Design Document

Data to Track

The system must capture and maintain the following information:

1. Event details: title, type, start/end time, venue, capacity, and status = active or cancelled
2. Student details: name, email, roll number, and associated college.
3. Student registrations: link between a student and an event.
4. Attendance: whether a registered student attended the event.
5. Feedback: rating 1to 5 and optional comment for each student per event.

Database Schema

Entities & Relationships

1. **College**: manages multiple students and events.
2. **Student**: can register for multiple events.
3. **Event**: belongs to a college, can have multiple registrations, attendance records, and feedback.
4. **Registration**: link between student and event.
5. **Attendance**: indicates whether the student was present in the event.

Table Structure

- **Colleges**: college_id, name, domain
- **Students**: student_id, college_id, name, email, roll_no
- **Events**: event_id, college_id, title, type, start_time, end_time, venue, capacity, is_cancelled
- **Registrations**: reg_id, student_id, event_id, registered_at

- **Attendance**: att_id, student_id, event_id, present, marked_at
- **Feedback**: fb_id, student_id, event_id, rating, comment, submitted_at

API Design

Event APIs

- **POST /events**: Create new event
- **GET /events**: List events (filter by type, date, etc.)

Student APIs

- **POST /students**: Add or update a student

Registration APIs

- **POST /registrations**: Register student to an event

Attendance APIs

- **POST /attendance**: Mark attendance for a student

Feedback APIs

- **POST /feedback**: Submit feedback for an event

Reporting APIs

- **GET /reports/events/popularity**: Registrations per event
- **GET /reports/events/attendance**: Attendance percentage per event
- **GET /reports/events/feedback**: Average feedback score per event
- **GET /reports/students/participation**: Events attended by a student
- **GET /reports/students/top**: Top active students

Workflows

Student Registration to Reporting

1. Student browses events.
2. Student registers for an event.

3. Attendance is marked on the event day.
4. Student submits feedback after attending.
5. Reports are generated using event, registration, attendance, and feedback data.

Assumptions & Edge Cases

1. Duplicate registrations are not allowed.
2. If an event is cancelled, registration, attendance, and feedback will be blocked.
3. Attendance can only be recorded for registered students.
4. If a student does not provide feedback, reports calculate averages only from submitted ratings.
5. If an event has no attendees, attendance percentage defaults to 0.

Reporting Requirements

1. **Event Popularity Report** → Number of registrations per event.
2. **Student Participation Report** → Number of events attended by each student.
3. **Attendance Report** → Attendance percentage per event.
4. **Feedback Report** → Average rating for each event.
5. **Bonus Reports**: Top 3 most active students, event popularity filtered by event type.

Campus Event Management Platform - Design Document

Project Overview

A comprehensive solution for managing college events, student registrations, attendance tracking, and feedback collection. The system supports multiple colleges with isolated data and provides rich reporting capabilities.

2. AI-Assisted Development

AI Tools Used

Cascade (Windsurf): For code generation, refactoring, and debugging-

ChatGPT: For brainstorming and architectural decisions

Key AI Contributions

- Initial API endpoint structure generation

- Database schema design assistance

- Query optimization suggestions

- Error handling patterns

Decisions & Deviations

Followed AI Suggestions:

- RESTful API design
- Database normalization
- Error handling middleware
- Pydantic models for request/response validation

Data Model

Database Schema

~~~~
```mermaid
erDiagram
    COLLEGES ||--o{ STUDENTS : has
    COLLEGES ||--o{ EVENTS : hosts
    STUDENTS ||--o{ REGISTRATIONS : makes
    EVENTS ||--o{ REGISTRATIONS : receives
    EVENTS ||--o{ ATTENDANCE : tracks
    STUDENTS ||--o{ ATTENDANCE : has
    EVENTS ||--o{ FEEDBACK : receives
    STUDENTS ||--o{ FEEDBACK : provides

    COLLEGES {
        string college_id PK
        string name
        string domain
    }
~~~~

```
STUDENTS {
    string student_id PK
    string college_id FK
    string name
    string email
    string roll_no
    datetime created_at
}

EVENTS {
    string event_id PK
    string college_id FK
    string title
    string type
    datetime start_time
    datetime end_time
    string venue
    integer capacity
    boolean is_cancelled
    datetime created_at
}

REGISTRATIONS {
    string reg_id PK
    string college_id FK
    string event_id FK
    string student_id FK
    datetime registered_at
}

ATTENDANCE {
    string att_id PK
    string college_id FK
    string event_id FK
    string student_id FK
    boolean present
    string method
    datetime timestamp
}

FEEDBACK {
    string fb_id PK
    string college_id FK
    string event_id FK
    string student_id FK
    integer rating
    string comment
    datetime submitted_at
}
```

API Design

Authentication

- API Key in `X-API-Key` header
- College context in `X-College-ID` header

Endpoints

Events

- `GET /events` - List all events
- `POST /events` - Create new event
- `GET /events/{event_id}` - Get event details
- `GET /events/{event_id}/registrations` - List event registrations
- `GET /events/{event_id}/attendance` - Get attendance report

- `GET /events/{event_id}/feedback` - Get event feedback

Students

- `POST /students` - Register new student
- `GET /students/{student_id}/events` - Get student's registered events
- `GET /students/{student_id}/attendance` - Get student's attendance

Registrations

- `POST /registrations` - Register student for event
- `DELETE /registrations/{reg_id}` - Cancel registration

Reports

- `GET /reports/event-popularity` - Most popular events
- `GET /reports/attendance-summary/{event_id}` - Event attendance
- `GET /reports/student-participation` - Student participation
- `GET /reports/top-active-students` - Most active students

5. Workflows

Event Registration Flow

```mermaid
sequenceDiagram
    participant Student
    participant API
    participant DB

    Student->>API: POST /api/v1/registrations
    API->>DB: Check event capacity
    API->>DB: Check duplicate registration
    API->>DB: Create registration
    API-->>Student: Registration confirmed
```

Attendance Marking Flow

```
mermaid
sequenceDiagram
    participant Staff
    participant API
    participant DB

    Staff->>API: POST /api/v1/attendance
    API->>DB: Verify registration
    API->>DB: Mark attendance
    API-->>Staff: Attendance recorded
```

6. Assumptions & Edge Cases

Assumptions

1. Each college is an independent tenant
2. Event IDs are unique within a college
3. Students can only register for events at their college
4. Feedback is anonymous but tied to registration
5. System handles up to 50 colleges × 500 students × 20 events

Edge Cases Handled

1. Duplicate registrations
2. Event capacity limits
3. Time conflicts (future enhancement)
4. Cancelled events
5. Missing feedback
6. Invalid date ranges
7. Concurrent registrations

7. Scalability Considerations

Data Partitioning

- Data is partitioned by `college_id`
- Indexes on all foreign keys
- Composite indexes for common query patterns

Performance Optimizations

- Denormalized counts for reports
- Pagination for large result sets
- Caching for frequently accessed data

8. Future Enhancements

1. User authentication and authorization
2. Email notifications
3. Bulk operations
4. Advanced reporting dashboard
5. Mobile app integration
6. Real-time updates using WebSockets

Setup

```
bash
# Install dependencies
pip install -r requirements.txt

# Initialize database
python -c "from database import init_db; init_db()"

# Seed sample data
python seed.py

# Run server
uvicorn main:app –reload
```

Data Model

1. Each college is completely independent
2. Event IDs are unique within a college but not globally
3. Students can only register for events at their own college
4. Feedback is anonymous but can be linked to registration for verification

Security

1. All endpoints require `X-College-ID` header for multi-tenancy
2. No authentication/authorization implemented (would be required in production)
3. Rate limiting not implemented (would be required in production)

Scalability

1. Designed to handle ~50 colleges
2. Each college has ~500 students
3. ~20 events per semester per college
4. SQLite is sufficient for this scale, but could be migrated to PostgreSQL if needed

Edge Cases Handled

1. Duplicate registrations (prevented by unique constraint)
2. Event capacity limits (enforced at registration)
3. Time conflicts (not currently enforced)
4. Cancelled events (marked but not automatically handled in all queries)

Future Enhancements

1. Add user authentication and authorization
2. Implement event search and filtering
3. Add email notifications
4. Support bulk operations (import/export)
5. Add more comprehensive reporting
6. Implement API versioning
7. Add request validation and error handling
8. Add API documentation (OpenAPI/Swagger)