**Online Boating House Booking Portal**

**Overview:**

The Online Boat House Booking Platform serves as a digital gateway for users seeking unique and immersive accommodations on the water. This web-based service provides a user-friendly environment for discovering, selecting, and reserving boat houses tailored to individual preferences.

**Users of the System:**

Authentication and Authorization : Role based authentication

**Roles:**

1.               **Admin**

2.               **Customer**

**Admin should have permission for CRUD operation**

**Role based Menu Options**

**Admin:**

1.    Register

2.    Login

3.    Post Boat Details

4.    Edit Boat Details

5.    Delete Boat Details

6.    View All Boats

7.    View All Bookings

8.    Change Status of Booking

9.    View Payment History

10. Logout

**User**:

1. Register

2. Login

3. View All Boats

4. Add Booking

5. Delete Booking

6. Make the payment for Boat

7. View Payment History

8. Logout

**Functional Requirements:**

·      Build an application to book boats, speed boats or cruise ships.

·      Booking date should not be less than the current date.

·      The application should have signup, login, dashboard, booking page for user and login, dashboard for the admin.

·      This application should have a provision to maintain a database for customer information, Admin information, Booking information.

·      Also, an integrated platform required for customers and admin.

·      View booking module to include options for adding / modifying / removing the existing booking and customer management.

·      User’s allowed to Make payments online

**Non-Functional Requirements:**

1. **Security**: The system must implement robust security measures to protect user data, including user authentication, secure data storage, and encrypted data transmission.
2. **Scalability**:The system should be designed to handle an increasing number of bookings, payments, and users.
3. **Usability:** The user interface should be intuitive and user-friendly, with responsive design for mobile and desktop users.
4. **Availability:** The system should be available 24/7 with minimal downtime for maintenance.
5. **Logging and Auditing**: Support logging and auditing of system activities for monitoring and troubleshooting.

**Technology Stack**

**>Front End**

React,HTML,CSS

**> Back End**

Java, Spring boot, MySQL for database

**> Authentication**

JWT for User Authentication

**Application Flow:**

**User side:**

The application flow for the portal begins with user registration, where prospective users create accounts by providing personal information. Upon logging in, users access the user dashboard and view the available boats with payment details. The user can book the boat and will be able to make payments, as well as view the entire payment history.

**Create a data flow diagram if user has logged in.**

**Admin side:**

The administrative flow within the portal begins with administrators accessing the admin dashboard, providing a comprehensive overview of boats offered with payment details. The admin can view the list of users profiles.The admin can view the booking details submitted by the users. The admin can view the payment history.

**Modules of the Application:**

**ADMIN**:

Ø Register

Ø Login

Ø Dashboard

§ Boat House Page (Add, edit and delete)

§ Customer Booking details

§ Payment History Page

**Customer**:

Ø Register

Ø Login

Ø Dashboard

§ Add booking

§ View booking(edit , delete , make payment)

Ø Boat House Booking Page

Make payment

**Password Validations**:

1. Basic email validation should be performed.

 2. Basic mobile number validation should be performed.

 3. Basic password should be performed.

**Client-Side Validation:**

Implement client-side validation using HTML5 attributes and JavaScript to validate user input before making API requests.

Provide immediate feedback to users for invalid input, such as displaying error messages near the input fields.

**Server-Side Validation:**

Implement server-side validation in the controllers to ensure data integrity.

Validate user input and API responses to prevent unexpected or malicious data from affecting the application.

Return appropriate validation error messages to the user interface for any validation failures.

**Exception Handling**

Implement exception handling mechanisms in the controllers to gracefully handle errors and exceptions.

Define custom exception classes for different error scenarios, such as API communication errors or database errors.

Log exceptions for debugging purposes while presenting user-friendly error messages to users. Record all the exceptions and errors handled store in separate table “**ErrorLogs**”.

**Error Pages:**

Create custom error pages for different HTTP status codes (e.g., **404** Not Found, **500** Internal Server Error) to provide a consistent and user-friendly error experience.

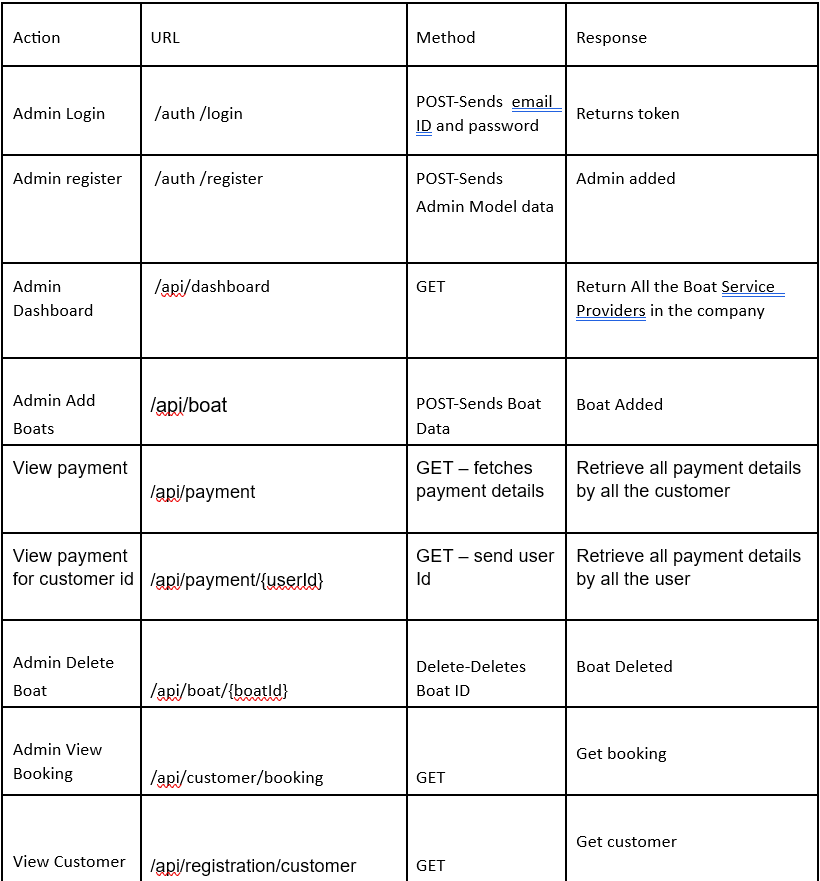
Ensure that error pages contain helpful information and guidance for users.

Thus, create a reliable and user-friendly web application that not only meets user expectations but also provides a robust and secure experience, even when faced with unexpected situations.

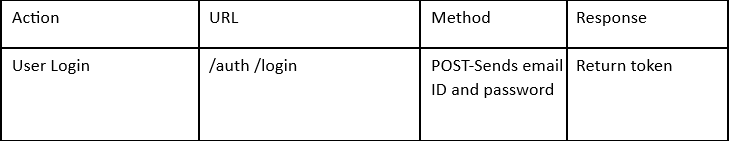
**Project Tasks:**

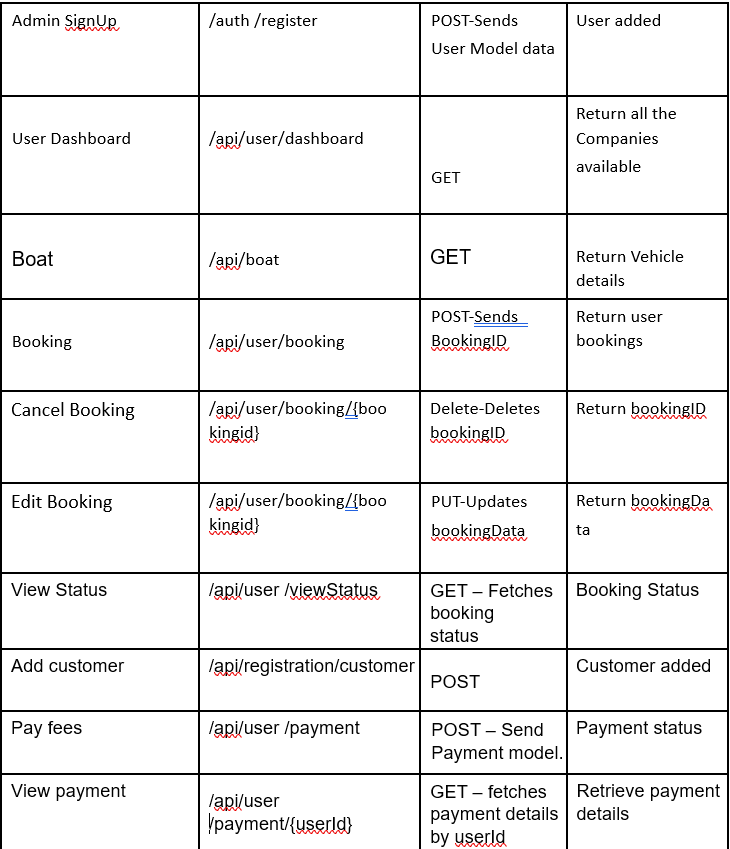
1. API Endpoints:

**Admin Side:**

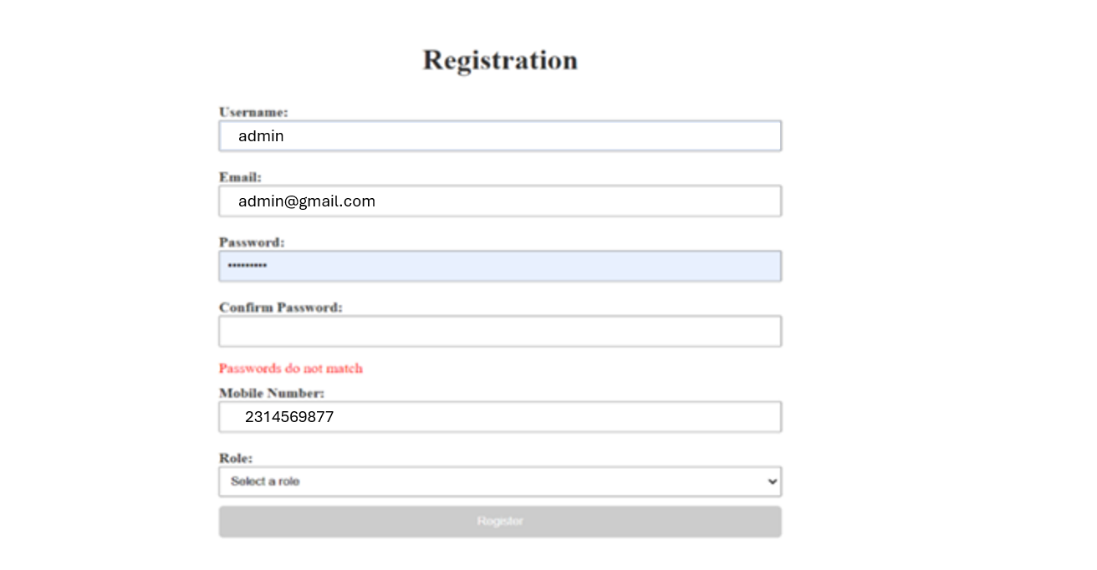


**User Side:**

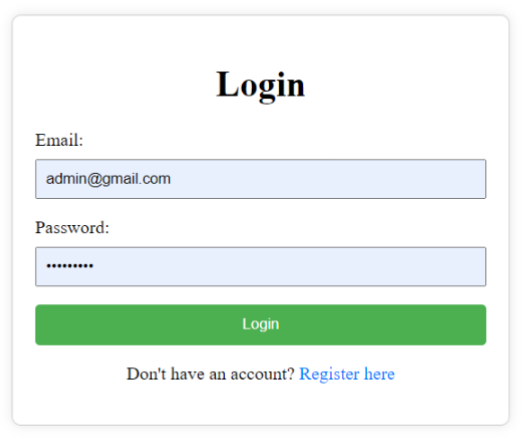
****



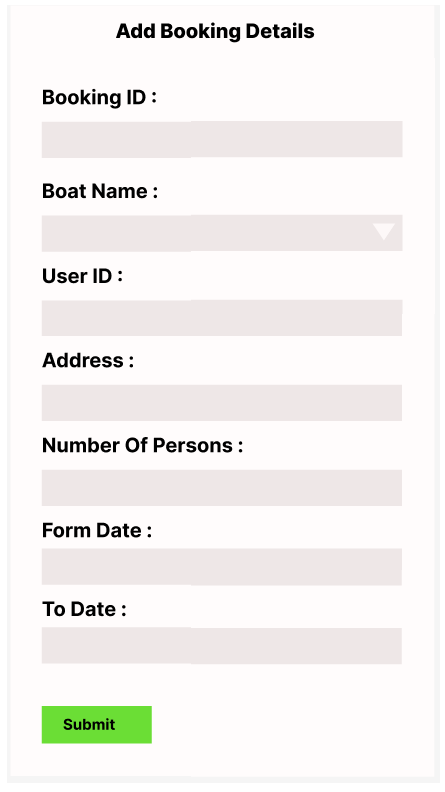
**Front end Screen Shots:**



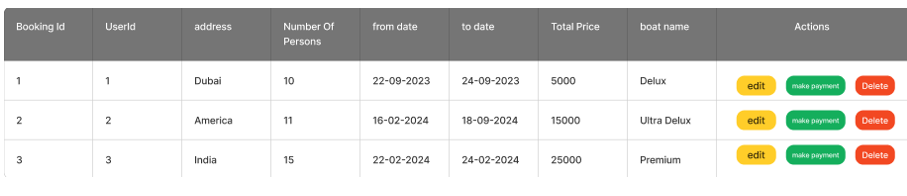
**Login**: Design a login page the existing customer can log in using the registered email id and password.



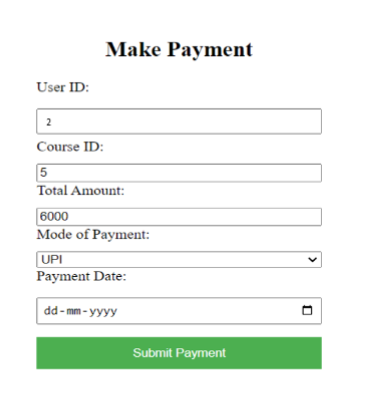
**AddBooking**: Design an AddBooking pagethat helps the users to view their booked boat details.



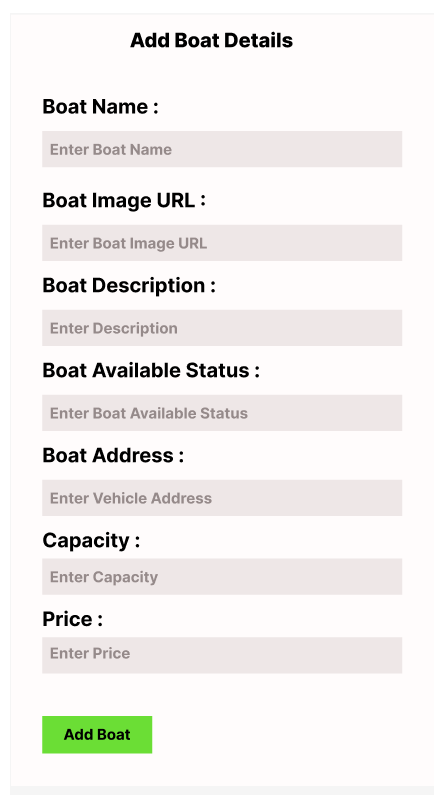
**User View Booking:** [view-booking]



**User can make payment**



**Admin Add Boat**: admin can add boat to the inventory.



**Admin View Boat**: Design an view boat component (Name the component as **view-boat** for angular app).



**Admin Views Payment History**



**Backend Requirements:**

**Model Class:**

**Entities:**

User

Customer

Boat

Booking

Passengers

Payment

**User:**

{

email: String

userId: Long

password: String

username: String

mobileNumber: String

userRole: String (ADMIN/CUSTOMER)

}

**class Customer**

{

customerId : Long

customerName : String

email : String

mobileNumber : Long

@OneToMany

bookings : List<Booking>

@OneToMany

payments : List<Payment>

@OneToOne

user : User

@ManytoMany

boats : List<Boat>

}

**class Boat**

{

boatid : Long

boatName : String

boatImageurl : String

boatLocation : String

boatCategory : String

boatAvailable status : String

price : Long

capacity : Integer

@OnetoMany

bookings : List<Booking>

@ManytoMany

customers : List<Customer>

}

**class Booking**

{

bookingId : Long

noOfPersons : Integer

fromDate : Date

toDate : Date

totalPrice : Double

@ManyToOne

customer : Customer

@ManyToOne

boat : Boat

@OnetoMany

passengers : List<Passenger>

@OnetoOne

payment : Payment

}

**class Passenger**

{

passengerId : Long

firstname : String

lastname : String

age : Integer

gender : String

@ManyToOne

booking : Booking

@ManyToOne

customer : Customer

}

**class Payment**

{

paymentId : Long

status : String

amountPaid : Double

@OneToOne

booking : Booking

@ManyToOne

customer : Customer

}

**Platform Prerequisites (Do’s and Don’ts):**

Ø The react app should run in port 8081.

Ø The spring boot app should run in port 8080.

**Other Important Key factors in the application:**

• Should use Custom Exceptions mandatory.

• Tables should have proper relationship and keys

• Frontend Application should be menu driven.

• Proper Menu / Navigation for corresponding role

• Client side validations and server side validations are mandatory

• Error should be handled.

• Follow best programmer practice while developing

• Provide proper Naming Conventions

**Key points to remember:**

1.    The id (for frontend) and attributes(backend) mentioned in the SRS should not be modified at any cost. Failing to do may fail test cases.

2.    Remember to check the screenshots provided with the SRS.

3.    Strictly adhere to the proper project scaffolding (Folder structure), coding conventions, method definitions and return types.

4.    Adhere strictly to the endpoints mentioned in API endpoints section.

5.    Don't delete any files in a project environment.

**How To Run the Project**

**API endpoint:**

8080

**Platform Guidelines:**

To run the command use **Terminal**in the platform.

**Spring Boot:**

Navigate to the springapp directory => **cd springapp**

To start/run the application '**mvn spring-boot:run**'

To Connect Database

Cmd: **mysql -u root –protocol=tcp -p**

Password: examly

**FRONTEND:**

**Step 1:**

Open the terminal

Use “nvm use 14” command to change node version to 14

**Step 1:**

Use "cd reactapp" command to go inside the reactapp folder

Install Node Modules **- "**npm install**"**

**Step 2:**

Write the code inside src folder

Create the necessary components

**Step 3:**

Click the run test case button to run the test cases

**Note :**

* Click PORT 8081 to view the result / output
* If any error persists while running the app , delete the node modules and reinstall them