**Movie Booking Application**

-By **Sabarish** **Iyer**

Contents

[Summary 4](#_Toc218537453)

[GitHub 4](#_Toc218537454)

[Business requirements covered 5](#_Toc218537455)

[System Architecture 5](#_Toc218537456)

[Micro Services List 5](#_Toc218537457)

[Eureka Server - Dashboard 5](#_Toc218537458)

[Primary Flow of Admin Creating A Movie 6](#_Toc218537459)

[Entities 6](#_Toc218537460)

[Movie Entity: 6](#_Toc218537461)

[Movie And Theatre Entity: 6](#_Toc218537462)

[Data Transfer Object (DTO) 7](#_Toc218537463)

[Movie DTO: 7](#_Toc218537464)

[Movie Controller 8](#_Toc218537465)

[Create Movie Interface: 8](#_Toc218537466)

[Create Movie Method Implementation: 8](#_Toc218537467)

[Movie Service 9](#_Toc218537468)

[Add Movie Method Implementation: 9](#_Toc218537469)

[Movie Repository 9](#_Toc218537470)

[Exception Created for Similar Movie Check Details 10](#_Toc218537471)

[API Gateway Property to pass all Movies related URLs to Movie Module 10](#_Toc218537472)

[Postman Screenshots for Add Movie 10](#_Toc218537473)

[Admin Adding Movie: 10](#_Toc218537474)

[Admin Adding Movie with incomplete Information: 11](#_Toc218537475)

[User trying to Add Movie: 11](#_Toc218537476)

[Swagger Screenshots for Add Movie 12](#_Toc218537477)

[Postman Screenshots for other Endpoints 14](#_Toc218537478)

[Registering New User 14](#_Toc218537479)

[Register Validation Check 14](#_Toc218537480)

[Login User 15](#_Toc218537481)

[Login User with wrong Password: 15](#_Toc218537482)

[Forgot password Check: 16](#_Toc218537483)

[Password Reset: 16](#_Toc218537484)

[All Movies 17](#_Toc218537485)

[Movie Search 17](#_Toc218537486)

[Book Ticket for Movie 18](#_Toc218537487)

[Get All Tickets for Movie 18](#_Toc218537488)

[Movie Status Change: 19](#_Toc218537489)

[Invalid Movie Tickets Update 20](#_Toc218537490)

[Delete Movie 21](#_Toc218537491)

[Delete Movie by User 21](#_Toc218537492)

[Caching and session management 22](#_Toc218537493)

[Caching Implemented 22](#_Toc218537494)

[Session Management 22](#_Toc218537495)

[Swagger 23](#_Toc218537496)

[Swagger - User: 23](#_Toc218537497)

[Swagger - Movies: 23](#_Toc218537498)

[Swagger - Tickets: 24](#_Toc218537499)

[Exception - Custom & Handling 25](#_Toc218537500)

[Exception User Already Exists: 25](#_Toc218537501)

[Custom Exception Super Class for all Custom Exceptions: 25](#_Toc218537502)

[Global Exception Handler 25](#_Toc218537503)

[Postman Showing Custom Exception 26](#_Toc218537504)

[Testing 27](#_Toc218537505)

[Service Test of Get By ID 27](#_Toc218537506)

[Controller Test of View All: 28](#_Toc218537507)

[Maven Test: 29](#_Toc218537508)

[Code Quality and Coverage 29](#_Toc218537509)

[Sonar Qube: 29](#_Toc218537510)

[Front-End Screenshots 30](#_Toc218537511)

[Home Page 30](#_Toc218537512)

[Register 30](#_Toc218537513)

[Register Validation 31](#_Toc218537514)

[Login Page 31](#_Toc218537515)

[Homepage for Logged-In User 32](#_Toc218537516)

[Movie Booking Page 32](#_Toc218537517)

[Updated Home Page 33](#_Toc218537518)

[My Bookings 33](#_Toc218537519)

[Forgot Password 34](#_Toc218537520)

[Forgot Password Check for Old Password 34](#_Toc218537521)

[Admin Home Page: 35](#_Toc218537522)

[Update Options 36](#_Toc218537523)

[Movie Deletion 37](#_Toc218537524)

[New Movie Creation 37](#_Toc218537525)

[Button to Reset Movie Status 38](#_Toc218537526)

# 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>
* 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

A screenshot of a computer

AI-generated content may be incorrect.

# 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

@AllArgsConstructor

@NoArgsConstructor

@Embeddable

public class MovieAndTheater {

@NonNull

private String movieName;

@NonNull

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> {

}

## Exception Created for Similar Movie Check 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 to 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:

A screenshot of a computer

AI-generated content may be incorrect.

### Admin Adding Movie with incomplete Information:

A screenshot of a computer program

AI-generated content may be incorrect.

### User trying to Add Movie:

A screenshot of a computer

AI-generated content may be incorrect.

## Swagger Screenshots for Add Movie

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

# Postman Screenshots for other Endpoints

## Registering New User

A screenshot of a computer

AI-generated content may be incorrect.

### Register Validation Check

A screenshot of a computer

AI-generated content may be incorrect.

## Login User

A screenshot of a computer

AI-generated content may be incorrect.

### Login User with wrong Password:

A screenshot of a computer

AI-generated content may be incorrect.

## Forgot password Check:

A screenshot of a computer

AI-generated content may be incorrect.

### Password Reset:

A screenshot of a computer

AI-generated content may be incorrect.

## All Movies

A screenshot of a computer

AI-generated content may be incorrect.

## Movie Search

A screenshot of a computer

AI-generated content may be incorrect.

## Book Ticket for Movie

A screenshot of a computer

AI-generated content may be incorrect.

## Get All Tickets for Movie

A screenshot of a computer

AI-generated content may be incorrect.

## Movie Status Change:

A screenshot of a computer

AI-generated content may be incorrect.

### Invalid Movie Tickets Update

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

## Delete Movie

A screenshot of a computer

AI-generated content may be incorrect.

### Delete Movie by User

A screenshot of a computer

AI-generated content may be incorrect.

# 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

### Swagger - User:

A screenshot of a computer

AI-generated content may be incorrect.

### Swagger - Movies:

A screenshot of a computer

AI-generated content may be incorrect.

### Swagger - Tickets:

A screenshot of a computer

AI-generated content may be incorrect.

# 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 serialVersionUIDPerException;

private final HttpStatus status;

private final String message;

protected CustomException(long serialVersionUIDPerException, HttpStatus status, String message) {

this.serialVersionUIDPerException = serialVersionUIDPerException;

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

A screenshot of a computer

AI-generated content may be incorrect.

# Testing

### Service Test of 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());

assertThrows(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);

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

}

### Controller Test of 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());

assertThrows(MovieNotFoundException.class, () -> movieController.viewAllMovies());

}

### Maven Test:

A computer screen shot of a program

AI-generated content may be incorrect.

## Code Quality and Coverage

### Sonar Qube:

A screenshot of a computer

AI-generated content may be incorrect.

# Front-End Screenshots

### Home Page

A screenshot of a computer

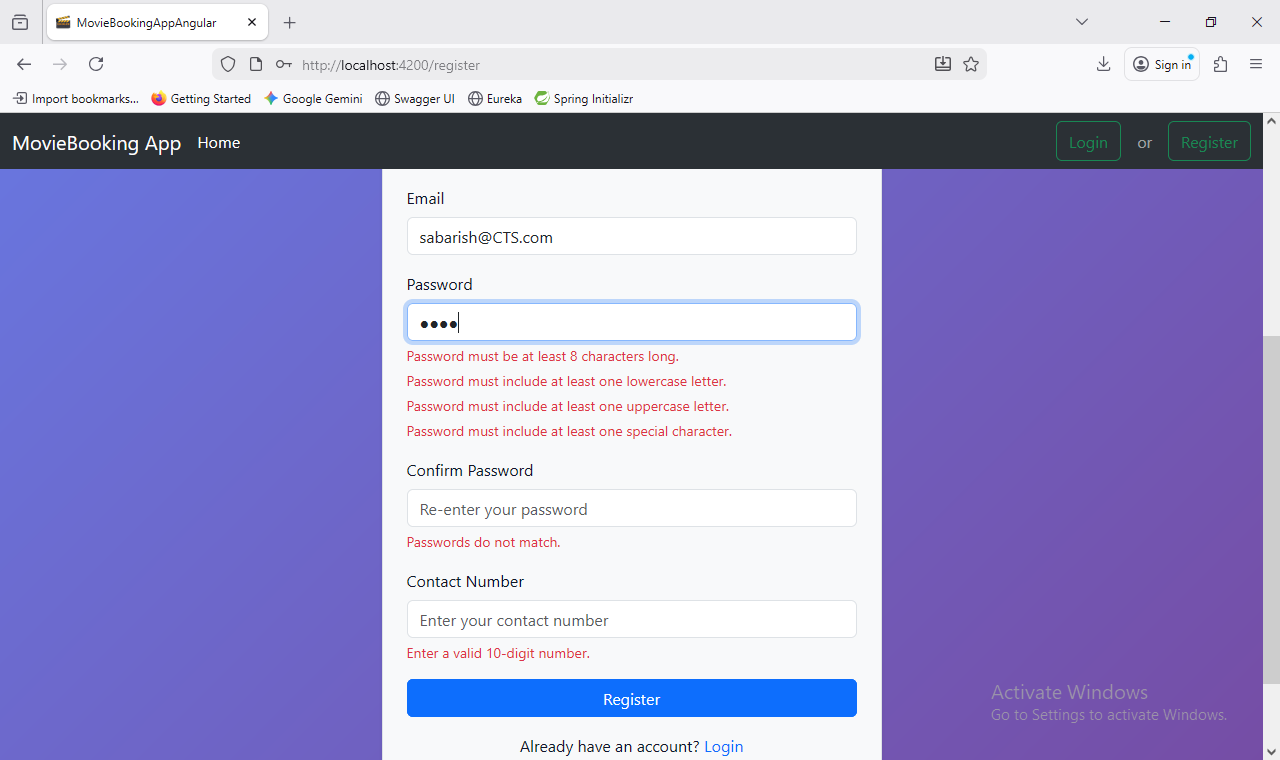
AI-generated content may be incorrect.

### Register

A screenshot of a computer

AI-generated content may be incorrect.

### Register Validation



### Login Page

A screenshot of a computer

AI-generated content may be incorrect.

### Homepage for Logged-In User

A screenshot of a computer

AI-generated content may be incorrect.

### Movie Booking Page

A screenshot of a computer

AI-generated content may be incorrect.

### Updated Home Page

A screenshot of a computer

AI-generated content may be incorrect.

### My Bookings

A screenshot of a computer

AI-generated content may be incorrect.

### Forgot Password

A screenshot of a computer

AI-generated content may be incorrect.

### Forgot Password Check for Old Password

A screenshot of a computer

AI-generated content may be incorrect.

### Admin Home Page:

A screenshot of a computer

AI-generated content may be incorrect.

### Update Options

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

### Movie Deletion

A screenshot of a computer

AI-generated content may be incorrect.

### New Movie Creation

A screenshot of a computer

AI-generated content may be incorrect.

### Button to Reset Movie Status

A screenshot of a computer

AI-generated content may be incorrect.