Campus Event Manager – Design Document

1. Assumptions

- A user can either be an **Admin** or a **Student**.
- Admins create and manage events, while students only register and give feedback.
- Every registration connects a student to an event. Attendance and feedback are tied to this registration.
- Feedback is optional, but when given it must include a rating between 1–5.
- For the prototype, authentication is simplified (user_id and role are passed in API requests).
- SQLite is used for quick setup, but the schema works for PostgreSQL too.

---

2. Key Decisions

- Chose **Django + Django Ninja** for backend (quick API development, automatic docs).
- Used a **custom User model** with role field instead of Django's built-in auth system (simpler for assignment).
- React was used for a minimal student interface with Tailwind for styling.
- Django Admin was kept for Admin features (create events, view registrations, mark attendance).
- Focused only on MVP features instead of building a large system.

---

3. Database Schema (ER Design)

Entities

- **User** → id, name, email, password, role (admin/student)
- **Event** → id, title, description, type, start_datetime, end_datetime, location, capacity, created_by (Admin FK)
- **Registration** → id, event (FK), student (FK), registered_at, status
- **Attendance** → id, registration (FK), present, checked_in_at
- **Feedback** → id, registration (FK), rating, comment, submitted_at

Relationships

- One **Admin** → Many **Events**
- One **Student** → Many **Registrations**
- One **Registration** → One **Attendance**
- One **Registration** → One **Feedback**

---

4. API Design

Admin APIs

- **POST /events/** → Create an event
- **GET /events/** → List events with registrations count
- **GET /events/{id}/registrations/** → View students registered
- **POST /attendance/** → Mark attendance
- **GET /reports/event-popularity/** → Events sorted by registrations
- **GET /reports/student-participation/** → Count of events attended per student

Student APIs

- **GET /events/** → Browse events
- **POST /events/{id}/register/** → Register for event
- **GET /my-registrations/** → View student's registrations + attendance
- **POST /feedback/** → Submit feedback

---

5. Workflows

Registration Flow

1. Student browses events.
2. Student registers for an event → registration record created.
3. Registration prevents duplicates for same student + event.

Attendance Flow

1. On event day, Admin marks attendance for each registration.
2. Attendance is stored against that registration.

Feedback Flow

1. After attending, student submits feedback tied to their registration.
2. Rating is stored and used in reports.

Reporting Flow

1. Admin requests reports via API.
2. Backend queries DB for registrations, attendance, and feedback.
3. Results are returned in JSON or displayed in Django Admin.

---

6. Reports (MVP)

- **Event Popularity Report** → Registrations per event, sorted.
- **Student Participation Report** → Number of events attended by each student.
- **Feedback Summary** → Average rating per event.
- **Top Students Report** → Top 3 students with highest attendance.

---

7. Edge Cases

- Prevent duplicate registrations (unique constraint on student + event).
- Attendance only possible for registered students.
- Cancelled events should block new registrations.
- Feedback ratings must be between 1–5.
- If no feedback is submitted, event average should be shown as "No feedback yet".

---

8. Implementation Notes

- Used Django Admin for quick management (instead of building a separate admin UI).
- Used React only for the student-facing features (events list, registrations, feedback).
- Seed data (fixtures/seed.json) is provided for quick demo.
- Automatic Swagger docs available at /api/docs.