**Data to Track**

The platform will track **event creation**, **student registration**, **attendance**, and **feedback** for events across multiple colleges.

* Events: Name, description, college, organizer, date/time, venue, category, status (active/cancelled)
* Users: Student or Admin details, college, role
* Registrations: Which students registered for which events, timestamp
* Attendance: Marked attendance per event per student
* Feedback: Ratings/comments after event by attendees

**Database Schema (ER Diagram/Table Sketch)**

Entity-relationship design includes:

* **Colleges**: id, name, location, configuration
* **Users**: id, college\_id, name, email, password\_hash, role
* **Events**: id, college\_id, created\_by(user id), name, description, category\_id, venue, date, status
* **Event\_Categories**: id, name
* **Registrations**: id, event\_id, user\_id, registered\_at
* **Attendance**: id, event\_id, user\_id, present, marked\_at
* **Feedback**: id, event\_id, user\_id, rating, comments, submitted\_at
* **Notifications**: id, user\_id, message, sent\_at, read\_status

Relationships are:

* Many users per college, many events per college.
* Users (as students) register for many events.
* Attendance records link users and events.
* Feedback ties attendee (user) and event.
* Events created by users (admins) in a college.

**API Design**

The main RESTful API endpoints are as follows:

**Authentication:**

* POST /api/auth/login — login
* POST /api/auth/register — register
* GET /api/auth/profile — read profile
* PUT /api/auth/profile — update profile

**Events:**

* GET /api/events — list/search events
* POST /api/events — create event (admin)
* GET /api/events/:id — event details
* PUT /api/events/:id — update
* DELETE /api/events/:id — cancel event

**Registrations:**

* POST /api/registrations/:eventId — register student
* DELETE /api/registrations/:eventId — unregister
* GET /api/registrations/my-registrations — student's list
* GET /api/registrations/event/:eventId — event’s attendee list

**Attendance:**

* POST /api/attendance/:eventId — mark student
* GET /api/attendance/event/:eventId — all attendance
* GET /api/attendance/stats/:eventId — stats/report

**Feedback:**

* POST /api/feedback/:eventId — submit feedback
* GET /api/feedback/event/:eventId — event feedback report

**Reports:**

* GET /api/reports/dashboard — global analytics
* GET /api/reports/events — event stats
* GET /api/reports/students — student metrics
* GET /api/reports/top-students — leaderboard
* GET /api/reports/event-types — category analytics

**Workflow Diagrams (Sequence Overview)**

**Student Registration:**

1. Student sees event listing → clicks register
2. Platform checks for existing registration (duplicate prevention) → creates registration record
3. Student receives confirmation

**Attendance Marking:**

1. On event day, admin opens attendee list
2. Marks as present/absent for each registered student
3. Attendance record is saved

**Reporting:**

1. Admin requests analytics dashboard
2. API queries registration, attendance, feedback tables
3. Data baseline: participation rates, attendance, feedback scores

**Assumptions & Edge Cases**

* **Duplicate registrations:** System checks if registration for (user, event) exists before creating
* **Missing feedback:** Feedback is optional; absent entries shown as “not received” in reporting
* **Cancelled events:** On event cancel (PUT or DELETE /api/events/:id), all future attendance/feedback data associated is flagged as obsolete; students notified automatically
* **Multi-college:** All major data tables are scoped to college\_id
* **Role-Based Access:** Students can only register/feedback, Admins manage events/attendance, Super Admins manage colleges
* **API security:** All endpoints require JWT, role checks, and input validation