

# Team 1 – Movie Ticket Booking System.

**GitHub Url :** [https://github.com/Piyusshh/Team\\_1\\_Movie\\_Booking](https://github.com/Piyusshh/Team_1_Movie_Booking)

Team Members: -

1. Piyush Agrawal
2. Ninad Dadmal
3. Ayesha Bee
4. Sejal Tajane
5. Prachi Kalantari

Entities =>

1. City
2. Theater
3. Movie
4. Show
5. Bookings

**Functional requirements: -**

Functionalities of Admin:

1. Admin can manage cities
  - a. Add city
  - b. View all cities. Can also view a city by id.
  - c. Update cities
  - d. Delete city
2. Admin can manage theatres
  - a. Add theatre
  - b. View all theatres. Can also view a theatre by id.
  - c. Update theatre
  - d. Delete theatre

### 3. Admin can manage movies

- a. Add movies
- b. View all movies. Can also view a movie by id.
- c. Update movies
- d. Delete movies

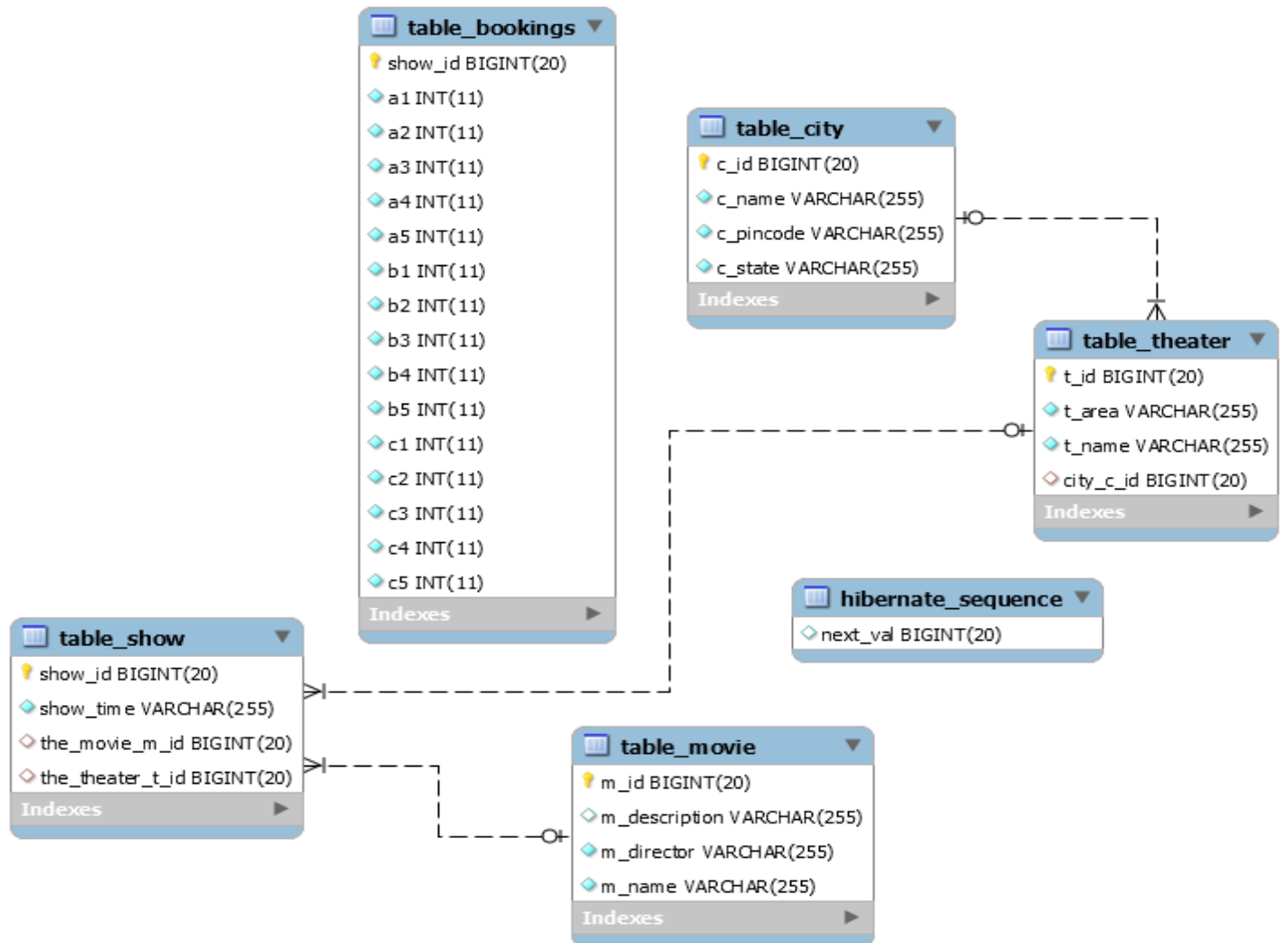
### 4. Admin can manage shows

- a. Add shows
- b. View all shows. Can also view a show by id.
- c. Update show
- d. Delete show

## Functionalities of Customer:

- Customer can view all the cities.
- Customer can view all theatres of a particular city using city id.
- Customer can view all movies of a theatre.
- Customer can view all the shows for a movie.
- Customer can view all the seats available for a particular show.
- Customer can also book a seat.

## Data Base Structure



## Movie Ticket Booking

This is the movie ticket booking REST API project. We used **Spring boot** for API implementation and **Hibernate** for Database connectivity.

**Function Requirements** => Below is the list of URL endpoints available for the User and Admin to make integrations for their platform: -

1. **Admin** => All the Admin or merchant URL endpoint starts with the **/api**.

a. To add the city to the DB

- i. URL: **/api/city**
- ii. Method: - POST

Body

```
1 {
2   "name": "Mumbai",
3   "pincode": "120021",
4   "state": "Maharashtra"
5 }
```

^ Response

Status: 200    Time: 248 ms

```
1 {
2   "pincode": "120021",
3   "name": "Mumbai",
4   "state": "Maharashtra",
5   "id": 10
6 }
```

b. To add Theater to the DB => here theater is reference to the key\_id of the city as primary key so makes sure to pass the valid city id

- i. URL: **/api/{CityID}/theater**
- ii. Method: POST

## Body

```
1 {  
2   "name": "Mumbai mall",  
3   "area": "Dharavi"  
4 }
```

## Response

Status: 200 Time: 59 ms

```
1 {  
2   "t_name": "Mumbai mall",  
3   "t_area": "Dharavi",  
4   "t_id": 11  
5 }
```

### c. To add the Movie to the DB

- i. URL: **/api/movie**
- ii. Method: POST

## Body

```
1 {  
2   "_name": "Dr. Strange",  
3   "_director": "Strange directors",  
4   "_description": "description about the Dr Starngel movie here"  
5 }
```

## Response

Status: 200 Time: 48 ms

PRETTY ☒ S

```
1 {  
2   "theShow": null,  
3   "_director": "Strange directors",  
4   "_description": "description about the Dr Starngel movie here",  
5   "_name": "Dr. Strange",  
6   "_id": 12  
7 }
```

- d. To add a show => TO add a show pass the theater Id and Movie id as a URL parameter and the show time in the request body.
- i. URL: **/api/{theater\_id}/{movie\_id}/show**
  - ii. Method: POST

```
Body
1 {
2   "time": "4/4/2020 - 3:30 PM"
3 }
```

Response

```
Status: 200    Time: 44 ms
1 {
2   "show_Id": 13,
3   "show_time": "4/4/2020 - 3:30 PM"
4 }
```

2. **User** => All the User or the Customer Endpoints starts with **/user**.
- a. After user selects the show pass the show id to the below api and in return you will get a list of seats available for the show.
    - i. URL: **/user/show/{showID}**
    - ii. Method: GET

```
Status: 200    Time: 23 ms

1  {
2    "show_id": 13,
3    "hibernateLazyInitializer": {},
4    "c2": 0,
5    "a3": 0,
6    "b4": 0,
7    "c4": 0,
8    "a5": 0,
9    "c5": 0,
10   "a2": 0,
11   "b1": 0,
12   "c1": 0,
13   "b2": 0,
14   "c3": 0,
15   "b5": 0,
16   "a1": 0,
17   "a4": 0,
18   "b3": 0
19 }
```

- b. After user selects the show pass the show id to the below URL and in return, you'll get a list of seats available for the show for the user to book.

- i. URL: `/user/show/{showID}/bookings`
- ii. Method: POST

Body

```
1 {  
2   "show_id": 13,  
3   "hibernateLazyInitializer": {},  
4   "c2": 1,  
5   "a3": 1,  
6   "b4": 0,  
7   "c4": 0,  
8   "a5": 0,  
9   "c5": 1,  
10  "a2": 1,  
11  "b1": 0,  
12  "c1": 1,  
13  "b2": 0,  
14  "c3": 0,
```

Response

Status: 200    Time: 31 ms

```
1 {  
2   "show_id": 13,  
3   "c2": 1,  
4   "a3": 1,  
5   "b4": 0,  
6   "c4": 0,  
7   "a5": 0,  
8   "c5": 1,  
9   "a2": 1,  
10  "b1": 0,  
11  "c1": 1,  
12  "b2": 0,  
13  "c3": 0,  
14  "b5": 0,  
15  "a1": 0,  
16  "a4": 1,  
17  "b3": 1,  
18 }
```

iii.



### 3. Login =>

- a. Customer or admin can Login by entering username, Password and selecting Role.
  - i. URL: **/user/show/{showID}/bookings**
  - ii. Method : POST
  - iii. Login Customer

Body

```
1 {  
2   "name": "ayesha",  
3   "password": "ayesha123",  
4   "role": "customer"  
5 }
```

Response

---

Status: 200    Time: 88 ms

```
1 Login as customer
```

#### iv. Login Admin

##### Body

```
1 {  
2   "name": "piyush",  
3   "password": "piyush123",  
4   "role": "admin"  
5 }
```

##### Response

---

Status: 200    Time: 11 ms

1    Login as admin

# JUNIT Test Case

We have added 2 custom test cases

1. Test city => asserts true for original value from data base and expected value

```
31
32 @Test
33 public void testCity() throws Exception {
34
35     mockMvc.perform(get("/Testcity/1")).andExpect(status().isOk())
36             .andExpect(content().contentType("application/json;"))
37             .andExpect(jsonPath("$.pincode").value("590056")).andExpect(jsonPath("$.state").value("Karnataka"));
38
39 }
40
```

JUnit 5  
Finished after 6.164 seconds  
Runs: 2/2 Errors: 0 Failures: 0  
> TestCity [Runner: JUnit 5] (0.283 s) Failure Trace

2. Test Movie => asserts true for original value from data base and expected value

```
26 @Test
27 public void testMovie() throws Exception {
28
29     mockMvc.perform(get("/Testmovie/3")).andExpect(status().isOk())
30             .andExpect(content().contentType("application/json;"))
31             .andExpect(jsonPath("$.director").value("WAR 3 directors")).andExpect(jsonPath("$.name").value("WAR 3"));
32
33 }

```

JUnit 5  
Finished after 4.727 seconds  
Runs: 2/2 Errors: 0 Failures: 0  
> TestMovie [Runner: JUnit 5] (0.231 s) Failure Trace

# Exception Handling

We have added following custom Exceptions

1. Invalid login => throws **invalid exception** when a user enter wrong credentials.

Body

```
1 {  
2     "name": "ayesha",  
3     "password": "piyush123",  
4     "role": "admin"  
5 }
```

Response

Status: 200    Time: 13 ms

```
1 invalid user
```

2. City Login => throws **invalid exception** when a particular city id not found in database.

Status: 200    Time: 18 ms

```
1 {  
2     "name": null,  
3     "state": null,  
4     "id": 0,  
5     "pincode": null  
6 }
```

