

# AI Logs

## Prompt 1 (Developer → LLM):

"I need to build a minimal event reporting prototype for a campus event platform. Suggest a simple schema and minimal APIs to support event creation, registration, attendance and feedback."

## AI (suggested):

"Use tables: `Colleges`, `Students`, `Events`, `Registrations`, `Attendance`, `Feedback`. Keep `event_id` as UUID. Minimal endpoints: `POST /events`, `POST /register`, `POST /attendance`, `POST /feedback`, `GET /reports/...`."

---

## Prompt 2 (Developer → LLM):

"What constraints should I add to prevent duplicate registrations and to compute attendance percentage?"

## AI (suggested):

"Add a unique constraint on `(event_id, student_id)` in `Registrations`. For attendance percentage:  $\text{attendance\_percentage} = (\text{present\_count} / \text{total\_registrations}) * 100$ ."

---

## Prompt 3 (Developer → LLM):

"I only have a few hours — should I use SQLite or Postgres for the prototype?"

## AI (suggested):

"Use **SQLite** for speed and simplicity in a time-bound prototype. Keep SQL compatible with Postgres for later migration."

---

**Note:** Save screenshots of this chat window (these three Q&A blocks are enough). Place them under `AI_logs/` in your repo.

---

Topic	AI Suggestion	My Decision (followed / deviated)	Rationale
DB choice	Suggested SQLite for prototype	<b>Followed</b>	Very quick to implement and acceptable for this assignment.
IDs	Use UUIDs for events	<b>Followed</b>	Ensures global uniqueness across colleges.
Separate DB per college	AI: option to separate	<b>Deviated</b> (kept single dataset)	Single DB simplifies reporting across colleges and faster to implement.
Attendance model	Separate Attendance table per check-in	<b>Followed (simplified)</b>	Implement a single Attendance table linked to Registration with a status field.
Detailed auth	AI suggested auth layer	<b>Deviated</b> (stubbed)	Authentication is out-of-scope given time constraints; will document as an improvement.

## 1. Tech stack (prototype)

- **Node.js (LTS)** + **Express** for server and APIs.
- **SQLite** (file-based) as prototype DB for speed and zero infra.
- Use **Sequelize** ORM for simple migrations/models (portable to Postgres later) — alternatively use **knex** if you prefer query builder.

## 2. Project layout

- Simple structure:

```
bash                                                                    Copy code

/backend
├─ package.json
├─ src/
│  ├─ models/      # Sequelize models
│  ├─ routes/
│  ├─ controllers/
│  ├─ db.js        # Sequelize initialization
└─ app.js
```

## 3. IDs

- Use **UUID v4** for **event\_id**, **registration\_id**.
- **student\_id** remains as college roll string plus **college\_id** column.

## 4. API style

- RESTful JSON endpoints:
  - **POST /api/events** — create event (admin)
  - **GET /api/events** — list/filter events
  - **POST /api/register** — register student
  - **POST /api/attendance** — mark attendance
  - **POST /api/feedback** — submit feedback ↓