Team 1 – Movie Ticket Booking System.

GitHub Url : 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

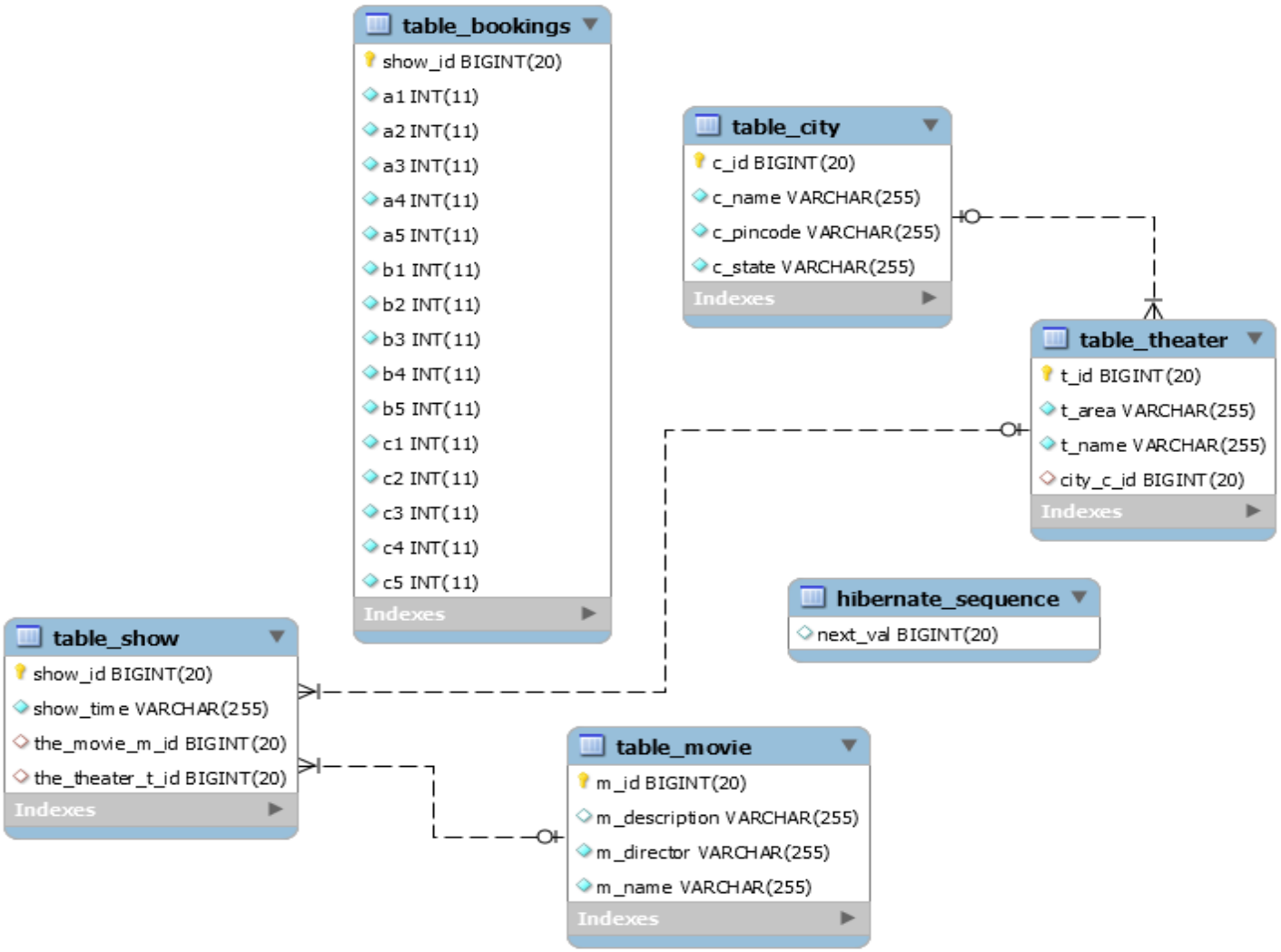
3. Admin can manage movies
    a. Add movies
    b. View all movies. Can also view a movie by id.
    c. Update movies
    d. Delete movies

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



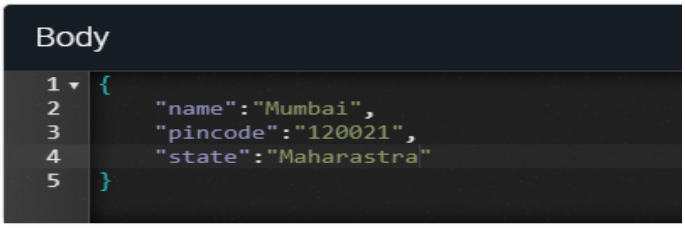
Movie Ticket Booking

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



Response

    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

```
{
    "name":"Mumbai mall",
    "area":"Dharavi"
}
```

Response

Status: 200 Time: 59 ms

```
{
    "t_name": "Mumbai mall",
    "t_area": "Dharavi",
    "t_id": 11
}
```

c. *To add the Movie to the DB*

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

Body

```
{
    "_name":"Dr. Strange",
    "_director":"Strange directors",
    "_description":"description about the Dr Starnge movie here"
}
```

Response

Status: 200 Time: 48 ms PRETTY

```
{
    "theShow": null,
    "_director": "Strange directors",
    "_description": "description about the Dr Starnge movie here",
    "_name": "Dr. Strange",
    "_id": 12
}
```

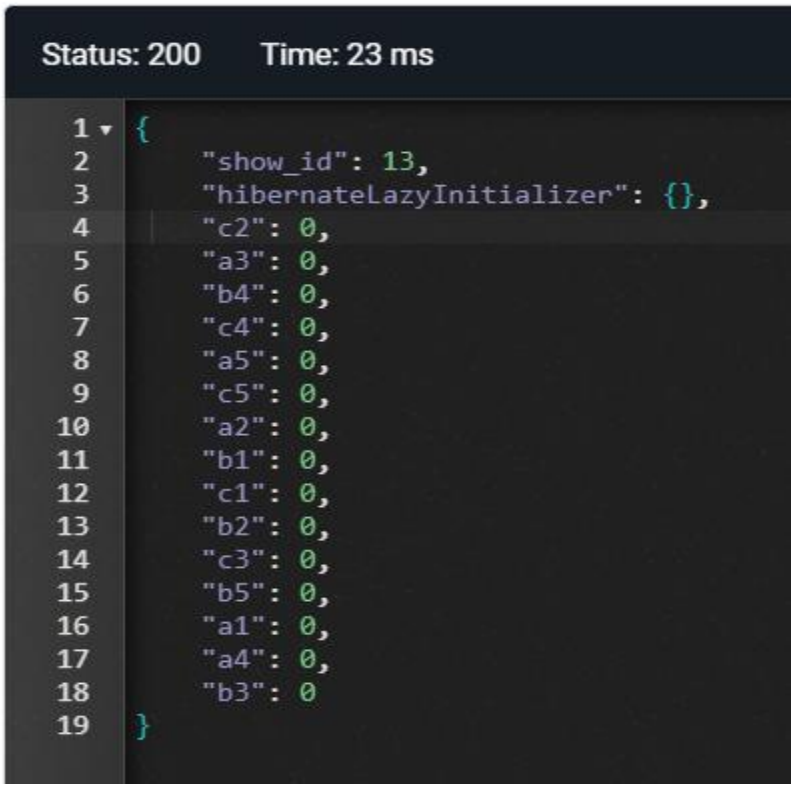
d. *To add a show* => TO add a show pass the theater Id and Movie id as a URL parameter and the show time in the request body.

   i. URL: **/api/{theater_id}/{movie_id}/show**
   ii. Method: POST

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



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


```
{
    "show_id": 13,
    "hibernateLazyInitializer": {},
    "c2": 1,
    "a3": 1,
    "b4": 0,
    "c4": 0,
    "a5": 0,
    "c5": 1,
    "a2": 1,
    "b1": 0,
    "c1": 1,
    "b2": 0,
    "c3": 0,
```

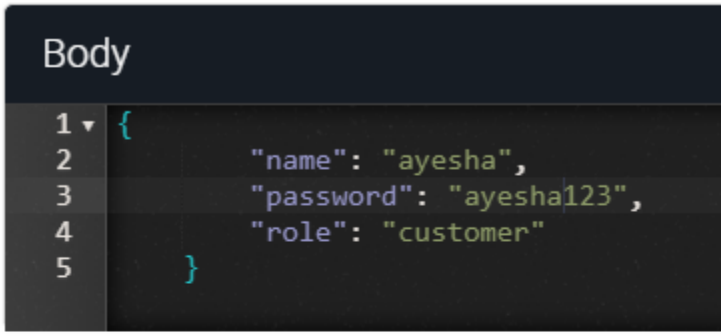

Response

iii.

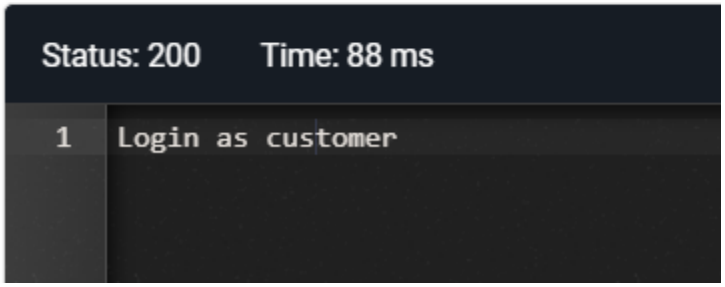
Status: 200 Time: 31 ms


```
{
    "show_id": 13,
    "c2": 1,
    "a3": 1,
    "b4": 0,
    "c4": 0,
    "a5": 0,
    "c5": 1,
    "a2": 1,
    "b1": 0,
    "c1": 1,
    "b2": 0,
    "c3": 0,
    "b5": 0,
    "a1": 0,
    "a4": 1,
    "b3": 1
}
```

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

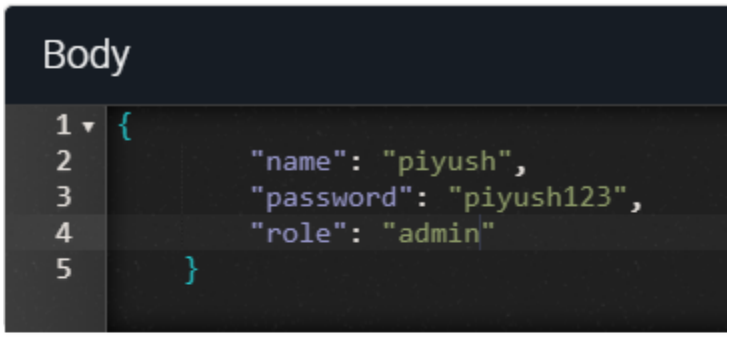


Response



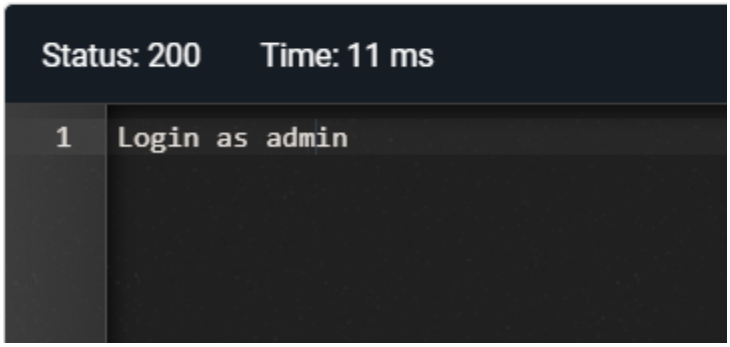
iv. Login Admin

Body



Response

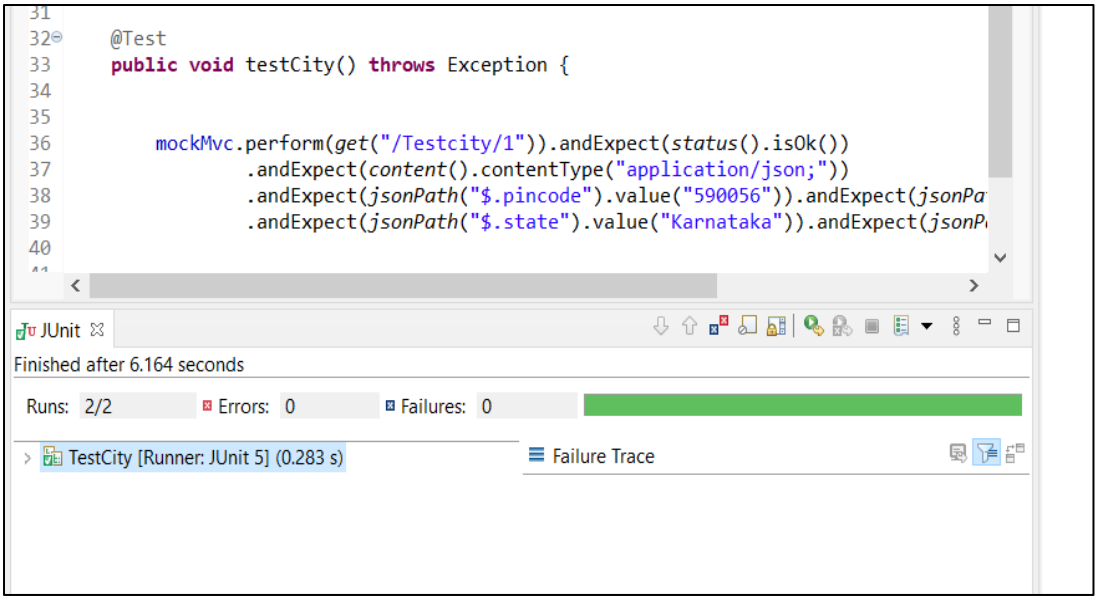
Status: 200 Time: 11 ms



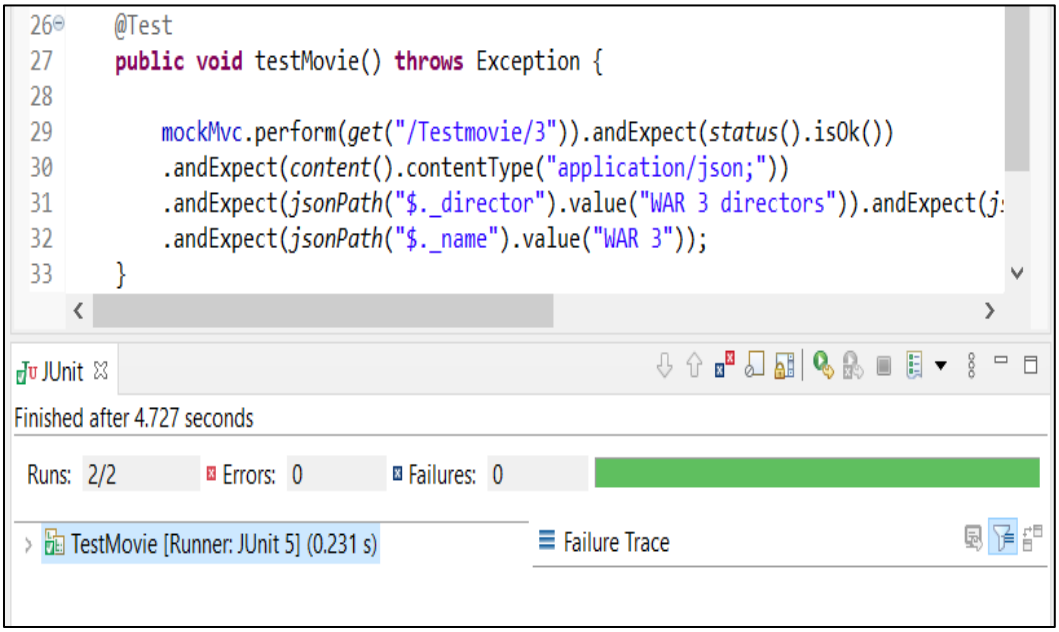
JUNIT Test Case

We have added 2 custom test cases

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



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



Exception Handling

We have added following custom Exceptions

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

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