

Project: Online Movie Ticket Booking

By

TEAM MEMBERS

1. EPSIKHA PATNAIK
2. GUJJALA ANUSHA
3. VISHNU
4. RAVITEJA
5. MOUNIKA
6. JAHNAVI
7. SANDEEP
8. VASU
9. DHIRAJ
10. RAVI SAIPRIYA

ABSTRACT

The project “Online Movie Ticket Booking System” is dedicated to the general requirements of multiplex theaters. The main objective of the project is to create an Online Movie Ticket Booking processing that allows customers to know about new movies, their schedules, cinema locations, class and ticket price etc. In the booking process when customer selects his city then cinemas of that city are filtered. In next step he/she selects his desired cinema where he/she wish to see movie, then selects movie and other details like show date, show time, class and no of tickets. Based on given parameters a graphical layout of seat status is visible to the customer. Now customer can select his desired seat location and number of seats. The Administrator will be able to see all booked and canceled tickets.

FUNCTIONAL SPECIFICATION

Project Code:	
Project Name:	Online Movie Ticket Booking

FUNCTIONAL SPECIFICATION

Table of Contents

- 1.INTRODUCTION
- 2.SYSTEM OVERVIEW
- 3.SUB-SYSTEM DETAILS
- 4.DATA ORGANIZATION
- 5. REST APIs to be Built
- 6.ASSUMPTIONS
- 7.EXPECTATIONS
- 8.ACCEPTANCE CRITERIA
- 9.TRACEABILITY TO REQUIREMENTS

FUNCTIONAL SPECIFICATION

1. Introduction

Online Movie ticket booking offers customers almost every service for booking cinema tickets and providing information about the movies and their schedule online including booking tickets, Transactions and payments. Movie Ticket Booking System has been developed to override the problems of existing manual system.

Admin can use Online Movie Ticket Booking System to insert and delete data such as movie description, movie schedule which will update the related webpages and will be accessible by the customers.

Online Movie Ticket Booking System provide another way for the customers to buy cinema ticket. This system reduces work load on customers, it is an automatic ticket booking system. This system is basically aimed to provide complete information of the movie and schedule to the customer, according to which customers can book the tickets.

1.1 Scope and Overview:

Scope of Online Movie Ticket Booking Services customers can book tickets 24 hours a day from anywhere in the world and interact with multiplex's website to know about currently running movies and their schedule or service information provided by the multiplex. Today the need of simplicity has driven application software programming to a new level. This project is a transaction related information storing project which will be used by the various multiplexes for online movie ticket booking through internet. Customers can view all currently running movies and book their tickets for any specific date and time. This application has a user friendly interface so that the customer and administrator can easily and efficiently use the software and its features.

1.3 Benefits

- To save yourtime efficiency.
- Easy to gather information about movies and theatres.
- To provide easy and faster access booking information.
- To provide user friendly environment.

1.4 Definitions, Acronyms, Abbreviations

MYSQL -	It is a relational model database serve produced by Microsoft.
SRS -	Software Requirements Specification.
EPM -	Employee Payroll Management
SLI -	Service Layer Interface

2. System Overview

The main purpose of this project is to provide a reliable, secure, efficient and user friendly environment to the customers and management authorities. Also benefit to the customer with efficient and faster service.

2.1. Product perspective

The Project “Online Movie Ticket Booking System” as a wide scope as it is generalized software and can be easily used in any ticket booking process system with little or no change. The Changes in software can be easily accommodated. The addition and deletion of the modules in software can be easily adjusted. This project has a lot of scope for further enhancement too. This project can save money and efforts in managing the record, just a mouse click can make the task easy and faster.

2.2. Functional Requirements:

The “Online Movie Ticket Booking” should support basic functionalities for all below listed users.

➤ User

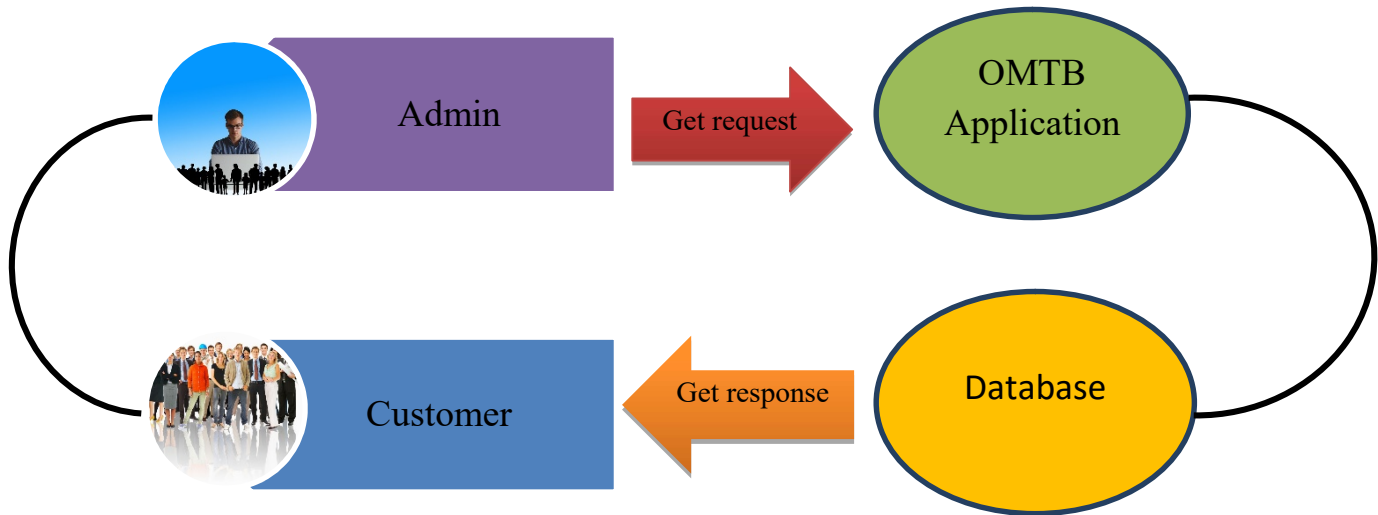
- Register into the application
- Sign into the application to maintain the record of activities
- Change Password
- Update Profile
- Registration according to category (customer or admin).

➤ Admin

- Login to Portal
- Create Admin Account
- Add/Edit/delete the Customers.
- Add/Edit/delete the Movie.
- Add/Edit/delete the show.

- Add/Edit/Delete the screen.
- Add/Edit/Delete the theatre.

2.3. Functional overview:



2.4. Assumptions and Dependencies

Assumptions:

- The code should be free with compilation errors/syntax errors.
- The product must have an interface which is simple enough to understand.

Dependencies:

- All necessary hardware and software are available for implementing and use of the tool.
- The proposed system would be designed, developed and implemented based on the software requirements specifications document.
- End users should have basic knowledge of computer and we also assure that the users will be given software training documentation and reference material.
- The system is not required to save generated reports

3.1 External interface requirements

3.1.1 User interfaces

The software provides good graphical interface for the front end which is self explanatory.

3.1.2 Hardware interfaces

- Memory minimum of 1GB RAM
- Hard disk of 40 GB
- Monitor
- Mouse
- Keyboard
- Printer

3.1.3 Software interfaces

- | | |
|--------------------|------------------|
| • Operating System | Windows XP |
| • Front End | Angular |
| • Backend | MYSQL, Hibernate |
| • Server Side | Spring Boot |
| • Server | Tomcat |

3.1.4 Project Overview

The following subsections provide the complete overview of the software specifications requirements documentation for the Online Movie Ticket Booking. The entire SRS is documented in view of User and the following sub sections are arranged to give a complete outlook of the software, its perspective, features, system requirements and users know how it is.

3.2 Sub-Functional Overview:

3.2.1 Admin

This module helps the administrator to add the customers and booking seats for the customers and update the status for Movies releases and its availability and to add or edit or delete the seats, screen, theatre etc.

3.2.2 User

This module helps to add and update the details of the user like the personal detail and the booking details and transactions.

3.2.3 Theatre

This module helps to search the theatre details based on theatre at particular city and area.

3.2.4 Screen

This module helps to select the screen based on the theatreid.

3.2.5 Seats

This module helps to select the customer seat as per the availability according to theatres and screens.

3.3 Booking

This module helps customers or user to book the movies as per the requirements.

Booking should be done along with transactions.

3.3.1 Movie

This module helps user or customer to select the movie as per the releases.

3.4 Design constraints

- The system runs under Windows XP.

- The application is developed on Angular and MYSQL server as back end..

3.5 Authentication & Authorization

Authentication is a process designed to confirm whether a user has the right to perform a specific operation or access a specific resource (e.g. a file).

The purpose of authorization is to control the access, therefore it takes place after authentication, i.e. the operation of establishing user id.

Take the example of an online Movie Ticket Booking system - where, after logging in, the user can perform certain operations during an active session. When the session expires, the user loses their rights until the next correct login.

3.5.1 Attributes

3.5.2 Reliability

In order to ensure reliability, this system is being designed using software that is established to be stable and easy to use.

3.5.3 Availability

This system is designed to run 24/7 and be readily available to the user.

3.5.4 Security

The access to the software is given only to valid operators. We need a specific ID and password to get access to the software.

3.5.6 Environment:

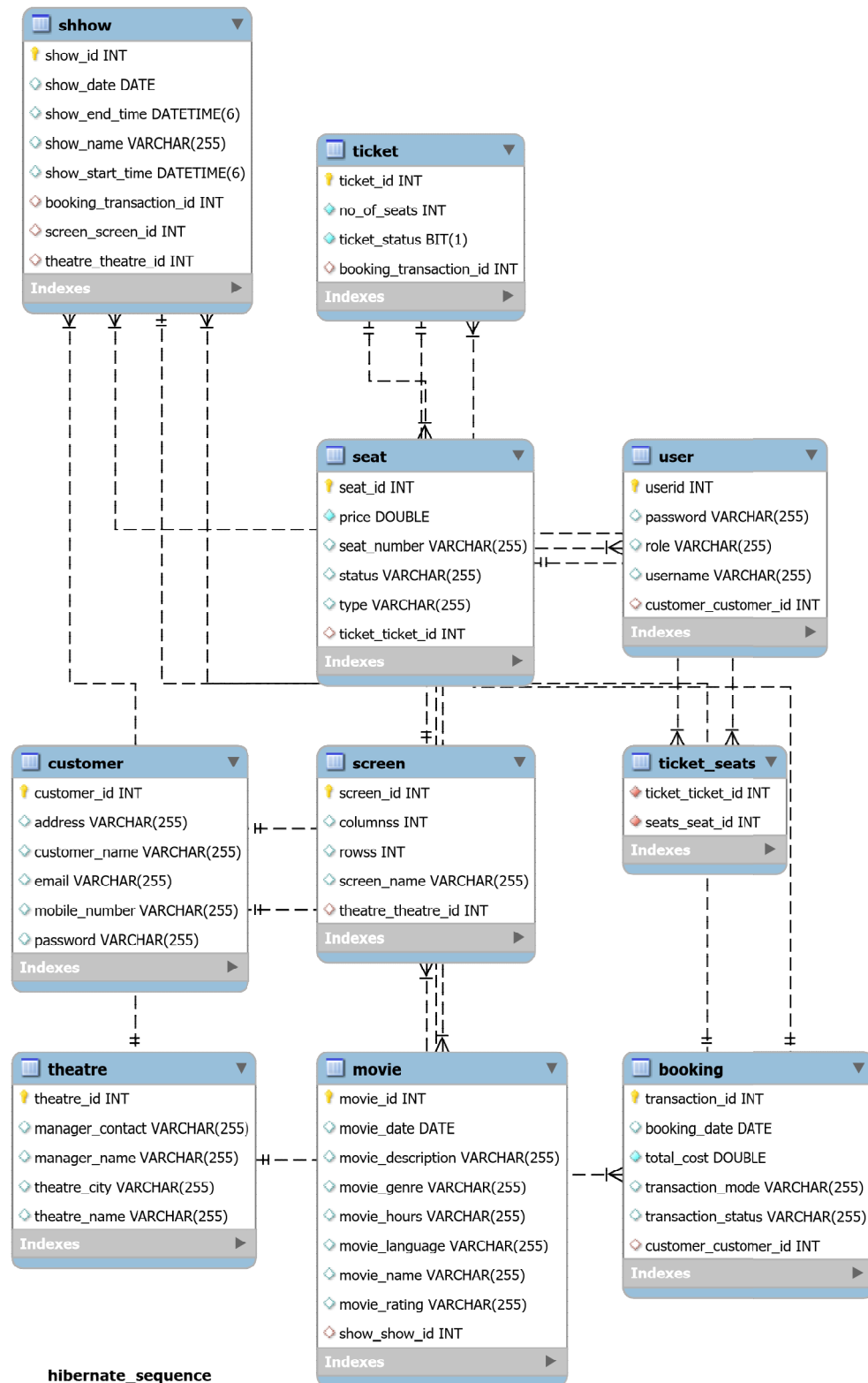
The system will be developed on any Windows OS machine using J2EE, Hibernate and Spring.

- Intel hardware machine (PC P4-2.26 GHz, 512 MB RAM, 40 GB HDD)
- Server – Apache Tomcat 8 or higher
- Database – MYSQL or higher
- JRE 8

- Eclipse IDE or Spring Tool Suite

4.0 Database Organization

This section explains the data storage requirements of ONLINE MOVIE TICKET BOOKING data description along with suggested table (database) structure. The following section explains few of the tables (fields) with description.



Online Movie Ticket Booking

➤ USER DATABASE

(1) USER :

Field name	Description
User id	User id is auto generated after registration and it is used as Login id.
Password	User password
Username	Username of the customer or admin
Role	User role
Customer_customer_id	Customer id

(2)CUSTOMER:

Field name	Description
Customer id	Customer id
Address	Address of customer
Customer name	Name of the customer
Email	Customer email
Phone number	10 digit contact number of customer
Password	Customer password

(3) MOVIE:

Field name	Description
Movie id	Movie id
Movie date	Screening date of movie
Movie description	Description type of movie(behind story)
Movie genre	Type of movie(Eg:action, adventure)
Movie hours	Duration of movie
Movie language	Language of movie(English,teluguetc)
Movie name	Name of the movie
Movie rating	Ranking of movie
Show_show_id	Show id

(4) THEATRE :

Field name	Description
Theatre id	Theatre id
Manager contact	10 digit contact number of manager
Manager name	Name of the manager
Theatre city	City in which theatre is located
Theatre name	Name of the theatre

Online Movie Ticket Booking

(5) SCREEN :

Field name	Description
Screen id	Screen id for the movie
Columns	No of column
Rows	No of the rows
Screen name	Name of the screen
Theatre_theatre_id	Theatre id

(6) SEAT :

Field name	Description
Seat id	Seat id for the customers
Price	Price of the ticket
Seat number	Seat number
Status	Status for seat availability
Seat type	Type of the seat
Ticket_ticket_id	Address of the user

(7) SHOW :

Field name	Description
Show id	Show id for the movie
Show date	Date for the show
Show name	Name of the movie
Show start time	Start time for the movie
Show end time	End time for the movie
Booking transaction id	Transaction id for the booking
Screen screen id	Screen id for the show
Theatre theatre id	theatre id for the movie

(8) TICKET :

Field name	Description
Ticket id	Ticket id for the required movie
No of seats	No of available seats
Ticket status	Status of ticket availability
Booking transaction id	Transaction id for booking a movie ticket

(9) TICKET SEATS :

Field name	Description
Ticket ticket id	Ticket id for seats
Seats seat id	Seat id for the required ticket id.

(10) BOOKING:

Field name	Description
Transaction id	Transaction id for booking
Booking date	Date for booking movie ticket
Total cost	Cost for the movie ticket
Transaction mode	Mode of transaction
Transaction status	Status for the transaction
Customer customer id	Customer for the transaction and booking

FUNCTIONAL SPECIFICATION

5. REST APIs to be Built.

1. Creating User Entity: Create Spring Boot with Micro services Application with Spring Data JPA

Technology stack:

- Spring Boot
- Spring REST
- Spring Data JPA

Here will have multiple layers into the application:

1. Create an Entity: User
2. Create a UserRepository interface and will make use of Spring Data JPA
 - a. Will have findByUserName method.
 - b. Add the User details

Create a UserService class and will expose all these services.

4. Finally, create a UserRestController will have the following Uri's:



Online Movie Ticket Booking

URL	METHOD	DESCRIPTION	Format
/user/username	GET	Give a single user description searched based on username	JSON
/user/userid	GET	Give a single user description searched based on user id	String
/users	POST	Add the user details	JSON
/users	PUT	Update the user details	JSON
/user/userid	DELETE	Delete user by id	String

POST http://localhost:9070/user/adduser... Send

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

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (8) Test Results Status: 200 OK Time: 4.39 s Size: 447 B Save Response

Pretty Raw Preview Visualize JSON

```
2  "userid": 154,
3  "username": "vidhya",
4  "password": "capg@1",
5  "role": "customer",
6  "customer": {
7    "customerId": 153,
8    "customerName": "vidhya",
9    "address": null,
10   "mobileNumber": null,
11   "email": null,
12   "password": "capg@1"
13 }
```

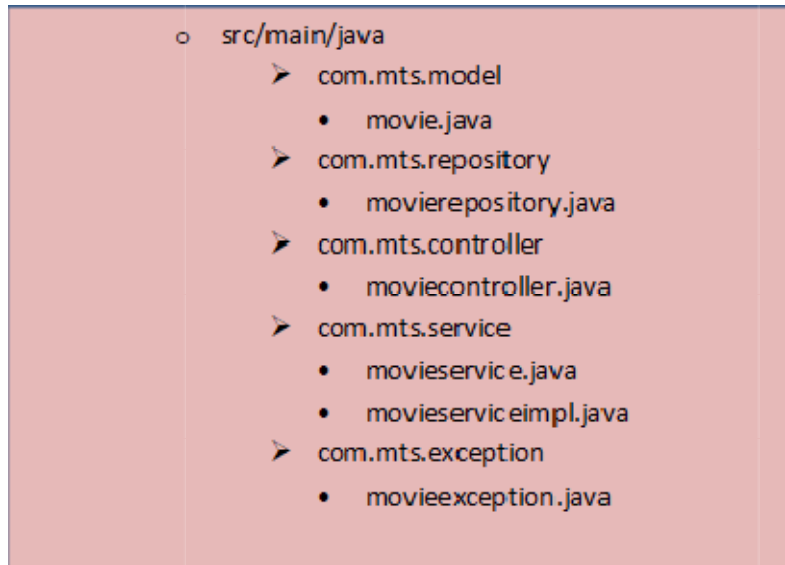
Activate Windows

2. Creating Movie Entity:

Build a RESTful resource for Movie manipulations, where CRUD operations to be carried out. Here will have multiple layers into the application:

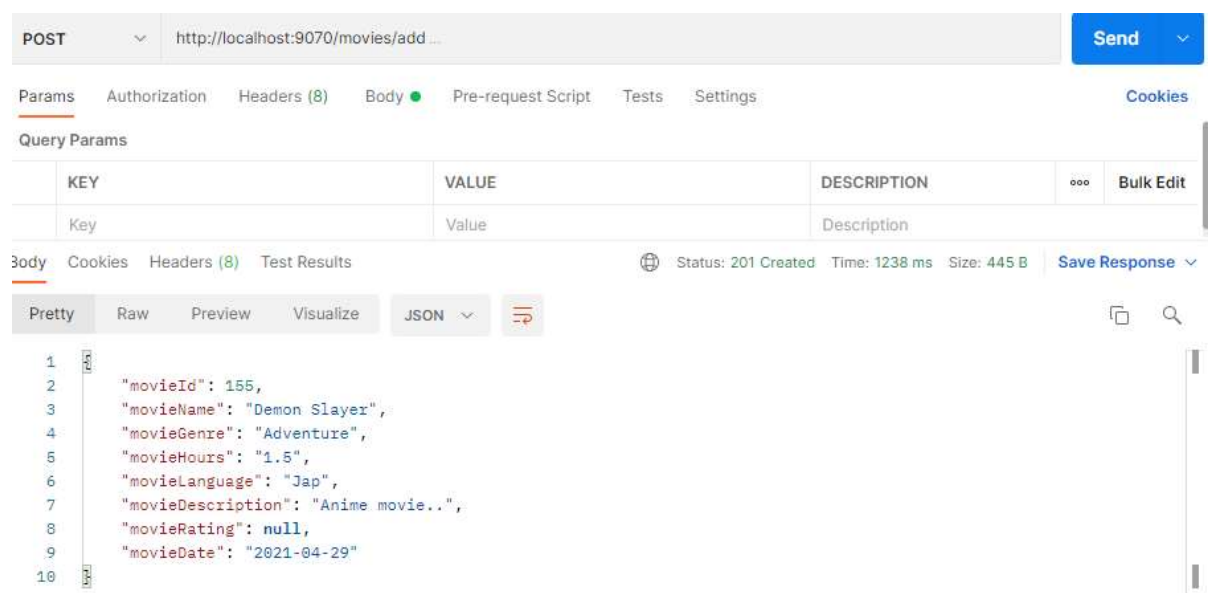
1. Create an Entity: Movie
2. Create a MovieRepository interface and will make use of Spring Data JPA
 - a) Will have findByMovieName method.
 - b) Add the Movie details method.
 - c) Will have deleteMovieById method.
 - d) Will have findAllMovie method.
3. Create a MovieService class and will expose all these services.

Online Movie Ticket Booking



4. Finally, create a MovieRestController will have the following Uri's:

URL	METHOD	DESCRIPTION	FORMAT
/movie/moviename	GET	Give a single Movie description searched based on moviename	JSON
/movie/movieid	GET	Give a single user description searched based on Movie id	String
/movies	POST	Add the user details	JSON
/movies	PUT	Update the movie details	JSON
/movie/movieid	DELETE	Delete movie by id	String



2. Creating Customer Entity:

Build a RESTful resource for customer manipulations, where CRUD operations to be carried out. Here will have multiple layers into the application:

Create an Entity: Customer

2. Create a CustomerRepository interface and will make use of Spring Data JPA

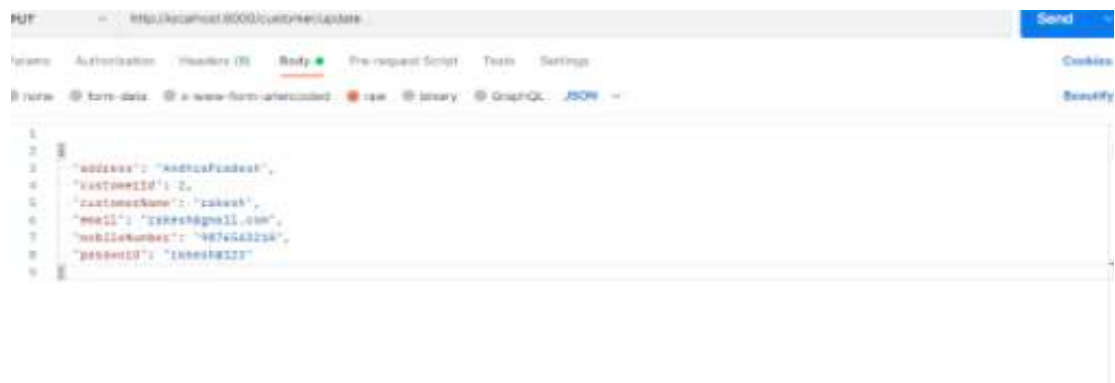
- a) Will have findByCustomerName method.
- b) Add the Movie details method.
- c) Will have deleteCustomerById method.
- d) Will have findAllCustomermethod.

3. Create a CustomerService class and will expose all these services



4. Finally, create a CustomerRestController will have the following Uri's:

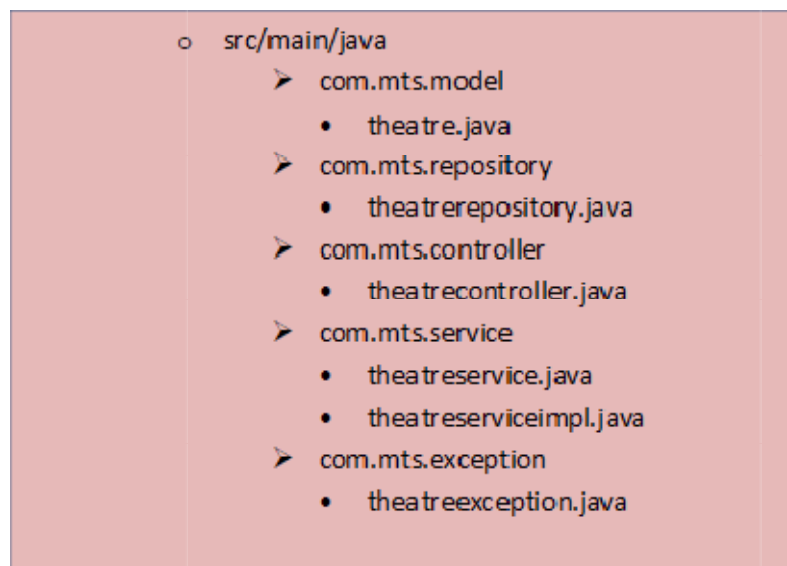
URL	METHOD	DESCRIPTION	FORMAT
/customer/customername	GET	Give a single user description searched based on Customername	JSON
/customer/customerid	GET	Give a single user description searched based on Customer id	String
/customers	POST	Add the Customer details	JSON
/customers	PUT	Update the Customer details	JSON
/customer/customerid	DELETE	Delete Customer by id	String



2. Creating Entity:

Build a RESTful resource for manipulations, where CRUD operations to be carried out. Here will have multiple layers into the application:

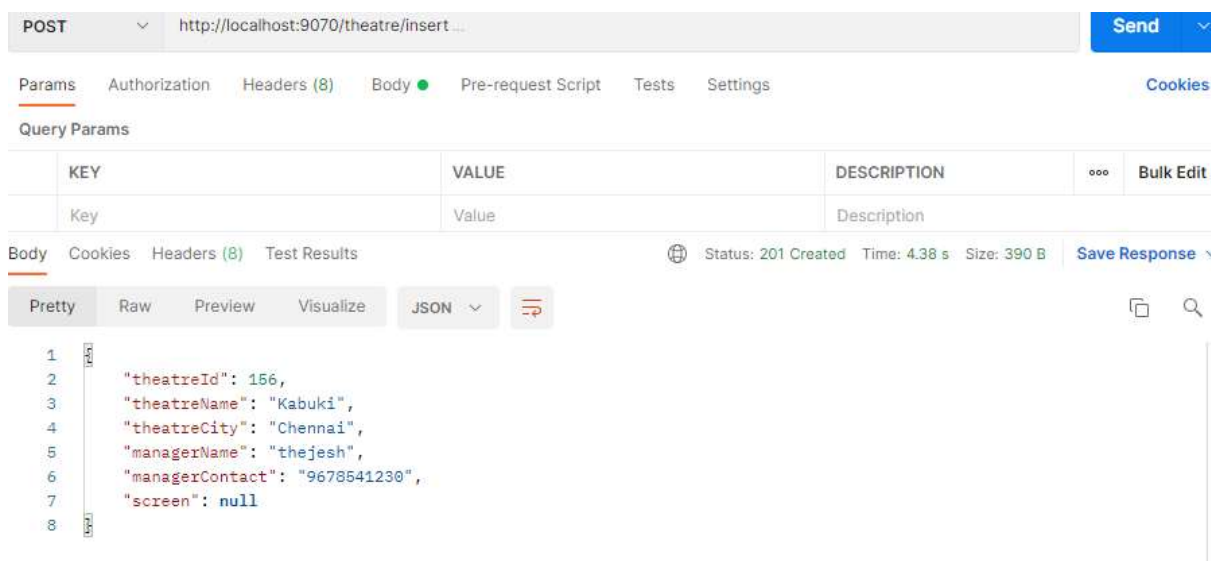
1. Create an Entity: Theatre
2. Create a TheatreRepository interface and will make use of Spring Data JPA
 - a) Will have findByTheatreName method.
 - b) Add the Movie details method.
 - c) Will have deleteTheatreById method.
 - d) Will have findAllTheatre method.
3. Create a TheatreService class and will expose all these services.



Online Movie Ticket Booking

4. Finally, create a TheatreRestController will have the following Uri's:

URL	METHOD	DESCRIPTION	FORMAT
/theatre/theatrename	GET	Give a single Theatre description searched based on theatre id	JSON
/theatre/theatreid	GET	Give a single Theatre description searched based on Theatre id	String
/theatres	POST	Add the Theatre details	JSON
/theatres	PUT	Update the Theatre details	JSON
/theatre/theatreid	DELETE	Delete Theatre by id	String

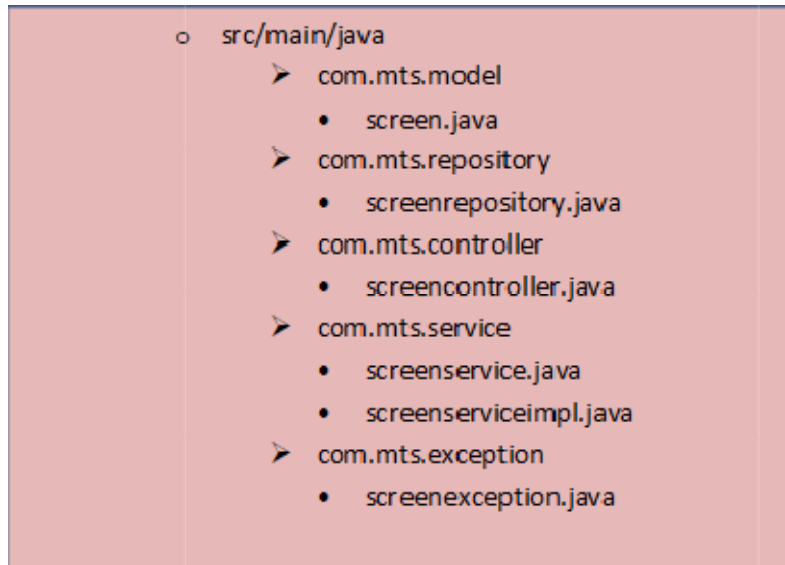


2. Creating Screen Entity:

Build a RESTful resource for Movie manipulations, where CRUD operations to be carried out. Here will have multiple layers into the application:

1. Create an Entity: Screen
2. Create a ScreenRepository interface and will make use of Spring Data JPA
 - a) Will have `findByScreenName` method.
 - b) Add the Screen details method.
 - c) Will have `deleteScreenById` method.
 - d) Will have `findAllScreen` method.
3. Create a ScreenService class and will expose all these services.

Online Movie Ticket Booking



4. Finally, create a ScreenRestController will have the following Uri's:

URL	METHOD	DESCRIPTION	FORMAT
/screen/screen name	GET	Give a single user description searched based on screen name	JSON
/screen/screen id	GET	Give a single user description searched based on screen id	String
/screens	POST	Add the screen details	JSON
/screens	PUT	Update the screen details	JSON
/screen/screen id	DELETE	Delete screen by id	String

POST http://localhost:9070/screens/add?theatreId=... Send

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

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> theatreId		
Key	Value	Description

Body Cookies Headers (8) Test Results Status: 200 OK Time: 84 ms Size: 323 B Save Response

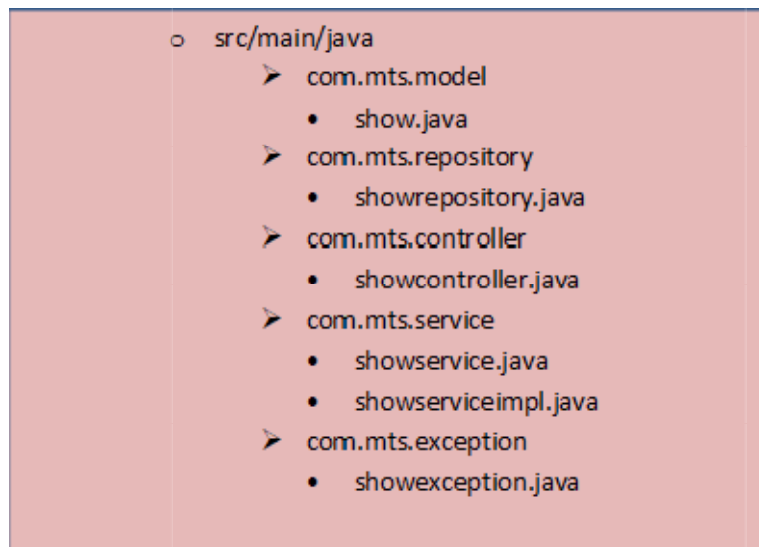
Pretty Raw Preview Visualize JSON

```
1 {
2   "screenId": 0,
3   "show": null,
4   "screenName": "screen2",
5   "rows": 4,
6   "columns": 5
7 }
```

2. Creating Show Entity:

Build a RESTful resource for show manipulations, where CRUD operations to be carried out. Here will have multiple layers into the application:

1. Create an Entity: show
2. Create a showRepository interface and will make use of Spring Data JPA
 - a) Will have findByshowName method.
 - b) Add the Movie details method.
 - c) Will have deleteshowById method.
 - d) Will have findAllshow method.
3. Create a showService class and will expose all these services.



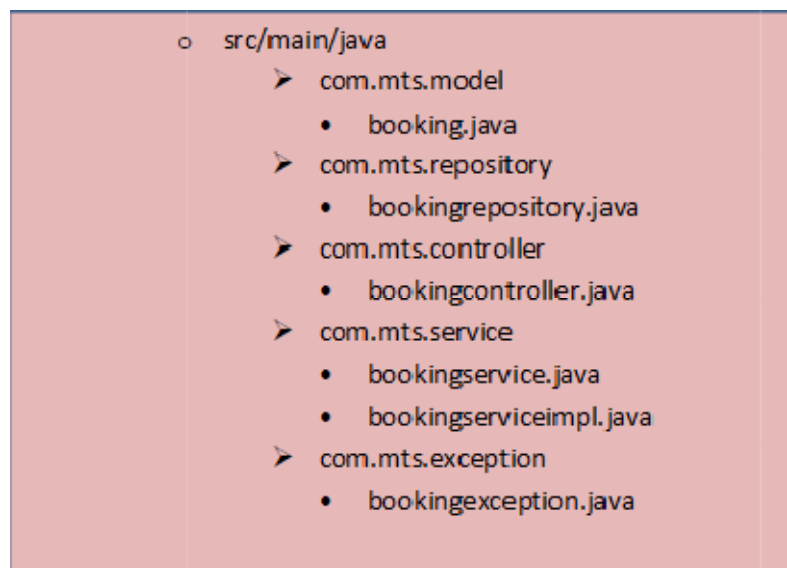
4. Finally, create a showRestController will have the following Uri's:

URL	METHOD	DESCRIPTION	FORMAT
/show/show id	GET		JSON
/show/show id	GET	Give a single user description searched based on user id	String
/shows	POST	Add the user details	JSON
/shows	PUT	Update the user details	JSON
/show/show id	DELETE	Delete user by id	String

2. Creating booking Entity:

Build a RESTful resource for booking manipulations, where CRUD operations to be carried out. Here will have multiple layers into the application:

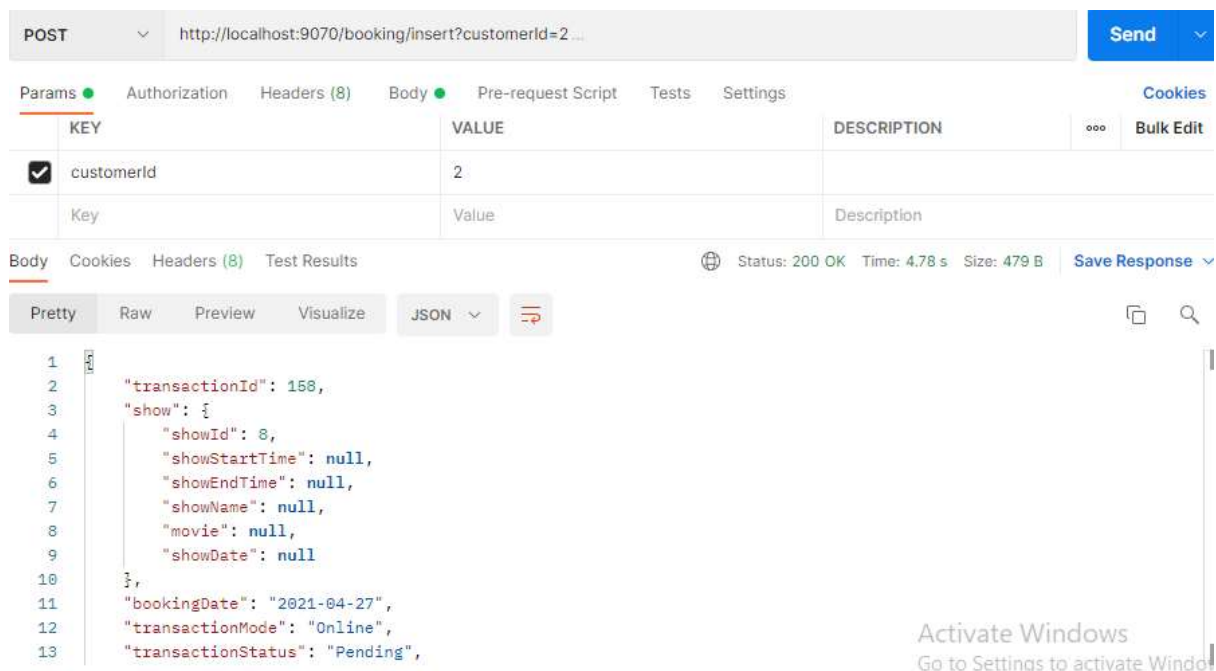
1. Create an Entity:
2. Create a BookingRepository interface and will make use of Spring Data JPA
 - a) Will have findByBookingName method.
 - b) Add the Booking details method.
 - c) Will have deleteBookingById method.
 - d) Will have findAllBookingmethod.
3. Create a BookingService class and will expose all these services.



4. Finally, create a BookingRestController will have the following Uri's:

URL	METHOD	DESCRIPTION	FORMAT
/booking/transaction id	GET		JSON
/booking/transaction id	GET	Give a single user description searched based on user id	String
/bookings	POST	Add the user details	JSON
/bookings	PUT	Update the user details	JSON
/booking/bookind id	DELETE	Delete user by id	String

Online Movie Ticket Booking



2. Creating ticket Entity:

Build a RESTful resource for ticket manipulations, where CRUD operations to be carried out. Here will have multiple layers into the application:

1. Create an Entity: ticket
2. Create a ticketRepository interface and will make use of Spring Data JPA
 - a) Will have `findByticketName` method.
 - b) Add the ticket details method.
 - c) Will have `deleteticketById` method.
 - d) Will have `findAllticket` method.

Creating Seat Entity:

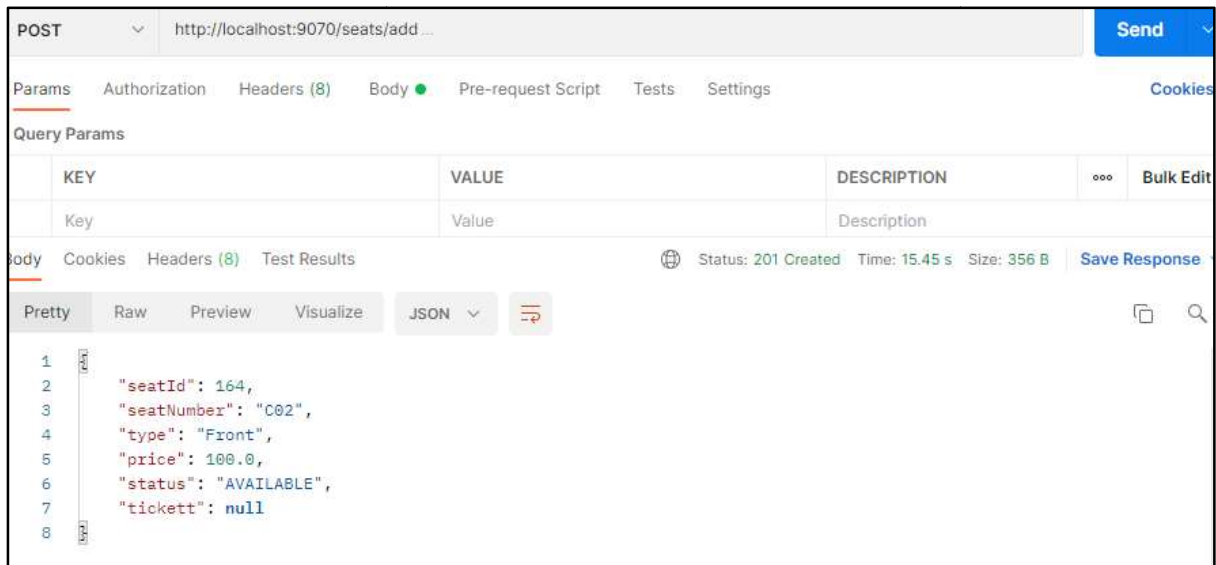
Build a RESTful resource for seat manipulations, where CRUD operations to be carried out. Here will have multiple layers into the application:

1. Create an Entity: seat
2. Create a seatRepository interface and will make use of Spring Data JPA
 - a) Will have findByseatName method.
 - b) Add the seat details method.
 - c) Will have seatMovieById method.
 - d) Will have findAllseat method.
3. Create a seatService class and will expose all these services.

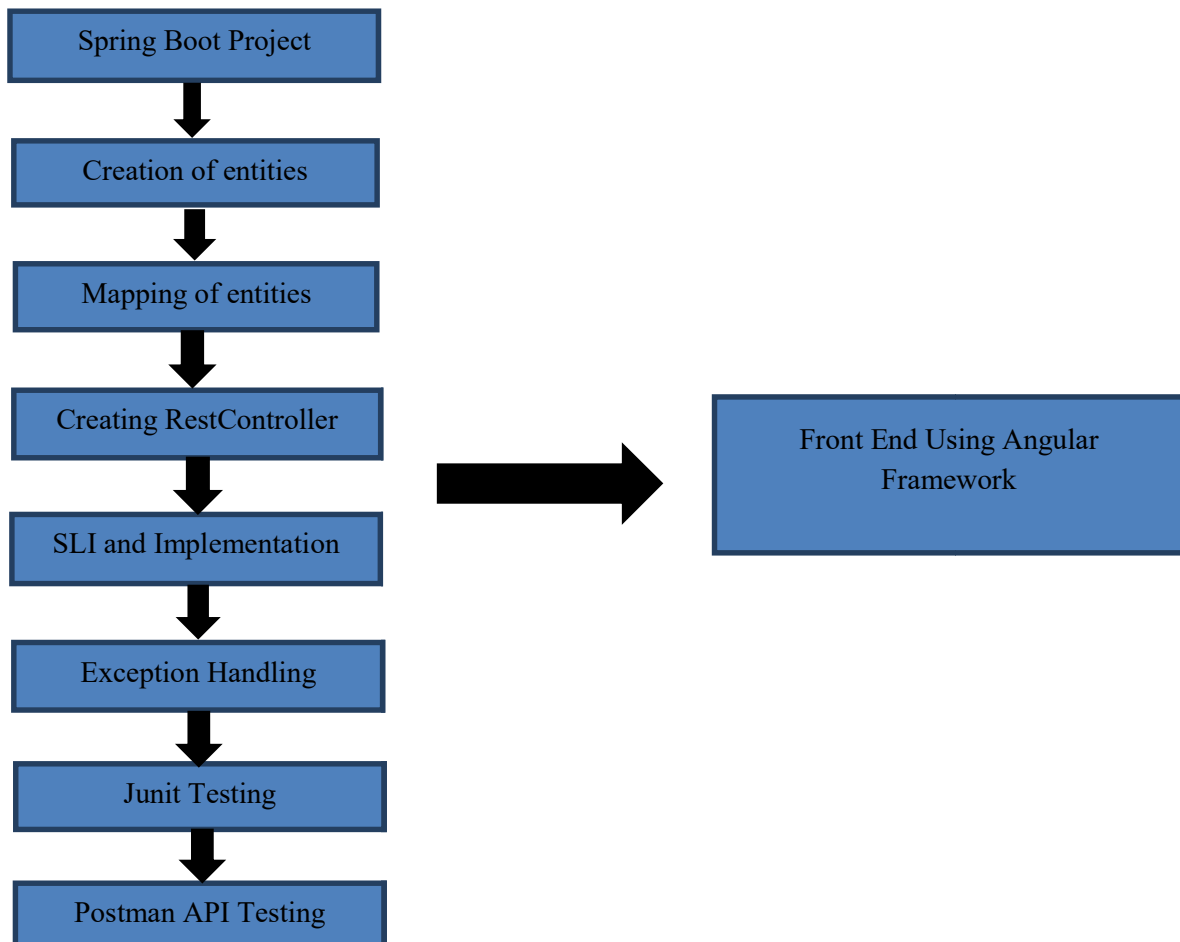


3. Finally, create a seatRestController will have the following Uri's:

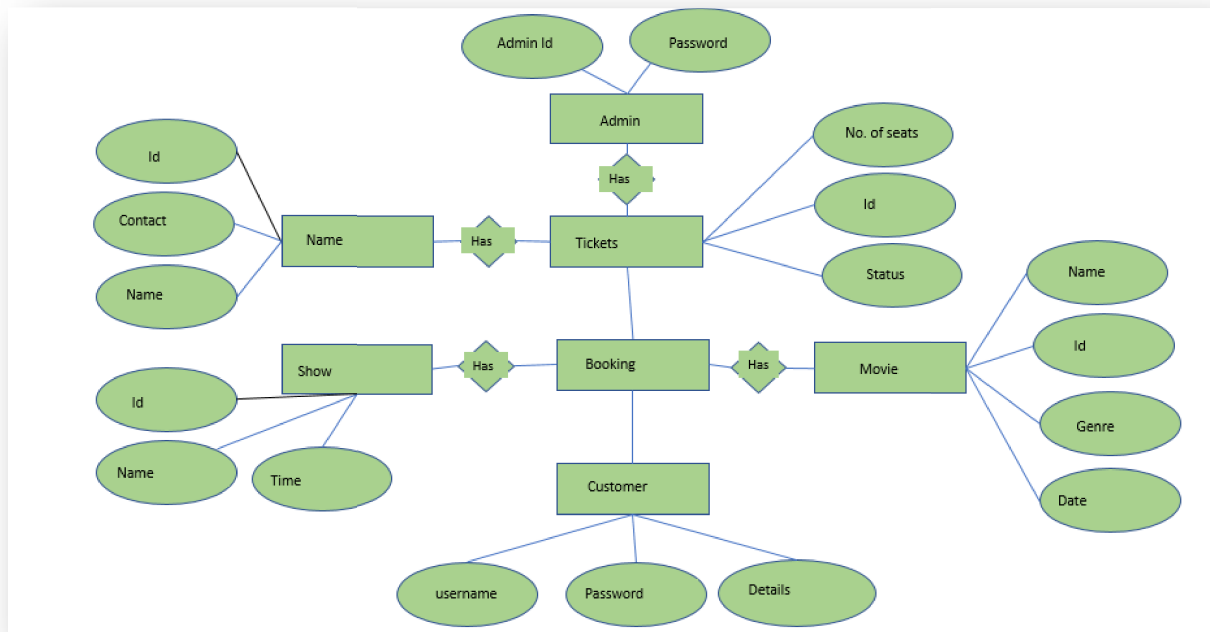
URL	METHOD	DESCRIPTION	FORMAT
	GET		JSON
/seat/seat id	GET	Give a single user description searched based on user id	String
/seats	POST	Add the user details	JSON
/seats	PUT	Update the user details	JSON
/seat id	DELETE	Delete user by id	String



5. SEQUENCE DIAGRAM:



1. ER DIAGRAM:



2. User Sequence Diagram:

User sequence diagram is used for the customer/user requirements to login into the application and to use the available details based on:

Movie

Theatre

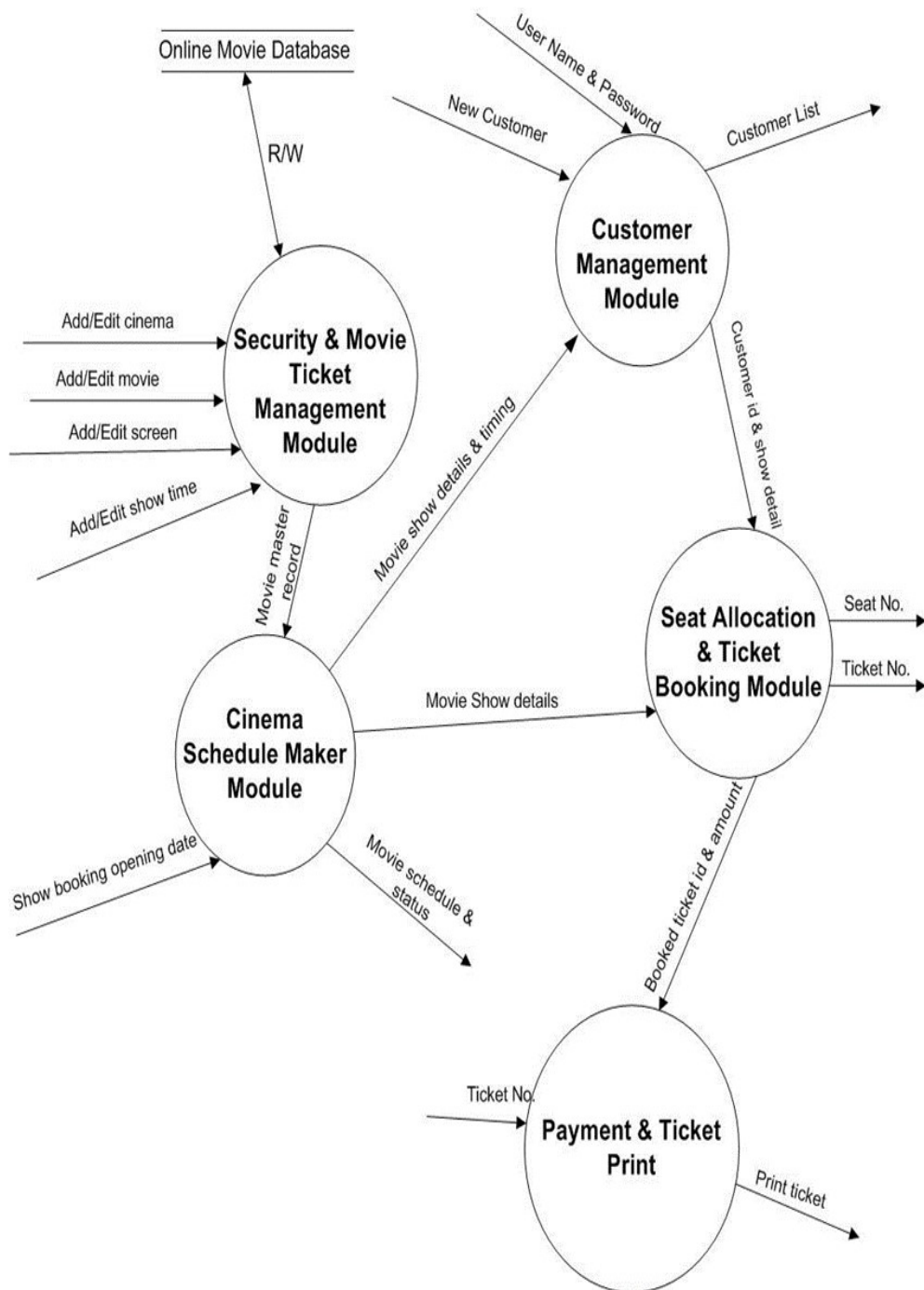
Screen

Seat

Ticket

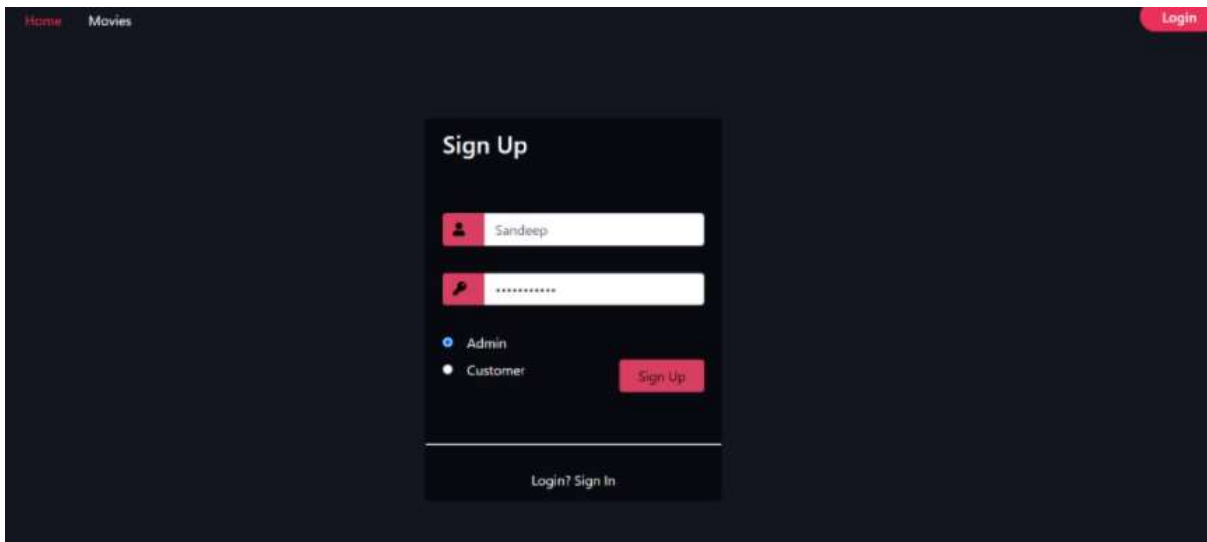
Booking

Online Movie Ticket Booking



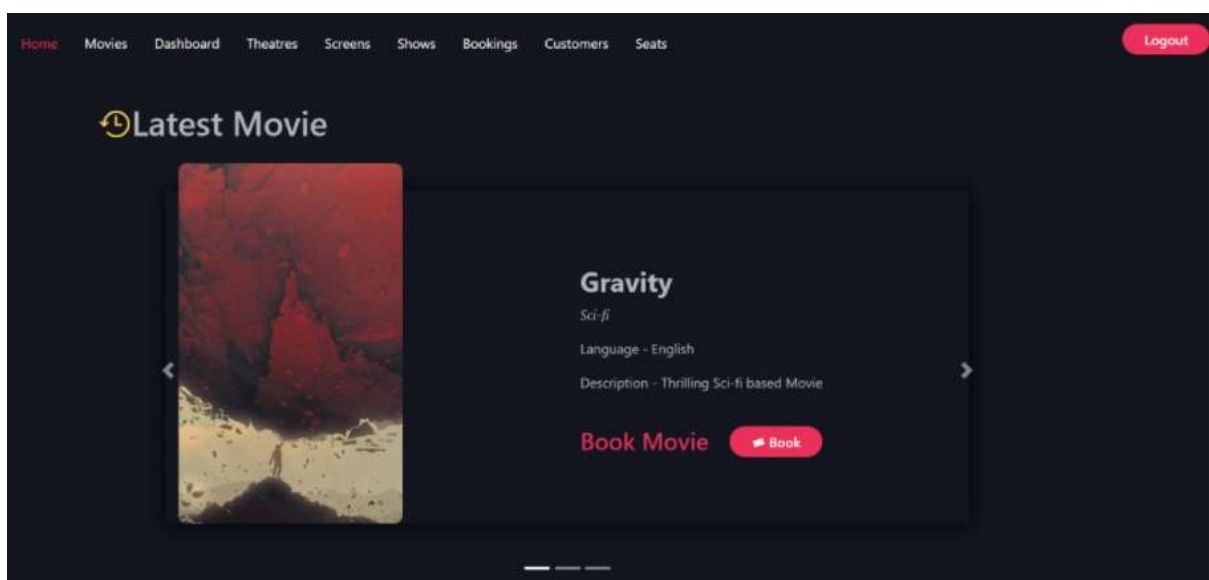
FRONTEND OUTPUT:

Admin and User Login Page:



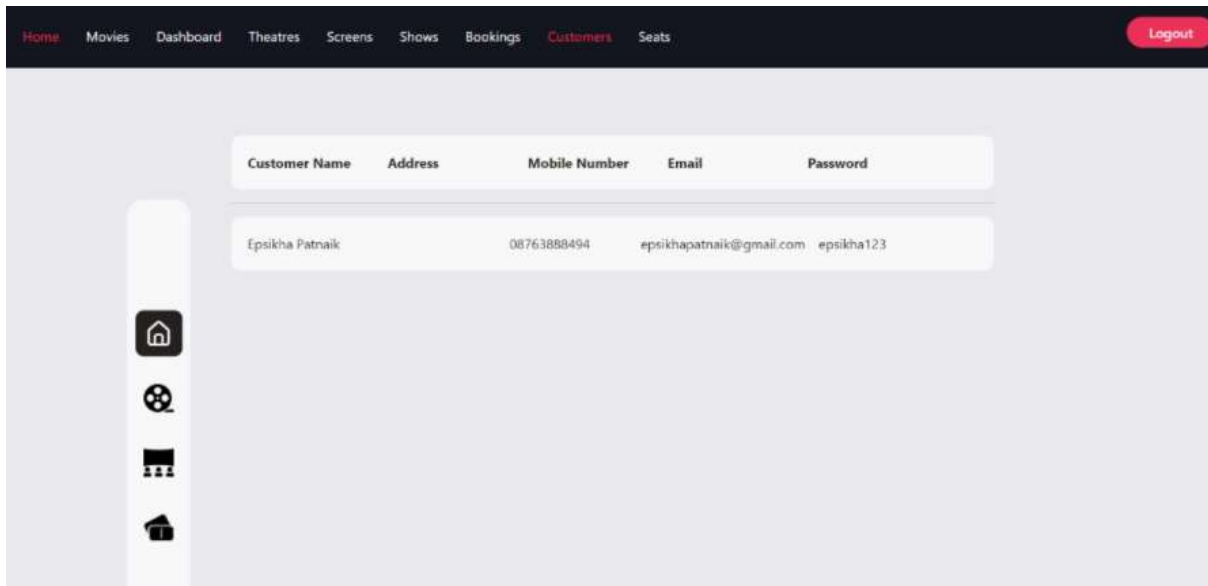
The screenshot shows a 'Sign Up' form on a dark background. At the top left are links for 'Home' and 'Movies', and at the top right is a 'Login' button. The form itself is centered and contains two input fields: the first is for a name, with 'Sandeep' entered, and the second is for a password, shown as asterisks. Below these fields are two radio buttons: 'Admin' (selected) and 'Customer'. A red 'Sign Up' button is to the right of the radio buttons. At the bottom of the form, there is a link that says 'Login? Sign In'.

Admin page:



Online Movie Ticket Booking

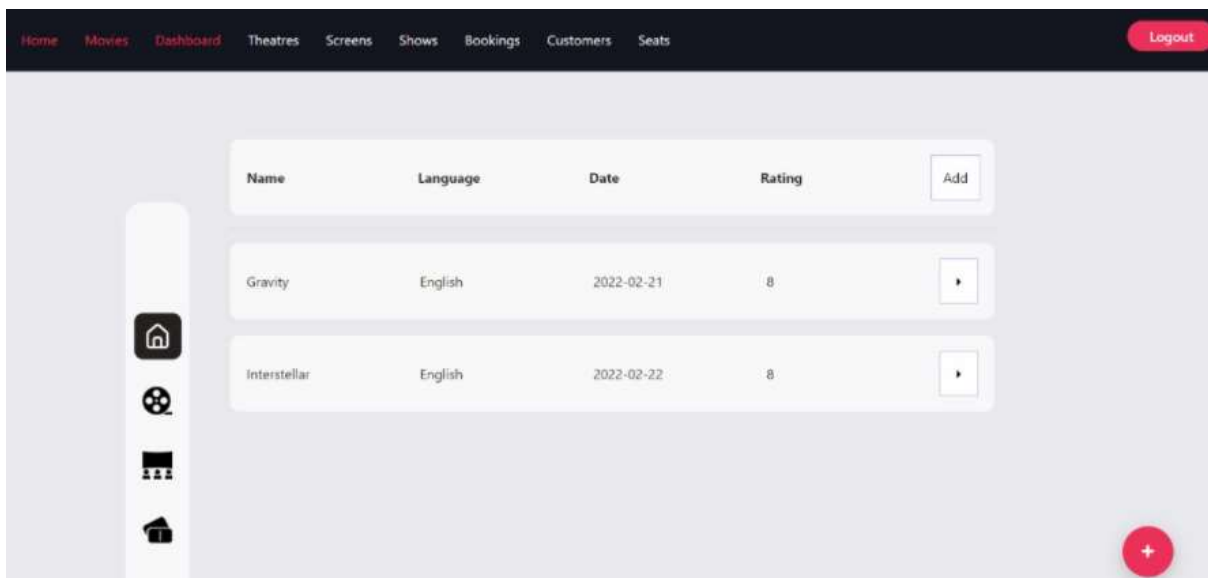
CustomerPage :





The screenshot shows the CustomerPage interface. At the top is a dark navigation bar with links: Home, Movies, Dashboard, Theatres, Screens, Shows, Bookings, Customers (highlighted), and Seats. A Logout button is in the top right. On the left is a vertical sidebar with icons for Home, Movies, Theatres, and Screens. The main content area features a table with customer information.

Customer Name	Address	Mobile Number	Email	Password
Epsikha Patnaik		08763888494	epsikhapatnaik@gmail.com	epsikha123

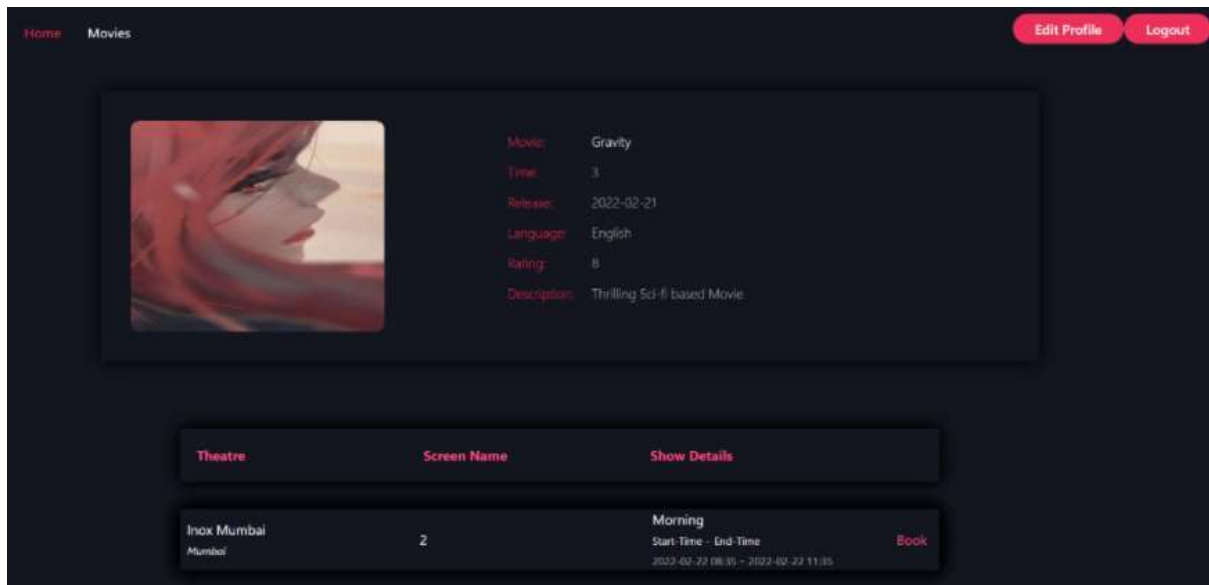
Movie booking page:



The screenshot shows the Movie booking page interface. It has the same navigation bar and sidebar as the CustomerPage. The main content area displays a table of movies with an 'Add' button and a red '+' button at the bottom right.

Name	Language	Date	Rating	Add
Gravity	English	2022-02-21	8	
Interstellar	English	2022-02-22	8	

Online Movie Ticket Booking



BACKEND OUTPUT :

JUnit Testing Output:



Online Movie Ticket Booking

Postman API Testing Outputs:

Output for Admin:

The screenshot shows a Postman interface for a POST request to `http://localhost:9070/admin/registeradmin/Ravi/Priy@14$...`. The status is 200 OK. The response body, viewed in 'Pretty' mode, contains the text `"CREATED"`.

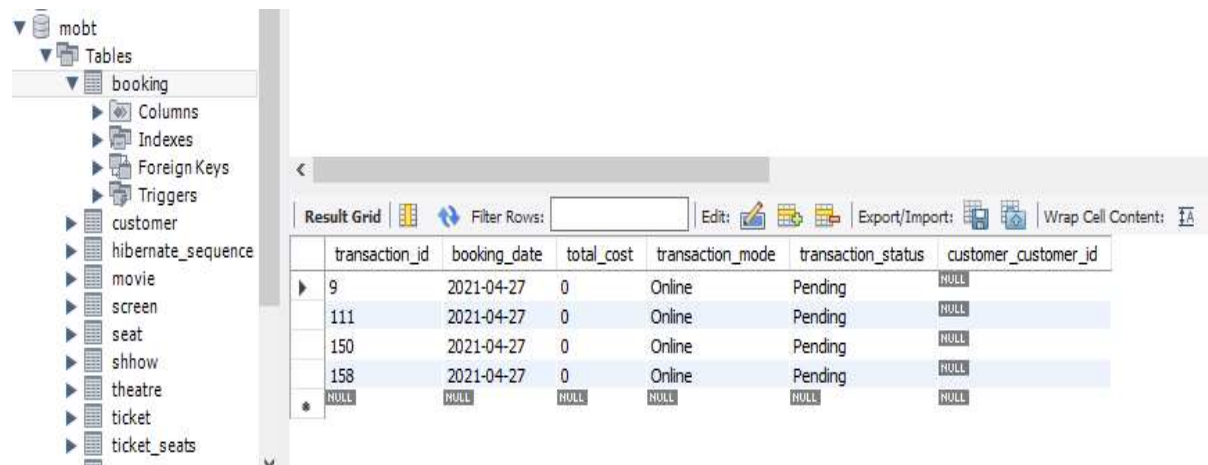
KEY	VALUE	DESCRIPTION
Key	Value	Description

Output for Booking:

The screenshot shows a Postman interface for a POST request to `http://localhost:9070/booking/insert?customerId=2...`. The status is 200 OK. The response body, viewed in 'Pretty' mode, contains a JSON object with booking details.

```
1 {
2   "transactionId": 158,
3   "show": {
4     "showId": 8,
5     "showStartTime": null,
6     "showEndTime": null,
7     "showName": null,
8     "movie": null,
9     "showDate": null
10  },
11   "bookingDate": "2021-04-27",
12   "transactionMode": "Online",
13   "transactionStatus": "Pending",
```

DATABASE CREATION OUTPUT:



The screenshot displays a database management interface. On the left, a tree view shows the 'mobt' database containing several tables: booking, customer, hibernate_sequence, movie, screen, seat, shhow, theatre, ticket, and ticket_seats. The 'booking' table is selected, and its details are shown on the right. The 'Result Grid' for the 'booking' table is displayed, showing a list of transactions. The grid has columns for transaction_id, booking_date, total_cost, transaction_mode, transaction_status, and customer_customer_id. The data shows four transactions with transaction IDs 9, 111, 150, and 158, all dated 2021-04-27, with a total cost of 0, transaction mode of 'Online', and transaction status of 'Pending'. The customer_customer_id is NULL for all transactions. A row with all NULL values is also shown at the bottom of the grid.

transaction_id	booking_date	total_cost	transaction_mode	transaction_status	customer_customer_id
9	2021-04-27	0	Online	Pending	NULL
111	2021-04-27	0	Online	Pending	NULL
150	2021-04-27	0	Online	Pending	NULL
158	2021-04-27	0	Online	Pending	NULL
NULL	NULL	NULL	NULL	NULL	NULL

