

**PENGEMBANGAN APLIKASI KATALOG FILM BERBASIS WEB (MOVIE
CATALOG APP)**



Disusun Oleh:

Muhammad Fawaz H. F1D002310079

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS MATARAM
2025**

BAB I

PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi web modern telah mendorong meningkatnya kebutuhan akan sistem informasi yang mampu menyajikan data secara terstruktur, interaktif, dan mudah diakses. Salah satu bentuk kebutuhan tersebut dapat dilihat pada pengelolaan dan penyajian informasi film, seperti yang diterapkan oleh platform populer semacam The Movie Database (TMDB).

Namun, platform berskala besar tersebut memiliki kompleksitas sistem yang tinggi dan tidak selalu sesuai untuk kebutuhan pembelajaran maupun pengembangan aplikasi berskala kecil. Oleh karena itu, diperlukan sebuah aplikasi katalog film yang lebih sederhana, terfokus pada fitur inti, namun tetap menerapkan konsep pengembangan web modern.

Aplikasi Movie Catalog App dikembangkan sebagai solusi untuk mengelola data film berdasarkan genre, menampilkan detail film, serta menyediakan fitur ulasan (review) oleh pengguna. Aplikasi ini dirancang dengan pendekatan client-server, menggunakan REST API pada sisi backend dan Single Page Application (SPA) pada sisi frontend, sehingga dapat menjadi sarana penerapan konsep Fullstack Web Modern.

1.2 Rumusan Masalah

Berdasarkan latar belakang tersebut, rumusan masalah dalam pengembangan aplikasi ini adalah sebagai berikut:

1. Bagaimana membangun aplikasi katalog film berbasis web dengan arsitektur client-server?
2. Bagaimana menerapkan autentikasi dan otorisasi pengguna berbasis JSON Web Token (JWT)?
3. Bagaimana mengelola data film, genre, dan review menggunakan REST API dan database relasional?

1.3 Batasan Masalah

Agar pengembangan aplikasi tetap terfokus, ditetapkan batasan masalah sebagai berikut:

1. Aplikasi hanya mencakup pengelolaan data film, genre, dan review.

2. Sistem memiliki dua peran pengguna, yaitu Admin dan User.
3. Admin bertugas mengelola data genre dan film.
4. User hanya dapat melihat data film dan memberikan review.
5. Backend menggunakan Node.js dan Express.js, frontend menggunakan Vue.js 3, dan database menggunakan MySQL.
6. Autentikasi dilakukan menggunakan JWT, tanpa integrasi pihak ketiga.

1.4 Tujuan Penelitian

1.4.1 Tujuan Akademis

Tujuan akademis dari pengembangan aplikasi ini adalah untuk memenuhi tugas Ujian Akhir Semester (UAS) pada mata kuliah Pemrograman Web Lanjut, serta sebagai implementasi konsep REST API, autentikasi, dan SPA.

1.4.2 Tujuan Praktis

TTujuan praktis dari aplikasi ini adalah menyediakan sistem katalog film yang dapat digunakan untuk mengelola dan menyajikan informasi film secara terstruktur dan mudah diakses oleh pengguna.

BAB II

ANALISIS DAN PERANCANGAN SISTEM

2.1 Analisis Kebutuhan Sistem

Tahap analisis kebutuhan dilakukan untuk memastikan fitur-fitur yang dibangun pada aplikasi Movie Catalog dapat mendukung pengelolaan data film dan review secara tepat. Karena aplikasi memiliki peran Admin dan User, kebutuhan sistem dibagi menjadi spesifikasi fungsional dan non-fungsional agar fitur berjalan konsisten dan aman.

2.1.1 Kebutuhan Fungsional

Kebutuhan fungsional mendefinisikan interaksi yang harus bisa dilakukan oleh sistem untuk setiap aktor (Admin dan User). Berdasarkan alur aplikasi dan API yang dibangun, kebutuhan fungsional meliputi:

1. Autentikasi & Role-Based Access
 - Sistem menyediakan login menggunakan email dan password terenkripsi.
 - Sistem membedakan hak akses Admin dan User.
 - Endpoint tertentu (CRUD genre & movie) hanya bisa diakses Admin.
2. Manajemen Genre (Admin)
 - Admin dapat menambah, mengubah, melihat, dan menghapus data movie.
 - Movie terhubung ke genre serta menyimpan tanggal rilis, durasi, dan poster.
3. Manajemen Movie (Admin)
 - Admin dapat menambah, mengubah, melihat, dan menghapus data movie.
 - Movie terhubung ke genre serta menyimpan tanggal rilis, durasi, dan poster.
4. Akses Katalog (User & Public)
 - Pengguna umum dapat melihat daftar genre dan daftar film.
 - Pengguna dapat mencari film berdasarkan judul dan filter berdasarkan genre.
5. Manajemen Review (User)
 - User dapat menambahkan review pada movie yang dipilih.
 - User hanya dapat mengedit dan menghapus review miliknya sendiri.
 - Admin dapat menghapus review jika diperlukan.

2.1.2 Kebutuhan Non-Fungsional

Kebutuhan non-fungsional berkaitan dengan kualitas sistem, teknologi yang digunakan, serta batasan teknis yang harus dipenuhi.

1. Integritas Data
 - Relasi antar tabel dijaga dengan foreign key (movie → genre, review → user & movie).
 - Validasi input mencegah penyimpanan data kosong
2. Keamanan
 - Password disimpan dalam bentuk hash menggunakan bcrypt.
 - Autentikasi API menggunakan JWT untuk setiap request yang protected.
3. Ketersediaan (Availability)
 - Aplikasi berbasis web sehingga dapat diakses lewat browser tanpa instalasi tambahan.
 - API menggunakan CORS agar frontend dapat mengakses backend.

2.2 Perancangan Alur Kerja

Perancangan alur kerja bertujuan untuk memastikan pengelolaan data film dan review berjalan terstruktur serta membatasi akses sesuai peran.

1. Fase Inisiasi (Akun & Akses) Admin membuat akun pengguna (atau pengguna melakukan register). Sistem menyimpan akun dan menerapkan role-based access agar admin memiliki hak kelola, sedangkan user hanya dapat mengakses katalog dan review.
2. Fase Katalog (Akses Publik & Pencarian) User membuka halaman Movies → Sistem menampilkan daftar film dan genre → User dapat melakukan filter berdasarkan genre atau mencari judul film.
3. Fase Pengelolaan Data (Admin) Admin login → Menu Admin → Tambah/Ubah/Hapus genre dan movie → Data tersimpan ke database dan langsung tersedia untuk user.
4. Fase Review (User) User login → Buka detail movie → Input skor dan komentar review → Review tersimpan dan tampil di detail movie. Jika user mengubah/menghapus review, sistem memastikan hanya pemilik review (atau admin) yang bisa melakukan aksi.

2.3 Perancangan Basis Data

Perancangan basis data menggunakan model relasional dengan MySQL (query SQL langsung, tanpa ORM). Struktur tabel mengikuti kebutuhan katalog film dan review. Tabel utama meliputi:

1. Tabel Users

Menyimpan identitas pengguna (id, name, email, password_hash, role). Kolom role menggunakan ENUM ('admin', 'user') untuk pembatasan akses.

2. Tabel Genres

Menyimpan data genre film (id, name, description, popularity_score, is_active, created_at, updated_at). Dipakai untuk mengelompokkan film dan mendukung fitur filter.

3. Tabel Movies

Menyimpan data film (id, genre_id, title, synopsis, release_date, duration_minutes, poster_url, created_at, updated_at).genre_id menjadi Foreign Key ke tabel genres.

4. Tabel Reviews

Menyimpan ulasan pengguna (id, movie_id, user_id, score, comment, is_spoiler, review_date, created_at, updated_at).movie_id mengacu ke movies dan user_id mengacu ke users sebagai foreign key.

2.4 Perancangan Antarmuka (*User Interface*)

Antarmuka dirancang minimalis dan mudah digunakan agar user dapat mencari film dan memberi review dengan cepat.

1. Dashboard (Admin)Menampilkan ringkasan data menggunakan card (jumlah film, jumlah genre) agar mudah dibaca cepat.
2. Movies List (Public)Menampilkan grid kartu film berisi poster, judul, genre, dan tombol view. Tersedia form filter berdasarkan genre dan pencarian judul.
3. Movie Detail + ReviewMenampilkan detail film lengkap serta daftar review. User yang login dapat menambahkan, mengedit, atau menghapus review miliknya.
4. Form Admin (Create/Edit Movie)Form sederhana berisi input judul, genre, sinopsis, tanggal rilis, durasi, dan URL poster. Fokus pada efisiensi input data.

BAB III

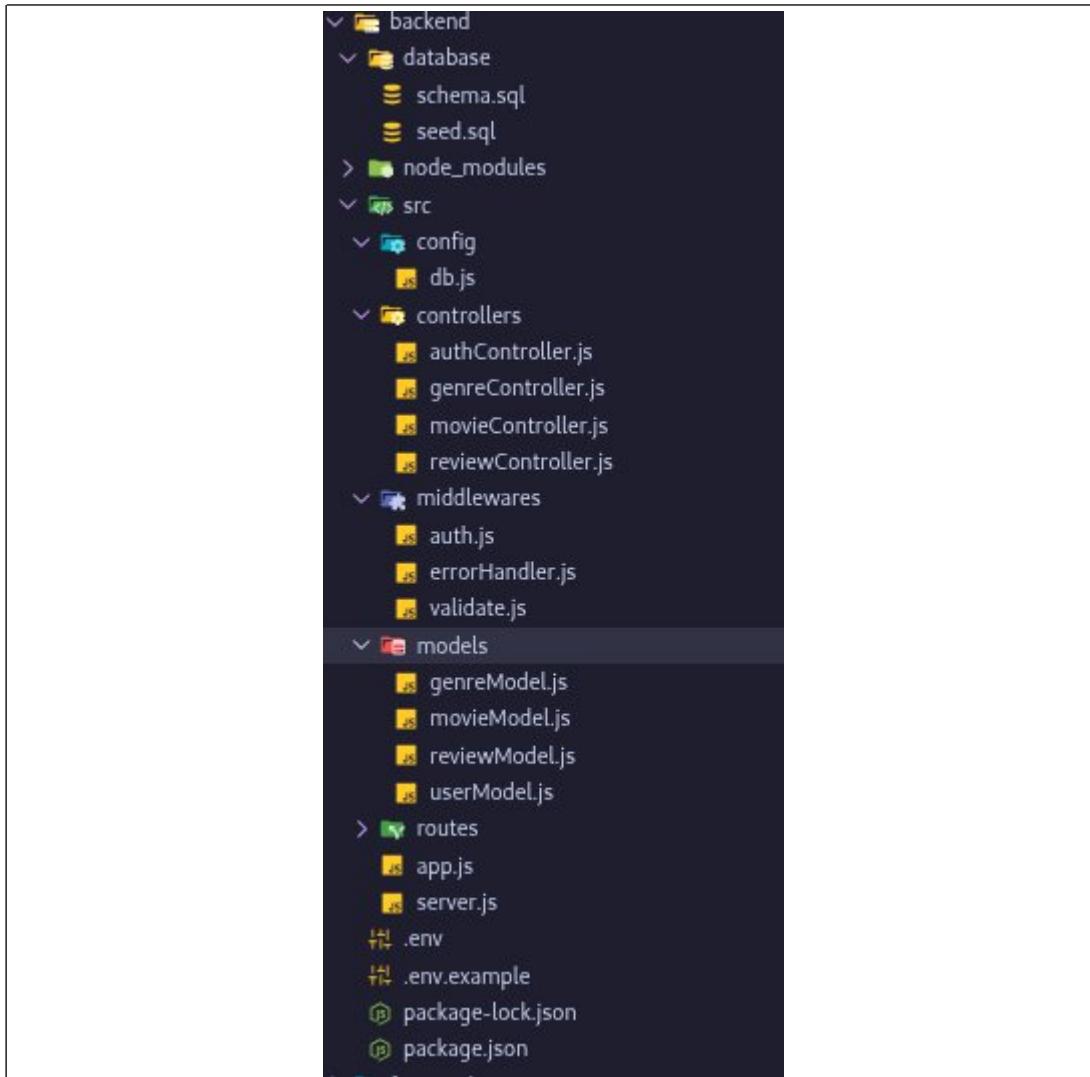
IMPLEMENTASI DAN EVALUASI SISTEM

3.1 Implementasi *Backend*

Sisi server aplikasi Movie Catalog dibangun menggunakan Node.js dengan Express.js. Arsitektur backend menerapkan pola MVC (Model–View–Controller) agar logika bisnis, routing, dan akses database terpisah dengan jelas. Pengelolaan data menggunakan MySQL melalui library mysql2 (promise) dan query SQL langsung (tanpa ORM).

3.1.1 Struktur Folder *Backend*

Pengorganisasian kode bersifat hierarkis untuk menjaga keterbacaan dan memudahkan debugging:



Gambar 3.1 Struktur Folder Backend

1. **config/**: Menyimpan konfigurasi utama, khususnya db.js untuk koneksi database MySQL.
2. **controllers/**: Merupakan inti dari logika:
 - authController.js → login & register
 - genreController.js → CRUD genre
 - movieController.js → CRUD movie + detail
 - reviewController.js → CRUD review
3. **middleware/**: Berisi fungsi perantara untuk validasi dan keamanan:
 - auth.js → verifikasi JWT + role admin
 - validate.js → cek body request kosong
 - errorHandler.js → global error handler JSON
4. **routes/**: routes/Definisi endpoint API:
authRoutes.js, genreRoutes.js, movieRoutes.js, reviewRoutes.js
Semua rute disatukan di app.js.
5. **app.js**: Inisialisasi Express, middleware global, dan rute utama.
6. **Server.js**: Entry point untuk menjalankan server.

3.1.2 Konfigurasi Environment (.env)

Untuk menjaga keamanan, data sensitif disimpan di file .env.

Konfigurasi utama meliputi:

```
PORt=4000
DB_HOST=localhost
DB_USER=pwsflix
DB_PASSWORD=pawas
DB_NAME=myflix
DB_PORT=3306
JWT_SECRET=supersecret
JWT_EXPIRES_IN=1d
```

3.1.3 Middleware Autentikasi dan Otorisasi

Sistem keamanan API dibangun dengan JSON Web Token (JWT). Middleware auth.js bertugas memeriksa token pada header Authorization di setiap request yang bersifat protected. Jika token valid, informasi user disimpan ke req.user agar bisa dipakai di controller.

Selain itu, terdapat middleware `isAdmin` untuk menerapkan Role-Based Access Control (RBAC). Middleware ini memastikan endpoint administratif (CRUD genre dan movie) hanya bisa diakses oleh pengguna dengan role admin.

Contoh kode middleware (`auth.js`):

```
const import jwt from "jsonwebtoken";
import { findById } from "../models/userModel.js";

export const auth = async (req, res, next) => {
    const header = req.headers.authorization;
    if (!header) return res.status(401).json({ message: "No token provided" });
    const token = header.split(" ")[1];
    try {
        const decoded = jwt.verify(token, process.env.JWT_SECRET);
        const user = await findById(decoded.id);
        if (!user) return res.status(401).json({ message: "User not found" });
        req.user = user;
        next();
    } catch (err) {
        return res.status(401).json({ message: "Invalid token" });
    }
};

export const isAdmin = (req, res, next) => {
    if (req.user?.role !== "admin") {
        return res.status(403).json({ message: "Admin only" });
    }
    next();
};
```

3.1.4 Implementasi Logika Bisnis (Controller)

Logika bisnis dipusatkan di layer controller. Contoh pada `reviewController.js`, alur yang diterapkan meliputi:

- Menerima input review (`score`, `comment`, `is_spoiler`) dari `req.body`.
- Mengambil user login dari `req.user` untuk mengikat review dengan pemiliknya.
- Menyimpan review ke database melalui model `reviewModel.js`.
- Membatasi update/delete review hanya oleh pemilik review atau admin.

Contoh kode controller (`reviewController.js`):

```
import {
    getReviewsByMovie,
    createReview,
    updateReview,
    deleteReview,
    getReviewById,
} from "../models/reviewModel.js";
```

```

export const listReviews = async (req, res, next) => {
  try {
    const reviews = await getReviewsByMovie(req.params.movieId);
    res.json(reviews);
  } catch (err) {
    next(err);
  }
};

export const createReviewController = async (req, res, next) => {
  try {
    const review = await createReview(req.params.movieId, {
      ...req.body,
      user_id: req.user.id,
    });
    res.status(201).json(review);
  } catch (err) {
    next(err);
  }
};

export const updateReviewController = async (req, res, next) => {
  try {
    const existing = await getReviewById(req.params.id);
    if (!existing) return res.status(404).json({ message: "Review not found" });
    if (existing.user_id !== req.user.id && req.user.role !== "admin") {
      return res.status(403).json({ message: "Not authorized" });
    }
    const review = await updateReview(req.params.id, req.body);
    res.json(review);
  } catch (err) {
    next(err);
  }
};

export const deleteReviewController = async (req, res, next) => {
  try {
    const existing = await getReviewById(req.params.id);
    if (!existing) return res.status(404).json({ message: "Review not found" });
    if (existing.user_id !== req.user.id && req.user.role !== "admin") {
      return res.status(403).json({ message: "Not authorized" });
    }
    await deleteReview(req.params.id);
    res.json({ message: "Review deleted" });
  } catch (err) {
    next(err);
  }
};

```

3.2 Implementasi *Frontend* (Vue.js 3)

Antarmuka pengguna dikembangkan sebagai Single Page Application (SPA) menggunakan Vue.js 3 dan Vite. Hal ini memberikan pengalaman navigasi yang cepat tanpa reload halaman serta memudahkan pengelolaan state dan routing.

3.2.1 Routing dan Navigasi

Navigasi antar halaman dikelola oleh Vue Router. Rute dipisah menjadi area publik (Movies List, Movie Detail, Login/Register) dan area privat (Dashboard, Form Admin). Route Guard diterapkan untuk membatasi akses halaman tertentu jika user belum login atau bukan admin. Contoh konfigurasi route guard:

```
router.beforeEach((to, from, next) => {
  const auth = useAuthStore();

  if (to.meta.requiresAuth && !auth.isLoggedIn) {
    return next({ name: "login" });
  }
  if (to.meta.adminOnly && !auth.isAdmin) {
    return next({ name: "home" });
  }
  next();
});
```

3.2.2 Integrasi API

Komunikasi dengan backend menggunakan Axios. Token JWT disisipkan otomatis ke setiap request melalui interceptor, sehingga user tetap terautentikasi saat mengakses endpoint protected.

Contoh interceptor Axios:

```
api.interceptors.request.use((config) => {
  const auth = useAuthStore();

  if (auth?.token) {
    config.headers.Authorization = `Bearer ${auth.token}`;
  }

  return config;
});
```

3.2.3 Komponen Manajemen Data

Frontend dibuat dengan pendekatan component-based menggunakan Vue 3, sehingga setiap fitur terpisah dalam komponen/store masing-masing.

1. Komponen Movie & GenreMenampilkan daftar film dan genre, mendukung filter dan pencarian judul.
2. Komponen ReviewUser dapat menambahkan, mengedit, dan menghapus review miliknya pada detail movie. Admin memiliki akses lebih luas untuk mengelola review.
3. Komponen Admin (Form Movie)Form khusus admin untuk membuat dan mengedit movie. Data diambil dari API dan langsung diperbarui di daftar

BAB IV

PENUTUP

4.1 Kesimpulan

Berdasarkan seluruh tahapan pengembangan aplikasi Movie Catalog (Mini TMDB)—mulai dari analisis kebutuhan, perancangan database, implementasi backend & frontend, hingga pengujian endpoint—dapat ditarik beberapa kesimpulan berikut:

1. Aplikasi berhasil dibangun sebagai katalog film modern dengan dua peran utama (Admin dan User) yang terstruktur, sehingga alur pengelolaan data dan interaksi pengguna berjalan jelas dan terkontrol.
2. Seluruh fitur inti yang dirancang telah berjalan sesuai kebutuhan, mencakup:
 - Keamanan Akses: Autentikasi JWT dan Role-Based Access Control memastikan endpoint admin terlindungi.
 - Manajemen Data: Admin dapat mengelola genre dan movie secara penuh (CRUD), sementara user dapat mengakses katalog secara publik.
 - Interaksi Pengguna: Fitur review memungkinkan user memberikan penilaian film serta mengelola ulasan miliknya sendiri.
3. Pemilihan stack Node.js + Express.js + MySQL di backend dan Vue 3 + Vite di frontend menghasilkan sistem yang responsif, modular, dan mudah dipelihara. Pemisahan frontend-backend membuat pengembangan lebih terstruktur..
4. Pengujian API menunjukkan alur sistem berjalan stabil, khususnya pada proses autentikasi, akses publik, dan perlindungan endpoint admin serta review berbasis kepemilikan.

4.2 Saran

Untuk peningkatan kualitas dan skalabilitas di tahap berikutnya, berikut beberapa rekomendasi:

1. Peningkatan Keamanan: Tambahkan refresh token dan mekanisme rate-limiting untuk mencegah brute force login dan penyalahgunaan token.
2. Fitur Favorit & Watchlist: Menambah fitur penyimpanan film favorit agar interaksi user lebih personal.
3. Analitik Admin: Dashboard admin dapat diperluas dengan grafik/statistik (jumlah review, rating rata-rata, genre paling populer).

4. Optimasi Infrastruktur: Gunakan Docker untuk deployment agar lebih konsisten lintas environment, serta siapkan CI/CD untuk otomatisasi build dan testing.