Campus Event Management Platform: Design Document

1. Data to Track

The system is built on a relational data model to track event and student interactions. The core entities and their attributes are as follows:

- **Events**: Details about each event.
  - `EventID`: Unique identifier for the event.
  - `Title`: Name of the event.
  - `Description`: Detailed information about the event.
  - `Date`: The date and time of the event.
  - `Location`: Venue of the event.
  - `EventType`: Category of the event (e.g., Workshop, Fest, Tech Talk).
  - `CollegeID`: A foreign key linking the event to a specific college.
- **Students**: Information about each registered student.
  - `StudentID`: Unique identifier for the student.
  - `Name`: Full name of the student.
  - `Email`: Student's email address (must be unique).
  - `CollegeID`: A foreign key linking the student to their college.
- **Registrations**: Links students to the events they have signed up for.
  - `RegistrationID`: Unique identifier for the registration.
  - `StudentID`: Foreign key to the `Students` table.
  - `EventID`: Foreign key to the `Events` table.
  - `RegistrationDate`: Timestamp of when the student registered.
- **Attendance**: Tracks which registered students attended an event.
  - `AttendanceID`: Unique identifier for the attendance record.
  - `RegistrationID`: Foreign key to the `Registrations` table.
  - `CheckInTime`: Timestamp of when the student checked in.
- **Feedback**: Collects ratings and comments from students after an event.
  - `FeedbackID`: Unique identifier for the feedback record.
  - `RegistrationID`: Foreign key to the `Registrations` table.
  - `Rating`: A rating from 1 to 5.
  - `Comments`: Optional text feedback.

---

2. Database Schema

The database is structured to support the relationships between these entities. We are using a relational database (MySQL) to maintain data integrity. The schema includes tables for each entity, with primary and foreign keys to establish relationships.

Table Sketch:

- **`colleges`**: `id` (PK), `name`
- **`events`**: `id` (PK), `college_id` (FK), `title`, `description`, `date`, `location`, `event_type`
- **`students`**: `id` (PK), `college_id` (FK), `name`, `email`

- **`registrations`**: `id` (PK), `student_id` (FK), `event_id` (FK), `registration_date`
- **`attendance`**: `id` (PK), `registration_id` (FK), `check_in_time`
- **`feedback`**: `id` (PK), `registration_id` (FK), `rating`, `comments`

---

3. API Design

The system uses a **RESTful API** approach. All endpoints are well-defined and use standard HTTP methods for communication.

- **Registration:**
  - `POST /api/registrations`: Registers a student for an event.
- **Attendance:**
  - `POST /api/attendance`: Marks a student as attended for a specific registration.
- **Feedback:**
  - `POST /api/feedback`: Submits a rating and optional comments for a registration.
- **Reports:**
  - `GET /api/reports/popularity`: Returns a list of events sorted by the number of registrations.
  - `GET /api/reports/registrations/:event_id`: Returns the total count of registrations for a single event.
  - `GET /api/reports/attendance/:event_id`: Calculates and returns the attendance percentage for a single event.
  - `GET /api/reports/feedback/:event_id`: Calculates and returns the average feedback score for a single event.
  - `GET /api/reports/top-students`: Returns the top 3 students who have attended the most events.

---

4. Workflows

The main user workflow can be represented in a sequence diagram, showing the interaction between the Student App (Frontend), the API Server (Backend), and the Database.

Workflow for Registration & Reporting:

1. **Student App** sends a `POST` request to `API Server` with `studentID` and `eventID`.
2. **API Server** validates the request and inserts a new row into the `registrations` table in the **Database**.
3. On the day of the event, an admin marks attendance, which triggers another API call to record the `CheckInTime` in the `attendance` table.
4. For reporting, the **Student App** or **Admin Portal** sends a `GET` request to a report endpoint on the **API Server**.
5. **API Server** executes a complex query on the **Database** (e.g., to count registrations or calculate percentages).
6. **API Server** returns the formatted data back to the **Student App** to be displayed.

5. Assumptions & Edge Cases

- **Scale:** The system is designed to handle medium-scale usage (`~50` colleges, `~500` students/college) with a single, centralized database.
- **Uniqueness:** Event IDs are assumed to be globally unique to prevent conflicts across colleges.
- **Duplicate Registrations:** The database is designed with a unique composite key on `(student_id, event_id)` to automatically prevent a student from registering for the same event more than once.
- **Missing Data:** Feedback (`rating` and `comments`) is optional and handled gracefully; reports will filter out any null values.
- **Cancelled Events:** This prototype does not account for cancelled events. In a full system, an `is_cancelled` field would be added to the `events` table.