Software Requirements Specification

for

<EduFinder>

Version 1.0 approved

Prepared by <Chee Seng, Ryan, Sizhe, Tarun, Wenhao>

Nanyang Technological University

16 Nov 2025

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Lim Kiat Yang Ryan	3rd November 2025	Added 1.x Introduction.	1.0
Peng Sizhe	3rd November 2025	Added 2.1–2.5 Product Perspective, Functions, User Classes, Operating Environment, and Constraints	1.0
Tarun Ilangoven	3rd November 2025	Added 3.x External Interface Requirements	1.0
Peng Sizhe	4th November 2025	Drafted 2.6 User Documentation; added 2.7 Assumptions and Dependencies.	1.0
Yu Wenhao	5th November 2025	Added 4.x System Features; completed 6.1 Database Requirements and 6.2 Legal Requirements.	1.0
Lim Kiat Yang Ryan	5th November 2025	Added 7 Appendix A and 8 Appendix B.	1.0
Yong Chee Seng	6th November 2025	Added 5.x Non-Functional Requirements; added 6.x Other Requirements.	1.0
Yong Chee Seng	8th November 2025	Expanded 2.6 User Documentation; introduced B.6 Backend Testing (unit and integration).	1.0
Yu Wenhao	8th November 2025	Expanded 7 Appendix A: Data Dictionary; updated 8 Appendix B: Analysis Models for clarity.	1.0
Yong Chee Seng	9th November 2025	Revised Appendix B diagrams for clarity.	1.0

1. Introduction

1.1 Purpose

This software requirements specification is written to provide a guideline for the production and development of the EduFinder application. It outlines the basic functional and non-functional requirements, and details the specifications of the application.

The primary objectives of the EduFinder application are:

- School Information and Comparison: Deliver a comprehensive, data-driven analysis that empowers users with the tools to make informed decisions about school selection, including in-depth comparisons based on various criteria such as location, academic performance, etc.

- School Navigation and Transit Optimization: Deliver real-time, location-based navigation tools that guide users efficiently to their selected schools. The system integrates with real-time traffic data to calculate transit durations, and suggest the most optimal routes.

- Interactive Chatbox for Personalized Assistance: Incorporate an intelligent chatbox that provides real-time, interactive support for users. The chatbox will assist with answering questions, offering guidance on school comparison, providing navigation suggestions, and aiding users with any additional inquiries they may have.

1.2 Document Conventions

This Software Requirements Specification follows the IEEE830-1998 standard. This includes the below formatting conventions:

- **Font**: Times New Roman
- **Heading 1**: Size 18, Bold
- **Heading 2**: Size 14, Bold
- **Heading 3**: Size 12, Bold
- **Heading 4**: Size 12, Bold
- **Content**: Size 12
- **Spacing in Content**: 1 line spaced

Furthermore, priorities for higher-level requirements are assumed to be inherited by detailed requirements.

Further conventions on special terms used throughout this document are described in Appendix A: Data Dictionary.

1.3 Intended Audience and Reading Suggestions

The intended audience of this document are the various stakeholders involved in the entire development cycle of the EduFinder Application, not limited to Testers, Product Managers and Software Engineers.

1.3.1 Testers

Testers should begin with **Section 2.2: Product Functions**, which extensively lists out the intended features of this application. After understanding the various functions, they can move on to **Appendix B.6**, to understand the testing techniques applied to this application.

1.3.2 Product Managers

For Product Managers, we highly recommend beginning with Section **2.2: Product Functions,** which outlines the core features and objectives of this application. After reviewing the product functions, they should read Section **3.3: User Interfaces** to familiarize themselves with the application design. Following this, Section **4: System Features** illustrates the core functionalities of the application.

1.3.3 Software Engineers

Software Engineers can start from **4: System Features** to understand each use case and their requirements. Then, **Appendix B: Analysis Model** should provide a detailed understanding of how EduFinder's code should be structured.

1.4 Product Scope

For decades, choosing the right secondary school has always been a hassle for both parents and students. There is still no comprehensive and user-friendly platform that showcases all the secondary schools across the country. For most students, the Primary School Leaving Examination (PSLE) marks a pivotal point in their academic journey—ushering them into the next chapter of their education. Yet, with a multitude of schools to choose from, parents and students often find themselves overwhelmed and uncertain.

Currently, a handful of websites and apps attempt to solve this issue by displaying school information. However, these platforms are fragmented, often lacking the depth and functionality required to guide families through such an important decision. Therefore, we came up with the problem statement: How can we help parents and students decide or choose their preferred school by assimilating different functionalities with precise map navigation to create an all-for-one seamless experience?

There is where EduFinder steps in—our revolutionary solution that transforms the entire school selection process into a seamless, intuitive experience. EduFinder isn't just another school directory. It is a **smart, data-driven search engine that allows** users to filter schools by criteria that matter most to them, i.e. academic focus, extracurricular activities. proximity, and much more. We have also taken a step further to integrate **Google Maps API** to offer real-time navigation and transit times. Now, users can discover the most efficient routes to get to desired schools.

EduFinder is the **ultimate tool** for making one of the most important decisions in a student's educational journey. We are not just helping you choose a school—we are helping you shape a student's future.

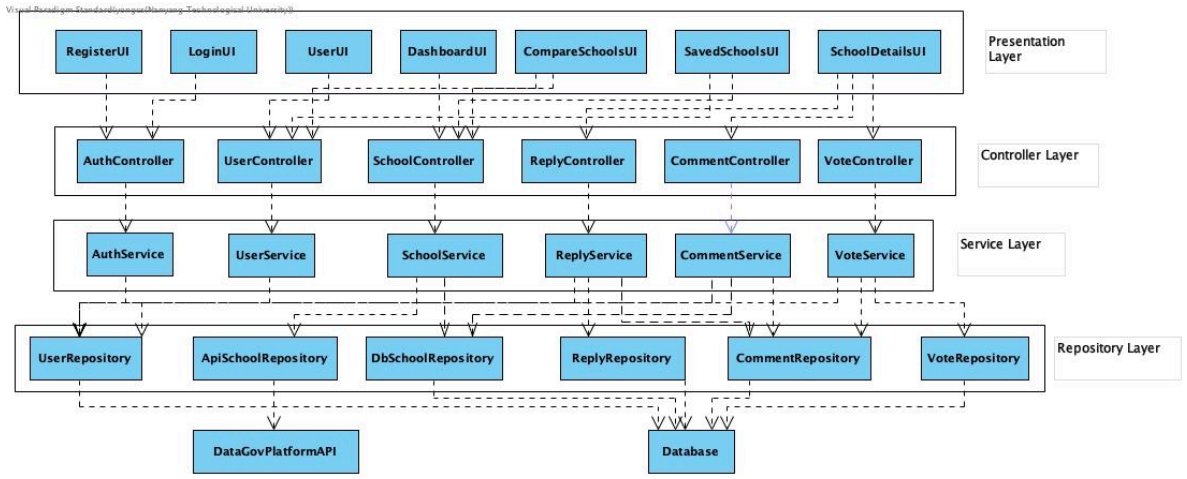
2. Overall Description

2.1 Product Perspective

EduFinder is a software product designed to eliminate the user challenge of fragmented and inconsistent information when searching for educational institutions. It functions as a central, integrated platform that is not a subsystem of a larger parent product.

The system is classified as a specialized Data Aggregator and Community Platform. Its operation is critically dependent on interfacing with one major external system: the Government API (e.g., MOE API), from which it retrieves and caches the primary, authoritative school data. A key constraint is the external interface rate limit, requiring the system to limit API calls to a maximum of five per day. The application is the source and manager of its second key component: the User Interaction Layer, which handles user-generated content such as school comparisons, comments, replies, and votes.

The system architecture diagram can be found below for a better understanding of EduFinder's operations.



2.2 Product Functions

The EduFinder system provides four primary functional categories that allow users to discover, evaluate, and discuss educational institutions. The major functions are summarized below.

2.2.1 School Searching and Filtering

1. Allows users to query schools by their specific name.
2. Filters the school list based on a user-defined cut-off point range (4–32).
3. Refines the search results by a user-selected region (location).
4. Filters the results further based on user-selected subjects and Co-Curricular Activities (CCAs).
5. Filters schools by education level (e.g., Primary, Secondary, Junior College)
6. Filters schools by nature code (e.g., Co-ed, Boys, Girls).
7. Filters schools by type (e.g., Government, Government-aided).

2.2.2 School Viewing and Comparison

1. Displays a comprehensive profile page with detailed information when a user selects a school.
2. Presents a side-by-side comparison interface, allowing users to evaluate the key attributes of multiple selected schools.
3. Allows user to enter a Natural Language Prompt to add context for Edufinder to evaluate the options better

2.2.3 User Personalization and Bookmarking

1. Manages user authentication, including account creation (Signup) and access (Login).
2. Allows authenticated users to save schools as favourites (bookmark).
3. Provides users with access to their personal list of saved schools for future reference.

2.2.4 Community Engagement

1. Allows authenticated users to post new comments on a school's profile page.
2. Supports the creation of discussion threads by allowing users to post replies to existing comments.
3. Incorporates a voting mechanism (upvote/downvote) for users to rank the quality and relevance of comments.

2.3 User Classes and Characteristics

The EduFinder system is mainly for two types of users, unauthenticated users who are browsing the website without creating an account or logging in, and authenticated users who have previously created accounts and are logged in for the session.

The most critical user class would be the authenticated users as they are the most frequent users and have the most functionalities available to them.

2.3.1 Unauthenticated User

Aspect	Description
Security or Privilege Levels	Low Privilege.
Frequency of Use	Low to medium.
Subset of Function Used	Read-only. Limited to core functions: - Search - Basic Public School Profile Viewing
Characteristics	Users that only check the school's information seldom or for one-time usage. Prioritizes speed and ease of access. Does not require to compare schools directly.

2.3.2 Authenticated User

Aspect	Description
Security or Privilege Levels	High Privilege
Frequency of Use	Medium to high
Subset of Function Used	Full user subset. - Search - Basic Public School Profile Viewing - Filtering Schools - Compare Schools - Voting Schools - Commenting on Schools - Reply to Comments
Characteristics	Regular users who frequently explore schools, compare options, and engage with the community. They are typically parents or students who need to evaluate schools meticulously, compare multiple attributes, and review past feedback or community discussions before making informed decisions.

2.4 Operating Environment

This section describes where EduFinder runs in production, what device permissions it requires, which external services it uses, and the recommended development/test environment.

2.4.1 Production Environment and Device Requirements

1. **Supported Platforms**: Modern desktop and mobile browsers (Chrome, Firefox, Safari, Edge) on Windows 10+, macOS 10.15+, and current Linux distributions. Mobile access is supported via iOS Safari and Android Chrome.
2. **Network**: Stable internet required; all traffic between client, backend, and third-party services MUST use HTTPS/TLS.
3. **Offline behaviour**: Client should use cached school data when offline or when external APIs are unavailable; UI must indicate offline/cached status.
4. **Device capability**: Location services access is required only for location-dependent features (nearby school discovery, navigation). The app requests location permission at point of use and only proceeds after explicit user consent which must be logged.

2.4.2 External Services and Usage Constraints

1. Singapore Government Open Data API (data.gov.sg)
    a. **Role**: source of authoritative school metadata (examples: school general information; subjects; CCAs; MOE programmes).
    b. **Format**: JSON
    c. **Auth**: no authentication required.
    d. **Caching policy**: authoritative data is cached; refreshes limited to a maximum of five refreshes per 24-hour window to respect rate limits. On failure, serve the last-known cache.
2. Google Map API (mapsplatform.google.com/lp/maps-apis/)
    a. **Role**: map rendering, geocoding, and travel-time estimation across transport modes.
    b. **Auth**: API key with enforced quotas
    c. **Caching policy**: implement graceful fallback to cached estimates when quotas are exhausted.
    d. **Privacy constraint**: transmit only coordinates strictly required for the request; raw location storage is subject to the data retention policy.

2.4.3 Development and Local Test Environment

Local services (recommended):

- **Frontend**: React + Vite + TypeScript.
- **Backend**: Java Spring Boot.
- **Database**: local MySQL instance seeded with representative sample data (users, comments, replies, votes, favourites).

To set up, refer to the README.md in the repository.

2.4.3.1 Frontend: React.js + Vite with TypeScript

React.js is an open-source JavaScript library designed for building dynamic and responsive user interfaces, particularly for single-page applications. EduFinder's frontend leverages React's component-based architecture to create reusable UI elements such as search filters, school detail cards, and comment sections. Each component manages its own state, resulting in a modular and maintainable codebase.

Combined with Vite, a modern build tool, the frontend benefits from fast development server startup, hot module replacement, and optimized production builds.

TypeScript adds static typing to JavaScript, reducing runtime errors and improving developer productivity by enabling better tooling support and clearer contracts between components. Styling is handled with Tailwind CSS and Flowbite, ensuring a consistent, responsive, and accessible design across devices.

2.4.3.2 Backend: Java Spring Boot

Spring Boot is an open-source Java framework designed to simplify the development of robust, production-ready backend applications. EduFinder's backend uses Spring Boot to implement RESTful APIs that handle user authentication, school data retrieval, comment management, and voting logic.

Spring Boot's modular design allows developers to integrate only the necessary dependencies, keeping the application lightweight while still supporting advanced features such as security, validation, and exception handling. Its built-in support for dependency injection and layered architecture (controllers, services, repositories) ensures clean separation of concerns and maintainable code.

The backend also integrates with external API (e.g., Singapore Government Open Data API) but prioritizes **cached data** to minimize external calls, in line with system constraints. Spring Boot's scalability and strong ecosystem make it well-suited for EduFinder's long-term growth.

2.4.3.3 Database: MySQL

MySQL is an open-source, relational database management system (RDBMS) widely used for web applications requiring reliable, scalable, and structured data storage. Unlike lightweight embedded databases, MySQL operates as a dedicated server process, enabling concurrent access from multiple clients and supporting large-scale deployments. This makes it an ideal choice for EduFinder, where persistent storage of user accounts, saved schools, comments, replies, and votes is essential.

MySQL follows a client-server architecture, where the database server manages data storage, indexing, and query execution, while applications connect through standard protocols (e.g., JDBC). It supports SQL queries, transactions, and advanced indexing strategies, ensuring efficient retrieval of school data and user-generated content.

One of MySQL's key strengths is its ACID compliance (Atomicity, Consistency, Isolation, Durability), which guarantees reliable transactions and data integrity even under concurrent usage. Its mature ecosystem includes replication, clustering, and backup tools, making it suitable for both development and production environments.

In EduFinder, MySQL integrates seamlessly with Spring Boot through JPA repositories, abstracting database operations into simple repository interfaces. This reduces boilerplate code while maintaining flexibility for complex queries. MySQL's indexing and query optimization features ensure that frequent operations — such as filtering schools, retrieving comments, or fetching a user's saved favourites — are performed quickly and efficiently.

2.5 Design and Implementation constraints

2.5.1 Budget Constraint

Due to limited development budget, the use of advanced commercial APIs or paid cloud services is restricted.

2.5.2 Framework Constraint

The frontend must be implemented with React + Vite + TypeScript, and the backend must be implemented using Java Spring Boot, which is decided by the team.

2.5.3 Database Constraint

The system must use MySQL as the relational database, which is integrated through JPA repositories.

2.5.4 External API Dependency

EduFinder relies on real-time data from the Singapore Government Open Data API and Google Maps API. Any downtime, rate limits, or changes in these APIs can affect system performance.

2.5.5 Hosting Constraint

Currently, the system is developed and tested in a local environment due to limited resources. The architecture, however, is designed to be scalable and cloud-ready. Cloud or centralized hosting is considered a future deployment plan and is not within the current project scope.

2.5.6 Integration Constraint

The system must ensure compatibility between the external APIs and the internal database schema. Regular updates to maintain data consistency is required.

2.6 User Documentation

The User documentation will include:

2.6.1 User Manual (For User)

The core features of EduFinder and how to utilize them are as follows:

2.6.1.1 Searching For School

The searching for schools feature is the main functionality of EduFinder and can be accessed by both authenticated and unauthenticated users. When users first enter the landing page, all schools populate the results table. Users can

- Search School by Name by keying in the name in the search box
- Search School by Filter by opening the filter menu and selecting appropriate filters
- Deselect selected filters: Once filters are selected / deselected, the table automatically refreshes the table based on the filters.

2.6.1.2 Viewing School Details

Users can click into school pages and can view:

- School Details (e.g. CCAs, Mother Tongues Offered, Website etc.)
- Google Maps View of Schools Location: If users are authenticated and have keyed in their postal code, they can see travel routes (via car, public transport, bike and walking) from their homes to the school, and their respective travel times
- Comments of other users on the school, and their replies and vote summaries: Authenticated users are able to post comments, replies to other comments and upvote/downvote other comments

2.6.1.3 Saving Schools

Authenticated users can save schools from the landing page. They can

- Click on the bookmark icon to save the school
- Visit their Saved Schools Tab to see what schools they have saved
- Unsave schools from Saved Schools Tab as well as landing page

2.6.1.4 Comparing Schools

Authenticated users can compare between schools side-by-side. Users can

- Select saved schools to add to the comparison table
- Deselect saved schools from the comparison table
- Enter a Natural Language Prompt to generate a comparison result

2.7 Assumptions and Dependencies

2.7.1 Assumptions

The following conditions are presumed to hold true for EduFinder's operation. System functionality may be degraded if assumptions cannot meet.

1. User Permissions
    a. End users will grant the application access to device location when prompted, enabling features such as nearby school search and navigation.
2. Connectivity
    a. Users are expected to maintain a stable internet connection to interact with EduFinder's services.
3. External Service Availability
    a. Government open data APIs (e.g., MOE school dataset) will remain publicly accessible, updated regularly, and within their documented rate limits (maximum of five refreshes per day).
    b. Google Maps API will remain available within the limits of its free usage quota.
4. User Agreements
    a. All users will explicitly accept the Terms and Conditions and Privacy Policy before account creation.

2.7.1 Dependencies

1. Data Sources
    a. Singapore Government Open Data API provides authoritative school information.
    b. Google Maps API supplies map rendering and geolocation services.
2. Database Infrastructure
    a. A properly configured MySQL database is required to persist user accounts, comments, votes, and favourites.
3. Data Quality
    a. The accuracy and timeliness of external datasets are essential for generating meaningful school recommendations.
4. Regulatory Compliance
    a. EduFinder and its maintaining organization must comply with Singapore's PDPA for personal data storage and handling.
5. Security Mechanisms
    a. The system relies on secure communication protocols (HTTPS/TLS 1.2 or higher) and robust input validation to mitigate vulnerabilities such as XSS, SQL injection, and CSRF.

3. External Interface Requirements

3.1 User Interfaces

How users interact with each page is covered earlier in **Section 2.6.1 User Manual**.

3.1.1 Landing Page

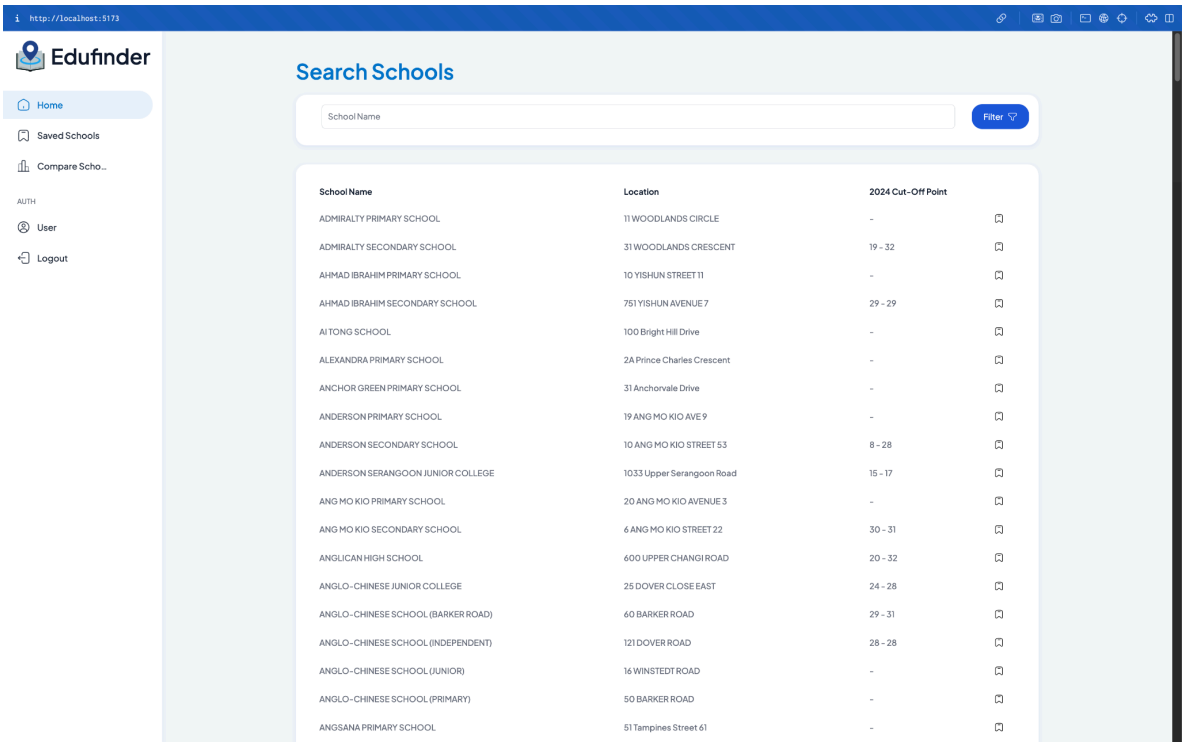


Figure 1: Landing Page when you click on the URL

3.1.2 Filter Pop-Up

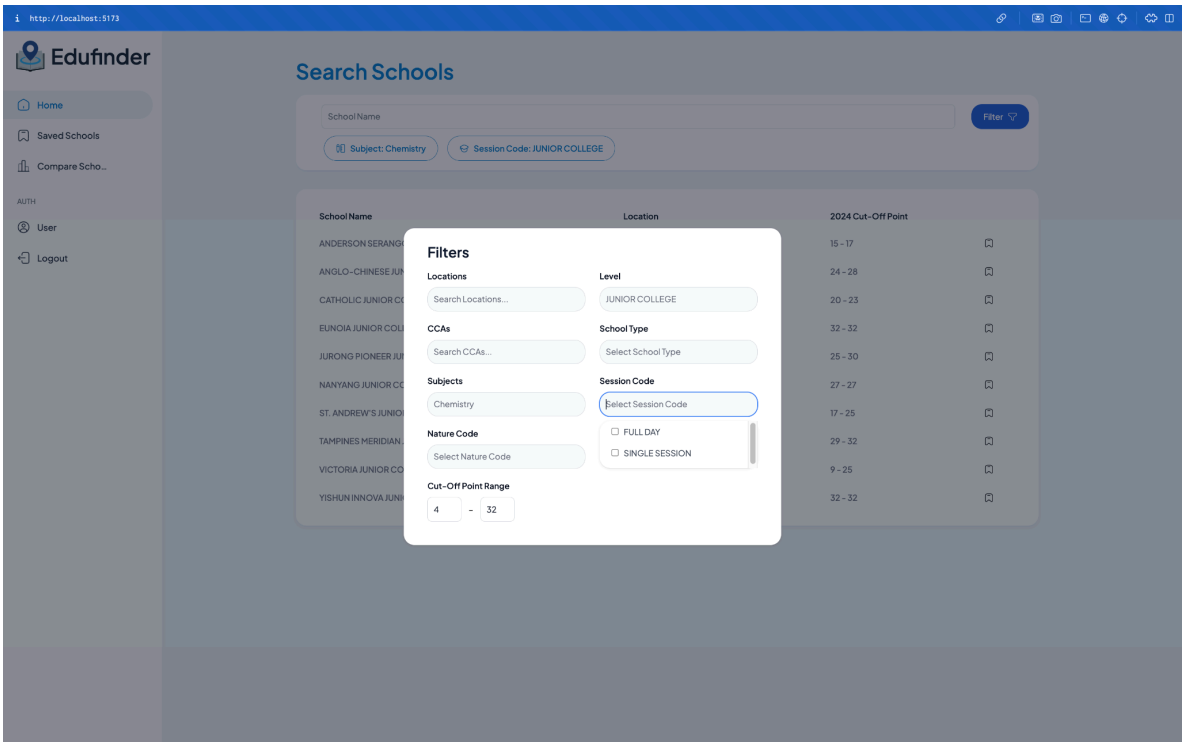


Figure 2: Filter Page to filter school based on preferences

3.1.3 School Page

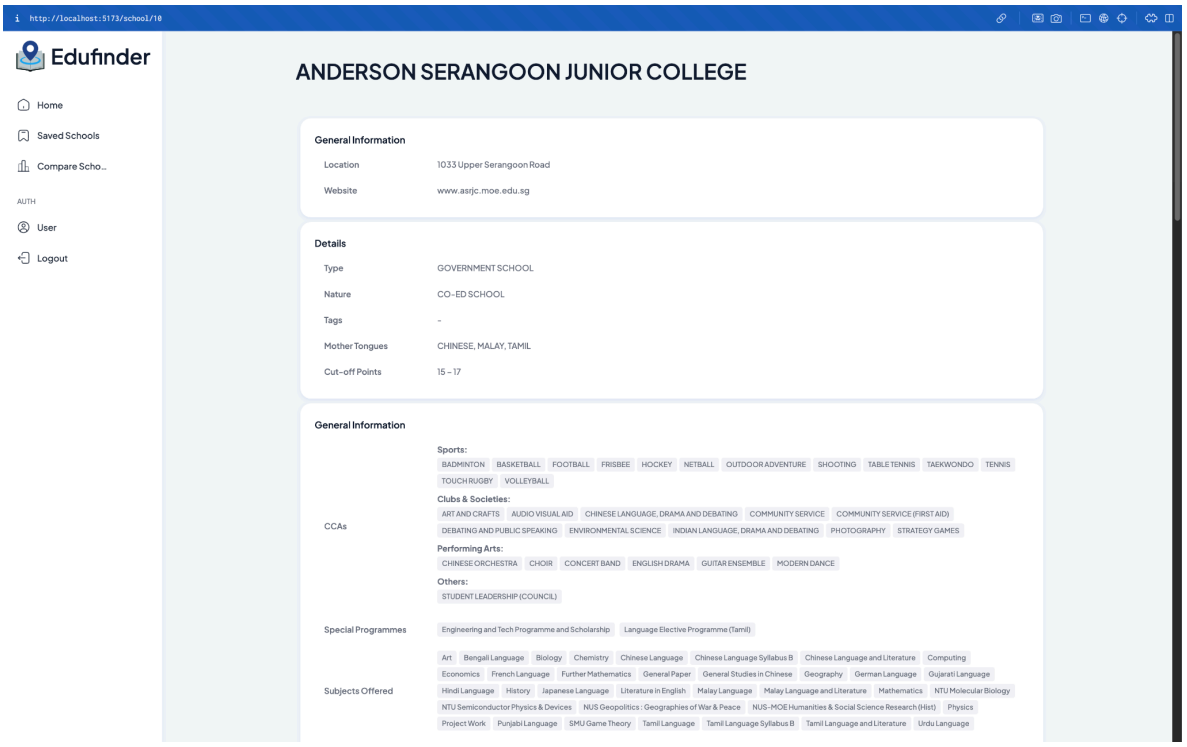


Figure 3: School Page (Top) displaying school general information

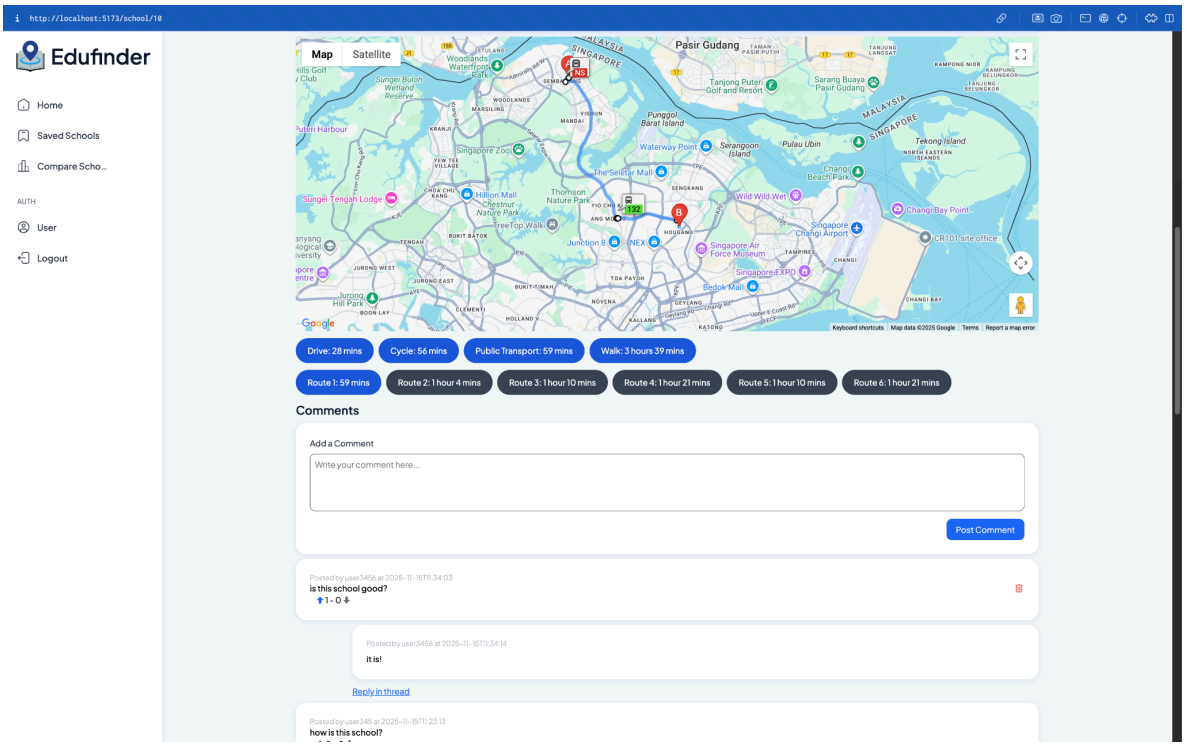


Figure 4: School Page (bottom) with map, directions and comment section

3.1.4 Saved Schools

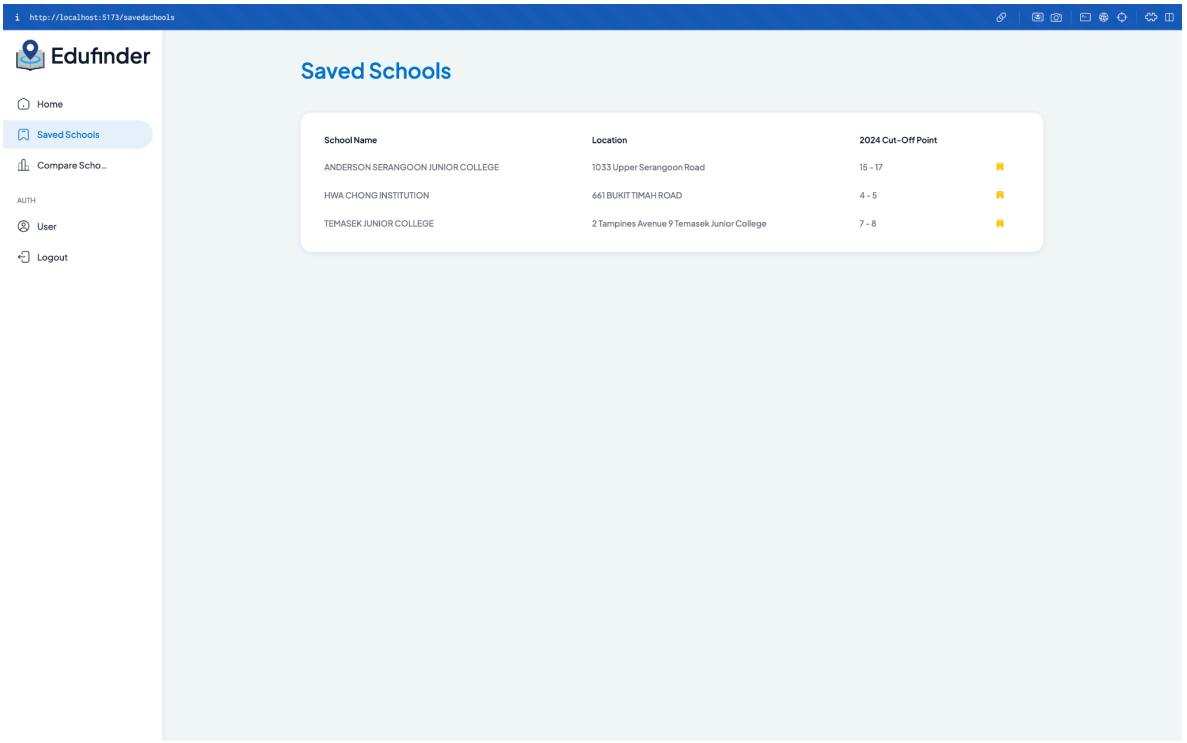


Figure 4: Saved Schools Page

3.1.5 Compare Schools Page

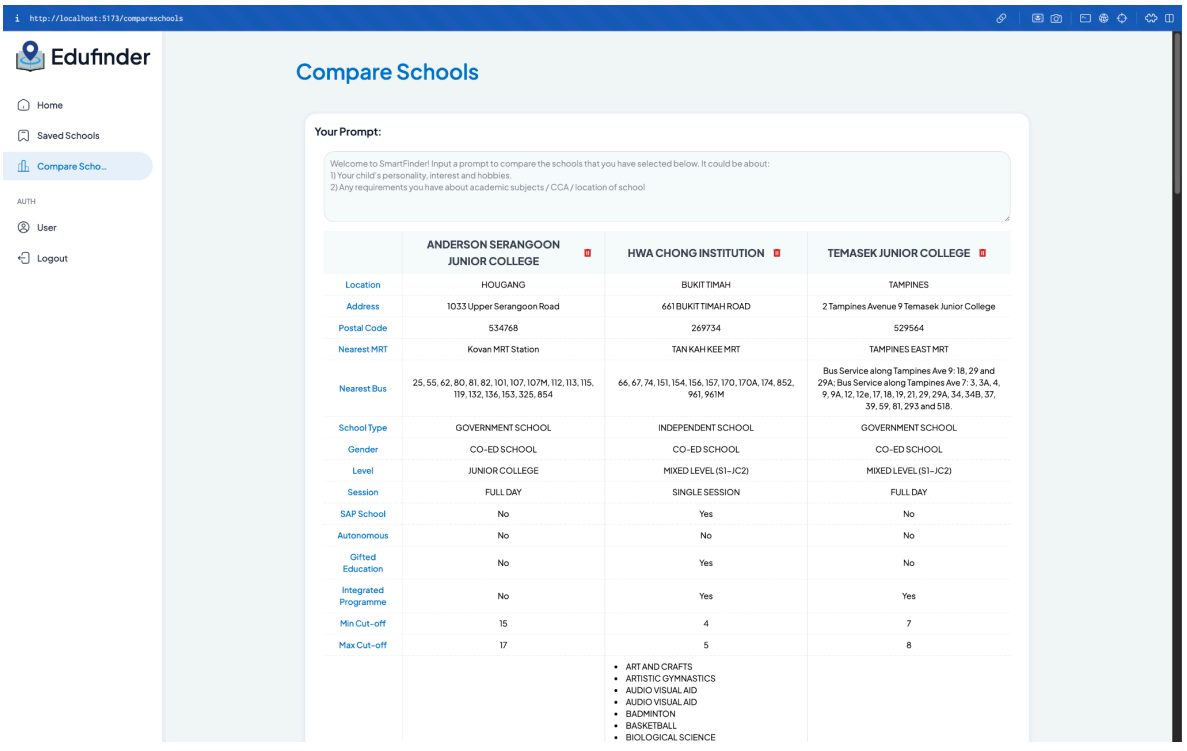


Figure 5: Compare Schools Page with Prompt Box

3.1.6 Login Page

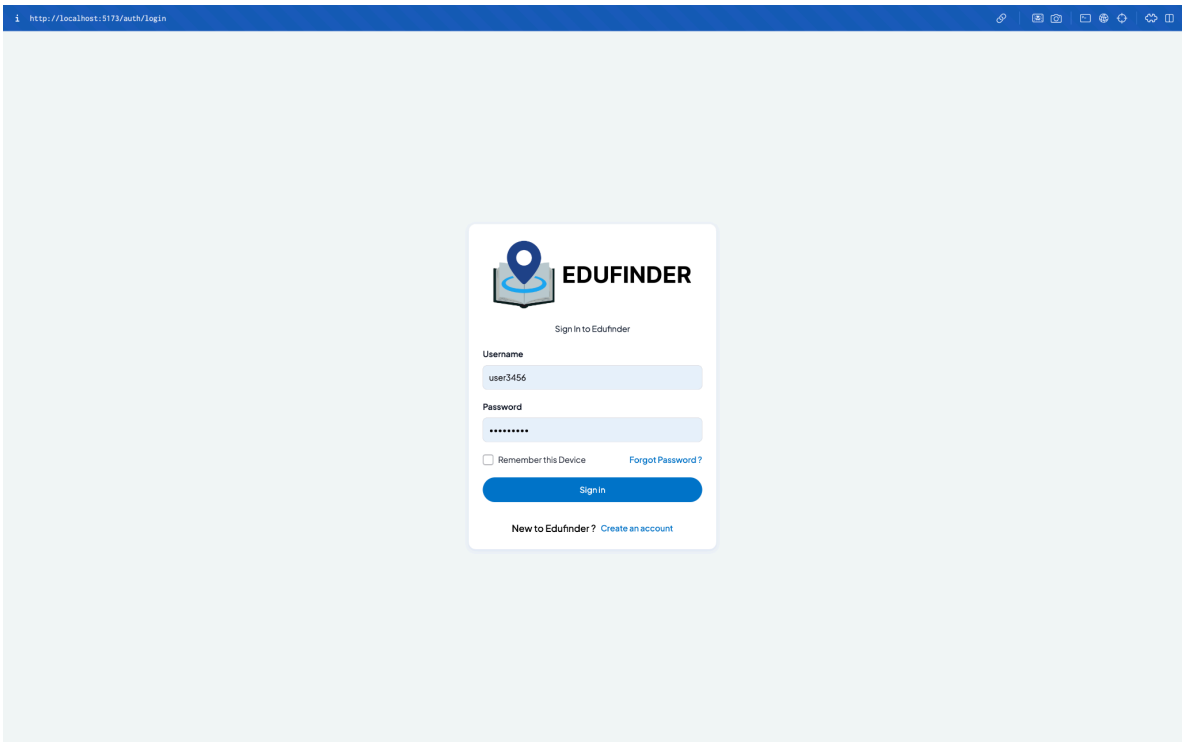


Figure 6: Login Page

3.1.7 Register Page

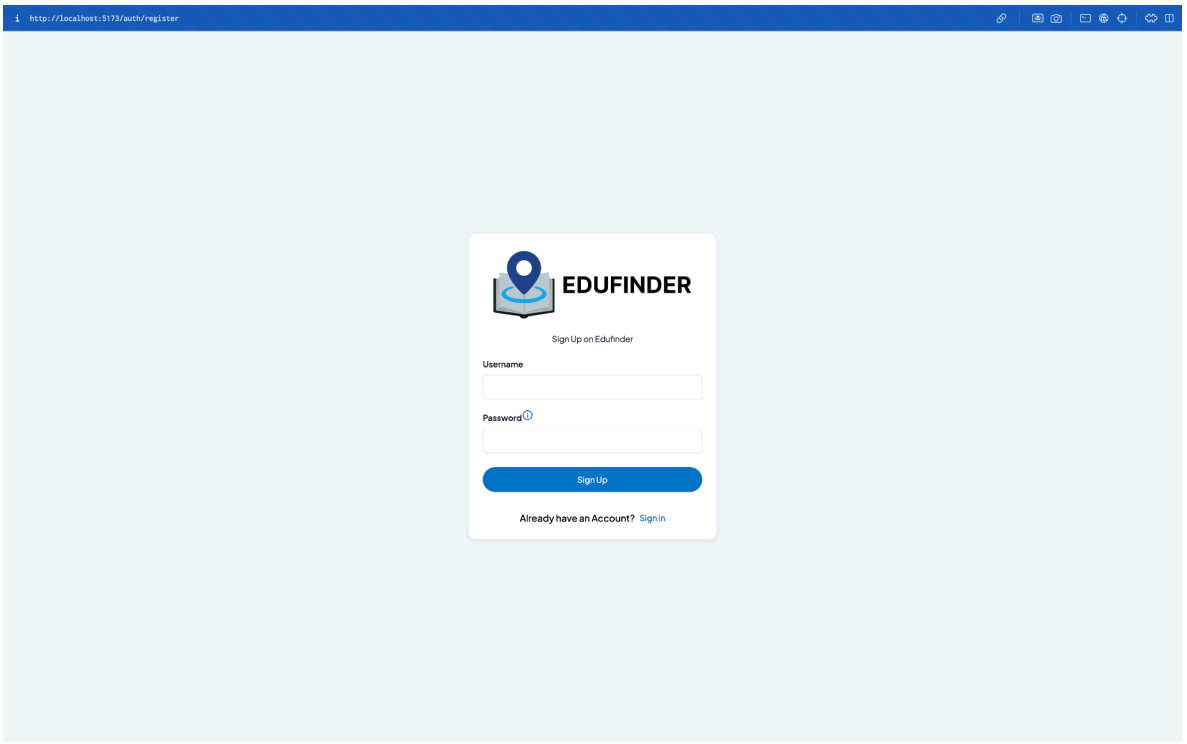


Figure 7: Register Page

3.1.8 User Info Page

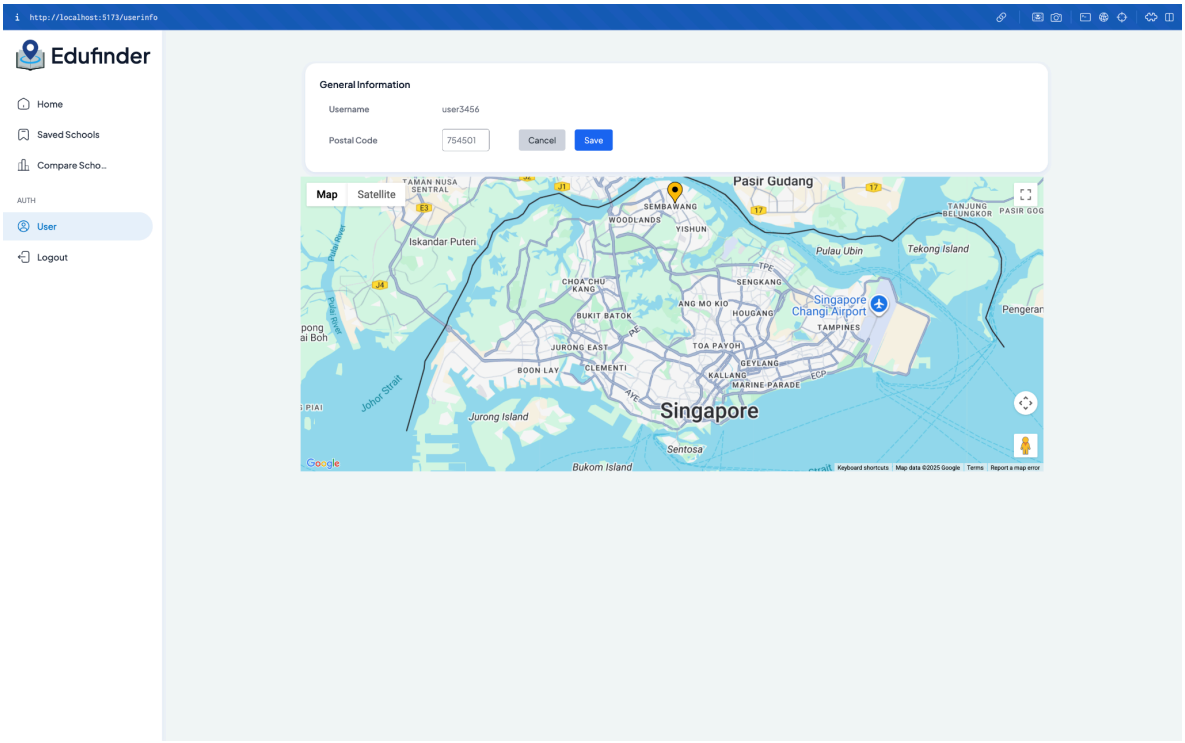


Figure 8: User Info Page

3.2 Hardware Interfaces

Edufinder is implemented to operate primarily as a web application, minimizing the need for direct hardware interfaces. However, the following highlights the logical and physical characteristics of the interfaces between the software and the hardware.

1. **Device types**

   The software is designed to be compatible with various devices, preferably those with access to the Internet, such as laptops, desktop computers, tablets and smartphones.

2. **Data and Control Interactions**

   The user's device interacts with the software through standard input/output hardwares, such as keyboards, touchscreens, and mice for input; displays for output; and network interface cards for internet connectivity.

3.3 Software Interfaces

There are a number of connections between the multiple softwares implemented to allow our web applications to function smoothly and properly.

Front-End Development:

- React JS was utilized for user interface rendering. Our system integrates with react js to provide rich interactive controls and smooth in-app navigation and animations, all to better user experience.
- Tailwind CSS(Cascading style sheets) is chosen to give a utility heavy and highly responsive styling of the webpage.
- Typescript was implemented to enforce static typing and maintainability across all the UI logic.

We have made sure that all these software can go hand-in-hand to provide a seamless client-side navigation in the front-end aspect of the web application.

Back-End Integration:

- Java executes all the business logic and functional workflows essential for the web application.
- Java cannot work on its own. This is where Spring Boot comes in. Spring Boot supplies the framework for API endpoints, database integration and overall server orchestration.

This stack enables a robust, scalable and accessible backend architecture for processing data and communicating with the front-end.

Database Connectivity:

- mySQL is used to generate primary relational databases to persist all structures application data, such as usernames, passwords and school information. We used mySQL because it provides reliable storage, referential integrity and optimized querying via SQL.

APIs:

- Data.gov.sg API

  This API allows developers to fetch up-to-date public datasets, in this case school data and information through REST calls, enabling automated ingestion of structured data without manual download. This is highly secure as government data is encrypted correctly.

- Google Maps API

  This API provides developers with a powerful suite of tools to integrate interactive maps, geolocation, and navigation features into the application. It enables access to high quality, customizable map data and functionalities, allowing for developers to create dynamic user experiences tailored to the requirements of the web application.

3.4 Communications Interfaces

For EduFinder, diverse communication interfaces are vital in ensuring a seamless, hassle-free and fluid user experience.

- Web Socket: Utilization of Web Socket protocol to exchange real-time, bi-directional data between the client and the server. This is very crucial for customer service support features, allowing for instant communication between the users and the support panel.
- HTTP/HTTPS: All interactions and sensitive data transmissions with EduFinder will happen over the HTTP/HTTPS protocols. This ensures alignment with modern web servers, message integrity and encrypted data transmission.
- API Communication: The conventions we will be utilizing are the RESTful API conventions, with JSON-formatted primary data interchange. We have used APIs from Google for maps integrations, and MOE for school data collection.
- NLP Chatbot Integration: NLP is integrated for enhanced user interactions. Our chatbot communicates with backend services through API calls, enabling automated query handling or user support. We attain this through clearly defined user inputs, and request/response payloads.
- Network Protocols: Standard network protocols such as TCP/IP will be used for network communication, following the criteria for internet communication.

4. System Features

4.1 Query School

4.1.1 Description and Priority

Users should be able to query Edufinder for schools using filters they have selected. This feature has high priority as users need to be able to apply respective filters to find relevant schools to them.

4.1.2 Stimulus/Response Sequences

Use Case ID:	UCQ-1		
Use Case Name:	Query School		
Created By:	Yong Chee Seng	Last Updated By:	Yong Chee Seng
Date Created:	4 September 2025	Date Last Updated:	15 November 2025

Actor:	User (Initiating)
Description:	Users shall be able to query schools based on different criteria, and the system shall show a list of schools that satisfy all the criteria.
Preconditions:	1. The system initiated **UCSY-1 Fetch School Data from Government API** and retrieved the latest school dataset.
Postconditions:	1. The system displays a list of schools that match all selected criteria or shows “no results” if none is found.
Priority:	High
Frequency of Use:	Estimated 50–200 per day across all users
Flow of Events:	1. The user navigates to the “Query School” page. 2. The system displays the full list of schools. 3. The system displays available query criteria (e.g., name, region, subject, CCA). 4. The user enters one or more criteria. 5. The user submits the query. 6. The system validates the query input. 7. The system filters the school data according to criteria. 8. The system displays a list of matching schools.
Alternative Flows:	AFS6: **No Criteria Entered** 1. The system continues showing all schools. AFS8: **No Results Found** 1. If no school satisfies all the criteria, the system displays the message “No result found”.

Exceptions:	NA
Includes:	1. UCSY-1 Fetch School Data from Government API
Special Requirements:	1. The system shall allow selection of criteria from predefined options where applicable (e.g., when filtering by subject, the system shall provide a list of available subjects instead of requiring manual text input). 2. The system shall complete the filter within 2 seconds under normal load condition.
Assumptions:	1. The **"Query School"** page is the main entry point of the application.
Notes and Issues:	NA

4.1.3 Functional Requirements

REQ-1. The system shall display a list of all schools in the home page.

REQ-2. The user shall be able to query the system for schools using one or multiple search requirements.

REQ-2.1. If one or more search requirements are provided, the system shall only include schools that satisfy all search requirements in the results.

REQ-2.2. The user shall be able to filter schools by name.

REQ-2.2.1. The name input shall be a text string of 0–512 characters.

REQ-2.2.2. The name input shall be a string of 0 characters by default.

REQ-2.2.3. If 0 characters are provided (empty input), the query shall not filter by name.

REQ-2.2.4. If 1 or more characters are provided, the system shall include only schools whose name contains the input substring (case-insensitive) in the results.

REQ-2.3. The user shall be able to filter schools based on their location.

REQ-2.3.1. The location input shall be a multi-choice dropdown with all locations available.

REQ-2.3.2. The location input shall be empty (no selected locations) by default.

REQ-2.3.3. If 0 locations are provided, the query shall not filter by location.

REQ-2.3.4. If 1 or more locations are provided, the system shall include only schools whose location is in the selected locations in the results.

REQ-2.4. The user shall be able to filter schools based on their Co-Curricular Activities (CCA) offered.

REQ-2.4.1. The CCA input shall be a multi-choice dropdown with all CCA available.

REQ-2.4.2. The CCA input shall be empty (no selected CCA) by default.

REQ-2.4.3. If 0 CCA are provided, the query shall not filter by CCA.

REQ-2.4.4. If 1 or more CCA are provided, the system shall include only schools whose 1 or more CCAs offered are in the selected CCA in the results.

REQ-2.5. The user shall be able to filter schools based on their subjects offered.

REQ-2.5.1. The subjects input shall be a multi-choice dropdown with all subjects available.

REQ-2.5.2. The subjects input shall be empty (no selected subjects) by default.

REQ-2.5.3. If 0 subjects are provided, the query shall not filter by subjects.

REQ-2.5.4. If 1 or more subjects are provided, the system shall include only schools whose 1 or more subjects offered are in the selected subjects in the results.

REQ-2.6. The user shall be able to filter schools by cut-off point range.

REQ-2.6.1. The cut-off point range shall consist minimum cut-off point and maximum cut-off point.

REQ-2.6.2. The minimum cut-off point input shall be an integer within 4-32.

REQ-2.6.2.1. The minimum cut-off point input shall be 4 by default.

REQ-2.6.3. The maximum cut-off point input shall be an integer within 4-32.

REQ-2.6.3.1. The maximum cut-off point input shall be larger than or equal to the minimum cut-off point.

REQ-2.6.3.2. The maximum cut-off point input shall be 32 by default.

REQ-2.6.4. The system shall include only schools whose cut-off points are within minimum and maximum cut-off point (inclusive) in the results.

REQ-2.7. The user shall be able to filter schools based on their level.

REQ-2.7.1. The level input shall be a multi-choice dropdown with all levels available.

REQ-2.7.2. The level input shall be empty (no selected level) by default.

REQ-2.7.3. If 0 levels are provided, the query shall not filter by level.

REQ-2.7.4. If 1 or more levels are provided, the system shall include only schools whose level is in the selected levels in the results.

REQ-2.8. The user shall be able to filter schools based on their nature code.

REQ-2.8.1. The nature code input shall be a multi-choice dropdown with all nature codes available.

REQ-2.8.2. The nature code input shall be empty (no selected nature code) by default.

REQ-2.8.3. If 0 nature codes are provided, the query shall not filter by nature code.

REQ-2.8.4. If 1 or more nature codes are provided, the system shall include only schools whose nature code is in the selected nature codes in the results.

REQ-2.9. The user shall be able to filter schools based on their type.

REQ-2.9.1. The type input shall be a multi-choice dropdown with all types available.

REQ-2.9.2. The type input shall be empty (no selected type) by default.

REQ-2.9.3. If 0 types are provided, the query shall not filter by type.

REQ-2.9.4. If 1 or more types are provided, the system shall include only schools whose type is in the selected types in the results.

REQ-2.10. The user shall be able to filter schools based on their session codes.

REQ-2.10.1. The session code input shall be a multi-choice dropdown with all session codes available.

REQ-2.10.2. The session code input shall be empty (no selected session code) by default.

REQ-2.10.3. If 0 session codes are provided, the query shall not filter by session code.

REQ-2.10.4. If 1 or more session codes are provided, the system shall include only schools whose session code is in the selected session codes in the results.

4.2 View School Details

4.2.1 Description and Priority

After querying using filters, or just selecting a school that shows up initially, users should be able to click into a school and find more details about it, including comments on the school's page and replies to them . This feature is of high priority as users need to evaluate a school's suitability by examining more of its attributes and the public's perception of the school through comments and replies.

4.2.2 Stimulus/Response Sequences

Use Case ID:	UCD-1		
Use Case Name:	View School Details		
Created By:	Yong Chee Seng	Last Updated By:	Yong Chee Seng
Date Created:	4 September 2025	Date Last Updated:	15 November 2025

Actor:	User (Initiating)
Description:	The user shall be able to view the details of a selected school, and the system shall display the full information of that school.
Preconditions:	1. The system has received the identifier of the selected school (e.g., unique ID). 2. The system initiated **UCSY-1 Fetch School Data from Government API** and retrieved the latest school dataset.
Postconditions:	1. The system displays the full details of the selected school. 2. The system displays any available user comments associated with the school. 3. The system displays any available user replies to the comments associated with the school.
Priority:	High
Frequency of Use:	Estimated 100–500 per day across all users
Flow of Events:	1. The user selects a school from the query results (or other entry points). 2. The system displays the full details of the selected school. 3. The system retrieves users' comments related to the school, and the replies to these comments. 4. System displays the retrieved user comment and replies
Alternative Flows:	AFS2: **School Not Found** 1. If a school with the identifier is not found, the system displays error "School not found".

	AFS4: **Comments Or Replies Unavailable** 1. If comments or replies cannot be retrieved, the system displays error "Comments / Replies unavailable at the moment".
Exceptions:	NA
Includes:	1. UCSY-1 Fetch School Data from Government API
Special Requirements:	1. The system shall show the details within 2 seconds under normal load condition.
Assumptions:	NA
Notes and Issues:	NA

4.2.3 Functional Requirements

REQ-1: The user shall be able to view school details when selecting a school from the query results.

REQ-1.1: The system shall display the general information of school:

REQ-1.1.1: The system shall display the school's name.

REQ-1.1.2: The system shall display the school's level.

REQ-1.1.3: The system shall display the school's type.

REQ-1.1.4: The system shall display the school's nature code.

REQ-1.1.5: The system shall display the school's session code.

REQ-1.1.6: The system shall display the school's cut-off point range.

REQ-1.1.7: The system shall display the school's mother tongue.

REQ-1.1.8: The system shall display the school's tags.

REQ-1.1.8.1: The system shall display whether the school is SAP.

REQ-1.1.8.2: The system shall display whether the school is autonomous.

REQ-1.1.8.3: The system shall display whether the school is gifted.

REQ-1.1.8.4: The system shall display whether the school is IP.

REQ-1.2: The system shall display geographical information of the school:

REQ-1.2.1: The system shall display the school's address.

REQ-1.2.2: The system shall display the school's postal code.

REQ-1.2.3: The system shall display the school's location.

REQ-1.2.4: The system shall display the school's nearby MRT station(s).

REQ-1.2.5: The system shall display the school's nearby bus station(s).

REQ-1.2.6: The system shall display an interactive map interface showing the school's location.

REQ-1.2.7: If the user is authenticated and the user profile includes valid postal code, the system shall display route information from the school to the user's postal code.

REQ-1.2.7.1: The system shall display the computed travel route on the interactive map.

REQ-1.2.7.2: The system shall display the estimated travel time for the selected route.

REQ-1.2.7.3: The system shall provide route information for driving.

REQ-1.2.7.4: The system shall provide route information for cycling.

REQ-1.2.7.5: The system shall provide route information for public transport.

REQ-1.2.7.6: The system shall provide route information for walking.

REQ-1.3: The system shall display the contact information of the school:

REQ-1.3.1: The system shall display the school's official website as a hyperlink.

REQ-1.3.2: The system shall display the school's email address as a hyperlink that opens the user's default mail client.

REQ-1.3.3: The system shall display the school's phone number as a hyperlink that initiates a call on supported devices.

REQ-1.3.4: The system shall display the school's fax number.

REQ-1.4: The system shall display the subjects offered by the school.

REQ-1.4.1: The system shall display subjects in ascending lexicographical order.

REQ-1.5: The system shall display the CCAs offered by school.

REQ-1.5.1: The system shall display CCAs in ascending lexicographical order.

REQ-1.6: The system shall display the MOE programmes offered by school.

REQ-1.6.1: The system shall display MOE programmes in ascending lexicographical order.

REQ-2: The system shall have a comment section under the school detail page.

REQ-2.1: The system shall display all comments related to the specific school.

REQ-2.1.1: The system shall display comments in descending chronological order.

REQ-2.1.2: The system shall display upvote count for each comment.

REQ-2.1.3: The system shall display downvote count for each comment.

REQ-2.1.4: If the user is authenticated, the system shall display the user's vote.

4.3 Save School

4.3.1 Description and Priority

Users should be able to save schools they are more interested in and visit frequently. This feature is **medium-high** priority as users need to save schools for the feature of comparing schools.

4.3.2 Stimulus/Response Sequences

Use Case ID:	UCS-4		
Use Case Name:	Save School as Favourite		
Created By:	Yong Chee Seng	Last Updated By:	Yong Chee Seng
Date Created:	4 September 2025	Date Last Updated:	15 November 2025

Actor:	User (Initiating)
Description:	The user shall be able to save a school as favourite.
Preconditions:	1. The user has logged in to an account. 2. The user has navigated to a valid entry point (e.g., Query School).
Postconditions:	1. The system saves the school. 2. The system visually indicates the saved status of the school.
Priority:	Medium
Frequency of Use:	Estimated 5-10 per day across all users
Flow of Events:	1. The user navigates to a valid entry point. 2. The user selects a school to save as favourite. 3. The system stores the school as a favourite for the user. 4. The system displays a clear sign the school is saved as favourite.
Alternative Flows:	AFS3a: **User Not Logged In** 1. The system displays the message “You need to login to save the school”. AFS3b: **User Already Saved as Favourite** 1. If the user has already saved the school previously, the system removes it from saved schools and updates the UI indicator. AFS4: **System Error** 1. If the school cannot be stored or removed due to a system error, system displays: “Unable to save school. Please try again later.”

Exceptions:	NA
Includes:	UCA-1 Login
Special Requirements:	1. The system shall complete the process within 5 seconds under normal load conditions.
Assumptions:	NA
Notes and Issues:	NA

4.3.3 Functional Requirements

REQ-1: The user shall be able to save a school as favourite.

REQ-1.1: The system shall enforce authentication to save as favourite.

REQ-1.2: If the user has not previously saved the school as a favourite, the system shall add the school to the user's favourites.

REQ-1.3: If the user has already saved the school as a favourite, the system shall remove the school from the user's favourites.

REQ-1.4: The user shall be able to view a list of their saved schools on a dedicated Favourites page.

4.4 Compare Schools

4.4.1 Description and Priority

Users should be able to select two or more schools that they have saved and compare between the attributes of the schools selected. They should be able to also input a Natural Language prompt to add context to their situation, such that the system is able to better recommend them a suitable school. The priority for this is **medium-high** as while it is useful for users to compare schools, it is not the priority of the system.

4.4.2 Stimulus/Response Sequences

Use Case ID:	UCD-2		
Use Case Name:	Compare School		
Created By:	Yong Chee Seng	Last Updated By:	Yong Chee Seng
Date Created:	4 September 2025	Date Last Updated:	15 November 2025

Actor:	User (Initiating)
Description:	The user shall be able to view and compare the details of 2 or more selected schools side by side.
Preconditions:	1. The user has logged in to an account. 2. The user has more than 2 schools saved to his account. 3. The system initiated **UCSY-1 Fetch School Data from Government API** and retrieved the latest school dataset.
Postconditions:	1. The system displays the full details of all selected schools side by side. 2. System suggests the most suitable school after taking into account the user's Natural Language input
Priority:	Medium-High
Frequency of Use:	Estimated 10–50 per day across all users
Flow of Events:	1. User selects more than 2 schools from their saved schools 2. The system displays the full details of the all selected schools side by side. 3. The system prompts user to input a Natural Language prompt to add context to their situation 4. The system suggests the most suitable school from the selected schools and justifies based on the user's prompt.
Alternative Flows:	NA
Exceptions:	NA
Includes:	1. UCSY-1 Fetch School Data from Government API 2. UCA-1 Login

Special Requirements:	1. The system shall show the details within 2 seconds under normal load condition. 2. The system shall present the comparison of both schools in a responsive layout that avoids UI overlay issues or excessive horizontal scrolling across all supported devices. 3. The system shall return the LLM-generated text within 30 seconds of submitting the user's prompt.
Assumptions:	NA
Notes and Issues:	NA

4.4.3 Functional Requirements

REQ-1: The user shall be able to compare 2 or more schools side by side.

REQ-1.1: The user shall be able to select 2 or more schools from their saved schools.

REQ-1.2: The system shall display the attributes of the selected schools side-by-side.

REQ-2: The user shall be able to initiate school comparison via natural language queries (NLP for comparing schools).

REQ-2.1: The system shall accept natural-language comparison requests of 1 to 256 characters.

REQ-2.2: The system shall process the NLP query to identify target schools.

REQ-2.2.1: The system shall display a paragraph regarding the reasons.

4.5 Login

4.5.1 Description and Priority

Users should be able to login to their account if they have previously signed up. The priority for this is **medium-high** as users can still browse Edufinder even without logging in, albeit with fewer features.

4.5.2 Stimulus/Response Sequences

Use Case ID:	UCA-1		
Use Case Name:	Login		
Created By:	Yong Chee Seng	Last Updated By:	Yong Chee Seng
Date Created:	4 September 2025	Date Last Updated:	15 November 2025

Actor:	User (Initiating)
Description:	The user shall be able to login to their account by providing valid credentials.
Preconditions:	1. The user has signed up for an account.
Postconditions:	1. The system recognizes the user as authenticated and grants access to logged-in user functionalities. 2. The system displays the account username in the UI.
Priority:	Medium-High
Frequency of Use:	Estimated 5-10 per day across all users
Flow of Events:	1. The user navigates to the “Login” page. 2. User inputs username and password. 3. The system retrieves the user account information corresponding to the entered username. 4. If the user exists, the system validates the entered password against the stored password. 5. If the password is correct, the system creates a session for the authenticated user. 6. The system displays the message “Login successfully”. 7. The system redirects the user to the home page.
Alternative Flows:	AFS4a: **User Not Found** 1. If the entered username does not exist, the system displays error “Wrong username or password”. AFS4b: **User Unavailable** 1. If the system cannot access user data, the system displays error “Login unavailable. Please try again later”.

	AFS5: **Incorrect Password** 1. If the password is incorrect, the system displays error "Wrong username or password".
Exceptions:	NA
Includes:	NA
Special Requirements:	1. The system shall complete the login process within 5 seconds under normal load conditions.
Assumptions:	NA
Notes and Issues:	NA

4.5.3 Functional Requirements

REQ-1: The user shall be able to log into their account.

REQ-1.1: The user shall be able to log into their account with valid credentials.

REQ-2: The system shall save user sessions using a secure method after successful login.

REQ-2.1: The system shall automatically invalidate user sessions no later than 7 days after the last successful login..

REQ-3: The user shall be able to log out of their account.

REQ-3.1: The system shall invalidate user sessions after logout.

4.6 Sign Up

4.6.1 Description and Priority

Users should be able to sign up for an account if they do not have an existing one. The priority for this is **medium-high** as users can still browse Edufinder even without creating an account, albeit with fewer features.

4.6.2 Stimulus/Response Sequences

Use Case ID:	UCA-2		
Use Case Name:	Signup		
Created By:	Yong Chee Seng	Last Updated By:	Yong Chee Seng
Date Created:	4 September 2025	Date Last Updated:	15 November 2025

Actor:	User (Initiating)
Description:	The user shall be able to sign up an account by providing valid credentials.
Preconditions:	NA
Postconditions:	1. The system stores user credentials securely. 2. The system allows user to login with the provided credentials
Priority:	Medium-High
Frequency of Use:	Estimated 3-5 per day across all users
Flow of Events:	1. The user navigates to the “Signup” page. 2. The user inputs username, password, and confirm password. 3. The system checks whether the username is already taken. 4. If the username is available, the system validates that the password meets the security policy (e.g., minimum length of 8 characters, includes uppercase, lowercase, number, and special symbol). 5. If the password is secure, the system verifies that the password and confirms the password match. 6. If the confirm password is the same as the password, the system stores user credentials. 7. The system displays the message “Account created. You can login now”. 8. The system redirects the user to the “Login” page.

Alternative Flows:	AFS4a: **Username Taken** 1. If the entered username exists, the system displays error "Username already exists. Please choose another". AFS4b: **User Unavailable** 1. If the system cannot access user data, the system displays error "Signup unavailable. Please try again later". AFS5: **Weak Password** 1. If the password does not meet the security policy, the system displays error: "Password does not meet security requirements". AFS6: **Password Mismatch** 1. If the password and confirm password do not match, the system displays the error: "Passwords do not match". AFS7: **System Error** 1. If credentials cannot be stored due to a system error, system displays error: "Signup unavailable. Please try again later".
Exceptions:	NA
Includes:	NA
Special Requirements:	1. The system shall enforce a strong password policy (e.g., minimum length of 8 characters, including uppercase, lowercase, numbers, and special symbols). 2. The system shall complete the signup process within 5 seconds under normal load conditions. 3. The system shall store all user passwords using industry-standard encryption methods (e.g., bcrypt). 4. The system shall ensure that users explicitly agree to the **Terms and Conditions** and **Privacy Policy** in a clear and visible manner before account creation.
Assumptions:	NA
Notes and Issues:	NA

4.6.3 Functional Requirements

REQ-1: The user shall be able to create an account.

REQ-1.1: The user shall provide a username.

REQ-1.1.1: The username shall be a string of 6-14 characters.

REQ-1.1.2: The username shall be unique across the user database.

REQ-1.2: The user shall provide a password.

REQ-1.2.1: The password shall be a string of at least 8 characters.

REQ-1.2.2: The password shall meet minimum complexity requirements:

REQ-1.2.2.1: The password shall have at least 1 lowercase letter.

REQ-1.2.2.2: The password shall have at least 1 uppercase alphabet.

REQ-1.2.2.3: The password shall have at least 1 number.

REQ-1.2.2.4: The password shall have at least 1 non-alphanumeric character.

4.7 Create Comment

4.7.1 Description and Priority

Users should be able to comment on school pages. The priority for this is **medium-high** as users should be able to give valuable insights for other users to see on the school pages. It is quite common for users to want to comment on the schools.

4.7.2 Stimulus/Response Sequences

Use Case ID:	UCS-1		
Use Case Name:	Create Comment		
Created By:	Yong Chee Seng	Last Updated By:	Yong Chee Seng
Date Created:	4 September 2025	Date Last Updated:	15 November 2025

Actor:	User (Initiating)
Description:	The user shall be able to create and submit a comment under any “School Details” page.
Preconditions:	1. The user has logged in to an account. 2. The user has navigated to a valid “School Details” page.
Postconditions:	1. The system stores the submitted comment. 2. The system displays the new comment under the related school’s comments section.
Priority:	Low
Frequency of Use:	Estimated 3-5 per day across all users
Flow of Events:	1. The user navigates to the “School Details” page. 2. The user inputs a comment into the comment input field. 3. The user submits the comment. 4. The system validates the input (e.g., not empty, within allowed length). 5. If valid, the system stores the comment. 6. The system refreshes or updates the page. 7. The system displays the newly created comment in the comments section.
Alternative Flows:	AFS2: **User Not Logged In** 1. The system displays the message “You need to login to comment”. AFS5: **Invalid Input** 1. The system displays a clear error message explaining why input is invalid (e.g., empty or too long).

	AFS6: **System Error** 1. If a comment cannot be stored due to a system error, the system displays error: “Create comment unavailable. Please try again later”.
Exceptions:	NA
Includes:	UCA-1 Login
Special Requirements:	1. The system shall complete the comment creation process within 5 seconds under normal load conditions. 2. System shall protect against automated/bot attack (e.g., reCAPTCHA or an equivalent). 3. The system shall sanitize and validate user input to prevent malicious attacks (e.g., XSS, SQL injection).
Assumptions:	NA
Notes and Issues:	NA

4.7.3 Functional Requirements

REQ-1: The user shall be able to create comments.

REQ-1.1: The system shall enforce authentication to create comments.

REQ-1.2: Each comment shall be associated with a specific school.

REQ-1.3: The comment shall be a string with at most 1024 characters.

4.8 Reply to Comment

4.8.1 Description and Priority

Users should be able to reply to other comments in school pages. The priority for this is **medium** as the frequency of people replying will not be very high. Many people will express their sentiments about the comment by voting on it rather than replying.

4.8.2 Stimulus/Response Sequences

Use Case ID:	UCS-2		
Use Case Name:	Reply Comment		
Created By:	Yong Chee Seng	Last Updated By:	Lim Kiat Yang Ryan
Date Created:	4 September 2025	Date Last Updated:	4 Nov 2025

Actor:	User (Initiating)
Description:	The user shall be able to reply to a comment under any “School Details” page.
Preconditions:	1. The user has logged in to an account. 2. The user has navigated to a valid “School Details” page.
Postconditions:	1. The system stores the submitted reply. 2. The system displays the new reply under the related comments.
Priority:	Low
Frequency of Use:	Estimated 3-5 per day across all users
Flow of Events:	1. The user navigates to the “School Details” page. 2. The user selects to reply to a comment. 3. User inputs reply into the input field. 4. The user submits the reply. 5. System validates the input (e.g., not empty, within allowed length). 6. If valid, the system stores the reply. 7. The system refreshes or updates the page. 8. The system displays the newly created reply under the related comment.
Alternative Flows:	AFS3: **User Not Logged In** 1. The system displays the message “You need to login to reply”.

	AFS6: **Invalid Input** 1. The system displays a clear error message explaining why input is invalid (e.g., empty or too long). AFS7: **System Error** 1. If reply cannot be stored due to a system error, system displays error: “Reply unavailable. Please try again later”.
Exceptions:	NA
Includes:	UCA-1 Login
Special Requirements:	1. The system shall complete the reply creation process within 5 seconds under normal load conditions. 2. System shall protect against automated/bot attack (e.g., reCAPTCHA or an equivalent). 3. The system shall sanitize and validate user input to prevent malicious attacks (e.g., XSS, SQL injection).
Assumptions:	NA
Notes and Issues:	NA

4.8.3 Functional Requirements

REQ-1: The user shall be able to reply to comments.

REQ-1.1: The system shall enforce authentication to reply to comments.

REQ-1.2: Each reply shall be associated with a specific comment.

REQ-1.3: The reply shall be a string with at most 1024 characters.

4.9 Vote Comment

4.9.1 Description and Priority

Users should be able to vote on comments. The priority for this is **medium-high** as the voting mechanism is important to let other users know the quality of a comment. Furthermore, users are more likely to vote than to reply to a comment.

4.9.2 Stimulus/Response Sequences

Use Case ID:	UCS-3		
Use Case Name:	Vote Comment		
Created By:	Yong Chee Seng	Last Updated By:	Lim Kiat Yang Ryan
Date Created:	4 September 2025	Date Last Updated:	4 November 2025

Actor:	User (Initiating)
Description:	The user shall be able to upvote or downvote a comment under any “School Details” page.
Preconditions:	1. The user has logged in to an account. 2. The user has navigated to a valid “School Details” page.
Postconditions:	1. The system stores the submitted upvote or downvote. 2. The system displays the new vote count under the related comment.
Priority:	Low
Frequency of Use:	Estimated 5-10 per day across all users
Flow of Events:	1. The user navigates to the “School Details” page. 2. The user selects a comment to upvote or downvote. 3. The system stores the vote. 4. System refreshes or updates the page 5. The system displays the new upvote and downvote count under the related comments.
Alternative Flows:	AFS3a: **User Not Logged In** 1. The system displays the message “You need to login to upvote or downvote”. AFS3b: **User Already Voted** 1. If the user has already voted the comment, the system toggles the vote.

	AFS4: **System Error** 1. If reply cannot be stored due to a system error, system displays error: “Upvote and downvote unavailable. Please try again later”.
Exceptions:	NA
Includes:	UCA-1 Login
Special Requirements:	1. The system shall complete the process within 5 seconds under normal load conditions. 2. System shall protect against automated/bot attack (e.g., reCAPTCHA or an equivalent).
Assumptions:	NA
Notes and Issues:	NA

4.9.3 Functional Requirements

REQ-1: The user shall be able to upvote a comment if logged in.

REQ-1.1: If the user has not previously voted, an upvote shall be added.

REQ-1.2: If the user has already upvoted the comment, the upvote shall be removed (toggle behavior).

REQ-1.3: If the user had already previously downvoted the comment, the downvote shall be removed and an upvote shall be added.

REQ-2: The user shall be able to downvote a comment if logged in.

REQ-2.1: If the user has not previously voted, a downvote shall be added.

REQ-2.2: If the user has already downvoted the comment, the downvote shall be removed (toggle behavior).

REQ-2.3: If the user had already previously upvoted the comment, the upvote shall be removed and a downvote shall be added.

4.10 Fetch School Data from Government API

4.10.1 Description and Priority

The system should be able to fetch data from the government API. The priority for this is **high** as school data is at the core of the application.

4.10.2 Stimulus/Response Sequences

Use Case ID:	UCSY-1		
Use Case Name:	Fetch School Data from Government API		
Created By:	Yong Chee Seng	Last Updated By:	Yong Chee Seng
Date Created:	4 September 2025	Date Last Updated:	15 November 2025

Actor:	Government API
Description:	The system shall retrieve available school data from MOE API.
Preconditions:	NA
Postconditions:	1. The system caches the retrieved school data in the system. 2. The system returns school data when requested.
Priority:	High
Frequency of Use:	Estimated 100-500 per day across all users
Flow of Events:	1. The system initiates **UCSY-1 Fetch School Data from Government API.** 2. The system calls the MOE API to retrieve all school data. 3. The Government API returns all school data. 4. The system caches the retrieved school data in the system.
Alternative Flows:	NA
Exceptions:	EX1: **MOE API Unavailable** 1. The system throws an exception to be caught by initiating a process.
Includes:	NA
Special Requirements:	1. The system shall complete the process within 5 minutes under normal load conditions. 2. The system shall initiate the process on startup before any user request. 3. The system shall initiate the process at a configurable interval between 1 and 7 days.
Assumptions:	NA
Notes and Issues:	NA

4.11 Edit User Profile

4.11.1 Description and Priority

The user should be able to edit their user profile. The priority for this is **medium** as the user profile such as postal code is needed to display school details in full.

4.11.2 Stimulus/Response Sequences

Use Case ID:	UCS-5		
Use Case Name:	Edit User Profile		
Created By:	Yong Chee Seng	Last Updated By:	
Date Created:	15 November 2025	Date Last Updated:	

Actor:	User (Initiating)
Description:	The user shall be able to edit their profile under the “User” page.
Preconditions:	1. The user has logged in to an account. 2. The user has navigated to a “User” page.
Postconditions:	1. The system stores the edited profile. 2. The system displays the new profile under the “User” page.
Priority:	Medium
Frequency of Use:	Estimated 3-5 per day across all users
Flow of Events:	1. The user navigates to the “User” page. 2. The user inputs a new profile into the corresponding field. 3. The user submits the input. 4. The system validates the input. 5. If valid, the system stores the new profile. 6. The system refreshes or updates the page. 7. The system displays the profile in the “User” page.
Alternative Flows:	AFS2: **User Not Logged In** 1. The system displays the message “You need to login”. AFS5: **Invalid Input** 1. The system displays a clear error message explaining why input is invalid (e.g., empty or too long). AFS6: **System Error** 1. If a comment cannot be stored due to a system error, the system displays error: “Create comment unavailable. Please try again later”.
Exceptions:	1. NA

Includes:	1. UCA-1 Login
Special Requirements:	1. The system shall complete the edit profile process within 5 seconds under normal load conditions. 2. System shall protect against automated/bot attack (e.g., reCAPTCHA or an equivalent). 3. The system shall sanitize and validate user input to prevent malicious attacks (e.g., XSS, SQL injection).
Assumptions:	NA
Notes and Issues:	NA

4.11.3 Functional Requirements

REQ-1. The user shall be able to edit their user profile.

REQ-1.1. The system shall enforce authentication to edit their user profile.

REQ-1.2. The user shall be able edit their postal code.

REQ-1.2.1. The postal code shall be a string with 6 digits.

REQ-1.2.2. The postal code shall exist in Singapore.

4.12 Edit User Role

4.12.1 Description and Priority

The admin should be able to edit a user's role. The priority for this is **low** as the admin feature is not crucial to the application.

4.12.2 Stimulus/Response Sequences

Use Case ID:	UCS-6		
Use Case Name:	Edit User Role		
Created By:	Yong Chee Seng	Last Updated By:	
Date Created:	15 November 2025	Date Last Updated:	

Actor:	Admin (Initiating)
Description:	The admin shall be able to edit the user role.
Preconditions:	1. The admin has logged in to an admin account.
Postconditions:	1. The system stores the edited role.
Priority:	Low
Frequency of Use:	Estimated 1 per month across all users
Flow of Events:	1. The admin inputs user ID and role 2. The admin submits the input. 3. The system stores the new role.
Alternative Flows:	AFS1: **User Not Admin** 1. The system displays the error message “Insufficient permission”. AFS3a: **User Not Found** 1. The system displays the error message “User not found". AFS3b: **Demoting Last Admin** 1. If the user is the last admin and is demoted to user, the system displays the error message “User is last admin”.
Exceptions:	1. NA
Includes:	1. UCA-1 Login
Special Requirements:	1. The system shall complete the edit role process within 5 seconds under normal load conditions.
Assumptions:	NA
Notes and Issues:	NA

4.12.3 Functional Requirements

REQ-1. The admin shall be able to edit all user's roles.

REQ-1.1. The system shall enforce authentication to edit user roles.

REQ-1.2. The admin shall be able to edit the user role to admin.

REQ-1.3. The admin shall be able to edit the user role to user.

REQ-1.3.1. The user shall not be the last admin.

4.13 Edit School Cut Off Point

4.13.1 Description and Priority

The admin should be able to edit a school's cut-off point. The priority for this is **low** as the admin feature is not crucial to the application.

4.13.2 Stimulus/Response Sequences

Use Case ID:	UCS-7		
Use Case Name:	Edit School Cut Off Point		
Created By:	Yong Chee Seng	Last Updated By:	
Date Created:	15 November 2025	Date Last Updated:	

Actor:	Admin (Initiating)
Description:	The admin shall be able to edit a school cut-off point.
Preconditions:	1. The admin has logged in to an admin account.
Postconditions:	1. The system stores the edited cut-off point.
Priority:	Low
Frequency of Use:	Estimated 350 per year across all users
Flow of Events:	1. The admin inputs school ID, minimum cut-off point, and maximum cut-off point 2. The admin submits the input. 3. The system stores the edited cut-off point.
Alternative Flows:	AFS1: **User Not Admin** 1. The system displays the error message “Insufficient permission”. AFS3a: **School Not Found** 1. The system displays the error message “School not found". AFS3b: **Invalid Cut Off Point Range** 1. The system displays the error message “Cut-off point must be in the range of 4-32”. AFS3c: **Maximum Cut Off Point Smaller Than Minimum Cut Off Point** 1. The system displays the error message “Maximum cut-off point larger or equal to minimum cut-off point”.
Exceptions:	1. NA
Includes:	1. UCA-1 Login

Special Requirements:	1. The system shall complete the edit cut-off point process within 5 seconds under normal load conditions.
Assumptions:	NA
Notes and Issues:	NA

4.13.3 Functional Requirements

REQ-1. The admin shall be able to edit all schools' cut off points.

REQ-1.1. The system shall enforce authentication to edit school cut-off points.

REQ-1.2. The admin shall be able to edit the school's minimum cut-off point.

REQ-1.2.1. The minimum cut-off point input shall be an integer within 4-32.

REQ-1.3. The admin shall be able to edit the school's maximum cut-off point.

REQ-1.3.1. The maximum cut-off point input shall be an integer within 4-32.

REQ-1.3.2. The maximum cut-off point input shall be larger than or equal to the minimum cut-off point.

5. Non-Functional Requirements

5.1 Performance Requirements

1. The system shall respond to school query requests within 2 seconds for 95% of queries under standard load conditions.
    1.1. The system shall use optimization techniques (e.g., caching) for school query requests.
2. The system shall respond to database queries within 5 seconds for 95% of operations under standard load conditions.
3. The system shall support at least 100 concurrent users with no more than 10% increase in average response time compared to single-user load.

5.2 Security, Authentication, Persistence Requirements

5.2.1 Security

1. The system shall use HTTPS with TLS 1.2 or higher to encrypt all communications between client and server.
2. The system shall sanitize and validate all user inputs to prevent security vulnerabilities such as XSS, SQL injection, CSRF, and command injection.
3. The system shall implement rate limiting and/or CAPTCHA to mitigate automated bot attacks.
4. The system shall implement throttling, monitoring, and alerts to mitigate and detect DoS/DDoS attacks.

5.2.2 Data Persistence

1. The system shall persist only necessary information via client-side storage (e.g., cookies, localStorage).
    1.1. The system shall invalidate client-side cookies no later than 7 days after creation.
2. The system shall persist user-specific data for authenticated users in the server-side database.
    2.1. System shall protect all persisted information against unauthorized access and tampering

5.2.3 Authentication

1. The system shall protect session information against unauthorized access and tampering using secure mechanisms (e.g., HTTP-only Secure cookies, signed tokens).
2. The system shall implement protections against brute-force login attempts (e.g., rate limiting, captcha, temporary lockout).
3. The system shall hash and salt all passwords using an industry-standard algorithm (e.g., bcrypt, Argon2) before storage.

5.3 Software Quality Attributes

5.3.1 Usability

1. The system shall be intuitive to users with minimal technical knowledge:
    1.1. 80% of users shall be able to make a simple school search within 2 minutes of using the website
        1.1.1. A simple school search will encompass a simple name search (e.g. XXX secondary school) or a single-field filter (e.g. filter by zone code)
    1.2. System UI shall adhere to Nielsen Norman group's usability heuristics, with minimalistic design, clear labels and consistent layout
2. The system shall provide accessibility support in compliance with WCAG 2.1 AA guidelines:
    2.1. System shall focus on screen reader compatibility and keyboard navigation support
3. The system shall provide helpful validation messages and error hints on erroneous user inputs
4. The system shall provide feedback (e.g., loading indicators or transition pages) for operations exceeding 1 second, to inform users that the system is processing the request.

5.3.2 Reliability

1. System shall be available 99.5% of the time, excluding periods of planned maintenance
2. System shall display a fallback maintenance page in the event of downtime
3. System shall schedule planned maintenance during periods of low web traffic, between 1:00 AM and 6:00 AM SGT

5.3.3 Maintenability

1. System shall support unit testing and integration testing
    1.1. Unit test coverage should be >80%
    1.2. System shall undergo user acceptance testing (UAT) before release with >10 representative users
        1.2.1. UAT shall pass with >90% task completion and >80% user satisfaction rating

5.3.4 Logging

1. The system shall maintain logs for user actions and application errors.
2. The system shall ensure that sensitive information (e.g., passwords, session tokens) is never logged in plain text.
3. The system shall restrict access to logs to authorized personnel only.

6. Other Requirements

6.1 Data Requirements

6.1.1 Data Storage

1. The system shall abstract data access so that the underlying database technology can be replaced with any database system, provided it supports the required data operations (e.g., CRUD, queries, indexing).

6.1.2 Data Collection

1. The system shall fetch school data from the Singapore Open Government Dataset via the data.gov.sg API on startup.
2. The system shall refresh school data asynchronously from the Singapore Open Government Dataset via the data.gov.sg API at least once per week
3. The system shall update the cut-off point data within 7 days of the data becoming available.
4. System shall cross-compare and validate newly fetched data to ensure consistency and accuracy before updating the database

6.1.3 Data Protection and Privacy

1. The system and its maintaining organization shall comply with Singapore PDPA regulations for the storage and handling of personal data.
2. The system shall only collect the minimum necessary data required for functionality.
3. The system shall implement mechanisms to enforce data retention limits in accordance with PDPA.
4. System shall provide a clear and accessible privacy policy describing:
    4.1. What data is collected
    4.2. Why it is collected
    4.3. How long it will be retained
    4.4. How it will be used
5. The system shall only collect personal data with the user's explicit consent.
    5.1. The system shall require the user to agree to the privacy policy when creating an account.
6. The system shall allow users to request removal of their personal data by contacting the maintaining party.

6.2 Legal Requirements

1. System shall provide a clear and accessible Terms and Conditions describing:
    1.1. Actions that are considered violations.
    1.2. Consequences of such violations (e.g., warnings, account suspension, or deletion).
2. System shall require users to explicitly agree to the Terms and Conditions when creating an account.
3. System shall allow authorized administrators to delete, restrict, or suspend user accounts that violate the Terms and Conditions.

7. Appendix A: Data Dictionary

A.1 School

Term	**Description**
School	An educational institution in Singapore. The system's list of schools and their information shall be sourced from the Singapore Open Government Dataset: https://data.gov.sg/
Search Criteria	The fields used to filter schools in the system. Includes: ● Name ● Location ● CCAs ● Subjects ● Cut-off Point Range ● Level ● Nature Code ● Type ● Session Code
Name	The official name of the school; can be used as a school identifier.
Location	The DGP (Development Guide Plan) code of the school, such as Woodlands or Jurong West. Represents the planning area as defined by the Urban Redevelopment Authority.
CCA	Co-Curricular Activities offered by the school.
Subjects	Academic subjects offered by the school.
Programmes	MOE programmes offered by the school.
Cut-off Point	A student's PSLE score used to determine the posting group and indicative level for subjects. Range: 4–32 (4 = best, 32 = worst).
Level	Education level of the school (e.g., Primary, Secondary, Junior College).
Nature Code	Type of school nature (e.g., Co-ed, Boys, and Girls).
Type	Type of school (e.g., Government, Government-aided, etc.).

Session Code	The academic session type (e.g., Single session and Full day).
Address	Full street address of the school.
Postal Code	Postal code for the school address.
Nearby MRT Station	MRT stations that are close to the school.
Nearby Bus Station	Bus stops that are close to the school.
Website	URL of the school's official website.
Email	Official school email address.
Phone Number	Official contact phone number for the school.
Fax Number	Official fax number for the school.
Mother Tongue	Mother tongue language offered by the school (e.g., Co-ed, Boys, and Girls).
SAP	Indicate whether a school has **Special Assistance Plan** status. These are schools that offer bicultural education.
Autonomous	Indicate whether a school has **autonomous** status. These are schools that follow the national syllabus, but offer a wider range of programmes.
Gifted	Indicate whether a school has a **Gifted Education Programme (GEP)** in secondary level.
IP	Indicate whether a school offers an **Integrated Programme.** These are schools with a 6-year programme with no O-Level exams, going straight to A-Levels/IB.

A.2 User

Term	**Description**
User ID	Unique identifier for each user.
Username	Unique username chosen by the user.
Password	User's password.
Created At	Timestamp the user signed up.
Role	User's role. Either **User** or **Admin**.
Postal Code	User's postal code.

A.3 Comment

Term	**Description**
Comment ID	Unique identifier for each comment.
User ID	Identifier of the user who created the comment.
School ID	Identifier of the school associated with the comment.
Content	Content of the comment.
Upvote Count	Number of upvotes for the comment. Updated whenever a user upvotes or removes an upvote.
Downvote Count	Number of downvotes for the comment. Updated whenever a user downvotes or removes a downvote.
Created At	Timestamp the comment was created.

A.4 Reply

Term	**Description**
Reply ID	Unique identifier for each reply.
User ID	Identifier of the user who created the reply.
Comment ID	Identifier of the parent comment.
Content	Content of the reply.
Created At	Timestamp the reply was created.

A.5 Vote

Term	**Description**
Upvote	Users like or agree with the comment.
Downvote	Users dislike or disagree with the comment.
Vote ID	Unique identifier for each vote.
Comment ID	Identifier of the comment being voted.
User ID	Identifier of the user who created the vote.
Type	Upvote or downvote.

A.6 Favourite School

Term	**Description**
User ID	Identifier of the user who saved the school.
School ID	Identifier of the saved school.

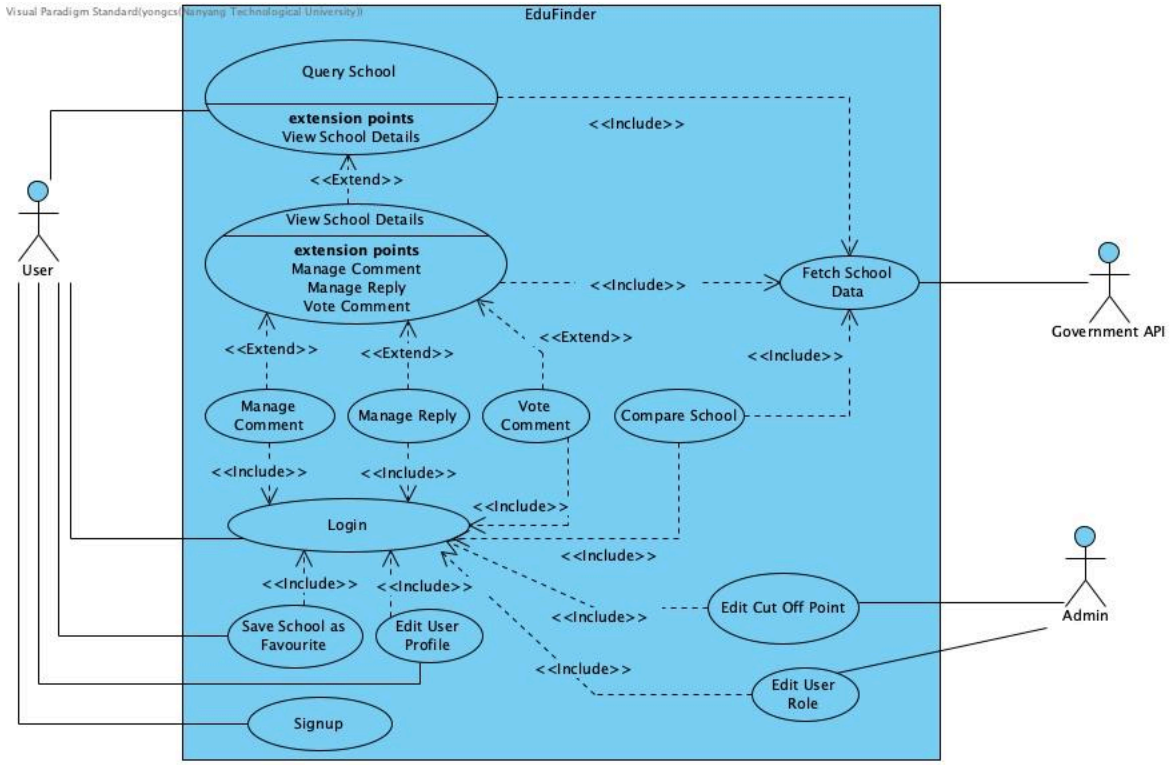
A.7 Actor

Term	**Description**
Government API	The API exposed by Singapore Open Government Dataset: https://data.gov.sg/.
User	Anonymous or authenticated user using Edufinder.
Admin	Administrator of Edufinder. Able to perform admin features such as edit school cut off point.

8. Appendix B: Analysis Models

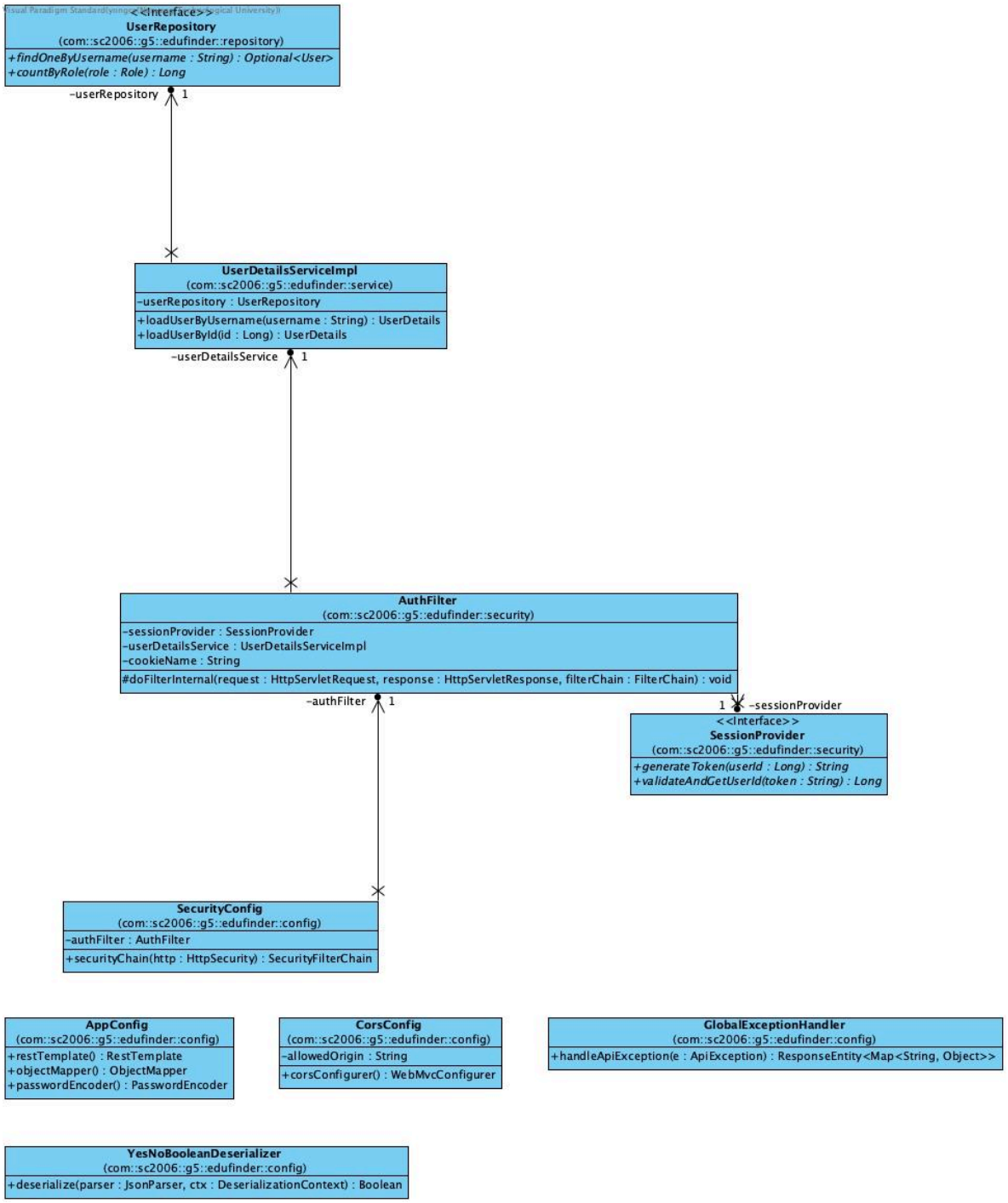
This section presents all the analysis models. Due to the aspect ratio, they may not be easily readable in this report. The same diagrams are available in the **/docs/analysis-model/** directory in both **.jpg** and **.xml** formats if needed.

B.1 Use Case Diagram

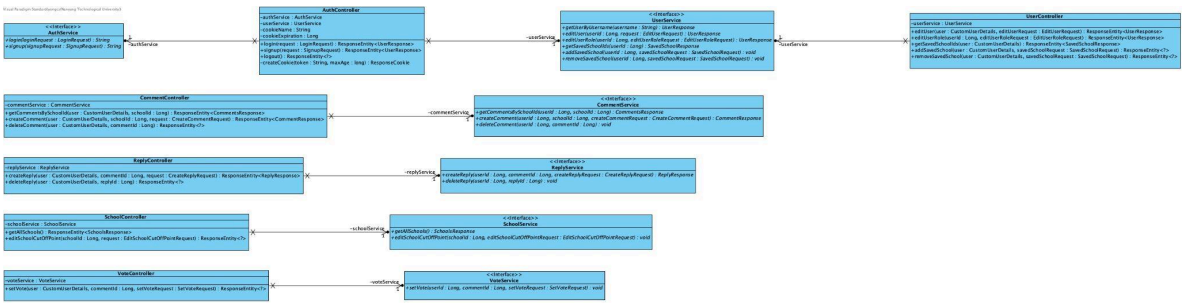


B.2 Class Diagrams

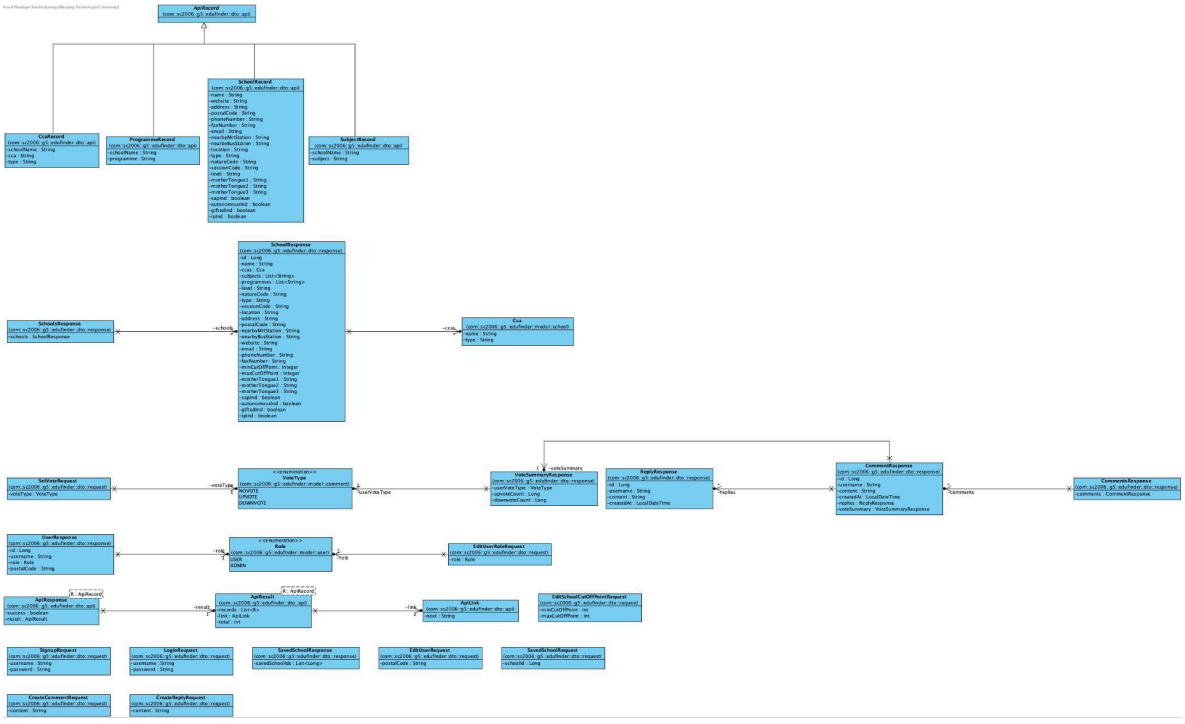
B.2.1 Config



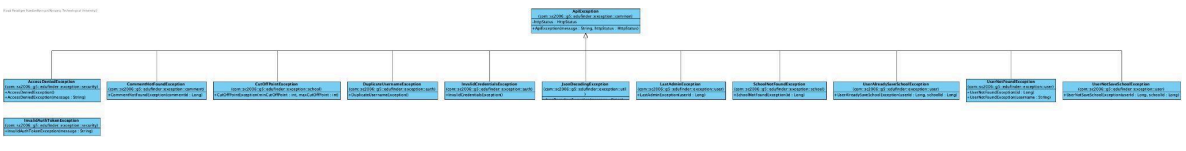
B.2.2 Controller



B.2.3 DTO

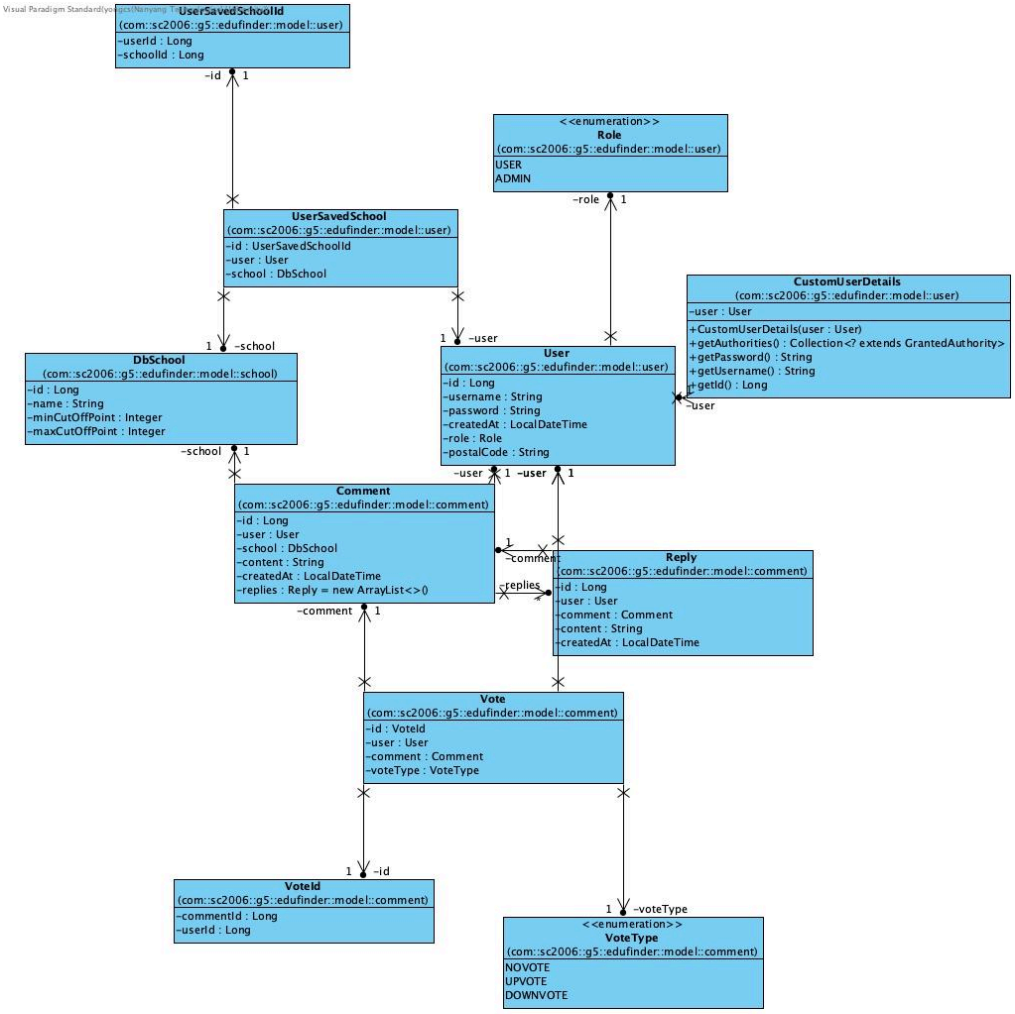


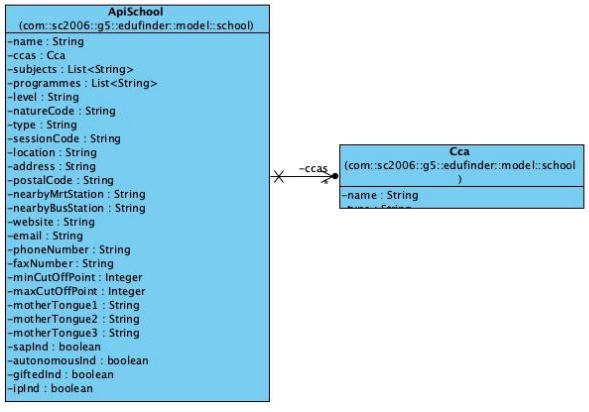
B.2.4 Exception



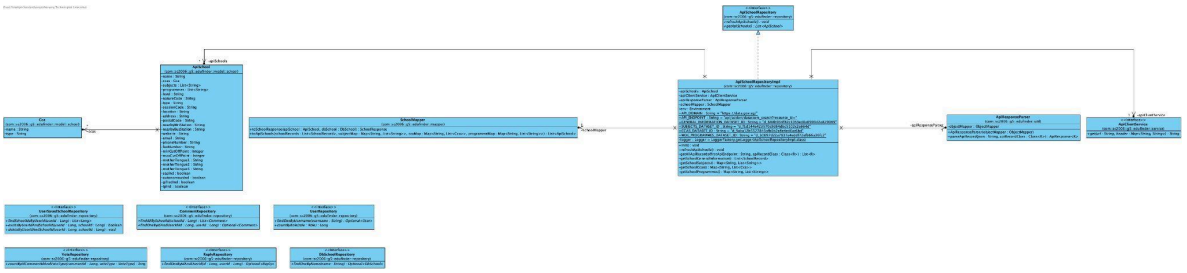
B.2.5 Mapper

B.2.6 Model

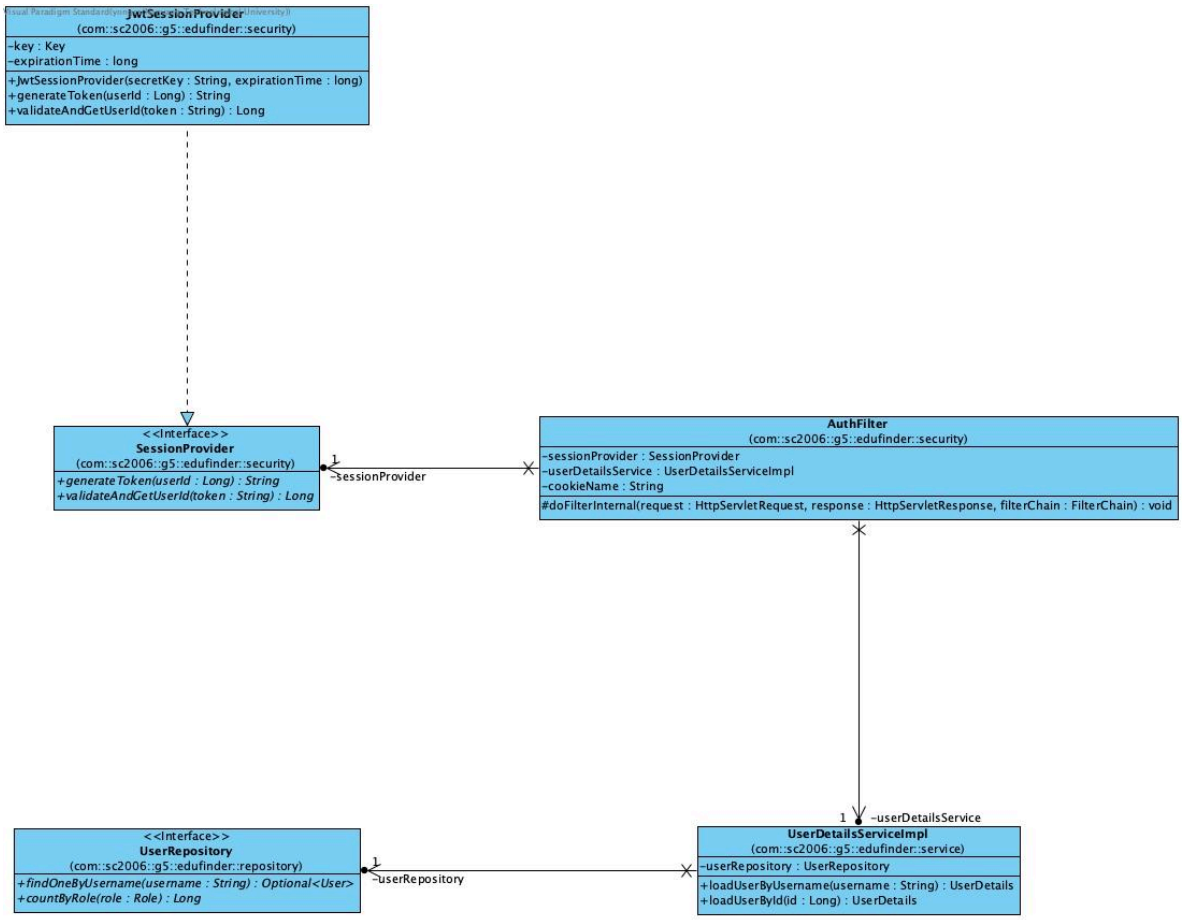




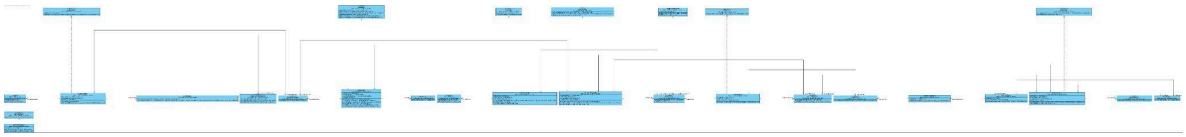
B.2.7 Repository



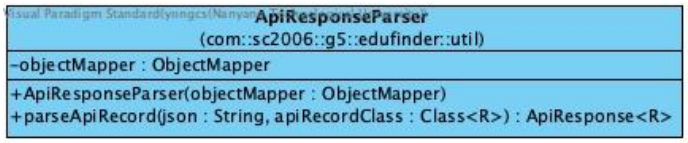
B.2.8 Security



B.2.9 Service



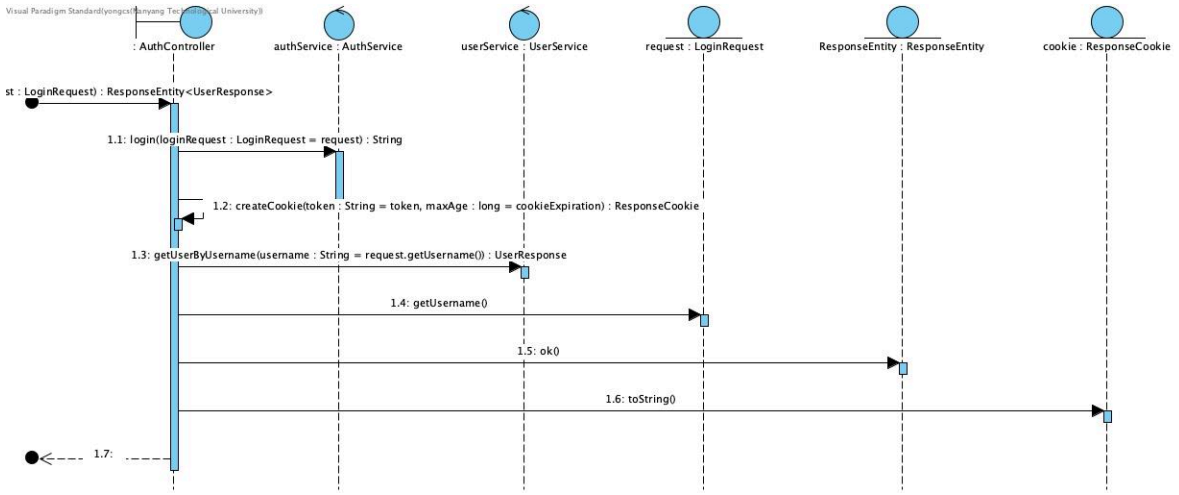
B.2.10 Util



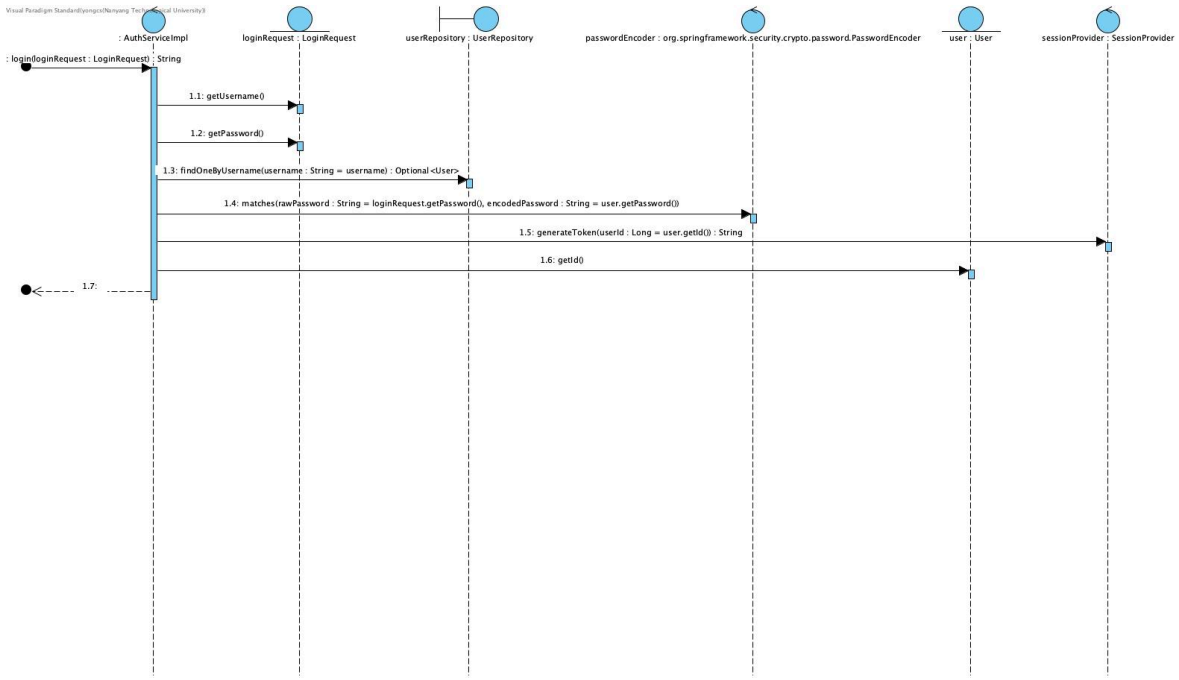
B.3 Sequence Diagrams

B.3.1 Login

B.3.1.1 AuthController.login()

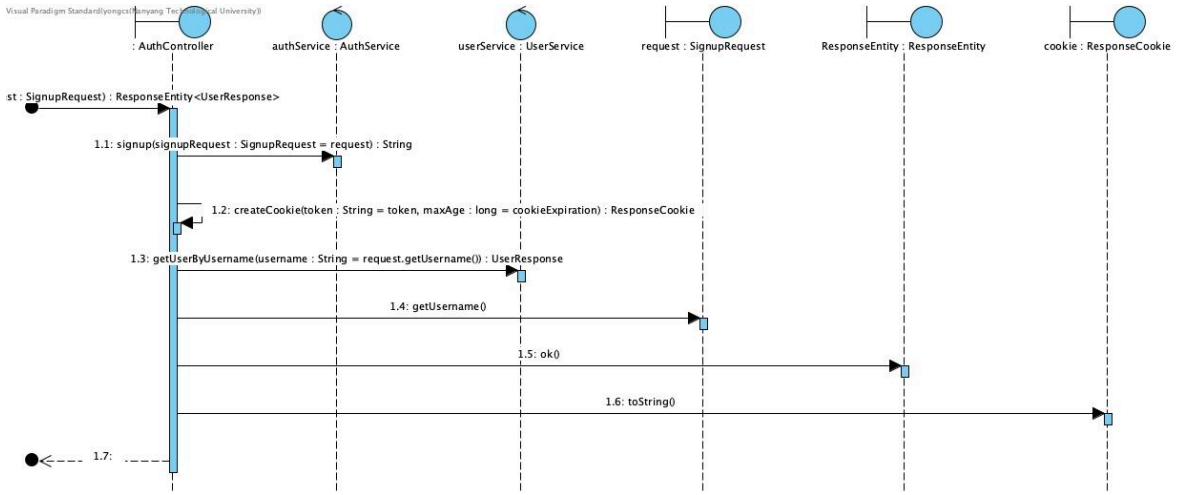


B.3.1.2 AuthServiceImpl.login()

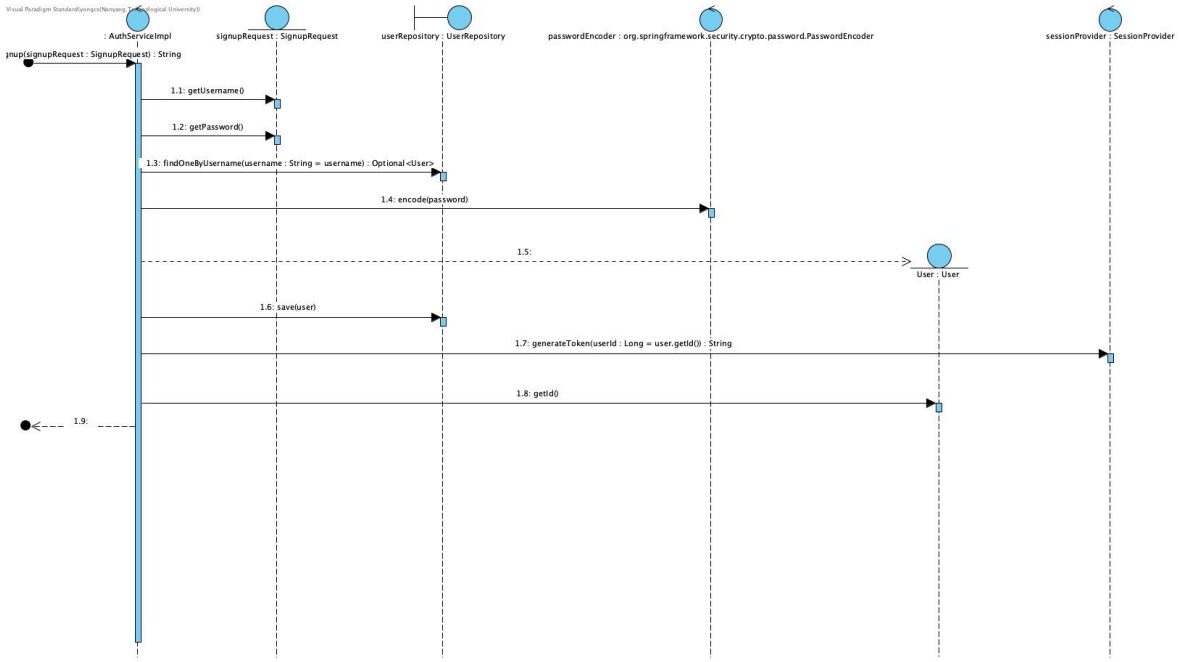


B.3.2 Signup

B.3.2.1 AuthController.signup()

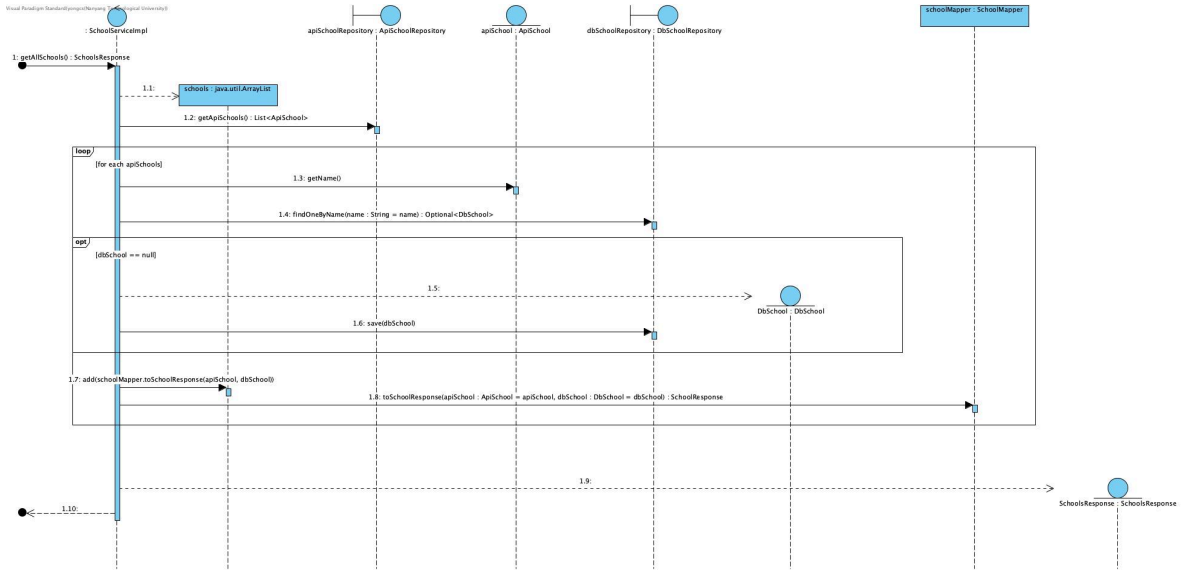


B.3.2.1 AuthServiceImpl.signup()

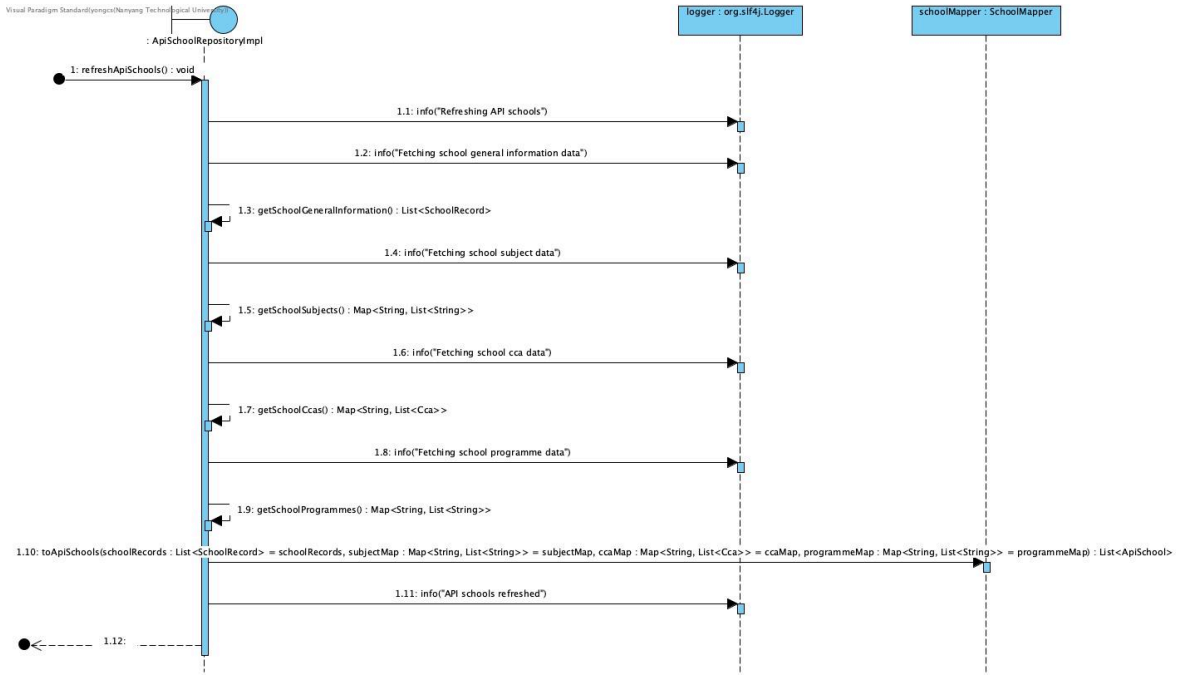


B.3.3 GetAllSchools

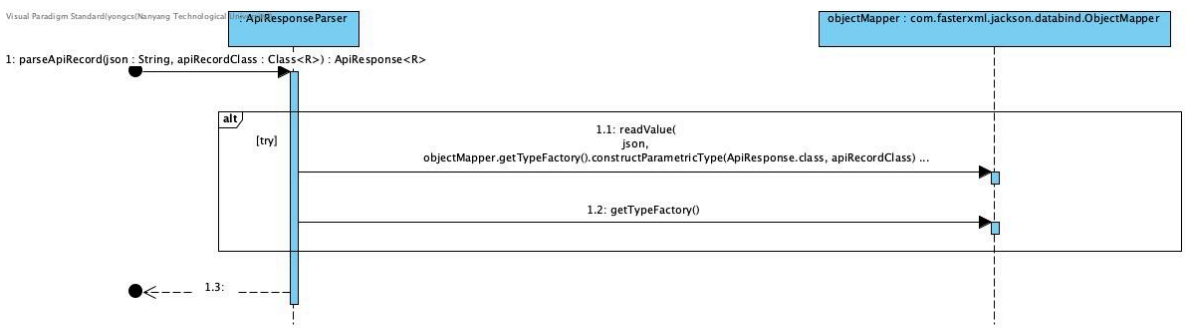
B.3.3.1 SchoolServiceImpl.getAllSchools()



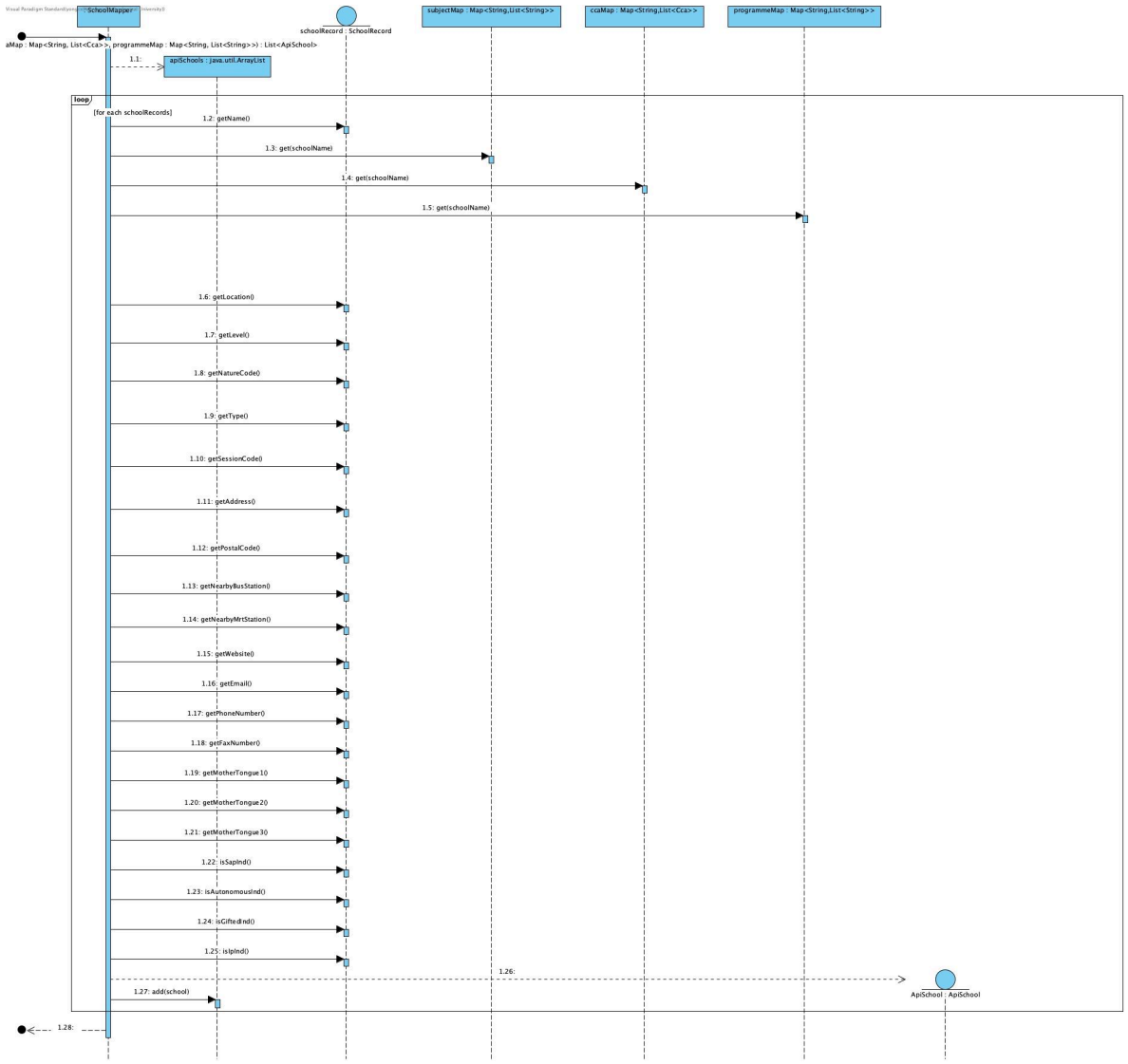
B.3.3.2 ApiSchoolRepositoryImpl.getAllSchools()



B.3.3.3 ApiResponseParser.parse()

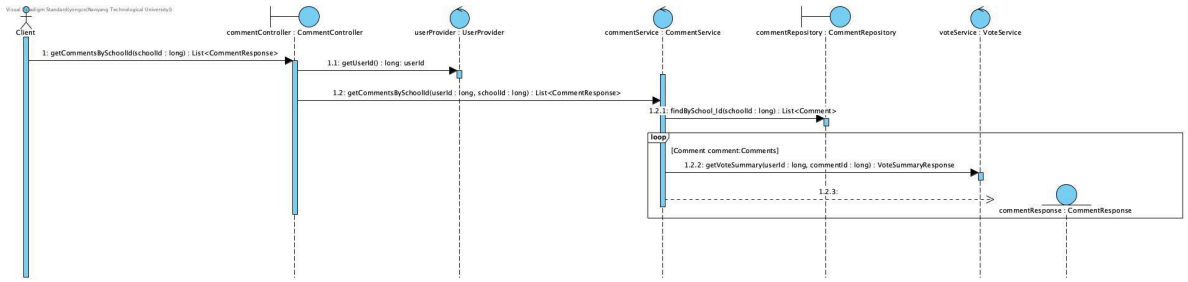


B.3.3.4 SchoolMapper.toApiSchools()

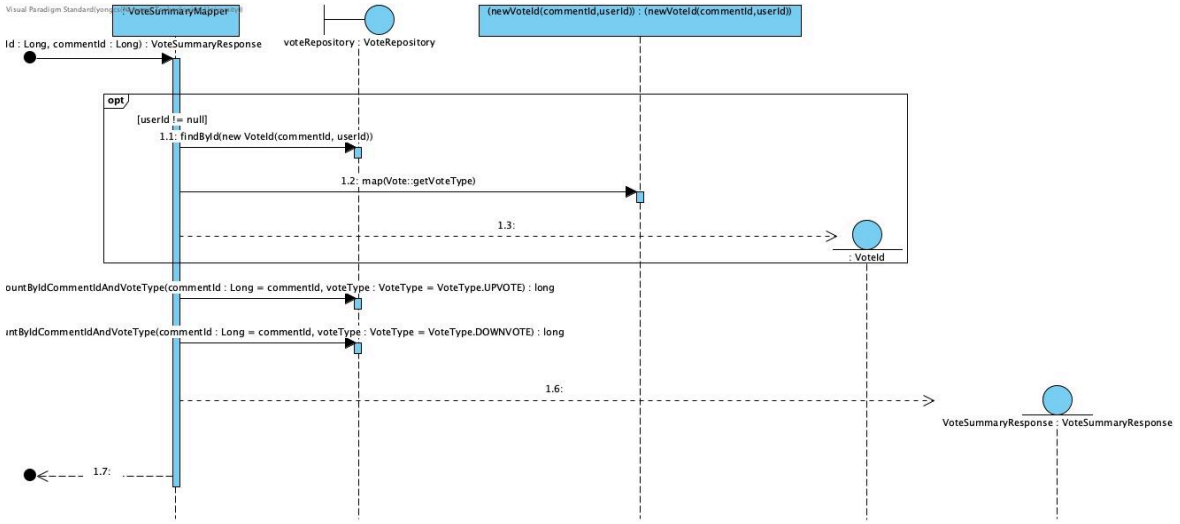


B.3.4 GetCommentsBySchoolId

B.3.4.1 CommentServiceImpl.getCommetsById()

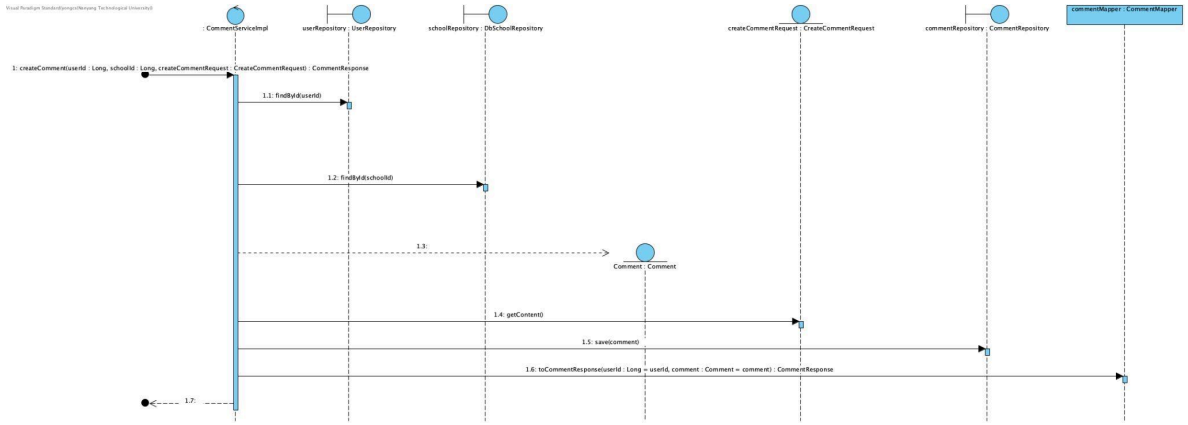


B.3.4.2 VoteSummaryMapper.buildVoteSummary()

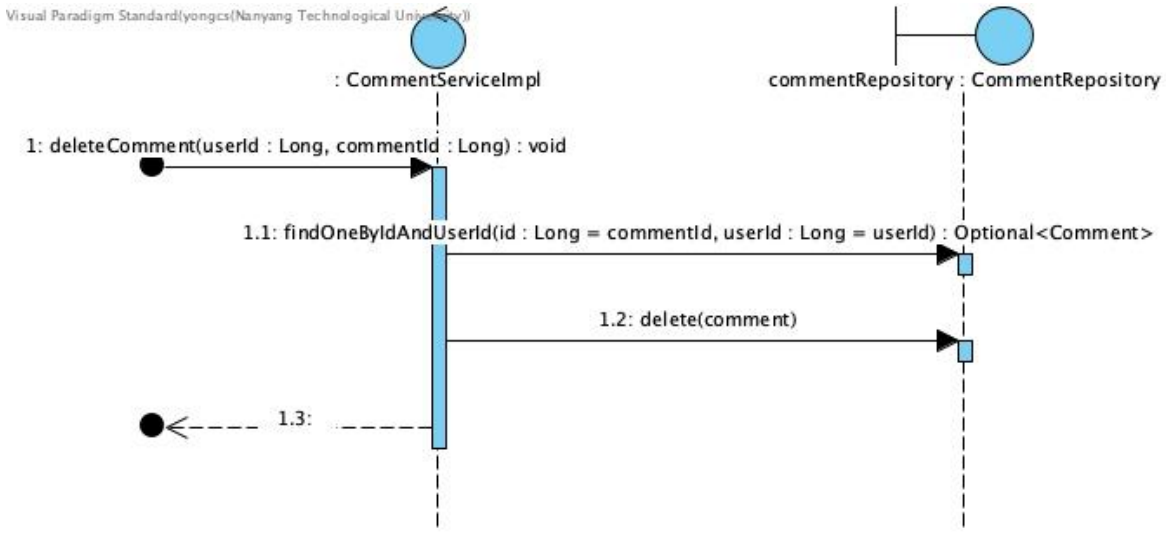


B.3.5 ManageComment

B.3.5.1 CommentServiceImpl.createComment()

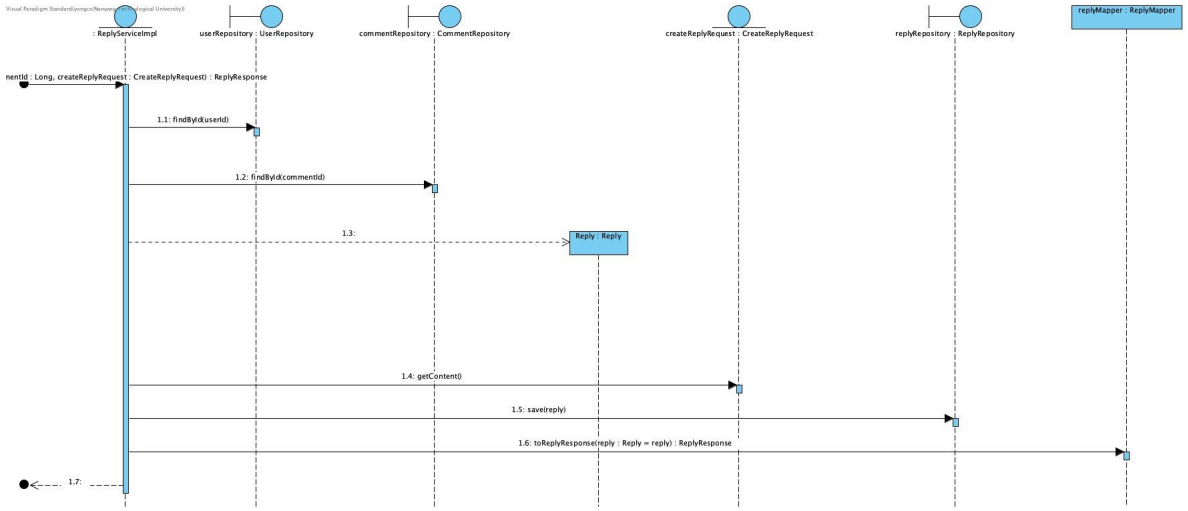


B.3.5.2 CommentServiceImpl.deleteComment()

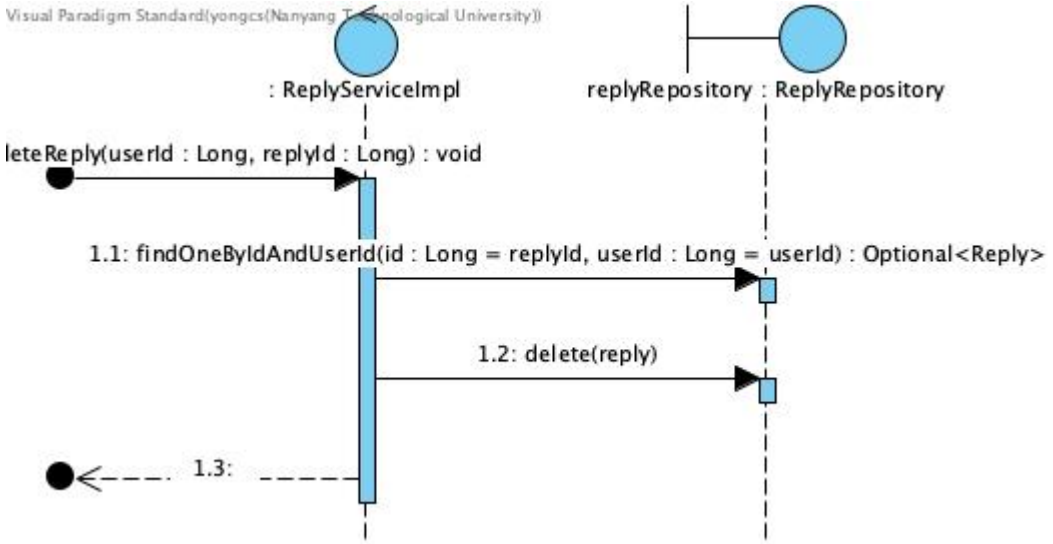


B.3.6 ManageReply

B.3.6.1 ReplyServiceImpl.createReply()

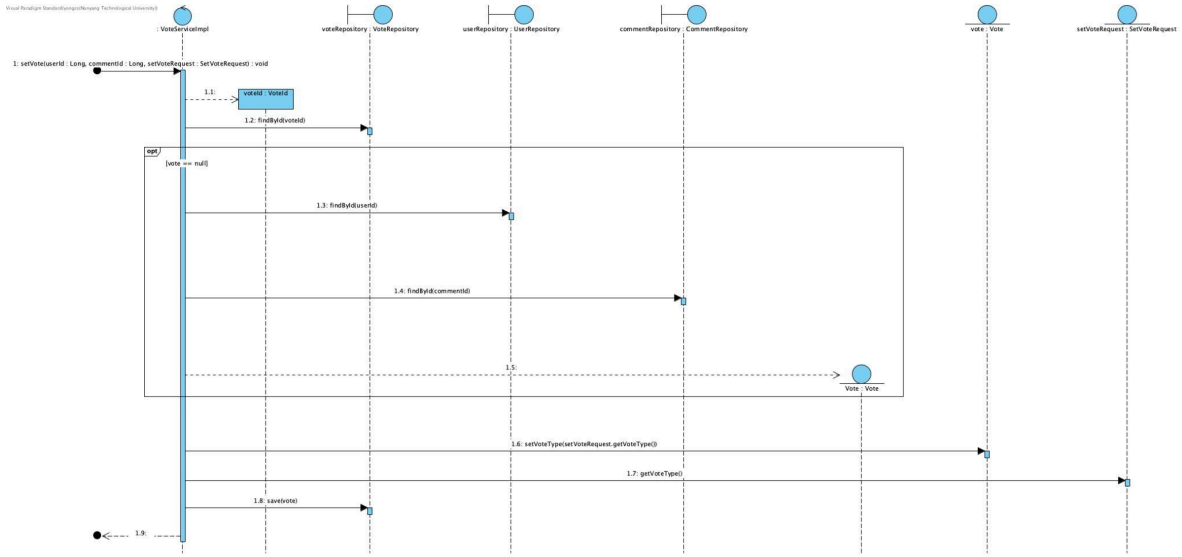


B.3.6.2 ReplyServiceImpl.deleteReply()



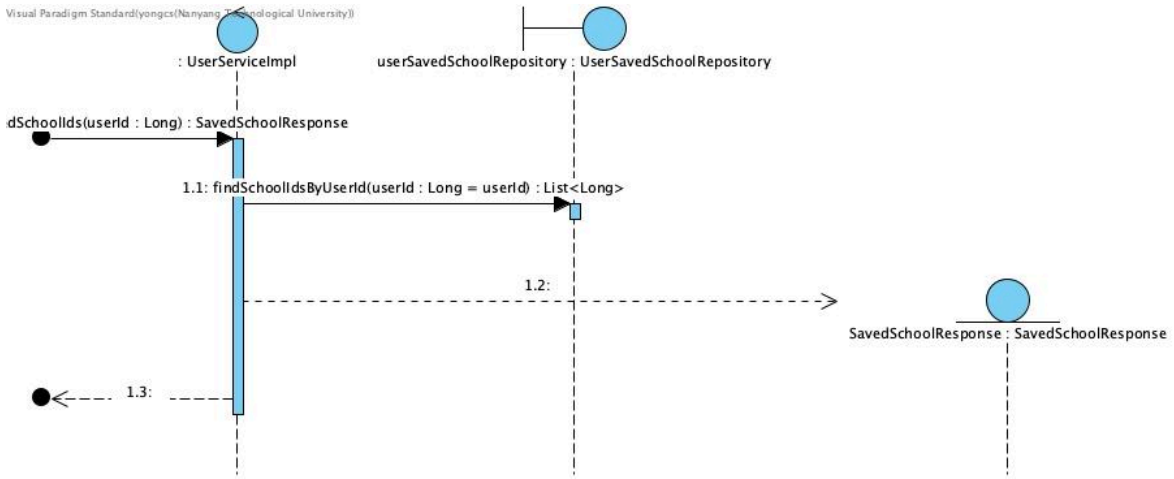
B.3.7 SetVote

B.3.7.1 VoteServiceImpl.setVote()



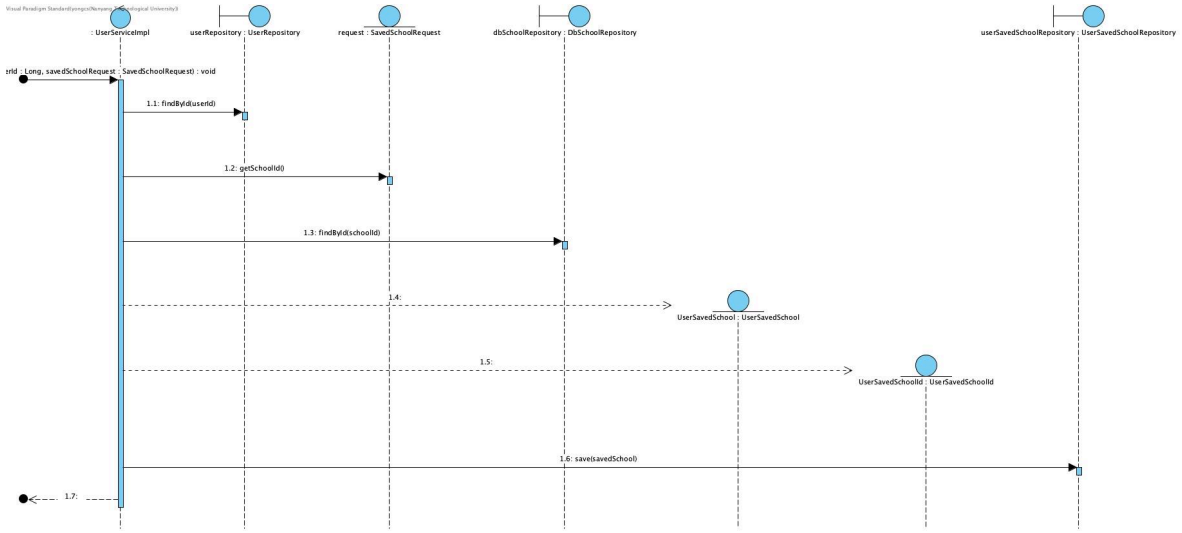
B.3.8 GetSavedSchool

B.3.8.1 UserServiceImpl.getSavedSchool()

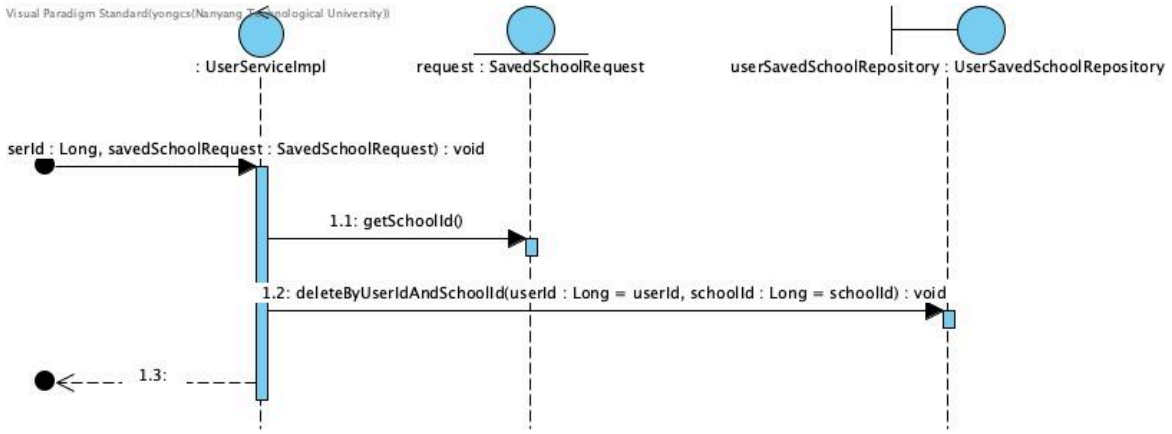


B.3.9 ManageSavedSchool

B.3.9.1 UserServiceImpl.addSavedSchool()

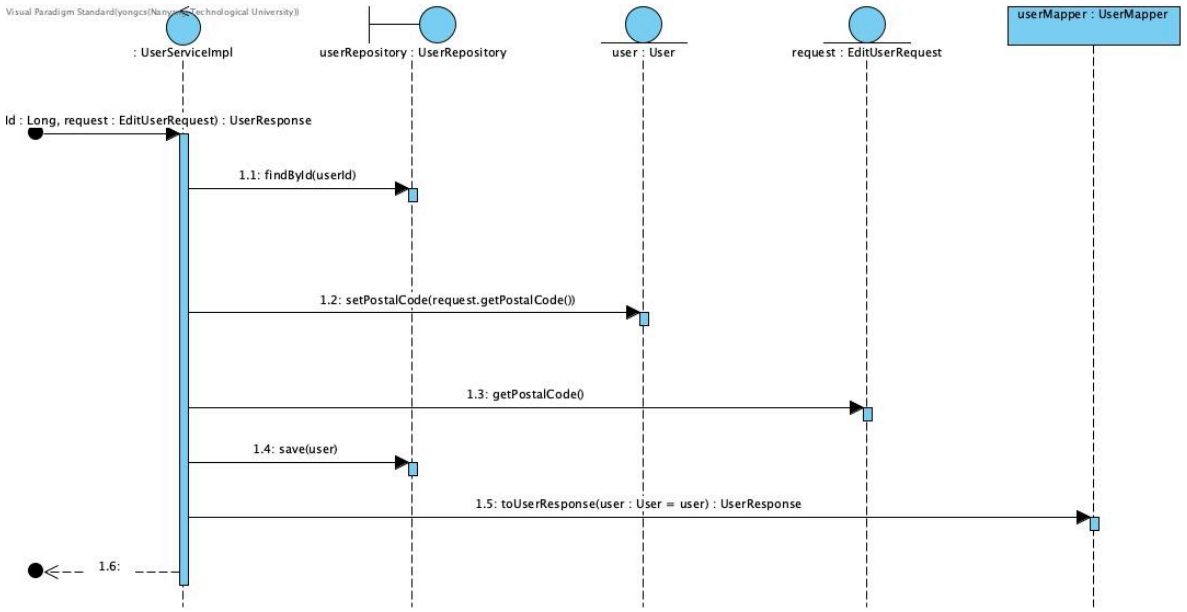


B.3.9.2 UserServiceImpl.removeSavedSchool()

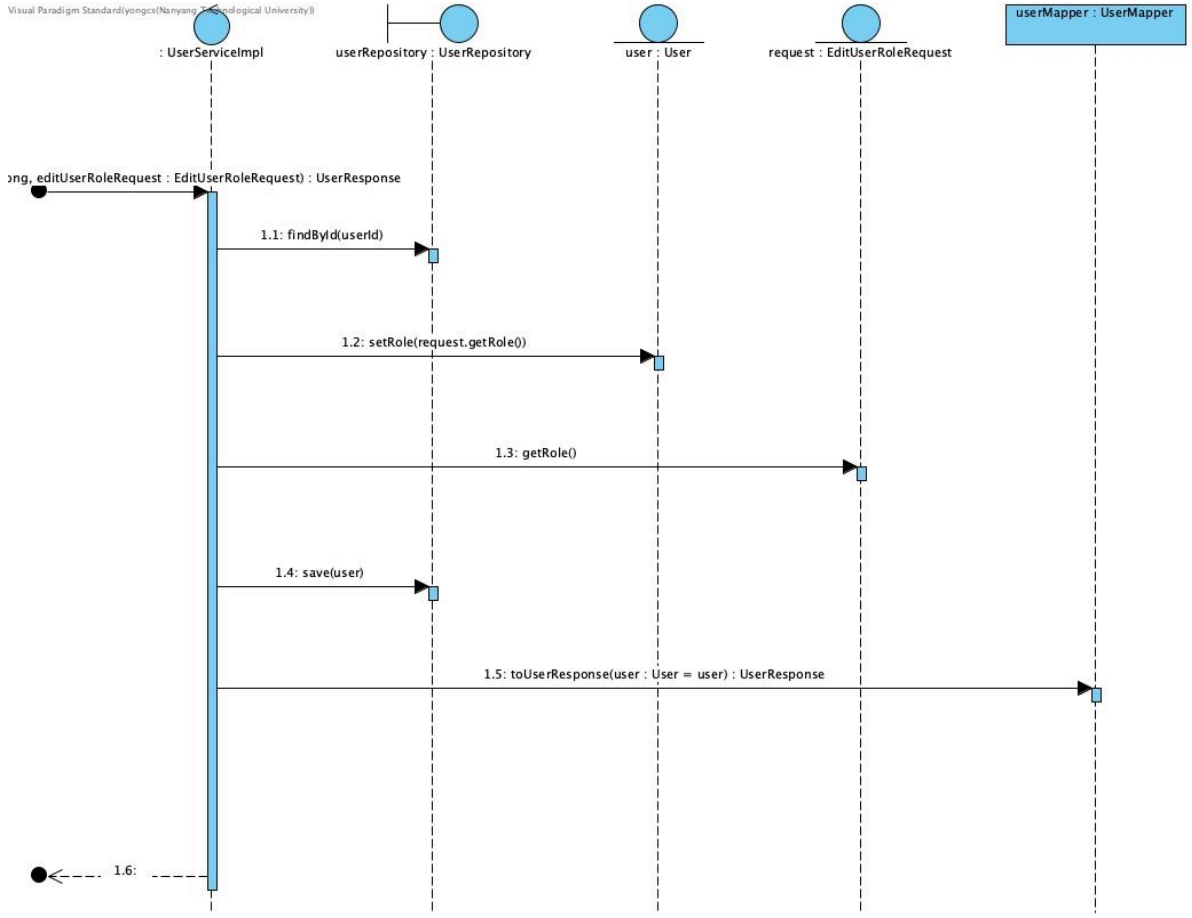


B.3.10 ManageUser

B.3.10.1 UserServiceImpl.editUser()

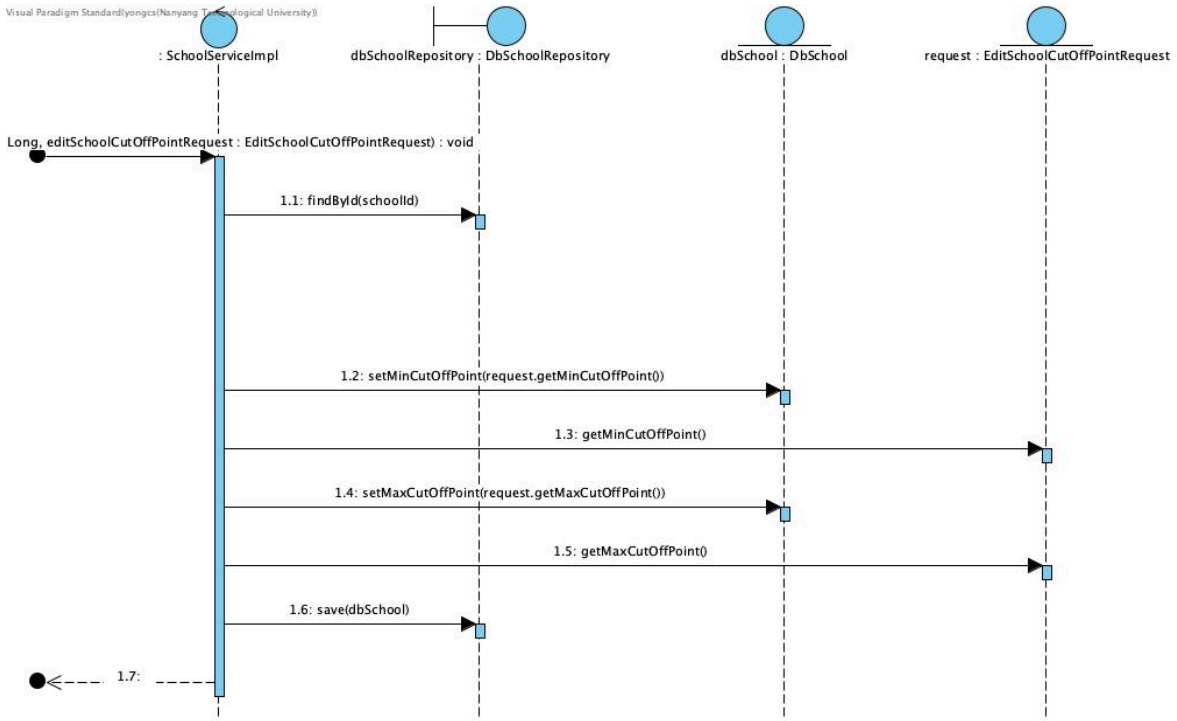


B.3.10.2 UserServiceImpl.editUserRole()



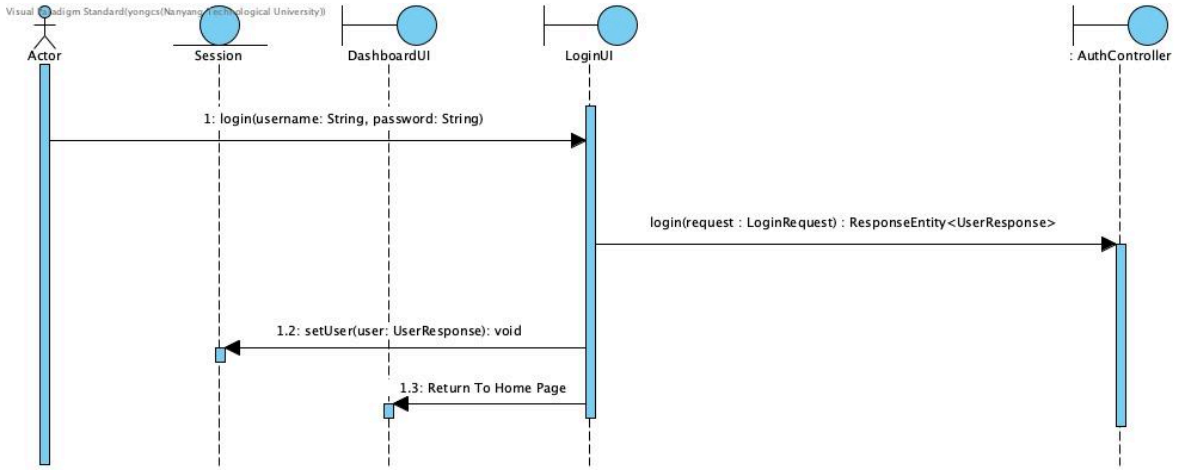
B.3.11 ManageSchool

B.3.11.1 SchoolServiceImpl.editSchoolCutOffPoint()

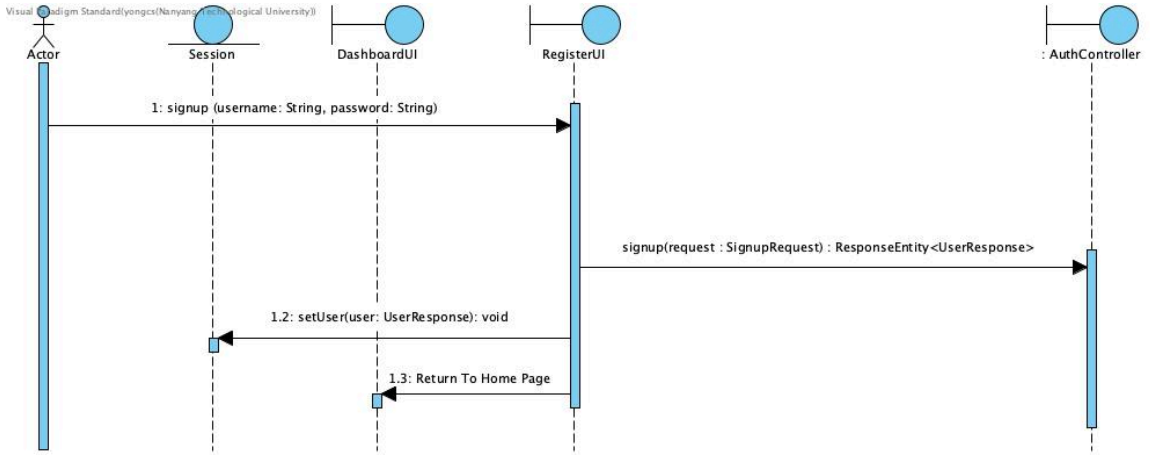


B.4 Sequence Diagrams (Frontend)

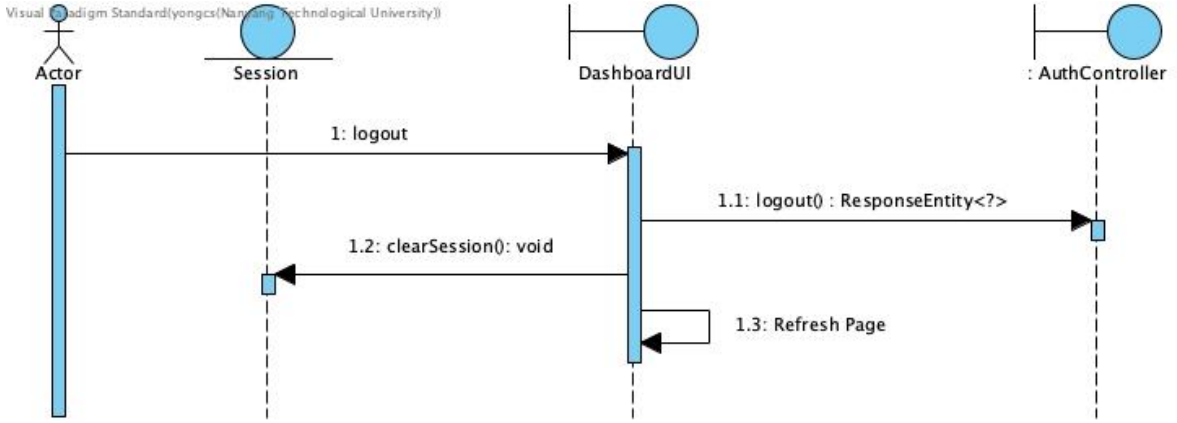
B.4.1 Login



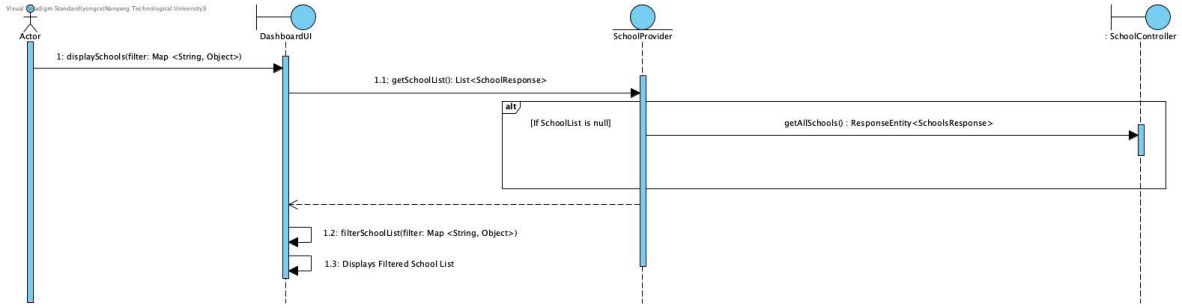
B.4.2 Signup



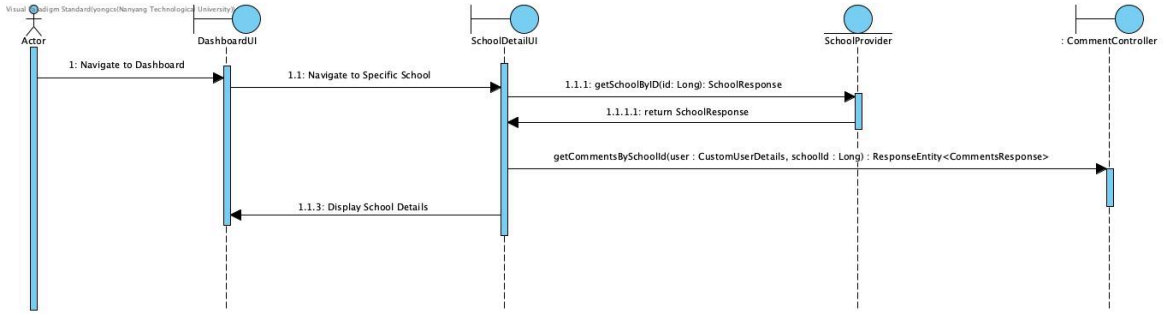
B.4.3 Logout



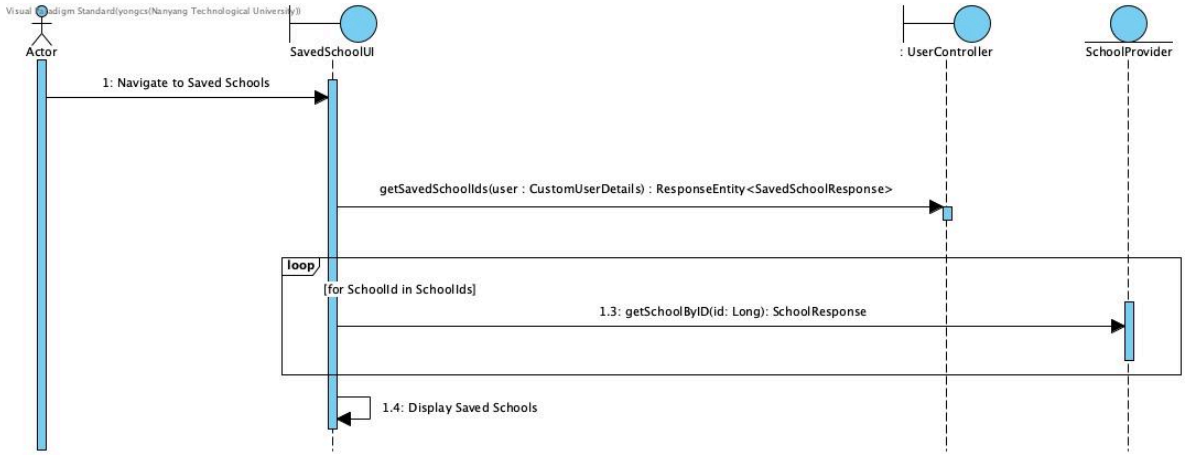
B.4.4 DisplaySchools



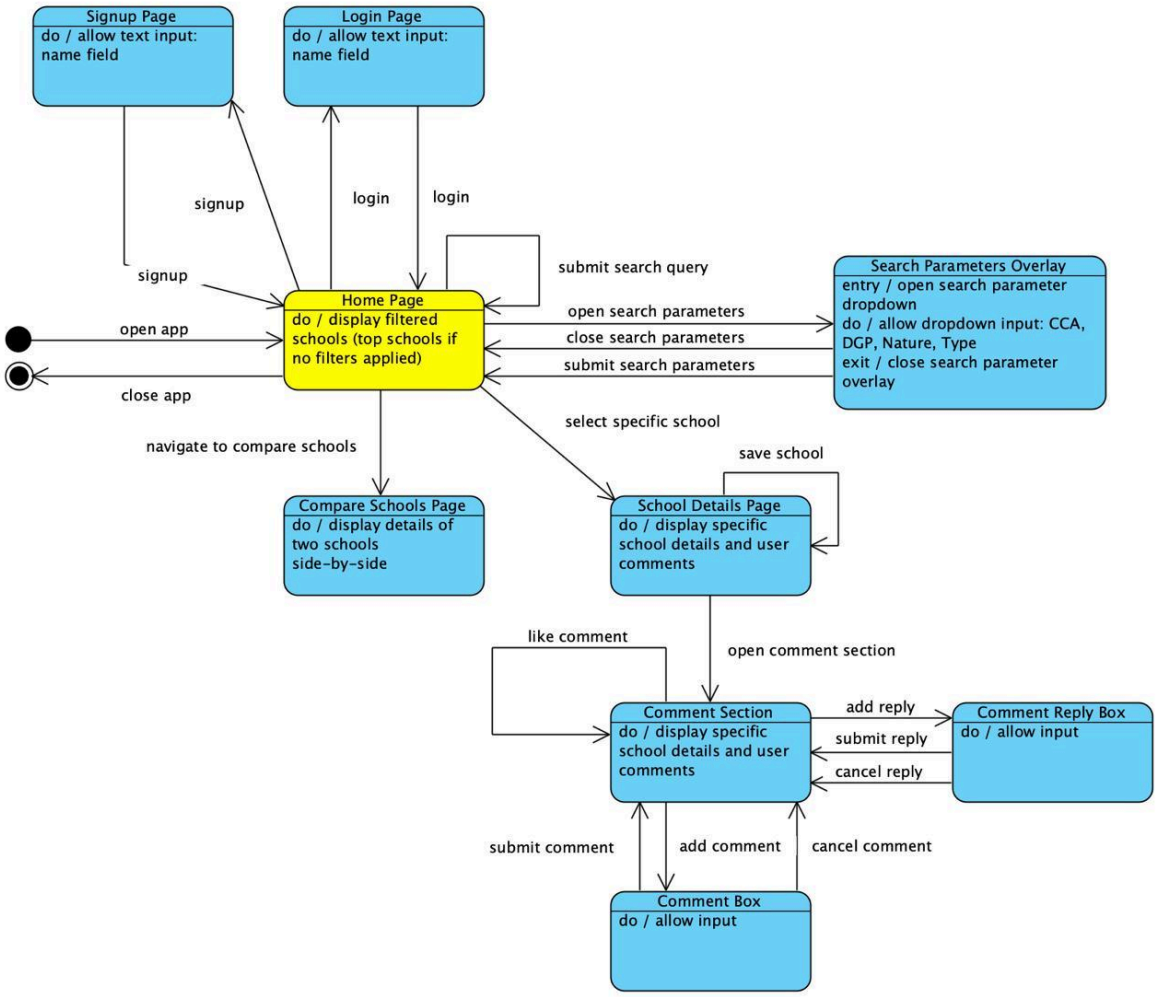
B.4.5 DisplaySchoolDetails



B.4.6 ViewSavedSchools



B.5 Dialog Map



B.6 Backend Testing

The backend includes automated tests to verify that all components function correctly before deployment.

Testing Framework: JUnit
Directory: /backend/edufinder/src/test/java/com/sc2006/g5/edufinder/

Unit Testing:

- **Approach**: White-Box testing
- **Purpose**: To test each component in isolation by mocking its dependencies.
- **Scope**: Covers core services, controllers, and other critical classes to ensure each behaves as expected under various conditions.

Integration Testing:

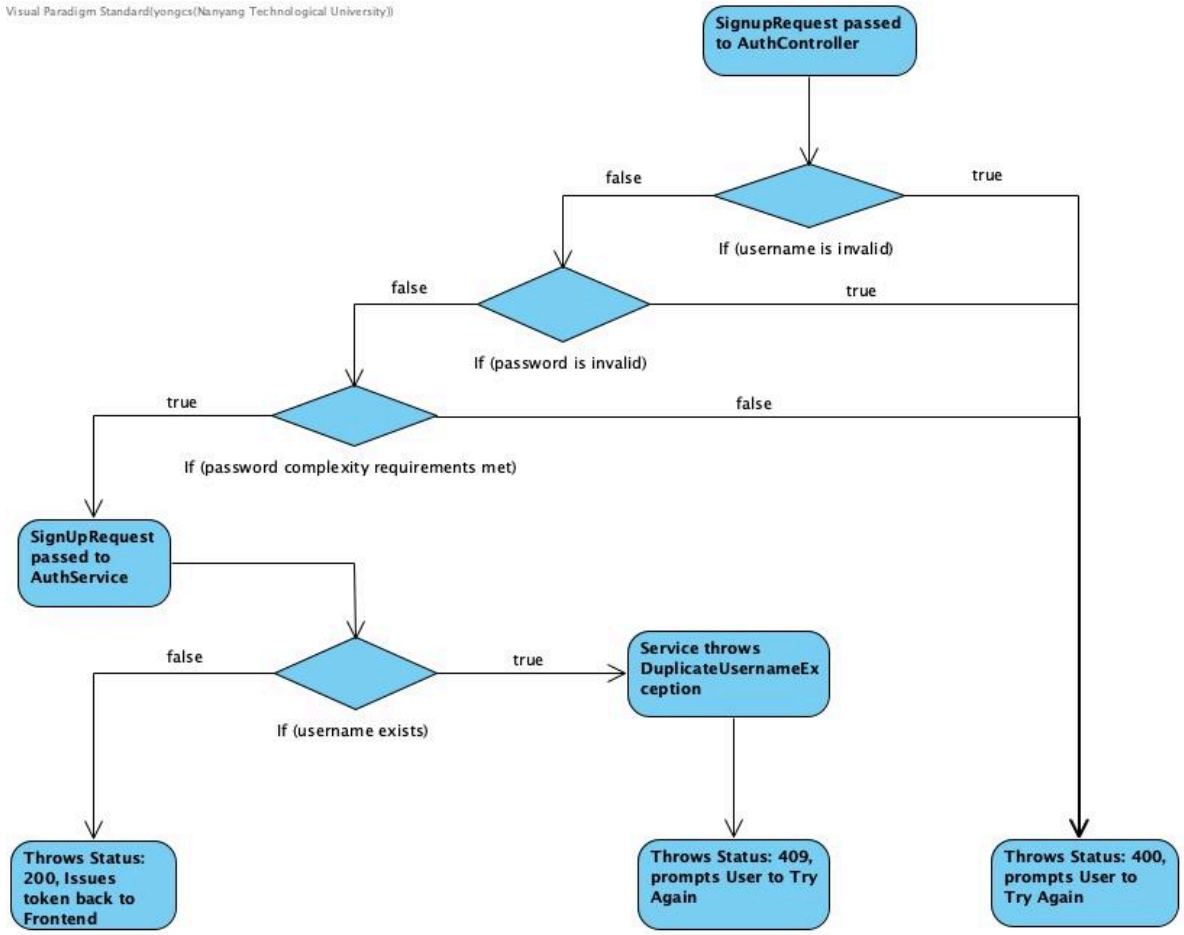
- **Approach**: Black-box testing
- **Purpose**: To validate that all components work correctly together by testing API endpoints end-to-end.
- **Scope**: Confirms proper interaction between layers such as controllers, services, and the database.

B.6.1 Unit Testing

We chose to demonstrate unit testing using **AuthServiceTest** and **AuthControllerTest**, as these components contain more complex logic compared to others. The complete unit test results can be found in **/docs/Unit Test Results.html**.

B.6.1.1 Signup

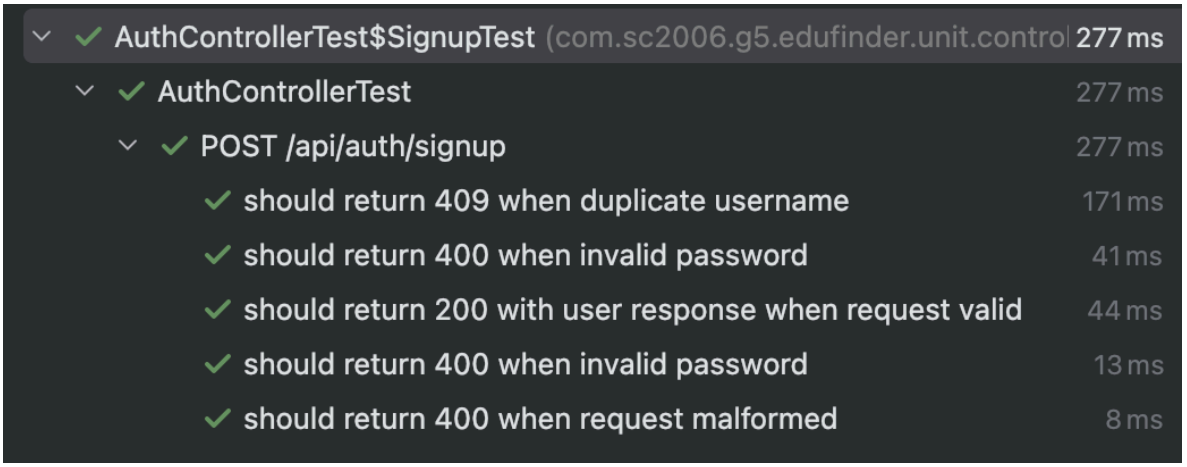
Control Flow Diagram:



Component: AuthController
Function:
Validates request input (username and password) before delegating to AuthService, and sets the authentication token issued by the service in the response cookie.

Test ID	Input	Equivalence Class	Expected Output
T1	username: valid_username password: AbCd123@	Valid	Status: 200 Token: Issued and Set Body: User Information
T2	{}	Malformed input	Status: 400
T3	password: AbCd123@	Missing Username	Status: 400
T4	username: valid_username	Missing Password	Status: 400
T5	username: user1 password: AbCd123@	Short Username	Status: 400
T6	username: user12341234123 password: AbCd123@	Long Username	Status: 400
T7	username: user12 password: AbCd123@	Valid Shortest Username	Status: 200
T8	username: user1234123412 password: AbCd123@	Valid Longest Username	Status: 200
T9	username: valid_username password: ABCD123@	No Lower Case Password	Status: 400
T10	username: valid_username password: abcd123@	No Upper Case Password	Status: 400
T11	username: valid_username password: Abcdefg@	No Digit Password	Status: 400
T12	username: valid_username password: Abcd1234	No Special Symbol Password	Status: 400
T13	username: valid_username password: Abab12@	Short Password	Status: 400
T14	username: invalid_username password: AbCd123@	Service throw DuplicateUsernameException	Status: 409

Result:



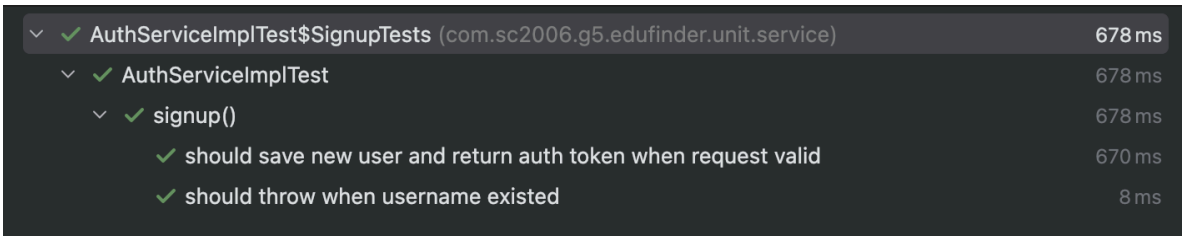
Note that some test cases are grouped together due to similar nature:

1. T2, T3, T4
2. T5, T6, T7, T8
3. T9, T10, T11, T12, T13

Component: AuthService
Function:
Check if the username exists, create a user and return a token.

Path ID	Predicate Coverage	Input Condition	Expected Output
P1	DuplicateUsername = false	Username does not exist	Password is encoded; user is saved; token with user ID is returned
P2	DuplicateUsername = true	Username already exists	DuplicatedUsernameException is thrown

Result:

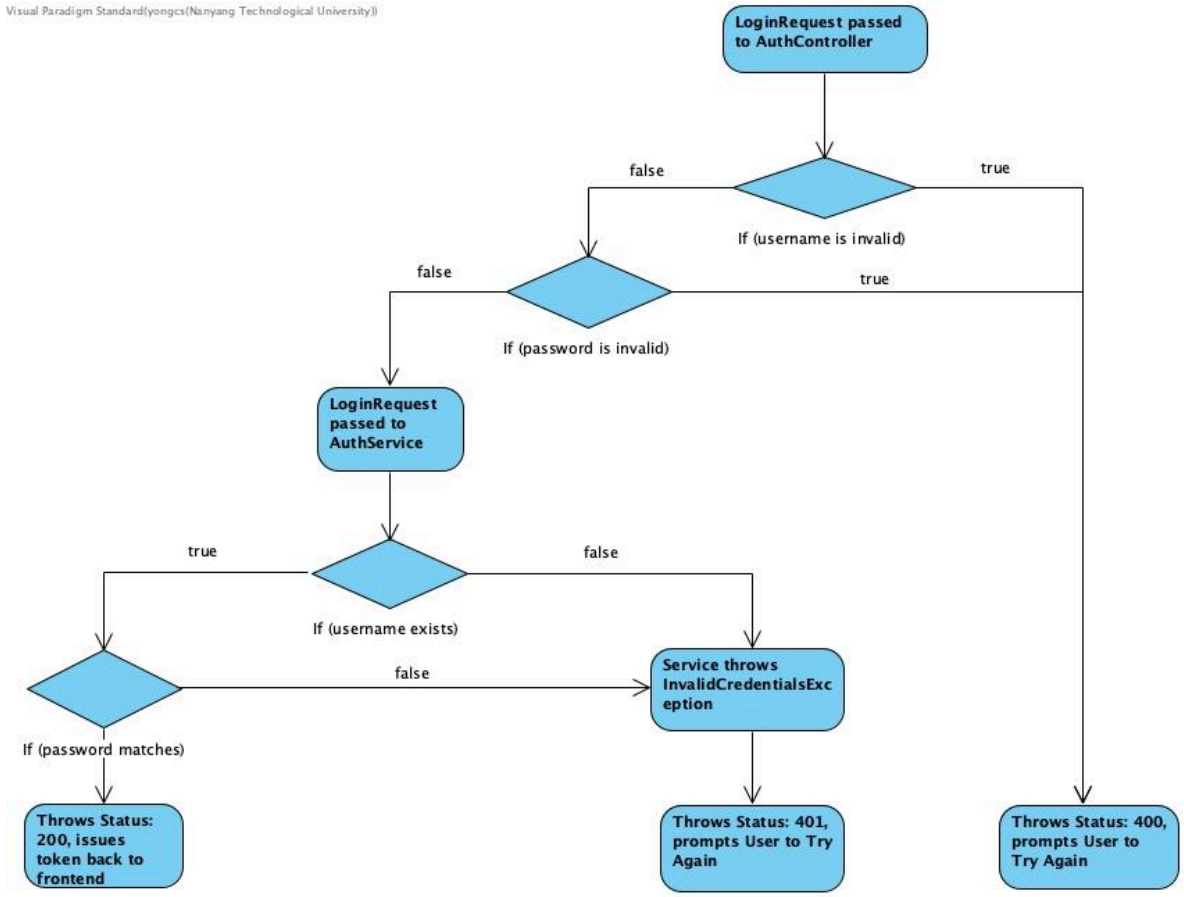


Note that some test cases are grouped together due to similar nature:

1. T2, T3, T4

B.6.1.2 Login

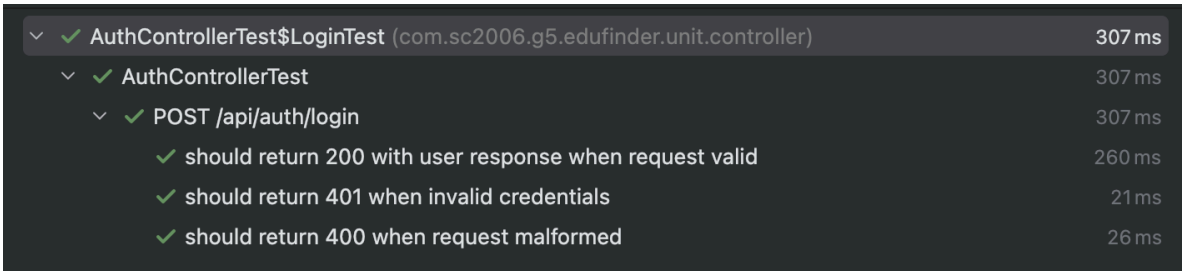
Control Flow Diagram:



Component: AuthController
Function:
Validates request input (username and password) before delegating to AuthService, and sets the authentication token issued by the service in the response cookie.

Test ID	Input	Equivalence Class	Expected Output
T1	username: valid_username password: AbCd123@	Valid	Status: 200 Token: Issued and Set Body: User Information
T2	{)	Malformed input	Status: 400
T3	password: AbCd123@	Missing Username	Status: 400
T4	username: valid_username	Missing Password	Status: 400
T5	username: valid_username password: wrong_password	Service throw InvalidCredentialsException	Status: 401

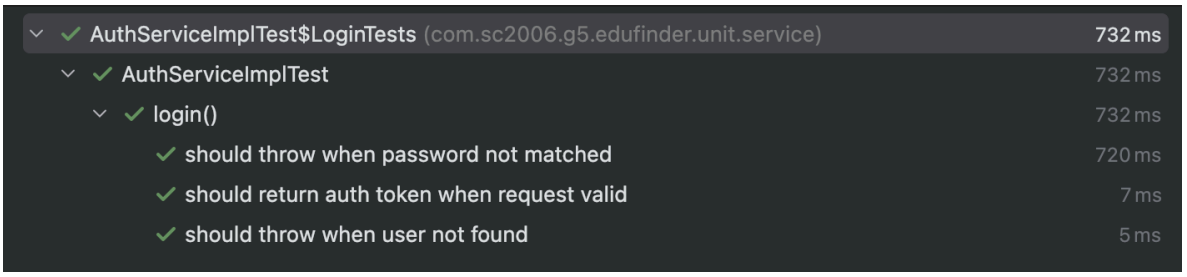
Result:



Component: AuthService
Function:
Check if the username exists, password matches and return a token.

Path ID	Predicate Coverage	Input Condition	Expected Output
P1	UsernameExisted = true PasswordMatched = true	Credential is correct	token with user ID is returned
P2	UsernameExisted = false	Username doesn't exist	InvalidCredentialsException is thrown
P3	UsernameExisted = true PasswordMatched = false	Password doesn't match	InvalidCredentialsException is thrown

Result:



B.6.2 Integration Testing

Due to time constraint, we have only written integration testing for two important endpoints **GET /api/schools** and **POST /api/auth/login**.

B.6.2.1 GET /api/schools

Test Objective:
To confirm that valid API requests return a 200 OK status with the correct school data integrated from all sources and properly persisted in the database.

Expected Result:

1. HTTP Status: 200
2. Response body contains two schools with accurate details for name, contact information, subjects, CCAs, and programmes.
3. Corresponding entries are successfully stored in the database.

Result: