Campus Event Management System – Design Document

1. Introduction

Done by – Mohit R from Reva University
This project was implemented by me with the help of AI-assisted coding (vibe coding approach using tools like ChatGPT, Cursor, and other AI Tools). The goal was to build a **Campus Event Management Platform** that can record and report essential event-related activities.

This design specifically focuses on the **reporting system**, enabling meaningful insights into **event participation, attendance trends, and feedback analysis**. The solution was built to be **simple, minimal, and functional**, keeping in mind the requirements of the assignment.

2. Data to Track

For the reporting system to work, the following key data points were tracked:

- **Event Creation**: Event details like name, type, date, college, description, and capacity.
- **Student Registration**: Students registering for events, their status, and timestamps.
- **Attendance**: Whether a registered student attended the event.
- **Feedback**: Ratings (1–5) and optional comments from students who attended.

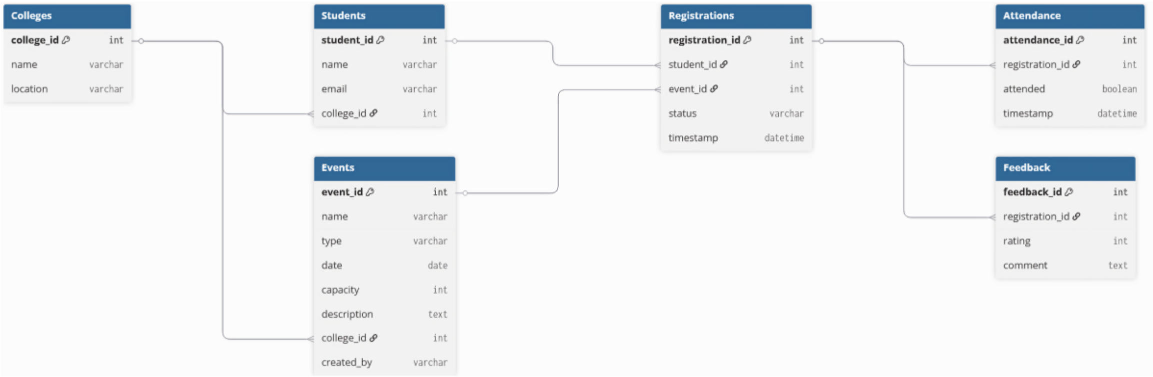
This data serves as the backbone for generating reports such as **event popularity**, **student participation**, and **top active students**.

3. Database Schema

The system is built on SQLite with the following main tables:

- Colleges – College_id, name, location.
- Students – Student_id, name, email, college_id.
- Events – Event_id, name, type, date, capacity, description, college_id, created_by.
- Registrations – Registration_id, student_id, event_id, status, timestamp.
- Attendance – Registration_id, attended (Boolean), timestamp.
- Feedback – Registration_id, rating (1–5), comment.

For this Prototype, I Am not including a separate Admins table. Event creation is tracked using a simple created by field in the events table. In a full-scale system, a dedicated Admins table can be added in the future to manage staff accounts, permissions and track which admin created which events



4. API Design

The APIs expose core functionalities to interact with the system.

- Colleges – List all colleges, get details.
  → /colleges, /colleges/{id}

- Students – Register student, fetch student details.
  → /students, /students/{id}, /students (POST)

- Events – Create events, fetch events, filter by college.
  → /events, /events/{id}, /events (POST)

- Registrations – Register students to events, list by student/event.
  → /registrations (POST), /registrations/student/{id}, /registrations/event/{id}

- Attendance – Mark attendance, fetch attendance by event.
  → /attendance (POST), /attendance/event/{id}

- Feedback – Submit and view feedback.
  → /feedback (POST), /feedback/event/{id}

- Reports – Event popularity, student participation, top students.
  → /reports/event-popularity, /reports/student-participation, /reports/top-students

5. Workflows

The workflows describe how the system handles **student event participation** from registration through reporting.

a. Student Registration Flow

1. A student browses available events.
2. The student selects an event and registers through the /registrations API.
3. A new record is created in the **Registrations** table, linking the student and event with a status (Registered).

b. Attendance Flow

1. On the day of the event, the student checks in.
2. The system records this via the /attendance API.
3. The attendance record is linked to the student's registration ID.

c. Feedback Flow

1. After attending, the student can submit feedback (rating and comment).
2. This is stored in the **Feedback** table using the /feedback API.

d. Reporting Flow

1. Admins or staff can view event reports via dedicated APIs.
2. Reports include:
   - **Event Popularity** – number of registrations per event.
   - **Attendance Reports** – attendance percentage per event.
   - **Student Participation** – number of events attended per student.
   - **Top Students** – most active participants across colleges.

The flow can be summarized as:

Registration → Attendance → Feedback → Reporting

This ensures every student's journey from registering for an event to contributing feedback is tracked, and meaningful reports can be generated

6. UI

For this project, I built a **very simple frontend using HTML, CSS, and JavaScript**. The purpose of the UI was not to create a full-fledged portal or student app, but to:

- Test and validate the APIs visually.
- Showcase how the backend data (students, events, registrations, attendance, and reports) can be displayed on a clean and minimal interface.
- Make it easier for reviewers/recruiters to try the system without writing API calls manually.

The frontend is deliberately minimalistic — just enough to demonstrate how endpoints work and how data flows from the database to the user interface. The core focus of this assignment was on **backend design, database structure, and reporting APIs**, and the UI plays a supportive role to illustrate those capabilities.

7. Assumptions and Edge Casses

To keep this prototype simple, I made a few assumptions and noted possible edge cases:

Assumptions:

- Each event belongs to exactly one college.
- Event IDs and Student IDs are unique system wide.
- A student can register for multiple events, but only once per event.
- Feedback is optional but can only be given if a student has attended the event.
- Attendance is linked to registration, meaning a student must be registered to be marked present.
- The system is designed for small to medium scale (SQLite) but can be extended to larger databases like PostgreSQL or MySQL.
- Event creation is tracked with a created by field (instead of a dedicated Admin table).

Edge Cases:

- Duplicate Registrations: If a student tries to register twice, the API should block or ignore duplicates.
- Cancelled Registrations: Students may cancel; these are tracked with a status field.
- Attendance without Registration: Not allowed — attendance only works if registration exists.
- Missing Feedback: Some students may not provide feedback; reports handle averages accordingly.
- Event Capacity: The prototype does not enforce strict limits, but in a full system, registrations should not exceed event capacity.
- Data Scale: For now, SQLite is fine, but with ~50 colleges × 500 students × 20 events per semester, scaling would need optimization.