**Campus Event Reporting System – Design & Implementation Document**

**1️⃣ Approach**

Objective:

Build a system to track college events, student registrations, attendance, and feedback, and generate reports on event popularity and student participation.

Assumptions & Decisions:

Each college can host multiple events.

Students can register for multiple events.

Attendance is linked to registrations.

Feedback is optional and numeric (1–5).

Event cancellations are tracked but don’t delete historical attendance or feedback.

Duplicate registrations are prevented.

Used ChatGPT to brainstorm DB schema, API design, and workflows. Some suggestions were customized for practical implementation.

**Tools Used:**

Database: SQLite

Language: Python

AI Tool: ChatGPT for design ideas, queries, and workflow suggestions

A screenshot of a computer program

AI-generated content may be incorrect.

2️⃣ **Design Document**

a) Data to Track

=> Event Creation: title, date, type, college.

=> Student Registration: student\_id, event\_id, timestamp.

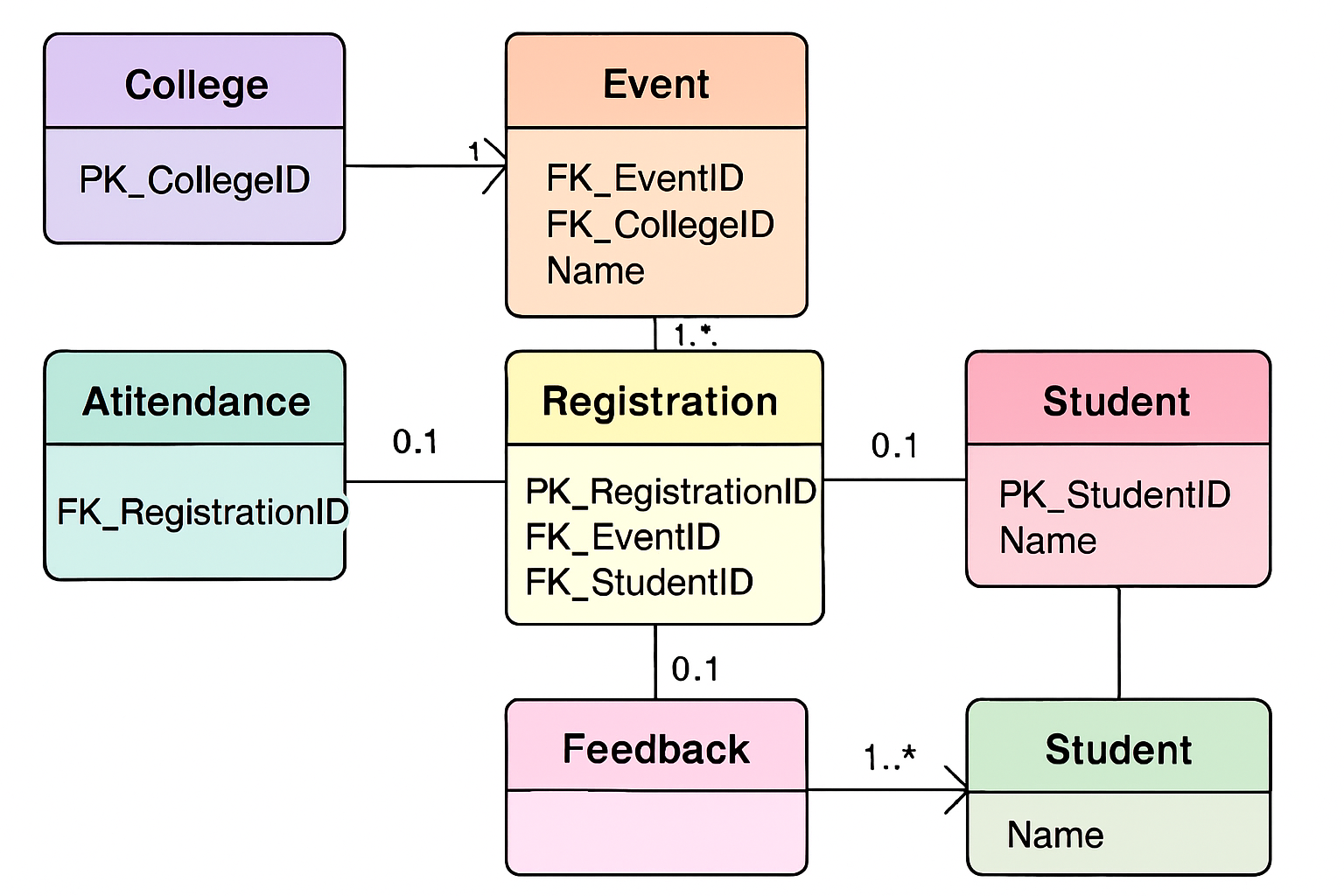
=> Attendance: registration\_id, status (present/absent), timestamp.

=> Feedback: registration\_id, rating (1–5), optional comment, timestamp.

**b) Database Schema**

**ER Diagram:**

Student (id PK, college\_id FK, name, email)



**c) API Design / Endpoints**

| **Endpoint** | **Method** | **Description** | **Request** | **Response** |
| --- | --- | --- | --- | --- |
| /events | POST | Create event | {title, type, date, college\_id} | {event\_id, status} |
| /students/register | POST | Register student | {student\_id, event\_id} | {registration\_id, status} |
| /attendance | POST | Mark attendance | {registration\_id, status} | {success} |
| /feedback | POST | Submit feedback | {registration\_id, rating, comment} | {success} |
| /reports/event\_popularity | GET | Registrations per event | - | [event\_id, title, total\_registrations] |
| /reports/student\_participation | GET | Student attendance | - | [student\_id, name, attended\_events\_count] |
| /reports/event\_feedback | GET | Average feedback score | - | [event\_id, title, avg\_rating] |

**d) Workflows**

**1. Registration → Attendance → Feedback → Reporting**

Student -> Registration API -> Creates Registration

Registration -> Attendance API -> Marks Attendance

Registration -> Feedback API -> Optional Feedback

Reports API -> Generates Event Popularity / Student Participation / Feedback Reports

**Sequence Diagram (Text Version):**

Student

|

v

Registration API ----> Registration Table

|

v

Attendance API ----> Attendance Table

|

v

Feedback API ------> Feedback Table

|

v

Reports API ------> Queries Tables -> Generates Reports

**e) Assumptions & Edge Cases**

* Duplicate registration attempts are rejected.
* Missing feedback is recorded as NULL.
* Cancelled events are flagged but historical data remains.
* Attendance duplicates should be removed.

**3️⃣ Prototype Implementation (Python + SQLite):**

**import sqlite3**

**# Connect to SQLite DB**

**conn = sqlite3.connect("D:\Downloads\CampusEventReporting-Prototype\instance\event\_reporting.db")**

**cur = conn.cursor()**

**print("📌 Tables in the database:")**

**cur.execute("SELECT name FROM sqlite\_master WHERE type='table';")**

**print(cur.fetchall())**

**print("\n📌 Events:")**

**cur.execute("SELECT \* FROM event;")**

**for row in cur.fetchall():**

**print(row)**

**print("\n📌 Students:")**

**cur.execute("SELECT \* FROM Student;")**

**for row in cur.fetchall():**

**print(row)**

**print("\n📌 Registrations:")**

**# cur.execute("SELECT \* FROM Registration;")**

**# for row in cur.fetchall():**

**#     print(row)**

**# Total registrations per event**

**cur.execute("""**

**SELECT e.title, COUNT(r.id) AS total\_registrations**

**FROM event e**

**LEFT JOIN registration r ON e.id = r.event\_id**

**GROUP BY e.id**

**ORDER BY total\_registrations DESC**

**""")**

**print("📌 Event Popularity Report:")**

**for row in cur.fetchall():**

**print(f"Event: {row[0]}, Total Registrations: {row[1]}")**

**print("\n📌 Attendance:")**

**cur.execute("SELECT \* FROM Attendance;")**

**for row in cur.fetchall():**

**print(row)**

**cur.execute("""**

**SELECT s.name, COUNT(DISTINCT a.id) AS attended\_events**

**FROM student s**

**JOIN registration r ON s.id = r.student\_id**

**JOIN attendance a ON r.id = a.event\_id**

**WHERE a.status='present'**

**GROUP BY s.id**

**ORDER BY attended\_events DESC**

**""")**

**print("\n📌 Student Participation Report:")**

**for row in cur.fetchall():**

**print(f"Student: {row[0]}, Events Attended: {row[1]}")**

**# print("\n📌 Feedback:")**

**# cur.execute("SELECT \* FROM Feedback;")**

**# for row in cur.fetchall():**

**#     print(row)**

**cur.execute("""**

**SELECT e.title, ROUND(AVG(f.rating), 2) AS avg\_feedback**

**FROM event e**

**LEFT JOIN registration r ON e.id = r.event\_id**

**LEFT JOIN feedback f ON r.id = f.event\_id**

**GROUP BY e.id**

**ORDER BY avg\_feedback DESC**

**""")**

**print("\n📌 Average Feedback Score per Event:")**

**for row in cur.fetchall():**

**print(f"Event: {row[0]}, Average Rating: {row[1]}")**

**cur.execute("""**

**DELETE FROM attendance**

**WHERE rowid NOT IN (**

**SELECT MIN(rowid)**

**FROM attendance**

**GROUP BY event\_id**

**)**

**""")**

**conn.commit()**

**conn.close()**

**4️⃣ Sample Reports:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**5️⃣ Conclusion**

* System tracks events, registrations, attendance, and feedback.
* Generates meaningful reports: event popularity, student participation, average feedback.
* Handles edge cases like duplicate registration, missing feedback, and cancelled events.
* Prototype in Python + SQLite is functional, easily extendable to API-based frameworks.