# Movie Catalog Service

## ****1. Project Setup****

Follow these steps to set up and run the Movie Catalog Service on your local machine:

### ****1.1 Prerequisites****

Before starting, ensure you have the following installed on your machine:

**Java 8**

**Maven** (for managing dependencies)

**Spring Boot** (for running the application)

**Database** (H2 Database is used for simplicity in this example)

### ****1.2 Clone the Repository****

**Repository URL:** [Fazal9638/movie-catalogue-api](https://github.com/Fazal9638/movie-catalogue-api)

Clone the project repository to your local machine using the following command:

git clone https://github.com/Fazal9638/movie-catalogue-api.git

### ****1.3 Build the Project****

Navigate to the project directory and build the project using Maven:

cd movie-catalog-service

mvn clean install

### ****1.4 Run the Application****

You can run the application with the following command:

mvn spring-boot:run

This will start the Spring Boot application locally, typically on http://localhost:8080.

### ****1.5 Database Setup****

By default, the project uses an **H2 database** in-memory mode for local development. You can configure a different database (such as MySQL or PostgreSQL) by modifying the application.properties file.

**2. Project Code Understanding**

The Movie Catalog Service is designed to manage movies, directors, and ratings in a catalog. Here’s an overview of the key components:

### ****2.1 MovieService****

**Purpose:** The MovieService interface defines operations related to movie management, including creation, update, deletion, and retrieval of movies.

**Key Methods:**

createMovie(MovieDTO movie): Creates a new movie.

updateMovie(UUID movieId, MovieDTO movie): Updates an existing movie.

deleteMovie(UUID movieId): Deletes a movie.

fetchMovieById(UUID movieId): Fetches a movie by its ID.

fetchAllMovies(Pageable pageable): Fetches all movies with pagination.

fetchMoviesByDirectorId(UUID directorId): Fetches all movies by a specific director.

fetchMoviesByRatingGreaterThanEqualTo(Double rating, Pageable pageable): Fetches movies by rating.

### ****2.2 DirectorService****

**Purpose:** The DirectorService interface defines operations related to managing directors, including creating, updating, and retrieving directors.

**Key Methods:**

createDirector(Director director): Creates a new director.

updateDirector(UUID directorId, Director director): Updates an existing director.

deleteDirector(UUID id): Deletes a director.

getDirectorById(UUID id): Fetches a director by ID.

getAllDirectors(): Fetches all directors.

### ****2.3 RatingService****

**Purpose:** The RatingService interface manages movie ratings, including creation, update, and retrieval of ratings.

**Key Methods:**

createRating(RatingDTO ratingDTO): Creates a new rating.

updateRating(UUID ratingId, RatingDTO ratingDTO): Updates an existing rating.

deleteRating(UUID ratingId): Deletes a rating.

getRatingById(UUID ratingId): Retrieves a rating by its ID.

getAllRatings(): Retrieves all ratings.

### ****2.4 Data Transfer Objects (DTOs)****

**MovieDTO:** Represents the data of a movie, including fields like title, release date, director, and rating.

**DirectorDTO:** Represents the director's details, including name and biography.

**RatingDTO:** Represents the rating for a movie, including score, review, and associated movie ID.