Task: Implement CRUD operations for a movie database using a RESTful API in Java.

Requirements:

General: Create a RESTful API that allows users to perform CRUD (Create, Read, Update, Delete) operations on a movie database tables.

Language: The API should be implemented using Java and any relevant web technologies (Spring or Servlets).

Database (Any Sql type: MySql, Postgres or others):
The movie table should contain the following fields: id (auto-generated Primary key), title, genre, release year, director (relation one-to-many: director can direct many movies), actors (relation many-to-many: each actor can play in many movies), reviews (relation one-to-many: one movie can have many reviews).

The person table should contain the following fields: id (auto-generated Primary key), firstName, lastName, birthDate.
This table will store people which can be as well as director of the movie or an actor, or even both of them in the same movie.
For storing relation between movie and director you need to use directorId as Foreign key in movie table.
For storing relation between movie and actors you need to use additional table for relation with columns: movieId-actorId.

The review table should contain the following fields: id (auto-generated Primary key), text, rating (floating number between 1.0 to 10.0), date with time in UTC timezone.
For storing relation between movie and reviews you need to use movieUd as Foreign key in review table row.

API:
The API should allow users to:
a. Create a new movie by providing the title, genre, release year, directorId (person table be created via another endpoint or used as stub data - mocked with file, etc), and actor ids (person table be created via another endpoint or used as stub data - mocked with file, etc).
b. Retrieve a movie by providing its ID.
c. Update a movie by providing its ID and updated fields.
d. Delete a movie by providing its ID.
e. Retrieve all movies in the database.

The API should also support relations between the movies and other entities in the database, including:
a. Reviews: Each movie can have many reviews, and each review belongs to only one movie.
b. Actors: Each movie can have many actors, and each actor can appear in many movies.
c. Director: Each movie can have only one director, and each director can direct many movies.

The API should allow users to retrieve summary information about the movies in the database, such as how many movies there are by genre, year, etc (USE JAVA STREAMS and COLLECTIONS for processing Database response).
The API should allow users to filter requests by categories such as genre, year, director, and actors, and should support sorting and pagination of responses.

(USE JAVA STREAMS and COLLECTIONS for processing Database response).

The API should return appropriate HTTP status codes and error messages for each operation. (USE JAVA exceptions in business logic, checked when failed flow can be re-processed with another parameter, unchecked - when the logic can not be re-processed).

The API should validate user input to ensure that required fields are not empty and that the release year is a valid year format.

The code should be well-structured and follow best practices for Java programming (SOLID principle, design patterns, clean code)

Mandatory points:
- service API endpoints should return data in JSON format

Optional points:
- Use docker container for Database image (you will not need to install database system on local environment)
- Use any UI views (JS native, JS libraries, JSP, etc) OR use Postman for displaying responses
- The API should use appropriate authentication and authorization mechanisms to ensure that only authorized users can perform operations on the movie database.
- The API should be well-documented with clear instructions for how to use each endpoint.
- Unit tests should be included to ensure that the API functions correctly and handles errors appropriately.