Campus Event Platform - System Design Document

Project: Campus Event Management System

1. Executive Summary

The Campus Event Platform is a complete web application that aims to centralize event discovery and registration for university students. The system offers a simple and responsive interface for browsing campus events. It also includes strong backend features for managing registrations, capacity limits, and support for multiple campuses.

Key Objectives

- Simplify event discovery across several university campuses
- Provide a smooth registration experience with real-time feedback
- Ensure a scalable structure that can support future improvements
- Maintain a responsive design that works on all device types

2. System Architecture Overview

2.1 Architectural Pattern : The application uses a client-server layout with a clear separation of tasks:

- Frontend (Client): Single-page application for user interface and experience
- Backend (Server): RESTful API that offers business logic and data access
- Database: Persistent storage for data with relational integrity

2.2 Technology Stack : Layer Technology Purpose Justification

- Presentation HTML5, CSS3, Bootstrap 5 User Interface Modern responsive design, mobile-first approach
- Client Logic Vanilla JavaScript Frontend Behaviour No framework overhead, direct DOM manipulation
- API Layer Flask (Python) REST API Server Lightweight, flexible, rapid development
- Data Layer SQLite Database Storage Zero-configuration, ideal for prototypes
- Styling Bootstrap Icons Visual Elements Consistent icons

2.3 Data Flow Architecture

[Browser] to [JavaScript Client] to [Flask API] to [SQLite Database]

Example Registration Flow:

- User clicks "Register Now" → JavaScript captures the event
- Client sends POST request → /api/v1/colleges/{college_id}/events/{event_id}/register
- Flask checks the request → Validates the database constraints
- Database operations → Insert registration record

- Success response → Client updates UI dynamically

3. Database Design

3.1 Entity Relationship Model :The database schema includes four main entities with these relationships:

Colleges (1) → (N) Events (One-to-Many)

Colleges (1) → (N) Students (One-to-Many)

Events (N) ← → (N) Students via Registrations (Many-to-Many)

3.2 Table Specifications

3.2.1 Colleges Table

sql script

```
CREATE TABLE colleges (
    id TEXT PRIMARY KEY,          -- Unique identifier (e.g., 'MIT', 'CMU')
    name TEXT NOT NULL,           -- Full institution name
    location TEXT NOT NULL,       -- Geographic location
    contact_email TEXT NOT NULL   -- Administrative contact
);
```

3.2.2 Events Table

sql script

```
CREATE TABLE events (
    id TEXT PRIMARY KEY,
    college_id TEXT NOT NULL,
    title TEXT NOT NULL,          -- Event name
    description TEXT,
    event_type TEXT,
    start_datetime TEXT,
```

```
    end_datetime TEXT,
    location TEXT,              -- Venue information
    capacity INTEGER,
    status TEXT DEFAULT 'active',
    FOREIGN KEY (college_id) REFERENCES colleges (id)
);
```

3.2.3 Students Table

sql script

```
CREATE TABLE students (
    id TEXT PRIMARY KEY,
    college_id TEXT NOT NULL,
    email TEXT UNIQUE NOT NULL,    -- Student email (unique)
    name TEXT NOT NULL,
    department TEXT,
    FOREIGN KEY (college_id) REFERENCES colleges (id)
);
```

3.2.4 Registrations Table

sql script

```
CREATE TABLE registrations (
    id TEXT PRIMARY KEY,          -- Registration unique ID
    event_id TEXT NOT NULL,       -- Foreign key to events
    student_id TEXT NOT NULL,     -- Foreign key to students
    registration_date TIMESTAMP,  -- When registration occurred
    status TEXT DEFAULT 'registered', -- Registration status
    FOREIGN KEY (event_id) REFERENCES events (id),
    FOREIGN KEY (student_id) REFERENCES students (id),
```

```
    UNIQUE(event_id, student_id)   -- Prevent duplicate registrations
);
```

3.3 Data Integrity Constraints

- Primary Keys: Ensure unique identification of all records
- Foreign Keys: Maintain referential integrity between related tables
- Unique Constraints: Prevent duplicate registrations and emails
- Not Null Constraints: Ensure required fields are filled in
- Default Values: Provide sensible defaults for status fields

4. API Design Specification

4.1 API Architecture Principles

- RESTful Design: Uses standard HTTP methods and status codes
- JSON Communication: All requests and responses are in JSON format
- Stateless Operations: Each request contains all necessary information
- Error Handling: Consistent error response format across all endpoints

4.2 Base Configuration

- Base URL: /api/v1
- Content-Type: application/json
- Authentication: Not implemented in current version (future improvement)

4.3 Endpoint Specifications

4.3.1 Get College Events

GET /api/v1/colleges/{college_id}/events

Purpose: Retrieve all active events for a specific college

Response Format:

json data file

```
[
  {
    "id": "MIT-2025-001",
    "college_id": "MIT",
    "title": "Intro to AI Workshop",
    "description": "A beginner-friendly workshop on Artificial Intelligence.",
    "event_type": "workshop",
    "start_datetime": "2025-10-15T10:00:00",
    "end_datetime": "2025-10-15T13:00:00",
    "location": "Room 404",
    "capacity": 50,
    "registration_count": 12,
    "avg_rating": 4.5
  }
]
```

Status Codes:

- 200 OK: Successfully retrieved events
- 404 Not Found: College not found

4.3.2 Register for Event

http

```
POST /api/v1/colleges/{college_id}/events/{event_id}/register
Content-Type: application/json

{
  "student_id": "MIT-STU-001"
}
```

Purpose: Register a student for a specific event

Success Response:

json

```
{
  "message": "Successfully registered for the event!",
  "registration_id": "REG-MIT-2025-001-STU-001"
}
```

Status Codes:

- 201 Created: Registration successful
- 400 Bad Request: Invalid request data
- 404 Not Found: Event or student not found
- 409 Conflict: Event full or student already registered

5. Frontend Architecture

5.1 Component Structure : The frontend follows a modular JavaScript structure with clear separation of tasks:

5.1.1 Core Components

- Event Display Module: Manages event grid rendering and filtering
- Modal Management: Controls event detail popups and registration forms
- API Communication: Handles all interactions with the server
- UI State Management: Manages loading states, alerts, and user feedback

5.1.2 Key Functions

javascript

```
fetchEvents()
renderEventCard()      // Generates individual event display cards
showEventModal()       // Opens detailed event information
```

handleRegistration() // Processes user registration requests

filterAndRender() // Applies search/filter logic

5.2 User Interface Design

5.2.1 Layout Structure

- Navigation Header: College selector and branding
- Hero Section: Dynamic title and subtitle based on selected college
- Control Panel: Search bar and event type filters
- Event Grid: Responsive card layout using Bootstrap's grid system
- Modal Overlays: Detailed event information and registration forms

5.2.2 Responsive Design

- Desktop: 3-column event grid with full feature set
- Tablet: 2-column layout with touch-optimized interactions
- Mobile: Single column with collapsible controls

6. Security Considerations

6.1 Current Implementation

- Input Validation: Basic server-side validation of registration data
- SQL Injection Prevention: Parameterized queries throughout
- CORS Handling: Proper cross-origin request configuration

6.2 Future Security Enhancements

- Authentication System: Student login with session management
- Authorization Levels: Controls for admin and student permissions
- Rate Limiting: Prevent registration spam and API misuse
- Data Encryption: Secure sensitive student information
- HTTPS Enforcement: Secure data transmission

7. Performance Considerations

7.1 Current Optimizations

- Efficient Queries: Optimized database queries with proper indexing
- Client-Side Filtering: Reduces server load for search operations
- Minimal Dependencies: Lightweight technology stack
- Responsive Loading: Progressive loading indicators

7.2 Scalability Planning

- Database Migration Path: SQLite to PostgreSQL for production
- Caching Strategy: Event data caching for high-traffic times
- CDN Integration: Optimizes delivery of static assets
- Load Balancing: Support for multiple server instances

8. Deployment Architecture

8.1 Development Environment

- Local Development: Flask development server with hot reload
- Database: Single-file SQLite for easy development
- Testing: Integrated test suite with mock data

8.2 Production Recommendations

- Web Server: Gunicorn with Nginx reverse proxy
- Database: PostgreSQL with connection pooling
- Monitoring: Application performance monitoring (APM)
- Backup Strategy: Automated database backups and recovery

9. Future Enhancement Roadmap

9.1 Phase 2 Features

- User Authentication: Complete student login system
- Event Management Dashboard: Admin interface for event creation
- Email Notifications: Confirmations and reminders for registrations
- Calendar Integration: Export to personal calendar applications

9.2 Phase 3 Features

- Mobile Application: Native iOS/Android apps
- Advanced Analytics: Tracking event popularity and attendance
- Social Features: Event comments and peer recommendations
- Integration APIs: Connect with existing campus management systems

10. Conclusion

The Campus Event Platform offers a strong foundation for managing university events. Its well-structured layout supports current needs and future growth. The separation between frontend and backend, along with a thoughtfully designed database, creates a solution that is easy to maintain and scale. The chosen technologies prioritize quick development and deployment while allowing for evolution into a more complex system as needs expand. The RESTful API design makes it easy to integrate additional client applications, like mobile apps and administrative dashboards, in future phases.