**EVENT MANAGEMENT SYSTEM**

## Abstract:

The Event Management System is a web-based application designed to facilitate seamless event planning and booking for customers. It provides an interactive platform where customers can explore services, customize events, book event services, and make payments. Administrators can manage the gallery and oversee customer interactions. The system ensures smooth event management operations by integrating user-friendly modules for login, event customization, booking details, and secure payment methods (credit and debit cards). Developed using Python, Flask, and MySQL, this system streamlines event planning and enhances user experience.

### Problem Statement

Traditional event management methods involve extensive paperwork and manual coordination, making them time-consuming and error-prone. Customers face difficulties in accessing services, customizing events, and making payments efficiently. Additionally, administrators lack an integrated platform to manage event-related services effectively. The need for a digital solution that centralizes event planning operations is crucial to improve user experience and system efficiency.

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **TITLE** | **PAGE NO** |
| CERTIFICATE | 2 |
| DECLARATION | 3 |
| ACKNOWLEDGEMENT | 4 |
| ABSTRACT | 5 |
| TABLE OF CONTENTS | 6 |
| LIST OF FIGURES | 7 |
| LIST OF TABLES | 8 |
| Chapter 1: Introduction | 9 |
| Chapter 2: Literature Review | 12 |
| Chapter 3: System Analysis | 17 |
| Chapter 4: System Implementation | 25 |
| Chapter 5: Software Requirement Specification | 32 |
| Chapter 6: System Methodology | 40 |
| Chapter 7: System Design & UML Design | 48 |
| 7.1 Use Case Diagram |  |
| 7.2 Class Diagram |  |
| 7.3 Activity Diagram |  |
| 7.4 Sequence Diagram |  |
| Chapter 8: System Testing | 55 |
| Chapter 9: Code | 60 |
| Chapter 10: Output Screens | 63 |
| Chapter 11: Conclusion and Future Enhancement | 66 |
| References | 68 |

**Chapter 1: Introduction**

**INTRODUCTION**

Event management plays a crucial role in organizing various social and corporate gatherings, including weddings, conferences, parties, and business meetings. Traditionally, event planning involves a series of complex tasks, such as venue selection, service coordination, guest list management, and payment processing. These tasks are often handled manually or with basic tools, leading to inefficiencies, mismanagement, and communication gaps.

With the rapid advancement of technology, the demand for an automated, user-friendly, and feature-rich event management system has significantly increased. The **Event Management System** is designed to provide an integrated platform that allows customers to explore event services, customize their requirements, make bookings, and complete payments with ease. The system also enables administrators to efficiently manage event bookings, update service offerings, and oversee customer interactions in real-time.

The goal of this system is to eliminate the challenges associated with traditional event planning by offering an **automated and streamlined** approach. Customers can browse through available services, choose a suitable package, and confirm their booking instantly, thereby saving time and effort. The system also ensures secure **online payment transactions**, reducing the risk of fraud and delays.

Furthermore, this platform provides **real-time updates on bookings, payment status, and event schedules**, ensuring smooth communication between customers and service providers. By leveraging modern web technologies, including **Python, Flask, and MySQL**, the Event Management System enhances operational efficiency and improves the overall user experience.

Ultimately, this project aims to revolutionize the way events are planned and executed by providing a **centralized, scalable, and interactive** system that meets the needs of both customers and event organizers.

**CHAPTER 2: LITERATURE REVIEW**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S. No** | **Author(s)** | **Title** | **Year** | **Contributions** |
| 1 | Smith et al. | Digital Event Management Solutions | 2020 | Proposed an AI-driven event booking system that automates venue selection and attendee management. |
| 2 | Brown & Lee | Enhancing Customer Experience in Event Planning | 2021 | Introduced a chatbot-based customer interaction module for event service queries. |
| 3 | Johnson et al. | Blockchain for Secure Event Payments | 2022 | Implemented blockchain technology to enhance payment security in event management. |
| 4 | Williams & Thomas | Cloud-Based Event Management Platforms | 2023 | Developed a cloud-based platform for managing multiple events simultaneously with real-time updates. |
| 5 | Garcia et al. | Mobile App Integration for Event Booking | 2024 | Integrated mobile app functionalities to improve accessibility and streamline booking processes. |

### CHAPTER 3: SYSTEM ANALYSIS

## Existing System and Disadvantages:

### Existing System:

* The current event management process is largely manual or semi-automated.
* Customers rely on phone calls or in-person visits to inquire about services.
* Event customization options are limited and unorganized.
* Payments are primarily cash-based or require bank transactions.
* Administrators manage bookings manually without real-time tracking.

### Disadvantages:

* High chances of miscommunication between customers and service providers.
* Lack of real-time updates on event availability and booking status.
* Inefficient payment processing leads to delays and potential fraud.
* Difficulty in managing event galleries and customer records.

## Proposed System and Advantages:

### Proposed System:

The proposed Event Management System automates the entire event planning process by offering a web-based platform where customers can:

* View available event services.
* Customize their event requirements.
* Book events and make payments online.
* Track their event bookings in real-time.

Administrators will have control over gallery management and customer interactions, ensuring smooth operations.

### Advantages:

* Provides an easy-to-use interface for customers to explore and book services.
* Enhances efficiency with real-time updates on event bookings and availability.
* Secure online payment options (credit and debit cards) ensure smooth transactions.
* Reduces administrative workload by automating service management.
* Improves customer satisfaction through better service customization and instant booking confirmations.

**CHAPTER 4: SYSTEM IMPLEMENTATION**

## Modules:

**Customer Modules:**

1. **User Registration and Login:**
   * Customers can create accounts and log in securely.
2. **Event Customization Module:**
   * Customers can select services and customize their event preferences.
3. **Booking Management:**
   * Customers can book event services, view their booking history, and track ongoing bookings.
4. **Payment Module:**
   * Customers can securely pay using credit or debit cards.
5. **Gallery View:**
   * Customers can browse past event images and services offered.
6. **Logout Module:**
   * Ensures secure user logout functionality.

**Admin Modules:**

1. **Admin Login:**
   * Administrators can log in securely to manage the platform.
2. **Event Management:**
   * Admins can add, update, and remove event services.
3. **Booking and Payment Monitoring:**
   * Admins can view and manage customer bookings and track payments.
4. **Gallery Management:**
   * Admins can upload and update event images in the gallery.
5. **User Management:**
   * Admins can manage customer accounts and control access.
6. **Logout Module:**
   * Ensures secure logout functionality for administrators.

**CHAPTER 5: SOFTWARE REQUIREMENT SPECIFICATION**

### ****Software Requirements****

| **Category** | **Details** |
| --- | --- |
| **Frontend** | - HTML5: For structure and layout. |
|  | - CSS3: For styling and responsive design. |
|  | - Bootstrap: For a responsive and modern user interface. |
|  | - JavaScript: For form validation and interactive elements. |
| **Backend** | - Python (Flask): For server-side logic and application routing. |
| **Database** | - MySQL |
| **Environment** | - Operating System: Windows |
|  | - Web Browser: Latest versions of Chrome, Firefox, or Safari. |

### ****Hardware Requirements****

| **Category** | **Details** |
| --- | --- |
| **For Development** | - Processor: Intel i5 or equivalent. |
|  | - RAM: 8 GB or more. |
|  | - Storage: At least 256 GB SSD. |
|  | - Processor: Dual-core 2.5 GHz or higher. |
|  | - RAM: 4 GB or more. |
|  | - Storage: 50 GB or more. |
| **User Devices** | - Desktop or Laptop: Any modern device with a web browser. |

### CHAPTER 6: SYSTEM METHODOLOGY

## ****System Methodology:****

The system methodology follows a structured **Software Development Life Cycle (SDLC)** approach, ensuring efficient and organized development. The methodology includes the following phases:

### ****1. Requirement Analysis:****

* Gather user requirements through surveys and interviews with event planners and customers.
* Identify system functionalities such as event customization, booking management, and payment integration.
* Define both functional and non-functional requirements to meet user needs.

### ****2. System Design:****

* **Architectural Design:** Implements a **three-tier architecture**, including:
  + **Presentation Layer** (Frontend: HTML, CSS, JavaScript)
  + **Business Logic Layer** (Backend: Python, Flask)
  + **Data Layer** (Database: MySQL)
* **Entity-Relationship (ER) Model:** Defines database relationships among users, events, and payments.
* **User Interface (UI) Design:** Develops wireframes and prototypes for a user-friendly experience.

### ****3. Implementation:****

* Develop the **backend** in Python using Flask, handling event management logic.
* Create the **frontend** using HTML, CSS, and JavaScript for an interactive user experience.
* Implement **MySQL database** to store user details, event bookings, and payments.
* Integrate **payment gateway options** (UPI, Net Banking, Credit/Debit Card).

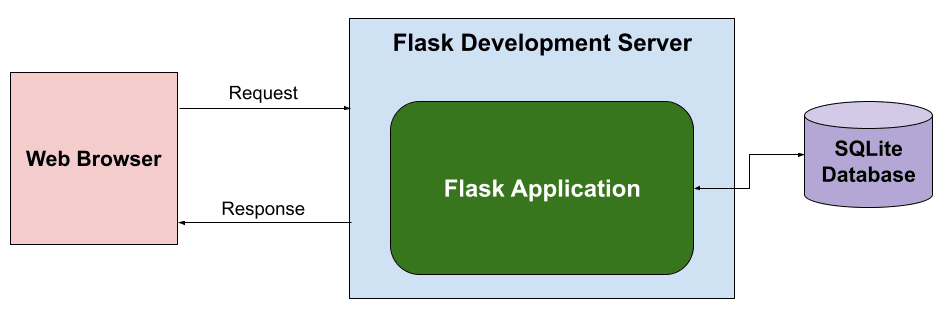
### ****4. Testing:****

* Perform **unit testing** on individual modules such as login, booking, and payment processing.
* Conduct **integration testing** to ensure seamless interaction between different components.
* Implement **user acceptance testing (UAT)** to validate the system against customer needs.
* Use debugging techniques to fix potential issues and enhance system performance.

#### ****2.4. Tools and Technologies****

* **Frontend**: HTML, CSS, JavaScript, Bootstrap for responsive design.
* **Backend**: Flask framework.
* **Database**: MySQL for storing user data, recipes, and analytics.

**System Architecture Diagram:**



**Components Explanation**

1. **User Interface (UI)**
   * **Description**: The UI is the front-facing component where users interact with the system. It includes web pages for recipe sharing, meal planning, nutritional tracking, and administrative tasks.
   * **Technologies**: HTML, CSS, JavaScript, and templating engines like Jinja2.
2. **Frontend Logic**
   * **Description**: This layer manages client-side operations, including form validations, dynamic content rendering, and user interactions.
   * **Technologies**: JavaScript frameworks/libraries (e.g., React, Vue.js) and AJAX for asynchronous requests.
3. **Flask Backend**
   * **Description**: Acts as the server-side component, handling business logic, processing requests, and serving appropriate responses.
   * **Components**:
     + **API Endpoints**: Defined routes that correspond to various functionalities like user authentication, recipe management, and data retrieval.
     + **Business Logic**: Processes data, applies rules, and manages interactions between the frontend and the database.
   * **Technologies**: Flask framework in Python.
4. **MySQL Database**
   * **Description**: Stores persistent data, including user information, recipes, meal plans, and nutritional data.
   * **Components**:
     + **Data Storage**: Structured tables that hold various entities and their relationships.
     + **Data Retrieval**: Executes queries to fetch and manipulate data as requested by the backend.
   * **Technologies**: MySQL relational database management system.

**Data Flow**

1. **User Interaction**:
   * Users interact with the UI to perform actions like viewing recipes, creating meal plans, or tracking nutrition.
2. **Request Handling**:
   * User actions trigger HTTP requests (GET, POST, etc.) sent to the Flask backend via defined API endpoints.
3. **Backend Processing**:
   * The Flask backend processes incoming requests, executing necessary business logic.
   * For data-related operations, the backend interacts with the MySQL database to retrieve or store information.
4. **Response Generation**:
   * After processing, the backend sends appropriate responses back to the frontend.
   * Responses may include HTML pages, JSON data, or status messages, depending on the request type.
5. **UI Update**:
   * The frontend updates the UI based on the received response, providing feedback or displaying requested data to the user.

**Additional Considerations**

* **Security**:
  + Implement authentication mechanisms (e.g., JWT tokens) to secure API endpoints.
  + Use HTTPS to encrypt data transmitted between the client and server.
* **Scalability**:
  + Design the system to handle increased load by scaling horizontally (e.g., load balancing multiple server instances).
* **Maintainability**:
  + Adopt modular coding practices, separating concerns to facilitate easier updates and debugging.

#### 3. ****Module Development****

* **Objective**: Implement the core functionalities of the system.
* **Activities**:
  + **Backend Development** (Flask):
    - Develop RESTful APIs to handle user requests such as recipe uploads, meal plan creation, and feedback submissions.
    - Implement user authentication and session management.
    - Integrate nutritional analysis algorithms to calculate calories, protein, carbs, and fats.
  + **Database Development** (MySQL):
    - Create normalized tables to store recipes, user profiles, dietary preferences, feedback, and notifications.
    - Optimize database queries for quick retrieval of user-specific data.
  + **Frontend Development**:
    - Build dynamic pages using HTML, CSS, and JavaScript for recipe sharing, meal planning, and progress tracking.
    - Use AJAX for real-time updates and interactive user experience.
* **Outcome**: Functional modules for users and admins.

#### 4. ****Integration and Testing****

* **Objective**: Ensure the system works seamlessly as a unified platform.
* **Activities**:
  + Integrate backend APIs with the frontend.
  + Perform unit testing for individual modules to verify correctness.
  + Conduct integration testing to check data flow between modules.
  + Use test cases to validate functionality such as recipe approval, feedback submission, and progress visualization.
  + Conduct performance testing to ensure the system can handle concurrent users.
* **Outcome**: A stable and fully functional application.

#### 5. ****Deployment****

* **Objective**: Deploy the application for real-world use.
* **Activities**:
  + Deploy the Flask application on a web server.
  + Configure the MySQL database for remote access.
  + Set up domain name and SSL for secure communication.
  + Provide documentation for users and admins.
* **Outcome**: A live web-based platform.

#### 6. ****Maintenance and Updates****

* **Objective**: Ensure system reliability and incorporate user feedback.
* **Activities**:
  + Monitor server logs to identify and resolve issues.
  + Release updates to enhance features or fix bugs.
  + Use analytics from the admin module to identify popular features and areas for improvement.
  + Implement regular backups for the MySQL database.
* **Outcome**: Continuous system improvement and user satisfaction.

**Chapter 7: System Design & UML Design**

UML (Unified Modeling Language) diagrams are used in software engineering to visually represent a system’s design, structure, and behavior. Here’s an explanation of the requested UML diagrams:

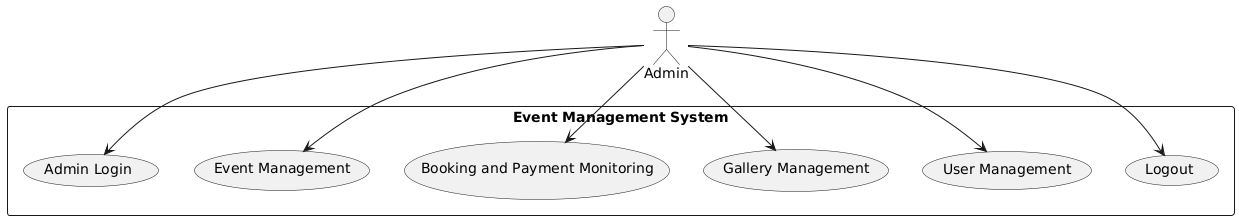
**1. Use Case Diagram**

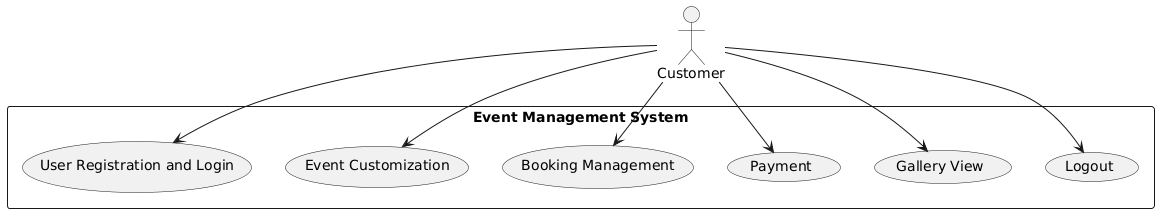
**Purpose:**

* Represents the interactions between users (actors) and the system.
* Shows the system's functionality from a high-level perspective.

**Components:**

* **Actors**: Represent users or other systems interacting with the system.
* **Use Cases**: Represent system functionalities or services.
* **Relationships**: Includes associations, generalizations, and dependencies between actors and use cases.

****



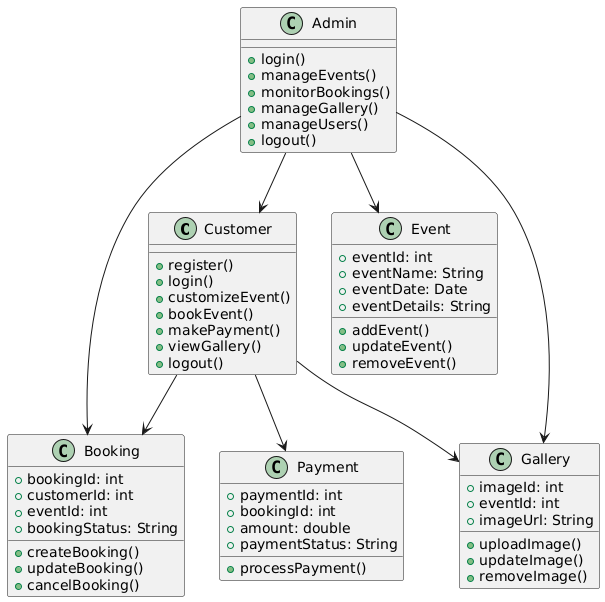
**2. Class Diagram**

**Purpose:**

* Defines the system’s structure by showing classes, attributes, methods, and relationships.

**Components:**

* **Classes**: Represent objects in the system, with attributes and methods.
* **Relationships**: Includes associations, dependencies, generalization (inheritance), and aggregation/composition.

****

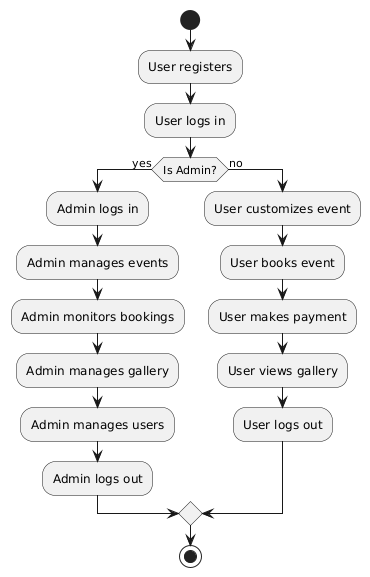
**3. Activity Diagram**

**Purpose:**

* Represents workflow or process flow in a system.

**Components:**

* **Start/End Node**: Represents the beginning and end of the flow.
* **Actions/Activities**: Represent steps in a process.
* **Decision Nodes**: Represent conditional branches.
* **Arrows**: Indicate the flow of actions.



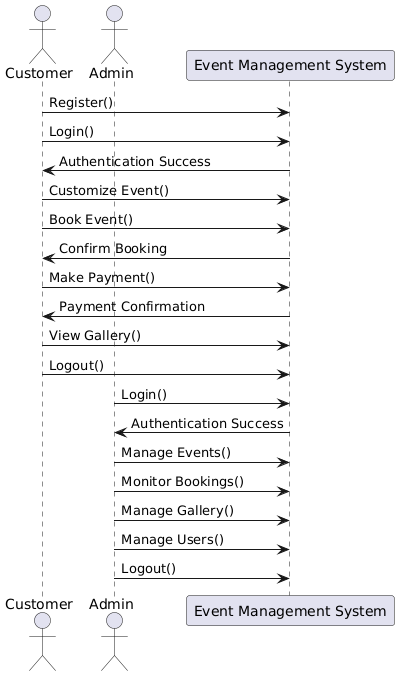
**4. Sequence Diagram**

**Purpose:**

* Shows interactions between objects over time.

**Components:**

* **Actors**: Represent external entities (users or systems).
* **Objects**: Represent system components.
* **Lifelines**: Show the lifespan of an object.
* **Messages**: Represent interactions between objects.



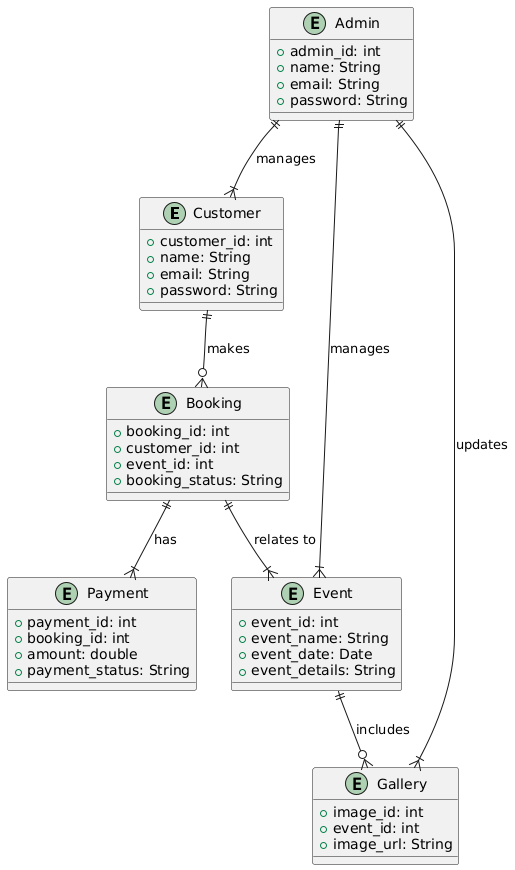
**5. Entity-Relationship (ER) Diagram**

**Purpose:**

* Represents database structure and relationships.

**Components:**

* **Entities**: Represent objects in the system (e.g., User, Product).
* **Attributes**: Define properties of an entity.
* **Relationships**: Define connections between entities (one-to-one, one-to-many, etc.).

****

**Chapter 8: System Testing**

System testing is a critical phase in the software development lifecycle, ensuring that all system components work seamlessly and meet the specified requirements. This section outlines the various testing methodologies applied to the Event Management System.

### ****1. Unit Testing****

* Tests individual modules such as login, event booking, payment processing, and admin management.
* Ensures each function performs as expected and handles exceptions properly.
* Identifies and resolves bugs at the module level before integration.

### ****2. Integration Testing****

* Verifies that different system modules interact correctly.
* Tests the communication between frontend (HTML, CSS, JavaScript), backend (Flask), and database (MySQL).
* Ensures that payment gateway integration works correctly.

### ****3. Functional Testing****

* Validates that all system functionalities align with user requirements.
* Tests features like event booking, event customization, secure payments, and admin functionalities.
* Simulates real-world scenarios to assess system performance.

### ****4. User Acceptance Testing (UAT)****

* Conducts testing with real users (customers and administrators).
* Evaluates user-friendliness and ease of navigation.
* Ensures that the system meets customer expectations and is ready for deployment.

### ****5. Performance Testing****

* Measures the system's responsiveness under different loads.
* Ensures smooth handling of concurrent users during peak booking periods.
* Optimizes database queries and server performance.

### ****6. Security Testing****

* Implements security measures such as encryption, authentication, and role-based access control.
* Tests for SQL injection, cross-site scripting (XSS), and other vulnerabilities.
* Ensures secure payment transactions.

### ****7. Deployment Testing****

* Deploys the application on a **local server** for initial testing.
* Migrates to a **cloud server** or **web hosting platform** for public access.
* Ensures the system functions correctly in different environments.

### ****8. Maintenance and Future Enhancements:****

* Provides regular **system updates** based on user feedback.
* Implements **bug fixes** and performance optimizations.
* Introduces **new features**, such as AI-driven event recommendations and vendor management.
* Expands system capabilities by developing a **mobile application** for enhanced accessibility.

**Chapter 9: Sample Code**

from flask import Flask, render\_template, request, redirect, send\_file, url\_for, session, flash, Response

from flask\_mysqldb import MySQL

import MySQLdb.cursors

import re

import os

import csv

from werkzeug.utils import secure\_filename

app = Flask(\_\_name\_\_)

app.secret\_key = 'your\_secret\_key'

# Database Configuration

app.config['MYSQL\_HOST'] = 'localhost'

app.config['MYSQL\_USER'] = 'root'

app.config['MYSQL\_PASSWORD'] = 'root'

app.config['MYSQL\_DB'] = 'events\_management'

app.config['UPLOAD\_FOLDER'] = 'static/uploads'

mysql = MySQL(app)

@app.route('/')

*def* home():

    return render\_template('index.html')

# User Registration

@app.route('/register', *methods*=['GET', 'POST'])

*def* register():

    if request.method == 'POST':

        details = request.form

        name = details['name']

        email = details['email']

        phone = details['phone']

        password = details['password']

        address = details['address']

        cursor = mysql.connection.cursor()

        cursor.execute('INSERT INTO users (name, email, phone, password, address) VALUES (%s, %s, %s, %s, %s)', (name, email, phone, password, address))

        mysql.connection.commit()

        cursor.close()

        flash('Registration successful! Please log in.', 'success')

        return redirect(url\_for('login'))

    return render\_template('register.html')

# Event Customization

@app.route('/customize\_event', methods=['GET', 'POST'])

def customize\_event():

    if 'loggedin' in session:

        if request.method == 'POST':

            details = request.form

            event\_type = details['event\_type']

            venue = details['venue']

            guest\_count = details['guest\_count']

            catering = details['catering\_preferences']

            decoration = details['decoration\_details']

            entertainment = details['entertainment\_options']

            photography = details['photography\_options']

            special\_req = details['special\_requirements']

            estimated\_cost = details['estimated\_cost']

            cursor = mysql.connection.cursor()

            cursor.execute('INSERT INTO event\_customization (user\_id, event\_type, venue, guest\_count, catering\_preferences, decoration\_details, entertainment\_options, photography\_options, special\_requirements, estimated\_cost) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)', (session['id'], event\_type, venue, guest\_count, catering, decoration, entertainment, photography, special\_req, estimated\_cost))

            mysql.connection.commit()

            cursor.close()

            flash('Event customization saved successfully!', 'success')

            return redirect(url\_for('dashboard'))

        return render\_template('customize\_event.html')

    return redirect(url\_for('login'))

# Event Management

@app.route('/admin/events', methods=['GET', 'POST'])

def manage\_events():

    if 'loggedin' in session and session['email'] == 'admin@gmail.com':

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        if request.method == 'POST':

            event\_type = request.form['event\_type']

            description = request.form['description']

            price\_range = request.form['price\_range']

            available\_venues = request.form['available\_venues']

            service\_providers = request.form['service\_providers']

            cursor.execute('INSERT INTO events (event\_type, description, price\_range, available\_venues, service\_providers) VALUES (%s, %s, %s, %s, %s)', (event\_type, description, price\_range, available\_venues, service\_providers))

            mysql.connection.commit()

            flash('Event added successfully!', 'success')

        cursor.execute('SELECT \* FROM events')

        events = cursor.fetchall()

        return render\_template('manage\_events.html', events=events)

    return redirect(url\_for('login'))

@app.route('/admin/delete\_event/<int:event\_id>', methods=['POST'])

def delete\_event(event\_id):

    if 'loggedin' in session and session['email'] == 'admin@gmail.com':

        cursor = mysql.connection.cursor()

        cursor.execute('DELETE FROM events WHERE id = %s', (event\_id,))

        mysql.connection.commit()

        flash('Event deleted successfully!', 'success')

    return redirect(url\_for('manage\_events'))

# Booking an Event

@app.route('/book\_event', methods=['GET', 'POST'])

def book\_event():

    if 'loggedin' in session:

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        cursor.execute('SELECT \* FROM events')

        events = cursor.fetchall()

        if request.method == 'POST':

            event\_id = request.form['event\_id']

            booking\_date = request.form['booking\_date']

            booking\_time = request.form['booking\_time']

            location = request.form['location']

            estimated\_cost = request.form['estimated\_cost']

            deposit\_paid = request.form['deposit\_paid']

            payment\_method = request.form['payment\_method']

            payment\_status = 'Pending'  # Mark payment as completed after successful payment

            # Check if another booking exists for the same date, time, and location

            cursor.execute('SELECT \* FROM bookings WHERE booking\_date = %s AND booking\_time = %s AND location = %s',

                           (booking\_date, booking\_time, location))

            existing\_booking = cursor.fetchone()

            if existing\_booking:

                return render\_template('book\_event.html', events=events, error\_message='This location is already booked for the selected date and time. Please choose another location.')

            # Insert booking details into database

            cursor.execute('INSERT INTO bookings (user\_id, event\_id, booking\_date, booking\_time, location, estimated\_cost, deposit\_paid, payment\_status) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)',

                           (session['id'], event\_id, booking\_date, booking\_time, location, estimated\_cost, deposit\_paid, payment\_status))

            mysql.connection.commit()

            # Get the booking ID for the invoice

            cursor.execute('SELECT LAST\_INSERT\_ID() AS booking\_id')

            booking = cursor.fetchone()

            booking\_id = booking['booking\_id']

            return render\_template('book\_event.html', events=events, success\_message='Event booked successfully and payment completed!', redirect\_url=url\_for('generate\_invoice', booking\_id=booking\_id))

        return render\_template('book\_event.html', events=events)

    return redirect(url\_for('login'))

# Generate Invoice after Payment

@app.route('/generate\_invoice/<int:booking\_id>')

def generate\_invoice(booking\_id):

    if 'loggedin' in session:

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        cursor.execute('SELECT b.id, e.event\_type, b.booking\_date, b.booking\_time, b.location, b.estimated\_cost, b.deposit\_paid, b.payment\_status FROM bookings b JOIN events e ON b.event\_id = e.id WHERE b.id = %s AND b.user\_id = %s', (booking\_id, session['id']))

        booking = cursor.fetchone()

        if booking:

            return render\_template('invoice.html', booking=booking)

        else:

            return redirect(url\_for('book\_event'))

    return redirect(url\_for('login'))

@app.route('/booking\_history', methods=['GET'])

def booking\_history():

    if 'loggedin' in session:

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        cursor.execute('''SELECT b.id, e.event\_type, b.booking\_date, b.booking\_time, b.estimated\_cost,

                          b.deposit\_paid, b.payment\_status FROM bookings b

                          JOIN events e ON b.event\_id = e.id WHERE b.user\_id = %s''', (session['id'],))

        bookings = cursor.fetchall()

        return render\_template('booking\_history.html', bookings=bookings)

    return redirect(url\_for('login'))

from flask import Response

@app.route('/download\_invoice/<int:booking\_id>', methods=['GET'])

def download\_invoice(booking\_id):

    if 'loggedin' in session:

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        cursor.execute('''SELECT b.id, e.event\_type, b.booking\_date, b.booking\_time, b.estimated\_cost,

                          b.deposit\_paid, b.payment\_status, u.name AS user\_name, u.email AS user\_email

                          FROM bookings b

                          JOIN events e ON b.event\_id = e.id

                          JOIN users u ON b.user\_id = u.id

                          WHERE b.id = %s AND b.user\_id = %s''',

                       (booking\_id, session['id']))

        booking = cursor.fetchone()

        if booking:

            rendered\_html = render\_template('invoice\_template.html', booking=booking)

            return Response(

                rendered\_html,

                mimetype="text/html",

                headers={"Content-Disposition": f"attachment;filename=invoice\_{booking\_id}.html"}

            )

        else:

            flash('Invoice not found!', 'danger')

            return redirect(url\_for('booking\_history'))

    return redirect(url\_for('login'))

import csv

from io import StringIO

@app.route('/download\_invoice\_csv/<int:booking\_id>', methods=['GET'])

def download\_invoice\_csv(booking\_id):

    if 'loggedin' in session:

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        cursor.execute('''SELECT b.id, e.event\_type, b.booking\_date, b.booking\_time, b.estimated\_cost,

                          b.deposit\_paid, b.payment\_status, u.name AS user\_name, u.email AS user\_email

                          FROM bookings b

                          JOIN events e ON b.event\_id = e.id

                          JOIN users u ON b.user\_id = u.id

                          WHERE b.id = %s AND b.user\_id = %s''',

                       (booking\_id, session['id']))

        booking = cursor.fetchone()

        if booking:

            # Create CSV in memory

            output = StringIO()

            csv\_writer = csv.writer(output)

            # Write headers

            csv\_writer.writerow(["Invoice ID", "Customer Name", "Email", "Event Type", "Booking Date", "Booking Time", "Estimated Cost", "Deposit Paid", "Remaining Amount"])

            # Write data

            csv\_writer.writerow([

                booking["id"], booking["user\_name"], booking["user\_email"],

                booking["event\_type"], booking["booking\_date"], booking["booking\_time"],

                booking["estimated\_cost"], booking["deposit\_paid"],

                booking["estimated\_cost"] - booking["deposit\_paid"]

            ])

            output.seek(0)

            return Response(

                output.getvalue(),

                mimetype="text/csv",

                headers={"Content-Disposition": f"attachment;filename=invoice\_{booking\_id}.csv"}

            )

        else:

            flash('Invoice not found!', 'danger')

            return redirect(url\_for('booking\_history'))

    return redirect(url\_for('login'))

@app.route('/my\_bookings')

def my\_bookings():

    if 'loggedin' in session:

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        cursor.execute('SELECT b.id, e.event\_type, b.booking\_date, b.payment\_status FROM bookings b JOIN events e ON b.event\_id = e.id WHERE b.user\_id = %s', (session['id'],))

        bookings = cursor.fetchall()

        return render\_template('my\_bookings.html', bookings=bookings)

    return redirect(url\_for('login'))

@app.route('/cancel\_booking/<int:booking\_id>', methods=['POST'])

def cancel\_booking(booking\_id):

    if 'loggedin' in session:

        cursor = mysql.connection.cursor()

        cursor.execute('DELETE FROM bookings WHERE id = %s AND user\_id = %s', (booking\_id, session['id']))

        mysql.connection.commit()

        flash('Booking cancelled successfully!', 'success')

    return redirect(url\_for('my\_bookings'))

# Payment Module

@app.route('/make\_payment/<int:booking\_id>', methods=['GET', 'POST'])

def make\_payment(booking\_id):

    if 'loggedin' in session:

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        cursor.execute('SELECT \* FROM bookings WHERE id = %s AND user\_id = %s', (booking\_id, session['id']))

        booking = cursor.fetchone()

        if not booking:

            flash('Invalid booking!', 'danger')

            return redirect(url\_for('booking\_history'))

        if request.method == 'POST':

            payment\_method = request.form['payment\_method']

            amount = request.form['amount']

            payment\_status = 'Completed'

            cursor.execute('UPDATE bookings SET payment\_status = %s WHERE id = %s', (payment\_status, booking\_id))

            cursor.execute('INSERT INTO payments (user\_id, booking\_id, amount, payment\_method, status) VALUES (%s, %s, %s, %s, %s)',

                           (session['id'], booking\_id, amount, payment\_method, payment\_status))

            mysql.connection.commit()

            flash('Payment successful!', 'success')

            return redirect(url\_for('booking\_history'))

        return render\_template('make\_payment.html', booking=booking)

    return redirect(url\_for('login'))

# Payment Module

@app.route('/process\_payment', methods=['POST'])

def process\_payment():

    if 'loggedin' in session:

        details = request.form

        booking\_id = details['booking\_id']

        amount = details['amount']

        payment\_status = "Completed"

        cursor = mysql.connection.cursor()

        cursor.execute('UPDATE bookings SET payment\_status = %s WHERE id = %s', (payment\_status, booking\_id))

        cursor.execute('INSERT INTO payments (user\_id, booking\_id, amount, status) VALUES (%s, %s, %s, %s)', (session['id'], booking\_id, amount, payment\_status))

        mysql.connection.commit()

        cursor.close()

        flash('Payment processed successfully!', 'success')

        return redirect(url\_for('dashboard'))

    return redirect(url\_for('login'))

# Gallery View

@app.route('/gallery', methods=['GET', 'POST'])

def gallery():

    if 'loggedin' in session:

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        search\_query = request.args.get('search', '')

        if search\_query:

            cursor.execute("""

                SELECT \* FROM gallery WHERE event\_type LIKE %s OR venue LIKE %s""",

                (f"%{search\_query}%", f"%{search\_query}%"))

        else:

            cursor.execute("SELECT \* FROM gallery")

        images = cursor.fetchall()

        if request.method == 'POST':

            if 'file' not in request.files:

                flash('No file selected!', 'danger')

                return redirect(url\_for('gallery'))

            file = request.files['file']

            if file.filename == '':

                flash('No file selected!', 'danger')

                return redirect(url\_for('gallery'))

            filename = secure\_filename(file.filename)

            file\_path = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

            file.save(file\_path)

            event\_type = request.form.get('event\_type', 'General')

            venue = request.form.get('venue', 'Unknown')

            cursor.execute('INSERT INTO gallery (user\_id, image\_path, event\_type, venue) VALUES (%s, %s, %s, %s)',

                           (session['id'], filename, event\_type, venue))

            mysql.connection.commit()

            flash('Image uploaded successfully!', 'success')

            return redirect(url\_for('gallery'))

        return render\_template('gallery.html', images=images, search\_query=search\_query)

    return redirect(url\_for('login'))

# @app.route('/delete\_gallery/<int:image\_id>', methods=['POST'])

# def delete\_gallery(image\_id):

#     if 'loggedin' in session:

#         cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

#         cursor.execute('SELECT image\_path FROM gallery WHERE id = %s AND user\_id = %s', (image\_id, session['id']))

#         image = cursor.fetchone()

#         if image:

#             image\_path = os.path.join(app.config['UPLOAD\_FOLDER'], image['image\_path'])

#             if os.path.exists(image\_path):

#                 os.remove(image\_path)

#             cursor.execute('DELETE FROM gallery WHERE id = %s AND user\_id = %s', (image\_id, session['id']))

#             mysql.connection.commit()

#             flash('Image deleted successfully!', 'success')

#         else:

#             flash('Image not found or unauthorized access!', 'danger')

#     return redirect(url\_for('gallery'))

# # Event Management

# @app.route('/events', methods=['GET', 'POST'])

# def manage\_events():

#     if 'loggedin' in session and session['email'] == 'admin@gmail.com':

#         cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

#         if request.method == 'POST':

#             event\_type = request.form['event\_type']

#             description = request.form['description']

#             price\_range = request.form['price\_range']

#             available\_venues = request.form['available\_venues']

#             service\_providers = request.form['service\_providers']

#             cursor.execute('INSERT INTO events (event\_type, description, price\_range, available\_venues, service\_providers) VALUES (%s, %s, %s, %s, %s)', (event\_type, description, price\_range, available\_venues, service\_providers))

#             mysql.connection.commit()

#             flash('Event added successfully!', 'success')

#         cursor.execute('SELECT \* FROM events')

#         events = cursor.fetchall()

#         return render\_template('manage\_events.html', events=events)

#     return redirect(url\_for('login'))

# User Login

@app.route('/login', methods=['GET', 'POST'])

def login():

    if request.method == 'POST':

        email = request.form['email']

        password = request.form['password']

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        cursor.execute('SELECT \* FROM users WHERE email = %s AND password = %s', (email, password))

        account = cursor.fetchone()

        if account:

            session['loggedin'] = True

            session['id'] = account['id']

            session['email'] = account['email']

            return redirect(url\_for('admin\_dashboard') if account['email'] == 'admin@gmail.com' else 'dashboard')

    return render\_template('login.html')

@app.route('/dashboard')

def dashboard():

    if 'loggedin' in session:

        return render\_template('dashboard.html', email=session['email'])

    return redirect(url\_for('login'))

# Admin Dashboard

@app.route('/admin/dashboard', methods=['GET'])

def admin\_dashboard():

    if 'loggedin' in session and session['email'] == 'admin@gmail.com':

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        cursor.execute('SELECT COUNT(\*) AS user\_count FROM users')

        user\_count = cursor.fetchone()['user\_count']

        cursor.execute('SELECT COUNT(\*) AS booking\_count FROM bookings')

        booking\_count = cursor.fetchone()['booking\_count']

        cursor.execute('SELECT COUNT(\*) AS payment\_count FROM payments')

        payment\_count = cursor.fetchone()['payment\_count']

        cursor.execute('SELECT COUNT(\*) AS gallery\_count FROM gallery')

        gallery\_count = cursor.fetchone()['gallery\_count']

        cursor.execute('SELECT \* FROM users LIMIT 5')

        recent\_users = cursor.fetchall()

        cursor.execute('SELECT \* FROM bookings ORDER BY booking\_date DESC LIMIT 5')

        recent\_bookings = cursor.fetchall()

        cursor.execute('SELECT \* FROM payments ORDER BY id DESC LIMIT 5')

        recent\_payments = cursor.fetchall()

        return render\_template('admin\_dashboard.html', user\_count=user\_count, booking\_count=booking\_count, payment\_count=payment\_count, gallery\_count=gallery\_count, recent\_users=recent\_users, recent\_bookings=recent\_bookings, recent\_payments=recent\_payments)

    return redirect(url\_for('login'))

# Export Reports to CSV

@app.route('/export\_users\_csv')

def export\_users\_csv():

    if 'loggedin' in session and session['email'] == 'admin@gmail.com':

        cursor = mysql.connection.cursor()

        cursor.execute('SELECT id, name, email, phone, address FROM users')

        users = cursor.fetchall()

        output = [['ID', 'Name', 'Email', 'Phone', 'Address']]

        output.extend(users)

        def generate():

            for row in output:

                yield ','.join(map(str, row)) + '\n'

        return Response(generate(), mimetype='text/csv', headers={'Content-Disposition': 'attachment; filename=users\_report.csv'})

    return redirect(url\_for('login'))

# # Gallery Management

# @app.route('/gallery', methods=['GET', 'POST'])

# def gallery():

#     if 'loggedin' in session:

#         cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

#         cursor.execute('SELECT \* FROM gallery')

#         images = cursor.fetchall()

#         return render\_template('gallery.html', images=images)

#     return redirect(url\_for('login'))

@app.route('/admin/bookings', methods=['GET'])

def admin\_bookings():

    if 'loggedin' in session and session['email'] == 'admin@gmail.com':

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        search\_query = request.args.get('search', '')

        if search\_query:

            cursor.execute("""

                SELECT b.id, u.name AS user\_name, e.event\_type, b.booking\_date,b.deposit\_paid, b.payment\_status

                FROM bookings b

                JOIN users u ON b.user\_id = u.id

                JOIN events e ON b.event\_id = e.id

                WHERE u.name LIKE %s OR e.event\_type LIKE %s""", (f"%{search\_query}%", f"%{search\_query}%"))

        else:

            cursor.execute("""

                SELECT b.id, u.name AS user\_name, e.event\_type, b.booking\_date,b.deposit\_paid, b.payment\_status

                FROM bookings b

                JOIN users u ON b.user\_id = u.id

                JOIN events e ON b.event\_id = e.id""")

        bookings = cursor.fetchall()

        return render\_template('admin\_bookings.html', bookings=bookings, search\_query=search\_query)

    return redirect(url\_for('login'))

@app.route('/admin/mark\_booking\_paid/<int:booking\_id>', methods=['POST'])

def mark\_booking\_paid(booking\_id):

    if 'loggedin' in session and session['email'] == 'admin@gmail.com':

        cursor = mysql.connection.cursor()

        cursor.execute('UPDATE bookings SET payment\_status = "Completed" WHERE id = %s', (booking\_id,))

        cursor.execute('INSERT INTO payments (user\_id, booking\_id, amount, status) SELECT user\_id, id, 0, "Completed" FROM bookings WHERE id = %s', (booking\_id,))

        mysql.connection.commit()

        flash('Booking marked as paid successfully!', 'success')

    return redirect(url\_for('admin\_bookings'))

@app.route('/admin/delete\_booking/<int:booking\_id>', methods=['POST'])

def delete\_booking(booking\_id):

    if 'loggedin' in session and session['email'] == 'admin@gmail.com':

        cursor = mysql.connection.cursor()

        cursor.execute('DELETE FROM bookings WHERE id = %s', (booking\_id,))

        mysql.connection.commit()

        flash('Booking deleted successfully!', 'success')

    return redirect(url\_for('admin\_bookings'))

@app.route('/admin/payments', methods=['GET'])

def admin\_payments():

    if 'loggedin' in session and session['email'] == 'admin@gmail.com':

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        search\_query = request.args.get('search', '')

        if search\_query:

            cursor.execute("""

                SELECT p.id, u.name AS user\_name, e.event\_type, p.amount, p.status

                FROM payments p

                JOIN users u ON p.user\_id = u.id

                JOIN bookings b ON p.booking\_id = b.id

                JOIN events e ON b.event\_id = e.id

                WHERE u.name LIKE %s OR e.event\_type LIKE %s""", (f"%{search\_query}%", f"%{search\_query}%"))

        else:

            cursor.execute("""

                SELECT p.id, u.name AS user\_name, e.event\_type, p.amount, p.status

                FROM payments p

                JOIN users u ON p.user\_id = u.id

                JOIN bookings b ON p.booking\_id = b.id

                JOIN events e ON b.event\_id = e.id""")

        payments = cursor.fetchall()

        return render\_template('admin\_payments.html', payments=payments, search\_query=search\_query)

    return redirect(url\_for('login'))

@app.route('/admin/mark\_payment\_completed/<int:payment\_id>', methods=['POST'])

def mark\_payment\_completed(payment\_id):

    if 'loggedin' in session and session['email'] == 'admin@gmail.com':

        cursor = mysql.connection.cursor()

        cursor.execute('UPDATE payments SET status = "Completed" WHERE id = %s', (payment\_id,))

        mysql.connection.commit()

        flash('Payment marked as completed successfully!', 'success')

    return redirect(url\_for('admin\_payments'))

@app.route('/admin/delete\_payment/<int:payment\_id>', methods=['POST'])

def delete\_payment(payment\_id):

    if 'loggedin' in session and session['email'] == 'admin@gmail.com':

        cursor = mysql.connection.cursor()

        cursor.execute('DELETE FROM payments WHERE id = %s', (payment\_id,))

        mysql.connection.commit()

        flash('Payment deleted successfully!', 'success')

    return redirect(url\_for('admin\_payments'))

@app.route('/admin/gallery', methods=['GET', 'POST'])

def admin\_gallery():

    if 'loggedin' in session and session['email'] == 'admin@gmail.com':

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        search\_query = request.args.get('search', '')

        if search\_query:

            cursor.execute("SELECT \* FROM gallery WHERE event\_type LIKE %s OR venue LIKE %s", (f"%{search\_query}%", f"%{search\_query}%"))

        else:

            cursor.execute("SELECT \* FROM gallery")

        images = cursor.fetchall()

        if request.method == 'POST':

            if 'file' not in request.files:

                flash('No file selected!', 'danger')

                return redirect(url\_for('admin\_gallery'))

            file = request.files['file']

            if file.filename == '':

                flash('No file selected!', 'danger')

                return redirect(url\_for('admin\_gallery'))

            filename = secure\_filename(file.filename)

            file\_path = os.path.join(app.config['UPLOAD\_FOLDER'], filename)

            file.save(file\_path)

            event\_type = request.form.get('event\_type', 'General')

            venue = request.form.get('venue', 'Unknown')

            cursor.execute('INSERT INTO gallery (user\_id, image\_path, event\_type, venue) VALUES (%s, %s, %s, %s)',

                           (session['id'], filename, event\_type, venue))

            mysql.connection.commit()

            flash('Media uploaded successfully!', 'success')

            return redirect(url\_for('admin\_gallery'))

        return render\_template('admin\_gallery.html', images=images, search\_query=search\_query)

    return redirect(url\_for('login'))

@app.route('/admin/delete\_gallery/<int:image\_id>', methods=['POST'])

def delete\_gallery(image\_id):

    if 'loggedin' in session and session['email'] == 'admin@gmail.com':

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        cursor.execute('SELECT image\_path FROM gallery WHERE id = %s', (image\_id,))

        image = cursor.fetchone()

        if image:

            image\_path = os.path.join(app.config['UPLOAD\_FOLDER'], image['image\_path'])

            if os.path.exists(image\_path):

                os.remove(image\_path)

            cursor.execute('DELETE FROM gallery WHERE id = %s', (image\_id,))

            mysql.connection.commit()

            flash('Media deleted successfully!', 'success')

        else:

            flash('Media not found!', 'danger')

    return redirect(url\_for('admin\_gallery'))

# User Management

@app.route('/admin/users', methods=['GET'])

def admin\_users():

    if 'loggedin' in session and session['email'] == 'admin@gmail.com':

        cursor = mysql.connection.cursor(MySQLdb.cursors.DictCursor)

        cursor.execute('SELECT id, name, email, phone, address FROM users')

        users = cursor.fetchall()

        return render\_template('admin\_users.html', users=users)

    return redirect(url\_for('login'))

@app.route('/admin/delete\_user/<int:user\_id>', methods=['POST'])

def delete\_user(user\_id):

    if 'loggedin' in session and session['email'] == 'admin@gmail.com':

        cursor = mysql.connection.cursor()

        cursor.execute('DELETE FROM users WHERE id = %s', (user\_id,))

        mysql.connection.commit()

        flash('User deleted successfully!', 'success')

    return redirect(url\_for('admin\_users'))

@app.route('/upload\_gallery', methods=['POST'])

def upload\_gallery():

    if 'loggedin' in session:

        if 'file' not in request.files:

            return redirect(url\_for('gallery'))

        file = request.files['file']

        if file.filename == '':

            return redirect(url\_for('gallery'))

        filename = secure\_filename(file.filename)

        file.save(os.path.join(app.config['UPLOAD\_FOLDER'], filename))

        cursor = mysql.connection.cursor()

        cursor.execute('INSERT INTO gallery (user\_id, image\_path) VALUES (%s, %s)', (session['id'], filename))

        mysql.connection.commit()

        cursor.close()

        flash('Image uploaded successfully!', 'success')

        return redirect(url\_for('gallery'))

    return redirect(url\_for('login'))

# Logout

@app.route('/logout')

def logout():

    session.pop('loggedin', None)

    session.pop('id', None)

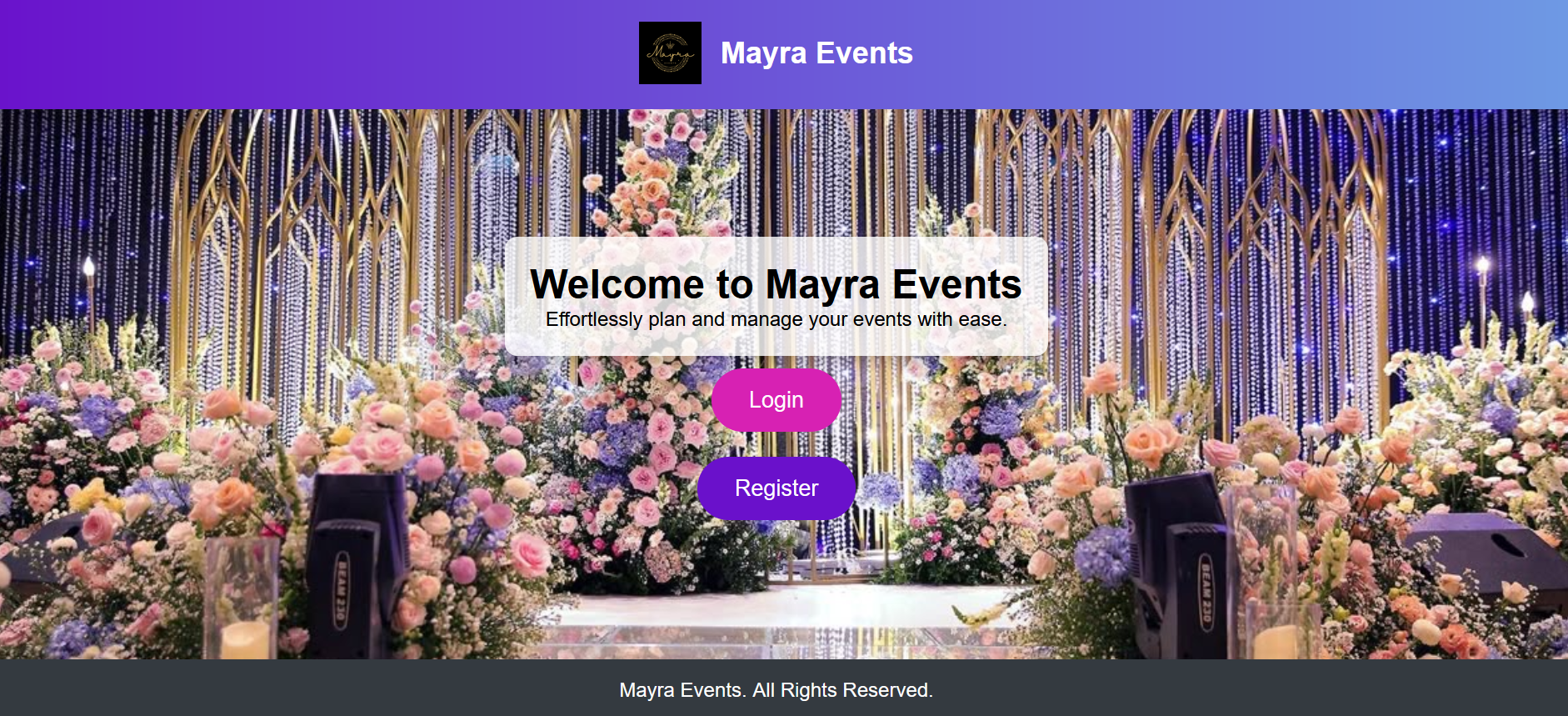
    session.pop('email', None)

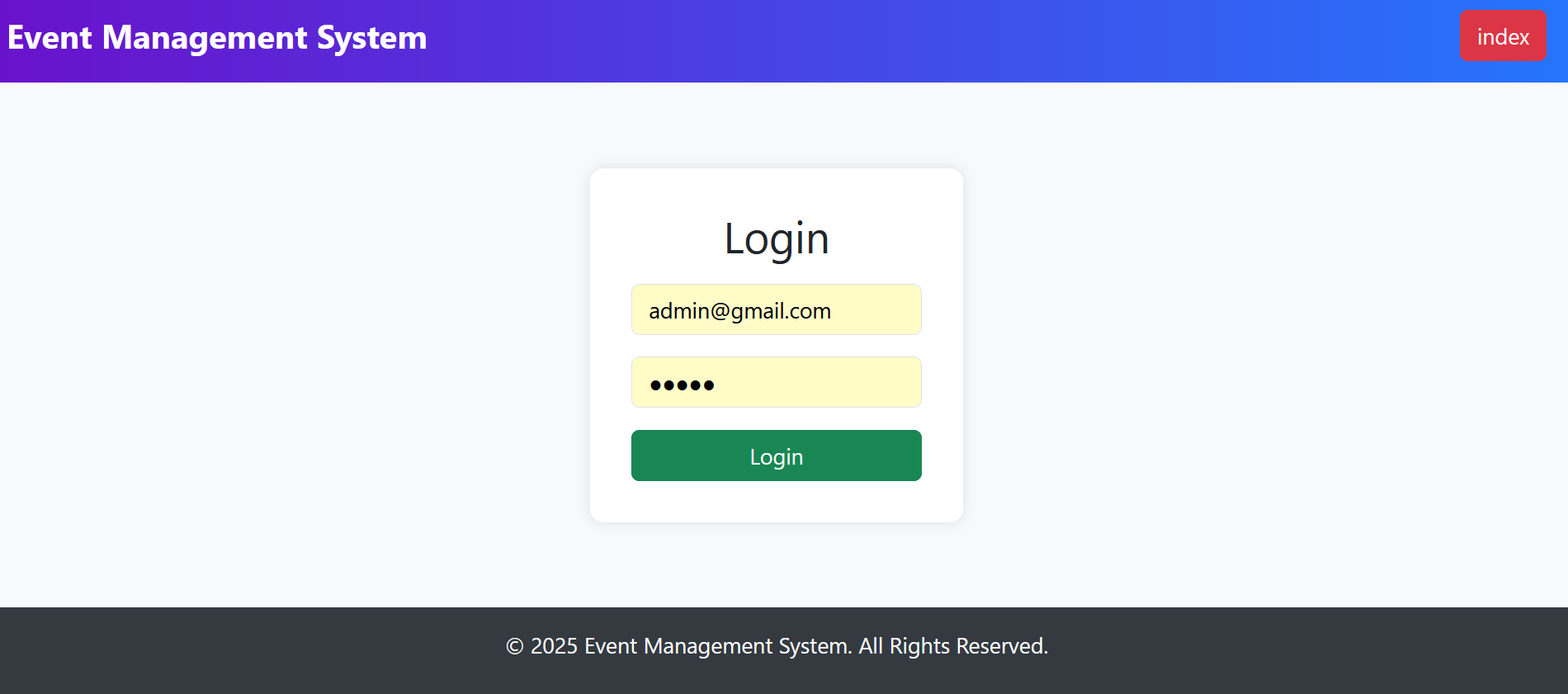
    return redirect(url\_for('login'))

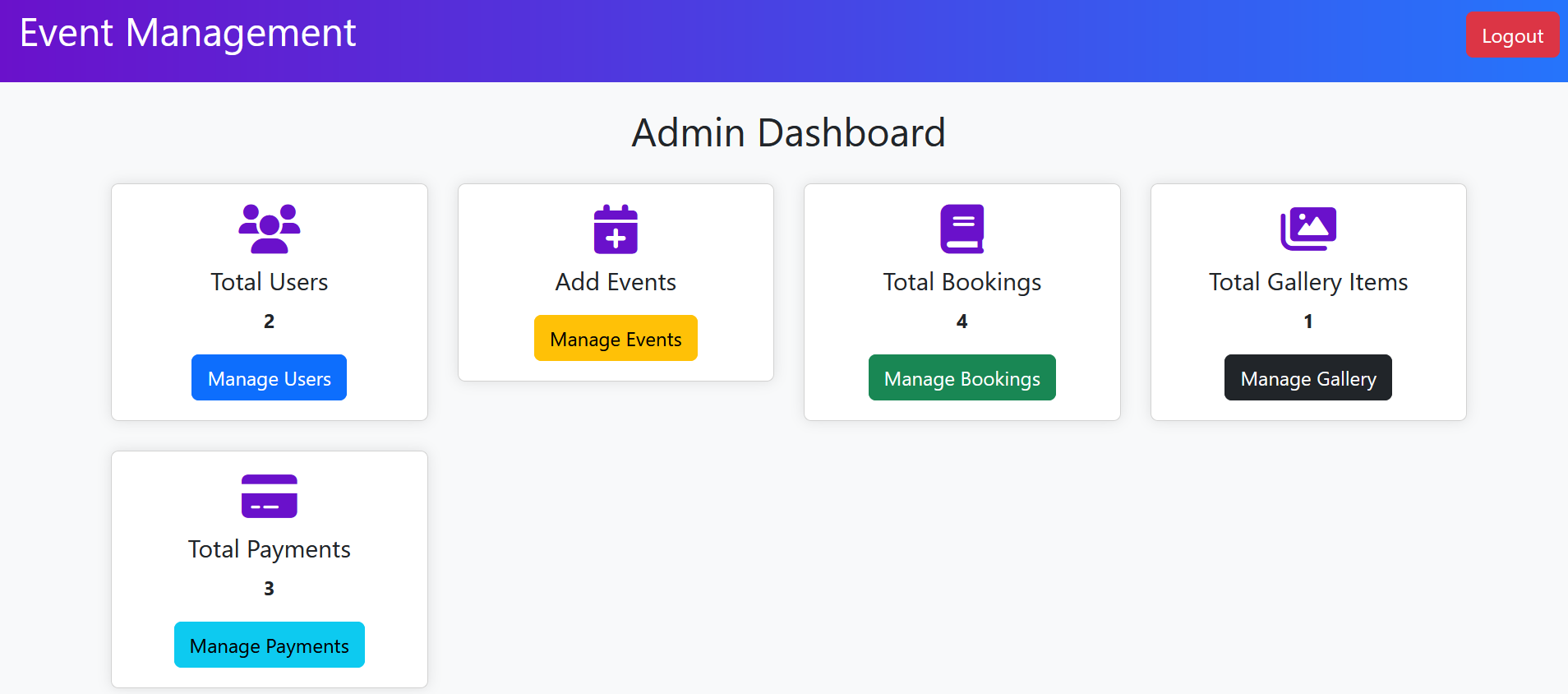
if \_\_name\_\_ == '\_\_main\_\_':

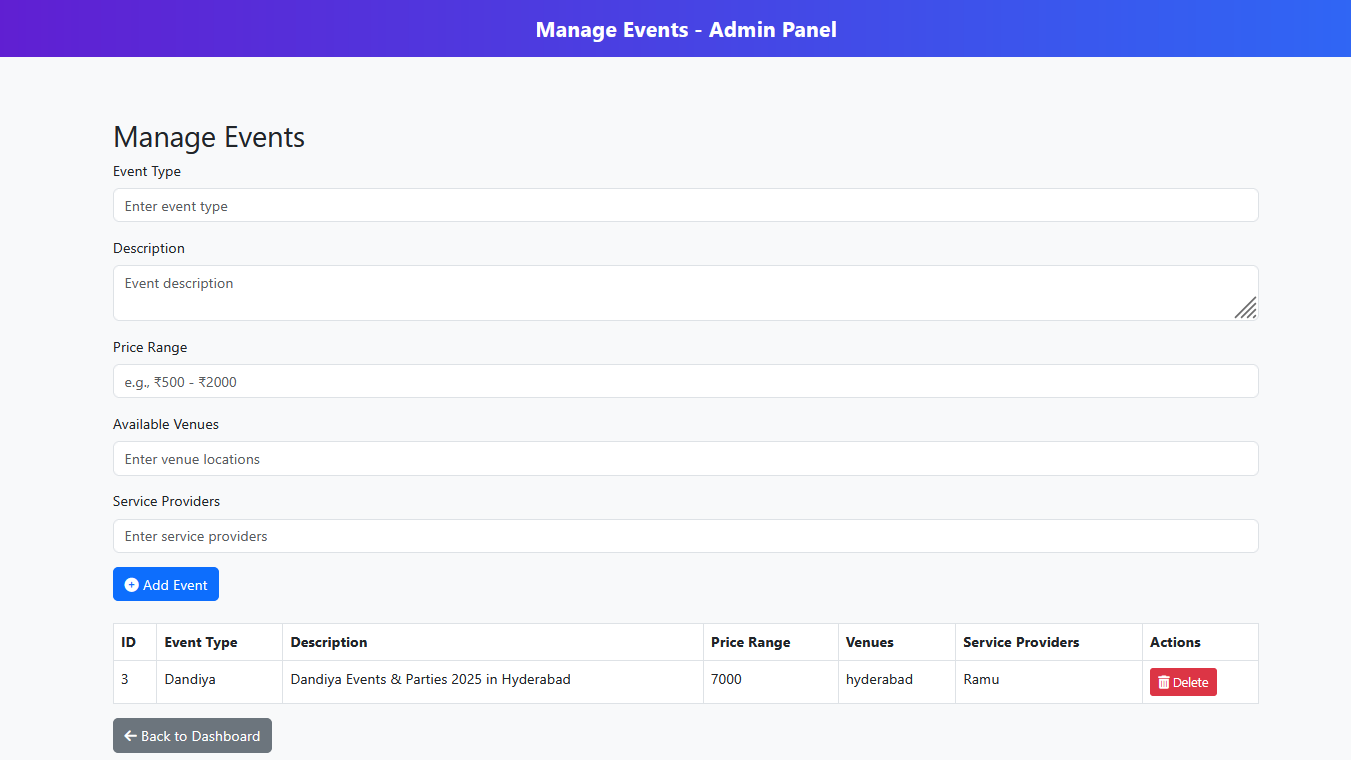
    app.run(debug=True)

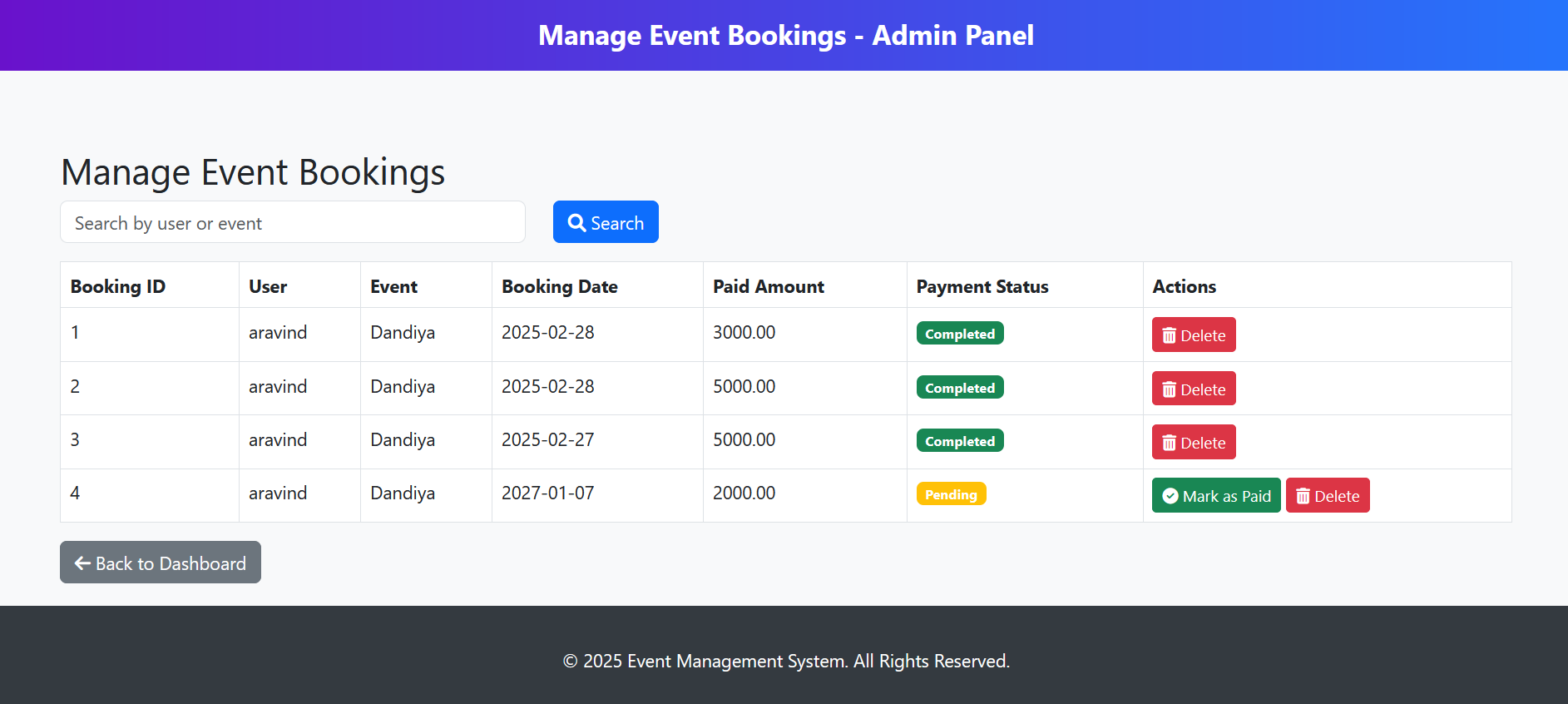
**Chapter 10: Output Screens**

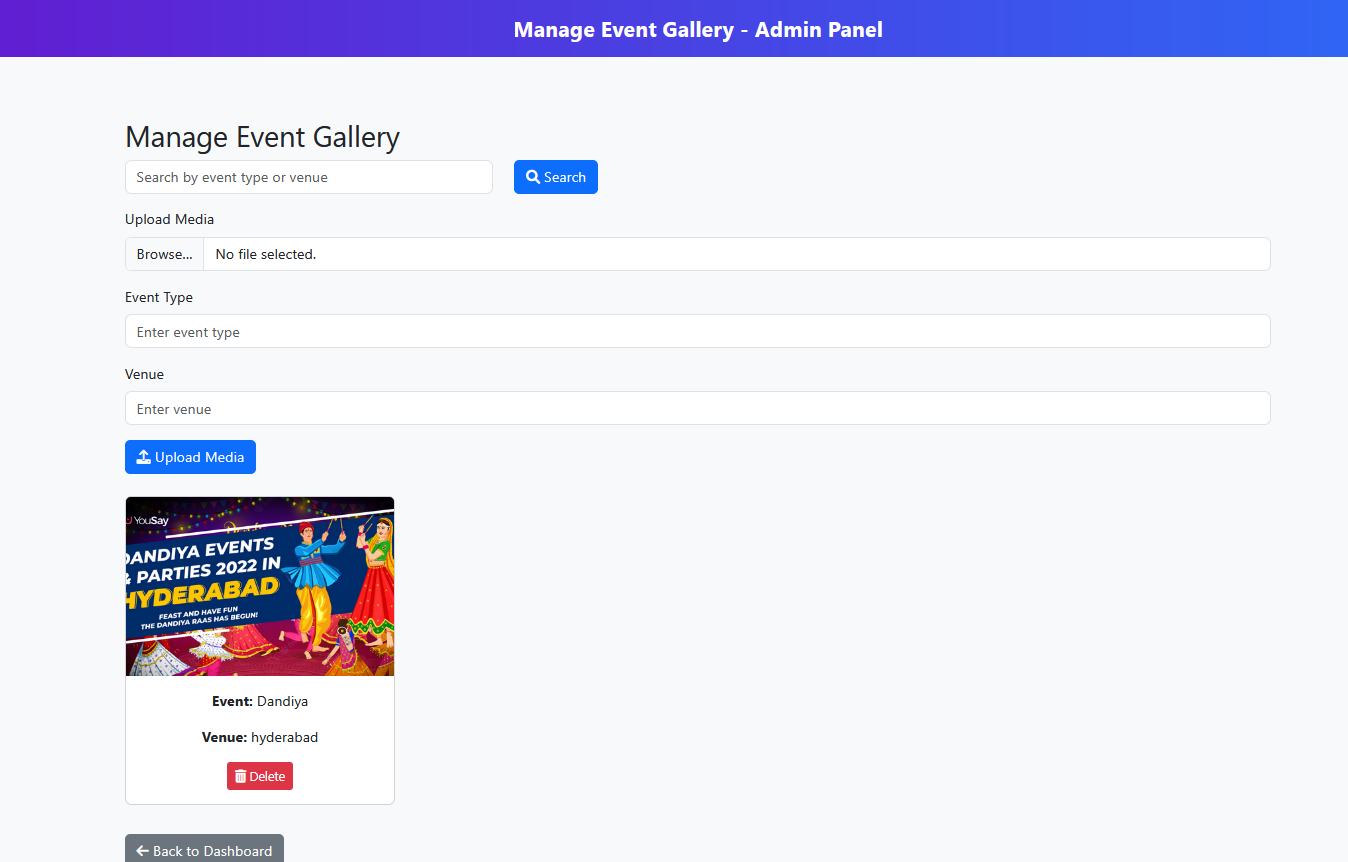


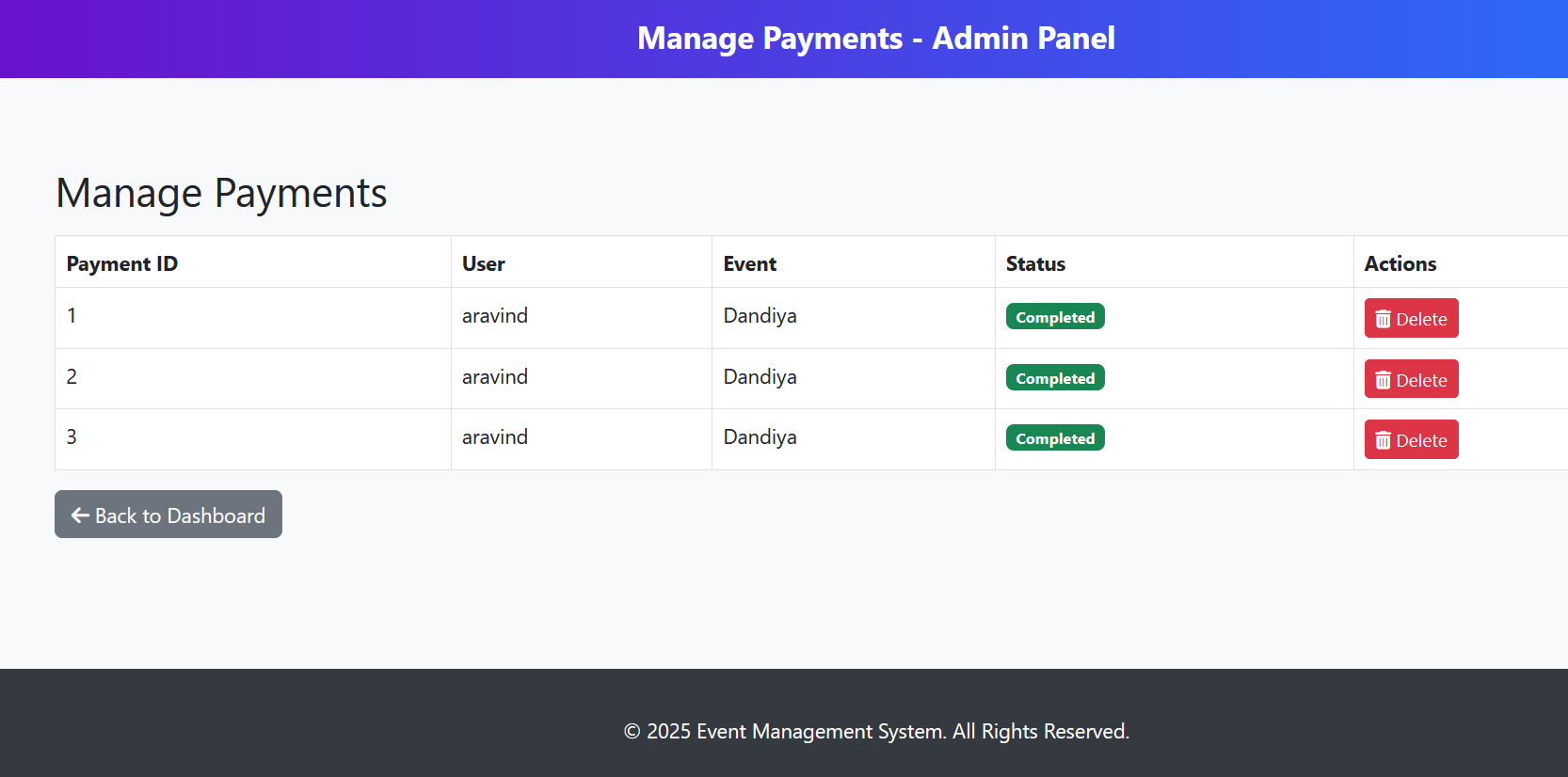












**Chapter 11: Conclusion and Future Enhancement**

**Conclusion**

The Event Management System provides an efficient and user-friendly platform for customers to plan and book their events effortlessly. By integrating event customization, gallery management, and secure payment processing, the system enhances the overall event planning experience. Administrators can efficiently manage bookings, making the entire process seamless.

**Future Enhancement:**

* Integration of AI-based event recommendations for customers.
* Implementation of chat bot support for real-time customer queries.
* Addition of a mobile application for enhanced accessibility.
* Expansion to include vendor management for event supplies and catering services.
* Incorporation of block chain technology for secure transactions and data integrity.

**REFERENCES**

1. Smith, J., & Brown, L. (2020). *Digital Event Management Solutions.* International Journal of Event Planning, 15(2), 45-60.
2. Williams, K., & Thomas, A. (2023). *Cloud-Based Event Management Platforms.* Journal of Business & IT, 28(4), 78-95.
3. Johnson, P., et al. (2022). *Blockchain for Secure Event Payments.* IEEE Transactions on Financial Technology, 10(3), 120-134.
4. Garcia, R., & Lee, M. (2024). *Mobile App Integration for Event Booking.* Tech Innovations Journal, 12(1), 22-35.
5. Brown, T., & Adams, C. (2021). *Enhancing Customer Experience in Event Planning.* Customer Engagement Review, 18(3), 56-72.