Documentation & Design

---

1. My Approach

When I started this project, I first thought about the real problem: **colleges conduct a lot of events, but there is no simple way to manage registrations, attendance, and feedback in one place**.
So, my goal was to build a **centralized system** where both students and admins can interact easily.

Assumptions & Decisions

- Each student belongs to **exactly one college**.
- A student can register for **multiple events**, but **only once per event** (no duplicates).
- Attendance is only recorded if the student has already registered.
- Feedback is **optional**, but if given, it must be **one per student per event**.
- Database used: **SQLite** (lightweight and sufficient for a prototype).
- Backend built with **Flask**, and frontend with **HTML + Bootstrap + Chart.js** for simple UI and reports.

Use of AI (LLM Tools)

- I used **ChatGPT** to brainstorm the **data model** and **API structure**.
- I followed the AI's suggestions for:
    - Designing entities (College, Student, Event, Registration, Feedback).
    - Creating REST API endpoints.
    - Writing SQL queries for reports.
- I **deviated** when building the UI — instead of plain HTML, I added **Bootstrap and charts** to make it more presentable.

---

2. Design Document

Data to Track

- **Event creation** → title, type (workshop/fest/seminar), time, location.
- **Student registration** → student linked to event.
- **Attendance** → whether a registered student attended.
- **Feedback** → rating and comment after the event.

Database Schema (ER Diagram in Words)

- **College (id, name)**
  → has many students and events.
- **Student (id, college_id, roll_no, name, email)**
  → belongs to a college, can register for many events.
- **Event (id, college_id, title, type, start_time, end_time, location, description)**
  → belongs to a college, can have many registrations and feedbacks.
- **Registration (id, student_id, event_id, attended, checkin_time)**
  → links student to event, records attendance.
- **Feedback (id, student_id, event_id, rating, comment)**
  → one feedback per student per event.

API Design

Main Endpoints

- `POST /colleges` → Add a college
- `POST /students` → Register student
- `POST /events` → Create event
- `GET /events` → List all events
- `POST /register` → Student registers for event
- `POST /attendance` → Mark attendance
- `POST /feedback` → Submit feedback

Reports

- `GET /reports/event-popularity` → Event name + registrations
- `GET /reports/student-participation` → Student name + events attended
- `GET /reports/average-feedback?event_id=1` → Average rating for event
- `GET /reports/top-active?limit=3` → Top active students

Workflows

Registration → Attendance → Feedback → Reporting

1. A student **registers** for an event (entry created in `Registration`).
2. On event day, **attendance** is marked (updates `Registration`).
3. After event, student can **submit feedback** (entry in `Feedback`).
4. Admin generates **reports**:
   - Popular events
   - Participation per student
   - Average feedback score
   - Top active students

Assumptions & Edge Cases

- Duplicate registrations are blocked using **unique constraints**.
- Missing feedback does not break reports — averages are calculated only from available entries.
- Cancelled events are ignored in reporting.
- Attendance is only valid if the student was registered.

✅ This document summarizes my **approach, assumptions, design, workflows, and edge cases** in a clear way.