SOFTWARE ENGINEERING

Inizio
Event management system

SOFTWARE DESIGN SPECIFICATION

Group Members:
Riya Agarwal(IFI2022011)
Anamika Garg(IFI2022005)
Avantika Soni(IIB2022045)
Simaran Maurya(IIB2022021)
Shreya Sinha(IIB2022034)
Tanisha Prakash(IIB2022018)

Software Design Specification

1.1 Purpose of this Document

This Software Design Document delineates the design aspects of the "Inizio" event management system,explaining the system's expected inputs, outputs, classes, and functions. It further explains the interactions between classes to fulfill the specified requirements.

1.2 Scope of the Development Project

In Scope:
Comprehensive event planning assistance.
Audience engagement and networking.
Troubleshooting and assistance.
Networking opportunities.
Ticketing services.
Out of Scope:
Non-event related services.
Highly specialized or industry-specific events.

1.3 Definitions, Acronyms, and Abbreviations

IEEE: Institute of Electrical and Electronics Engineers
SRS: Software Requirements Specification

1.4 References

R. S. Pressman, Software Engineering: A Practitioner’s Approach, 5th Ed, McGraw-Hill, 2001.
IEEE SRS template

1.5 Overview of Document

This SDD is divided into several sections:
1.Introduction
2.Conceptual Architecture/Architecture Diagram
3.Logical Architecture
4.Execution Architecture
5.Design Decisions and Trade-offs
6.Pseudocode for Components
7.Appendices

2. Conceptual Architecture/Architecture Diagram: This section will provide an overview of the system's components, modules, structure, relationships, and user interface issues.

3. Logical Architecture: This section will describe the logical architecture of the system, including its description and components.

4. Execution Architecture: This section will define the runtime environment, processes, and deployment view of the system**.**

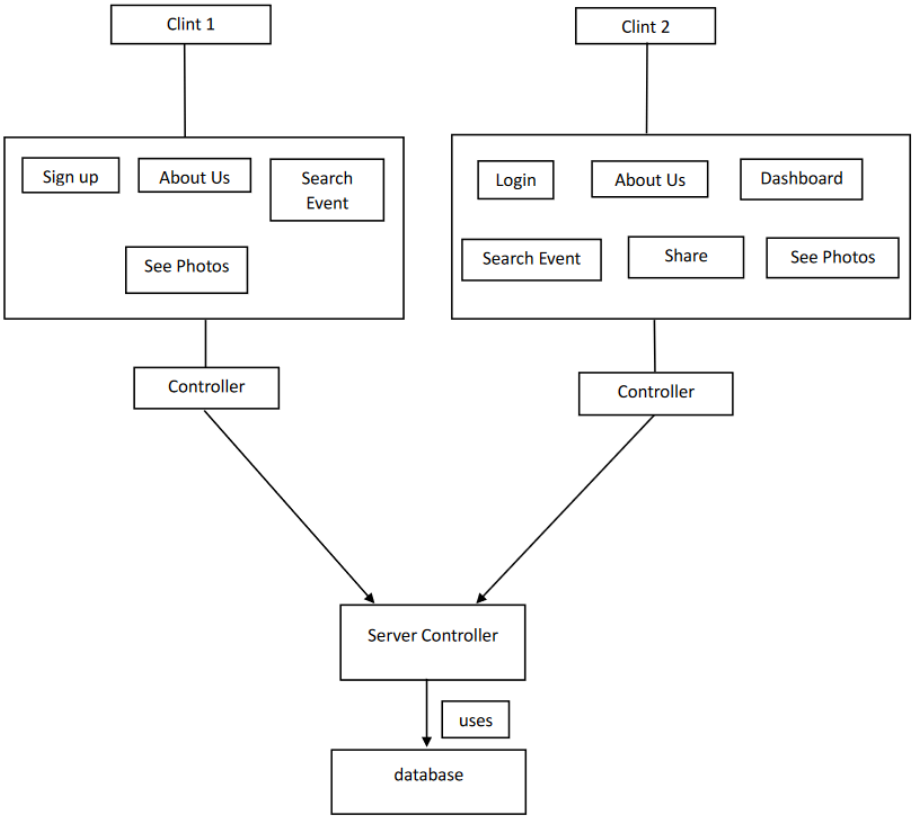
5. Design Decisions and Trade-offs: This section will elaborate on the decisions made during the design process and justify why they were chosen over other alternatives.

6. Pseudocode for Components: This section will provide pseudocode for various components of the system, aiding in understanding their functionality.

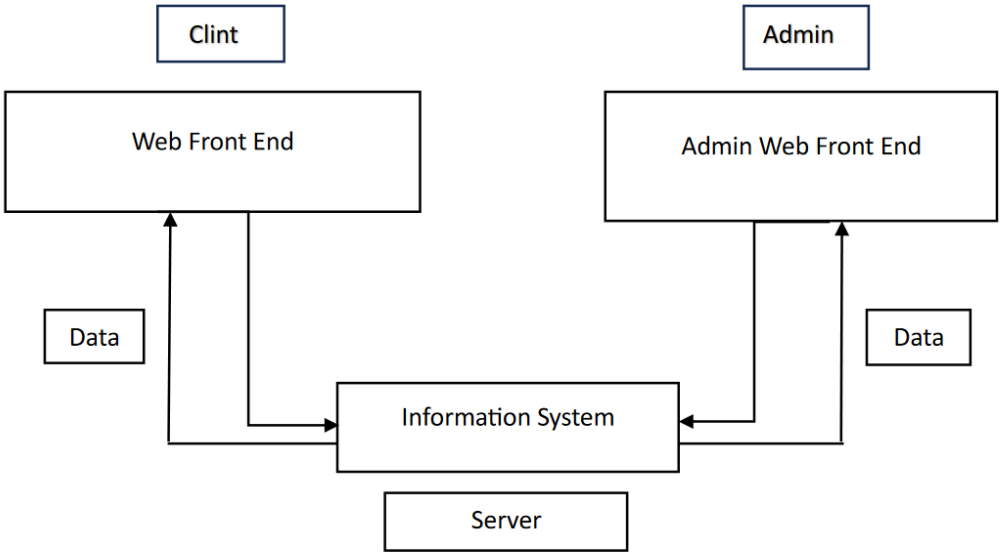
7. Appendices: This section will contain any supplementary material deemed necessary for the document.

2. Conceptual Architecture/Architecture Diagram

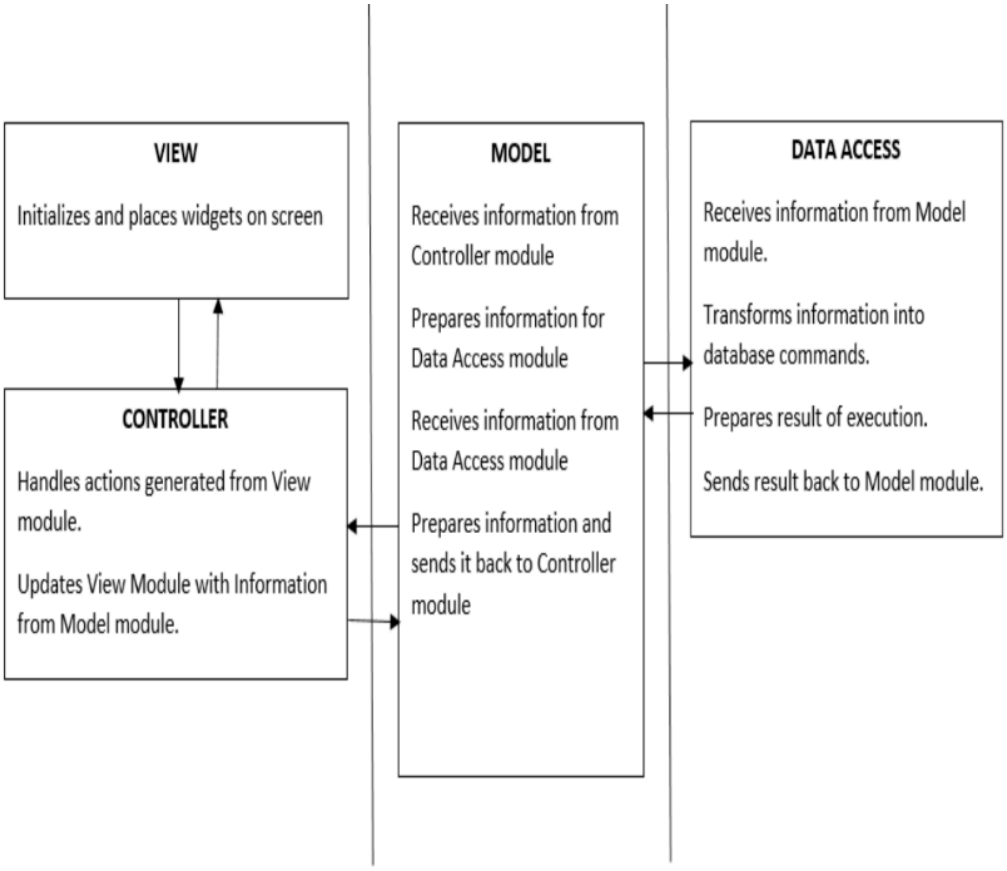
Architecture Diagram 1:



Architecture Diagram 2 :



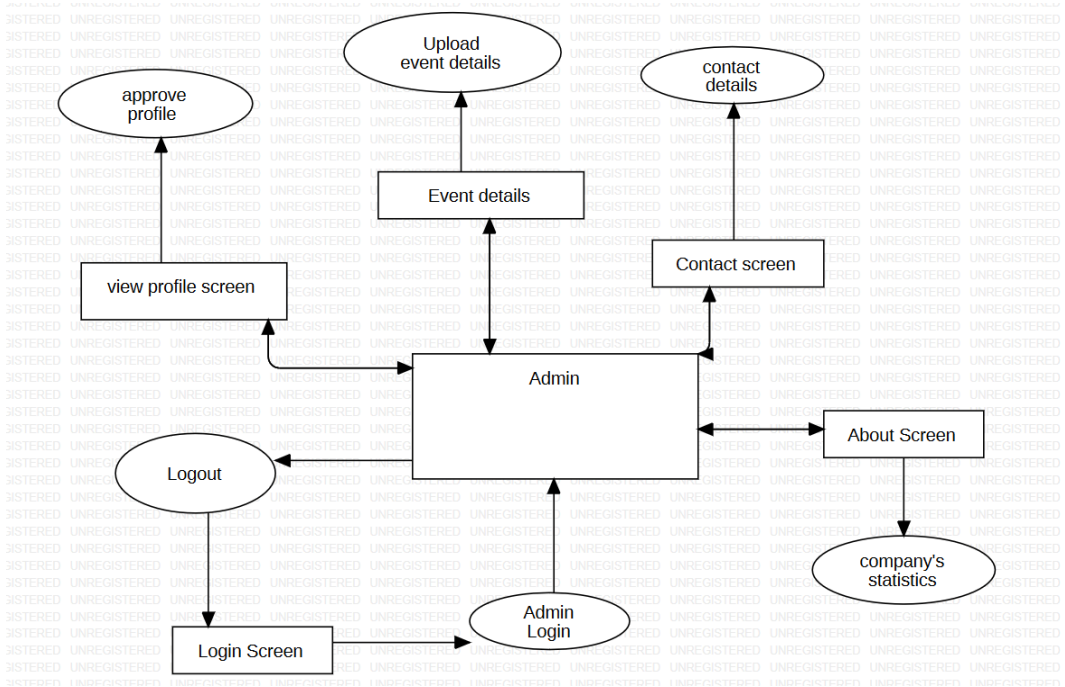
2.1 Overview of modules / components:



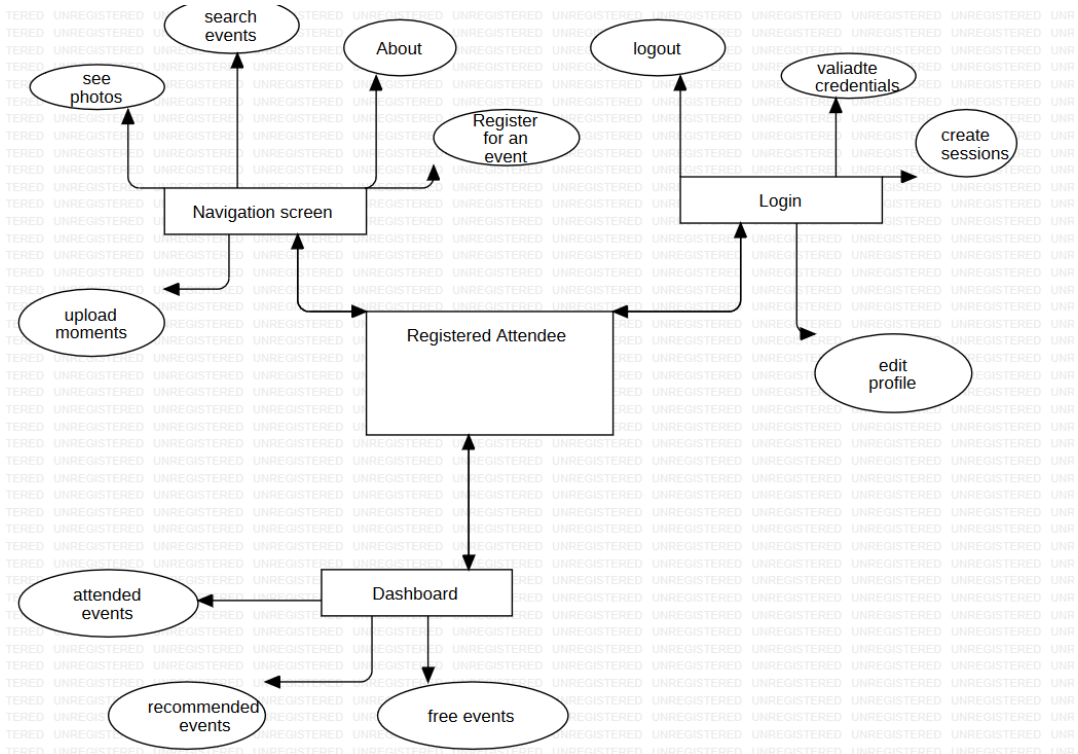
NOTE:
The horizontal lines represent the separation of modules.
More than one box within the same section represents sub-modules.

2.2 Structure and relationships

2.2.1 Admin's Side



2.2.2 User's side:



NOTE:
The boxes represent individual screens.
The circles represent actions that screens will do.
The arrows represent navigation between screens.

User Interface Issue:
Designing an effective user interface for an event management system involves addressing the needs and expectations of various user roles, including administrators, registered users, non-registered users, and sponsors. Here are some common UI issues and suggestions for each user type:

Administrator Interface:
Complexity: Admin interfaces often tend to become cluttered with too many options. Simplify the layout by grouping related functionalities and providing clear navigation paths.
Permissions: Ensure that administrators have access to all necessary tools and controls but restrict access to sensitive functionalities based on roles (e.g., financial data access only for finance admins).
Dashboard: Create a dashboard with customizable widgets for quick access to key metrics, such as upcoming events, registrations, etc.
Notification Management: Implement a system for managing notifications and alerts to keep admins informed about important updates or issues.

Registered User Interface:
User Profiles: Allow users to easily update their profiles, including personal information, preferences, and communication settings.
Event Registration: Streamline the event registration process with a clear and intuitive interface. Provide options for managing registrations, such as viewing past registrations, canceling registrations, etc.
Payment Processing: Integrate secure and user-friendly payment processing tools to facilitate seamless transactions during registration or for purchasing event-related items.
Feedback Mechanism: Include feedback forms or surveys after events to gather user feedback and improve future event experiences.

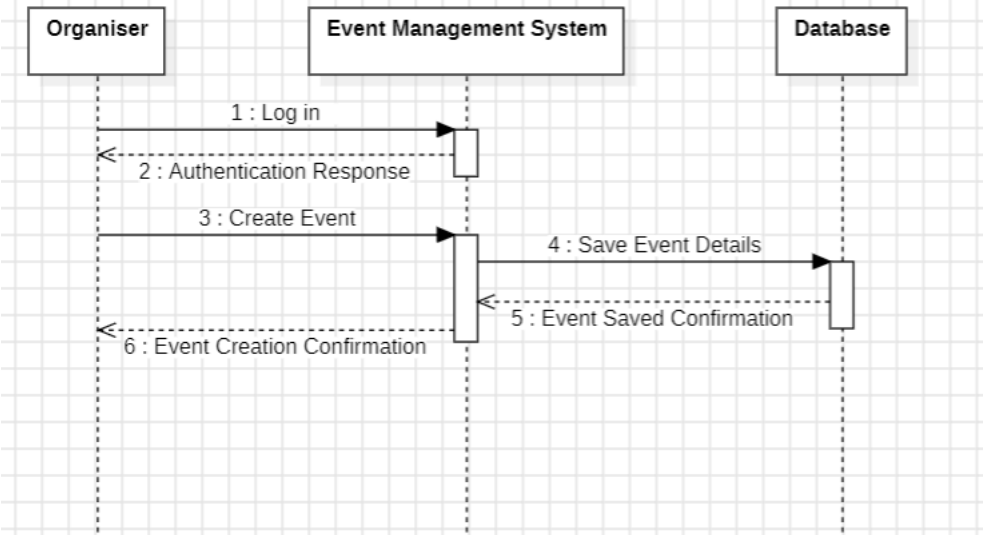
Non-Registered User Interface:
Clear Registration Path: Encourage non-registered users to sign up by highlighting the benefits (e.g., early access to events, personalized recommendations).
Event Preview: Provide a preview of upcoming events with basic information (date, time, location, description) to attract non-registered users to explore further.

Guest Registration: Allow non-registered users to register for events as guests, with the option to create an account during the registration process.
User-friendly Signup: Keep the signup process simple and minimize required fields to reduce friction and encourage signups.

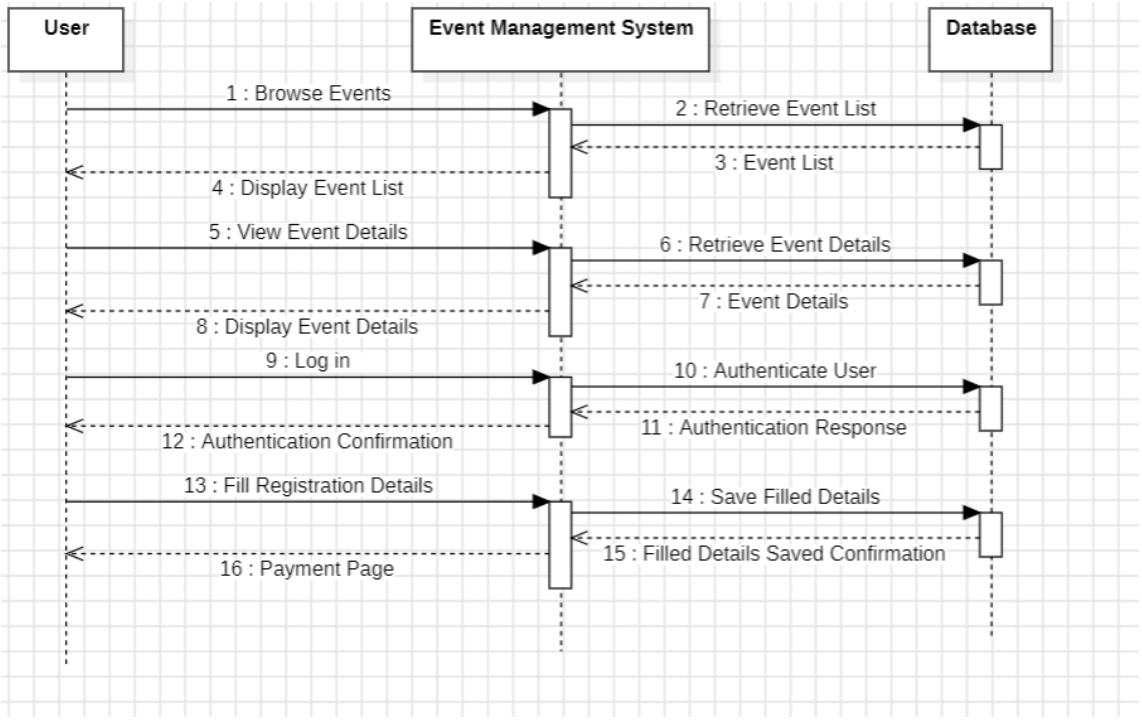
Sponsor Interface:
Sponsorship Opportunities: Clearly showcase available sponsorship opportunities, including benefits, pricing, and exposure levels.
Proposal Submission: Create a streamlined process for sponsors to submit sponsorship proposals or inquiries.
Tracking and Analytics: Provide sponsors with access to analytics and tracking tools to monitor the performance and ROI of their sponsorships.
Communication Channels: Enable direct communication channels between sponsors and event organizers for smooth coordination and collaboration.
Additionally, ensure that the UI design is responsive and accessible across different devices and screen sizes to cater to a diverse user base. Regular usability testing and gathering feedback from users across these roles can also help identify and address any usability issues or pain points in the interface.

SEQUENCE DIAGRAM

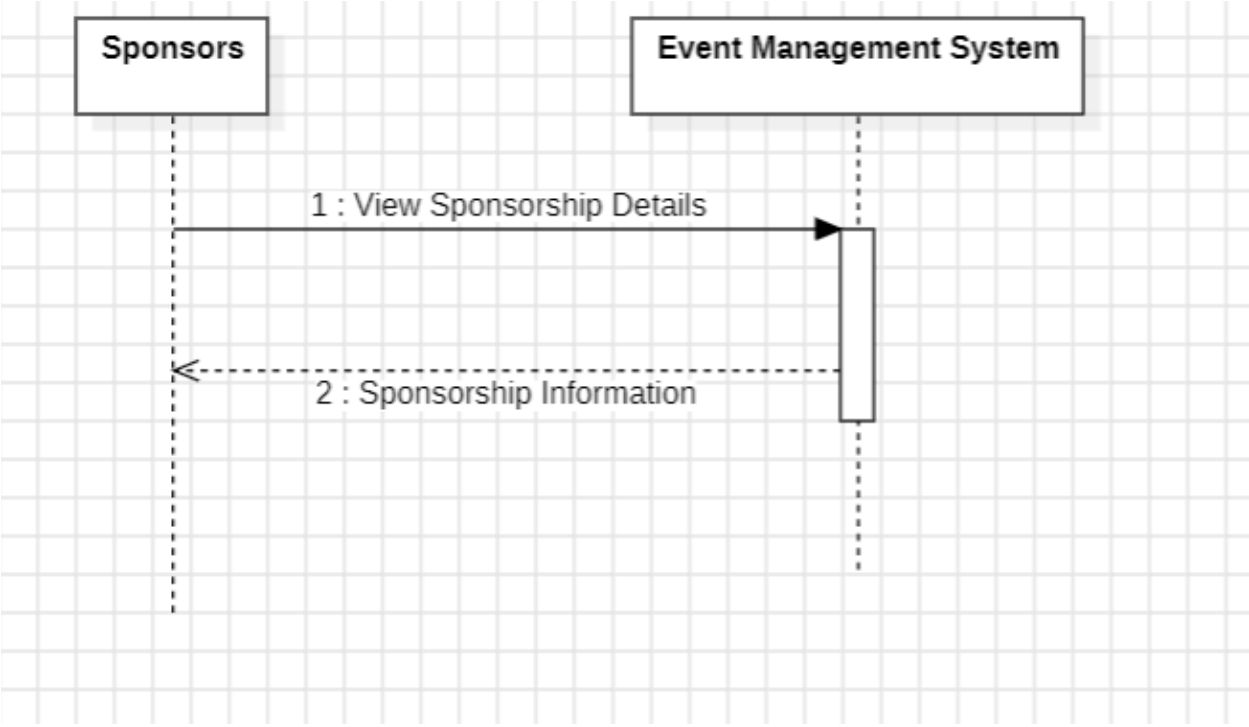
1. Event creation



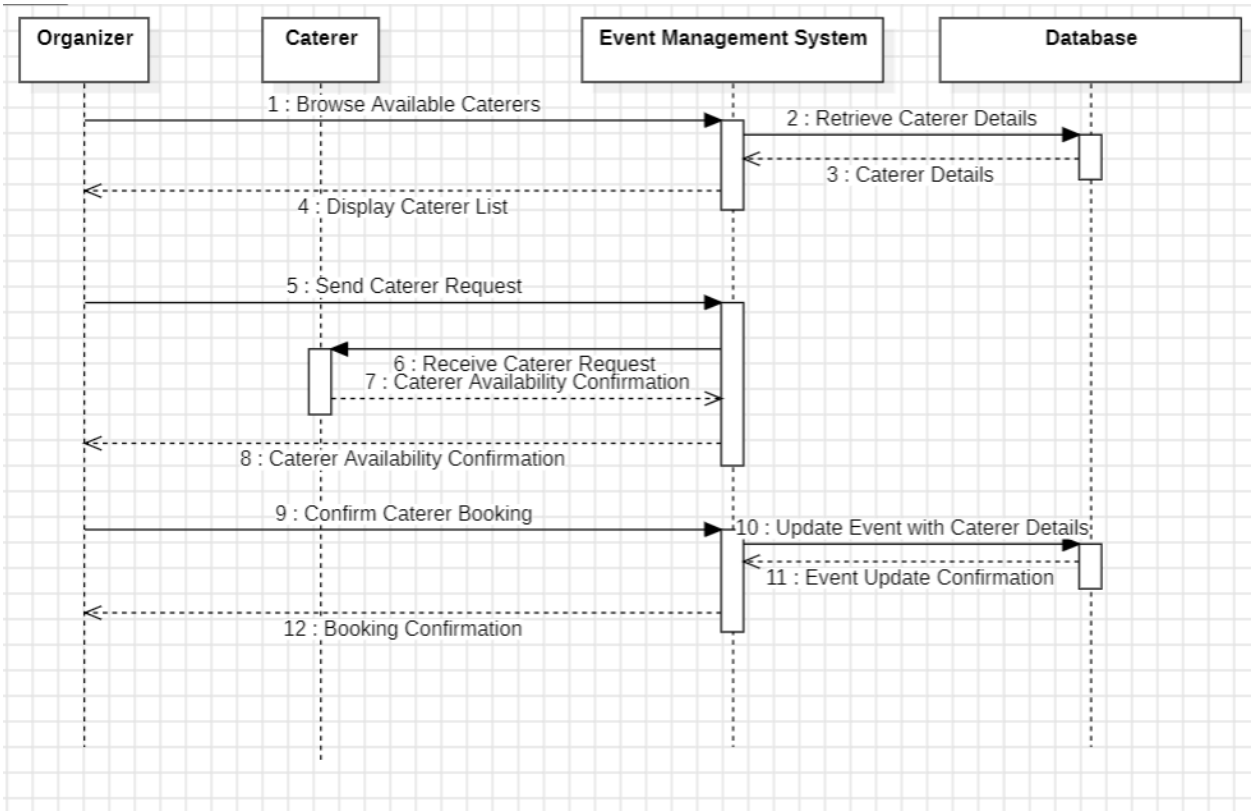
2. Event Registration



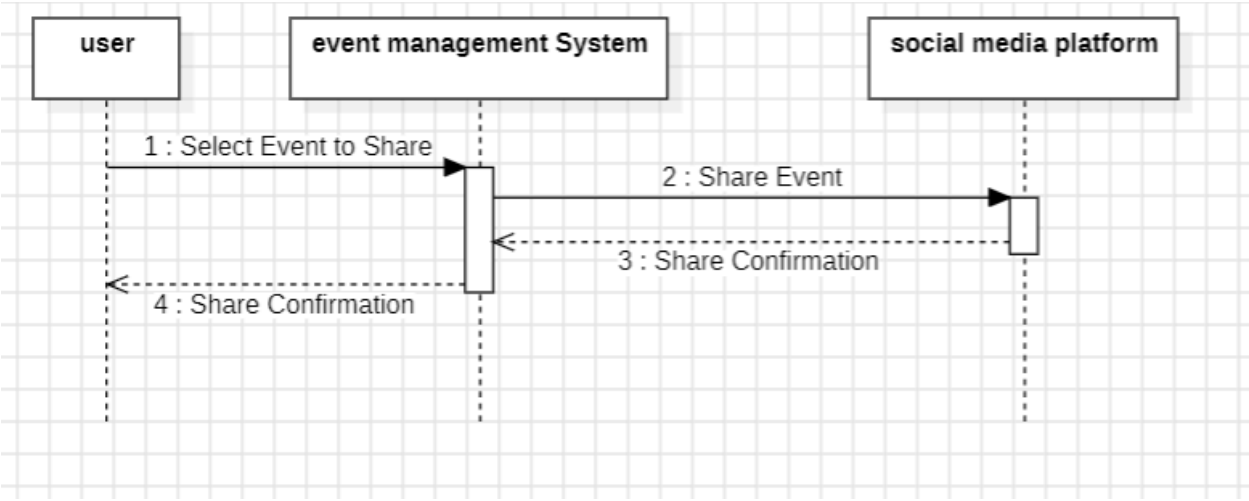
3. Sponsorship



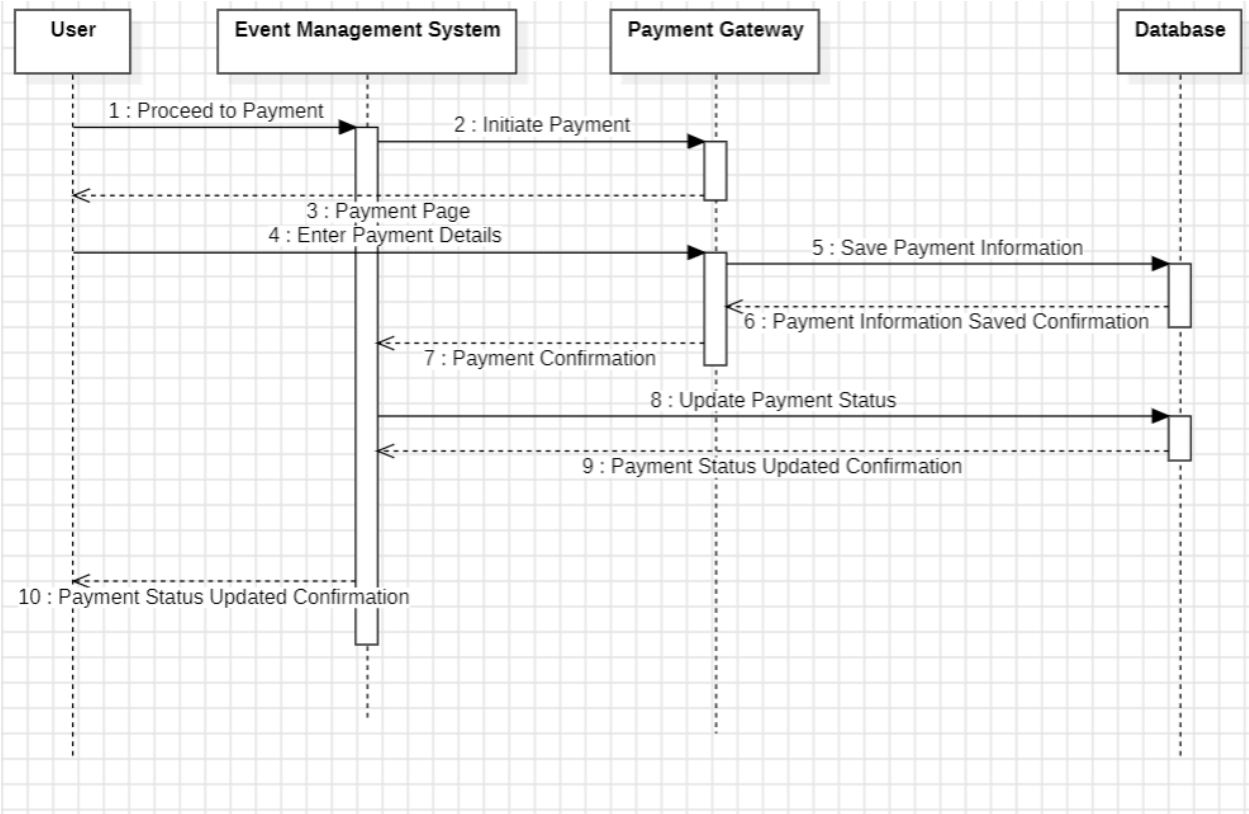
4. Caterers



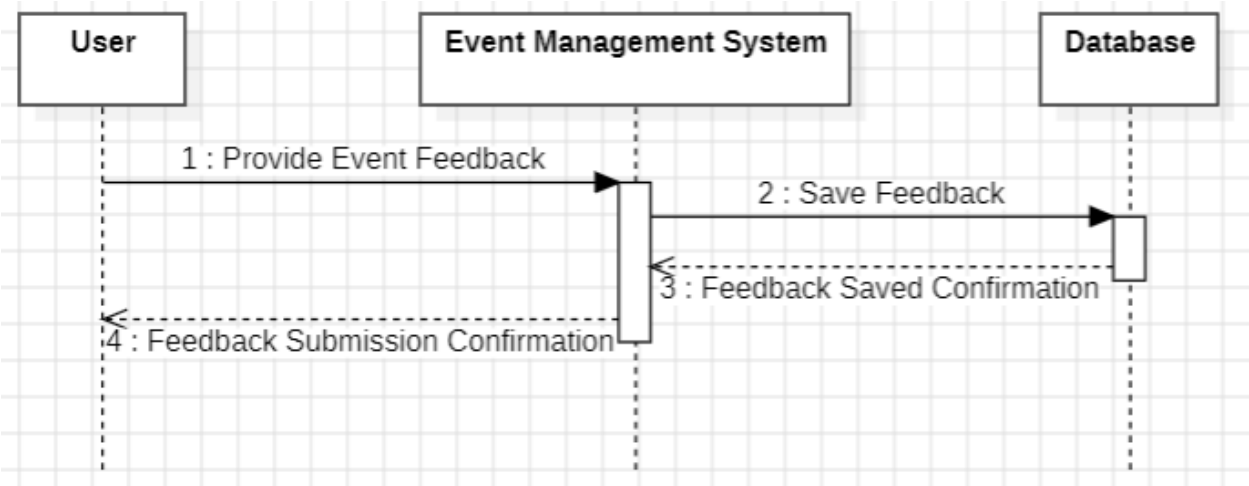
5, Sharing on Social Media



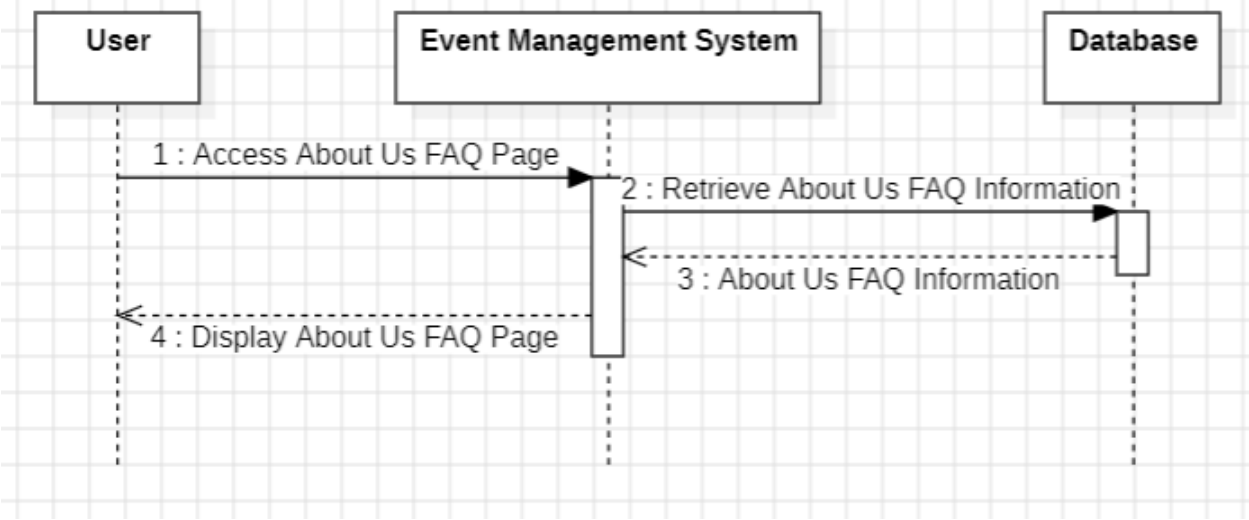
6. Payment



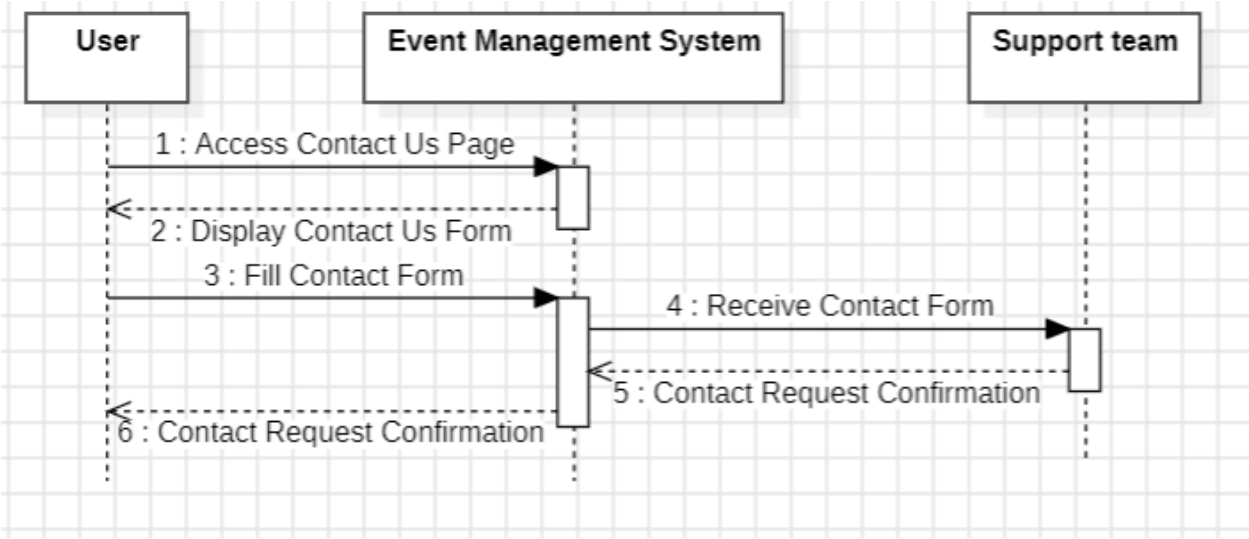
7. Feedback



8. About us/FAQ

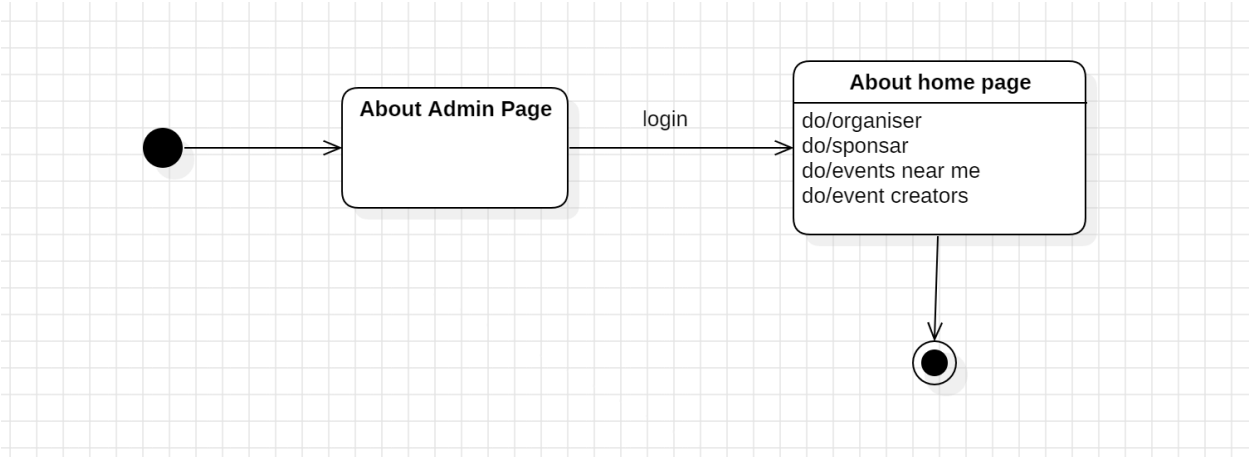


9. Contact page

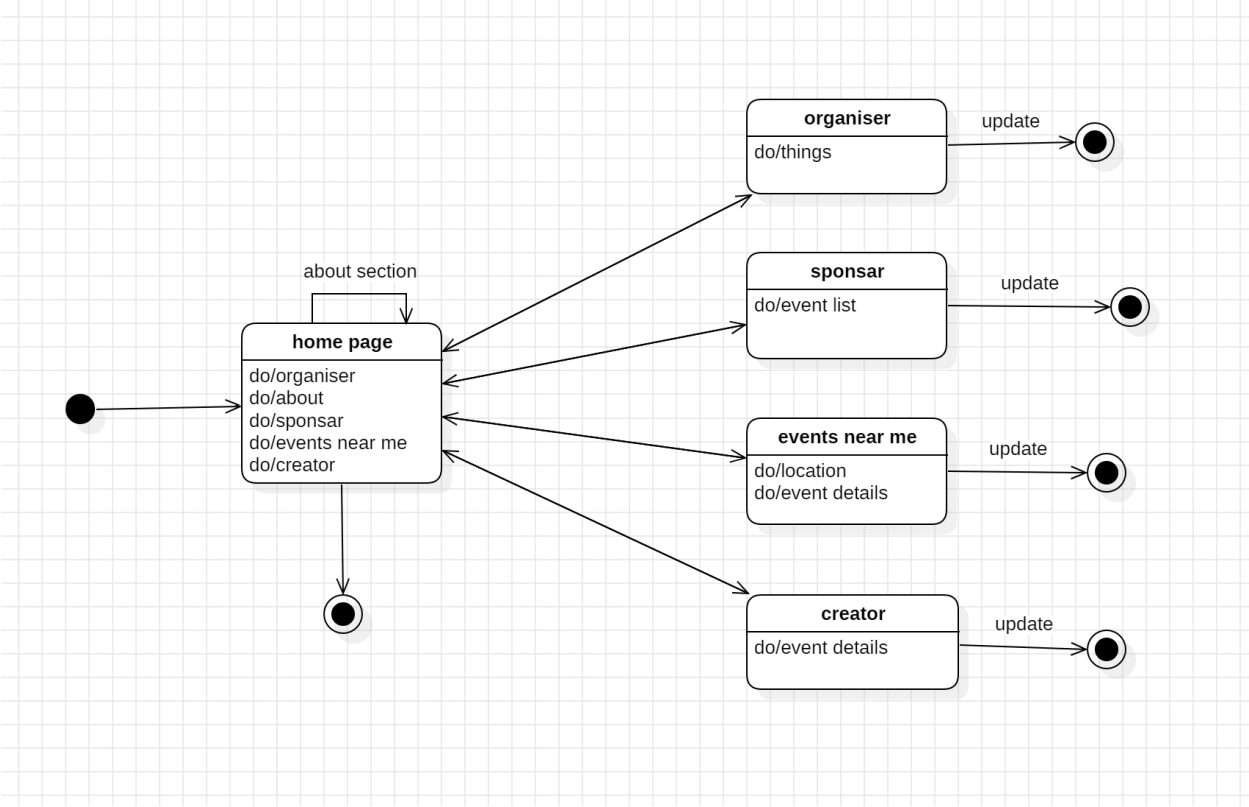


State Diagram

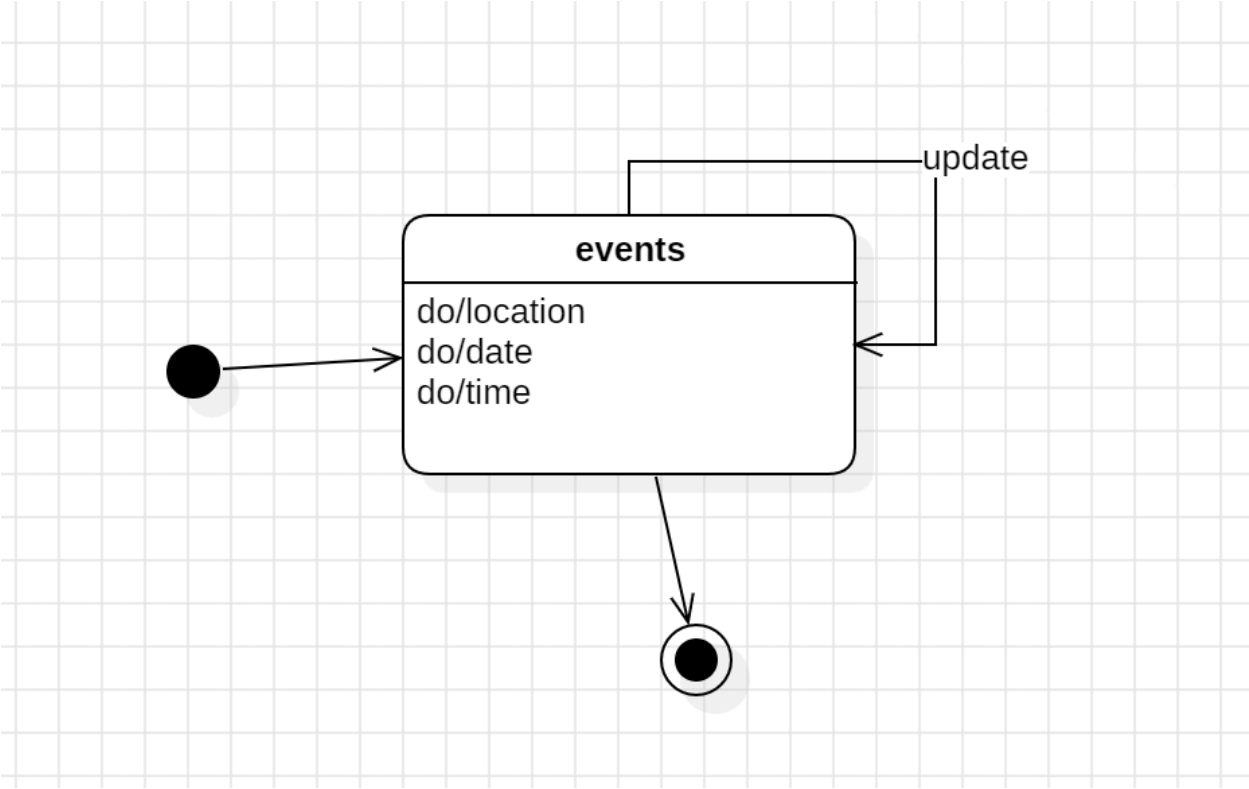
1. Admin Page



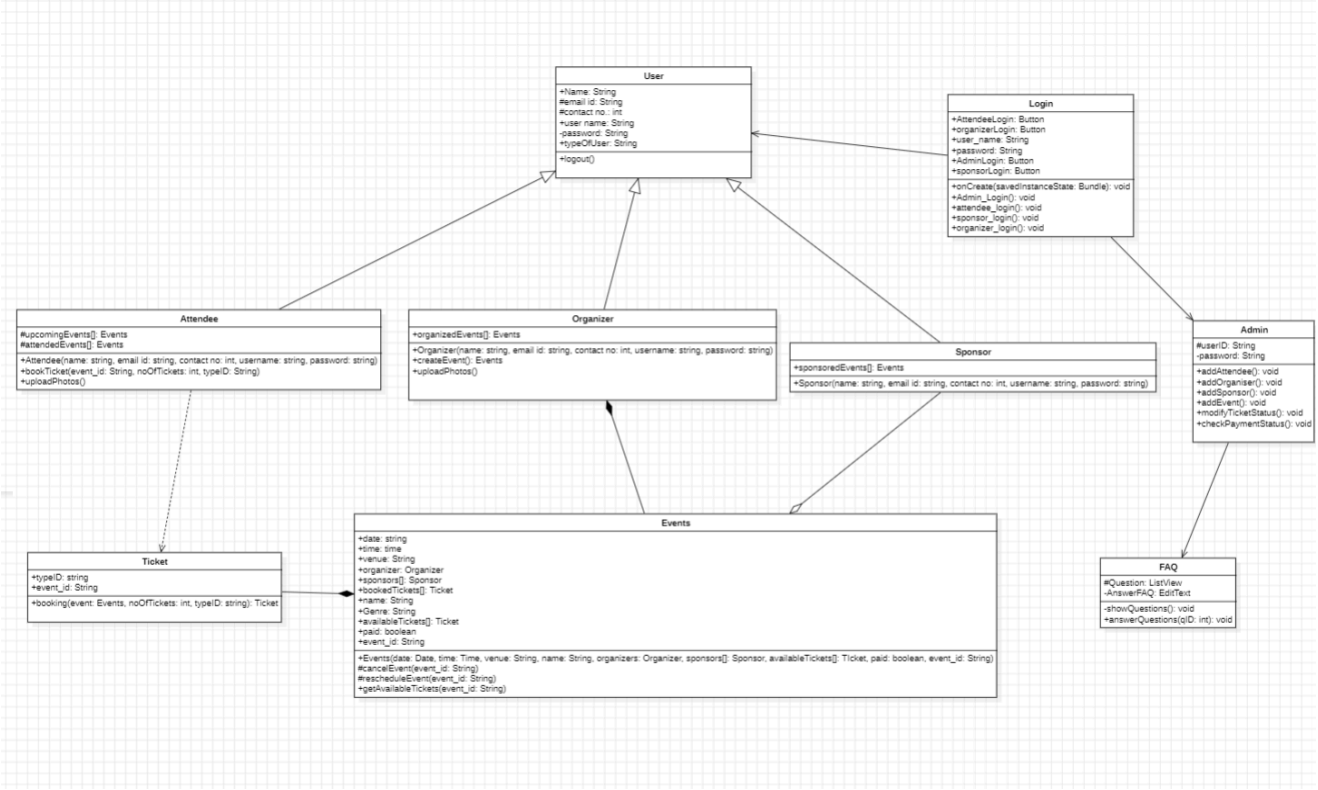
2. Home Page



3. Subpage (events near me)



CLASS DIAGRAM



SEQUENCE DIAGRAM DESCRIPTION:

Event Creation: The organizer provides event details, which are validated by the system and then stored in the database.

Event Registration (Listing): An attendee registers for an event, the system verifies the registration details, and updates the attendee database accordingly.

Event Details Page: A user requests details for a specific event, and the system retrieves the relevant information from the database to display it.

User Authentication: A user attempts to log in, the system verifies the provided credentials against stored data, and grants access if valid.

Sponsorship: Upon user request, the system retrieves sponsor details from the database and presents them to the user.

Social Media Sharing: The user selects to share an event on social media, the system provides sharing options and facilitates the sharing process.

Payment Process: The user initiates payment, the system collects payment information, processes it securely, and communicates with the payment gateway.

Feedback and Ratings: An attendee provides feedback and ratings for an event, the system stores this feedback in the database for future reference.

About Us/FAQ Page: The user accesses the page to find general information, the system retrieves and displays this information from the database.

Contact Us Page: The user accesses the page to contact support, the system provides contact options and facilitates communication with the user.

CLASS DIAGRAM DESCRIPTION:

1. Class Name: User
   Description: This class allows the new user to register and allow them to enter the system.

   Methods:

1.1 Logout()

Method Description:
It is designed to facilitate the process of terminating a user's session or session-related activities within a software system or application. Upon invocation, this method performs the necessary actions to invalidate the user's current session and ensure that they are securely logged out of the system.

Child Classes : Attendee, Organizer, Sponsor

2. Class name: Attendee
   Methods:

2.1 Attendee()

Method Description:
It is a constructor method with parameters : name, email id , contact, username, password which helps in registering new users as type – attendee into the system.

2.2 bookTicket()

Method Description:
The bookTicket method is a crucial functionality. It facilitates the process of reserving and purchasing tickets for various events or activities offered within the

system. This method allows users to select their desired event, specify the number of tickets they wish to purchase, and complete the booking transaction securely.

Upon invocation, the bookTicket method prompts the user to provide relevant details such as the event name or ID, the number of tickets required, and any additional preferences or requirements, such as seating preferences or special accommodations. The method then validates the availability of the requested tickets for the specified event and checks for any potential conflicts or constraints, such as overlapping bookings or capacity limitations.

Once validated, the method proceeds to reserve the requested tickets, updating the availability status and allocating the designated seats or slots to the user. Depending on the implementation, the method may also handle payment processing, allowing users to securely complete the transaction using various payment methods supported by the system.

Parameters:
event_id: The unique identifier or code associated with the event for which tickets are being booked.
noOfTickets: The number of tickets requested by the user for booking.
typeID: The identifier for the type of tickets (if necessary) to be booked.

2.3 uploadPhotos()

Method Description:
The uploadPhotos is a functionality within the Attendee class enabling attendees to upload photos related to the event they are participating in. This method facilitates the process of adding visual content captured during the event, allowing attendees to share their experiences and memories with others.

Upon invocation, the uploadPhotos method prompts the attendee to select one or multiple photos from their device or an external source. The method then processes these photos, verifying their format, size, and any other relevant criteria to ensure compatibility and compliance with the system's requirements. Once validated, the method uploads the photos to the designated storage location within the system.

3. Class name: Organizer
Methods:

3.1 Organizer()

Method Description:

It is a constructor method with parameters : name, email id , contact, username, password which helps in registering new users as type – organizer into the system.

3.2 createEvent()

Method Description:
The createEvent method is designed for organizers to create and configure new events within the system. This method empowers organizers to define event details such as the event name, date and time, location, description, and any other relevant information necessary for attendees to understand and participate in the event.
Upon invocation, the createEvent method prompts the organizer to input the required event details through a user-friendly interface or form.
Once validated, the method proceeds to create a new event record within the system's database, storing all the specified details and assigning a unique identifier or code to the event for future reference.

Return Value:
It returns the event details of type Events.

3.3 uploadPhotos()

Method Description:
The functionality of uploadPhotos is the same as uploadPhotos method within the Attendee class. It enables organizers to upload photos related to the event they are organizing.

4. Class name: Sponsor
Methods:

4.1 Sponsor()

Method Description:
It is a constructor method with parameters : name, email id , contact, username, password which helps in registering new users as type – sponsor into the system.

5. Class name: Login
Methods:

5.1 onCreate()

Method Description:
When an Activity first calls or launches, the onCreate(Bundle savedInstanceState) method is responsible for creating the activity. Whenever orientation(i.e. from

horizontal to vertical or vertical to horizontal) of activity gets changed the object of Bundle class will save the state of an Activity. Basically Bundle class is used to store the data of activity whenever the above condition occurs in app. SetContentView is used to fill the window with the UI provided from the layout file. This method takes input as username and password and as a result opens the landing page if login is successful.

Parameters:
savedInstanceState, username, Password

Output: Launch the Activity, SignIn

5.2 admin_Login(), attendee_login(), organizer_login(), or sponsor_login()

Method Description:
This method takes input as username and password from the user (or admin) and checks whether it is authorized login or not and if the result is successful it leads to another activity page.

Parameters:
Username, password
Output: landing page of login successful

6. Class name: Events
Methods:

6.1 Events()

Method Description:
It is a constructor method with parameters: name, date, time, venue, organizer, event_id, availableTickets and paid to specify if it's tickets are paid or not.
It is invoked by the organizer while creating a new event.

6.2 cancelEvent()

Method Description:
The cancelEvent method allows organizers to cancel events that cannot proceed as planned due to unforeseen circumstances, logistical issues, or other reasons necessitating event cancellation. This method allows organizers to efficiently communicate the cancellation decision to registered attendees, manage refunds or compensations, and update the event status within the system.

Parameters:

event_id: The unique identifier or code associated with the event to be canceled. This parameter enables the method to locate and modify the corresponding event record within the system's database.

6.3 rescheduleEvent()

Method Description:
The rescheduleEvent method allows organizers to reschedule events due to unforeseen circumstances, logistical issues, or other reasons necessitating event rescheduling.

Parameters:
event_id: The unique identifier or code associated with the event to be rescheduled. This parameter enables the method to locate and modify the corresponding event record within the system's database.

6.4 getAvailableTickets()

Method Description:
The getAvailableTickets method returns the number of tickets available to be booked for an event.

Parameters:
event_id: The unique identifier or code associated with the event.

Returns:
Number of tickets available.

7. Class name: Tickets
Methods:

7.1 booking()

Method Description:
Called by attendee to book the ticket for an event by passing event_id, number of tickets and type of tickets.
Returns the tickets of type Ticket to the attendee class from where it is called.

8.1 State Diagram Description:

Initial state is being shown by starting with a black dot. Final State is being shown by the black dot surrounded by an empty circle.

8.1.1 Admin Page:

This is the page that the user sees first and has the sign-up or login information on it. You will be sent to the homepage after logging in.

8.1.2 Home page:

The home page will have an About page and four more tabs that lead to the Organizers section. These tabs will help you work with the event organizer to improve the experience of your event. Sponsors: this will make it simple for you to locate partners for your event. Organizers: those in charge of the event will keep you informed about its specifics.Events around me will enable you to locate nearby events based on your interests.

8.1.3 Events:

By providing your location and an appropriate filter, you can easily locate events nearby that fit into your calendar.

CODE

Events Page

```
import React, { useEffect, useMemo, useState } from "react";
import api from "../../Services/api";


import {
  Alert,
  Button,
  Container,
  Col,
  Form,
  FormGroup,
```

```
  Input,
  InputGroup,
  InputGroupAddon,
  Label
} from "reactstrap";
import "./events.css";


import '../../components/NavigationBar/assets/css/main.css'
function EventsPage({ history }) {


  const [title, setTitle] = useState("");
  const [description, setDescription] = useState("");
  const [price, setPrice] = useState();
  const [thumbnail, setThumbnail] = useState(null);
  const [date, setDate] = useState("");
  const [eventType, setEventType] = useState("Event Type");
  const [error, setError] = useState(false);
  const [success, setSuccess] = useState(false);
  const [dropdownOpen, setOpen] = useState(false);
  const user = localStorage.getItem("user");


  useEffect(() => {
    var today = new Date();
    var dd = today.getDate();
    var mm = today.getMonth() + 1; //January is 0!
    var yyyy = today.getFullYear();
    if (dd < 10) {
      dd = '0' + dd
    }
    if (mm < 10) {
      mm = '0' + mm
    }
    today = yyyy + '-' + mm + '-' + dd;
    document.getElementById("datefield").setAttribute("min", today);


    if (!user) {
      history.push("/login");
    }
    // eslint-disable-next-line react-hooks/exhaustive-deps
```

```
}, []);


const toggle = () => setOpen(!dropdownOpen);


const preview = useMemo(() => {
  return thumbnail ? URL.createObjectURL(thumbnail) : null;
}, [thumbnail]);


const handleEventSubmit = async (event) => {
  event.preventDefault();
  const eventData = new FormData();


  eventData.append("thumbnail", thumbnail);
  eventData.append("eventType", eventType);
  eventData.append("title", title);
  eventData.append("price", price);
  eventData.append("description", description);
  eventData.append("date", date);


  try {
    if (
      title !== "" &&
      description !== "" &&
      price !== null &&
      eventType !== "" &&
      date !== "" &&
      thumbnail !== null
    ) {
      await api.post("/event", eventData, { headers: { user } });
      setSuccess(true);
      setTimeout(() => {
        setSuccess(false);
        history.push("/");
      }, 2000);
    } else {
      setError(true);
      setTimeout(() => {
        setError(false);
```

```
    }, 2000);
   }
  } catch (er) {
   Promise.reject(er);
   console.log(er.message);
  }


  return "";
};
return (
  <Container style={{ marginTop: "5%" }}>


   <center className="logo"><h2>Create your event</h2></center>


   <Form onSubmit={handleEventSubmit}>


    <FormGroup row>
     <Label for="exampleTitle" sm={2}>Event Title</Label>
     <Col sm={10}>
      <Input placeholder="Enter event title"
       bsSize="lg"
       id="title"
       type="text"
       value={title}
       onChange={(event) => {
        setTitle(event.target.value);
       }} />
     </Col>
    </FormGroup>
    <FormGroup row>
     <Label for="exampleText" sm={2}>Event Description</Label>
     <Col sm={10}>
      <Input
       bsSize="lg"
       placeholder="Enter event description"
       id="description"
       type="textarea"
       value={description}
       onChange={(event) => {
```

```
      setDescription(event.target.value);
     }}
    />
  </Col>
</FormGroup>
<FormGroup row>
  <Label for="exampleSelect" sm={2}>Event Type</Label>
  <Col sm={10}>
    <Input type="select" name="select" id="exampleSelect"
     onChange={(event) => {
      setEventType(event.target.value);
     }}
    >
     <option disabled selected>Select event-type</option>
     <option value="webinar">Webinar</option>
     <option value="seminar" >Seminar</option>
     <option value="workshop" >Workshop</option>
    </Input>
  </Col>
</FormGroup>
<FormGroup row>
  <Label for="exampleText" sm={2}>Event Price</Label>
  <Col sm={10}>
    <InputGroup>
     <InputGroupAddon addonType="prepend">₹</InputGroupAddon>
     <Input
      placeholder="Enter price of event"
      id="price"
      type="number"
      min="0"
      step="any"
      value={price}
      onChange={(event) => {
       setPrice(event.target.value);
      }}
     />
    </InputGroup>
  </Col>
</FormGroup>


<FormGroup row>
  <Label for="datefield" sm={2}>Select Date</Label>
```

```
        <Col sm={10}>
          <Input
            placeholder="Enter date of event"
            id="datefield"
            type="date"
            value={date}
            onChange={(event) => {
              setDate(event.target.value);
            }}
          />
        </Col>
      </FormGroup>
      <FormGroup row>
        <Label for="exampleFile" sm={2}>Thumbnail</Label>
        <Col sm={10}>
          <Input id="thumbnail"
            type="file"
            onChange={(event) => {
              setThumbnail(event.target.files[0]);
            }} />
        </Col>
      </FormGroup>


      <br />
      <FormGroup>
        <Button className="submit-btn" color="success" size="lg">Create
Event</Button>
        <Button
          className="secondary-btn"
          onClick={() => {
            history.push("/");
          }}
          color="danger"
          size="lg"
        >
          Cancel
        </Button>
      </FormGroup>
      <FormGroup>
      </FormGroup>
    </Form>
```

```
      {error ? (
        <Alert color="danger" className="event-validation">
          Missing required information
        </Alert>
      ) : (
          ""
        )}


      {success ? (
        <Alert color="success" className="event-validation">
          Event was created successfully
        </Alert>
      ) : (
          ""
        )}
    </Container>
  );
}


export default EventsPage;
```

Login Page:

```
import React, { useState, useContext } from "react";
import api from "../../Services/api";
import { Button, Form, FormGroup, Alert } from "reactstrap";
import { UserContext } from '../../user-context';

import SignIn from '../../assets/LoginRegister/images/signin-image.jpg'

import { MdLock, MdEmail } from "react-icons/md";

function  Login({ history }) {
  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState(false);
```

```
const [errorMessage, setErrorMessage] = useState("");
const { setIsLoggedIn } = useContext(UserContext);

const handleSubmit = async (event) => {
  event.preventDefault();

  const response = await api.post("/login", { email, password });
  const user_id = response.data.user_id || false;
  const user = response.data.user || false;

  try {
    if (user && user_id) {
      localStorage.setItem("user", user);
      localStorage.setItem("user_id", user_id);

      setIsLoggedIn(true);
      history.push("/");
    } else {
      const { message } = response.data;
      setError(true);
      setErrorMessage(message);
      setTimeout(() => {
        setError(false);
        setErrorMessage("");
      }, 2000);
    }
  } catch (err) {
    setError(true);
    setErrorMessage("Error is " + err);
  }
};

return (

  <div className="main">
    <section className="sign-in">
      <div className="container">
        <div className="signin-content">
          <div className="signin-image">
            <figure><img src={SignIn} alt="sing up image" /></figure>
            <a className="signup-image-link"
              style={{ cursor: "pointer" }}
              onClick={() => {
```

```
            history.push("/register");
          }}>Create an account</a>
        </div>

        <div className="signin-form">
          <h2 className="form-title">Log In</h2>
          <Form onSubmit={handleSubmit}>
            <div className="form-group" style={{ justifyContent: "center" }}>
              <label for="your_name"><MdEmail fontSize="large" /></label>
              <input
                onChange={(event) => {
                  setEmail(event.target.value);
                }}
                type="email"
                name="email"
                id="exampleEmail"
                placeholder="Enter your email here"
              />
            </div>
            <div className="form-group" style={{ justifyContent: "center" }}>
              <label for="your_pass"><MdLock fontSize="large" /></label>
              <input
                onChange={(event) => {
                  setPassword(event.target.value);
                }}
                type="password"
                name="password"
                id="examplePassword"
                placeholder="Enter your password here"
              />
            </div>
            <FormGroup>
              <Button color="success" className="submit-btn"
size="lg">Submit</Button>
            </FormGroup>

          </Form>
          {errorMessage ? (
            <Alert color="danger" className="event-validation">
              {errorMessage}
            </Alert>
          ) : (
            ""
```

```
            )}
          </div>
        </div>
      </div>
    </section>

  </div>
 );
}

export default Login;
```