Index

Introduction

The project, Web-Based Event Management System, is a dynamic and user-friendly platform, designed to streamline the process of organizing and managing events, providing both organizers and participants with a seamless and efficient experience.

The platform enables users to create, manage, and participate in events with ease, ensuring a centralized and efficient approach to event organization.

Objectives of the Project:

- Simplification of Event Management: Automating the creation and registration process for events to save time and reduce errors.
- Enhanced User Accessibility: Providing an intuitive interface that caters to both event organizers and participants.
- Efficient Data Handling: Ensuring smooth coordination by organizing and managing event and participant details effectively using python and MySQL.

Key Features:

1. Create Events: Organizers can easily create new events by entering essential details such as event name, date, time, and description.
2. Register for Events: Participants can register for events with just a few clicks, ensuring hassle-free enrollment.
3. View Upcoming Events: Users can explore a list of upcoming events, with details displayed in an organized and user-friendly manner.
4. Manage Participants: Organizers can view, edit, or delete participant information, ensuring smooth coordination and management.

Languages

1. HTML

- Creating user interfaces for features such as event creation, registration, and dashboards.
- Displaying data like upcoming events and participant lists in a structured manner.

2. Tailwind CSS

- Enhancing the visual appeal of the web application.
- Ensuring responsiveness and consistency across different devices.
- Simplifying the styling process, saving time, and reducing CSS file size.

3. Python

- Handling backend logic and processing user requests.
- Managing data flow between the frontend and the database.
- Implementing custom functionalities like participant management and event validation.

4. Flask

- Managing routes for different parts of the application (e.g., /create-event, /register).
- Serving HTML templates and integrating them with backend logic.
- Handling user input and connecting with the database.

5. MySQL

- Storing event details, participant information, and registration data.
- Enabling efficient data retrieval and manipulation through SQL queries.
- Providing a secure and scalable solution for data storage.

Data Structures

1. Events Table

```
mysql> desc events;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| id             | int          | NO   | PRI | NULL    | auto_increment |
| event_name     | varchar(255) | NO   | UNI | NULL    |                |
| event_date     | date         | NO   |     | NULL    |                |
| event_location | varchar(255) | NO   |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

2. Participants

```
mysql> desc participants;
+-------------------+--------------+------+-----+---------+----------------+
| Field             | Type         | Null | Key | Default | Extra          |
+-------------------+--------------+------+-----+---------+----------------+
| id                | int          | NO   | PRI | NULL    | auto_increment |
| event_id          | int          | YES  | MUL | NULL    |                |
| participant_name  | varchar(255) | NO   |     | NULL    |                |
| participant_email | varchar(255) | NO   |     | NULL    |                |
| participant_phone | varchar(50)  | YES  |     | NULL    |                |
+-------------------+--------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

3. Participants registered in an event

```
mysql> desc farewell;
+-------------------+--------------+------+-----+---------+----------------+
| Field             | Type         | Null | Key | Default | Extra          |
+-------------------+--------------+------+-----+---------+----------------+
| id                | int          | NO   | PRI | NULL    | auto_increment |
| participant_name  | varchar(255) | NO   |     | NULL    |                |
| participant_email | varchar(255) | NO   |     | NULL    |                |
| participant_phone | varchar(50)  | NO   |     | NULL    |                |
+-------------------+--------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
```

Source Code

```
from flask import Flask, render_template, request, redirect, url_for
from datetime import date
import mysql.connector

app = Flask(__name__)

db_config = {
    'host': 'localhost',
    'user': 'root',
    'password': '', #your password
    'database': '' #databse name
}

def get_db_connection():
    return mysql.connector.connect(**db_config)

def init_database():
    # Creating events table if it doesn't exist
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS events (
        id INT AUTO_INCREMENT PRIMARY KEY,
        event_name VARCHAR(255) NOT NULL UNIQUE,
        event_date DATE NOT NULL,
        event_location VARCHAR(255) NOT NULL
    )
    ''')
    conn.commit()
    cursor.close()
    conn.close()

def create_event_table(event_name):
    # Creating table for participants of a specific event
    table_name = event_name.replace(" ", "_").lower()
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute(f'''
    CREATE TABLE IF NOT EXISTS `{table_name}` (
        id INT AUTO_INCREMENT PRIMARY KEY,
        participant_name VARCHAR(255) NOT NULL,
        participant_email VARCHAR(255) NOT NULL,
        participant_phone VARCHAR(50) NOT NULL
    )
    ''')
    conn.commit()
    cursor.close()
    conn.close()

@app.route('/')
def index():
    # Displays upcoming events
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute('SELECT * FROM events WHERE event_date >= %s ORDER BY
event_date ASC', (date.today(),))
    events = cursor.fetchall()
    cursor.close()
    conn.close()
    return render_template('index.html', events=events)
```

Source Code

```
@app.route('/create', methods=['POST'])
def create():
    # Creatinga new event
    event_name = request.form['event_name']
    event_date = request.form['event_date']
    event_location = request.form['event_location']

    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute("""
    INSERT INTO events (event_name, event_date, event_location)
    VALUES (%s, %s, %s)
    """, (event_name, event_date, event_location))
    conn.commit()
    create_event_table(event_name)

    cursor.close()
    conn.close()

    return redirect('/')

@app.route('/register', methods=['POST'])
def register_participant():
    # Retrieve event_name and other form data
    event_name = request.form.get('event_name')
    participant_name = request.form['participant_name']
    participant_email = request.form['participant_email']
    participant_phone = request.form['participant_phone']

    table_name = event_name.replace(" ", "_").lower()

    conn = get_db_connection()
    cursor = conn.cursor()

    try:
        cursor.execute(f"""
        INSERT INTO `{table_name}` (participant_name, participant_email, participant_phone)
        VALUES (%s, %s, %s)
        """, (participant_name, participant_email, participant_phone))
        conn.commit()
    except mysql.connector.Error as e:
        return f"Database error: {e}", 500
    finally:
        cursor.close()
        conn.close()

    return redirect('/')
```

Source Code

```
@app.route('/event/<event_name>')
def view_event_participants(event_name):
    # View participants of a specific event
    table_name = event_name.replace(" ", "_").lower()

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    cursor.execute('SELECT * FROM events WHERE event_name = %s', (event_name,))
    event = cursor.fetchone()
    cursor.execute(f'SELECT * FROM `{table_name}`')
    participants = cursor.fetchall()

    cursor.close()
    conn.close()

    return render_template('event_details.html', event=event, participants=participants)

@app.route('/edit_participant/<event_name>/<int:participant_id>', methods=['GET', 'POST'])
def edit_participant(event_name, participant_id):
    # Edit participant details
    table_name = event_name.replace(" ", "_").lower()

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    if request.method == 'POST':
        participant_name = request.form['participant_name']
        participant_email = request.form['participant_email']
        participant_phone = request.form['participant_phone']

        cursor.execute(f"""
        UPDATE `{table_name}`
        SET participant_name = %s, participant_email = %s, participant_phone = %s
        WHERE id = %s
        """, (participant_name, participant_email, participant_phone, participant_id))
        conn.commit()
        cursor.close()
        conn.close()

        return redirect(url_for('view_event_participants', event_name=event_name))

    # Gets the prelfilled data in edit_participant.html
    cursor.execute(f'SELECT * FROM `{table_name}` WHERE id = %s', (participant_id,))
    participant = cursor.fetchone()

    cursor.close()
    conn.close()

    return render_template('edit_participant.html', event_name=event_name, participant=participant)
```

Source Code

```
@app.route('/delete_participant/<event_name>/<int:participant_id>', methods=['POST'])
def delete_participant(event_name, participant_id):
    # Delets participant from  event
    table_name = event_name.replace(" ", "_").lower()

    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute(f'DELETE FROM `{table_name}` WHERE id = %s', (participant_id,))
    conn.commit()

    cursor.close()
    conn.close()

    return redirect(url_for('view_event_participants', event_name=event_name))

if __name__ == '__main__':
    init_database()
    app.run(debug=True)
```

Functions
&
Working

1. get_db_connection():
    - **Purpose:** Establishes and returns a connection to the MySQL database using the provided configuration. This function is used throughout the app to interact with the database.

```
db_config = {
    'host': 'localhost',
    'user': 'root',
    'password': '', #my password
    'database': '' #databse name
}

def get_db_connection():
    return mysql.connector.connect(**db_config)
```

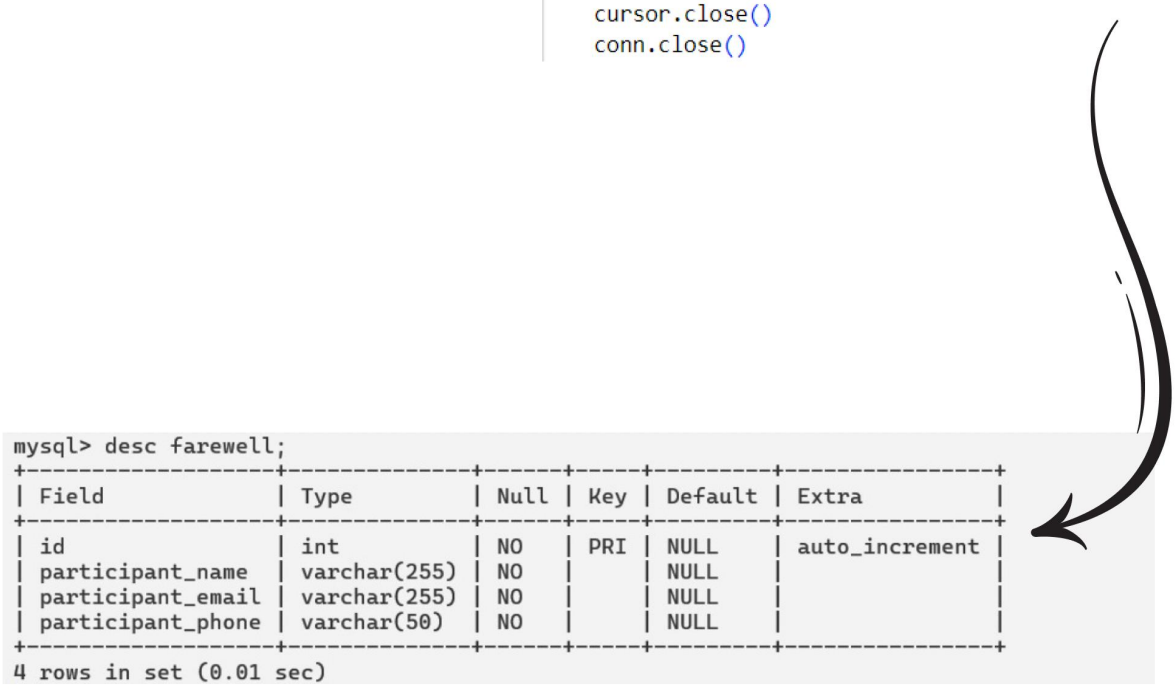
2. init_database():
   - **Purpose:** Initializes the database by creating the events table if it doesn't exist. This ensures that the database structure is ready to store event data when the application starts.

```python
def init_database():
    # Creating events table if it doesn't exist
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute('''
    CREATE TABLE IF NOT EXISTS events (
        id INT AUTO_INCREMENT PRIMARY KEY,
        event_name VARCHAR(255) NOT NULL UNIQUE,
        event_date DATE NOT NULL,
        event_location VARCHAR(255) NOT NULL
    )
    ''')
    conn.commit()
    cursor.close()
    conn.close()
```

```
mysql> desc events;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| id             | int          | NO   | PRI | NULL    | auto_increment |
| event_name     | varchar(255) | NO   | UNI | NULL    |                |
| event_date     | date         | NO   |     | NULL    |                |
| event_location | varchar(255) | NO   |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

3. create_event_table(event_name):
    - **Purpose:** Creates a dynamic table for each event where participant data will be stored. The table name is generated by converting the event name into a valid SQL table name (by replacing spaces with underscores). This allows each event to have a unique table for its participants.

```python
def create_event_table(event_name):
    # Creating table for participants of a specific event
    table_name = event_name.replace(" ", "_").lower()
    conn = get_db_connection()
    cursor = conn.cursor()
    cursor.execute(f'''
    CREATE TABLE IF NOT EXISTS `{table_name}` (
        id INT AUTO_INCREMENT PRIMARY KEY,
        participant_name VARCHAR(255) NOT NULL,
        participant_email VARCHAR(255) NOT NULL,
        participant_phone VARCHAR(50) NOT NULL
    )
    ''')
    conn.commit()
    cursor.close()
    conn.close()
```

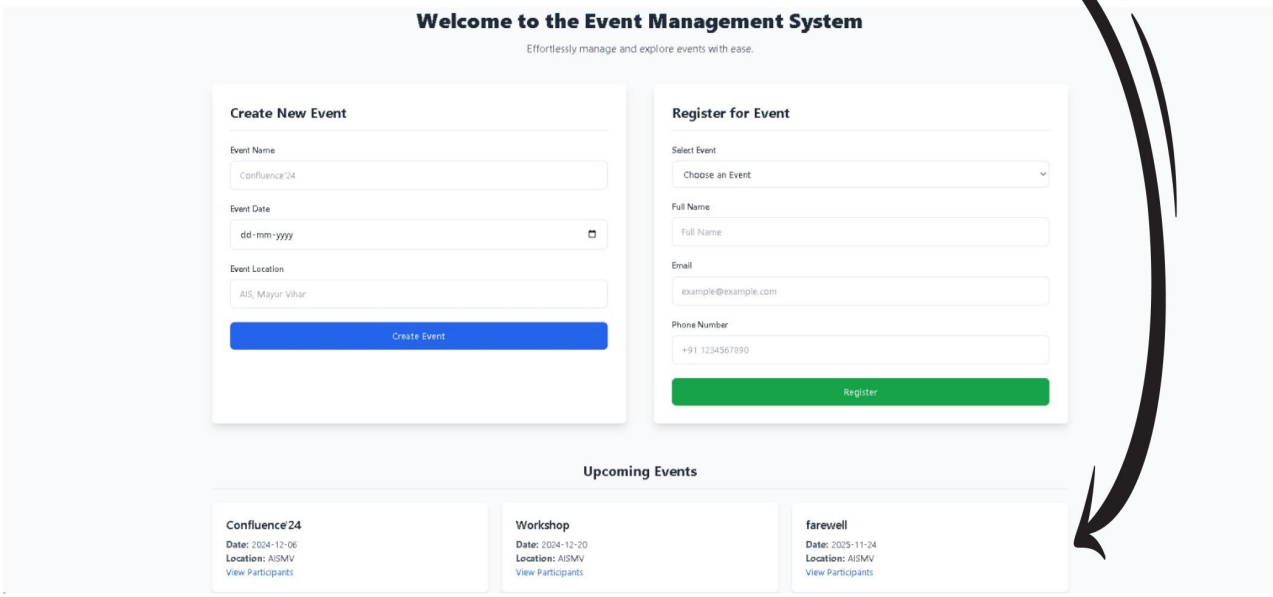


```
mysql> desc farewell;
+-------------------+--------------+------+-----+---------+----------------+
| Field             | Type         | Null | Key | Default | Extra          |
+-------------------+--------------+------+-----+---------+----------------+
| id                | int          | NO   | PRI | NULL    | auto_increment |
| participant_name  | varchar(255) | NO   |     | NULL    |                |
| participant_email | varchar(255) | NO   |     | NULL    |                |
| participant_phone | varchar(50)  | NO   |     | NULL    |                |
+-------------------+--------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
```

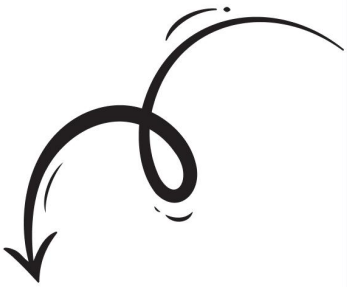
4. index():
    - **Purpose:** Displays all upcoming events stored in the events table. The function fetches events that have a date greater than or equal to today and renders them in the index.html template for the user to view.

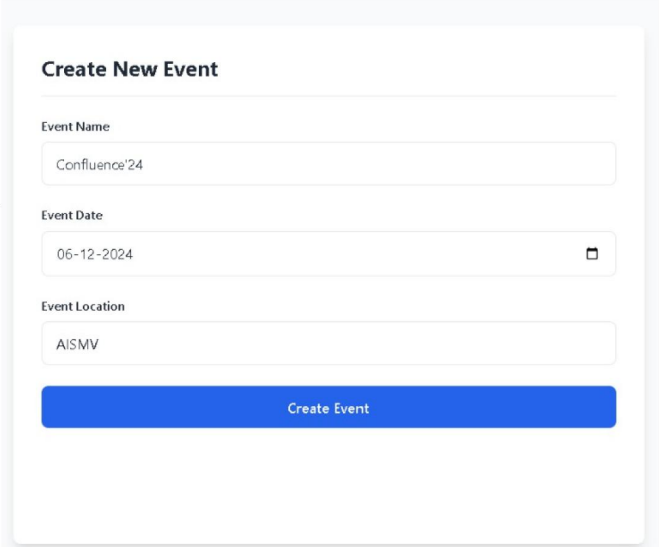
```python
@app.route('/')
def index():
    # Displays upcoming events
    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute('SELECT * FROM events WHERE event_date >= %s ORDER BY event_date ASC', (date.today(),))
    events = cursor.fetchall()
    cursor.close()
    conn.close()
    return render_template('index.html', events=events)
```

```
mysql> select * from events;
+----+---------------+------------+----------------+
| id | event_name    | event_date | event_location |
+----+---------------+------------+----------------+
|  1 | Confluence'24 | 2024-12-06 | AISMV          |
|  4 | Workshop      | 2024-12-20 | AISMV          |
|  5 | farewell      | 2025-11-24 | AISMV          |
+----+---------------+------------+----------------+
3 rows in set (0.00 sec)
```



5. create():
    - **Purpose:** Handles the form submission for creating a new event. It collects the event name, date, and location from the form, inserts the data into the events table, and creates a corresponding table for the event's participants using create_event_table().

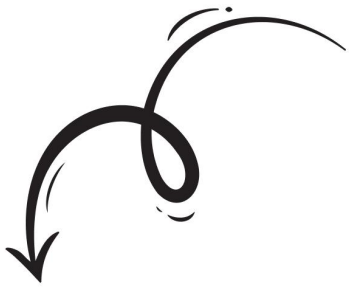


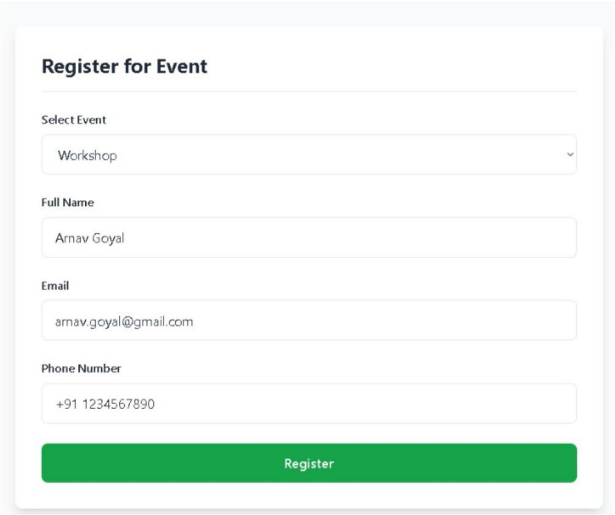



```python
@app.route('/create', methods=['POST'])
def create():
    # Creatinga new event
    event_name = request.form['event_name']
    event_date = request.form['event_date']
    event_location = request.form['event_location']

    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute('''
    INSERT INTO events (event_name, event_date, event_location)
    VALUES (%s, %s, %s)
    ''', (event_name, event_date, event_location))
    conn.commit()
    create_event_table(event_name)

    cursor.close()
    conn.close()

    return redirect('/')
```

```
mysql> select * from events;
+----+---------------+------------+----------------+
| id | event_name    | event_date | event_location |
+----+---------------+------------+----------------+
|  1 | Confluence'24 | 2024-12-06 | AISMV          |
|  4 | Workshop      | 2024-12-20 | AISMV          |
|  5 | farewell      | 2025-11-24 | AISMV          |
+----+---------------+------------+----------------+
3 rows in set (0.00 sec)
```

6. register_participant():
    - **Purpose:** Allows users to register as participants for an event. It takes participant details (name, email, phone) and inserts them into the appropriate event's table (which is dynamically created based on the event name). It then redirects the user back to the event's page.





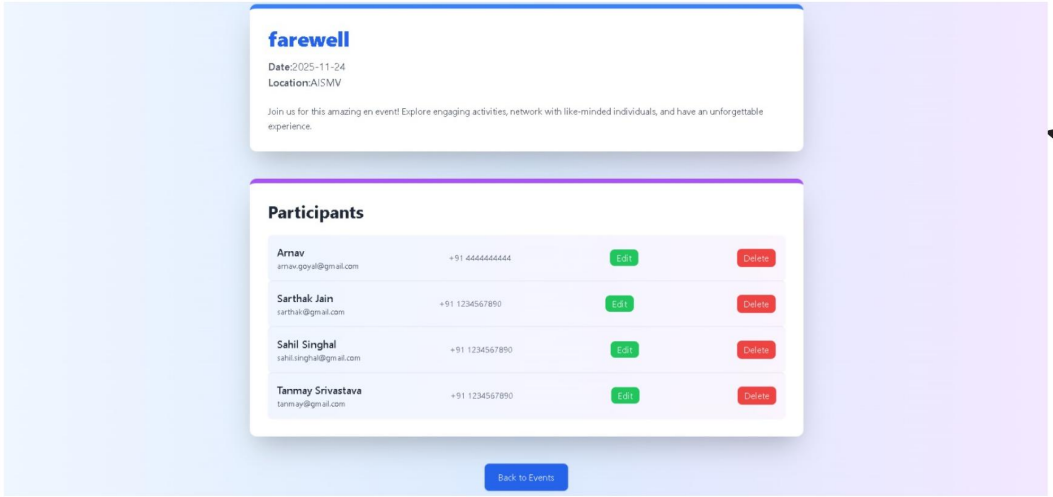

```python
@app.route('/register', methods=['POST'])
def register_participant():
    # Retrieve event_name and other form data
    event_name = request.form.get('event_name')
    participant_name = request.form['participant_name']
    participant_email = request.form['participant_email']
    participant_phone = request.form['participant_phone']

    table_name = event_name.replace(" ", "_").lower()

    conn = get_db_connection()
    cursor = conn.cursor()

    try:
        cursor.execute(f'''
        INSERT INTO `{table_name}` (participant_name, participant_email, participant_phone)
        VALUES (%s, %s, %s)
        ''', (participant_name, participant_email, participant_phone))
        conn.commit()
    except mysql.connector.Error as e:
        return f"Database error: {e}", 500
    finally:
        cursor.close()
        conn.close()

    return redirect('/')
```

```
mysql> select * from farewell;
+----+------------------+-------------------------+-------------------+
| id | participant_name | participant_email       | participant_phone |
+----+------------------+-------------------------+-------------------+
|  1 | Arnav            | arnav.goyal@gmail.com   | +91 4444444444    |
|  2 | Sarthak Jain     | sarthak@gmail.com       | +91 1234567890    |
|  3 | Sahil Singhal    | sahil.singhal@gmail.com | +91 1234567890    |
|  4 | Tanmay Srivastava | tanmay@gmail.com       | +91 1234567890    |
+----+------------------+-------------------------+-------------------+
4 rows in set (0.00 sec)
```

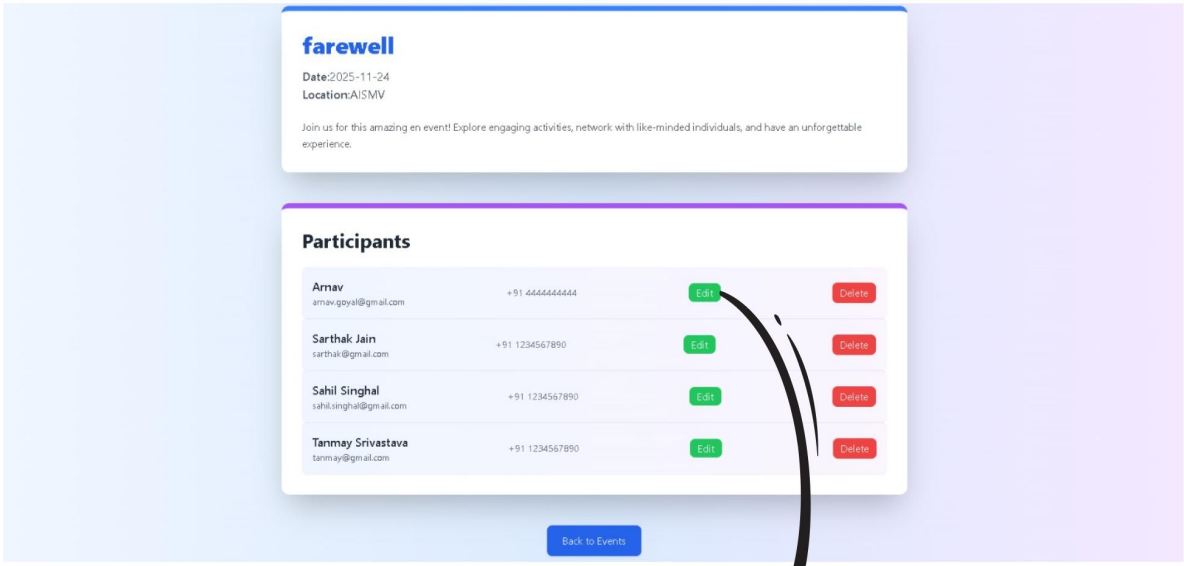
7. view_event_participants(event_name):
    - **Purpose:** Displays the details of a specific event and its participants. It fetches the event's information from the events table and the list of participants from the dynamically created table for that event, rendering the data in the event_details.html template


```python
@app.route('/event/<event_name>')
def view_event_participants(event_name):
    # View participants of a specific event
    table_name = event_name.replace(" ", "_").lower()

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    cursor.execute('SELECT * FROM events WHERE event_name = %s', (event_name,))
    event = cursor.fetchone()
    cursor.execute(f'SELECT * FROM `{table_name}`')
    participants = cursor.fetchall()

    cursor.close()
    conn.close()

    return render_template('event_details.html', event=event, participants=participants)
```

```
mysql> select * from farewell;
+----+-------------------+-------------------------+-------------------+
| id | participant_name  | participant_email       | participant_phone |
+----+-------------------+-------------------------+-------------------+
|  1 | Arnav             | arnav.goyal@gmail.com   | +91 4444444444    |
|  2 | Sarthak Jain      | sarthak@gmail.com       | +91 1234567890    |
|  3 | Sahil Singhal     | sahil.singhal@gmail.com | +91 1234567890    |
|  4 | Tanmay Srivastava | tanmay@gmail.com        | +91 1234567890    |
+----+-------------------+-------------------------+-------------------+
4 rows in set (0.00 sec)
```



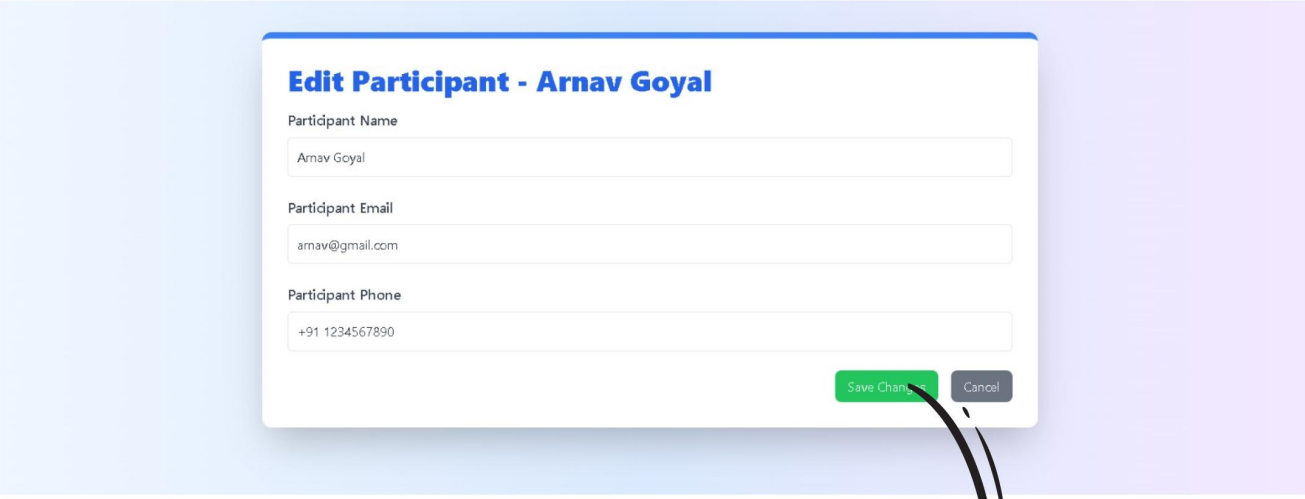
8. edit_participant(event_name, participant_id):
    - **Purpose:** Handles the editing of a participant's details. When the form is submitted with updated participant information, the function updates the participant's record in the database. If accessed via a GET request, it pre-fills the participant's existing details in the form.




```python
@app.route('/edit_participant/<event_name>/<int:participant_id>', methods=['GET', 'POST'])
def edit_participant(event_name, participant_id):
    # Edit participant details
    table_name = event_name.replace(" ", "_").lower()

    conn = get_db_connection()
    cursor = conn.cursor(dictionary=True)

    if request.method == 'POST':
        participant_name = request.form['participant_name']
        participant_email = request.form['participant_email']
        participant_phone = request.form['participant_phone']

        cursor.execute(f'''
        UPDATE `{table_name}`
        SET participant_name = %s, participant_email = %s, participant_phone = %s
        WHERE id = %s
        ''', (participant_name, participant_email, participant_phone, participant_id))
        conn.commit()
        cursor.close()
        conn.close()

        return redirect(url_for('view_event_participants', event_name=event_name))

    # Gets the prelfilled data in edit_participant.html
    cursor.execute(f'SELECT * FROM `{table_name}` WHERE id = %s', (participant_id,))
    participant = cursor.fetchone()

    cursor.close()
    conn.close()

    return render_template('edit_participant.html', event_name=event_name, participant=participant)
```

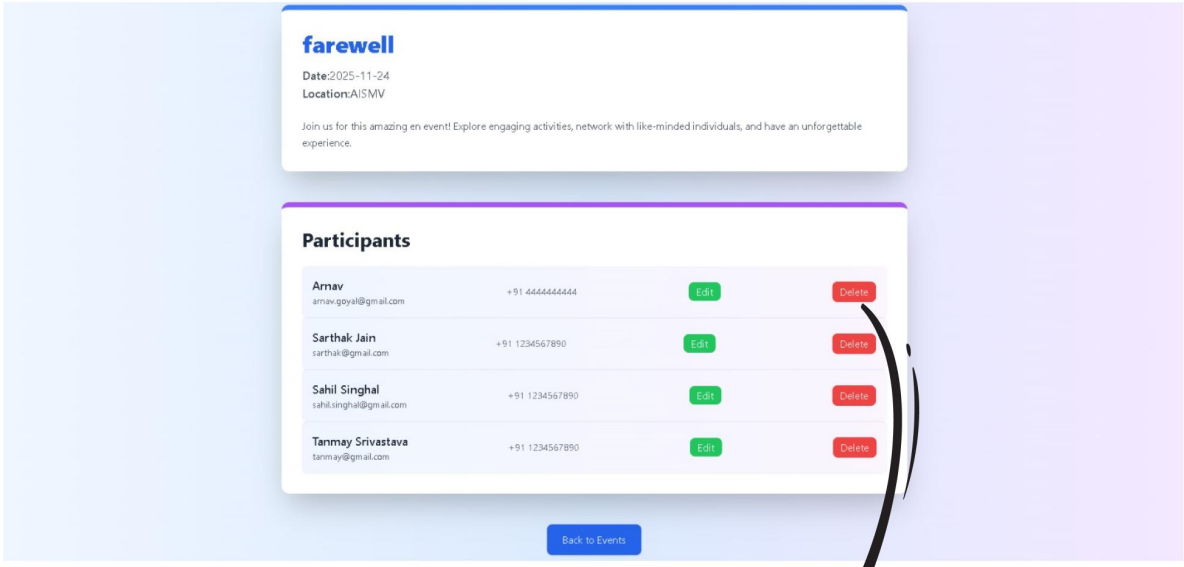
```
mysql> select * from farewell;
+----+-------------------+-------------------------+-------------------+
| id | participant_name  | participant_email       | participant_phone |
+----+-------------------+-------------------------+-------------------+
|  1 | Arnav             | arnav.goyal@gmail.com   | +91 4444444444    |
|  2 | Sarthak Jain      | sarthak@gmail.com       | +91 1234567890    |
|  3 | Sahil Singhal     | sahil.singhal@gmail.com | +91 1234567890    |
|  4 | Tanmay Srivastava | tanmay@gmail.com        | +91 1234567890    |
+----+-------------------+-------------------------+-------------------+
4 rows in set (0.00 sec)
```

next page




```
mysql> select * from farewell;
+----+-------------------+-------------------------+-------------------+
| id | participant_name  | participant_email       | participant_phone |
+----+-------------------+-------------------------+-------------------+
|  1 | Arnav Goyal       | arnav@gmail.com         | +91 1234567890    |
|  2 | Sarthak Jain      | sarthak@gmail.com       | +91 1234567890    |
|  3 | Sahil Singhal     | sahil.singhal@gmail.com | +91 1234567890    |
|  4 | Tanmay Srivastava | tanmay@gmail.com        | +91 1234567890    |
+----+-------------------+-------------------------+-------------------+
4 rows in set (0.00 sec)
```

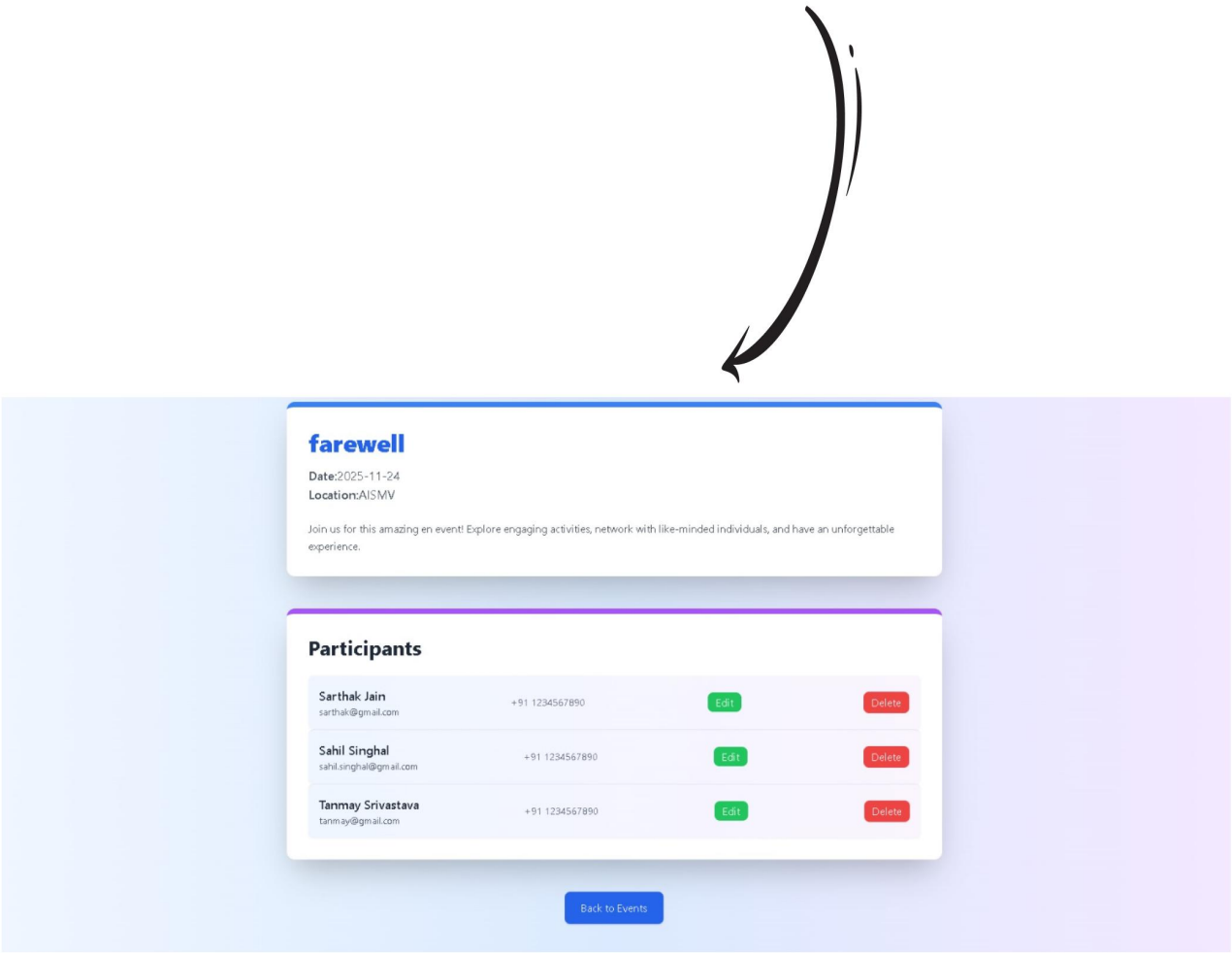
9. delete_participant(event_name, participant_id):
    - **Purpose: Deletes** a participant from a specific event. This function removes the participant's record from the corresponding event's table using their participant ID and then redirects back to the event's participant list.




```python
@app.route('/delete_participant/<event_name>/<int:participant_id>', methods=['POST'])
def delete_participant(event_name, participant_id):
    # Delets participant from  event
    table_name = event_name.replace(" ", "_").lower()

    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute(f'DELETE FROM `{table_name}` WHERE id = %s', (participant_id,))
    conn.commit()

    cursor.close()
    conn.close()

    return redirect(url_for('view_event_participants', event_name=event_name))
```

next page

```
mysql> select * from farewell;
+----+------------------+-------------------------+------------------+
| id | participant_name | participant_email       | participant_phone |
+----+------------------+-------------------------+------------------+
|  2 | Sarthak Jain     | sarthak@gmail.com       | +91 1234567890   |
|  3 | Sahil Singhal    | sahil.singhal@gmail.com | +91 1234567890   |
|  4 | Tanmay Srivastava | tanmay@gmail.com       | +91 1234567890   |
+----+------------------+-------------------------+------------------+
3 rows in set (0.00 sec)
```



Future Expansion

The Web-Based Event Management System can be enhanced with several advanced features to make it more robust and user-friendly. Integrating a user authentication system with role-based access control would ensure secure and personalized access for organizers and participants. The system could also include email and SMS notifications to confirm registrations and provide event updates. Adding payment integration would allow participants to pay for registrations seamlessly, while a dynamic dashboard for organizers could provide valuable insights into event statistics and participant data. Furthermore, features like event categorization, mobile-friendly design, and feedback options would significantly improve the overall user experience.

Key Expansion Ideas:

1. User Authentication: Secure login for admins and participants with role-based permissions.
2. Notifications: Email and SMS alerts for registrations and event reminders.
3. Payment Integration: Seamless payment options and financial record management.

Bibliography

1. Youtube.com
2. flask.palletsprojects.com
3. tailwindcss.com
4. w3schools.com
5. Github.com
6. Stackoverflow.com
7. Geeksforgeeks.com