**Campus Event Management Platform: Event Reporting System Design Document**

**1. Introduction**

This document outlines the design and implementation of a basic event reporting system for a Campus Event Management Platform. The platform consists of two main components: an **Admin Portal** (web-based) for college staff and a **Student App** (mobile-friendly) for student users. The primary goal of this system is to effectively track and report on event creation, student participation, and user feedback.

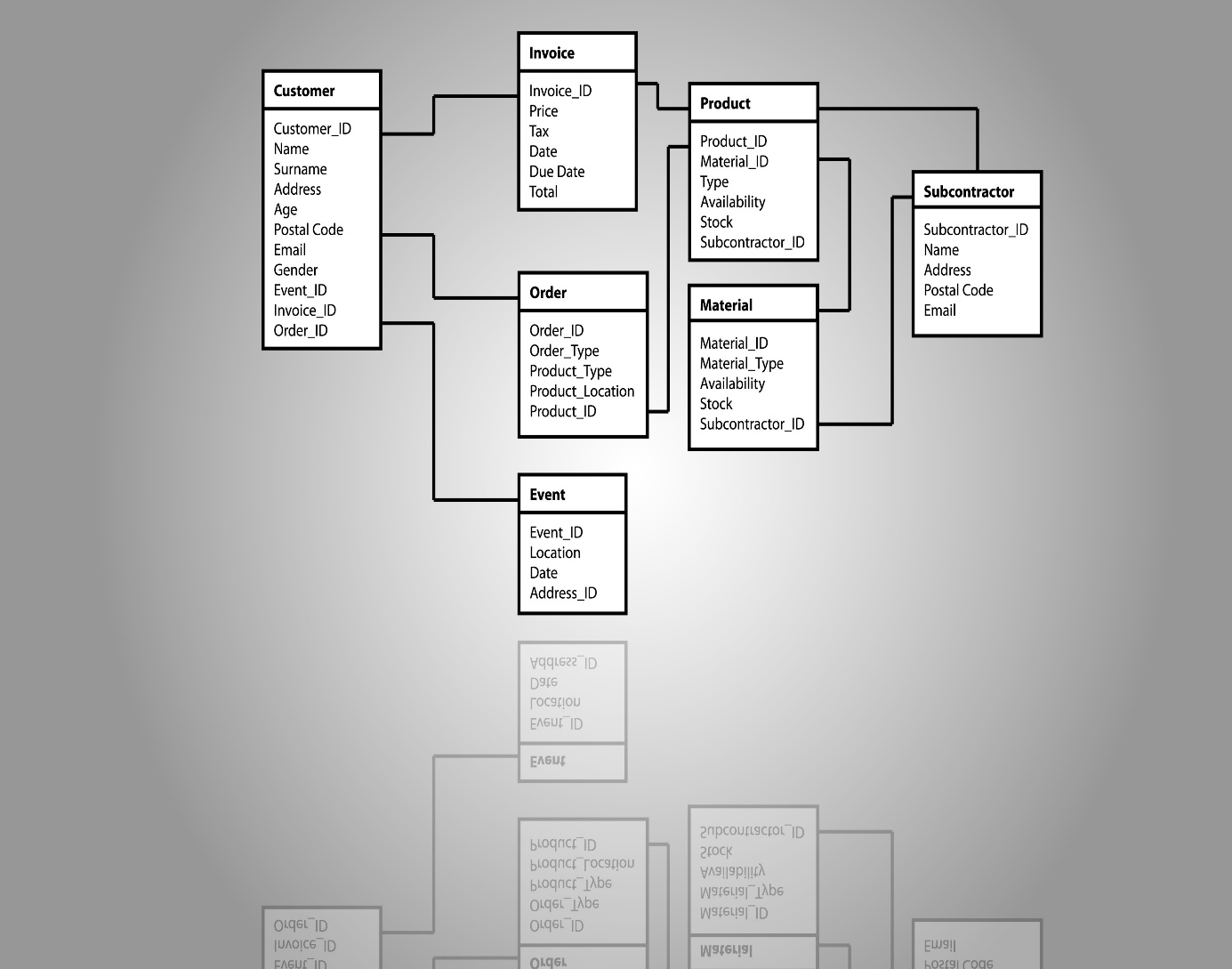
**2. Data to Track**

The system is designed to track four core types of data to provide a comprehensive view of campus events:

* **Event Creation:** Information about each event, including its name, type, date, and location.
* **Student Registration:** Records of students signing up for specific events.
* **Attendance:** Data on which registered students actually attended an event.
* **Feedback:** Ratings and comments provided by students after attending an event.

**3. Database Schema**

The system uses a relational database model with six tables to organize the data efficiently. This schema uses foreign keys to link related data and ensures data integrity.



* **college**: Stores information about each college.
  + college\_id (INT, PRIMARY KEY, AUTO\_INCREMENT)
  + name (VARCHAR(255))
  + location (VARCHAR(255))
* **student**: Stores student profiles.
  + student\_id (INT, PRIMARY KEY, AUTO\_INCREMENT)
  + college\_id (INT, FOREIGN KEY)
  + first\_name (VARCHAR(100))
  + last\_name (VARCHAR(100))
  + email (VARCHAR(255), UNIQUE per college)
* **event**: Stores details about each event.
  + event\_id (INT, PRIMARY KEY, AUTO\_INCREMENT)
  + college\_id (INT, FOREIGN KEY)
  + name (VARCHAR(255))
  + event\_type (VARCHAR(50))
  + date\_time (DATETIME)
  + location (VARCHAR(255))
* **registration**: Links a student to an event they have registered for.
  + registration\_id (INT, PRIMARY KEY, AUTO\_INCREMENT)
  + student\_id (INT, FOREIGN KEY)
  + event\_id (INT, FOREIGN KEY)
  + registered\_at (DATETIME)
  + UNIQUE constraint on (student\_id, event\_id) to prevent duplicate registrations.
* **attendance**: Tracks event attendance.
  + attendance\_id (INT, PRIMARY KEY, AUTO\_INCREMENT)
  + registration\_id (INT, FOREIGN KEY, UNIQUE)
  + marked\_at (DATETIME)
* **feedback**: Stores student feedback for an event.
  + feedback\_id (INT, PRIMARY KEY, AUTO\_INCREMENT)
  + registration\_id (INT, FOREIGN KEY, UNIQUE)
  + rating (TINYINT from 1-5)
  + comment (TEXT)
  + submitted\_at (DATETIME)

**4. API Design**

The platform uses a RESTful API to manage data, with dedicated endpoints for each user action.

**Admin Portal Endpoints**

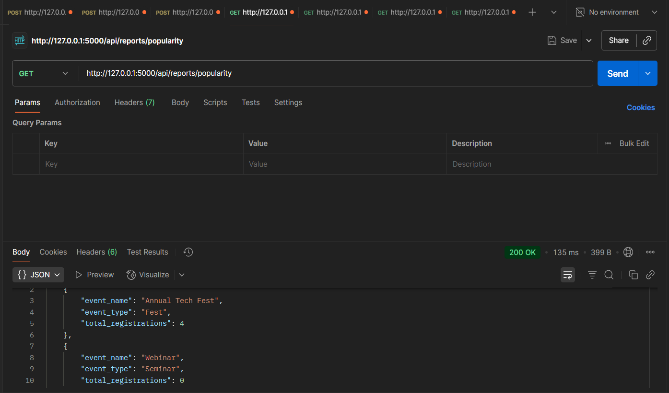
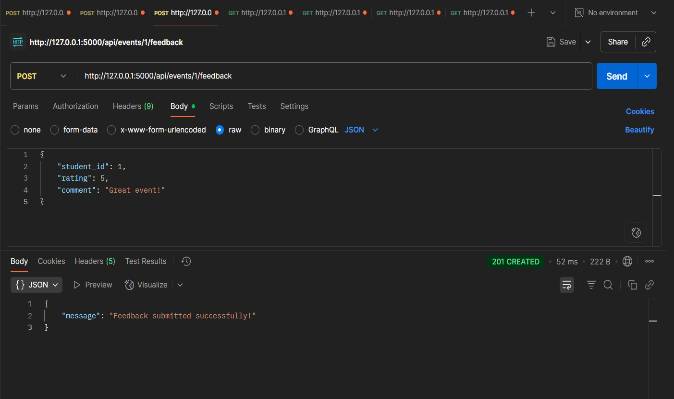
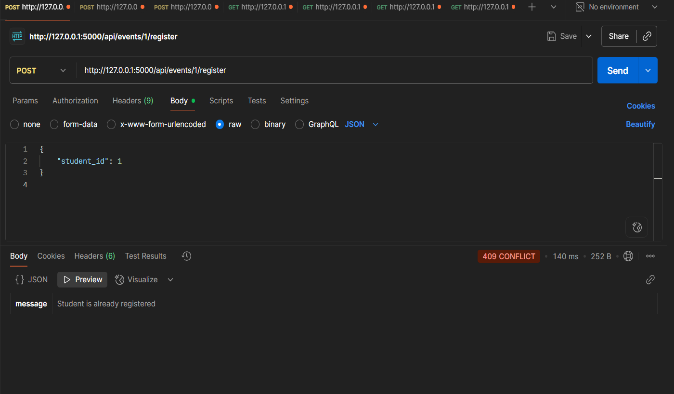
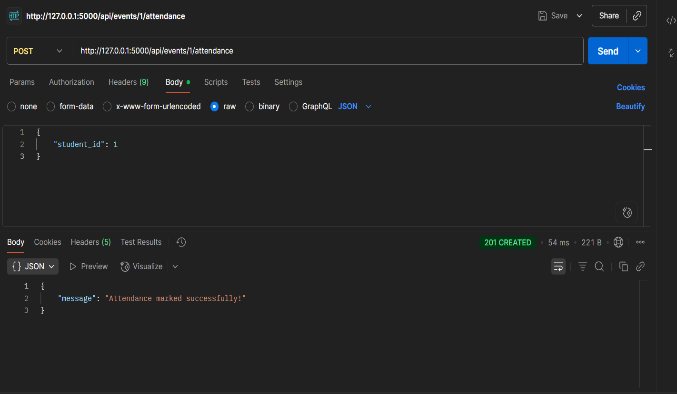
* POST /api/students: Create a new student record.
* POST /api/events: Create a new event record.
* POST /api/events/<event\_id>/attendance: Mark attendance for a student at an event.

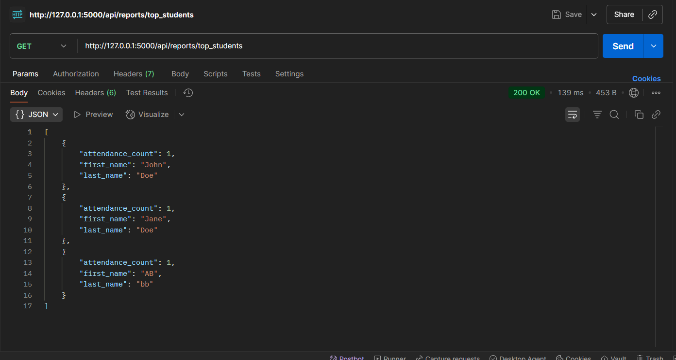
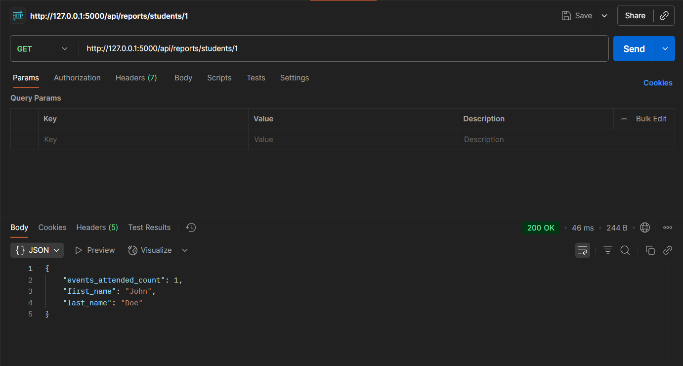
**Student App Endpoints**

* POST /api/events/<event\_id>/register: Register a student for a specific event.
* POST /api/events/<event\_id>/feedback: Submit a rating and comment for an event.

**Reporting Endpoints (Accessed by Admin)**

* GET /api/reports/popularity: Returns a list of all events sorted by the total number of registrations.
* GET /api/reports/students/<student\_id>: Shows the number of events a specific student has attended.
* GET /api/reports/top\_students: Returns a list of the top 3 students by attendance count.
* GET /api/reports/events?type=<event\_type>: Returns a list of events filtered by a specific type (e.g., Workshop, Fest).

Fig 1: API call for Registration Fig 2: API call for attendance

Fig 3: API call for Feedback Fig 4: API (GET) request for popularity

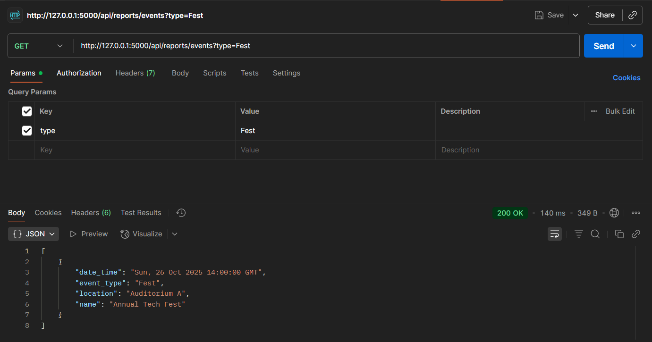
Fig 5: API (Get) request for Students Fig 6: API (GET) request for top student

Fig 7: API (GET) request for type of Even

**5. Workflows**

**Student Registration**

1. **Student Action:** The student's app sends a POST request with their student\_id to the /api/events/<event\_id>/register endpoint.
2. **Server Action:** The backend verifies that the student and event exist and checks for a duplicate registration.
3. **Database Action:** If validation passes, a new record is created in the registration table.
4. **Response:** The API returns a 201 Created status if successful or a 409 Conflict if the student is already registered.

**Attendance Marking**

1. **Admin Action:** A staff member uses the Admin Portal to send a POST request with a student\_id to the /api/events/<event\_id>/attendance endpoint.
2. **Server Action:** The backend first finds the student's registration record for the event.
3. **Database Action:** If a registration exists and attendance hasn't been marked, a new record is created in the attendance table, linked to the registration\_id.
4. **Response:** The API returns a 201 Created status for success, a 404 Not Found if the student wasn't registered, or a 409 Conflict if attendance was already marked.

**Generating Reports (Event Popularity)**

1. **Admin Action:** The admin portal sends a GET request to the /api/reports/popularity endpoint.
2. **Server Action:** The backend executes a SQL query that joins the event and registration tables. It counts the number of registrations for each event and sorts the results.
3. **Response:** The API returns a JSON array containing event names, types, and their total registration counts.

**6. Assumptions & Edge Cases**

* **Duplicate Registrations:** The system prevents duplicate registrations by using a UNIQUE constraint on the (student\_id, event\_id) combination in the registration table. The API checks for this before creating a record and returns a 409 Conflict status.
* **Missing Data:** Foreign key constraints are used to ensure that a registration, attendance, or feedback record cannot be created unless the referenced student and event already exist. The API returns a 404 Not Found or 400 Bad Request if a foreign key is invalid.
* **Missing Feedback/Attendance:** A student can register for an event but not attend or leave feedback. The system handles this by making the attendance and feedback records optional via LEFT JOINs in reports and by returning a success message without requiring attendance or feedback.

**OUTPUTS AND RESULTS**

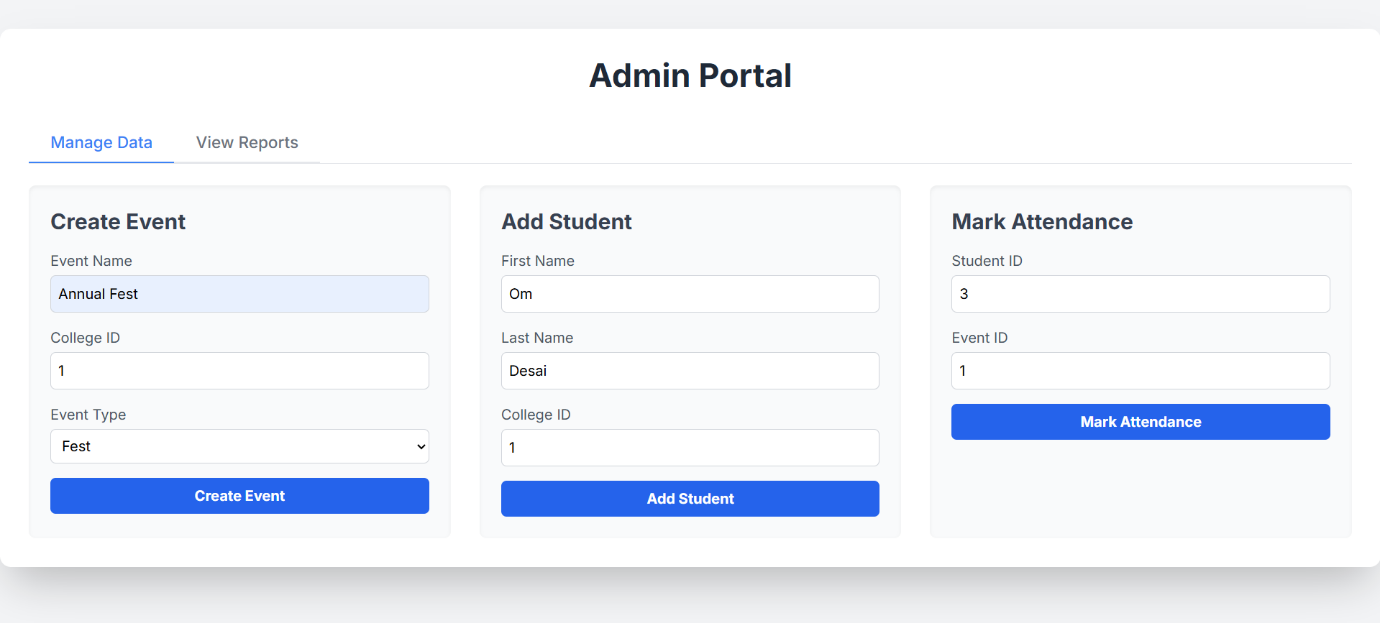


Fig: This is the Admin Portal where the college can create event mark attendance of the student and add a student for and event.

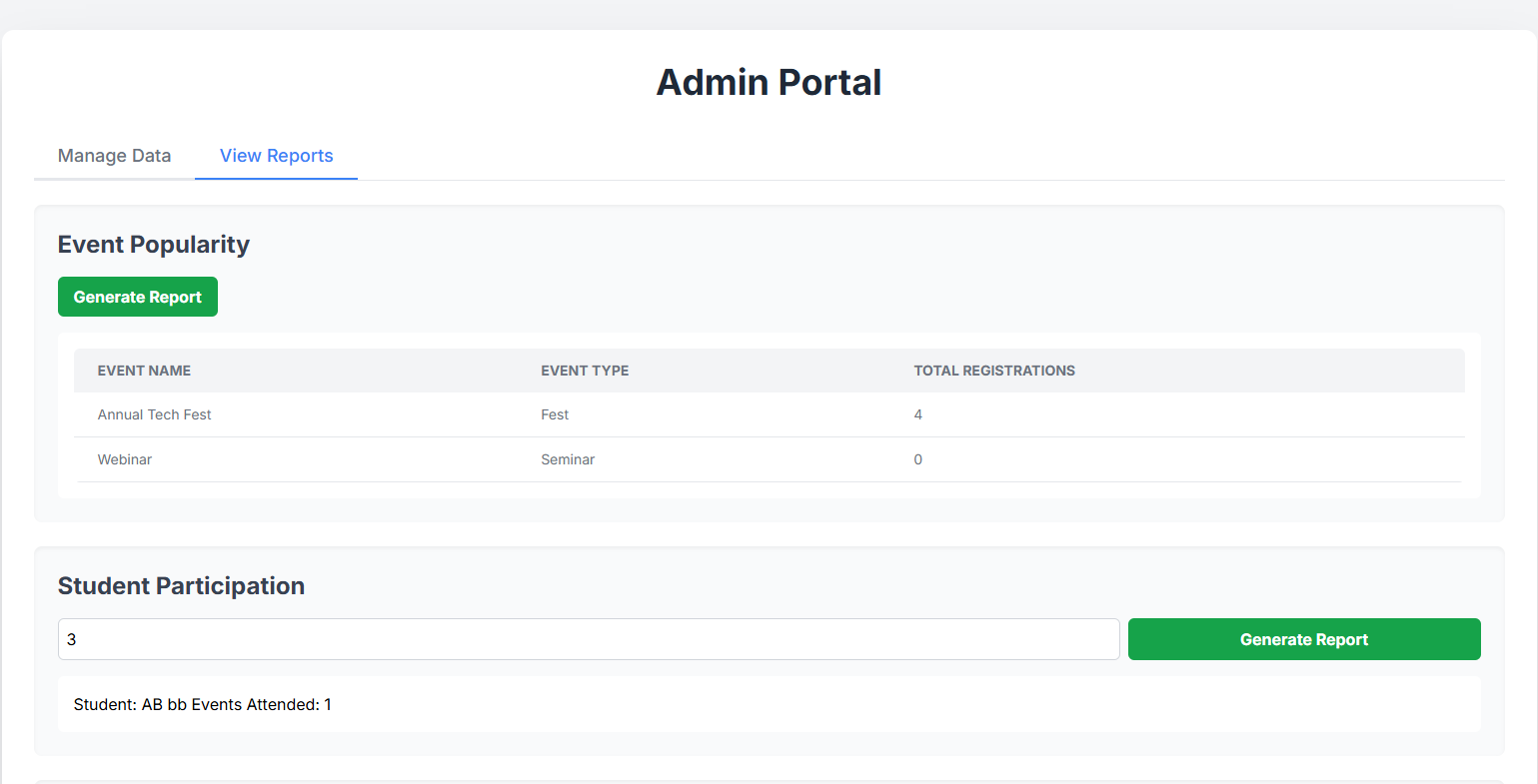


Fig: Also, the admin can view the report as Event Popularity, Student participation by Student ID

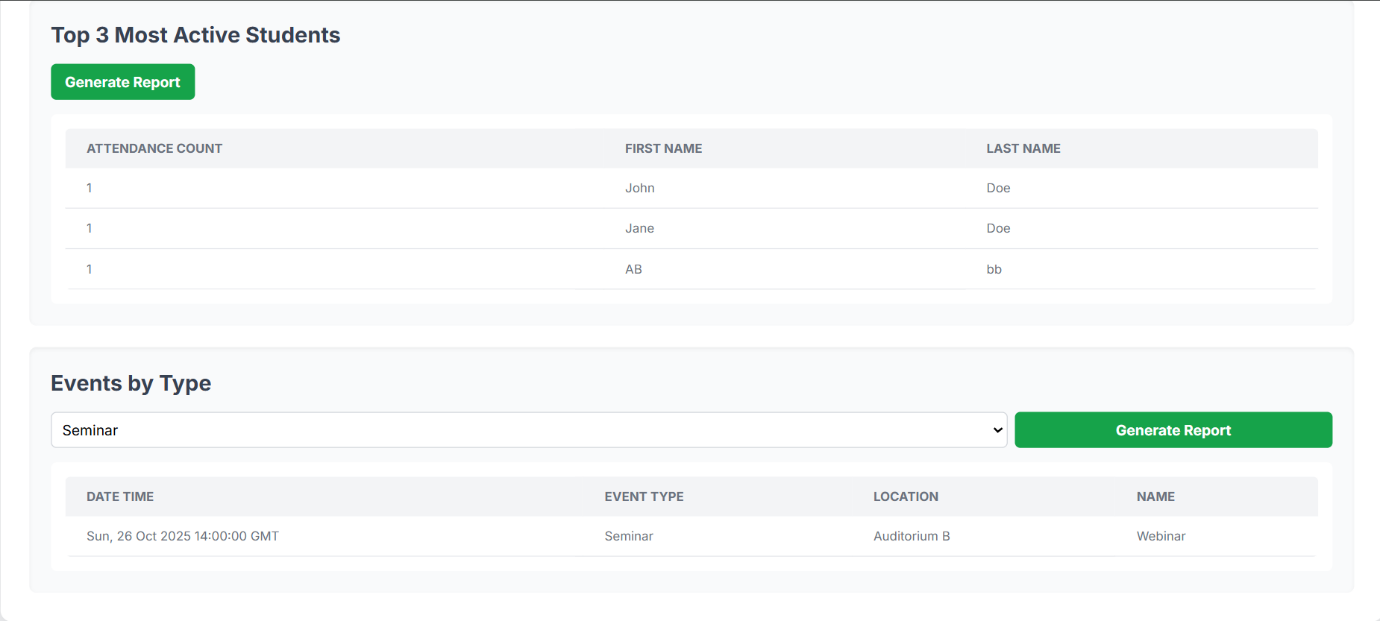
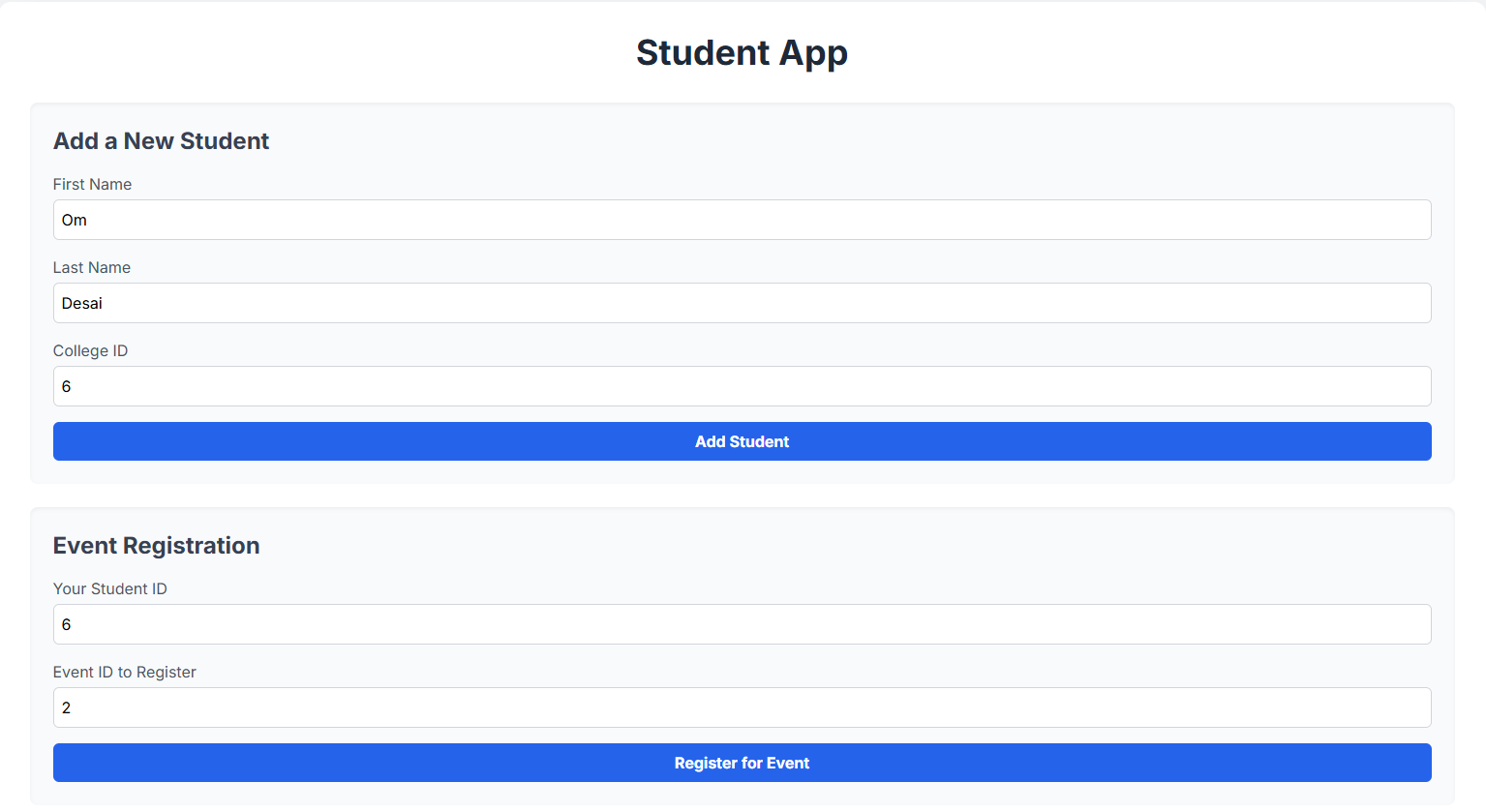
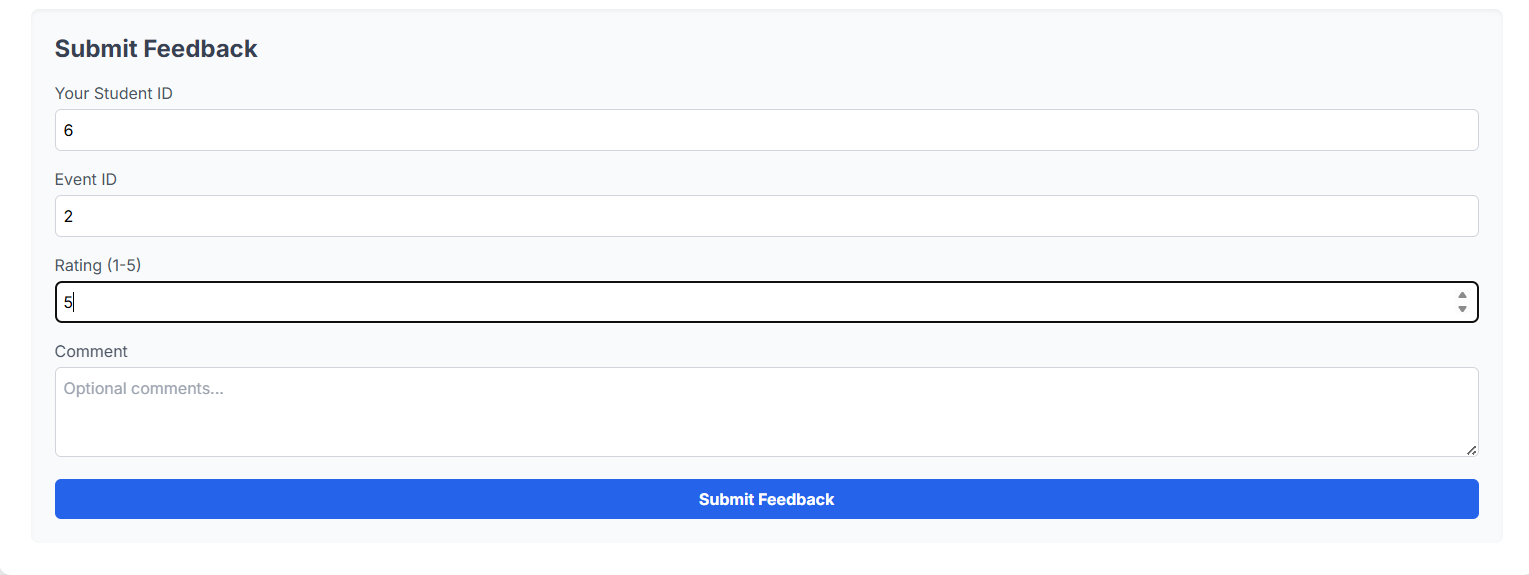


Fig: In the report Section it shows the top 2 active students and also the Events by type like Seminar, Fest, Workshop.

Fig: This is the student dashboard which helps student to register for and event by entering their name and college id also the student id is created.

Fig: Student can also submit the Feedback form for the event they have registered which will be noted for more improvement purposes.