

# Movie Ticket Booking Application

## Problem Statement

← Classroom

Movie Ticket Booking Application  
Movie Ticket Application

?

Problem

Submissions

Solutions

Doubts

Movie Booking Application

Easy • Score 120/120 • Spring Hibernate(Beginner)

Problem statement

Send feedback

Project Goal

Develop a backend Movie Booking Application enabling users to view a full movie list, add new movies with unique IDs, and retrieve specific movie information by ID. Implement strong error handling for duplicate IDs and non-existent movies. The goal is to create a smooth and dependable platform for managing movie data while maintaining data integrity across the system.

▼ Features of the Application:

The application should allow users to:

- View a list of all the available movies.
- Add new movies to the system.
- Retrieve details of a particular movie by providing its unique id.
- Delete a movie from the system using its id.
- Update details of a movie by providing its id.
- Get error messages if I try to add a movie with an existing id.
- Get an error message if I try to retrieve or delete a movie that does not exist in the system.
- Validate movie details like movie name, id, and movie director name.

Step 1

Download starter kit

Step 2

Complete project on local IDE

Step 3

Export code as .zip file

Step 4

Upload .zip file max 50mb

### Steps:

Use the following guidelines and hints to build the project.

1. We will be utilizing the Spring version **3.0.0** for this mini-project.

2. Create a class with the name **Movie** in the Model package with the following attributes and the required getters and setters:

- String id
- String movieName
- String movieDirector
- long movieRating
- String movieLanguage
- List writers
- List actors
- List genre

3. Create a class with the name **controller** in the controller package with the following rest APIs as mentioned below:

- **GET "/ticket/movies"**: It fetch the list of all the movies.
- **POST "/ticket/movie" (@Valid @RequestBody Movie movie, BindingResult bindingResult)**: It adds a new movie and throws RuntimeException if bindingResult has errors.
- **GET "/ticket/movie/{id}" (@PathVariable String id)**: It fetches a movie by the given.
- **DELETE "/ticket/movie/{id}" (@PathVariable String id)**: It deletes a movie by the given id.
- **PUT "/ticket/update/{id}" (@Valid @RequestBody Movie topic,@PathVariable String id)**:It updates a movie by the given id.

4. Create an Interface with the name **MovieServiceInterface** and its implementation with the name **MovieService** in the service package which handles the business logic for the APIs mentioned above.

5. Create the following Runtime Exception Handling classes in the Exceptions package for the below scenarios:

**a. IdAlreadyExist class**

- If the ID already exists
- It returns a **NOTFOUND** HTTP response status.

**b. IdNotFound class**

- It the id is not found
- It returns a **NOTFOUND** HTTP response status.

6. Implement validation for the following attributes:

- Restrict the 'movieName' to have a minimum length of 3 and a maximum length of 20.
- Restrict the 'id' to have a minimum value of 1.
- The name of the movieDirector can not be null.
- Restrict the movieRating to have a minimum value of 1 and a maximum value of 10.

7. You are supposed to throw a RuntimeException if any of the above validation criteria is failed.

END OF TASK

**Problem**SubmissionsSolutionsDoubts

▼ End Points To Be Created:

Movie Booking Endpoints:

- GET /ticket/movies: Retrieve the list of all movies associated with a ticket.
- GET /ticket/movie/{id} : Retrieve a movie based on the given id.
- POST /ticket/movie: Adds a movie in a ticket (Body: Movie movie, BindingResult bindingResult).
- PUT /ticket/update/{id}: Updates a movie in a ticket.
- DELETE /ticket/movie/{id}: Deletes a specific movie from the given ticket.

Testing on Postman:

After successfully creating the application, you need to test its functionality. Your application should be tested for the following scenarios:

1. Your application should test the following:

- You should be able to create a movie.
- All movies should be listed with the getAll method.
- You should be able to fetch a movie with its Id as a PathVariable.
- You should be able to delete a movie by its Id.
- You should be able to update a movie by sending a request with a JSON body.

2. Your application should throw an error if:

- The user is trying to register a new movie whose Id already exists.
- The user is trying to get a movie with an Id that does not exist.

3. Test all the exposed APIs of Backend on Postman, as follows:

## Output

The screenshot shows the Postman interface for a POST request to `localhost:8080/ticket/movie`. The request is configured with the following JSON body:

```
1 {
2   "id": "1",
3   "movieName": "Coding",
4   "movieDirector": "Coding Ninja",
5   "movieRating": 5,
6   "movieLanguage": "Hindi",
7   "writers": [
8     "Writers A",
9     "Writers B"
10  ],
11  "actors": [
12    "Ankush Singla"
13  ],
14  "genre": [
15    "Comedy",
16    "Serious"
17  ]
18 }
```

The status bar at the bottom indicates a successful response: Status: 200 OK, Time: 301 ms, Size: 123 B. The interface also shows tabs for Params, Authorization, Headers (8), Body, Pre-request Script, Tests, and Settings.

The screenshot shows the Postman interface for a GET request to `localhost:8080/ticket/movies`. The response is displayed in the Body tab, formatted as JSON:

```
2 {
3   "id": "1",
4   "movieName": "Coding",
5   "movieDirector": "Coding Ninja",
6   "movieRating": 5,
7   "movieLanguage": "Hindi",
8   "writers": [
9     "Writers A",
10    "Writers B"
11  ],
12  "actors": [
13    "Ankush Singla"
14  ],
15  "genre": [
16    "Comedy",
17    "Serious"
18  ]
19 }
20 }
```

The status bar at the bottom indicates a successful response: Status: 200 OK, Time: 36 ms, Size: 360 B. The interface also shows tabs for Params, Authorization, Headers (6), Body, Pre-request Script, Tests, and Settings.

HomeWorkspacesAPI NetworkExplore

Search Postman

Invite

Upgrade

GET localhost:8080/ticket/r

No Environment

localhost:8080/movie/1

Save

</>

GET

localhost:8080/ticket/movie/1

Send

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Body

Cookies

Headers (5)

Test Results

Status: 200 OK

Time: 9 ms

Size: 358 B

Save as Example

Pretty

Raw

Preview

Visualize

JSON

```
1  {
2    "id": "1",
3    "movieName": "Coding",
4    "movieDirector": "Coding Ninja",
5    "movieRating": 5,
6    "movieLanguage": "Hindi",
7    "writers": [
8      "Writers A",
9      "Writers B"
10   ],
11   "actors": [
12     "Ankush Singla"
13   ],
14   "genre": [
15     "Comedy",
16     "Serious"
17   ]
18 }
```

Online

Find and replace

Console

Postbot

Runner

Start Proxy

Cookies

Trash