Pemrograman Web Lanjut

SIB245007 | D-IV Sistem Informasi Bisnis

Pertemuan 1: **Arsitektur Web Modern dan Ekosistem Laravel**

Rencana Pembelajaran Semester (RPS) & Materi Pertemuan 1

Bagian 1

Identitas Mata Kuliah

Program Studi	D-IV Sistem Informasi Bisnis
Kode Mata Kuliah	SIB245007
Nama Mata Kuliah	Pemrograman Web Lanjut

> Mata kuliah ini memakai **Simple POS**, aplikasi kasir/point-of-sale UMKM yang kodenya nyata dan berkembang commit demi commit, sebagai studi kasus berjalan sepanjang satu semester.

Rencana Pembelajaran Semester

Pertemuan	Materi
1	Arsitektur Web Modern
2	HTTP & Arsitektur MVC
3	Frontend & Templating
4	Desain Basis Data & Migrasi
5	ORM & Relasi Data
6	Validasi & Keamanan Input
7	Autentikasi & Otorisasi
8	**UTS**
9	Pengolahan & Ekspor Data

Pertemuan	Materi
10	Arsitektur & Desain API
11	Perencanaan Proyek PBL
12	Pengembangan Fitur PBL
13	Integrasi & Pengujian PBL
14	Optimasi & Deployment PBL
15	Finalisasi Proyek PBL
16	Persiapan Ujian Akhir PBL
17	**UAS**

> Pertemuan 1–10 membangun fondasi teknis Laravel; Pertemuan 11–17 mengalihkannya menjadi proyek *Project Based Learning* (PBL) mandiri.

Komponen Evaluasi

Basis Evaluasi	Bobot
Aktivitas Partisipatif (Case Method)	0%
Hasil Proyek (Project Based Learning)	55%
Kognitif – Tugas mingguan (increment)	15%
Kognitif – Quiz	0%
Kognitif – UTS (progress proyek + presentasi)	5%
Kognitif – UAS (proyek final + presentasi)	25%
Total	**100%**

> **Tugas mingguan**: implementasi bertahap (*increment*) aplikasi Simple POS, dikumpulkan sebagai bukti kode dan dokumentasi sesuai rubrik.

Bagian 2

Yang Akan Kamu Pelajari

1. Membandingkan arsitektur **monolith**, **microservices**, dan **serverless** untuk memahami dasar pemilihan arsitektur sebuah aplikasi web
2. Memahami alasan **Laravel 13** dengan **SQLite** sebagai basis data zero-setup dipilih untuk proyek Simple POS yang akan kamu bangun sepanjang semester
3. Mengenali struktur folder proyek Laravel (`routes/`, `app/Http/Controllers/`, `database/migrations/`) sebagai perwujudan pola **MVC**

> Slide ini membahas konsep. Langkah instalasi, setup proyek, dan latihan praktik lengkap dibahas terpisah di luar slide ini.

Apa itu Arsitektur Web?

> **Arsitektur web:** cara lapisan-lapisan aplikasi (antarmuka, logika bisnis, akses data) diorganisasi dan di-deploy: satu unit, atau banyak unit terpisah.

- Pilihan arsitektur bukan sekadar teknis: ia menentukan berapa banyak proses deploy, titik gagal, dan komunikasi jaringan yang harus dikelola tim
- Arsitektur yang "lebih canggih" bukan berarti lebih baik: membangun food court untuk bisnis satu dapur hanya menghabiskan usaha untuk pipa penghubung, bukan fitur
- Tiga gaya yang akan kita bandingkan: **monolith**, **microservices**, **serverless**

Monolith

> **Monolith:** Aplikasi yang seluruh lapisannya (antarmuka, logika bisnis, akses data) berjalan dalam satu basis kode dan satu proses deploy.

- Menambah fitur = menambah kode pada proyek yang sama
- Deploy pembaruan = mengganti satu unit dengan versi baru
- Sering disalahpahami sebagai "kode berantakan"
- Monolith terstruktur (mis. dengan MVC) tetap rapi
- Lawan katanya bukan "modular", melainkan **"terdistribusi"**

Microservices

> **Microservices:** Arsitektur yang memecah aplikasi menjadi layanan independen, masing-masing berjalan dan di-deploy sendiri, saling berkomunikasi lewat jaringan.

- Setiap layanan: bahasa berbeda, jadwal deploy berbeda, skala sendiri-sendiri
- **Harganya:** 8 layanan = 8 proses deploy + 8 titik gagal + komunikasi jaringan antar-layanan
- Sepadan untuk **tim besar** dengan sistem berskala jutaan pengguna
- Untuk aplikasi skala kecil: ongkos jauh melebihi manfaatnya

Analogi: Restoran Keluarga vs. Food Court

Dua arsitektur yang baru saja kita definisikan, dalam analogi sehari-hari:

Restoran keluarga (monolith)

- Satu dapur, satu kasir, satu tim
- Semua orang tahu semua hal
- Ramai? Tambah kompor, bukan cabang baru

Food court (microservices)

- Setiap tenant: dapur, kasir, resep sendiri
- Unit-unit independen
- Satu tenant sepi tidak mengganggu yang lain

> Sebuah aplikasi dipilih sebagai **monolith** (restoran keluarga) bukan karena keterbatasan, tapi karena skalanya cocok: satu warung dengan satu-dua kasir tidak butuh sepuluh layanan terpisah.

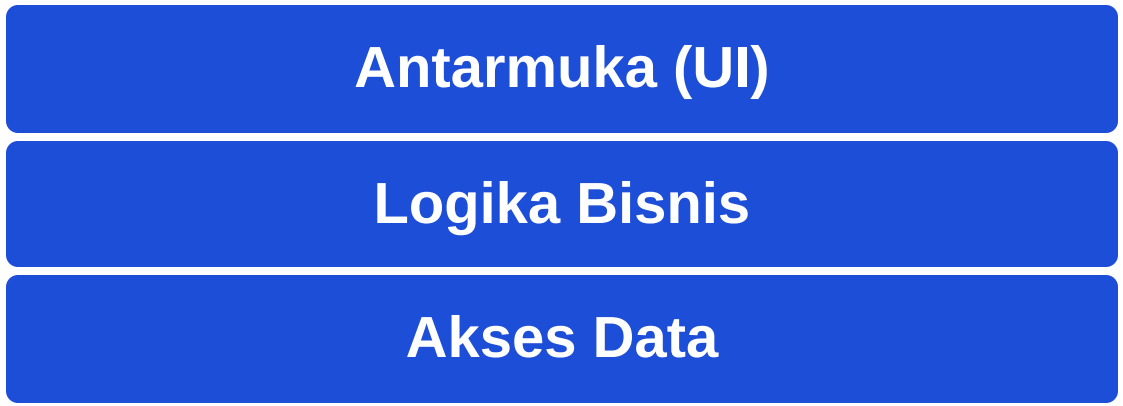
Serverless

> **Serverless:** Kode dieksekusi sebagai fungsi-fungsi kecil yang hanya berjalan saat dipicu peristiwa, tanpa proses server yang menyala terus-menerus.

- Namanya menyesatkan: server tetap ada, hanya bukan tanggung jawabmu
- Ada jeda **cold start** saat fungsi lama tidak dipanggil
- Biaya dihitung per eksekusi, bukan per jam server menyala
- Cocok: beban kerja naik-turun tajam (mis. proses gambar saat upload)
- Kurang cocok: aplikasi yang butuh koneksi basis data konsisten

Perbandingan Ketiga Arsitektur

Arsitektur	Deployment	Kompleksitas Awal	Cocok Untuk
Monolith	Satu unit	Rendah	Aplikasi skala kecil, MVP, tim kecil
Microservices	Banyak unit independen	Tinggi	Sistem skala besar, tim besar
Serverless	Fungsi per event	Sedang	Beban kerja sporadis



Monolith

Microservices

Mengapa Laravel?

> **MVC (Model-View-Controller):** Model mengurus data, View mengurus tampilan, Controller mengurus alur permintaan di antara keduanya.

- Satu proyek PHP: routing, autentikasi, ORM, template, siap pakai
- Dibanding PHP polos: struktur MVC konsisten sejak baris kode pertama
- Dibanding framework microservices-first: tetap produktif untuk tim 1–2 orang
- Ekosistem paket kuat: **Sanctum** (API), **Excel** (impor/ekspor), **Cashier** (pembayaran)
- Filosofi mirip Django (Python) & Ruby on Rails: *convention over configuration*

Satu Pintu Masuk: `public/index.php`

PHP polos

- URL memetakan langsung ke berkas
- `/produk.php` → menjalankan `produk.php`
- Setiap berkas berpotensi diakses langsung lewat URL

Laravel

- Semua request masuk lewat satu berkas: `public/index.php`
- Alamat URL didaftarkan di `routes/web.php`
- Berkas lain (Controller, Model) tidak bisa diakses langsung lewat URL

> Satu pintu masuk berarti setiap request bisa diproses seragam sebelum sampai ke kode aplikasi: dasar dari routing, middleware, dan autentikasi terpusat.

Alur Satu Request Laravel

> Pola alur ini berulang di setiap fitur aplikasi Laravel, dari halaman sederhana hingga endpoint REST API, semua mengikuti jalur yang sama.

Menyiapkan Proyek: Kenapa SQLite?

- Basis data disimpan dalam **satu berkas biasa**
- Tanpa proses server terpisah yang harus dinyalakan & dikonfigurasi
- Migrasi bisa langsung dijalankan di menit pertama
- Simple POS berjalan di atas SQLite bahkan untuk demonstrasi kelas

> Sebelum lanjut, pastikan empat alat berikut sudah terpasang: `php -v` (8.2+), `composer -V`, `node -v`, dan `git --version`.

Composer & npm: Manajer Dependensi

> **Composer:** manajer dependensi PHP yang membaca `composer.json`, mengunduh paket ke `vendor/`, lalu membuat `vendor/autoload.php`.

- Berkat `autoload.php`, setiap class langsung bisa dipakai, tidak perlu `require` / `include` manual seperti PHP polos
- **npm** adalah rekannya di dunia JavaScript: `package.json` mendaftarkan paket, `node_modules/` menyimpannya, dipakai Laravel untuk Vite & Tailwind
- Kedua folder (`vendor/`, `node_modules/`) hasil unduhan, tidak pernah di-commit ke Git

`.env` & Lapisan Konfigurasi

> **.env:** berkas konfigurasi yang memisahkan kredensial dan pengaturan lingkungan dari kode sumber.

- Alur baca konfigurasi: `.env` → helper `env()` → `config/*.php` → helper `config()` yang dipakai kode aplikasi
- App key di dalamnya dipakai untuk mengenkripsi session dan cookie
- Pemisahan ini memungkinkan konfigurasi berbeda per lingkungan (lokal, staging, produksi) tanpa mengubah kode

Berkas `.env` menyimpan data sensitif dan tidak boleh ikut di-commit. `.gitignore` bawaan Laravel sudah mengecualikannya.

Artisan & Migrasi

> **Artisan:** CLI bawaan Laravel untuk tugas pengembangan sehari-hari (migrasi, seeding, membuat boilerplate): alat bantu develop, bukan bagian dari aplikasi yang dilayani ke pengguna.

> **Migrasi:** berkas PHP yang mendefinisikan perubahan skema basis data secara terprogram, sehingga skema bisa dibangun ulang secara konsisten di mesin mana pun.

- Perlakukan migrasi seperti version control untuk skema: perubahan baru = berkas migrasi baru, jangan mengubah migrasi lama yang sudah dijalankan di tempat lain
- Opsi `--seed` mengisi tabel dengan data contoh yang realistis untuk latihan dan demonstrasi

Struktur Folder Laravel = Wujud MVC

Folder	Peran MVC	Isi
`routes/web.php`	Controller	Pendaftaran alamat URL
`app/Http/Controllers/`	Controller	Kelas pemroses request
`database/migrations/`	Model	Definisi skema basis data
`resources/views/`	View	Berkas Blade (Pertemuan 3)
`vendor/`	-	Paket Composer, tidak di-commit

> Struktur ini bukan kebetulan: ia mewujudkan pola MVC yang sama dengan diagram alur request sebelumnya.

Rangkuman

- **Monolith** menyatukan seluruh lapisan dalam satu basis kode & satu deploy, cocok untuk aplikasi skala kecil-menengah; **microservices** memecahnya dengan ongkos yang sepadan hanya untuk sistem besar; **serverless** cocok untuk beban kerja sporadis
- Laravel dipilih sebagai kerangka kerja karena strukturnya konsisten sejak awal (MVC) dan produktif untuk tim kecil; **SQLite** dipilih sebagai basis data karena zero-setup: satu berkas, tanpa server terpisah
- **Composer**/npm mengelola dependensi; **.env** memisahkan konfigurasi dari kode; **Artisan** & **migrasi** membangun skema basis data secara terprogram
- Struktur folder Laravel mewujudkan pola **MVC**, memisahkan tanggung jawab routing, logika bisnis, dan tampilan secara konsisten

Referensi & Diskusi

Dokumentasi resmi Laravel · Manual PHP

Kode lengkap: `github.com/se-polinema/simple-pos`

Pertemuan berikutnya: HTTP & Arsitektur MVC