**Party Event Management**

**Problem Statement:**

The process of planning and organizing events, especially parties and gatherings, often involves navigating through a multitude of service providers, each offering different aspects of event management. Current systems may lack a centralized and user-friendly platform that effectively connects party USERs with potential clients, resulting in inefficiencies and a fragmented user experience.

Clients seeking event services face challenges in finding relevant and available options, and the absence of a streamlined booking process can lead to confusion and delays. Party USERs, on the other hand, may struggle to efficiently showcase their services, manage bookings, and communicate effectively with clients using existing tools.

The Party USER System aims to address these challenges by providing a comprehensive and user-friendly solution. Party USERs can list their services, showcasing event types, venue options, catering services, entertainment, decorations, and pricing. Clients, in turn, benefit from a straightforward event search feature, allowing them to find and book relevant event services based on specific criteria.

**Users of the System:**

Authentication and Authorization : Role based authentication.

**Role:**

1.   ADMIN

2.   CUSTOMER

Admin should have permission for CRUD operation.

**Role based Menu Options**

**ADMIN**

* Register
* Login
* Post Event details
* Edit Event Details
* Delete Event Details
* View All Events
* Post Venue Details
* Edit Venue Details
* Delete Venue Details
* View All Bookings
* Change the status of Booking
* View Payment History
* Logout

**USER**

* Register
* Login
* View All Events
* View all Venues
* Book for the event by checking the venue availability
* Book the event date and pay amount to book the venue
* Cancel the event should reflect the venue availability for the date
* Check the Status of Booking
* Delete Booking
* Logout

**Functional Requirements:**

**Service Listings for Party USERs**: Party USERs can list their services, including event types (e.g., birthday parties, weddings, get-together, function celebration, product launch), venue options, catering services, entertainment, decorations, and pricing.

**User Authentication**: Users are required to create accounts or log in before booking services or listing their services. This ensures data security and user accountability.

**Event Search**: Search for available event services based on criteria such as event type, location, availability, and price. This simplifies the process of finding relevant event services.

**Event Booking**: Book event services for their parties and gatherings. They need to select the service, choose an event date, check the availability of venue, specify the number of attendees, and provide special requests or preferences.

**Booking Confirmation**: Party USERs are notified of new bookings. Need to do pay the total amount for verification process

**USER Dashboard:** Party USERs have access to a dashboard for managing their listed event services, viewing bookings. Also can view their booked event services, check the booking status, and communicate.

**Real-time Availability**: Party USERs can update the availability of their services in real-time, ensuring that clients are booking available slots.

While the above ones are the basic functional features expected, the below ones can be nice to have add-on features:

* Have appropriate filters for search.
* Email integration for intimate USERs.
* Multi-factor authentication for the sign-in process

**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 event listings, venues, 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.

**Application Flow:**

ADMIN side:

The administrative flow within the portal begins with administrators accessing the admin dashboard, providing a comprehensive overview of events and its package description. The view of listing the limited number of venues and the availability for the selected date. The admin can edit /delete the venue details.

The admin can view all the bookings for the events with venue and date. The admin should validate the booking and change the status from pending to verified, booked, rejected, etc. The admin can view all the payment history.

USER Side:

The application flow for the portal begins with user registration, where prospective students create accounts by providing personal information. Upon logging in, users access the USER dashboard and view the available of events packages and descriptions. The USER view all the venues and check the availability of venue for the selected date.

Once the venue is available for the event date, book for the event. By default all the booking status will be pending status. Once the booking status changed to verified, then the USER need to pay the 20% of the amount as advance.

After the advance paid the status will be changed as booked. After the balance amount paid the booking status will be changed to Confirmed.

The USER can also cancel the event by eleasing the venue and then from the amount paid 40% of cancellation charges will be deducted.

**Abstract**:

The Party USER System is an innovative web-based application designed to revolutionize the event planning experience, catering to a diverse range of celebrations, from birthdays to weddings and corporate events. This comprehensive platform acts as a central hub for party USERs to showcase their services and for clients to seamlessly browse, book, and manage event services. With a focus on user-friendly interactions and efficient event management, this system aims to enhance the overall experience for both USERs and clients in the dynamic world of event planning.

**Mandatory Modules**

**ADMIN:**

* Register
* Login
* Dashboard
* Events List
* Venues List
* Bookings List
* Payment details

**USER**

* Register
* Login
* Dashboard
  + Events List
  + Check Venue availability
  + Book for the event date with venue

**Technology Stack**

Front End

React, HTML, CSS

Back End

Java, Spring boot and MySQL for database

**Application assumptions**:

1.    The login page should be the first page rendered when the application loads.

2.    Logging out must again redirect to the login page.

3. Design forgot password and forgot email buttons in login page.

**Validation**

**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.

1. Basic email validation should be performed.

      2. Basic mobile number validation should be performed.

      3. Basic password should be performed

**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.

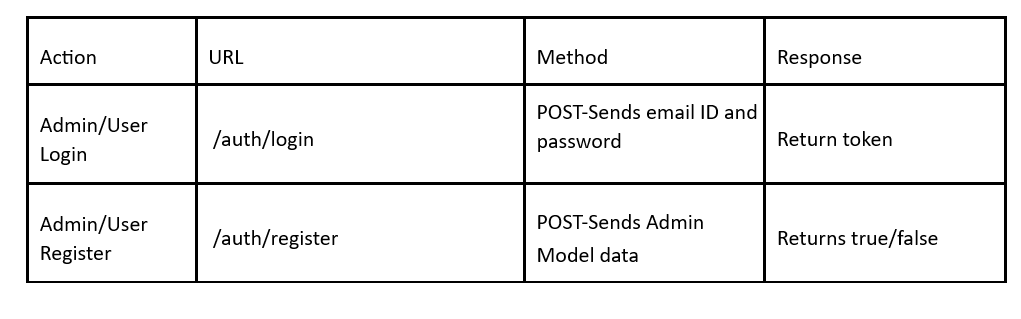
Define custom exception classes for different error scenarios, such as API communication errors or database errors.

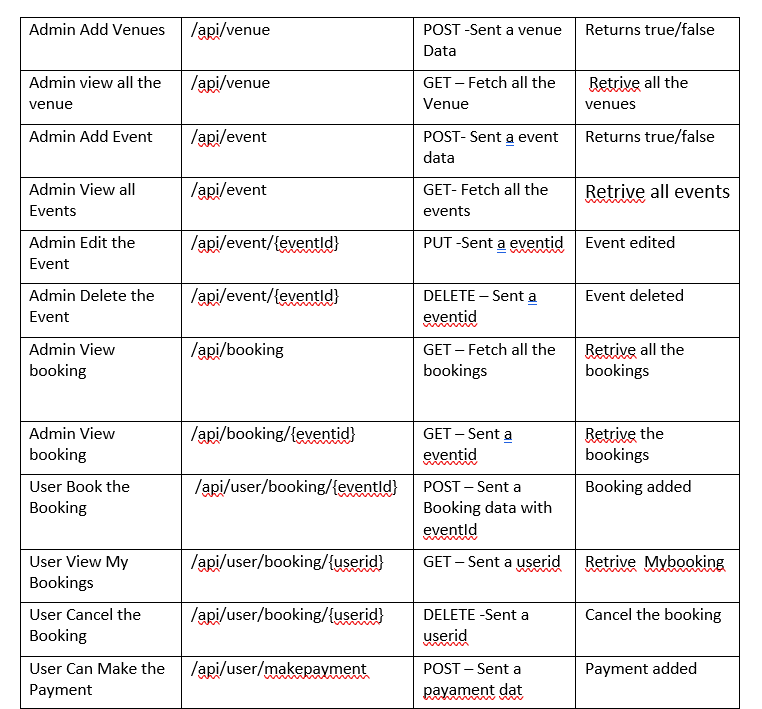
Log exceptions for debugging purposes while presenting user-friendly error messages to users. Record all the exceptions and errors handled store in separate table “**ErrorLogs**”.

**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.  
  
**API Endpoints**:  
  






**class User** {

private Long userId;

  private String email;

  private String password;

  private String username;

  private String userRole;

@OneToOne

  private Customer customer;

}

**class Customer**{

  private Long customerId;

private String customerName;

  private String address;

  @OneToMany

  private List<Booking> bookings;

  @ManyToMany

  private List<Event> events;

  @OneToOne

  private User user;

}

**class Event** {

   private Integer eventId;

private String eventType;

   private String description;

   String eventPackage;

   String maxParticipantsCount;

   private Integer charges;

   @ManyToMany

   private List<Customer> customers;

   @OneToMany

private List<Booking> bookings;

}

**public class** **Venue** {

private Integer venueId;

private String venueName;

private String venueLocation;

@OneToMany

@JoinColumn(name = "venue\_id")

private List<Booking> bookings;

}

**public class Booking**

{

           Integer bookingID

           DateTime submissionDate

           DateTime eventDate

           Integer bookingStatus

           Integer headcount

String eventType

           @ManyToOne

           Customer customer

@ManyToOne

           Venue venue

           @ManyToOne

           Event event

           @OneToOne

           Payment payment

}

**class Payment**

{

           Long paymentId

           String status

           Double totalAmount

           Date paymentDate

           String modeOfPayment

           @OneToOne   // Optional...only for reporting

           Customer customer

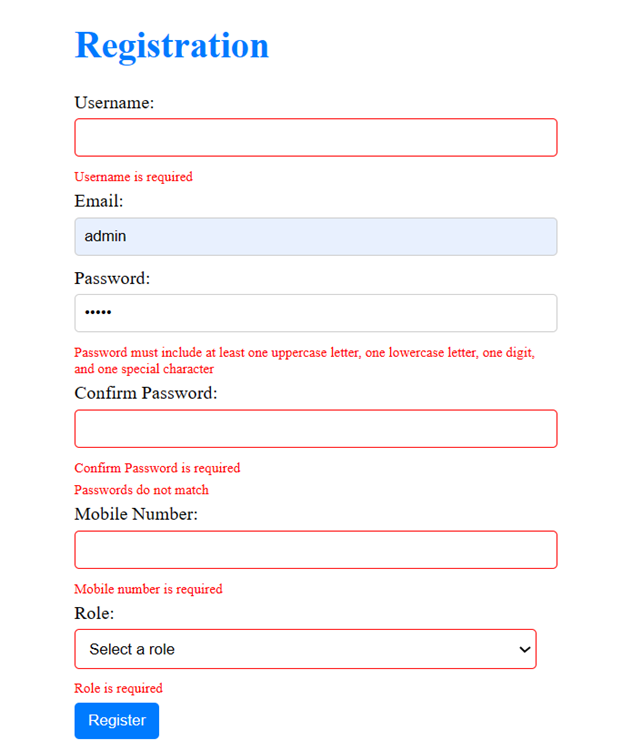
}

**Note**:

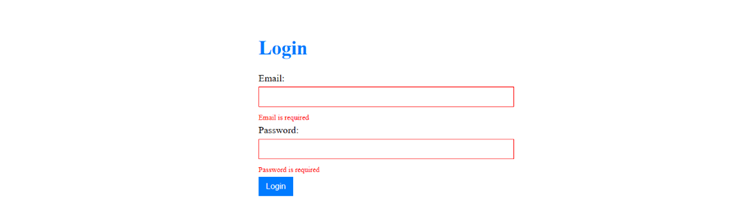
·      Create a folder of components inside the app to store all the components.

**Frontend Sample Screenshots: ADMIN SIDE:** 

Landing page :   
Registration page with validation



Login page with validation



On successful login , Based on role nav bar items will be shown accordingly  
  
Admin Side :

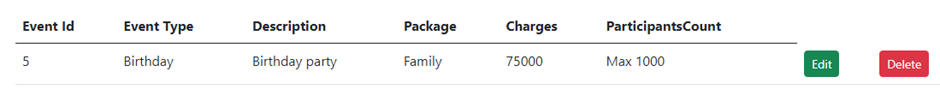


User Side :  
  


Admin add the event details:

A white lined paper with blue lines

Description automatically generated

Admin can view all events

On clicking the edit and delete , respective operation will done for that event.

The form to book for the event.

A white paper with blue lines

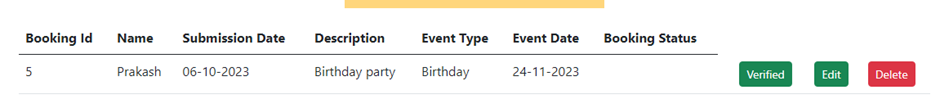
Description automatically generated

Name should be stored once the user is logged in successfully and it should be prepopulated in the input field “Name” as in the above screenshot.  
  
A blue line with white lines

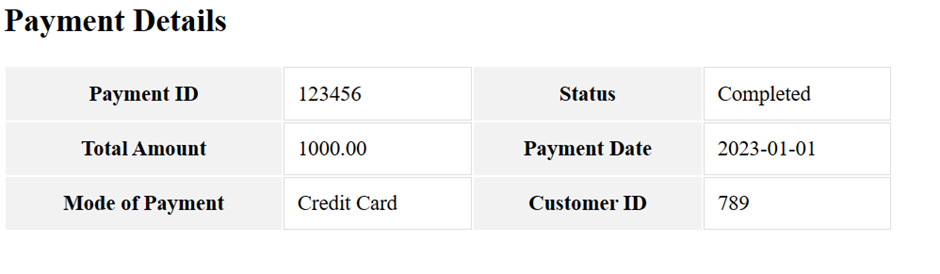
Description automatically generated

List of all event type (Added in admin side) is listed in dropdown , as shown in above screenshot provided

The Admin can change the Booking status from Pending to Verified, Booked, Confirmed.

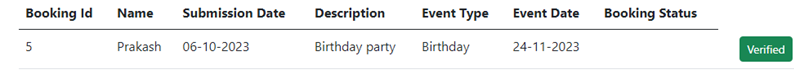


Admin can view the payment history of customers payment



**USER Side:**

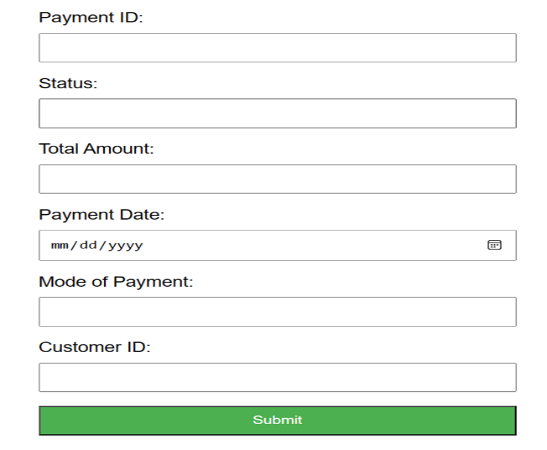
The USER can check the status of the bookings



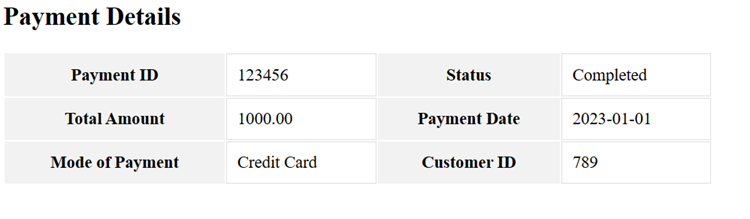
The USER can book for the event and the date. When he tries to book the same venue in the same date

OOPS ! Sorry this venue is not available

The USER can pay for the booking



USER can view the payment history



**Note:**

* You should use NotFound(), NoContent(), BadRequest(), CreatedAtAction() to handle the HTTP status code as return values for the Controller methods as mentioned.
* Don't delete any files in a project environment.

**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
* Error should be handled
* Follow best programmer practice while developing
* Provide proper Naming Conventions

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

* The react app should run in port 8081.
* The springapp should run in port 8080.

**HOW TO RUN THE PROJECT :**

**FRONTEND:**

**Step 1:**

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

**BACKEND:**

**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**'

Click on the Run Test Case button to pass all the test cases

**To Connect Database open terminal**

Cmd: mysql -u root –protocol=tcp -p

Password: examly