**Corporate Events Management Project**

**Problem Statement:**

Planning and managing corporate events can be a complex and time-consuming task, often fraught with challenges such as inefficient communication, limited access to diverse service offerings, and difficulties in coordinating event details. Event planners face the challenge of reaching potential clients and showcasing their services effectively, while corporate clients struggle to find suitable event services tailored to their specific needs.

Existing systems often lack a centralized platform that efficiently connects event planners and corporate clients, resulting in a fragmented and less-than-optimal experience for both parties. The absence of a streamlined process for booking event services, along with the complexities of managing and confirming bookings, further exacerbates these challenges.

**Users of the System:**

**Authentication and Authorization: Role based authentication.**

**Role:**

·      Admin

·      Organizer

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

·      View All Bookings

·      View All Payments

·      Logout

**Organizer**:

·      Register

·      Login

·      View All Events

·      Book for the event

·      Check the Status of Booking

·      Once the status of Booking is verified, the advance payment need to be paid.

·      Once the Booking confirmed the full payment need to be paid.

·      Delete Booking

·      Once the Booking confirmed same date same event cannot be booked

·      One day only one event can be booked by one organizer.

·      View all payment history

·      Logout

**Functional Requirements:**

·      **Service Listings for Event Planners**: Event planners can list their services, including event type (e.g., conferences, seminars, team-building events), venue options, catering services, audiovisual equipment, 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 for Corporate Clients**: Corporate clients can 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**: Corporate clients can book event services for their corporate events. They need to select the service, choose an event date, specify the number of attendees, and provide special requests or preferences.

·      **Booking Confirmation**: Corporate clients receive the confirmation for their booked event services. Event planners are also notified of new bookings.

·      **Event Planner Dashboard**: Event planners have access to a dashboard for managing their listed event services, viewing bookings, and handling client communications.

·      Payment details, including total cost, breakdown of charges, and payment confirmation, are integrated into the system, ensuring a smooth and transparent financial transaction for both parties.

·      Once the booking is verified, the 40% advance payment should be paid. Once the payment done and when the status change to Booked, the remaining 60% payment should be paid by the organizer.

·      Same date same event should not be booked by any other organizer.

·      The organizer check the status of the event booking.

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 the organizers to get the booking status.

·      Multi-factor authentication for the sign-in process

**Non-Functional Requirements:**

·      **Security**: The system must implement robust security measures to protect user data, including user authentication, secure data storage, and encrypted data transmission

·      **Scalability**: The system should be designed to handle an increasing number of events listings, bookings, payments, and users.

·      **Usability**: The user interface should be intuitive and user-friendly, with responsive design for mobile and desktop users.

·      **Availability**: The system should be available 24/7 with minimal downtime for maintenance.

·      **Logging and Auditing**: Support logging and auditing of system activities for monitoring and troubleshooting.

**Application Flow:**

**Organizer side:**

The application flow for the portal begins with user registration, where prospective organizers create accounts by providing personal information. Upon logging in, users access the organizer dashboard and view the events available with payment description.

The organizer will book for an event. The organizer booking status will be pending status. Once verified by the admin then the organizer need to pay the event amount.

Once the payment completed the booking status will be changed to verify to booked status

The organiser will do the payment and view all the payment history.

**Admin side:**

The administrative flow within the portal begins with administrators accessing the admin dashboard, providing a comprehensive overview of events details and booking list. The admin can view all the payment history.

The admin can add, edit and delete the events descriptions. The admin can view the list of booking submitted by organizer and change the status of the booking such as Pending, Verified, Booked, Confirmed, Rejected etc. The admin should not confirm the same date same event. The admin should get the error message that the same event is booked for same date.

The admin can view the list of payments.

**Application assumptions**:

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

·      Unless logged into the system, the user cannot navigate to any other pages.

·      Logging out must again redirect to the login page.

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

**Technology Stack**

**Front End**

React, HTML, CSS

**Back End**

Java, Spring Boot and MySQL for database

**Authentication**

JWT for User Authentication

**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 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 a

lts provides a robust and secure experience, even when faced with unexpected situations.

**Admin Side:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Action** | **URL** | **Method** | **Response** |
| User Register  (Both the Roles) | /auth/register | POST -Sends Admin Model data | Returns true/false |
| User Login (Both the Roles) | /auth/login | POST -Sends Email Id and Password | Return token userId,userRole and username |
| Add Event | /api/event | POST-Sends Event model data | Event Added |
| View Events | /api/events | GET -Fetches event data | Retrieve all events |
| View Events by Type | /api/admin/event/{eventtype} | GET -Fetches event data | Retrieve all events |
| Edit Event | /api/admin/update-event/{eventId} | PUT -Sends eventid | Events are updated |
| Add Facility | /api/facility | POST -Sends facility Model data | Facility added |
| View Facility | /api/facility | GET – Fetches Facility Data | Retrieve all facilities |
| Edit Booking status | /api/admin/booking/  changestatus/{bookingId} | PUT – Sends bookingid | Booking status updated |
| View Organizers | /api/registration/organizer | GET – Fetches Organizer Data | Retrieve all Organizer |
| View Booking by id | /api/admin/booking/{bookingId} | GET – Fetches Organizer Data | Retrieve the Data booking by id |

**Student Side**

|  |  |  |  |
| --- | --- | --- | --- |
| **Action** | **URL** | **Method** | **Response** |
| Add Booking | api/user/booking | POST – Sends Booking model data | Booking Added |
| View Events | /api/events | GET – Fetch Event Data | Retrive all Events |
| Add Organizer | /api/registration/organizer | POST- Sends Organizer data model | Organizer Added |
| Add Payment | /api/user/payments | POST – Sends Payment data model | Payment Status |
| View Payment | /api/user/payment | GET – Fetches Payment | Payment Details |
| View Booking by id | /api/user/booking/{bookingid} | GET – Fetches Booking Data | Retrive the Booking by id |
| Edit Booking | /api/user/booking/{bookingid} | PUT – Sends Booking id | Booking are Updated |

**Backend**

Create Model class in Springboot with the following datatypes.

Primary Model

**User**:

This class stores the user type (admin or the customer) and all user information.

class User

{

 Long userId;

 String userName;

 String email;

 String password;

 String userRole;

}

class Booking

{

Long bookingId;

Date submissionDate;

String description;

Date eventDate;

boolean bookingStatus;

Integer headcount;

Double amount;

@ManyToOne

private Event event;

@OneToOne

private Payment payment;

@ManyToOne

private Organizer organizer;

}

public class Event

{

Long eventId;

String eventType;

String description;

@OneToMany

List<Facility> facilities;

Integer participantsCount;

Integer eventCharges;

@OneToMany

List<Booking> bookings

@ManyToMany

List<Organizer> organizers;

}

public class Facility

{

Long facilityId;

String facilityDescription;

Integer price;

}

public class Organizer

{

Long organizerId

Integer mobileNumber;

@OneToOne

User user;

 @OneToMany

 List<Booking> bookings;

@ManyToMany

List<Event> events;

}

**Payment**:

This class stores the details of the payment.

public class Payment

{

          Long paymentId;

          Double amount;

          Date paymentDate;

          String modeOfPayement;

           @OneToOne

          Booking booking;

}

**Note:**

•            You should use NotFound(), NoContent(), BadRequest(), CreatedAtAction() to handle the HTTP status code as return values for the Controller methods as mentioned.

•            Use swagger/Index to view the API output screen in 8080 port.

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

**Note:**

·      Declare a public property **apiUrl** to store the backend url. (apiUrl – case-sensitive)

**·**Import model files, services and components as required.

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

·      Create a folder models inside app to store all the model interface.

**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

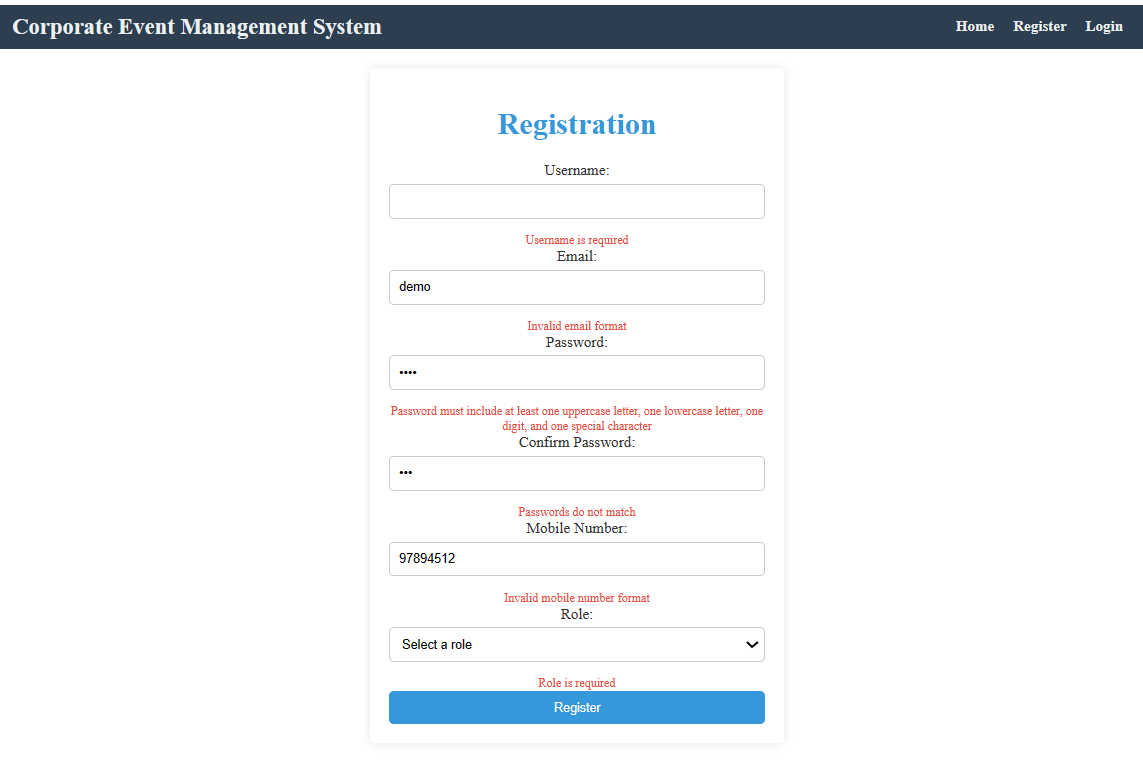
•            Provide proper Naming Conventions

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

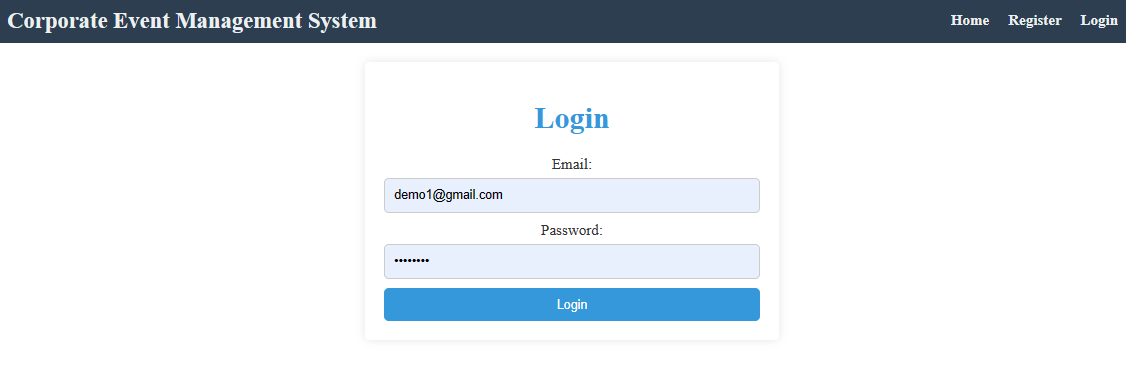
·      The react app should run in port 8081.

·      The spring app should run in port 8080.

Registration Page: The Users can register their details.



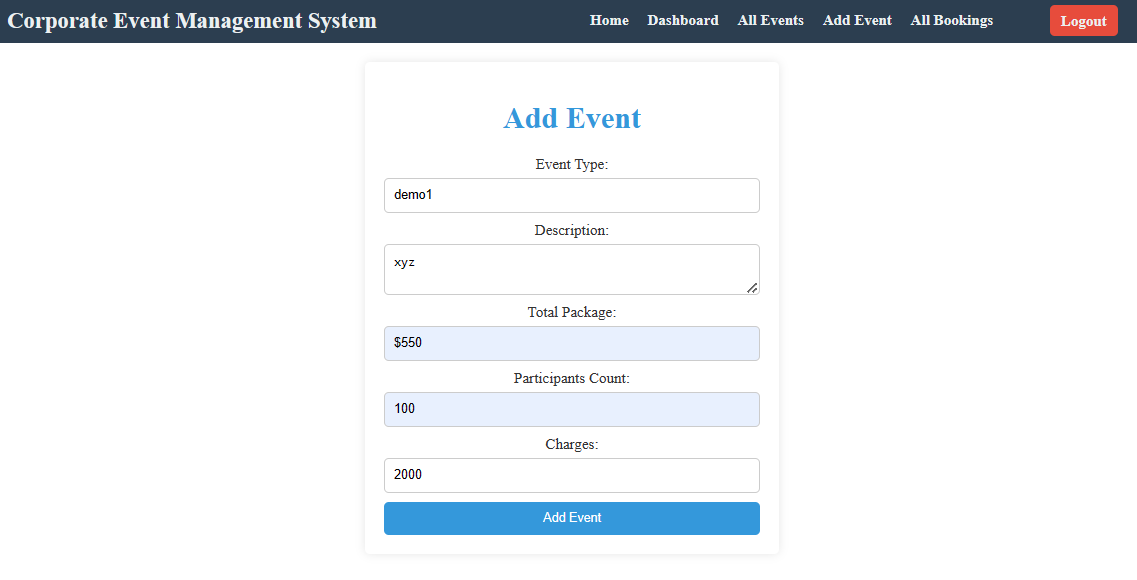
2.    Login Page:



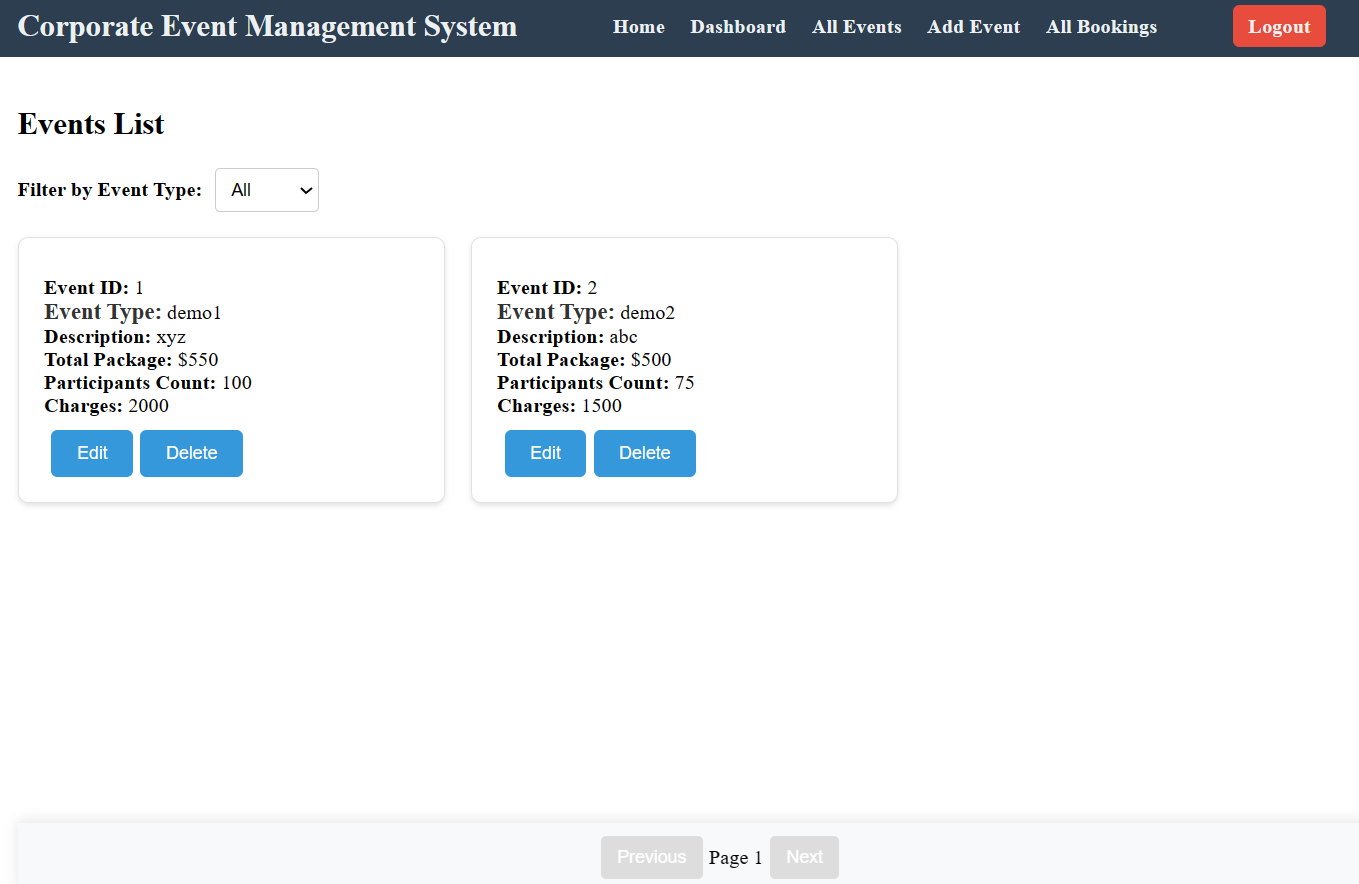
**3.    Admin NavBar:**



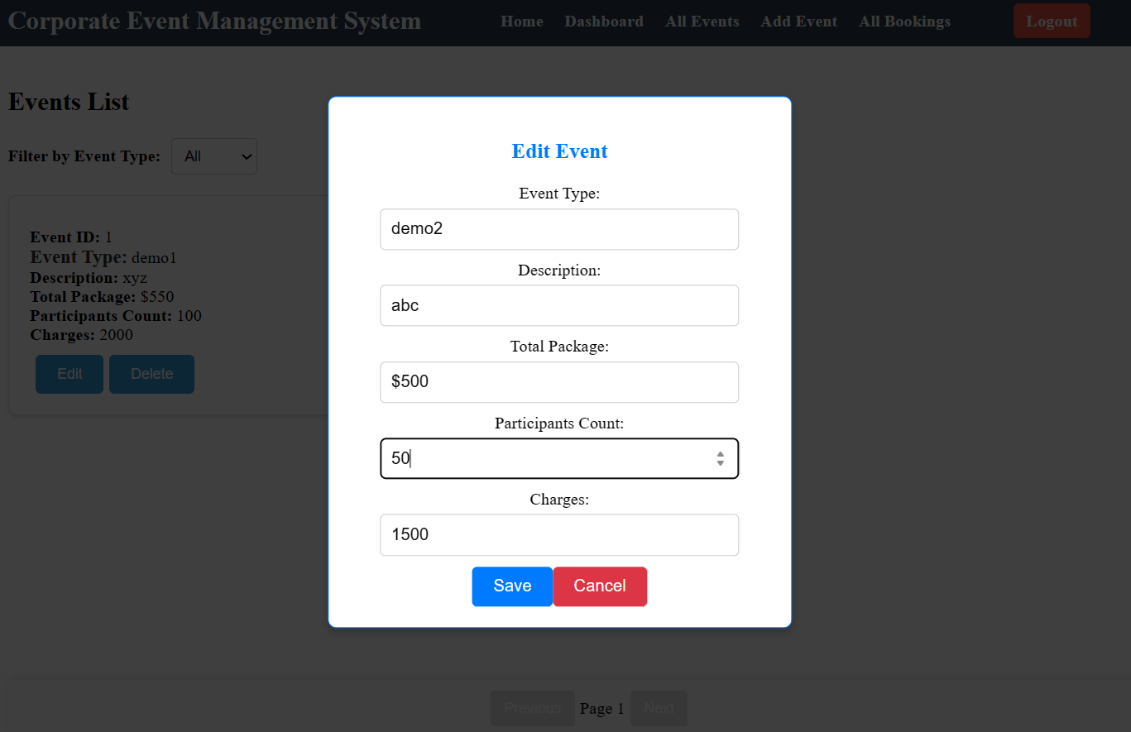
4.    Admin Side – The Admin can Add New Event



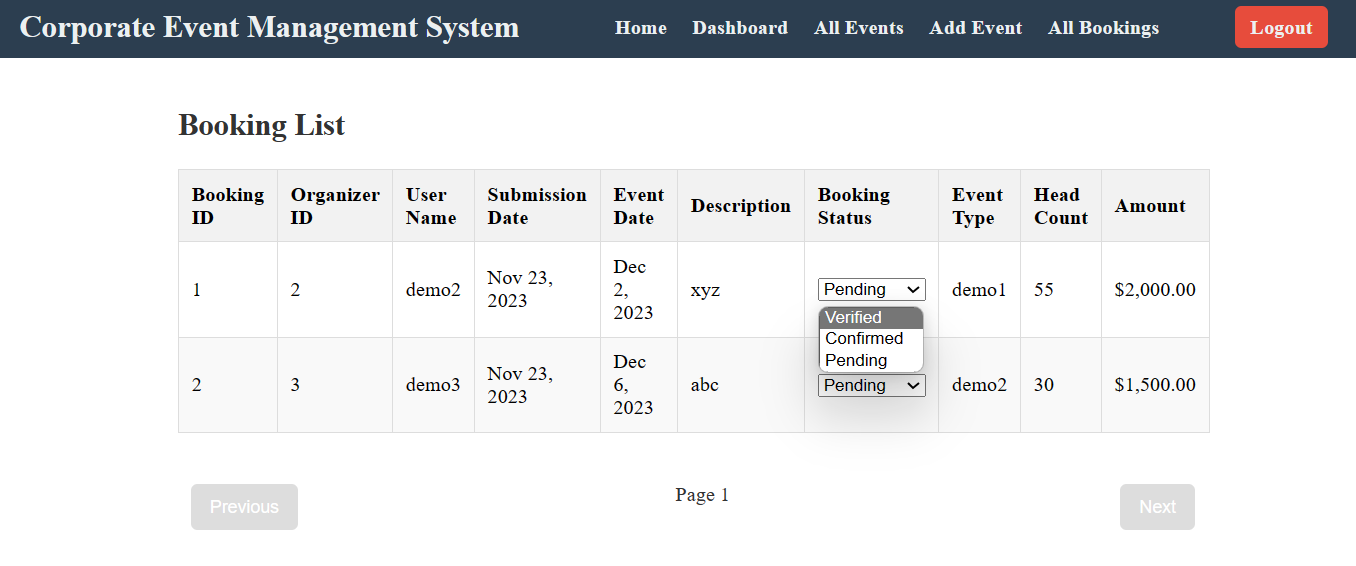
Admin Side – Can view all Events with filter by Events type along with Edit & delete actions



6.    Admin Side – Can Edit Event



7. Admin side – Can View All Bookings & can change the Booking status from Pending to Verified, Booked, Confirmed.

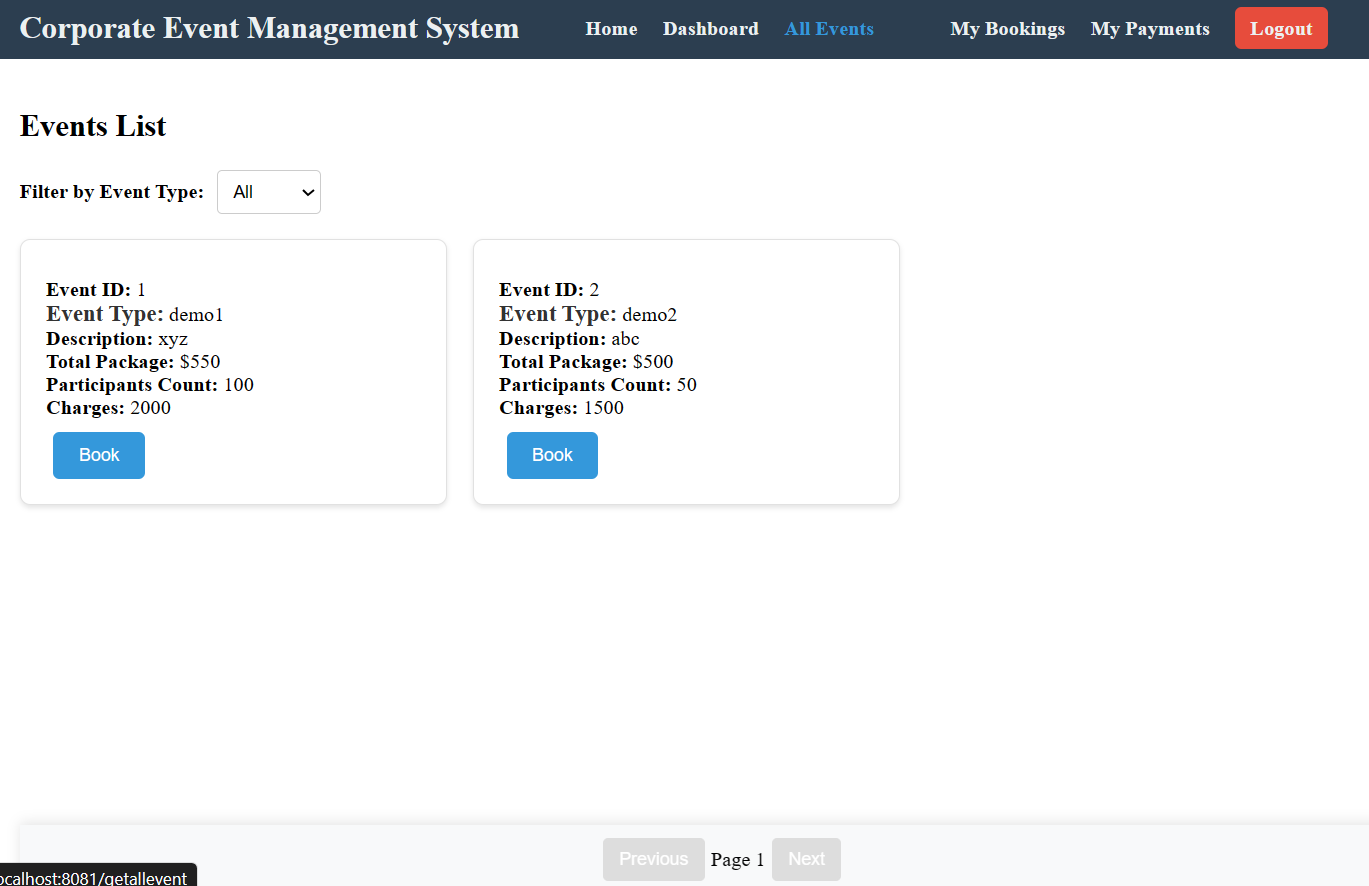


**ORGANIZER Side:**

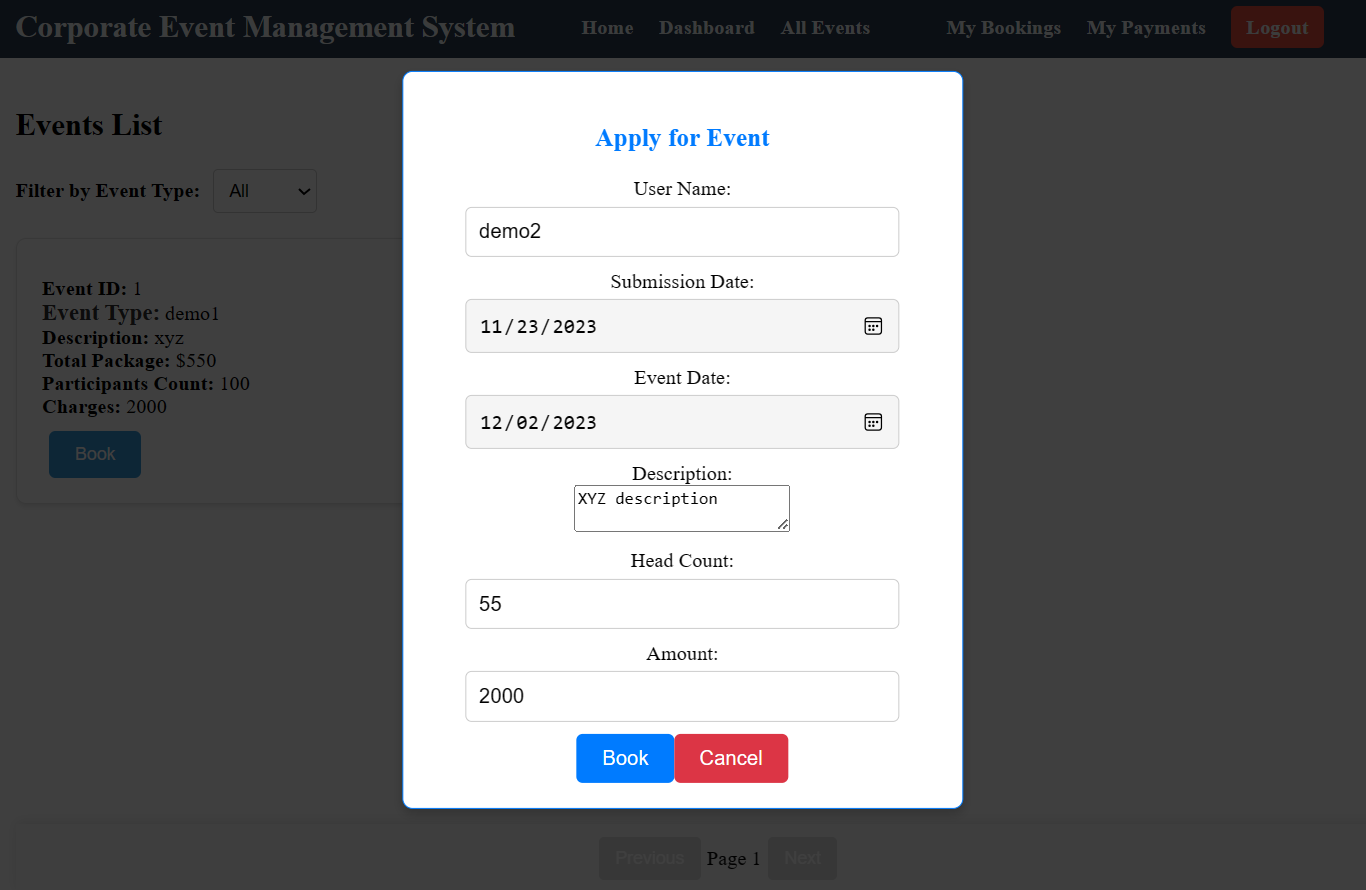
**8. Organizer NavBar:**



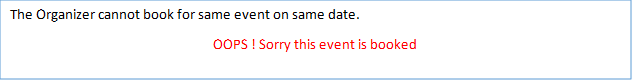
**9.    Organizer Side – Can View all the Available Events with filter by event Type:**

****

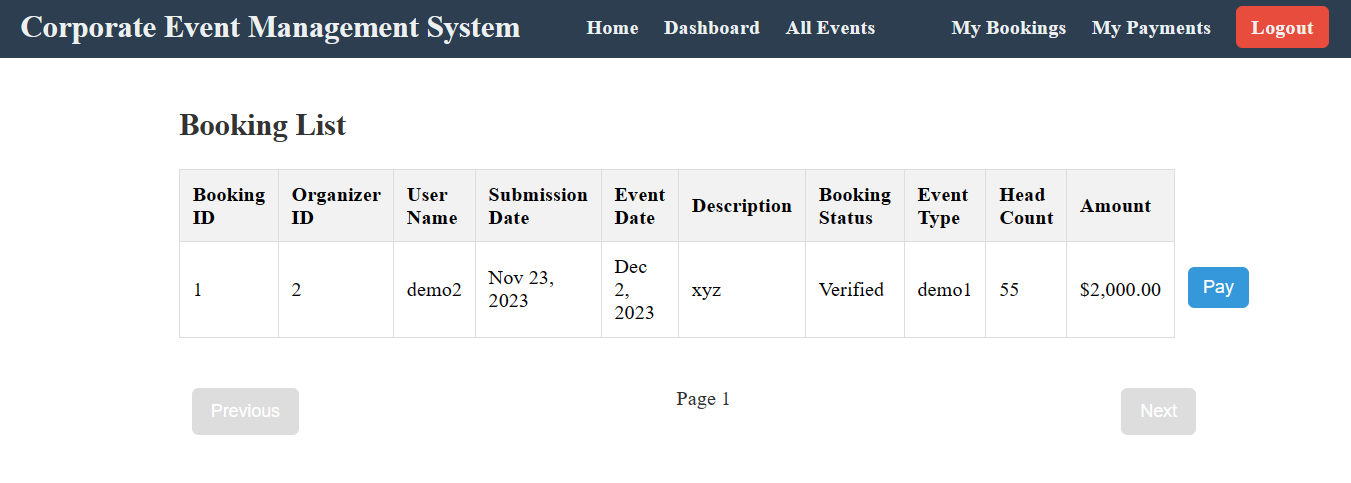
10. Organizer Side – Event Booking Form:



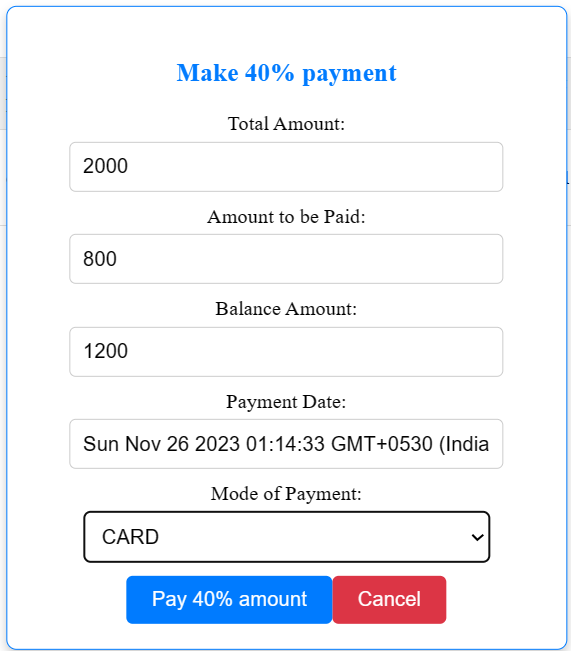
11. The organizer can book for the event and the date. When he tries to book the same event in same date



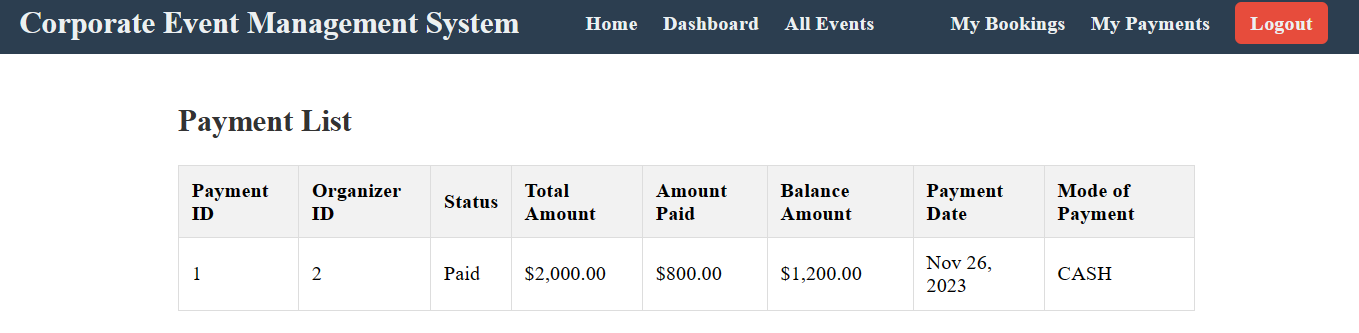
**12. The organizer can check the status of the bookings (if the Booking Status is verified by the ADMIN then PAY button should be poped up)**



13.**The Organizer can pay for the booking**



14. Organizer can view their own payment history

  
  
  
**HOW TO RUN THE PROJECT :**

**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.

**Spring Boot:**

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

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

**To Connect Database open terminal**

Cmd: mysql -u root –protocol=tcp -p

Password: examly