**Movie Booking APP**

By Sabarish Iyer

Summary:  
…

Bussiness Req Covered

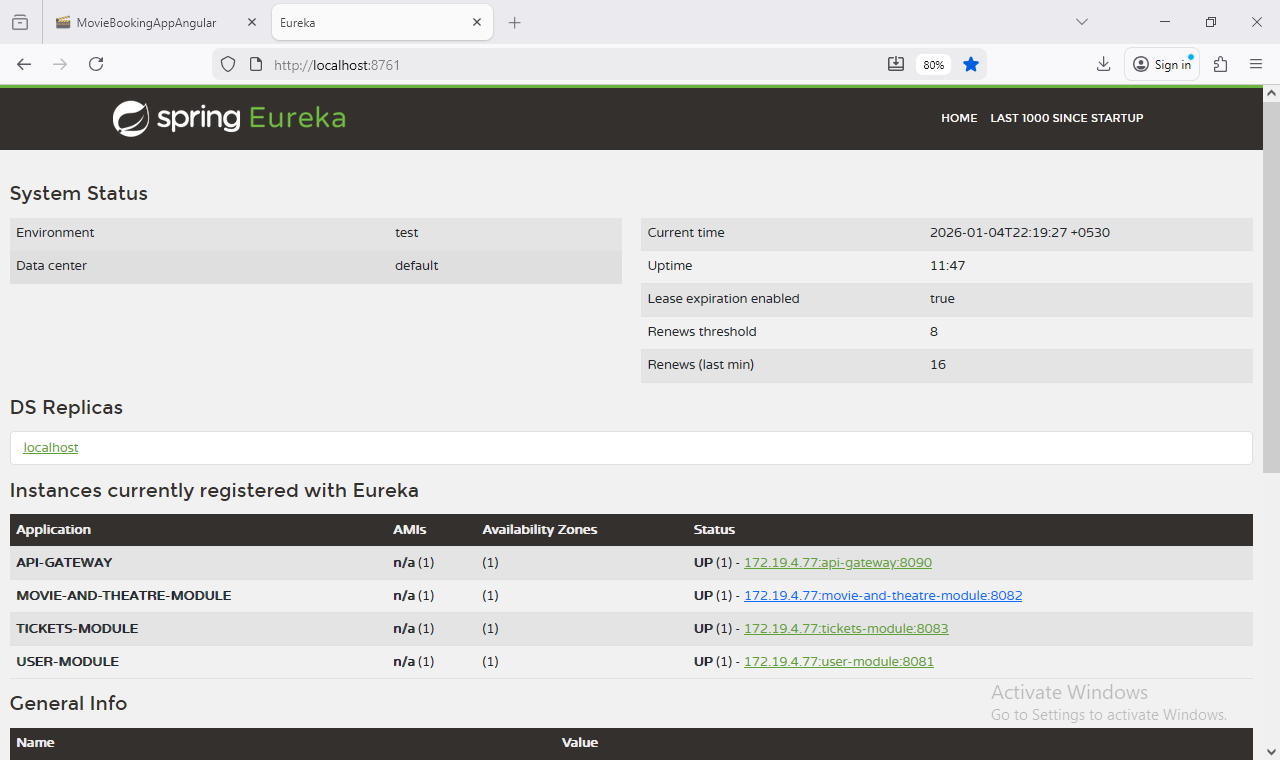
…

GitHub Repo URL:

**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:**



**Primary Flow of Admin Creating A Movie:**

**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;

}

}

**MovieAndTheatre Entity (Composite Key Requires Embeddable class):**

@Data

@AllArgsConstructor

@NoArgsConstructor

@Embeddable

public class MovieAndTheater {

@NonNull

private String movieName;

@NonNull

private String theatreName;

}

**Movie Data Transfer Object (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 Method:**

@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);

**Movie Controller - 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 Movie with Same 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

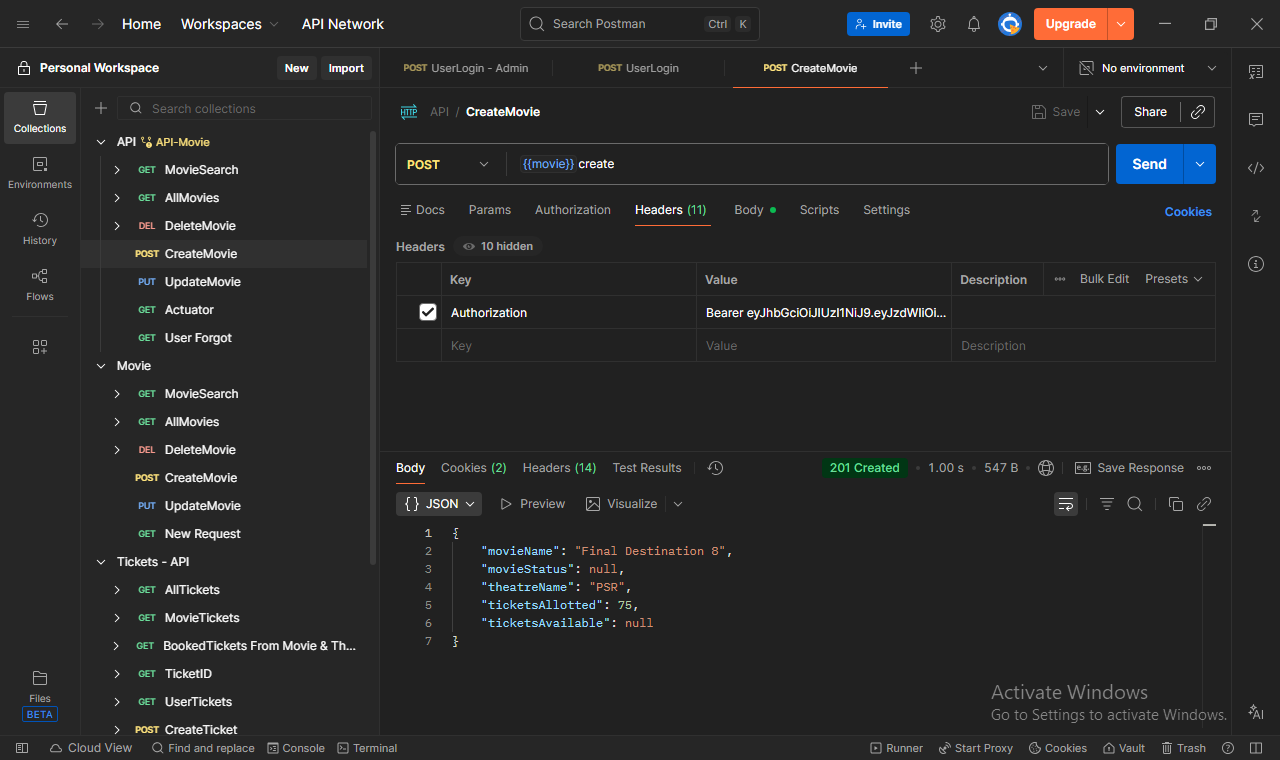
uri: lb://MOVIE-AND-THEATRE-MODULE

predicates:

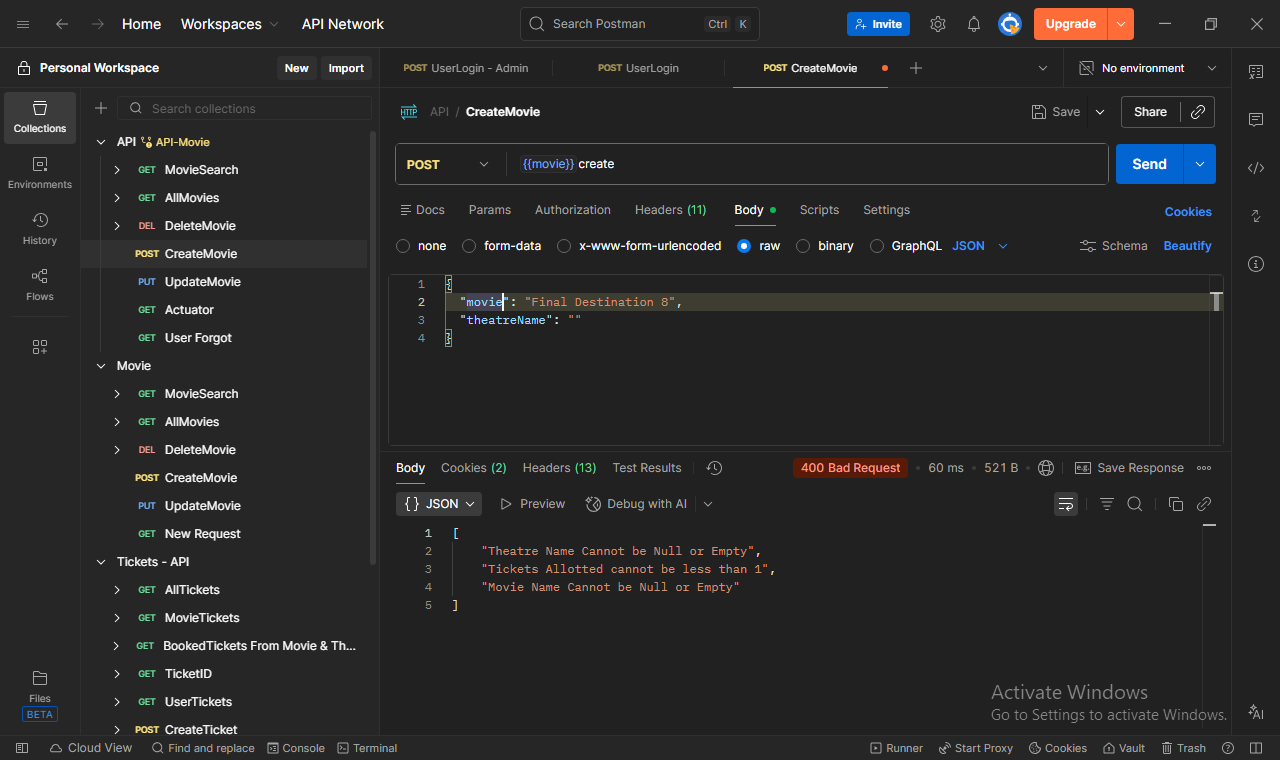
- Path=/api/v1.0/moviebooking/movies/\*\*

**Postman Screenshots for Add Movie:**

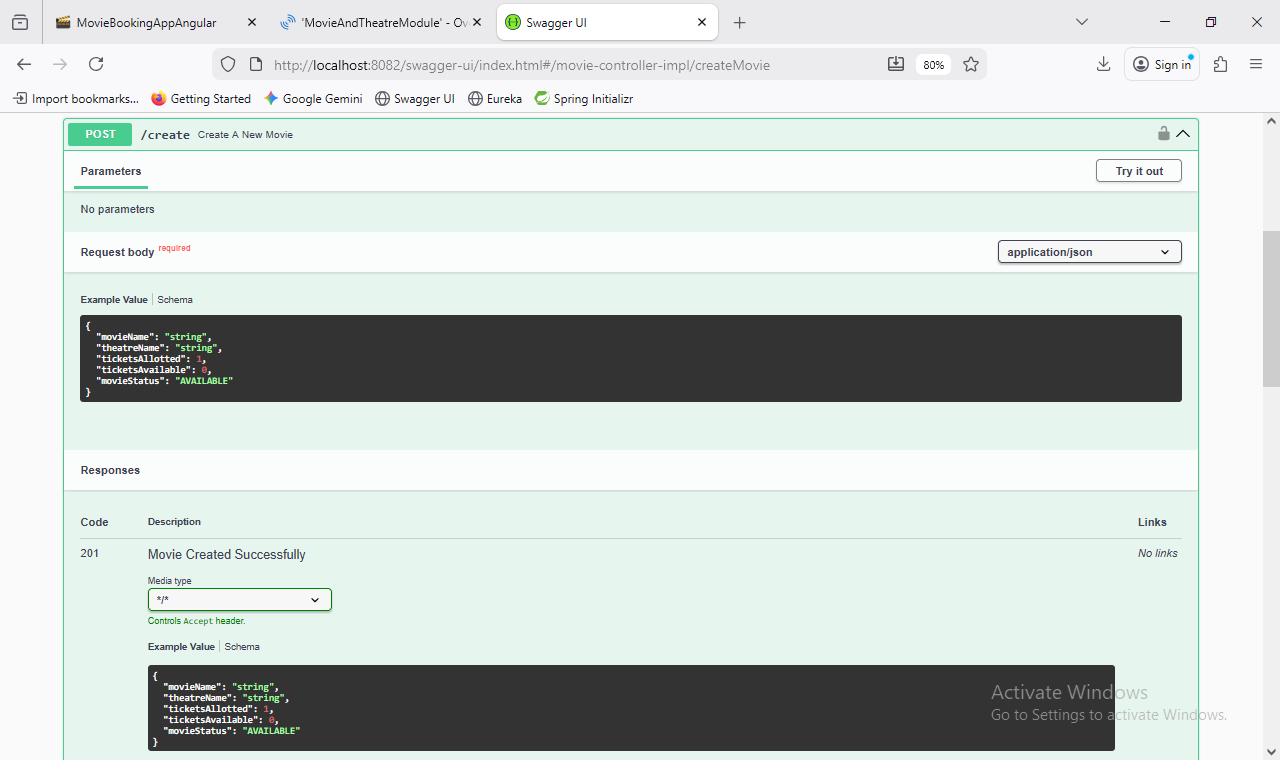
**Admin Adding Movie:**

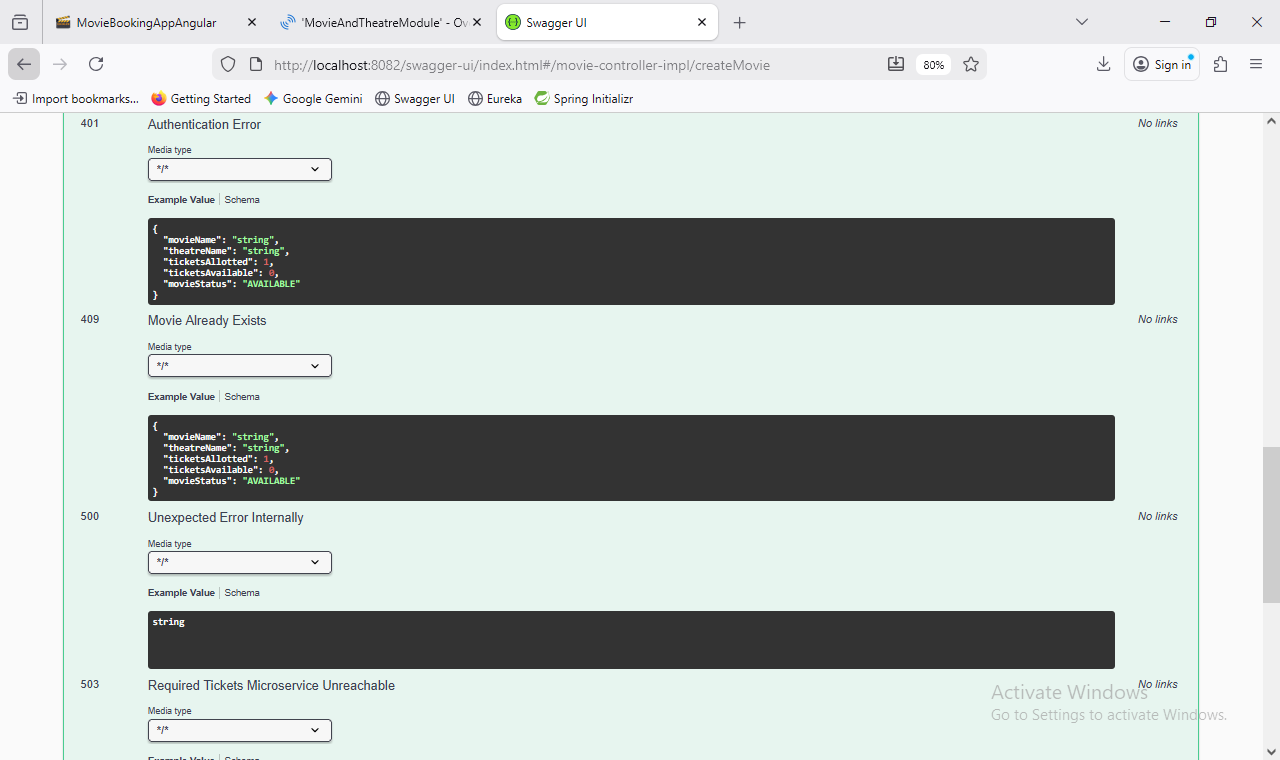


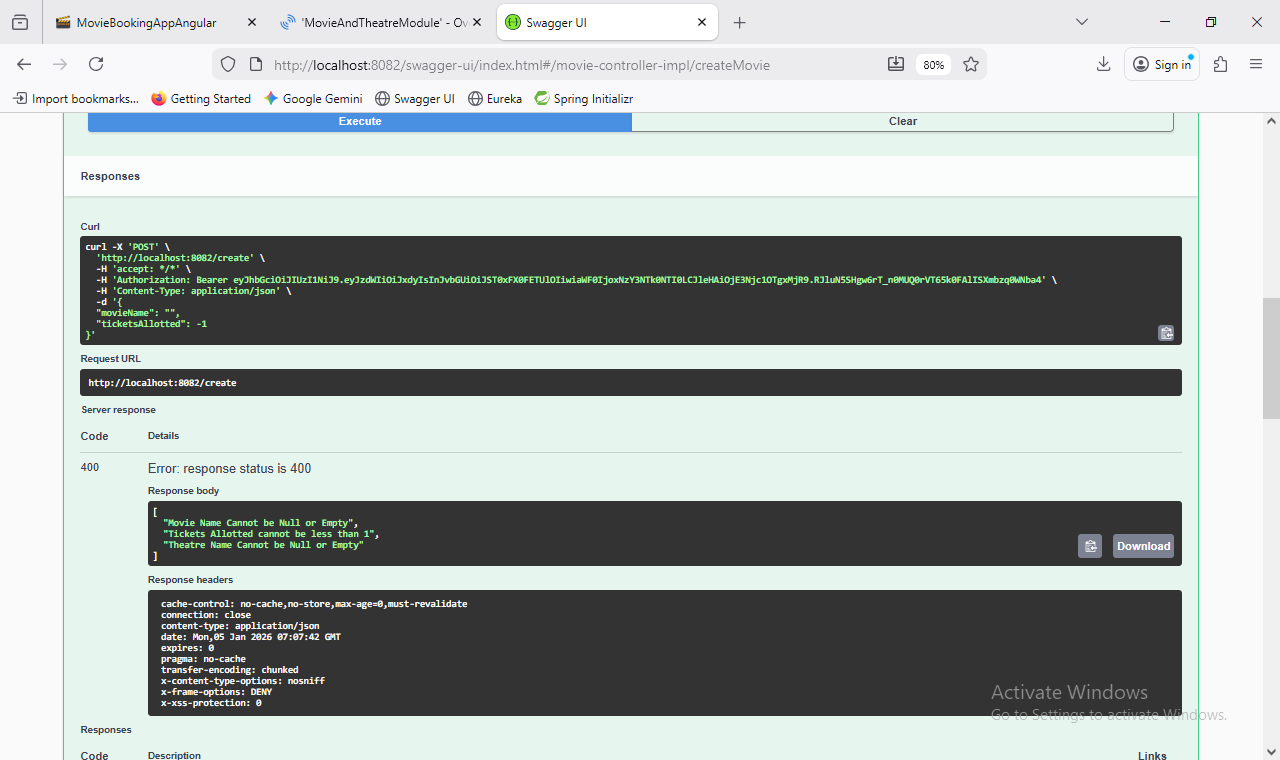
**Admin Adding Movie with incomplete Information:**

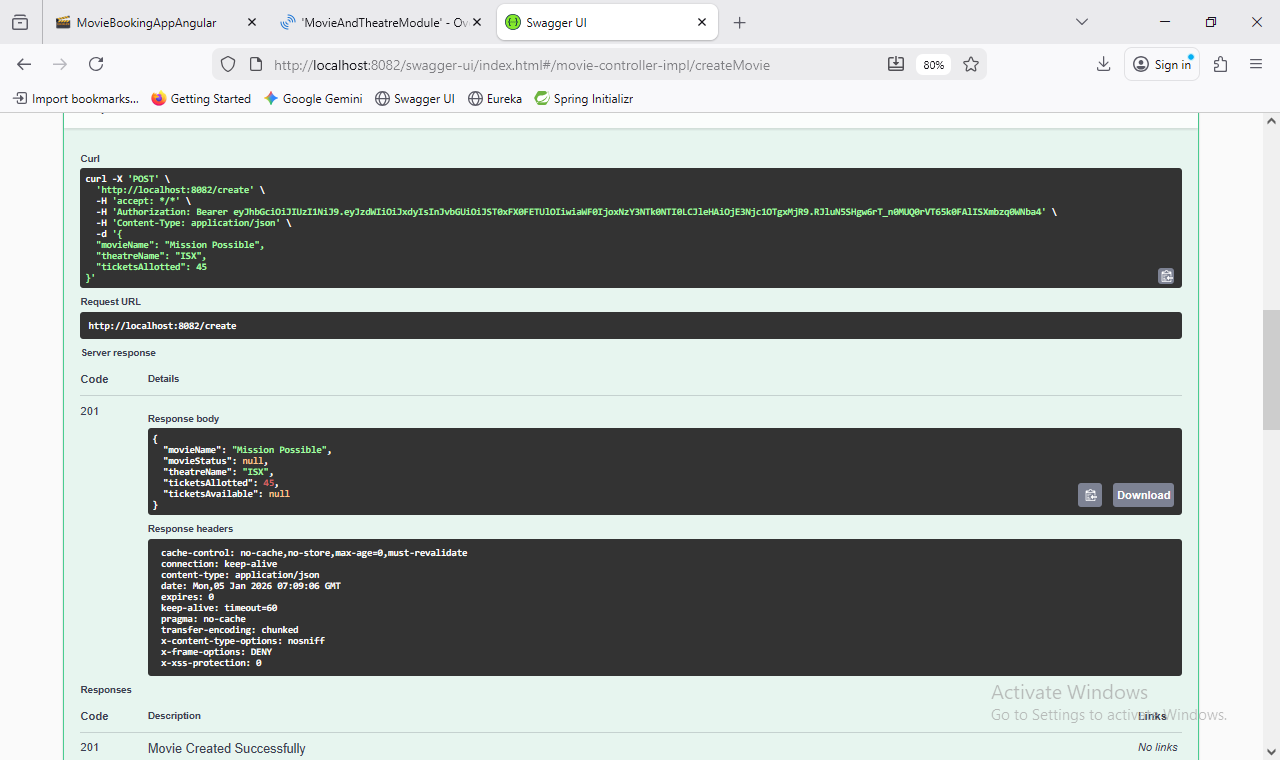


**User trying to Add Movie:**

**Swagger Screenshots for Add Movie:**

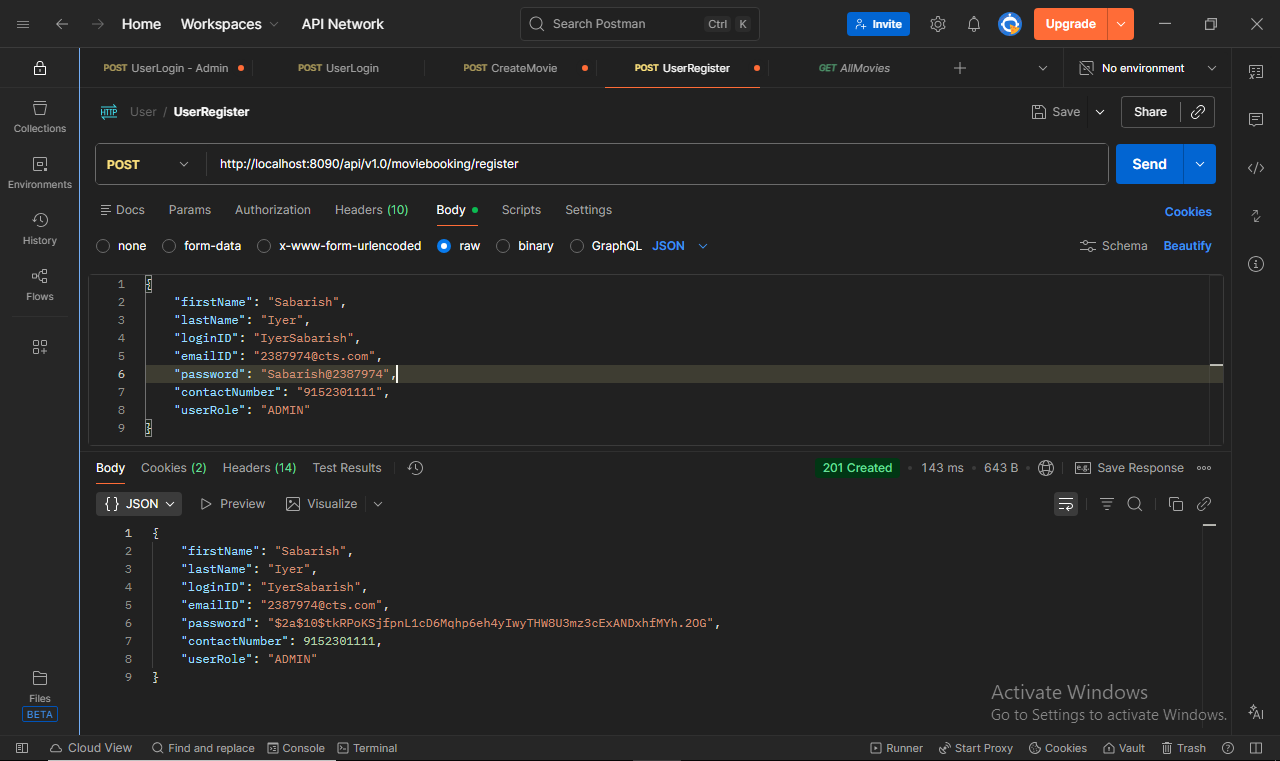




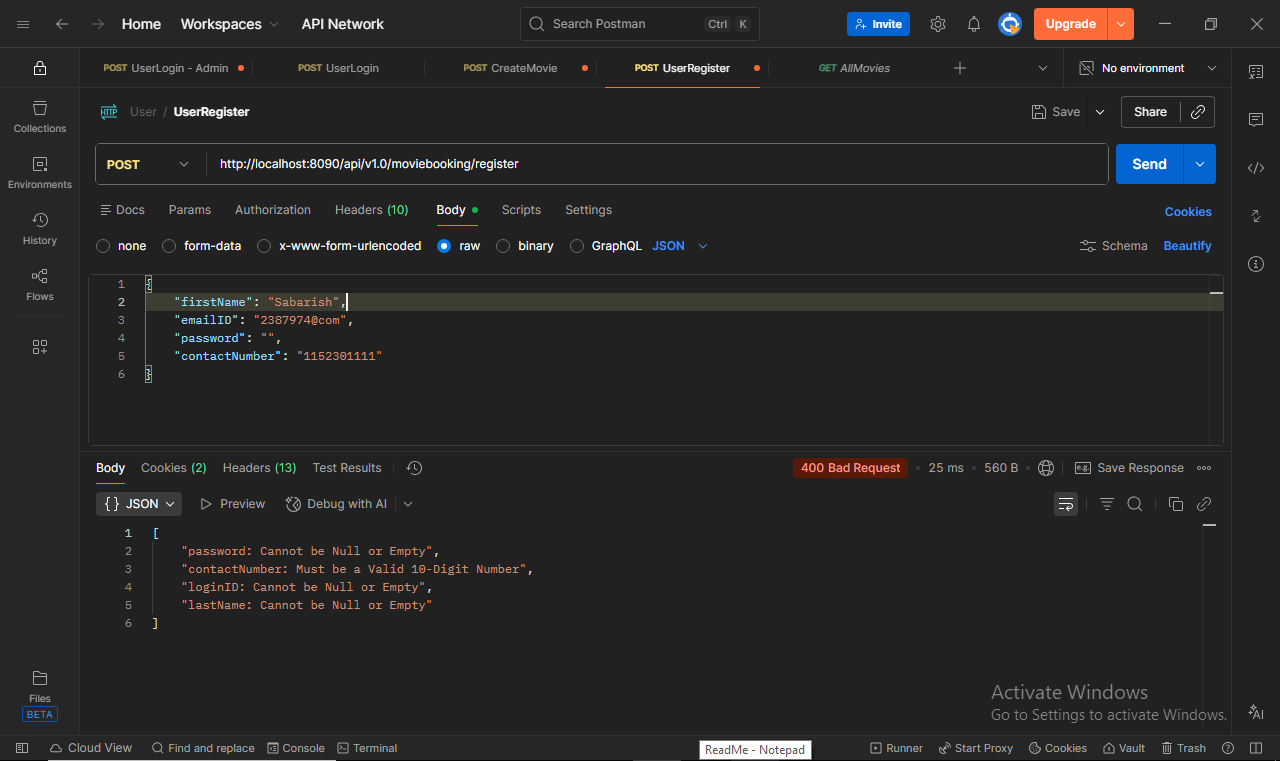


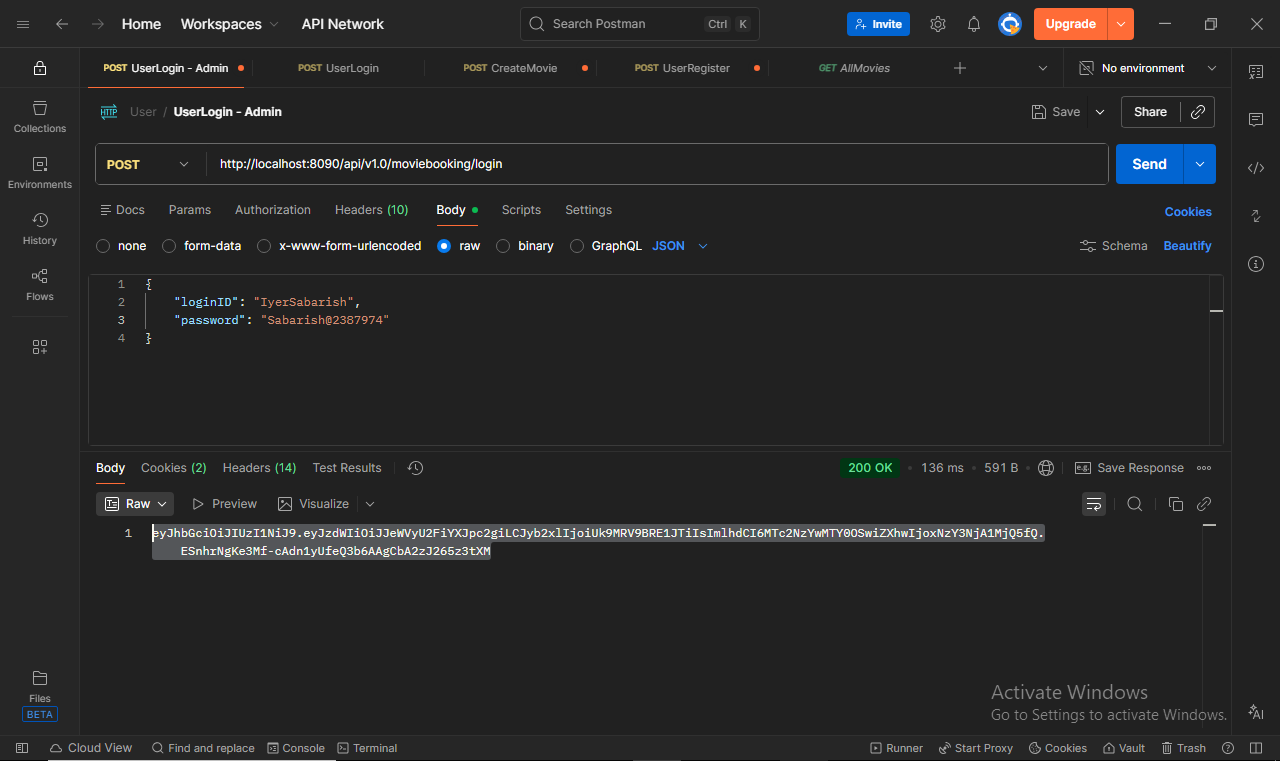
**Postman Screenshots for other Endpoints:**

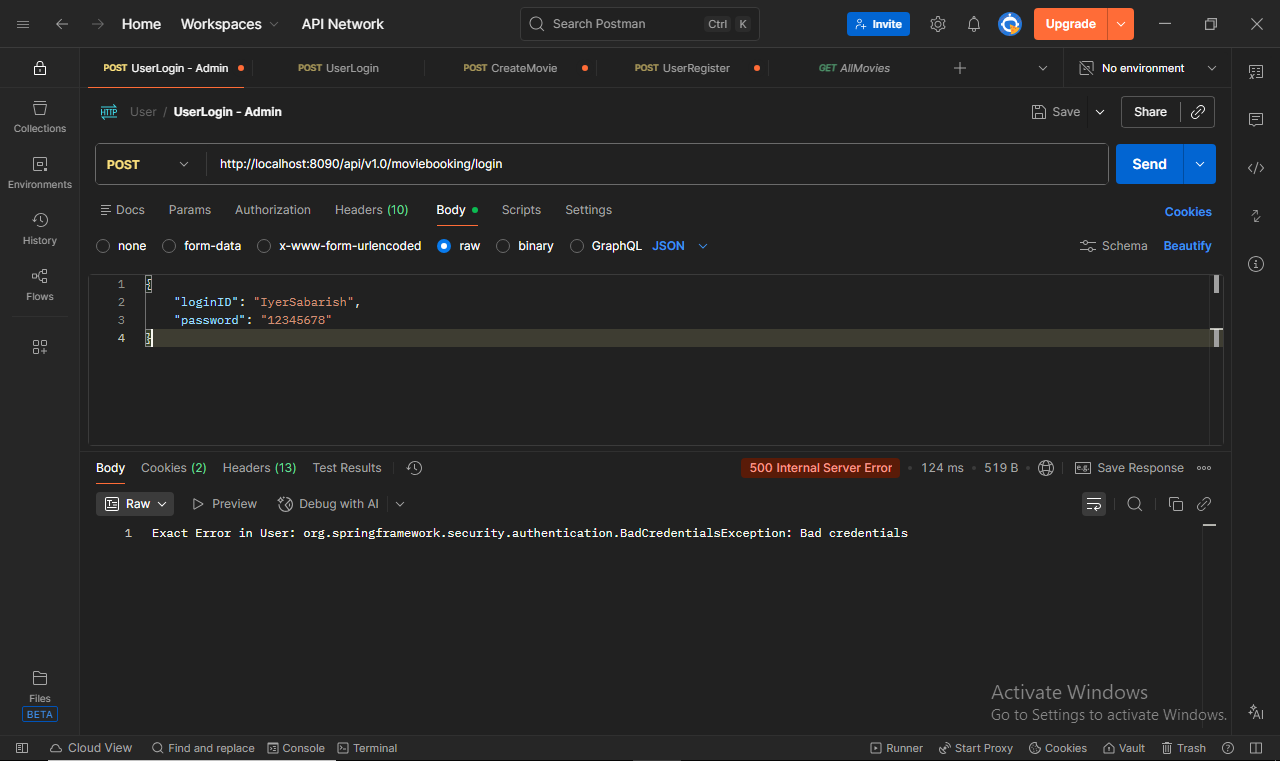
**Register New User:**



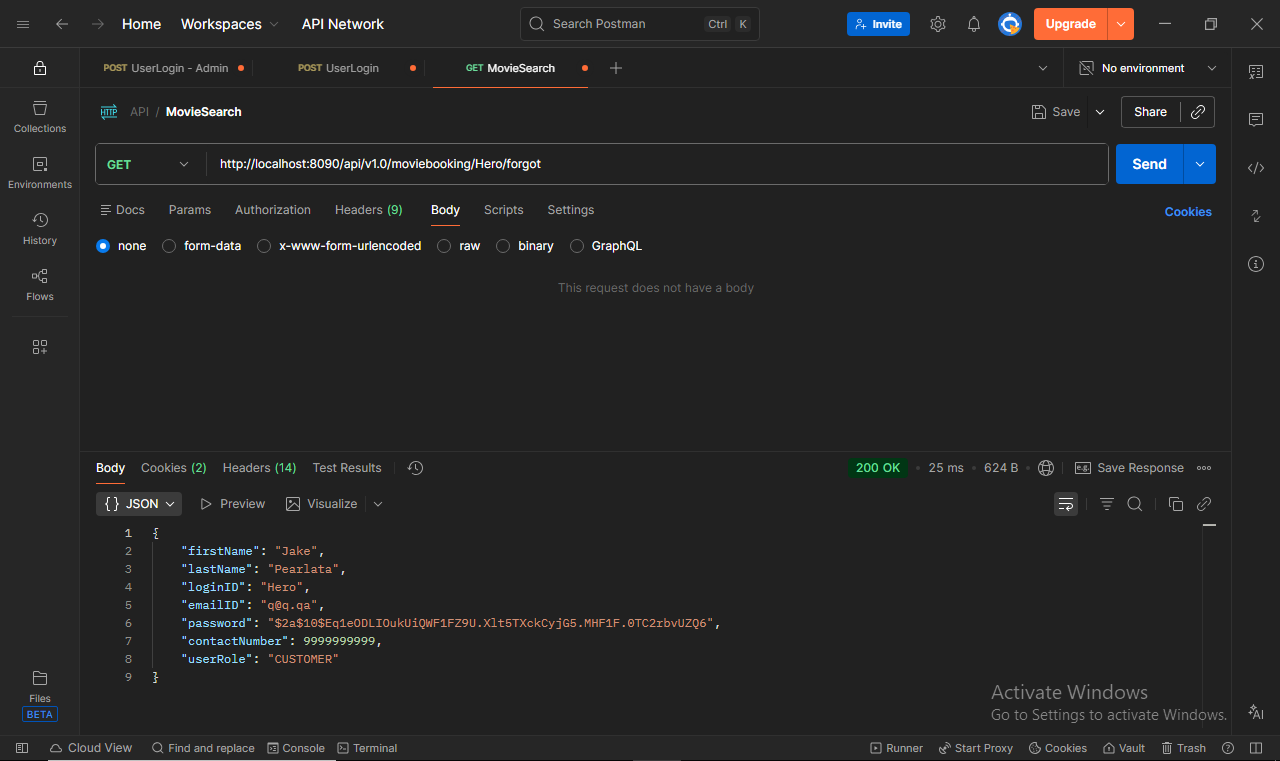
**Register Validation Check:**

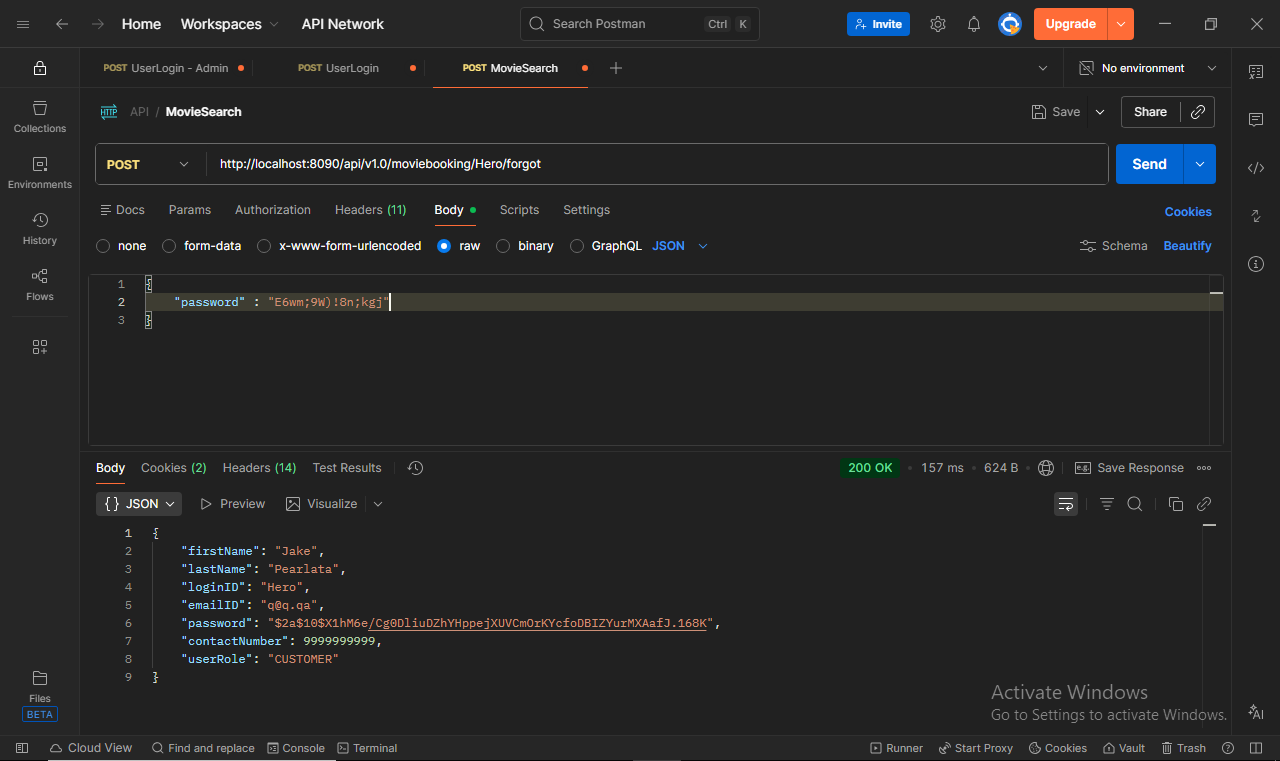


**Login User:**

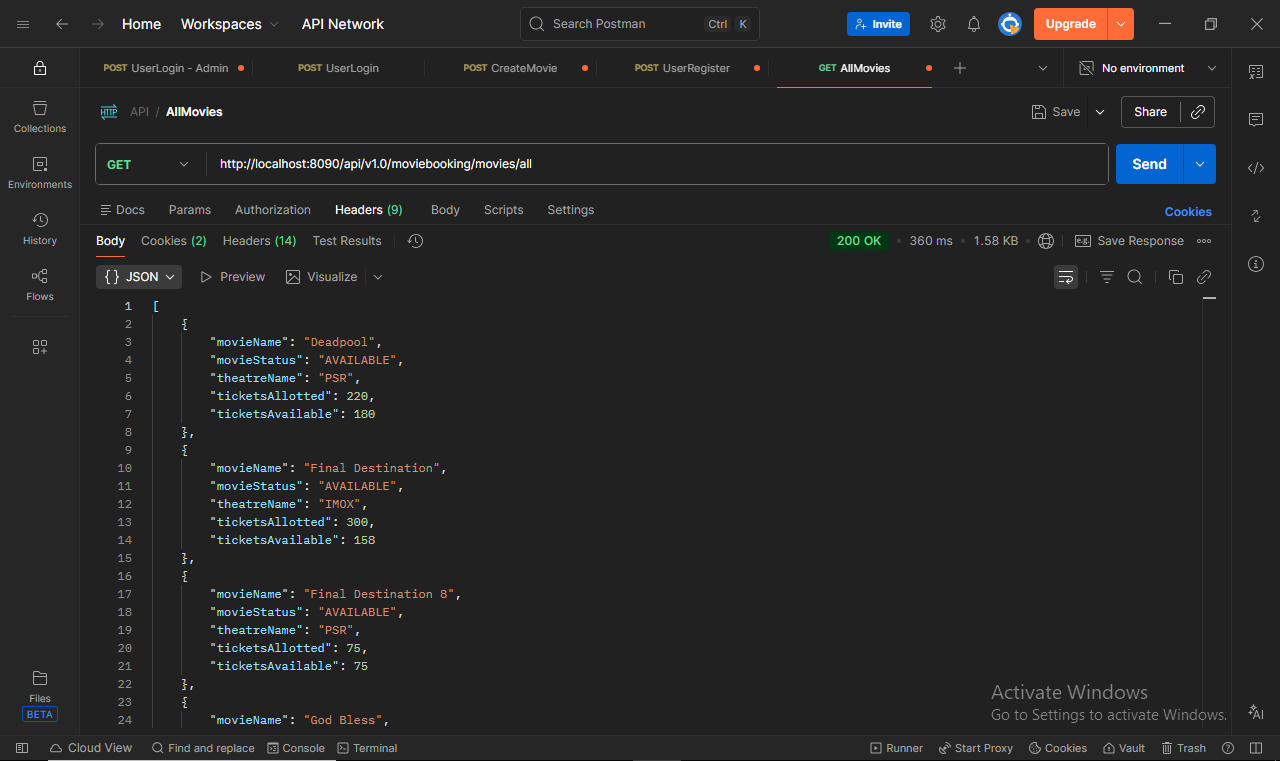
**Login User with wrong Password:**

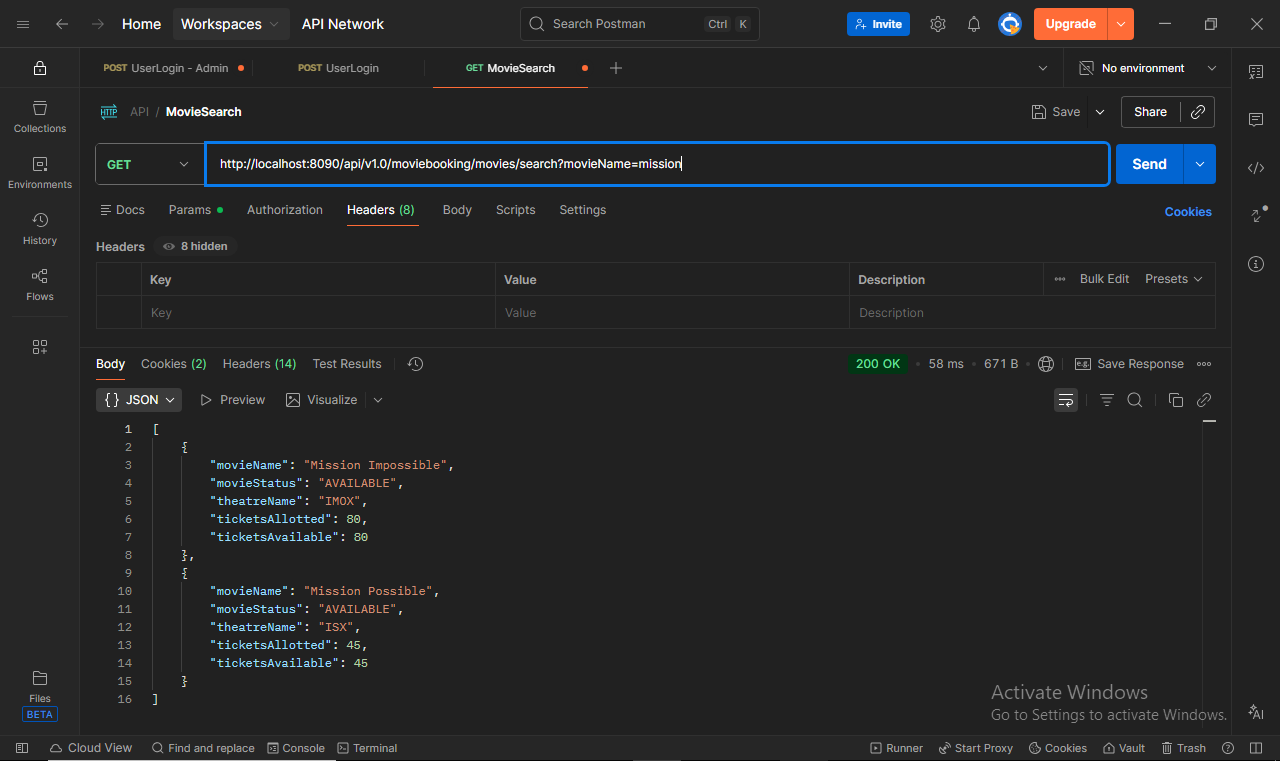
**Forgot password Check:**

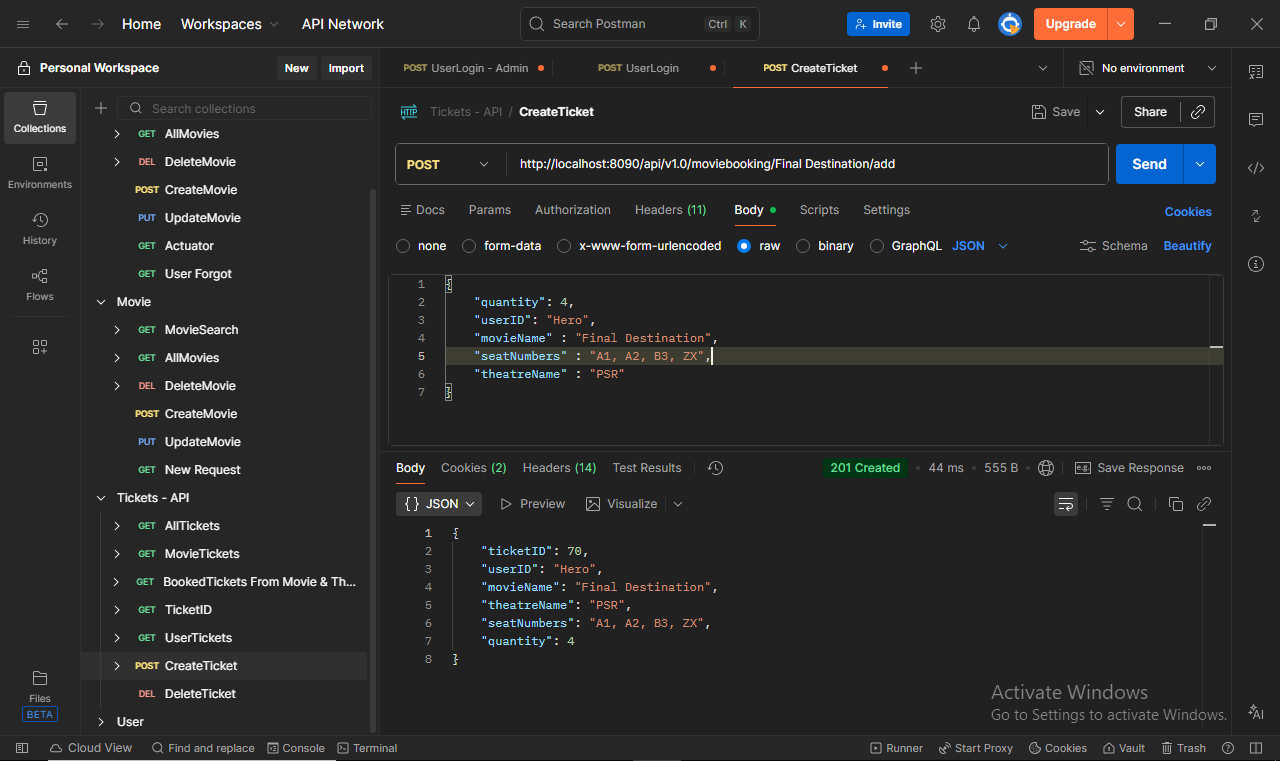


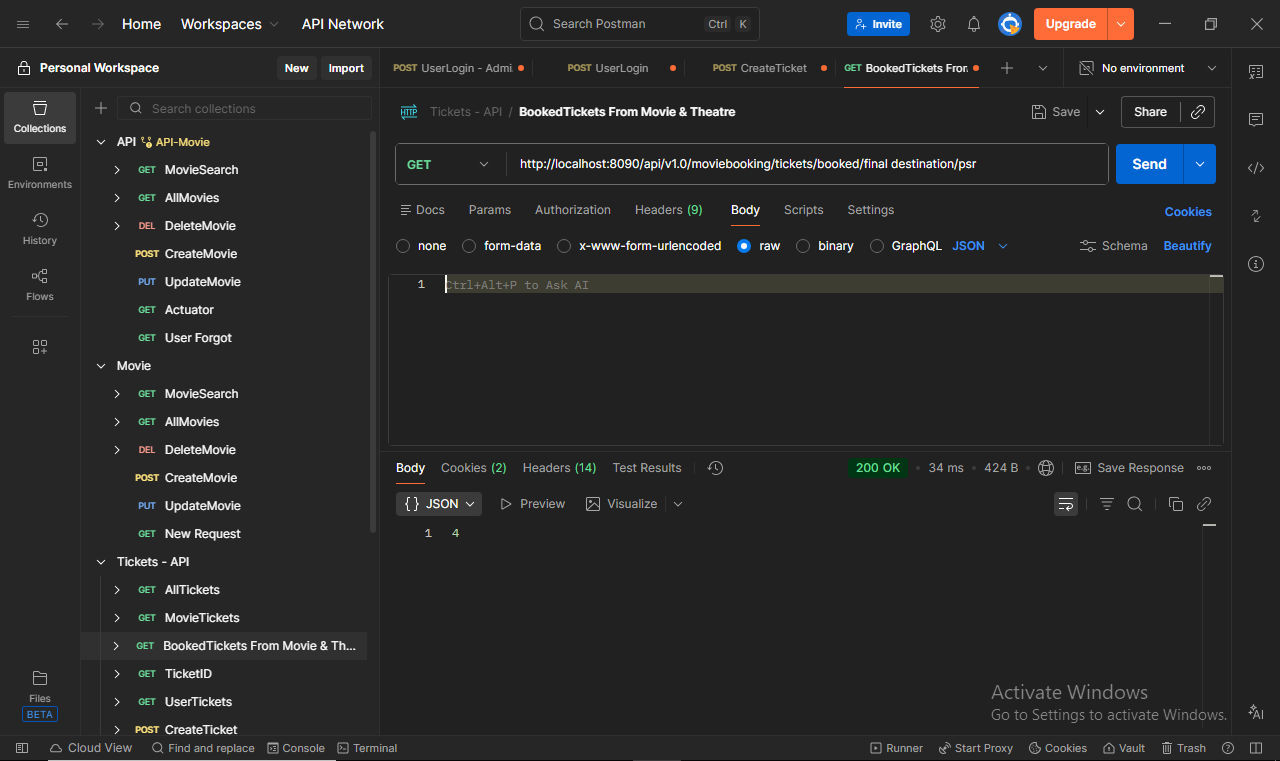
**Forgot Password - New Password:**

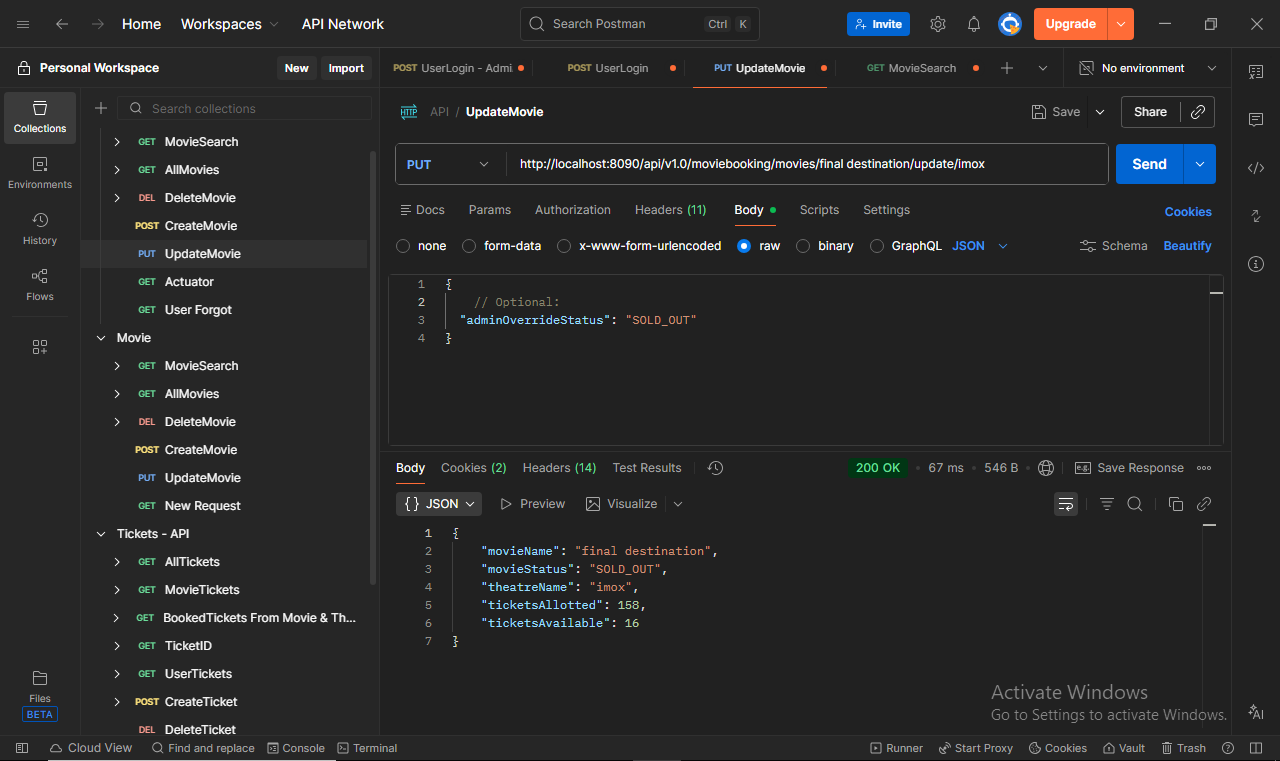
**All Movies:**



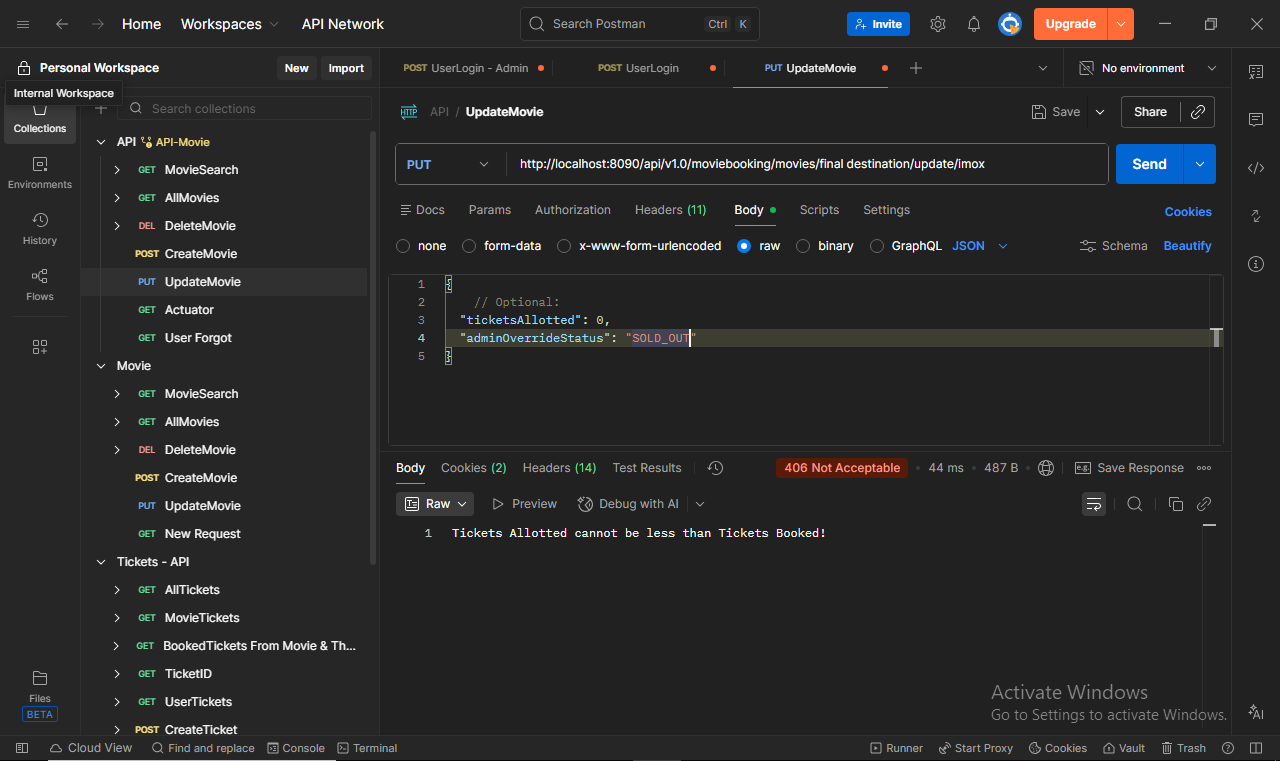
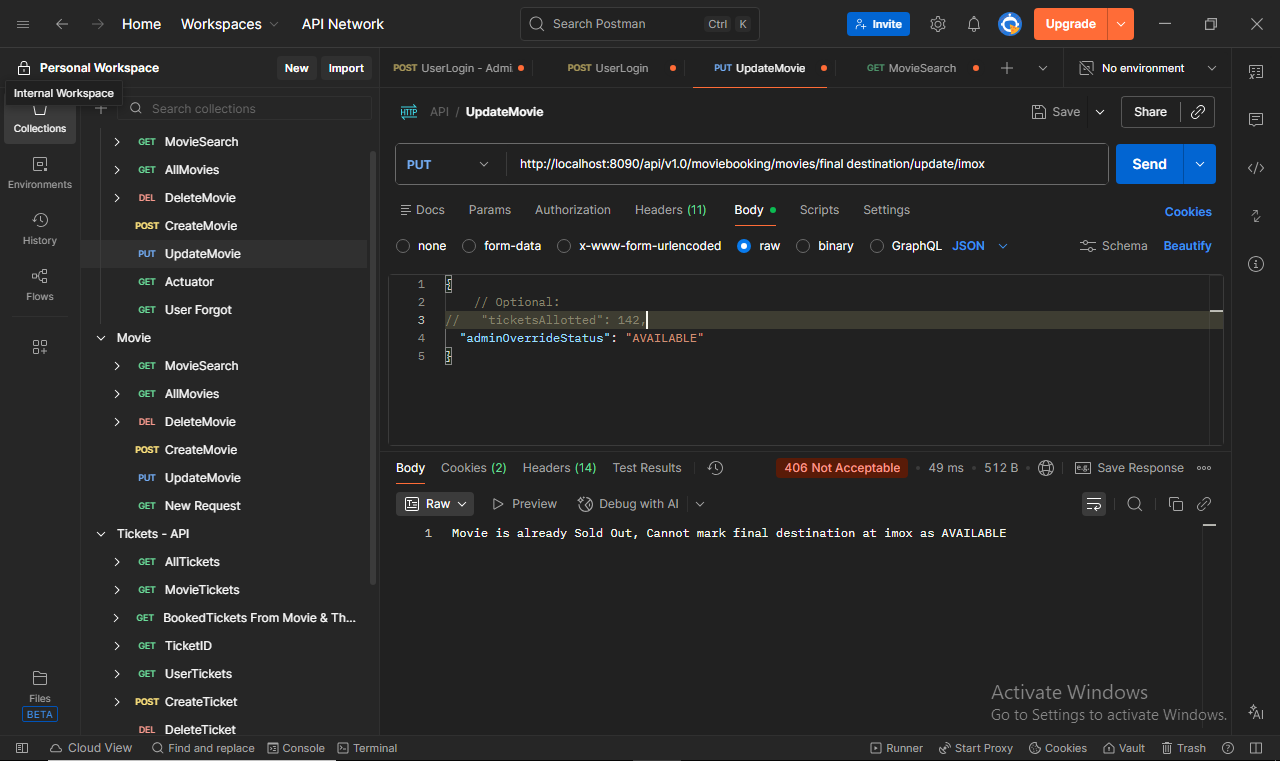
**Movie Search:**

**Book Ticket For A Movie:**

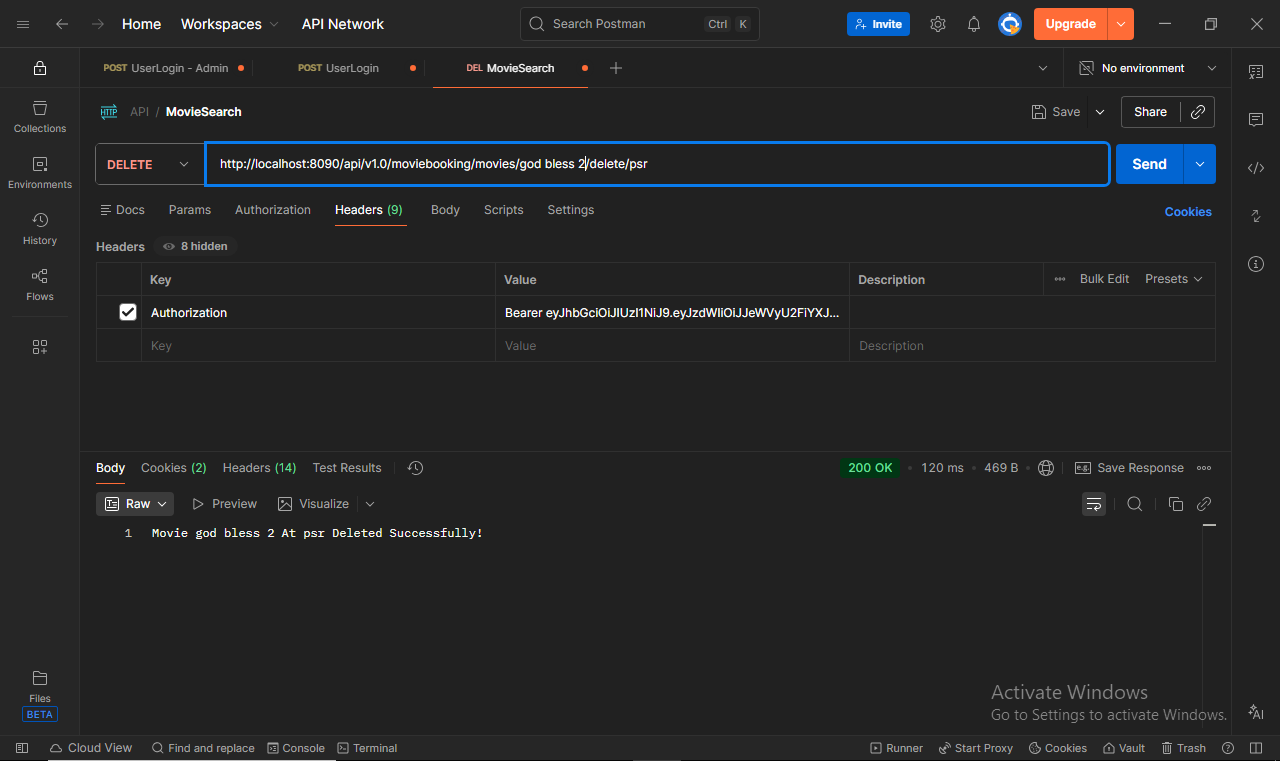
**Get All Tickets For A Movie:**

**Movie Status Change:**

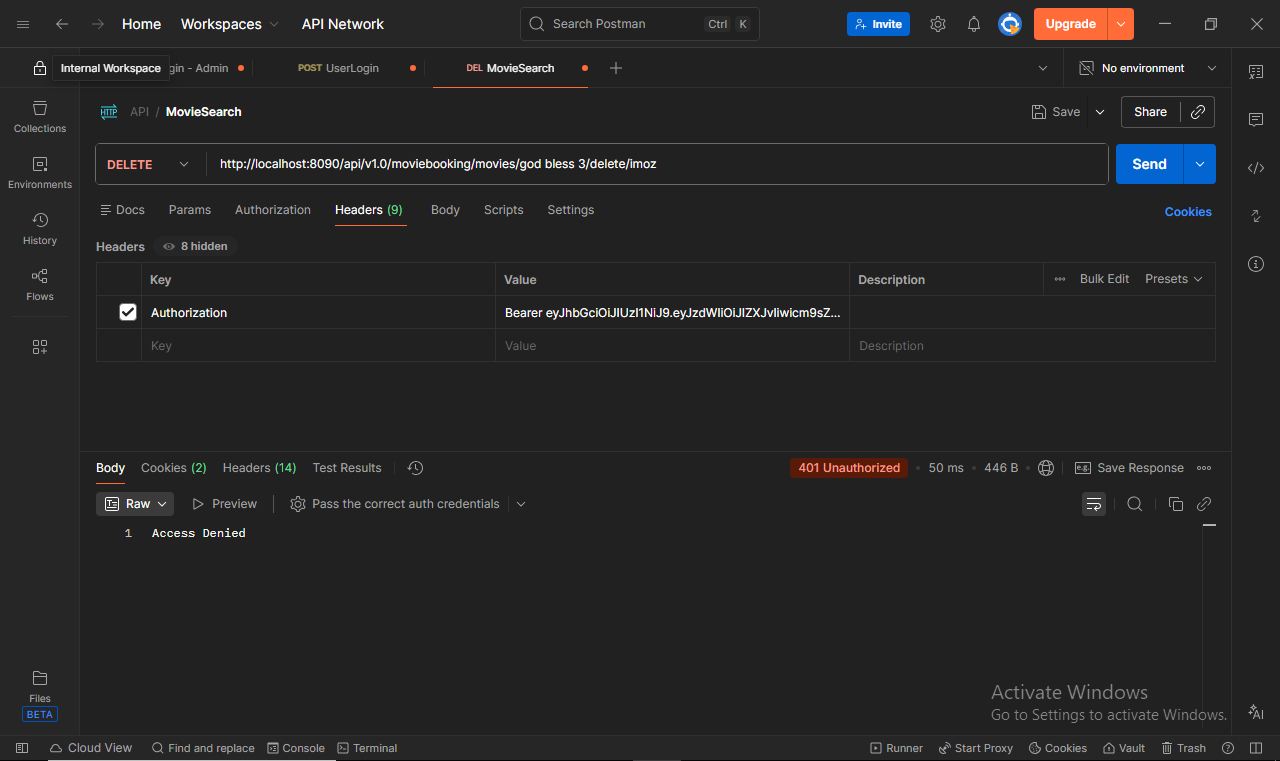
**Invalid Movie Tickets Update:**



**Delete Movie:**



**Delete Movie By User:**



**Caching Implemented:**

/\*\*

\* 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);

}

Ticket cannot be modified, hence can be cached for better performance.

**Session Management:**

httpSecurity

.csrf(AbstractHttpConfigurer::disable)

.authorizeHttpRequests(auth -> auth

.requestMatchers("/swagger-ui/\*\*", "/swagger-ui.html", "/v3/api-docs/\*\*").permitAll()

.requestMatchers(HttpMethod.GET, "/tickets/booked/\*\*").permitAll()

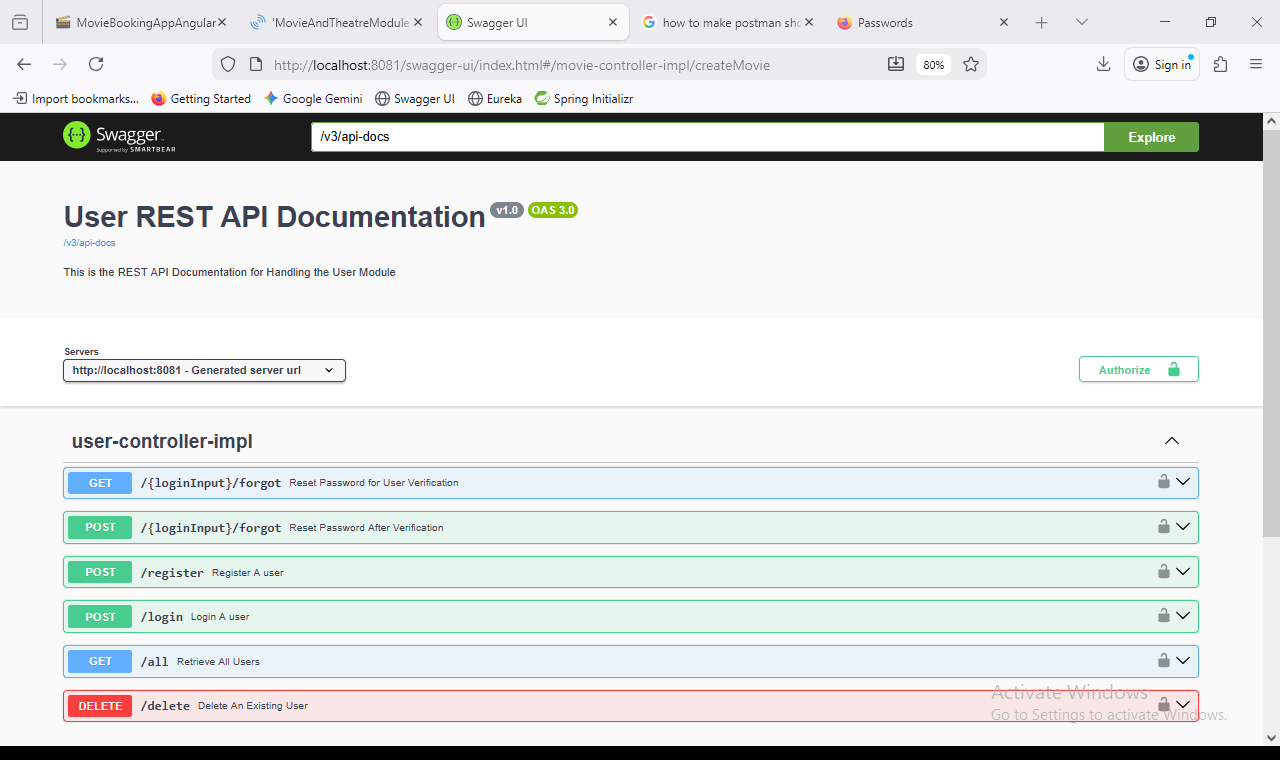
.anyRequest().authenticated()

)

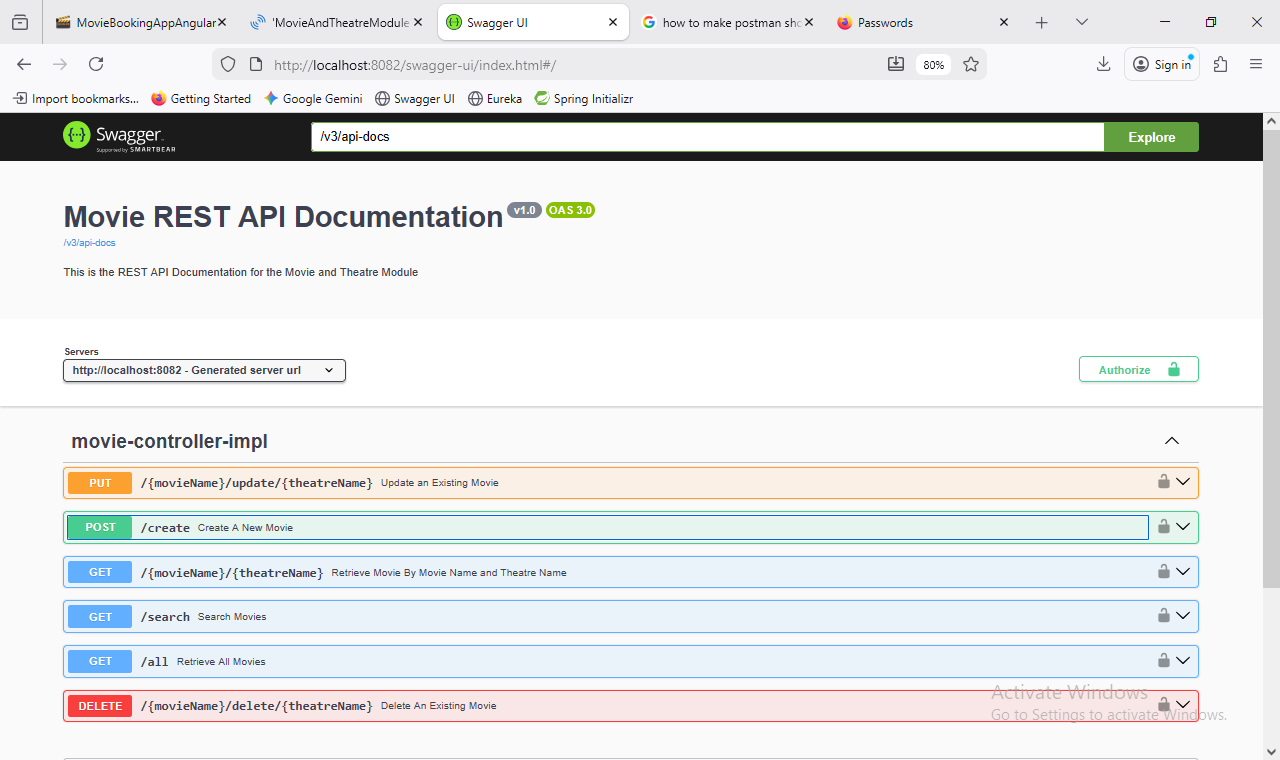
.sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS));

Stateless Session Tokens are used in security.

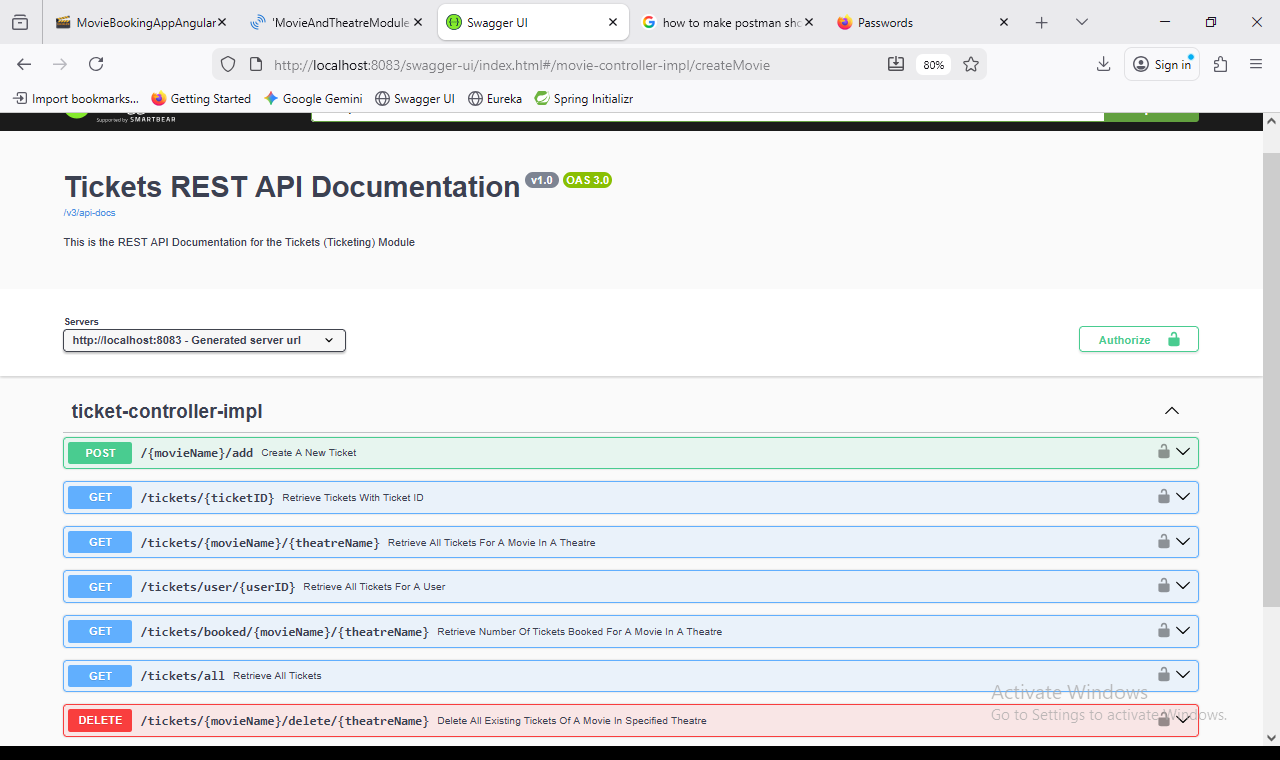
**Swagger - User:**



**Swagger - Movies:**



**Swagger - Tickets:**



**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)

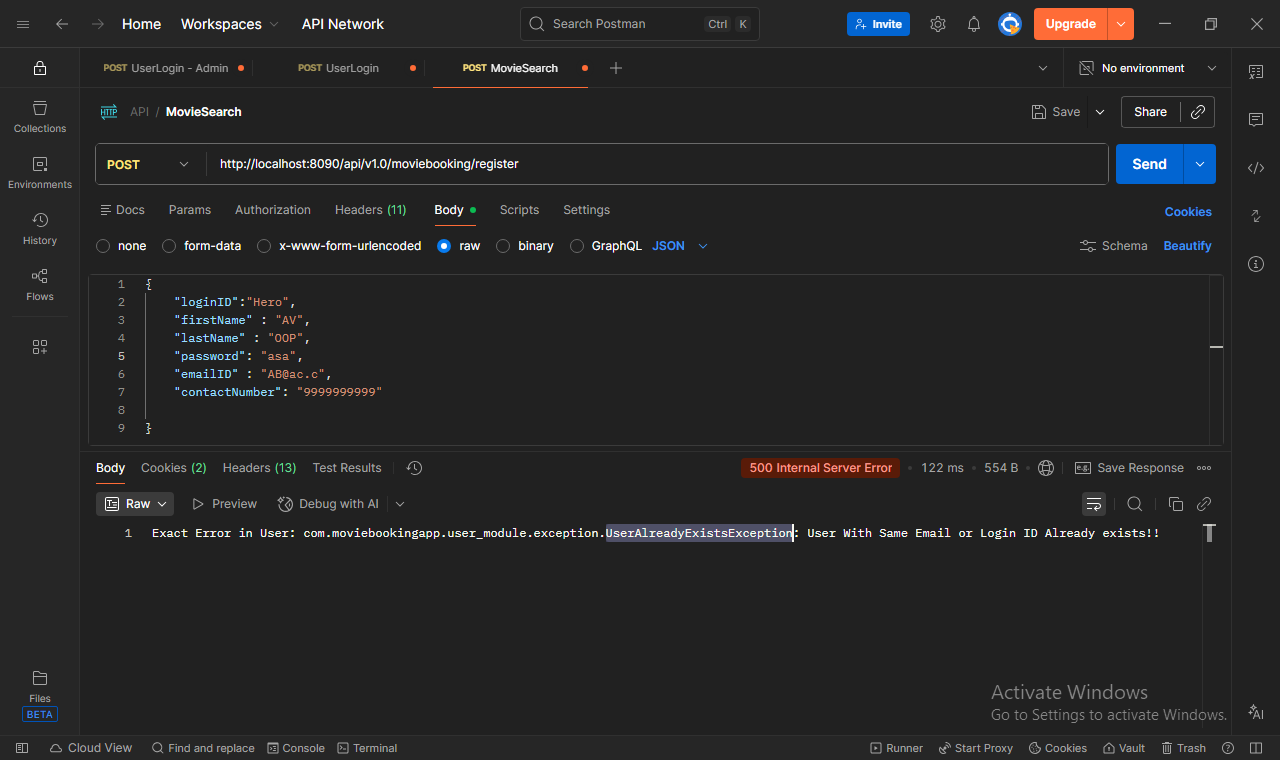
@ResponseStatus(HttpStatus.INTERNAL\_SERVER\_ERROR)

public String handleOtherExceptions(Exception ex) {

return "Exact Error in Movie: " + ex;

}

**Postman Showing Custom Exception:**



**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")

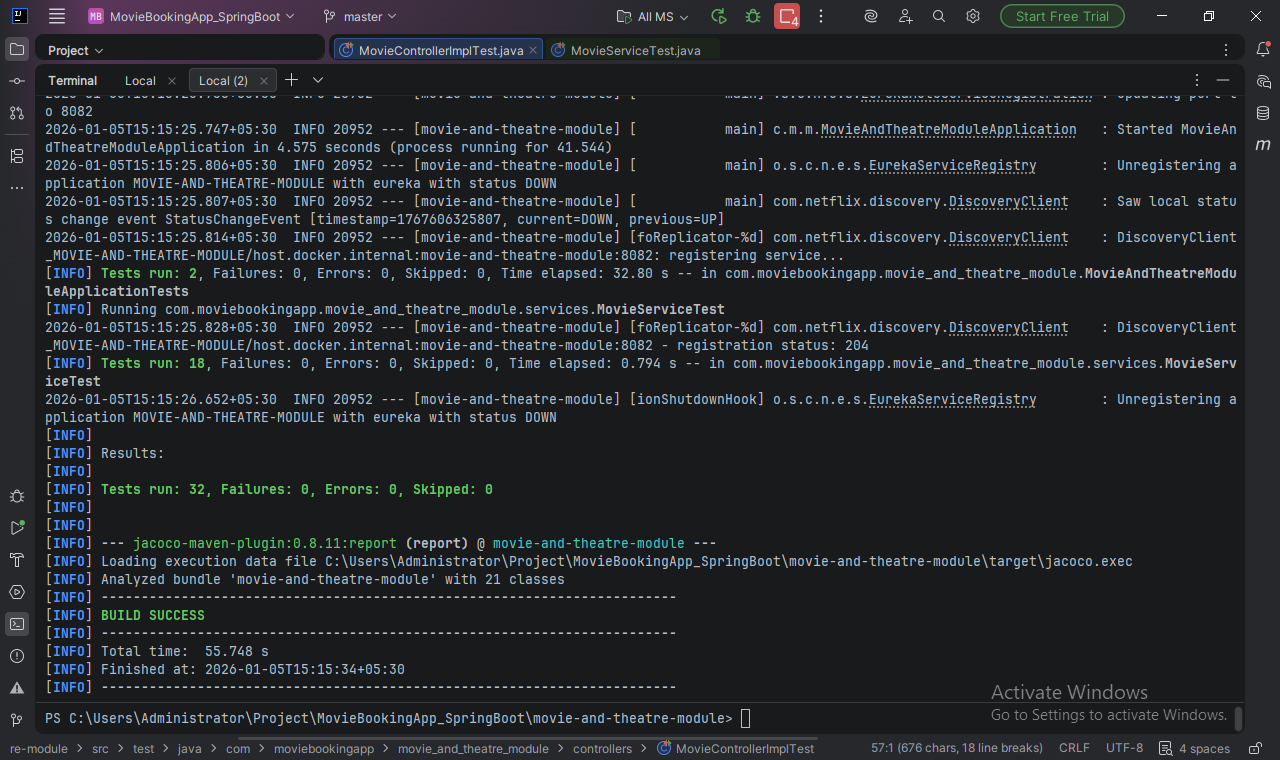
void viewAllMovies\_negative\_movieNotFound() {

when(movieService.getAllMovies()).thenThrow(new MovieNotFoundException());

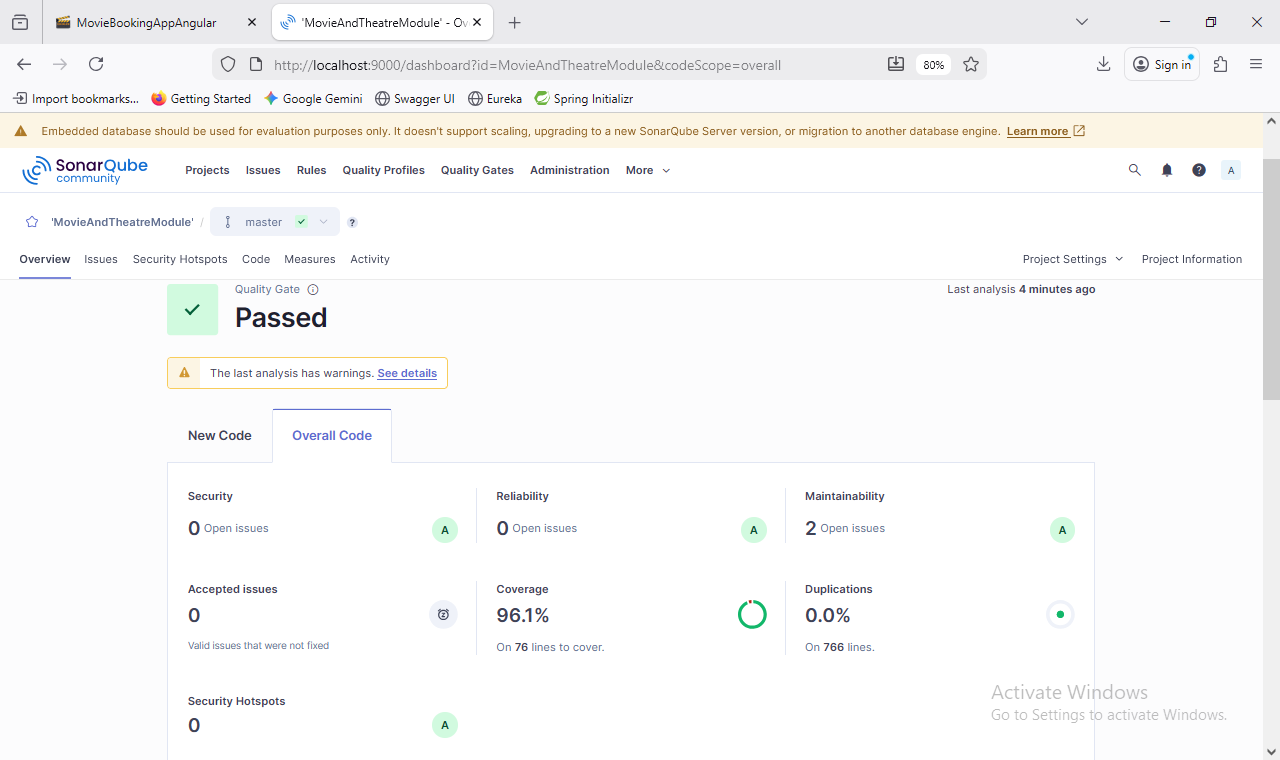
assertThrows(MovieNotFoundException.class, () -> movieController.viewAllMovies());

}

**Maven Test:**

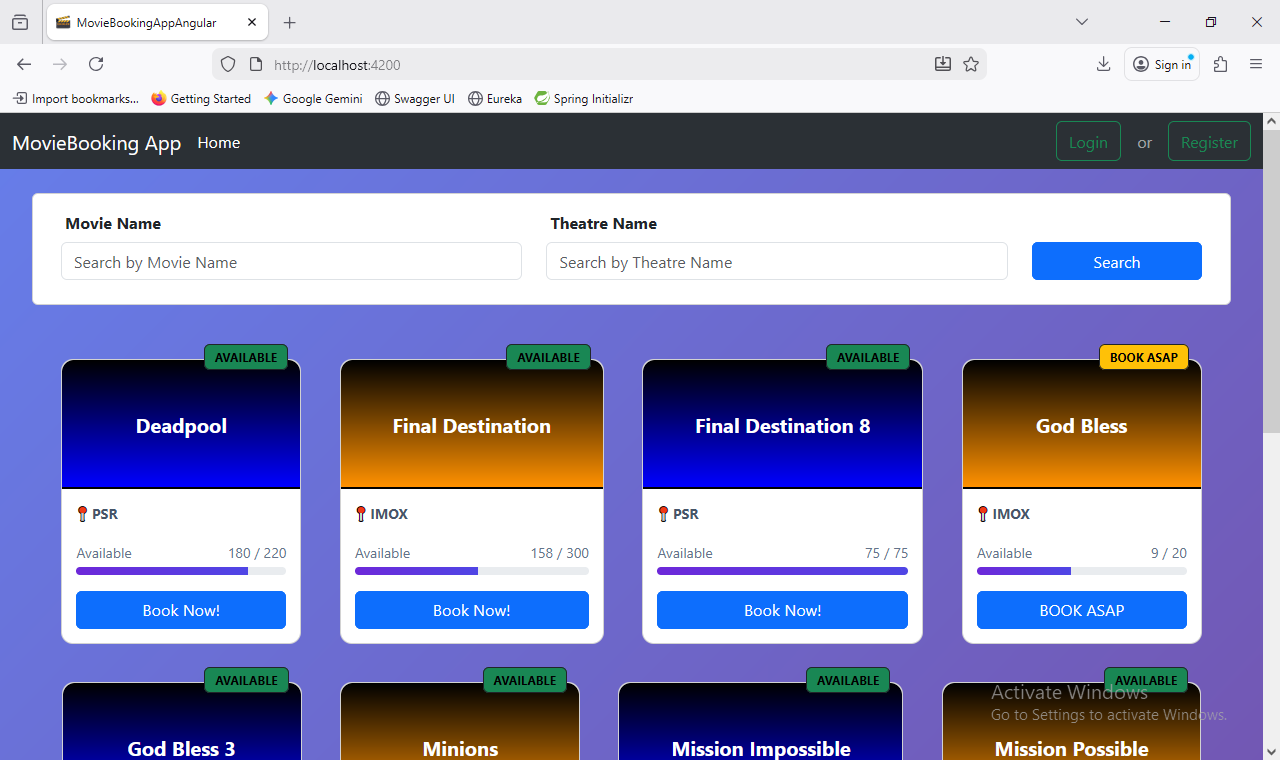


**Code Quality And Coverage:**

**Sonar Qube:**

**Front-End Screenshots:**

**Home Page:**



**Register:**

