LAB 2 REPORT: LAYERED ARCHITECTURE DESIGN

Project Name: Movie Ticket Booking System

Member Name: Ngô Đình Quyền - 23010485

Trần Hữu Kiên - 23010258

---

1. LAYER DEFINITIONS & RESPONSIBILITIES

This section defines the four logical layers of the system architecture, adhering to the strict layered pattern where each layer only interacts with the layer directly below it[2].

Layer	Purpose/Responsibility	Output/Artifacts
1. Presentation Layer (UI)	Handles HTTP requests, user authentication, input parsing, and formatting the JSON response for the client[3].	**Controllers** (e.g., `MovieController`, `TicketController`)
2. Business Logic Layer (Service)	Encapsulates core business rules, validations (e.g., checking seat availability), and transaction management. It orchestrates data access[4].	**Services** (e.g., `MovieService`, `BookingService`)
3. Persistence Layer (Data Access)	Responsible for mapping business objects to database entities and executing CRUD (Create, Read, Update, Delete) operations. It abstracts the SQL details[5].	**Repositories** (e.g., `MovieRepository`, `TicketRepository`)
4. Data Layer	The physical storage system that holds the raw data tables[6].	**Database Schema**

Layer	Purpose/Responsibility	Output/Artifacts
		(MySQL/PostgreSQL Tables)

Data Flow Description (View Movie Details)

The strict flow of control for a user requesting to view details of a specific movie is as follows[7]:

1. **Client Request:** User requests `GET /movies/123`.
2. **Layer 1 (Presentation):** `MovieController` receives the request.
3. **Layer 2 (Business Logic):** Controller calls `MovieService`.
4. **Layer 3 (Persistence):** Service calls `MovieRepository`.
5. **Layer 4 (Data):** Repository executes SQL query on the Database.
6. **Response:** Data returns up the stack (Repo $\rightarrow$ Service $\rightarrow$ Controller $\rightarrow$ Client JSON Response)[8].

---

2. COMPONENT IDENTIFICATION (Feature: View Movie Details)

This section breaks down the "View Movie Details" feature into concrete software components residing in the top three layers[9].

2.1. Component Roles

- **Layer 1: Presentation**
  - **Component:** `MovieController`
  - **Responsibility:** Receives the HTTP GET request for a specific movie ID, validates the input format, and delegates the processing to the `MovieService`[10].
- **Layer 2: Business Logic**
  - **Component:** `MovieService`
  - **Responsibility:** Receives the ID from the controller. It enforces business rules (e.g., ensuring the movie is currently scheduled/active) before asking the repository for data[11].
- **Layer 3: Persistence**
  - **Component:** `MovieRepository`
  - **Responsibility:** Constructs the specific database query (e.g., `SELECT * FROM movies WHERE id = ?`) to retrieve the movie entity[12].

2.2. Interface Definitions

To decouple the layers, we define the following interfaces[13]:

- **Service Interface (Provided to Layer 1):**

  Java

  ```
  // Defines what the Service layer offers to the Controller
  public interface IMovieService {
      MovieDTO getMovieDetails(String movieId);
  }
  ```

  14

- **Repository Interface (Provided to Layer 2):**

  Java

  ```
  // Defines what the Repository layer offers to the Service
  public interface IMovieRepository {
      MovieEntity findById(String movieId);
  }
  ```

  15

---

3. COMPONENT DIAGRAM MODELING

The following diagram illustrates the logical view of the architecture, showing components, interfaces (Lollipop/Socket notation), and strict downward dependencies.