


Event reporting system ideas



 Campus Drive Assignment – Webknot Technologies.pdf
PDF

Act as an expert in brainstorming ideas and building projects with 10+years of experience.

I got this assignment for a campus placement process , i have attached the task pdf for full details. in brief the project is about building a basic event reporting system for a campus event management platform with various features like event creation, student registration, attendance, feedback , and generating reports

I need your help in brainstorming ideas (dont give code)

1. what assumptions or edge cases i need to consider for this project
 2. what design for the database is ideal in this scenario(consider few different options)
 3. what api and workflow suits with the database design and features
 4. how do i show the outputs or reports for submission of projects
 5. any extra things to be added lmk
 6. im thinking of doing project in fast api and nextjs idk if its the optimal choice , but if not suggest the good ones i can consider
- also the dealine its tmr 3pm which is roughly 27 hrs from now (and in that i work for roughly 15-20 hrs max)

Perfect, I went through the assignment details  . Since you've got ~27
Campus Drive Assignment – Webkn...

hours and want to balance quality with feasibility, let's focus on **clarity, completeness, and smart scope** rather than building a huge system. I'll walk you through each of your points:

1. Assumptions & Edge Cases

You need to document these clearly in your submission:

- **Event lifecycle**
 - Events can be created, edited, or cancelled.
 - What happens if an event is cancelled after registrations? (Students notified? Data retained?)
- **Registration**
 - Prevent duplicate registration for the same student/event.
 - Handle registrations after event capacity is full.
 - Late/last-minute registrations (cutoff times).
- **Attendance**
 - Attendance can only be marked for registered students.

- Handle students checking in twice (ignore second).
 - **Feedback**
 - Allow feedback only if attendance is marked.
 - Missing feedback → handle in reports (default “not given”).
 - **Reports**
 - Handle events with 0 registrations (avoid divide-by-zero in % calculations).
 - Student who never attends any events → still included in participation reports?
 - **Scaling**
 - Unique IDs across multiple colleges (college-wise namespace vs global IDs).
 - Separation of college data or multi-tenant database.
-

2. Database Design (Options)

Option A – Simple Relational (good for prototype, SQLite/Postgres/MySQL)

Tables:

- **Colleges** → id, name, location
- **Students** → id, name, email, college_id
- **Events** → id, name, type, date, capacity, college_id
- **Registrations** → id, student_id, event_id, timestamp
- **Attendance** → id, registration_id, attended (bool), timestamp
- **Feedback** → id, registration_id, rating (1-5), comments

Option B – Denormalized for Speed (less joins, more duplication)

- **EventReports** table that stores precomputed counts (registrations, attendance, avg rating). Good for reporting dashboards.
- Useful if scale grows, but adds complexity.

Option C – Multi-Tenant Database

- One shared schema with `college_id` in every table (simple).
 - OR separate schemas per college (clean separation but harder to manage for 50 colleges).
- For your prototype: **Option A with `college_id` in every table** is clean and realistic.
-

3. API & Workflow

Core APIs (REST for simplicity with FastAPI):

- **Event Management**
 - **POST** `/events` → create event
 - **GET** `/events` → list events (filter by college/type/date)

- **Student Registration**
 - `POST /events/{event_id}/register`
 - Prevent duplicate registration
- **Attendance**
 - `POST /events/{event_id}/attendance (student_id)`
- **Feedback**
 - `POST /events/{event_id}/feedback (rating, comments)`
- **Reports**
 - `GET /reports/events/popularity` → sorted by registrations
 - `GET /reports/events/{id}/attendance` → % attendance
 - `GET /reports/students/{id}/participation` → # of events attended
 - Bonus: `GET /reports/students/top3`

Workflow Example:

1. Admin creates event (via web portal).
2. Student registers (via mobile app).
3. On event day → student checks in (attendance).
4. After event → student gives feedback.
5. Reports are generated from these.

4. Showing Outputs/Reports

You don't need a full UI since time is short. Options:

- **Simplest** → SQL query outputs or JSON API responses with screenshots.
- **Better** → A lightweight Next.js frontend or even Swagger UI (auto-generated by FastAPI) to display results.
- **Bonus polish** → Minimal dashboard page (table view of events, students, reports).

Reports you must show:

- Event Popularity Report (sorted registrations).
- Student Participation Report (events attended count).
- Attendance % per event.
- Avg feedback score per event.
- Bonus: Top 3 students report.

5. Extras to Add (Optional for “wow” factor)

- **Filters in reports** (by event type, by college).

- **Simple UI mockup** (Figma or hand-drawn wireframe). Even one page is enough.
- **Export reports** to CSV/Excel for realism.
- **QR-code check-in** for attendance (conceptual only, don't implement fully).

6. Tech Stack Choice

- **FastAPI + Next.js** → actually a great choice:
 - FastAPI → clean APIs, quick to implement, auto-docs.
 - Next.js → nice if you want polished frontend, but it may take more time.

Since you have 15–20 hrs:

- **Best trade-off:** FastAPI + SQLite/Postgres (focus backend + API + report outputs).
- If you want a UI: keep it very minimal → Next.js with just 2–3 pages (list events, report table).
- If UI takes too long → rely on FastAPI Swagger docs + JSON outputs + SQL queries (that's acceptable for submission).

✓ My recommendation for **your 27-hr window**:

- Do backend + database properly in FastAPI.
- Use SQLite (easy setup, works everywhere).
- Focus on correct workflows + API endpoints + report queries.
- Add **screenshots of Swagger UI + SQL queries as reports**.
- If time permits, add a small Next.js dashboard.

Would you like me to **draft the database schema (tables + ER diagram) and the API workflow diagram** so you can directly use them in your design doc, or do you prefer me to just give structured descriptions so you can create the diagrams yourself quickly?

Sources



whats a ideal thing to do

lets say the system will be used by 50 college with around 500+students with roughly 20 events per semester

- Should event IDs be unique across colleges?
- Would you keep data separate per college, or maintain one large dataset?

Great question 🙌 — this is where design choices make or break scalability and clarity. Let's walk through both options and then I'll tell you the **ideal approach for your scenario (50 colleges, ~500 students each, ~20 events per semester)**.

1. Should event IDs be unique across colleges?

- **Option A – Global Unique IDs**

- Each event gets a unique ID across the entire system (e.g., UUID or auto-increment without reset).
- Pros: Simpler queries when doing system-wide reporting (like "top events across all colleges").
- Cons: Event IDs don't have college context (event_id 101 doesn't mean anything by itself).

- **Option B – Scoped IDs per College**

- Each college has its own sequence for event IDs. (e.g., college_id + event_id composite key).
- Pros: Cleaner for admins — College A can say "Event #5" without clashing with College B's Event #5.
- Cons: Requires composite keys in queries (college_id + event_id).

✓ **Ideal for you:** Use **global unique IDs** (simplifies database integrity and joins). Just include a college_id column in Events table so you can filter by college when needed.

2. Separate data per college vs one large dataset?

- **Option A – Separate databases/schemas per college**

- Each college has its own tables (like students_collegeA, events_collegeA).
- Pros: Clean isolation (one college's data won't affect another).
- Cons: Maintenance nightmare — schema changes must be applied to all, queries across colleges become complex, scaling is harder.

- **Option B – One large dataset (multi-tenant)**

- One set of tables, but every row has a college_id.
- Pros: Easier to manage, extend, and run reports (per-college or global).
- Scales fine with your numbers (50 colleges × 500 students × 20 events = only ~500,000 records max per year, trivial for Postgres/MySQL).
- Cons: Need to ensure queries always filter by college_id for security and performance.

✓ **Ideal for you:** One large dataset with a college_id foreign key in all main tables.

This is the **standard SaaS multi-tenant design** — clean, manageable, and future-proof.

So, the ideal setup for your assignment

- **Global unique IDs** for events (primary key).
- **One shared database schema** with college_id as a foreign key in Students, Events, Registrations, etc.
- Always query data with college_id filter for isolation.

Example table (simplified):

```
sql
```

```
Events
```

```
-----
```

```
event_id (PK)
```

```
college_id (FK -> Colleges.college_id)
```

```
name
```

```
type
```

```
date
```

```
capacity
```

This way:

- You can easily do **college-specific reports** (WHERE college_id = ?)
- Or **global reports** across colleges (just don't filter).

👉 Question for you: do you want me to also sketch out the **ER diagram (tables + relations)** with this "one large dataset + global IDs" approach so you can directly paste it in your design doc, or would you prefer just a structured table design list?



i was thinking of doing separate database per college to keep the database easier to manage

That's a solid instinct 👍 — **separating databases per college feels cleaner on paper**, because each college gets its own sandbox, but let's weigh this carefully against your scenario and deadline.

✅ Pros of Separate Database per College

- **Isolation:** Bugs, bad data, or accidental queries in College A won't affect College B.
- **Easier to backup/restore per college:** If College C wants their database exported, you can hand it over easily.
- **Simple IDs:** Each college can have Event ID = 1, Student ID = 1, etc. (no collisions).

⚠️ Cons of Separate Database per College

- **Maintenance Overhead:** Any schema change (e.g., add "event category") must be applied across 50 DBs.
- **Cross-college reports are hard:** "Top 3 active students across all colleges" → requires aggregating from all DBs.
- **Scaling DevOps:** If this grows beyond 50 colleges (say 200+), managing so many DBs gets painful.

⚖️ Realistic Recommendation

Since your assignment mentions **~50 colleges × 500 students × 20 events** (≈ 500k rows/year max — trivial size), the **multi-tenant single DB with college_id** is actually *easier to manage long-term*.

But — since you want to keep it **easier for the prototype/demo**, here's a hybrid way to think about it:

- **Prototype/demo assignment:**
Use **one shared database** (faster to build + easier reporting).
- **Mention in your design doc:**
"In a real deployment, we could also use separate databases per college for stronger isolation, especially if colleges demand ownership of their data. However, for this prototype we used a single shared schema with a `college_id` key, as it simplifies reporting and aligns with the small scale (~50 colleges)."

That way, you cover both bases: you show you *considered* the isolation approach, but you implement the practical one within your 27-hour window. 🚀

👉 Do you want me to **draft two ER diagrams** for you — one for "single DB multi-tenant" and one for "separate DB per college" — so you can show in your report that you thought through both approaches? That'll give you bonus points in the "design decisions" section.



↳

****Maintenance Overhead:**** Any schema change (e.g., add "event category") must be applied across 50 DBs.

in that pov i feel one database would be managable for accesing reports, so lets go with that

can u suggest some example database design i can considered

Perfect 👍 — one shared database (multi-tenant with `college_id`) is the right call for your scale and especially for reporting. Let me suggest a **clean, normalized design** that you can use in your design doc.



Example Database Design (One Database for All Colleges)

Tables & Relationships

1. Colleges

- `college_id` (PK)
- `name`
- `location`

2. Students

- `student_id` (PK)
- `college_id` (FK → Colleges)
- `name`
- `email` (unique within college)
- `roll_number` (optional, unique per college)

3. Events

- **event_id** (PK)
- **college_id** (FK → Colleges)
- **title**
- **event_type** (Workshop / Seminar / Fest / Hackathon)
- **start_date**
- **end_date**
- **capacity**

4. Registrations

- **registration_id** (PK)
- **student_id** (FK → Students)
- **event_id** (FK → Events)
- **registered_at** (timestamp)
- **Unique constraint** → (**student_id**, **event_id**) (to avoid duplicates)

5. Attendance

- **attendance_id** (PK)
- **registration_id** (FK → Registrations)
- **status** (Present / Absent)
- **marked_at** (timestamp)

6. Feedback

- **feedback_id** (PK)
- **registration_id** (FK → Registrations)
- **rating** (1–5)
- **comments** (optional)
- **Constraint:** only one feedback per registration

ER Diagram (described in words)

- One **College** has many **Students** and many **Events**.
 - A **Student** can register for many **Events** (via **Registrations** table).
 - Each **Registration** can have one **Attendance** and one **Feedback** entry.
-

Why this design works

- Easy to **filter by college_id** to keep reports isolated.
- Supports **global queries** (e.g., “Top 3 students across all colleges”).
- Enforces constraints (no duplicate registration, no feedback without registration).

- Scales fine for your expected size (few hundred thousand rows).

Reports you can generate with this schema

- **Event Popularity** → Count of registrations per event (ORDER BY count DESC).
 - **Attendance %** → $\text{SUM}(\text{attendance.status} = \text{'Present'}) / \text{COUNT}(*).$
 - **Avg Feedback Score** → $\text{AVG}(\text{feedback.rating})$ per event.
 - **Student Participation** → Count of events attended by each student.
 - **Top 3 Active Students** → Order by attendance count DESC, LIMIT 3.
-

👉 Question for you: do you want me to **sketch this ER diagram visually** (boxes + relationships) so you can directly paste it into your design doc, or do you prefer me to keep it in structured text form only?