

EVENTLY

Project By:

Sr.No	Name	Email
1	Niraj Nitin Surve	dse21123406@git-india.edu.in
2	Mangesh Shantaram Poskar	dse21122855@git-india.edu.in
3	Rutikesh Ravindra Sawant	rutikeshsawant1234@gmail.com

ABSTRACT

The Event Management System Website (EMS) represents a sophisticated digital solution for modern event management needs. In a world marked by the increasing complexity and diversity of events, the EMS serves as a central hub for organizers and participants alike. This web-based platform empowers event planners, coordinators, and attendees by offering an intuitive, feature-rich environment that simplifies every aspect of event preparation and participation.

The EMS website encompasses a wide range of functionalities to facilitate the creation, promotion, and execution of events, spanning from small-scale gatherings to large-scale conferences and conventions. It provides a seamless experience for users, ensuring that events are meticulously planned, efficiently marketed, and successfully executed.

Accessible through desktop and mobile devices, the EMS website offers an integrated and personalized experience to all stakeholders. It adapts to the unique needs of each event, ensuring scalability and flexibility to accommodate varying sizes and requirements. With a focus on user-friendliness and efficiency, the EMS website is the cornerstone for creating memorable, engaging, and successful events in today's dynamic event landscape.

ACKNOWLEDGEMENT

In this project, we have made an effort. Nevertheless, it would not have been possible without the generous support and assistance of numerous people and organizations. We would like to express our gratitude towards Symbiosis Skills and Professional University. We would like to extend our sincere thanks to our trainer Mr. Chakradhar Shinde sir. We are highly indebted to our trainer for his kind cooperation, encouragement, guidance and constant supervision as well as for providing necessary information regarding the project & also for his support in completing the project.

DECLARATION

We declare that the project entitled "**Evently** " submitted as part of our academic requirements for Java FullStack at Symbiosis Skills and Professional University, is entirely our original work. We further declare that this project has not been duplicated or submitted in any form, in full or in part, to any other university or educational institution for the award of any degree or qualification.

To the best of our knowledge and belief, other than our self, no individual has submitted this project to any other university or institution for academic credit or certification.

LIST OF FIGURES

Figure No.	Figure Name	Page Number
4.1	Class Diagram	5
4.2	Data Flow Diagram	6

CONTENT

ABSTRACT	II
ACKNOWLEDGEMENT	III
DECLARATION	IV
LIST OF FIGURES	V
1. Introduction	1
2. Feature and functionality	2
3. Database Design	4
4. Diagram	5
5. Design and Implementation	7
6. Source Code	10
7. Conclusion and Future Work	21

Chapter 1

Introduction

In today's fast-paced world, managing events has evolved into a complex and demanding task. To address the challenges of event planning, coordination, and execution, Evently emerges as the ultimate solution. This cutting-edge Event Management System website is built around three core entities: User, Event, and Ticket, each playing a pivotal role in redefining the way events are organized and experienced.

The User class within Evently recognizes the central importance of user engagement. Catering not only to event organizers but also to attendees, Evently ensures a holistic and user-centric approach. Organizers find a user-friendly platform to effortlessly create, manage, and promote events. Meanwhile, attendees enjoy a seamless ticket booking experience and access to comprehensive event information, enriching their overall event participation.

At the core of Evently lies the Event class, which encapsulates the essence of every gathering. This class empowers event organizers to meticulously define event details, encompassing critical aspects such as event dates, times, venues, and objectives. With Evently's robust features, organizers can craft event agendas, manage attendee lists, and maintain essential event-related data with ease.

Complementing the Event class is the Ticket class, designed to simplify the ticketing process. Attendees benefit from a straightforward ticket registration and purchase system, while organizers effortlessly oversee ticket bookings and reservations. This class ensures efficient resource allocation, facilitates revenue tracking, and simplifies event check-in and access control.

As we delve deeper into the intricacies of Evently, we will explore how this innovative system simplifies event management for professionals and elevates the event experience for attendees. Evently's flexible architecture, user-friendly interface, and robust features empower organizers to curate unforgettable moments, ensuring the success of events regardless of their scale or nature. Join us on this journey to discover how Evently reshapes the event management landscape, combining technological sophistication with the artistry of creating memorable experiences.

Chapter 2

Features and Functionality

User Authentication and Authorization:

1. User Registration:

Users can sign up for an Evently account by providing basic information (name, email, Password). Implement email verification to ensure the validity of user accounts.

2. User Login:

Registered users can log in securely using their email and password.

Event Management:

1. Event Creation:

Event organizers can create new events by providing event details such as the event name, date, time, location, description, and images.

2. Event Editing and Deletion:

Event organizers can edit event details and delete events they've created.

Event Listings:

1. Event Listings:

Users can browse a list of upcoming events on the Evently platform.

Events are displayed with key information like event name, date, time, and location.

2. Search and Filters:

Users can search for events based on criteria like date, location, and category.

Implement filters to refine search results.

Event Registration:

1. Event Registration:

Registered users can easily register for events with a simple registration form.

Users receive confirmation emails upon successful registration.

2. Attendee List:

Event organizers can access a list of registered attendees for their events.

User Profiles:

1. User Profiles:

Users can create and update their profiles, adding personal information, profile pictures, and event preferences.

2. My Events:

Users can view a list of events they have registered for in their user profile.

Notifications:**1. Email Notifications:**

Event organizers and attendees receive email notifications for event-related actions, such as event registration confirmation and updates.

Used Tools & Technologies

Technology/Domain- Java FullStack

Front-End- ReactJS

Database- MySQL

Back-End- Java

Chapter 3

Database Design

- **Event Table –**

Field Name	DataType
<u>EventID</u>	Integer
EventName	String
EventDate	Date
EventLocation	String
EventDescription	String
EventCapacity	Integer
EventType	String

- **User Table –**

Field Name	DataType
<u>UserID</u>	Integer
FirstName	String
LastName	String
Email	String
Password	String
UserType	String
Organization	String

- **Ticket Table –**

Field Name	DataType
<u>TicketID</u>	Integer
EventID	String
UserID	Date
TicketType	String

Chapter 4

Diagram

- **Class Diagram**

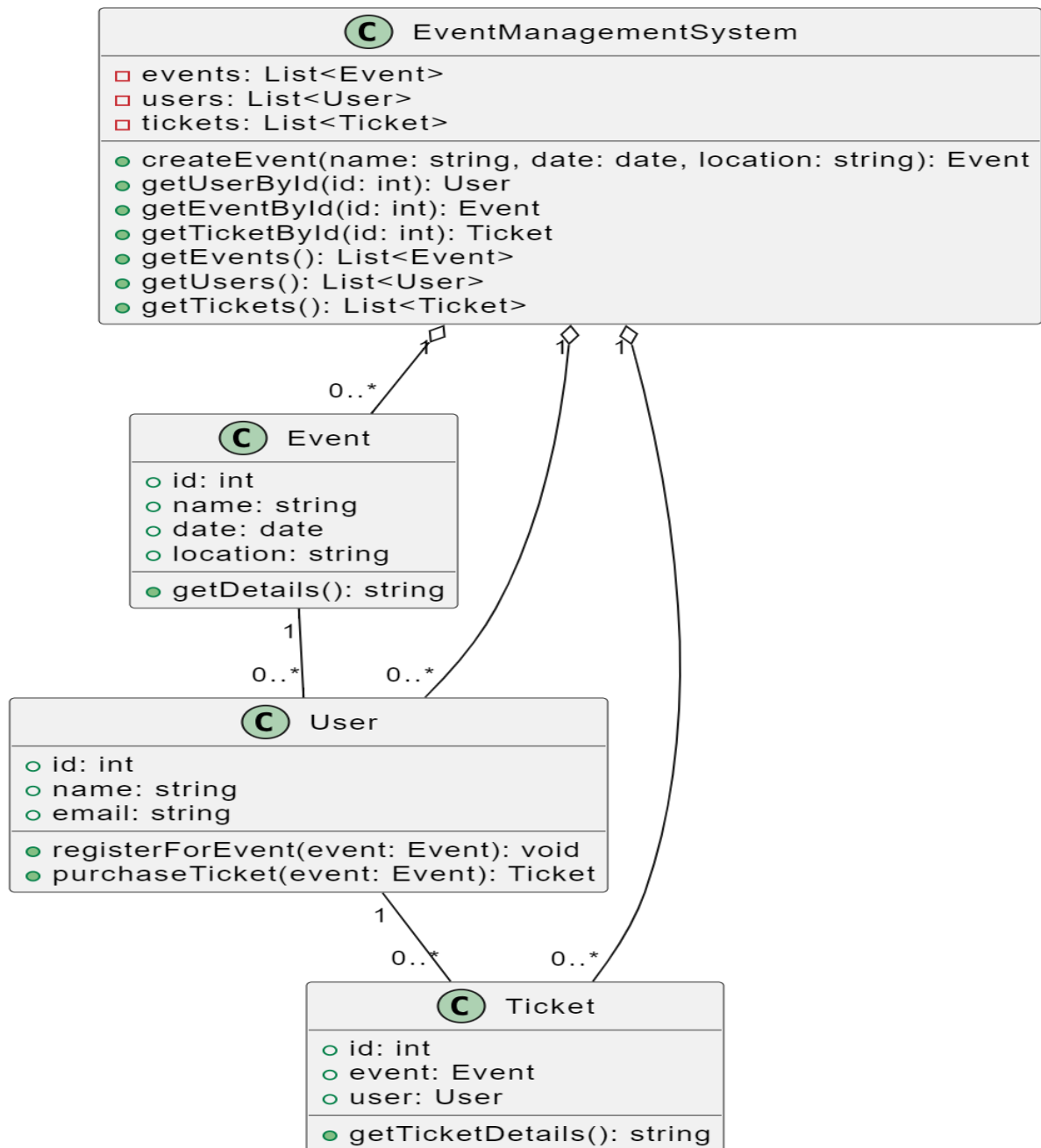


Fig 4.1: Class Diagram

- **Data Flow Diagram**

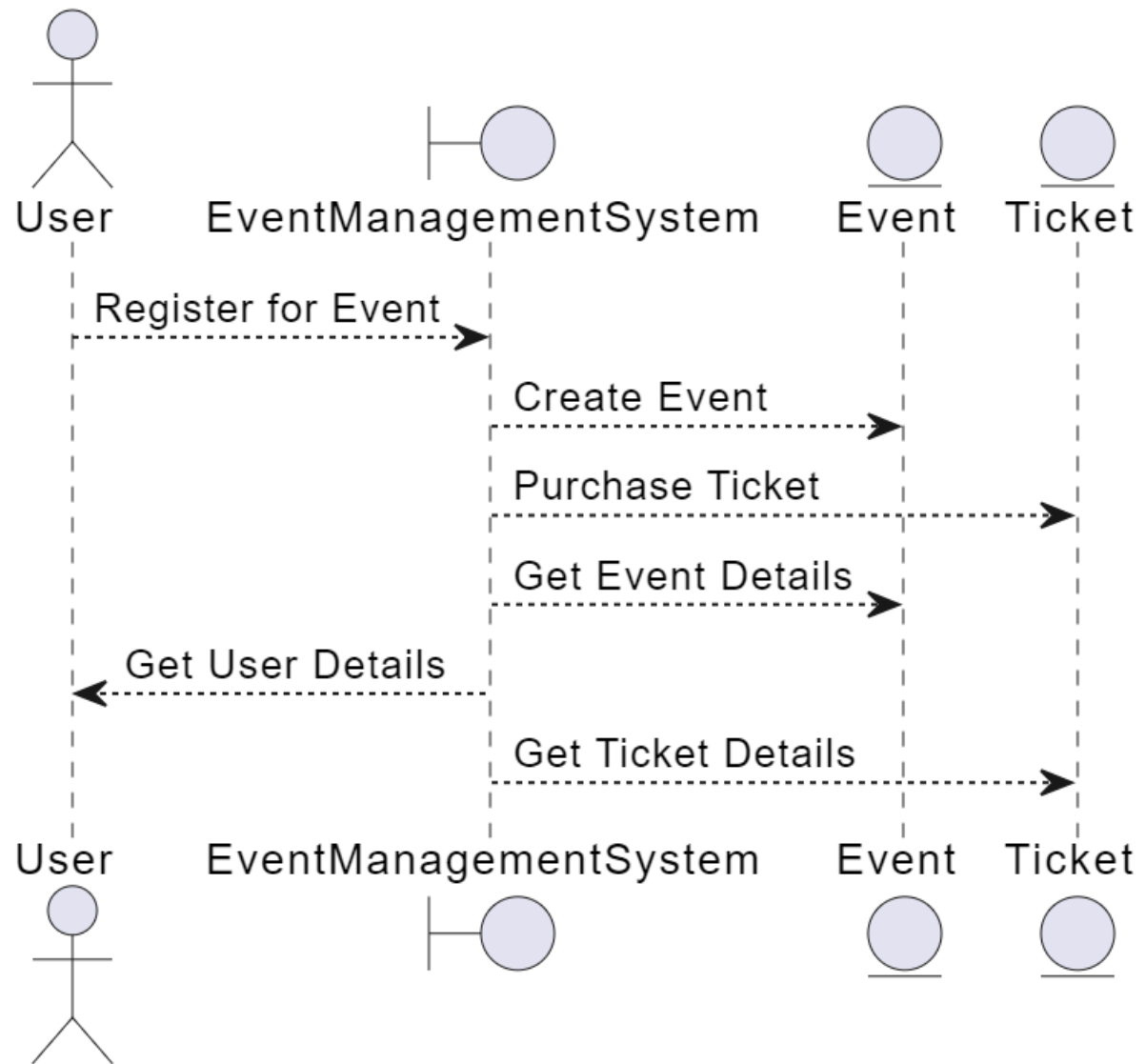
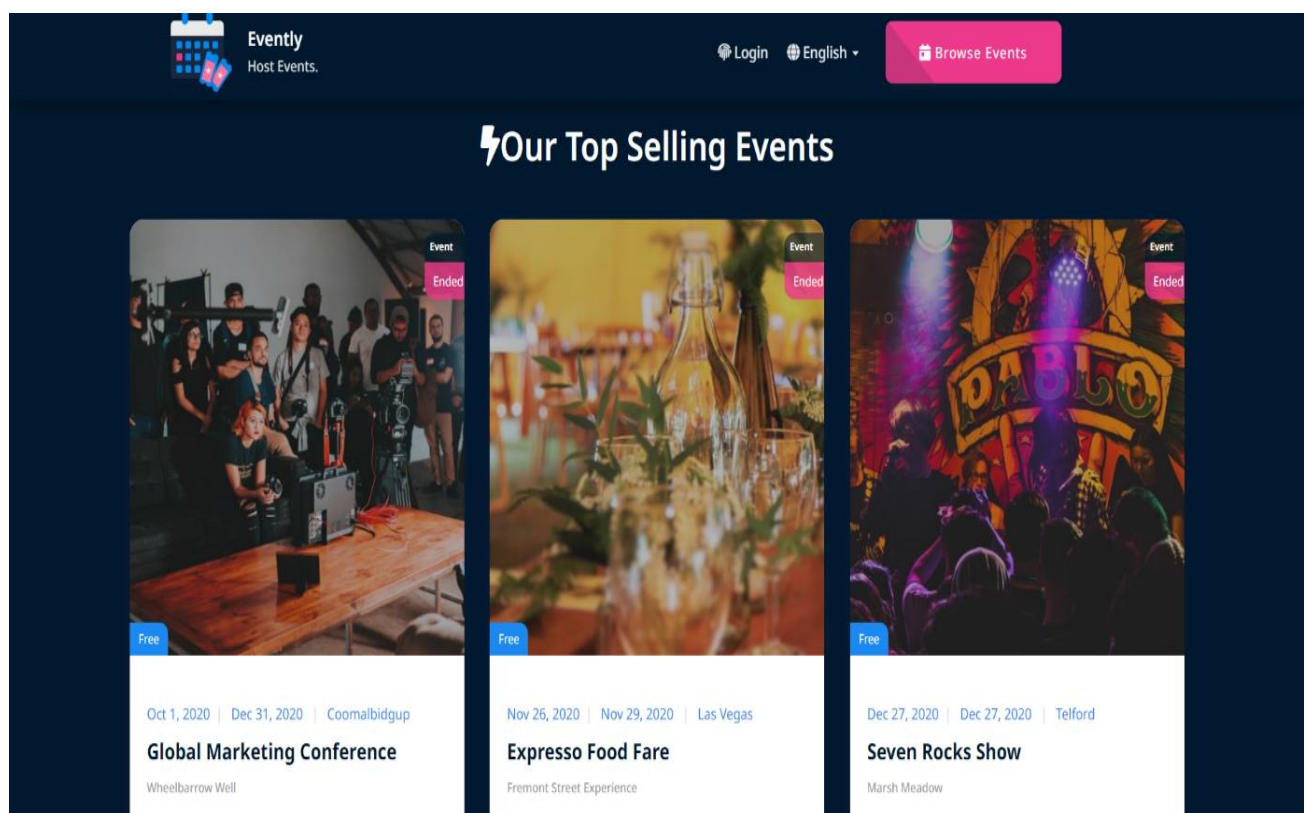
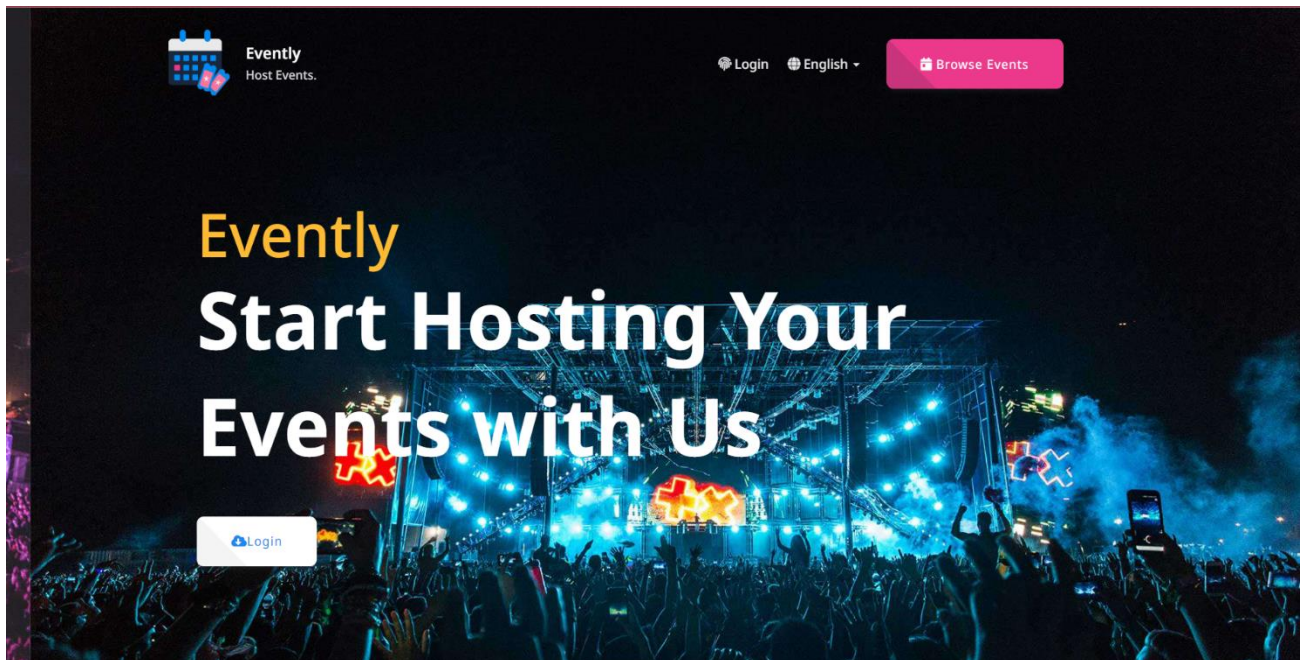
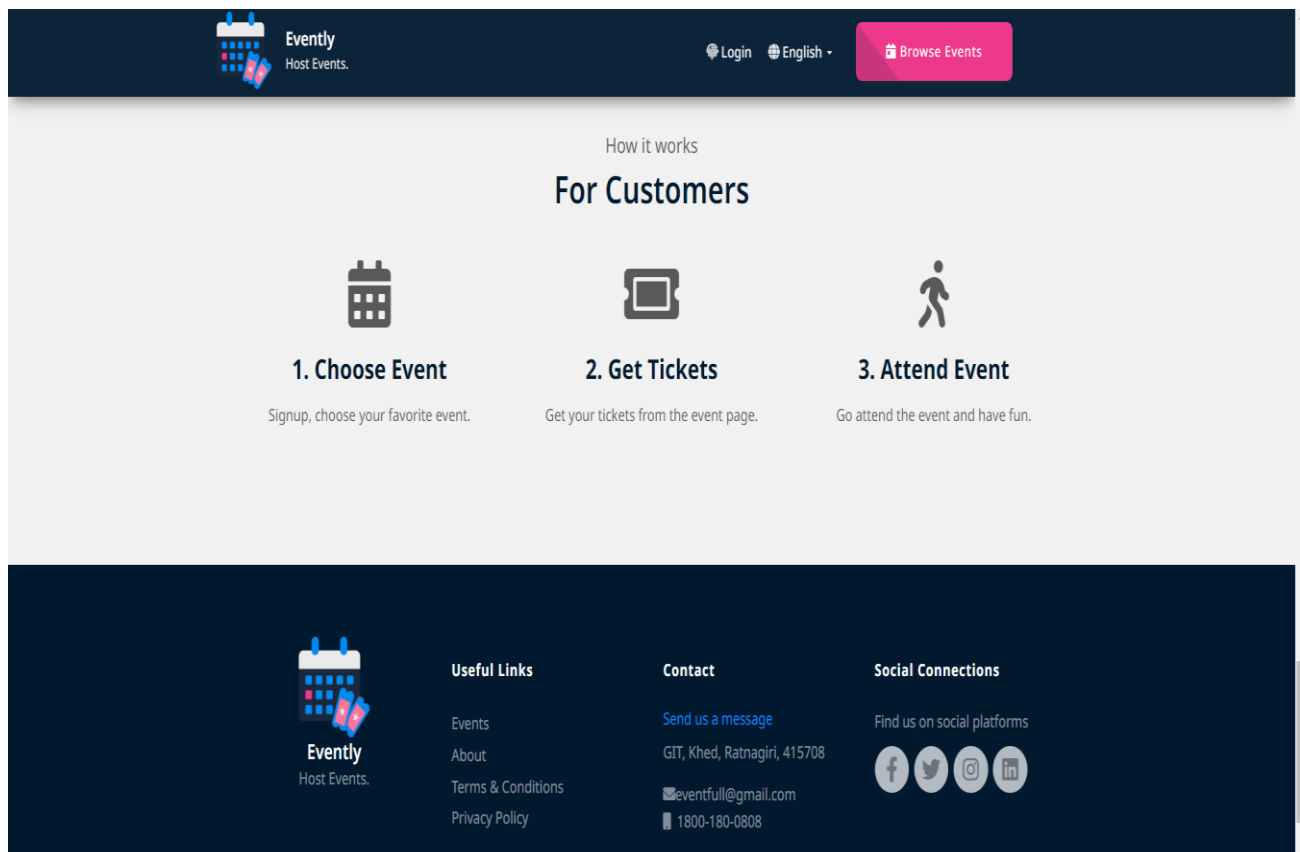
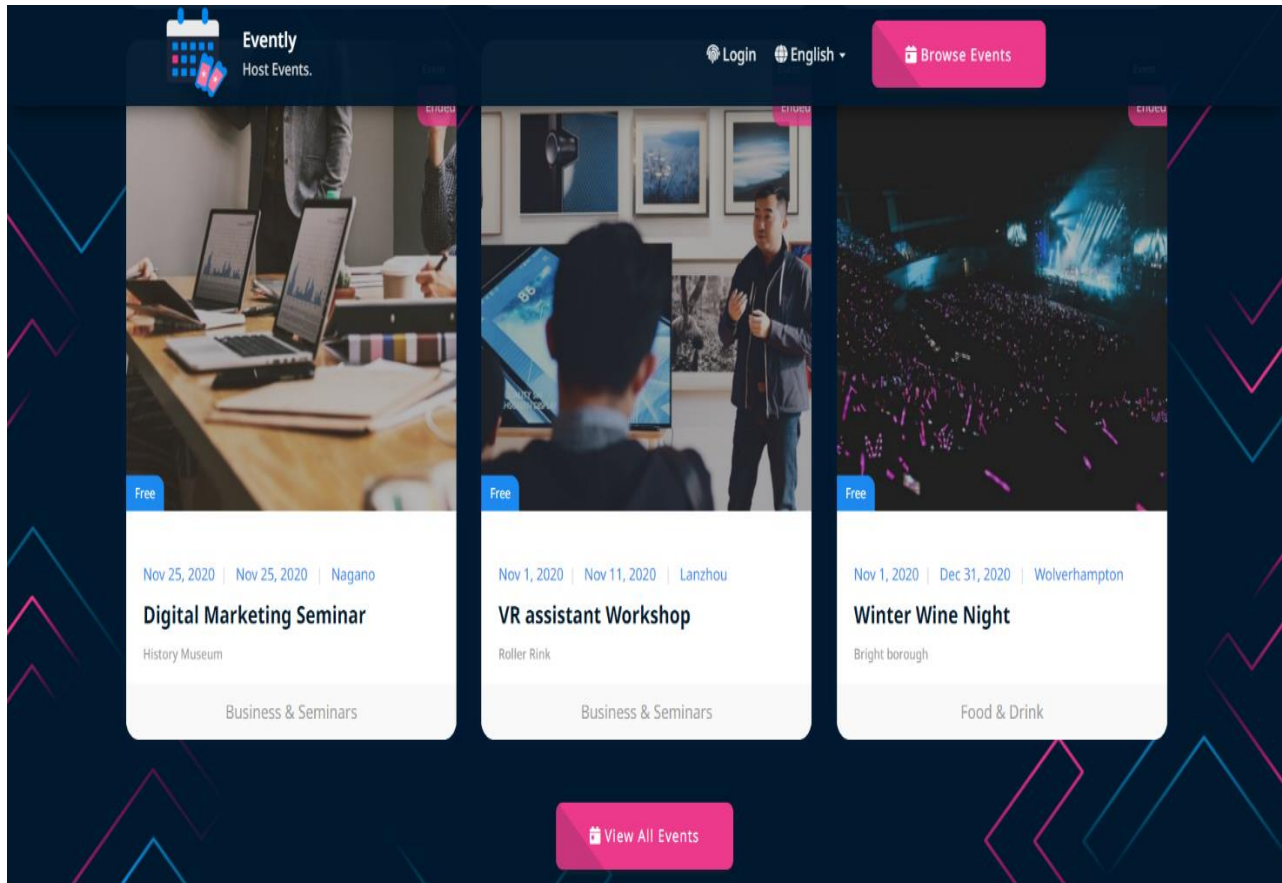


Fig 4.2: Data Flow Diagram


Chapter 5

Design and Implementation





Login



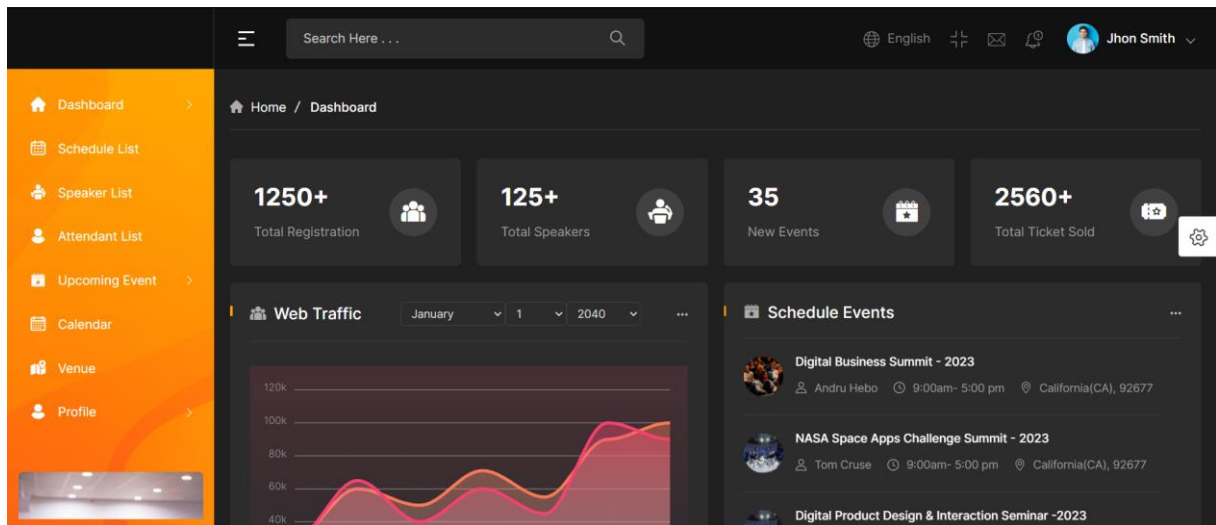
Select Role

▼

LOGIN

[Forgot password?](#)

[Signup as Admin?](#)



Chapter 6

Source Code

- **App.js**

```
import './App.css';
import './components/Fontawesomeicons';
import Home from './components/Home/Home';
import Contact from './components/Contact/Contact';
import Services from './components/Services/Services';
import About from './components/About/About';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import { Login } from './components/Login/Login';
import { Forgot } from './components/Forgot/Forgot';
import { AdminSignup } from './components/AdminSignup/AdminSignup';
import { PageNotFound } from './components/PageNotFound/PageNotFound';
import Navbar from './components/Navbar/Navbar';
import { ToastContainer } from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
import NewPassword from './components/NewPassword/NewPassword';
import { AdminDashboard } from './components/AdminDashboard/AdminDashboard';
import Statistics from './AdminPages/Statistics/Statistics';
import ManageStaff from './AdminPages/ManageStaff/ManageStaff';
function App() {
  return (
    <>
    <Router>
    <Routes>
    <Route exact path="/" element={<Navbar />} >
    <Route exact path="/" element={<Home />} />
  )
}
```



```

        <Route path="/contact" element={<Contact />} />

        <Route path="/services" element={<Services />} />
        <Route path="/about" element={<About />} />
        <Route path="/login" element={<Login />} />
        <Route path="/admin-signup" element={<AdminSignup />} />
        <Route path="/forgot" element={<Forgot />} />
        <Route path="/change-password" element={<NewPassword />}
/>

        <Route path="*" element={<PageNotFound />} />
    </Route>
    <Route path="/admin-dashboard/*" element={<AdminDashboard
/>}>

        <Route path="" element={<Statistics />} />
        <Route path="iam-service" element={<IAM />} />
        <Route path="staff-management" element={<ManageStaff />}
/>

    </Route>

    <Route path="" element={<FacilityStats />} />
    </Route>
</Routes>
</Router>
<ToastContainer hideProgressBar/>
</>
);
}

export default App;

```

- **Navbar.js**

```
import React, {useRef} from 'react';
import './navbar.css';
import { FontAwesomeIcon } from "@fortawesome/react-
fontawesome";
import {
  Link,
  NavLink,
  Outlet
} from "react-router-dom";
import Footer from '../Footer/Footer';

export default function Navbar() {
  const sidemenuRef = useRef(null);
  const barRef = useRef(null);
  const logoRef = useRef(null);

  const openmenu = () => {
    sidemenuRef.current.style.right = "0";
    barRef.current.style.display = "none";
    logoRef.current.style.margin = "auto";
  }

  const closemenu = () => {
    sidemenuRef.current.style.right = "-500px";
    barRef.current.style.display = "block";
    logoRef.current.style.margin = "0";
  }

  return(
    <>
      <div className='container' id='navContainer'>
        <nav id='nav'>
          <Link to="/"><img src='../Images/logo.svg'
ref={logoRef} alt="PSFMS" className='logo' /></Link>
          <ul id="sidemenu" ref={sidemenuRef}>
            <li><NavLink to="/"
>Home</NavLink></li>
            <li><NavLink to="./contact"
>Contact</NavLink></li>
            <li><NavLink to="./services"
>Events</NavLink></li>
```

```

        <li><NavLink to="./about"
>About</NavLink></li>
        <FontAwesomeIcon className='fa-solid
menubtn' onClick={closemenu} icon="xmark"></FontAwesomeIcon>
        </ul>
        <FontAwesomeIcon className='fa-solid
menubtn' ref={barRef} onClick={openmenu}
icon="bars"></FontAwesomeIcon>
        </nav>
    </div>
    <Outlet/>
    <Footer/>
  </>
  );
}

```

- **Login.js**

```

import React from "react";
import { Link, useNavigate } from "react-router-dom";
import './login.css';
import { FontAwesomeIcon } from "@fortawesome/react-
fontawesome";
import { useState } from "react";
import { toast } from 'react-toastify';
import { useForm } from "react-hook-form";
import axios from "axios";
const REACT_APP_SERVER_URL = process.env.REACT_APP_SERVER_URL

export const Login = () => {
  // Toggle Password
  const [type, setType] = useState('password');
  const [icon, setIcon] = useState('eye')

  const handleToggle = () => {
    if (type === 'password') {
      setIcon('eye-slash')
      setType('text')
    } else {
      setIcon('eye')
      setType('password')
    }
  }

  // Form Validation
  const { register, handleSubmit, reset, formState: {errors} } =
useForm({

```

```

    defaultValues: {
      email: '',
      password: '',
      role: 'Select Role'
    }
  });

  // Handling Errors
  toast.warn(errors.email?.message, {
    toastId: "error",
    autoClose: 1000
  });
  toast.warn(errors.password?.message, {
    toastId: "error",
    autoClose: 1000
  });

  const navigate = useNavigate();

  const onSubmit = async (data) => {
    const {role} = data;
    if(data.role === "Select Role"){
      toast.warn("Please select a role!", {
        toastId: "error",
        autoClose: 1000
      })
      return false;
    }else{
      data.role = role;
    }
    // Calling Admin Login API
    if (role === "Admin") {
      const {email, password} = data
      axios.post(`${REACT_APP_SERVER_URL}/admin-signin`, {
        email: email,
        password: password
      }).then(res=>{
        if(res.status === 200){
          navigate('/admin-dashboard')
          localStorage.setItem('TOKEN', res.data.token)
          localStorage.setItem('admin-email', res.data.email)
        }
        reset()
      }).catch(err=>{
        toast.warn('Invalid username or password!', {
          toastId: "error",
          autoClose: 1000
        })
      })
    }
  }

```

```

    });
  })
}

// Calling Principal Login API
if (role === "Principal") {
  const {email, password} = data
  axios.post(`${REACT_APP_SERVER_URL}/principal-signin`, {
    email: email,
    password: password
  }).then(res=>{
    if(res.status === 200){
      navigate('/principal-dashboard')
      localStorage.setItem('TOKEN', res.data.token)
      localStorage.setItem('principal-email',
res.data.email)
    }
    reset()
  }).catch(err=>{
    toast.warn('Invalid username or password!', {
      toastId: "error",
      autoClose: 1000
    });
  })
}

// Calling Staff Login API
if (role === "Staff") {
  const {email, password} = data
  axios.post(`${REACT_APP_SERVER_URL}/staff-signin`, {
    email: email,
    password: password
  }).then(res=>{
    if(res.status === 200){
      navigate('/staff-dashboard')
      localStorage.setItem('TOKEN', res.data.token)
      localStorage.setItem('staff-email', res.data.email)
    }
    reset()
  }).catch(err=>{
    toast.warn('Invalid username or password!', {
      toastId: "error",
      autoClose: 1000
    });
  })
}
}

```

```

return (
  <>
    <div className="loginContainer">
      <div className="login-card">
        <div className="head">
          <h2>Login</h2>
        </div>
        <form className="login-form" method='post'
onSubmit={handleSubmit(onSubmit)}>
          <div className="input-field">
            <input type="email" {...register('email',
{required: {value: true, message: 'Please enter email'},
pattern: {value: /^\\w+@[a-zA-Z_]+?\\. [a-zA-Z]{2,3}$/, message:
'Please enter valid email'}})} id="email" placeholder="Email"
/>

            </div>
            <div className="input-field">
              <input type={type} {...register('password',
{required: {value: true, message: 'Please enter password'},
minLength: {value: 8, message: 'Password should be at least of
8 characters'}, maxLength: {value: 12, message: 'Password
should be less than 13 characters'}})} id="password"
placeholder="Password" />
              <FontAwesomeIcon
                className="fa-solid"
                icon={icon}
                onClick={handleToggle}
              ></FontAwesomeIcon>
            </div>
            <select className="selectRole"
{...register('role')} id="role" >
              <option value="Select Role" hidden>Select
Role</option>
              <option value="User">User</option>
              <option value="Admin">Admin</option>
            </select>
            <button type="submit">LOGIN</button>
            <Link to="/forgot">Forgot password?</Link>
            <Link to="/admin-signup">Signup as Admin?</Link>
          </form>
        </div>
      </div>
    </>
  );
};

```

- **Controllers**

```
package com.eventmanagement.controllers;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.*;
import com.eventmanagement.dao.EventDao;
import com.eventmanagement.dao.AttendeeDao;
import com.eventmanagement.models.Event;
import com.eventmanagement.models.Attendee;

import javax.validation.Valid;
import java.util.List;

@Controller
@RequestMapping("/event")
public class EventController {

    @Autowired
    private EventDao eventDao;

    @Autowired
    private AttendeeDao attendeeDao;

    @GetMapping("/add")
    public String addEvent(Model model) {
        Event event = new Event();
        model.addAttribute("event", event);
        return "/event/addevent";
    }

    @PostMapping("/save")
    public String saveEvent(@Valid @ModelAttribute("event")
Event event, BindingResult result, Model model) {
        if (result.hasErrors()) {
            return "/event/addevent";
        }

        String successMsg = eventDao.saveEvent(event);
        model.addAttribute("successMsg", successMsg);
        return "success";
    }
}
```

```

    @GetMapping("/list")
    public String getEventList(Model model) {
        List<Event> eventList = eventDao.getAllEvents();
        model.addAttribute("eventList", eventList);
        return "/event/allevvents";
    }

    @GetMapping("/update/{id}")
    public String updateEvent(@PathVariable("id") Long id,
        Model model) {
        Event event = eventDao.getEventById(id);
        model.addAttribute("event", event);
        return "/event/updateevent";
    }

    @GetMapping("/delete/{id}")
    public String deleteEvent(@PathVariable("id") Long id,
        Model model) {
        String successMsg = eventDao.deleteEvent(id);
        model.addAttribute("successMsg", successMsg);
        return "success";
    }

    @GetMapping("/{eventId}/attendees")
    public String getEventAttendees(@PathVariable("eventId")
        Long eventId, Model model) {
        Event event = eventDao.getEventById(eventId);
        List<Attendee> attendees =
        attendeeDao.getAttendeesByEvent(event);
        model.addAttribute("event", event);
        model.addAttribute("attendees", attendees);
        return "/event/eventattendees";
    }
}

```

- **Models**

```

package com.eventmanagement.models;

import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;

```



```

import javax.persistence.OneToOne;
import javax.validation.Valid;
import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.Pattern;
import javax.validation.constraints.Size;

@Entity
public class Event {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long eventId;

    @NotEmpty(message = "Event Name cannot be empty.")
    @Size(min = 3, max = 50, message = "Event Name must be
between 3 and 50 characters.")
    private String eventName;

    @Pattern(regexp = "[789][0-9]{9}", message = "Phone should
start with [7,8,9] and must be 10 numbers only.")
    private String eventPhone;

    @Valid
    @OneToOne(cascade = CascadeType.ALL)
    private Address eventAddress;

    @ManyToMany
    @JoinTable(
        joinColumns = @JoinColumn(name = "eventId"),
        inverseJoinColumns = @JoinColumn(name =
"subjectId")
    )
    private List<Subject> eventSubjects;

    public Event() {
    }

    public Event(Long eventId, String eventName, String
eventPhone, Address eventAddress, List<Subject> eventSubjects)
    {
        this.eventId = eventId;
        this.eventName = eventName;
        this.eventPhone = eventPhone;
        this.eventAddress = eventAddress;
        this.eventSubjects = eventSubjects;
    }

    public Long getEventId() {
        return eventId;
    }

```

```
    }

    public void setEventId(Long eventId) {
        this.eventId = eventId;
    }

    public String getEventName() {
        return eventName;
    }

    public void setEventName(String eventName) {
        this.eventName = eventName;
    }

    public String getEventPhone() {
        return eventPhone;
    }

    public void setEventPhone(String eventPhone) {
        this.eventPhone = eventPhone;
    }

    public Address getEventAddress() {
        return eventAddress;
    }

    public void setEventAddress(Address eventAddress) {
        this.eventAddress = eventAddress;
    }

    public List<Subject> getEventSubjects() {
        return eventSubjects;
    }

    public void setEventSubjects(List<Subject> eventSubjects) {
        this.eventSubjects = eventSubjects;
    }

    @Override
    public String toString() {
        return "Event [eventId=" + eventId + ", eventName=" +
            eventName + ", eventPhone=" + eventPhone +
            ", eventAddress=" + eventAddress + ",
            eventSubjects=" + eventSubjects + " ]";
    }
}
```

Chapter 7

Conclusion & Future Work

• Conclusion

The Evently: Event Management System website represents a significant achievement in the realm of event planning and execution. It has successfully addressed critical objectives, making event management more accessible, efficient, and user-friendly. By providing event organizers with a powerful and intuitive platform, it has streamlined the creation, management, and promotion of events. Simultaneously, it has enhanced the experience for attendees, offering them a straightforward ticket booking process and access to comprehensive event information. The system's robust data management capabilities have empowered organizers to maintain attendee records, event details, and financial information securely. Moreover, its analytics and reporting tools have provided valuable real-time insights into event performance and attendee engagement, facilitating informed decision-making for future events. The system's customization and branding options have allowed each event to maintain its unique identity and align with the organizer's vision, enhancing the overall event experience. Data security has been a top priority, ensuring that user data remains protected and private.

• Future Work

Despite its current successes, the Event Management System website offers numerous opportunities for future enhancements and expansion. These prospects include the integration of payment gateways to facilitate secure online transactions, the development of mobile applications for greater accessibility, and the implementation of advanced analytics to provide deeper insights into attendee behaviour and event performance. Additionally, the system could benefit from improved user feedback mechanisms, more robust marketing and promotional tools, and enhanced user roles and permissions for organizational efficiency. The introduction of ticket scanning and check-in features, localization support, accessibility compliance, and the utilization of machine learning and AI algorithms for predictive analytics and personalization are all areas that warrant further exploration. Ensuring continuous security updates, fostering sustainability initiatives, and establishing feedback loops for platform improvements are vital for maintaining and improving the system's standing in the event planning industry. By addressing these future enhancements, the Event Management System can continue to evolve and offer even greater value to event organizers and attendees alike.