I - Introduction

Managing school clubs and student organizations can quickly become complicated, especially when it involves handling members, events, attendance, and finances all at once. Many clubs still rely on manual spreadsheets or scattered tools, which makes coordination difficult and prone to errors. To solve this issue, **OrganiByG7** was developed as a centralized, modern, and efficient management system designed specifically for clubs, associations, and small organizations.
This project aims to simplify daily operations, improve data accuracy, and help administrators and club presidents focus more on activities rather than administrative tasks.

II - Project Objective

The primary objective of OrganiByG7 is to offer a unified digital solution that simplifies the daily management of school clubs and organizations. More specifically, the project aims to:

• **Centralize administrative tasks** by gathering member management, events, attendance, and financial records in one platform.
• **Improve efficiency** by reducing manual work and automating repetitive tasks such as listing members, registering events, and updating attendance.
• **Enhance transparency** by maintaining accurate and accessible records of participation, payments, and club activities.
• **Support better organization workflows** through clear data structures that help administrators monitor operations in real time.
• **Provide a user-friendly system** that makes club administration easier for staff and creates a more organized experience for students.

III - Project Description

OrganiByG7 is a web-based management system that allows clubs and organizations to manage all their structure in one place. Built using **Flask (Python)** for the backend, **MySQL** for data management, and **HTML/CSS/JavaScript** for the frontend, the platform provides a user-friendly dashboard with essential features such as:

1- Member Management

Add, update, delete, and display members easily using a structured database. All member information is stored in the `members` table and displayed dynamically through templates.

2- Event Management

Create and manage events, including details like title, description, date, and price. Events are saved in the `events` table and sorted by date for faster access.

3- Attendance Tracking

A complete attendance system linking members and events using a many-to-many relationship. Attendance records are stored and updated in the `attendance` table.

4- Payments & Financial Tracking

Keep track of payments made by members for events. The system stores payment amounts, methods, and links them to the corresponding events and members.

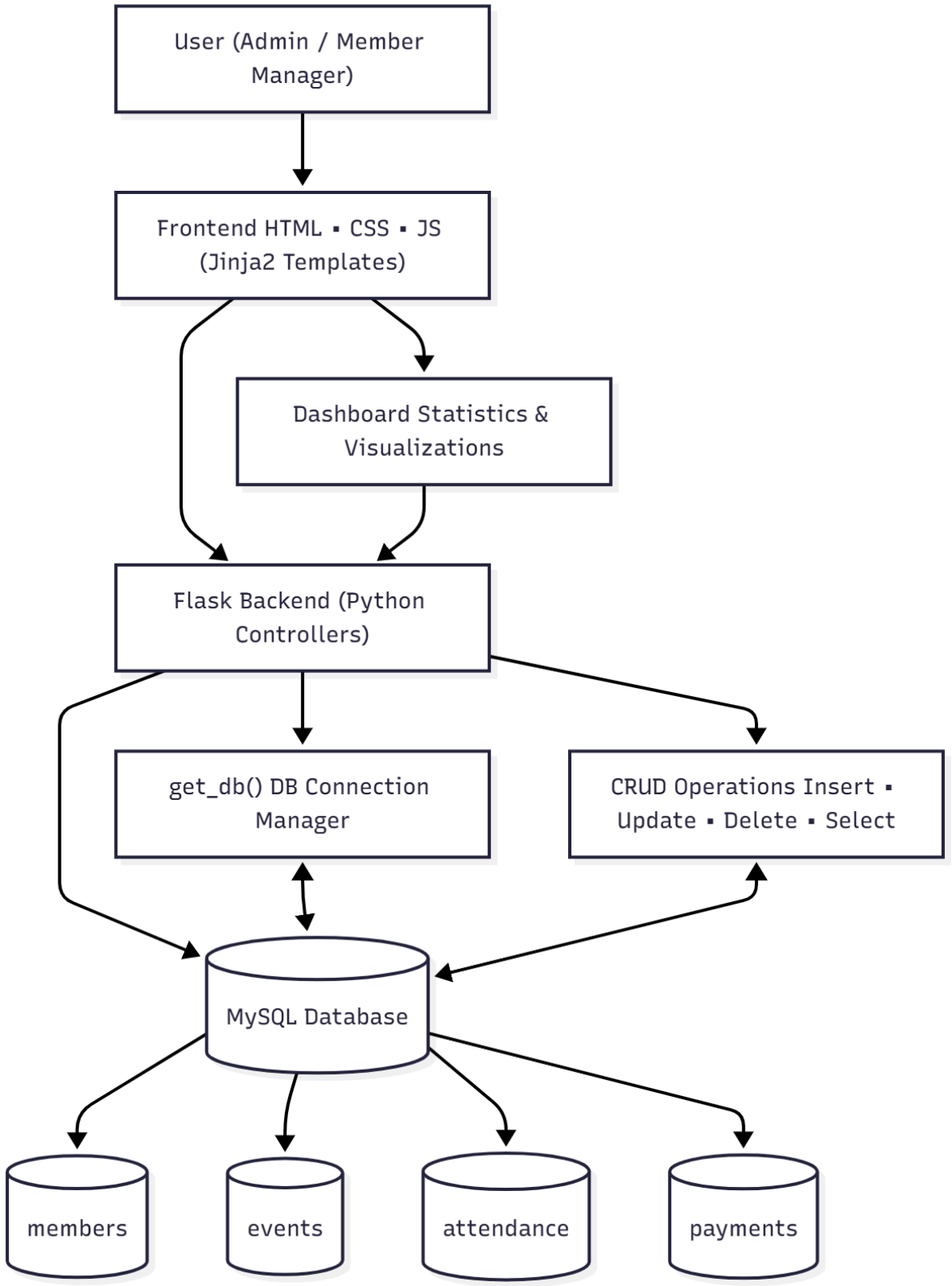
5- Dashboard & Statistics

An interactive dashboard provides real-time insights such as:

- Total number of members
- Upcoming events
- Revenues collected
- Attendance rates

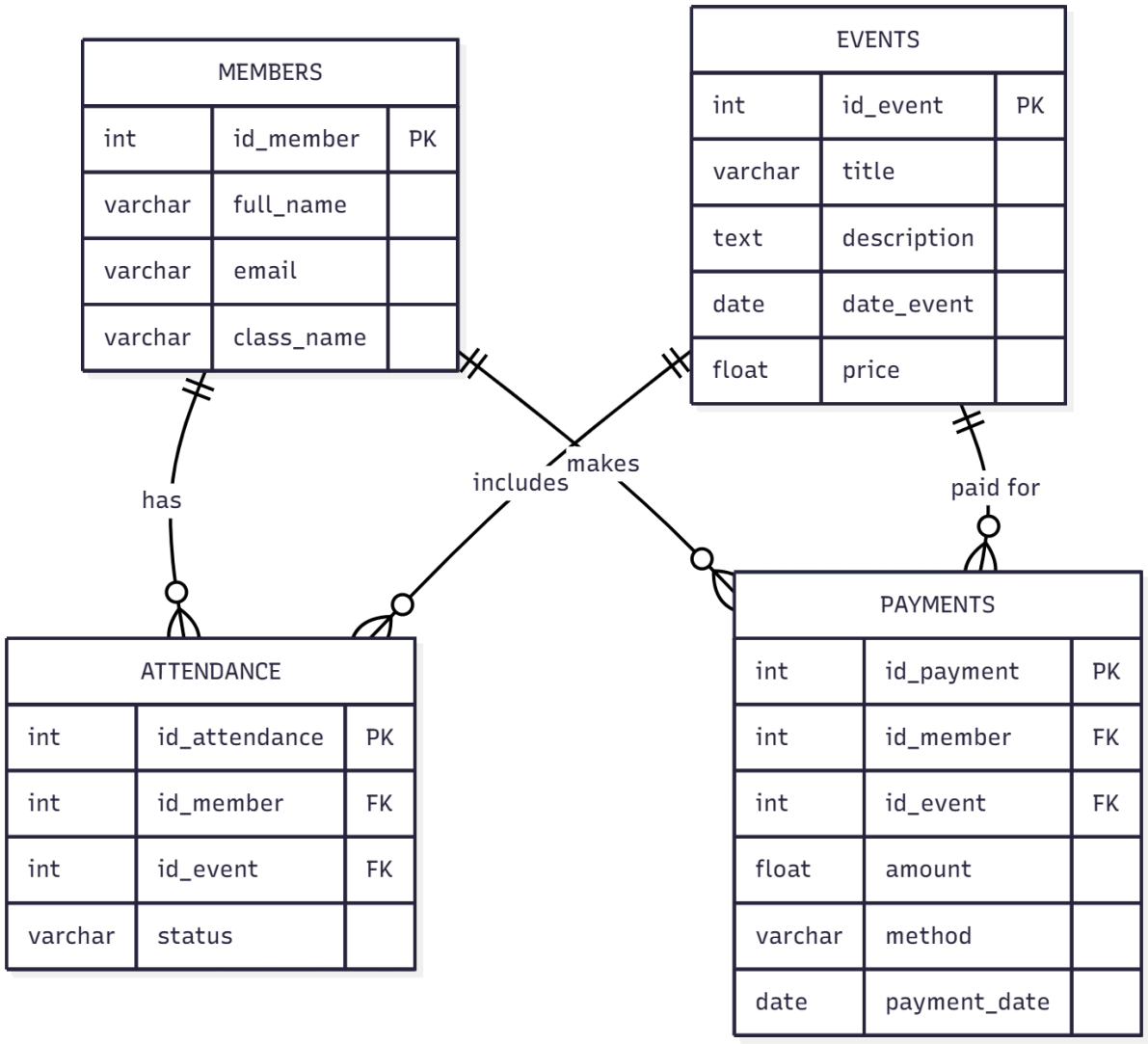
These statistics are generated via SQL queries and rendered to the frontend to maintain up-to-date management information.

IV - System Architecture

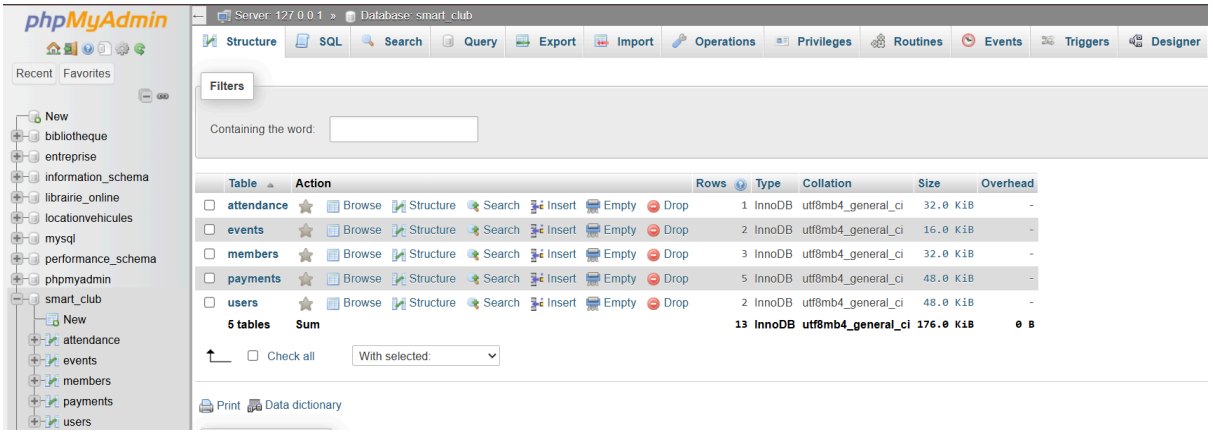


V - DataBase Design

1- ER Diagram

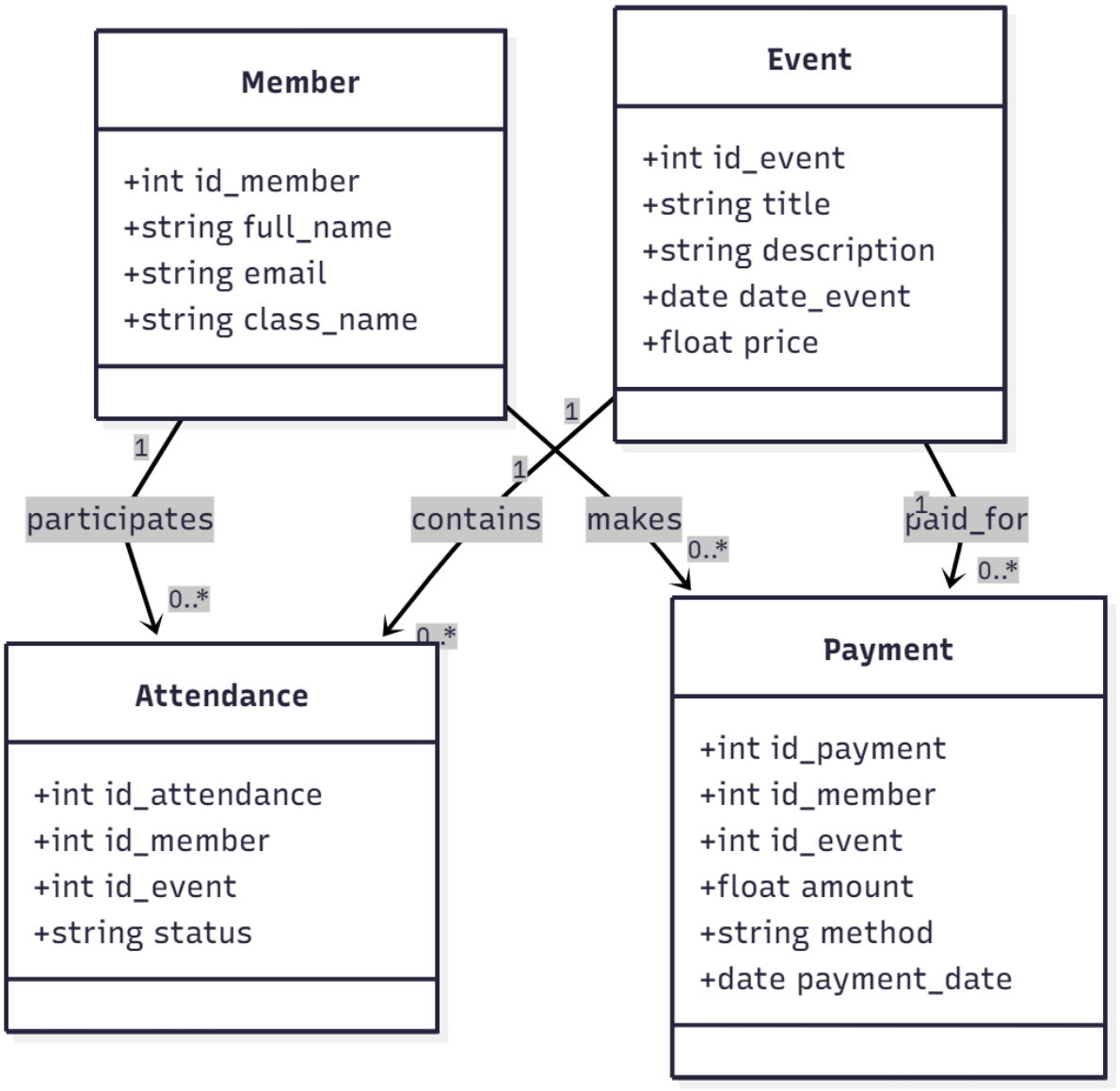


2- MySQL Tables:

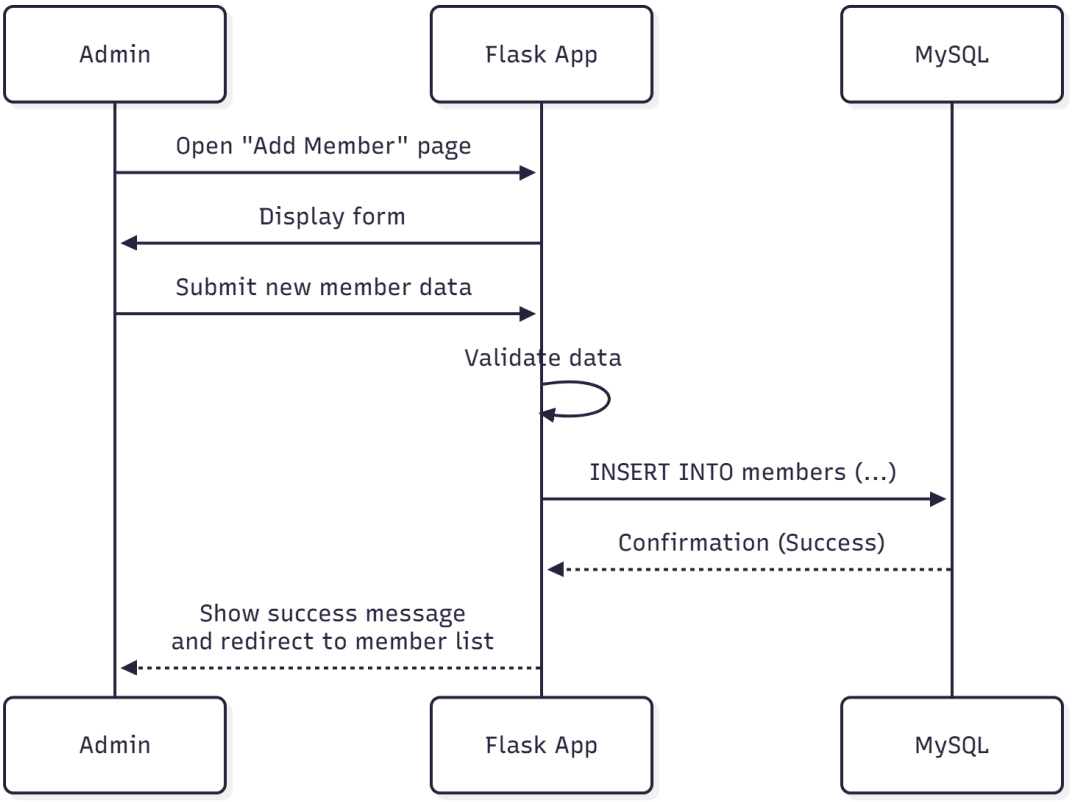


VI - Diagrams:

1- Class Diagram



2- Sequence Diagram



VI - Implémentation

The implementation of **OrganiByG7** relies heavily on MySQL for data storage and on Flask for handling requests, rendering templates, and executing SQL queries. The system uses structured CRUD operations to manage records across the four main tables: `members`, `events`, `attendance`, and `payments`.

1. SQL Connection Layer

The project uses the `mysql-connector-python` library to connect Flask to the MySQL server.
All database logic is centralized in **`db.py`**, which contains:

- A configuration object specifying host, user, password, and database name.
- A `get_db()` function that:
    - Creates a new MySQL connection if one does not already exist for the current request.

    - Stores that connection in Flask's `g` context.

- A `teardown_appcontext` function that automatically closes the connection after the request finishes.

This ensures that each HTTP request gets a clean, independent SQL session.

2. Data Flow: MySQL → Flask → HTML Pages

The data lifecycle works like this:

1. Browser sends a request (GET or POST).
2. Flask route receives it and calls `get_db()` to access the MySQL connection.
3. A cursor (`db.cursor(dictionary=True)`) executes SQL queries:
    - `SELECT` for reading
    - `INSERT` for adding
    - `UPDATE` for modifying
    - `DELETE` for removing
4. Query results are stored in Python dictionaries.
5. Flask passes the results to an HTML template through `render_template()`.
6. Jinja2 loops (`{% for x in data %}`) display the records inside tables or forms.

This mechanism makes the backend a bridge between SQL storage and the user interface.

3. Member Management (CRUD Using SQL)

A. Creating a Member

When the user submits the "Add Member" form:

Flask captures the POST data:

```
name = request.form['full_name']
```

- SQL query executed:

```
INSERT INTO members (full_name, email, class_name)
VALUES (%s, %s, %s);
```

- After committing the transaction, Flask reloads the member list.

B. Reading Members

To show the table of members, Flask runs:

```
SELECT * FROM members ORDER BY id DESC;
```

The results are returned as a dictionary list and displayed in Jinja2.

C. Updating a Member

For editable fields, Flask executes:

```
UPDATE members
SET full_name = %s, email = %s, class_name = %s
WHERE id = %s;
```

D. Deleting a Member

Using a route like `/delete-member/<id>`:

```
DELETE FROM members WHERE id = %s;
```

4. Event Management

Creating an Event

```
INSERT INTO events (title, description, date_event, price)
VALUES (%s, %s, %s, %s);
```

Listing Events

```
SELECT * FROM events ORDER BY date_event DESC;
```

This allows the interface to display upcoming events first.

Updating or Deleting Events

Standard SQL `UPDATE` and `DELETE` queries are used when administrators modify event details.

5. Attendance Tracking (Many-to-Many SQL Logic)

The `attendance` table links members to events.

Marking Attendance

The system uses a smart SQL technique:

```
INSERT INTO attendance (member_id, event_id, status)

VALUES (%s, %s, %s)

ON DUPLICATE KEY UPDATE status = VALUES(status);
```

This means:

- If the attendance record does not exist → create it.
- If it already exists → update the status instead.

This solves the problem of duplicates and keeps attendance consistent.

Displaying Attendance

```
SELECT a.*, m.full_name, e.title

FROM attendance a

JOIN members m ON a.member_id = m.id

JOIN events e ON a.event_id = e.id;
```

This provides meaningful information to the UI.

6. Payment Management (Data Linked to Members and Events)

Adding a Payment

```
INSERT INTO payments (member_id, event_id, amount, method,
payment_date)

VALUES (%s, %s, %s, %s, %s);
```

Showing Payments

```
SELECT p.*, m.full_name, e.title

FROM payments p

JOIN members m ON p.member_id = m.id

JOIN events e ON p.event_id = e.id;
```

Dashboard Revenue Calculation

```
SELECT SUM(amount) AS total_revenue FROM payments;
```

Flask uses this value to display financial statistics on the dashboard.

7. Rendering Data in Templates

Once the SQL queries return their results:

```
return render_template("members.html", members=data)
```

Inside HTML (Jinja2):

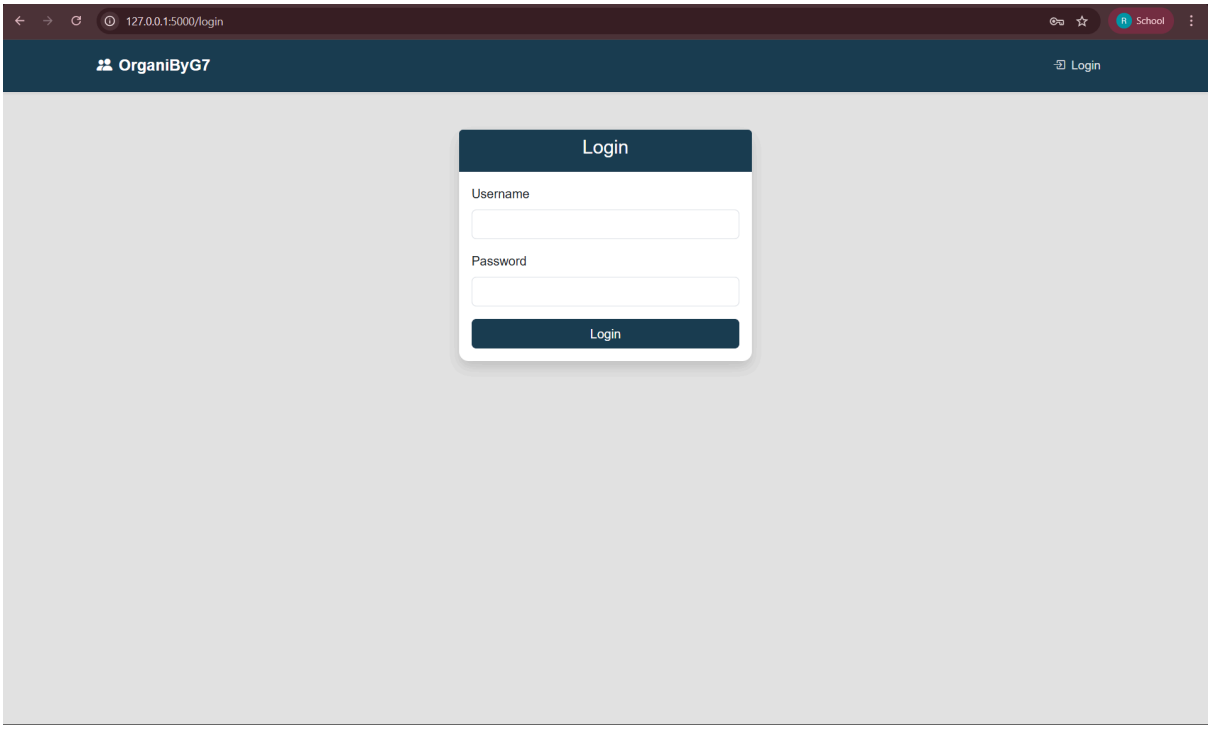
```
{% for member in members %}

<tr>

  <td>{{ member.full_name }}</td>

  <td>{{ member.email }}</td>

  <td>{{ member.class_name }}</td>

</tr>

{% endfor %}
```

This connects SQL data directly to the final visual interface.

VII - Interface

1- Dashboard Login:

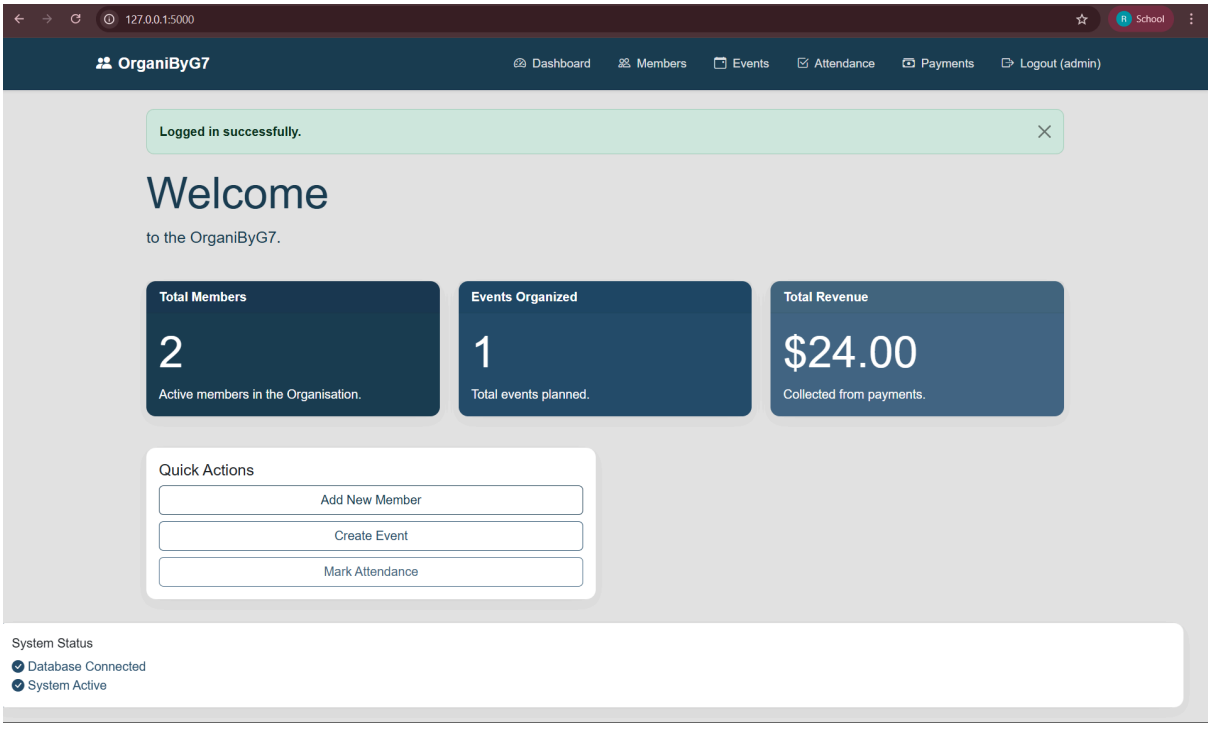
The Login page allows administrators to securely access the system. It verifies user credentials, prevents unauthorized access, and redirects the user to the Admin Dashboard once authenticated.



2- Dashboard Admin

The Admin Dashboard provides a quick overview of the club’s activity.
It displays real-time statistics such as:

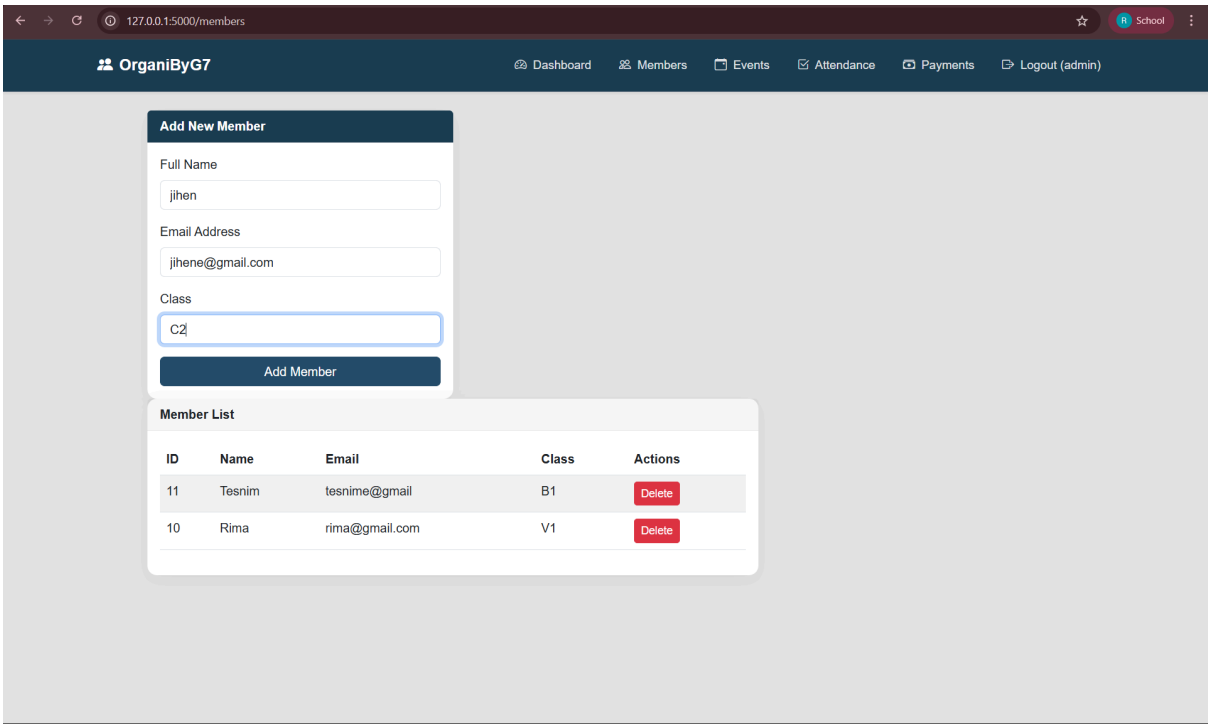
- Total members
- Number of events
- Upcoming events
- Payments collected



3- Members Manager

This module handles all operations related to members.
Features include:

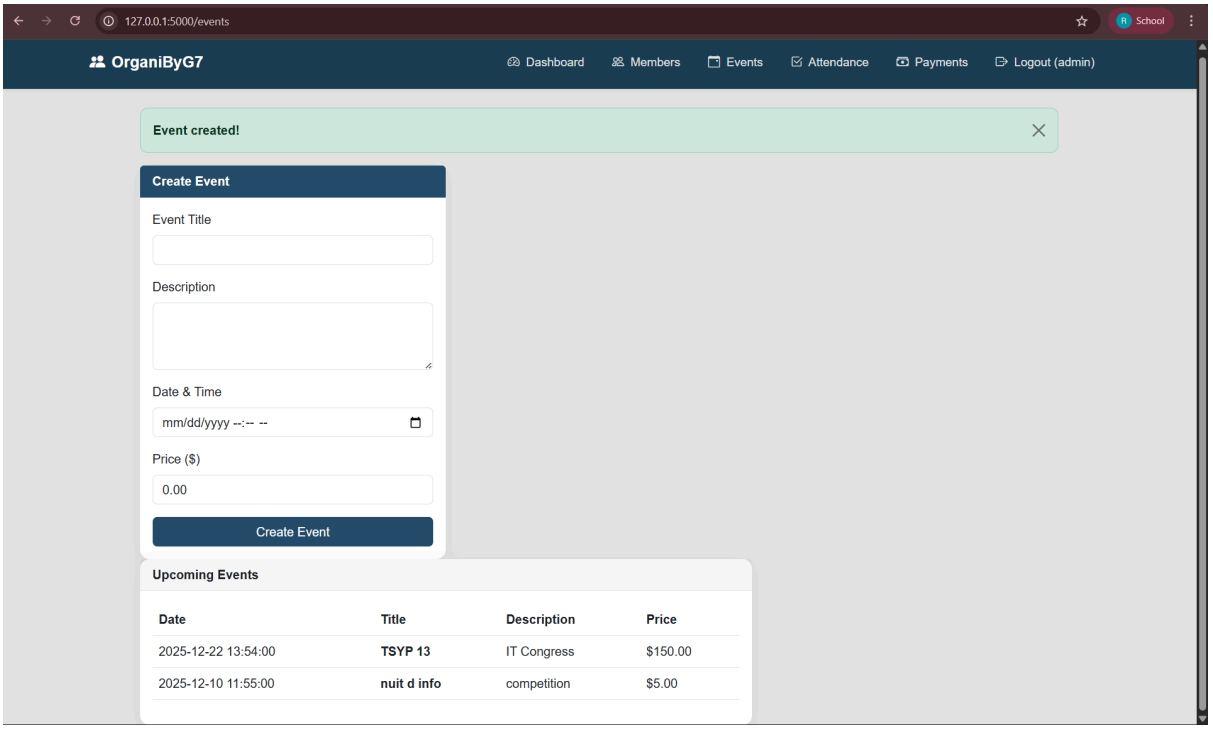
- Adding new members
- Updating member information
- Deleting members
- Viewing all members in a searchable table



4- Event Manager

The Event Manager allows the admin to create and manage events.
It includes:

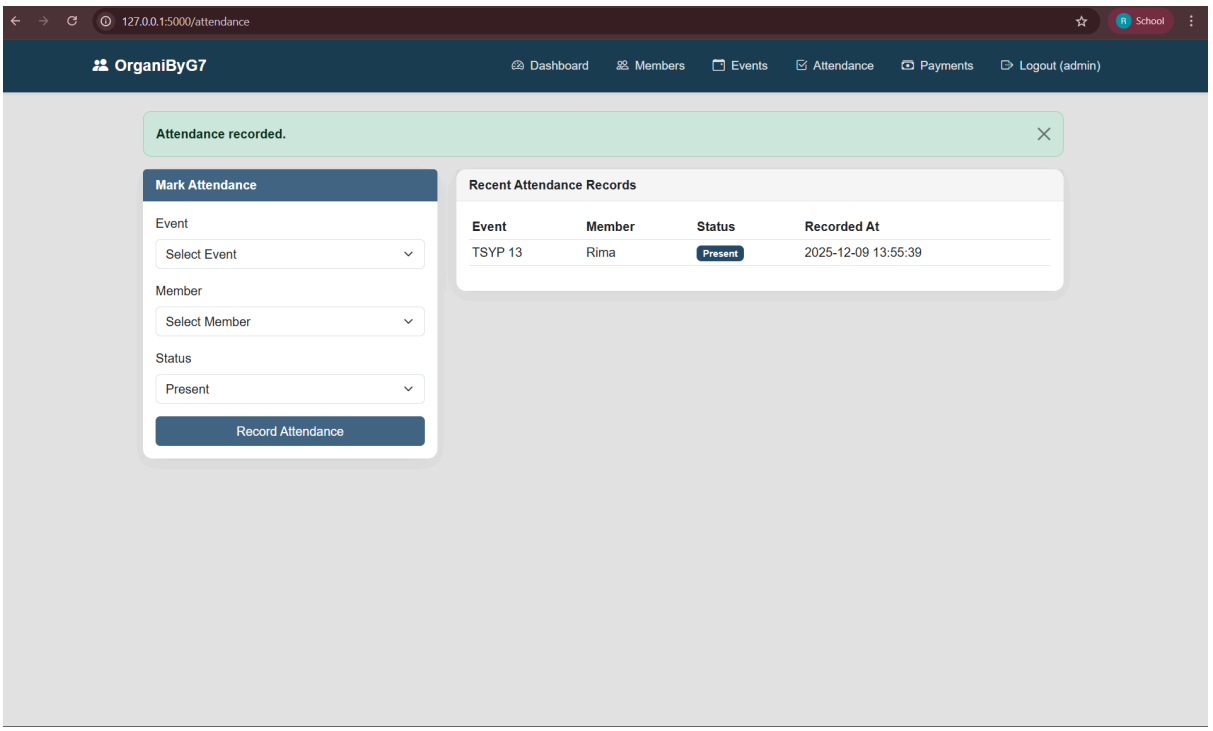
- Event title, description, date, and price
- Editing or deleting events
- Listing all events by date
  This module makes event planning simple and centralized.



5- Attendees Manager

The Attendees Manager links members to events.
It allows the admin to:

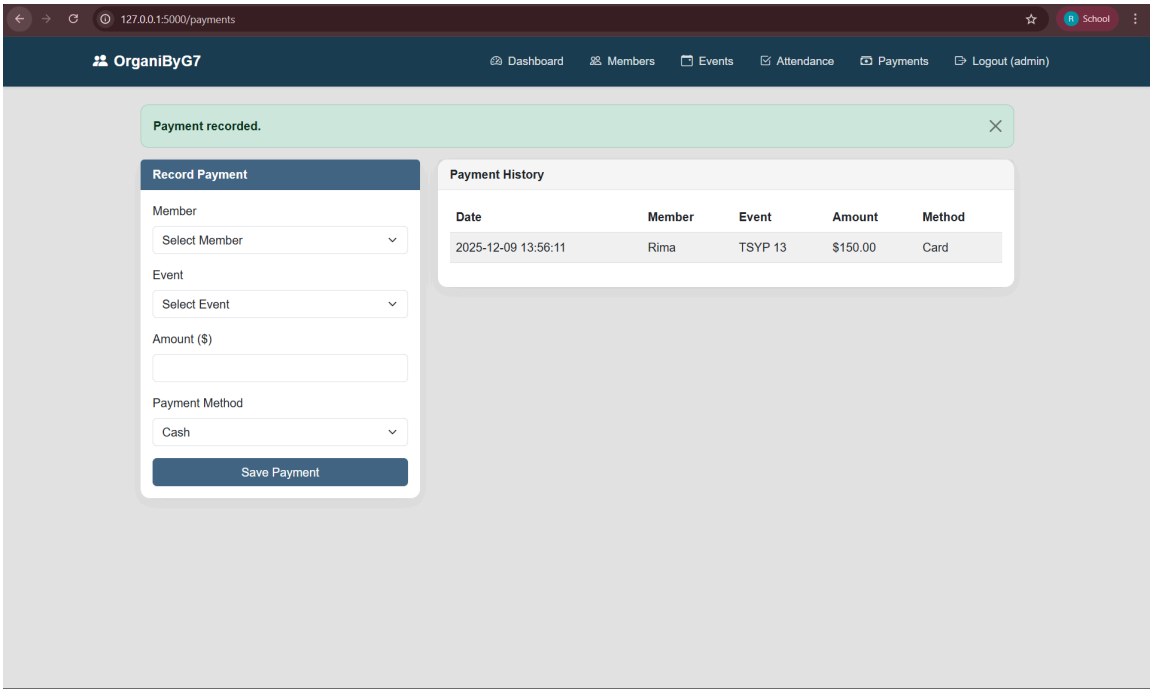
- Mark a member as *Present* or *Absent*
- View attendance lists for each event
- Update attendance without duplicates



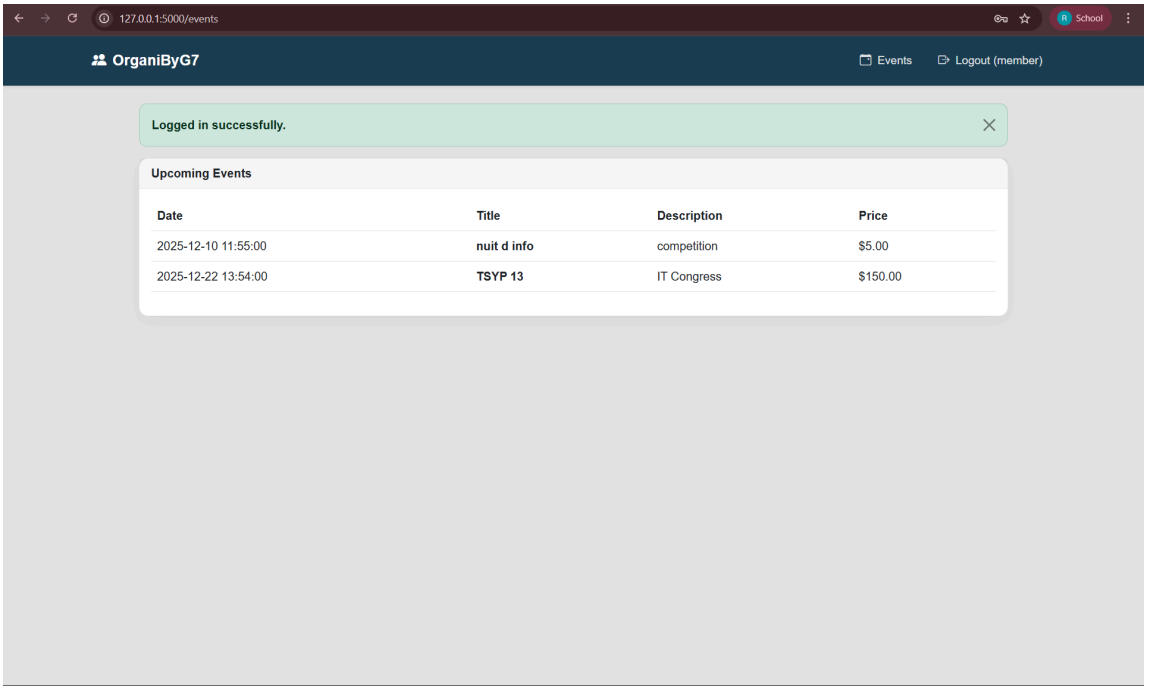
6- Payment Manager

This module manages all financial transactions related to events.
It allows:

- Recording payments from members
- Selecting payment method (Cash, Card, Transfer)
- Calculating total revenue
- Viewing payment history
  It ensures transparency and proper financial tracking.



7- Dashboard Member



VIII - Tool & Technology:

- **Python (Flask):**
  Used to build the backend of the website.
  It handles the logic, routes, forms, and communication with the database.
- **MySQL:**
  Used to store all the data (members, events, attendance, payments).
  It is reliable, organized, and perfect for structured information.
- **HTML / CSS / JS**
  Used to create the website pages:
  - HTML → page structure
  - CSS → design and colors
  - JavaScript → interactivity
- **Bootstrap**
  Used to make the design nicer and responsive.
  It gives ready-made styles for forms, buttons, and tables.
- **MVC Structure**
  Used to keep the project organized:
  - **Model:** database
  - **View:** HTML pages
  - **Controller:** Flask routes

- **MySQL Connector**
  Used by Python to connect to the MySQL database.
  It allows Flask to send queries and get results.

IX - Conclusion

This project allowed the development of a complete and functional management system for school clubs and small organizations. By combining Flask, MySQL, and a clean web interface, the application successfully centralizes essential tasks such as managing members, events, attendance, and payments.

Throughout the project, I gained practical experience in building a full-stack application, structuring a database, creating CRUD operations, and organizing the project using an MVC-inspired approach. The result is a system that is easy to use, reliable, and scalable for future improvements.

In the future, additional features such as user roles, email notifications, mobile responsiveness, and advanced analytics can be added to enhance the platform. Overall, this project strengthened my technical skills and provided a solid understanding of real-world application development.