Group Project Development Final Report

USTH Academic Suite

Vietnam France University

Group 28 – ICT

Members

Nguyen Minh Quan	22BA13258
Nghiem Vy Son	22BA13273
Pham Nhat Minh	23BI14298
Tran Quang Huy	23BI14196
Pham Quang Hung	22BA13156
Nguyen Long	22BA13203
Nguyen Trung Duong	22BA13095

Supervisor

Dr. Kieu Quoc Viet

ICT Lab

January, 2026

Contents

List of Figures

1. Introduction and Motivation

In modern universities, academic management is no longer a simple administrative task. With the increasing number of students, classes, programs, and learning activities, academic assistants are required to handle large volumes of data every day, ranging from student records and class schedules to room all location, requests for absence, and real-time announcements. However, many of these activities are still carried out using fragmented tools such as Excel spreadsheets, emails, Google Forms, and informal chat groups. Although these tools may work for small-scale management, they quickly become inefficient, inconsistent, and difficult to maintain as the system grows.

These challenges motivated us to develop the "Academic Suite – Student and Academic Management System", a centralized web-based platform designed specifically for academic assistants. The system aims to provide a single interface for managing users, subjects, classes, rooms, schedules, and student requests while supporting real-time notifications and data analysis. Instead of relying on scattered documents and communication channels, all academic activities can be managed consistently within an integrated environment.

The main objective of this project is not only to digitize the existing work flow but also to improve it by introducing automation, data integrity, and real time interaction. By adopting modern web technologies such as React, Firebase Authentication, and a serverless PostgreSQL database, the Academic Suite is designed to be scalable, reliable, and easy to maintain. Ultimately, this system helps academic assistants reduce administrative workload, minimize human errors, and focus more on improving the quality of academic services provided to students.

The Academic Suite is expected to significantly improve the efficiency and reliability of academic management activities at USTH. By centralizing all academic data into a single platform, the system reduces manual data duplication and minimizes the risk of inconsistencies across different departments. The integration of real-time notifications ensures that academic assistants are promptly informed about new requests and schedule updates, leading to faster response times and better communication with students.

2 Overall System Requirements

Our Academic Suite is a web-based academic management system designed to support academic assistants in managing students, classes, schedules, rooms, and academic requests through a centralized interface.

2.1 Software Requirements

Component	**Requirement**
Runtime Environment	Node.js version 18 or higher
Version Control System	Git
Code Editor	Visual Studio Code/Notepad++
Database Service	Neon PostgreSQL account
Authentication Service	Firebase project
Realtime Data Service	Firebase Firestore

Table 1: Software Requirements for Academic Suite

The software requirements presented in Table 1 describe the essential components needed to operate the Academic Suite. Node.js is required to run both frontend and backend services, while Git supports source code management and collaboration. A text editor, such as Visual Studio Code or Notepad++, is necessary for editing environment files

and maintaining the system. Neon PostgreSQL is used as the main relational database to store academic data, and Firebase provides secure user authentication. In addition, Firebase Firestore enables real-time notifications, allowing academic assistants to receive instant updates during daily operations.

2.2 Hardware Requirements

Component	**Requirement**
Memory (RAM)	4 GB of RAM (required), 8 GB recommended
Disk Storage	At least 2 GB of free disk space
Processor (CPU)	Any modern multi-core processor capable of running a web browser and Node.js.
Internet Connection	Stable internet connection

Table 2: Hardware Requirements for Academic Suite

The hardware requirements in Table 2 define the minimum system resources needed to run the Academic Suite smoothly. A minimum of 4 GB of RAM is required to support browser-based operations, while 8 GB is recommended to ensure better performance when handling multiple tasks simultaneously. At least 2 GB of free disk space is sufficient for storing the project files and dependencies because the system mainly operates in a web-based environment. Any modern multi-core processor is adequate for running the application, as most processing tasks are handled on the server side. Finally, a stable internet connection is essential since the system relies on cloud services such as Neon Database and Firebase for data storage and authentication.

3. Design Analysis

This section presents the design analysis of the Academic Suite system. It describes the main user roles, business processes, and use cases that define how academic assistants interact with the system. By analyzing these use cases, the core functional requirements are identified and translated into architectural decisions, which serve as the foundation for the system design described in the following sections.

3.1. Functional Requirements

This subsection outlines the core functional requirements of the Academic Suite system. These requirements define the specific actions the system must support for each user role, ensuring that the system meets the needs of administrators, academic assistants, lecturers, and students. The requirements are broken down into different functional areas such as account management, course management, and learning management, detailing the user interactions and system responses necessary to facilitate efficient academic operations.

3.1.1. Account Management

- **Login:** Authenticate users using email and password.
- **View Profile:** Display personal information (full name, student ID, major) retrieved from the user_profile or User table.

- **Logout:** Log out the user, clear local session/token, and redirect to the Login screen.

3.1.2. Course Management (Market)

- **View All Subjects:** Display a list of all available subjects in the system.
- **View Subject Details:** Show subject description, number of credits, and assigned lecturer.
- **Enrollment Check:** Automatically check whether the student has already enrolled in a subject to prevent duplicate enrollment.
- **Auto Enroll:** Automatically enroll the student in a subject (the server selects an available class for the subject and assigns the student).

3.1.3 Learning Management

- **My Courses:** View the list of subjects the student is currently enrolled in.
- **View Attendance:** View attendance history for each subject (date, check-in time, status: Present/Absent).
- **View Grades:** View the student's grades (retrieved via the API /api/grades/{studentId}).

3.1.4. Schedule Management

- **View Schedule:** Display upcoming class schedules, including start time, end time, and room.

3.2 Non-Functional Requirements

3.2.1. Performance

- Asynchronous Processing: The application uses Retrofit with callback-based asynchronous requests on Android to call APIs without blocking the UI (Main Thread).
- Query Optimization: The server utilizes SQL JOIN queries to retrieve aggregated data in a single request instead of multiple calls (e.g., the /api/subjects API returns lecturer information and enrollment status simultaneously).

3.2.2. Usability (UI/UX)

- BottomSheetDialog: Bottom Sheets are used for the *Course Detail* feature, allowing users to quickly view information without fully navigating to another screen, resulting in a smoother user experience.
- Visual Feedback:
  - Action buttons change color (Green → Gray) after successful enrollment.
  - Attendance status is color-coded (Green for *Present*, Red for *Absent*) for easy recognition.
- Navigation: The application uses a DrawerLayout (side menu) combined with flexible Fragment transitions to provide intuitive navigation.

3.2.3 Reliability & Data Integrity

- Constraint Checks: The database enforces foreign key constraints between enrollment, class, user and tables to ensure data consistency (e.g., preventing enrollment in non-existent classes).
- Error Handling: Both the server and client implement error-handling mechanisms (try–catch blocks and onFailure callbacks) to handle network or server errors, with user-friendly notifications displayed via Toast messages.

3.2.4. Security

- Role-Based Access Control (RBAC): The server distinguishes between *Lecturer* and *Student* roles when serving schedule data to ensure appropriate access.
- Data Transport: Retrofit is used to support secure HTTPS communication, depending on the server configuration (process.env.DATABASE_URL with SSL settings such as rejectUnauthorized: false).

3.2.5. Compatibility

- Android: The application is developed as a native Android application.
- Database: The system is compatible with PostgreSQL, specifically using NeonDB.

3.3. Business Processes

This section describes the main business processes supported by the Academic Suite. A business process refers to a set of activities carried out by users during everyday operations and demonstrates how the system supports and improves these activities.

3.3.1 Academic Assistant Daily Workflow

The Academic Assistant plays a central role in managing academic activities. A typical workday starts with signing into the Academic Suite through the authentication system. Once signed in, the assistant accesses a centralized dashboard displaying schedules, pending requests, and recent updates.

One of the main tasks of the academic assistant is managing class schedules. The assistant creates or updates schedules by assigning subjects, rooms, and time slots. The system supports this process by storing all scheduling information in a centralized database and minimizing conflicts through structured data management.

Throughout the day, the assistant receives student requests, such as absence reports or questions related to schedules. These requests are submitted through the system and appear immediately on the assistant's dashboard. The assistant reviews each request and either approves or rejects it. Once a decision is made, the system automatically notifies the student, removing the need for manual communication through emails or messaging apps.

In addition, the academic assistant manages class information and keeps track of room availability. When changes occur, such as rooms becoming un available or scheduling adjustments, the system ensures that all related

records are updated consistently. Real-time notifications further support the assistant by providing immediate alerts on new activities or changes.

3.3.2 Administrator Management Process

The Administrator role is responsible for managing master data within the system, including users, subjects, rooms, and academic structures. Administrators would typically configure the system by creating subjects, defining rooms, and managing user accounts. These actions ensure that academic assistants can carry out their daily operations efficiently with accurate and up-to-date data.

3.3.3 Student Interaction Process

Students interact with the system mainly to view academic information and submit requests. They can view their schedules, track academic updates, and submit academic-related requests. The system records these requests and forwards them to academic assistants for review. This organized process improves transparency and reduces reliance on informal communication channels.

3.3.4 Lecturer Interaction Process

Lecturers use the system to access teaching-related information, such as class schedules and assigned rooms. While they do not perform administrative tasks, the system supports their workflow by providing accurate and up-to-date academic information.

3.4 Use Case Diagram Overview

Use case diagrams are used to provide a high-level overview of the functional scope of the Academic Suite and to illustrate how different user roles interact with the system. These diagrams focus on *what* actions users can perform rather than *how* those actions are implemented. By presenting system functionalities from a role-based perspective, the diagrams help clarify responsibilities, access boundaries, and the overall structure of user interactions.

The Academic Suite defines four main actors: *Administrator*, *Academic Assistant*, *Lecturer*, and *Student*. Each actor interacts with a specific set of system functionalities through either the web portal or the mobile application. The following subsections describe the use case diagram associated with each role.

3.4.1 Administrator Use Case Diagram

The Administrator use case diagram illustrates the administrator's responsibility for managing master data and system configuration through the web portal. As shown in Figure 1, the administrator can manage subjects, rooms, classes, users, and grades. These core functionalities are grouped under higher-level management use cases to emphasize centralized control over academic data.

Extended use cases such as importing data from external sources, assigning roles, and resetting passwords further support administrative tasks. This diagram reflects the administrator's role in ensuring data consistency, correctness, and system readiness for daily academic operations.

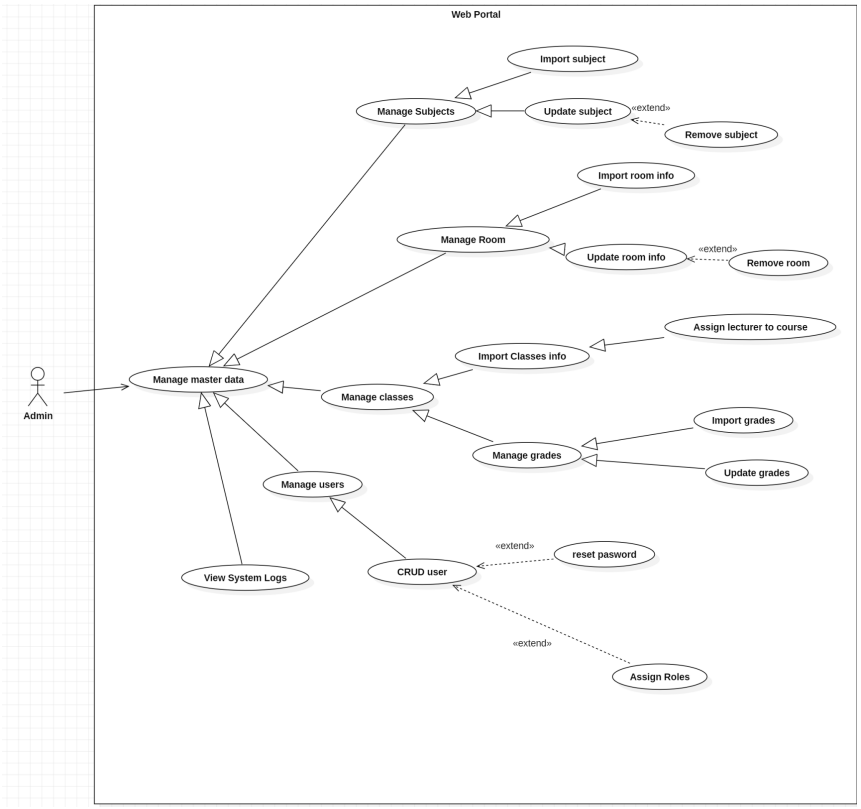

Figure 1: Administrator Use Case Diagram

3.4.2 Academic Assistant Use Case Diagram

The Academic Assistant use case diagram represents the core operational work flow of the Academic Suite. As illustrated in Figure 2, the academic assistant manages class schedules, including manual scheduling and schedule import, checks room availability, assigns proctors, manages exam schedules, and verifies exam eligibility.

In addition to scheduling tasks, the assistant handles student- and lecturer related requests, such as booking requests and makeup class requests. After processing a request, the system sends notifications to inform users of the outcome. This diagram highlights the assistant's central role in coordinating academic activities and emphasizes the importance of real-time interaction and automation within the system.

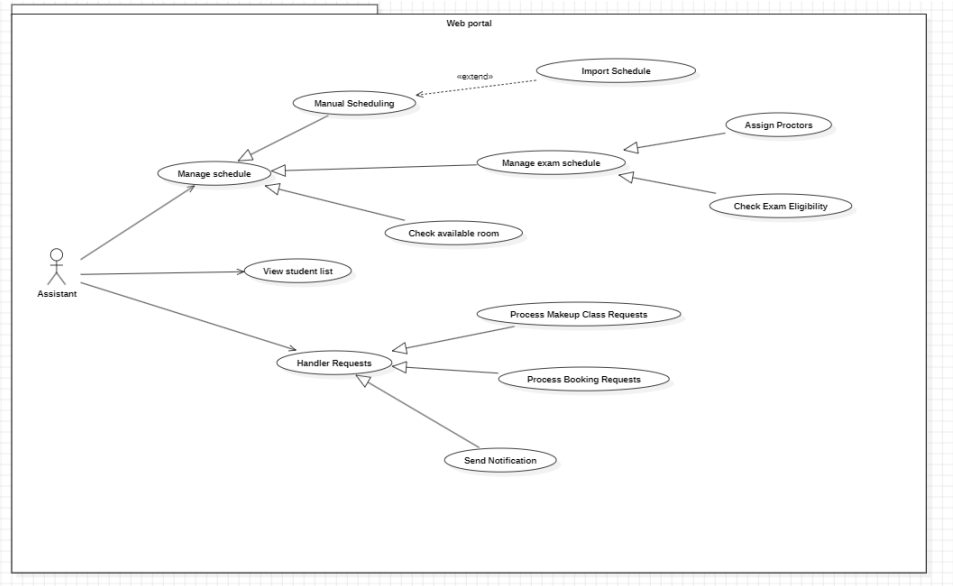

Figure 2: Academic Assistant Use Case Diagram

3.4.3 Lecturer Use Case Diagram

The Lecturer use case diagram describes interactions performed through the lecturer mobile application. As shown in Figure 3, lecturers can view assigned courses, access student lists, check attendance records, upload course-related documents, and view their teaching schedules.

Lecturers are also able to submit makeup class requests, which are later processed by academic assistants. Account management functionalities are included to allow lecturers to update personal information. This diagram reflects the lecturer's role as a teaching-focused user who interacts with the system primarily for instructional and informational purposes.

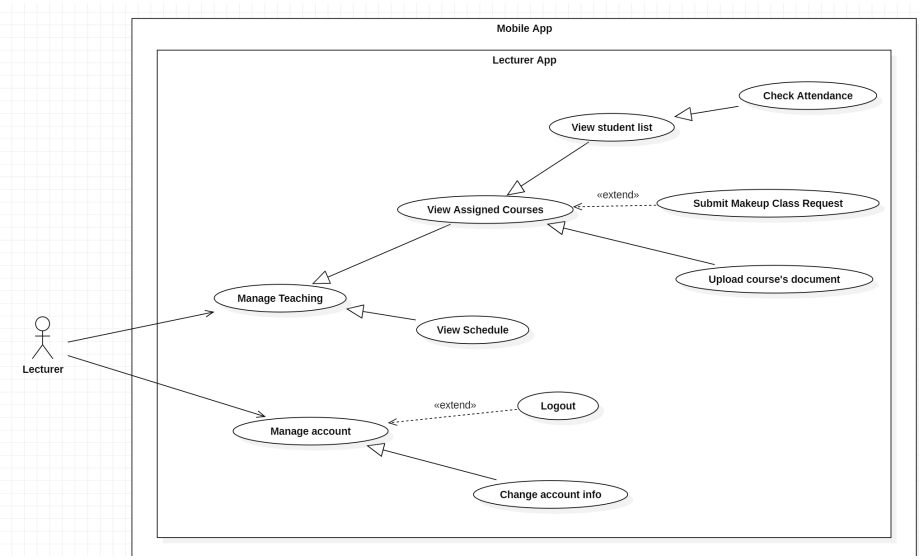

Figure 3: Lecturer Use Case Diagram

3.4.4 Student Use Case Diagram

The Student use case diagram focuses on enrollment management and learning progress tracking through the student mobile application. As illustrated in Figure 4, students can view all available courses, enroll in or drop courses, and access information related to enrolled classes, including schedules, attendance records, scores, and available course documents.

Students can also manage their personal accounts and monitor academic progress. This diagram emphasizes the student's role as an active participant in managing their academic activities while maintaining a clear separation from administrative functions.

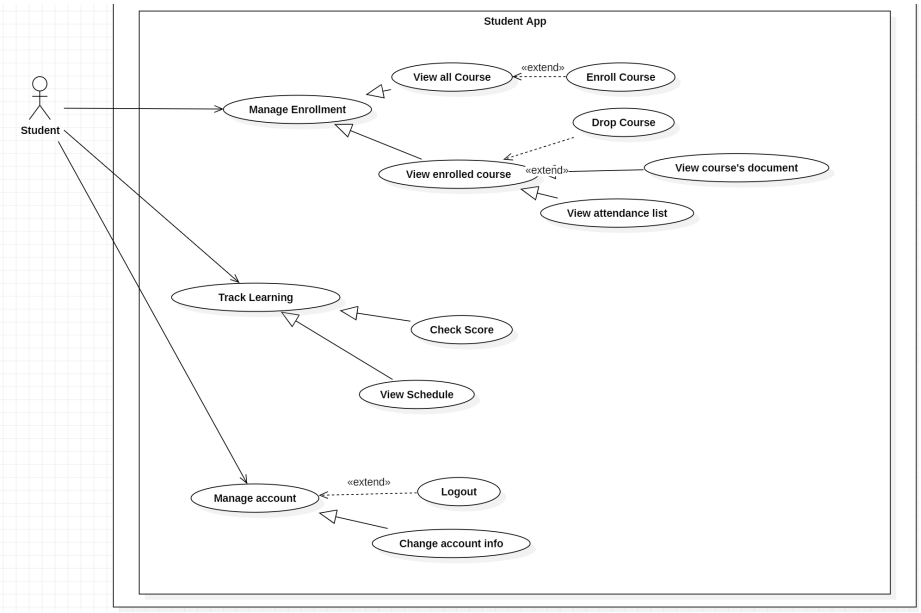

Figure 4: Student Use Case Diagram

Overall, these use case diagrams provide a comprehensive overview of system functionalities from the perspective of each user role. They establish a clear link between user responsibilities and system features and serve as a visual foundation for the detailed use case specifications presented in the following subsection.

3.5 Use Case Specification

This subsection presents detailed use case specifications derived from the use case diagrams introduced in the previous section. Each use case specification describes the interaction between users and the system, including actors, preconditions, main flows, alternative flows, and postconditions. These specifications provide a clear and structured description of system functionality and serve as a basis for system design and implementation.

3.5.1 UC01: Authentication

Actors: Administrator, Academic Assistant, Lecturer, Student

Description: This use case describes the process by which users authenticate themselves to access the Academic Suite system. Authentication is required before users can access any system functionality.

Preconditions:

- The user has a registered account in the system.
- The system is connected to the Firebase Authentication service. Main Flow:

1. The user opens the Academic Suite web or mobile application.
2. The system displays the login interface.
3. The user enters a valid email address and password.
4. The system sends the credentials to Firebase Authentication for verification.
5. Firebase Authentication validates the credentials.
6. Upon successful authentication, the system identifies the user role.
7. The user is redirected to the appropriate dashboard based on their role. Alternative Flow:

- *Invalid credentials:* If the entered credentials are incorrect, the system displays an error message and prompts the user to retry.
- *Forgot password:* The user may request a password reset, and the system sends a reset email via Firebase.

Postconditions:

- The user is successfully logged into the system and granted access according to their role.

3.5.2 UC02: Manage Subjects

Actor: Administrator

Description: This use case describes how the administrator manages subject information within the Academic Suite. It includes importing subject data, updating subject details, and removing obsolete subjects.

Preconditions:

- The administrator is authenticated and authorized.
- The system database is available.

Main Flow:

1. The administrator accesses the subject management module.
2. The system displays the list of existing subjects.
3. The administrator selects an action (import, update, or remove subject). 4. The system validates the input data.
5. The system stores the updated subject information in the database. 6. The system confirms the successful operation.

Alternative Flow:

- *Invalid data:* If the provided subject data is invalid or incomplete, the system displays an error message.
- *Remove subject:* If the subject is linked to existing classes, the system prevents deletion and notifies the administrator.

Postconditions:

- Subject information is updated consistently in the system.

3.5.3 UC03: Manage Classes

Actors: Administrator, Academic Assistant

Description: This use case describes how classes and schedules are managed within the Academic Suite, including class creation, scheduling, room as signment, and exam scheduling.

Preconditions:

- The user is authenticated.
- Subjects and rooms already exist in the system.

Main Flow:

1. The user accesses the class management or scheduling module.
2. The system displays available subjects, rooms, and time slots.
3. The user creates or updates a class schedule manually or by importing data.
4. The system checks room availability and scheduling conflicts.
5. The user confirms the schedule.
6. The system saves the class and schedule information.

Alternative Flow:

- *Schedule conflict:* If a conflict is detected, the system notifies the user and requests adjustments.
- *Import error:* If imported data is invalid, the system rejects the file and displays an error message.

Postconditions:

- Class and schedule data are stored and accessible to students and lecturers.

3.5.4 UC04: Handle Requests

Actor: Academic Assistant

Description: This use case describes how the academic assistant processes requests submitted by students and lecturers, such as booking requests and makeup class requests.

Preconditions:

- The academic assistant is authenticated.
- Pending requests exist in the system.

Main Flow:

1. The academic assistant accesses the request management dashboard.
2. The system displays all pending requests.
3. The assistant reviews a selected request.
4. The assistant approves or rejects the request.
5. The system updates the request status.
6. The system sends a notification to the requester.

Alternative Flow:

- *Incomplete request:* If required information is missing, the assistant may reject the request.

- *Schedule conflict:* If approval causes a conflict, the system alerts the assistant before confirmation.

Postconditions:

- Request status is updated and notifications are delivered.

3.5.5 UC05: View Analytics Dashboard

Actors: Administrator, Academic Assistant

Description: This use case describes how authorized users access analytical information and system statistics through a centralized dashboard. Preconditions:

- The user is authenticated.
- Analytical data exists in the system.

Main Flow:

1. The user navigates to the analytics dashboard.
2. The system retrieves aggregated academic data.
3. The system displays visual analytics such as charts and summaries.
4. The user reviews system performance and academic statistics.

Alternative Flow:

- *No data available:* The system displays an empty-state message. Postconditions:
- No system data is modified.

3.5.6 UC06: Schedule & Calendar

Actor: Lecturer & Student

Description: This use case describes how users view their academic

schedule. The calendar opens in Month View by default to provide an

overview, and users can switch to Week View for detailed timing.

Preconditions:

- The user is authenticated.
- Analytical data exists in the system.

Main Flow:

1. The user accesses the Schedule/Calendar feature from the menu.
2. The system displays the calendar in Month View by default.
3. The system indicates days containing classes with a small dot.
4. The user reviews the monthly overview.
5. The user selects the option to switch to Week View.
6. The system updates the interface to display the schedule in a weekly format

Alternative Flow:

– No classes scheduled: If a month or week has no classes, the system

displays an empty state or message indicating no scheduled activities.

– Navigation: The user may navigate between previous and next months/weeks using navigation controls.

.

3.5.7 UC07: Class Detail

Actors: Student, Lecturer

Description: This use case describes how users view detailed information

about their classes for a specific day. By selecting a date, users can see a list of classes containing essential details such as Subject Name, Room, Time, and Teacher.

Preconditions:

– The user is successfully authenticated.

– The user is currently viewing the Calendar.

Main Flow:

1. The user taps on a specific date in the calendar.

2. The system retrieves the schedule information for the selected date.

3. The system displays a list of all classes scheduled for that day below the calendar or in a pop-up.

4. For each class entry, the system shows the Subject Name, Room, Time, and Teacher Name.

Alternative Flow:

– No classes scheduled: If the user selects a date with no classes, the

system displays a message indicating No classes for this day.

– Missing details: If specific details are not available, the system displays “To Be Announced”

3.5.8 UC08: Enrolment

Actors: Student

Description: This use case describes how students view all available

subjects for the current semester and enroll in a class. The system ensures

students cannot duplicate enrollments by updating the button status.

Preconditions:

– The student is logged into the system.

– The registration period for the semester is open.

Main Flow:

1. The student navigates to the ”All Courses” section.

2. The system displays a list of all subjects available in the current semester.

3. The student locates a desired course and clicks the ”Enroll” button.

4. The system processes the enrollment request.

5. The system disables the button to prevent duplicate clicks.

Alternative Flow:

– Already Enrolled: If the student views the list and has already joined a class previously, the button is displayed as ”Enrolled” and is disabled immediately.

3.5.9 UC09: Document & Materials

Actors: Student

Description: This use case describes how students access their list of enrolled courses and view specific course materials. From the course list, students can enter a course detail screen to download documents uploaded by the lecturer..

Preconditions:

– The student is logged into the application.

– The student is currently enrolled in at least one course.

Main Flow:

1. The student navigates to the ”My Courses” section.
2. The system displays a list of all subjects the student is currently studying.
3. The student selects a specific course from the list.
4. The system redirects the student to the detail screen.
5. The system displays course information and a list of available documents.
6. The student clicks on a document name to view or download it.
7. The system opens the document preview or initiates the download process.

Alternative Flow:

– No Documents: If the lecturer has not uploaded any materials, the document section displays No materials available.

3.5.10 UC10: Scoreboard

Actors: Student

Description: This use case allows students to access their academic records directly from the main menu to track their learning progress. Students can view scores for different semesters and subjects

Preconditions:

– The student is logged into the system.

– Academic results have been updated in the system.

Main Flow:

1. The student selects the “scoreboard” feature from the navigation bar.
2. The system displays a summary list of semesters or directly lists the courses for the current semester.
3. The student selects a specific subject they want to check.
4. The system displays the detailed grade breakdown for that subject, including:

– Attendance Score

– Mid-term Exam Score

– Final Exam Score

5. The system displays the calculated Total Score and the status for the

course.

Alternative Flow:

– If the system has not yet input the grades, the app displays grading in progress for the specific score fields.

– If the student's total score is below the passing threshold, the system highlights the grade in red or marks the status as "Failed"

3.5.11 UC11: Take Attendance

Actors: Lecturer

Description: This use case describes how a lecturer records the atten-

dance status for a specific class session. The lecturer accesses the session

directly from the Schedule, ensuring the attendance is recorded for the

correct date and time slot.

Preconditions:

– The lecturer is logged in.

– The lecturer is currently viewing the Schedule

Main Flow:

1. The lecturer navigates to the Schedule screen.

2. The lecturer taps on a specific class block time slot on the calendar.

3. The system displays the details for that specific session.

4. The lecturer clicks the "Take Attendance" button within the detail view.

5. The system displays the list of enrolled students with checkboxes.

6. The lecturer marks the status:

– Checked: Present.

– Unchecked: Absent.

7. The lecturer clicks the "Save" button.

8. The system saves the data and displays a success message.

Alternative Flow:

– Future/Past Restriction: If the lecturer tries to take attendance for

a future session that hasn't started yet, the system will disable the "Take Attendance" button or show a warning.

– Cancel: The lecturer can close the session details without saving changes.

3.5.12 UC12: Upload Documents

Actors: Lecturer

Description: This use case allows the lecturer to upload learning ma-

terials for a specific course. The uploaded documents are linked directly to the selected course, ensuring only enrolled students of that course can

access them.

Preconditions:

– The lecturer is logged in.

– The lecturer has at least one assigned course for the semester.

Main Flow:

1. The lecturer accesses the Assigned Courses list.
2. The lecturer selects the specific course they want to add materials to.
3. The system displays the course dashboard.
4. The lecturer selects the ”Documents” tab section.
5. The lecturer clicks the Upload button.
6. The lecturer selects a file from their device.
7. The system uploads the file and links it to the current course ID.
8. The system refreshes the list to show the newly added document.

Alternative Flow:

– Delete Document: The lecturer can remove an existing document
from the list if it was uploaded by mistake.

– Upload Failure: If the internet connection is interrupted during the
upload, the system displays ”Upload failed”.

4. Architecture and system design

4.1 Mobile app

4.1.1 Overview

The system is divided into two distinct user flows based on roles

1. **Student Flow:** Focuses on viewing schedules, course registration, and tracking academic results.
2. **Lecturer Flow:** Focuses on managing teaching schedules and class information.

4.1.2 System Modules

The system is divided into three logical functional areas based on user roles and shared utilities:

A. Authentication & Identity Module

- **Role:** Acts as the security gateway.
- **Function:** Role-Based Access Control to route users to either the Student or Lecturer interface upon successful login.

B. Student Module

Focused on the academic lifecycle of the student:

- **Enrollment Subsystem:** Allows students to browse the course catalog and submit registration requests.

- **Academic Tracking:** Visualizes grading data, calculating weighted scores and displaying attendance history.
- **Personal Schedule:** Renders a weekly or monthly calendar highlighting learning sessions.

C. Lecturer Module

- **Teaching Dashboard:** Aggregates assigned classes and displays immediate teaching schedules .
- **Class Management:** Provides detailed views of specific class cohorts.

4.1.3 System Design

A. Presentation Layer (UI)

This layer is responsible for rendering the graphical user interface and capturing user interactions.

- **Navigation Bar:** Manage the global navigation structure and host different functional screens.
- **View Components:** Consist of modular screens that display specific data sets.
- **Adapters:** Specialized components responsible for efficiently mapping data collections into scrollable UI elements.

B. Network & Connectivity Layer

This layer acts as the bridge between the mobile app and the backend server.

- **RetrofitClient:** A centralized networking component configured to communicate with RESTful endpoints.
- **Request Management:** Handles asynchronous HTTP requests
- **Data serialization:** Automatically converts raw JSON responses from the server into structured Data Entities used by the app

C. Controller Layer and Appflow

- Handles user interactions .
- Initiates network requests and processes the responses

Appflow:

1. Login Screen
   - Activity: The user enters Email and Password. The app calls the login API.
   - Logic:
     - If successful: Saves USER_ID and USER_NAME to local storage (SharedPreferences).
     - Role-based Navigation:
       * If Role is STUDENT, navigate to StudentMainActivity.
       * If Role is LECTURER, navigate to LecturerMainActivity.
2. Student Flow
   - Main Screen (Calendar):
     - Opens the Calendar by default.

- The app automatically calls the getStudentSchedule API to fetch class schedules, parses ISO-8601 dates/times, and displays red dots on dates with scheduled classes.

- Menu (Navigation Drawer):
  - The menu Header automatically displays the Name, Student ID, and Major by calling the getProfile API.
- Course Registration (All Courses):
  - Displays the list of subjects (via getSubjects).
  - Smart Enroll Button:
    - If the API returns isEnrolled = true, the button turns gray (Disabled).
    - If not, the button remains green (Enabled).
  - Action: Click Enroll – Calls enroll Auto API – Shows success notification – Immediately updates button status.
- My Courses:
  - Displays the list of registered subjects .
  - Click Detail – Navigate to the Course Detail screen.
  - On the Detail screen: Call the get Attendance API to view attendance history (Present/Absent).

D. Data Layer

- Java objects that mirror the JSON structure from the API

E. Interaction Flow Design (Sequence Diagrams)

1. LoginActivity

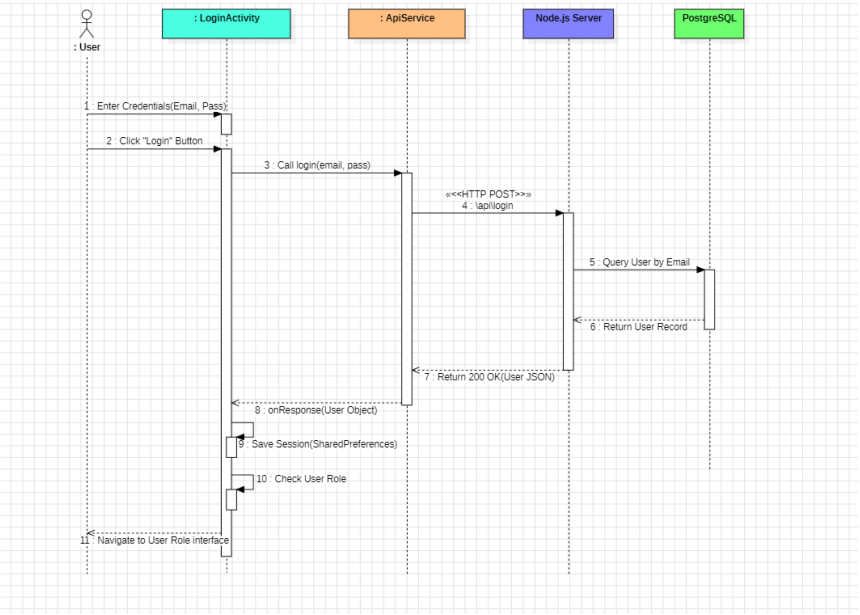

Figure 5: Login Sequence Diagram

Step 1: The User enters credentials in the LoginActivity.

Step 2: The ApiService sends a POST request to the Node.js Server

Step 3: The Server queries PostgreSQL to validate the user.

Step 4: Upon success, the Client receives the User Object, saves the session into SharedPreferences, and navigates to the specific UI based on the User Role

2. Calendar Fragment

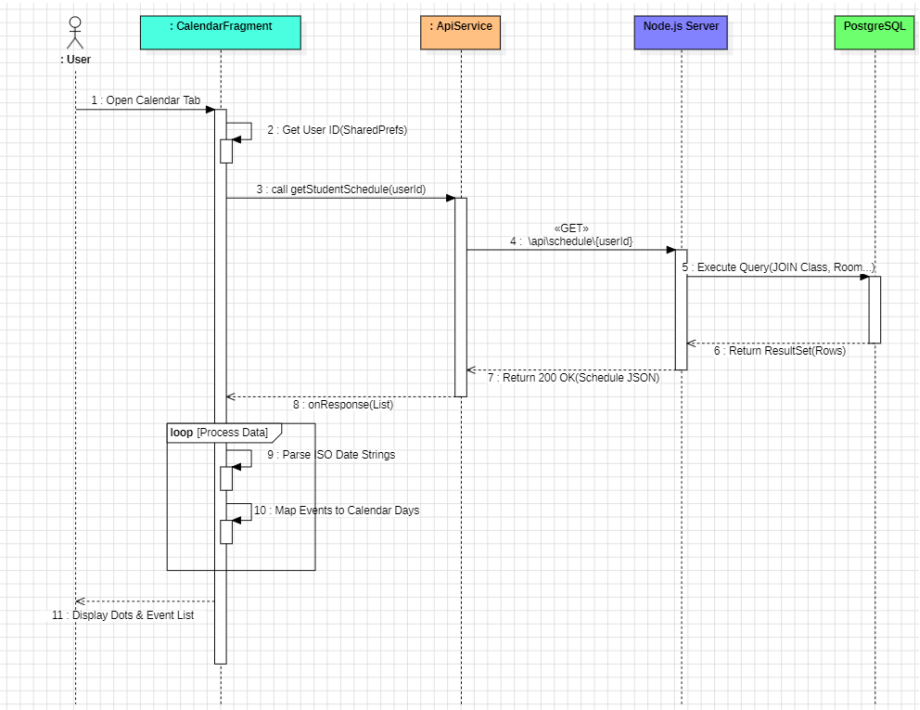

Figure 6: Calender Sequence Diagram

Step 1: The CalendarFragment initializes and retrieves the stored User ID.

Step 2: The app calls get StudentSchedule(userId) via the API.

Step 3: The Server performs a JOIN query (Class, Room) and returns the schedule JSON.

Step 4: The Client parses the ISO date strings and maps events to the calendar view, displaying visual indicators (dots) for busy days.

3. All Course Fragment

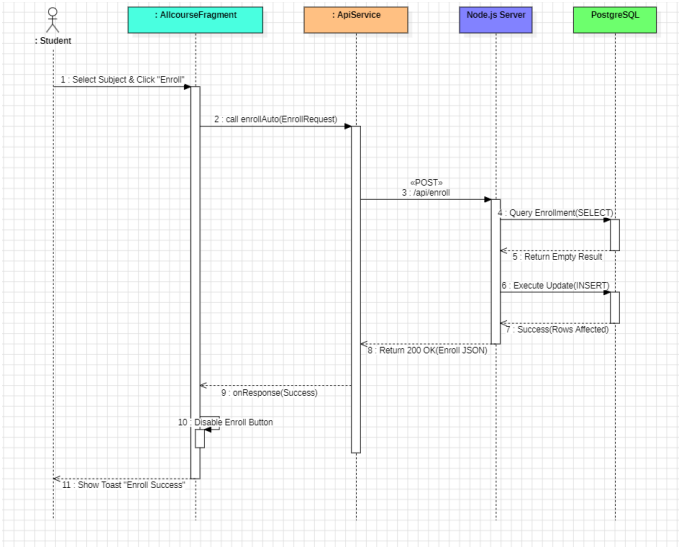

Figure 7: All Course Sequence Diagram

Step 1: The Student selects a subject in AllCourseFragment and clicks "Enroll".

Step 2: The app triggers the enrollAuto API call.

Step 3: The Server verifies eligibility and inserts a new record into the Enrollment table.

Step 4: Upon receiving a generic 200 OK success response, the UI immediately disables the "Enroll" button and shows a success Toast notification.

4. Attendance

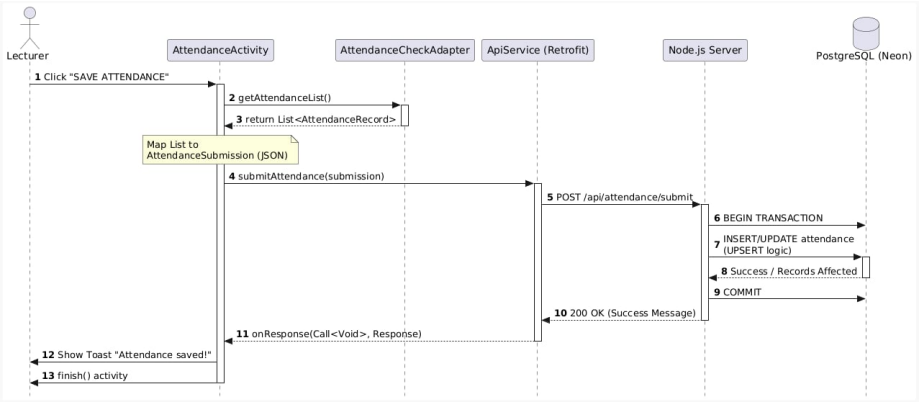

Figure 8: Attendance Checking Sequence Diagram

Step 1: Loading the Class List

When the Lecturer opens the attendance screen, the app automatically asks the server for the list of students in that class. The server retrieves the latest data from the database and sends it back to the app to be displayed.

Step 2: Taking Attendance

The Lecturer sees the list of students and simply taps to mark them as "Present" or "Absent". These changes happen immediately on the screen.

Step 3: Saving to the System

Once finished, the Lecturer clicks the save button. The app packages all the attendance data and sends it securely to the server.

Step 4: Confirmation

The server saves the records into the database. Finally, the app shows a "Saved Successfully" message and closes the screen, confirming the job is done.

4.2 Web app

The Web App is a centralized administrative platform designed specifically for Academic Assistants to manage university operations. It is built using a modern, scalable architecture that separates frontend presentation from backend business logic and real-time notification services.

4.2.1 Technology Stack

The web application utilizes a specialized technology stack to ensure performance and reliability:

Frontend: Built with React 18 and TypeScript, using Vite as the build tool for fast development.

UI styling is managed via TailwindCSS, while Zustand is used for efficient state management.

Backend Services: The core logic runs on Node.js and Express (TypeScript), utilizing Prisma ORM for structured database interactions.

Database: Employs Neon Database, a serverless PostgreSQL solution, for main academic data storage and Firebase Cloud Messaging (FCM) for handling real-time notifications.

Authentication: Integrated with Firebase Authentication on the frontend, with token verification performed by the Firebase Admin SDK on the backend.

4.2.2 High-Level Architecture

The system follows a distributed service topology:

1.**Client Layer (Port 5173)**: The React frontend handles user interactions and communicates with two distinct backend services.

2.**Core API Service (Port 4000)**: The primary REST API that manages authentication, user records, subjects, classes, rooms, and schedules.

3.**Realtime Service (Port 5002)**: A dedicated service for managing Firestore-based notifications and live status updates.

4.2.3 Core Functional Modules

Following a refactoring to focus on administrative efficiency, the current web portal concentrates on two primary modules:

Authentication Module: Manages secure login and role verification to ensure only authorized assistants can access administrative features.

Assistant Portal: A comprehensive dashboard providing tools for:
User & Role Management: Creating and updating user accounts and roles.

Academic Resource Management: Handling the CRUD (Create, Read, Update, Delete)
operations for Subjects, Classes, and Rooms.

Scheduling: Manual and bulk (Excel) scheduling of classes while checking for room availability and conflicts.

Request Handling: Centralized processing of student and lecturer requests, such as makeup classes or room bookings.

4.2.4 Data Flow and Interaction

The web app's data flow is designed for integrity and real-time responsiveness:

Authentication Flow: The client authenticates with Firebase, receives an ID token, and sends it to the Core Service for verification and retrieval of detailed user data (e.g., role, full name) from

the PostgreSQL database.

Management Flow: When an assistant updates a schedule, the Core Service validates the request against existing data in the Neon DB. Upon success, it can trigger events to the Realtime Service, which updates Firestore to push notifications to relevant users.

The interactions between components to do specific tasks on the web app is shown below.

1. Account Management

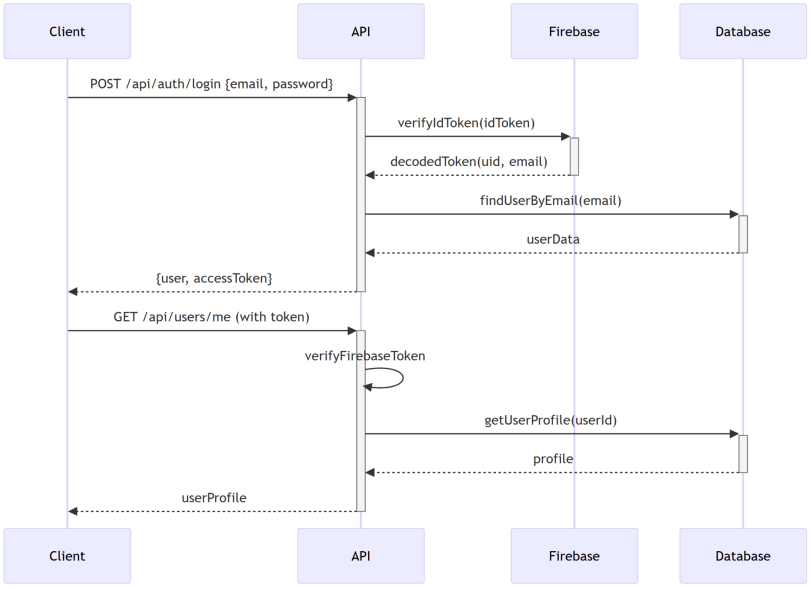

Figure 9: Account Management Sequence Diagram

Step 1: Client Sends Login Request

The Client sends a POST request to the API at /api/auth/login with the parameters email and password to initiate the login process.

Step 2: API Verifies the Token

The API sends a request to Firebase to verify the ID token using the verifyIdToken(idToken) method.

Step 3: Firebase Verifies and Returns Token Data

Firebase processes the request and responds with a decoded token, which contains the user ID (uid) and the email associated with the token.

Step 4: API Retrieves User Data

The API then queries the Database to retrieve the user's data by calling findUserByEmail(email).

Step 5: Database Returns User Data

The Database responds with the user data, including details such as the user's account information and other relevant data.

Step 6: API Sends Response to Client

The API sends a response to the Client, which includes the user's information and an access token to authenticate future requests.

Step 7: Client Requests User Profile

After receiving the access token, the Client sends a GET request to the API at /api/users/me with the access token to retrieve the profile information.

Step 8: API Verifies Firebase Token

The API verifies the Firebase token to ensure the request is authenticated and the client has the necessary permissions to access the user profile.

Step 9: API Retrieves User Profile from Database

The API queries the Database again, this time using the getUserProfile(userId) method to fetch detailed profile information associated with the user's ID.

Step 10: Database Returns User Profile

The Database responds with the user's profile data, such as personal details, cohort, and other relevant information.

Step 11: API Sends Profile to Client

The API sends the retrieved user profile information back to the Client, completing the profile retrieval process.

2. Request & Authorization

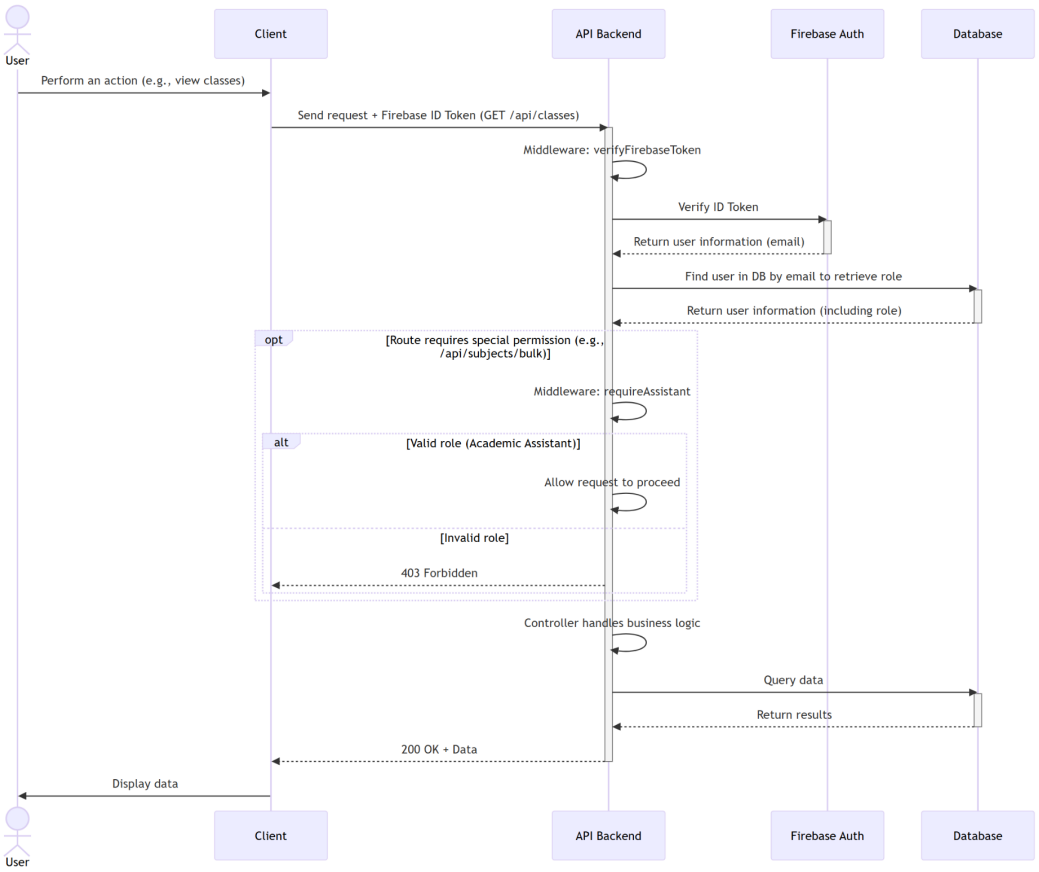

Figure 10: Request & Authorization Sequence Diagram

Step 1: User initiates an action

The user performs an action on the client application, such as viewing the list of classes.

Step 2: Client sends authenticated request

The client sends an HTTP request (e.g., GET /api/classes) to the API Backend, attaching a Firebase ID Token.

Step 3: API verifies authentication token

The API Backend executes the verifyFirebaseToken middleware and sends the ID Token to Firebase Auth for validation.

Step 4: Firebase returns user identity

Firebase Auth verifies the token and returns the authenticated user's information (e.g., email).

Step 5: API retrieves user role from database

Using the email, the API queries the database to fetch the user record and determine the user's role.

Step 6: Authorization check (if required)

If the requested route requires special permission (e.g., Academic Assistant), the requireAssistant middleware checks the role. If invalid, the API returns **403 Forbidden**.

If valid, the request proceeds.

Step 7: Business logic execution and data query

The API controller processes the request and queries the database for the required data.

Step 8: Response returned to user

The API sends a **200 OK** response with the requested data to the client, which then displays it to the user.

3. Enrollment Management

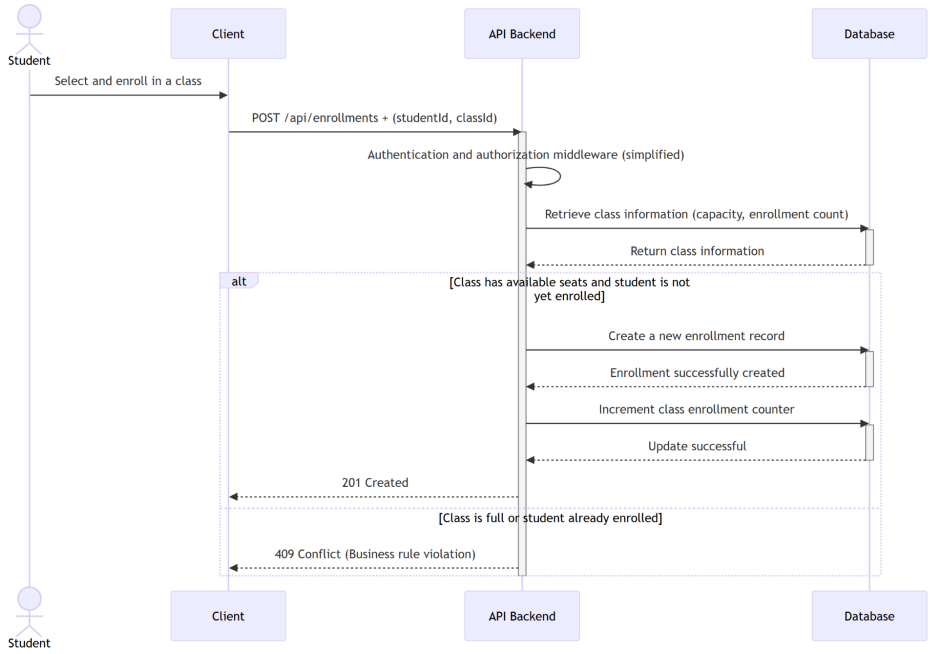

Figure 11: Enrollment Management Sequence Diagram

Step 1: Student initiates enrollment

The student selects a class to enroll in through the client application.

Step 2: Client submits enrollment request

The client sends a POST /api/enrollments request to the API Backend with studentId and classId.

Step 3: API performs authentication and authorization

The API Backend executes authentication and authorization middleware to validate the request.

Step 4: API retrieves class status

The API queries the database to obtain class information, including capacity and current enrollment count.

Step 5: System evaluates enrollment conditions

The API checks whether the class has available seats and whether the student is not already enrolled.

Step 6: Enrollment is created (if conditions are met)

If valid, the API creates a new enrollment record in the database.

Step 7: Class enrollment counter is updated

The API increments the class enrollment counter to reflect the new enrollment.

Step 8: Response is returned to the client

- **201 Created** is returned if enrollment succeeds.
- **409 Conflict** is returned if the class is full or the student is already enrolled.

4.2.5 Advantages of the Web Infrastructure

The transition to a serverless PostgreSQL architecture (Neon) provides several benefits to the academic suite:

Scalability: Automatic scaling to handle varying loads during peak registration periods.

Data Reliability: Strong consistency and SSL-secured connections for sensitive academic records.

Efficiency: Connection pooling and serverless management reduce the administrative overhead for the system's maintenance.

5. Implementation and Results

5.1. Backend & System Implementation

Based on the database schema, CRUD functions have been added for every main module of the system.

Figure 12: Database Schema

Figure 12: Database Schema

This database schema is designed to support a comprehensive educational system, managing users, subjects, enrollments, schedules, grading, and attendance. At its core, the User table stores information about the students, their roles, and personal details. The Subject table defines academic subjects, their credits, and related details, while SubjectPrerequisite manages prerequisite relationships between subjects. Class represents individual class sessions, with attributes like max capacity, instructor, and class timings. The Enrollment table tracks student registrations for classes. The ClassSchedule table defines the schedule of each class session, linking to rooms and times. The GradeItems and GradeRecords tables manage the grading system, including items like assignments or exams and the actual grades awarded to students. Request stores student or system requests, including status and payload information, while the Notification table handles communication between the system and users. Lastly, the Attendance table tracks student attendance for scheduled classes.

This schema is designed to efficiently handle data for scheduling, grading, attendance, and student management, making it a robust solution for an academic platform.

5.2. Database Design

For the database and data layer, we have chosen a Hybrid Approach, combining Firebase Firestore as the primary database and Neon PostgreSQL as the backup system. This approach leverages the strengths of both technologies to ensure real-time updates, scalability, and efficient data processing.

5.2.1. Notifications: Firebase Cloud Messaging + Neon Database

The data flow for notifications is shown below:

1. Create notification → Write to Firestore (realtime)

2. Background job → Sync to Neon (for analytics)

3. Frontend → Subscribe Firestore (realtime updates)

4. Analytics → Query Neon (complex reports)

By using Firebase as the primary database, the system ensures that users receive real-time updates, enabling immediate reflection of changes across all connected clients. Neon PostgreSQL is used as a backup system, providing an efficient solution for storing historical data. It can also handle complex queries and analytics, which enables advanced reporting and insights without compromising on speed or accuracy. Moreover, the system remains
cost-efficient by leveraging Firestore's scalability and Neon's powerful querying capabilities for archival data.

5.2.2. Requests: Neon PostgreSQL (Primary) + Polling/WebSocket

The data flow for notifications is shown below:

1. Create/Update request → Write to Neon (ACID)

2. Frontend → Poll API every 5-10s

3. Frontend → WebSocket server (optional for stronger realtime requests)

4. Analytics → Query Neon (complex JOIN queries)

This implementation allows the system to handle complex queries, such as joining user, class and subject. Neon's support for ACID transactions ensures strong data consistency and reliability, processing requests accurately and safely even in the case of failures or interruptions. The database's powerful query and transaction handling capabilities also facilitate seamless workflow management, ensuring that each step in the process is efficiently executed and tracked. Additionally, Neon's robust querying engine enables efficient analytics and reporting, generating detailed, real-time insights that are crucial for data-driven
decision-making.

5.3. Experimental Results and System Outcomes

5.3.1. Backend Functional Verification

Table 3: Overall backend verification result

Component	Detailed Status
Database Connection	Connected and reachable
Health Check	/health endpoint is working
Subjects API	Endpoints are ready for data implementation
Rooms API	Endpoints are ready for data implementation
Users API	Endpoint are ready, users added

Backend functional verification was conducted using a shell-based API testing script to ensure that core system components were correctly configured and operational. The tests confirmed successful database connectivity, proper health check responses, and correct behavior of essential RESTful APIs, including user, subject, and room endpoints.

Firebase-based authentication was also verified, demonstrating that the backend could correctly process and validate ID tokens. In addition, CORS configuration and
frontend–backend port alignment were validated, ensuring smooth communication between system components. Overall, these results confirm that the backend system is stable and ready for further performance and analytical evaluation.

5.3.2. Performance Evaluation

The experimental results indicate that almost all core functional requirements of the system have been successfully implemented. Key functionalities such as authentication, role-based access control, subject and class management, enrollment processing, scheduling, and notification handling were verified through backend testing.

To evaluate backend performance, response time measurements were conducted using a shell-based testing script that repeatedly invoked selected API endpoints under local testing conditions. The average response time was calculated by sending multiple consecutive requests to the same endpoint and computing the mean latency. The latency of the GET\ api\subject endpoint is evaluated using a shell-based testing script executed in a Git Bash environment on Windows. The average response time after 10 iterations was 76ms. Endpoints involving more complex logic and database interactions, such as enrollment and schedule-related APIs, exhibited slightly higher latency but remained within acceptable limits for an interactive web-based academic system.

These results indicate that the backend is capable of handling typical user requests
efficiently, while maintaining consistent performance across different types of API operations.

5.3.3. User Interfaces

This subsection presents an overview of the web application's user interface to provide a visual demonstration of the implemented system. An administrator user is created in the database to illustrate the main pages and interaction flows of the Academic Suite web application, highlighting how users access and navigate core functionalities.

After login successfully, the web will navigate to the Academic Management section.

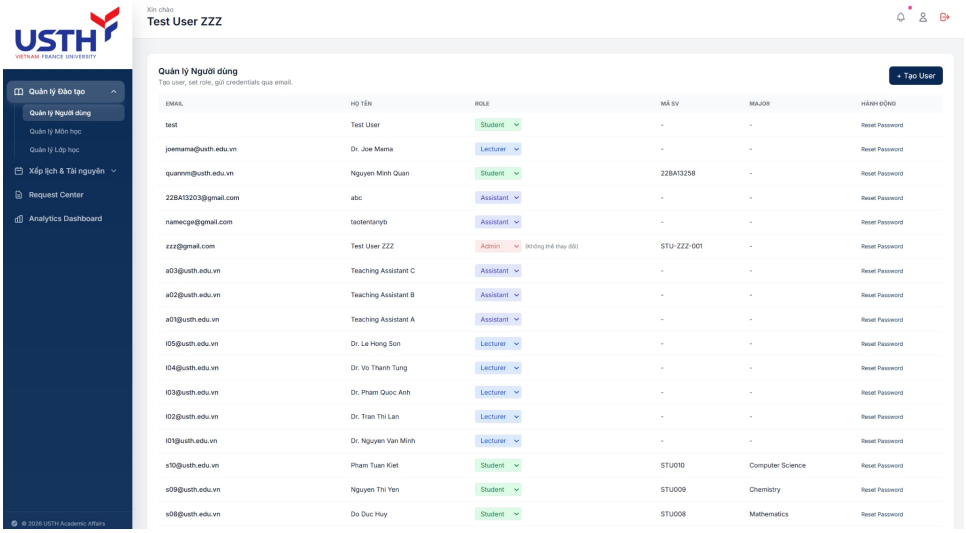

Figure 13: User Management subsection in Academic Management

From here,the Administrator can manage the subjects and classes.

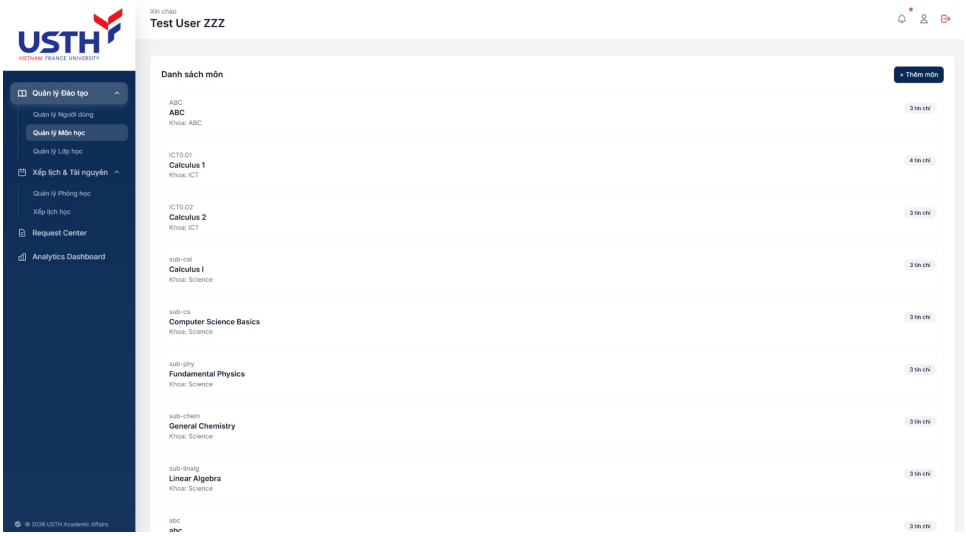

Figure 14: Subject Management

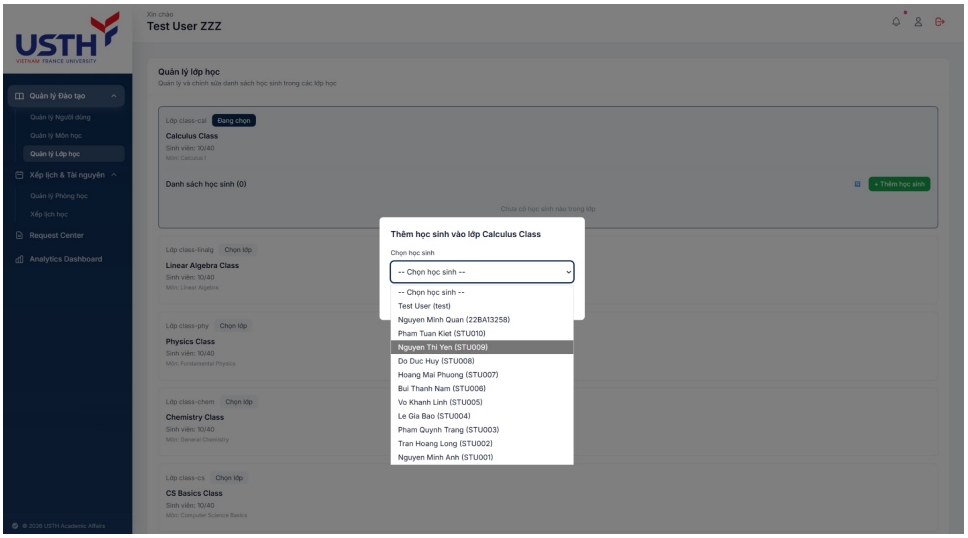

Figure 15: Add students in Calculus class

Administrators are also granted access to Room Management and Schedule Management.

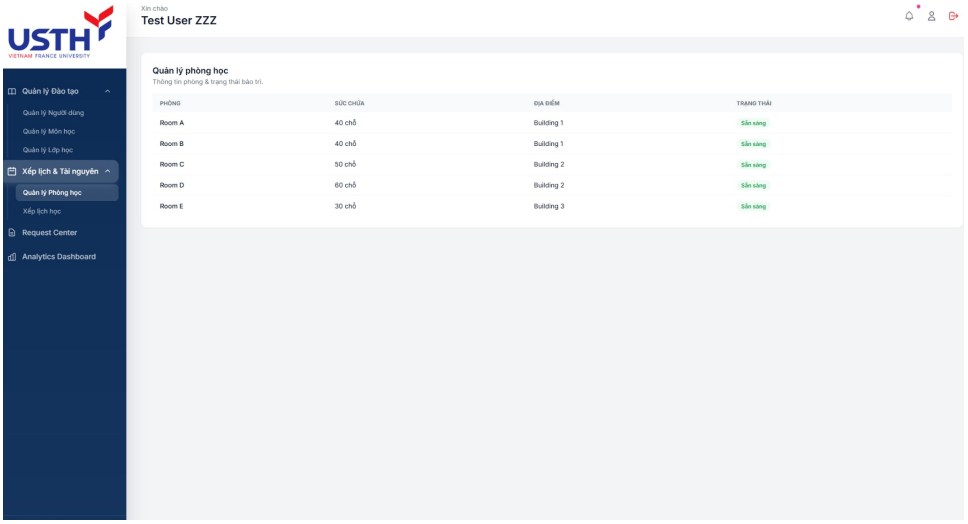

Figure 16: Room Management Interface

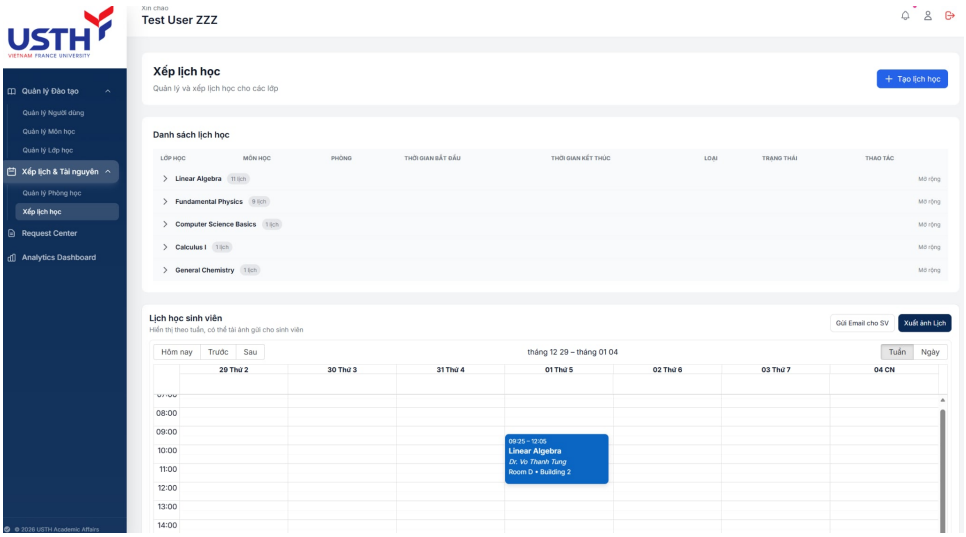

Figure 17: Schedule Management Interface

6. Conclusion and Future Work

This project presented the design and implementation of an Academic Suite web application aimed at supporting core academic management workflows in a centralized and structured manner. The system successfully integrates a modular backend architecture, a relational database for reliable data management, and a web-based user interface to enable efficient interaction between users and academic data. Through functional verification and
performance evaluation, the results demonstrate that the implemented system is stable, responsive, and capable of supporting typical academic operations. Overall, the project establishes a solid technical foundation for an academic management platform while highlighting opportunities for further enhancement and expansion.

Despite the successful implementation of core system functionalities, several challenges and limitations were identified during development. First, the notification module has not yet been fully implemented due to limited familiarity with Firebase services, particularly in designing and integrating real-time notification workflows. As a result, notification-related features remain at a conceptual or partially implemented level. In addition, the role of the administrator within the system has not been fully specified or implemented in detail. While basic
role-based access control exists, advanced administrative responsibilities such as system monitoring, request moderation, and user management workflows require further clarification and development. These limitations highlight areas for future improvement and extension of the system.