**Design Document – Campus Event Management System**

This document explains how I approached designing my Webknot assignment project: a **Campus Event Management System**. While working on this, I tried to think not just as a coder but also as a student who wants an event system that’s actually easy to use. My focus was on keeping the design clean, simple, and flexible enough for improvements later.

**1. Data to Track**

The first thing I asked myself was: *What exactly do we need to track so the system makes sense?* After some thought, I decided the essential data points are:

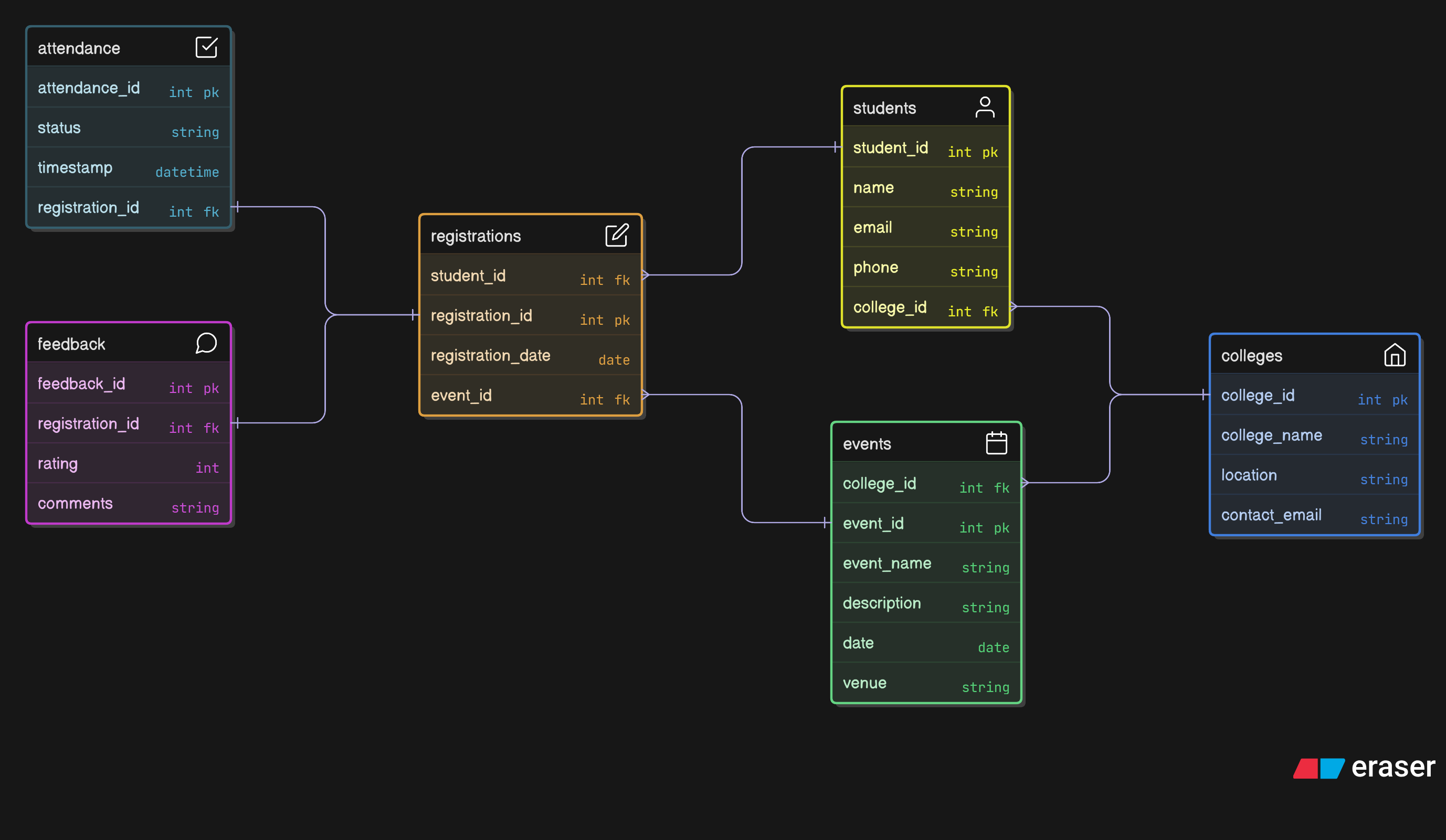
* **Events:** title, description, type (workshop, seminar, cultural), date, time, venue, maximum participants, and associated college.
* **Students:** name, email, student ID, and their college.
* **Registrations:** which student registered for which event, with a timestamp.
* **Attendance:** whether a registered student actually showed up.
* **Feedback:** ratings (1–5 stars) plus optional comments from students after the event.

I kept the list short and focused so that all the core features (like reporting and analysis) could still be supported without making things unnecessarily complicated.

**2. Database Schema**

I chose a **relational database** with six main tables because it’s organized, avoids duplication, and makes reporting easier.

* **Colleges** – Info about each college/university.
* **Students** – Profiles linked to a college.
* **Events** – Details of events, also linked to a college.
* **Registrations** – Connects students and events.
* **Attendance** – Tracks who attended.
* **Feedback** – Stores ratings and comments.



**Why I designed it this way:**

* A **college** can have many students and many events.
* A **student** can join multiple events, but only once per event.
* An **event** can have multiple registrations, attendance entries, and feedback.

This schema felt logical and expandable. If I ever want to add features like certificates or notifications, it’ll still fit neatly.

**3. API Design**

For the backend, I structured **RESTful APIs** because they’re simple, standard, and easy for the frontend to consume.

* **Events**: create, list, view details.
* **Registrations**: register students while preventing duplicates.
* **Attendance**: mark attendance only for registered students.
* **Feedback**: collect post-event feedback.
* **Reports**: generate insights like most popular events or top active students.

The API will be available at `http://localhost:8000`

These are the API end points

**### Events**

- `POST /events` - Create event

- `GET /events` - List events (with filters: college\_id, event\_type)

- `GET /events/{id}` - Get specific event

**### Registrations**

- `POST /events/{id}/register` - Register for event (returns 409 if already registered)

- `POST /events/{id}/attendance` - Mark attendance

- `POST /events/{id}/feedback` - Submit feedback (rating 1-5)

**### Reports**

- `GET /reports/event-popularity` - Events sorted by registrations

- `GET /reports/student-participation` - Students by events attended

- `GET /reports/top-active-students?limit=N` - Top active students

I liked this structure because it breaks down big tasks into small, clear endpoints.

**4. Workflows**

To keep things practical, I mapped out the flow like this:

1. Students **view events → register**.
2. On the event day, **attendance** is marked.
3. Afterward, students **submit feedback**.
4. Admins can then **view reports** (popular events, active students, etc.).

This step-by-step workflow gave me confidence that nothing would get left out.

**5. Assumptions & Edge Cases**

While designing, I made a few assumptions:

* Students cannot register for the same event twice.
* Feedback is optional — not everyone fills it.
* If an event gets canceled, related registrations, attendance, and feedback should also be cleaned up.
* Attendance is only valid for registered students (no random entries).

I wanted the system to be student-friendly but strict enough to avoid messy data.

**6.Reports**

These are the queries for the report

- `GET /reports/event-popularity` - Events sorted by registrations

- `GET /reports/student-participation` - Students by events attended

- `GET /reports/top-active-students?limit=N` - Top active students

Example API Calls

**1. Create Event**

```bash

curl -X POST "http://localhost:8000/events" \

-H "Content-Type: application/json" \

-d '{

"title": "Python Workshop",

"description": "Learn Python fundamentals",

"event\_type": "workshop",

"date": "2024-02-15T10:00:00",

"location": "Computer Lab A",

"max\_participants": 30

}'

```

**2. Register for Event**

```bash

curl -X POST "http://localhost:8000/events/1/register" \

-H "Content-Type: application/json" \

-d '{

"student\_id": 1

}'

```

**3. Mark Attendance**

```bash

curl -X POST "http://localhost:8000/events/1/attendance" \

-H "Content-Type: application/json" \

-d '{

"student\_id": 1

}'

```

**4. Submit Feedback**

```bash

curl -X POST "http://localhost:8000/events/1/feedback" \

-H "Content-Type: application/json" \

-d '{

"student\_id": 1,

"rating": 5,

"comment": "Excellent workshop!"

}'

```

**5. Get Event Popularity Report**

```bash

curl "http://localhost:8000/reports/event-popularity"

```

**6. Get Top Active Students**

```bash

curl "http://localhost:8000/reports/top-active-students?limit=3"

```

**7. Reflection**

Honestly, this was one of those assignments where I felt both challenged and excited. I had to make decisions at every step — from keeping the schema simple to choosing REST APIs over something heavier. I also spent time imagining how students and admins would actually use the system, which made the design more practical.

What I’m most proud of is how I balanced functionality and simplicity. I didn’t try to cram in everything, but I made sure the essentials (registrations, attendance, feedback, reports) worked smoothly. At the same time, I kept the design modular, so I can add extra features later without breaking the core.

This project was more than just “finishing a task” for me. It was about thinking like a developer, learning to justify my choices, and building something that I’d genuinely want to use in a real college setting.