Design Document

Data to Track

- **College** — id, name, domain (scopes data).
- **Event** — id, collegeId, title, description, type, startTime, endTime, location, state (draft|published|cancelled|completed), eventCode.
- **Student** — id, studentRoll (human: R001…), name, email, collegeId. Unique: (studentRoll, collegeId).
- **Registration** — id, eventId, studentId, collegeId, registeredAt, attendanceStatus (registered|present|absent|late). Unique: (eventId, studentId).
- **Feedback** — id, eventId, studentId, collegeId, rating (1–5), comments.

Schema

Collage

Column	Type	Null?	Default	Constraints / Notes
id	TEXT	NO	—	PK. Use readable IDs like C001
name	TEXT	NO	—	
domain	TEXT	YES	NULL	optional
createdAt	DATETIME	NO	CURRENT_TIMESTAMP	
updatedAt	DATETIME	NO	CURRENT_TIMESTAMP	

- **Primary Key:** id
- **Foreign Keys:** none
- **Indexes:** none required (PK indexed)
- **Example: { id: "C001", name: "Reva University", domain: "reva.edu" }.**

Student

Column	Type	Null?	Default	Constraints / Notes
id	TEXT	NO	—	PK. e.g., S001
studentRoll	TEXT	NO	—	Human readable (e.g., R005), used in API requests
name	TEXT	YES	NULL	
email	TEXT	YES	NULL	validate format if present

Column	Type	Null?	Default	Constraints / Notes
collegeId	TEXT	NO	—	FK → Colleges(id)
createdAt	DATETIME	NO	CURRENT_TIMESTAMP	
updatedAt	DATETIME	NO	CURRENT_TIMESTAMP	

- **Primary Key:** id
- **Foreign Keys:** collegeId → Colleges(id) (ON DELETE CASCADE in prototype)
- **Indexes:** Unique index on (studentRoll, collegeId) to prevent duplicate rolls per college
- **Example:** { id: "S005", studentRoll: "R005", name: "Kavana", email: "kavana@example.com", collegeId: "C001" }.

Events

Column	Type	Null?	Default	Constraints / Notes
id	TEXT	NO	—	PK. e.g., E001
collegeId	TEXT	NO	—	FK → Colleges(id)
title	TEXT	NO	—	
description	TEXT	YES	NULL	optional
type	TEXT	YES	NULL	e.g., workshop, fest, seminar
startTime	DATETIME	YES	NULL	optional
endTime	DATETIME	YES	NULL	optional
location	TEXT	YES	NULL	optional
state	TEXT	NO	'draft'	enum: ` draft
eventCode	TEXT	YES	NULL	optional human code
createdAt	DATETIME	NO	CURRENT_TIMESTAMP	
updatedAt	DATETIME	NO	CURRENT_TIMESTAMP	

- **Primary Key:** id
- **Foreign Keys:** collegeId → Colleges(id)
- **Indexes:** index on collegeId; consider index on state if filtering often
- **Example:** { id: "E001", collegeId: "C001", title: "AI Workshop", type: "workshop", state: "published" }.

Registrations

Column	Type	Null?	Default	Constraints / Notes
id	TEXT	NO	—	PK
eventId	TEXT	NO	—	FK → Events(id)
studentId	TEXT	NO	—	FK → Students(id)
collegeId	TEXT	NO	—	FK → Colleges(id) (denormalized for fast queries)
registeredAt	DATETIME	NO	CURRENT_TIMESTAMP	
attendanceStatus	TEXT	NO	'registered'	enum: ` registered
createdAt	DATETIME	NO	CURRENT_TIMESTAMP	
updatedAt	DATETIME	NO	CURRENT_TIMESTAMP	

- **Primary Key:** id
- **Foreign Keys:** eventId → Events(id), studentId → Students(id), collegeId → Colleges(id)
- **Indexes:**
- Unique index on (eventId, studentId) to prevent duplicate registrations (DB-enforced)
- Index on collegeId (for reports)
- Index on eventId (for event-scoped queries)
- **Example:** { id: "R0001", eventId: "E001", studentId: "S005", collegeId: "C001", attendanceStatus: "present" }

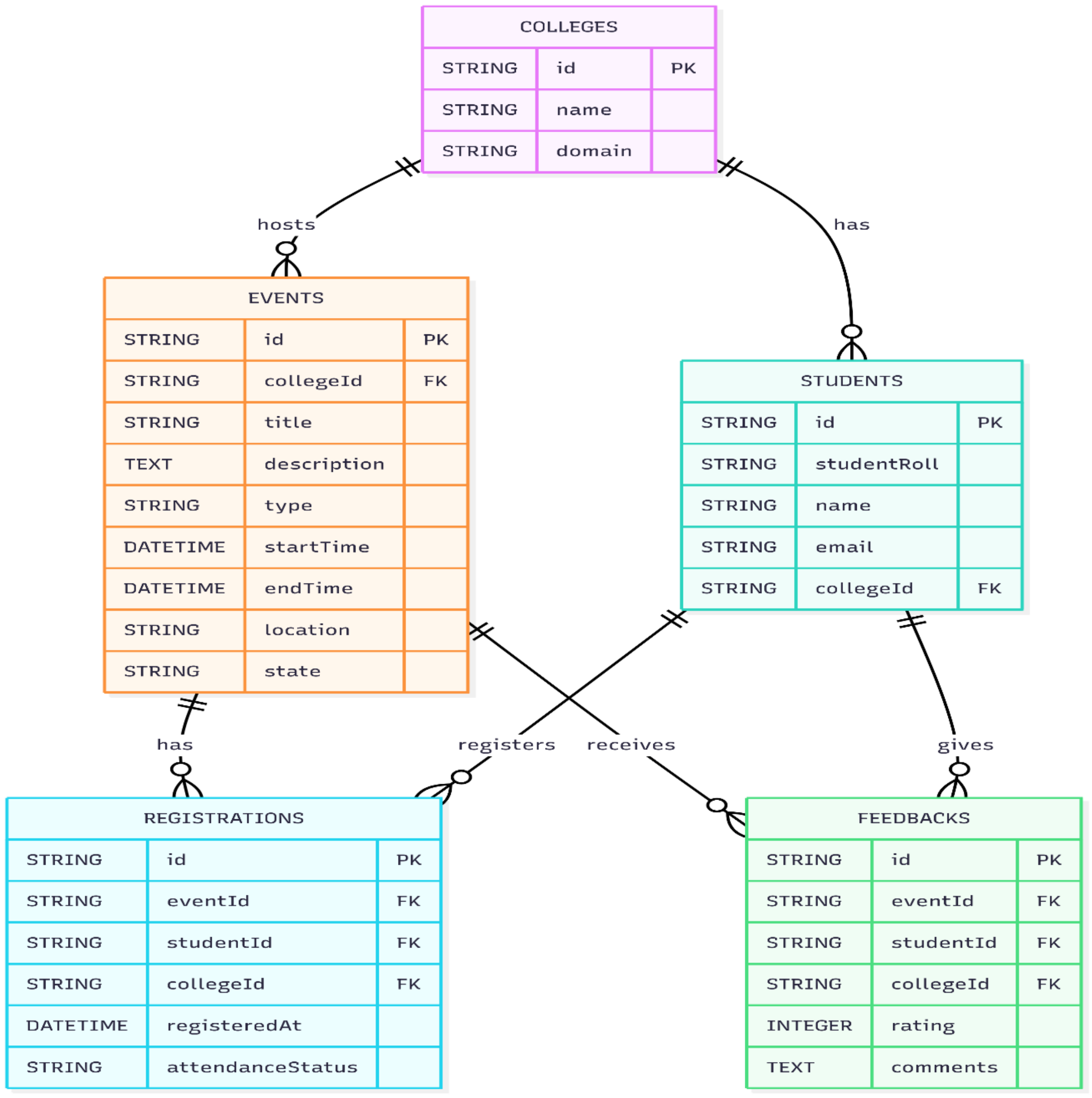
Feedback

Column	Type	Null?	Default	Constraints / Notes
id	TEXT	NO	—	PK
eventId	TEXT	NO	—	FK → Events(id)
studentId	TEXT	NO	—	FK → Students(id)
collegeId	TEXT	NO	—	FK → Colleges(id) (denormalized)
rating	INTEGER	NO	—	1..5 (validate at application level)
comments	TEXT	YES	NULL	optional
createdAt	DATETIME	NO	CURRENT_TIMESTAMP	

Column	Type	Null?	Default	Constraints / Notes
updatedAt	DATETIME	NO	CURRENT_TIMESTAMP	

- **Primary Key:** id
- **Foreign Keys:** eventId → Events(id), studentId → Students(id), collegeId → Colleges(id)
- **Indexes:** index on collegeId (optional)
- **Optional Constraint:** unique (eventId, studentId) if you want to restrict to one feedback per student per event
- **Example:** { id: "F001", eventId: "E002", studentId: "S005", collegeId: "C001", rating: 4, comments: "Learned a lot!" }.

Diagram



API Design

API Design (Theory)

The backend provides REST APIs under the /api prefix. All data is exchanged in **JSON**.

1. Event Management

- **POST /api/events** → Create a new event.
  Requires: collegeId, title. Optional: type, startTime, endTime, location, state.
- **GET /api/events** → List events.
  Supports filters by collegeId, state, and type.
- **DELETE /api/events/:id** → Permanently delete an event (hard delete).
- **POST /api/events/:id/cancel** → Cancel an event (soft delete by updating state to cancelled).

2. Student Registration

- **POST /api/register** → Register a student for an event.
  - Creates the student if not already present.
  - Prevents duplicate registrations (unique combination of eventId + studentId).
  - Allowed only if the event is in the published state.

3. Attendance Tracking

- **POST /api/attendance** → Update a student's attendance for an event.
  - Required fields: eventId, studentRoll, collegeId, status.
  - Status can be: registered, present, absent, late.
  - Returns updated registration details.

4. Feedback Collection

- **POST /api/feedback** → Submit feedback for an event.
  - Required fields: eventId, studentRoll, collegeId, rating (1–5).
  - Optional: comments.
  - Stores ratings for use in reports (e.g., average feedback per event).

5. Reporting APIs

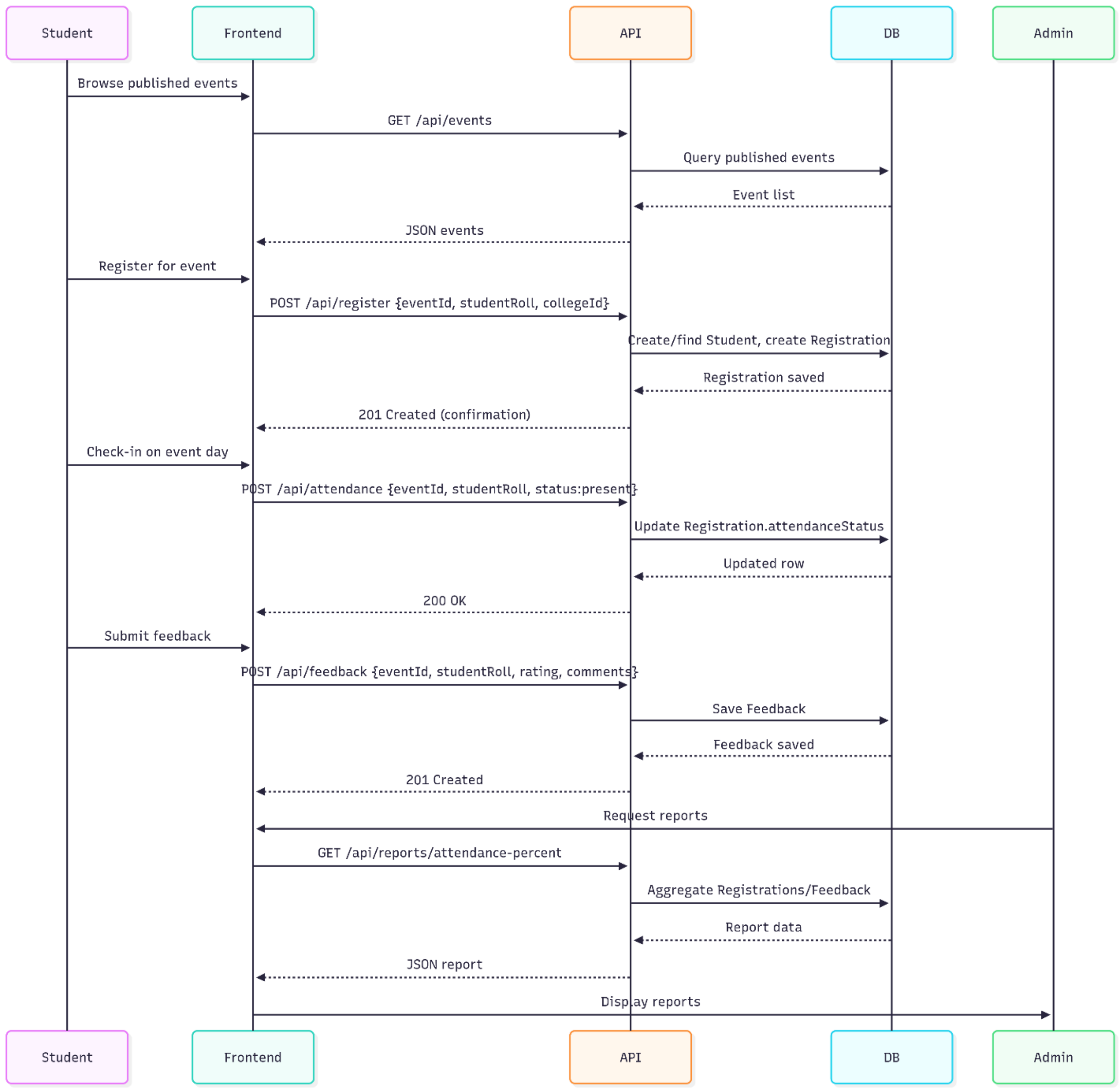
- **GET /api/reports/event-popularity** → Returns events sorted by number of registrations.
- **GET /api/reports/attendance-percent** → Shows attendance percentage per event.
- **GET /api/reports/avg-feedback** → Returns average rating and total feedback count per event.

- **GET /api/reports/student-participation** → Shows how many events each student registered for and attended.
- **GET /api/reports/top-active-students** → Lists top N students ranked by number of events attended.

Endpoint	**Method**	**Purpose**	**Notes**
/api/events	POST	Create a new event	Requires collegeId, title; optional: type, time, location, state.
/api/events	GET	List events (with filters)	Query params: collegeId, state, type. Returns array of events.
/api/events/:id	DELETE	Delete an event (hard delete)	Removes event permanently. Risk: may orphan related registrations/feedback.
/api/events/:id/cancel	POST	Cancel an event (soft delete)	Updates event state to cancelled. Registrations remain but new signups blocked.
/api/register	POST	Register a student for an event	Creates student if not found. Only allowed if event is published. Prevents duplicate registration.
/api/attendance	POST	Mark attendance for a student	Requires eventId, studentRoll, collegeId, status (present/absent/late).
/api/feedback	POST	Submit feedback for an event	Requires eventId, studentRoll, collegeId, rating (1–5). comments optional.
/api/reports/event-popularity	GET	Show events sorted by registrations	Query param: collegeId. Returns list of events with registration counts.
/api/reports/attendance-percent	GET	Show attendance percentage per event	Query params: collegeId, optional eventId. Returns registered vs present stats.
/api/reports/avg-feedback	GET	Show average feedback per event	Query param: collegeId. Returns avg rating and feedback count.
/api/reports/student-participation	GET	Show participation per student	Query param: collegeId. Returns events registered vs attended for each student.

Endpoint	**Method**	**Purpose**	**Notes**
/api/reports/top-active-students	GET	Show top active students	Query param: collegeId. Returns top N students ranked by attendance.

Workflows


Student
Frontend
API
DB
Admin
Browse published events
GET /api/events
Query published events
Event list
JSON events
Register for event
POST /api/register {eventId, studentRoll, collegeId}
Create/find Student, create Registration
Registration saved
201 Created (confirmation)
Check-in on event day
POST /api/attendance {eventId, studentRoll, status:present}
Update Registration.attendanceStatus
Updated row
200 OK
Submit feedback
POST /api/feedback {eventId, studentRoll, rating, comments}
Save Feedback
Feedback saved
201 Created
Request reports
GET /api/reports/attendance-percent
Aggregate Registrations/Feedback
Report data
JSON report
Display reports

The workflow begins with the **student** browsing published events and registering. The system verifies the event status, creates a student record if needed, and saves the registration. On the event day, the student checks in, and their **attendance status** is updated. After the event, the student submits feedback, which is stored in the database. Finally, the **admin** requests reports, and the system aggregates registrations, attendance, and feedback data to generate insights such as event popularity, attendance percentage, and average feedback.

Assumptions & Edge Cases

Assumptions

- **Unique student identity** → A student is uniquely identified by (studentRoll, collegeId).
- **Readable IDs** → Prototype uses simple IDs (C001, E001, S001) for clarity.
- **Event states** → Only published events allow registration.
- **One college scope** → Prototype includes one college (C001), but schema supports multiple.
- **No authentication** → Prototype has open endpoints; in real deployment, JWT/roles would be added.
- **Feedback rules** → Students can give one feedback per event (no duplicates expected).

Edge Cases & Handling

- **Duplicate Registration** → Blocked by DB unique constraint (eventId, studentId). API returns 409 Conflict.
- **Unpublished / Cancelled Events** → Registration attempts rejected with 400 Bad Request.
- **Missing Student** → If student doesn't exist at registration → created. For attendance/feedback → 404 Not Found.
- **Attendance Without Registration** → API returns 404 Registration not found.
- **Invalid Feedback Rating** → API validates rating ∈ [1,5]; else returns 400 Bad Request.
- **Cancelled Events After Registration** → Existing registrations remain, but new ones are disallowed.
- **Missing Feedback** → Reports still run; events without feedback show avgRating = null or 0.