

# Movie Booking Application

**By Sabarish Iyer**

## CONTENTS

Summary.....	5
GitHub.....	5
Backend:.....	5
Frontend:.....	5
Business requirements covered.....	6
System Architecture.....	6
Micro Services List.....	6
Eureka Server.....	6
Dashboard.....	6
API Gateway.....	6
Primary Flow of Admin Creating A Movie.....	7
Entities.....	7
Movie Entity:.....	7
Movie And Theatre Entity:.....	7
Data Transfer Object (DTO).....	8
Movie DTO:.....	8
Movie Controller.....	9
Create Movie Interface:.....	9
Create Movie Method Implementation:.....	9
Movie Service.....	9
Add Movie Method Implementation:.....	9
Movie Repository.....	10
Custom Exception.....	10
Exception for Same Movie Details.....	10
API Gateway Property.....	10
Pass all Movies related URLs to Movie Module.....	10
Postman Screenshots for Add Movie.....	11
Admin Adding Movie:.....	11
Admin Adding Movie with incomplete Information:.....	11
User trying to Add Movie:.....	12
Swagger Screenshots for Add Movie.....	12
Create Movie Endpoint.....	12
Responses by Create Movie.....	13
Successfully Created a Movie.....	13
Incomplete Details Provided.....	14
Postman Screenshots for other Endpoints.....	14
Register User.....	14
Registering New User.....	14
Login User.....	15
Logging an Existing User In.....	15
Login User with wrong Password:.....	16

Forgot Password.....	16
Forgot password Check.....	16
Password Reset.....	17
Get All Movies.....	17
Search for a Movie.....	18
Book Ticket for Movie.....	18
Get All Tickets for Movie.....	19
Movie Status Change:.....	19
Invalid Movie Update.....	20
Invalid Status Change.....	20
Invalid Number of Tickets Allotted.....	20
Delete Movie.....	21
By Admin.....	21
By User.....	21
Caching and session management.....	22
Caching Implemented.....	22
Session Management.....	22
Swagger.....	23
User Documentation.....	23
Movies Documentation.....	23
Tickets Documentation.....	24
Exception - Custom & Handling.....	24
Custom Exception Super Class for all Custom Exceptions:.....	24
Global Exception Handler.....	25
Postman Showing Custom Exception.....	26
Testing and Code Quality.....	26
Service Test.....	26
Get By ID.....	26
Controller Test.....	27
View All:.....	27
Maven Test:.....	28
Sonar Qube:.....	28
Front-End Screenshots.....	29
Home Page.....	29
Register.....	29
Register Form.....	29
Register Validation.....	30
Login Page.....	30
Homepage for Logged-In User.....	31
Movie Booking Page.....	31
Home Page.....	32
My Bookings.....	32

Sold Out Movie.....	33
Blocks Ticket Booking.....	33
Forgot Password.....	33
Verify If User exists.....	33
Check for Old Password.....	34
Admin Home Page:.....	34
Update Options.....	35
Movie Status Update.....	35
Button to Reset Movie Status.....	35
Tickets Allotted Update.....	36
Movie Deletion.....	36
Confirm Before Deletion.....	36
New Movie.....	37
Creating Using New Movie Card.....	37

## SUMMARY

This project implements a Movie Ticket Booking System using a microservices architecture. The application is composed of five independently deployable modules: User Module (authentication and authorization), Movie Module (movie and theatre management), Tickets Module (ticket booking and order management), API Gateway (centralized routing and security), and Eureka Server (service discovery).

The system supports role-based access control using JWT-based stateless authentication. End users can register, log in, search and view movies, and book tickets, while administrators can manage movies and monitor booking status. All APIs are documented using Swagger / Java Docs, validated using Bean Validation, and protected using Spring Security.

The application follows clean separation of concerns using controller interfaces for API contracts and controller implementations for execution logic. Business logic resides in service layers with centralized exception handling and optional caching for improved performance.

The frontend is built using Angular (standalone architecture) and communicates with backend services through the API Gateway. Unit tests, Maven build reports, and static code analysis demonstrate code quality and maintainability.

This project satisfies the required business use cases while maintaining extensibility for future enhancements.

## GitHub

### Backend:

- Link: <https://github.com/Sabarish-2/BookAMovie-Spring-Boot>

To run;

```
mvn clean package
```

```
java -jar target/*.jar
```

### Frontend:

- Link: [https://github.com/Sabarish-2/MovieBookingApp\\_Angular](https://github.com/Sabarish-2/MovieBookingApp_Angular)

To run;

```
npm install
```

```
npm run build
```

## BUSINESS REQUIREMENTS COVERED

- User Registration and Login with Role-based access (Admin / Customer)
- View and Search Movies by name and theatre
- Book Tickets for a selected Movie and theatre
- Persist booked tickets and allow users to view their bookings
- Admin management of Movies (Add / Delete) and its status updates
- Centralized API routing and Security handled by API Gateway
- Exception Handling and Security
- Swagger and Java Documentation

## SYSTEM ARCHITECTURE

### Micro Services List

1. Movie And Theatre Module - Handles All Movies Data and Movie CRUD Operation.
2. Tickets Module - Handles All Tickets and Ticket Booking Mechanism.
3. User Module - Handles Login, Register and Forgot Password Mechanism.
4. API Gateway - Routes All Requests to respective Micro-service.
5. Eureka Server - Stores and gives an instance of required Micro-service from load-balancer.

## Eureka Server

### Dashboard

The screenshot shows the Spring Eureka dashboard at <http://localhost:8761>. The top navigation bar includes links for HOME and LAST 1000 SINCE STARTUP. The main content area is divided into several sections:

- System Status:** Displays various system metrics:
  - Environment: test
  - Data center: default
  - Current time: 2026-01-04T22:19:27 +0530
  - Uptime: 11:47
  - Lease expiration enabled: true
  - Renew threshold: 8
  - Renews (last min): 16
- DS Replicas:** Shows a list of registered hosts:
  - localhost
- Instances currently registered with Eureka:** A table listing registered applications and their details:

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - <a href="http://172.19.4.77:api-gateway:8090">172.19.4.77:api-gateway:8090</a>
MOVIE-AND-THEATRE-MODULE	n/a (1)	(1)	UP (1) - <a href="http://172.19.4.77:movie-and-theatre-module:8082">172.19.4.77:movie-and-theatre-module:8082</a>
TICKETS-MODULE	n/a (1)	(1)	UP (1) - <a href="http://172.19.4.77:tickets-module:8083">172.19.4.77:tickets-module:8083</a>
USER-MODULE	n/a (1)	(1)	UP (1) - <a href="http://172.19.4.77:user-module:8081">172.19.4.77:user-module:8081</a>
- General Info:** A section with a link to activate Windows.

## API Gateway

All client requests are routed through the API Gateway, which handles centralized routing and security. The gateway forwards requests to MovieService, Tickets Service, and UserService based on configured routes, while Eureka Server enables service discovery.

## PRIMARY FLOW OF ADMIN CREATING A MOVIE

### Entities

#### Movie Entity:

```
@Entity  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Movie {  
  
    @EmbeddedId  
    private MovieAndTheater movieAndTheatre;  
  
    @Min(1)  
    @Column(nullable = false)  
    private int ticketsAllotted;  
  
    @Enumerated(EnumType.STRING)  
    private MovieStatus adminOverrideStatus;  
  
    public Movie(MovieAndTheater movieAndTheatre, int ticketsAllotted) {  
        this.ticketsAllotted = ticketsAllotted;  
        this.movieAndTheatre = movieAndTheatre;  
    }  
}
```

#### Movie And Theatre Entity:

- Composite Key Requires Embeddable class

```
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
@Embeddable
```

```

public class MovieAndTheater {
    @NotNull
    private String movieName;
    @NotNull
    private String theatreName;
}

```

## Data Transfer Object (DTO)

Movie DTO:

```

@Data
@NoArgsConstructor
@AllArgsConstructor
public class MovieDTO {

    @NotBlank(message = "com.moviebookingapp.movie_and_theatre_module.dtos.movieName.invalid")
    private String movieName;

    @NotBlank(message = "com.moviebookingapp.movie_and_theatre_module.dtos.theatreName.invalid")
    private String theatreName;

    @Min(value = 1L, message = "com.moviebookingapp.movie_and_theatre_module.dtos.ticketsAllotted.invalid")
    private int ticketsAllotted;

    private Integer ticketsAvailable;

    private MovieStatus movieStatus;

    public MovieDTO(String movieName, String theatreName, int ticketsAllotted) {
        this.movieName = movieName;
        this.theatreName = theatreName;
        this.ticketsAllotted = ticketsAllotted;
    }
}

```

## Movie Controller

Create Movie Interface:

```
@Operation(summary = "Create A New Movie")
@ApiResponse(responseCode = "201", description = "Movie Created Successfully")
@ApiResponse(responseCode = "400", description = "Validation Error in Movie Details Provided")
@ApiResponse(responseCode = "409", description = "Movie Already Exists")
 ResponseEntity<MovieDTO> createMovie(@Valid @RequestBody MovieDTO movieDTO);
```

Create Movie Method Implementation:

```
/**
 * Creates a new movie.
 *
 * @param movieDTO The movie details to create.
 * @return A response entity containing the created movie.
 */
@Override
@PostMapping("create")
@PreAuthorize("hasRole('ADMIN')")
public ResponseEntity<MovieDTO> createMovie(MovieDTO movieDTO) {
    return new ResponseEntity<>(movieService.addMovie(movieDTO), HttpStatus.CREATED);
}
```

## Movie Service

Add Movie Method Implementation:

```
/**
 * Adds a new movie to the system.
 *
 * @param movieDTO Data transfer object containing movie details.
 * @return The added movie as a DTO.
 * @throws MovieAlreadyExistsException if the movie already exists.
 */
@Override
public MovieDTO addMovie(MovieDTO movieDTO) {
```

```

Movie newMovie = mapper.map(movieDTO);

if (movieRepository.findById(newMovie.getMovieAndTheatre()).isPresent()) {
    throw new MovieAlreadyExistsException(
        "Movie " + movieDTO.getMovieName() + " at " + movieDTO.getTheatreName() + "
    Already Exists!");
}

Movie savedMovie = movieRepository.save(newMovie);
return mapper.map(savedMovie);
}

```

## Movie Repository

```

@Repository
public interface MovieRepository extends JpaRepository<Movie, MovieAndTheater>,
JpaSpecificationExecutor<Movie> {
}

```

## Custom Exception

### Exception for Same Movie Details

```

public class MovieAlreadyExistsException extends CustomException {

    @Serial
    private static final long serialVersionUID = 9L;

    public MovieAlreadyExistsException(String message) {
        super(serialVersionUID, HttpStatus.CONFLICT, message);
    }
}

```

## API Gateway Property

### Pass all Movies related URLs to Movie Module

- id: movie-and-theatre-module
- uri: lb://MOVIE-AND-THEATRE-MODULE
- predicates:
- Path=/api/v1.0/moviebooking/movies/\*\*

## Postman Screenshots for Add Movie

### Admin Adding Movie:

The screenshot shows the Postman interface with the 'CreateMovie' POST request selected. The 'Headers' tab contains an 'Authorization' header with a Bearer token. The 'Body' tab shows a JSON payload for a movie named 'Final Destination 8' at PSR with 75 allotted tickets. The response status is '201 Created' with a 'movieId' of 1.

```
1 {
2   "movieName": "Final Destination 8",
3   "movieStatus": null,
4   "theatreName": "PSR",
5   "ticketsAllotted": 75,
6   "ticketsAvailable": null
7 }
```

### Admin Adding Movie with incomplete Information:

The screenshot shows the Postman interface with the 'CreateMovie' POST request selected. The 'Body' tab has a raw JSON payload with only 'movieName' set to 'Final Destination 8'. The response status is '400 Bad Request' with validation errors: 'Theatre Name Cannot be Null or Empty', 'Tickets Allotted cannot be less than 1', and 'Movie Name Cannot be Null or Empty'.

```
1 [
2   "Theatre Name Cannot be Null or Empty",
3   "Tickets Allotted cannot be less than 1",
4   "Movie Name Cannot be Null or Empty"
5 ]
```

## User trying to Add Movie:

The screenshot shows the Postman application interface. On the left, the 'Personal Workspace' sidebar lists various collections and environments. The main workspace shows a 'POST CreateMovie' request under the 'API / CreateMovie' section. The 'Body' tab is selected, displaying the following JSON payload:

```
1 {
2   "movieName": "Final Destination 8",
3   "theatreName": "PSR",
4   "ticketsAllotted": 75
5 }
```

Below the body, the status bar indicates a **401 Unauthorized** response with a duration of 323 ms and a size of 392 B. A note says 'Pass the correct auth credentials'. The bottom right corner of the window has an 'Activate Windows' watermark.

## Swagger Screenshots for Add Movie

### Create Movie Endpoint

The screenshot shows the Swagger UI interface for the 'MovieAndTheatreModule'. The top navigation bar includes links for 'Import bookmarks...', 'Getting Started', 'Google Gemini', 'Swagger UI', 'Eureka', and 'Spring Initializr'. The main area displays the 'POST /create' endpoint for creating a new movie. The 'Parameters' section shows 'No parameters'. The 'Request body' section is marked as 'required' and contains the following schema:

```
{
  "movieName": "string",
  "theatreName": "string",
  "ticketsAllotted": 0,
  "ticketsAvailable": 0,
  "movieStatus": "AVAILABLE"
}
```

The 'Responses' section shows a successful response (201) with the description 'Movie Created Successfully'. The 'Media type' dropdown is set to 'application/json'. The example response is identical to the request body schema:

```
{
  "movieName": "string",
  "theatreName": "string",
  "ticketsAllotted": 0,
  "ticketsAvailable": 0,
  "movieStatus": "AVAILABLE"
}
```

A watermark for 'Activate Windows' is visible in the bottom right corner.

## Responses by Create Movie

The screenshot shows the Swagger UI interface for the 'MovieAndTheatreModule'. It displays four error responses for the 'createMovie' endpoint:

- 401 Authentication Error**:  
Media type: application/json  
Example Value | Schema  
{  
 "movieName": "string",  
 "theatreName": "string",  
 "ticketsAllotted": 0,  
 "ticketsAvailable": 0,  
 "movieStatus": "AVAILABLE"  
}
- 409 Movie Already Exists**:  
Media type: application/json  
Example Value | Schema  
{  
 "movieName": "string",  
 "theatreName": "string",  
 "ticketsAllotted": 0,  
 "ticketsAvailable": 0,  
 "movieStatus": "AVAILABLE"  
}
- 500 Unexpected Error Internally**:  
Media type: application/json  
Example Value | Schema  
string
- 503 Required Tickets Microservice Unreachable**:  
Media type: application/json

## Successfully Created a Movie

The screenshot shows the Swagger UI interface for the 'MovieAndTheatreModule'. It displays a successful response for the 'createMovie' endpoint:

Curl:  
curl -X 'POST' '\<http://localhost:8082>/create'\  
-H 'Accept: \*/\*'\  
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJxdyfisInVbUDI5T0xFX0FETU10Ii1d#01joxNzY3NTk0NTI0LClcHAIOje3Njci0TgXWJR9.K31uK5HigdrT\_n0WUQ0hYT65k0FA1ISXmbzqWkbA4'\  
-H 'Content-Type: application/json'  
-d '{  
 "movieName": "Mission Possible",  
 "movieStatus": "null",  
 "theatreName": "TSK",  
 "ticketsAllotted": 45,  
 "ticketsAvailable": null  
}'

Request URL:  
<http://localhost:8082/create>

Server response

Code	Details
201	Response body { "movieName": "Mission Possible", "movieStatus": "null", "theatreName": "TSK", "ticketsAllotted": 45, "ticketsAvailable": null } Response headers cache-control: no-cache,no-store,max-age=0,must-revalidate connection: keep-alive content-type: application/json date: Mon, 05 Jan 2026 07:09:06 GMT expires: keep-alive: timeout=60 pragma: no-cache transfer-encoding: chunked x-content-type-options: nosniff x-frame-options: DENY x-xss-protection: 0

Responses

Code	Description
201	Movie Created Successfully

## Incomplete Details Provided

The screenshot shows the MovieBookingAppAngular Swagger UI interface. In the top navigation bar, the URL is `http://localhost:8082/swagger-ui/index.html#/movie-controller-impl/createMovie`. The main area displays a curl command and a request URL for creating a movie. The request URL is `http://localhost:8082/create`. The server response shows an error status of 400 with the message "Error: response status is 400". The response body contains an array of validation errors: ["Movie Name Cannot be Null or Empty", "Tickets Allotted cannot be less than 1", "Theatre Name Cannot be Null or Empty"]. Below the response, the response headers are listed, including cache-control, connection, content-type, last-modified, expires, pragma, transfer-encoding, x-content-type-options, x-frame-options, and x-xss-protection.

## POSTMAN SCREENSHOTS FOR OTHER ENDPOINTS

### Register User

#### Registering New User

The screenshot shows a Postman collection named "User / UserRegister". It contains a POST request for the endpoint `http://localhost:8090/api/v1.0/moviebooking/register`. The request body is in raw JSON format, containing user registration details: {"firstName": "Sabarish", "lastName": "Iyer", "loginID": "IyerSabarish", "emailID": "2387974@cts.com", "password": "Sabarish@2387974", "contactNumber": "9152301111", "userRole": "ADMIN"}. The response status is 201 Created, and the response body is identical to the request body. A success message "Activate Windows Go to Settings to activate Windows." is visible at the bottom right.

## Register Validation Check

The screenshot shows the Postman interface with a collection named "User". A POST request is made to `http://localhost:8090/api/v1.0/moviebooking/register`. The request body contains:

```
1 "firstName": "Sabarish",
2 "emailID": "2387974@com",
3 "password": "",
4 "contactNumber": "1152301111"
```

The response status is **400 Bad Request**, with a message indicating validation errors:

```
1 [
2   "password: Cannot be Null or Empty",
3   "contactNumber: Must be a Valid 10-Digit Number",
4   "loginID: Cannot be Null or Empty",
5   "lastName: Cannot be Null or Empty"
6 ]
```

## Login User

Logging an Existing User In.

The screenshot shows the Postman interface with a collection named "User". A POST request is made to `http://localhost:8090/api/v1.0/moviebooking/login`. The request body contains:

```
1 {
2   "loginID": "IyerSabarish",
3   "password": "Sabarish@2387974"
4 }
```

The response status is **200 OK**, with a complex JSON object returned:

```
1 {"yJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJjeWVyl2F1YyJpZgIiLClyb2xLTjo1Uk9MPV9BRE1JTjTsTm1hdCT6MtC2NzYwMTY0OSwizXhwIjoxNzY3NjA1MjQ5fQ,E5nhzNgKe3MF-cAdm1yUfeQb6AAgCbA2zJ26523TXI}
```

## Login User with wrong Password:

The screenshot shows the Postman application interface. In the top navigation bar, there are several tabs: 'POST UserLogin - Admin' (highlighted), 'POST UserLogin', 'POST CreateMovie', 'POST UserRegister', and 'GET AllMovies'. Below the tabs, the URL is set to 'http://localhost:8090/api/v1.0/moviebooking/login'. The 'Body' tab is selected, showing a raw JSON payload:

```
1 "loginID": "TyerSabarish",
2 "password": "12345678"
```

Under the 'Body' tab, the status is '500 Internal Server Error' with a response time of 124 ms and a size of 519 B. The response body contains the error message: 'Exact Error in User: org.springframework.security.authentication.BadCredentialsException: Bad credentials'.

## Forgot Password

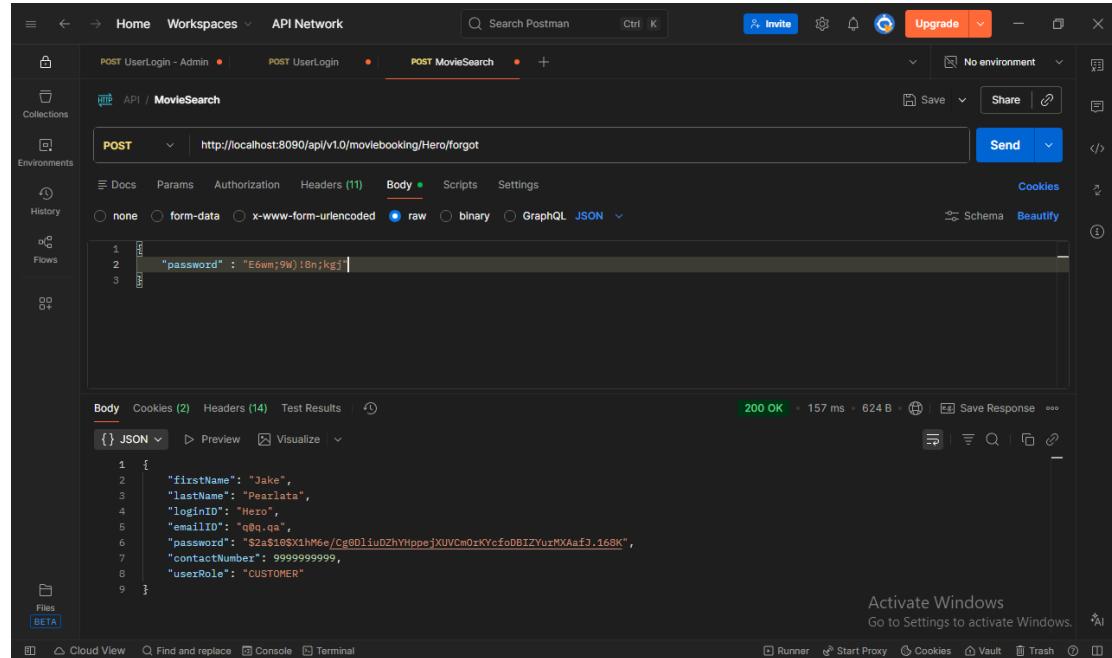
### Forgot password Check

The screenshot shows the Postman application interface. In the top navigation bar, there are several tabs: 'POST UserLogin - Admin' (highlighted), 'POST UserLogin', 'GET MovieSearch' (highlighted), and '+'. Below the tabs, the URL is set to 'http://localhost:8090/api/v1.0/moviebooking/Hero/forgot'. The 'Body' tab is selected, showing a raw JSON payload:

```
{ "firstName": "Jake",
  "lastName": "Pearlata",
  "loginID": "Hero",
  "emailID": "q0q.qa",
  "password": "$2a$10$Eq1e0DLIoukUiQwF1FZ9U.Xlt5TXckCyjG8.MHF1F.0TC2rbvUZQ6",
  "contactNumber": 9999999999,
  "userRole": "CUSTOMER"}
```

Under the 'Body' tab, the status is '200 OK' with a response time of 25 ms and a size of 624 B. The response body is empty, indicated by the message 'This request does not have a body'.

## Password Reset



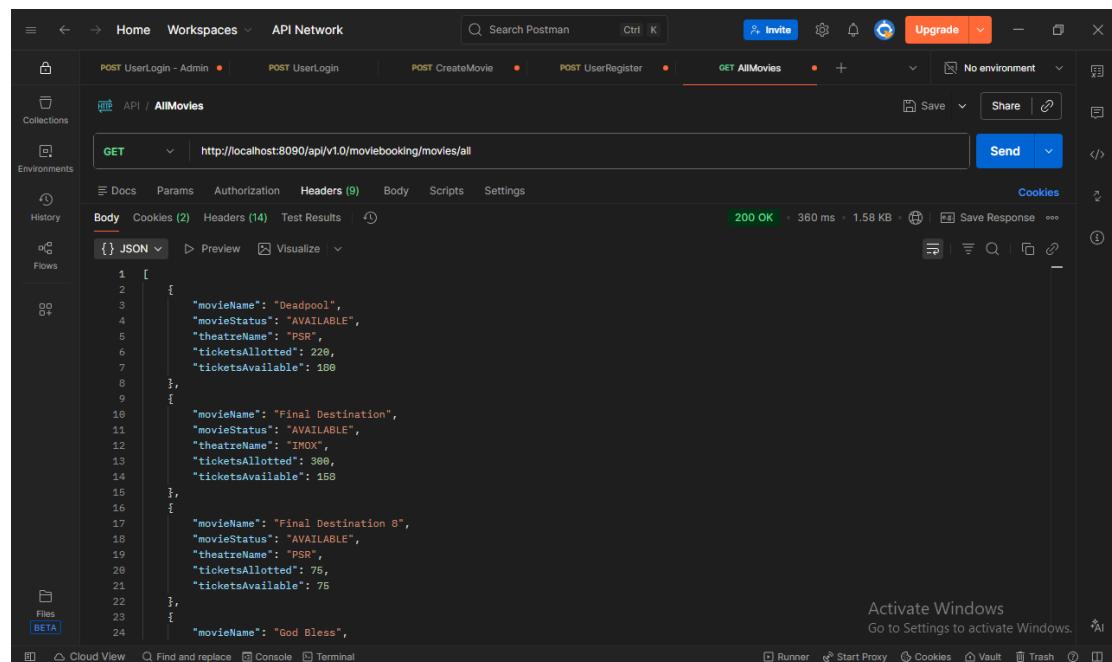
The screenshot shows the Postman interface with the following details:

- Request Method:** POST
- Request URL:** http://localhost:8090/api/v1.0/moviebooking/Hero/forgot
- Body Type:** raw
- Body Content:**

```
1 ["password": "E6wm;9w!Bn;kg3"]
```
- Response Status:** 200 OK
- Response Headers:** 157 ms, 624 B
- Response Body (JSON):**

```
1 {
  2   "firstName": "Jake",
  3   "lastName": "Pearlata",
  4   "loginId": "Hero",
  5   "emailId": "q0q.qa",
  6   "password": "$2a$10$X1hM6e/Cg@Olu0ZhYHpejXUVcm0zKYcf0DBIZYurMXAafJ.168K",
  7   "contactNumber": 9999999999,
  8   "userRole": "CUSTOMER"
}
```

## Get All Movies



The screenshot shows the Postman interface with the following details:

- Request Method:** GET
- Request URL:** http://localhost:8090/api/v1.0/moviebooking/movies/all
- Body Type:** JSON
- Body Content:**

```
1 [
  2   {
  3     "movieName": "Deadpool",
  4     "movieStatus": "AVAILABLE",
  5     "theatreName": "PSR",
  6     "ticketsAllotted": 220,
  7     "ticketsAvailable": 180
  8   },
  9   {
  10    "movieName": "Final Destination",
  11    "movieStatus": "AVAILABLE",
  12    "theatreName": "IMOX",
  13    "ticketsAllotted": 300,
  14    "ticketsAvailable": 158
  15  },
  16  {
  17    "movieName": "Final Destination 8",
  18    "movieStatus": "AVAILABLE",
  19    "theatreName": "PSR",
  20    "ticketsAllotted": 75,
  21    "ticketsAvailable": 75
  22  },
  23  {
  24    "movieName": "God Bless",
  25  }
```
- Response Status:** 200 OK
- Response Headers:** 360 ms, 1.58 kB
- Response Body (JSON):**

```
1 [
  2   {
  3     "movieName": "Deadpool",
  4     "movieStatus": "AVAILABLE",
  5     "theatreName": "PSR",
  6     "ticketsAllotted": 220,
  7     "ticketsAvailable": 180
  8   },
  9   {
  10    "movieName": "Final Destination",
  11    "movieStatus": "AVAILABLE",
  12    "theatreName": "IMOX",
  13    "ticketsAllotted": 300,
  14    "ticketsAvailable": 158
  15  },
  16  {
  17    "movieName": "Final Destination 8",
  18    "movieStatus": "AVAILABLE",
  19    "theatreName": "PSR",
  20    "ticketsAllotted": 75,
  21    "ticketsAvailable": 75
  22  },
  23  {
  24    "movieName": "God Bless",
  25  }
```

## Search for a Movie

The screenshot shows the Postman interface with a collection named "API / MovieSearch". A GET request is made to `http://localhost:8090/api/v1.0/moviebooking/movies/search?movieName=mission`. The response status is 200 OK, and the JSON body contains two movie records:

```
[{"id": 1, "movieName": "Mission Impossible", "movieStatus": "AVAILABLE", "theatreName": "IMOX", "ticketsAllotted": 88, "ticketsAvailable": 88}, {"id": 10, "movieName": "Mission Possible", "movieStatus": "AVAILABLE", "theatreName": "ISX", "ticketsAllotted": 45, "ticketsAvailable": 45}]
```

## Book Ticket for Movie

The screenshot shows the Postman interface with a collection named "Tickets - API". A POST request is made to `http://localhost:8090/api/v1.0/moviebooking/Final Destination/add`. The raw body of the request is:

```
{"quantity": 4, "userID": "Hero", "movieName": "Final Destination", "seatNumbers": "A1, A2, B3, ZX", "theatreName": "PSR"}
```

The response status is 201 Created, and the JSON body of the response is:

```
{"ticketID": 70, "userID": "Hero", "movieName": "Final Destination", "theatreName": "PSR", "seatNumbers": "A1, A2, B3, ZX", "quantity": 4}
```

## Get All Tickets for Movie

The screenshot shows the Postman interface with a collection named "API - API-Movie". A specific endpoint, "GET BookedTickets From Movie & Theatre", is selected. The request method is set to "GET" and the URL is "http://localhost:8090/api/v1.0/moviebooking/tickets/booked/destination/psr". The "Body" tab is active, showing raw JSON input: "1". The response status is "200 OK" with a response time of 34 ms and a body size of 424 B. The response content is empty JSON: "[]".

## Movie Status Change:

The screenshot shows the Postman interface with a collection named "API". An endpoint named "PUT UpdateMovie" is selected. The request method is "PUT" and the URL is "http://localhost:8090/api/v1.0/moviebooking/movies/destination/update/imox". The "Body" tab is active, showing raw JSON input: "1", "2", "3", "4", "5", "6", "7". The response status is "200 OK" with a response time of 67 ms and a body size of 546 B. The response content is a JSON object: {"movieName": "final destination", "movieStatus": "SOLD\_OUT", "theatreName": "imox", "ticketsAllotted": 188, "ticketsAvailable": 16}.

## Invalid Movie Update

### Invalid Status Change

The screenshot shows the Postman interface with a PUT request to `http://localhost:8090/api/v1.0/moviebooking/movies/final destination/update/imox`. The request body contains the following JSON:

```
1 // Optional:  
2 // "ticketsAllotted": 142 |  
3 "adminOverrideStatus": "AVAILABLE"  
4
```

The response status is **406 Not Acceptable**, and the error message is: **Movie is already Sold Out, Cannot mark final destination at imox as AVAILABLE**.

## Invalid Number of Tickets Allotted

The screenshot shows the Postman interface with a PUT request to `http://localhost:8090/api/v1.0/moviebooking/movies/final destination/update/imox`. The request body contains the following JSON:

```
1 // Optional:  
2 "ticketsAllotted": 0,  
3 "adminOverrideStatus": "SOLD_OUT"  
4
```

The response status is **406 Not Acceptable**, and the error message is: **Tickets Allotted cannot be less than Tickets Booked!**.

# Delete Movie

By Admin

The screenshot shows the Postman interface with the following details:

- Request Type:** DELETE
- URL:** `http://localhost:8090/api/v1.0/moviebooking/movies/god bless 2/delete/pqr`
- Headers:** Authorization (Value: Bearer eyJhbGciOiJIUzI1NiJ9eyJzdWIiOiJZJvliwiicm9sZ...)
- Body:** Raw response: "Movie god bless 2 At pqr Deleted Successfully!"
- Status:** 200 OK
- Time:** 120 ms
- Size:** 469 B

By User

The screenshot shows the Postman interface with the following details:

- Request Type:** DELETE
- URL:** `http://localhost:8090/api/v1.0/moviebooking/movies/god bless 3/delete/lmox`
- Headers:** Authorization (Value: Bearer eyJhbGciOiJIUzI1NiJ9eyJzdWIiOiJZJvliwiicm9sZ...)
- Body:** Raw response: "Pass the correct auth credentials"
- Status:** 401 Unauthorized
- Time:** 50 ms
- Size:** 446 B

## CACHING AND SESSION MANAGEMENT

### Caching Implemented

- Tickets cannot be modified, hence can be cached for better performance.

```
/**  
 * Retrieves a ticket by its ID.  
 *  
 * @param ticketID The ID of the ticket to retrieve.  
 * @return The ticket as a DTO.  
 * @throws TicketNotFoundException if the ticket is not found.  
 */  
  
@Override  
@Cacheable("ticket")  
public TicketDTO getTicketByID(Long ticketID) {  
    Ticket ticket = ticketRepository.findById(ticketID)  
        .orElseThrow(() -> new TicketNotFoundException(ticketID));  
    return mapper.map(ticket);  
}
```

### Session Management

- Stateless Session Tokens are used in security.

```
httpSecurity  
    .csrf(AbstractHttpConfigurer::disable)  
    .authorizeHttpRequests(auth -> auth  
        .requestMatchers("/swagger-ui/**", "/swagger-ui.html",  
        "/v3/api-docs/**").permitAll()  
        .requestMatchers(HttpMethod.GET, "/tickets/booked/**").permitAll()  
        .anyRequest().authenticated()  
    )  
    .sessionManagement(session ->  
        session.sessionCreationPolicy(SessionCreationPolicy.STATELESS));
```

## SWAGGER

### User Documentation

The screenshot shows the User REST API Documentation page in a browser. The title is "User REST API Documentation v1.0 OAS 3.0". It includes a "Servers" dropdown set to "http://localhost:8081 - Generated server url" and an "Authorize" button. The main section lists several API endpoints under the "user-controller-impl" category:

- GET /{loginInput}/forgot Reset Password for User Verification
- POST /{loginInput}/forgot Reset Password After Verification
- POST /register Register A user
- POST /login Login A user
- GET /all Retrieve All Users
- DELETE /delete Delete An Existing User

A red box highlights the "DELETE /delete" endpoint. A watermark for "Activate Windows" is visible in the bottom right corner.

### Movies Documentation

The screenshot shows the Movie REST API Documentation page in a browser. The title is "Movie REST API Documentation v1.0 OAS 3.0". It includes a "Servers" dropdown set to "http://localhost:8082 - Generated server url" and an "Authorize" button. The main section lists several API endpoints under the "movie-controller-impl" category:

- PUT /{movieName}/update/{theatreName} Update an Existing Movie
- POST /create Create A New Movie
- GET /{movieName}/{theatreName} Retrieve Movie By Movie Name and Theatre Name
- GET /search Search Movies
- GET /all Retrieve All Movies
- DELETE /{movieName}/delete/{theatreName} Delete An Existing Movie

A red box highlights the "DELETE /{movieName}/delete/{theatreName}" endpoint. A watermark for "Activate Windows" is visible in the bottom right corner.

## Tickets Documentation

The screenshot shows a browser window displaying the 'Tickets REST API Documentation' at <http://localhost:8083/swagger-ui/index.html#/movie-controller-impl/createMovie>. The page title is 'Tickets REST API Documentation v1.0 OAS 3.0'. It includes a navigation bar with links like 'Import bookmarks...', 'Getting Started', 'Google Gemini', 'Swagger UI', 'Eureka', and 'Spring Initializer'. Below the title, it says 'This is the REST API Documentation for the Tickets (Ticketing) Module'. A 'Servers' dropdown is set to '<http://localhost:8083> - Generated server url'. An 'Authorize' button is visible. The main content area is titled 'ticket-controller-impl' and lists various API endpoints:

- POST** /{movieName}/add Create A New Ticket
- GET** /tickets/{ticketID} Retrieve Tickets With Ticket ID
- GET** /tickets/{movieName}/{theatreName} Retrieve All Tickets For A Movie In A Theatre
- GET** /tickets/user/{userID} Retrieve All Tickets For A User
- GET** /tickets/booked/{movieName}/{theatreName} Retrieve Number Of Tickets Booked For A Movie In A Theatre
- GET** /tickets/all Retrieve All Tickets
- DELETE** /tickets/{movieName}/delete/{theatreName} Delete All Existing Tickets Of A Movie In Specified Theatre

## EXCEPTION - CUSTOM & HANDLING

### Exception User Already Exists:

```
public class UserAlreadyExistsException extends CustomException {  
  
    @Serial  
    private static final long serialVersionUID = 9L;  
  
    public UserAlreadyExistsException(String message) {  
        super(serialVersionUID, HttpStatus.CONFLICT, message);  
    }  
}
```

### Custom Exception Super Class for all Custom Exceptions:

```
@Getter  
public abstract class CustomException extends RuntimeException {
```

```

private final long serialVersionUID;
private final HttpStatus status;
private final String message;
protected CustomException(long serialVersionUID, HttpStatus status, String message)
{
    this.serialVersionUID = serialVersionUID;
    this.status = status;
    this.message = message;
}

```

### Global Exception Handler

- To handle all Exceptions and show Logs instead of throwing errors

```

// Handle Custom Errors
@ExceptionHandler(CustomException.class)
public ResponseEntity<String> handleCustomExceptions(CustomException ex) {
    return new ResponseEntity<>(ex.getMessage(), ex.getStatus());
}

// Handle All other Errors
@ExceptionHandler(Exception.class)
@ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
public String handleOtherExceptions(Exception ex) {
    return "Exact Error in Movie: " + ex;
}

```

## Postman Showing Custom Exception

The screenshot shows the Postman application interface. In the center, there is a request card for a POST method to 'MovieSearch' at 'http://localhost:8090/api/v1.0/moviebooking/register'. The request body is set to 'raw' JSON and contains the following data:

```
1 {
2     "loginID": "Hero",
3     "firstName": "AV",
4     "lastName": "OOP",
5     "password": "asa",
6     "emailID": "AB@ac.c",
7     "contactNumber": "9999999999"
8 }
```

Below the request, the response section shows a 500 Internal Server Error. The error message is: 'Exact Error in User: com.moviebookingapp.user\_module.exception.UserAlreadyExistsException: User With Same Email or Login ID Already exists!!'

## TESTING AND CODE QUALITY

### Service Test

#### Get By ID

```
@Test
@DisplayName("GetMovieByID-Positive")
void test_GetMovieByID_positive() {
    when(movieRepository.findById(movieAndTheater)).thenReturn(Optional.of(movie));
    when(mapper.map(movie)).thenReturn(movieDTO);
    when(ticketsClient.getBookedTickets(any(), any())).thenReturn(ResponseEntity.ok(1L));
```

```
MovieDTO actualMovie = movieService.getMovieByID(movieName, theatreName);
```

```
assertEquals(movieDTO, actualMovie);
}
```

```
@Test
@DisplayName("GetMovieByID-Negative-MovieNotFound")
void test_GetMovieByID_negative_movieNotFound() {
    when(movieRepository.findById(movieAndTheater)).thenReturn(Optional.empty());
```

```

        assertsThrows(MovieNotFoundException.class, () -> movieService.getMovieByID(movieName,
theatreName));

    }

@Test
@DisplayName("GetMovieByID-Negative-FeignRuntimeInPrivateMethod")
void test_GetMovieByID_negative_feignRuntimeInPrivateMethod() {
    when(movieRepository.findById(movieAndTheater)).thenReturn(Optional.of(movie));
    when(mapper.map(movie)).thenReturn(movieDTO);
    when(ticketsClient.getBookedTickets(movieName, theatreName)).thenReturn(null);

    assertsThrows(RuntimeException.class, () -> movieService.getMovieByID(movieName,
theatreName));
}

```

## Controller Test

View All:

```

@Test
@DisplayName("ViewAllMovies-Positive")
void viewAllMovies_positive() {
    when(movieService.getAllMovies()).thenReturn(List.of(movieDTO));

    ResponseEntity<List<MovieDTO>> response = movieController.viewAllMovies();

    assertEquals(HttpStatus.OK, response.getStatusCode());
    assertEquals(List.of(movieDTO), response.getBody());
}

@Test
@DisplayName("ViewAllMovies-Negative-MovieNotFound")
void viewAllMovies_negative_movieNotFound() {
    when(movieService.getAllMovies()).thenThrow(new MovieNotFoundException());

    assertsThrows(MovieNotFoundException.class, () -> movieController.viewAllMovies());
}

```

## Maven Test:

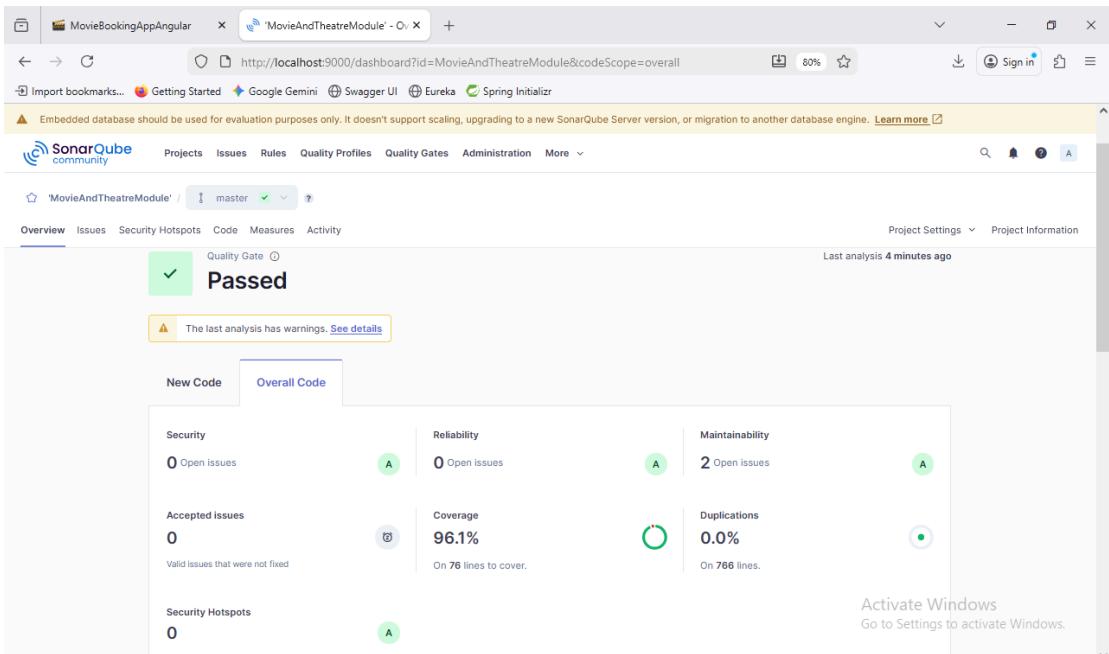
The screenshot shows a terminal window within a code editor interface. The terminal output is as follows:

```
0 8082
2026-01-05T15:15:25.747+05:30 INFO 20952 --- [movie-and-theatre-module] [main] c.m.m.MovieAndTheatreModuleApplication : Started MovieAndTheatreModuleApplication in 4.375 seconds (process running for 41.544)
2026-01-05T15:15:25.806+05:30 INFO 20952 --- [movie-and-theatre-module] [main] o.s.c.n.e.s.EurekaServiceRegistry : Unregistering application MOVIE-AND-THEATRE-MODULE with eureka with status DOWN
2026-01-05T15:15:25.807+05:30 INFO 20952 --- [movie-and-theatre-module] [main] com.netflix.discovery.DiscoveryClient : Saw local status change event StatusChangeEvent [timestamp=1767606325807, current=DOWN, previous=UP]
2026-01-05T15:15:25.814+05:30 INFO 20952 --- [movie-and-theatre-module] [foReplicator-%d] com.netflix.discovery.DiscoveryClient : DiscoveryClient _MOVIE-AND-THEATRE-MODULE/host.docker.internal:movie-and-theatre-module:8082: registering service...
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 32.80 s -- in com.moviebookingapp.movie_and_theatre_module.MovieAndTheatreModuleApplicationTests
[INFO] Running com.moviebookingapp.movie_and_theatre.services.MovieServiceTest
2026-01-05T15:15:25.828+05:30 INFO 20952 --- [movie-and-theatre-module] [foReplicator-%d] com.netflix.discovery.DiscoveryClient : DiscoveryClient _MOVIE-AND-THEATRE-MODULE/host.docker.internal:movie-and-theatre-module:8082 - registration status: 204
[INFO] Tests run: 18, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.794 s -- in com.moviebookingapp.movie_and_theatre_module.MovieServiceTest
2026-01-05T15:15:26.652+05:30 INFO 20952 --- [movie-and-theatre-module] [ionShutdownHook] o.s.c.n.e.s.EurekaServiceRegistry : Unregistering application MOVIE-AND-THEATRE-MODULE with eureka with status DOWN
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 32, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- jacoco-maven-plugin:0.8.11:report (report) @ movie-and-theatre-module ---
[INFO] Loading execution data file C:\Users\Administrator\Project\MovieBookingApp_SpringBoot\movie-and-theatre-module\target\jacoco.exec
[INFO] Analyzed bundle 'movie-and-theatre-module' with 21 classes
[INFO]
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 55.748 s
[INFO] Finished at: 2026-01-05T15:15:34+05:30
[INFO]
```

Activate Windows  
Go to Settings to activate Windows.

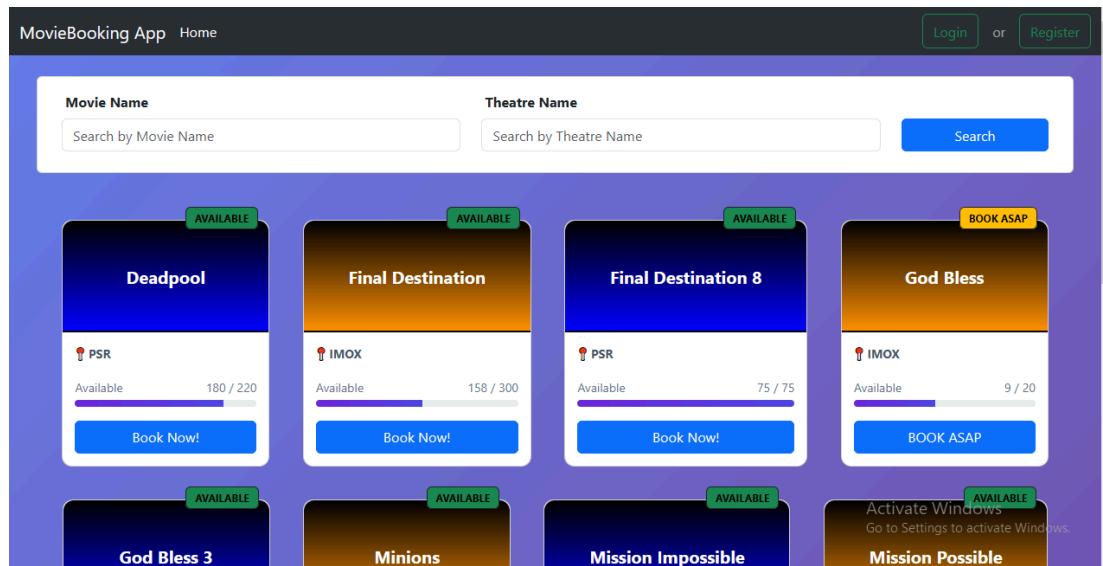
PS C:\Users\Administrator\Project\MovieBookingApp\_SpringBoot\movie-and-theatre-module> 57:1 (676 chars, 18 line breaks) CRLF UTF-8 4 spaces

## Sonar Qube:



## FRONT-END SCREENSHOTS

### Home Page



### Register

#### Register Form

The screenshot shows the register form page of the MovieBooking App. At the top, there is a navigation bar with the text "MovieBooking App" and "Home" on the left, and "Login" or "Register" on the right. Below the navigation bar is a large white "Register" button. The form itself consists of several input fields: "First Name" (placeholder: Enter your first name), "Last Name" (placeholder: Enter your last name), "Login ID" (placeholder: Choose a login ID), "Email" (placeholder: Enter your email address), "Password" (placeholder: Create a password), and "Confirm Password" (placeholder: Enter your password again). To the right of the form, there is a message: "Activate Windows" followed by "Go to Settings to activate Windows.".

## Register Validation

The screenshot shows the registration form for the MovieBooking App. The fields entered are:

- Email: sabarish@CTS.com
- Password: A password consisting of four dots ('....').
- Confirm Password: Re-entered password field.
- Contact Number: An empty field.

Validation errors displayed:

- >Password must be at least 8 characters long.
- Password must include at least one lowercase letter.
- Password must include at least one uppercase letter.
- Password must include at least one special character.
- Passwords do not match.
- Enter your contact number
- Enter a valid 10-digit number.

Buttons and links:

- Register button
- Already have an account? [Login](#)
- Login or Register buttons
- Activate Windows link: Go to Settings to activate Windows.

## Login Page

The screenshot shows the login form for the MovieBooking App. The fields entered are:

- Email ID / Login ID: Hero
- Password: A password consisting of ten dots ('.....').

Buttons and links:

- Login button
- Forgot Password?
- Already have an account? [Register](#)
- Login or Register buttons
- Activate Windows link: Go to Settings to activate Windows.

## Homepage for Logged-In User

The screenshot shows the homepage of the MovieBooking App. At the top, there is a navigation bar with links for "MovieBooking App", "Home", and "MyBookings". On the right side of the nav bar are "Hi, Hero" and "Logout" buttons. Below the nav bar is a search section with fields for "Movie Name" and "Theatre Name", each with a "Search" button. The main content area displays movie availability in a grid format:

Movie Name	Theatre Name	Status
Deadpool	PSR	AVAILABLE
Final Destination	IMOX	AVAILABLE
Final Destination 8	PSR	AVAILABLE
God Bless	IMOX	BOOK ASAP
God Bless 3		AVAILABLE
Minions		AVAILABLE
Mission Impossible		AVAILABLE
Mission Possible		AVAILABLE

Each movie listing includes a "Book Now!" button. A promotional message at the bottom right encourages users to "Activate Windows" by going to settings.

## Movie Booking Page

The screenshot shows the "Book Tickets" page. At the top, there is a navigation bar with links for "MovieBooking App", "Home", and "MyBookings". On the right side of the nav bar are "Hi, Hero" and "Logout" buttons. The main content area has a blue header "Book Tickets". Below it is a summary box containing movie details:

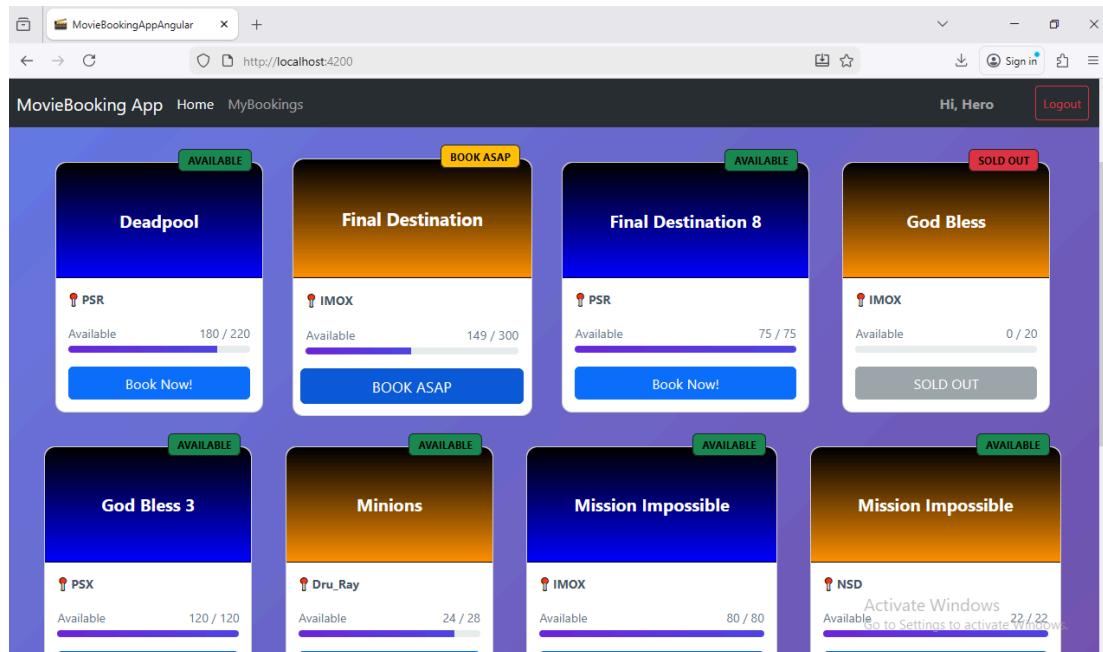
**Movie:** Final Destination  
**Theatre:** IMOX  
**Status:** AVAILABLE  
**Available:** 158 / 300  
**Seats:** O3,O4,O5,O6,O7,O8,O9,O10,P1

Below the summary box is a "Number of Seats" input field with the value "9" and a "Book Now" button.

A promotional message at the bottom right encourages users to "Activate Windows" by going to settings.

## Home Page

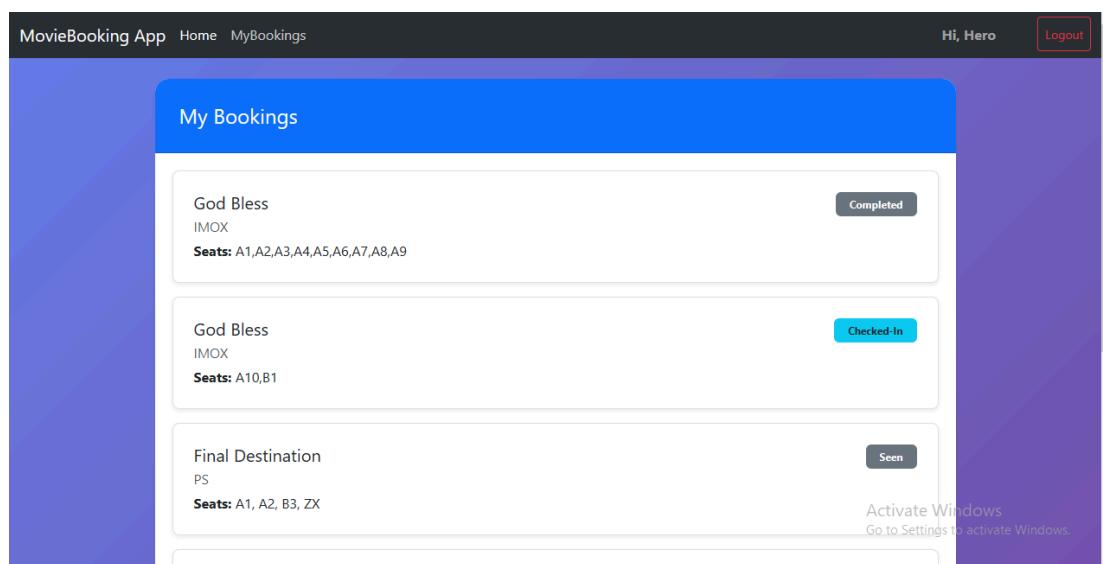
Status and Value change after Tickets booked



The screenshot shows the MovieBooking App's home page. It displays a grid of movie cards. Each card includes the movie title, a status indicator (e.g., AVAILABLE, SOLD OUT), the cinema name, and seat availability. Buttons for booking or checking availability are also present.

Movie Title	Cinema	Status	Available Seats
Deadpool	PSR	AVAILABLE	180 / 220
Final Destination	IMOX	BOOK ASAP	149 / 300
Final Destination 8	PSR	AVAILABLE	75 / 75
God Bless	IMOX	SOLD OUT	0 / 20
God Bless 3	PSX	AVAILABLE	120 / 120
Minions	Dru_Ray	AVAILABLE	24 / 28
Mission Impossible	IMOX	AVAILABLE	80 / 80
Mission Impossible	NSD	AVAILABLE	22 / 22 Activate Windows Go to Settings to activate Windows.

## My Bookings

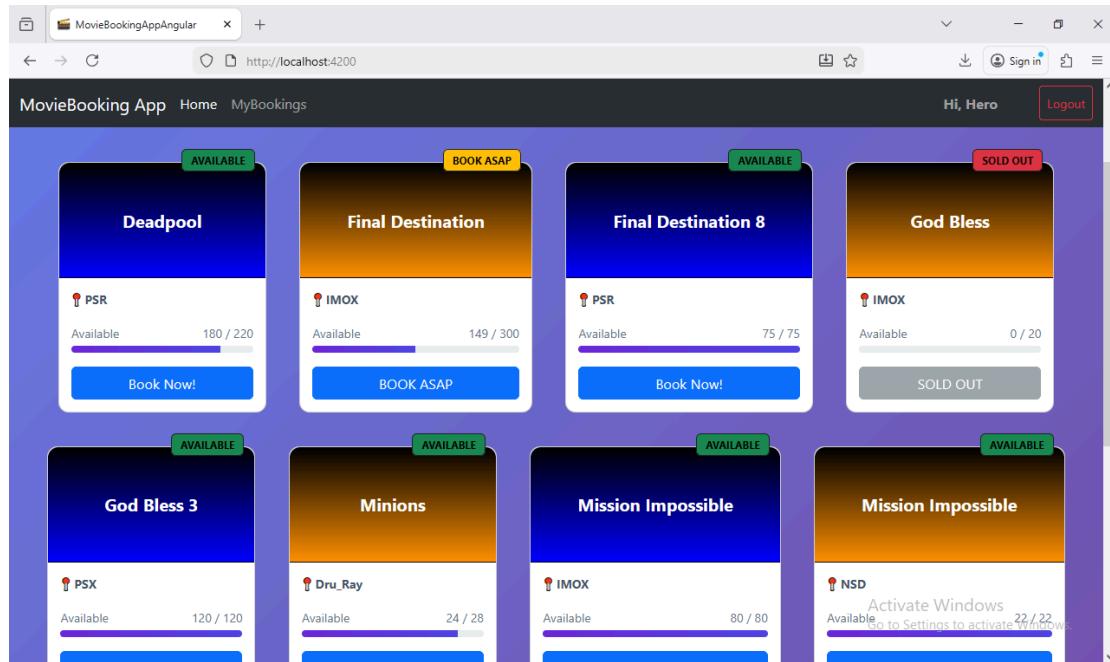


The screenshot shows the 'My Bookings' page. It lists past bookings with details like movie title, cinema, and seat information. Each booking has a status indicator (Completed, Checked-In, Seen) and a link to activate Windows.

Booking Details	Status	Action
God Bless IMOX Seats: A1,A2,A3,A4,A5,A6,A7,A8,A9	Completed	
God Bless IMOX Seats: A10,B1	Checked-In	
Final Destination PS Seats: A1, A2, B3, ZX	Seen	

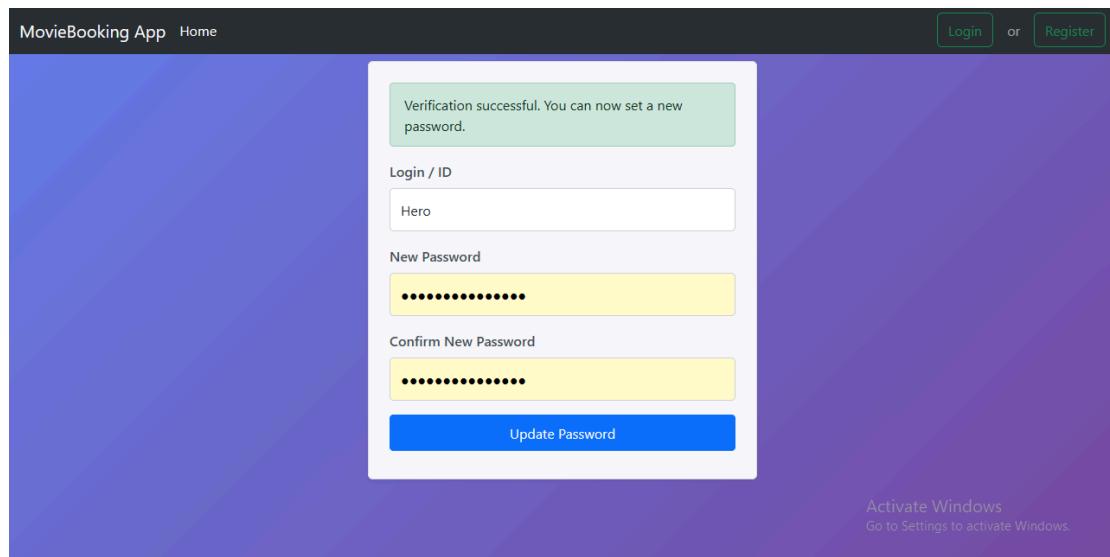
## Sold Out Movie

Blocks Ticket Booking

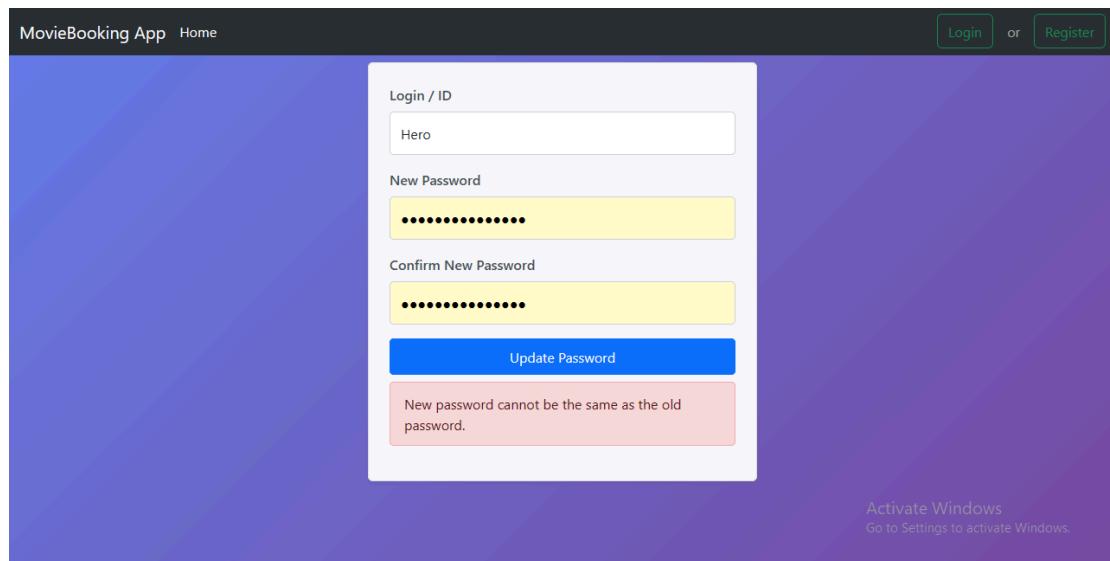


## Forgot Password

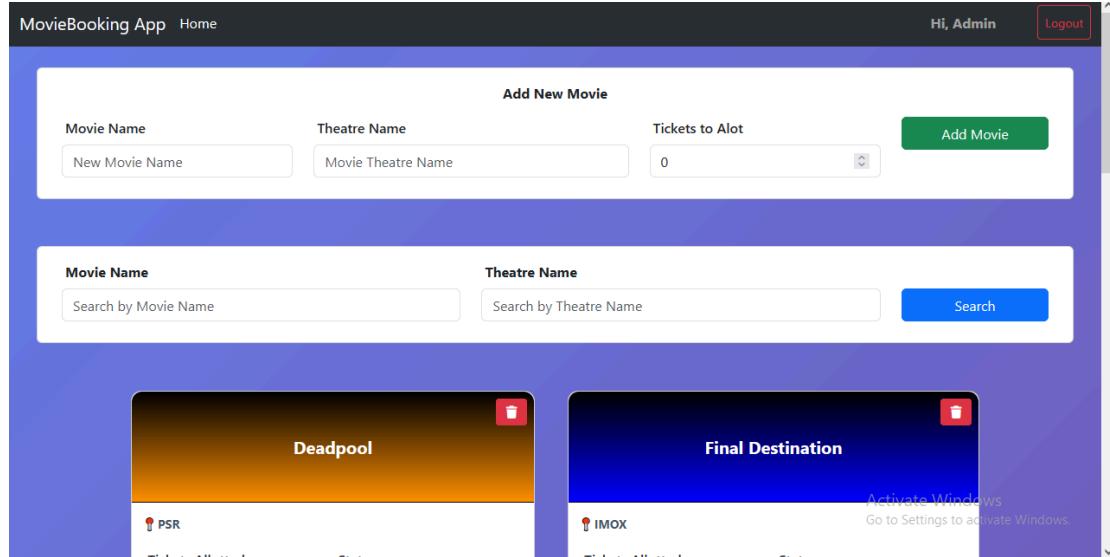
Verify If User exists



## Check for Old Password



## Admin Home Page:



## Update Options

### Movie Status Update

The screenshot shows the MovieBooking App interface. At the top, there's a navigation bar with "MovieBooking App" and "Home" on the left, and "Hi, Admin" and "Logout" on the right. Below the navigation bar, there are four movie cards arranged in a grid:

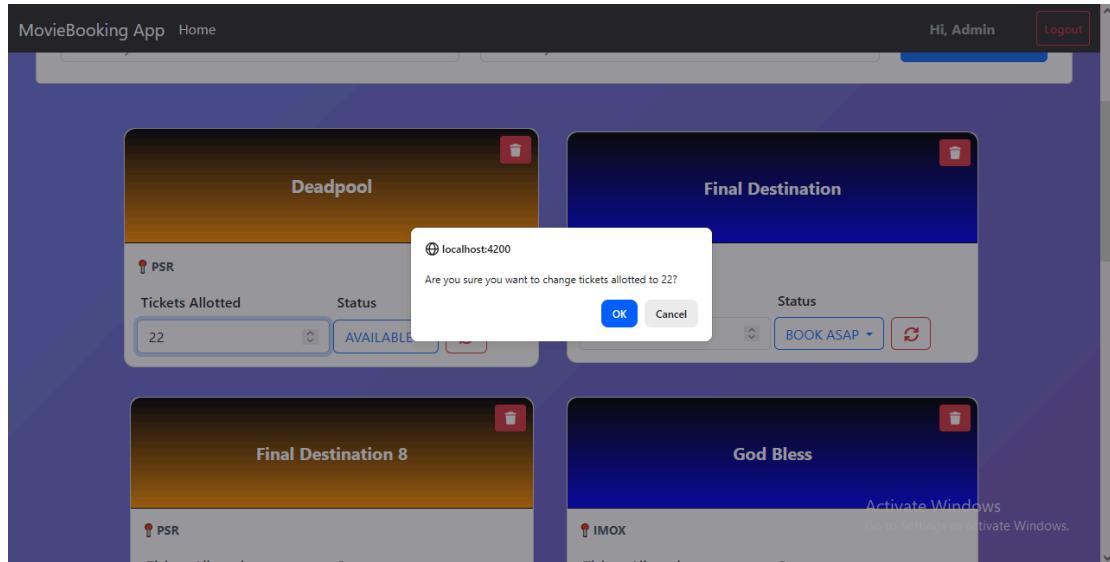
- Deadpool**: Located at PSR, Tickets Allotted: 220, Status: AVAILABLE. A dropdown menu is open over the status button, showing "Available", "Sold Out", and "Book ASAP".
- Final Destination**: Located at IMOX, Tickets Allotted: 300, Status: BOOK ASAP. A dropdown menu is open over the status button, showing "Available", "Sold Out", and "Book ASAP".
- Final Destination 8**: Located at PSR, Tickets Allotted: 220, Status: AVAILABLE.
- God Bless**: Located at IMOX, Tickets Allotted: 300, Status: BOOK ASAP. A message at the bottom right says "Activate Windows Go to Settings to activate Windows."

### Button to Reset Movie Status

The screenshot shows the MovieBooking App interface. At the top, there's a navigation bar with "MovieBooking App" and "Home" on the left, and "Hi, Admin" and "Logout" on the right. Below the navigation bar, there are four movie cards arranged in a grid:

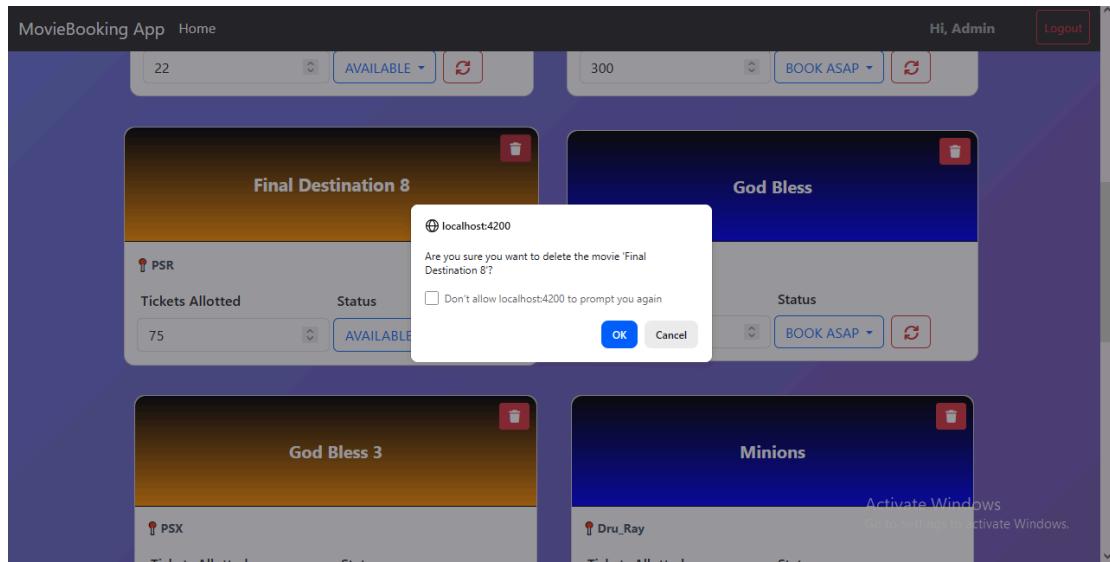
- Mission Impossible**: Located at IMOX, Tickets Allotted: 80, Status: AVAILABLE. A confirmation dialog box is overlaid on the card, asking "localhost:4200 Are you sure you want to reset the status? OK Cancel".
- Mission Possible**: Located at ISX, Tickets Allotted: 45, Status: AVAILABLE.
- World**: Located at PSR, Tickets Allotted: 30, Status: BOOK ASAP. A message at the bottom right says "Activate Windows Go to Settings to activate Windows."

## Tickets Allotted Update



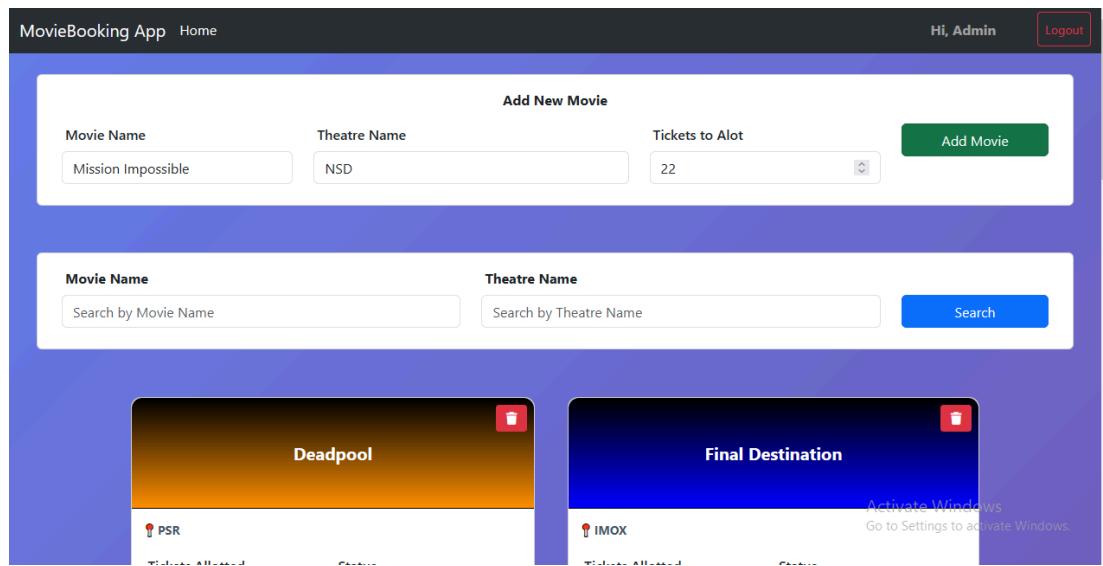
## Movie Deletion

Confirm Before Deletion



## New Movie

Creating Using New Movie Card



## Conclusion

This project successfully implements a Movie Ticket Booking application using a microservices-based architecture. All required business functionalities—including user authentication, movie management, ticket booking, and administrative controls—have been implemented with proper validation, exception handling, and role-based security. The system leverages an API Gateway for centralized routing and security, Eureka Server for service discovery, and Swagger for API documentation. Unit testing, caching, and stateless session management ensure maintainability, performance, and scalability. Overall, the application meets the specified requirements and demonstrates clean architectural design and industry-standard development practices.