**Campus Event Management Platform  
  
Design Document  
  
*Submitted By*Name : Darshan Ramesh  
Degree : B.Tech in Information Science and Engineering  
University : REVA University  
SRN : R22EQ064  
  
Date: 7th September 2025**

**Project: Campus Drive Assignment – Webknot Technologies**

**1. Introduction / Scope**

The Campus Event Management Platform is designed to simplify how colleges organize and manage events while improving the student experience. The system consists of two main components:

- Admin Portal (Web): Used by faculty and staff to create and manage events such as hackathons, workshops, seminars, and cultural fests.

- Student App (Mobile): Enables students to browse upcoming events, register, mark attendance, and provide feedback after participation.

The primary goals of this platform are:

- To streamline the process of event creation and student registrations.

- To ensure accurate attendance tracking and feedback collection.

- To generate meaningful reports on event popularity, student participation, and overall engagement.

The scope of this design document is to provide a clear and minimalistic blueprint for the system, covering the following aspects:

1. The type of data to be tracked.

2. The database schema and relationships.

3. API endpoints to support event lifecycle operations.

4. Workflows describing registration, attendance, and reporting.

5. Edge cases and scalability considerations for real-world use.

This design focuses on a prototype-level implementation but ensures that the solution can scale to multiple colleges and thousands of students in the future.

**2. Assumptions**

Unique Identifiers:  
- Event IDs are globally unique across all colleges.  
- Student IDs are unique within each college.

Student Registration:  
- A student can register for multiple events but only once per event.  
- Registration is required before attendance can be marked.

Attendance Tracking:  
- Attendance can only be marked on the event day by authorized staff.  
- Only registered students can have their attendance recorded.

Feedback Collection:  
- Feedback (rating 1–5) is optional.  
- Only students who attended the event can submit feedback.

System Scale:  
- The prototype is designed for ~50 colleges, ~500 students each, and ~20 events per semester.  
- SQLite is used for the prototype; the system can be migrated to PostgreSQL/MySQL for production.

Data Integrity:  
- Duplicate registrations and invalid feedback are prevented using database constraints.  
- Cancelled events will automatically invalidate their associated registrations, attendance, and feedback

3. Data to Track

1. Colleges

• - college\_id (unique identifier)

• - name (college name)

2. Students

• - student\_id (unique identifier)

• - name

• - roll\_no

• - email

• - college\_id (foreign key → Colleges)

3. Events

• - event\_id (unique identifier)

• - title

• - type (Workshop, Fest, Seminar, etc.)

• - date

• - college\_id (foreign key → Colleges)

4. Registrations

• - registration\_id (unique identifier)

• - student\_id (foreign key → Students)

• - event\_id (foreign key → Events)

• - timestamp (time of registration)

5. Attendance

• - attendance\_id (unique identifier)

• - student\_id (foreign key → Students)

• - event\_id (foreign key → Events)

• - status (Present/Absent)

6. Feedback

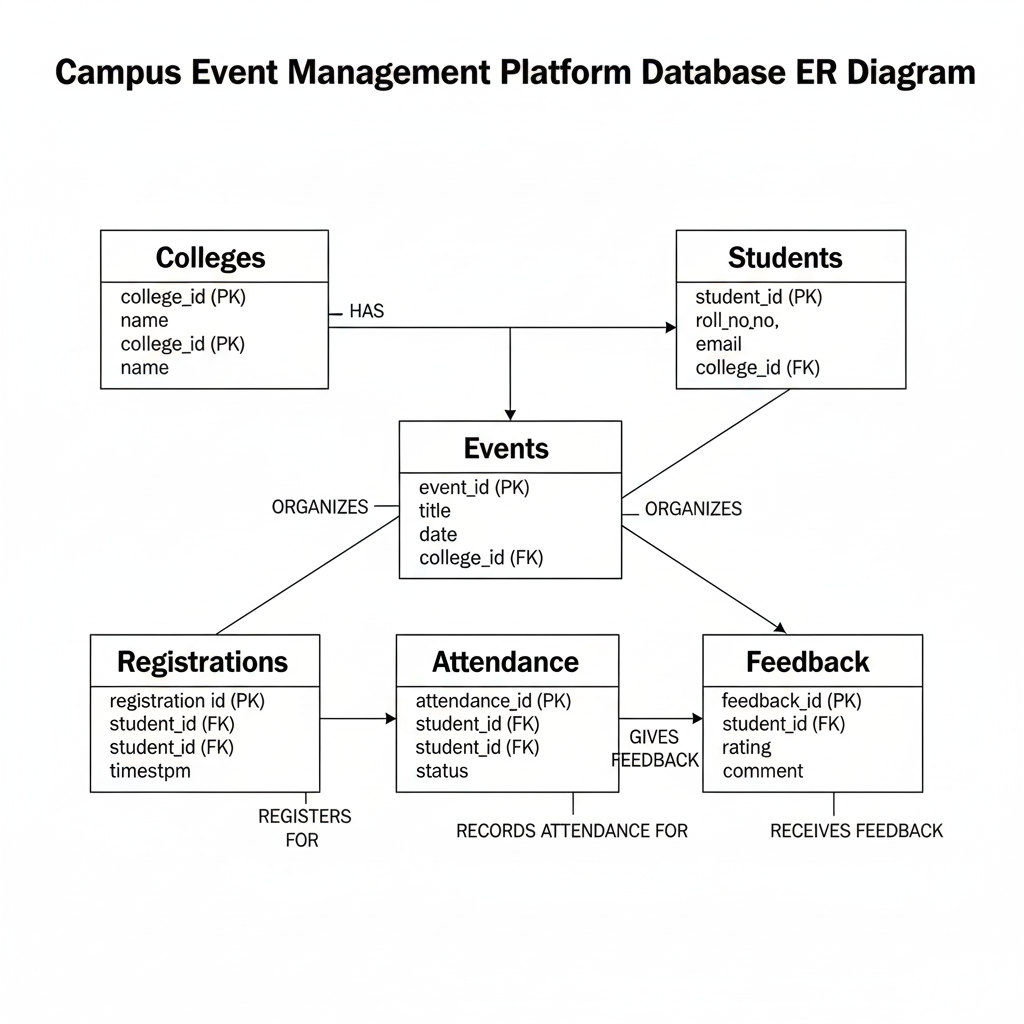
• - feedback\_id (unique identifier)

• - student\_id (foreign key → Students)

• - event\_id (foreign key → Events)

• - rating (1–5)

• - comment (optional text)

4. ER Diagram   
  


5. **API Design Document**

1. Introduction This document outlines the API design for the Campus Event Management Platform, detailing the communication interface between the front-end applications (Admin Portal and Student App) and the back-end system. The API is a RESTful web service, following standard HTTP methods and conventions for resource-based interactions. The design focuses on providing clear, intuitive, and scalable endpoints for all core functionalities, from event management to data reporting.

2. API Endpoints The API endpoints are organized by function. All endpoints will use the base URL /api/v1/ to ensure versioning and a consistent structure. Each endpoint is described below with its HTTP method, URL, and a brief description.

| Endpoint | HTTP Method | Description |
| --- | --- | --- |
| /api/v1/events | POST | Creates a new event. |
| /api/v1/events/{eventId} | GET | Retrieves details for a specific event. |
| /api/v1/events/{eventId}/register | POST | Registers a student for an event. |
| /api/v1/events/{eventId}/attendance | POST | Marks a student's attendance at an event. |
| /api/v1/events/{eventId}/feedback | POST | Submits a student's feedback for an event. |
| /api/v1/reports/popularity | GET | Generates a report on event popularity by registrations. |
| /api/v1/reports/attendance\_rate | GET | Calculates attendance percentage for all events. |
| /api/v1/reports/feedback | GET | Generates a report on average feedback scores. |
| /api/v1/reports/participation/{studentId} | GET | Lists events a specific student has attended. |
| /api/v1/reports/top-students | GET | Identifies and lists the top 3 most active students. |

3. Request and Response Schemas This section details the expected JSON format for requests and the corresponding responses for each endpoint.

3.1. Create a New Event

* Endpoint: POST /api/v1/events
* Purpose: Allows college staff to create a new event.
* Request Body (JSON):

JSON

{

"title": "Data Science Workshop",

"type": "Workshop",

"date": "2025-10-20",

"description": "An introduction to data science.",

"college\_id": "C-REVA-1"

}

* Success Response (HTTP 201 Created):

JSON

{

"message": "Event created successfully.",

"event\_id": "E-DS-20251020"

}

3.2. Register a Student

* Endpoint: POST /api/v1/events/{eventId}/register
* Purpose: Enables students to register for an event.
* Request Body (JSON):

JSON

{

"student\_id": "S-REVA-22EQ064"

}

* Success Response (HTTP 200 OK):

JSON

{

"message": "Student registered successfully for the event."

}

3.3. Mark Attendance

* Endpoint: POST /api/v1/events/{eventId}/attendance
* Purpose: Records a student's attendance.
* Request Body (JSON):

JSON

{

"student\_id": "S-REVA-22EQ064",

"status": "Present"

}

* Success Response (HTTP 200 OK):

JSON

{

"message": "Attendance marked."

}

3.4. Submit Feedback

* Endpoint: POST /api/v1/events/{eventId}/feedback
* Purpose: Collects feedback and a rating from a student.
* Request Body (JSON):

JSON

{

"student\_id": "S-REVA-22EQ064",

"rating": 5,

"comment": "The workshop was very informative!"

}

* Success Response (HTTP 200 OK):

JSON

{

"message": "Feedback submitted successfully."

}

3.5. Reporting

Reporting endpoints return a JSON array containing the requested data.

* Endpoint: GET /api/v1/reports/popularity
* Purpose: Displays the most popular events by registration count.
* Success Response (HTTP 200 OK):

JSON

{

"report": [

{ "event\_id": "E-DS-20251020", "title": "Data Science Workshop", "registrations": 150 },

{ "event\_id": "E-FEST-20251105", "title": "Annual College Fest", "registrations": 450 }

]

}

* Endpoint: GET /api/v1/reports/participation/{studentId}
* Purpose: Lists all events a specific student attended.
* Success Response (HTTP 200 OK):

JSON

{

"student\_id": "S-REVA-22EQ064",

"attended\_events": [

{ "event\_id": "E-DS-20251020", "title": "Data Science Workshop", "attendance\_date": "2025-10-20" },

{ "event\_id": "E-FEST-20251105", "title": "Annual College Fest", "attendance\_date": "2025-11-05" }

]

}

4. Error Handling The API will return standard HTTP status codes to communicate success or failure.

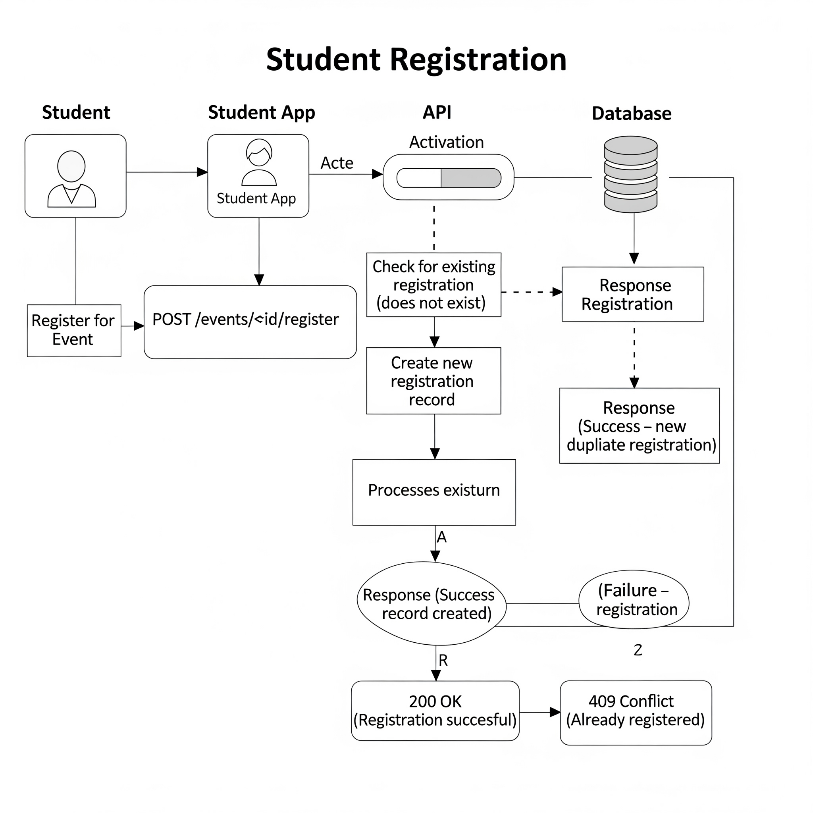
* 400 Bad Request: Malformed request or missing required fields.
* 401 Unauthorized: Authentication failed.
* 404 Not Found: The requested resource does not exist (e.g., an invalid eventId).
* 409 Conflict: An attempt to create a duplicate resource, such as a student trying to register twice for the same event.
* 500 Internal Server Error: An unexpected server-side issue.

6. **Workflows & Sequence Diagrams**1. Student Registration Workflow

This workflow details the process a student goes through to register for an event.

* Goal: To successfully record a student's registration in the database while preventing duplicates.
* Actors: Student, Student App, API, Database.

Sequence of Events:

1. Student browses events on the Student App.
2. The Student App sends a POST /events/{id}/register request to the API with the student's ID and event ID.
3. The API receives the request and validates the data. It checks the Database to ensure the student has not already registered for this event.
4. If the registration is unique, the API adds a new record to the Registrations table in the Database.
5. The Database confirms the action.
6. The API sends a 200 OK success response back to the Student App.
7. The Student App displays a confirmation message to the Student.
8. If the registration is a duplicate, the API sends a 409 Conflict error response, and the Student App displays an error message.

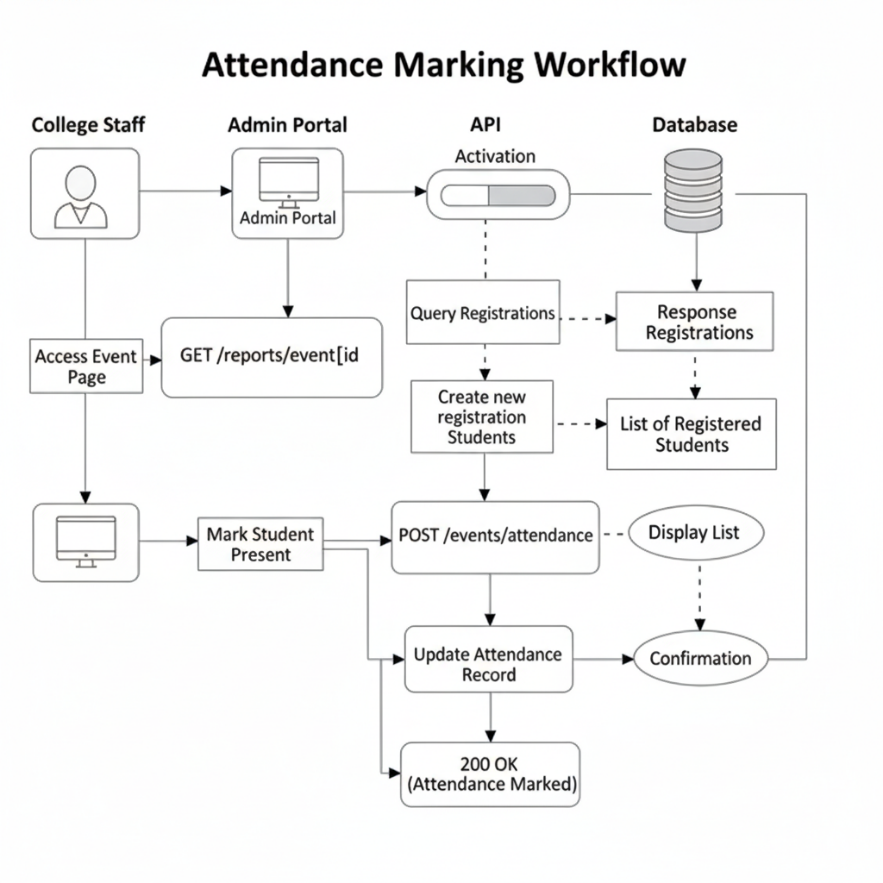
2. Attendance Marking Workflow

This workflow details how authorized staff can record student attendance.

* Goal: To accurately mark attendance for registered students on the day of an event.
* Actors: College Staff, Admin Portal, API, Database.

Sequence of Events:

1. College Staff accesses the event management page on the Admin Portal.
2. The Admin Portal displays a list of registered students for that event by querying the API with GET /reports/event/{id}.
3. On event day, the College Staff marks a student as Present.
4. The Admin Portal sends a POST /events/{id}/attendance request to the API with the student's ID.
5. The API receives the request, validates the data, and checks if the student is already registered.
6. The API updates the Attendance table in the Database, setting the status to 'Present'.
7. The Database confirms the update.
8. The API sends a 200 OK success response to the Admin Portal.

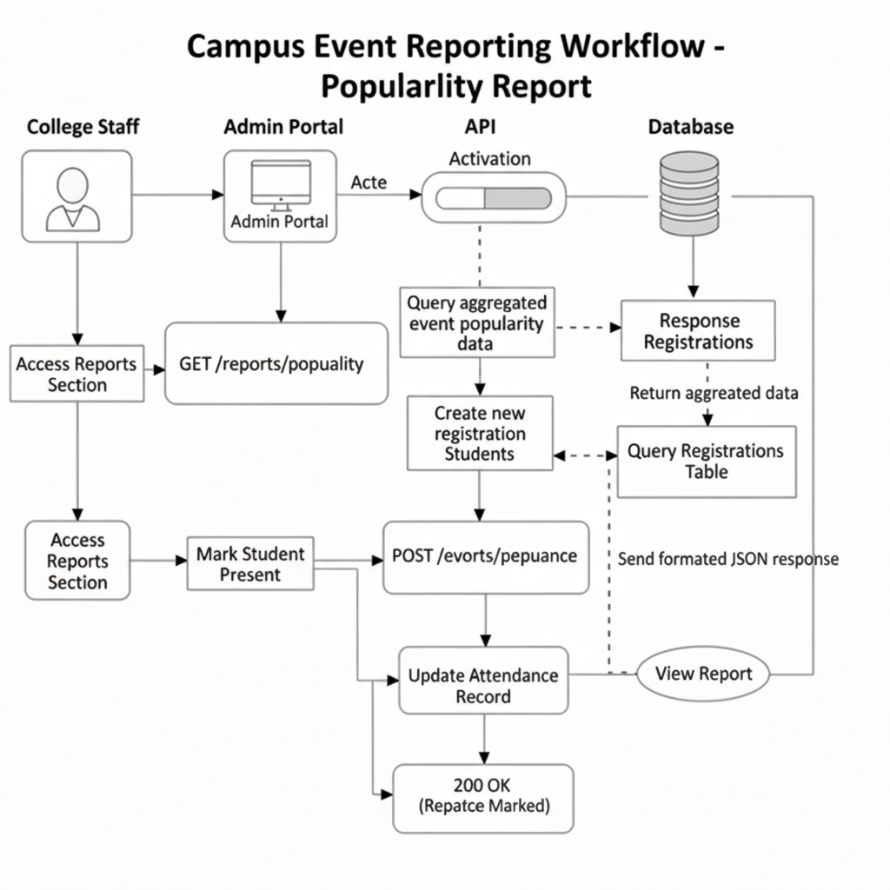


3. Reporting Workflow

This workflow describes how the system generates a report on event popularity.

* Goal: To provide the college staff with a report on event popularity, sorted by the number of registrations.
* Actors: College Staff, Admin Portal, API, Database.

Sequence of Events:

1. College Staff navigates to the reporting section of the Admin Portal.
2. The Admin Portal sends a GET /reports/popularity request to the API.
3. The API receives the request and executes a query on the Registrations table in the Database to count registrations per event.
4. The Database returns the aggregated data.
5. The API formats the data into a JSON response.
6. The API sends the response back to the Admin Portal.
7. The Admin Portal presents the data in a user-friendly format (e.g., a table or chart) to the College Staff.

**7. Assumptions & Edge Cases**1. Assumptions

These are the foundational beliefs and constraints we have considered for the initial design of the platform.

* System Scale: The prototype is built to handle a moderate scale, assuming approximately 50 colleges, each with 500 students and hosting around 20 events per semester. This scale justifies the use of a lightweight database like SQLite.
* Unique Identifiers: We assume that Event IDs must be unique across all colleges to avoid conflicts, while Student IDs are unique only within a single college.
* Data Structure: We will maintain one unified, central dataset for all colleges. This approach simplifies cross-college reporting and data management compared to having separate datasets for each college.
* Database Scalability: The prototype will use a lightweight database like SQLite. We assume that for a production environment, the system would be migrated to a more robust, scalable solution like PostgreSQL or MySQL.

2. Edge Cases & Handling

This section identifies potential issues or "edge cases" and details the system's planned response to maintain data integrity and a smooth user experience.

* Duplicate Registration: A student attempts to register for the same event more than once.
  + Handling: The API will prevent this by checking for an existing registration entry with the same student\_id and event\_id combination. If found, the request will be rejected with a 409 Conflict HTTP error.
* Missing Feedback: A student who attended an event does not submit a feedback rating.
  + Handling: The rating field in the Feedback table will be designed to allow null values. Reporting queries for average feedback will be designed to only calculate the average based on entries with a valid rating, thus ignoring records without feedback.
* Attendance without Registration: An authorized staff member attempts to mark attendance for a student who never registered for the event.
  + Handling: The API will perform a validation check against the Registrations table before marking attendance. If the student is not registered, the API will reject the request.
* Canceled Events: An event is canceled after students have already registered or submitted feedback.
  + Handling: The Events table will include a status field (e.g., active, canceled). Any API call related to a canceled event will automatically fail, and the front-end will display a clear message to the user.
* Invalid IDs: An API request is made with a nonexistent or invalid student\_id or event\_id.
  + Handling: The API will follow standard RESTful practices and return a 404 Not Found error, clearly indicating that the requested resource does not exist.