# PENDEKATAN SOAL TEORI LSP

## MODUL TES TEORI PROJECT PERPUSTAKAAN ONLINE DENGAN LARAVEL

---

⚠️ Beberapa hal yang perlu diperhatikan

- Soal – soal yang terlampir dalam dokumen ini telah disesuaikan dengan project dummy/Latihan lsp perpustakaan yang telah dibuat
- Jawaban yang terlampir tidak secara general pengertian namun lebih mengarah kepada pendekatan kode itu sendiri
- Link GitHub project dummy : GitHub
- Topik mendasar yang wajib dikuasai adalah MVC, Eloquent Relationship, Middleware, Validasi Blade, CRUD, Authentication, Error Handling

**T E K N O L O G I  Y A N G  D I G U N A K A N**

Laravel • PHP • MySQL • Tailwind CSS • Node.js

**BAGIAN 1: PENJELASAN PROJECT SECARA KESELURUHAN**

## 🏠 Apa Sih Project Ini?

Project ini adalah **Sistem Perpustakaan Digital** yang dibangun menggunakan **Laravel.** Bayangkan seperti ini:

Kamu punya perpustakaan di sekolah. Ada **Admin** (petugas perpustakaan) yang mengatur buku dan anggota. Ada **Siswa** yang bisa meminjam dan mengembalikan buku. Project ini membuat semua proses itu jadi digital lewat website!

## 🗂️ Struktur Folder Project

```
perpus-lsp-main/
├── app/                    ← 🧠 "OTAK" aplikasi (logic & aturan)
│   ├── Enums/              ← Definisi role (admin/siswa)
│   ├── Http/
│   │   ├── Controllers/    ← 🎮 "Remote Control" - mengatur apa yang terjadi
│   │   │   ├── Auth/       ← Login, Register, Logout
│   │   │   ├── BookController.php
│   │   │   ├── OrderController.php  (peminjaman)
│   │   │   ├── UserController.php
│   │   │   ├── SiswaController.php
│   │   │   └── DashboardController.php
│   │   ├── Middleware/     ← 🚪 "Satpam" - cek hak akses
│   │   └── Requests/       ← 📋 "Formulir Validasi"
│   ├── Models/             ← 📦 "Cetakan Data" - representasi tabel database
│   │   ├── User.php, Book.php, Orders.php, anggota.php, Book_Orders.php
│   └── Providers/
├── database/               ← 🗄️ "Rak Arsip"
│   ├── migrations/         ← 📐 "Blueprint" - desain tabel
│   └── seeders/            ← 🌱 "Data Contoh"
├── resources/views/        ← 🎨 "Tampilan" - apa yang dilihat user
│   ├── admin/              ← Halaman khusus admin
│   ├── siswa/              ← Halaman khusus siswa
│   └── auth/               ← Halaman login & register
├── routes/web.php          ← 🗺️ "Peta Jalan" - URL mana ke mana
├── .env                    ← ⚙️ "Pengaturan Rahasia"
└── bootstrap/app.php       ← 🚀 "Starter" - konfigurasi awal
```

🌐 Konfigurasi Project

**Name Project : Perpustakaan_lsp2**

- Install Composer
  “Composer Install”
- Copy paste file “.env.example” lalu rename manjadi “.env”
- Setup Database :
  DB_CONNECTION=mysql
  DB_HOST=127.0.0.1
  DB_PORT=3306
  DB_DATABASE=perpustakaan_lsp < *nama databasenya mengikuti nama database yang ingin kalian gunakan*
  DB_USERNAME=root
  DB_PASSWORD= < *jika phpMyAdmin kalian menggunakan password silahkan masukkan passwordnya*
- Buat Key Generate Laravel
  “php artisan key:generate”
- Migrasi database
  “php artisan migrate”
- Migrasi Seeder
  “php artisan db:seed”
- Jalankan server Laravel
  “php artisan serve”

🌐 Konfigurasi Project

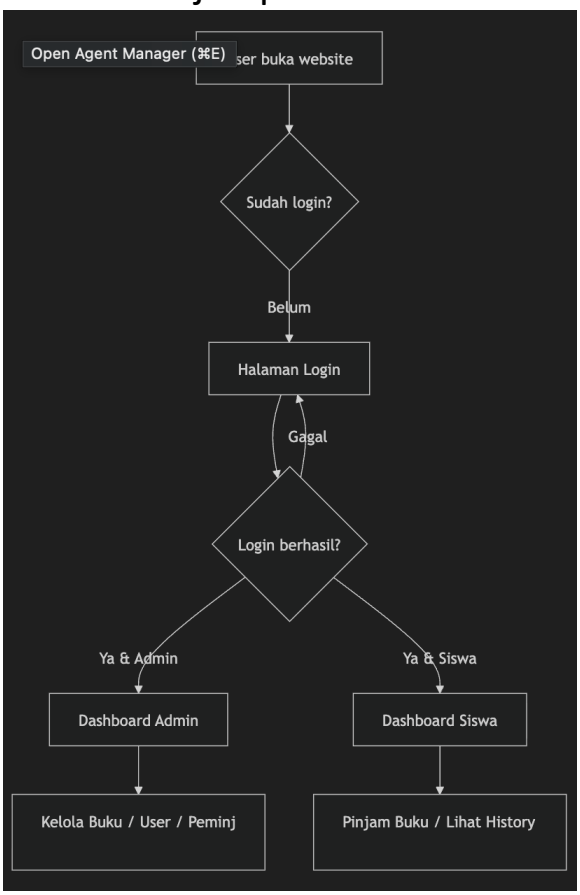
**Name Project : Perpustakaan_lsp**

- Install Composer
  “Composer Install”
- Copy paste file “.env.example” lalu rename manjadi “.env”
- Setup Database :
  DB_CONNECTION=mysql
  DB_HOST=127.0.0.1
  DB_PORT=3306

DB_DATABASE=perpustakaan_lsp < *nama databasenya mengikuti nama database yang ingin kalian gunakan*
DB_USERNAME=root
DB_PASSWORD= < *jika phpMyAdmin kalian menggunakan password silahkan masukkan passwordnya*

- Buat Key Generate Laravel
  "php artisan key:generate"
- Migrasi database
  "php artisan migrate"
- Migrasi Seeder
  "php artisan db:seed"
- Jalankan server Laravel
  "php artisan serve"
- Install Node JS di Google Chrome atau website pencarian lainnya *( install node js untuk os windows )*
  Beberapa keterangan :
  - Jangan centang pada step yang mengharuskan kalian mendownload chocolatey
- Install JS depenc.
  "npm install"
- Jika muncul error pada saat menjalankan npm install coba jalankan perintah
  "Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned" dan jalankan Kembali "npm install"
- Jalankan server Node js
  "npm run dev"
- Buat terminal baru, jangan close/kill terminal server node js, di terminal baru jalankan server Laravel
  "php artisan serve"

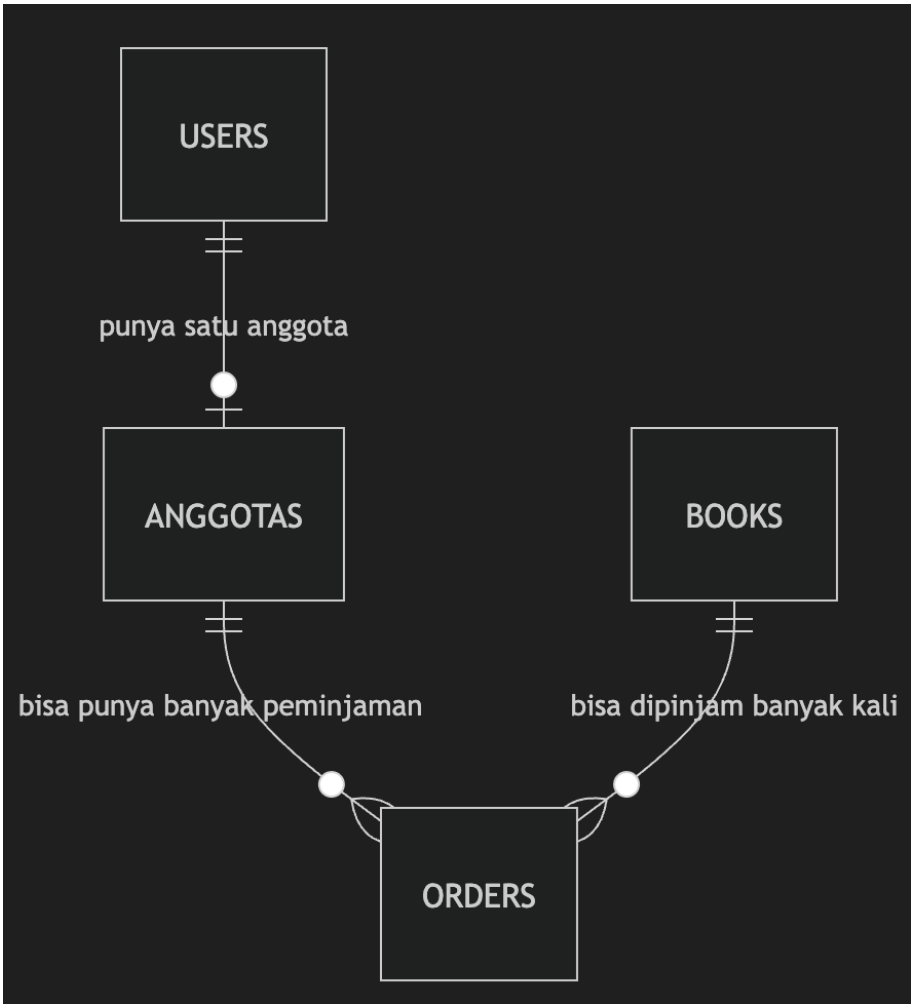
## 🔄 Alur Kerja Aplikasi




## 📊 Tabel Database (4 tabel utama)

| Tabel | Isi | Kolom |
|---|---|---|
| `users` | Akun login | id, username, password, role, anggota_id |
| `anggotas` | Data anggota/siswa | id, nis, nama, kelas, jurusan |
| `books` | Data buku | id, nama, pengarang, penerbit, stock, tahun_terbit |
| `orders` | Data peminjaman | id, book_id, anggota_id, tanggal_pinjam, tanggal_kembali |

## 🔗 Hubungan Antar Tabel




- **User → Anggota** : Setiap user siswa **punya 1 data anggota** (NIS, nama, kelas, jurusan)
- **Anggota → Orders** : Satu anggota **bisa pinjam banyak buku**
- **Book → Orders :** Satu buku **bisa dipinjam banyak anggota** (selama stock ada)

**BAGIAN 2: JAWABAN SOAL ESSAY**

## ✏️ Soal 1: Jelaskan konsep MVC pada Laravel serta fungsi masing-masing komponennya dalam sistem peminjaman buku

MVC = Model - View - Controller. Bayangkan seperti restoran :

| Komponen | Analogi | Fungsi | Contoh di Project Kita |
|---|---|---|---|
| **Model** | Dapur & Bahan | Mengatur data, berkomunikasi dengan database | Book.php , Orders.php , anggota.php , User.php |
| **View** | Menu & Penyajian | Menampilkan data ke pengguna (HTML) | Semua file .blade.php di `resources/views/` |
| **Controller** | Pelayan | Menerima pesanan user, minta Model ambil data, lalu kirim ke View | `BookController.php`, `OrderController.php`, dll |

**Cara kerja di project kita:**

1. **Model** → [Book.php](file:///Users/mac/Downloads/perpus-lsp-main/app/Models/Book.php) mendefinisikan struktur data buku. Dia tahu kolom apa saja yang bisa diisi (`$fillable`), dan relasinya dengan tabel lain (`orders()`).

2. **View** → [index.blade.php](file:///Users/mac/Downloads/perpus-lsp-main/resources/views/admin/Book/index.blade.php) menampilkan daftar buku dalam bentuk tabel HTML.

3. **Controller** → [BookController.php](file:///Users/mac/Downloads/perpus-lsp-main/app/Http/Controllers/BookController.php) jadi "pengatur lalu lintas":
   - `index()`: ambil semua buku dari Model, kirim ke View
   - `create()`: tampilkan form + terima data + simpan ke database
   - `update()`: ubah data buku
   - `destroy()`: hapus data buku

## ✏️ Soal 2: Uraikan langkah-langkah pembuatan project Laravel dari pembuatan folder hingga siap dijalankan

1. Install Composer (package manager PHP)
2. Buat project:

```bash
composer create-project laravel/laravel perpus-lsp-main
```

3. Konfigurasi `.env` - Atur koneksi database:

```
DB_DATABASE=perpustakaan_lsp
DB_USERNAME=root
DB_PASSWORD=
```

4. Buat database di MySQL: `CREATE DATABASE perpustakaan_lsp;`
5. Generate APP_KEY: `php artisan key:generate`
6. Buat migration: `php artisan make:migration create_books_table`
7. Jalankan migration: `php artisan migrate`
8. Buat Model & Controller: `php artisan make:model Book`, `php artisan make:controller BookController`
9. Buat seeder & jalankan: `php artisan db:seed`
10. Install frontend: `npm install`
11. Jalankan server: `php artisan serve` → akses di `http://localhost:8000`

## ✏️ Soal 3: Jelaskan fungsi file `.env` dan konfigurasi database MySQL

File `.env` = "buku catatan rahasia" berisi pengaturan penting yang TIDAK boleh dibagikan.
Konfigurasi database MySQL (dari [.env]

(file:///Users/mac/Downloads/perpus-lsp-main/.env)):

```env
DB_CONNECTION=mysql          ← Jenis database
DB_HOST=127.0.0.1            ← Alamat server (localhost)
DB_PORT=3306                 ← Port MySQL
DB_DATABASE=perpustakaan_lsp ← Nama database
DB_USERNAME=root             ← Username MySQL
DB_PASSWORD=                 ← Password (kosong = XAMPP default)
```

Konfigurasi penting lainnya: `APP_KEY` (kunci enkripsi), `APP_DEBUG=true` (mode debug), `SESSION_DRIVER=database` (session di database).

## ✏️ Soal 4: Apa yang dimaksud migration? Jelaskan manfaatnya

Migration = "blueprint/cetak biru" tabel database dalam bentuk kode PHP.
Contoh di [create_books_table.php]

(file:///Users/mac/Downloads/perpus-lsp-main/database/migrations/2026_01_21_064556_create_books_table.php):

```php
Schema::create('books', function (Blueprint $table) {
    $table->id();
    $table->string('nama', 50);
    $table->string('pengarang', 50);
    $table->integer('stock');
    $table->timestamps();
});
```

Manfaat:

1. Version Control → Dilacak perubahannya lewat Git
2. Konsistensi Tim → Semua punya struktur database sama
3. Rollback → Bisa mundur kalau ada kesalahan
4. Portabilitas → Bisa pindah jenis database
5. Otomatis → Cukup `php artisan migrate`
6. Dokumentasi → Kode = dokumentasi

## ✏️ Soal 5: Perbedaan migration dan model serta hubungannya

| Aspek | Migration | Model |
|---|---|---|
| **Apa** | Blueprint/desain tabel | Representasi tabel dalam PHP |
| **Fungsi** | MEMBUAT struktur tabel | MENGAKSES & MEMANIPULASI data |
| **Letak** | `database/migrations/` | `app/Models/` |
| **Kapan dijalankan** | Sekali saat setup | Setiap aplikasi berjalan |
| **Analogi** | Arsitek MENDESAIN rumah | Penghuni TINGGAL di rumah |

Hubungan: Migration membuat wadah (tabel), Model menggunakan wadah tersebut. Keduanya harus sinkron — kolom di migration harus ada di `$fillable` Model.

✏️ Soal 6: Struktur tabel transactions dan fungsi field-nya

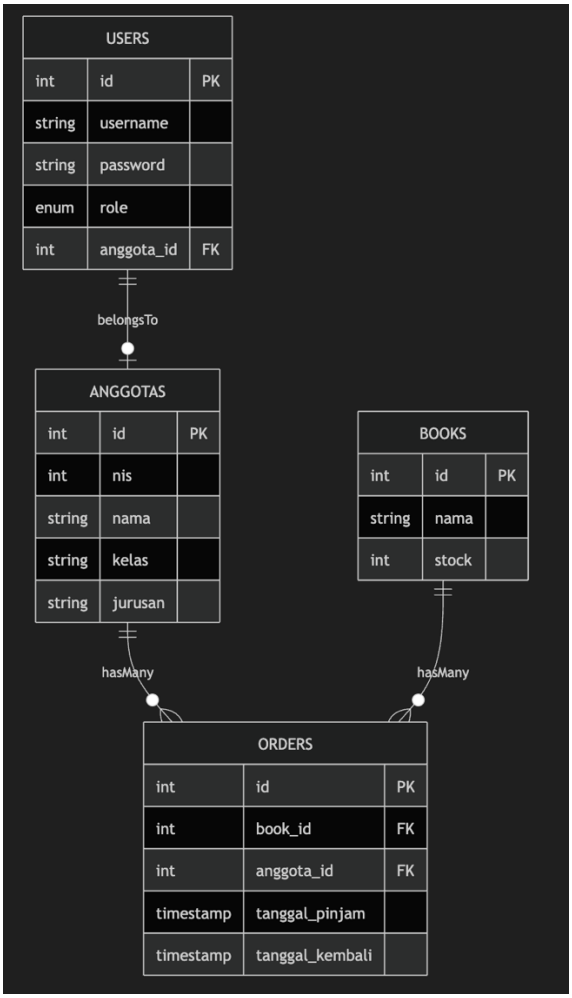
Tabel transaksi = `orders` ([create_orders_table.php]

(file:///Users/mac/Downloads/perpus-lsp-main/database/migrations/2026_01_21_064658_create_orders_table.php)):

| Field | Tipe | Fungsi |
|---|---|---|
| `id` | bigint auto increment | Primary Key, nomor unik transaksi |
| `book_id` | foreignId | FK ke `books` — buku APA yang dipinjam |
| `anggota_id` | foreignId | FK ke `anggotas` — SIAPA peminjam |
| `tanggal_pinjam` | timestamp | Tanggal peminjaman (otomatis terisi) |
| `tanggal_kembali` | timestamp nullable | Tanggal pengembalian — `NULL` = belum dikembalikan |
| `created_at/updated_at` | timestamp | Otomatis oleh Laravel |

[!IMPORTANT]
**tanggal_kembali** yang nullable adalah kunci logika: `NULL` = masih dipinjam, ada tanggal = sudah dikembalikan.

## ✏️ Soal 7: Relasi antara tabel users, books, dan transactions




1. Users → Anggotas (`belongsTo`): Setiap user siswa punya 1 data anggota; admin tidak punya
2. Anggotas → Orders (`hasMany`): Satu anggota bisa punya banyak peminjaman
3. Books → Orders (`hasMany`): Satu buku bisa dipinjam banyak kali
4. Orders → Books & Anggotas (`belongsTo`): Setiap order milik 1 buku dan 1 anggota

## ✏️ Soal 8: Fungsi `$fillable` dan dampak jika tidak didefinisikan

`$fillable` = "daftar ijin masuk" untuk data. Menentukan kolom yang BOLEH diisi secara massal.

Contoh di [Book.php](file:///Users/mac/Downloads/perpus-lsp-main/app/Models/Book.php):

```php
protected $fillable = ['nama', 'pengarang', 'penerbit', 'stock',
'tahun_terbit'];
```

Dampak jika tidak didefinisikan:

1. MassAssignmentException → Laravel MENOLAK simpan data
2. Keamanan Terancam → Tanpa proteksi, hacker bisa ubah field sensitif (misal `role` jadi `admin`)
3. Data Tidak Tersimpan → Kolom di luar `$fillable` diabaikan diam-diam

## ✏️ Soal 9: Middleware UserAkses dan pembatasan hak akses

Di project kita = `RoleMiddleware` ([RoleMiddleware.php](file:///Users/mac/Downloads/perpus-lsp-main/app/Http/Middleware/RoleMiddleware.php)). Middleware = "SATPAM" di pintu masuk.

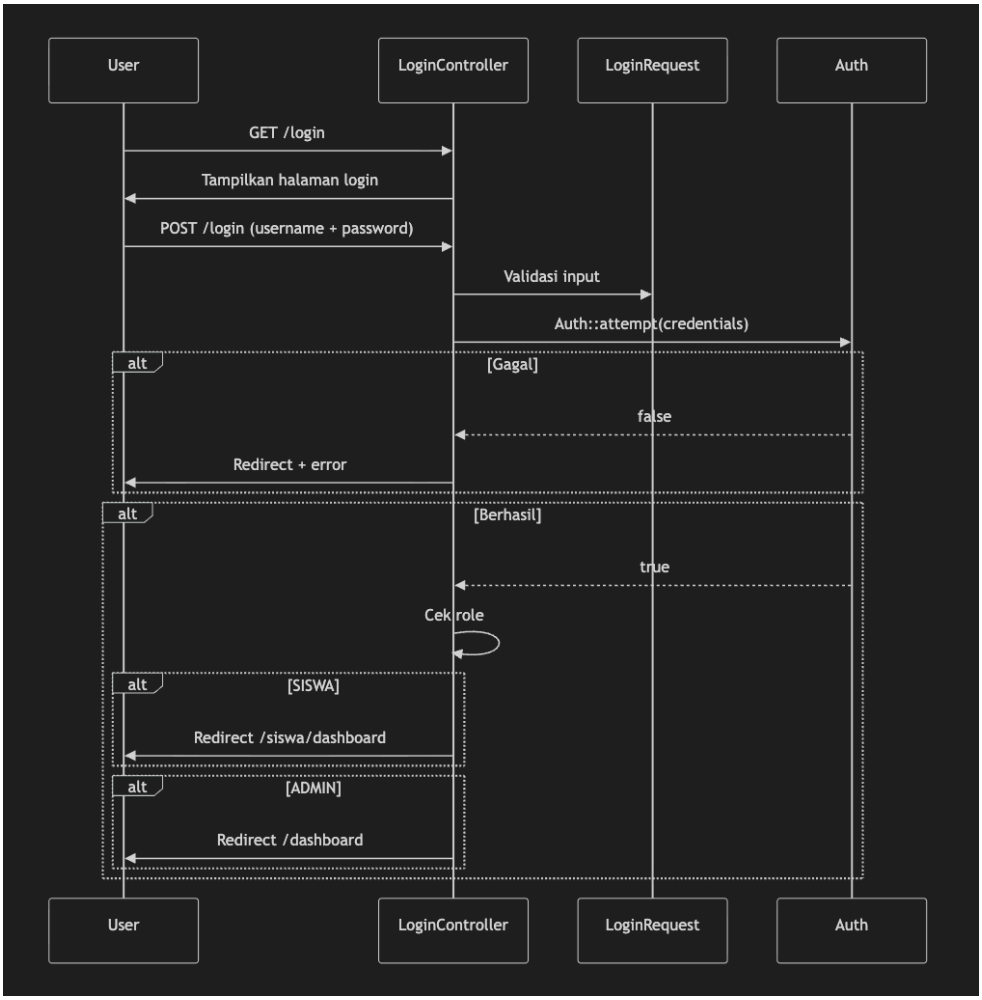
```php
public function handle(Request $request, Closure $next, string $role): Response
{
    $user = $request->user();
    if (!$user) { abort(401); }                    // Belum login → TOLAK
    if ($user->role->value !== $role) { abort(403); } // Role tidak cocok → TOLAK
    return $next($request);                         // Lolos → LANJUT
}
```

Penerapan di [web.php] (file:///Users/mac/Downloads/perpus-lsp-main/routes/web.php):

```php
Route::middleware(['auth', 'role:admin'])->group(...);  // Hanya admin
Route::middleware(['auth', 'role:siswa'])->group(...);  // Hanya siswa
```

Didaftarkan di [bootstrap/app.php](file:///Users/mac/Downloads/perpus-lsp-main/bootstrap/app.php): `'role' => RoleMiddleware::class`

## ✏️ Soal 10: Alur login hingga redirect ke dashboard sesuai role




Kode di [LoginController.php] (file:///Users/mac/Downloads/perpus-lsp-main/app/Http/Controllers/Auth/LoginController.php): `Auth::attempt()` cek ke database → berhasil → cek `$user->role` → redirect sesuai role.

## ✏️ Soal 11: Proses peminjaman buku oleh member

Di [SiswaController.php](file:///Users/mac/Downloads/perpus-lsp-main/app/Http/Controllers/SiswaController.php), fungsi `createOrder()`:

1. Buka form (`GET`): Ambil buku yang stock > 0 → tampilkan dropdown
2. Submit (`POST`): Validasi `book_id` → cek 3 hal:
   - Akun terhubung anggota? → Jika tidak → TOLAK
   - Stock ≥ 1? → Jika tidak → TOLAK
   - Sudah pinjam buku ini? → Jika sudah → TOLAK

3. Buat order: `Orders::create([anggota_id, book_id, tanggal_pinjam = now()])` — `tanggal_kembali` otomatis NULL
4. Kurangi stock: `$book->decrement('stock')`
5. Redirect ke dashboard + pesan sukses

## ✏️ Soal 12: Apa yang terjadi saat transaksi berubah menjadi "completed"

"Completed" = `tanggal_kembali` diisi (bukan NULL lagi).

Yang terjadi:

1. ✅ `tanggal_kembali` diisi tanggal pengembalian
2. ✅ Stock buku +1 (`$book->increment('stock')`)
3. ✅ Order berpindah dari "aktif" ke "history" (query filter `whereNull` vs `whereNotNull`)
4. ✅ Pengembalian ganda dicegah (cek `$order->tanggal_kembali === null`)

Admin: di [OrderController](file:///Users/mac/Downloads/perpus-lsp-main/app/Http/Controllers/OrderController.php) `update()`. Siswa: di [SiswaController](file:///Users/mac/Downloads/perpus-lsp-main/app/Http/Controllers/SiswaController.php) `processReturn()`.

## ✏️ Soal 13: Fungsi BorrowController dan ReturnController

Project kita tidak punya file terpisah. Fungsinya digabung:

| Fungsi | Admin (OrderController) | Siswa (SiswaController) |
|---|---|---|
| **Pinjam** | `create()` | `createOrder()` |
| **Kembalikan** | `update()` (isi tanggal_kembali) | `processReturn()` |
| **Lihat** | `index()` | `dashboard()`, `history()`, `returnBook()` |

## ✏️ Soal 14: Peran `Route::resource` dan kelebihannya

`Route::resource('books', BookController::class)` = shortcut untuk 7 route CRUD (index, create, store, show, edit, update, destroy).

Project kita TIDAK menggunakan `Route::resource`, melainkan route manual di [web.php](file:///Users/mac/Downloads/perpus-lsp-main/routes/web.php).

Kelebihan `Route::resource`:

1. Ringkas → 1 baris vs 7 baris
2. Standar RESTful → Mengikuti konvensi
3. Named routes otomatis → `books.index`, `books.create`, dll
4. Mudah di-maintain

## ✏️ Soal 15: Fungsi AuthController dalam autentikasi

Di project kita dibagi ke 3 controller:

1. [LoginController](file:///Users/mac/Downloads/perpus-lsp-main/app/Http/Controllers/Auth/LoginController.php): `show()` tampilkan form, `login()` proses login + redirect berdasarkan role
2. [RegisterController](file:///Users/mac/Downloads/perpus-lsp-main/app/Http/Controllers/Auth/RegisterController.php): `show()` tampilkan form, `register()` validasi + buat anggota + buat user siswa
3. [LogoutController](file:///Users/mac/Downloads/perpus-lsp-main/app/Http/Controllers/Auth/LogoutController.php): `logout()` keluar + hapus session + redirect ke login

## ✏️ Soal 16: Pentingnya validasi data dan contohnya

Validasi = "pengecekan kualitas" sebelum data masuk database.

Contoh di project:

| Controller | Rule | Penjelasan |
|---|---|---|
| RegisterController | `'nis' => 'required\|numeric\|unique:anggotas'` | Wajib, angka, unik |
| RegisterController | `'password' => 'required\|min:8'` | Wajib, minimal 8 karakter |
| BookController | `'nama' => 'required\|unique:books,nama,'.$id` | Unik kecuali milik sendiri |
| OrderController | `'book_id' => 'required'` | Wajib pilih buku |
| SiswaController | `'book_id' => 'required\|exists:books,id'` | Wajib + harus ada di tabel books |

## ✏️ Soal 17: Fungsi Blade Template

Blade = bahasa template Laravel untuk menulis HTML + PHP dengan syntax bersih.

| Syntax | Fungsi | Contoh |
|---|---|---|
| `{{ $var }}` | Tampilkan data (auto-escape XSS) | `{{ $book->nama }}` |
| `@extends` | Warisi template induk | `@extends('admin.layout')` |
| `@section/@yield` | Definisi & tampilkan konten | Isi halaman per page |
| `@foreach` | Looping | `@foreach($books as $book)` |
| `@if/@else` | Kondisi | `@if($errors->any())` |
| `@csrf` | Token keamanan | Di setiap form |
| `@method('PUT')` | Fake HTTP method | Untuk update/delete |

Keuntungan: Anti XSS, template inheritance, syntax bersih, performa cepat (compiled).

## ✏️ Soal 18: Perbedaan tampilan dan hak akses admin vs member

| Aspek | Admin | Siswa |
|---|---|---|
| **Layout** | `admin/layout.blade.php` | `siswa/layout.blade.php` |
| **Dashboard** | Statistik (jumlah user, buku, order) | Pinjaman aktif & history |
| **URL** | `/dashboard`, `/books`, `/orders` | `/siswa/dashboard`, `/siswa/order/create` |
| **Middleware** | `role:admin` | `role:siswa` |
| **Bisa kelola buku** | ✅ | ❌ |
| **Bisa kelola user** | ✅ | ❌ |
| **Bisa pinjam buku** | ✅ (lewat order) | ✅ (langsung) |
| **Bisa kembalikan** | ✅ (lewat edit order) | ✅ (lewat menu return) |

✏️ Soal 19: Bagaimana sistem mencegah buku yang sedang dipinjam tidak bisa dipinjam lagi

**3 Level perlindungan:**

**Level 1 — Filter tampilan:**

```php
$books = Book::where('stock', '>', 0)->get(); // Buku stock 0 TIDAK muncul
```

**Level 2 — Cek stock saat submit:**

```php
if ($book->stock < 1) { return error; } // Double check
```

**Level 3 — Cek duplikasi per siswa:**

```php
$existingOrder = Orders::where('anggota_id', $anggota->id)
    ->where('book_id', $request->book_id)
    ->whereNull('tanggal_kembali')->first();
if ($existingOrder) { return error; } // Sudah pinjam → TOLAK
```

Mekanisme stock otomatis: Pinjam → `decrement('stock')`, Kembalikan → `increment('stock')`.

[!NOTE]
Siswa berbeda TETAP bisa pinjam buku yang sama selama stock > 0. Ini benar karena perpustakaan bisa punya banyak eksemplar buku yang sama.

# 🔥 Soal Latihan UJIKOM Laravel — Level ADVANCED

Soal-soal ini dibuat lebih sulit dari soal LSP BNSP tahun lalu, mencakup topik yang kemungkinan besar muncul di UJIKOM berikutnya. Setiap soal disertai jawaban lengkap berdasarkan project `perpus-lsp-main`.

## 📌 KATEGORI A: ARSITEKTUR & KONSEP LARAVEL

### Soal 20: Jelaskan apa itu Eloquent ORM dan mengapa Laravel menggunakannya? Bandingkan penulisan query menggunakan Eloquent vs Raw SQL/Query Builder

Jawaban:

Eloquent ORM (Object-Relational Mapping) = cara Laravel "menerjemahkan" tabel database menjadi objek PHP. Setiap tabel punya 1 Model, setiap baris = 1 objek.

| Cara | Contoh Ambil Semua Buku | Kelebihan |
|---|---|---|
| **Raw SQL** | `DB::select('SELECT * FROM books')` | Cepat, kontrol penuh |
| **Query Builder** | `DB::table('books')->get()` | Aman dari SQL injection, mudah |
| **Eloquent ORM** | `Book::all()` | Paling ringkas, ada relasi, ada event |

Contoh perbandingan di project kita:

```php
// ❌ Raw SQL (ribet, rawan SQL injection)
$orders = DB::select('SELECT o.*, b.nama as book_nama, a.nama as anggota_nama
    FROM orders o
    JOIN books b ON o.book_id = b.id
    JOIN anggotas a ON o.anggota_id = a.id');

// ✅ Eloquent (bersih, aman, otomatis join lewat relasi)
$orders = Orders::with(['anggota', 'book'])->get();
```

Mengapa Laravel pakai Eloquent:

1. Readable → `Book::where('stock', '>', 0)->get()` lebih mudah dibaca
2. Relasi otomatis → `$order->book->nama` langsung ambil nama buku tanpa JOIN manual
3. Mass Assignment Protection → `$fillable` mencegah injeksi field berbahaya
4. Mutators & Casts → Otomatis konversi data (misal password di-hash)
5. Soft Deletes, Events, Scopes → Fitur lanjutan yang tidak ada di Query Builder

## Soal 21: Apa perbedaan antara `php artisan make:model Book -mcr` vs membuat Model, Migration, dan Controller secara terpisah? Kapan menggunakan masing-masing?

Jawaban:

bash

```
# Cara 1: SEMUA SEKALIGUS (1 perintah)
php artisan make:model Book -mcr
# -m = migration, -c = controller, -r = resource controller

# Cara 2: SATU PER SATU (3 perintah)
php artisan make:model Book
php artisan make:migration create_books_table
php artisan make:controller BookController
```

| Flag | Artinya | File yang dibuat |
|---|---|---|
| `-m` | Migration | `database/migrations/xxxx_create_books_table.php` |
| `-c` | Controller | app/Http/Controllers/BookController.php |
| `-r` | Resource | Controller dengan 7 method CRUD |
| `-f` | Factory | `database/factories/BookFactory.php` |
| `-s` | Seeder | database/seeders/BookSeeder.php |
| `--all` | Semua | Migration + Factory + Seeder + Controller + Policy |

Kapan pakai yang mana:

- `-mcr`: Saat mulai fitur baru dari nol (project baru)
- Terpisah: Saat menambahkan komponen ke fitur yang sudah ada

Di project kita, file-file dibuat terpisah karena kebutuhan customisasi (misal `OrderController` tidak standar resource).

Soal 22: Jelaskan apa itu Service Container dan Service Provider di Laravel. Berikan contoh bagaimana `AppServiceProvider` bisa digunakan untuk mendaftarkan service khusus

Jawaban:

**Service Container** = "kotak ajaib" Laravel yang menyimpan semua objek/class dan otomatis menyuntikkannya (inject) ke mana pun dibutuhkan.
**Service Provider** = "resep" yang memberitahu container cara membuat objek.
**Analogi: Container** = lemari es, Provider = yang mengisi lemari es dengan bahan makanan.

Contoh nyata di project kita:

php

```php
// Di bootstrap/app.php, Laravel otomatis mendaftarkan middleware:
$middleware->alias([
    'role' => \App\Http\Middleware\RoleMiddleware::class,
]);
// Container TAHU bahwa 'role' = RoleMiddleware. Saat route pakai 'role:admin',
// container otomatis membuat objek RoleMiddleware.
```

**Contoh penggunaan AppServiceProvider** (jika diterapkan di project kita):

php

```php
// app/Providers/AppServiceProvider.php
public function boot(): void
{
    // Contoh 1: Share data ke semua view
    View::share('appName', 'Perpustakaan LSP');

    // Contoh 2: Pagination default pakai Bootstrap
    Paginator::useBootstrap();

    // Contoh 3: Custom validation rule
    Validator::extend('valid_nis', function ($attribute, $value) {
        return strlen($value) === 6 && is_numeric($value);
    });
}
```

Soal 23: Apa yang dimaksud dengan Eager Loading dan Lazy Loading di Eloquent? Jelaskan problem N+1 Query dan bagaimana mengatasinya. Berikan contoh dari project ini

Jawaban:

Lazy Loading = ambil data relasi HANYA saat diakses (satu per satu → boros query)
Eager Loading = ambil data relasi SEKALIGUS di awal (1-2 query → efisien)

php

```php
// ❌ LAZY LOADING (N+1 Problem) — JANGAN BEGINI!
$orders = Orders::all(); // 1 query: SELECT * FROM orders
@foreach($orders as $order)
    {{ $order->book->nama }}     // +1 query PER order: SELECT * FROM books
WHERE id = ?
    {{ $order->anggota->nama }}  // +1 query PER order: SELECT * FROM anggotas
WHERE id = ?
@endforeach
// Jika ada 100 order → 1 + 100 + 100 = 201 QUERY! 💀
```

php

```php
// ✅ EAGER LOADING — CARA BENAR (yang dipakai di project kita)
$orders = Orders::with(['anggota', 'book'])->get();
// Query 1: SELECT * FROM orders
// Query 2: SELECT * FROM books WHERE id IN (1, 2, 3, ...)
// Query 3: SELECT * FROM anggotas WHERE id IN (1, 2, 3, ...)
// Total: HANYA 3 QUERY! 🚀
```

Contoh di OrderController.php :

php

```php
$orders = Orders::with(['anggota','book'])->get(); // ✅ Eager Loading
```

Contoh di UserController.php :

php

```php
$users = User::with('anggota')->get(); // ✅ Eager Loading
```

## 📌 KATEGORI B: DATABASE & MIGRATION LANJUTAN

Soal 24: Jelaskan apa itu Foreign Key Constraint, `cascadeOnDelete()`, dan `cascadeOnUpdate()`. Apa dampaknya terhadap integritas data? Berikan contoh dari project ini

Jawaban:

Foreign Key = kolom yang "menunjuk" ke primary key tabel lain. Menjamin data yang dirujuk PASTI ADA.

Contoh di [create_orders_table.php] (file:///Users/mac/Downloads/perpus-lsp-main/database/migrations/2026_01_21_064658_create_orders_table.php):

```php
$table->foreignId('book_id')
    ->constrained('books')       // FK menunjuk ke tabel books
    ->cascadeOnDelete()          // Kalau buku DIHAPUS → semua order buku itu
IKUT TERHAPUS
    ->cascadeOnUpdate();         // Kalau id buku BERUBAH → otomatis update di
sini juga
```

| Opsi | Artinya | Contoh dampak |
|---|---|---|
| `cascadeOnDelete()` | Hapus berantai | Buku dihapus → SEMUA order buku itu ikut terhapus |
| `cascadeOnUpdate()` | Update berantai | ID buku berubah 1 → 99, `book_id` di orders ikut jadi 99 |
| `nullOnDelete()` | Set null saat induk dihapus | Buku dihapus → `book_id` jadi NULL |
| `restrictOnDelete()` | TOLAK hapus jika masih ada anak | Buku TIDAK BISA dihapus selama masih ada order |

Dampak integritas data:

- Tanpa FK: Order bisa menunjuk `book_id = 999` padahal buku 999 tidak ada → DATA RUSAK
- Dengan FK: Database MENOLAK `book_id` yang tidak ada → DATA TERJAMIN

⚠️ Di project kita pakai `cascadeOnDelete`: menghapus buku = menghapus SEMUA history peminjaman. Dalam sistem nyata, lebih baik pakai `restrictOnDelete()` atau soft delete.

## Soal 25: Jelaskan perbedaan `timestamp()`, `dateTime()`, dan `date()`. Apa arti `useCurrent()` dan `nullable()`?

Jawaban:

| Tipe | Format | Contoh | Kapan Pakai |
|---|---|---|---|
| date() | YYYY-MM-DD | 2026-02-19 | Tanggal lahir |
| dateTime() | YYYY-MM-DD HH:MM:SS | 2026-02-19 17:30:00 | Timestamp tanpa timezone |
| timestamp() | YYYY-MM-DD HH:MM:SS | 2026-02-19 17:30:00 | Timestamp (timezone aware) |

Di project kita (tabel orders ):

```php
$table->timestamp('tanggal_pinjam')->useCurrent();
// useCurrent() = otomatis diisi waktu SEKARANG saat record dibuat

$table->timestamp('tanggal_kembali')->nullable();
// nullable() = BOLEH kosong (NULL), karena buku belum tentu sudah dikembalikan
```

`$table->timestamps()` = shortcut untuk `created_at` + `updated_at` (otomatis dikelola Laravel).

Soal 26: Apa itu Database Seeder? Bagaimana urutan seeder bisa mempengaruhi keberhasilan seeding?

Jawaban:

Seeder = file PHP untuk mengisi database dengan data awal/contoh.

Urutan di [DatabaseSeeder.php]
(file:///Users/mac/Downloads/perpus-lsp-main/database/seeders/DatabaseSeeder.php):

```php
$this->call([
    BookSeeder::class,      // 1️⃣ Buat buku dulu
    AnggotaSeeder::class,   // 2️⃣ Buat anggota dulu
    OrderSeeder::class,     // 3️⃣ Baru buat order (butuh book_id & anggota_id)
    UserSeed::class         // 4️⃣ Buat user (butuh anggota_id)
]);
```

URUTAN SANGAT PENTING! Karena Foreign Key:

- `OrderSeeder` butuh `book_id = 1` → `BookSeeder` HARUS duluan
- `OrderSeeder` butuh `anggota_id = 1` → `AnggotaSeeder` HARUS duluan

Kalau urutan salah:

SQLSTATE[23000]: Integrity constraint violation:
Cannot add or update a child row: a foreign key constraint fails

## 📌 KATEGORI C: CONTROLLER & LOGIC LANJUTAN

Soal 27: Jelaskan perbedaan `Request` biasa dan `FormRequest`. Kapan menggunakan masing-masing?

Jawaban:

| Aspek | Request biasa | `FormRequest` (custom) |
|---|---|---|
| **Validasi di mana** | Di dalam controller | Di file terpisah |
| **Reusable** | Tidak | Ya |
| **Controller jadi** | Gemuk | Ramping |
| **Authorize** | Tidak ada | Bisa cek hak akses |

**`FormRequest` di project kita — LoginRequest.php :**

```php
class LoginRequest extends FormRequest {
    public function authorize(): bool { return true; }
    public function rules(): array {
        return ['username' => ['required'], 'password' => ['required']];
    }
}
// Di controller cukup: public function login(LoginRequest $request) { ... }
```

**Request biasa — di BookController.php :**

```php
public function create(Request $request) {
    $validate = $request->validate([ 'nama' => 'required', ... ]);
}
```

Rekomendasi: `FormRequest` untuk validasi kompleks/reusable, `Request` untuk validasi sederhana.

Soal 28: Jelaskan perbedaan `findOrFail()` vs `find()` vs `firstOrFail()` vs `first()`. Apa yang terjadi jika data tidak ditemukan?

Jawaban:

| Method | Jika Data TIDAK Ada | Response |
|---|---|---|
| `find($id)` | Return `null` | Tidak ada error |
| `findOrFail($id)` | Throw `ModelNotFoundException` → **404** | Halaman 404 |
| `first()` | Return `null` | Tidak ada error |
| `firstOrFail()` | Throw `ModelNotFoundException` → **404** | Halaman 404 |

Di project kita — BookController.php :

```php
$book = Book::findOrFail($id); // Buku tidak ada → otomatis 404
```

Di SiswaController.php :

```php
$existingOrder = Orders::where(...)->first(); // Bisa null → lanjut proses
```

Soal 29: Jelaskan perbedaan `redirect()`, `redirect()->back()`, dan `redirect()->route()`. Apa fungsi `->with()` dan `->withErrors()`?

| Method | Fungsi |
| --- | --- |
| `redirect('/path')` | Pindah ke URL tertentu |
| `redirect()->back()` | Kembali ke halaman sebelumnya |
| `redirect()->route('name')` | Pindah ke named route |

**`->with()`** = flash message (sekali muncul):

```php
return redirect('/books')->with('success', 'Berhasil update data');
// Di view: {{ session('success') }} → muncul sekali, refresh → hilang
```

**`->withErrors()`** = kirim error ke form:

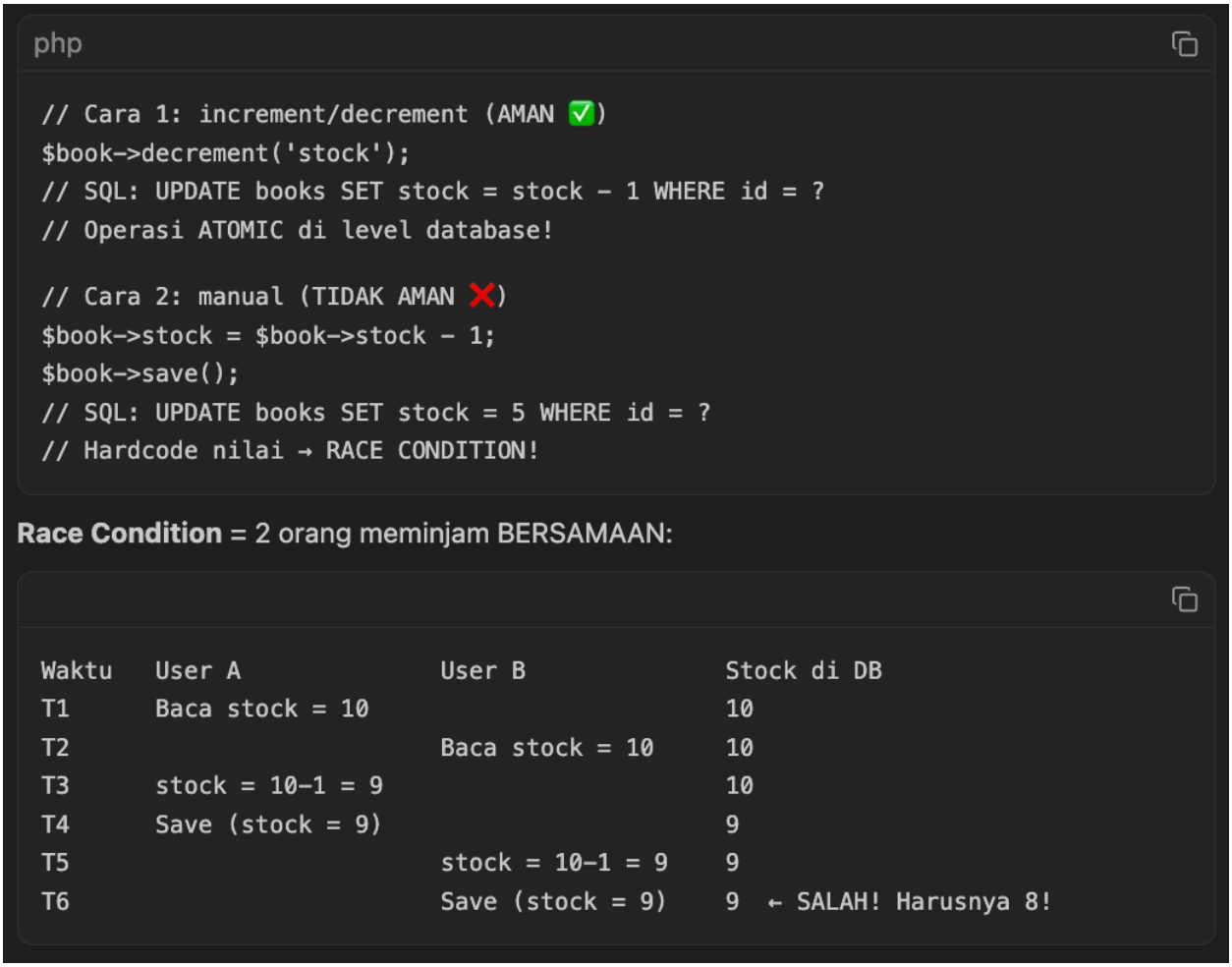
```php
return redirect()->back()->withErrors(['book_id' => 'Buku tidak tersedia']);
```

**`->withInput()`** = kembalikan data input (tidak perlu isi ulang form):

```php
return back()->withInput($request->except('password'));
```

Soal 30: Mengapa `$book->decrement('stock')` lebih aman daripada `$book->stock = $book->stock - 1; $book->save()`?

Jawaban:



```php
// Cara 1: increment/decrement (AMAN ✅)
$book->decrement('stock');
// SQL: UPDATE books SET stock = stock - 1 WHERE id = ?
// Operasi ATOMIC di level database!

// Cara 2: manual (TIDAK AMAN ❌)
$book->stock = $book->stock - 1;
$book->save();
// SQL: UPDATE books SET stock = 5 WHERE id = ?
// Hardcode nilai → RACE CONDITION!
```

**Race Condition** = 2 orang meminjam BERSAMAAN:

| Waktu | User A | User B | Stock di DB |
|---|---|---|---|
| T1 | Baca stock = 10 | | 10 |
| T2 | | Baca stock = 10 | 10 |
| T3 | stock = 10-1 = 9 | | 10 |
| T4 | Save (stock = 9) | | 9 |
| T5 | | stock = 10-1 = 9 | 9 |
| T6 | | Save (stock = 9) | 9 ← SALAH! Harusnya 8! |

Dengan `decrement()`, database menjalankan `stock = stock - 1` secara atomic → selalu benar.

## 📌 KATEGORI D: KEAMANAN & MIDDLEWARE

Soal 31: Jelaskan apa itu CSRF dan mengapa Laravel mewajibkan `@csrf` di form. Apa yang terjadi tanpa `@csrf`?

Jawaban:
CSRF = serangan dimana website lain mengirim request ke website kita atas nama user yang login.
Cara `@csrf` melindungi:

```html
<form method="POST" action="/books/create">
    @csrf  <!-- <input type="hidden" name="_token" value="abc123xyz"> -->
</form>
```

- Setiap form dapat token rahasia unik per session
- Saat submit, Laravel mencocokkan token
- Website hacker TIDAK TAHU token → request DITOLAK

Tanpa `@csrf`: Error `419 | Page Expired`

Soal 32: Jelaskan perbedaan middleware `auth` dan `guest`. Bagaimana kombinasinya mengamankan aplikasi?

Jawaban:

| Middleware | Membolehkan | Menolak | Redirect ke |
|---|---|---|---|
| auth | Sudah login | Belum login | `/login` |
| `guest` | Belum login | Sudah login | Dashboard |

Di web.php :

```php
Route::middleware('guest')->group(function () {
    Route::get('/login', ...);   // Belum login → tampilkan form
});
Route::middleware(['auth', 'role:admin'])->group(function () {
    Route::get('/dashboard', ...); // Sudah login & admin → tampilkan
});
```

Tanpa `auth` → orang akses dashboard tanpa login → DATA BOCOR.
Tanpa `role` → siswa akses halaman admin → BERBAHAYA.

## Soal 33: Jelaskan Enum di PHP dan bagaimana project ini menggunakannya untuk role. Apa kelebihannya vs string biasa?

Jawaban:

Di [UserRole.php]
(file:///Users/mac/Downloads/perpus-lsp-main/app/Enums/UserRole.php):

php

```
enum UserRole: string {
    case ADMIN = 'admin';
    case SISWA = 'siswa';
}
```

| Tanpa Enum | Dengan Enum |
| --- | --- |
| `$user->role = 'adminn'` (typo lolos!) | `UserRole::ADMIN` (typo = error compile) |
| Bisa diisi apa saja | HANYA `ADMIN` atau `SISWA` |

Penggunaan:

php

```
// Migration — database hanya terima 'admin' atau 'siswa'
$table->enum('role', array_column(UserRole::cases(), 'value'));
// LoginController — cek role type-safe
if ($user->role === UserRole::SISWA) { ... }
// User Model — auto-cast string ke Enum
protected $casts = ['role' => UserRole::class];
```

## 📌 KATEGORI E: VIEW & BLADE LANJUTAN

Soal 34: Jelaskan Template Inheritance di Blade. Mengapa project ini punya 2 layout berbeda?

Jawaban:

```
layout.blade.php (INDUK)
├── @yield('title')     ← Tempat judul
├── @yield('content')   ← Tempat isi

Book/index.blade.php (ANAK)
├── @extends('admin.layout')
├── @section('title', 'Daftar Buku')
└── @section('content') ... @endsection
```

| Directive | Fungsi |
|---|---|
| `@extends('admin.layout')` | Anak mewarisi template induk |
| `@yield('content')` | Di induk: taruh konten anak DI SINI |
| `@section('content')` | Di anak: ini konten yang saya kirim |
| `@include('partials.navbar')` | Sisipkan file blade lain |

2 layout karena admin dan siswa punya tampilan, navigasi, dan akses yang berbeda.

Soal 35: Jelaskan `@method('PUT')` dan `@method('DELETE')`. Mengapa HTML form tidak bisa langsung PUT/DELETE?

Jawaban:

HTML form hanya mendukung `GET` dan `POST`. Laravel menambahkan "method spoofing"

```html
<form method="POST" action="/books/1">
    @csrf
    @method('PUT')  <!-- <input type="hidden" name="_method" value="PUT"> -->
</form>
```

Laravel membaca `_method` dan memperlakukan request seolah `PUT`, walaupun dikirim sebagai `POST`.

```php
Route::put('/books/{id}', ...);    // Butuh @method('PUT')
Route::delete('/books/{id}', ...); // Butuh @method('DELETE')
```

## 📌 KATEGORI F: ERROR HANDLING & DEBUGGING

Soal 36: Jelaskan perbedaan error 401, 403, 404, 419, 422, 500 di Laravel

Jawaban:

| Kode | Nama | Kapan Muncul di Project Kita |
|---|---|---|
| **401** | Unauthorized | Akses route `auth` tanpa login |
| **403** | Forbidden | Siswa akses `/dashboard` admin → `abort(403)` |
| **404** | Not Found | `Book::findOrFail(999)` buku tidak ada |
| **419** | Page Expired | Form tanpa `@csrf` / session expired |
| **422** | Unprocessable | Validasi gagal (API mode) |
| **500** | Server Error | Bug kode, database connection failed |

Soal 37: Jelaskan `try-catch` dan bagaimana project ini menangani error registrasi

Jawaban:

Di [RegisterController.php](file:///Users/mac/Downloads/perpus-lsp-main/app/Http/Controllers/Auth/RegisterController.php):

```php
try {
    $anggota = anggota::create($anggotaValidated);
    $user = User::create($userValidated);
    return redirect('/login')->with('success', 'Register berhasil.');
} catch (\Exception $e) {
    return back()
        ->withInput($request->except('password'))
        ->with('error', 'Terjadi kesalahan: ' . $e->getMessage());
}
```

Pentingnya:

- User-friendly → Pesan error jelas, bukan halaman error jelek
- Data aman → `withInput()` kembalikan data yang sudah diisi
- Tidak crash → Aplikasi tetap berjalan meski ada error

## 📌 KATEGORI G: SKENARIO & STUDI KASUS (Level Tertinggi)

Soal 38: Jika diminta menambahkan fitur "Denda Keterlambatan", jelaskan perubahan di Migration, Model, Controller, dan View

Jawaban:

**1. Migration:**

```php
Schema::table('orders', function (Blueprint $table) {
    $table->integer('denda')->default(0);
    $table->integer('max_hari_pinjam')->default(7);
});
```

**2. Model Orders — hitung denda:**

```php
protected $fillable = [..., 'denda', 'max_hari_pinjam'];

public function hitungDenda(): int {
    if (!$this->tanggal_kembali) return 0;
    $hari = Carbon::parse($this->tanggal_pinjam)->diffInDays($this->tanggal_kembali);
    return max(0, $hari - $this->max_hari_pinjam) * 1000;
}
```

**3. Controller — saat pengembalian:**

```php
$order->denda = max(0, $hariPinjam - 7) * 1000;
```

**4. View:**

```html
<td>Rp {{ number_format($order->denda, 0, ',', '.') }}</td>
```

Soal 39: Bagaimana menambahkan fitur "Pencarian Buku" (search)? Jelaskan perubahan di Route, Controller, View

Jawaban:

Route: Tidak perlu berubah (tetap `GET /books`)

Controller:

```php
public function index(Request $request) {
    $query = Book::query();
    if ($request->filled('search')) {
        $search = $request->search;
        $query->where(function($q) use ($search) {
            $q->where('nama', 'like', "%{$search}%")
              ->orWhere('pengarang', 'like', "%{$search}%");
        });
    }
    $books = $query->paginate(10);
    return view('admin.Book.index', compact('books'));
}
```

**View:**

```html
<form method="GET" action="/books">
    <input name="search" value="{{ request('search') }}" placeholder="Cari buku...">
    <button type="submit">Cari</button>
</form>
{{ $books->links() }}
```

Soal 40: Jelaskan Soft Delete di Laravel. Jika diterapkan di model Book, apa yang berubah?

Jawaban:

Soft Delete = "hapus pura-pura". Data ditandai `deleted_at`, bukan benar-benar hilang.

```php
// Migration
$table->softDeletes(); // Tambah kolom deleted_at

// Model
use SoftDeletes;
```

| Operasi | Tanpa Soft Delete | Dengan Soft Delete |
|---|---|---|
| destroy($id) | Data HILANG permanen | `deleted_at` diisi (data masih ada) |
| `Book::all()` | Semua data | Hanya yang `deleted_at = NULL` |
| Lihat yang dihapus | Tidak bisa | `Book::withTrashed()->get()` |
| Kembalikan | Tidak bisa | `$book->restore()` |

Soal 41: Jelaskan Pagination di Laravel. Perbedaan `paginate()`, `simplePaginate()`, `cursorPaginate()`?

Jawaban:

| Method | Tampilan | Performa |
|---|---|---|
| `paginate(10)` | Nomor halaman (1, 2, 3...) | Count + Offset |
| `simplePaginate(10)` | Hanya Prev/Next | Tanpa Count, lebih cepat |
| `cursorPaginate(10)` | Prev/Next (cursor) | Tercepat, untuk data besar |

```php
// Controller
$books = Book::paginate(10);
// View
{{ $books->links() }}  // Otomatis render tombol pagination
```