A FIELD PROJECT REPORT

on

“Event Management System”

Submitted

By

221FA04113

Mercy Sikhinam

221FA04135

Srilatha

Madira

221FA04178

Mohan Vudumula

221FA04421

Tuta Prasanth

Under the guidance of

Mr. Sk Badar Saheb Sir

Assistant Professor

SCHOOL OF COMPUTING & INFORMATICS

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

VIGNAN'S FOUNDATION FOR SCIENCE, TECHNOLOGY AND RESEARCH Deemed to be UNIVERSITY

Vadlamudi, Guntur.

ANDHRA PRADESH, INDIA, PIN-522213.

VIGNAN'S
FOUNDATION FOR SCIENCE, TECHNOLOGY & RESEARCH
(Deemed to be University) - Estd. u/s 3 of UGC Act 1956

CERTIFICATE

This is to certify that the Field Project entitled **"Event Management System"** that is being submitted by 221FA04113 (Mercy Sikhinam), 221FA04135(Srilatha Madira), 221FA04178(Mohan Vudumula) and 221FA04421 (Prasanth Tuta) for partial fulfilment of Field Project is a bonafide work carried out under the supervision of Badar Saheb Sir, Assistant Professor, Department of CSE.

Mr. Sk. Bhadar Saheb Sir

Assistant Professor
Department of CSE

Dr. S. V. Phani Kumar

HOD,CSE

VIGNAN'S
FOUNDATION FOR SCIENCE, TECHNOLOGY & RESEARCH
(Deemed to be University) - Estd. u/s 3 of UGC Act 1956

DECLARATION

We hereby declare that the Field Project entitled "**Event Management System"** that is being submitted by 221FA04113(Mercy Sikhinam), 221FA04135(Madira Srilatha) , 221FA04178 (Mohan Vudumula) and 221FA04421(Prasanth Tuta) in partial fulfilment of Field Project course work. This is our original work, and this project has not formed the basis for the award of any degree. We have worked under the supervision of BadarSha Sir, Assistant Professor, Department of CSE.

By

221FA04113(Mercy Sikhinam)

221FA04135 (Madira Srilatha)

221FA04178 (Mohan Vudumula)

221FA04421 (Prasanth Tuta)

ABSTRACT

The Event Management System is a digital platform designed to efficiently manage the entire lifecycle of an event, from initial planning to post-event analysis. With the increasing demand for organized and seamless event execution in educational institutions, corporate environments, and social gatherings, this system provides a centralized solution that simplifies the complexities involved in event coordination. Traditional event management methods rely heavily on manual processes that are time-consuming, error-prone, and difficult to scale. This system addresses these limitations by offering a robust and scalable platform that automates tasks, enhances communication, and ensures a professional experience for both organizers and participants.

At the core of the Event Management System is its ability to handle a variety of events including seminars, conferences, workshops, cultural programs, and corporate functions. The system enables users to create events by entering detailed information such as event name, type, date, time, venue, and description. Once an event is created, it is made visible to potential attendees who can then register through an intuitive user interface. This process eliminates the need for physical registration forms, reduces paperwork, and ensures that participant data is securely stored and easily retrievable. By incorporating user authentication and role-based access control, the system ensures that only authorized users can manage event content, while participants can only view or interact with the events relevant to them.

One of the key advantages of the Event Management System is its real-time communication capability. Through automated notifications via email or SMS, users are kept informed about event updates, schedule changes, or cancellations. Organizers can broadcast messages to all registered participants, ensuring timely and consistent communication. This feature not only saves time but also helps in maintaining attendee engagement and satisfaction. Additionally, reminders and alerts are sent to participants before the event begins, thereby minimizing the chances of no-shows and improving overall attendance rates.

Another significant aspect of the system is its focus on user experience. The platform is designed to be user-friendly and accessible, even for individuals with limited technical expertise. The clean and responsive interface allows users to navigate seamlessly across different sections of the application, whether they are accessing it from a computer or a mobile device. This ensures that both organizers and participants can perform their tasks with ease, enhancing the usability of the system across diverse user groups. Moreover, the system is built using modern technologies that ensure speed, security, and scalability.

Data management is also a crucial component of the Event Management System. All participant

information, including registration details, attendance status, and feedback, is stored in a secure database. Organizers can generate reports to evaluate the success of their events based on user feedback, attendance metrics, and engagement levels. These reports provide valuable insights that can be used to improve future events. The system also supports feedback forms that participants can fill out after the event, allowing organizers to gather suggestions and assess overall satisfaction. This continuous feedback loop contributes to the refinement and success of upcoming events.

Security and privacy are paramount in any system dealing with user data. The Event Management System incorporates multiple layers of security to protect sensitive information. All data transfers are encrypted, and user authentication mechanisms prevent unauthorized access. Furthermore, the system follows standard data protection policies to ensure compliance with legal and organizational regulations. Regular backups and system updates are conducted to maintain system integrity and availability.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREIVATIONS

- ADS : ANDROID DATA SYNCHRONIZATION
- SDK : SOFTWARE DEVELOPMENT KIT
- ADT : ANDROID DEVELOPMENT KIT
- API : APPLICATION PROGRAMMING INTERFACE
- AOSP : ANDROID OPEN SOURCE PROJECT
- AVD : ANDROID VIRTUAL DEVICE
- CRM : CUSTOMER RELATIONSHIP MANAGEMENT
- UI : USER INTERFACE
- JSON : JAVA SCRIPT OBJECT NOTATION
- XAMPP : WINDOWS/LINUX APACHE MYSQL PERL PHP
- PHP : HYPERTEXT PREPROCESSOR
- IDE : INTEGRATED DEVELOPMENT ENVIRONMENT
- HTML : HYPER TEXT MARKUP LANGUAGE
- JDK : JAVA DEVELOPMENT KIT
- XML : EXTENSIBLE MARKUP LANGUAGE
- WIFI : WIRELESS FIDELITY
- AVD : ANDROID VIRTUAL DEVICES
- ADB : ANDROID DEBUG BRIDGE

CHAPTER - 1

INTRODUCTION

1. INTRODUCTION

1.1 Introduction

The **Event Management System** is a web-based application designed to streamline event planning, coordination, and execution. It provides a centralized platform for event organizers, attendees, and service providers to collaborate efficiently. By integrating key features such as event scheduling, ticketing, venue management, and real-time notifications, this system enhances the overall event management experience. The system aims to automate and optimize event-related tasks, reducing manual workload while improving accuracy and efficiency.

The Event Management System is an advanced web-based solution that simplifies the process of organizing and managing events. Traditional event planning involves manual coordination of tasks, which can lead to errors, delays, and inefficiencies. This system integrates automated scheduling, online registration, real-time notifications, and attendee tracking to ensure seamless event execution. Organizers can create event listings, set ticket pricing, manage venue bookings, and track participant engagement. Attendees can register, receive notifications, and access event details through an intuitive user interface. The system also offers reporting and analytics tools to measure event success and improve future planning. By leveraging modern technology, this system provides a comprehensive and efficient approach to event management.

Effective event management is crucial for the success of any event, whether corporate, educational, or entertainment-based. Traditional event planning methods often involve multiple manual processes, leading to inefficiencies, miscommunication, and increased operational costs. An automated event management system enhances organization, ensures smooth coordination, and improves attendee experience. Key benefits include real-time event updates, simplified registrations, digital ticketing, and automated reminders. Additionally, data-driven insights help organizers measure attendance, track revenue, and evaluate overall event success. The integration of digital tools reduces human errors and allows seamless execution of events with minimal manual intervention.

1.2 Literature Survey

A literature survey is a crucial step in software development, ensuring technological feasibility, resource evaluation, and identifying potential challenges. The research involved analyzing content management systems, web development frameworks, and role-based access control mechanisms. Agile development methodology, version control systems, and modern web technologies were studied to enhance skills. Existing digital publishing platforms and academic literature on collaborative content creation were reviewed. Best practices in educational content management and workflow optimization techniques were explored. Learning resources included online courses, research papers, technical documentation, and insights from developer communities. This research provided a solid foundation for developing an efficient digital publishing tool.

- Investigated technologies, frameworks, and best practices for content management.
- Acquired skills in Agile development, version control, and UX design.

1.3 Project Background

The **Event Management System** was conceptualized to overcome the challenges and inefficiencies associated with the traditional management of institutional events. In conventional settings, the organization of technical and non-technical events involves a high degree of manual coordination — including offline registrations, paper-based approvals, miscommunication between stakeholders, and lack of real-time updates. These factors often result in delayed execution, confusion among participants, and increased administrative overhead.

With the widespread adoption of digital platforms, institutions are increasingly looking for automated solutions to manage events in a centralized and streamlined manner. The proposed system digitizes the complete event lifecycle — from event creation and approval to participant registration, scheduling, and post-event result announcements.

The system is built using the **MERN stack (MongoDB, Express.js, React.js, and Node.js)**, ensuring a **scalable, user-friendly, and efficient platform**. It integrates essential functionalities like **role-based authentication, article tracking**, making it a reliable solution for institutions and organizations aiming to streamline their magazine publication process.

1.4 Objective

Effective event management is crucial for the success of any event, whether corporate, educational, or entertainment-based. Traditional event planning methods often involve multiple manual processes, leading to inefficiencies, miscommunication, and increased operational costs. An automated event management system enhances organization, ensures smooth coordination, and improves attendee experience. Key benefits include real-time event updates, simplified registrations, digital ticketing, and automated reminders. Additionally, data-driven insights help organizers measure attendance, track revenue, and evaluate overall event success. The integration of digital tools reduces human errors and allows seamless execution of events with minimal manual intervention.

1.5 Project Description

This section provides a detailed explanation of the features and functionalities of the Online Magazine Generator application.

1) User Roles and Features

1.1. **Admin**

a) **Authentication:** Secure login with an administrative dashboard.

b) **Content Management:** Review, approve/reject events, provide feedback, and create events.

1.2 Users

a) **Access:** View and register to events

1) Technical Features

1.1. Role-based access control

1.2. Secure user authentication

1.3. Responsive web design

1.4. Content categorization

1.5. Search and filter functionality

2) Technology Stack

2.1. **Frontend:** React.js

2.2. **Backend:** Node.js, Express.js

2.3. **Database:** MongoDB

2.4. **Authentication:** JSON Web Tokens (JWT)

.

CHAPTER - 2

SOFTWARE REQUIREMENT SPECIFICATION

2 SOFTWARE REQUIREMENTS SPECIFICATION

2.2 Requirement Analysis

For the purpose of accessibility and portability, we propose to develop a web-based system using the MERN stack, compatible with modern operating systems (Windows, macOS, Linux). The system will be a web-based software allowing easy access, retrieval, and management of digital content.

Key documentation includes:

1. Problem statement
2. Data flow diagrams
3. Use case diagram
4. Other UML diagrams.

The above mentioned documents gives us diagrammatical view of the system what we are going to develop.

2.3 Problem Statement

The problem statement focuses on:

1) Developing a robust digital content management system using MERN stack

2) Creating a streamlined workflow for article submission and review

3) Implementing a secure, scalable platform for educational content publishing

4) Enabling efficient magazine compilation and distribution

2.4 Functional Requirements

1. User Registration and Authentication

- Implemented using React forms with state management
- Backend validation with Express.js
- Data storage in MongoDB
- Collects:
    - First name
    - Email address

- Password (with bcrypt encryption)

2. **Event Submission Process**

- React-based dynamic form
- MongoDB schema for Event storage
- Features:
    - Takes all the necessary fields
    - Saves as per the category Technical or non technical

3. **Content Review Mechanism**

- Express.js middleware for authentication
- React dashboard for administrators
- MongoDB aggregation for content tracking

2.5 Software Requirement Specification

1. **Purpose**

The is a **MERN stack-based Event Management System** application for creating, and viewing the events online at one place . It is structured as a **Multi Page Application (MPA)** for a seamless user experience. The backend, built with **Node.js and Express.js**, handles content submissions, authentication, and magazine compilation. **MongoDB Atlas** ensures high availability and scalability.

To enhance performance, **React.js with Redux** optimizes state management, while **RESTful APIs** ensure efficient frontend-backend communication. The system supports multiple concurrent users, enabling real-time content creation, submission tracking, and administrative management.

2. Scope of the project

The platform serves two user roles:

1. Users (Participants):

- Register and log in using institutional credentials or OTP-based login.
- Browse upcoming **Technical** and **Non-Technical** events.
- Register for multiple events.
- View registered events and event schedules in the **User Dashboard**.
- Receive email and/or in-platform notifications for event updates.

2. Admin :

- Log in via secured admin portal.
- Create and manage event categories (Technical / Non-Technical).
- Add or edit event details (title, description, date, rules, team size, etc.).
- View real-time registrations and participant data.
- Send broadcast messages to registered users.
- Generate reports and export participant lists.

3. Technologies Used

- **Frontend Technologies:**
    - React.js, React Hooks, Redux for UI development and state management.
    - Axios for API calls, Material-UIfor responsive components

- **Backend Technologies:**
    - Node.js, Express.js for RESTful API development.
    - Mongoose , JSON Web Token (JWT) for database interactions and authentication.
    - Multer for handling file uploads.
- **DataBase:**
    - MongoDB Atlas for cloud-based storage.

4. Overview

The primary goal of the Event Management System is to create a centralized online platform that allows:

- **Organizers** to create and manage technical and non-technical events,
- **Participants (Users)** to register, view schedules, and receive updates,
- **Administrators** to oversee all event-related operations including approvals, analytics, and communication.

The backend, built with **Node.js and Express.js**, provides a **RESTful API**, while **MongoDB Atlas** ensures scalable cloud-based storage. The frontend, developed using **React.js with Redux**, optimizes state management and real- time updates. Security is reinforced with **JWT-based authentication**, while **Multer** handles file uploads. The system leverages a **microservices architecture** and **cloud- native deployment** for scalability and reliability.

2.6 Software Requirements

The software interface is the operating system, and application programming interface used for the development of the software.

- Operating System - Windows XP
- Frontend Framework - React.js
- Backend - Node.js
- Database - MongoDB
- Version - 1.0
- Technologies - React,Node.js,Express,MongoDB

2.7 Hardware Requirements

CLIENT			
OPERATING SYSTEM	SOFTWARE	DISK SPACE	RAM
Any operating system	Any Web Browser	500MB	4GB

Table 2.1 Client Requirements

SERVER				
OPERATING SYSTEM	SOFTWARE	PROCESSOR	RAM	DISK SPACE
Ubuntu 12.04 LTS	Node.js MongoDB	Intel Xeon	8GB	50GB

Table 2.2 Server Requirements

2.8 Functional Requirements (Modules)

1. The **Event Management System** is a MERN stack-based Single Page Application (SPA) designed to streamline the process of managing and participating in technical and non-technical events. It consists of **two main user roles**: **Participant**, **Administrators** — each with their distinct functionalities. Student Coordinators

2. Administrators (DEO)

   Admins are responsible for **approving events, managing system content**, and **overseeing the platform**

   operations.

   a. **Review and Approve Events**
   b. **Manage User Roles and Access**
   c. **Publish Announcements and Notifications**
   d. **Generate Reports and Analytics**

3. Users

i) Browse Events
ii) Register for Events
iii) View Event Details and Schedules
iv) Access Results and Certificates

2.9 Non-Functional Requirements

1. Performance requirements:

- The user interface should load within 5 seconds on standard internet connections.
- Article submissions and content uploads should be processed within 10 seconds.
- Search queries for browsing published magazines should return results in less than 5 seconds.
- Authentication and login verification should take no longer than 7 seconds.

2. Design Constraints:

- The Online Magazine Generator shall be developed using the MERN stack (MongoDB, Express.js, React.js, Node.js).
- The system shall follow Single Page Application (SPA) principles for smooth navigation.
- The system shall be designed with a responsive UI to support various screen sizes, including desktops, tablets, and mobile devices.

3. Standards Compliance:

- The codebase shall follow standard ESLint rules for JavaScript best practices.
- The user interface shall maintain a consistent design using Material-UI or Chakra UI components.
- Security protocols like JWT authentication shall be implemented to protect user data.

4. Availability:

- The system shall have 99.9% uptime, ensuring accessibility at all times except during maintenance.
- Administrators shall be notified of scheduled maintenance in advance.

5. Portability:

- The application shall be browser-independent and work on Chrome, Firefox, Edge, and Safari.
- It shall be compatible with Windows, macOS, and Ubuntu environments.
- The platform shall support cloud-based deployment for scalability.

6. Reliability:

- The system shall ensure data integrity by maintaining backups of submitted articles and published magazines.
- In case of server failure, data shall be automatically recovered from backups stored in MongoDB Atlas.
- The application shall undergo regular security updates to prevent unauthorized access.

2.10 External Interface Requirements

1. User Interface

The Event Management System provides a user-friendly and intuitive interface designed for Participants, Administrators The layout follows a Multi Page Application (MPA) approach using React.js with Material- UI for an enhanced user experience.

- **Authentication** is managed using a **login page** with JWT-based security.

- The application is **responsive**, supporting **desktop, tablet, and mobile devices**.

2.11 Feasibility study

The feasibility study evaluates the practicality of developing and implementing the **Event Management System** by assessing various factors, including economic, technical, and behavioral aspects. The purpose is to determine whether the project should proceed further.

1. Economic Feasibility

The project is economically viable as it primarily requires access to a computer and an internet connection. The system is built using open-source technologies (React.js, Node.js, MongoDB Atlas), eliminating licensing costs. Hosting expenses can be minimized by using cloud-based services with scalable pricing models. The platform reduces administrative workload by automating content approvals and magazine compilation, leading to cost savings in terms of labor and resources.

2. Technical Feasibility

The system is technically feasible as it leverages widely used and well-supported web technologies. It requires:

- A web server for backend processing (Node.js with Express.js).
- A database server for content storage (MongoDB Atlas).
- A high-speed internet connection for seamless access and content synchronization.

3. Behavioural Feasibility

The application is user-friendly and does not require technical expertise.

- Participants can easily submit and view events through an intuitive interface.
- Administrators have a structured dashboard for managing submissions and generating events.

CHAPTER - 3

ANALYSIS & DESIGN

3 ANALYSIS & DESIGN

3.2 Introduction

1. Purpose

The Software Design Document (SDD) serves as a blueprint for the Online Magazine Generator system, providing a structured representation of the software architecture, components, and interactions. It facilitates effective communication between developers, designers, and stakeholders, ensuring a seamless implementation process.

- **Project Purpose**

The aim is to develop a centralized online portal for organizing, managing, and tracking events conducted within an institution or organization. The platform allows participants to register for events, view event details, and get real-time updates, while Admins can add, modify, and manage events along with participant data.

2. Scope

- **Document Scope**

This document outlines the high-level architecture of the Event Management System, including:

- System design and architecture, defining how different components interact.
- User interface design, explaining the visual layout and accessibility features.
- Module interactions, detailing how different modules (submission, review, and publishing) function together.

This document does not cover:

- Low-level code implementation details such as class definitions or database queries.
- Internal module processing logic, which will be addressed in implementation documentation.

- **Project Scope**

The website supports two types of users:

1. **Users (Participants):**
    1. Register and log in using institutional credentials or OTP-based login.
    2. Browse upcoming **Technical** and **Non-Technical** events.
    3. Register for multiple events.
    4. View registered events and event schedules in the **User Dashboard**.
    5. Receive email and/or in-platform notifications for event updates.
2. **Admins (Event Organizers):**
    1. Log in via secured admin portal.
    2. Create and manage event categories (Technical / Non-Technical).
    3. Add or edit event details (title, description, date, rules, team size, etc.).
    4. View real-time registrations and participant data.
    5. Send broadcast messages to registered users.
    6. Generate reports and export participant lists.

3.3 System Overview

The Event Management Website is a responsive, modular web application built using the MERN (MongoDB, Express.js, React.js, Node.js) stack. The system supports dynamic event management, role-based access, and real-time updates, accessible through any modern web browser.

3.2.1.1 Visual Studio Code

Visual Studio Code (VS Code) is the primary development environment used for coding and debugging the Online Magazine Generator. It is a lightweight, cross-platform source code editor that supports multiple programming languages, including HTML, CSS, JavaScript, and Node.js. With its rich extension ecosystem, it provides powerful tools for debugging, version control, and real-time collaboration.

3.2.1.2 Node.js and Express.js

The system backend is built using **Node.js** with **Express.js**, a lightweight and flexible web application framework. **Node.js** is an asynchronous JavaScript runtime that

enhances the **scalability and performance** of the application. **Express.js** simplifies backend development by providing built-in middleware, routing mechanisms, and template engines.

3.2.1.3 MongoDB

MongoDB is the **NoSQL database** used for storing and managing **user information, article submissions, and editorial decisions**. It allows for **flexible schema design** and efficient data retrieval, making it ideal for handling structured and semi-structured data.

3.2.1.4 React.js

React.js is used for building the frontend user interface. It enables a dynamic and responsive UI, ensuring a seamless experience for users. With component-based architecture, React allows for better code organization, maintainability, and reusability.

3.2.1.5 GitHub for Version Control

GitHub is used for version control and collaborative development. The project repository maintains a structured codebase, allowing multiple contributors to work efficiently while keeping track of changes and updates.The Online Magazine Generator follows a microservices-based architecture, ensuring scalability, modularity, and performance while providing a user-friendly interface for content creators and reviewers.

3.4 System Architecture

1. **Architectural Design**

The **Event Management System** is designed as a **web-based** application that allows users to view and register to the events mentioned in the website. It follows a **three-tier architecture** consisting of **a front-end user interface, a business logic layer, and a database layer**. This architecture ensures scalability, maintainability, and security.

Attributes of a well-built web application architecture:

- **Solves business problems** by providing an easy-to-use platform for article management.

- **Ensures a responsive UI** with a clean and intuitive layout.
- **Supports analytics** to track a subeventissions, reviews, and engagement.
- **Implements security** measures like authentication and access control.
- **Logs errors efficiently** to improve debugging and maintenance.

2. Components of Web Application Architecture

The **Event Management System** consists of the following key components:

UI/UX components

These elements ensure a smooth user experience and include:

- User Dashboard
- Admin DashBoard
- Adding new Events
- Updating and deleting existing events

Structural Components

- Frontend (Client-Side): Developed using React.js, it provides a dynamic, responsive, and user-friendly interface.
- Backend (Server-Side): Built using Node.js with Express.js, it handles business logic, authentication, and API requests.
- Database Layer: Utilizes MongoDB as the primary database to store user data, events and review history.

When it comes to building the components, there are several models to choose from:

- 1 web server and 1 database
- 2 web servers and 2 databases
- More than 2 web servers and databases

Web Server & Database Models

The system follows a multi-server architecture for better reliability:

1. **One Web Server & One Database (Basic Model)** – Suitable for local testing and development.
2. **Two Web Servers & One Database** – Ensures redundancy and backup.
3. **Multiple Web Servers & Databases** – Ideal for large-scale deployment, ensuring high availability and load balancing.

3. Build and Deploy to a Web App in the Cloud

The Online Magazine System is deployed to a **cloud-based web application** using **AWS or Firebase Hosting**. The deployment process involves:

1. **Building the Project**
    - The project is built using **Node (for backend) and npm (for frontend)**.

2. **Accessing the Web App**
    - Once deployed, users can access the system via the provided **domain URL**.

4. Application Components

View Layer (Presentation Layer)

- Displays Events, submission forms, dashboards, and notifications.
- Developed using React.js, HTML, CSS, and Bootstrap.
- Interacts with the backend through RESTful APIs.

Business Logic Layer

- Handles user authentication, article validation, and review workflows.
- Implements role-based access control (RBAC).
- Developed using Node.js with Express.js.

Data Access Layer

- Manages database queries, CRUD operations, and indexing.
- Uses MongoDB with Mongoose ORM for efficient data handling.

Error Handling, Security & Logging

- JWT Authentication secures user sessions.
- Input validation prevents SQL Injection and XSS attacks.
- Error logging using Winston Logger & Cloud Monitoring.

5. Overall Software Architecture

The **Event Management System** follows a **microservices-based JavaScript =architecture** where:

1. **Frontend (React.js)** interacts with the **Backend (Node.js/Express.js)** via REST APIs.
2. Backend processes **article management, authentication, and review workflows**.
3. **MongoDB database** stores articles, user details, and reviews.

The architecture shown in above figure is used in the sync operation where the data from web pages goes to web server (Apache Tomcat) to database server (MySQL). JSP is used here because of the interaction it can offer with the databases and it is easy to deploy on the XAMPP web server and here it sits in middle as shown in figure. SQL insert query is written in JSP script to insert this data into MySQL database server.

3.5 Data Design

1. Database

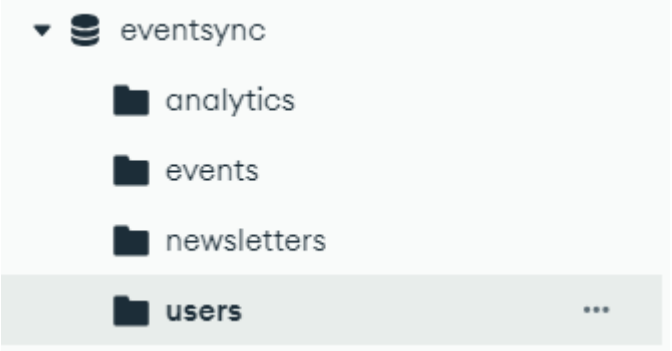

Table 3.1 Event Management Database

2. DataBase Schemas

- Analytics Schema

```
const mongoose = require('mongoose');

const analyticsSchema = new mongoose.Schema({
  eventId: { type: mongoose.Schema.Types.ObjectId,
  ref: 'Event', required: true },
  participantCount: { type: Number, required: true },
  date: { type: Date, default: Date.now },
});

module.exports = mongoose.model('Analytics',
analyticsSchema);
```

- Event Schema

```
const mongoose = require('mongoose');

const eventSchema = new mongoose.Schema({
  name: { type: String, required: true },
  description: { type: String },
  location: { type: String },
  duration: { type: String },
  startDate: { type: Date },
  endDate: { type: Date },
  picture: { type: String },
  applyLink: { type: String },
  type: { type: String, enum: ['tech', 'non-tech'],
  required: true },
  category: {
    type: String,
    enum: [
      'hackathon',
      'tech-fest',
      'contest',
      'sports', // Added for non-technical events
      'cultural', // Added for non-technical events
      'art-fair',
      'music-festival',
      'other',
    ],
    required: true,
  },
  createdBy: { type: mongoose.Schema.Types.ObjectId,
  ref: 'User', required: true },
  createdAt: { type: Date, default: Date.now },
});

module.exports = mongoose.model('Event',
eventSchema);
```

- Events table schema

```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const userSchema = new mongoose.Schema({
  name: { type: String, required: true },
  email: { type: String, required: true, unique:
  true },
  password: { type: String, required: true },
  role: { type: String, enum: ['user', 'admin'],
  default: 'user' },
  phone: { type: String },
  college: { type: String },
  profilePicture: { type: String },
  registeredEvents: [{ type: mongoose.Schema.Types.
  ObjectId, ref: 'Event' }], // New field
  createdAt: { type: Date, default: Date.now },
});

// Hash password before saving
userSchema.pre('save', async function (next) {
  if (!this.isModified('password')) return next();
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password,
  salt);
  next();
});

// Method to compare passwords
userSchema.methods.comparePassword = async function
(candidatePassword) {
  return await bcrypt.compare(candidatePassword,
  this.password);
};

module.exports = mongoose.model('User', userSchema);
```

- Users table schema

```
// Method to compare passwords
userSchema.methods.comparePassword = async function
(candidatePassword) {
  return await bcrypt.compare(candidatePassword,
  this.password);
};

module.exports = mongoose.model('User', userSchema);
```

CHAPTER - 4

MODELING

4 MODELING

4.2 Design

Requirements gathering followed by careful analysis leads to a systematic Object Oriented Design (OOAD). Various activities have been identified and are represented using Unified Modeling Language (UML) diagrams. UML is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software-intensive system under development.

4.2.1 Use Case Diagram

In the Unified Modeling Language (UML), the use case diagram is a type of behavioral diagram defined by and created from a use-case analysis. It represents a graphical over view of the functionality of the system in terms of actors, which are persons, organizations or external system that plays a role in one or more interaction with the system. These are drawn as stick figures. The goals of these actors are represented as use cases, which describe a sequence of actions that provide something of measurable value to an actor and any dependencies between those use cases.

In this application there are two actors and below is the use case diagram of this application.

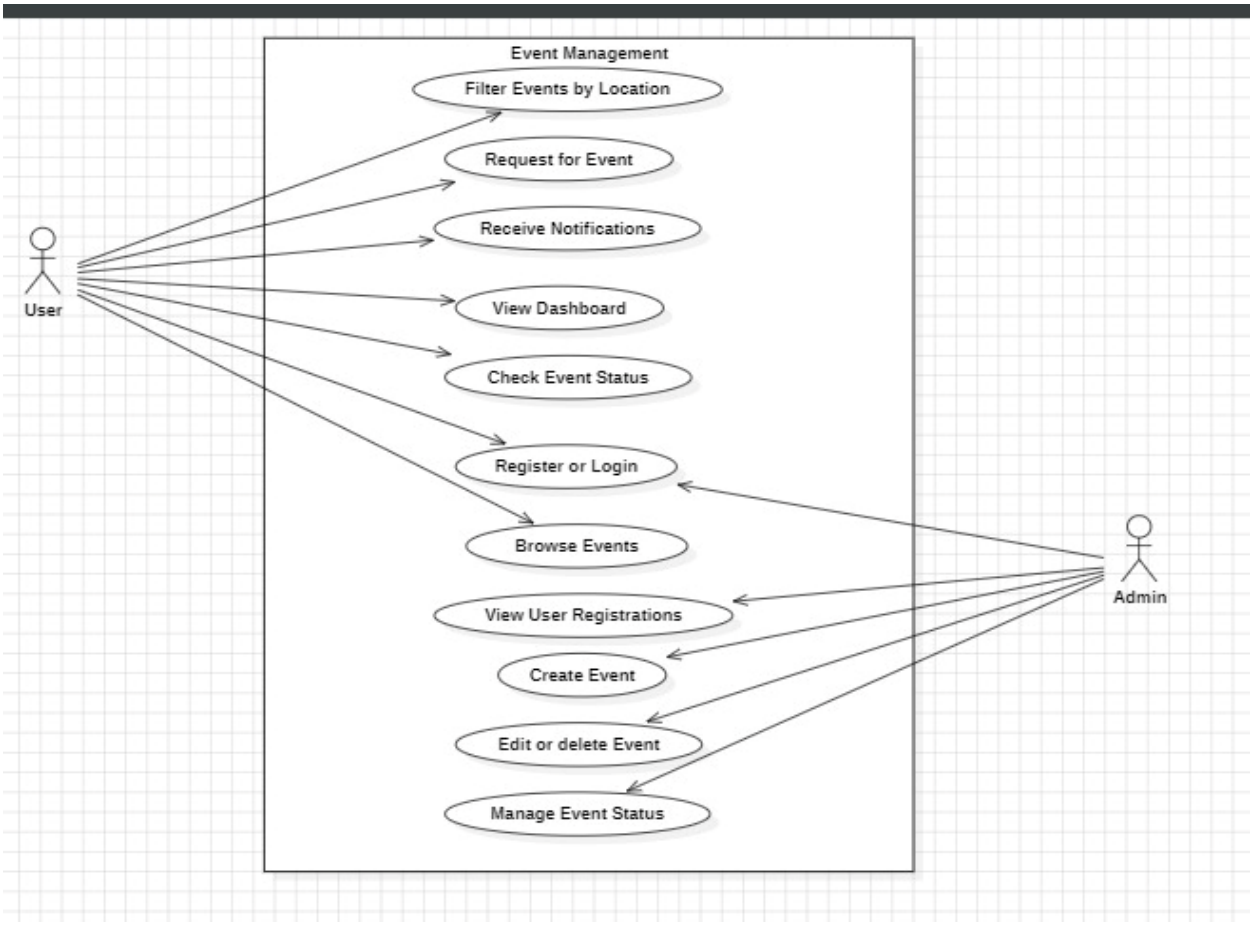

Figure 4-1 Use Case Diagram for System

4.1.2 Sequence Diagram

UML sequence diagrams are used to show how objects interact in a given situation. An important characteristic of a sequence diagram is that time passes from top to bottom: the interaction starts near the top of the diagram and ends at the bottom (i.e. Lower equals later).

A popular use for them is to document the dynamics in an object-oriented system. For each key, collaboration diagrams are created that show how objects interact in various representative scenarios for that collaboration.

Sequence diagram is the most common kind of interaction diagram, which focuses on the message interchange between a numbers of lifelines.

The following nodes and edges are typically drawn in a UML sequence diagram: lifeline, execution specification, message, combined fragment, interaction use, state invariant, continuation, destruction occurrence.

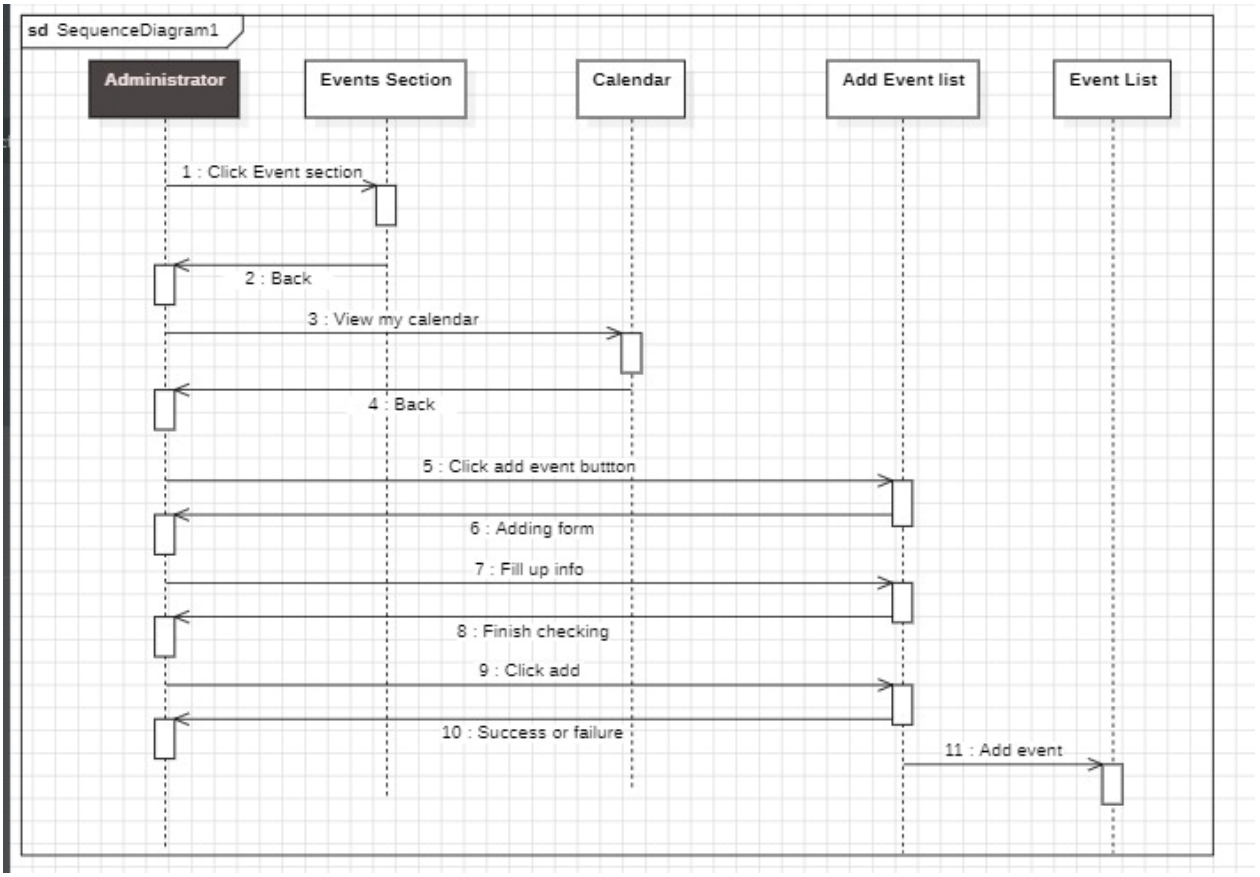

Figure 4-2 Sequence Diagram

4.1.3 Activity Diagram

Activity diagram is another important diagram in UML to describe dynamic aspects of the system. Activity diagram is basically a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system. So the control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagrams deals with all type of flow control by using different elements like fork, join etc. Activity is a particular operation of the system.

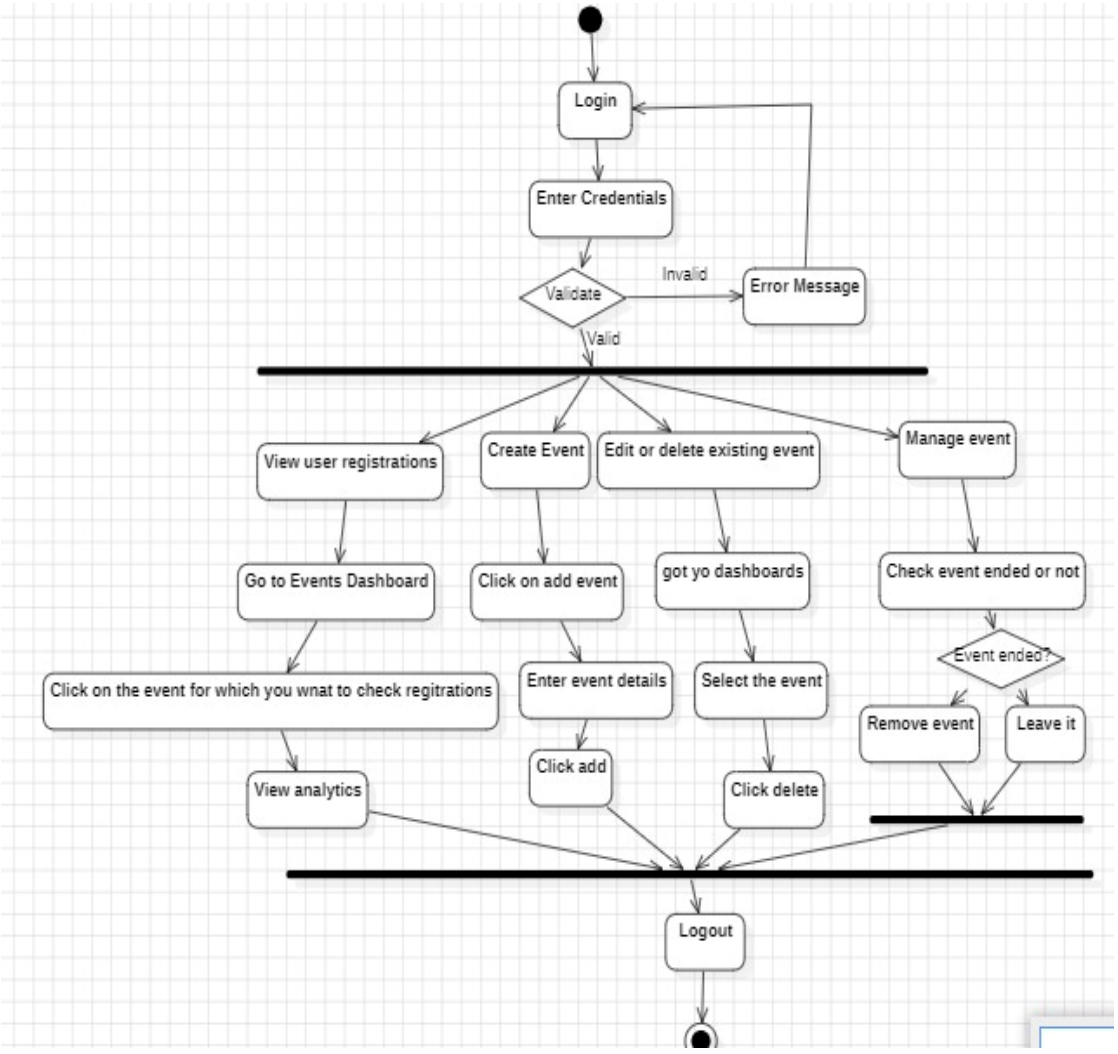

Figure 4-3 Activity Diagram for System

4.1.4 Class Diagram

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes.

The class diagram is the main building block of object-oriented Modelling. It is used both for general conceptual modelling of the application, and for detailed modelling translating the models into programming code. Class diagrams can also be used for data modelling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed.

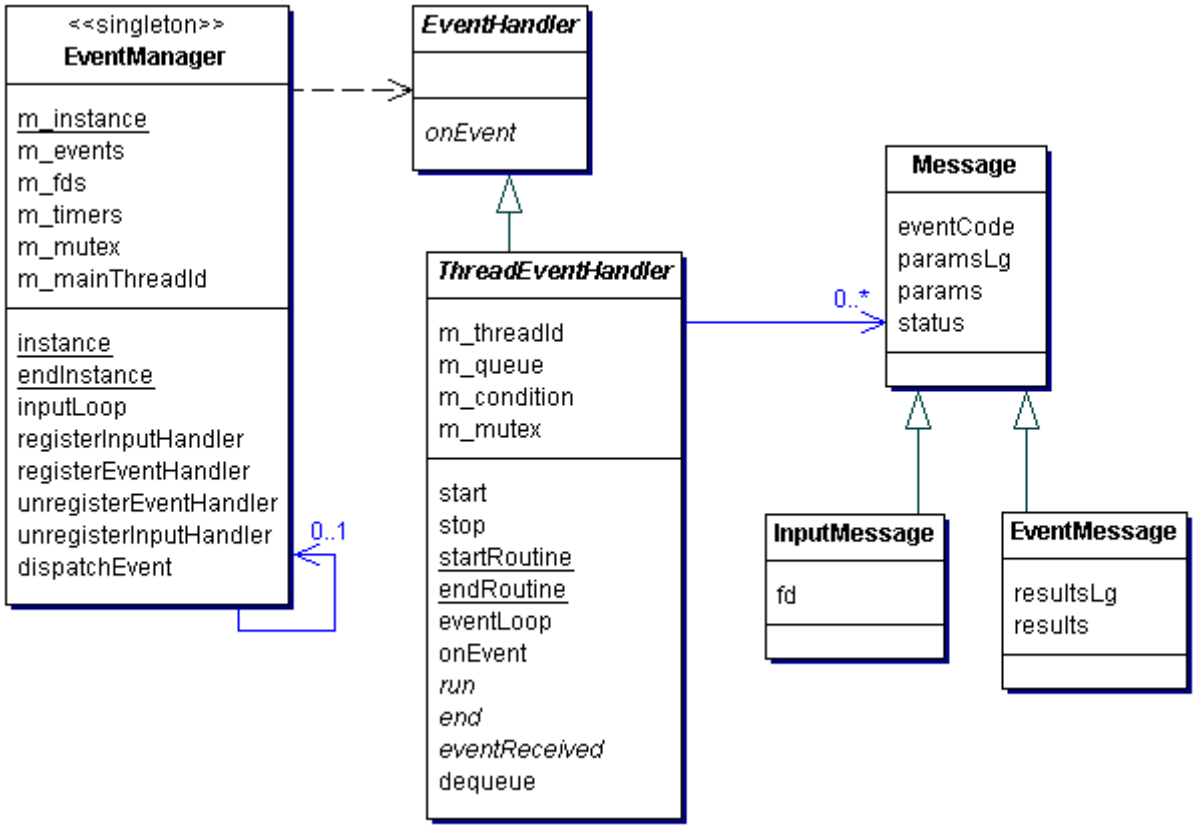

Figure 4-4 Class Diagram for CMS

4.1.5 Statechart Diagram

Statechart diagram is one of the five UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime and these states are changed by events. Statechart diagrams are useful to model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

Statechart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. The most important purpose of Statechart diagram is to model lifetime of an object from creation to termination.

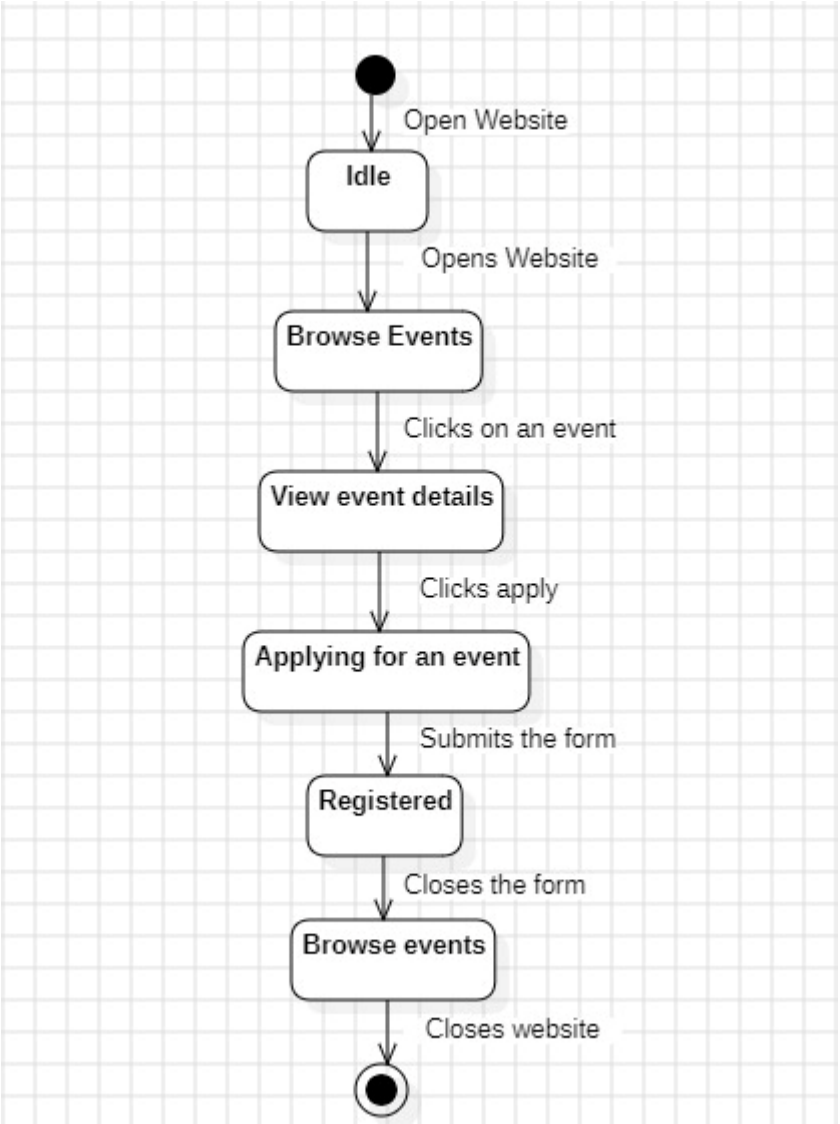

Figure 4-5 Deployment Diagram of the system

CHAPTER - 5

IMPLEMENTATION

5 IMPLEMENTATION

5.2 Sample Code

1. Code for Home Page

```
import { useState, useEffect, useRef } from 'react';
import { motion, AnimatePresence } from 'framer-motion';
import { Link, useNavigate } from 'react-router-dom';
import { FaFacebookF, FaInstagram, FaLinkedinIn, FaTwitter, FaPhone, FaArrowDown } from 'react-icons/fa';
import axios from 'axios';

function Home() {
  const [currentSlide, setCurrentSlide] = useState(0);
  const [ongoingEvents, setOngoingEvents] = useState([]);
  const [loading, setLoading] = useState(true);
  const [formData, setFormData] = useState({ name: '', email: '' });
  const [formError, setFormError] = useState('');
  const [formSuccess, setFormSuccess] = useState('');
  const eventsSectionRef = useRef(null);
  const navigate = useNavigate();

  // Hero slider images
  const slides = [
    { image: 'https://images.unsplash.com/photo-1540575467063-178a50c2df87?q=80&w=2070&auto=format&fit=crop', alt: 'Tech Event' },
    { image: 'https://images.unsplash.com/photo-1514525253161-7a46d19cd819?q=80&w=2074&auto=format&fit=crop', alt: 'Cultural Event' },
    { image: 'https://png.pngtree.com/thumb_back/fh260/back_our/20190621/ourmid/
    pngtree-blue-sky-white-clouds-silhouette-marathon-sports-poster-background-material-image_181472.jpg', alt: 'Sports Event' },
  ];

  // Autoplay for hero slider
  useEffect(() => {
    const interval = setInterval(() => {
      setCurrentSlide((prev) => (prev === slides.length - 1 ? 0 : prev + 1));
    }, 5000);
    return () => clearInterval(interval);
  }, [slides.length]);
```

2. **Code for Login**

```
import { useState, useEffect } from 'react';
import axios from 'axios';
import { useNavigate } from 'react-router-dom';
import { jwtDecode } from 'jwt-decode';

function Login() {
  const [isLoginMode, setIsLoginMode] = useState(true);
  const [formData, setFormData] = useState({ email: '', password: '', name:
  const [error, setError] = useState(null);
  const [success, setSuccess] = useState(null);
  const navigate = useNavigate();

  useEffect(() => {
    // Reset messages and form data when switching modes
    setError(null);
    setSuccess(null);
    setFormData({ email: '', password: '', name: '' });
  }, [isLoginMode]);

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    setError(null);
    setSuccess(null);

    // Basic validation
    if (!formData.email || !formData.password) {
      setError('Email and password are required');
      return;
    }
    if (!isLoginMode && !formData.name) {
      setError('Name is required for registration');
      return;
    }
```

3. Code for Admin Dashboard

```
import { useState, useEffect, useRef } from 'react';
import { motion } from 'framer-motion';
import axios from 'axios';
import { jwtDecode } from 'jwt-decode';
import Modal from 'react-modal';
import { FaCalendarAlt, FaClock, FaEdit, FaTrash, FaSearch, FaSun, FaMoon, FaPlus, FaUsers, FaCalendarCheck, FaFire } from 'react-icons
import { useNavigate } from 'react-router-dom';

// Bind modal to app element for accessibility
Modal.setAppElement('#root');

function AdminDashboard() {
  const [admin, setAdmin] = useState(null);
  const [events, setEvents] = useState([]);
  const [filteredEvents, setFilteredEvents] = useState([]);
  const [stats, setStats] = useState({ totalEvents: 0, upcomingEvents: 0, totalUsers: 0, recentEvents: [] });
  const [activeTab, setActiveTab] = useState('add-event');
  const [searchQuery, setSearchQuery] = useState('');
  const [filterStatus, setFilterStatus] = useState('all');
  const [darkMode, setDarkMode] = useState(false);
  const [newEvent, setNewEvent] = useState({
    name: '',
    type: 'tech',
    category: '',
    startDate: '',
    endDate: '',
    description: '',
    picture: '',
    applyLink: '',
  });
```

```
const [eventError, setEventError] = useState('');
const [eventSuccess, setEventSuccess] = useState('');
const [adminFormData, setAdminFormData] = useState({
  name: '',
  email: '',
  contact: '',
  profilePicture: '',
  password: '',
});
const [adminFormError, setAdminFormError] = useState('');
const [adminFormSuccess, setAdminFormSuccess] = useState('');
const [modalIsOpen, setModalIsOpen] = useState(false);
const [editEvent, setEditEvent] = useState(null);
const [deleteConfirm, setDeleteConfirm] = useState(null);
const [previewEvent, setPreviewEvent] = useState(null);
const dragRef = useRef(null);
const navigate = useNavigate();

// Fetch admin data, events, and stats on mount
useEffect(() => {
  const token = localStorage.getItem('token');
  if (!token) {
    console.log('No token found, redirecting to login');
    navigate('/login');
    return;
  }
```

```
// Fetch events and stats
const fetchData = async () => {
  try {
    // Fetch events
    console.log('Fetching events from /api/events');
    const eventsRes = await axios.get('http://localhost:5000/api/events', {
      headers: { 'x-auth-token': token },
    });
    console.log('Fetched Events:', eventsRes.data);
    const fetchedEvents = eventsRes.data;
    setEvents(fetchedEvents);
    setFilteredEvents(fetchedEvents);

    // Calculate stats
    const now = new Date();
    const totalEvents = fetchedEvents.length;
    const upcomingEvents = fetchedEvents.filter((event) => new Date(event.startDate) > now).length;
    const recentEvents = fetchedEvents
      .sort((a, b) => new Date(b.createdAt) - new Date(a.createdAt))
      .slice(0, 3);

    // Fetch total users
    console.log('Fetching users from /api/users');
    const usersRes = await axios.get('http://localhost:5000/api/users', {
      headers: { 'x-auth-token': token },
    });
    console.log('Fetched Users:', usersRes.data);
    const totalUsers = usersRes.data.length;

    setStats({ totalEvents, upcomingEvents, totalUsers, recentEvents });
  } catch (err) {
    console.error('Error fetching data:', err);
    if (err.response) {
      console.error('Response status:', err.response.status);
```

4. Code for User Dashboard

```
import { useState, useEffect } from 'react';
import { motion } from 'framer-motion';
import axios from 'axios';
import { jwtDecode } from 'jwt-decode';
import { FaCalendarAlt, FaClock, FaEdit, FaSearch, FaSun, FaMoon } from 'react-icons/fa';

function UserDashboard() {
  const [user, setUser] = useState(null);
  const [registeredEvents, setRegisteredEvents] = useState([]);
  const [filteredEvents, setFilteredEvents] = useState([]);
  const [activeTab, setActiveTab] = useState('events');
  const [searchQuery, setSearchQuery] = useState('');
  const [filterStatus, setFilterStatus] = useState('all');
  const [darkMode, setDarkMode] = useState(false);
  const [formData, setFormData] = useState({
    name: '',
    email: '',
    phone: '',
    college: '',
    profilePicture: '',
    password: '',
  });
  const [formError, setFormError] = useState('');
  const [formSuccess, setFormSuccess] = useState('');

  // Fetch user data and registered events on mount
  useEffect(() => {
    const token = localStorage.getItem('token');
    if (token) {
      const decoded = jwtDecode(token);
      setUser(decoded);
      setFormData({
        name: decoded.name,
        email: decoded.email,
        phone: '',
        college: '',
```

```
// Handle search and filter
useEffect(() => {
  let filtered = registeredEvents;

  // Search by event name
  if (searchQuery) {
    filtered = filtered.filter((event) =>
      event.name.toLowerCase().includes(searchQuery.toLowerCase())
    );
  }

  // Filter by status
  if (filterStatus !== 'all') {
    const now = new Date();
    filtered = filtered.filter((event) => {
      const start = new Date(event.startDate);
      const end = new Date(event.endDate);
      if (filterStatus === 'ongoing') return start <= now && now <= end;
      if (filterStatus === 'upcoming') return start > now;
      if (filterStatus === 'completed') return end < now;
      return true;
    });
  }

  setFilteredEvents(filtered);
}, [searchQuery, filterStatus, registeredEvents]);
```

5. Code for App.jsx

```jsx
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import { ThemeProvider } from './context/ThemeContext';
import Navbar from './components/Navbar';
import Footer from './components/Footer';
import Home from './pages/Home';
import Events from './pages/Events';
import Technical from './pages/Technical';
import NonTechnical from './pages/NonTechnical';
import About from './pages/About';
import ContactUs from './pages/ContactUs';
import Login from './pages/Login';
import AdminDashboard from './pages/AdminDashboard';
import UserDashboard from './pages/UserDashboard';

function App() {
  return (
    <ThemeProvider>
      <Router>
        <div>
          <Navbar />
          <Routes>
            <Route path="/" element={<Home />} />
            <Route path="/events" element={<Events />} />
            <Route path="/events/technical" element={<Technical />} />
            <Route path="/events/non-technical" element={<NonTechnical />} />
            <Route path="/about" element={<About />} />
            <Route path="/contact" element={<ContactUs />} />
            <Route path="/login" element={<Login />} />
            <Route path="/admin" element={<AdminDashboard />} />
            <Route path="/user-dashboard" element={<UserDashboard />} />
            <Route path="*" element={<div style={{ textAlign: 'center', padding: '2rem' }}>404 - Page Not Found</div>} />
          </Routes>
          <Footer />
        </div>
      </Router>
    </ThemeProvider>
  );
```

6. Code for NavBar

```
import { useState, useEffect, useContext } from 'react';
import { Link, useNavigate } from 'react-router-dom';
import { jwtDecode } from 'jwt-decode';
import { FaUser, FaBars, FaTimes, FaSun, FaMoon } from 'react-icons/fa';
import { ThemeContext } from '../context/ThemeContext';

function Navbar() {
  const { theme, toggleTheme } = useContext(ThemeContext);
  const [isLoggedIn, setIsLoggedIn] = useState(false);
  const [isAdmin, setIsAdmin] = useState(false);
  const [showDropdown, setShowDropdown] = useState(false);
  const [isMobileMenuOpen, setIsMobileMenuOpen] = useState(false);
  const navigate = useNavigate();

  // Function to check login state
  const checkLoginState = () => {
    const token = localStorage.getItem('token');
    if (token) {
      setIsLoggedIn(true);
      try {
        const decoded = jwtDecode(token);
        if (decoded.role === 'admin') {
          setIsAdmin(true);
        } else {
          setIsAdmin(false);
        }
      } catch (err) {
        console.error('Invalid token:', err);
        setIsLoggedIn(false);
        setIsAdmin(false);
        localStorage.removeItem('token');
      }
    } else {
      setIsLoggedIn(false);
      setIsAdmin(false);
    }
  };
```

5.3 Screen Captures

1. **home screen**

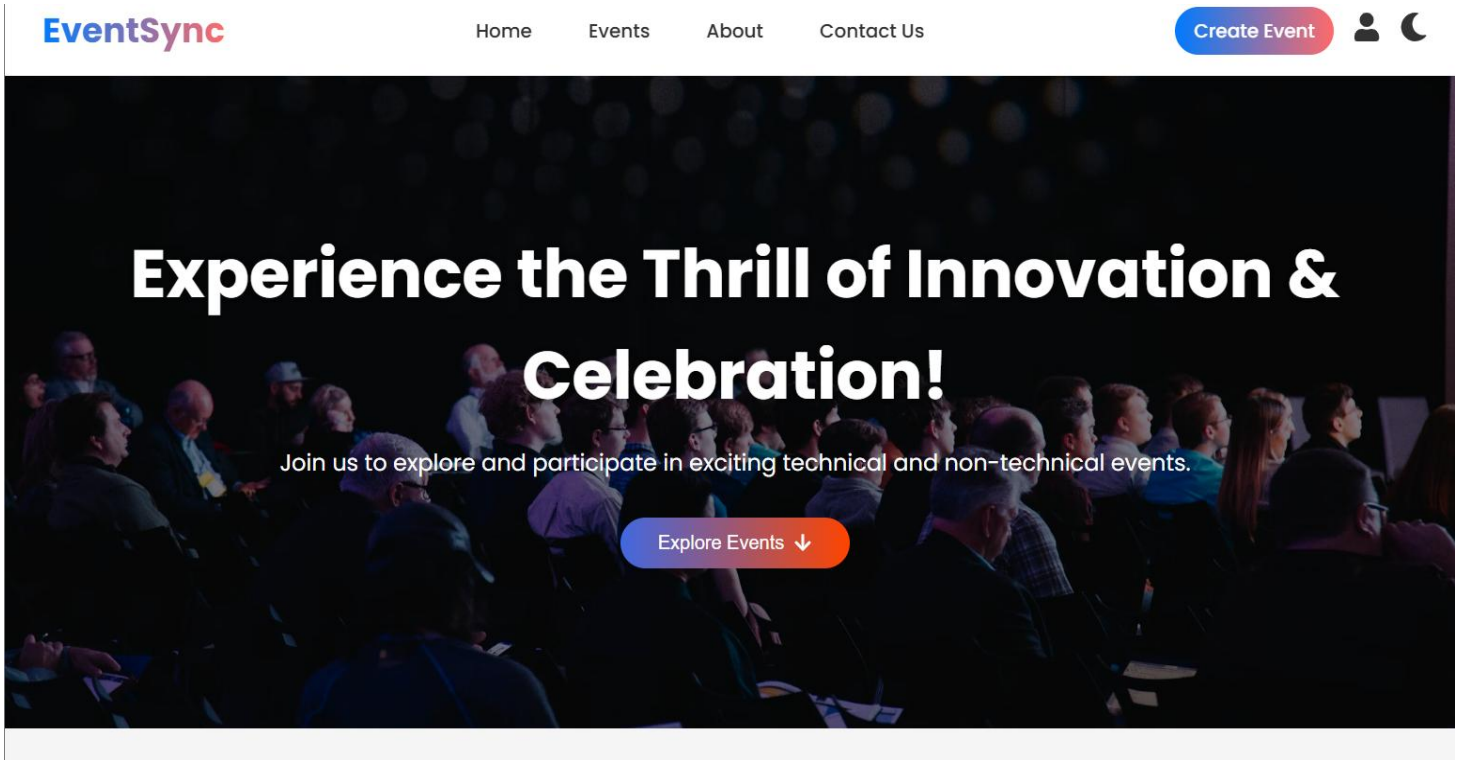

Figure 5-1 homescreen Activity

Description: After launching the application a home screen will come into view.

2. **Login Screen**

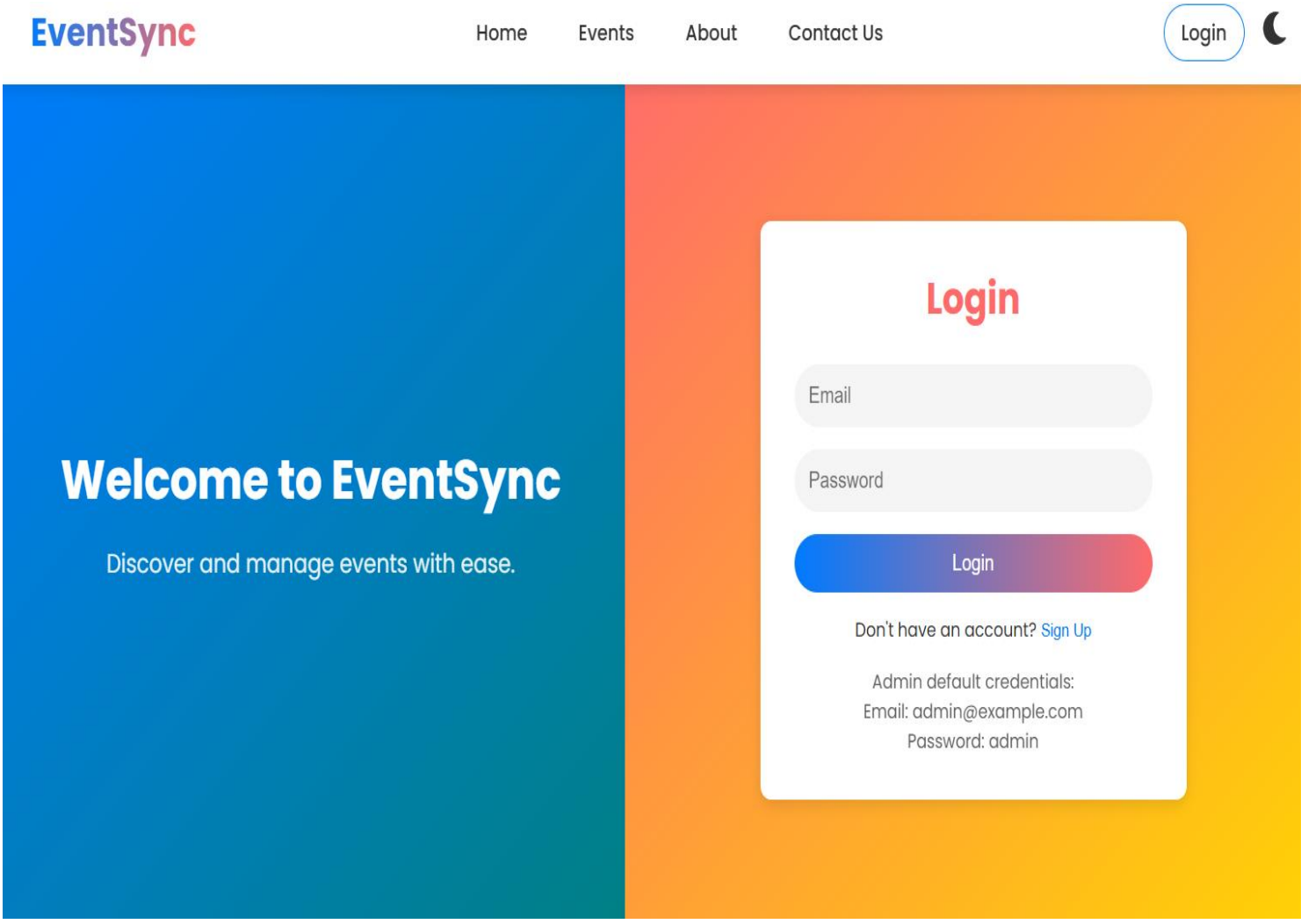

Figure 5-2 Admin and Student Login Page

Description: Displays the login interface for both Admin and Students roles in the Event management system.

3. Admin Dashboard Page

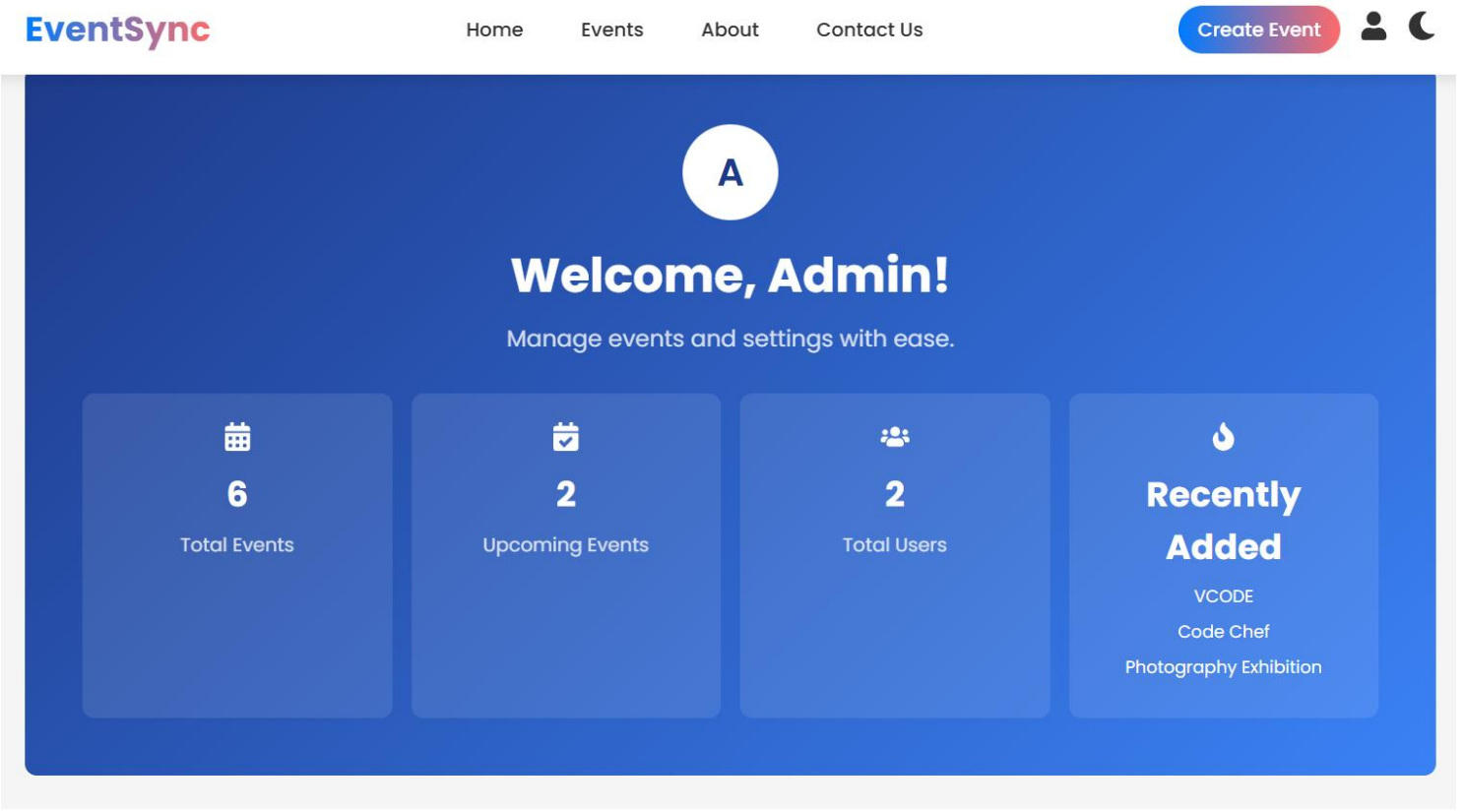

Figure 5-3 Admin Dashboard

Description: Shows the Admin dashboard where admin can add events, update and delete events and view analytics

4. **Template to add new event**

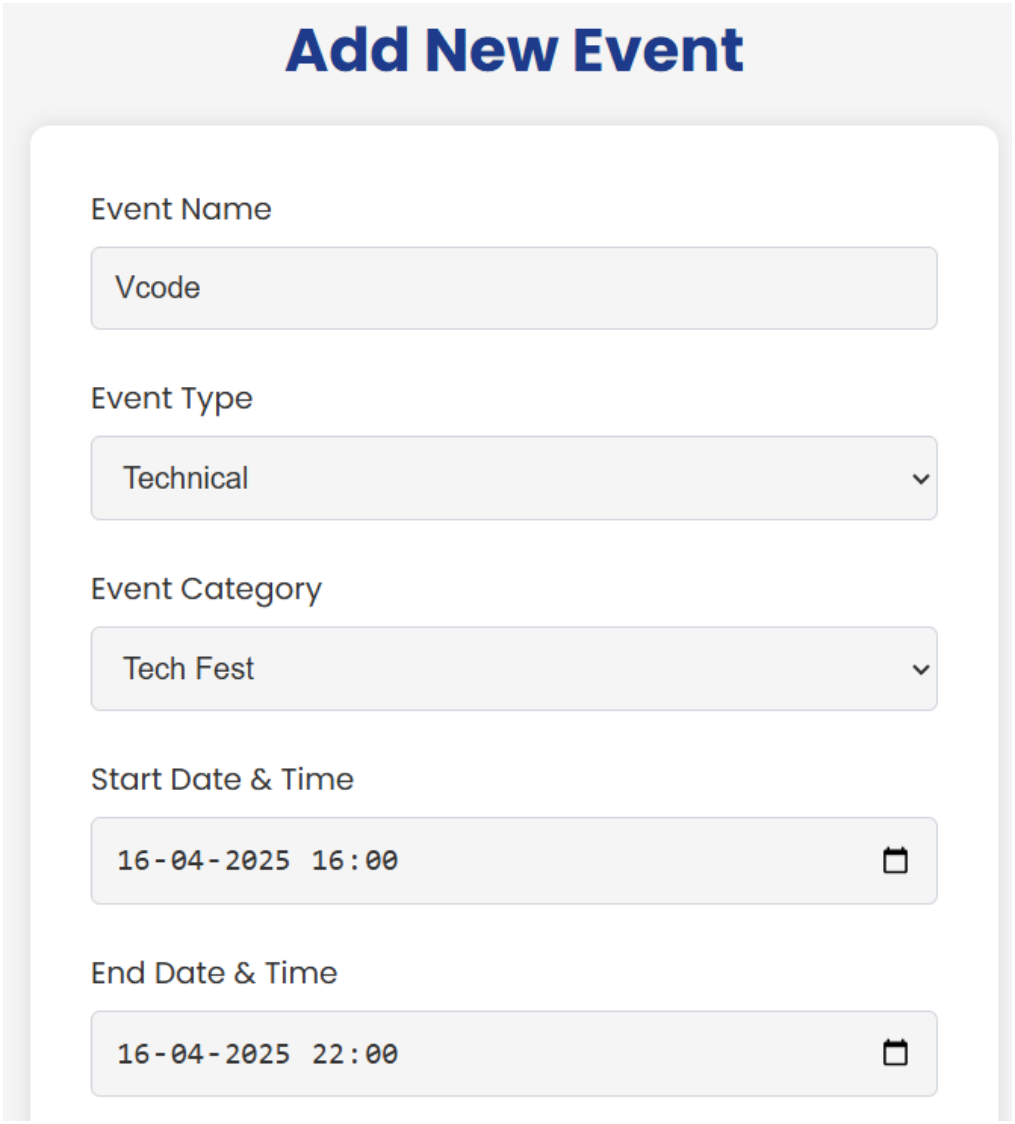

Figure 5-4 Template to add new event

Description: Here is the template where admin can add the details of events and add the event to the website.

5. **Preview Event**

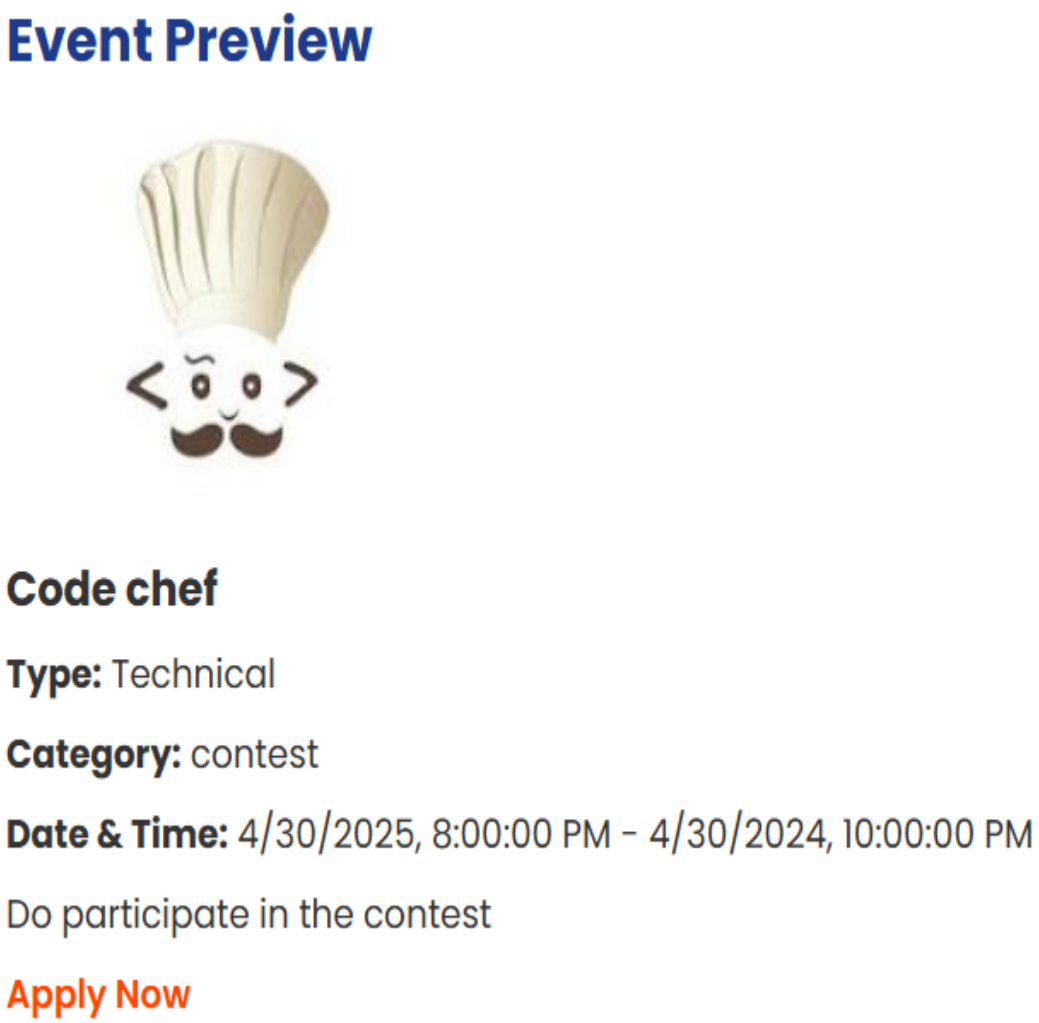

Figure 5-5 Preview Event

Description: Displays the preview of the event that is going to be added

6. **User dashboard**

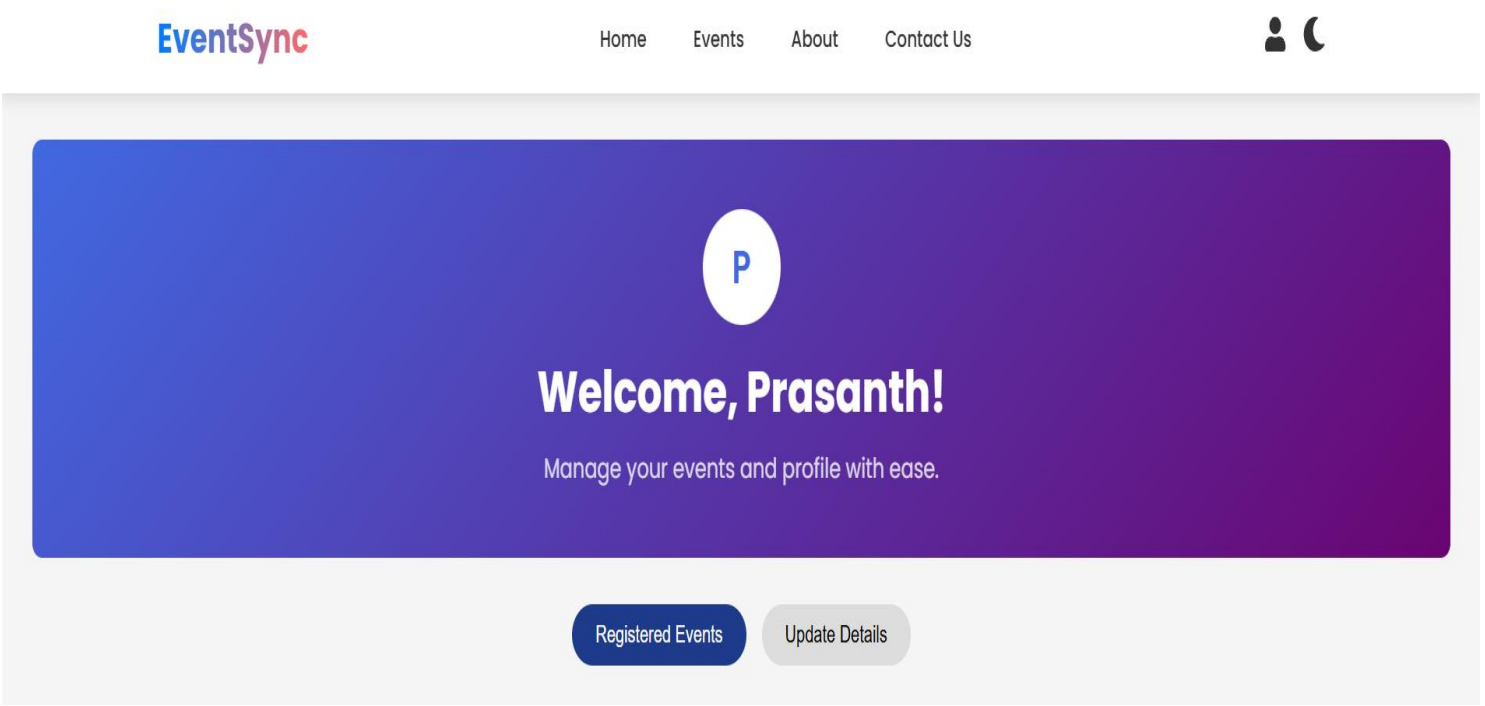

Figure 5-6 user dashboard

Description: This image shows how the user dashboard looks like

5.2.6 Footer

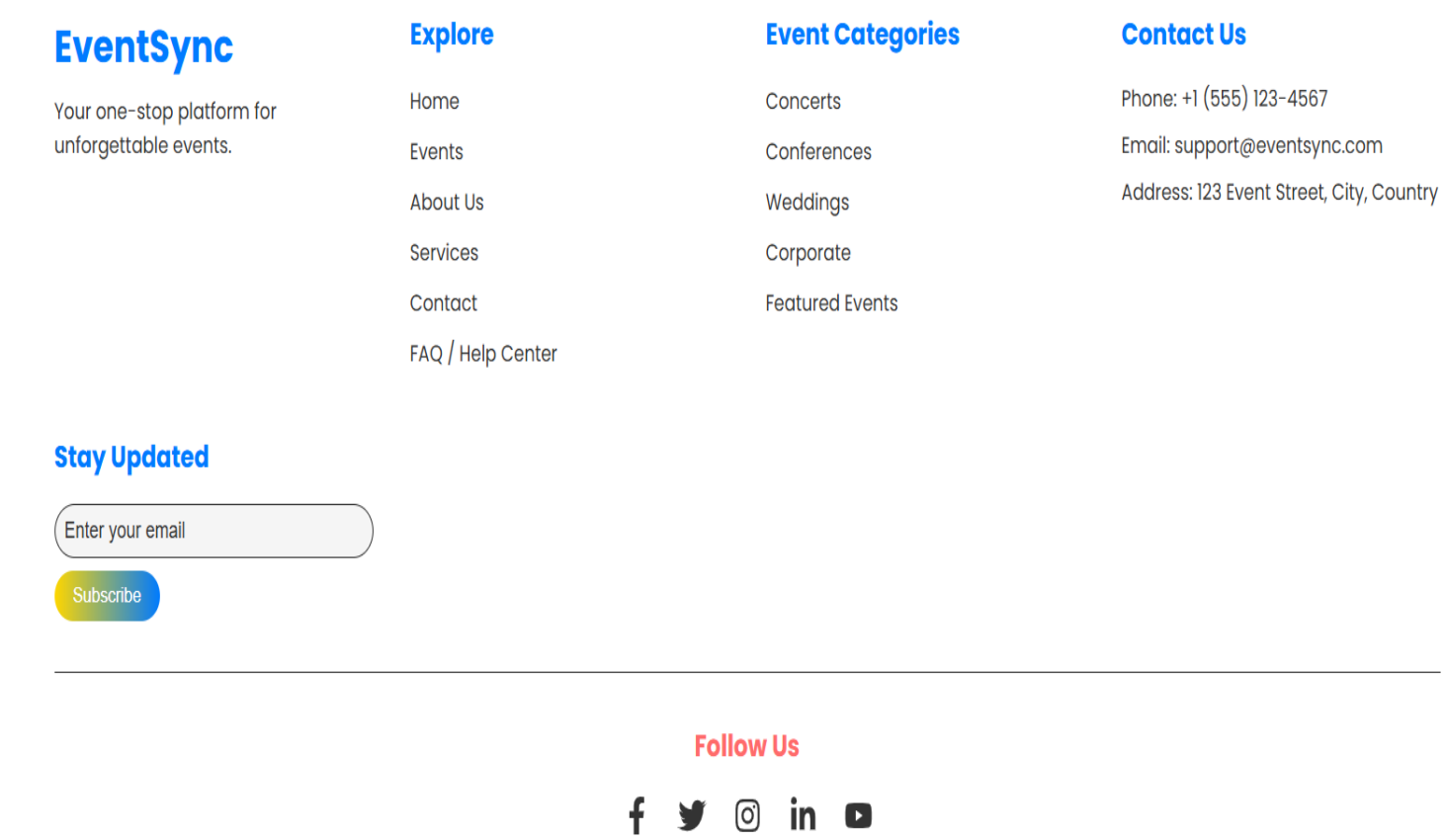

Figure 5-7 Footer Image

5.2.7 Sample Event Card

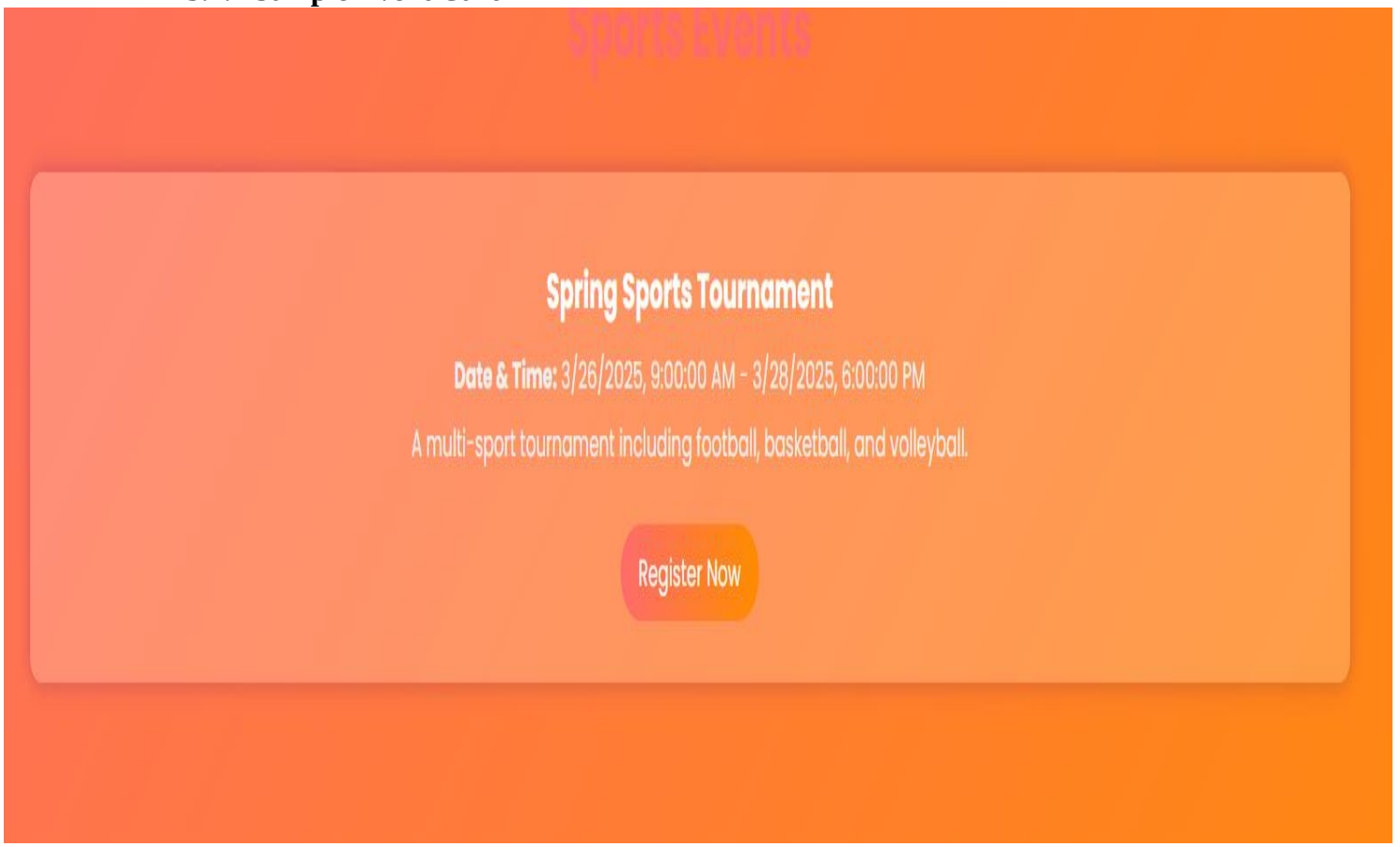

Figure 5-8 Sample Event Card

Description: This Event shows the sample event card structure

5.2.8 Non Technical Events Page

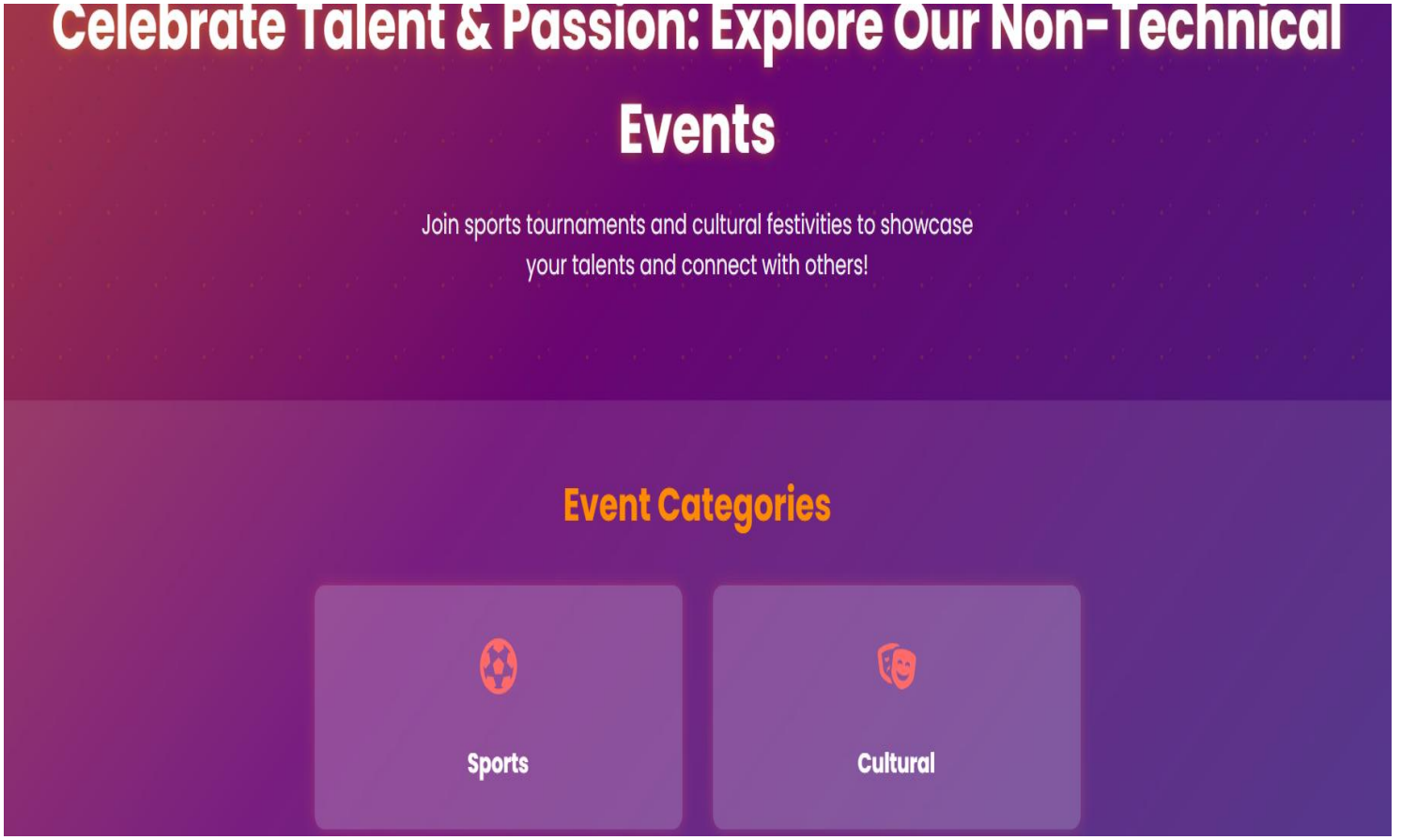

Figure 5-9 Non Technical Page

Description: This page shows the categories of non technical events.

5.2.9 Technical Events Page

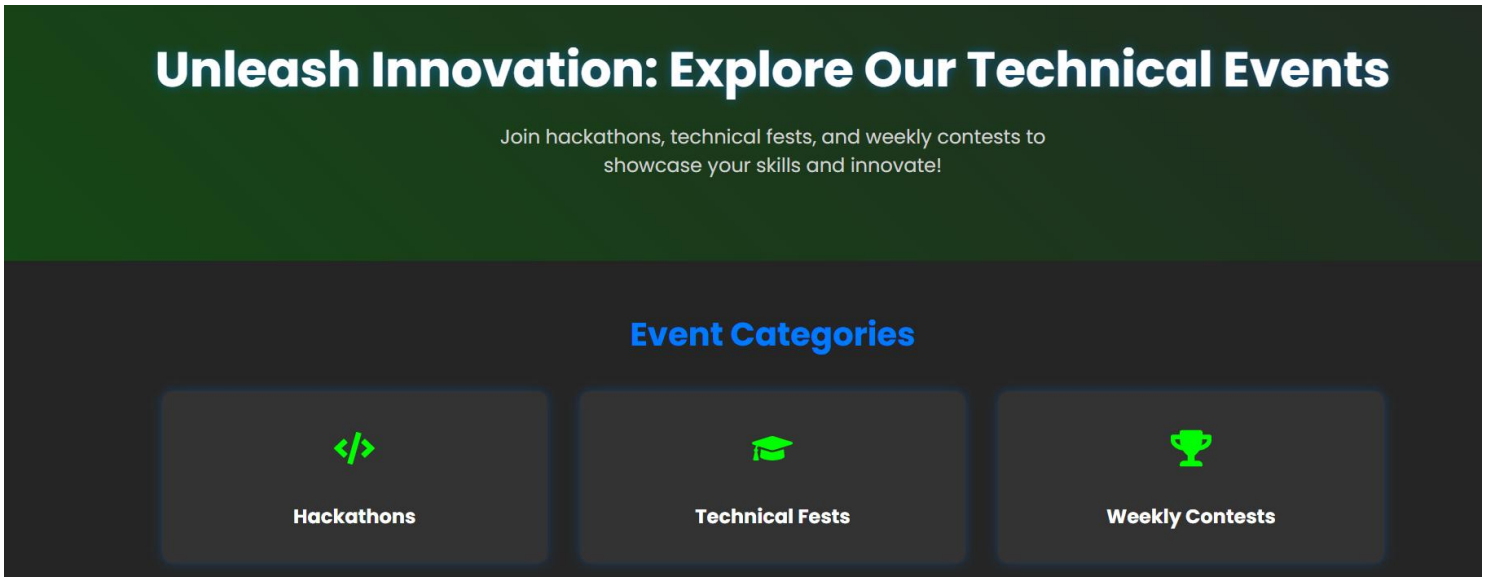

Figure 5-10 Technical Events Page

Description: This page shows the categories of technical events

CHAPTER - 6

TESTING

6 TESTING

6.2 Software Testing

Software testing is the process of validating and verifying that a software application meets the technical requirements which are involved in its design and development. It is also used to uncover any defects/bugs that exist in the application. It assures the quality of the software. There are many types of testing software viz., manual testing, unit testing, black box testing, performance testing, stress testing, regression testing, white box testing etc. Among these performance testing and load testing are the most important one for an android application and next sections deal with some of these types.

6.3 Black box Testing

Black box testing treats the software as a "black box"—without any knowledge of internal implementation. Black box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, fuzz testing, model-based testing, traceability matrix, exploratory testing and specification-based testing.

6.4 White box Testing

White box testing is when the tester has access to the internal data structures and algorithms including the code that implement these.

6.5 Performance Testing

Performance testing is executed to determine how fast a system or sub-system performs under a particular workload. It can also serve to validate and verify other quality attributes of the system such as scalability, reliability and resource usage.

6.6 Load Testing

Load testing is primarily concerned with testing that can continue to operate under specific load, whether that is large quantities of data or a large number of users.

6.7 Manual Testing

Manual Testing is the process of manually testing software for defects. Functionality of this application is manually tested to ensure the correctness. Few examples of test case for Manual Testing are discussed later in this chapter.

Test Case 1	
Test Case Name	Adding a new Event
Description	If all fields given.
Output	Event added successfull

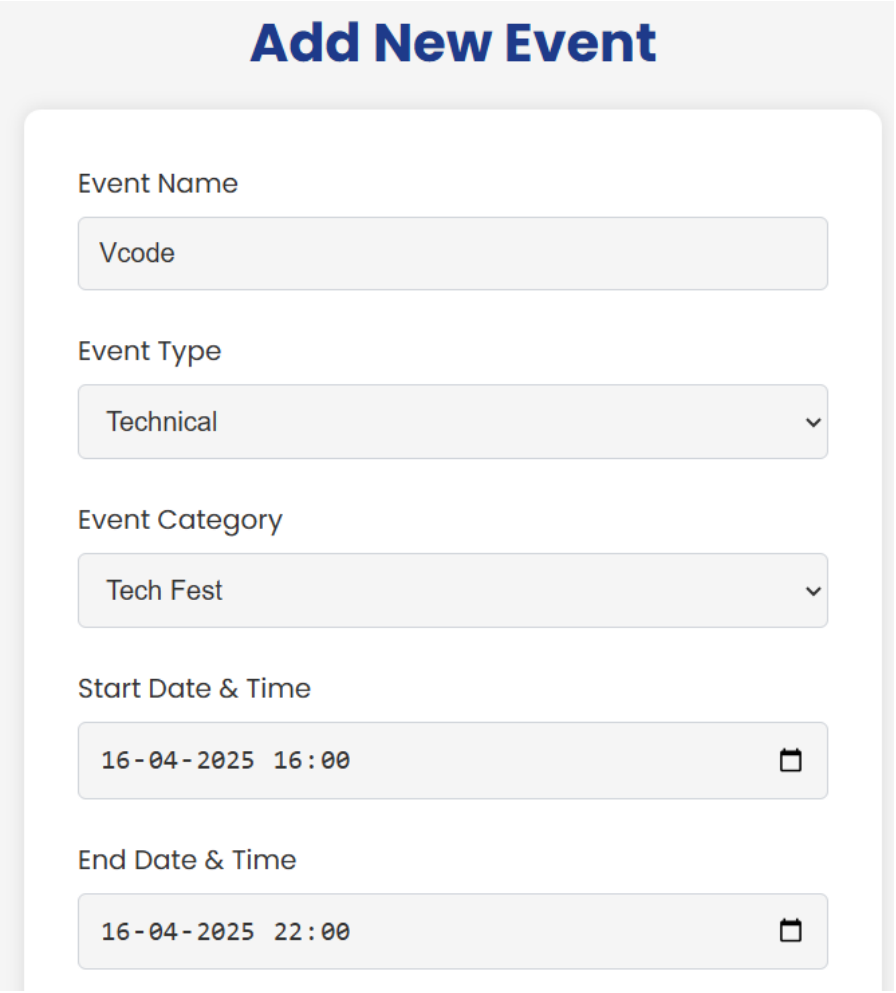

Figure 6-1 Test Case for Inserting New Event

Test Case 2	
Test Case Name	Updation of admin dashboard
Description	If new events are added or existing events are removed
Output	Increase or decrease in the count of events

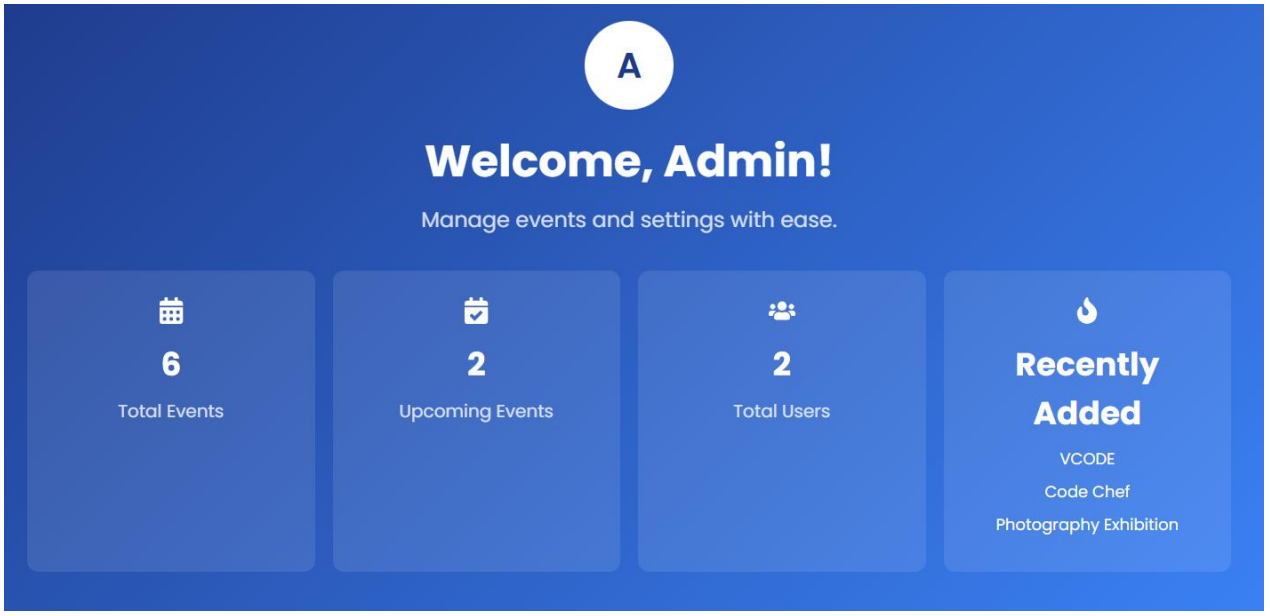

Figure 6-2 Event details updation in admin dashboard

CHAPTER - 7

RESULTS &CHALLENGES

7 RESULTS AND CHALLENGES

7.2 Results

The **Event Management System** was successfully implemented and tested, ensuring smooth event organization and attendee management. The authentication system worked efficiently, allowing secure login and registration for both event organizers and attendees. The event creation module enabled organizers to add event details, set ticket prices, and manage attendee lists seamlessly. The ticket booking system functioned as expected, providing users with a hassle-free experience while selecting seats and making payments. The dashboard for event organizers displayed **real-time statistics on ticket sales, attendee check-ins, and revenue generation**, making event management more efficient. Performance tests confirmed that the system could handle multiple concurrent users without significant slowdowns. Both **manual and automated testing** validated the correctness of the features, ensuring smooth UI navigation, error handling, and secure transactions. The system successfully generated downloadable event reports in **Excel and PDF formats** for better record-keeping. Additionally, **fraud detection mechanisms** prevented unauthorized access and duplicate ticket usage, ensuring fairness in ticket distribution. Overall, the system met its objectives by enhancing the efficiency, security, and user experience in event management.

7.3 Challenges

During the development of the **Event Management System**, several challenges were encountered and addressed. One major challenge was ensuring **secure user authentication**, as the system needed to protect personal information and prevent unauthorized access. Implementing **real-time updates** for ticket sales and event statistics required **efficient database optimization and WebSocket integration**. The **payment gateway integration** posed difficulties due to varying transaction security requirements, necessitating compliance with **PCI DSS (Payment Card Industry Data Security Standard)** regulations. **Scalability** was another challenge, as the system needed to handle large-scale events with thousands of attendees. **UI/UX design** underwent multiple iterations to ensure ease of use for both event organizers and attendees. **Handling edge cases**, such as attendees attempting **multiple ticket purchases without funds, last-minute cancellations, and refund processing**, required strict validation checks. Ensuring **device compatibility** across mobile phones, tablets, and desktops was also challenging. Lastly, maintaining **server stability under peak loads** during ticket release events was a critical challenge that was resolved through **load balancing and caching mechanisms**. Despite these challenges, continuous testing and optimizations ensured a robust and efficient system.

CHAPTER - 8

CONCLUSIONS & FUTURE WORK

8 CONCLUSION

8.2 Conclusions

The Event Management System successfully streamlined the process of event organization, ticket booking, and attendee management while improving security and user experience. The system ensures that event organizers can easily create and manage events, track ticket sales, and generate reports, reducing the need for manual record-keeping. Attendees benefit from a smooth ticket booking experience with secure payment processing and instant ticket generation. The integration of real-time dashboards allows organizers to monitor event performance, track revenue, and manage attendees effortlessly. The automated notification system ensures that users receive timely reminders about upcoming events and ticket confirmations. Security features such as encrypted user data storage, fraud detection mechanisms, and role-based access control enhance system reliability. Performance testing confirmed that the system can handle high user traffic and concurrent transactions efficiently. The user-friendly UI/UX design ensures that both organizers and attendees can navigate the platform effortlessly. By automating event planning and ticketing, the system significantly enhances efficiency, transparency, and security in event management. Future improvements can focus on AI-based event recommendations, personalized user experiences, and enhanced mobile app integration to further enhance usability.

8.3 Scope for future work

The **Event Management System** has significant potential for future enhancements to improve efficiency, security, and usability. One possible improvement is the integration of **AI-powered event recommendations**, which can suggest events to users based on their interests and past attendance. A **mobile application** can be developed to provide a more **seamless and responsive experience** for event organizers and attendees. **Offline ticket booking with automatic syncing** when the internet is available can be implemented to support users in areas with poor connectivity. **Advanced fraud detection mechanisms**, such as **facial recognition for ticket validation or QR code-based check-ins**, can be explored to enhance security. The payment system can be **expanded to include cryptocurrency payments** and **more local payment options** for global accessibility. The **ticket resale and transfer feature** can be introduced, allowing users to legally resell or transfer tickets within the platform. **Push notifications** can be added to keep attendees informed about event updates, last-minute changes, or promotional offers. **Multi-language support** can be introduced to cater to a diverse user base. Integration with **social media platforms** can enhance event promotion and increase attendee engagement. These improvements will make the system even more **robust, scalable, and user-friendly**.

8.4 Limitations

Despite its effectiveness, the **Event Management System** has certain limitations that can be addressed in future updates. One limitation is **server dependency**, as the system requires **a stable internet connection** for real-time updates and transactions. **Scalability concerns** may arise for extremely large-scale events with hundreds of thousands of attendees, requiring additional **cloud infrastructure support**. **Payment gateway restrictions** may vary across countries, affecting ticket sales in different regions. Fraud prevention remains a challenge, as users might attempt to **resell the same ticket multiple times** or use unauthorized transactions. **Refund and cancellation policies** need strict guidelines to prevent misuse. **Device compatibility issues** may occur on older smartphones or browsers that do not support advanced UI elements. **Handling last-minute bulk bookings** during high-demand events can put pressure on the server, leading to slow response times. **Data privacy concerns** related to user personal information, ticket transactions, and event details require strong security measures. Despite these challenges, the system remains a **valuable tool for automating event management, improving efficiency, and enhancing user experience**.

BIBLIOGRAPHY

[1] Meta Platforms, Inc. “React – A JavaScript library for building user interfaces” with no author [online] available at: https://reactjs.org/

[2] OpenJS Foundation “Express – Fast, unopinionated, minimalist web framework for Node.js” with no author [online] available at: https://expressjs.com/

[3] Node.js Foundation “Node.js – JavaScript runtime built on Chrome’s V8 JavaScript engine” with no author [online] available at: https://nodejs.org/

[4] MongoDB Inc. “MongoDB – The Developer Data Platform” with no author [online] available at: https://www.mongodb.com/

[5] Postman, Inc. “Postman API Platform” with no author [online] available at: https://www.postman.com/

[6] Auth0 by Okta “JWT.IO – JSON Web Tokens” with no author [online] available at: https://jwt.io/

[7] Mozilla Developer Network “Using HTTP cookies” with no author [online] available at: https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies

[8] Tailwind Labs, Inc. “Tailwind CSS – Rapidly build modern websites” with no author [online] available at: https://tailwindcss.com/

[9] puppeteer.github.io “Puppeteer Core – Headless Chrome Node API” with no author [online] available at: https://pptr.dev/

[10] Git-scm.com “Git – Local branching on the cheap” with no author [online] available at: https://git- scm.com/