# ****Movie Catalog System - Understanding Document****

## ****1. Introduction****

This document provides a comprehensive overview of the Movie Catalog System, outlining its purpose, key components, and functionality. The system is designed to manage a movie catalog where movies can be stored, updated, rated, and associated with directors. The system also provides the ability to retrieve movies based on different criteria.

## ****2. System Overview****

The Movie Catalog System consists of three main entities: **Movies**, **Directors**, and **Ratings**. The system allows users to:

Create, update, and delete movies and directors.

Rate movies and update those ratings.

Retrieve movies based on various filters (e.g., by director, rating, etc.).

Support pagination for movie lists.

The system's backend is implemented using Java and Spring Boot, utilizing RESTful services to handle the various operations on the entities. The data is transferred through Data Transfer Objects (DTOs) such as MovieDTO, DirectorDTO, and RatingDTO.

## ****3. Key Components and Responsibilities****

### ****3.1 Movie Management****

Handles operations related to movies, including creating, updating, retrieving, listing, and deleting movie records along with facility to fetch movies by director and search movies with rating threshold. Each movie is associated with a director and may have ratings.

· **Controller Layer**: Exposes RESTful endpoints to create, update, retrieve (single and all), and delete movies.

· **Service Layer**: Contains business logic for handling movie operations including validation, data transformation, and interactions with other services (like Director or Rating).

· **Repository Layer**: Interfaces with the database to persist and retrieve movie records using JPA.

· **Entity Layer**: Defines the Movie entity, including fields like id, title, genre, releaseDate, and its relationship to a Director and Ratings.

### ****3.2 Director Management****

Manages director data by providing endpoints to add, update, retrieve, list, and delete directors. Directors are linked to one or more movies.

· **Controller Layer**: Provides endpoints to perform CRUD operations on directors.

· **Service Layer**: Manages business rules such as preventing deletion of directors linked to existing movies and converting between DTOs and entities.

· **Repository Layer**: Handles direct interaction with the Director table in the database.

· **Entity Layer**: Defines the Director entity with attributes like id, name, and relationship mappings to Movie entities.

### ****3.3 Rating Management****

Enables users to rate movies by submitting a score and optional review. Supports full CRUD operations for ratings, tied to specific movies.

· **Controller Layer**: Offers endpoints to create, update, retrieve, list, and delete movie ratings.

· **Service Layer**: Implements business rules such as validating the rating score (0–10), associating ratings with the correct movie, and converting between entities and DTOs.

· **Repository Layer**: Performs database operations for the Rating entity, including custom queries if needed.

· **Entity Layer**: Defines the Rating entity with fields like id, rating, review, and its association to a specific Movie.

## ****5. Data Flow****

The data flow within the system occurs through interaction with the following components:

**Controller Layer**: Receives requests from the client, processes them, and calls the appropriate service method.

**Service Layer**: Contains the core business logic, calling the necessary methods in the repository or database to perform operations.

**Repository Layer**: Interacts with the database to persist and retrieve entities (such as Movie, Director, and Rating).

## ****6. Error Handling****

The system handles various errors, including:

**Invalid Input**: When an invalid MovieDTO, DirectorDTO, or RatingDTO is provided, a 400 Bad Request error is returned.

**Not Found**: When a movie, director, or rating is requested by ID and not found, a 404 Not Found error is returned.

**Server Errors**: If an unexpected error occurs on the server, a 500 Internal Server Error is returned with custom/generic message so system does not leak any stack trace.

Here is comprehensive documentation for the three classes involved in your unit test setup: RatingControllerTest, RatingController, and RatingDTO. This documentation explains their roles in test coverage, their responsibilities, and how they interconnect.

1. ****Controller and it’s API with Test Coverage****

**[Movie](https://github.com/Fazal9638/movie-catalogue-api/blob/main/Documentation/Movie%20Components.docx)**

**[Director](https://github.com/Fazal9638/movie-catalogue-api/blob/main/Documentation/Director%20Components.docx)**

**[Rating](https://github.com/Fazal9638/movie-catalogue-api/blob/main/Documentation/RatingController.docx)**

## ****8. Future Enhancements****

To keep the development simple at this stage, we have skipped implementing security. However, in the future, we can introduce role-based access control (RBAC) or JWT-based security to protect sensitive endpoints and manage user access appropriately.

## ****9. Conclusion****

The Movie Catalog System is designed to efficiently manage movies, directors, and ratings, providing a seamless experience for users to interact with the catalog. With the service layer abstracting the core business logic, the system is easily maintainable and extendable.