

User Manual for Online Movie Booking Application

By:

Soumosir Dutta
sdutta7@umd.edu

Table of Contents

PROJECT INTRODUCTION

Setup guide

API guide

Security Guide

Logging Guide

Validation for apis

Secure Login

Test guide

Project Introduction

Movie Book is an online platform for booking movies. It will be a common platform for both theater owners and customers. We aim to develop a system where theater owners can manage their theaters and add new movies etc. It also offers information about upcoming movies, their timings, cast, and other movie details.

Project Setup :-

1. Java version 11

```
java --version
openjdk 11.0.12 2021-07-20
OpenJDK Runtime Environment Microsoft-25199 (build 11.0.12+7)
OpenJDK 64-Bit Server VM Microsoft-25199 (build 11.0.12+7, mixed mode)
```



2. Development and running test case environment

IntelliJ Idea 2021.2.3



3. <https://dev.mysql.com/downloads/mysql/>

Download mysql server

Run DB in 3306 port

Make a root user and set password as "password"

```
url: jdbc:mysql://localhost:3306/bookmyDB
username: root
password: password
```

If using brew

```
brew services start mysql
mysql_secure_installation
```

```
mysql -u root -p
```

Reference : <https://flaviocopes.com/mysql-how-to-install/>

4. Clone the repo in any directory

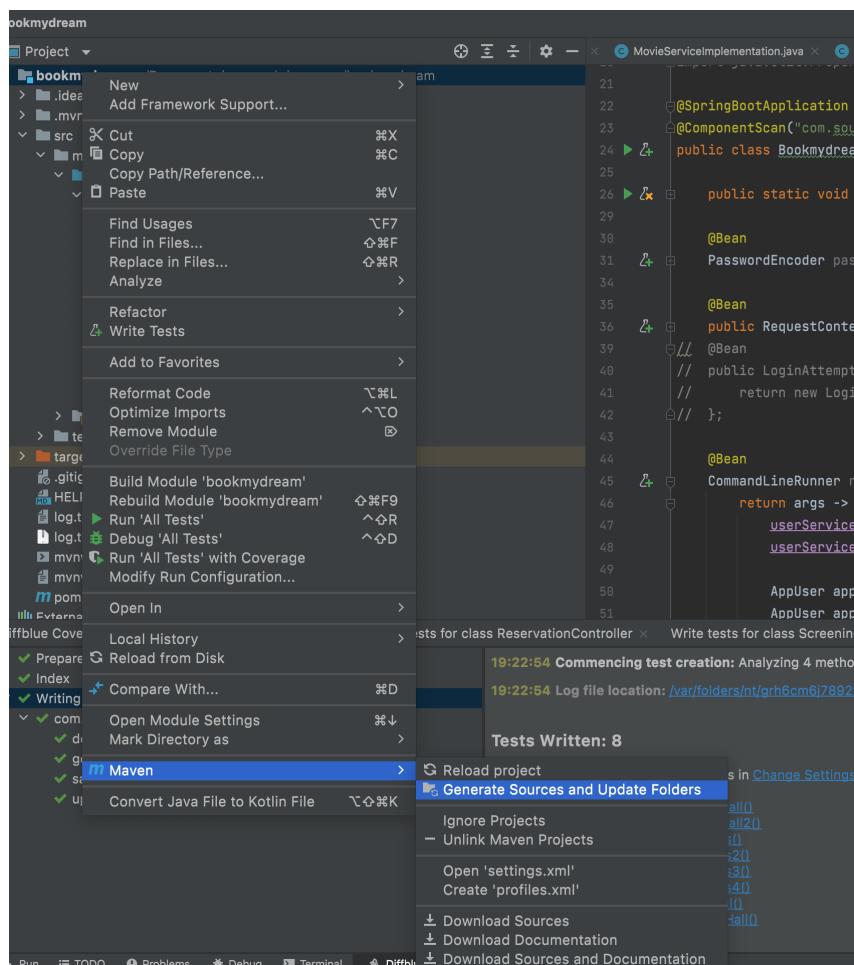
```
git clone git@code.umd.edu:sdutta7/ENPM809WFall2022Project-sdutta7.git
```

Git checkout Phase2

5. Open the folder in intellij

And wait for some time for the project to load

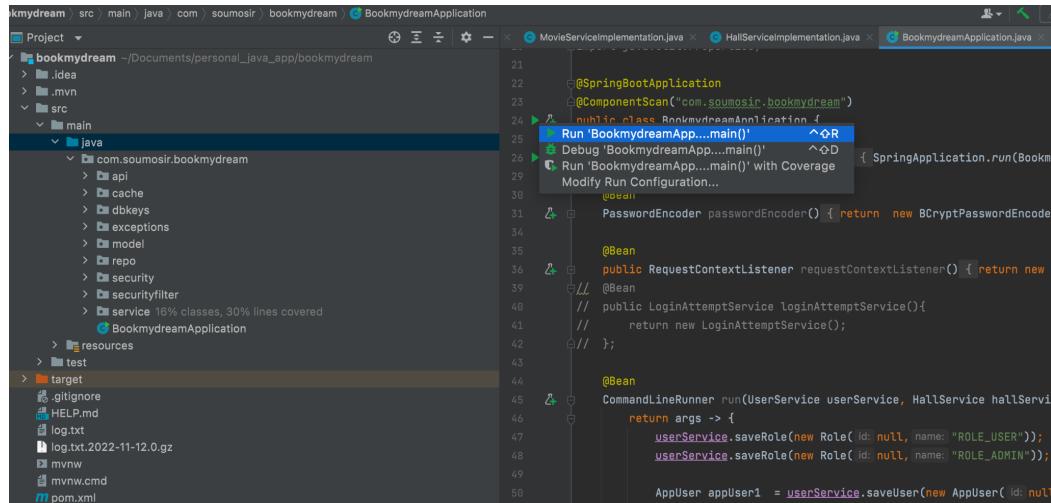
From IntelliJ, you can install maven dependency by installing by clicking pom.xml and selecting reload the project. The screenshot is below for reference.



6.

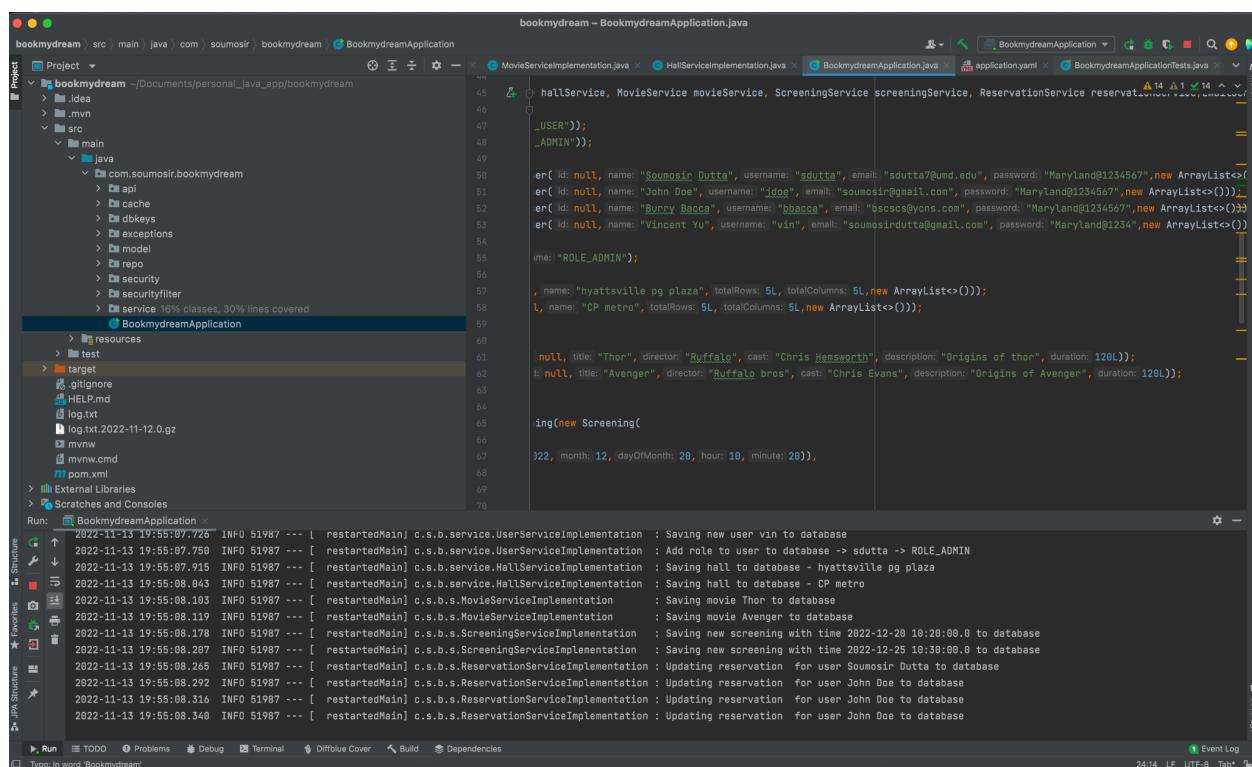
Navigat to src/main/java/com/soumosir/bookmydream/BookmydreamApplication.java
and

Run BookMyDreamAplication.class by the green button



```
1  package com.soumosir.bookmydream;
2
3  import org.springframework.boot.SpringApplication;
4  import org.springframework.boot.autoconfigure.SpringBootApplication;
5  import org.springframework.context.annotation.Bean;
6  import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
7  import org.springframework.web.context.request.RequestContextListener;
8
9  @SpringBootApplication
10 @ComponentScan("com.soumosir.bookmydream")
11 public class BookmydreamApplication {
12
13     public static void main(String[] args) {
14         SpringApplication.run(BookmydreamApplication.class, args);
15     }
16
17     @Bean
18     public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
19
20     @Bean
21     public RequestContextListener requestContextListener() { return new RequestContextListener(); }
22
23     @Bean
24     CommandLineRunner run(UserService userService, HallService hallService) {
25         return args -> {
26             userService.saveRole(new Role(id: null, name: "ROLE_USER"));
27             userService.saveRole(new Role(id: null, name: "ROLE_ADMIN"));
28
29             AppUser appUser1 = userService.saveUser(new AppUser(id: null,
30
31             name: "Soumosir Dutta", username: "sdutta", email: "sdutta7@umd.edu", password: "Maryland@1234567", new ArrayList<>());
32             er(id: null, name: "John Doe", username: "jdoe", email: "soumosir@gmail.com", password: "Maryland@1234567", new ArrayList<>());
33             er(id: null, name: "Burry Bacco", username: "bbacco", email: "bsccscs@vns.com", password: "Maryland@1234567", new ArrayList<>());
34             er(id: null, name: "Vincent Yu", username: "vin", email: "soumosirdu@gmail.com", password: "Maryland@1234", new ArrayList<>());
35
36             im("ROLE_ADMIN");
37
38             , name: "hyattsville pg plaza", totalRows: 5L, totalColumns: 5L, new ArrayList<>());
39             l, name: "CP metro", totalRows: 5L, totalColumns: 5L, new ArrayList<>());
40
41
42             null, title: "Thor", director: "Ruffalo", cast: "Chris Hemsworth", description: "Origins of thor", duration: 128L);
43             null, title: "Avenger", director: "Ruffalo bros", cast: "Chris Evans", description: "Origins of Avenger", duration: 128L);
44
45             ing(new Screening(
46                 2022, month: 12, dayOfMonth: 20, hour: 10, minute: 20));
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
```

One the app is up and running, you should be able to see this in console



```
2022-11-13 19:55:07.726 INFO 51987 --- [ restartedMain] c.s.b.service.UserServiceImplementation : Saving new user vin to database
2022-11-13 19:55:07.750 INFO 51987 --- [ restartedMain] c.s.b.service.UserServiceImplementation : Add role to user to database -> ROLE_ADMIN
2022-11-13 19:55:07.915 INFO 51987 --- [ restartedMain] c.s.b.service.HallServiceImplementation : Saving hall to database - hyattsville pg plaza
2022-11-13 19:55:08.043 INFO 51987 --- [ restartedMain] c.s.b.service.HallServiceImplementation : Saving hall to database - CP metro
2022-11-13 19:55:08.103 INFO 51987 --- [ restartedMain] c.s.b.s.MovieServiceImplementation : Saving movie Thor to database
2022-11-13 19:55:08.119 INFO 51987 --- [ restartedMain] c.s.b.s.MovieServiceImplementation : Saving movie Avenger to database
2022-11-13 19:55:08.178 INFO 51987 --- [ restartedMain] c.s.b.s.ScreeningServiceImplementation : Saving new screening with time 2022-12-20 10:20:00.0 to database
2022-11-13 19:55:08.207 INFO 51987 --- [ restartedMain] c.s.b.s.ScreeningServiceImplementation : Saving new screening with time 2022-12-25 10:30:00.0 to database
2022-11-13 19:55:08.265 INFO 51987 --- [ restartedMain] c.s.b.s.ReservationServiceImplementation : Updating reservation for user Soumosir Dutta to database
2022-11-13 19:55:08.292 INFO 51987 --- [ restartedMain] c.s.b.s.ReservationServiceImplementation : Updating reservation for user John Doe to database
2022-11-13 19:55:08.316 INFO 51987 --- [ restartedMain] c.s.b.s.ReservationServiceImplementation : Updating reservation for user John Doe to database
2022-11-13 19:55:08.340 INFO 51987 --- [ restartedMain] c.s.b.s.ReservationServiceImplementation : Updating reservation for user John Doe to database
```

API - Login - POST - <https://localhost:8443/api/login>

```
--data-urlencode 'username=<string>' \
--data-urlencode 'password=<string>'
```

Example :-

Run the curl command or by post man to login as admin

```
curl -k --location --request POST 'https://localhost:8443/api/login' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'username=sdutta' \
--data-urlencode 'password=Maryland@1234567'
```

Sample Response

```
{"access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWliOjzZHV0dGEiLCJyb2xlcyl6WyJST0xFETUIOliwiUk9MVRV9VU0VSII0sImlzcyI6Imh0dHBzOi8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2luliwiZXhwIjoxNjY4MzkwMTU2fQ.oopZii9RUET9opQTes50g_lBbncncb5cXR3fGnTaFVo", "refresh_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWliOjzZHV0dGEiLCJpc3MiOiJodHRwczovL2xvY2FsaG9zdDo4NDQzL2FwaS9sb2dpbiIsImV4cCI6MTY2ODM5MTM1Nn0.l7ONqcixAqrGbcDhDBey7MBxZORZLnLi-hUpunzu3jM"}
```

OR Postman for

The screenshot shows the Postman interface for a POST request to <https://localhost:8443/api/login>. The 'Body' tab is selected, showing two form-data fields: 'username' (sdutta) and 'password' (Maryland@1234567). The response status is 200 OK, with a response time of 150 ms and a size of 849 B. The response body is displayed in JSON format, containing the access token and refresh token.

KEY	VALUE	DESCRIPTION	...	Bulk Edit
username	sdutta			
password	Maryland@1234567			

```
1 {  
2   "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.  
3     eyJzdWliOjzZHV0dGEiLCJyb2xlcyl6WyJST0xFETUI0IiwiUk9MVRV9VU0VSII0sImlzcyI6Imh0dHBzOi8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2luliwiZXhwIjoxNjY4MzkwMTU2fQ.oopZii9RUET9opQTes50g_lBbncncb5cXR3fGnTaFVo",  
4     "refresh_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.  
5       eyJzdWliOjzZHV0dGEiLCJpc3MiOiJodHRwczovL2xvY2FsaG9zdDo4NDQzL2FwaS9sb2dpbiIsImV4cCI6MTY2ODM5MTM1Nn0.l7ONqcixAqrGbcDhDBey7MBxZORZLnLi-hUpunzu3jM"  
6 }
```

Copy the access token to Authorization bearer For subsequent request

API - Get User by username - GET - <https://localhost:8443/api/user/sdutta> {Auth_token}

Example

```
curl -k --location --request GET 'https://localhost:8443/api/user/sdutta' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWlOIjZHV0dGEiLCJyb2xlcyI6WyJST0xFX0FETUIOliwiUk9MRV9
VU0VSII0sImIzcyI6Imh0dHBzOi8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2lulwiZXhwIjoxNjY4Mzg5NzUzfQ.SGfce9
TzhFU3Z1o3ZT2t4mTbuH6PBrhOFFEB5sQi50' \
--data-binary "
```

OR

The screenshot shows the Postman application interface. At the top, it says "bookmy / user_get". On the right, there are "Save", "Send", and other buttons. Below the header, the method is set to "GET" and the URL is "https://localhost:8443/api/user/sdutta". The "Authorization" tab is selected, showing a "Bearer..." dropdown and a note: "Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables." A token field contains the copied JWT token. Other tabs include "Params", "Headers (9)", "Body", "Pre-request Script", "Tests", "Settings", and "Cookies". At the bottom, the response is displayed in "Pretty" JSON format:

```
1 {"id": 3,
2   "name": "Soumosir Dutta",
3   "username": "sdutta",
4   "email": "sdutta7@umd.edu",
5   "roles": [
6     "ROLE_ADMIN",
7     "ROLE_USER"
8   ]
9 }
10 }
```

API - registering a new user - POST - '<https://localhost:8443/api/user/register>'

```
{  
  "name": "<string>",  
  "username": "<string>",  
  "password": "<string>",  
  "email" : "<string>"  
}
```

Example

```
curl --location --request POST 'https://localhost:8443/api/user/register' \  
--header 'Content-Type: application/json' \  
--data-raw '{  
  "name": "Chanish yuto",  
  "username": "cha7",  
  "password": "MaryLand@123",  
  "email" : "aoumosir@gmail.com"  
}'
```

The screenshot shows the Postman interface with the following details:

- Method:** POST
- URL:** https://localhost:8443/api/user/register
- Body:** JSON (selected)
- Body Content:**

```
1 {  
2   "name": "Chanish yuto",  
3   "username": "cha7",  
4   "password": "MaryLand@123",  
5   "email" : "aoumosir@gmail.com"  
6 }
```
- Response Status:** 201 Created
- Response Body:**

```
1 {  
2   "id": 17,  
3   "name": "Chanish yuto",  
4   "username": "cha7",  
5   "email": "aoumosir@gmail.com",  
6   "roles": [  
7     "ROLE_USER"  
8   ]  
9 }
```

Note there is validation matcher too

The screenshot shows the Postman interface. At the top, it says "bookmy / user_post register". Below that, a "POST" method is selected with the URL "https://localhost:8443/api/user/register". A "Send" button is present. The "Body" tab is open, showing "raw" and "JSON" options, with "JSON" currently selected. The JSON payload is:

```
1 {"name": "Chanish yuto",
2  "username": "cha7",
3  "password": "Mary",
4  "email" : "aoumosir@gmail.com"}  
5  
6
```

Below the body, the response status is 403 Forbidden, with a 161 ms duration and 604 B size. The response body is:

```
1 {"error_message": "Password length should be greater
2  than 8 and less than 30 :Mary :: Email or
3  Username already exists!"}
```

API - Modify a user - PUT - <https://localhost:8443/api/user/<username>> {Auth_token}

```
{  
  "name": "<string>",  
  "username": "<string>",  
  "password": "<string>",  
  "email" : "<string>"  
}
```

Example :

```
curl -k --location --request PUT 'https://localhost:8443/api/user/sdutta' \  
--header 'Authorization: Bearer  
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJzZHV0dGEiLCJyb2xlcyI6WyJST0xFETUIOliwiUk9MRV9VU0VSII0sImIz  
cyl6lmh0dHBzOi8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2luliwiZXhwIjoxNjY4Mzg5NzUzfQ.SGfce9rTzhFU3Z1o3ZT2t4mTbuH6PB  
rhOFfEB5sQj50' \  
--header 'Content-Type: application/json' \  
--data-raw '{  
  "name": "Soumosir Admin",  
  "username": "sdutta",  
  "password": "Mary@1234567",  
  "email" : "oumosir@gmail.com"  
}'
```

PUT https://localhost:8443/api/user/sdutta

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL **JSON**

```

1 {
2   "name": "Soumosir Admin",
3   "username": "sdutta",
4   "password": "Mary@1234567",
5   "email": "oumosir@gmail.com"
6 }
```

Body Cookies Headers (13) Test Results 201 Created 358 ms 581 B

Pretty Raw Preview Visualize JSON

```

1 {
2   "id": 3,
3   "name": "Soumosir Admin",
4   "username": "sdutta",
5   "email": "oumosir@gmail.com",
6   "roles": [
7     "ROLE_ADMIN",
8     "ROLE_USER"
9   ]
10 }
```

Deleting a user

API - Deleting a user - DELETE - <https://localhost:8443/api/user/<username>> {Auth_token}

Example

```
curl -k --location --request DELETE 'https://localhost:8443/api/user/sdutta' \
```

```
--header 'Authorization: Bearer
```

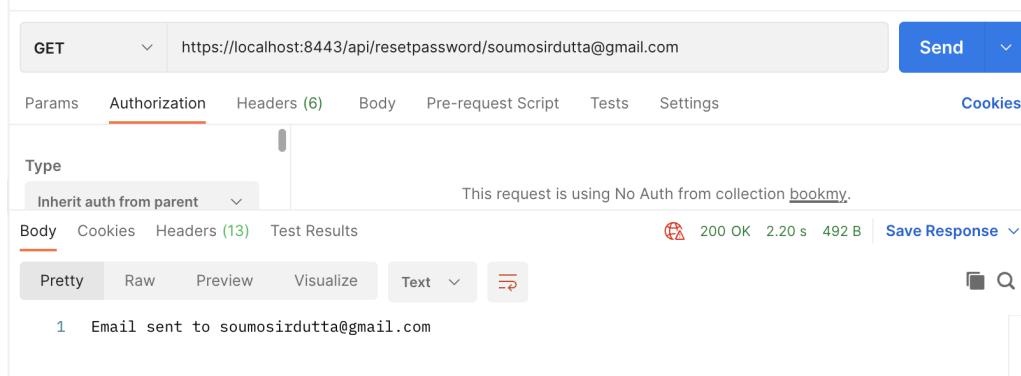
```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWliOiJzZHV0dGEiLCJyb2xlcyI6WyJST0xFX0FETUIOliwiUk9MRV9VU0VSII0sImIzcyl6lmh0dHBzOi8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2luliviZXhwIjoxNjY4Mzg5NzUzfQ.SGfce9rTzhFU3Z1o3ZT2t4mTbuH6PBrhOFfEB5sQj50' \
--header 'Content-Type: application/json' \
--data-raw "
```

RESET a user password (REQUEST)

API - RESET user password - GET - <https://localhost:8443/api/resetpassword/<email>>

Example

```
curl --location --request GET 'https://localhost:8443/api/resetpassword/soumosirdutta@gmail.com'
```



The screenshot shows a Postman interface with a GET request to `https://localhost:8443/api/resetpassword/soumosirdutta@gmail.com`. The response status is 200 OK, 2.20 s, 492 B. The response body contains the message: "1 Email sent to soumosirdutta@gmail.com".

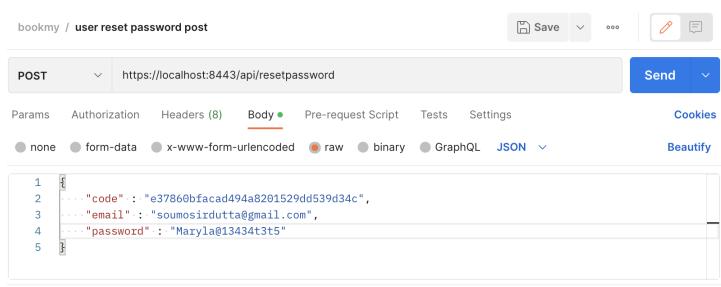
RESET a userpassword (POST)

API - RESET user password - POST - <https://localhost:8443/api/resetpassword/>

```
{
  "code" : "string",
  "email" : "string",
  "password" : "string"
}
```

Example

```
curl --location --request POST 'https://localhost:8443/api/resetpassword' \
--header 'Content-Type: application/json' \
--data-raw '{
  "code" : "e37860bfac494a8201529dd539d34c",
  "email" : "soumosirdutta@gmail.com",
  "password" : "Maryla@13434t3t5"
}'
```



The screenshot shows a Postman interface with a POST request to `https://localhost:8443/api/resetpassword`. The request body is a JSON object with fields: "code": "e37860bfac494a8201529dd539d34c", "email": "soumosirdutta@gmail.com", "password": "Maryla@13434t3t5".

GET all the movies (USER and ADMIN allowed)

API - GET all the movies - GET - <https://localhost:8443/api/movie> {Auth_token}

Example :-

```
curl --location --request GET 'https://localhost:8443/api/movie' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWlOjzZHV0dGEiLCJyb2xlcyI6WyJST0xFX0FETUIOliwiUk9MRV9
VU0VSII0sImzcyl6lmh0dHBzOj8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2luliwiZXhwIjoxNjY4Mzg5NzUzfQ.SGfce9r
TzhFU3Z1o3ZT2t4mTbuH6PBrhOFfEB5sQj50'
```

The screenshot shows the Postman application interface. At the top, it says "bookmy / movies_get". Below that, a "GET" request is selected with the URL "https://localhost:8443/api/movie". The "Authorization" tab is active, showing "Bearer Token" selected. A note in a callout box says: "Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables." Below this, there's a "Token" field containing a long JWT token. The "Body" tab is selected, showing a JSON response with two movie objects:

```
1 [{"id": 9, "title": "Thor", "director": "Ruffalo", "cast": "Chris Hemsworth", "description": "Origins of thor", "duration": 120}, {"id": 10, "title": "Avenger", "director": "Ruffalo bros", "cast": "Chris Evans", "description": "Origins of Avenger", "duration": 120}]
```

Add a movie (ADMIN Only)

API - add the movies - POST - <https://localhost:8443/api/movie> {Auth_token}

```
{  
  "duration": number, "title": string, "cast": string "director": string "description": string  
}  
}
```

Example :-

```
curl -k --location --request POST 'https://localhost:8443/api/movie/' \  
-header 'Authorization: Bearer  
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9eyJzdWliOjzZHV0dGEiLCJyb2xlcyI6WyJST0xFX0FETUIOliwiUk9MRV9VU0VSII0sImlzcyI6Imh0dHBzOi8v  
bG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2luliwiZXhwIjoxNjY4Mzg5NzUzfQ.SGfce9rTzhFU3Z1o3ZT2t4mTbuH6PBrhOFFEB5sQj50' \  
-header 'Content-Type: application/json' \  
-data-raw '{  
  "duration": 60,  
  "title": "ABBAAtt",  
  "cast": "rittiktt",  
  "director": "A234gtt",  
  "description": "Sensational Movie"  
}'
```

The screenshot shows the Postman application interface. At the top, there is a header bar with 'POST' selected, a URL field containing 'https://localhost:8443/api/movie/', and a 'Send' button. Below the header are tabs for 'Params', 'Authorization' (which is set to 'Bearer'), 'Headers (9)', 'Body' (which is currently selected), 'Pre-request Script', 'Tests', and 'Settings'. To the right of these tabs are 'Cookies' and 'Beautify' buttons. The 'Body' tab contains a JSON payload with the following content:

```
1  {  
2    "duration": 60,  
3    "title": "ABBAAtt",  
4    "cast": "rittiktt",  
5    "director": "A234gtt",  
6    "description": "Sensational Movie"  
7  }
```

Below the body editor, there are tabs for 'Body', 'Cookies', 'Headers (13)', and 'Test Results'. The 'Body' tab is selected. On the right side of the interface, there is a status bar showing '201 Created' with a timestamp of '169 ms' and a size of '573 B', along with a 'Save Response' button. At the bottom, there are buttons for 'Pretty' (selected), 'Raw', 'Preview', 'Visualize', and 'JSON' (with a dropdown menu). There is also a search icon.

The response body, visible in the 'Pretty' tab, is:

```
1  {  
2    "id": 18,  
3    "title": "ABBAAtt",  
4    "director": "A234gtt",  
5    "cast": "rittiktt",  
6    "description": "Sensational Movie",  
7    "duration": 60  
8  }
```

Edit a movie(Admin)

API - Edit a movie - POST - https://localhost:8443/api/movie/<movie_id> {Auth_token}

```
{
  "duration" : number, "title": string, "cast": string "director" : string "description" : string
}
```

Example :-

```
curl -k -location --request PUT 'https://localhost:8443/api/movie/18' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9eyJdWliOjZHZV0dGEiLCJyb2xlcyI6WyJST0xFX0FETUIOliwUk9MRV9VU0VSII0slmzcyl6lmh0dHBzOi8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2lulwiZXhwlijoxNjY4Mzg5NzUzfQ.SGfce9rTzhFU3Z1o3ZT2t4mTbuH6PBrhOFFEB5sQj50' \
--header 'Content-Type: application/json' \
--data-raw '{
  "duration" : 60,
  "title": "ABBA",
  "cast": "rittik Gus",
  "director" : "Nolan",
  "description" : "Sensational Movie"
}'
```

The screenshot shows the Postman interface with a PUT request to <https://localhost:8443/api/user/sdutta>. The Body tab is selected, showing a JSON payload:

```

1 {
2   "name": "Soumosir Admin",
3   "username": "sdutta",
4   "password": "Mary@1234567",
5   "email": "oumosir@gmail.com"
6 }
```

The response at the bottom indicates a 201 Created status with 358 ms response time and 581 B size.

Body

```

1 {
2   "id": 3,
3   "name": "Soumosir Admin",
4   "username": "sdutta",
5   "email": "oumosir@gmail.com",
6   "roles": [
7     "ROLE_ADMIN",
8     "ROLE_USER"
9   ]
10 }
```

Delete a movie(Admin)

API - Delete a movie - DELETE - https://localhost:8443/api/movie/<movie_id> {Auth_token}

```
curl --location --request DELETE 'https://localhost:8443/api/movie/18' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWlOIjZHV0dGEiLCJyb2xlcyI6WyJST0xFX0FETUIOliwiUk9MRV9
VU0VSII0sImIzcyl6lmh0dHBzOi8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2lulwiZXhwIjoxNjY4Mzg5NzUzfQ.SGfce9
TzhFU3Z1o3ZT2t4mTbuH6PBrhOFFEB5sQj50' \
--data-raw "
```

The screenshot shows the Postman application interface. At the top, there is a header bar with a dropdown menu set to "DELETE", a URL input field containing "https://localhost:8443/api/movie/18", a "Send" button, and a dropdown arrow. Below the header, there are tabs for "Params", "Authorization" (which is selected and has a green dot), "Headers (7)", "Body", "Pre-request Script", "Tests", and "Settings". The "Body" tab is currently active, indicated by an orange underline. Under the "Body" tab, there are several options: "none", "form-data", "x-www-form-urlencoded", "raw" (which is selected and highlighted in orange), "binary", "GraphQL", and "JSON" (with a dropdown arrow). To the right of these options is a "Beautify" button. The main body area is a large text input field containing the curl command shown in the previous code block. At the bottom of the interface, there are tabs for "Body" (underlined in orange), "Cookies", "Headers (11)", and "Test Results". To the right of these tabs are status indicators: a globe icon, "200 OK", "187 ms", "370 B", and a "Save Response" button. Below these status indicators are buttons for "Pretty", "Raw", "Preview", "Visualize", and "Text" (with a dropdown arrow). To the far right are icons for a clipboard and a magnifying glass.

Get all halls(admin and user allowe)

API - Get all halls - GET - <https://localhost:8443/api/hall/> {Auth_token}

Example :-

```
curl -k --location --request GET 'https://localhost:8443/api/hall/' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWliOiJzZHV0dGEiLCJyb2xlcyI6WyJST0xFETUIOliwiUk9MRV9
VU0VSII0slmIzcyl6lmh0dHBzOi8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2lulwiZXhwlijoxNjY4Mzg5NzUzfQ.SGfce9r
TzhFU3Z1o3ZT2t4mTbuH6PBrhOFfEB5sQj50' \
--header 'Content-Type: application/json' \
--data-raw "
```

The screenshot shows the Postman interface with the following details:

- Request Method:** GET
- URL:** <https://localhost:8443/api/hall/>
- Authorization:** Type: Bearer... (selected)
- Headers:** (9) (not explicitly listed in the screenshot but present in the UI)
- Body:** (empty)
- Response Status:** 200 OK (235 ms, 540 B)
- ResponseBody (Pretty):**

```
1 {
2   "id": 7,
3   "name": "hyattsville pg plaza",
4   "totalRows": 5,
5   "totalColumns": 5
6 },
7 {
8   "id": 8,
9   "name": "CP metro",
10  "totalRows": 5,
11  "totalColumns": 5
12 }
13 }
```

Add a new hall (admin only)

API - Add new hall - POST - <https://localhost:8443/api/hall/> {Auth_token}

```
{ "name": "string", "totalRows": number, "totalColumns": number }
```

```
curl -k --location --request POST 'https://localhost:8443/api/hall/' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWlOjzZHV0dGEiLCJyb2xlcyI6WyJST0xFETUIOliwiUk9MRV9
VU0VSII0slmIzcyl6lmh0dHBzOi8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2lulwiZXhwIjoxNjY4Mzg5NzUzfQ.SGfce9r
TzhFU3Z1o3ZT2t4mTbuH6PBrhOFFEB5sQj50' \
--header 'Content-Type: application/json' \
--data-raw '{

    "name": "DC Hall",
    "totalRows": 3,
    "totalColumns": 6
}'
```

The screenshot shows the Postman application interface. At the top, it displays a POST request to <https://localhost:8443/api/hall/>. The 'Body' tab is selected, showing the JSON payload:

```
1 {
2     ...
3     "name": "DC Hall",
4     "totalRows": 3,
5     "totalColumns": 6
6 }
```

Below the request, the response details are shown: a 201 Created status with a response time of 290 ms and a size of 516 B. The response body is displayed in Pretty, Raw, Preview, and Visualize formats, and is also JSON-formatted:1 {
2 "id": 19,
3 "name": "DC Hall",
4 "totalRows": 3,
5 "totalColumns": 6
6 }

Edit a hall by id (admin only)

API - edit hall - PUT - https://localhost:8443/api/hall/<hall_id> {Auth_token}

```
{ "name": "string", "totalRows": number, "totalColumns": number }
```

Example

```
curl -k --location --request PUT 'https://localhost:8443/api/hall/19' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWlOIjZHZV0dGEiLCJyb2xlcyI6WyJST0xFETUIOliwiUk9MRV9VU0VSII0sImlz
cyl6lmh0dHBzOi8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2luliviZXhwlijoxNjY4Mzg5NzUzfQ.SGfce9rTzhFU3Z1o3ZT2t4mTbuH6PB
rhOFFEB5sQj50' \
--header 'Content-Type: application/json' \
--data-raw '{
  "name": "CC Hall",
  "totalRows": 3,
  "totalColumns": 4
}'
```

The screenshot shows the Postman application interface for making a PUT request to the URL `https://localhost:8443/api/hall/19`. The request method is set to PUT. The Body tab is selected, showing the JSON payload:

```
1 {
2   ...
3   ...
4   ...
5   ...
6 }
```

The response section shows the following details:

- Method: PUT
- URL: `https://localhost:8443/api/hall/19`
- Status: 201 Created
- Time: 190 ms
- Size: 519 B
- Save Response button

The Body section displays the response JSON:

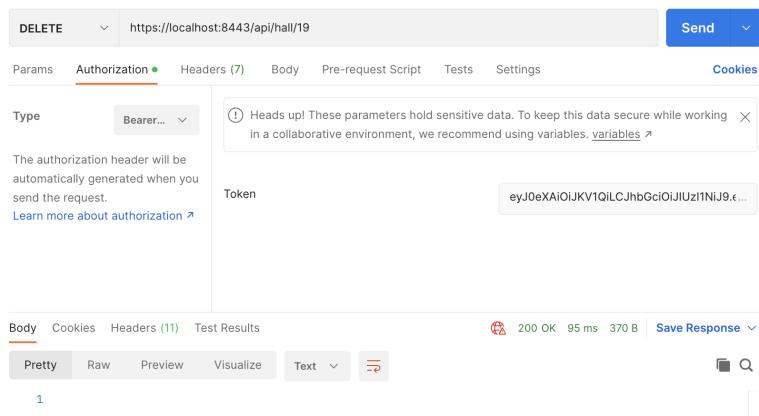
```
1
2   "id": 19,
3   "name": "CC Hall",
4   "totalRows": 3,
5   "totalColumns": 4
6 }
```

Delete a hall(admin only)

API - delete hall - DELETE - https://localhost:8443/api/hall/<hall_id> {Auth_token}

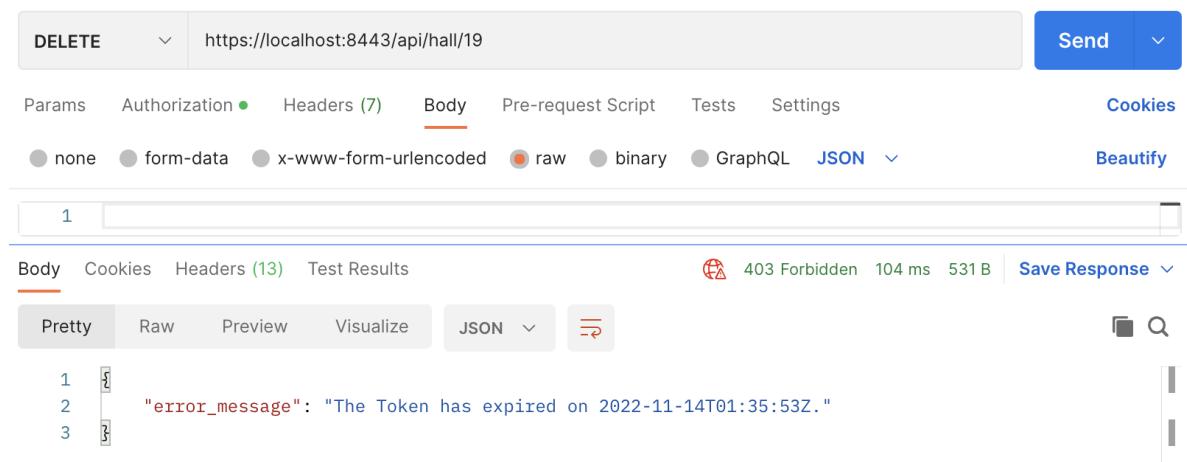
Example :-

```
curl -k --location --request DELETE 'https://localhost:8443/api/hall/19' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWliOiJzZHV0dGEiLCJyb2xlcyI6WyJST0xFETUIOliwiUk9MRV9
VU0VSII0slmzcyI6lmh0dHBzOi8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2lulwiZXhwIjoxNjY4Mzg5NzUzfQ.SGfce9r
TzhFU3Z1o3ZT2t4mTbuH6PBrhOFFeB5sQj50' \
--data-binary "
```



The screenshot shows the Postman interface with a successful DELETE request. The URL is https://localhost:8443/api/hall/19. The response status is 200 OK, 95 ms, 370 B.

Token expires after 5mins



The screenshot shows the Postman interface with a failed DELETE request due to token expiration. The URL is https://localhost:8443/api/hall/19. The response status is 403 Forbidden, 104 ms, 531 B.

```
1
2 "error_message": "The Token has expired on 2022-11-14T01:35:53Z."
3
```

Screenings GET (Admin and user both)

[Screening is like event or the particular movie running on a particular hall at a particular time]

API - get all screenings - GET - <https://localhost:8443/api/screening/> {Auth_token}

Example:

```
curl -k --location --request GET 'https://localhost:8443/api/screening' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWlOjzZHV0dGEiLCJyb2xlcyI6WyJST0xFX0FETUIOliwiUk9MRV9
VU0VSII0sImLzcyl6lmh0dHBzOj8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2lulwiZXhwIjoxNjY4MzkyMjgwQ.M_ttW_el
_tbCN5rEsEq0QSfaxcaQeM6sib0dxuaeRk' \
--header 'Content-Type: application/json' \
--data-raw "
```

The screenshot shows the Postman interface with the following details:

- Method:** GET
- URL:** https://localhost:8443/api/screening
- Authorization:** Bearer [REDACTED]
- Body:** (Empty)
- Headers:** (12) - includes Content-Type: application/json
- Response Status:** 200 OK, 211 ms, 758 B
- Response Body (Pretty JSON):**

```
1 [
2   {
3     "id": 11,
4     "startTime": "2022-12-20T15:20:00.000+00:00",
5     "movieId": 9,
6     "movieTitle": "Thor",
7     "hallId": 7,
8     "hallName": "hyattsville pg plaza",
9     "endTime": "2022-12-20T17:20:00.000+00:00"
10    },
11    {
12      "id": 12,
13      "startTime": "2022-12-25T15:30:00.000+00:00",
14      "movieId": 9,
15      "movieTitle": "Thor",
16      "hallId": 7,
17      "hallName": "hyattsville pg plaza",
18      "endTime": "2022-12-25T17:30:00.000+00:00"
19    }
20 ]
```

Screening Add a new screening (Admin)

API - Add a new screening - POST - <https://localhost:8443/api/screening/> {Auth_token}

```
{ "startTime": "string date", "movie":{ "id": num }, "hall" :{ "id": num} }
```

Example :-

```
curl -k --location --request POST 'https://localhost:8443/api/screening' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWlOIjZHV0dGEiLCJyb2xlcyI6WyJST0xFETUIOliwiUk9MRV9
VU0VSII0sImIzcyl6lmh0dHBzOi8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2lulwiZXhwIjoxNjY4MzkyMjgwQ.M_ttW_el
_tbCN5rEsEq0QSfaxcaQeM6sib0dxuaeRk' \
--header 'Content-Type: application/json' \
--data-raw '{
  "startTime": "2022-12-20T19:21:00.000+00:00",
  "movie":{ "id": 10 },
  "hall" :{ "id": 8}
}'
```

The screenshot shows the Postman interface with the following details:

- Method:** POST
- URL:** https://localhost:8443/api/screening
- Headers:** Authorization (set to Bearer with a long token), Content-Type (application/json)
- Body:** Raw JSON payload:

```
{
  "startTime": "2022-12-20T19:21:00.000+00:00",
  "movie":{ "id": 10 },
  "hall" :{ "id": 8}
}
```
- Response:** Status 201 Created, 130 ms, 623 B. The response body is:

```
{
  "id": 20,
  "startTime": "2022-12-20T19:21:00.000+00:00",
  "movieId": 10,
  "movieTitle": "Avenger",
  "hallId": 8,
  "hallName": "CP metro",
  "endTime": "2022-12-20T21:21:00.000+00:00"
}
```

Screening Edit a existing Screening(Admin)

API - edit screening - PUT - <https://localhost:8443/api/screening/<id>> {Auth_token}

```
{ "startTime": "string date", "movie":{ "id": num }, "hall" : {"id": num} }
```

Example -

```
curl --location --request PUT 'https://localhost:8443/api/screening/20' \
--header 'Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9eyJzdWliOjZHZV0dGEiLCJyb2xlcyl6WyJST0xFX0FETUIOliwiUk9MRV9VU0VSII0sImlzcyI6Imh0dHBzOi8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2lulawiZXhwlijoxNjY4MzkyMjgwfQ.M_ttW_el_tbCN5rEsEq0QSfaxcaQeM6sib0dxuaeQRk' \
--header 'Content-Type: application/json' \
--data-raw '{
  "startTime": "2022-12-20T19:22:00.000+00:00",
  "movie":{ "id": 10 },
  "hall" : {"id": 8}
}'
```

Changed start time

The screenshot shows the Postman interface with a PUT request to <https://localhost:8443/api/screening/20>. The Body tab is selected, displaying the following JSON payload:

```
1
2
3   ...
4   ...
5   ...
6 }
```

The response tab shows a 201 Created status with the following JSON body:

```
1
2
3   ...
4   ...
5   ...
6 }
```

Changed hall , movie and start time -

The screenshot shows a Postman request to `https://localhost:8443/api/screening/20`. The Body tab contains the following JSON:

```
1 {
2     ...
3         "startTime": "2022-12-20T19:22:00.000+00:00",
4         "movie": { "id": 9 },
5         "hall": { "id": 7 }
6 }
```

The response status is 201 Created, with a response body:

```
1 {
2     "id": 20,
3     "startTime": "2022-12-20T19:22:00.000+00:00",
4     "movieId": 9,
5     "movieTitle": "Thor",
6     "hallId": 7,
7     "hallName": "hyattsville pg plaza",
8     "endTime": "2022-12-20T21:22:00.000+00:00"
9 }
```

Screening Delete a existing Screening(Admin)

API - DeleteScreening - DELETE - <https://localhost:8443/api/screening/<id>> {Auth_token}

```
curl -k --location --request DELETE 'https://localhost:8443/api/screening/20' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWlOjzZHV0dGEiLCJyb2xlcyI6WyJST0xFX0FETUIOliwiUk9MRV9
VU0VSII0sImIzcyl6lmh0dHBzOi8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2luliwiZXhwIjoxNjY4MzkyMjgwQ.M_ttW_el
_tbCN5rEsEq0QSfaxcaQeM6sib0dxuaeRk' \
--data-raw "
```

The screenshot shows a Postman request to `https://localhost:8443/api/screening/20`. The Body tab contains the number 1.

The response status is 200 OK, with a response body:

```
1
```

Reservations - User and Admin (Note we are not adding payment functionality as of now)

Get reservation done by that particular user. Admin can see all reservations

API - Get Reservation - DELETE - [https://localhost:8443/api/reservation/ {Auth_token}](https://localhost:8443/api/reservation/)

Example :-

```
curl -k --location --request GET 'https://localhost:8443/api/reservation' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWliOiJqZG9lIiwicm9sZXMiOlsIUK9MRV9VU0VSII0sImlzcyI6Imh0dHBzOi8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2luliwiZXhwIjoxNjY4MzkzMDC5fQ.MJuy2oOLLA0jGvmzbBGhMeiSE5Z5N_iNcDQ4KB7ar2g' \
--data-binary "
```

The screenshot shows the Postman application interface. At the top, there is a header bar with 'GET' selected, a URL field containing 'https://localhost:8443/api/reservation', and a 'Send' button. Below the header, there are tabs for 'Params', 'Authorization' (which is currently active and highlighted in green), 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', 'Settings', and 'Cookies'. The 'Authorization' tab shows a dropdown menu set to 'Bearer Token'. A tooltip message reads: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables.' Below the tabs, there are buttons for 'Body', 'Cookies', 'Headers (12)', and 'Test Results'. On the right side, there is a status indicator showing '200 OK' with a response time of '84 ms' and a size of '772 B', along with a 'Save Response' button. The main content area displays a JSON response with line numbers from 1 to 26. The JSON data consists of three reservation objects:

```
1 {
2   "id": 14,
3   "timestamp": "2022-11-14T00:55:08.292+00:00",
4   "rowNumber": 1,
5   "columnNumber": 3,
6   "username": "jdoe",
7   "screeningId": 11
8 },
9 {
10  "id": 15,
11  "timestamp": "2022-11-14T00:55:08.316+00:00",
12  "rowNumber": 1,
13  "columnNumber": 4,
14  "username": "jdoe",
15  "screeningId": 11
16 },
17 {
18  "id": 16,
19  "timestamp": "2022-11-14T00:55:08.340+00:00",
20  "rowNumber": 1,
21  "columnNumber": 5,
22  "username": "jdoe",
23  "screeningId": 11
24 }
25
26 }
```

Add Reservations (user - { can only Add his reservation})

API - Add Reservation - POST - <https://localhost:8443/api/reservation/> {Auth_token}

```
{ "screening" : { "id" : num }, "rowId": num, "columnId": num }
```

```
curl -k --location --request POST 'https://localhost:8443/api/reservation/' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWliOiJqZG9Iliwicm9sZXMiOlsiUk9MRV9VU0VSII0sImlzcyI6Imh0dH
BzOi8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2luliwZXhwljoxNjY4MzkzMDC5fQ.MJuy2oOLLA0jGvmzbBGhMeiSE5
Z5N_iNcDQ4KB7ar2g' \
--header 'Content-Type: application/json' \
--data-raw '{
  "screening" : {
    "id" : 11
  },
  "rowId": 2,
  "columnId": 1
}'
```

The screenshot shows the Postman application interface. At the top, there is a header bar with 'POST' selected as the method, the URL 'https://localhost:8443/api/reservation/' entered, and a 'Send' button. Below the header are tabs for 'Params', 'Authorization', 'Headers (9)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Body' tab is active, showing the JSON payload:

```
1
2   ...
3     "screening" : {
4       "id" : 11
5     },
6     "rowId": 2,
7     "columnId": 1
8 }
```

Below the body editor, there are tabs for 'Body', 'Cookies', 'Headers (13)', and 'Test Results'. The 'Body' tab is active, showing the raw JSON response:

```
1
2   {
3     "id": 21,
4     "timestamp": "2022-11-14T01:56:08.039+00:00",
5     "rowNumber": 2,
6     "columnNumber": 1,
7     "username": "jdoe",
8     "screeningId": 11
9 }
```

At the bottom right, there are buttons for 'Save Response' and a search icon.

Update Reservations (user - { can only Update his reservation})

API - Edit Reservation - PUT - <https://localhost:8443/api/reservation/<id>> {Auth_token}

```
{ "screening": { "id": num }, "rowId": num, "columnId": num }
```

```
curl -k --location --request PUT 'https://localhost:8443/api/reservation/21' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWlOIjQZG9lIiwicm9sZXMiOlsiUk9MRV9VU0VSII0sImlzcyI6Imh0dHBzOj8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2lulwiZXhwIjoxNjY4MzkzMDC5fQ.MJuy2oOLLA0jGvmzbBGhMeiSE5Z5N_iNcDQ4KB7ar2g'
--header 'Content-Type: application/json' \
--data-raw '{
  "screening": {
    "id": 11
  },
  "rowId": 4,
  "columnId": 4
}'
```

The screenshot shows the Postman application interface. At the top, there is a header bar with 'PUT' selected, the URL 'https://localhost:8443/api/reservation/21', and a 'Send' button. Below the header are tabs for 'Params', 'Authorization' (which is set to 'Bearer'), 'Headers (9)', 'Body' (which is currently selected), 'Pre-request Script', 'Tests', 'Settings', and 'Cookies'. Under the 'Body' tab, there are options for 'none', 'form-data', 'x-www-form-urlencoded', 'raw' (which is selected), 'binary', and 'GraphQL'. To the right of these options is a 'JSON' dropdown and a 'Beautify' button. The main body area contains the JSON payload:

```
1 {
2   ...
3   "screening": {
4     ...
5     "id": 11
6   },
7   ...
8 }
```

At the bottom of the interface, there are tabs for 'Body', 'Cookies', 'Headers (13)', and 'Test Results'. The 'Test Results' tab is active, showing a status message: '201 Created 288 ms 588 B | Save Response'. Below this, there are buttons for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'JSON' (with a dropdown menu). On the far right, there are icons for a file and a search function.

The 'Body' section also displays the JSON response received from the server:

```
1 {
2   "id": 21,
3   "timestamp": "2022-11-14T01:56:08.039+00:00",
4   "rowNumber": 4,
5   "columnNumber": 4,
6   "username": "jdoe",
7   "screeningId": 11
8 }
```

Delete Reservations (user - { can only Delete own reservation})

API - Delete Reservation - DELETE - <https://localhost:8443/api/reservation/<id>> {Auth_token}

```
curl --location --request DELETE 'https://localhost:8443/api/reservation/21' \
--header 'Authorization: Bearer
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWlOjZG9lIiwicm9sZXMiOlsiUk9MRV9VU0VSIl0sImlzcyI6Imh0dH
BzOi8vbG9jYWxob3N0Ojg0NDMvYXBpL2xvZ2luliwiZXhwIjoxNjY4MzkzMDC5fQ.MJuy2oOLLA0jGvmzbBGhMeiSE5
Z5N_iNcDQ4KB7ar2g' \
--data-binary "
```

The screenshot shows the Postman application interface. At the top, there is a header bar with 'DELETE' selected, a URL field containing 'https://localhost:8443/api/reservation/21', and a 'Send' button. Below the header are tabs for 'Params', 'Authorization' (selected), 'Headers (7)', 'Body' (selected), 'Pre-request Script', 'Tests', and 'Settings'. Under 'Body', there are options for 'none', 'form-data', 'x-www-form-urlencoded', 'raw' (selected), 'binary', and 'GraphQL', with 'JSON' dropdown and 'Beautify' link. The main body area has a single entry labeled '1'. At the bottom, there are tabs for 'Body' (selected), 'Cookies', 'Headers (11)', and 'Test Results'. The status bar at the bottom right shows '200 OK 170 ms 370 B' and a 'Save Response' button. Below the status bar are buttons for 'Pretty', 'Raw', 'Preview', 'Visualize', and 'Text' (selected). There is also a search icon.

Security features :-

1. Rate limiting

Cannot access api more than 60 times in 1 mins from an IP (Using a cache for this ip:attempts)

Too many request : message **HTTP code 429**

The screenshot shows the Postman interface with a GET request to `https://localhost:8443/api/screening/`. The 'Body' tab is selected, showing a single digit '1'. The response status is 429 Too Many Requests, with a timestamp of 7 ms and a body size of 403 B. The response content is 'Too many requests'.

2. Brute force login attempts limited to 10 from an IP (Using cache ip : attempts)

The screenshot shows the Postman interface with a POST request to `https://localhost:8443/api/login`. The 'Body' tab is selected, showing a table with two rows: 'username' with value 'jdoe' and 'password' with value '1212'. The response status is 403 Forbidden, with a timestamp of 43 ms and a body size of 377 B. The response content is 'Description'.

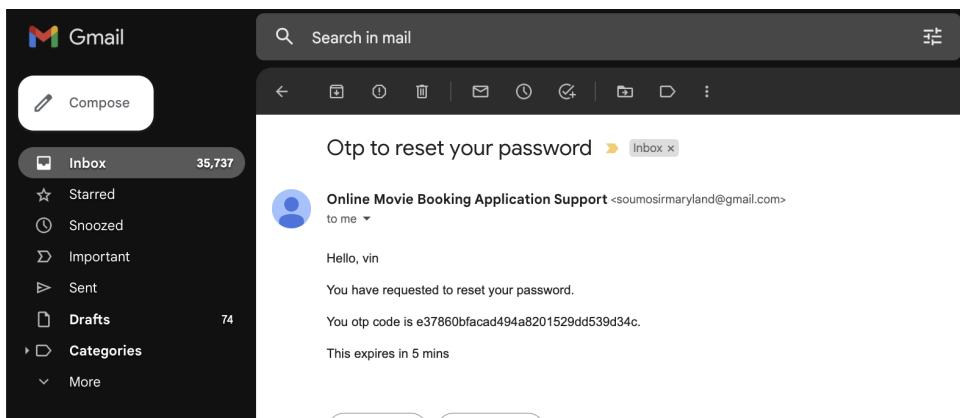
```
2022-11-13 21:14:57.744 ERROR 51987 --- [nio-8443-exec-1] c.s.b.s.CustomAuthenticationFilter      : User on ip 0:0:0:0:0:0:1 entered wrong username and password
2022-11-13 21:14:58.369 INFO 51987 --- [nio-8443-exec-2] c.s.b.s.CustomAuthenticationFilter      : Attempting login for Username : jdoe ,
2022-11-13 21:14:58.377 INFO 51987 --- [nio-8443-exec-2] c.s.b.service.UserServiceImplementation : Ip 0:0:0:0:0:0:1 is blocked
2022-11-13 21:14:58.390 ERROR 51987 --- [nio-8443-exec-2] c.s.b.s.CustomAuthenticationFilter      : An internal error occurred while trying to authenticate the user.
```

3. Forgot password uses secret code sent to email address to change password (using cache for this email : code)

RESET a user password (REQUEST)

```
curl --location --request GET 'https://localhost:8443/api/resetpassword/soumosirdutta@gmail.com'
```

The screenshot shows a Postman request configuration. The method is set to GET, the URL is <https://localhost:8443/api/resetpassword/soumosirdutta@gmail.com>, and the 'Authorization' tab is selected. The response status is 200 OK, and the body of the response is: "1 Email sent to soumosirdutta@gmail.com".



bookmy / user reset password post

POST

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL **JSON** Beautify

```

1
2   "code" : "e37860bfac494a8201529dd539d34c",
3   "email" : "soumosirdutta@gmail.com",
4   "password" : "Maryla@13434t3t5"
5

```

User cannot view other user :-

Cannot see/edit/delete user jdoe using bbacca user

GET

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Type Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [variables](#)

Body Cookies Headers (12) Test Results Save Response

Pretty Raw Preview Visualize **JSON**

```

1
2   "id": 5,
3   "name": "Burry Bacca",
4   "username": "bbacca",
5   "email": "bscscs@ycns.com",
6   "roles": [
7     "ROLE_USER"
8   ]
9

```

GET

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Type Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [variables](#)

Body Cookies Headers (11) Test Results Save Response

Pretty Raw Preview Visualize **Text**

1

User cannot see/edit/add reservations of other user

Jdoe 's reservation

The screenshot shows a Postman interface with a GET request to `https://localhost:8443/api/reservation`. The Authorization tab is selected, showing a Bearer Token. The response body is displayed in Pretty JSON format, showing three reservation objects:

```
1 [
2   {
3     "id": 14,
4     "timestamp": "2022-11-14T00:55:08.292+00:00",
5     "rowNumber": 1,
6     "columnNumber": 3,
7     "username": "jdoe",
8     "screeningId": 11
9   },
10  {
11    "id": 15,
12    "timestamp": "2022-11-14T00:55:08.316+00:00",
13    "rowNumber": 1,
14    "columnNumber": 4,
15    "username": "jdoe",
16    "screeningId": 11
17  },
18  {
19    "id": 16,
20    "timestamp": "2022-11-14T00:55:08.340+00:00",
21    "rowNumber": 1,
22    "columnNumber": 5,
23    "username": "jdoe",
24    "screeningId": 11
25  }
26 ]
```

Bacca's reservation - same api

The screenshot shows a Postman interface with a GET request to `https://localhost:8443/api/reservation`. The Authorization tab is selected, showing a Bearer Token. The response body is displayed in Pretty JSON format, showing an empty array:

```
1 []
```

Trying to access a Admin route POST api/screening/ using USER_ROLE user

The screenshot shows the Postman interface with the following details:

- Method:** POST
- URL:** https://localhost:8443/api/screening/
- Authorization:** Bearer... (selected)
- Headers:** (8)
- Body:** (empty)
- Pre-request Script:** (empty)
- Tests:** (empty)
- Settings:** (empty)
- Cookies:** (empty)

A tooltip message in the Authorization section reads: "Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables." A token value is shown in the Token field: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9....

At the bottom, the response status is 403 Forbidden with 106 ms response time and 377 B size. Buttons for "Save Response" and "Pretty" are visible.

We get forbidden

Secure Communication by SSL certificate

Api listens on https:

References -

<https://www.baeldung.com/spring-boot-https-self-signed-certificate>

Port given is 8443

Validation

We validation for user email, password, name,

Email - (>3 characters and <30 characters)

Password - (>8 characters and <30 characters)

Atleast 1 upper case character

Atleast 1 lower case character

Atleast 1 special characters

Atleast 1 digit

Name - (>3 characters and <30 characters)

Username - (>3 characters and <30 characters) Alphanumeric character - A-Z, a-z ,0-9

We use model validation for hall name , seat row and seat column

Hall name - >3 characters and <30 characters

Seat row number (1-200)

Seat column number (1-200)

We use string validation for movie title, movie , director and cast (>3 characters and <30 characters)

Title >3 characters and <30 characters

director >3 characters and <30 characters

Cast >3 characters and <30 characters

Description less than 200 characters

Screening

Starttime - must be a valid time of format YYYY-MM-DDTHH:MM:

2022-12-20T15:20:00.000+00:00

No two movies can screen at the same interval in same hall

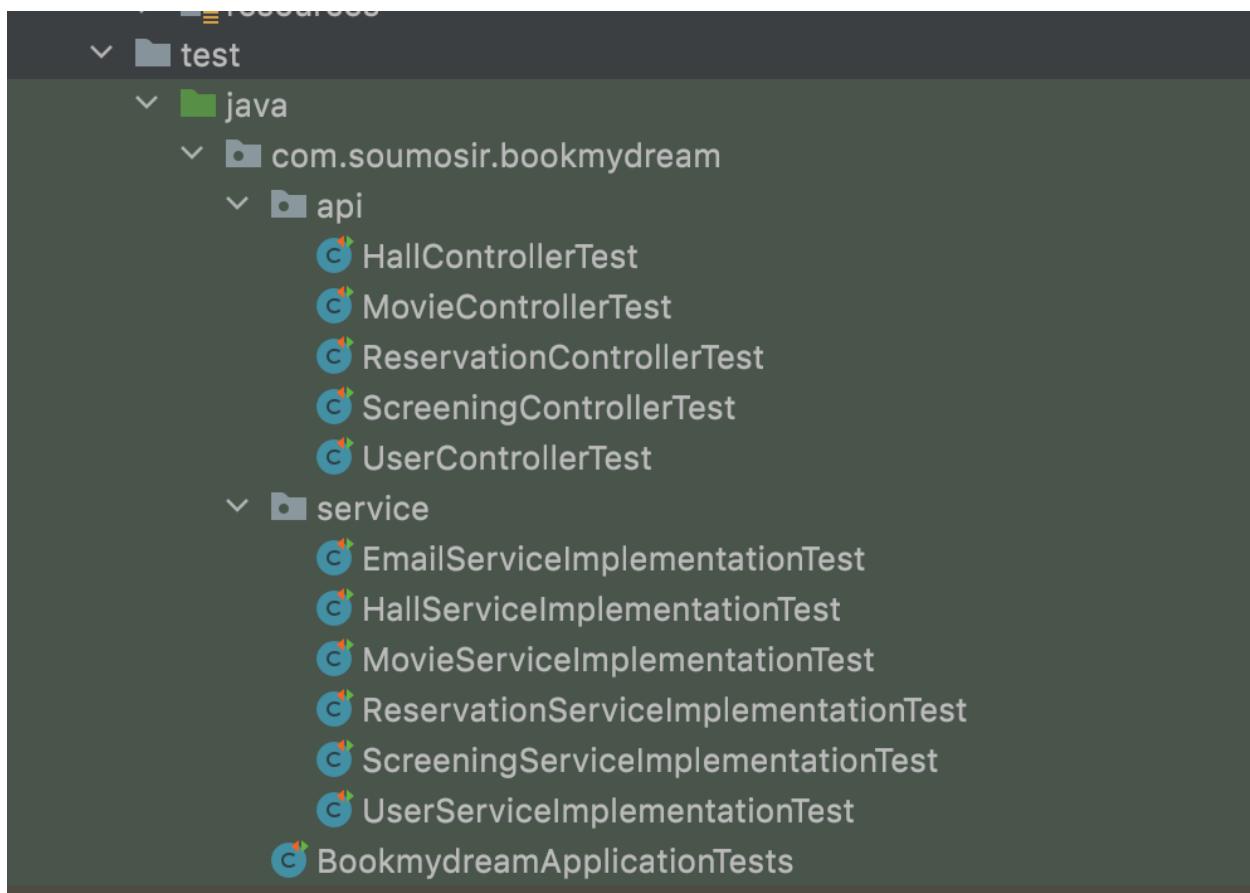
Reservation

No two person can have the same seat for a screening(unique hall and movie for a time)

Running tests:

**There are 6 services test classes
5 rest api controller test classes**

In `src/test/java/com/soumosir/bookmydream`



Go to each one and run the file by green button

You need to run all the test individually

Example : -

Test result seen

Rest controllers -

1. Hall controller 8 tests

The screenshot shows the IntelliJ IDEA interface with the HallControllerTest.java file open. The code defines a test class for the HallController. Below the code editor, the 'Run' tool window displays the test results for HallControllerTest, showing 8 tests passed in 1 second.

```
package com.soumosir.bookmydream.api;
import ...
@ContextConfiguration(classes = {HallController.class})
@ExtendWith(SpringExtension.class)
class HallControllerTest {
    @Autowired
    private HallController hallController;
    @MockBean
    private HallService hallService;
    /**
     * Method under test: {@link HallController#getHalls()}
     */
    @Test
    void testGetHalls() throws Exception {
```

Test Results:

- testDeleteHall2() 902 ms
- testDeleteHall() 69 ms
- testSaveHall() 333 ms
- testUpdateHall() 75 ms
- testDeleteHall() 43 ms
- testGetHalls2() 43 ms
- testGetHalls3() 44 ms
- testGetHalls4() 48 ms

Output log (partial):

```
22:03:29.059 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating CacheAwareContextLoaderDelegate from class [org.springframework.test.context.CacheAwareContextLoaderDelegate]
22:03:29.123 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating BootstrapContext using constructor [public org.springframework.test.context.BootstrapContext()]
22:03:29.252 [main] DEBUG org.springframework.test.context.support.ActiveProfilesUtils - Could not find an 'annotation declaring class' for annotation type [org.springframework.test.context.ActiveProfiles]
22:03:29.252 [main] DEBUG org.springframework.test.context.support.AbstractTestContextBootstrapper - Instantiating TestContextBootstrapper for test class [com.soumosir.bookmydream.api.HallControllerTest]
22:03:29.252 [main] DEBUG org.springframework.test.context.support.AbstractDelegatingSmartContextLoader - Delegating to AnnotationConfigContextLoader
22:03:29.252 [main] DEBUG org.springframework.test.context.support.AbstractGenericContextLoader - Loading ApplicationContext for merged context configuration
22:03:29.252 [main] INFO org.springframework.test.context.support.DefaultTestContextBootstrapper - Loaded default TestExecutionListener class names from location [META-INF/spring.factories]
22:03:29.252 [main] DEBUG org.springframework.test.context.support.DefaultTestContextBootstrapper - Using TestExecutionListeners: [org.springframework.test.context.support.DependencyInjectionTestExecutionListener, org.springframework.test.context.support.DirtiesContextTestExecutionListener, org.springframework.test.context.support.DistributedTestExecutionListener, org.springframework.test.context.support.EmailTestExecutionListener, org.springframework.test.context.support.ExpectedExceptionTestExecutionListener, org.springframework.test.context.support.GherkinDslTestExecutionListener, org.springframework.test.context.support.GroupTestExecutionListener, org.springframework.test.context.support.RootTestExecutionListener]
22:03:29.252 [main] DEBUG org.springframework.test.context.support.AbstractTestContextBootstrapper - Before test class: context [DefaultTestContext@11-0-2]
22:03:29.291 [main] DEBUG org.springframework.test.context.support.AbstractDelegatingSmartContextLoader - Delegating to AnnotationConfigContextLoader
22:03:29.292 [main] DEBUG org.springframework.test.context.support.AbstractGenericContextLoader - Loading ApplicationContext for merged context configuration
22:03:29.318 [main] DEBUG org.springframework.core.env.StandardEnvironment - Activating profiles []
22:03:29.318 [main] DEBUG org.springframework.test.context.support.AnnotationConfigContextLoader - Registering component classes: [class com.soumosir.bookmydream.api.HallController]
22:03:29.412 [main] DEBUG org.springframework.context.support.GenericApplicationContext - Refreshing org.springframework.context.support.GenericApplicationContext
22:03:29.412 [main] DEBUG org.springframework.beans.factory.support.DefaultListableBeanFactory - Creating shared instance of singleton bean 'hallController'
```

2. Movie controller 6 tests

The screenshot shows the IntelliJ IDEA interface with the MovieControllerTest.java file open. The code defines a test class for the MovieController. Below the code editor, the 'Run' tool window displays the test results for MovieControllerTest, showing 6 tests passed in 1 second.

```
package com.soumosir.bookmydream.api;
import ...
@ContextConfiguration(classes = {MovieController.class})
@ExtendWith(SpringExtension.class)
class MovieControllerTest {
    @Autowired
    private MovieController movieController;
    @MockBean
    private MovieService movieService;
    /**
     * Method under test: {@link MovieController#getMovies()}
     */
    @Test
    void testGetMovies() throws Exception {
```

Test Results:

- testDeleteMovie() 695 ms
- testUpdateMovie() 319 ms
- testDeleteMovie2() 52 ms
- testSaveMovie() 51 ms
- testGetMovies() 49 ms
- testGetMovies2() 40 ms

Output log (partial):

```
22:45:39.441 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating CacheAwareContextLoaderDelegate from class [org.springframework.test.context.CacheAwareContextLoaderDelegate]
22:45:39.485 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating BootstrapContext using constructor [public org.springframework.test.context.BootstrapContext()]
22:45:39.588 [main] DEBUG org.springframework.test.context.BootstrapUtils - Instantiating TestContextBootstrapper for test class [com.soumosir.bookmydream.api.MovieControllerTest]
22:45:39.538 [main] DEBUG org.springframework.test.context.support.AbstractDelegatingSmartContextLoader - Delegating to AnnotationConfigContextLoader
22:45:39.616 [main] DEBUG org.springframework.test.context.support.ActiveProfilesUtils - Could not find an 'annotation declaring class' for annotation type [org.springframework.test.context.ActiveProfiles]
22:45:39.617 [main] DEBUG org.springframework.test.context.support.DefaultTestContextBootstrapper - @TestExecutionListeners is not present for class [com.soumosir.bookmydream.api.MovieControllerTest]
22:45:39.617 [main] INFO org.springframework.test.context.support.DefaultTestContextBootstrapper - Loaded default TestExecutionListener class names from location [META-INF/spring.factories]
22:45:39.635 [main] INFO org.springframework.test.context.support.DefaultTestContextBootstrapper - Using TestExecutionListeners: [org.springframework.test.context.support.DependencyInjectionTestExecutionListener, org.springframework.test.context.support.DirtiesContextTestExecutionListener, org.springframework.test.context.support.DistributedTestExecutionListener, org.springframework.test.context.support.EmailTestExecutionListener, org.springframework.test.context.support.ExpectedExceptionTestExecutionListener, org.springframework.test.context.support.GherkinDslTestExecutionListener, org.springframework.test.context.support.GroupTestExecutionListener, org.springframework.test.context.support.RootTestExecutionListener]
22:45:39.637 [main] DEBUG org.springframework.test.context.support.AbstractDirtiesContextTestExecutionListener - Before test class: context [DefaultTestContext@11-0-2]
22:45:39.656 [main] DEBUG org.springframework.test.context.support.AbstractDelegatingSmartContextLoader - Delegating to AnnotationConfigContextLoader to load context configuration
22:45:39.656 [main] DEBUG org.springframework.test.context.support.AbstractGenericContextLoader - Loading ApplicationContext for merged context configuration
22:45:39.689 [main] DEBUG org.springframework.core.env.StandardEnvironment - Activating profiles []
```

3. reservation controller 8 tests

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "bookmydream" located at "Documents/personal...". It contains a "src" directory with "main" and "test" packages. The "test" package contains Java classes: HallControllerTest, MovieControllerTest, ReservationControllerTest, ScreeningControllerTest, UserControllerTest, and BookmydreamApplicationTests.
- Code Editor:** The "ReservationControllerTest.java" file is open. The code is a JUnit test for the "ReservationController". It imports various annotations and beans, and defines a test method "testDeleteReservation".
- Run Tab:** The "ReservationControllerTest" is selected in the dropdown. Below it, the "Test Results" section shows the execution details: 6 tests passed in 1 second and 449 milliseconds. Each test result includes the name, duration, and the command used to run the test.

4. Screening controller 8 test

The screenshot shows the IntelliJ IDEA interface with several windows open. The left sidebar displays the project structure for 'bookmydream' under 'Project'. The main editor window shows the source code for 'ScreeningControllerTest.java'. The bottom-left corner shows the 'Run' tool window with the output of a test run, indicating 8 tests passed in 1 second. The bottom-right corner shows the 'Test Results' tool window, which lists all test methods for 'ScreeningControllerTest' along with their execution times.

```
package com.soumosir.bookmydream.api;
import ...;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.boot.test.autoconfigure.web.servlet.MockMvcMvcBuilder;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.boot.test.context.SpringBootTestConfiguration;
import org.springframework.test.context.junit.jupiter.SpringExtension;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.ResultActions;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;

@SpringBootTestConfiguration(classes = {ScreeningController.class})
@ExtendWith(SpringExtension.class)
class ScreeningControllerTest {
    @Autowired
    private ScreeningController screeningController;

    @MockBean
    private ScreeningService screeningService;

    /**
     * Method under test: {@link ScreeningController#getScreenings()}
     */
    @Test
    void testGetScreenings() throws Exception {
        when(this.screeningService.getScreenings()).thenReturn(new ArrayList<>());
        MockMvcRequestBuilder requestBuilder = MockMvcRequestBuilders.get(urlTemplate: "/api/screenings");
        MockMvcBuilders.standaloneSetup(this.screeningController).StandaloneMockMvcBuilder
            .build() MockMvc
    }
}
```

Method	Time (ms)
ScreeningControllerTest#testGetScreenings()	1sec 695 ms
ScreeningControllerTest#testGetScreenings2()	1sec 695 ms
ScreeningControllerTest#testGetScreenings3()	1sec 84 ms
ScreeningControllerTest#testGetScreenings4()	53 ms
ScreeningControllerTest#testSaveScreening()	45 ms
ScreeningControllerTest#testDeleteScreening()	297 ms
ScreeningControllerTest#testUpdateScreening()	55 ms
ScreeningControllerTest#testGetScreenings()	79 ms
ScreeningControllerTest#testDeleteScreening2()	26 ms
ScreeningControllerTest#testGetScreenings2()	56 ms

5. User controller - 24 tests

The screenshot shows the IntelliJ IDEA interface with the code editor displaying the `UserControllerTest.java` file. The run results window below shows 24 tests passed in 2 seconds and 87 ms.

```

package com.soumosir.bookmydream.api;
import ...
@ContextConfiguration(classes = {UserController.class})
@ExtendWith(SpringExtension.class)
class UserControllerTest {
    @MockBean
    private EmailService emailService;

    @MockBean
    private ForgotPasswordService forgotPasswordService;

    @Autowired
    private UserController userController;

    @MockBean
    private UserService userService;

    /**
     * Method under test: {@link UserController#getUser()}
     */
}

```

Run: UserControllerTest

Tests passed: 24 of 24 tests – 2 sec 87 ms

- Test Results: 2 sec 87 ms
 - UserControllerTest
 - testResetUserPassword2() 2 sec 87 ms
 - testDeleteUser2() 910 ms
 - testDeleteUser3() 147 ms
 - testDeleteUser3() 57 ms
 - testResetUserPasswordCodeR 112 ms
 - testGetUser2() 85 ms
 - testGetUser3() 45 ms
 - testGetUsers() 33 ms
 - testSaveUser() 38 ms
 - testUpdateUser2() 118 ms
 - testUpdateUser3() 40 ms
 - testUpdateUser4() 49 ms
 - testResetUserPasswordCodeR 30 ms
 - testUpdateUser() 41 ms

Services

6. Email service 4 tests

The screenshot shows the IntelliJ IDEA interface with the code editor displaying the `EmailServiceImplementationTest.java` file. The run results window below shows 3 tests passed in 214 ms, with 1 ignored.

```

package com.soumosir.bookmydream.service;
import ...
@ContextConfiguration(classes = {EmailServiceImplementation.class})
@ExtendWith(SpringExtension.class)
class EmailServiceImplementationTest {
    @Autowired
    private EmailServiceImplementation emailServiceImplementation;

    @MockBean
    private JavaMailSender javaMailSender;

    /**
     * Method under test: {@link EmailServiceImplementation#sendEmail(String, String, String)}
     */
    @Test
    void testSendEmail() throws MailException {
        doNothing().when(this.javaMailSender).send((MimeMessage) any());
        when(this.javaMailSender.createMimeMessage()).thenReturn(new MimeMessage((Session) null));
        this.emailServiceImplementation.sendEmail("janedoe", "jane.doe@example.org", verify(this.javaMailSender).createMimeMessage());
    }
}

```

Run: EmailServiceImplementationTest

Tests passed: 3, ignored: 1 of 4 tests – 214 ms

- Test Results: 214 ms
 - EmailServiceImplementationTest 214 ms
 - testSendEmail2() 157 ms
 - testSendEmail3() 35 ms
 - testSendEmail4() 22 ms
 - testSendEmail() 22 ms

7. Hall service 18 tests

8. Movie Service tests - 12 tests

The screenshot shows an IntelliJ IDEA interface with the following details:

- Project Tree:** The project is named "bookmydream" and contains packages for "main" and "test". The "test" package contains Java classes for testing various controllers and services, along with a main application test.
- Code Editor:** The current file is "MovieServiceImplementationTest.java". It includes annotations like @ContextConfiguration and @ExtendWith(SpringExtension.class). The class "MovieServiceImplementationTest" is annotated with @MockBean and @Autowired. It has a private field "MovieRepo movieRepo" and a protected field "private MovieServiceImplementation movieServiceImplementation". A JUnit test method "void testSaveMovie()" is shown, which is disabled with the reason "TODO: Complete this test". The code editor also highlights some TODO comments and a diff message from "Diffblue Cover".
- Run Tab:** Shows the run configuration for "MovieServiceImplementationTest" with a status of "Tests passed: 12, ignored: 2 of 14 tests - 575 ms".
- Test Results:** A detailed view of the test execution results, including test names, execution times, and log output for each test case.

9. Reservation service 18 tests

The screenshot shows the IntelliJ IDEA interface with the code editor displaying `ReservationServiceImplementationTest.java` and the toolbars and side panels visible.

```

package com.soumosir.bookmydream.service;
import ...
@ContextConfiguration(classes = {ReservationServiceImplementation.class})
@ExtendWith(SpringExtension.class)
class ReservationServiceImplementationTest {
    @MockBean
    private AppUserRepo appUserRepo;

    @MockBean
    private HallRepo hallRepo;

    @MockBean
    private ReservationRepo reservationRepo;

    @Autowired
    private ReservationServiceImplementation reservationServiceImplementation;

    @MockBean
    private ScreeningRepo screeningRepo;
}

```

The `Test Results` window shows 18 tests passed in 954ms, with detailed logs for each test method.

10. Screening service 7 tests

The screenshot shows the IntelliJ IDEA interface with the code editor displaying `ScreeningServiceImplementationTest.java` and the toolbars and side panels visible.

```

package com.soumosir.bookmydream.service;
import ...
@ContextConfiguration(classes = {ScreeningServiceImplementation.class})
@ExtendWith(SpringExtension.class)
class ScreeningServiceImplementationTest {
    @MockBean
    private HallRepo hallRepo;

    @MockBean
    private MovieRepo movieRepo;

    @MockBean
    private ScreeningRepo screeningRepo;

    @Autowired
    private ScreeningServiceImplementation screeningServiceImplementation;

    /**
     * Method under test: {@link ScreeningServiceImplementation#getScreeningsByHall(Hall)}
     */
}

```

The `Test Results` window shows 7 tests passed in 361ms, with detailed logs for each test method.

11. User service tests - 33

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Structure:** The project is named "bookmydream" and contains packages for ".idea", ".mvn", "src", "main", and "test". The "test" package contains sub-packages "java" and "service", which in turn contain various test classes like "HallControllerTest", "MovieControllerTest", etc., and "UserServiceImplementationTest".
- Code Editor:** The code editor shows the source code for "UserServiceImplementationTest.java". It includes annotations such as `@ContextConfiguration(classes = {UserServiceImplementation.class})`, `@ExtendWith(SpringExtension.class)`, and `@MockBean` for dependencies like `AppUserRepo`, `ForgotPasswordService`, `HttpServletRequest`, `LoginAttemptService`, and `PasswordEncoder`.
- Run Tab:** The "Run" tab shows the result of running "UserServiceImplementationTest" with 33 tests passed and 7 ignored.
- Test Results:** The "Test Results" section provides a detailed log of the test execution. It lists 33 test methods with their execution times and corresponding log output from the Spring framework.

References

<https://www.baeldung.com/spring-security-block-brute-force-authentication-attempts>

<https://spring.io/projects/spring-security>

<https://spring.io/guides/tutorials/rest/>