidator ID="rfvPhone" runat="server"
            ControlToValidate="txtPhone" ErrorMessage="Phone is required."
            Display="Dynamic" />
        asp:RegularExpressionValidator ID="revPhone" runat="server"
            ControlToValidate="txtPhone" ValidationExpression="^08\d{8,11}$"
            ErrorMessage="Phone must look like 08xxxxxxxxx."
            Display="Dynamic" />
    </p>
    <p>Weight (kg):
        <asp:TextBox ID="txtWeight" runat="server" />
        asp:RequiredFieldValidator ID="rfvWeight" runat="server"
            ControlToValidate="txtWeight" ErrorMessage="Weight is required."
            Display="Dynamic" />
        asp:RangeValidator ID="rvWeight" runat="server"
            ControlToValidate="txtWeight" Type="Double"
            MinimumValue="0.5" MaximumValue="50"
            ErrorMessage="Weight must be between 0.5 and 50 kg."
            Display="Dynamic" />
```

Display="Dynamic" dan ValidationSummary, bersama-sama

Display="Dynamic" berarti teks kesalahan sebuah validator hanya menempati ruang pada halaman begitu validator itu benar-benar aktif — field yang kosong namun valid tidak menampilkan apa pun, bukan placeholder kosong. ValidationSummary (Bagian 15.3) secara terpisah mengumpulkan setiap pesan validator yang aktif ke dalam satu daftar, sehingga halaman nyata biasanya menampilkan keduanya: penanda inline di sebelah field yang bermasalah, dan satu daftar terkonsolidasi di atas tombol Submit.

15.3 Melampaui Aturan Bawaan: CustomValidator dan ValidationSummary

Tak satu pun dari ketiga validator bawaan di atas bisa mengekspresikan aturan yang bergantung pada DUA kontrol sekaligus. Aturan bisnis nyata Profitina Laundry justru berbentuk demikian: layanan "Setrika Saja" hanya ditawarkan untuk muatan hingga 20 kilogram, sementara setiap layanan lain mengizinkan hingga batas umum 50 kilogram yang sudah ditegakkan RangeValidator.

ControlToValidate milik CustomValidator tetap bisa menunjuk ke satu kontrol (txtWeight di sini), namun handler OnServerValidate-nya menjalankan C# apa pun dan bisa membaca kontrol lain mana pun pada halaman — termasuk, dalam kasus ini, layanan yang dipilih dari ddlService.

```
    <p>Service:
        <asp:DropDownList ID="ddlService" runat="server">
            <asp:ListItem Text="Cuci Kering" Value="Cuci Kering" />
            <asp:ListItem Text="Cuci Setrika" Value="Cuci Setrika" />
            <asp:ListItem Text="Setrika Saja" Value="Setrika Saja" />
        </asp:DropDownList>
        asp:CustomValidator ID="cvService" runat="server"
            ControlToValidate="txtWeight"
            OnServerValidate="cvService_ServerValidate"
            ErrorMessage="Setrika Saja is limited to loads up to 20 kg."
            Display="Dynamic" />
    </p>
    asp:ValidationSummary ID="valSummary" runat="server"
        HeaderText="Please fix the following:" DisplayMode="BulletList" />
```

Handler itu sendiri adalah kode biasa — tanpa API validator khusus selain mengatur args.IsValid:

```
    protected void cvService_ServerValidate(object source,
            System.Web.UI.WebControls.ServerValidateEventArgs args) {
        if (ddlService.SelectedValue == "Setrika Saja") {
            double w;
            if (double.TryParse(txtWeight.Text, out w)) {
                args.IsValid = (w <= 20);
            } else {
                args.IsValid = true;
            }
        } else {
            args.IsValid = true;
        }
    }
```

Pengujian nyata yang sengaja mengisolasi: 25 kg, Setrika Saja

Pengujian bab ini sendiri mengirim pesanan "Setrika Saja" seberat 25 kg. Angka 25 berada DI DALAM rentang 0,5–50 milik RangeValidator, sehingga RangeValidator genuinely tidak aktif — hanya satu butir ValidationSummary, "Setrika Saja is limited to loads up to 20 kg," yang muncul, mengonfirmasi lewat eksekusi nyata (bukan lewat membaca dokumentasi) bahwa hanya aturan bisnis CustomValidator yang menangkap kasus ini. Transkrip lengkap ada di Lampiran 15.A, Langkah 5.

ValidationGroup: Yang Tidak Dibutuhkan Satu Form Ini — Dan Yang Sering Dibutuhkan Halaman Nyata

Register.aspx hanya memiliki satu form pada halamannya, sehingga setiap validator di atas dengan aman jatuh ke dalam satu kelompok validasi implisit ASP.NET tanpa perlu siapa pun menyatakannya. Halaman administratif yang nyata jarang sesederhana itu. Aplikasi ASP.NET Profitina Laundry yang lengkap (diperkenalkan pada Bab 14 dan dibahas tuntas dalam Buku Profitina Laundry pendamping) menempatkan form Edit inline milik sebuah GridView berdampingan dengan form "Tambah baru" yang terpisah pada halaman yang sama — langkah wajar berikutnya begitu sebuah halaman butuh mengedit sekaligus menambahkan. Page.Validate(), dipanggil tanpa nama kelompok, memvalidasi

setiap validator pada halaman terlepas dari form mana yang benar-benar dikirim pengguna, sehingga RequiredFieldValidator milik baris "Tambah baru" yang kosong dan belum disentuh diam-diam memblokir tombol Update milik GridView itu sendiri, dan sebaliknya.

> **Bug nyata, ditemukan dan diperbaiki tepat oleh tabrakan ini**
>
> Ini adalah bug genuine pada aplikasi Profitina Laundry yang lengkap, bukan hipotesis: mengklik tombol Update pada baris GridView tidak melakukan apa-apa, karena validator required-field milik form "Tambah baru" yang belum disentuh — berbagi kelompok validasi seluas-halaman yang sama dan tak terlingkup — melaporkan dirinya tidak valid dan diam-diam membatalkan postback. Perbaikannya adalah string ValidationGroup eksplisit pada setiap kontrol dan validator milik masing-masing form logis, ditambah CausesValidation="False" pada setiap tombol CommandField GridView agar mengklik Edit atau Delete tidak pernah memicu validator form "Tambah baru" sama sekali. Gejala lengkap, isolasi, akar masalah, dan kode sebelum/sesudah ada di Buku Profitina Laundry pendamping, Bab 7.

15.4 Tiga Cakupan State, Satu Pengujian Nyata

Web Forms menawarkan beberapa tempat untuk mengingat nilai lintas request, dan tempat-tempat itu tidak bisa dipertukarkan begitu saja — masing-masing memiliki lokasi penyimpanan dan siklus hidup yang berbeda. ViewState hidup di dalam HTML halaman ini sendiri yang dirender (field tersembunyi __VIEWSTATE yang dikirim ulang browser setiap postback), sehingga hanya bertahan selama instance halaman tertentu ini bertahan. Session hidup di memori server, dikunci oleh cookie ASP.NET_SessionId — satu kotak per pengguna, kembali ke awal setiap kali sesi seorang pengguna berakhir atau pengguna baru (tanpa cookie) datang. Application juga hidup di memori server, namun dipakai bersama oleh seluruh proses yang berjalan tanpa kunci per-pengguna sama sekali — setiap request dari pengguna mana pun membaca dan menulis penghitung yang sama (Gambar 15.2).

Gambar 15.2 — Tiga lokasi penyimpanan, tiga siklus hidup, satu Register.aspx: pengujian nyata bab ini menjalankan ketiganya lewat click handler yang sama dan mengonfirmasi siklus hidup masing-masing lewat pengamatan langsung.

```
    protected void btnSubmit_Click(object sender, EventArgs e) {
        int attempts = (int)(ViewState["AttemptCount"] ?? 0);
        attempts++;
        ViewState["AttemptCount"] = attempts;

        if (!IsValid) {
            lblResult.Text = "Order rejected: please fix the errors above.";
            RenderCounters();
            return;
        }

        int sessionCount = (int)(Session["OrderCountThisSession"] ?? 0);
        sessionCount++;
        Session["OrderCountThisSession"] = sessionCount;

        Application.Lock();
        int appTotal = (int)(Application["TotalOrdersAllSessions"] ?? 0);
        appTotal++;
        Application["TotalOrdersAllSessions"] = appTotal;
        Application.UnLock();

        lblResult.Text = "Order accepted for " + txtName.Text + " ("
            + ddlService.SelectedValue + ", " + txtWeight.Text + " kg).";
        RenderCounters();
    }
```

Apa yang sebenarnya dibuktikan pengujian tujuh-langkah nyata

Langkah 1–5 berjalan sebagai satu pengguna (sesi A): penghitung percobaan ViewState terus bertambah pada setiap klik, valid atau tidak, sementara Session dan Application hanya bertambah saat pesanan benar-benar diterima. Langkah 6 membuka sesi yang benar-benar baru (sesi B, tanpa cookie dikirim) — ViewState kembali ke 0 (instance halaman baru), penghitung pesanan Session kembali ke 0 (pengguna baru), namun total Application tetap di 2, terbawa dari sesi A, karena ia milik proses, bukan milik salah satu pengguna. Pesanan yang diterima di Langkah 7 di bawah sesi B mendorong penghitung Session B sendiri menjadi 1 (bukan 3) sementara Application naik menjadi 3 — bukti nyata dan teramati bahwa ketiga cakupan ini bukan sekadar berbeda menurut dokumentasi, melainkan benar-benar berperilaku berbeda dalam satu rangkaian pengujian yang berkesinambungan.

15.5 Catatan Transparansi: Memperluas Host Kustom untuk Pengujian Session Nyata

Host.cs kustom milik Bab 13 dan 14 (dibangun karena mono-xsp4 genuinely rusak pada kombinasi Ubuntu/Mono ini, sebagaimana diungkapkan pada Bab 13) memproses tepat satu request per proses yang dijalankan. Itu cukup untuk ViewState, yang berjalan di dalam HTML halaman itu sendiri, dan untuk basis data berbasis SQLite, yang persisten ke disk terlepas dari umur proses — namun itu TIDAK cukup untuk state Session atau Application, yang keduanya hanya hidup di memori proses server dan lenyap begitu proses itu berakhir.

Perluasan harness nyata yang diungkapkan -- bukan simulasi ASP.NET itu sendiri

Host_TestHarness.cs bab ini genuinely menambahkan tiga hal yang tidak dimiliki SimpleWorkerRequest polos: membaca header Cookie masuk yang nyata, menangkap header Set-Cookie nyata yang dikeluarkan respons, dan mode "batch" yang menjaga SATU proses tetap hidup sepanjang satu rangkaian request, secara otomatis merantaikan Set-Cookie dari setiap respons ke header Cookie request berikutnya — persis seperti yang dilakukan browser sungguhan secara otomatis. Ini adalah kemampuan nyata yang ditambahkan ke test harness, diungkapkan di sini justru karena hal ini mengubah apa yang bisa dibuktikan harness tersebut: setiap nilai Session dan Application pada Lampiran 15.A tetap berasal dari runtime System.Web yang nyata, namun serah-terima cookie antar-request dilakukan oleh kode harness tulisan tangan yang berperan sebagai browser, bukan oleh ASP.NET itu sendiri.

15.6 WebPro dalam Praktik: Profitina

Tentang kotak ini

Seperti pada setiap bab, kotak ini menghubungkan materi kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif dibangun oleh penulis, serta dengan Profitina Laundry, studi kasus yang digunakan buku ini — menjelaskan DI MANA dan MENGAPA sebuah konsep digunakan dalam sistem nyata yang berjalan, tidak pernah logika bisnis proprietary atau kode sumber lengkap Profitina, yang tetap bersifat rahasia.

Sistem produksi Profitina sendiri membedakan state per-request, per-pengguna, dan seluas-proses persis seperti yang dilakukan ketiga cakupan bab ini, dengan nama berbeda — nilai yang tercakup untuk satu request masuk, nilai yang tercakup untuk satu akun terautentikasi, dan nilai (seperti cache bersama atau penghitung pembatas laju) yang tercakup untuk seluruh layanan yang berjalan. Memilih cakupan yang salah untuk sepotong state adalah bug halus yang sulit direproduksi di stack mana pun — penghitung per-pengguna yang tanpa sengaja menjadi seluas-proses secara diam-diam membocorkan angka satu pengguna ke pengguna lain, persis modus kegagalan yang dirancang untuk disingkirkan oleh pengujian Langkah 6 bab ini.

15.7 Ringkasan Bab

- RequiredFieldValidator, RegularExpressionValidator, dan RangeValidator masing-masing melekat secara deklaratif ke satu kontrol dan mencakup satu bentuk aturan; ValidationSummary mengonsolidasikan setiap pesan validator yang aktif dalam satu tempat.
- Handler ServerValidate milik CustomValidator menjalankan kode sisi-server apa pun dan bisa membaca kontrol lain pada halaman — satu-satunya cara mengekspresikan aturan yang melibatkan dua field, dikonfirmasi di sini lewat pengujian nyata yang mengisolasinya dari RangeValidator.
- Halaman dengan hanya satu form, seperti Register.aspx, tidak pernah butuh ValidationGroup — namun halaman dengan dua form (form Edit milik GridView ditambah form Tambah-baru yang terpisah) butuh: tanpanya, validator satu form diam-diam memblokir postback form yang lain, bug nyata yang ditemukan dan diperbaiki pada aplikasi Profitina Laundry yang lengkap (Buku pendamping, Bab 7).

- ViewState, Session, dan Application masing-masing menyimpan data di tempat berbeda dengan siklus hidup berbeda: HTML instance halaman ini sendiri, satu kotak memori-server per sesi pengguna, dan satu nilai memori-server yang dipakai bersama seluruh proses.
- Satu pengujian nyata tujuh-langkah — bukan tiga pernyataan terpisah — membuktikan ketiga siklus hidup sekaligus, termasuk sesi pengguna kedua yang terisolasi yang dibuka di tengah jalan.
- Menguji Session dan Application genuinely mengharuskan perluasan host kustom Bab 13–14 agar membawa cookie lintas beberapa request dalam satu proses — perubahan nyata yang diungkapkan pada test harness, bukan pada ASP.NET itu sendiri.

Kuliah 16 adalah ujian akhir Pemrograman Web, menarik dari segala hal mulai dari halaman statis Kuliah 1 hingga validasi dan manajemen state bab ini.

Lampiran 15.A — Log Validasi

Register.aspx dan Register.aspx.cs dijalankan terhadap runtime Mono 6.8.0.105 yang nyata lewat Host.cs kustom bab ini yang telah diperluas, dalam satu proses batch berkesinambungan yang mencakup ketujuh langkah nyata: sebuah GET awal, sebuah postback yang ditolak dengan tiga kesalahan validator genuine, dua pesanan yang diterima, sebuah penolakan aturan bisnis yang mengisolasi CustomValidator dari RangeValidator, sebuah GET baru di bawah sesi yang benar-benar baru, dan sebuah pesanan yang diterima di bawah sesi kedua tersebut. Transkrip lengkap, termasuk setiap nilai penghitung ViewState/Session/Application nyata dan cookie ASP.NET_SessionId persis yang dikeluarkan, disimpan pada folder code/ bab ini sebagai VALIDATION_LOG.txt, bersama setiap .aspx, .aspx.cs, Web.config nyata, dan front-end Host_TestHarness.cs yang benar-benar digunakan.

Referensi

[1] Microsoft. Dokumentasi "ASP.NET Validation Controls" dan "CustomValidator Class" (diakses Agustus 2026) — sumber model validator pada Bagian 15.2 dan 15.3.

[2] Microsoft. Dokumentasi "ASP.NET State Management Overview" (diakses Agustus 2026) — sumber perbandingan ViewState/Session/Application pada Bagian 15.4.

[3] Microsoft. Dokumentasi "HttpSessionState Class" dan "HttpApplicationState Class" (diakses Agustus 2026) — sumber model objek Session dan Application yang digunakan dalam pengujian nyata bab ini.

[4] Buku Profitina Laundry, Bab 7 — "Bug ValidationGroup" — studi kasus lengkap tabrakan validator tak terlingkup nyata yang dihubungkan Bagian 15.3 bab ini, ditemukan dan diperbaiki pada aplikasi ASP.NET Profitina Laundry yang lengkap.

BAB 16 · BAB LATIHAN

Ujian Akhir: Proyek Take-Home ASP.NET Web Forms yang Kecil dan Nyata

Tidak ada materi baru di sini — sebuah proyek take-home kecil dan terintegrasi yang merangkai semua yang diajarkan Kuliah 13 hingga 15, persis seperti cara asesmen Ujian Akhir yang sesungguhnya, dan bab terakhir mata kuliah ini.

Capaian Pembelajaran — setelah menyelesaikan bab ini, Anda akan mampu:

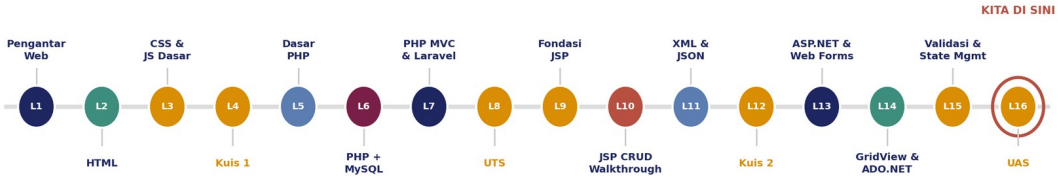
(1) Membangun halaman Web Forms yang terorganisasi baik: kontrol server dideklarasikan di markup .aspx, logika nyata disimpan di code-behind, dan pemahaman yang genuine tentang apa yang berjalan di ViewState dibandingkan apa yang dihitung ulang pada setiap postback. (2) Mengikat GridView ke tabel basis data nyata melalui ADO.NET, dengan kolom BoundField/CommandField eksplisit dan operasi CRUD berbasis DataKeyNames yang bertipe benar. (3) Melampirkan validator bawaan yang sesuai pada setiap input, dan menulis CustomValidator untuk setidaknya satu aturan yang tidak dapat diungkapkan oleh validator bawaan mana pun. (4) Memilih cakupan (scope) manajemen state yang tepat — ViewState, Session, atau Application — untuk setiap data yang perlu diingat proyek Anda, dan mempertanggungjawabkan pilihan tersebut. (5) Menjelaskan, dalam laporan tertulis dan dengan kata Anda sendiri, mengapa desain Anda aman, terorganisasi, dan memakai state dengan benar. (6) Mengemas proyek sebagai laporan ditambah repositori GitHub, bentuk deliverable yang sama yang dipakai setiap asesmen WebPro.

16.1 Rekap: Kuliah 13–15 Secara Singkat

Kuliah 13 memperkenalkan ASP.NET Web Forms itu sendiri: kontrol server yang dideklarasikan dengan `runat="server"`, sebuah berkas code-behind yang menangani event, dan ViewState yang secara diam-diam membawa data halaman itu sendiri melintasi setiap postback di dalam sebuah hidden field. Kuliah 14 menghubungkan model kontrol server yang sama itu ke basis data nyata dan persisten — sebuah GridView dengan kolom BoundField dan CommandField eksplisit, DataKeyNames yang melacak primary key sesungguhnya, dan event RowEditing/RowUpdating/RowDeleting yang terhubung ke perintah ADO.NET yang genuine, termasuk sebuah bug unboxing Int64-vs-Int32 nyata yang ditemukan dan diperbaiki oleh pengujian nyata mata kuliah ini. Kuliah 15 menambahkan dua bagian lagi yang dibutuhkan setiap aplikasi Web Forms nyata: kontrol validasi (RequiredFieldValidator, RegularExpressionValidator, RangeValidator, dan CustomValidator untuk aturan yang melibatkan lebih dari satu field) dan perbedaan nyata yang telah didemonstrasikan antara ViewState, Session, dan Application — tiga tempat berbeda untuk mengingat data, masing-masing dengan masa hidupnya sendiri (Gambar 16.1).

Peta Jalan Mata Kuliah WebPro

Bab ini adalah Kuliah 16, Ujian Akhir: checkpoint atas Kuliah 13-15, dan kuliah terakhir mata kuliah ini.



Gambar 16.1 — Bab ini berada di Kuliah 16 dalam peta jalan 16 kuliah WebPro: Ujian Akhir, dan kuliah terakhir mata kuliah ini.

16.2 Cara Menggunakan Bab Latihan Ini

Apa bab ini — dan bukan apa

Ini adalah Bab Latihan: bab praktik, bukan bab materi baru. Seperti setiap asesmen WebPro, Ujian Akhir yang sesungguhnya adalah satu PROYEK take-home kecil — satu fitur ASP.NET Web Forms yang berjalan, diserahkan sebagai laporan ditambah repositori GitHub, bukan serangkaian pertanyaan jawaban-singkat bernomor. Bagian 16.3 di bawah membahas keempat deliverable proyek tersebut (masing-masing dapat ditelusuri ke kuliah tertentu, lihat Gambar 16.2), masing-masing dengan panduan menuju praktik yang baik, bukan solusi lengkap yang sudah dikerjakan atau kode sumber acuan. Seperti Bab 4, 8, dan 12, bab ini tidak menyertakan subfolder code/ — proyek ini adalah milik Anda untuk dirancang dan dibangun, menggunakan kode Bab 13-15 yang benar-benar dijalankan sebagai acuan BAGAIMANA membangunnya, bukan APA yang harus diserahkan.

Asal Setiap Deliverable Ujian Akhir

Ujian Akhir adalah satu proyek ASP.NET kecil take-home -- setiap deliverable di Bagian 16.3 dapat ditelusuri kembali ke Kuliah 13-15.

#	Deliverable Ujian Akhir	Berasal dari
1	Halaman Web Forms yang Terorganisasi Baik	K13 -- kontrol server, pemisahan code-behind, model ViewState/postback yang genuine
2	Fitur CRUD Nyata GridView + ADO.NET	K14 -- BoundField/CommandField, DataKeyNames, perintah ADO.NET terparameterisasi
3	Kontrol Validasi dan Cakupan State yang Tepat	K15 -- validator bawaan + CustomValidator, dan pilihan ViewState/Session/Application yang disengaja
4	Laporan Tertulis: Analisis Desain & State	K13-K15 -- menjelaskan, dengan kata Anda sendiri, mengapa desainnya aman, terorganisasi, dan memakai cakupan state yang tepat

Gambar 16.2 — Setiap deliverable di Bagian 16.3 dapat ditelusuri kembali ke Kuliah 13, 14, atau 15.

Sebelum Anda mulai

Sketsalah kolom GridView fitur Anda dan tentukan, untuk setiap data yang perlu diingat proyek Anda, apakah data itu masuk ViewState, Session, atau Application — SEBELUM menulis markup .aspx apa pun. Ujian Akhir menghargai fitur kecil yang terorganisasi dengan benar dan memakai cakupan state yang tepat, dibandingkan fitur besar yang kebetulan berjalan karena menyimpan segalanya di Session demi kemudahan.

16.3 Proyek Mini Ujian Akhir

Rancang dan bangun satu fitur ASP.NET Web Forms yang kecil dan nyata — baik berupa fitur baru yang ditambahkan ke situs Profitina Laundry dari Bab 13-15 (log masukan pelanggan, daftar inventaris sederhana, pelacak permintaan layanan), atau fitur kecil setara untuk situs kecil lain pilihan Anda — selama fitur itu mendemonstrasikan setiap deliverable di bawah ini.

16.3.1 Memilih Fitur Anda

Pilih fitur yang dibangun di sekitar satu tabel basis data kecil yang setidaknya membutuhkan sebuah halaman yang menampilkan dan mengedit baris yang ada (bentuk yang sama seperti Orders.aspx pada Bab 14) dan setidaknya satu formulir input yang tervalidasi (bentuk yang sama seperti Register.aspx pada Bab 15). Satu tabel yang dirancang dengan baik lengkap dengan GridView dan satu formulir tervalidasi lebih baik daripada fitur yang lebih besar dengan beberapa halaman yang setengah jadi.

Petunjuk

Jika Anda tidak dapat menjelaskan, masing-masing dalam satu kalimat, apa yang ditampilkan GridView Anda dan apa yang dikumpulkan formulir tervalidasi Anda, kemungkinan besar fitur tersebut terlalu besar cakupannya untuk Ujian Akhir. Usahakan sesuatu yang dapat Anda selesaikan, diuji dari ujung ke ujung, jauh sebelum tenggat waktu.

16.3.2 Deliverable yang Diwajibkan & Laporan

Serahkan proyek Anda sebagai repositori GitHub (berkas sumber, termasuk halaman .aspx/.aspx.cs Anda, Web.config, dan skema tabel Anda) ditambah laporan tertulis singkat. Laporan sebaiknya membahas secara singkat: fitur Anda dan rancangan tabelnya, validator mana yang Anda gunakan dan mengapa, penjelasan spesifik tentang cakupan state mana (ViewState, Session, atau Application) yang Anda pilih untuk setiap data yang diingat proyek Anda dan mengapa cakupan itu benar, serta satu paragraf yang menjelaskan, dengan kata Anda sendiri, bagaimana rancangan Anda melindungi dari kegagalan unboxing cast seperti yang ditemukan oleh pengujian nyata Bab 14.

Mengapa penjelasan cakupan state harus spesifik untuk proyek Anda

Pengujian nyata Bab 15 sendiri membuktikan, dengan mengamati langsung sesi kedua di tengah pengujian, bahwa ViewState direset per instance halaman, Session direset per pengguna, dan Application dibagi oleh semua orang. Laporan adalah tempat Anda menunjukkan bahwa Anda memahami MENGAPA pilihan cakupan proyek Anda sendiri benar untuk setiap nilai spesifik — menunjuk pada kode Anda sendiri, bukan mengulang definisi umum ketiga cakupan tersebut.

16.3.3 Checklist Halaman Web Forms yang Terorganisasi Baik

Deklarasikan kontrol Anda di markup .aspx dengan `runat="server"`, simpan semua logika nyata — event handler, akses data, logika validasi — di berkas code-behind, dan hindari menyisipkan SQL atau aturan bisnis langsung di markup. Jika proyek Anda menyimpan nilai per-instance-halaman apa pun yang bertahan melintasi postback dan tidak seharusnya masuk basis data, gunakan ViewState untuk itu secara sengaja, seperti AttemptCount pada Bab 15 — bukan karena ASP.NET kebetulan menjadikan ViewState sebagai default, tetapi karena Anda memutuskan bahwa nilai tertentu itu memang milik satu instance halaman ini.

> **Berkas code-behind yang berupa satu event handler raksasa yang melakukan segalanya bukanlah Web Forms yang terorganisasi baik**
>
> Halaman contoh Bab 13 dan 14 sendiri memisahkan tanggung jawab: sebuah metode helper privat (seperti BindGrid()) yang hanya mengikat data, event handler terpisah untuk setiap operasi GridView, dan markup yang hanya mendeklarasikan kontrol dan propertinya. Sebuah metode Page_Load sepanjang 200 baris yang memvalidasi, melakukan query, dan me-render segalanya sekaligus belum benar-benar memanfaatkan tujuan pemisahan code-behind.

16.3.4 Checklist GridView + ADO.NET CRUD Nyata

Ikat sebuah GridView dengan `AutoGenerateColumns="false"` dan kolom BoundField/CommandField eksplisit ke tabel basis data nyata melalui ADO.NET, dengan DataKeyNames melacak primary key sesungguhnya. Setiap INSERT, UPDATE, dan DELETE yang menyertakan nilai dari postback harus menggunakan perintah terparameterisasi — SQL yang digabung-string di mana pun dalam pengumpulan tugas Anda adalah kegagalan otomatis untuk deliverable ini. Saat membaca kembali primary key dari DataKeyNames, unbox dengan `Convert.ToInt32(...)` alih-alih cast langsung `(int)`, perbaikan yang dibutuhkan pengujian nyata Bab 14 untuk alasan yang persis sama.

> **Cast (int) langsung pada nilai DataKeyNames tidak dijamin aman**
>
> Pengujian nyata Bab 14 sendiri menemukan sebuah `System.InvalidCastException` genuine dari pola persis ini: penyedia data mengembalikan primary key sebagai `System.Int64`, dan cast unboxing langsung ke `(int)` melempar exception. `Convert.ToInt32(...)` berhasil terlepas dari apakah tipe boxed yang mendasarinya Int32 atau Int64 — verifikasi perilaku sesungguhnya penyedia Anda sendiri, jangan mengasumsikan bahwa itu cocok dengan apa pun yang kebetulan dihasilkan pengujian berbasis SQLite mata kuliah ini.

16.3.5 Checklist Validasi & Cakupan State

Lampirkan RequiredFieldValidator, dan setidaknya satu dari RegularExpressionValidator atau RangeValidator, pada setiap input yang dikumpulkan formulir Anda. Tulis setidaknya satu CustomValidator yang mengungkapkan aturan yang bergantung pada lebih dari satu kontrol — bentuk yang sama seperti aturan "Setrika Saja dibatasi 20 kg" pada Bab 15, yang tidak dapat diungkapkan oleh satu validator bawaan mana pun. Untuk setiap data yang diingat proyek Anda lintas request, pilih secara eksplisit ViewState, Session, atau Application, dan mampu mempertanggungjawabkan pilihan itu dalam laporan Anda.

Petunjuk

Tanyakan, untuk setiap nilai yang Anda tergoda untuk simpan: apakah ini milik satu instance halaman (ViewState), satu pengguna yang login lintas banyak halaman (Session), atau semua orang yang menggunakan aplikasi sekaligus (Application)? Pengujian nyata Bab 15 sendiri membuktikan ketiga cakupan ini berperilaku berbeda dalam satu pengujian nyata yang berkesinambungan — salah memilih cakupan membuat sebuah nilai direset padahal seharusnya bertahan, atau bocor antar-pengguna padahal seharusnya tidak.

16.4 Format Ujian Akhir: Proyek Open-Book, Take-Home

Ujian Akhir adalah proyek open-book, take-home, diserahkan sebagai repositori GitHub ditambah laporan tertulis singkat — bentuk deliverable yang sama seperti Kuis 1, UTS, dan Kuis 2, hanya saja mencakup jalur ASP.NET secara spesifik. Tidak ada komponen berwaktu di kelas.

Open-book berarti referensi, bukan kolaborator

Anda boleh berkonsultasi dengan buku ajar, dokumentasi resmi ASP.NET/ADO.NET, dan catatan Anda sendiri saat membangun proyek Anda. Anda TIDAK BOLEH menyalin repositori mahasiswa lain atau menyerahkan kode yang bukan benar-benar milik Anda sendiri — proyek take-home open-book menguji apakah Anda dapat MENERAPKAN apa yang telah Anda pelajari, bekerja secara mandiri.

Kriteria	Yang Dinilai	Bobot
Desain	Rancangan tabel dan rencana halaman: apakah fitur memiliki tujuan yang jelas, tata letak kolom GridView yang masuk akal, dan rencana validator serta cakupan state mana yang akan dipakai sebelum .aspx mana pun ditulis?	20%
Implementasi	Apakah fitur benar-benar berjalan dengan benar: halaman Web Forms yang terorganisasi baik, fitur CRUD GridView + ADO.NET nyata dengan penanganan DataKeyNames yang aman, validasi yang berfungsi termasuk CustomValidator, dan state disimpan pada cakupan yang benar?	50%
Analisis & evaluasi	Apakah penjelasan cakupan state dan keamanan dalam laporan menunjukkan pemahaman nyata dengan kata mahasiswa sendiri, terkait dengan proyeknya sendiri, bukan definisi umum dari buku ajar?	25%
Kesimpulan	Refleksi singkat dan jujur: apa yang berhasil, apa yang akan dilakukan berbeda dengan waktu yang lebih banyak.	5%

16.5 WebPro dalam Praktik: Profitina

> **Tentang kotak ini**
>
> Seperti pada setiap bab, kotak ini menghubungkan materi mata kuliah dengan Profitina, platform AI-visibility (Generative Engine Optimization) Indonesia yang nyata dan aktif, dibangun oleh penulis, serta dengan Profitina Laundry, studi kasus yang dipakai buku ini — menjelaskan DI MANA dan MENGAPA sebuah konsep dipakai dalam sistem nyata yang berjalan, tidak pernah logika bisnis atau kode sumber lengkap Profitina yang bersifat rahasia.

Setiap fitur entri-data dan penampilan-daftar milik app.profitina.com sendiri mengikuti persis bentuk Ujian Akhir ini: input yang tervalidasi, daftar berbasis data dengan primary key yang bertipe benar, dan pilihan yang disengaja tentang cakupan mana — request, sesi pengguna, atau state proses bersama — yang memiliki setiap data. Tidak satu pun dari ketiga kuliah ASP.NET mata kuliah ini bersifat kebetulan: sebuah halaman yang benar dalam validasi, kebenaran CRUD, dan cakupan state dalam skala kecil di sini sedang melakukan, dalam skala kecil, persis disiplin yang harus dijalankan sistem produksi dalam skala yang jauh lebih besar.

16.6 Ringkasan Bab

- Bab ini tidak memperkenalkan materi baru — ini adalah checkpoint yang mencakup Kuliah 13 hingga 15, dan bab terakhir Pemrograman Web.
- Ujian Akhir, seperti setiap asesmen WebPro, adalah satu PROYEK take-home kecil (repositori GitHub ditambah laporan), bukan serangkaian pertanyaan jawaban-singkat terpisah.
- Keempat deliverable proyek memetakan ke tiga kuliah yang direkap: halaman Web Forms yang terorganisasi baik (K13), fitur CRUD GridView + ADO.NET nyata (K14), dan validasi plus cakupan state yang benar (K15).
- Bobot penilaian terberat pada Implementasi (50%), diikuti Analisis & Evaluasi (25%), Desain (20%), dan Kesimpulan (5%) — bentuk rubrik yang sama yang dipakai ulang setiap asesmen WebPro.

Fitur yang berjalan dengan benar di komputer mahasiswa sendiri bukanlah hal yang sama dengan fitur yang aman dari kegagalan unboxing cast dan benar tentang cakupan mana yang memiliki data mana — dan satu-satunya cara penilai dapat membedakannya adalah kode mahasiswa sendiri dan laporannya sendiri yang menjelaskannya.

Lampiran 16.A — Checklist Periksa-Mandiri (Tidak Dinilai)

Sebelum menyerahkan repositori dan laporan Anda, jalankan proyek Anda melalui checklist ini. Ini hanya memakan beberapa menit dan menangkap sebagian besar kesalahan yang paling sering dilihat penilai pada pengumpulan Ujian Akhir pertama.

- Apakah GridView Anda memakai AutoGenerateColumns="false" dengan kolom BoundField/CommandField eksplisit dan DataKeyNames yang benar?
- Apakah setiap INSERT, UPDATE, dan DELETE yang menyentuh nilai postback memakai perintah terparameterisasi — dengan nol penggabungan string ke SQL di mana pun?
- Apakah Anda meng-unbox primary key DataKeyNames mana pun dengan Convert.ToInt32(...) alih-alih cast (int) langsung?

- Apakah setiap input memiliki setidaknya RequiredFieldValidator, dan apakah setidaknya satu CustomValidator mengungkapkan aturan yang melibatkan lebih dari satu kontrol?
- Untuk setiap data yang diingat, dapatkah Anda menyebutkan cakupan mana (ViewState/Session/Application) yang dipakainya dan menjelaskan mengapa cakupan itu — bukan yang lain — benar?
- Apakah semua logika nyata berada di berkas code-behind, dengan markup .aspx hanya mendeklarasikan kontrol dan propertinya?
- Apakah paragraf cakupan state dalam laporan menjelaskan pilihan spesifik proyek INI sendiri, bukan definisi umum yang disalin dari ketiga cakupan tersebut?
- Akankah seorang penilai yang belum pernah melihat proyek ini mampu menyiapkan dan menjalankannya hanya dari instruksi dalam laporan?

Referensi

[1] Format asesmen bab latihan ini dimodelkan langsung dari Ujian Akhir nyata mata kuliah ini (EF234301 Pemrograman Web): proyek take-home open-book yang diserahkan sebagai repositori GitHub ditambah laporan tertulis, dinilai berdasarkan Desain (20%), Implementasi (50%), Analisis & Evaluasi (25%), dan Kesimpulan (5%) — bentuk rubrik yang sama yang dipakai setiap asesmen WebPro (Kuis 1, UTS, Kuis 2, Ujian Akhir). Format ini dipakai ulang apa adanya di sini; hanya brief proyek spesifik di atas (yang merekap konten nyata Kuliah 13-15) dan logistik pengumpulan administratif yang telah diadaptasi untuk buku ajar ini.

[2] Microsoft. "ASP.NET Web Forms", "GridView Web Server Control Overview", dan "ASP.NET Validation Controls" documentation (diakses Agustus 2026).

[3] Microsoft. "ASP.NET State Management Overview" documentation (diakses Agustus 2026) — sumber untuk panduan pemilihan cakupan ViewState/Session/Application di Bagian 16.3.5.