Full Stack Application Development

Assignment

10-05-2025

SCHOOL VACCINATION SYSTEM

Submitted by:
Harshitha Palligunthala
2023tm93691

Table of Contents

System Overview

The **School Vaccination Management System** is designed to facilitate the efficient management of student vaccination efforts in schools. The system allows school coordinators to:

- Simulate authentication for access to the system
- View a dashboard with key vaccination metrics
- Manage students (add individually or via bulk CSV upload)
- Schedule and manage vaccination drives
- Record and track vaccinations
- Generate reports

The application stack includes:

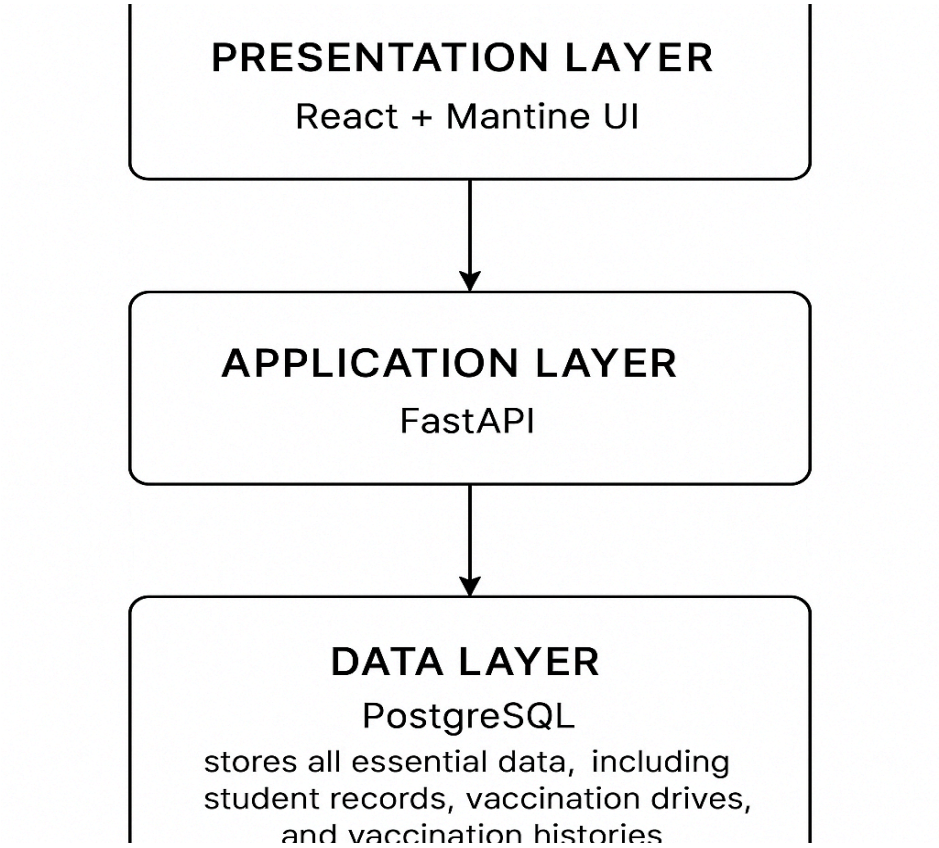
- **Frontend**: React.js with Mantine UI components for a modern, interactive UI
- **Backend**: FastAPI for building RESTful APIs
- **Database**: PostgreSQL for data persistence

Frontend: React.js with Mantine UI
React.js is a powerful JavaScript library for building interactive and component-driven user interfaces. Combined with Mantine UI, it provides a clean, modern design system with built-in components, themes, and responsive layouts, enabling rapid development of a professional-grade frontend.

Backend: FastAPI
FastAPI is a high-performance Python web framework designed for building fast and scalable RESTful APIs. It offers automatic data validation, interactive API documentation (Swagger/OpenAPI), and asynchronous capabilities, making it ideal for backend services that require speed and clarity.

Database: PostgreSQL
PostgreSQL is a robust, open-source relational database known for its reliability and powerful querying capabilities. It stores structured application data such as student records, vaccination drives, and vaccination logs securely and efficiently.

Application Architecture



The application follows a **3-tier architecture**:

1. **Presentation Layer**:
    - **Frontend** built with **React.js** using **Mantine** UI components, which provides a user-friendly interface for school coordinators to interact with the system.
2. **Application Layer**:

    - **Backend** developed using **FastAPI** to handle API requests, business logic, and data validation.
3. **Data Layer**:
    - **PostgreSQL Database** stores all essential data, including student records, vaccination drives, and vaccination histories.

Key Components:

- Frontend communicates with the backend via HTTP requests.
- Backend manages data processing, validation, and business logic.
- Database persists all application data securely.

Route-centric approach with FastAPI is **More efficient** for API development and **Easier to maintain** with fewer layers.

Key Features Highlighted at the Frontend

Authentication Flow

- Dual storage (session/local)
- Time-based auto logout
- Protected route wrapper

Navigation System

- Icon-based menu
- Active route highlighting
- Programmatic navigation

Design System

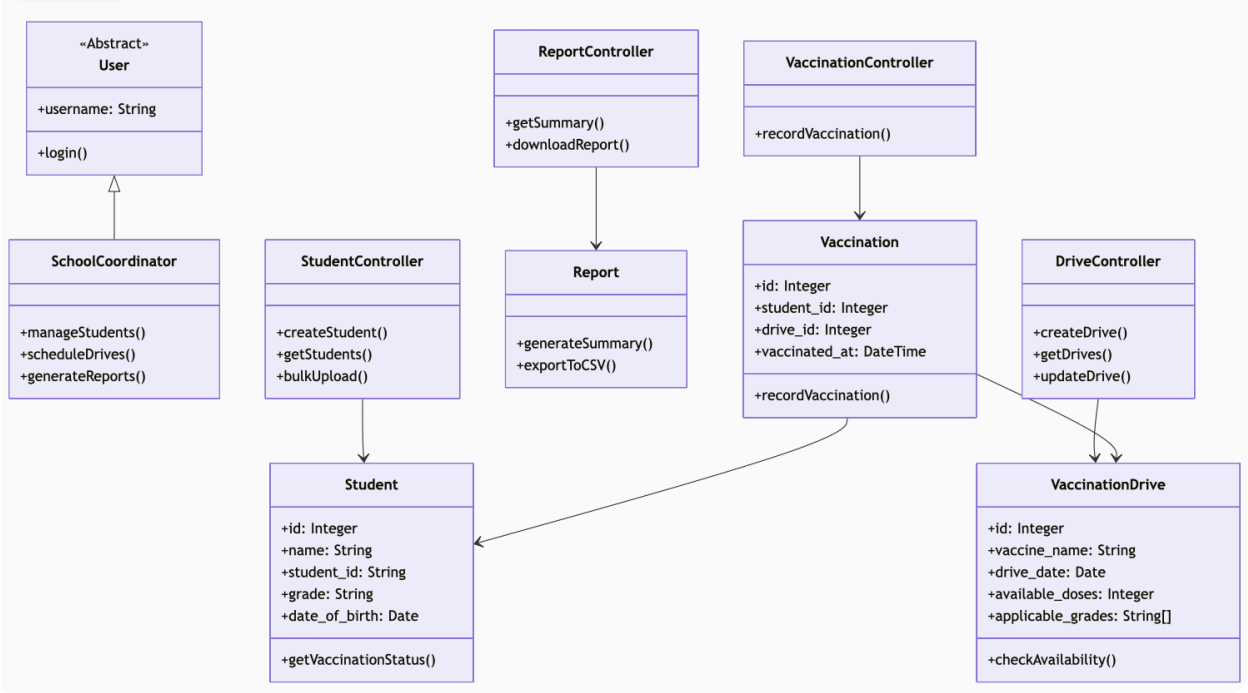
- Custom theme extension
- Consistent spacing/radius
- Color palette management

Technical Highlights

- Clean component composition
- Type-safe implementation
- Responsive-first approach
- Centralized layout management
- Notification system integration

System Design Diagrams

Class diagram:



1. User (Abstract Class)

- Represents a general user in the system.
- Contains common properties and methods for all users.
- Attributes:
    - username: String – the identifier for the user.
- Methods:
    - login() – simulated login functionality (e.g., hardcoded credentials).

2. SchoolCoordinator (inherits from User)

- A specific user type that manages the portal operations.
- Responsibilities:
    - manageStudents() – handles student creation, updates, or deletion.

    - scheduleDrives() – responsible for creating and managing vaccination drives.
    - generateReports() – can generate and download summary reports.

3. StudentController

- Manages operations related to students.
- Methods:
    - createStudent() – adds a new student.
    - getStudents() – retrieves a list of all students.
    - bulkUpload() – allows CSV upload of multiple students at once.

4. Student

- Represents a student entity in the system.
- Attributes:
    - id: Integer, name: String, student_id: String, grade: String, date_of_birth: Date
- Methods:
    - getVaccinationStatus() – returns the vaccination record(s) for the student.

5. DriveController

- Handles creation and management of vaccination drives.
- Methods:
    - createDrive() – creates a new vaccination drive.
    - getDrives() – fetches all or filtered vaccination drives.
    - updateDrive() – modifies an existing drive's information.

6. VaccinationDrive

- Represents a scheduled vaccination drive.
- Attributes:
    - id: Integer, vaccine_name: String, drive_date: Date, available_doses: Integer, applicable_grades: String[]
- Methods:

    - checkAvailability() – verifies if vaccines are available for a particular grade or student.

7. VaccinationController

- Manages recording of vaccination data.
- Methods:
    - recordVaccination() – logs the vaccination of a student under a specific drive.

8. Vaccination

- Entity representing a vaccination event.
- Attributes:
    - id: Integer, student_id: Integer, drive_id: Integer, vaccinated_at: DateTime
- Methods:
    - recordVaccination() – stores the vaccination details linking student and drive.

9. ReportController

- Responsible for retrieving vaccination reports.
- Methods:
    - getSummary() – fetches a summarized report.
    - downloadReport() – exports the report in a downloadable format (CSV).

10. Report

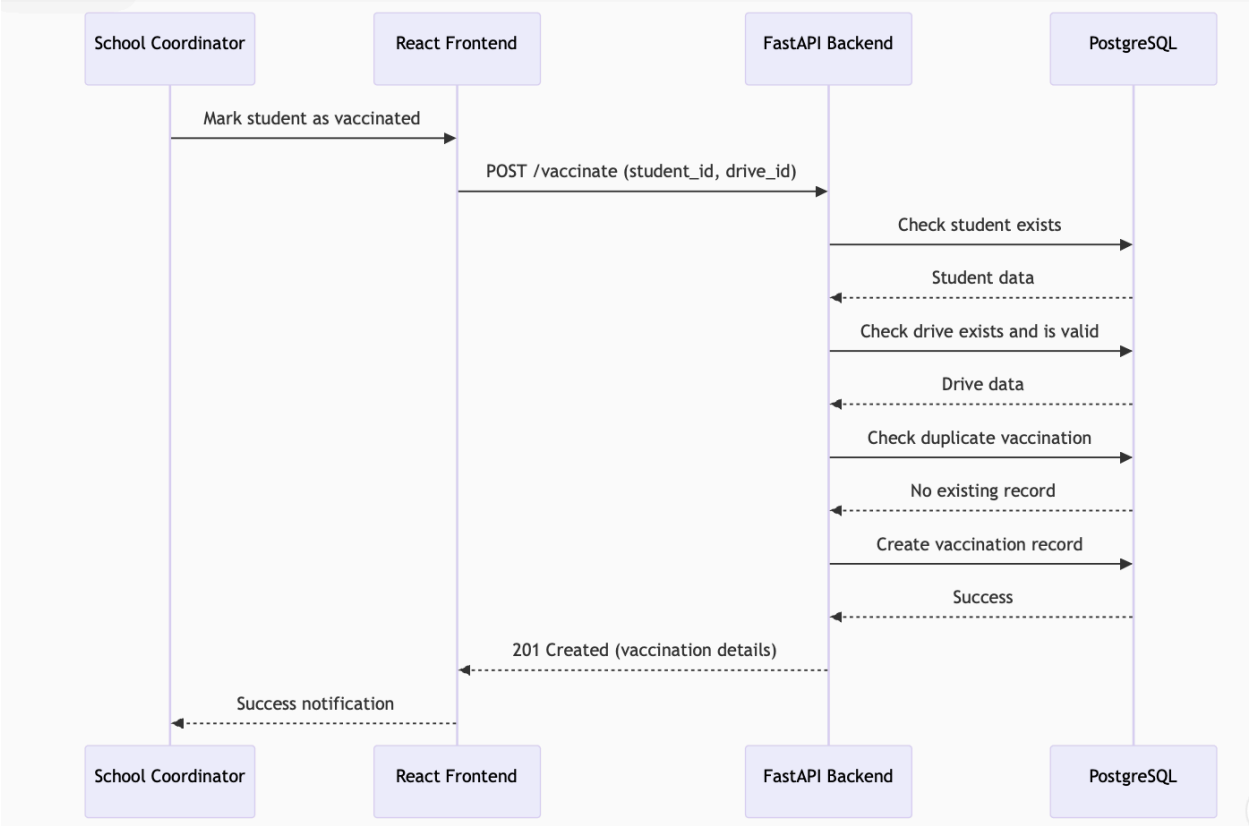
- Represents a report entity generated from student and vaccination data.
- Methods:
    - generateSummary() – compiles data into a readable summary.
    - exportToCSV() – converts the report into a downloadable CSV file.

Relationships

- SchoolCoordinator uses StudentController, DriveController, VaccinationController, and ReportController.
- Vaccination is associated with both Student and VaccinationDrive through student_id and drive_id.
- Report is generated from data provided by Student and Vaccination.

Sequence Diagram:

Vaccination Recording Flow:



Purpose:

This sequence diagram illustrates the step-by-step process of recording a vaccination event in the system. It involves interaction between the school coordinator (end user), frontend UI, FastAPI backend, and the PostgreSQL database.

Actors & Components Involved:

- School Coordinator: The user who initiates the action to mark a student as vaccinated.
- React Frontend: The client-side interface which collects input and triggers the backend API call.
- FastAPI Backend: The server-side logic that handles the vaccination logic.
- PostgreSQL: The database storing student, drive, and vaccination records.

Step-by-Step Flow:

1. Mark Student as Vaccinated (Initiation)
   The school coordinator triggers the action via the frontend UI (e.g., selecting a student and a drive).
2. POST Request to /vaccinate
   The frontend sends a POST request to the FastAPI endpoint /vaccinate, passing student_id and drive_id.
3. Check if Student Exists
   The backend queries the PostgreSQL database to ensure the student exists.
4. Check if Drive is Valid
   The backend checks whether the vaccination drive exists and is applicable for the student's grade and date.
5. Check for Duplicate Vaccination
   The backend verifies that this student hasn't already been vaccinated in the same drive to avoid duplicate entries.
6. Create Vaccination Record
   If all checks pass, the backend inserts a new record into the Vaccination table.
7. Respond with Success
   A 201 Created response is returned with the vaccination details.
8. Notify the Coordinator
   The frontend shows a success notification to the school coordinator confirming the vaccination was recorded.

Key Validations in Backend Logic:

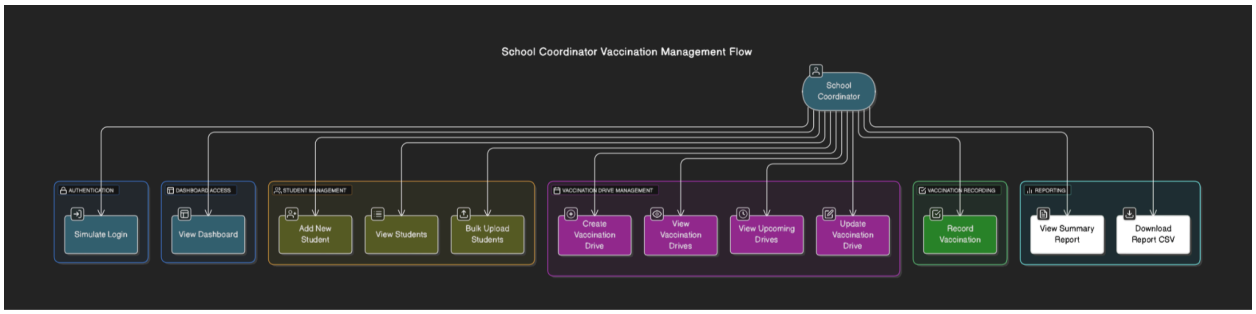
- Student existence and eligibility.
- Drive validity and availability.

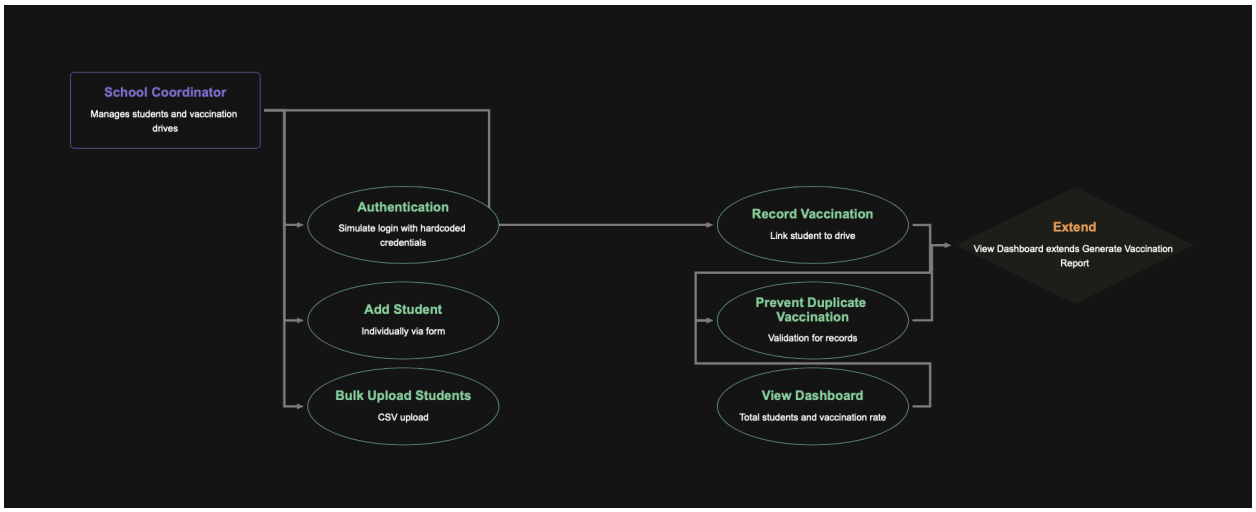
- Duplicate vaccination prevention.

Status Codes:

- 201 Created — Vaccination successfully recorded.
- 400/404 (not shown in diagram) — Possible if student or drive is not found, or if duplicate detected.

Use Case Diagram





Actors:

1. School Coordinator (Primary Actor)
    - Manages students, vaccination drives, and reports.
    - Authenticates into the system.

Use Cases:

1. Authentication

    - Simulate Login (Hardcoded credentials for demo)
2. Student Management
    - Add Student (Individually via form)
    - Bulk Upload Students (CSV upload)
    - View Students (List with pagination)
    - Delete Student
3. Vaccination Drive Management
    - Schedule Vaccination Drive (Future dates, min. 15 days ahead)
    - View Upcoming Drives (Next 30 days)
    - Update Drive Details (Edit doses, grades, etc.)
4. Vaccination Recording
    - Record Vaccination (Link student to drive)
    - Prevent Duplicate Vaccination (Validation)
5. Reports & Analytics
    - View Dashboard (Total students, vaccination rate, upcoming drives)
    - Generate Vaccination Report (Summary statistics)
    - Download Report as CSV
6. System Validations
    - Validate Student Eligibility (Grade check)
    - Check Drive Availability (Dose limits, dates)

Key Relationships:

- Include Relationships:
    - Record Vaccination includes Validate Student Eligibility and Check Drive Availability.
    - Bulk Upload Students includes Validate CSV Format.
- Extend Relationships:
    - View Dashboard extends Generate Vaccination Report.

Frontend-Backend Interaction

The interaction between the frontend (React) and backend (FastAPI) follows a structured RESTful API approach, ensuring seamless data flow and state management. Below is an elaborated breakdown of the key interaction patterns:

1. Authentication Flow

- **Simulated Login Mechanism**:
  The frontend sends a hardcoded username (e.g., "admin") to the backend via a POST /simulate-login request. Since this is a simulated system, no actual password validation occurs.
    - Backend Response:
      The backend generates a mock JWT token (e.g., "simulated_jwt_token") and returns it in the response. This token is stored in the frontend's state (or local storage) and attached to subsequent API requests in the Authorization header.
    - Security Consideration:
      While this approach suffices for development, a production system would integrate OAuth2 or session-based authentication for secure access control.

2. Data Fetching

- **Dashboard Data Aggregation**:
  Upon successful login, the frontend immediately fetches dashboard analytics by making parallel requests to:
    - GET /reports/ → Retrieves summarized metrics (total students, vaccination rate, etc.).
    - GET /drives/upcoming/ → Fetches drives scheduled within the next 30 days (filtered backend-side).
    - Error Handling:
      If no drives exist, the backend returns an empty array, and the frontend displays a user-friendly empty state (e.g., "No upcoming drives scheduled").
- **Student & Drive Management**:
    - The student list is fetched via GET /students/ with optional query params (?grade=5 or ?vaccinated=true) for filtering.
    - Drive data is retrieved via GET /drives/, with upcoming drives cached to minimize redundant API calls.

3. Data Modification

- **CRUD Operations**:

    - Student Creation/Update:
      The frontend sends a structured JSON payload (e.g., student details) to POST /students/ or PUT /students/{id}. On success, the backend returns the updated record, and the frontend updates its local state to reflect changes without a full page reload.
    - Bulk Uploads:
      CSV files are sent as multipart/form-data to POST /students/upload-csv/. The backend validates each row before insertion and returns a summary (success/fail counts).
- **Vaccination Recording**:
    - When marking a student as vaccinated, the frontend sends { student_id, drive_id } to POST /vaccinate/.
    - The backend checks for duplicate vaccinations and valid drive dates before committing to the database. A failed validation returns a 4XX error with actionable feedback.

4. Real-Time Feedback & State Sync

- Optimistic Updates:
  For smoother UX, the frontend temporarily updates the UI before receiving API confirmation (e.g., showing a "Vaccinated" badge immediately). If the backend fails, the UI reverts with an error toast.
- Pessimistic Updates for Critical Actions:
  Drive scheduling (POST /drives/) waits for backend validation (e.g., no date conflicts) before updating the UI to ensure data consistency.

5. Error Handling & User Feedback

- Structured Error Responses:
  The backend returns consistent error formats (e.g., { "detail": "Drive date invalid" }), allowing the frontend to display contextual alerts.
- Retry Mechanism:
  Failed requests (e.g., due to network issues) automatically retry **twice** before showing an error.

Key Considerations for Production

1. Authentication: Replace simulated login with JWT + refresh tokens.

2. WebSockets: For real-time updates (e.g., new drive notifications).
3. Request Batching: Combine /reports and /drives/upcoming into a single dashboard endpointto reduce round trips.

This structured interaction ensures predictable behavior, efficient data flow, and responsive feedback for end-users.

API Endpoints

Authentication

- **POST /simulate-login**
  Simulated login endpoint that accepts any credentials and returns a token.

Student Management

- **POST /students/**
  Adds a new student record.
- **GET /students/**
  Lists all students in the system.
- **POST /students/upload-csv/**
  Bulk upload of student records via a CSV file.

Vaccination Drive Management

- **POST /drives/**
  Creates a new vaccination drive.
- **GET /drives/**
  Lists all vaccination drives.
- **GET /drives/upcoming/**
  Retrieves all upcoming vaccination drives (within the next 30 days).
- **PUT /drives/{drive_id}**
  Updates an existing vaccination drive.

Vaccination Recording

- **POST /vaccinate/**
  Records a vaccination event, linking a student to a specific drive.

Reports

- **GET /reports/**
  Retrieves summary statistics such as total students, vaccinated students, and vaccination rate.
- **GET /reports/download/**
  Downloads a vaccination report as a CSV file.

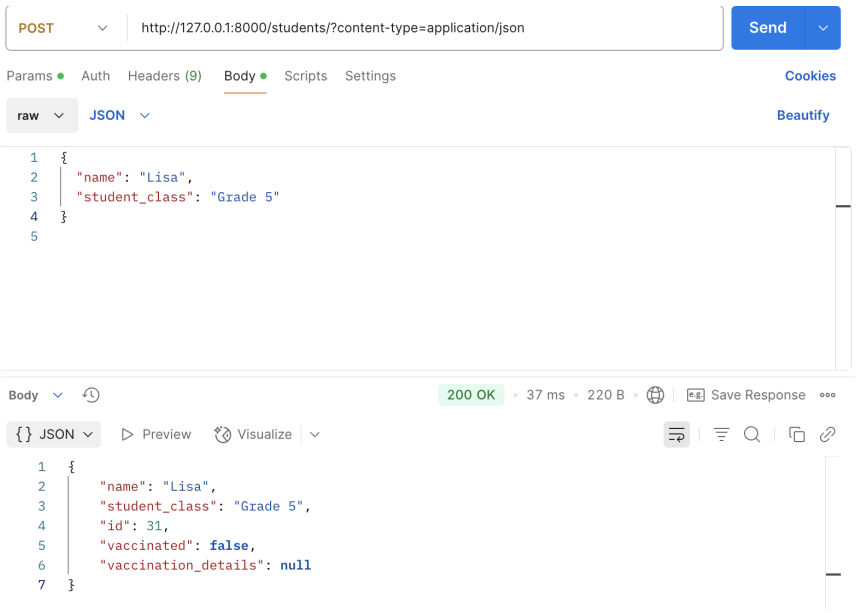
API Documentation

Detailed API documentation is available in **Swagger UI** when running the backend locally at:
http://localhost:8000/docs

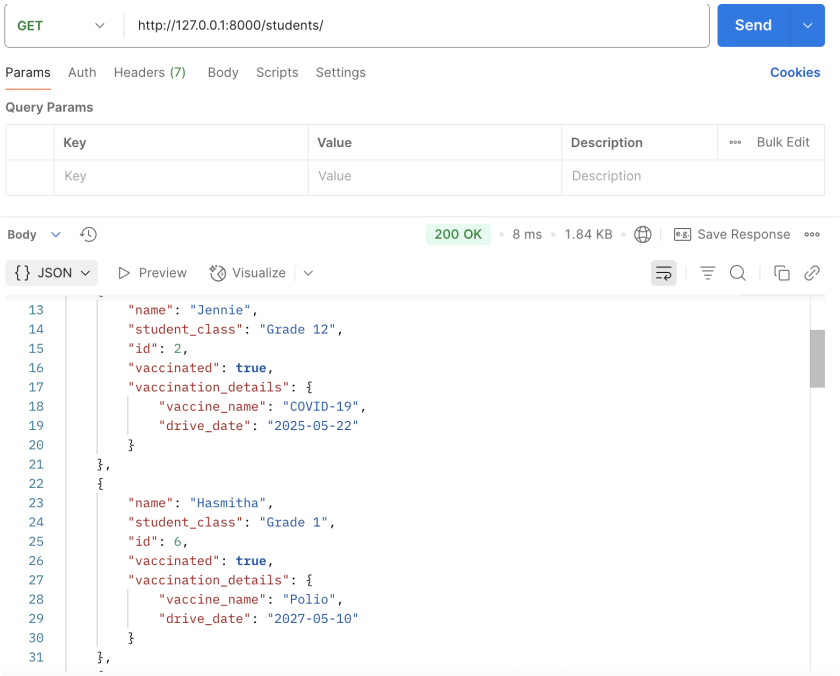
API Request/Response:

Students:

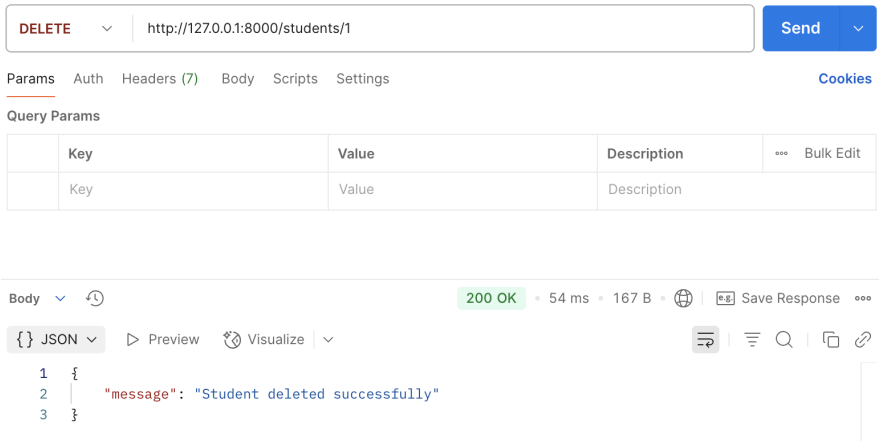
- Endpoint: POST /students/



- Endpoint: GET /students/

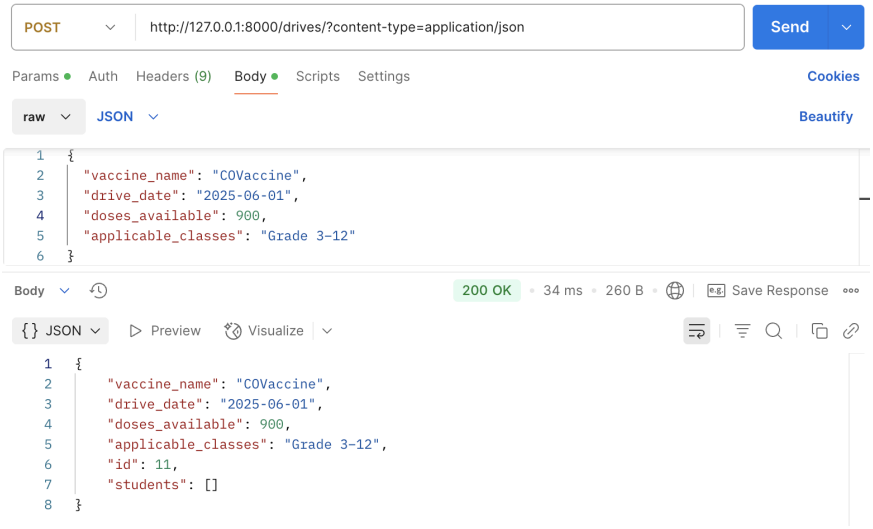


- Endpoint: DELETE /students/1

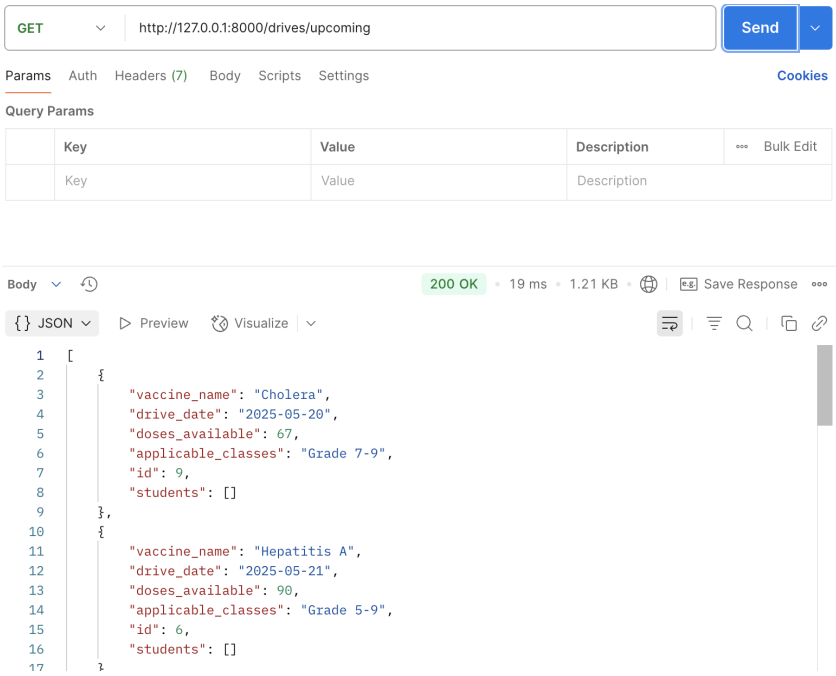


Drives:

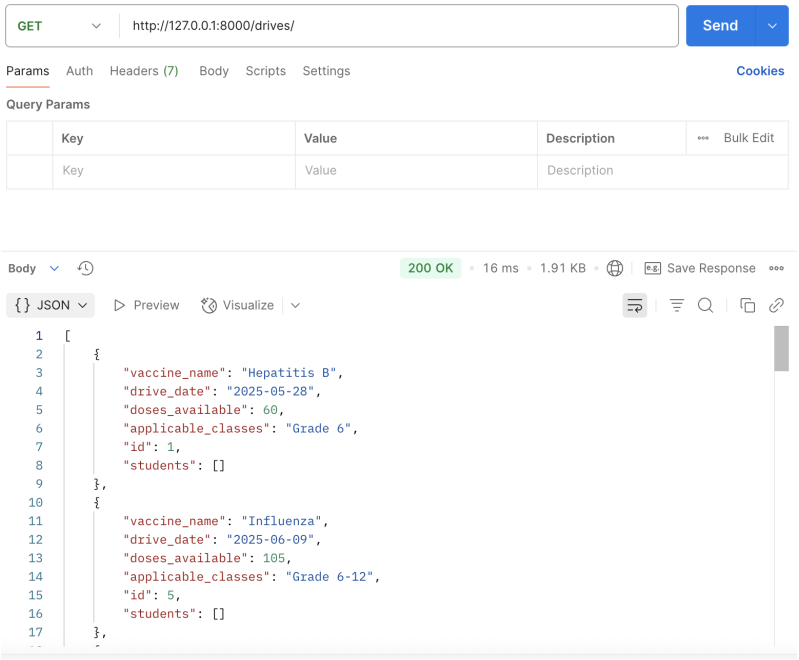
- Endpoint: POST /drives/



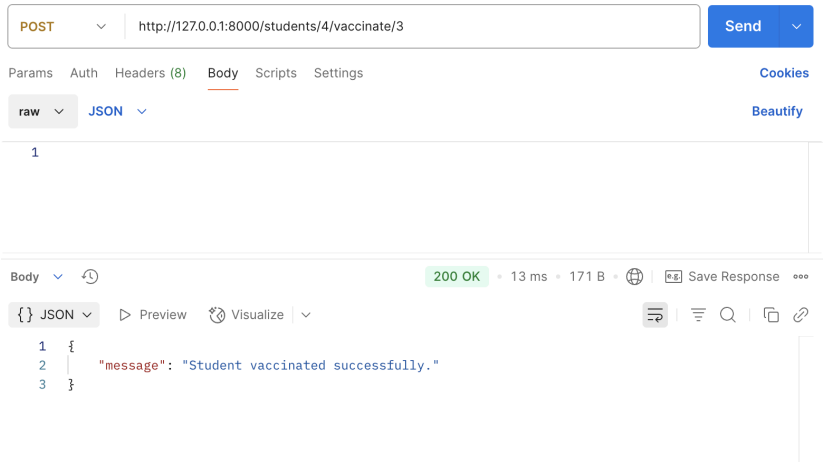
- Endpoint: GET /drives/upcoming



- Endpoint: GET /drives/

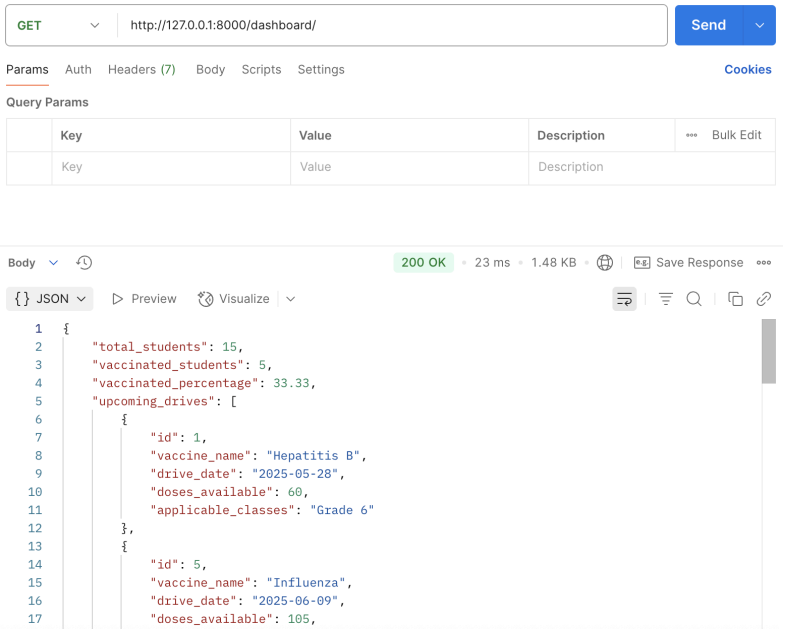


- Endpoint: POST /drives/students/4/vaccinate/3

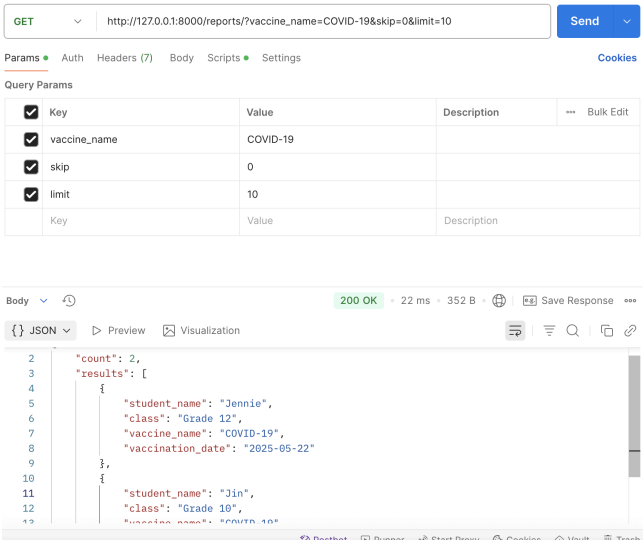


- Endpoint: PUT /drives/1

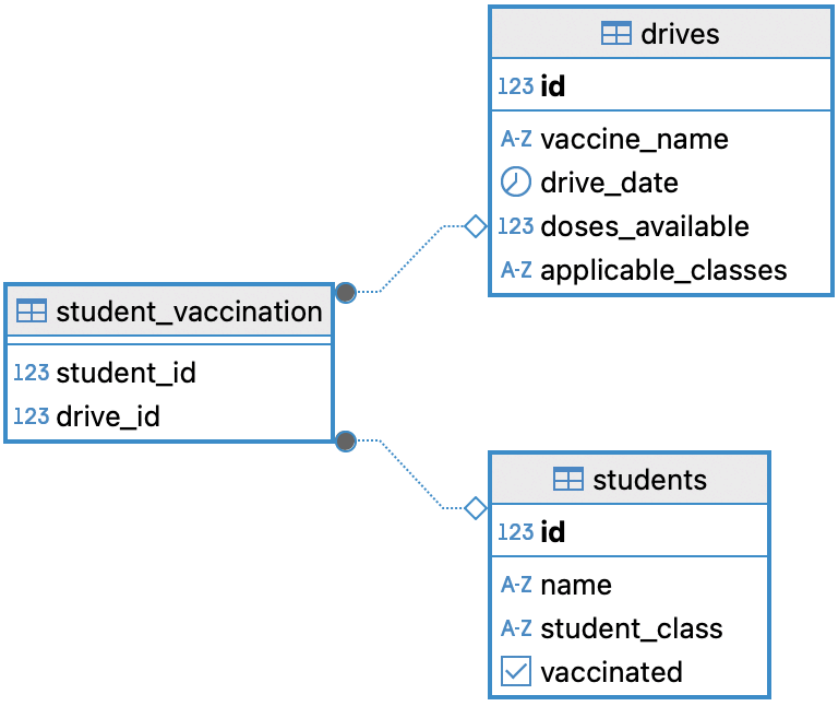
- Endpoint: GET /dashboard



- Endpoint: GET /reports



Database Schema



Tables:

➢ students:

Column Name	#	Data type	Identity	Collation	Not Null
123 id	1	serial4			[v]
A-Z name	2	varchar		default	[ ]
A-Z student_cla	3	varchar		default	[ ]
☑ vaccinated	4	bool			[ ]

➢ vaccination_drives:

Column Name	#	Data type	Identity	Collation	Not Null
123 id	1	serial4			[v]
A-Z vaccine_name	2	varchar		default	[ ]
drive_date	3	date			[ ]
123 doses_available	4	int4			[ ]
A-Z applicable_class	5	varchar		default	[ ]

➢ Student_vaccination:

Column Name	#	Data type	Identity	Collation	Not Null
123 student_ic	1	int4			[ ]
123 drive_id	2	int4			[ ]

1. drives Table

- **Purpose**: Stores information about vaccination drives
- **Fields**:
  - id (Primary Key): Unique identifier for each drive (integer)
  - vaccine_name: Name of the vaccine being administered (string)
  - drive_date: Scheduled date of the vaccination drive (date)
  - doses_available: Number of vaccine doses available (integer)
  - applicable_classes: Grade/class levels eligible for this drive (string)

2. students Table

- **Purpose**: Contains student records and vaccination status
- **Fields**:
    - id (Primary Key): Unique identifier for each student (integer)
    - name: Full name of the student (string)
    - student_class: Grade/class level of the student (string)
    - vaccinated: Vaccination status flag (boolean)

3. student_vaccination Table

- **Purpose**: Junction table recording which students attended which drives
- **Fields**:
    - student_id (Foreign Key): References students.id (integer)
    - drive_id (Foreign Key): References drives.id (integer)

Relationships

1. **Many-to-Many Relationship**:
    - A student can attend multiple vaccination drives
    - A drive can include multiple students
    - This relationship is implemented through the student_vaccination junction table
2. **Foreign Key Constraints**:
    - student_vaccination.student_id → students.id
    - student_vaccination.drive_id → drives.id

Assumptions

1. **Authentication**:
    - Simulated login with hardcoded credentials and password validation.
    - The system assumes a single admin role for access control.
2. **Data Validation**:
    - The backend ensures vaccination drives are scheduled at least 15 days in advance , past drives are not edited.

    - The backend prevents duplicate vaccinations for the same student and drive.
3. **UI/UX**:
    - The dashboard view shows the total no.of students registered for the drive, and the vaccinated students count along with the vaccination rate in percentage. upcoming vaccination drives (within the next 30 days) is also shown.
    - Empty states are handled gracefully with informative messages.
    - Bulk CSV uploads require a specific format (detailed in the application).
4. **Performance**:
    - In-memory storage is used for demonstration purposes. In production, a robust database solution and optimization strategies should be employed.

User Stories Implementation

1. User Story 1.1: Dashboard Overview

As a school coordinator, I should be presented with a landing page on successful login showing key metrics and navigation options.

Acceptance Criteria

Implemented:

- Displays real-time statistics:
    - Total students registered
    - Number/percentage of vaccinated students
    - Upcoming drives (next 30 days)
- Shows empty states (e.g., "No drives scheduled") when no data exists.
- Provides quick navigation to:
    - Student management
    - Drive scheduling
    - Reports

Technical Implementation

- **Frontend**:
    - Fetches data from /reports and /drives/upcoming on page load.
    - Uses Mantine's <Paper> and <SimpleGrid> for responsive cards.
- **Backend**:
    - /reports aggregates data via SQL queries (e.g., COUNT vaccinated students).
    - /drives/upcoming filters drives using WHERE drive_date BETWEEN NOW() AND NOW() + 30d.

2. User Story 1.2: Student Management

As a school coordinator, I should be able to add/edit students individually or via bulk CSV upload.

Acceptance Criteria

Implemented:

- **Form for individual entry**:
    - Fields: Name, Student ID, Grade.
    - Validation: Unique student_id, grade format (e.g., "5" or "12").
- **Bulk CSV upload**:
    - Accepts CSV with headers: name, student_id, grade.
    - Returns success/failure counts.
- **Search/filter** by name, Id.

Technical Implementation

- **Frontend**:
    - Uses <Modal> for CSV upload with file validation.
    - Table with pagination (@mantine/core <Table>).
- **Backend**:
    - POST /students: Validates data using Pydantic.
    - POST /students/upload-csv: Parses CSV with csv.DictReader.

3. User Story 1.3: Report Generation

As a coordinator, I should generate vaccination reports with filters and download options.

Acceptance Criteria

Implemented:

- **Tabular view** of students with columns:
    - Name, Grade, Vaccination Status, Vaccine Name, Date.
- **Filters**: By vaccine name.
- **Download as CSV/PDF**:
    - CSV: Generated via FastAPI's StreamingResponse.
    - PDF: Uses react-to-pdf (frontend).

Technical Implementation

- **Backend**:
    - GET /reports/download?format=csv: Streams CSV using StringIO.
- **Frontend**:
    - <MultiSelect> for filters.
    - "Export" button triggers PDF generation.

4. User Story 1.4: Vaccination Drive Scheduling

As a coordinator, I should schedule drives with date, vaccine, and dose constraints.

Acceptance Criteria

Implemented:

- **Form inputs**:
    - Vaccine name, date (≥15 days ahead), doses, applicable grades.
- **Conflict prevention**:
    - Blocks overlapping drives for the same grades.
- **Edit/Cancel**:
    - Only allowed for future drives.

Technical Implementation

    - Disables past drives in the date picker.

5. User Story 1.5: Vaccination Recording

As a coordinator, I should mark students as vaccinated during a drive.

Acceptance Criteria

Implemented:

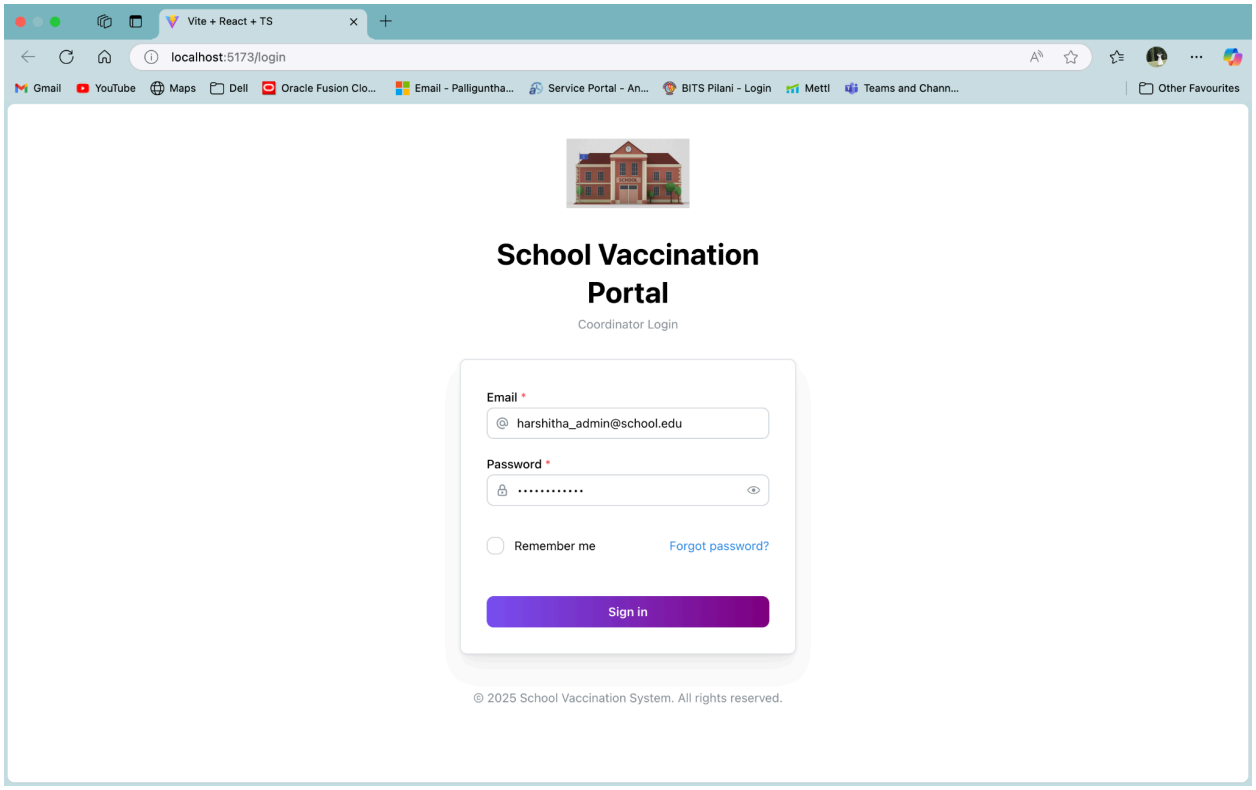
- **Prevents duplicates**:
    - Same student + same drive → Error.

Technical Implementation

- **Backend**:
    - POST /vaccinate: Checks UNIQUE(student_id, drive_id) constraint.
- **Frontend**:
    - Dropdown to select a drive

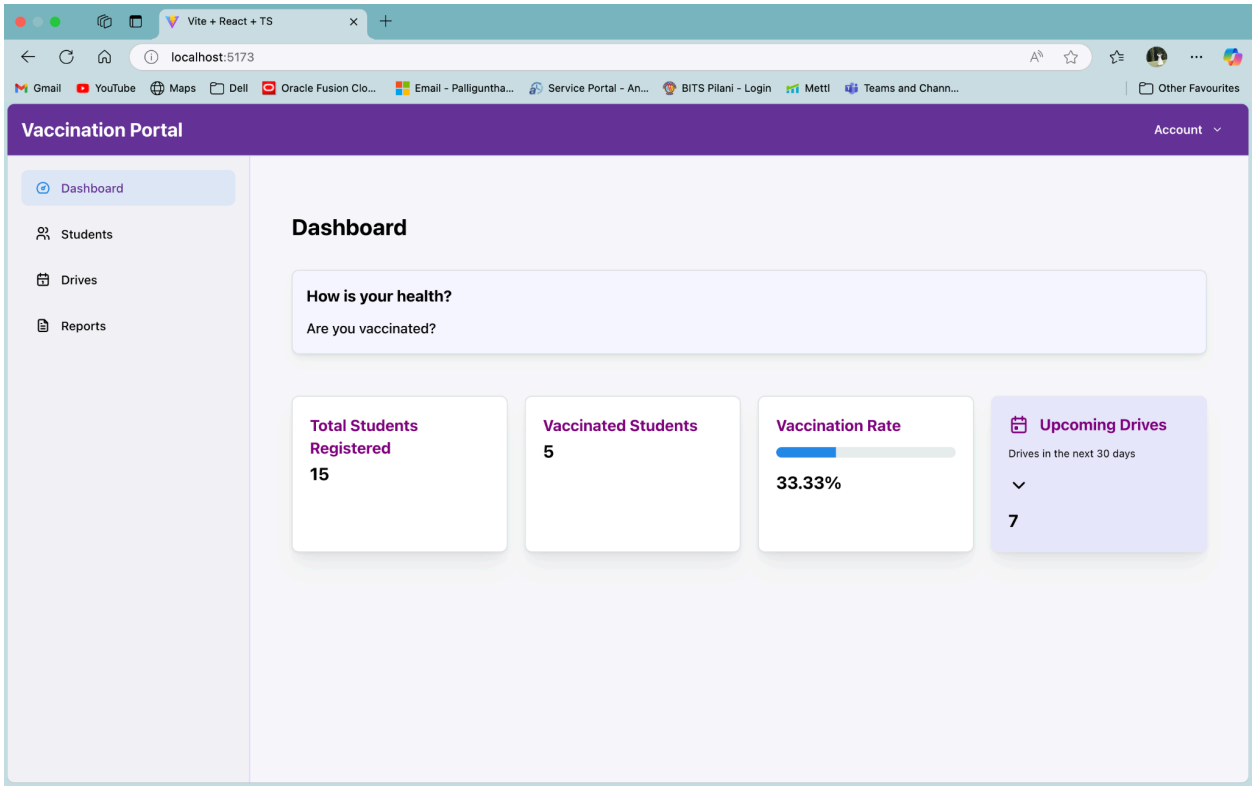
UI/UX Wireframes

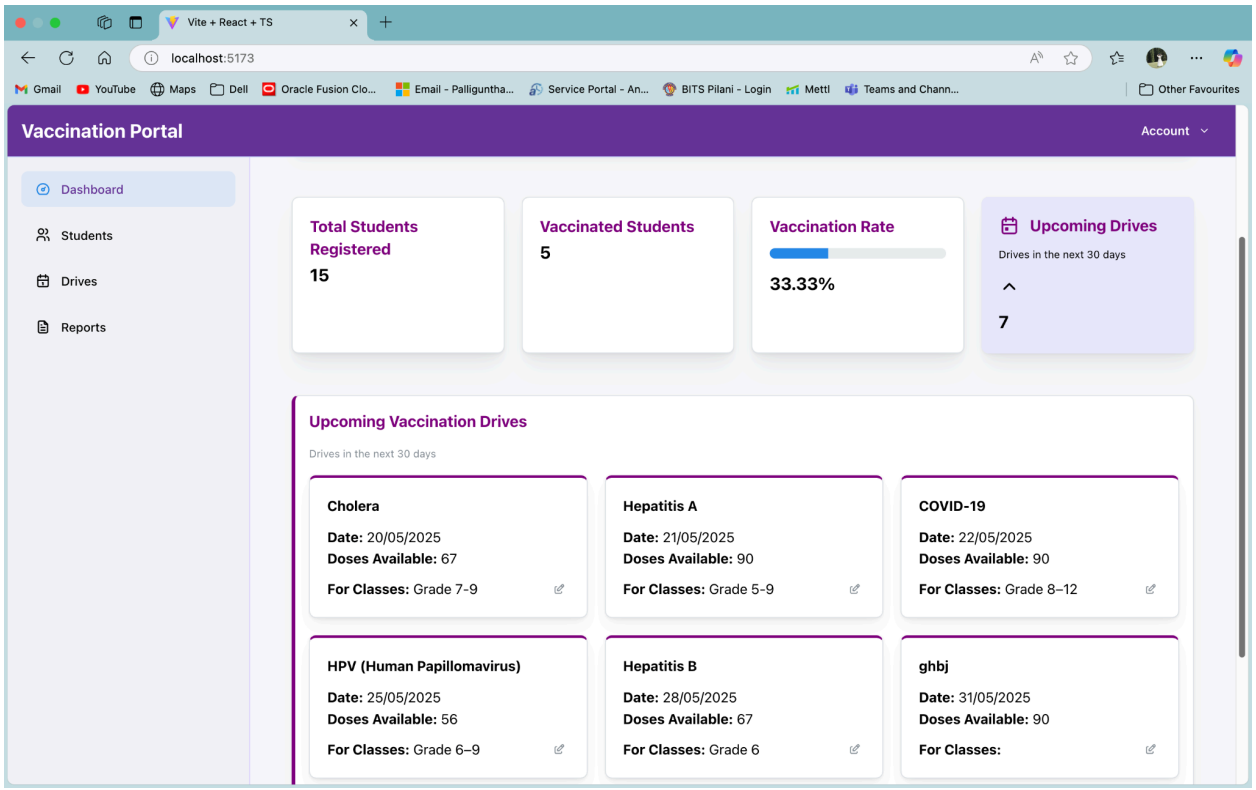
Login View:



Dashboard View:

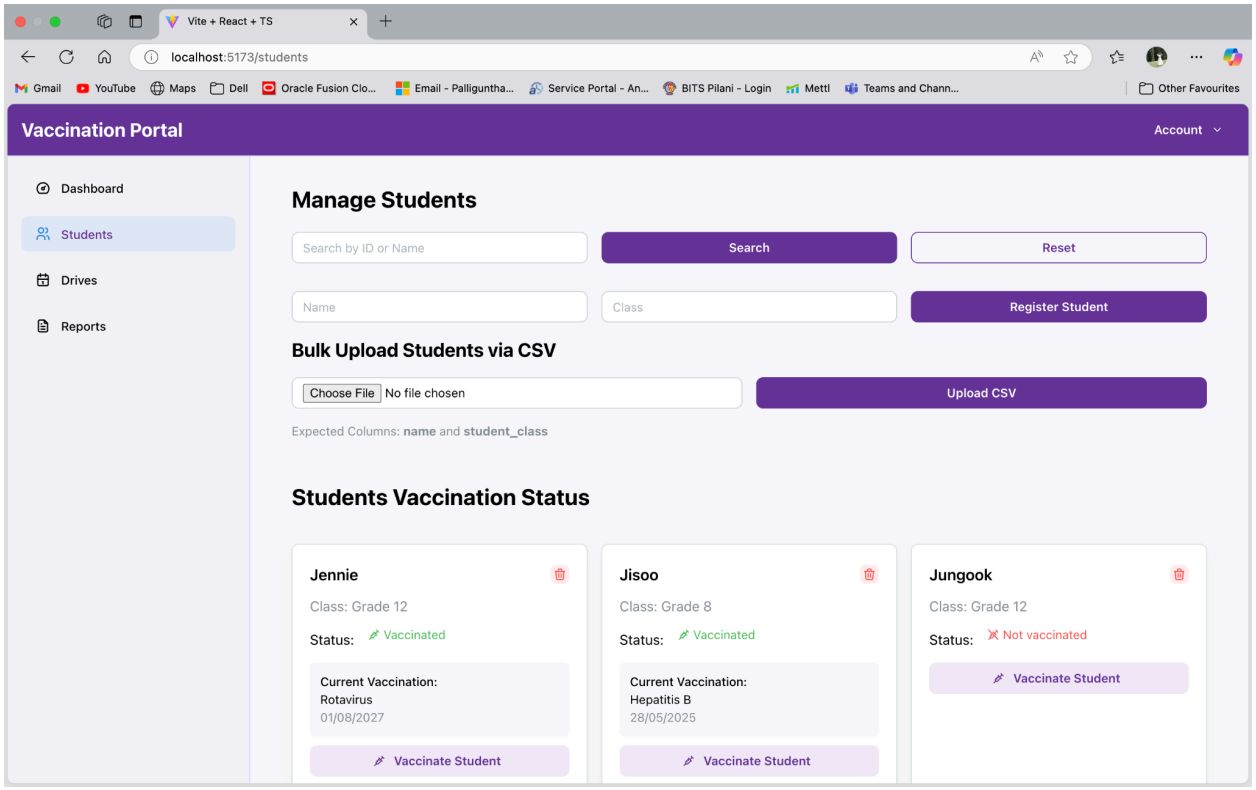
- *Dashboard showing key metrics and upcoming drives.*

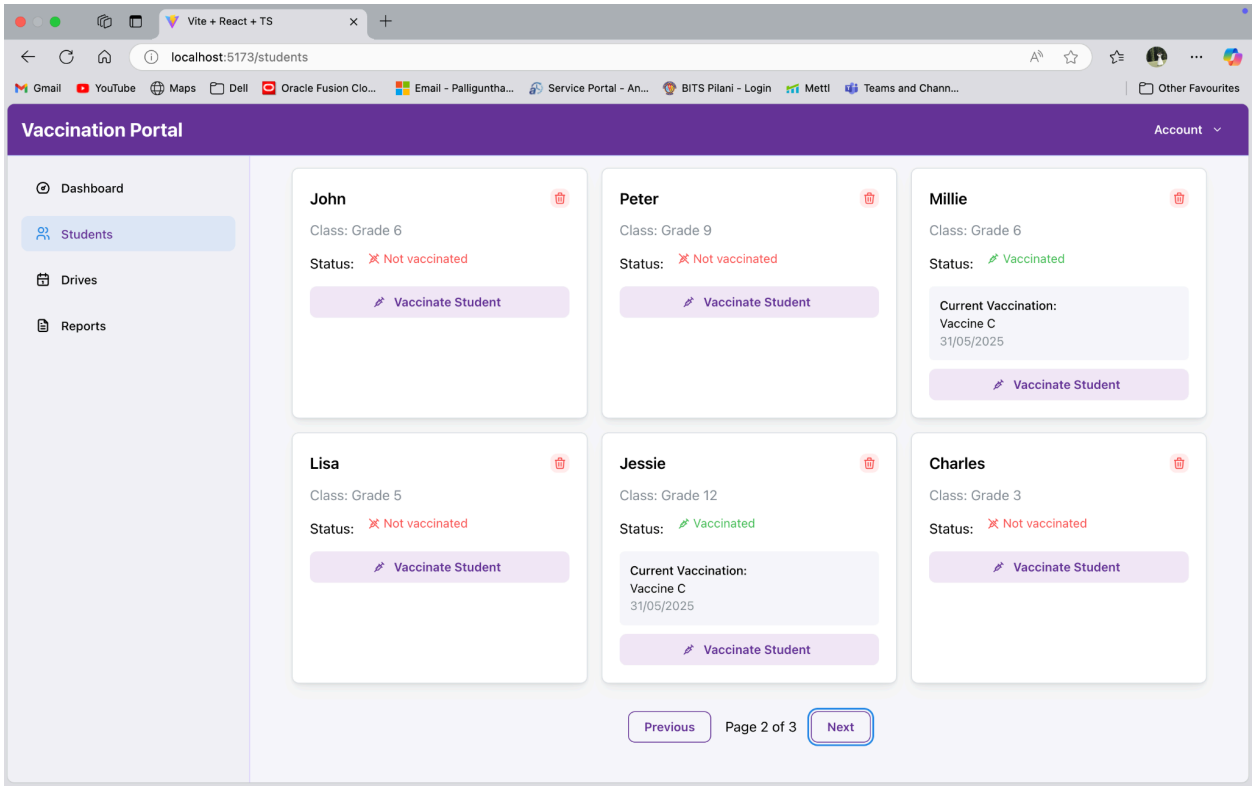




Students View:

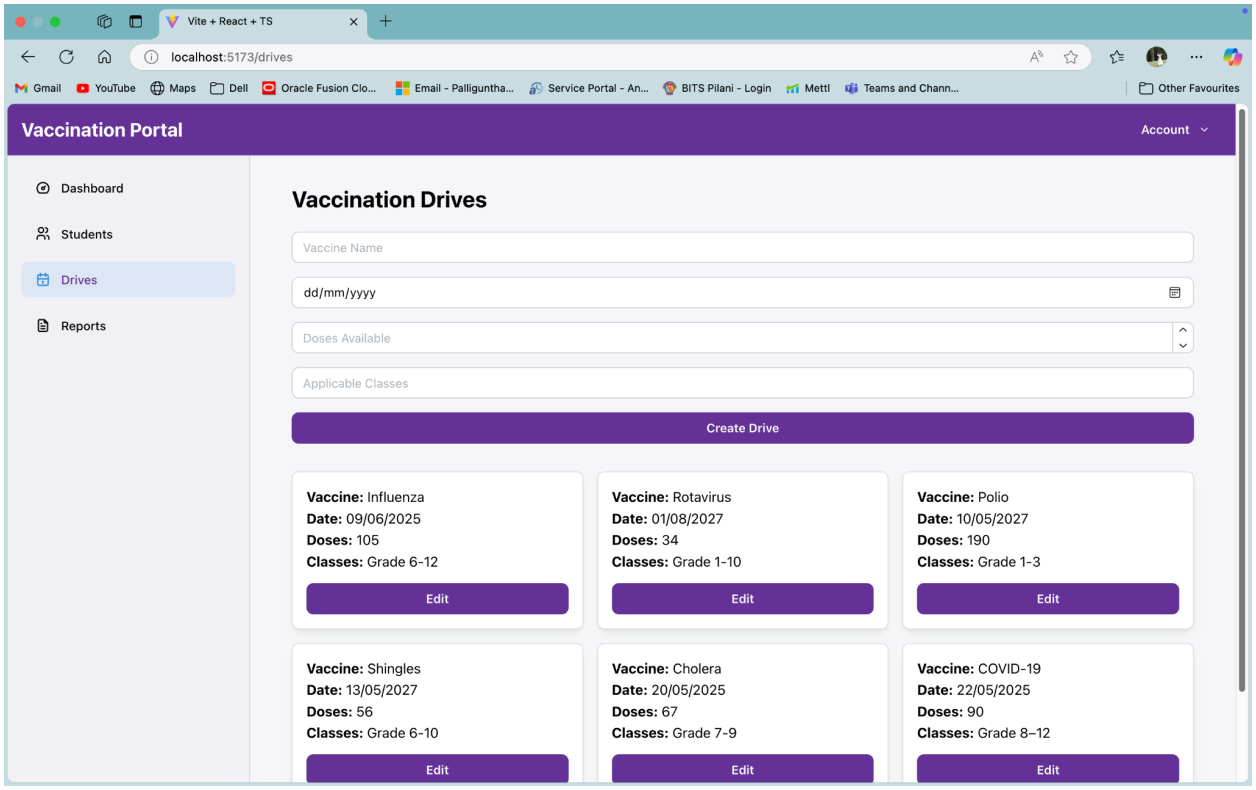
- *Student management interface with add form.*
- *Pagination applied for student details and vaccination status*



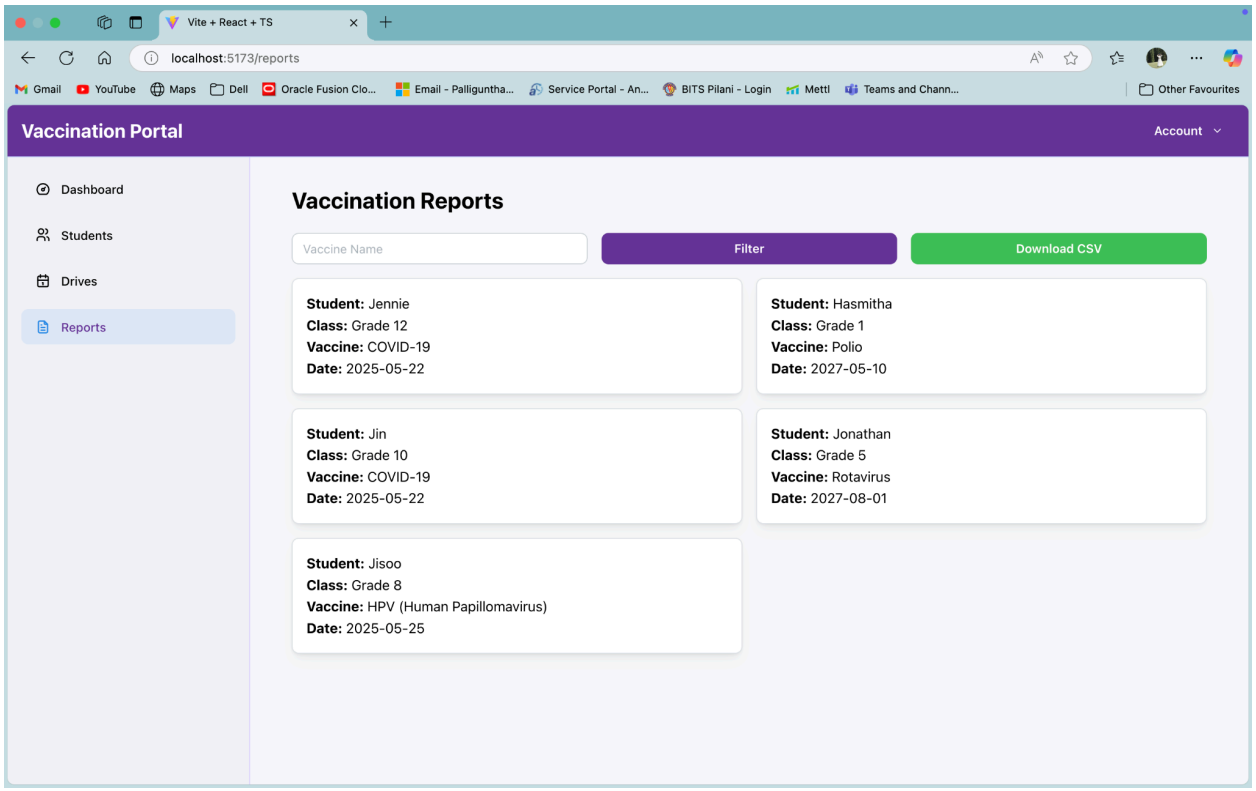


Drives View:

- *Vaccination drive management interface.*
- *Pagination applied.*

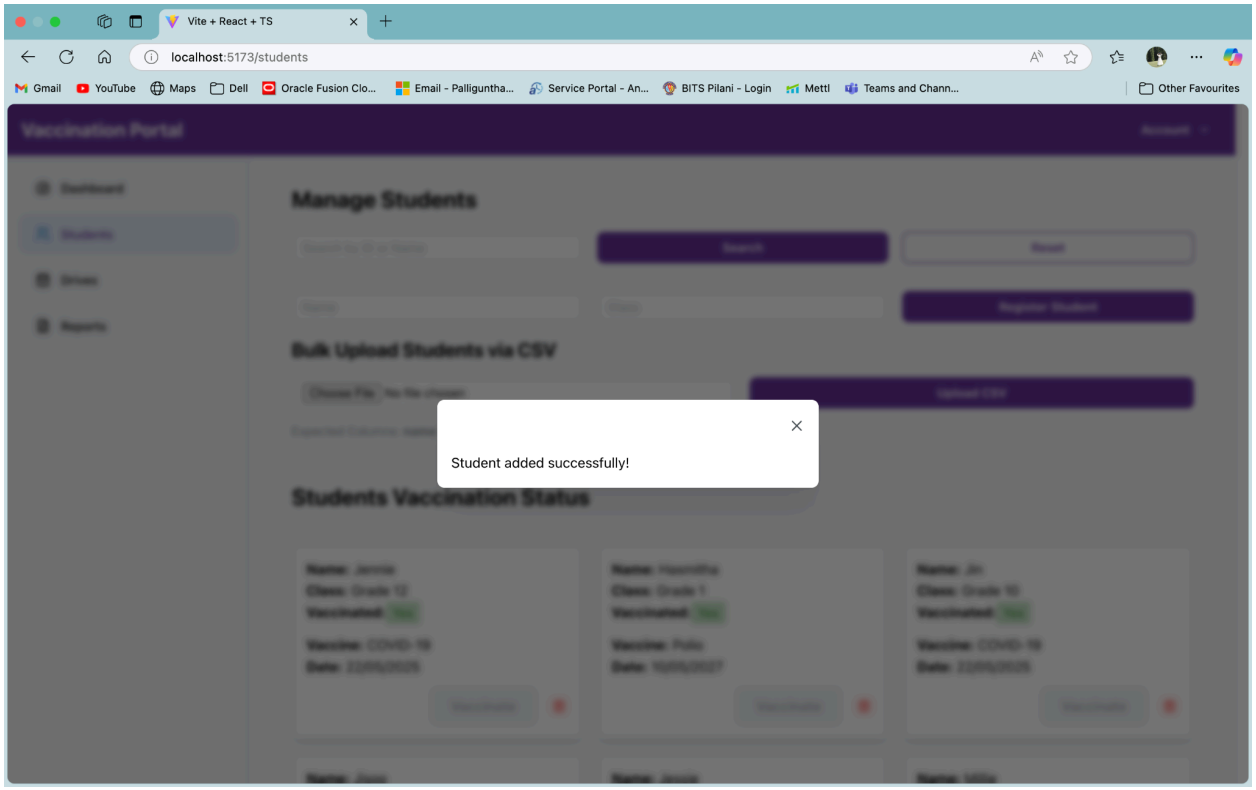


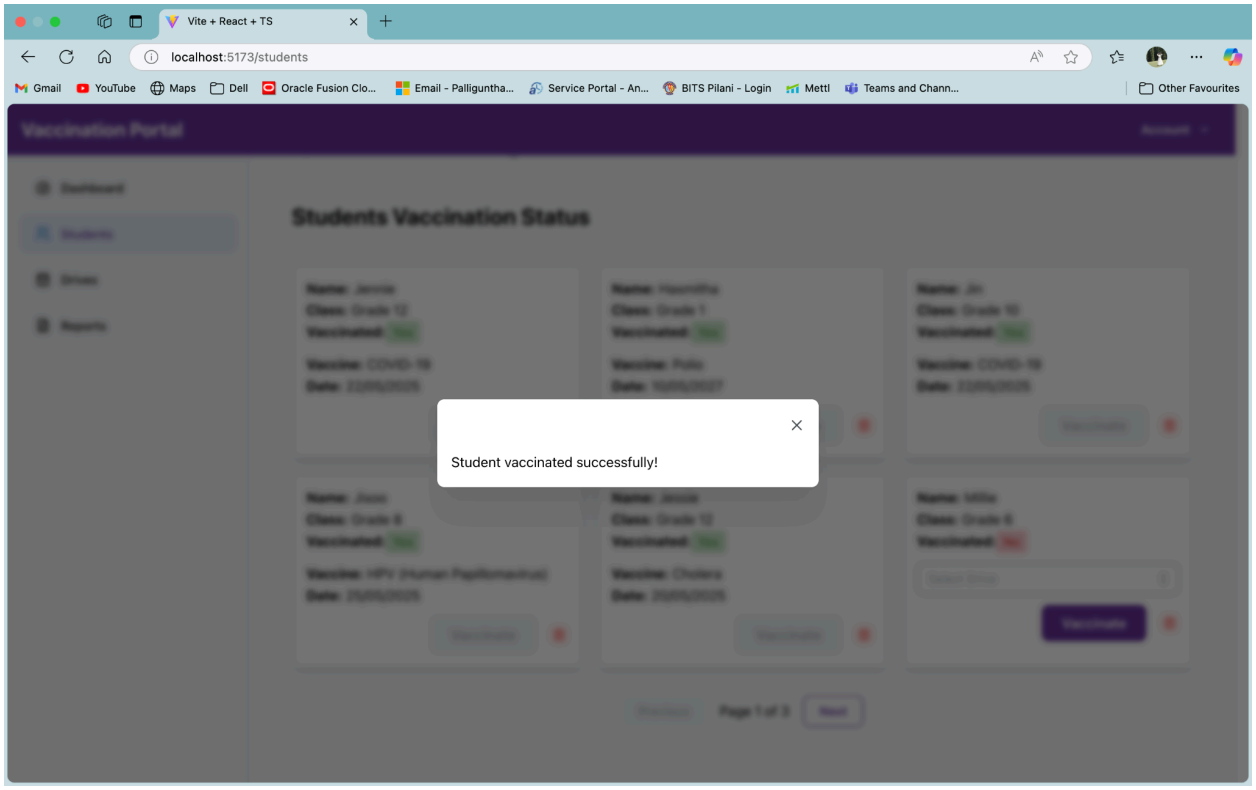
Reports View:

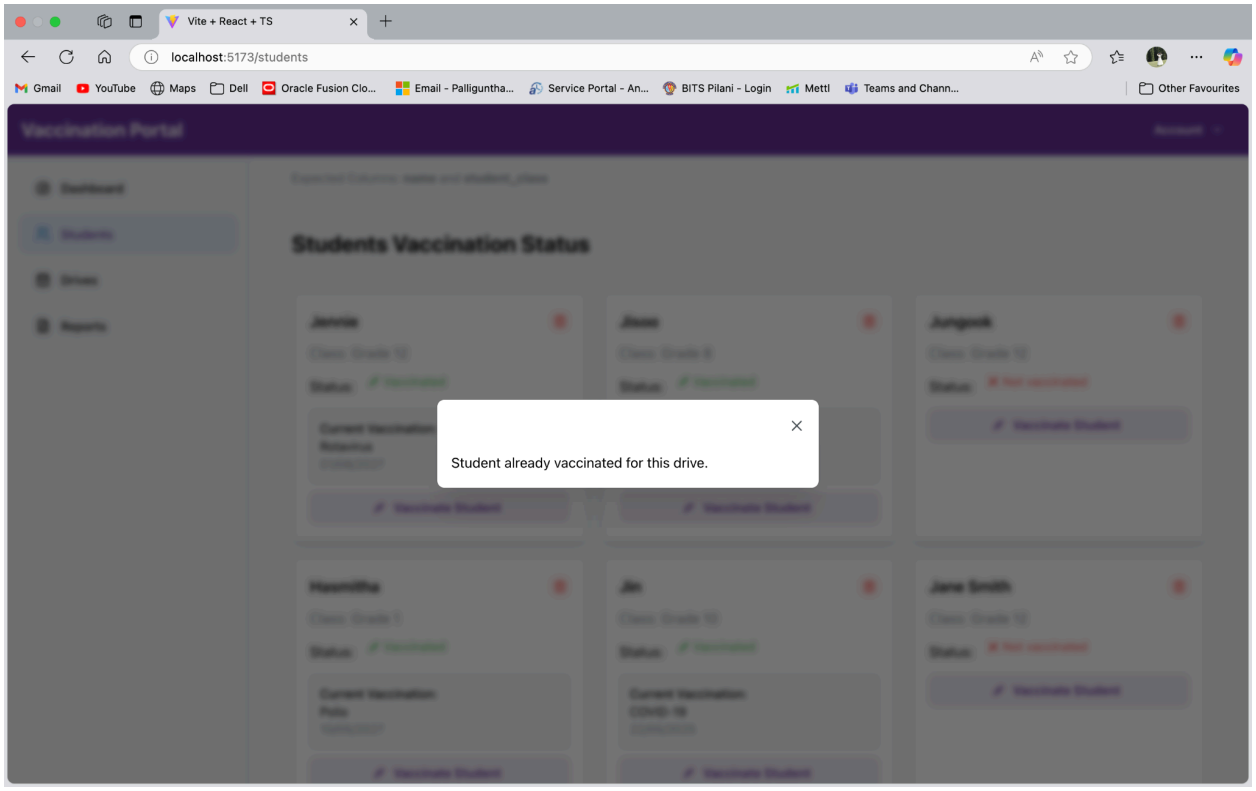


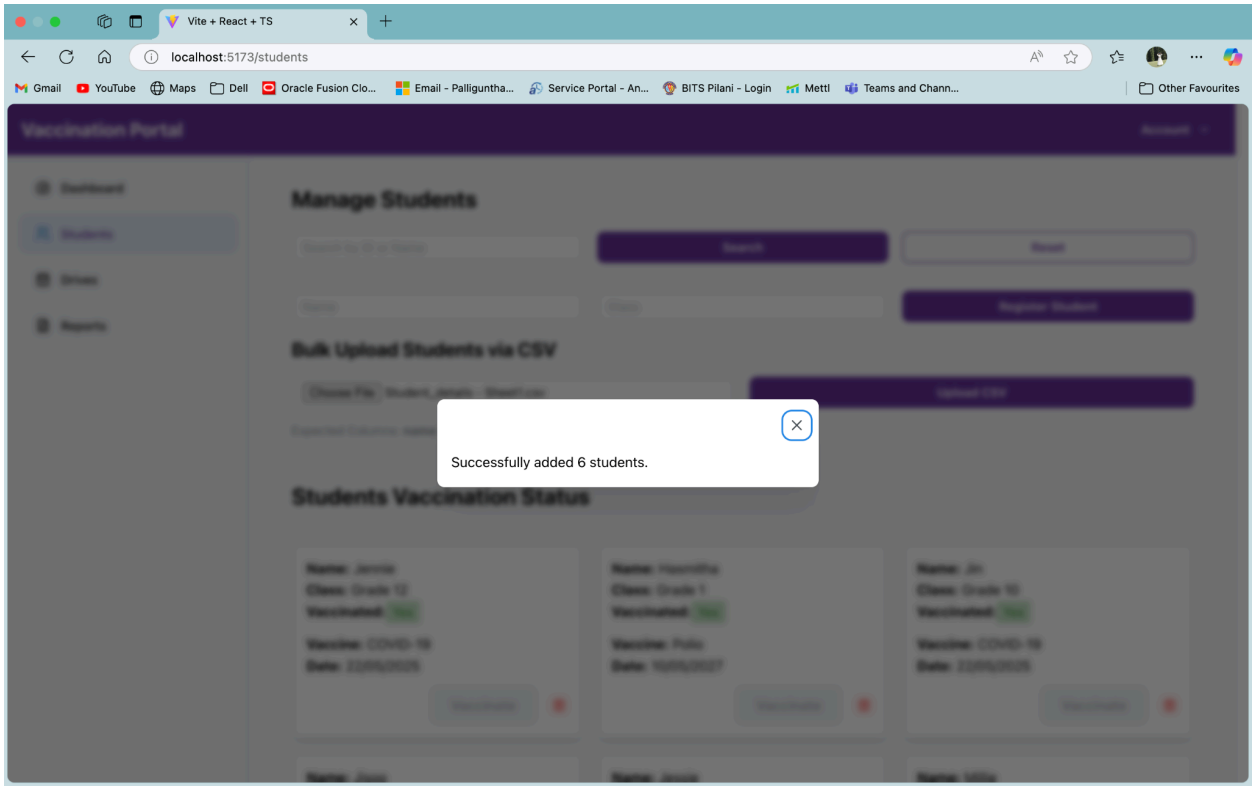
Output Snapshots

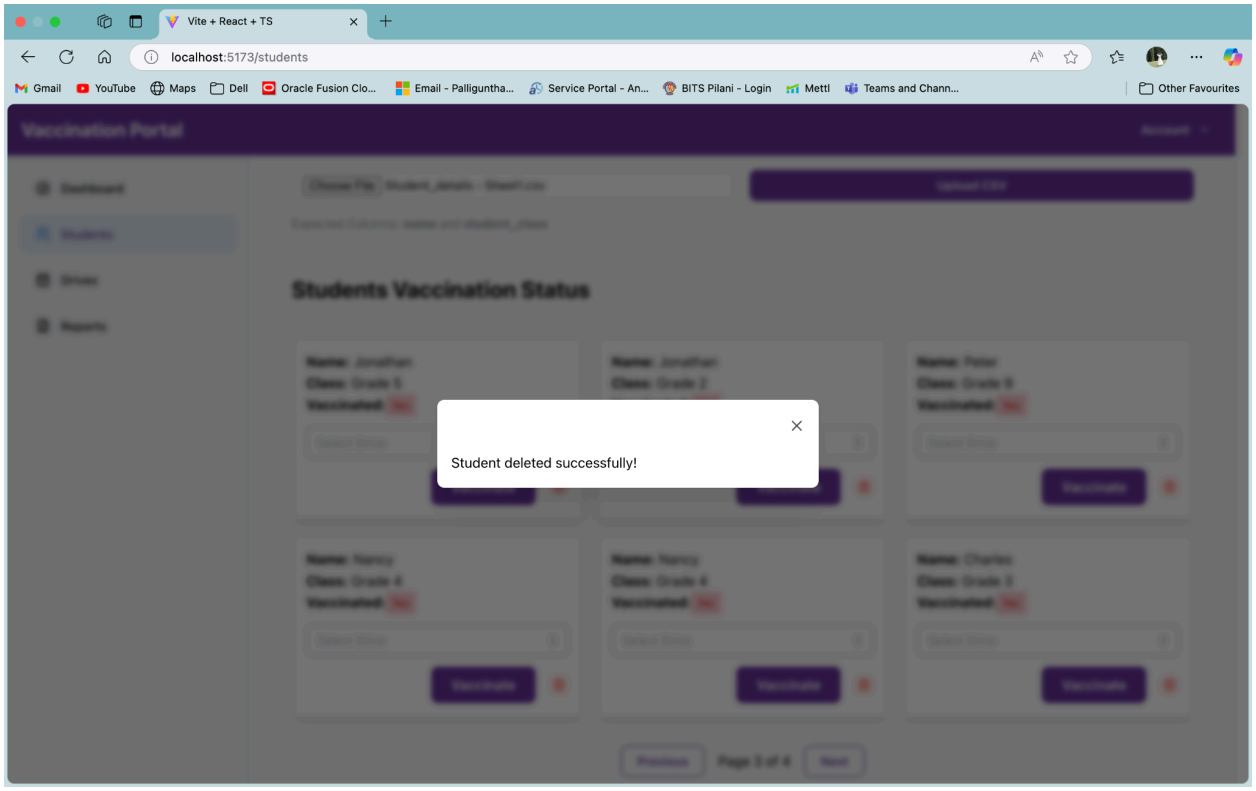
API Responses

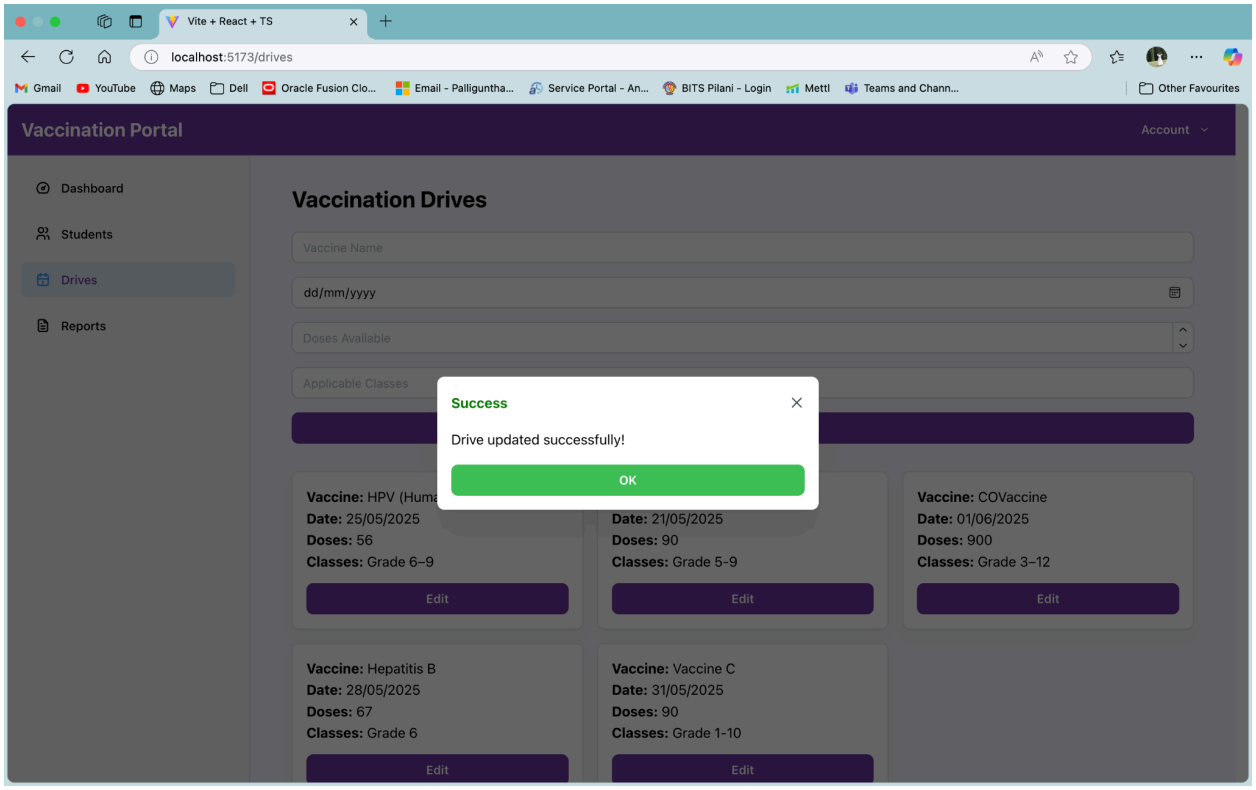


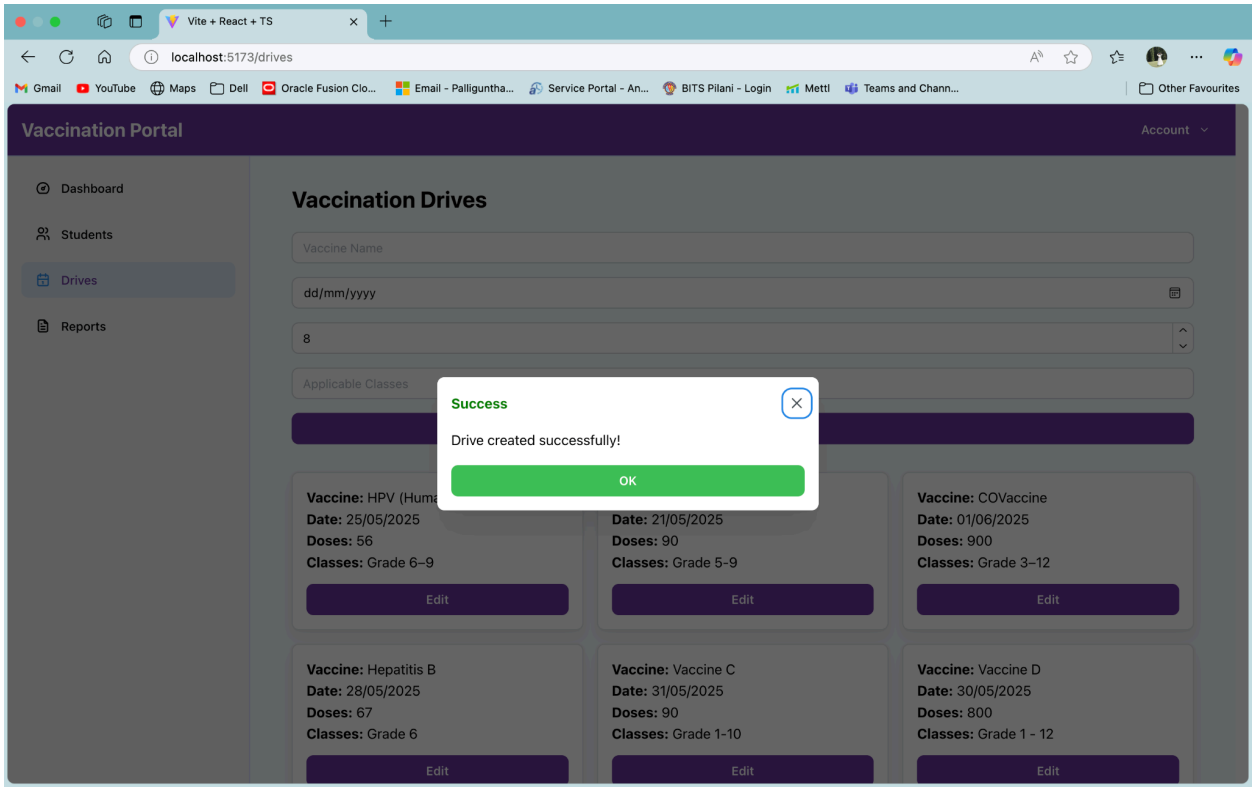

Vite + React + TS
localhost:5173/students
Gmail
YouTube
Maps
Dell
Oracle Fusion Clo...
Email - Palliguntha...
Service Portal - An...
BITS Pilani - Login
Mettl
Teams and Chann...
Other Favourites
Vaccination Portal
Students Vaccination Status
Student vaccinated successfully!

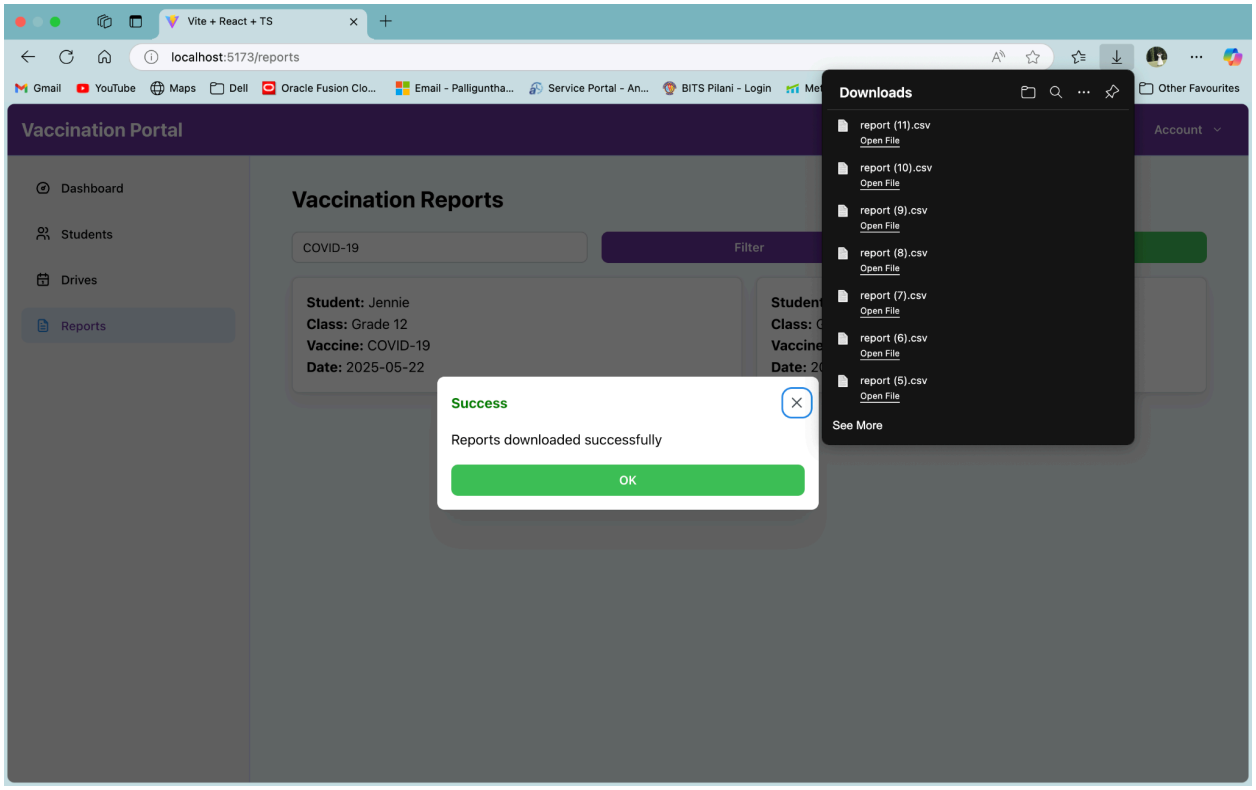

Vite + React + TS
localhost:5173/students
Gmail
YouTube
Maps
Dell
Oracle Fusion Clo...
Email - Palliguntha...
Service Portal - An...
BITS Pilani - Login
Mettl
Teams and Chann...
Other Favourites
Vaccination Portal
Students Vaccination Status
Student already vaccinated for this drive.

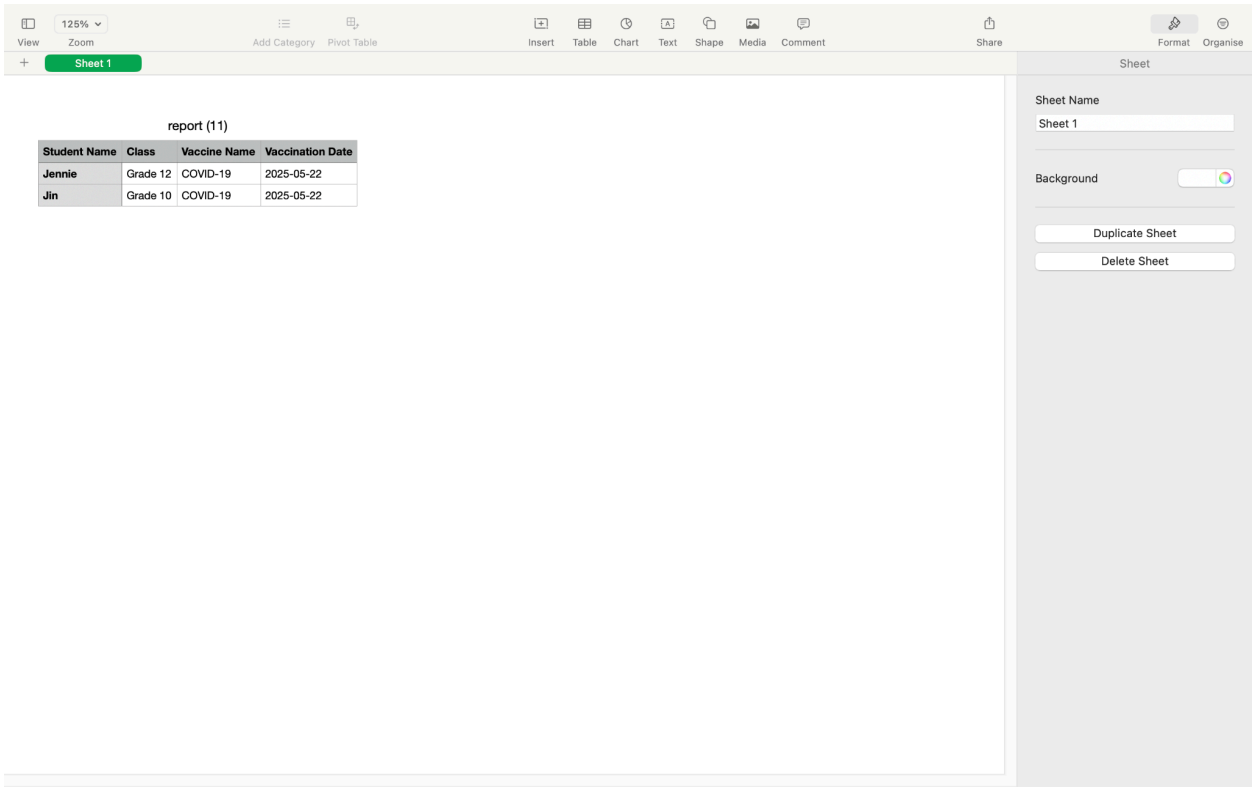

Vite + React + TS
localhost:5173/students
Gmail
YouTube
Maps
Dell
Oracle Fusion Clo...
Email - Palliguntha...
Service Portal - An...
BITS Pilani - Login
Mettl
Teams and Chann...
Other Favourites
Vaccination Portal
Manage Students
Bulk Upload Students via CSV
Successfully added 6 students.
Students Vaccination Status

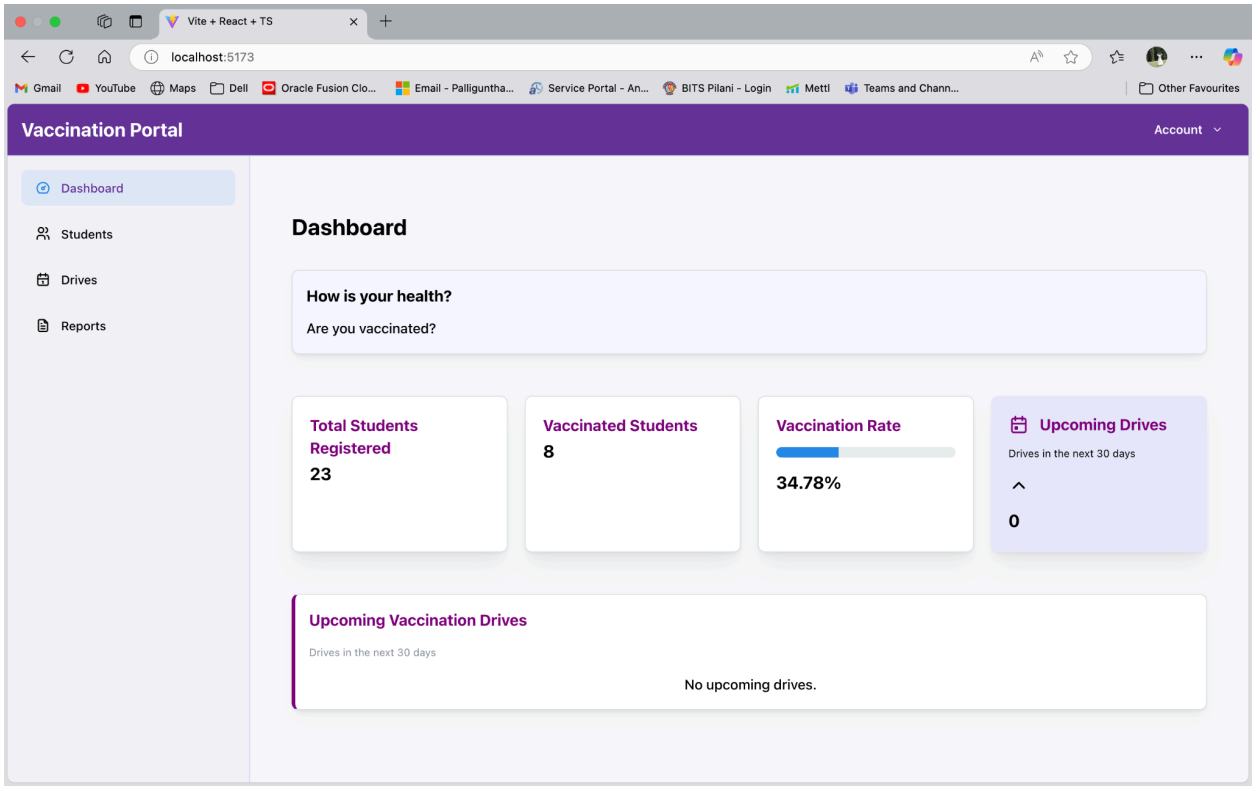

Vite + React + TS
localhost:5173/students
Gmail
YouTube
Maps
Dell
Oracle Fusion Clo...
Email - Palliguntha...
Service Portal - An...
BITS Pilani - Login
Mettl
Teams and Chann...
Other Favourites
Vaccination Portal
Students Vaccination Status
Student deleted successfully!


Vite + React + TS
localhost:5173/drives
Vaccination Portal
Account
Dashboard
Students
Drives
Reports
Vaccination Drives
Vaccine Name
dd/mm/yyyy
Doses Available
Applicable Classes
Success
Drive updated successfully!
OK
Vaccine: HPV (Huma
Date: 25/05/2025
Doses: 56
Classes: Grade 6–9
Edit
Date: 21/05/2025
Doses: 90
Classes: Grade 5-9
Edit
Vaccine: COVaccine
Date: 01/06/2025
Doses: 900
Classes: Grade 3–12
Edit
Vaccine: Hepatitis B
Date: 28/05/2025
Doses: 67
Classes: Grade 6
Edit
Vaccine: Vaccine C
Date: 31/05/2025
Doses: 90
Classes: Grade 1-10
Edit


Vite + React + TS
localhost:5173/drives
Vaccination Portal
Account
Dashboard
Students
Drives
Reports
Vaccination Drives
Vaccine Name
dd/mm/yyyy
8
Applicable Classes
Success
Drive created successfully!
OK
Vaccine: HPV (Huma
Date: 25/05/2025
Doses: 56
Classes: Grade 6–9
Edit
Date: 21/05/2025
Doses: 90
Classes: Grade 5-9
Edit
Vaccine: COVaccine
Date: 01/06/2025
Doses: 900
Classes: Grade 3–12
Edit
Vaccine: Hepatitis B
Date: 28/05/2025
Doses: 67
Classes: Grade 6
Edit
Vaccine: Vaccine C
Date: 31/05/2025
Doses: 90
Classes: Grade 1-10
Edit
Vaccine: Vaccine D
Date: 30/05/2025
Doses: 800
Classes: Grade 1 - 12
Edit

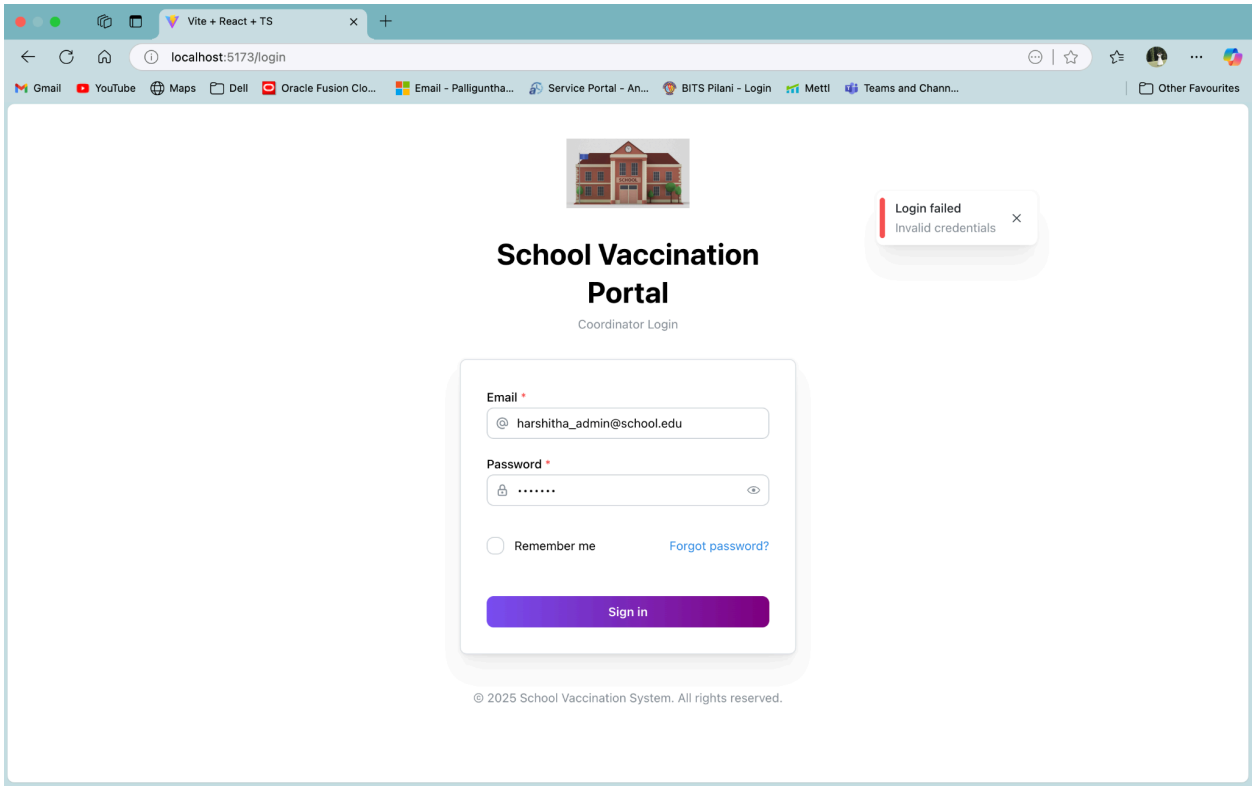

Vite + React + TS
localhost:5173/reports
Gmail
YouTube
Maps
Dell
Oracle Fusion Clo...
Email - Palliguntha...
Service Portal - An...
BITS Pilani - Login
Other Favourites
Vaccination Portal
Account
Dashboard
Students
Drives
Reports
Vaccination Reports
COVID-19
Filter
Student: Jennie
Class: Grade 12
Vaccine: COVID-19
Date: 2025-05-22
Success
Reports downloaded successfully
OK
Downloads
report (11).csv
Open File
report (10).csv
Open File
report (9).csv
Open File
report (8).csv
Open File
report (7).csv
Open File
report (6).csv
Open File
report (5).csv
Open File
See More


View
Zoom
Add Category
Pivot Table
Insert
Table
Chart
Text
Shape
Media
Comment
Share
Format
Organise
Sheet 1
Sheet
Sheet Name
Sheet 1
Background
Duplicate Sheet
Delete Sheet
report (11)
Student Name
Class
Vaccine Name
Vaccination Date
Jennie
Grade 12
COVID-19
2025-05-22
Jin
Grade 10
COVID-19
2025-05-22

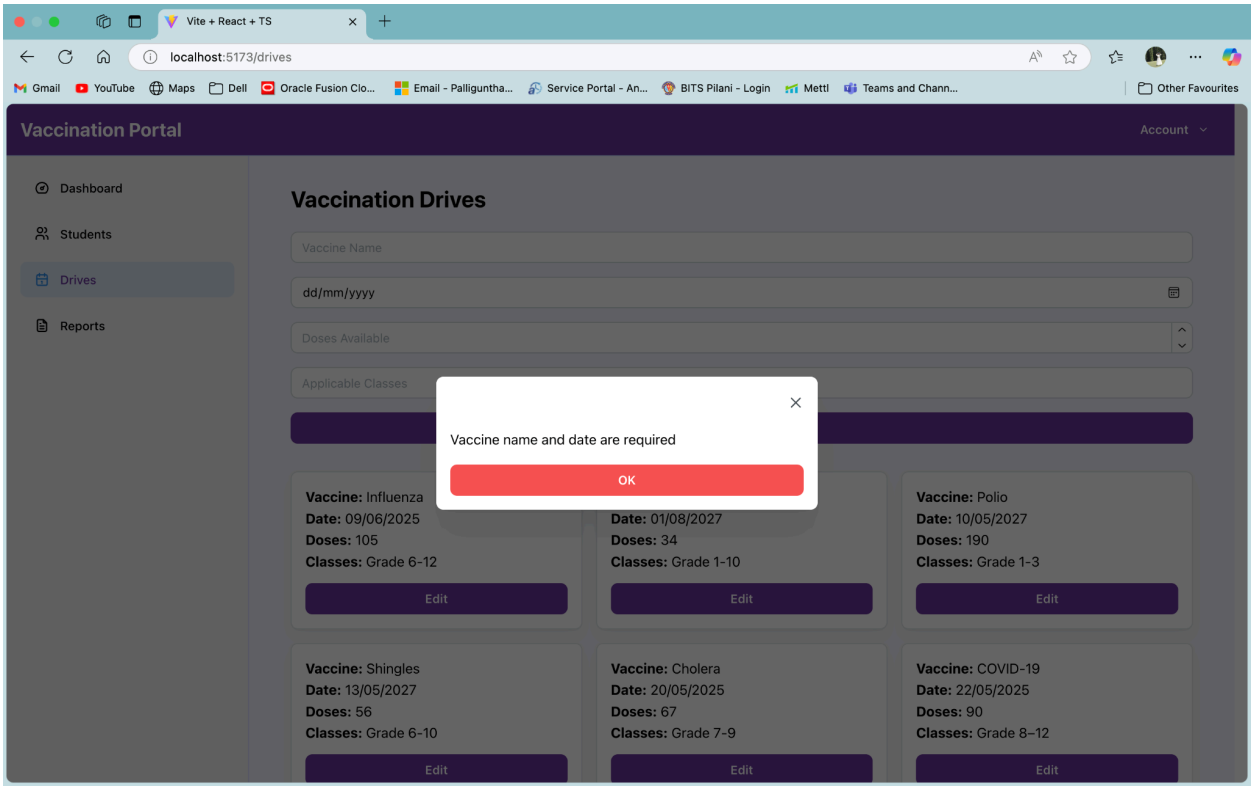


Error Response and Handling

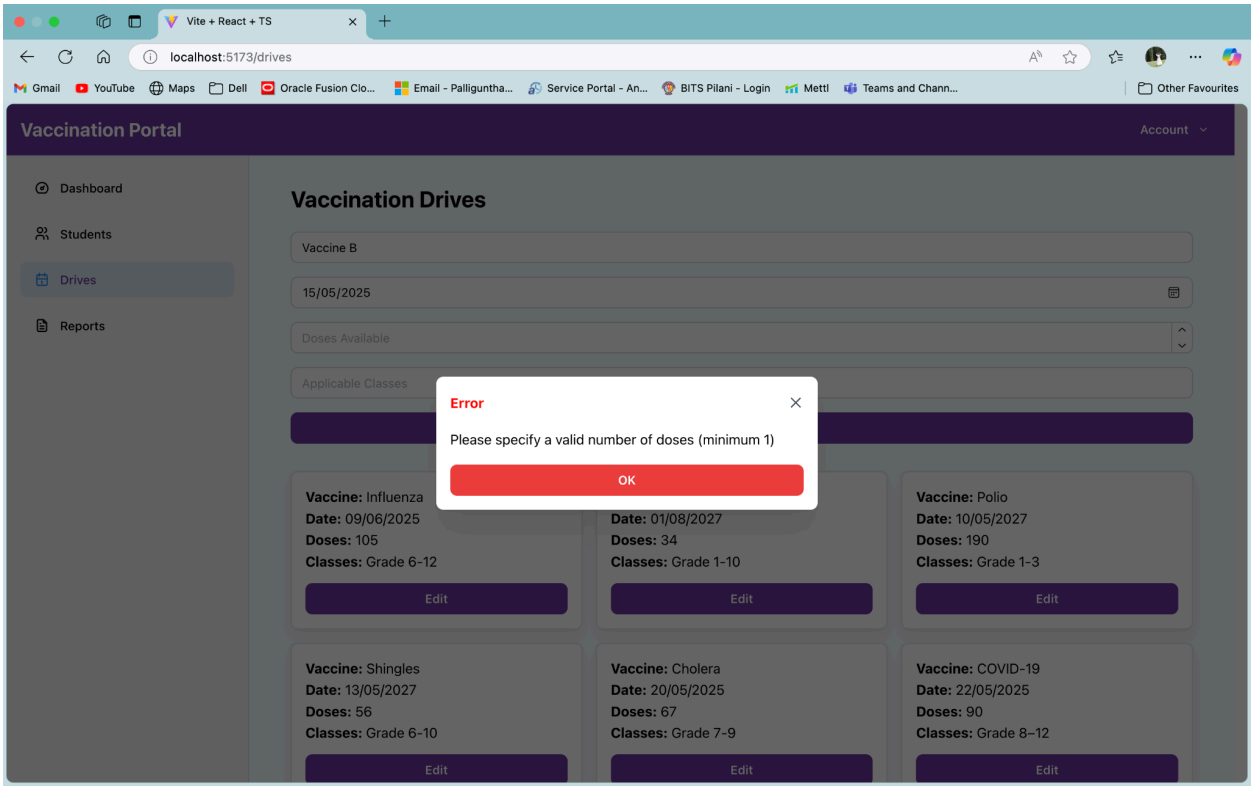
Invalid Login Credentials: When incorrect credentials are submitted on the login page, a clear "Login failed" error message is displayed to the user.



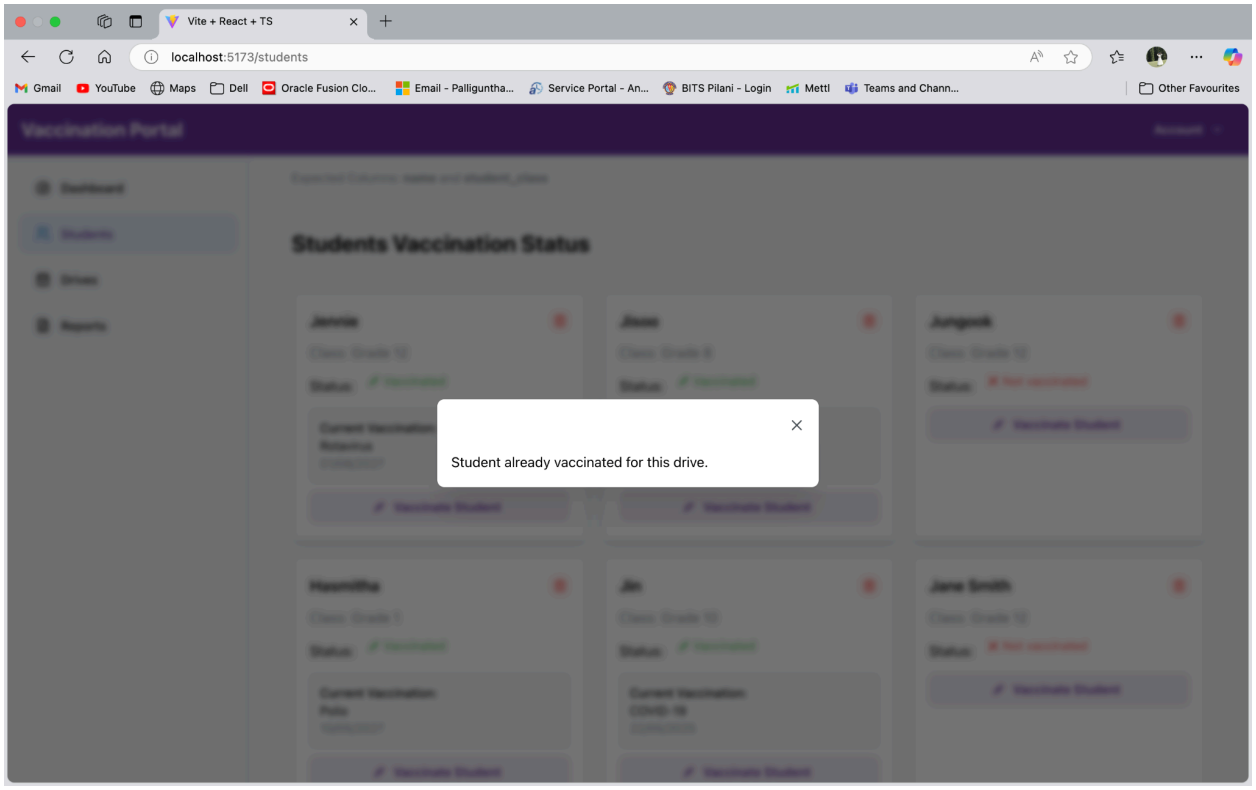
Missing Drive Details: Attempting to create a vaccination drive without specifying both the vaccine name and date triggers a validation error popup indicating that these fields are mandatory.



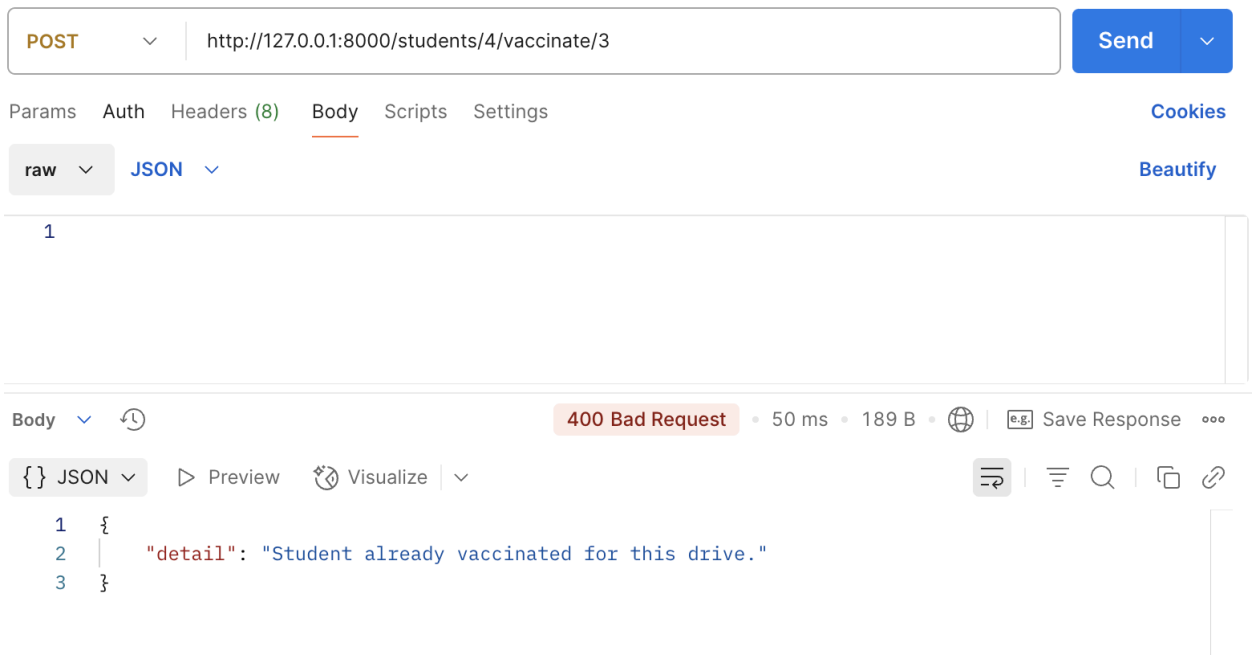
Missing Dose Count: If the number of available doses is not provided during drive creation, the system prompts the user with an error message: "Please specify a valid number of doses (minimum 1)".



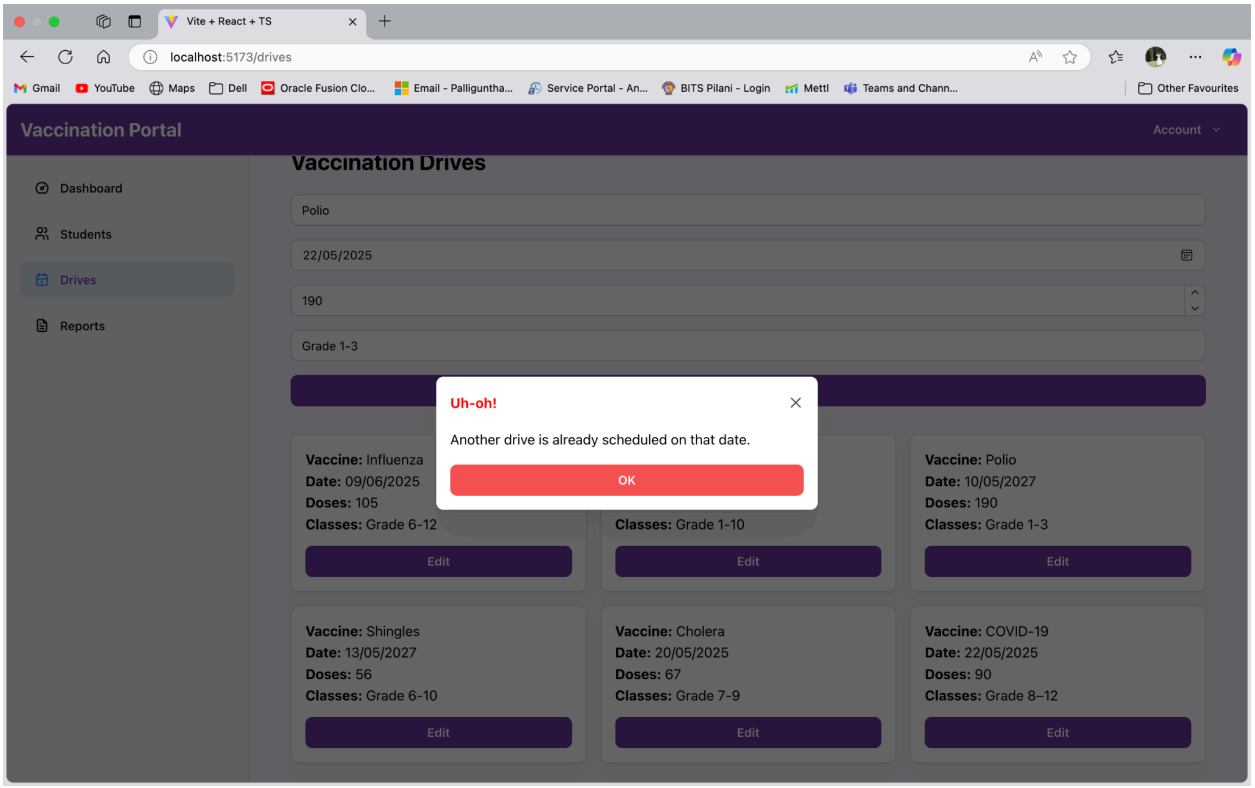
Duplicate Vaccination Prevention: For students who have already been vaccinated for a drive, the system disables the Vaccinate button for that vaccine to prevent multiple entries for the same student.



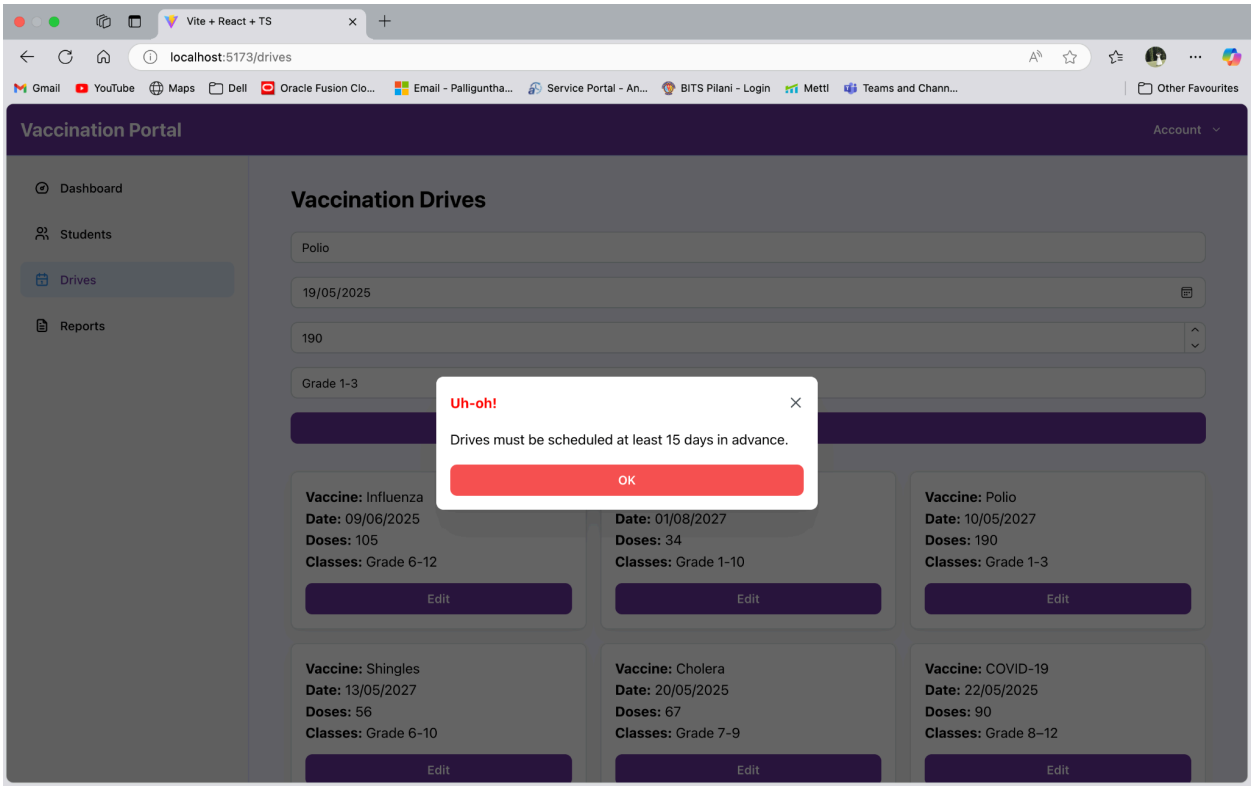
a



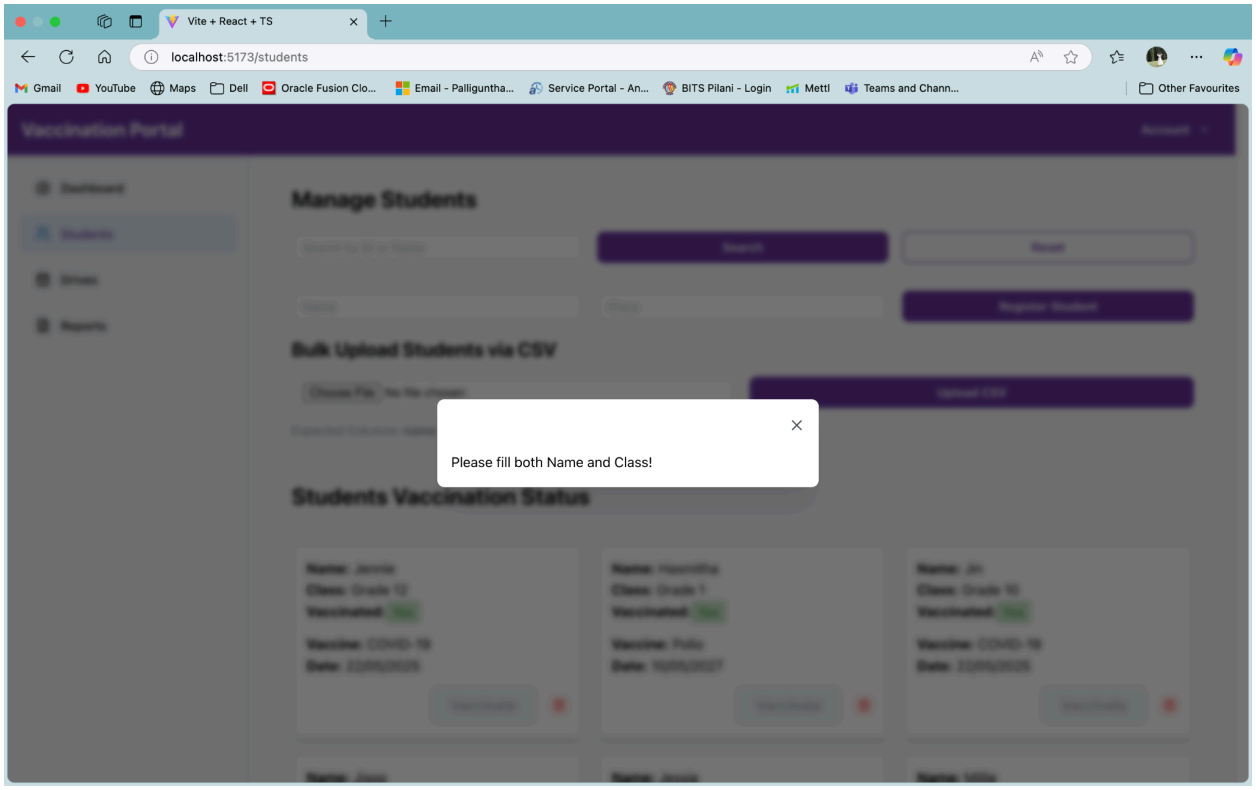
Date Conflict Check: System checks for existing drives on the same date for overlapping grade levels.Returns 409 Conflict with details: "Another drive is already on that date".



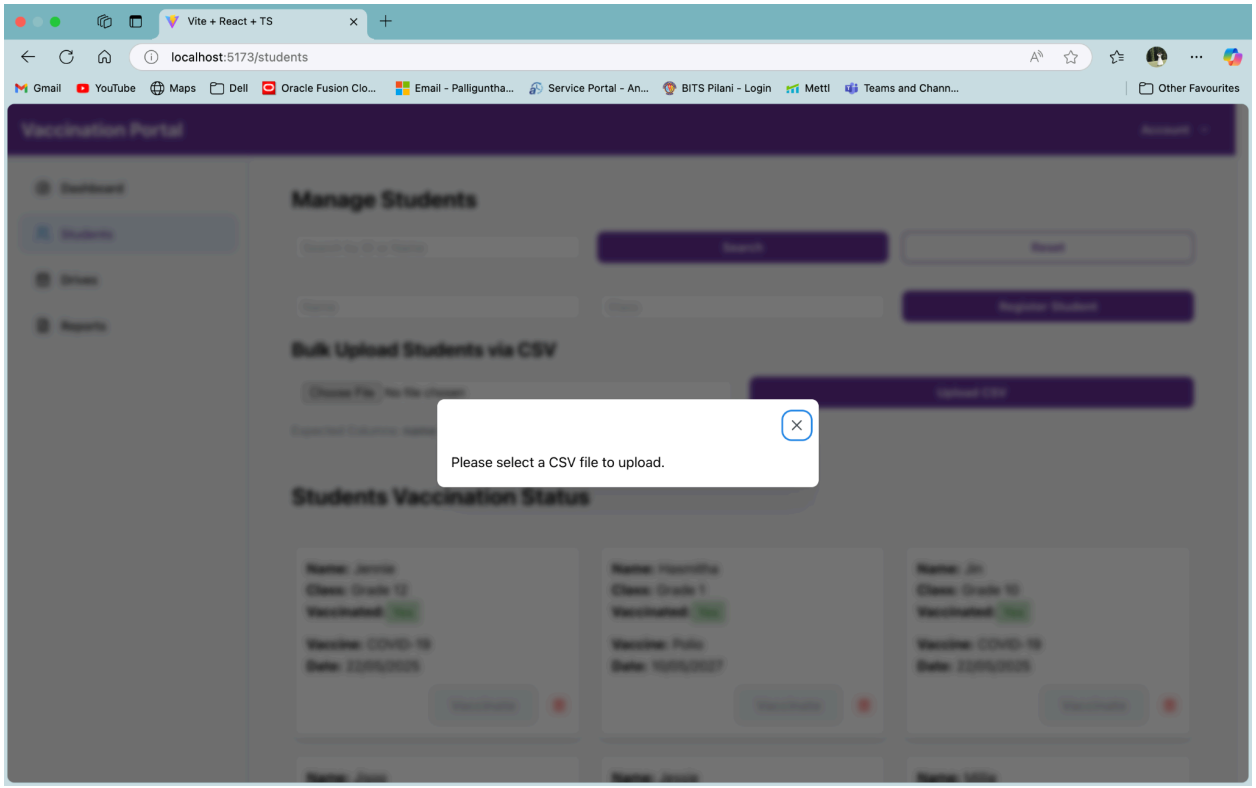
Scheduled Date check: Backend rejects dates fewer than 15 days from the current date.Returns 400 Bad Request with message: "Drives must be scheduled 15 days in advance".



Incomplete Student Entry: If an attempt is made to create a student record without entering both the student's name and class, an error message appears instructing the user to "Please fill in both name and class."



Missing File for Bulk Upload: When trying to submit the bulk CSV upload form without selecting a file, an error popup notifies the user: "Please select a CSV file."



Local Setup Instructions

Prerequisites

Ensure the following software is installed on your system:

- Node.js (v16 or above)
- Python (version 3.8 or above)
- PostgreSQL (version 12 or above)
- npm or yarn (Node.js package manager)

Backend Setup

1. Clone the Repository

- git clone <repository_url>
- cd <project_root>

2. Create and Activate Virtual Environment

```
# Create a virtual environment

python -m venv venv

# Activate the environment

# On Linux/Mac

source venv/bin/activate

# On Windows

venv\Scripts\activate
```

3. Install Backend Dependencies

- pip install -r requirements.txt

4. Set Up PostgreSQL

- Create a new PostgreSQL database named vaccination_db.
- Update the database connection string in config.py to point to this database.

5. Run Database Migrations

- alembic upgrade head

6. Start the Backend Server

- uvicorn app.main:app --reload

Once the server starts, the backend will be available at:

- Backend URL: http://localhost:8000
- API Documentation: http://localhost:8000/docs

Frontend Setup

1. Navigate to the Frontend Directory

- cd frontend

2. Install Frontend Dependencies

- npm install

3. Start the Frontend Development Server

- npm run dev

By default, the frontend will run at http://localhost:5173 or as specified in the terminal output.

Application URLs

Once both servers are running, access the application via:

- Frontend: http://localhost:5173
- Backend: http://localhost:8000
- API Docs: http://localhost:8000/docs

References

This project was developed using the following technologies, frameworks, and resources. Below are the official documentation and references that were consulted during development:

1. Frontend Technologies

- **React.js**
  - Official Documentation: https://react.dev/
  - Used for building the interactive user interface with component-based architecture.
- **Mantine UI**
  - Official Documentation: https://mantine.dev/
  - Provided pre-built, customizable components for a responsive and modern UI.
- **Axios**
  - GitHub Repository: https://github.com/axios/axios

    - Used for making HTTP requests from the frontend to the backend API.
- **React Router**
    - Official Documentation: https://reactrouter.com/
    - Enabled client-side routing for seamless navigation between views.

2. Backend Technologies

- **FastAPI**
    - Official Documentation: https://fastapi.tiangolo.com/
    - Used for building a high-performance RESTful API with automatic OpenAPI/Swagger documentation.
- **Uvicorn**
    - GitHub Repository: https://github.com/encode/uvicorn
    - ASGI server for running FastAPI in production.
- **SQLAlchemy**
    - Official Documentation: https://www.sqlalchemy.org/
    - ORM (Object-Relational Mapping) tool for interacting with PostgreSQL.
- **Alembic**
    - Official Documentation: https://alembic.sqlalchemy.org/
    - Database migration tool for managing schema changes.

3. Database

- **PostgreSQL**
    - Official Documentation: https://www.postgresql.org/docs/
    - Robust relational database used for storing and managing application data.
- **psycopg2**
    - Documentation: https://www.psycopg.org/docs/
    - PostgreSQL adapter for Python, used with SQLAlchemy for database operations.

4. Development & Testing Tools

- **Postman**

    - Official Website: https://www.postman.com/
    - Used for API testing and documentation.
- **Swagger UI**
    - GitHub Repository: https://github.com/swagger-api/swagger-ui
    - Integrated with FastAPI for interactive API documentation.
- **Docker**
    - Official Documentation: https://docs.docker.com/
    - Used for containerization and deployment.

5. Additional Resources

- **Mermaid.js** (For Diagrams)
    - Documentation: https://mermaid.js.org/
    - Used to generate flowcharts, sequence diagrams, and database schemas in documentation.
- **Chart.js** (For Data Visualization)
    - Official Documentation: https://www.chartjs.org/
    - Used for displaying vaccination statistics on the dashboard.

Attributions

This project was developed by referring to the official documentation of the above technologies. Special thanks to the open-source communities for providing these powerful tools.

Github link : Student_Vaccination_GitRepo