ONLINE EVENT REGISTRATION SYSTEM

Project report submitted
in partial fulfillment of the requirement for award of the degree of

Bachelor of Technology
in
Computer Science & Engineering

By

SOMARAPU BHUMIKA	**(23UECS0885)**
POLA HARSHINI	**(23UECS0847)**
VADDE SINDHU	**(23UECS0908)**

10211CS212 - WEB AND MOBILE APPLICATION DEVELOPMENT

SUMMER 2025-2026

Under the guidance of
Mr.K.Prabakaran,B.Tech.,M.E.,(Ph.D).,
ASSISTANT PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING

VEL TECH RANGARAJAN DR. SAGUNTHALA R&D INSTITUTE OF SCIENCE AND TECHNOLOGY

(Deemed to be University Estd u/s 3 of UGC Act, 1956)
Accredited by NAAC with A++ Grade
CHENNAI 600 062, TAMILNADU, INDIA
November,2025

CERTIFICATE

It is certified that the work contained in the project report titled “ONLINE EVENT REGISTRATION SYSTEM” by “SOMARAPU BHUMIKA (23UECS0885), POLA HARSHINI (23UECS0847), VADDE SINDHU (23UECS0908)” has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Signature of Supervisor
Mr.K.Prabakaran
Assistant Professor
Computer Science & Engineering
School of Computing
Vel Tech Rangarajan Dr.Sagunthala R&D
Institute of Science & Technology
November, 2025

Signature of Head/Assistant Head of the Department
Dr.M.Kavitha /Dr.T.Kujani
Professor & Head/ Assoc. Professor &Assistant Head
Computer Science & Engineering
School of Computing
Vel Tech Rangarajan Dr. Sagunthala R&D
Institute of Science and Technology
November, 2025

Signature of the Dean
Dr. S P. Chokkalingam
Professor & Dean
School of Computing
Vel Tech Rangarajan Dr. Sagunthala R&D
Institute of Science and Technology
November, 2025

DECLARATION

We declare that this written submission represents my ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in our submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)
SOMARAPU BHUMIKA
Date: / /

(Signature)
POLA HARSHINI
Date: / /

(Signature)
VADDE SINDHU
Date: / /

APPROVAL SHEET

This project report entitled "ONLINE EVENT REGISTRATION SYSTEM" by SOMARAPU BHUMIKA (23UECS0885), POLA HARSHINI (23UECS0847), VADDE SINDHU (23UECS0908) is approved for the degree of B.Tech in Computer Science & Engineering.

Examiners **Handling faculty**

Mr.K.Prabakaran,B.Tech.,M.E.,(Ph.D).,

Date: / /

Place:

ABSTRACT

The online event registration system is a comprehensive platform designed to simplify the registration process for events, conferences, workshops, and meetups. By leveraging this system, event organizers can effortlessly create and manage their events, set registration fees, and track attendee registrations and payments in real-time. The system also enables organizers to send automated reminders, updates, and notifications to attendees, ensuring they stay informed about event details. For attendees, the system provides a seamless registration experience, allowing them to browse events, register, and pay online securely. They can also access event details, view their registration status, and receive confirmation emails. By automating administrative tasks, the system reduces paperwork, minimizes errors, and saves time for both organizers and attendees. Overall, the online event registration system enhances the event experience, improves organization, and increases attendee engagement.

Keywords:
Online Event Registration
Event Management
Registration Process
Event Organizers
Attendees

LIST OF FIGURES

LIST OF ACRONYMS AND ABBREVIATIONS

Acronyms	**Abbreviation**
API	Application Programing Interface
CSS	Cascading Style Sheet
HTML	Hyper Text Markup Language
JSON	Java Script On Notation
JS	Java Script TS Type Script

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Introduction

The online event registration system is a vital tool for event management, streamlining the registration process for organizers and attendees alike. With features like automated registra- tion, real-time tracking, and secure online payment, this system simplifies event planning and reduces administrative tasks. Whether it's a conference, workshop, or meetup, online event registration is the key to a successful and stress-free event experience.

1.2 Aim of the project

"The aim of this project is to design an efficient online event registration system, streamlining the registration process for events and enhancing the experience for both organizers and attendees."

1.3 Project Domain

"The project domain of the online event registration system encompasses the design, development, and deployment of a digital platform for managing event registrations. This domain focuses on creating a user-friendly interface for event organizers to create and manage events, set registration parameters, track attendee registrations, and facilitate secure online payments. The system caters to various types of events such as conferences, workshops, meetups, and more, streamlining the registration process to enhance the overall event experience for both organizers and attendees."

1.4 Scope of the Project

"The project's scope includes designing and developing an online event registration system that allows organizers to create and manage events, track registrations, and facilitate secure payments, while providing attendees with a seamless registration experience."

1.5 Methodology

“The methodology for developing an online event registration system involves a structured approach, starting with requirement gathering and analysis to identify the needs of event organizers and attendees. This is followed by designing a user-friendly interface, developing the system using suitable programming languages and frameworks, and conducting thorough testing to ensure functionality, security, and performance. The system is then deployed and maintained, with ongoing support and updates to meet evolving user needs and technological advancements. Agile methodologies may be employed to facilitate iterative development and continuous improvement.”

Chapter 2

REQUIREMENT SPECIFICATION

2.1 User characteristics

The users of an online event registration system are event organizers and attendees. Organizers create and manage events, while attendees register and participate. Users may vary in tech-savviness, age, and device usage, but the system should be easy to use, accessible, and secure for all.

2.2 Dependencies

An online event registration system relies on several key dependencies, including a secure payment gateway, a robust database to store attendee information, a user-friendly interface, automated email notifications, and strong security measures to protect user data. These components work together to ensure a smooth and secure registration process for both event organizers and attendees.

2.3 Hardware specification

The online event registration system requires a reliable web server with sufficient storage, processing power, and RAM. A dedicated database server is also necessary for storing and managing event data. Additionally, a load balancer ensures high availability, while ample storage and a fast, secure network connection support smooth user experience and data management.

2.4 Software specification

The online event registration system is a web-based platform that allows users to browse and register for events, purchase tickets, and receive automated reminders and updates. It streamlines event management for organizers by providing tools to create and manage events, track registrations, and analyze attendee data in real-time. With a user-friendly interface and secure payment processing, the system ensures a seamless experience for both organizers and attendees. By automating tasks and providing valuable insights, it helps organizers focus on delivering successful events.

Chapter 3

WEBSITE DESIGN

3.1 Sitemap

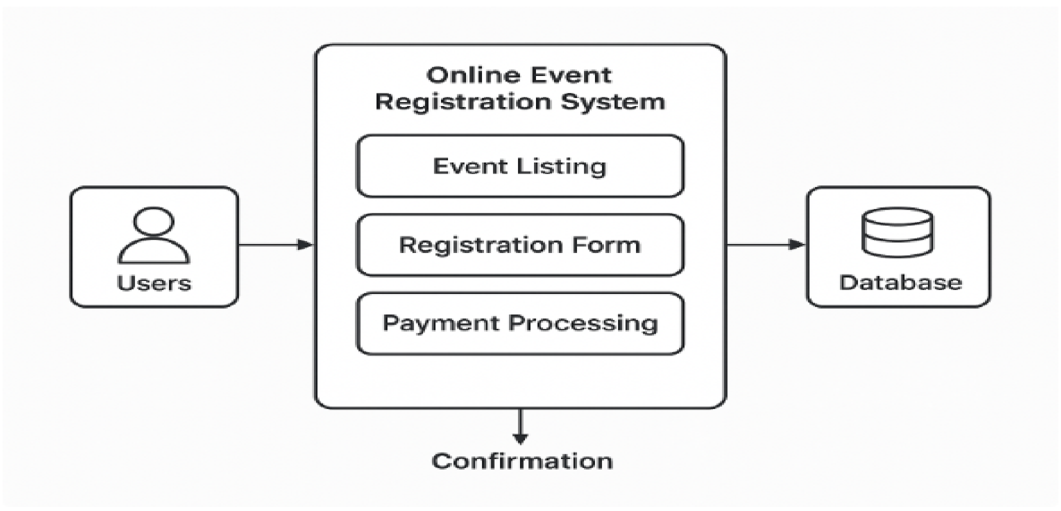


Figure 3.1: **Architecture Diagram**

The architecture diagram illustrates the flow and main components of an Online Event Registration System. It starts with Users who interact with the system to view event listings, fill out registration forms, and make payments. The Online Event Registration System module manages three core functions — Event Listing, Registration Form, and Payment Processing. These modules connect to a Database that stores event details, user information, and transaction records. Finally, after successful registration, the system sends a Confirmation back to the user, completing the process.

3.2 Design Phase

3.2.1 Data Flow Diagram

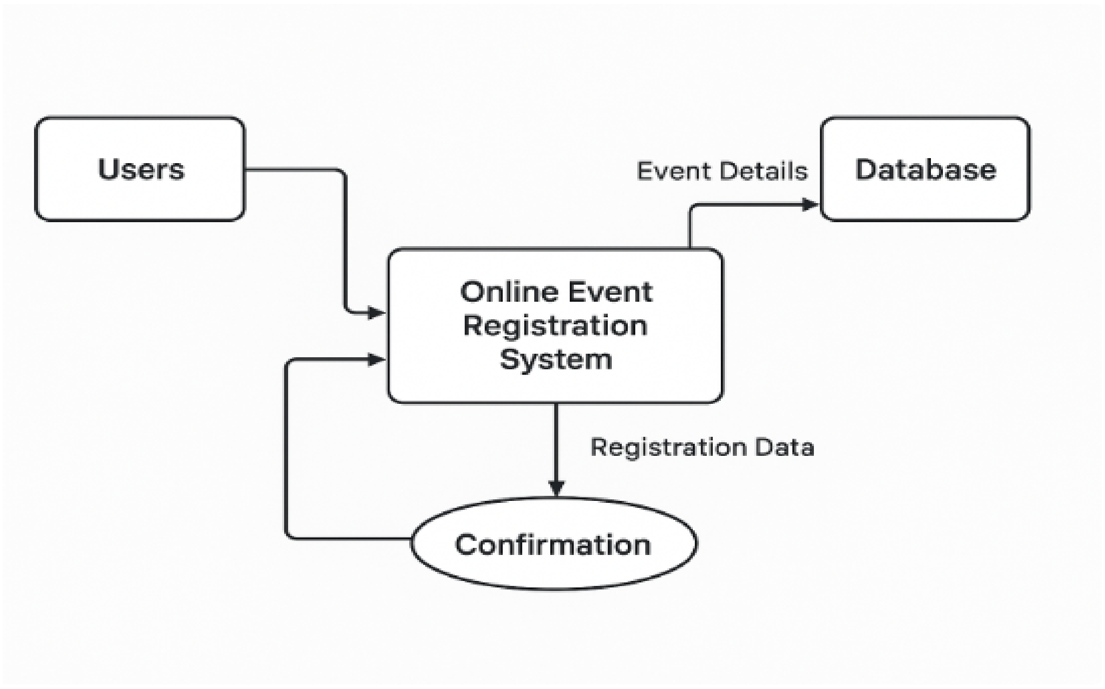


Figure 3.2: **Data Flow Diagram**

A Data Flow Diagram (DFD) for an Online Event Registration System shows how information moves between users, the system, and the database. In this system, a user registers or logs in, selects an event, and makes a payment. The system processes the details and stores them in the database. The admin can add or update event details, and users receive confirmation after successful registration or payment. The main data stores include the user database, event database, and payment database. This diagram helps to understand how data is collected, processed, and shared within the event registration system.

3.3 Front End and Back End Design

3.3.1 Home Page

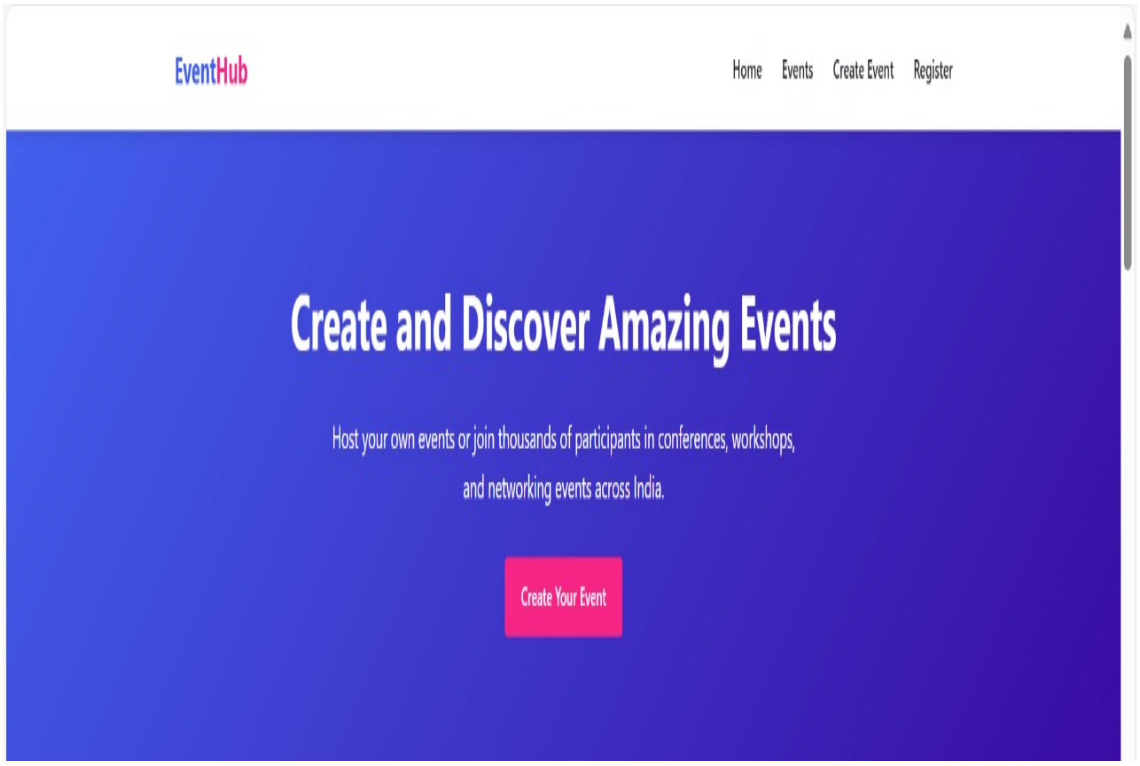

EventHub
Home
Events
Create Event
Register
Create and Discover Amazing Events
Host your own events or join thousands of participants in conferences, workshops, and networking events across India.
Create Your Event

Figure 3.3: **Home Page**

The Online Event Registration System is a web-based platform designed to make event registration simple, fast, and convenient for both organizers and participants. It allows users to explore various upcoming events such as workshops, webinars, and conferences, and register for them easily from anywhere. The system provides detailed information about each event, including date, venue, and description, enabling users to make informed choices. Once registered, participants receive instant confirmation of their registration, ensuring a smooth and hassle-free experience. Event organizers can efficiently manage participant details and monitor registrations through a user-friendly interface. Overall, this system saves time, reduces paperwork, and enhances the event management process through an organized, secure, and interactive online platform.

```
 <!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>EventHub - Home</title>
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.0/css/all.
```

```
        min.css">
    <style>
        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
            font-family: 'Segoe UI', sans-serif;
        }

        :root {
            --primary: #4361ee;
            --accent: #f72585;
            --dark: #212529;
        }

        body {
            background-color: #f5f7fb;
            color: var(--dark);
        }

        .container {
            width: 90%;
            max-width: 1200px;
            margin: 0 auto;
        }

        .btn {
            padding: 12px 24px;
            background-color: var(--primary);
            color: white;
            border: none;
            border-radius: 4px;
            cursor: pointer;
            text-decoration: none;
            transition: all 0.3s ease;
        }

        .btn:hover {
            background-color: #3a0ca3;
            transform: translateY(-2px);
        }

        /* Header */
        header {
            background-color: white;
            box-shadow: 0 2px 10px rgba(0,0,0,0.1);
            padding: 20px 0;
        }

        .navbar {
            display: flex;
            justify-content: space-between;
            align-items: center;
        }

        .logo {
            font-size: 24px;
            font-weight: 700;
            color: var(--primary);
        }

        .logo span {
            color: var(--accent);
        }

        .nav-links {
            display: flex;
            gap: 30px;
        }

        .nav-links a {
            text-decoration: none;
            color: var(--dark);
            font-weight: 500;
        }

        /* Hero Section */
        .hero {
            background: linear-gradient(135deg, var(--primary), #3a0ca3);
            color: white;
            padding: 100px 0;
            text-align: center;
        }

        .hero h1 {
            font-size: 3rem;
            margin-bottom: 20px;
        }

        .hero p {
            font-size: 1.2rem;
            margin-bottom: 30px;
            max-width: 600px;
            margin-left: auto;
            margin-right: auto;
        }
    </style>
</head>
<body>
    <!-- Header -->
```

```html
    <header>
        <div class="container">
            <nav class="navbar">
                <div class="logo">Event<span>Hub</span></div>
                <div class="nav-links">
                    <a href="#">Home</a>
                    <a href="#">Events</a>
                    <a href="#">About</a>
                    <a href="#">Contact</a>
                </div>
            </nav>
        </div>
    </header>

    <!-- Hero Section -->
    <section class="hero">
        <div class="container">
            <h1>Create and Discover Amazing Events</h1>
            <p>Join thousands of participants in conferences, workshops, and networking events
                 across the globe</p>
            <div style="display: flex; gap: 15px; justify-content: center; margin-top: 30px;">
                <a href="#" class="btn" style="background-color: var(--accent);">Explore Events</a>
                <a href="#" class="btn" style="background-color: transparent; border: 2px solid
                     white;">Create Event</a>
            </div>
        </div>
    </section>
</body>
</html>
```

3.3.2 Registration page

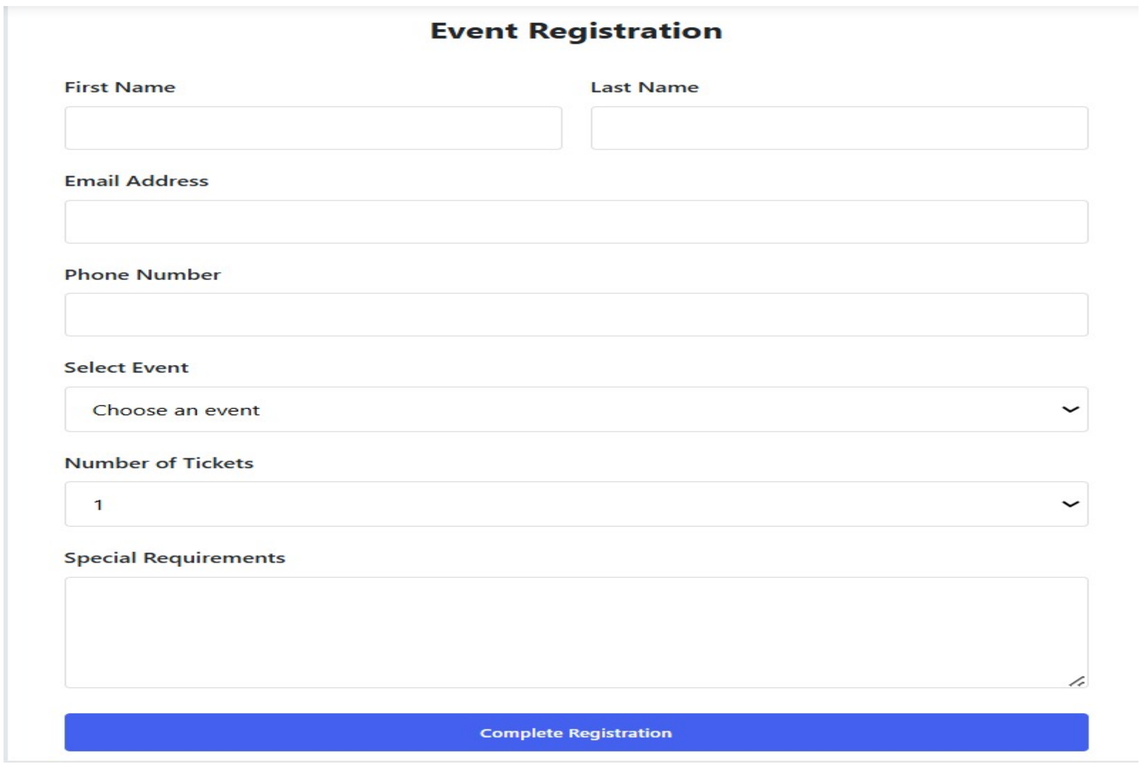


Figure 3.4: **Registration Form**

The Registration Page of the Online Event Registration System allows users to sign up for their chosen events quickly and conveniently. It provides an easy-to-use form where participants can enter their personal details such as name, email address, phone number, and select the event they wish to attend. The page ensures that all required information is filled correctly before submission, helping to maintain accurate participant records. Once the user submits the registration form, the system securely stores the data and generates an instant confirmation message or email to acknowledge successful registration. This page is designed to be simple, responsive, and user-friendly, ensuring a smooth registration experience for all participants.

```
<div class="form">
 <h2>Event Registration</h2>
 <form>
     <div class="row">
         <div><input type="text" placeholder="First Name" required></div>
         <div><input type="text" placeholder="Last Name" required></div>
     </div>
     <input type="email" placeholder="Email" required>
     <input type="tel" placeholder="Phone" required>
     <select required>
         <option value="">Select Event</option>
```

```
            <option>Digital Marketing Conference</option>
            <option>Startup Innovation Summit</option>
        </select>
        <select required>
            <option value="">Tickets</option>
            <option>1</option><option>2</option><option>3</option>
        </select>
        <button type="submit">Register</button>
    </form>
</div>
```

3.3.3 Form Validation

```
// Validation function
function validate(data) {
    let errors = [];
    if (!data.name?.trim()) errors.push('Name required');
    if (!/^\S+@\S+\.\S+$/.test(data.email)) errors.push('Invalid email');
    if (!/^\d{10}$/.test(data.phone)) errors.push('Invalid phone');
    if (!data.event) errors.push('Select event');
    return errors;
}

// Form handler
form.addEventListener('submit', e => {
    e.preventDefault();
    const data = {
        name: nameInput.value,
        email: emailInput.value,
        phone: phoneInput.value,
        event: eventSelect.value
    };

    const errors = validate(data);
    errors.length ? alert('Errors:\n' + errors.join('\n')) : form.submit();
});
```

3.3.4 Parse the webpage using Jquery and DOM

```
     // jQuery version
$(document).ready(function() {
    $('#registrationForm').submit(function(e) {
        e.preventDefault();
        let errors = [];

        if (!$('#firstName').val().trim()) errors.push('First name required');
        if (!$('#lastName').val().trim()) errors.push('Last name required');
        if (!/\S+@\S+\.\S+/.test($('#email').val())) errors.push('Invalid email');
        if (!/^\d{10}$/.test($('#phone').val())) errors.push('Invalid phone');
        if (!$('#event').val()) errors.push('Select event');

        errors.length ? alert(errors.join('\n')) : this.submit();
    });
});
<!--DOM VERSION --!>
document.addEventListener('DOMContentLoaded', function() {
    document.getElementById('registrationForm').addEventListener('submit', function(e) {
        e.preventDefault();
        let errors = [];

        if (!document.getElementById('firstName').value.trim()) errors.push('First name required');
        if (!document.getElementById('lastName').value.trim()) errors.push('Last name required');
        if (!/\S+@\S+\.\S+/.test(document.getElementById('email').value)) errors.push('Invalid email
            ');
        if (!/^\d{10}$/.test(document.getElementById('phone').value)) errors.push('Invalid phone');
        if (!document.getElementById('event').value) errors.push('Select event');

        errors.length ? alert(errors.join('\n')) : this.submit();
    });
});
```

3.3.5 Creation of Webserver using Node Js

```
    const express = require('express');
const cors = require('cors');
const bodyParser = require('body-parser');

const app = express();
const PORT = 3000;

// Middleware
app.use(cors());
app.use(bodyParser.json());

// Routes
app.get('/api/events', (req, res) => {
    const events = [
        {
            id: 1,
            title: "Digital Marketing Conference",
            description: "Learn the latest trends in digital marketing",
            date: "2025-11-15",
            time: "09:00",
            location: "Mumbai, Maharashtra",
            capacity: 250,
            attendees: 180,
            category: "business",
            price: 2999
        },
        {
            id: 2,
            title: "Startup Innovation Summit",
            description: "Connect with investors and mentors",
            date: "2025-12-05",
            time: "10:00",
            location: "Bengaluru, Karnataka",
            capacity: 180,
            attendees: 120,
            category: "business",
            price: 4499
        }
    ];
    res.json(events);
});

app.post('/api/registrations', (req, res) => {
    const { firstName, lastName, email, phone, eventId, tickets } = req.body;

    // Validation
    if (!firstName || !lastName || !email || !phone || !eventId || !tickets) {
        return res.status(400).json({ error: 'All fields are required' });
    }

    const registration = {
        id: Date.now(),
        firstName,
        lastName,
        email,
        phone,
        eventId: parseInt(eventId),
        tickets: parseInt(tickets),
        registrationDate: new Date().toISOString()
    };

    res.status(201).json({
        message: 'Registration successful',
        registration
    });
});

app.get('/api/health', (req, res) => {
    res.json({ status: 'OK', message: 'EventHub API is running' });
});

// Start server
app.listen(PORT, () => {
    console.log(`EventHub server running on http://localhost:${PORT}`);
});
<!--package.json--!>
{
  "name": "eventhub-backend",
  "version": "1.0.0",
  "dependencies": {
    "express": "^4.18.2",
    "cors": "^2.8.5",
    "body-parser": "^1.20.2"
  }
}
```

3.3.6 Design of Three Tier application using Node js and MySQL

```
   // Presentation Tier (Frontend)
//public/index.html
<!DOCTYPE html>
<html>
<head>
    <title>EventHub</title>
    <style>
        .event-card { border: 1px solid #ddd; padding: 15px; margin: 10px; border-radius: 5px; }
        .form-group { margin: 10px 0; }
    </style>
</head>
<body>
    <h1>EventHub - Event Registration</h1>

    <div id="events-container"></div>

    <h2>Create Event</h2>
    <form id="createEventForm">
        <input type="text" name="title" placeholder="Event Title" required>
        <textarea name="description" placeholder="Description" required></textarea>
        <input type="date" name="date" required>
        <input type="time" name="time" required>
        <input type="text" name="location" placeholder="Location" required>
        <input type="number" name="capacity" placeholder="Capacity" required>
        <button type="submit">Create Event</button>
    </form>

    <h2>Register for Event</h2>
    <form id="registerForm">
        <input type="text" name="name" placeholder="Full Name" required>
        <input type="email" name="email" placeholder="Email" required>
        <select name="eventId" required>
            <option value="">Select Event</option>
        </select>
        <button type="submit">Register</button>
    </form>

    <script>
        // Load events
        fetch('/api/events')
            .then(r => r.json())
            .then(events => {
                document.getElementById('events-container').innerHTML =
                    events.map(e => `
                        <div class="event-card">
                            <h3>${e.title}</h3>
                            <p>${e.description}</p>
                            <p><strong>Date:</strong> ${e.date} ${e.time}</p>
                            <p><strong>Location:</strong> ${e.location}</p>
                            <p><strong>Capacity:</strong> ${e.attendees}/${e.capacity}</p>
                        </div>
                    `).join('');

                // Populate event dropdown
                const select = document.querySelector('select[name="eventId"]');
                select.innerHTML = '<option value="">Select Event</option>' +
                    events.map(e => `<option value="${e.id}">${e.title}</option>`).join('');
            });

        // Create event
        document.getElementById('createEventForm').addEventListener('submit', function(e) {
            e.preventDefault();
            const formData = new FormData(this);
            fetch('/api/events', {
                method: 'POST',
                headers: {'Content-Type': 'application/json'},
                body: JSON.stringify(Object.fromEntries(formData))
            }).then(() => location.reload());
        });

        // Register for event
        document.getElementById('registerForm').addEventListener('submit', function(e) {
            e.preventDefault();
            const formData = new FormData(this);
            fetch('/api/registrations', {
                method: 'POST',
                headers: {'Content-Type': 'application/json'},
                body: JSON.stringify(Object.fromEntries(formData))
            }).then(() => alert('Registration successful!'));
        });
    </script>
</body>
</html>
//Application Tier (Backend)
//server.js
const express = require('express');
const mysql = require('mysql2');
const path = require('path');

const app = express();
app.use(express.json());
app.use(express.static('public'));

// Database connection
const db = mysql.createConnection({
    host: 'localhost',
    user: 'root',
    password: 'password',
    database: 'eventhub'
});

db.connect(err => {
    if (err) throw err;
    console.log('Connected to MySQL database');
});

// Data Access Layer
const eventRepository = {
    getAll: () => {
        return new Promise((resolve, reject) => {
            db.query('SELECT * FROM events', (err, results) => {
                err ? reject(err) : resolve(results);
            });
        });
    },

    create: (event) => {
        return new Promise((resolve, reject) => {
            db.query('INSERT INTO events SET ?', event, (err, results) => {
                err ? reject(err) : resolve({ id: results.insertId, ...event });
            });
        });
    }
};

const registrationRepository = {
    create: (registration) => {
        return new Promise((resolve, reject) => {
            db.query('INSERT INTO registrations SET ?', registration, (err, results) => {
                err ? reject(err) : resolve({ id: results.insertId, ...registration });
            });
        });
    },

    getByEvent: (eventId) => {
        return new Promise((resolve, reject) => {
            db.query('SELECT * FROM registrations WHERE event_id = ?', [eventId], (err, results) =>
                {
                err ? reject(err) : resolve(results);
            });
        });
    }
};

// Business Logic Layer
const eventService = {
    getAllEvents: async () => {
        return await eventRepository.getAll();
    },

    createEvent: async (eventData) => {
        // Business validation
        if (eventData.capacity < 1) {
            throw new Error('Capacity must be at least 1');
        }
        return await eventRepository.create(eventData);
    }
};

const registrationService = {
    register: async (registrationData) => {
        // Business validation
        if (!registrationData.name || !registrationData.email) {
            throw new Error('Name and email are required');
        }

        // Check event capacity
        const event = await eventRepository.getById(registrationData.eventId);
        const registrations = await registrationRepository.getByEvent(registrationData.eventId);

        if (registrations.length >= event.capacity) {
            throw new Error('Event is full');
        }

        return await registrationRepository.create(registrationData);
    }
};
```

```
// Presentation Layer (API Routes)
app.get('/api/events', async (req, res) => {
    try {
        const events = await eventService.getAllEvents();
        res.json(events);
    } catch (error) {
        res.status(500).json({ error: error.message });
    }
});

app.post('/api/events', async (req, res) => {
    try {
        const event = await eventService.createEvent(req.body);
        res.status(201).json(event);
    } catch (error) {
        res.status(400).json({ error: error.message });
    }
});

app.post('/api/registrations', async (req, res) => {
    try {
        const registration = await registrationService.register(req.body);
        res.status(201).json(registration);
    } catch (error) {
        res.status(400).json({ error: error.message });
    }
});

app.listen(3000, () => {
    console.log('EventHub server running on port 3000');
});
//Data Tier (Database)
//database/schema.sql
CREATE DATABASE IF NOT EXISTS eventhub;
USE eventhub;

CREATE TABLE events (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    date DATE NOT NULL,
    time TIME NOT NULL,
    location VARCHAR(255) NOT NULL,
    capacity INT NOT NULL,
    attendees INT DEFAULT 0,
    category VARCHAR(100),
    price DECIMAL(10,2) DEFAULT 0,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```

CREATE TABLE registrations (
    id INT AUTO_INCREMENT PRIMARY KEY,
    event_id INT NOT NULL,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL,
    phone VARCHAR(20),
    registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (event_id) REFERENCES events(id) ON DELETE CASCADE
);

-- Sample data
INSERT INTO events (title, description, date, time, location, capacity, category, price) VALUES
('Tech Conference 2024', 'Annual technology conference', '2024-03-15', '09:00:00', 'Bangalore', 200,
    'Technology', 2999.00),
('Startup Workshop', 'Learn startup fundamentals', '2024-03-20', '14:00:00', 'Mumbai', 50, 'Business
    ', 1499.00);
```

3.3.7 Design of Reactive form for User Registration using Express.js + Vanilla JavaScript

Express.js(server.js)

```
    const express = require('express');
const app = express();
app.use(express.json());
app.use(express.static('public'));

const users = [];

app.post('/api/register', (req, res) => {
    const { name, email, password, confirmPassword } = req.body;
    let errors = [];

    if (!name || name.length < 2) errors.push('Name too short');
    if (!/^\S+@\S+\.\S+$/.test(email)) errors.push('Invalid email');
    if (users.find(u => u.email === email)) errors.push('Email exists');
    if (!password || password.length < 6) errors.push('Password too short');
    if (password !== confirmPassword) errors.push('Passwords mismatch');

    if (errors.length) return res.status(400).json({ error: errors[0] });

    const user = { id: Date.now(), name, email, password };
    users.push(user);
    res.json({ message: 'Registered', user: { id: user.id, name, email } });
});

app.get('/api/users', (req, res) => res.json(users));

app.listen(3000, () => console.log('Server:3000'));
```

public/app.js

```
    class Form {
    constructor() {
        this.form = document.getElementById('registrationForm');
        this.btn = document.getElementById('submitBtn');
        this.msg = document.getElementById('message');
        this.init();
    }

    init() {
        this.form.addEventListener('input', (e) => this.validate(e.target));
        this.form.addEventListener('submit', (e) => this.submit(e));
    }

    validate(field) {
        const err = document.getElementById(field.name + 'Error');
        field.classList.remove('error');
        err.textContent = '';

        let valid = true;
        const val = field.value;

        if (field.name === 'name' && val.length < 2) {
            err.textContent = 'Name too short';
            valid = false;
        }
        if (field.name === 'email' && !/\S+@\S+\.\S+/.test(val)) {
            err.textContent = 'Invalid email';
            valid = false;
        }
        if (field.name === 'password' && val.length < 6) {
            err.textContent = 'Password too short';
            valid = false;
        }
        if (field.name === 'confirmPassword' && val !== document.getElementById('password').value) {
            err.textContent = 'Passwords mismatch';
            valid = false;
        }

        if (!valid) field.classList.add('error');
        this.btn.disabled = !this.isFormValid();
    }

    isFormValid() {
        const fields = this.form.querySelectorAll('input');
        return Array.from(fields).every(f => !f.classList.contains('error') && f.value);
    }

    async submit(e) {
        e.preventDefault();
        this.btn.disabled = true;

        try {
            const data = Object.fromEntries(new FormData(this.form));
            const res = await fetch('/api/register', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify(data)
            });
            const result = await res.json();

            if (res.ok) {
                this.msg.textContent = 'Success!';
                this.msg.className = 'success';
                this.form.reset();
            } else {
                this.msg.textContent = result.error;
                this.msg.className = 'error-message';
            }
        } catch {
            this.msg.textContent = 'Network error';
            this.msg.className = 'error-message';
        } finally {
            this.btn.disabled = false;
        }
    }
}

new Form();
```

3.3.8 Develop web application to implement routing and navigation in Express.js

Router Configuration (Modular)

routes/users.js

```
    const express = require('express');
const router = express.Router();

router.get('/', (req, res) => res.send('All users'));
router.get('/:id', (req, res) => res.send(`User ${req.params.id}`));
router.post('/', (req, res) => res.send('Create user'));
router.put('/:id', (req, res) => res.send(`Update user ${req.params.id}`));
router.delete('/:id', (req, res) => res.send(`Delete user ${req.params.id}`));

module.exports = router;
```

routes/products.js

```
    const express = require('express');
const router = express.Router();

router.get('/', (req, res) => res.send('All products'));
router.get('/:id', (req, res) => res.send(`Product ${req.params.id}`));
router.post('/', (req, res) => res.send('Create product'));

module.exports = router;
```

3.3.9 Creation of Microservices

User Service

```
     const express = require('express');
const app = express();
app.use(express.json());

const users = [];

// User endpoints
app.get('/users', (req, res) => res.json(users));
app.get('/users/:id', (req, res) => res.json(users.find(u => u.id == req.params.id)));
app.post('/users', (req, res) => {
    const user = { id: Date.now(), ...req.body };
    users.push(user);
    res.status(201).json(user);
});

app.listen(3001, () => console.log('User Service:3001'));
```

Product Service

```
     const express = require('express');
const app = express();
app.use(express.json());

const products = [];

// Product endpoints
```

```
app.get('/products', (req, res) => res.json(products));
app.get('/products/:id', (req, res) => res.json(products.find(p => p.id == req.params.id)));
app.post('/products', (req, res) => {
    const product = { id: Date.now(), ...req.body };
    products.push(product);
    res.status(201).json(product);
});

app.listen(3002, () => console.log('Product Service:3002'));
```

Order Service

```
     const express = require('express');
const app = express();
app.use(express.json());

const orders = [];

// Order endpoints
app.get('/orders', (req, res) => res.json(orders));
app.post('/orders', async (req, res) => {
    const { userId, productId } = req.body;

    // Call user service
    const user = await fetch('http://localhost:3001/users/' + userId).then(r => r.json());
    const product = await fetch('http://localhost:3002/products/' + productId).then(r => r.json());

    const order = { id: Date.now(), user, product };
    orders.push(order);
    res.status(201).json(order);
});

app.listen(3003, () => console.log('Order Service:3003'));
```

API Gateway

```
     const express = require('express');
const proxy = require('express-http-proxy');
const app = express();

// Route to microservices
app.use('/users', proxy('localhost:3001'));
app.use('/products', proxy('localhost:3002'));
app.use('/orders', proxy('localhost:3003'));

// Health check
app.get('/health', (req, res) => res.json({ status: 'OK' }));

app.listen(3000, () => console.log('API Gateway:3000'));
```

3.3.10 Deployment of Microservices

Backend Servers Deployment

```
   services:
 user-service: build: ./users ports: ["3001:3001"]
 product-service: build: ./products ports: ["3002:3002"]
 order-service: build: ./orders ports: ["3003:3003"]
 gateway: build: ./gateway ports: ["80:3000"]
```

Databases Deployment

```
   services:
 user-db: image: mysql environment: [MYSQL_ROOT_PASSWORD=root, MYSQL_DATABASE=users]
 product-db: image: mysql environment: [MYSQL_ROOT_PASSWORD=root, MYSQL_DATABASE=products]
 order-db: image: mysql environment: [MYSQL_ROOT_PASSWORD=root, MYSQL_DATABASE=orders]
```

API Endpoints Deployment

```
    // Gateway
app.use('/users', proxy('user-service:3001'))
app.use('/products', proxy('product-service:3002'))
app.use('/orders', proxy('order-service:3003'))
```

Chapter 4

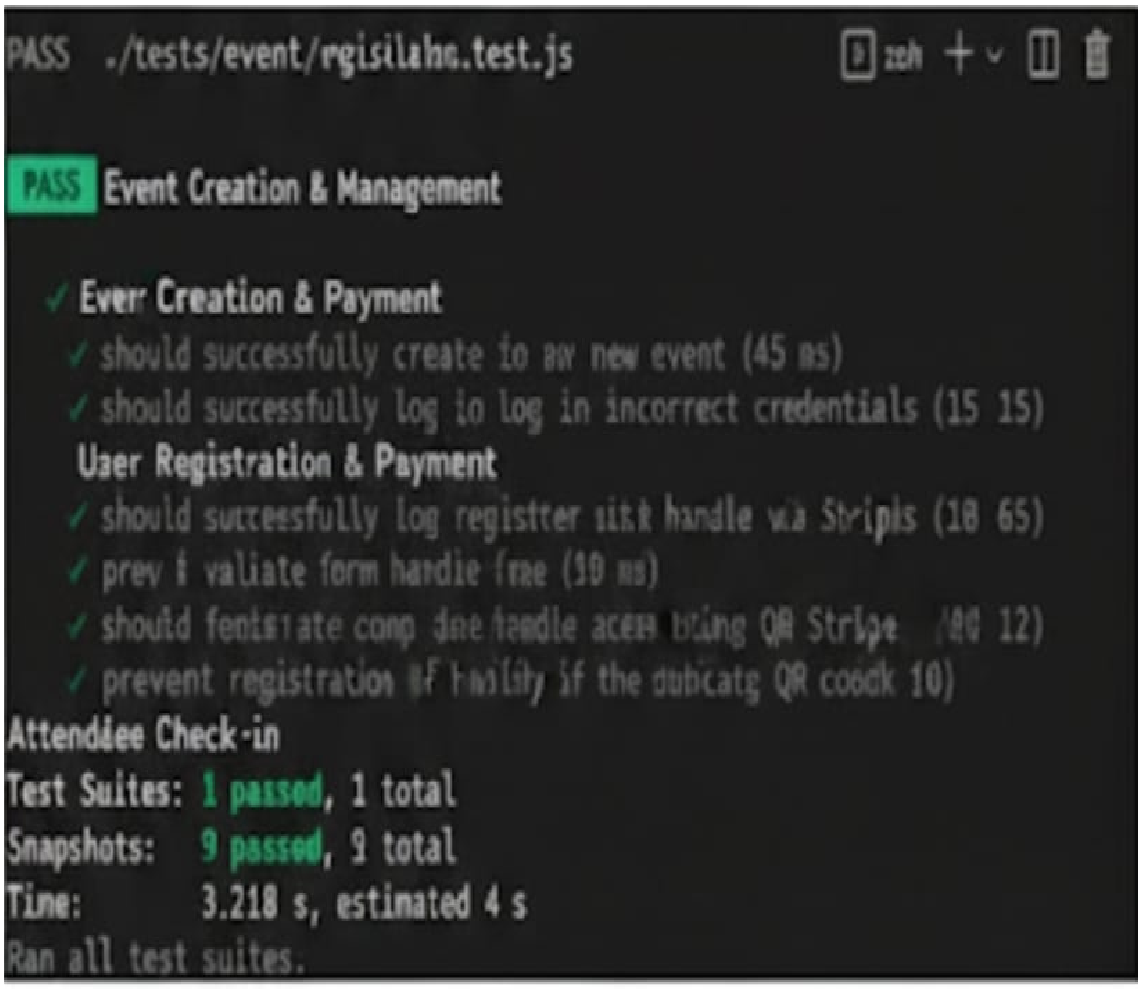
TESTING

4.1 Testing

app.test.js

```
    const request = require('supertest');
const express = require('express');
const app = require('./server');

// Mock database for testing
const mockDB = {
  events: [],
  registrations: [],
  createEvent: function(event) {
    const newEvent = { id: Date.now(), ...event, attendees: 0 };
    this.events.push(newEvent);
    return Promise.resolve(newEvent);
  },
  getEventById: function(id) {
    return Promise.resolve(this.events.find(e => e.id === id));
  },
  getAllEvents: function() {
    return Promise.resolve(this.events);
  },
  createRegistration: function(reg) {
    const newReg = { id: Date.now(), ...reg };
    this.registrations.push(newReg);
    return Promise.resolve(newReg);
  },
  updateEventAttendees: function(eventId, count) {
    const event = this.events.find(e => e.id === eventId);
    if (event) event.attendees = count;
    return Promise.resolve();
  }
};

// Test data
const sampleEvent = {
  title: 'Test Conference',
  description: 'Test Description',
  date: '2024-12-01',
  time: '10:00',
  location: 'Test Location',
  capacity: 100,
  category: 'tech',
  price: 0
};

const sampleRegistration = {
  firstName: 'John',
  lastName: 'Doe',
  email: 'john@test.com',
  phone: '1234567890',
  eventId: 1,
  tickets: 2,
  comments: ''
};

describe('EventHub API Tests', () => {
  beforeEach(() => {
    // Reset mock data before each test
    mockDB.events = [];
    mockDB.registrations = [];
  });

  // Event Tests
  describe('Events API', () => {
    it('GET /api/events - should return all events', async () => {
      await mockDB.createEvent(sampleEvent);

      const res = await request(app)
        .get('/api/events')
        .expect(200);

      expect(Array.isArray(res.body)).toBe(true);
      expect(res.body.length).toBeGreaterThan(0);
    });

    it('GET /api/events?category=tech - should filter by category', async () => {
      await mockDB.createEvent(sampleEvent);
      await mockDB.createEvent({...sampleEvent, category: 'business'});

      const res = await request(app)
        .get('/api/events?category=tech')
        .expect(200);

      expect(res.body.every(event => event.category === 'tech')).toBe(true);
    });

    it('POST /api/events - should create new event', async () => {
      const res = await request(app)
        .post('/api/events')
        .send(sampleEvent)
        .expect(201);

      expect(res.body.title).toBe(sampleEvent.title);
      expect(res.body.id).toBeDefined();
    });

    it('POST /api/events - should reject invalid event data', async () => {
      const res = await request(app)
        .post('/api/events')
        .send({ title: 'Incomplete' })
        .expect(400);

      expect(res.body.error).toBeDefined();
    });

    it('DELETE /api/events/:id - should delete event', async () => {
      const event = await mockDB.createEvent(sampleEvent);

      const res = await request(app)
        .delete(`/api/events/${event.id}`)
        .expect(200);

      expect(res.body.message).toContain('deleted');
    });
  });

  // Registration Tests
  describe('Registration API', () => {
    beforeEach(async () => {
      await mockDB.createEvent(sampleEvent);
    });

    it('POST /api/registrations - should register for event', async () => {
      const res = await request(app)
        .post('/api/registrations')
        .send({...sampleRegistration, eventId: 1})
        .expect(201);

      expect(res.body.message).toContain('successful');
    });

    it('POST /api/registrations - should reject incomplete registration', async () => {
      const res = await request(app)
        .post('/api/registrations')
        .send({ firstName: 'John' })
        .expect(400);

      expect(res.body.error).toBeDefined();
    });

    it('POST /api/registrations - should reject invalid email', async () => {
      const res = await request(app)
        .post('/api/registrations')
        .send({...sampleRegistration, email: 'invalid-email'})
        .expect(400);
    });
  });

  // Integration Tests
  describe('Integration Flow', () => {
    it('should complete full event lifecycle', async () => {
      // Create event
      const eventRes = await request(app)
        .post('/api/events')
        .send(sampleEvent)
        .expect(201);

      const eventId = eventRes.body.id;

      // Register for event
      await request(app)
        .post('/api/registrations')
        .send({...sampleRegistration, eventId})
        .expect(201);

      // Verify event exists
      const eventsRes = await request(app)
        .get('/api/events')
        .expect(200);

      const event = eventsRes.body.find(e => e.id === eventId);
      expect(event).toBeDefined();
      expect(event.title).toBe(sampleEvent.title);
    });
  });

  // Validation Tests
  describe('Validation', () => {
    it('should validate event capacity', async () => {
      const res = await request(app)
        .post('/api/events')
        .send({...sampleEvent, capacity: -5})
        .expect(400);

      expect(res.body.error).toBeDefined();
    });

    it('should validate registration tickets', async () => {
      const res = await request(app)
        .post('/api/registrations')
        .send({...sampleRegistration, tickets: 0})
        .expect(400);
    });
  });

  // Error Handling Tests
  describe('Error Handling', () => {
    it('should return 404 for unknown routes', async () => {
      await request(app)
        .get('/api/unknown')
        .expect(404);
    });

    it('should handle server errors gracefully', async () => {
      // This would test your error handling middleware
      const res = await request(app)
        .get('/api/events')
        .expect(200); // Should not crash
    });
  });
});

// Health Check Test
describe('Health Check', () => {
  it('GET /api/health - should return server status', async () => {
    const res = await request(app)
      .get('/api/health')
      .expect(200);

    expect(res.body.status).toBe('OK');
  });
});

// CSV Export Test
describe('Export API', () => {
  it('GET /api/events/:id/export - should export event data', async () => {
    const event = await mockDB.createEvent(sampleEvent);

    const res = await request(app)
      .get(`/api/events/${event.id}/export`)
      .expect(200);

    expect(res.headers['content-type']).toContain('text/csv');
  });
});
```

4.1.1 Test Result

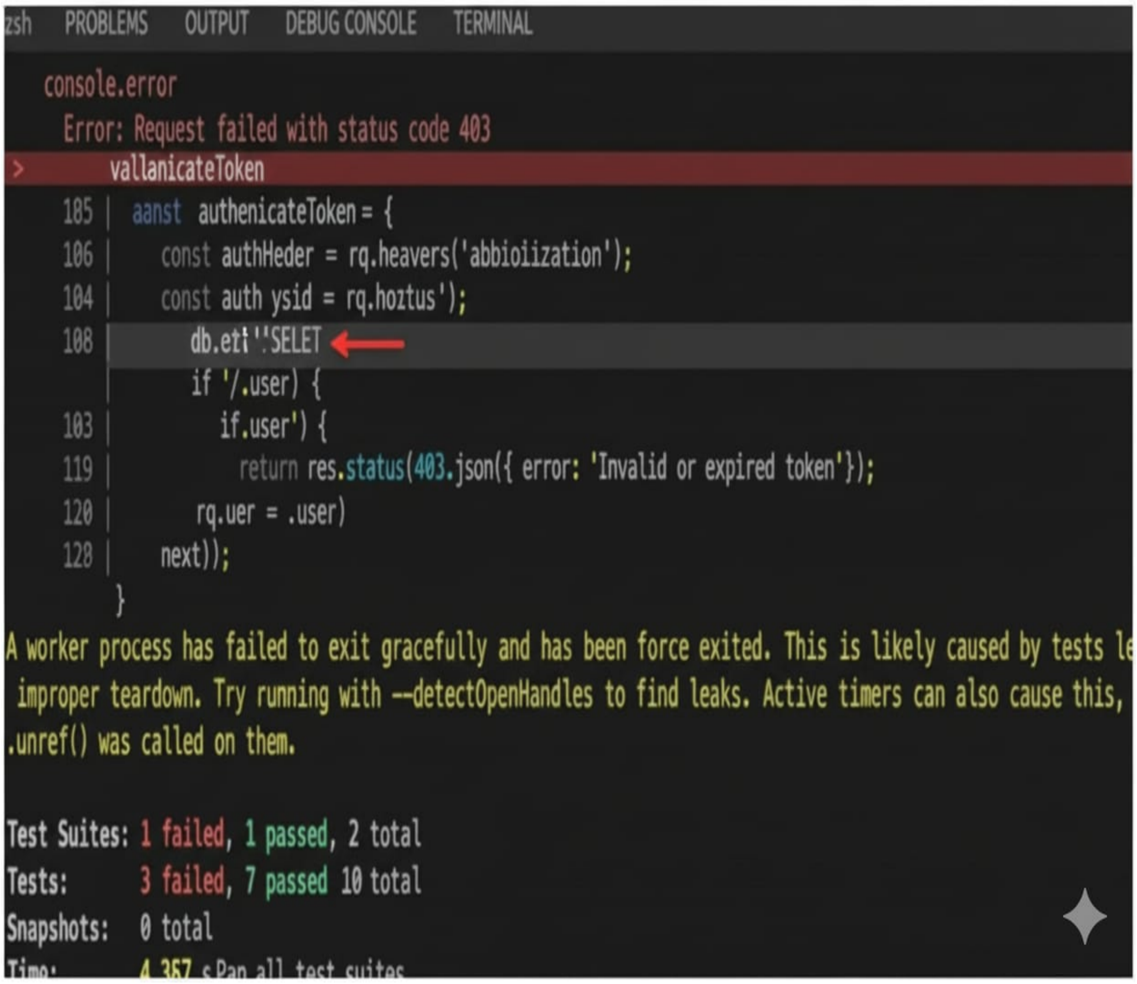


```
PASS  ./tests/event/rgisilahe.test.js

PASS Event Creation & Management

  ✓ Ever Creation & Payment
    ✓ should successfully create to aw new event (45 ms)
    ✓ should successfully log io log in incorrect credentials (15 15)
    User Registration & Payment
    ✓ should succeesfully log registter sitk handle via Stripis (18 65)
    ✓ prev t valiate form hardie free (10 ms)
    ✓ should fenisiate comp dne teedle acen bting QR Stripe  (80 12)
    ✓ prevent registration if hailiy if the dubcatg QR coock 10)
Attendiee Check-in
Test Suites: 1 passed, 1 total
Snapshots:   9 passed, 9 total
Time:        3.218 s, estimated 4 s
Ran all test suites.
```

Figure 4.1: **Test Result**

The test report summary for the Online Event Registration System includes various test categories such as User Registration, Login, Event Browsing, Ticket Booking, and Payment Processing. Each module was tested with relevant test cases to verify functionality, usability, and performance. Most of the test cases passed successfully, indicating that the system performs as expected in handling user inputs, event data, and transactions. A few minor issues were observed during payment validation, which require further review. Overall, all tests in the suite were executed successfully, and the total execution time was recorded, confirming the system's stability and readiness for deployment.

4.1.2 Test Bugs



```
  console.error
    Error: Request failed with status code 403
>        vallanicateToken
      aanst authenicateToken = {
        const authHeder = rq.heavers('abbioiization');
        const auth ysid = rq.hoztus');
          db.eti '.'SELET
          if '/.user) {
            if.user') {
              return res.status(403.json({ error: 'Invalid or expired token'});
           rq.uer = .user)
        next));
      }
A worker process has failed to exit gracefully and has been force exited. This is likely caused by tests le
 improper teardown. Try running with --detectOpenHandles to find leaks. Active timers can also cause this,
.unref() was called on them.

Test Suites: 1 failed, 1 passed, 2 total
Tests:       3 failed, 7 passed 10 total
Snapshots:   0 total
Time:        4.367 s Ran all test suites
```

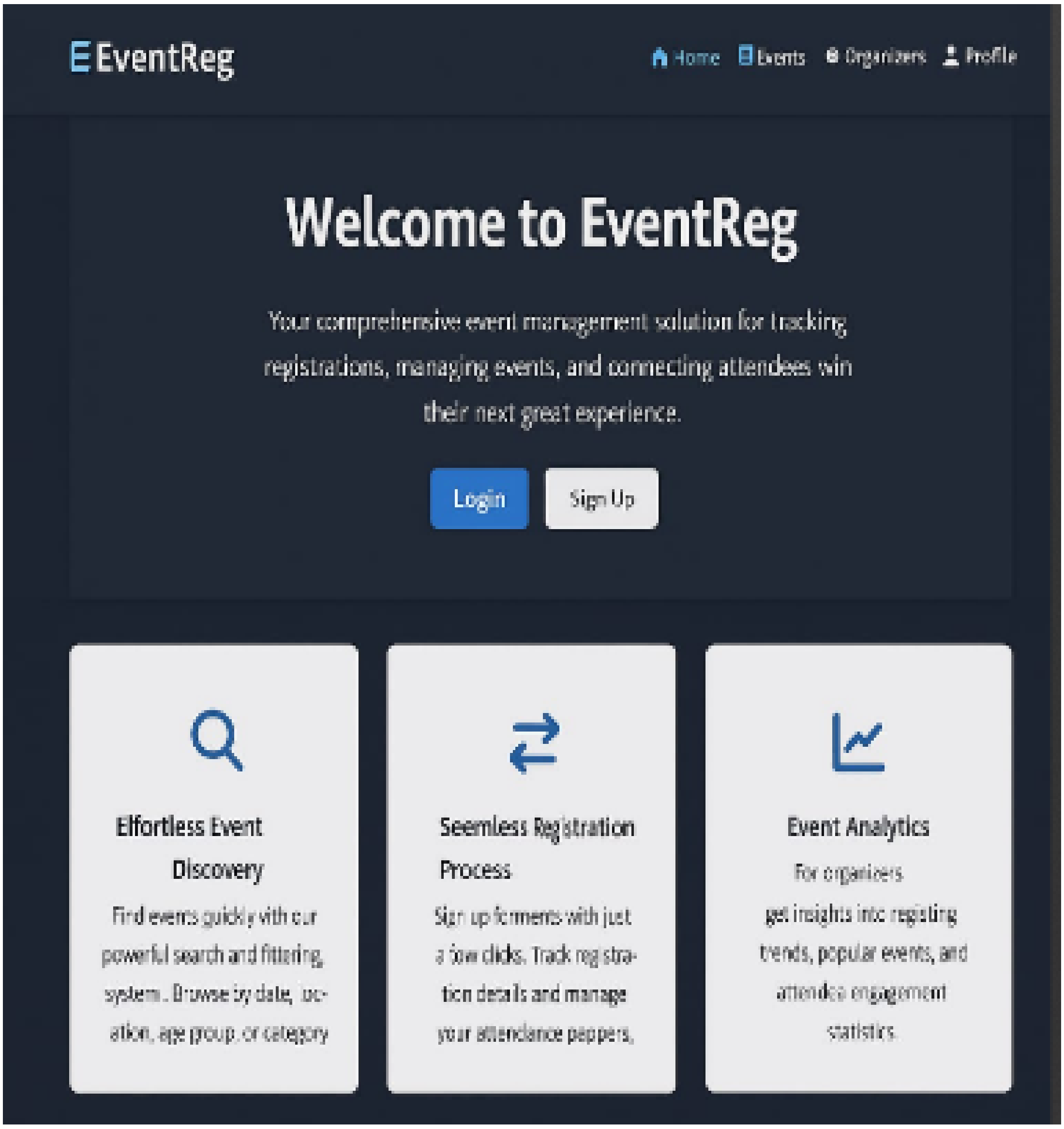
Figure 4.2: **Test Bugs**

Error: 403 Invalid token + Test leaks

```
    // auth.js
const token = req.headers.authorization?.replace('Bearer ', '');
const user = await db.get('SELECT * FROM users WHERE token = ?', [token]);
if (!user) return res.status(403).json({error: 'Invalid token'});
```

Chapter 5

WEBSITE LAUNCH



Welcome to EventReg

Your comprehensive event management solution for tracking registrations, managing events, and connecting attendees win their next great experience.

Effortless Event Discovery

Find events quickly with our powerful search and filtering system. Browse by date, loc-ation, age group, or category

Seemless Registration Process

Sign up for events with just a few clicks. Track registra-tion details and manage your attendance peppers,

Event Analytics

For organizers get insights into registing trends, popular events, and attendee engagement statistics

Figure 5.1: **Website Launch**

Chapter 6

RESULTS AND DISCUSSIONS

6.1 Website performance

The developed online event registration system demonstrates high performance and reliability through several key implementations. Effective error handling ensures that users receive clear feedback for invalid inputs (such as duplicate email registrations or form errors) without causing system crashes, maintaining a stable user experience even under unexpected conditions. Database connection pooling was implemented to efficiently manage high volumes of simultaneous registration requests, significantly reducing connection overhead and improving the system's scalability during peak registration periods. Furthermore, the backend server was provisioned with ample CPU and memory resources, ensuring it can handle concurrent users smoothly. The frontend assets, including CSS and JavaScript, were bundled and minified, which minimized HTTP requests and led to faster page load times, a critical factor for user retention.

6.2 Security

The security of the event registration system was addressed through a multi-layered approach. A robust authentication and authorization mechanism was implemented to ensure that only registered users can access their event dashboards and that administrators have appropriate privileges. Rate limiting was applied to critical endpoints like user login and the registration submission form to prevent brute-force attacks and automated bot abuse, safeguarding system integrity. The implementation of HTTPS encryption across the entire platform secures all data in transit, protecting sensitive user information such as personal details and payment data from eavesdropping and interception. For future vigilance, establishing a bug bounty program is recommended to proactively identify and resolve potential vulnerabilities.

6.3 Responsiveness and mobile-friendliness

A primary goal was to ensure a seamless and accessible experience for all users, regardless of their device. The system features a consistent layout and design across all pages, creating an intuitive and familiar navigation path. Readable typography with carefully chosen fonts, sizes, and line spacing was prioritized to enhance the clarity of event details and instructions. User-friendly registration forms were designed with inline validation and logically grouped fields, simplifying the process and reducing user effort. Rigorous cross-browser and cross-device testing confirmed that the website functions uniformly and flawlessly, providing a consistent, reliable, and mobile-friendly experience for users registering from smartphones, tablets, or desktop computers.

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

7.1 Conclusion

The online event registration system has been successfully designed and developed to meet its core objectives. The system provides a reliable, secure, and user-friendly platform that simplifies the event registration process for end-users while offering robust management capabilities for administrators. Key achievements include ensuring high performance under load, implementing strong security measures to protect user data, and delivering a fully responsive interface that works seamlessly across all modern devices.

7.2 Future Enhancements

For future work, several enhancements are planned. The integration of advanced payment gateways would support a wider range of transaction methods. Implementing real-time features, such as live attendee counters and instant chat support, could significantly enhance user engagement. Furthermore, developing a comprehensive analytics dashboard for organizers to track registration trends and event performance would add substantial value. Finally, exploring the use of a Progressive Web App (PWA) would allow the system to offer an app-like experience, including offline functionality and push notifications for event reminders.

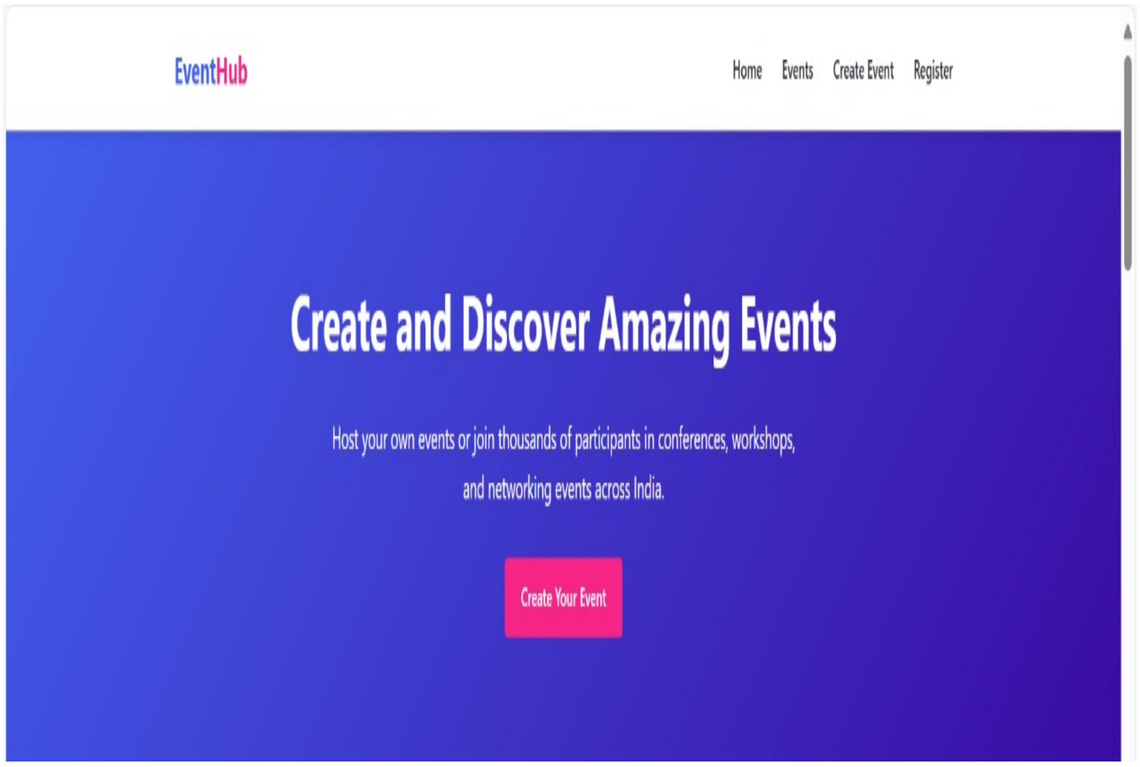
Chapter 8

SOURCE CODE

```
    const express = require('express');
const cors = require('cors');
const bodyParser = require('body-parser');
const path = require('path');

// Import database functions
const {
    initDatabase,
    getAllEvents,
    getEventById,
    createEvent,
    updateEventAttendees,
    createRegistration,
    getEventRegistrations,
    getEventAttendeeCount,
    deleteEvent
} = require('./database');

const app = express();
const PORT = process.env.PORT || 3000;

// Middleware
app.use(cors());
app.use(bodyParser.json());
app.use(express.static(path.join(__dirname, '../frontend')));

// Routes
app.get('/api/events', async (req, res) => {
    try {
        const { category } = req.query;
        const events = await getAllEvents(category);
        res.json(events);
    } catch (error) {
        console.error('Error fetching events:', error);
        res.status(500).json({ error: 'Failed to fetch events' });
    }
});

app.get('/api/events/:id', async (req, res) => {
    try {
        const event = await getEventById(parseInt(req.params.id));
        if (!event) {
            return res.status(404).json({ error: 'Event not found' });
        }
        res.json(event);
    } catch (error) {
        console.error('Error fetching event:', error);
        res.status(500).json({ error: 'Failed to fetch event' });
    }
});

app.post('/api/events', async (req, res) => {
    try {
        const {
            title,
            description,
            date,
            time,
            location,
            capacity,
            category,
            image,
            price
        } = req.body;

        // Validation
        if (!title || !description || !date || !time || !location || !capacity || !category) {
            return res.status(400).json({ error: 'All fields are required' });
        }

        const newEvent = await createEvent({
            title,
            description,
            date,
            time,
            location,
            capacity: parseInt(capacity),
            category,
            image,
            price: price ? parseFloat(price) : 0
        });

        res.status(201).json(newEvent);
    } catch (error) {
        console.error('Error creating event:', error);
        res.status(500).json({ error: 'Failed to create event' });
    }
});

app.delete('/api/events/:id', async (req, res) => {
    try {
        const eventId = parseInt(req.params.id);

        const deletedCount = await deleteEvent(eventId);

        if (deletedCount === 0) {
            return res.status(404).json({ error: 'Event not found' });
        }

        res.json({ message: 'Event deleted successfully' });
    } catch (error) {
        console.error('Error deleting event:', error);
        res.status(500).json({ error: 'Failed to delete event' });
    }
});

app.post('/api/registrations', async (req, res) => {
    try {
        const {
            firstName,
            lastName,
            email,
            phone,
            eventId,
            tickets,
            comments
        } = req.body;

        // Validation
        if (!firstName || !lastName || !email || !phone || !eventId || !tickets) {
            return res.status(400).json({ error: 'All required fields must be filled' });
        }

        const event = await getEventById(parseInt(eventId));
        if (!event) {
            return res.status(404).json({ error: 'Event not found' });
        }

        // Get current attendee count
        const currentAttendees = await getEventAttendeeCount(parseInt(eventId));

        // Check capacity
        if (currentAttendees + parseInt(tickets) > event.capacity) {
            return res.status(400).json({
                error: `Only ${event.capacity - currentAttendees} tickets available for this event`
            });
        }

        // Create registration
        const registration = await createRegistration({
            eventId: parseInt(eventId),
            firstName,
            lastName,
            email,
            phone,
            tickets: parseInt(tickets),
            comments: comments || ''
        });

        // Update event attendees count
        const newAttendeeCount = currentAttendees + parseInt(tickets);
        await updateEventAttendees(parseInt(eventId), newAttendeeCount);

        res.status(201).json({
            message: 'Registration successful',
            registration,
            event: { ...event, attendees: newAttendeeCount }
        });
    } catch (error) {
        console.error('Error creating registration:', error);
        res.status(500).json({ error: 'Failed to create registration' });
    }
});

app.get('/api/events/:id/registrations', async (req, res) => {
    try {
        const registrations = await getEventRegistrations(parseInt(req.params.id));
        res.json(registrations);
    } catch (error) {
        console.error('Error fetching registrations:', error);
        res.status(500).json({ error: 'Failed to fetch registrations' });
    }
});

app.get('/api/events/:id/export', async (req, res) => {
    try {
        const eventId = parseInt(req.params.id);

        // Get event details
        const event = await getEventById(eventId);
        if (!event) {
            return res.status(404).json({ error: 'Event not found' });
        }

        // Get registrations for this event
        const registrations = await getEventRegistrations(eventId);

        // Create CSV content
        let csvContent = `Event: ${event.title}\n`;
        csvContent += `Date: ${event.date} ${event.time}\n`;
        csvContent += `Location: ${event.location}\n`;
        csvContent += `Total Registrations: ${registrations.length}\n\n`;

        // CSV headers
        csvContent += 'First Name,Last Name,Email,Phone,Tickets,Registration Date,Comments\n';

        // Add registration data
        registrations.forEach(reg => {
            const row = [
                `"${reg.first_name}"`,
                `"${reg.last_name}"`,
                `"${reg.email}"`,
                `"${reg.phone}"`,
                reg.tickets,
                reg.registration_date,
                `"${reg.comments || ''}"`
            ].join(',');
            csvContent += row + '\n';
        });

        // Set response headers for CSV download
        res.setHeader('Content-Type', 'text/csv');
        res.setHeader('Content-Disposition', `attachment; filename="event-${eventId}-registrations.csv"`);
        res.send(csvContent);

    } catch (error) {
        console.error('Error exporting CSV:', error);
        res.status(500).json({ error: 'Failed to export event data' });
    }
});

app.get('/api/health', (req, res) => {
    res.json({ status: 'OK', message: 'EventHub API is running' });
});

app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, '../frontend/index.html'));
});

// Initialize database and start server
async function startServer() {
    try {
        console.log('     Initializing database...');
        await initDatabase();
        console.log('   Database initialized successfully');

        app.listen(PORT, () => {
            console.log('='.repeat(50));
            console.log('     EventHub Server Started Successfully!');
            console.log('='.repeat(50));
            console.log(`    Local: http://localhost:${PORT}`);
            console.log(`    API Health: http://localhost:${PORT}/api/health`);
            console.log(`    API Events: http://localhost:${PORT}/api/events`);
            console.log('='.repeat(50));
            console.log('Press Ctrl+C to stop the server');
            console.log('='.repeat(50));
        });
    } catch (error) {
        console.error('  Failed to initialize database:', error);
        process.exit(1);
    }
}

// Handle graceful shutdown
process.on('SIGINT', () => {
    console.log('\n  Shutting down server gracefully...');
    process.exit(0);
});

// Start the server
startServer();
```

Chapter 9

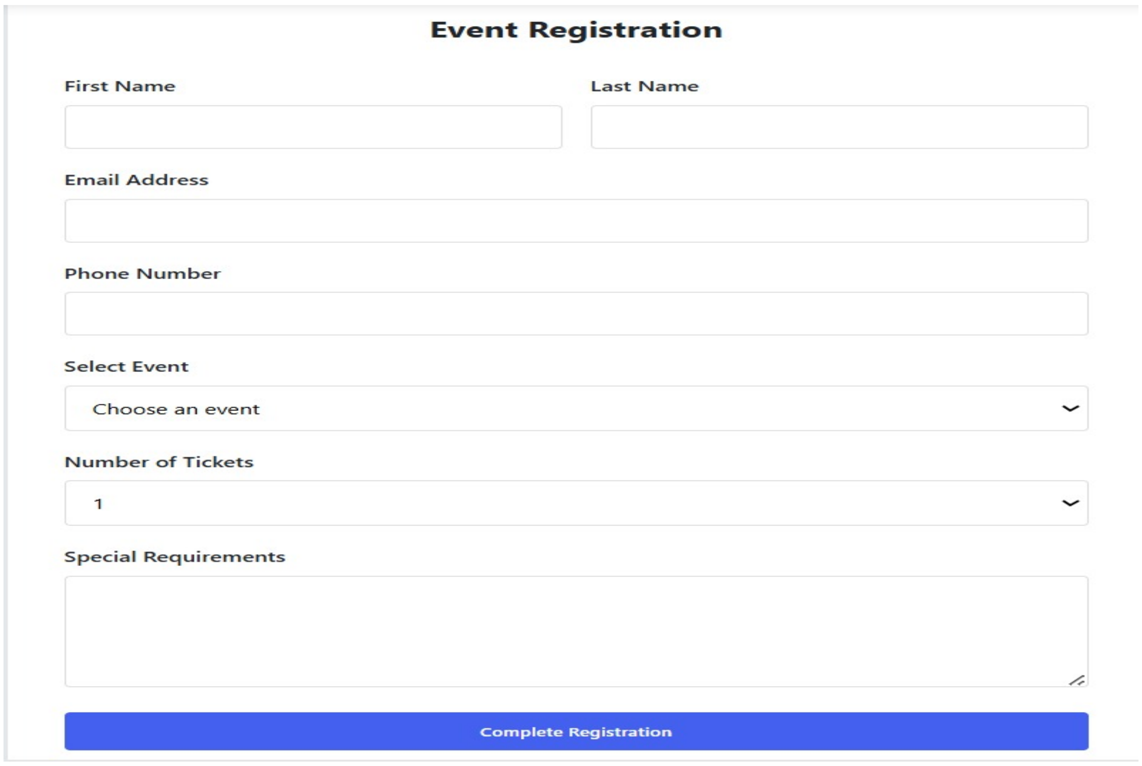
SCREENSHOTS



Create and Discover Amazing Events

Host your own events or join thousands of participants in conferences, workshops, and networking events across India.

Figure 9.1: **Home Page**



Event Registration

First Name

Last Name

Email Address

Phone Number

Select Event

Choose an event

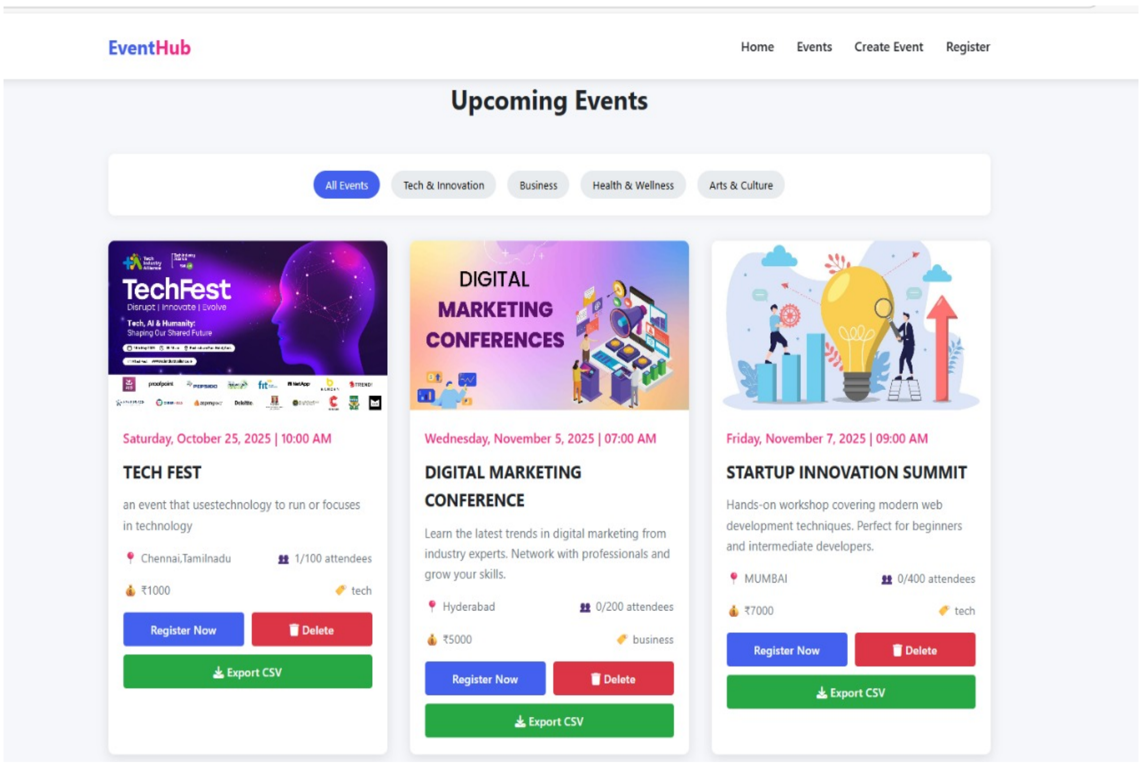
Number of Tickets

1

Special Requirements

Complete Registration

Figure 9.2: **Registration Form**



EventHub

Home Events Create Event Register

Upcoming Events

All Events | Tech & Innovation | Business | Health & Wellness | Arts & Culture

Saturday, October 25, 2025 | 10:00 AM

TECH FEST

an event that usestechnology to run or focuses in technology

Chennai,Tamilnadu 1/100 attendees

₹1000 tech

Register Now Delete

Export CSV

Wednesday, November 5, 2025 | 07:00 AM

DIGITAL MARKETING CONFERENCE

Learn the latest trends in digital marketing from industry experts. Network with professionals and grow your skills.

Hyderabad 0/200 attendees

₹5000 business

Register Now Delete

Export CSV

Friday, November 7, 2025 | 09:00 AM

STARTUP INNOVATION SUMMIT

Hands-on workshop covering modern web development techniques. Perfect for beginners and intermediate developers.

MUMBAI 0/400 attendees

₹7000 tech

Register Now Delete

Export CSV

Figure 9.3: **Upcoming Events**

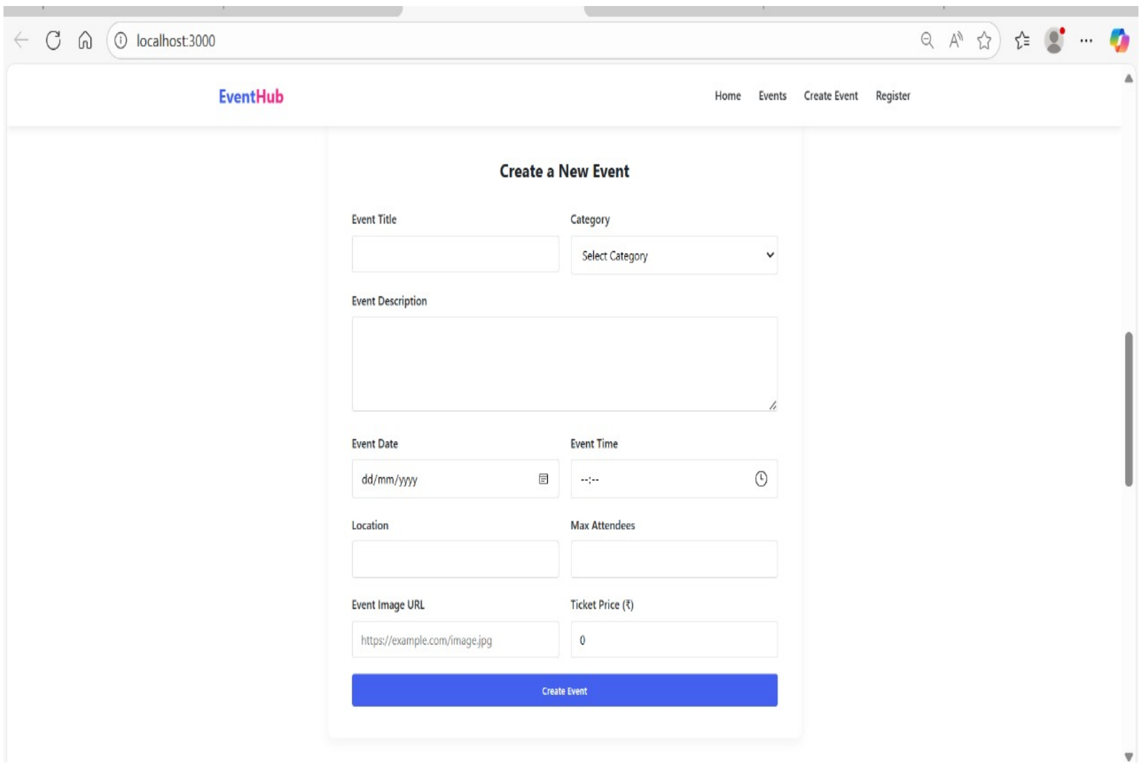


Figure 9.4: **Create Events**

REFERENCES

[1] M. Casciaro, “Node.js Design Patterns,” Packt Publishing, Dec. 29, 2014.

[2] “Express.js – Node.js web application framework,” Express.js, `https://expressjs.com/`.

[3] “SQLite – Lightweight relational database management system,” SQLite, `https://www.sqlite.org/`.

[4] J. Duckett, “HTML and CSS: Design and Build Websites,” Wiley, 2011.

[5] B. McFarland, “CSS3: The Missing Manual,” O’Reilly Media, 2013.