Design Document: Campus Event Reporting System

1. Data to Track
2. Database Schema
3. API Design
4. Workflows
5. Assumptions & Edge Cases

1. Data to Track

The system needs to track the following data:

- **Students** → Name, College ID
- **Events** → Title, Type (e.g., Tech, Cultural), College ID
- **Registrations** → Which student registered for which event
- **Attendance** → Whether a registered student attended the event or not
- **Feedback** → Ratings (1–5) provided by students for events.

---

2. Database Schema

The system uses an **SQLite relational database** with the following tables:

1. **Students**
    - id (PK)
    - name
    - college_id
2. **Events**
    - id (PK)
    - title
    - type
    - college_id
3. **Registrations**
    - id (PK)
    - student_id (FK → Students.id)
    - event_id (FK → Events.id)
4. **Attendance**
    - Id (PK)

- student_id (FK → Students.id)
- event_id (FK → Events.id)
- status (present/absent)

5. **Feedback**
    - id (PK)
    - student_id (FK → Students.id)
    - event_id (FK → Events.id)
    - rating (1–5)

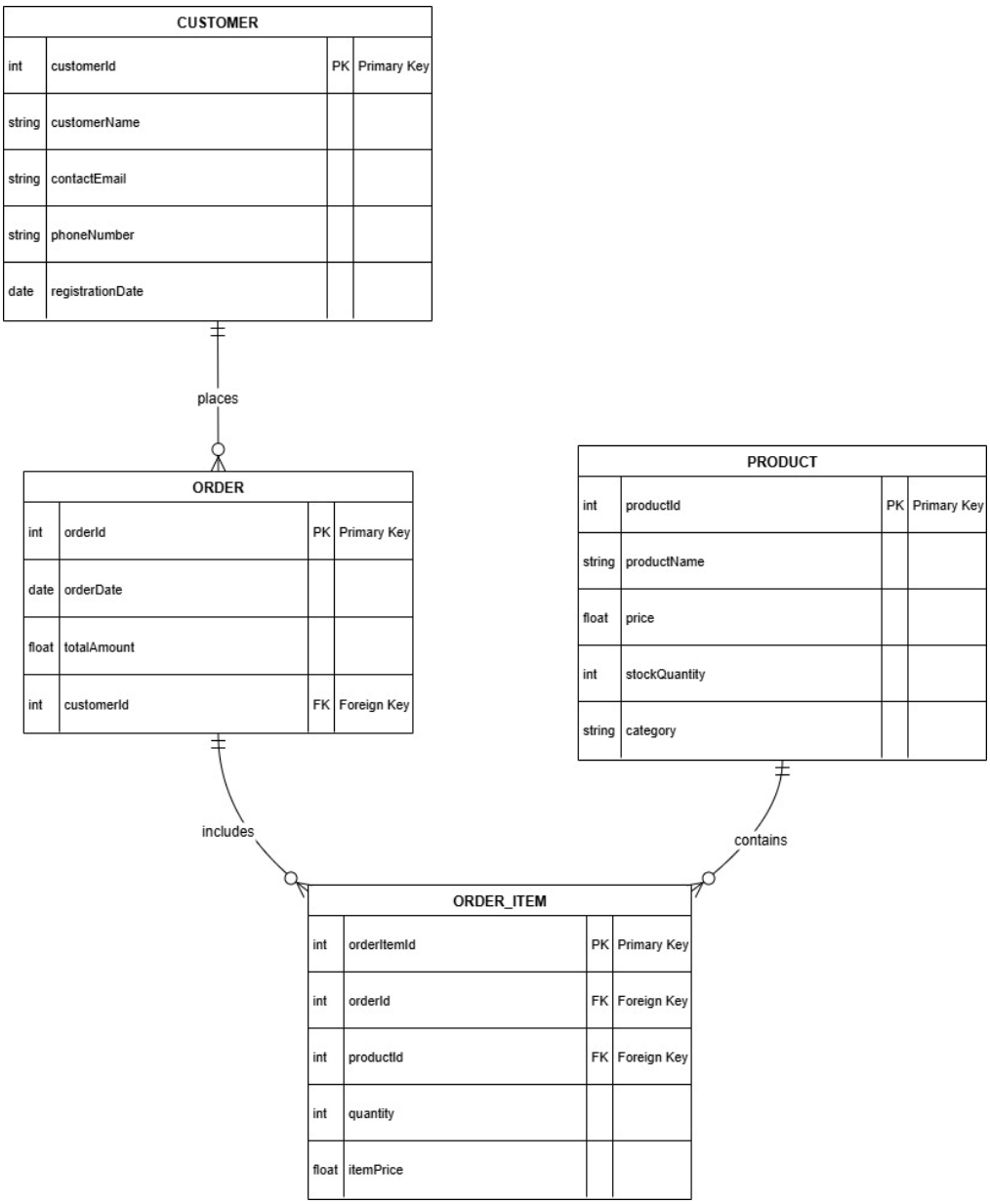

Figure 1: ER Diagram for Database Schema

3. API Design

Method	Endpoint	Description
POST	/register_student	Register a new student
POST	/create_event	Create a new event
POST	/register_event	Register a student for an event
POST	/mark_attendance	Mark attendance for a student
POST	/submit_feedback	Submit feedback for an event
GET	/report/registrations	Get number of registrations per event
GET	/report/attendance	Get attendance count per event
GET	/report/feedback	Get average feedback rating per event

EXAMPLE: 1. Create Event API

```
{
"title": "Tech Talk: AI for Beginners",
"type": "Workshop",
"college_id": 101
}
```

2. Register Student

```
{
 "name": "Alice",
 "college_id": 101
}
```

3. Mark Attendance

```
{
 "student_id": 1,
 "event_id": 1,
 "status": "present"
}
```

4. Submit feedback

```
{
 "student_id": 1,
 "event_id": 1,
 "rating": 5
}
```

4. Workflows

Student Flow:

1. Student registers in the system.
2. Student registers for an event.
3. Student attends the event.
4. Student provides feedback.
5. Admin generates reports.

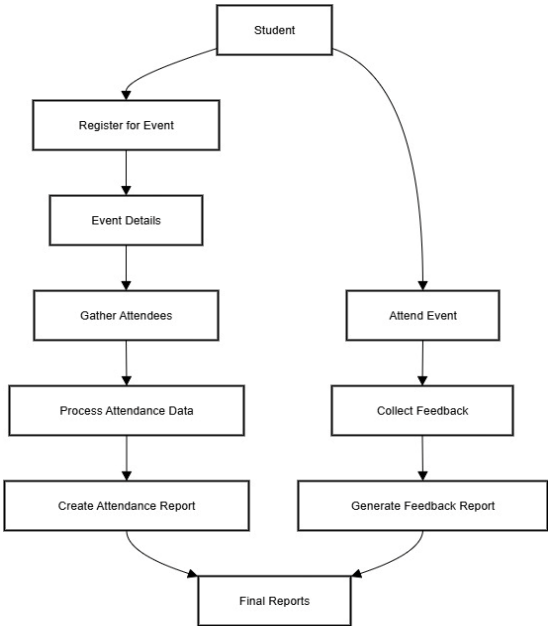

Figure 2: Workflow for Student → Event → Reports

- Actors: Student, System, Database
- Arrows showing requests (POST/GET) and responses.

---

5. Assumptions & Edge Cases

- Each student has a unique ID.
- Students must be registered before attending events.
- Duplicate registrations for the same event are not allowed.
- Feedback is optional.
- If no feedback is given, reports show None.
- Admins can only view reports, not alter attendance/feedback manually.

---

6. Reports

1. **Event Popularity Report**

Sorted by number of registrations.

Sample output:

```
[
  {"event": "Hackathon 2025", "registrations": 120},
  {"event": "Tech Talk", "registrations": 80},
  {"event": "Cultural Fest", "registrations": 60}
]
```

2. **Student Participation Report**

Shows how many events each student attended.

Sample Output:

```
[
  {"student": "Alice", "events_attended": 5},
  {"student": "Bob", "events_attended": 3},
  {"student": "Charlie", "events_attended": 2}
]
```

3. **Bonus: Top 3 Most Active Students**

Based on highest number of attended events.

Sample Output:

```
[
  {"student": "Alice", "events_attended": 5},
  {"student": "David", "events_attended": 4},
  {"student": "Bob", "events_attended": 3}
]
```
