Software Requirements Specification

for

Study Buddy

Version 1.0 approved

Prepared by

CHUA QIN DI
NICHOLAS CHANG CHIA KUAN
REDDIPALLI SAI DHARSHAK SATHAVAHANA
SHARMA HIMANSHU
VONA TANISHA
WU YIQING

NTU, SCEC Group E3, Mission: Passable

2 April 2025

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
	10 Apr 25	Added the descriptions for SRS Updated and added diagrams Included GUI displays of the app	1.0

1. Introduction

1.1 Purpose

This document outlines the software requirements and specifications of *Study Buddy*, a productivity and collaboration tool designed to help students manage study sessions, track tasks, and stay accountable through social features.

This document serves as a comprehensive guide for the development team, testers, and other stakeholders to ensure a shared understanding of the system's functionality and constraints. It covers the core components of the application, including the front-end, back-end and API integrations.

1.2 Document Conventions

This Software Requirements Specification (SRS) document follows standard typographical conventions to ensure clarity and consistency. Text in bold represents GUI elements, user inputs, or other emphasis. Text in italics denotes newly introduced terms, important notes, or areas requiring special attention. Source code, API endpoints, and database queries are presented in monospace font.

Priorities for high-level requirements are inherited by detailed requirements unless explicitly stated otherwise.

Font: Arial
Heading1: Size 18, Bold
Heading2: Size 14, Bold
Heading3: Size 12, Bold
Content: Size 11
Spacing in content: 1 line spaced
Further conventions on special terms used throughout this document are described in **6.1: Data Dictionary.**

1.3 Intended Audience and Reading Suggestions

This SRS document is structured to cater to the needs of various stakeholders, including the development team, documentation writers, project managers, marketing team, testers, and end-users of *Study Buddy*.

Readers are encouraged to begin with Section 1 for a high-level understanding, followed by Section 2: Overall Description for system context and user environment.
Section 3 and Section 4 are most relevant for technical stakeholders, covering system features and interface requirements in detail.
Section 5 and Section 6 contain nonfunctional requirements, diagrams, and use cases that further define expected behaviors and system structure.

1.4 Product Scope

Study Buddy is a mobile application designed to support students in managing their academic responsibilities and fostering peer collaboration. The app offers an interactive platform where users can create personalized task schedules, monitor academic progress, and engage in collaborative study sessions with friends and peers.

The goal of Study Buddy is to make studying more engaging, structured, and socially accountable, ultimately helping students stay on track and improve their academic performance. By integrating task management, social features, and progress tracking into a single app, Study Buddy encourages motivation and productivity in both individual and group learning contexts.

This app aligns with broader educational goals of promoting self-directed learning, digital literacy, and student well-being through technology. Study Buddy Version 1.0 serves as the initial release and foundational product, with scalability in mind for future integration with institutional learning tools and expanded features.

1.5 References

1. Collaborative Study App – Source Code Repository
   https://github.com/rsds26/collaborative-study-app

2. React Native Documentation:
   https://reactnative.dev/

3. Supabase – Open Source Backend-as-a-Service:
   https://supabase.com/

4. Expo – Framework and Platform for Universal React Apps
   https://expo.dev/

5. Google Maps Platform – Location and Mapping APIs
   https://developers.google.com/maps

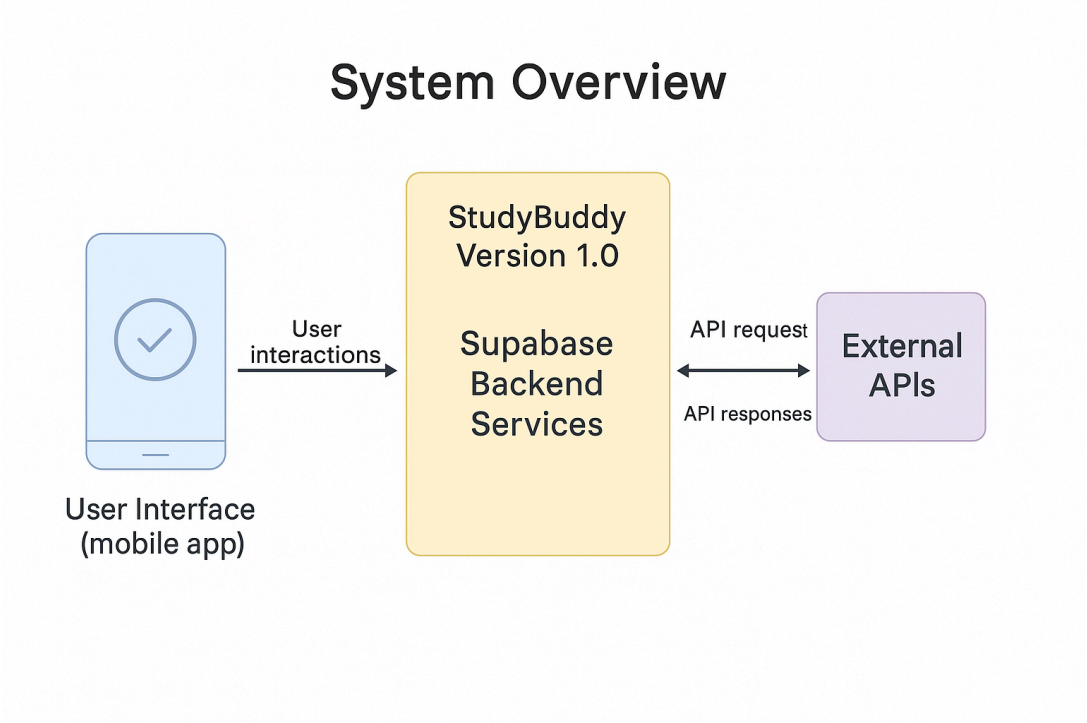
6. IEEE 830-1998 – IEEE Recommended Practice for Software Requirements Specifications
   https://ieeexplore.ieee.org/document/720574

2. Overall Description

2.1 Product Perspective

Study Buddy Version 1.0 is a new, self-contained mobile application designed to support students in organizing their academic responsibilities and encouraging collaborative learning. It is not a follow-on or replacement of any existing system, although it draws inspiration from productivity and educational apps commonly used by students, such as calendar planners.

Study Buddy functions as a standalone solution, but it is designed with scalability in mind, enabling potential future integration with institutional learning management systems (LMS) through API endpoints.



System Overview Diagram

2.2 Product Functions

Study Buddy provides features that empower students to manage tasks, monitor progress, and collaborate with peers. Major functions include:

2.2.1 Account Management

a. User Sign Up
b. User Login
c. User Sign out

2.2.2 Task Management

a. Add Task
b. Edit Task
c. Remove Task
d. Mark Task as Completed

2.2.3 Friend Management

a. Add Friend
b. Remove Friend

2.2.4 Friend's Task

a. View Friend's Task

2.2.5 Search Nearby Library

a. Input postal code
b. Search nearby libraries

2.2.6 Timetable

a. View Timetable

2.2.7 Progress Overview

a. Overall Dashboard View
b. Progress Bar
c. Outstanding Tasks

2.3 User Classes and Characteristics

The anticipated user classes include:

1. **Students (Primary Users)**

Aspect	**Description**
Frequency of Use	Frequent
Age	13-26
Subset of Product Functions Used	All
Technical Expertise	● Low ● Owns a mobile phone with stable internet connection
Characteristics	● This group comprises of secondary school students to tertiary level students ● Users may use the app daily to track academic progress

2. **Administrator**

Aspect	**Description**
Frequency of Use	Frequent
Subset of Product Functions Used	All
Technical Expertise	● High ● Expected to have strong technical skills to handle system maintenance and support.
Characteristics	● Manage user accounts, moderate content, and address technical issues. ● Security features to handle sensitive student data. ● Support for troubleshooting and resolving user inquiries.

2.4 Operating Environment

The production and development environment of *Study Buddy* will be covered under this section.

2.4.1 User Device Permissions

Study Buddy requires access to internet connection to operate as all functionalities of the application are based on HTTP requests between frontend and backend server.

2.4.2 Development Environment

Development Environment	**Description**
Front-end: React Native, Typescript, Expo	React Native is an open source framework for building Android and iOS applications using React and the app platform's native capabilities. With React Native, developers use JavaScript and TypeScript to access the platform's APIs as well as to describe the appearance and behavior of UI using React components: bundles of reusable, nestable code Expo is a production-grade React Native Framework. Expo provides features like file-based routing, high-quality universal libraries, and the ability to write plugins that modify native code without having to manage native files.
Back-end: Supabase	Supabase is an open-source Backend-as-a-Service (BaaS) platform that enables developers to build secure and scalable web and mobile applications quickly. It is powered by a PostgreSQL database, offering full SQL support along with real-time capabilities, authentication, storage, and more.

2.5 Design and Implementation Constraints

1. The app uses Supabase as the database
2. The app relies on Supabase for user login and registration. Hence, it is subject to whether Supabase services are available. In cases where Supabase services are unreachable , the app may not function as expected
3. The app integrates with third-party APIs (e.g., Google Maps API) for features like location-based library search. These services are external dependencies and are subject to availability, rate limits, and data accuracy provided by the external providers.
4. The UI and functionality must behave consistently across iOS (14+) and Android (10+) devices, which may involve resolving differences in layout rendering, permissions handling, and API behaviors between platforms.

2.6 User Documentation

A demonstration of how the app works along with a guide to install and run the app is provided in the README.md file on the GitHub repository which contains the source code for this app

2.7 Assumptions and Dependencies

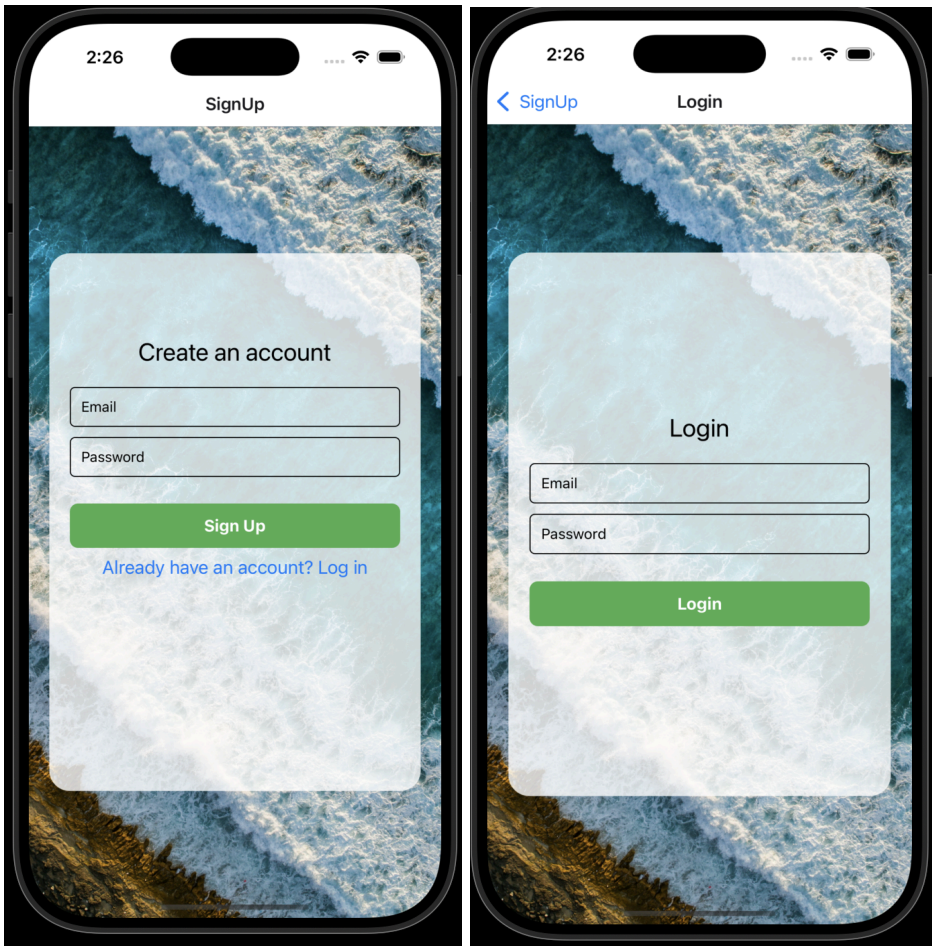
Assumptions:
1. The app relies on the availability of third-party API for information such as location data and opening hours.
2. It is assumed that the information obtained from the API is accurate
3. In cases where the API services become unavailable, the app may not meet some of the requirements specified in this document.
4. The end-users are required to have a stable internet connection to use the application.
5. The application's performance is contingent on the chosen cloud hosting service's reliability and uptime.

3. External Interface Requirements

3.1 User Interfaces

This section will showcase all GUI of *Study Buddy's* mobile application

3.1.1 User Authentication



The User Authentication consists of the **Sign Up** and **Login** Page.
Sign Up:
New users can create an account by entering a valid email and password.

- A Sign Up button submits the information to register the user.
- A link is provided to redirect existing users to the login page.

Login:

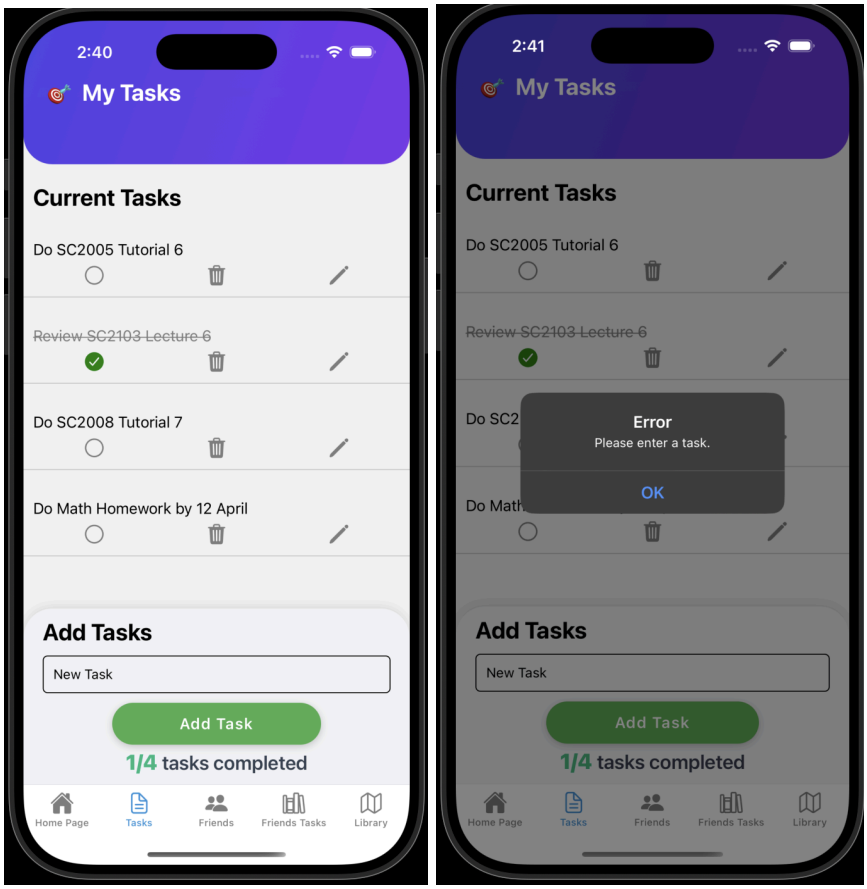
- Existing users can log in using their email and password.
- The Login button authenticates the credentials and grants access to the app.

3.1.2 Home Page / Dashboard Overview

The Home Page provides an overview of the outstanding tasks that the users have.
In the **“Today’s Focus”** section, two uncompleted tasks are shown by default to help users prioritize their day.
By clicking the **“View All”** button, users can expand the list to view all remaining tasks in a drop-down format.
Additionally, the **“Task Progress”** and **“Study Streak”** features are designed to motivate users by visually tracking their productivity and consistency.

Users can also **Sign Out** of their account at the top left of the **Home Page**.

3.1.3 Task Page



The **Task Page** allows users to view, add, edit, complete, or delete their tasks.

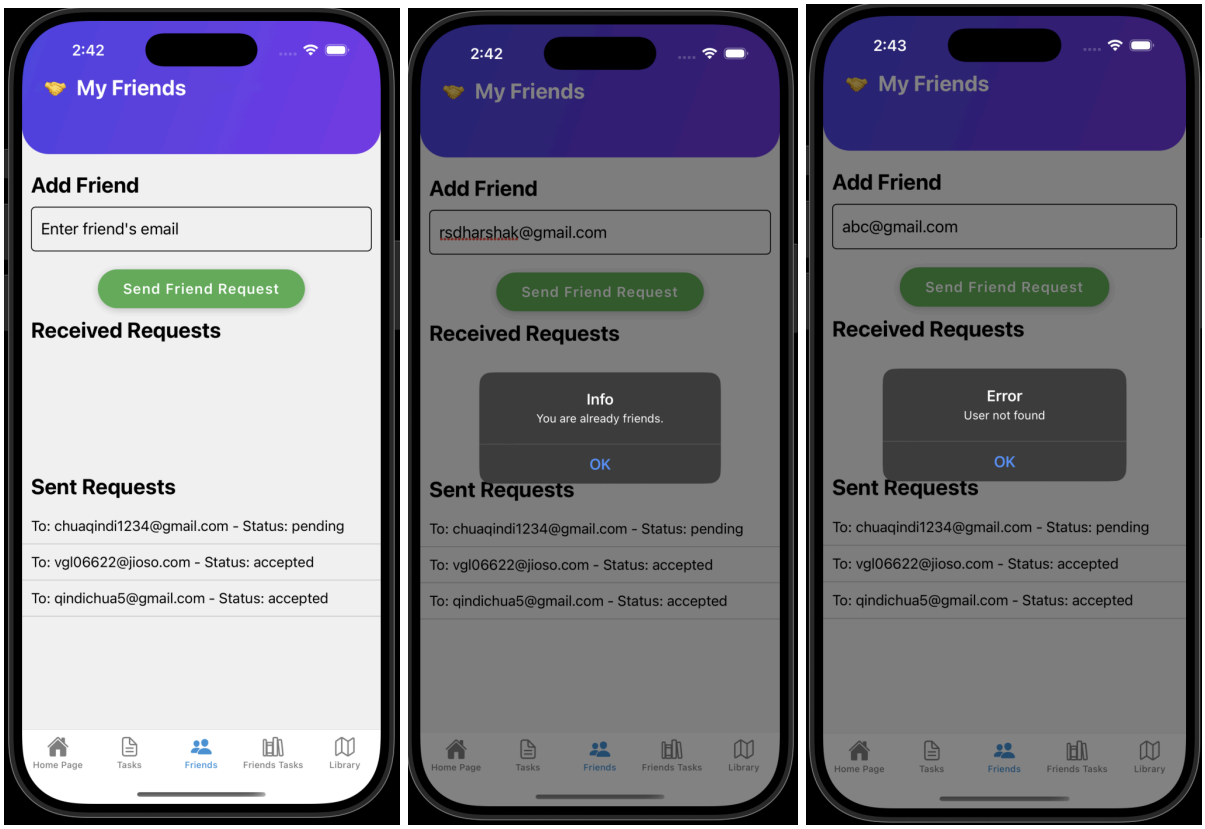
Current Tasks Section:
Displays a list of all pending and completed tasks. Each task is accompanied by:

- A circle icon to mark it as complete or incomplete.
- A trash icon to delete the task.
- A pencil icon to edit the task text.

Task Completion Indicator:
Completed tasks are visually distinguished by a strikethrough effect and a green checkmark inside the circle.

Add Tasks Section:
Users can type a new task into the input field and press the green **Add Task** button. If the input is empty and the user attempts to add a task, an **error message** appears, as shown in the second screenshot, prompting the user with *"Please enter a task."*

3.1.4 Friends Tab



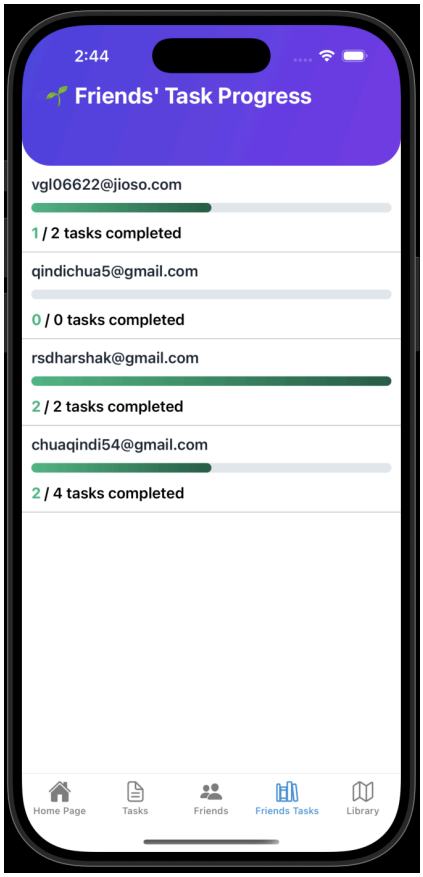
The **Friends Tab** allows users to manage friend connections within the app. It includes features for sending, receiving, and tracking friend requests.

Key Components:

Add Friend:
Users can enter another user's email address and tap the **"Send Friend Request"** button to initiate a friend request.

Error Handling:
Empty or Invalid Input: Request is blocked if the email field is empty or improperly formatted.
User Not Found: An error popup ("User not found") appears if the entered email is not associated with any registered user.
Already Friends: An info popup ("You are already friends") appears if the user has already added the entered email as a friend.

3.1.5 View Friend's Tasks

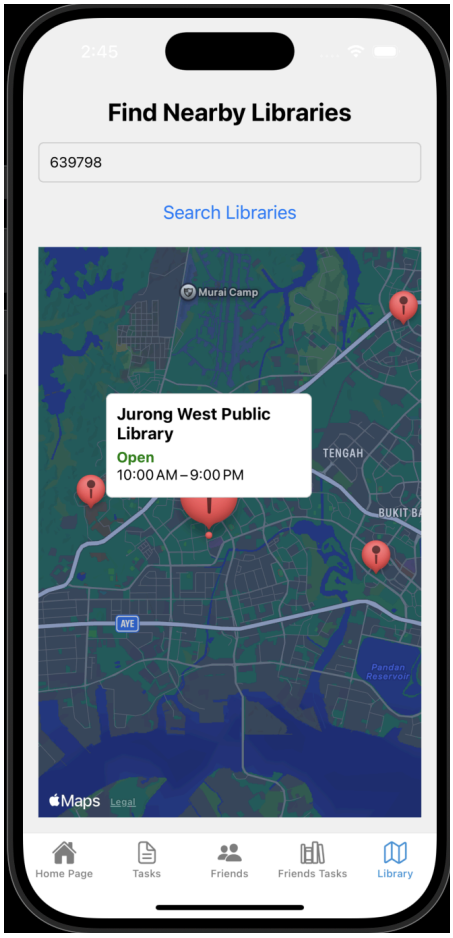


The **Friends' Task Progress** page allows users to view the task completion status of their friends for added motivation and accountability.

Task Progress Indicator:
Below each progress bar, the app shows the number of tasks completed versus the total tasks (e.g., "1 / 2 tasks completed").

Progress Visualization:
A horizontal bar visually reflects each friend's task completion ratio, helping users quickly compare progress.
This feature encourages social accountability and fosters a sense of shared productivity among users.

3.1.6 Search Nearby Library



The **Search Nearby Library** feature helps users locate public libraries based on a provided postal code.

Map Integration:
Libraries are shown as red location pins. Tapping a pin reveals key details, such as:

- Library name (e.g., Jurong West Public Library)
- Current status (e.g., "Open")
- Operating hours

This feature enhances convenience by helping users discover and access nearby study spaces quickly.

3.2 Hardware Interfaces

This section describes the hardware interfaces required for *Study Buddy* to perform effectively and reliably.

Requirements	**Description**
Operating System	Mobile Devices with Android 10 or above, or iOS 14 or above Minimum 2 GB RAM
Network Connection	Wireless Network Interface Card (WNIC), or cellular modem for internet access
Interaction	A responsive capacitive touchscreen for navigation and input

3.3 Software Interfaces

1. Front-End: React Native
   Role: Provides user interface and handles user interactions.
   Communication: Sends HTTP requests to the back-end for data such as task information and friend information

2. Back-End: Supabase
   Role: Manages business logic, data processing, and security.
   Communication: Exposes RESTful APIs for the front-end to send and receive data; communicates with the database via SQL queries.
   Key Functions: Handles authentication, retrieves user data

3. Database: Supabase/ PostgresQL
   Role: Stores persistent data such as user profiles, task information, friend list information
   Communication: Receives SQL commands from the back-end to store and retrieve data.
   Key Data: User accounts, task information, friend lists

4. Third Party APIs: Google Maps API
   Role: Provides geolocation services and interactive map features (e.g., finding nearby libraries or study locations)
   Communication: Accessed via HTTPS requests or SDKs integrated into React Native. Returns JSON responses for location data.
   Key Data: Location data, location opening hours

3.4 Communications Interfaces

Requirement	Description
Network Protocols	HTTPS is used for all communication between the app and Supabase back-end to ensure secure data transmission.
API Communication	RESTful APIs are used for operations such as authentication, CRUD operations on tasks, and user data. JSON is the primary data format for request/response bodies.
Authentication Protocols	OAuth 2.0 and email/password authentication methods are supported via Supabase. All sessions are handled with secure tokens (JWT).
Encryption Standards	All data-in-transit is encrypted via TLS (HTTPS). Sensitive data such as passwords are never stored in plain text and are hashed using Supabase's internal security mechanisms.
Email Integration	Supabase provides email-based verification and password reset capabilities.

4. System Features

4.1 Functional Requirements

4.1.1 The app requires users to register for a new account.

4.1.1.1 The system must request for the user's password and email.

4.1.1.2 The system must validate all information provided by the user for user registration.

4.1.1.2.1 The system must validate that the username provided is not already registered.

4.1.1.2.2 The system must validate that the email provided is valid and not registered.

4.1.1.2.3 The system must validate that the password provided must meet the following requirements.

4.1.1.2.3.1 Minimum 8 characters in length.

4.1.1.2.3.2 At least 1 uppercase letter.

4.1.1.2.3.3 At least 1 numeric digit.

4.1.1.2.4 The system must display error messages when validation fails, specifying the reason.

4.1.1.3 The system must save the user’s information to the database upon successful registration.

4.1.1.4 The system must send an email verification link to complete the registration process.

4.1.2 The app requires users to login to their account to use the app.

4.1.2.1 The app must request the user for their username and password for verification.

4.1.2.1.1 Username and Password must be registered in the database.

4.1.2.2 The app must reject the login request if the login credentials are invalid.

4.1.2.2.1 The login request must be rejected if the email is incorrect.

4.1.2.2.2 The login request must be rejected if the password is incorrect.

4.1.2.3 If the app rejects the login request, the app must inform the user of the rejection.

4.1.2.4 The app must return to home page with the user logged in upon successful verification.

4.1.2.5 The app must allow the user to sign out of their account.

4.1.2.5.1 The users shall be able to tap on a ‘sign out’ button to sign out of their account.

4.1.2.5.2 The users shall be shown the login page for their account after sign out.

4.1.3 The app allows users to manage their profile.

4.1.3.1 The system must be able to display user’s information on their profile such as name, password, email, and courses.

4.1.3.2 Users shall be able to update their information on their profile.

4.1.3.2.1 The system must validate all information provided by the user for profile update.

4.1.3.2.2 The system must save the user's updated information to the database.

4.1.4 The app allows users to manage their friend list.

4.1.4.1 The system must be able to display the user's friend list.

4.1.4.2 Users shall be able to add friends by sending a friend request.

4.1.4.2.1 Username must exist in the database in order to send a friend request.

4.1.4.3 The system shall validate the entered email before processing:

4.1.4.3.1 The system shall verify the email is not empty.

4.1.4.3.2 The system shall shall verify that no existing relationship (pending or accepted) exists between the users.

4.1.4.3.3 The system shall prevent users from sending friend requests to themselves.

4.1.4.4 Users must be able to accept or reject friend requests.

4.1.4.4.1 The system shall refresh the friends list to include the newly accepted friend.

4.1.5 The app allows users to manage courses in their profile.

4.1.5.1 Users shall be able to add courses.

4.15.2 The system must be able to display the user's courses.

4.1.5.2.1 Courses must be registered into the database via Add Course in order for users to view them.

4.1.5.3 Users shall be able to remove their own courses.

4.1.5.3.1 Course requested to be removed by the user must be deleted from database.

4.1.5.4 The system shall allow users to view their friend's courses.

4.1.5.4.1 User must be registered as friends with another user in the database in order for users to view another user's courses.

4.1.6 The app allows users to manage the status of tasks in their timetable.

4.1.6.1 Users shall be able to add tasks.

4.1.6.1.1 The system shall provide a text input field for users to enter new task titles.

4.1.6.1.2 The system shall validate that the task title is not empty before submission.

4.1.6.2 Users will be able to edit existing tasks.

4.1.6.3 The system must be able to display the user's tasks.

4.1.6.2.1 Tasks must be register into the database via Add Task in order for users

to view them.

4.1.6.2.2 The system shall display task manipulation options (complete, edit, delete) for each task.

4.1.6.4 Users shall be able to remove tasks.

4.1.6.3.1 Tasks requested to be removed by the user must be deleted from database.

4.1.6.5 Users shall be able to mark tasks as completed.

4.1.6.6 The system will display the task completion statistics on the home screen and task screen.

4.1.6.6.1 The system shall update task statistics in real time when tasks are added, completed or deleted.

4.1.7 The app allows users to find the nearest library.

4.1.7.1 The system shall display an input field that allows users to enter a postal code.

4.1.7.2 The system shall validate that the postal code is not empty before processing the Search.

4.1.7.3 The system shall search for libraries within a 7-kilometer radius of the specified location.

4.1.7.4 The system displays each library on an active map as a pin. Tapping a pin shall reveal a call-out containing:

4.1.7.4.1 Library name

4.1.7.4.2 Current open/closed status

4.1.7.4.3 Operating hours

4.1.8 The app allows users to view the Home-page dashboard after logging in.

4.1.8.1 The system must display a personalised greeting banner that includes the user's first name and a welcome message upon successful login.

4.1.8.2 The system shall display the user's profile picture and a "Sign Out" link in the banner.

4.1.8.2.1 Selecting "Sign Out" shall execute the sign-out flow defined in 4.1.2.5.

4.1.8.3 The system shall present a "Today's Focus!" section that lists high-priority tasks due within the next 24 hours.

4.1.8.4 The system provides a Task progress section that displays:

4.1.8.4.1 The ratio of completed tasks for the current day.

4.1.8.4.2 A progress bar that fills proportionally.

4.1.8.5 The system shall provide a Study Streak card that shows the number of consecutive days in which the user completed at least one task.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

5.1.1 Usability

5.1.1.1 The app must be simple and intuitive for users.
5.1.1.2 Users should be able to complete common workflows without consulting external documentation.
5.1.1.3 The interface should follow standard mobile UI/UX design principles (e.g., consistent navigation, responsive layout).
5.1.1.4 All icons and buttons must include tooltips or labels for clarity.
5.1.1.5 Key functions (e.g., adding a task should be accessible within three taps or fewer.

5.1.2 Reliability

5.1.2.1 User data must be encrypted and secured.
5.1.2.2 Data should be backed up automatically at least once per week to ensure recovery in case of failure.
5.1.2.3 The app must be able to handle at least 200 concurrent users without any performance issues.
5.1.2.4 Users should be able to maintain at least 30 tasks without any performance issues.
5.1.2.5 Users should be able to maintain at least 50 friends without any performance issue.
5.1.2.6 The system must be upgradable as user count increases.
5.1.2.7 The system must maintain an annual uptime of at least 99.9%.
5.1.2.8 Error messages must be user-friendly and should not expose internal system logic or sensitive information.

5.1.3 Performance

5.1.3.1 App screens must load and display data in under 4 seconds on average devices.
5.1.3.2 Once a task/course/friend is added, the system must register the data into the database and display it to the users within 2 seconds.
5.1.3.3 The app should consume minimal device resources (CPU, battery) during normal usage

5.1.4 Supportability

5.1.4.1 The app must be fully compatible with Android (v10.0 and above) and iOS (v14.0 and above.
5.1.4.2 The UI must adapt to various screen sizes and resolutions across different mobile devices.
5.1.4.3 Codebase should be modular and follow clean architecture principles to simplify maintenance and future updates.

5.2 Safety Requirements

1. The application must ensure the confidentiality, integrity, and privacy of all user data stored and transmitted, in order to prevent unauthorized access, data leaks, or breaches.
2. All sensitive operations, such as account creation, login, and task/friend management, must be conducted over secure, encrypted connections (HTTPS/TLS).
3. Administrative access to user data shall be logged and audited at least once a month to ensure accountability and prevent unauthorized tampering or misuse.
4. The system must prevent actions that could result in accidental data loss, such as permanent deletions without user confirmation dialogs.
5. In the event of a system crash or API failure, the application should provide user-friendly error messages to avoid confusion or data corruption.

5.3 Security Requirements

1. The system must enforce secure user authentication through email and password using Supabase's authentication service.
    a. Passwords must:
        i. Be at least 8 characters long
        ii. contain at least one uppercase letter and one number
        iii. be encrypted and never stored in plain text.
2. Sensitive user data stored in the database (e.g., email, password hashes) must follow Supabase's internal security and encryption standards.
3. All user inputs must be validated and sanitized to prevent common vulnerabilities such as SQL injection.

5.4 Software Quality Attributes

Availability: The system should be available 99.9% of the time, excluding scheduled maintenance windows, to ensure consistent access for users.

Reliability: The application should have an error rate of less than 0.1% to ensure reliable performance. It should function as expected without crashes or data loss.

Usability: The application must be easy to use. Users should be able to learn how to use the app without a manual. Interfaces must be clean, intuitive, and consistent across platforms.

Maintainability: The code should be modular, well-documented, and follow standard design patterns. This allows future developers to update or add features easily.

5.5 Business Rules

Users are assigned roles that define their permissions within the application. There are two main roles: primary users and administrators.

1. Primary users (Students) can access only their own data. They can view, edit, and manage their tasks, courses and friends.

2. Administrators have full access to all user data. This access is granted only for system support, maintenance, and troubleshooting.

Access to sensitive user data is strictly role-based, with regular users only having access to their data, and administrators to all user data for support and maintenance purposes only. Any administrative actions must be logged for transparency and reviewed regularly.

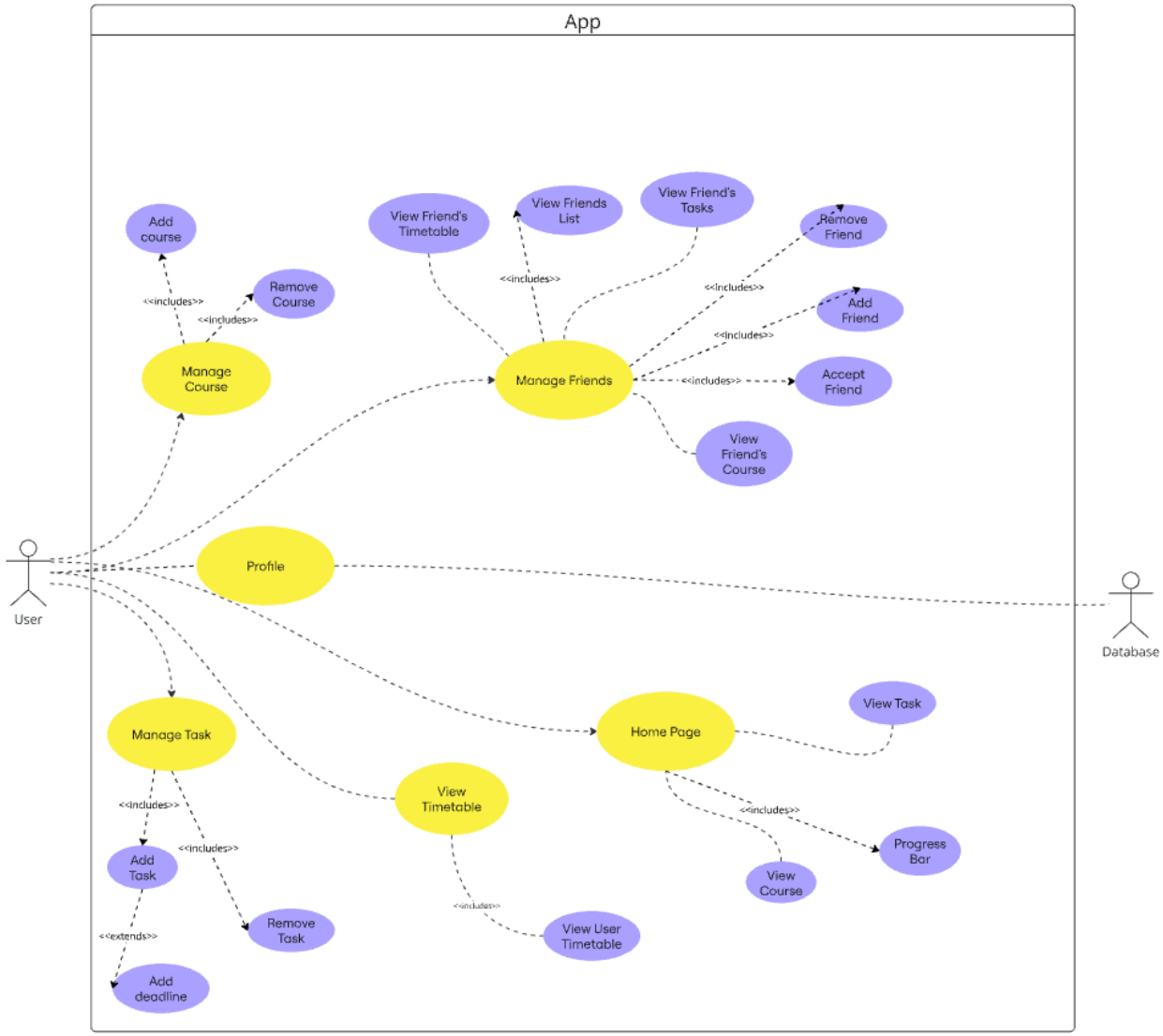
6. Other Requirements

6.1 Data Dictionary

Term	Definition
Course	A course is a basic unit of teaching. A course must be either compulsory or elective. A course must include lectures and tutorials . Some courses may have labs.
Task	A piece of work to be done or undertaken.
Timetable	A chart showing the times at which events are scheduled to take place or when tasks are scheduled to be completed.
Notifications	An alert (typically a pop-up or other message) generated by an application to notify the user of a new message or update.
Friend (Online)	A person met and communicated with through the internet.
Friend Request	An online invitation to allow a user to become a Friend of the sender. Receiver can accept or reject the Friend Request
Friend List	A list of people that a user is connected to on a social networking site.
Study Group	A group of people who gather virtually to collaboratively study without needing to be physically present in the same location.
Username	Unique username chosen by the user
Password	Encrypted password for authentication Requires at least 8 characters long, containing 1 capital letter and 1 number
Email	User’s email address (must be unique)
Add Course	A function allowing users to add a course associated to the user
Add Task	A function allowing users to add a task associated to the user
Create Study Group	A function allowing users to create a Study Group by adding Friends into the group
Task Status	The state of a task, such as "Pending", "Completed", or "Overdue".
Friendship Status	The state of the user’s relationship with another user, which could be "Pending", "Accepted", or "Rejected".
Library Search	A function that allows users to search for nearby libraries by postal code.

6.2 Use Case Diagram

Refer to https://github.com/softwarelab3/2006-SCEC-E3/tree/main/lab5/Diagrams for a clearer image.



6.3 Use Case Descriptions

Use Case ID:	UC-DB-1		
Use Case Name:	Register		
Created By:	Chua Qin Di	Last Updated By:	Chua Qin Di
Date Created:	3 Feb 2025	Date Last Updated:	2 Apr 2025

Actor:	User
Description:	**Register** allows new users to create an account in our system database. To create an account, users must input Username, Email, Password
Preconditions:	1. Username must not be taken 2. Email must not be linked to existing account 3. Passwords must meet requirements: must be at least 8 characters long, containing 1 capital letter and 1 number.
Postconditions:	
Priority:	High
Frequency of Use:	Once
Flow of Events:	1. User inputs required information: Username, Email, Password. 2. System validates the information. 3. System creates the account. 4. System updates account information in database. 5. System directs the user to the homepage.
Alternative Flows:	Username taken 1. System displays “User account already exists” 2. System prompts user to input a different username 3. Continue from main flow step 1 Email linked to existing account 1. System displays “Email linked to existing account” 2. System prompts user to log in to existing account or register account 3. If log in, System directs the user to the Login page 4. If register account, continue from main flow step 1 Password does not meet requirements 1. System displays “Password does not meet requirements” 2. System prompts user to re-enter their password. 3. Continue from main flow Step 1.
Exceptions:	

Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	UC-DB-2		
Use Case Name:	Login		
Created By:	Chua Qin Di	Last Updated By:	Chua Qin Di
Date Created:	3 Feb 2025	Date Last Updated:	25 March 2025

Actor:	User
Description:	**Login** allows users to enter the app and use the features intended for registered users of the app. To login to the system, the user must input their username and the corresponding password.
Preconditions:	User must have an existing account in the database
Postconditions:	System displays the main homepage
Priority:	High
Frequency of Use:	Occasional
Flow of Events:	1. User enters username and password on the login page. 2. System validates the user's username and password. 3. System shows the main homepage.
Alternative Flows:	Invalid Username 1. System cannot find the username in the database. 2. System displays an error message. 3. System prompts the user to input username and password or register an account. 4 Use case resumes at main flow step 1 or Register account page Invalid Password 1. Password does not match for the username. 2. System displays an error message. 3. System prompts user to input username and password. 4. Use case resumes at main flow step 1.
Exceptions:	
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	UC-PF-1		
Use Case Name:	Manage Profile		
Created By:	Wu Yiqing	Last Updated By:	Wu Yiqing
Date Created:	3 Feb 2025	Date Last Updated:	25 March 2025

Actor:	User
Description:	**Manage Profile** allows users to view and update personal information.
Preconditions:	1. User must have an existing account in the database. 2. The user must have logged into the app.
Postconditions:	
Priority:	High
Frequency of Use:	Occasional
Flow of Events:	1. User navigates to profile settings. 2. User selects the option to update profile. 3. User inputs updated information. (e.g., name, password and email) 4. User submits updated information. 5. System validates the updated information. 6. System updates the new profile into database. 7. System displays new profile for user.
Alternative Flows:	
Exceptions:	Invalid Data Entry 1. User enters an invalid information format. 2. System displays error message and rejects update profile request. 3. User must rectify the invalid format before proceeding. User cancels update profile request. 1. User chooses to exit the update profile page. 2. App returns to home page.
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	UC-FR-1		
Use Case Name:	Manage Friends		
Created By:	Vona Tanisha	Last Updated By:	Vona Tanisha
Date Created:	3 Feb 2025	Date Last Updated:	25 March 2025

Actor:	User
Description:	**Manage Friends** allows users to add existing users as friends or allows users to accept another user's Friend Request or remove friends from their friend list.
Preconditions:	Friend must be an existing user in the database
Postconditions:	Request will be sent to friend, pending his approval Friend shows up on Friend List
Priority:	Medium
Frequency of Use:	Occasional
Flow of Events:	1. User inputs another existing user's username to add as friend 2. System displays "Friend request sent" 3. User chooses to press Accept or Reject Friend Request 4. User removes friend from Friend List
Alternative Flows:	Cannot add yourself as Friend 1. System displays error message and prompt user to input another username Username does not exist 1. System displays error message and prompt user to input another username Already your friend 1. System displays error message and prompt user to input another username
Exceptions:	
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	UC-CO-1		
Use Case Name:	Manage Course		
Created By:	Chua Qin Di	Last Updated By:	Chua Qin Di
Date Created:	3 Feb 2025	Date Last Updated:	25 March 2025

Actor:	User
Description:	**Manage Course** allows users to add, remove or view their current courses into the database.
Preconditions:	
Postconditions:	
Priority:	Medium
Frequency of Use:	Occasional
Flow of Events:	1. User selects Add Course, View Course or Remove Course a. Add Course allows users to add their courses into their course list by inputting course ID b. View Course allows users to view the information of the selected course c. Remove Course allows users to remove the course from their course list
Alternative Flows:	
Exceptions:	
Includes:	Add Course, View Course, View Friend's Course, Remove Course
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	UC-TS-1		
Use Case Name:	Manage Task		
Created By:	Chua Qin Di	Last Updated By:	Chua Qin Di
Date Created:	3 Feb 2025	Date Last Updated:	25 March 2025

Actor:	User
Description:	**Manage Task** allows users to add, view or remove their tasks on the database
Preconditions:	
Postconditions:	
Priority:	High
Frequency of Use:	Frequent
Flow of Events:	1. User selects Add Task, View Task or Remove Task a. Add Task allows users to add tasks to complete onto the their task list by inputting the description of the task and the deadline b. View Task allows users to view the information of the selected task c. Remove Task allows users to remove the task from the task list
Alternative Flows:	1. User can mark task as completed
Exceptions:	
Includes:	Add Task, View Task, View Friend's Task, Remove Task
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	UC-TB-1		
Use Case Name:	View Timetable		
Created By:	Vona Tanisha	Last Updated By:	Vona Tanisha
Date Created:	3 Feb 2025	Date Last Updated:	25 March 2025

Actor:	User
Description:	**View Timetable** allows users to view their personal timetable and their Friends timetables
Preconditions:	Must be friends to view other users timetable
Postconditions:	
Priority:	High
Frequency of Use:	Frequent
Flow of Events:	1. User can Choose to view personal timetable 2. User can choose a Friends timetable to view
Alternative Flows:	1. If the user is not connected to the internet a. The system notifies the user that they will not be able to view a friend's timetable till internet connection is restored.
Exceptions:	
Includes:	View personal timetable, View Friends Timetable
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	UC-NT-1		
Use Case Name:	Notifications		
Created By:	Wu Yiqing	Last Updated By:	Wu Yiqing
Date Created:	3 Feb 2025	Date Last Updated:	25 March 2025

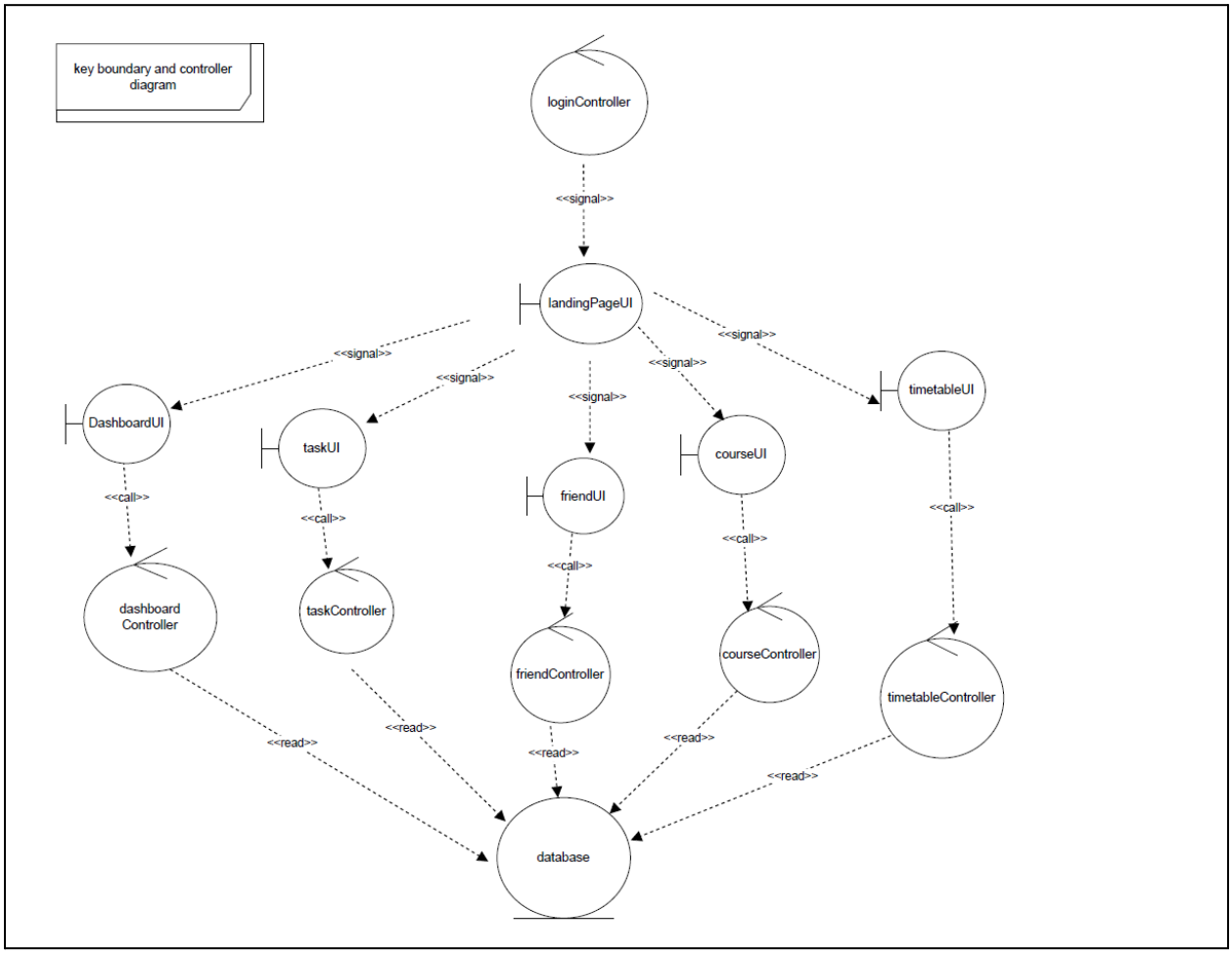
Actor:	User, Database
Description:	**Notifications** are alerts received by users when the system detects upcoming deadlines for tasks, and messages sent to the user from other users.
Preconditions:	1. Upcoming deadlines for task 2. Incoming friend requests 3. Friend notifies user
Postconditions:	
Priority:	High
Frequency of Use:	Frequent
Flow of Events:	1. System detects that a message is sent to the user 2. The app displays an alert on screen for the user with the information
Alternative Flows:	E1. User is not connected to internet 1. System delays notification until user is connected to the internet
Exceptions:	
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	UC-PG-1		
Use Case Name:	ProgressBar		
Created By:	Chua Qin Di	Last Updated By:	Chua Qin Di
Date Created:	3 Feb 2025	Date Last Updated:	25 March 2025

Actor:	User
Description:	**ProgressBar** is a graphical UI to show users the completion rate of their current tasks
Preconditions:	1. Have at least one current task
Postconditions:	
Priority:	High
Frequency of Use:	Frequent
Flow of Events:	1. The app displays a graphical progress bar to show the percentage of tasks to be completed
Alternative Flows:	No ongoing tasks 1. The app will display a message indicating that there are no tasks to be completed at the moment
Exceptions:	
Includes:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

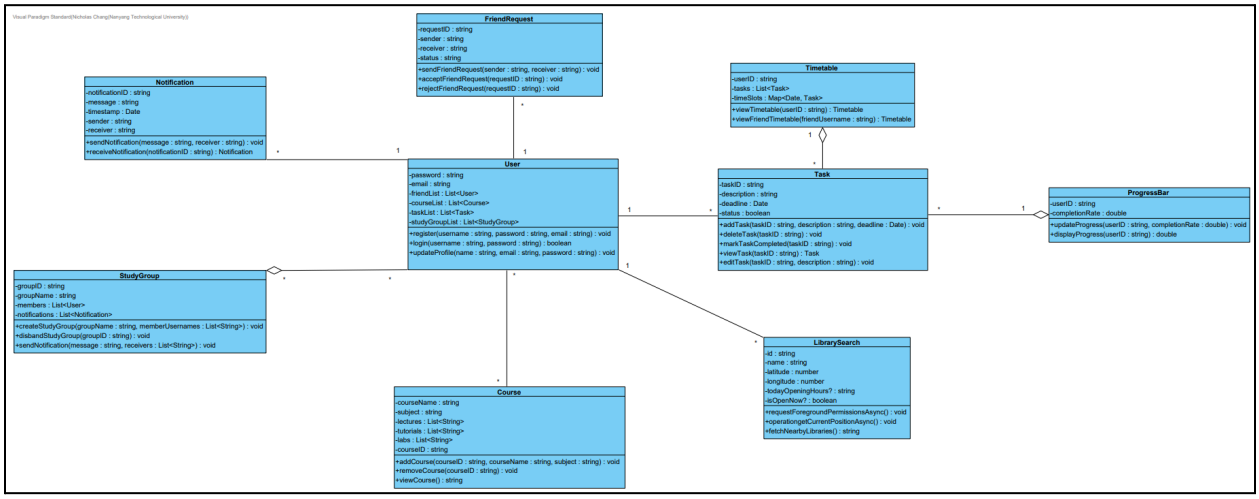
6.4 Key Boundary Classes & Control Classes

Refer to https://github.com/softwarelab3/2006-SCEC-E3/tree/main/lab5/Diagrams for a clearer image.



6.5 Class Diagram

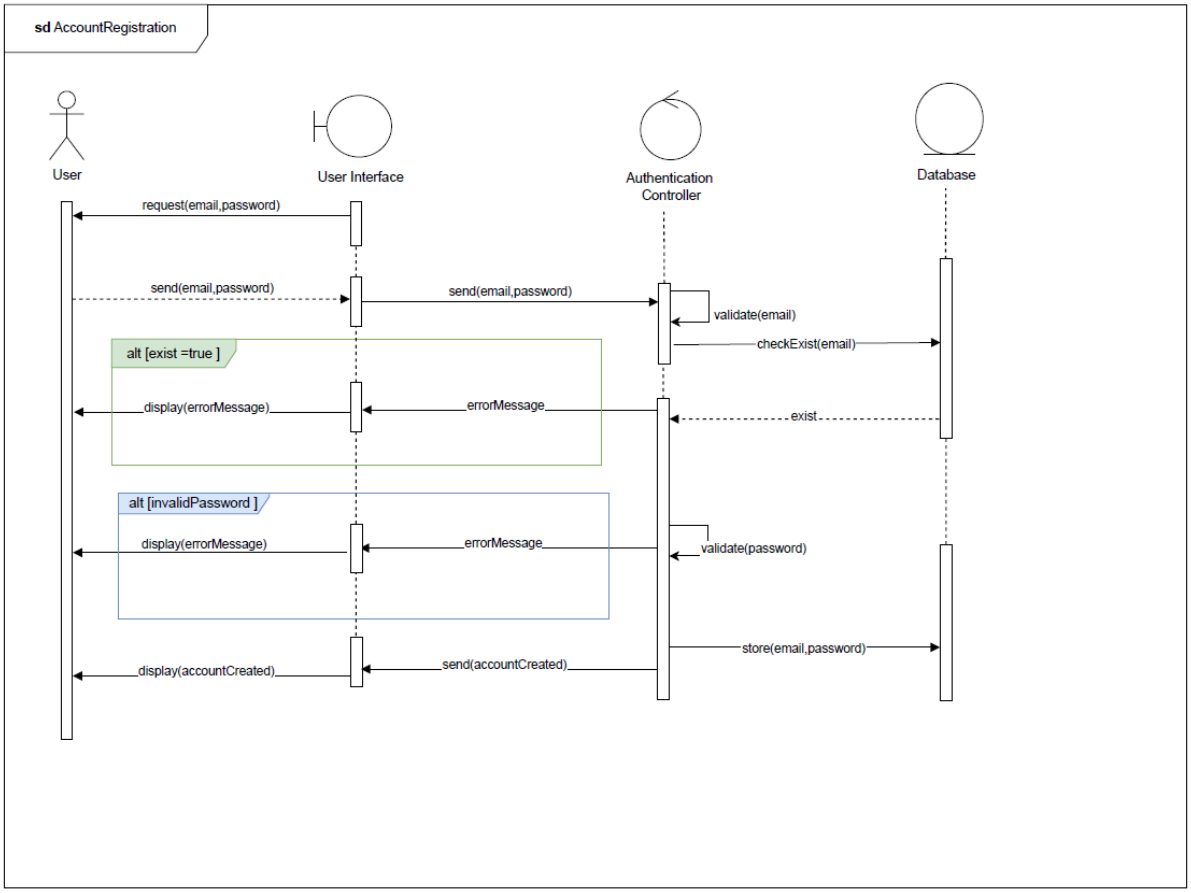
Please refer to https://github.com/softwarelab3/2006-SCEC-E3/tree/main/lab5/Diagrams for a clearer image.



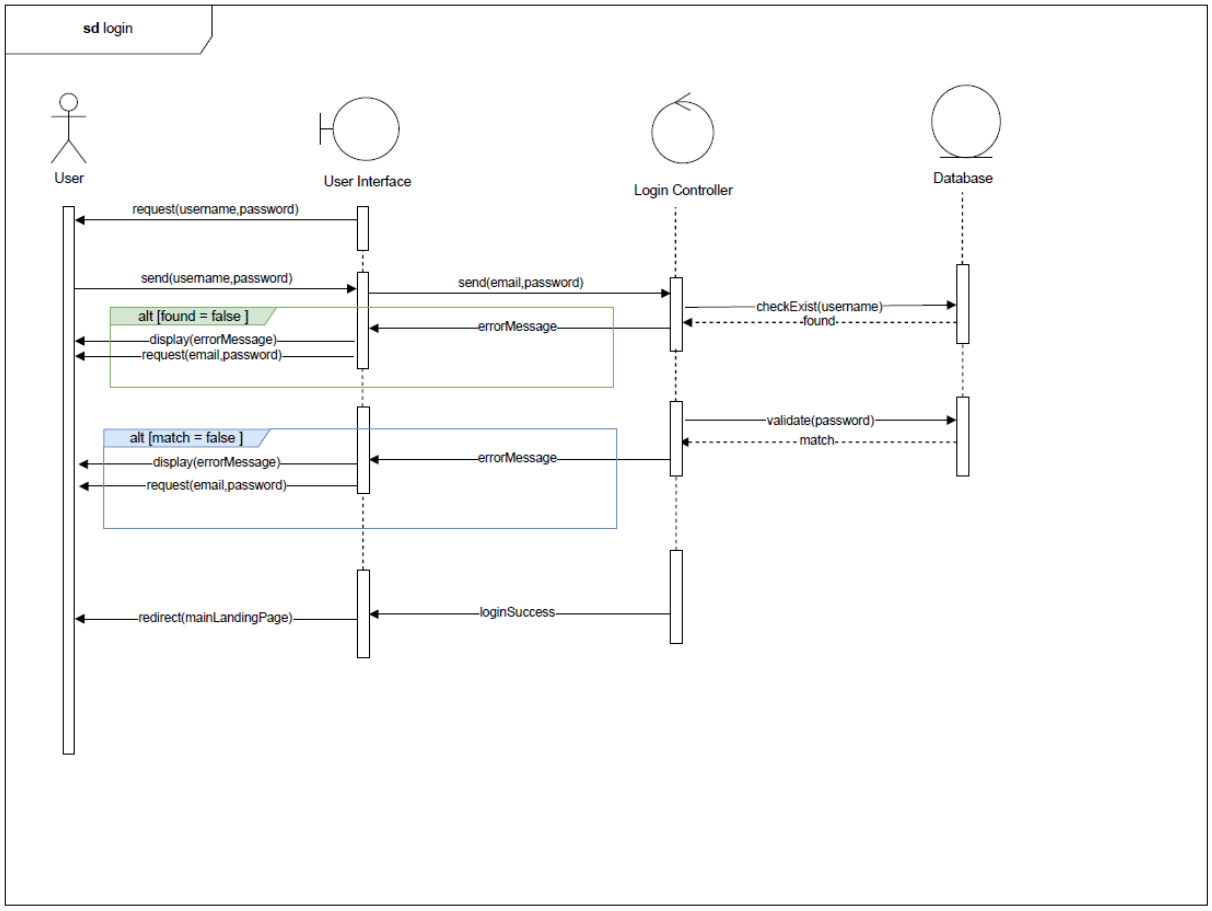
6.6 Sequence Diagrams

Please refer to https://github.com/softwarelab3/2006-SCEC-E3/tree/main/lab5/Diagrams for a clearer image.

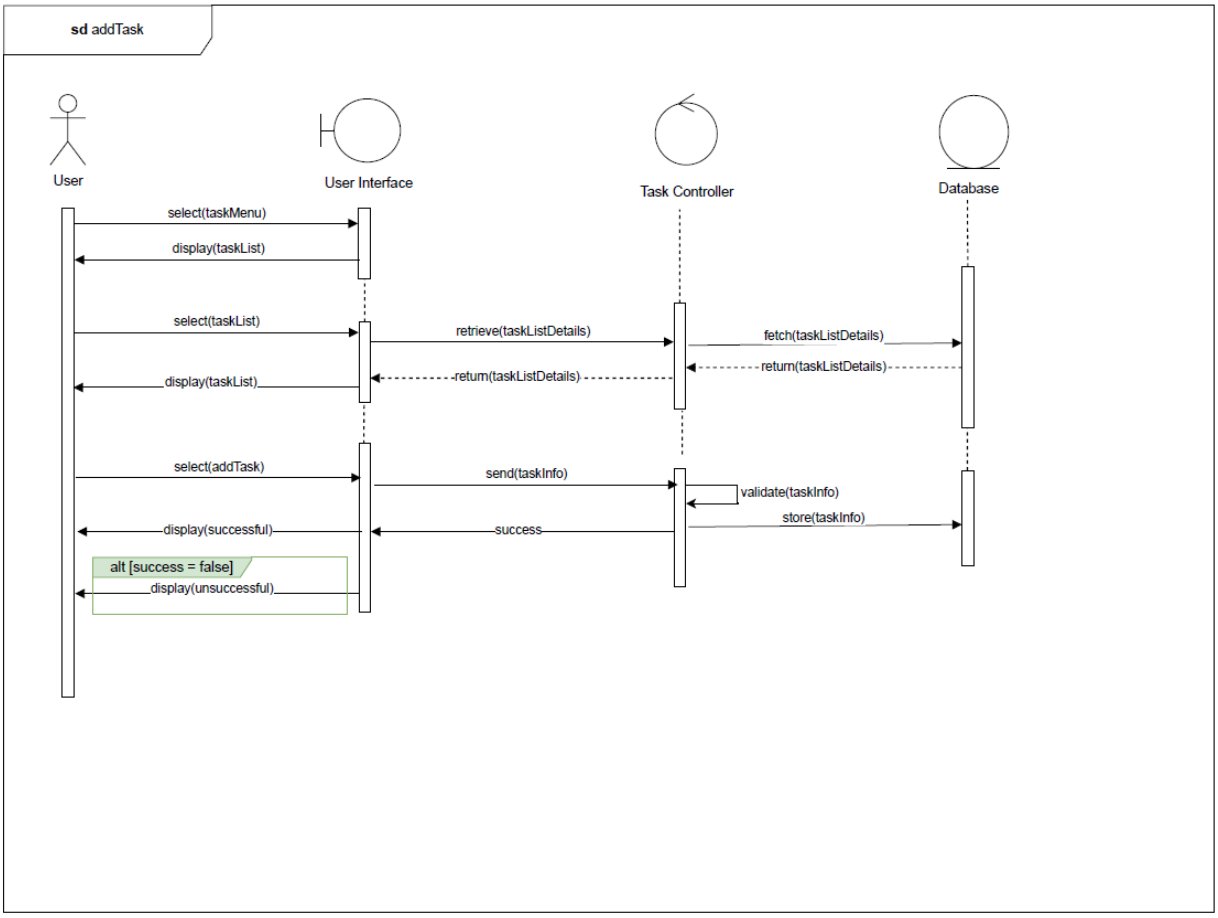
sd accountRegistration



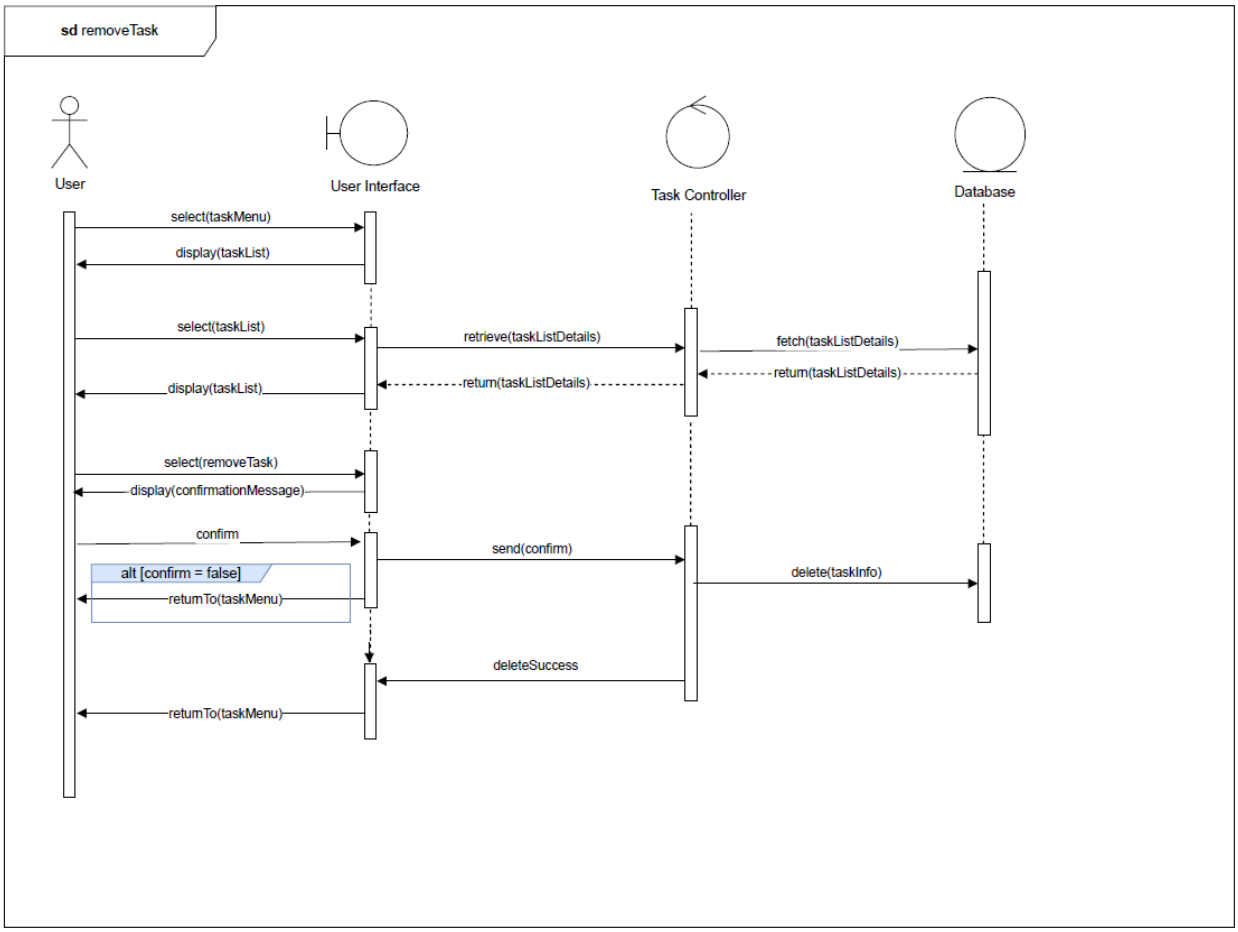
sd login


sd login
User
User Interface
Login Controller
Database
request(username,password)
send(username,password)
send(email,password)
checkExist(username)
found
alt [found = false]
errorMessage
display(errorMessage)
request(email,password)
validate(password)
match
alt [match = false]
errorMessage
display(errorMessage)
request(email,password)
redirect(mainLandingPage)
loginSuccess

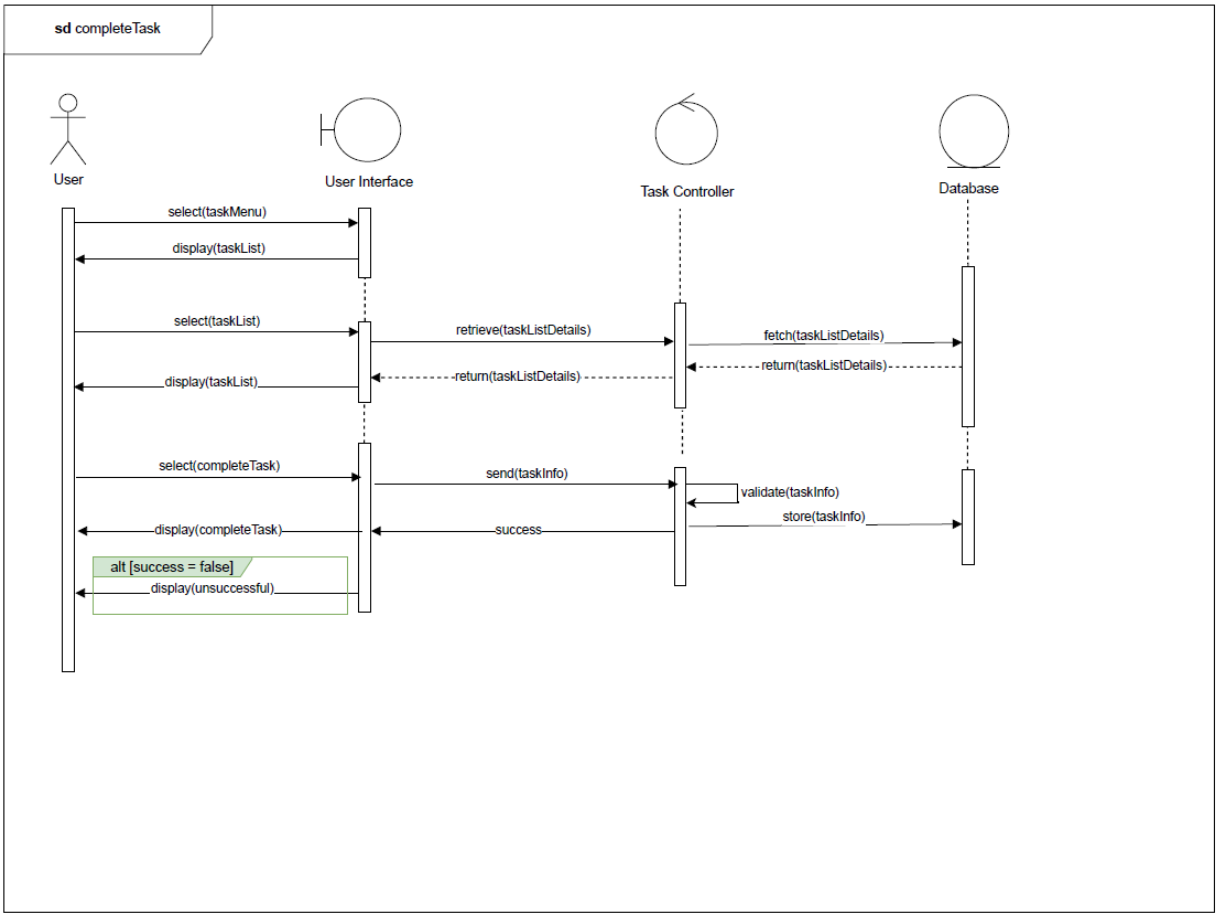
sd addTask


sd addTask
User
User Interface
Task Controller
Database
select(taskMenu)
display(taskList)
select(taskList)
retrieve(taskListDetails)
fetch(taskListDetails)
return(taskListDetails)
return(taskListDetails)
display(taskList)
select(addTask)
send(taskInfo)
validate(taskInfo)
store(taskInfo)
display(successful)
success
alt [success = false]
display(unsuccessful)

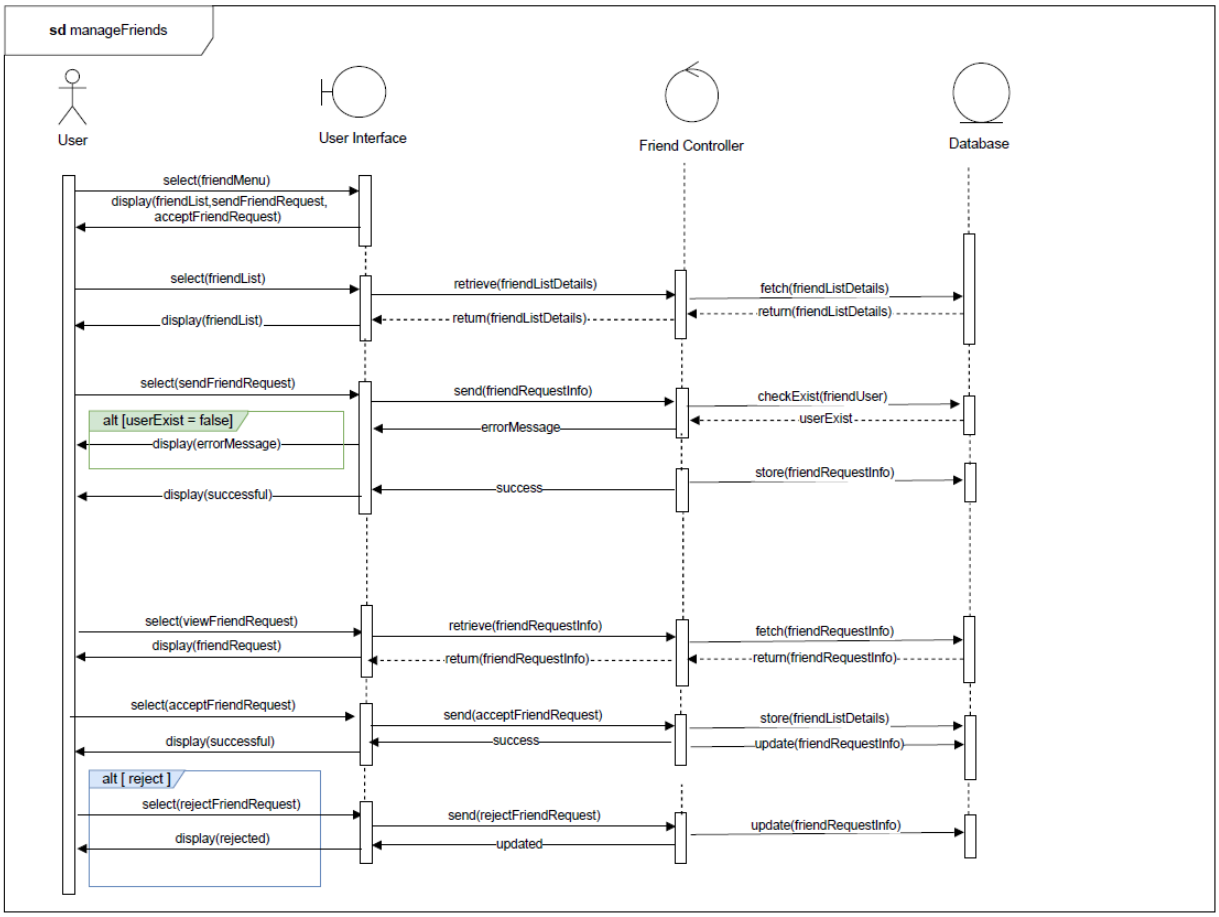
sd removeTask



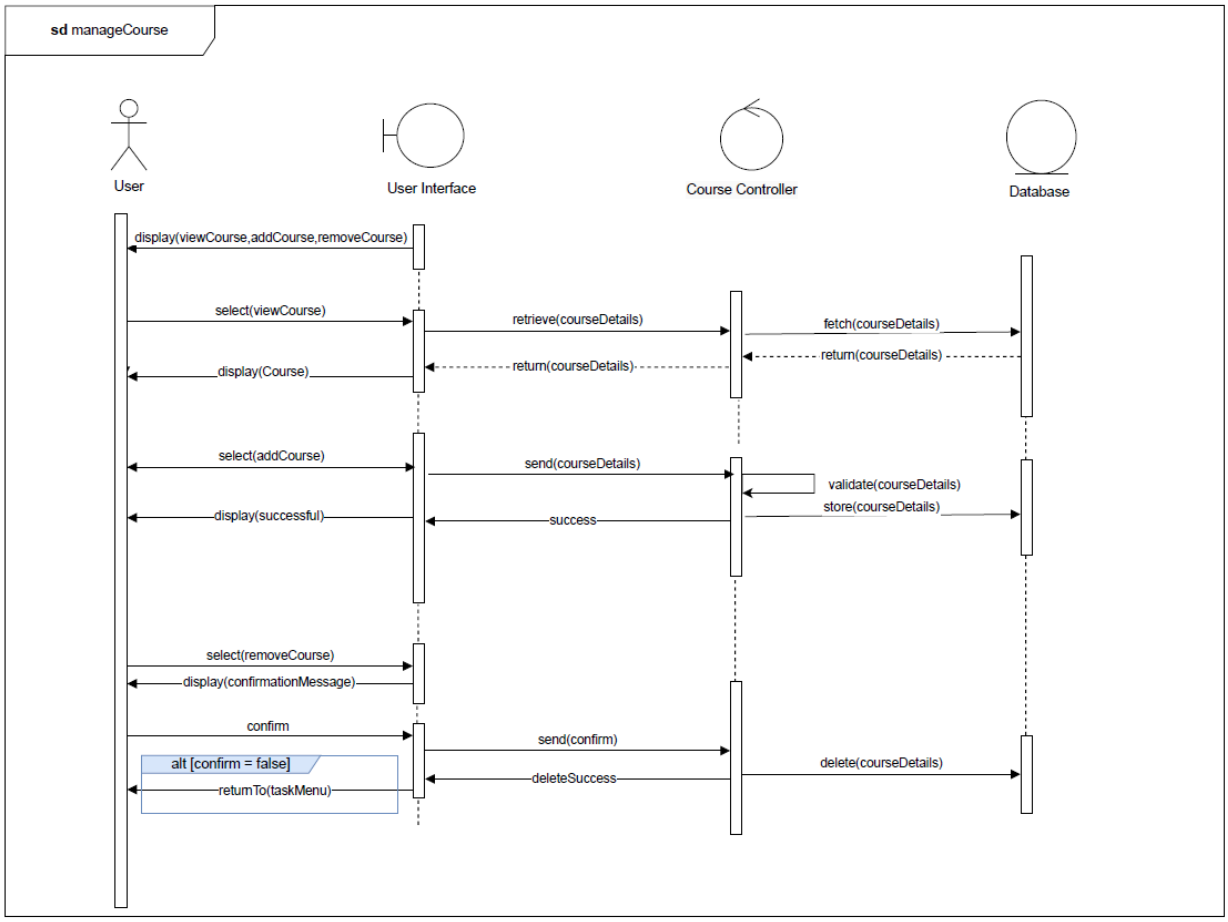
sd completeTask



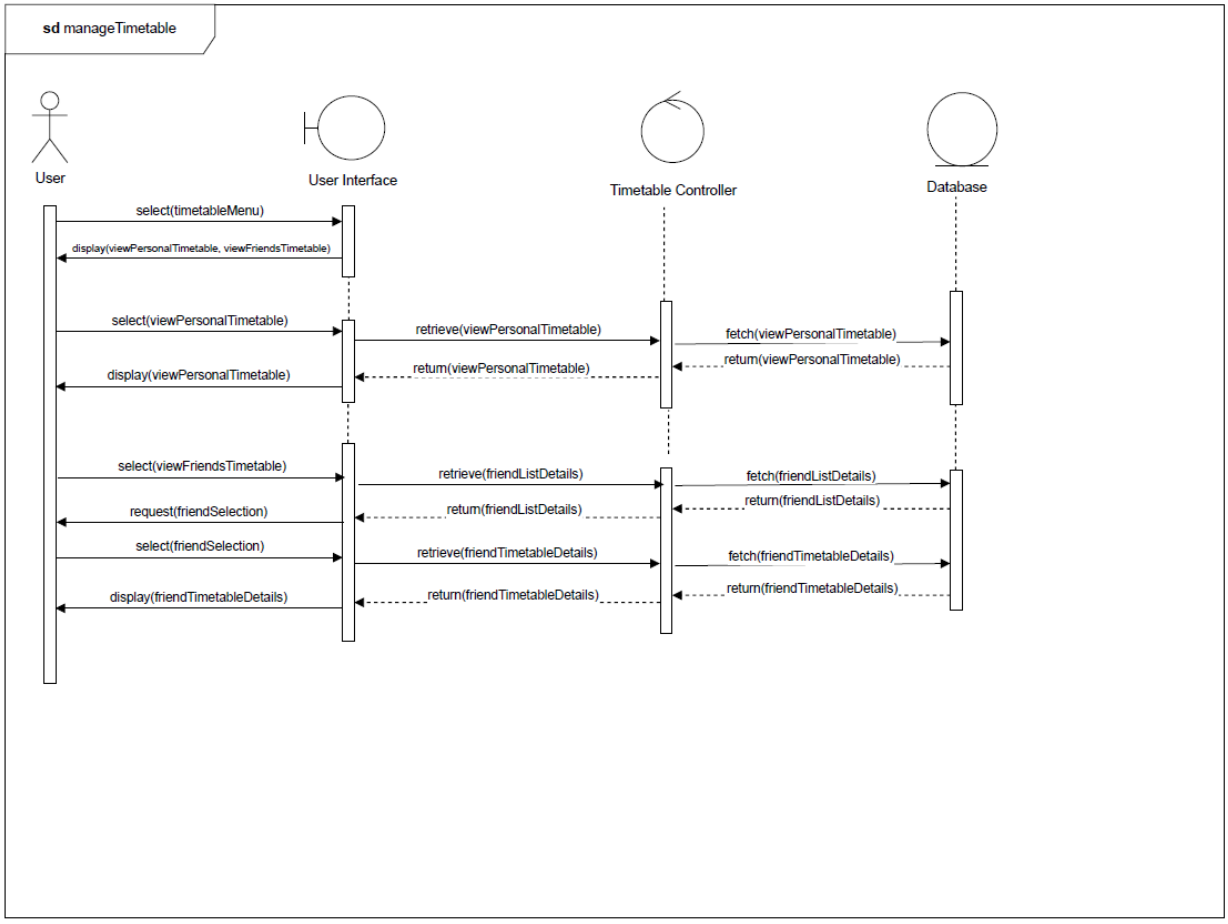
sd manageFriends



sd manageCourse

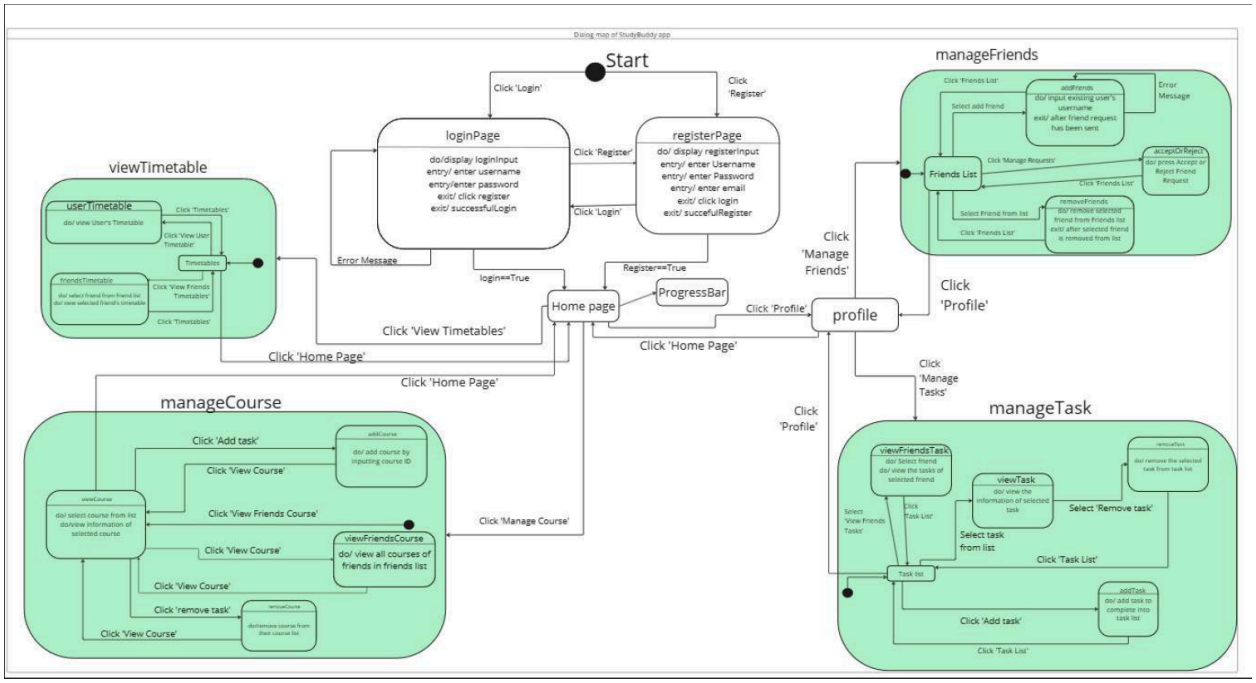


sd manageTimetable



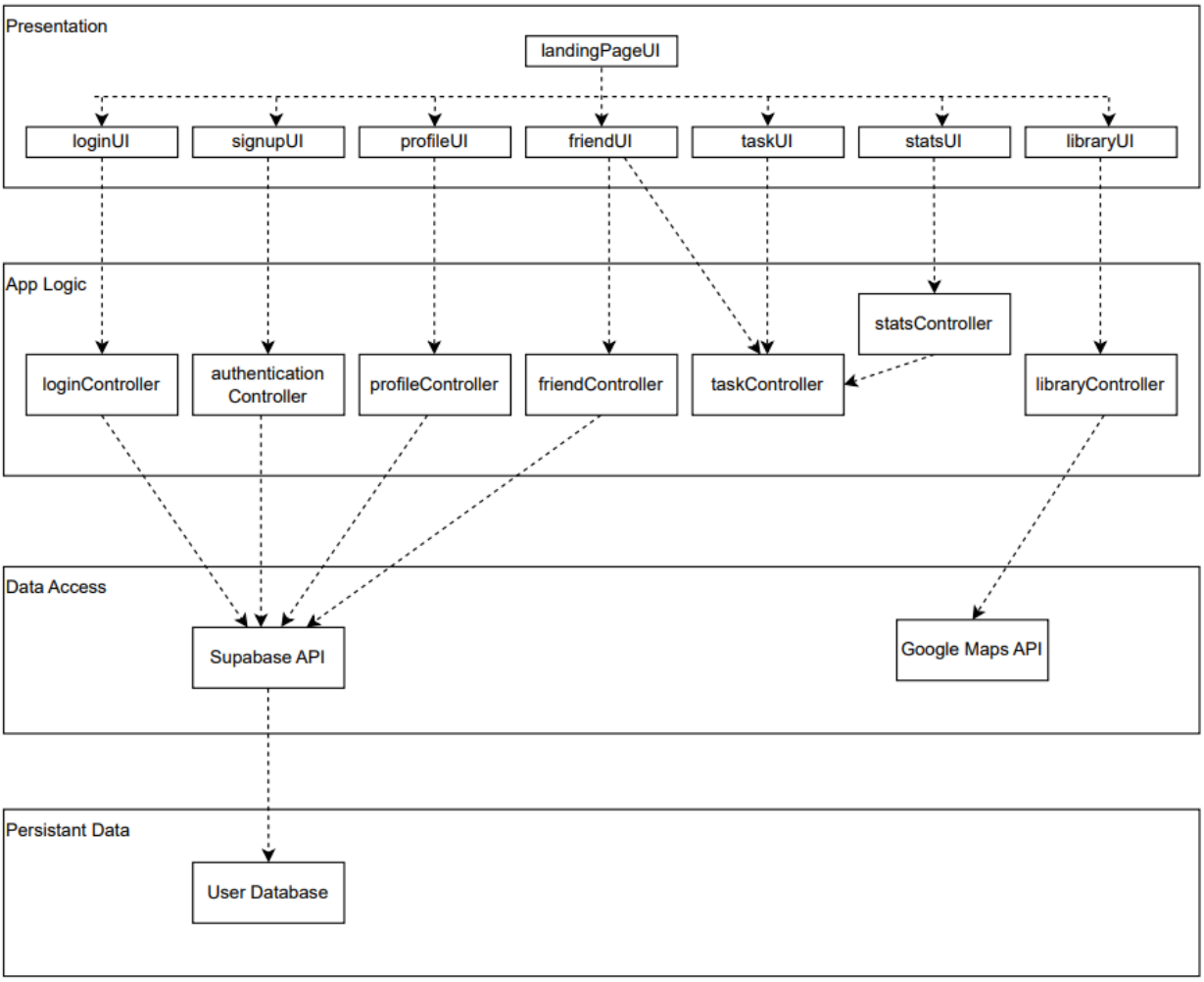
6.7 Dialog Map

Refer to https://github.com/softwarelab3/2006-SCEC-E3/tree/main/lab5/Diagrams for a clearer image.



6.8 System Architecture

Refer to https://github.com/softwarelab3/2006-SCEC-E3/tree/main/lab5/Diagrams for a clearer image.



6.9 Testing

6.9.1 Black Box Testing

Equivalence Class & Boundary Value Testing

Scenario 1: Account Sign Up

Equivalence Class for Email:

Valid EC	Valid Email address
Invalid EC	Empty field or invalid Email address

Equivalence Class for Password:

Valid EC	Strong password that meets requirement
Invalid EC	Empty field or password does not meet requirement

Test Case 1	Invalid account sign up		
Test Scenario	Account sign up	**Test Case ID**	T-SU-1
Test Description	Sign up with invalid credentials	**Test Priority**	High
Prerequisite	Stable Internet Connection	**Post-Requisite**	N/A

No.	Email	Password	Expected Output	Actual Output	Test Result
1	[empty]	abc	Sign up failed: Unable to validate email address: invalid format	Sign up failed: Unable to validate email address: invalid format	Pass
2	example@gmail.com	[empty]	Sign up failed: Signup requires a valid password	Sign up failed: Signup requires a valid password	Pass

3	a@gmail.com	abc	Sign up failed: Email address "a@gmail.com" is invalid	Sign up failed: Email address "a@gmail.com" is invalid	Pass
4	a@gmail.com	Password1234!	Sign up failed: Email address "a@gmail.com" is invalid	Sign up failed: Email address "a@gmail.com" is invalid	Pass
5	chuaqindi02@gmail.com	abc	Sign up failed: Password should be at least 6 characters.	Sign up failed: Password should be at least 6 characters.	Pass

Test Case 2	Sign up successful		
Test Scenario	Account sign up	**Test Case ID**	T-SU-2
Test Description	Sign up invalid credentials	**Test Priority**	High
Prerequisite	Stable Internet Connection	**Post-Requisite**	N/A

No.	Email	Password	Expected Output	Actual Output	Test Result
1	Any valid email. Eg. chuaqindi02@gmail.com	Password123!	Check your email to confirm sign up!	Check your email to confirm sign up!	Pass

Scenario 2: Account Login

Equivalence Class for Email:

Valid EC	Existing Email address in database
Invalid EC	Empty field or Email address does not exist in database

Equivalence Class for Password:

Valid EC	Correct password associated with email
Invalid EC	Empty field or incorrect password

Test Case 1	Invalid Login		
Test Scenario	Account Login	**Test Case ID**	T-LO-1
Test Description	Login with invalid credentials	**Test Priority**	High
Prerequisite	Stable Internet Connection	**Post-Requisite**	N/A

No.	Email	Password	Expected Output	Actual Output	Test Result
1	[empty]	abc	Login failed: missing email or phone	Login failed: missing email or phone	Pass
2	chuaqindi02@gmail.com	[empty]	Login failed: Invalid login credentials	Login failed: Invalid login credentials	Pass
3	a@gmail.com	abc	Login failed: Invalid login credentials	Login failed: Invalid login credentials	Pass
4	a@gmail.com	Password1234!	Login failed: Invalid login credentials	Login failed: Invalid login credentials	Pass
5	chuaqindi02@gmail.com	abc	Login failed: Invalid login credentials	Login failed: Invalid login credentials	Pass

6	chuaqindi02@gmail.com	IncorrectPassword1234!@	Login failed: Invalid login credentials	Login failed: Invalid login credentials	Pass

Test Case 2	Login Successful		
Test Scenario	Account Login	**Test Case ID**	T-LO-2
Test Description	Login with valid credentials	**Test Priority**	High
Prerequisite	Stable Internet Connection	**Post-Requisite**	N/A

No.	Email	Password	Expected Output	Actual Output	Test Result
1	Any existing email. eg. chuaqindi02@gmail.com	Password1234! (assuming correct password)	Login successful	Login successful	Pass

Scenario 3: Add Task

Equivalence Class for Task:

Valid EC	Non-empty field
Invalid EC	Empty field

Test Case 1

Test Case	Invalid Task		
Test Scenario	Add Task	**Test Case ID**	T-TA-1
Test Description	Add invalid task	**Test Priority**	Medium
Prerequisite	Stable Internet Connection	**Post-Requisite**	N/A

No.	Task input	Expected Output	Actual Output	Test Result
1	[empty]	Failed: No Input	Failed: No Input	Pass

Test Case 2

Test Case	Valid Task		
Test Scenario	Add Task	**Test Case ID**	T-TA-2
Test Description	Add valid task	**Test Priority**	Medium
Prerequisite	Stable Internet Connection	**Post-Requisite**	N/A

No.	Task input	Expected Output	Actual Output	Test Result
1	Any Non empty Field: Eg. Do Homework	Task added successfully	Task added successfully	Pass

Test Case 3

Test Case	Task Completed		
Test Scenario	Edit Task	**Test Case ID**	T-TA-3
Test Description	Task Completed	**Test Priority**	Medium
Prerequisite	Stable Internet Connection	**Post-Requisite**	N/A

No.	Task input	Expected Output	Actual Output	Test Result
1	Click the checkbox on the task description to indicate “Completed”	Task completed successfully	Task completed successfully	Pass

Test Case 4

Test Case	Delete Task		
Test Scenario	Edit Task	**Test Case ID**	T-TA-4
Test Description	Delete Task	**Test Priority**	Medium
Prerequisite	Stable Internet Connection	**Post-Requisite**	N/A

No.	Task input	Expected Output	Actual Output	Test Result
1	Click the “trash bin” icon on the task description to delete the task	Task deleted successfully	Task deleted successfully	Pass

Scenario 4: Add Friend

Equivalence Class for Friend's email:

Valid EC	Existing and verified email in database
Invalid EC	Email does not exist in database or email not verified

Test Case 1

Test Case	Empty Friend's Email		
Test Scenario	Add Friend	**Test Case ID**	T-FR-1
Test Description	Empty Friend's Email	**Test Priority**	Medium
Prerequisite	Stable Internet Connection	**Post-Requisite**	N/A

No.	Friend email input	Expected Output	Actual Output	Test Result
1	[empty]	Error Message: "Please enter an email address"	Error Message: "Please enter an email address"	Pass

Test Case 2

Test Case	Invalid Friend's Email		
Test Scenario	Add Friend	**Test Case ID**	T-FR-2
Test Description	Add invalid Friend email	**Test Priority**	Medium
Prerequisite	Stable Internet Connection	**Post-Requisite**	N/A

No.	Friend email input	Expected Output	Actual Output	Test Result
1	abc@gmail.com	Error Message: User not found!	Error Message: User not found!	Pass

Test Case 3

Test Case	Add existing Friend's email		
Test Scenario	Add Friend	**Test Case ID**	T-FR-3
Test Description	Add existing Friend's email	**Test Priority**	Medium
Prerequisite	Stable Internet Connection	**Post-Requisite**	N/A

No.	Friend email input	Expected Output	Actual Output	Test Result
1	Input existing Friend's email	Error Message: "You are already friends."	Error Message: "You are already friends."	Pass

Scenario 5: Search Nearby Library

Equivalence Class for Library:

Valid EC	Valid Library that exists on Google Maps API
Invalid EC	Library that does not exist on Google Maps API

Equivalence Class for Postal Code:

Valid EC	Valid Singapore Postal Code
Invalid EC	Invalid Singapore Postal Code. i.e Contains alphabets or number of digits not equal 6

Test Case 1

Test Case	Invalid Postal Code		
Test Scenario	Search Nearby Library	**Test Case ID**	T-SE-1
Test Description	Invalid Postal Code	**Test Priority**	Medium
Prerequisite	Stable Internet Connection	**Post-Requisite**	N/A

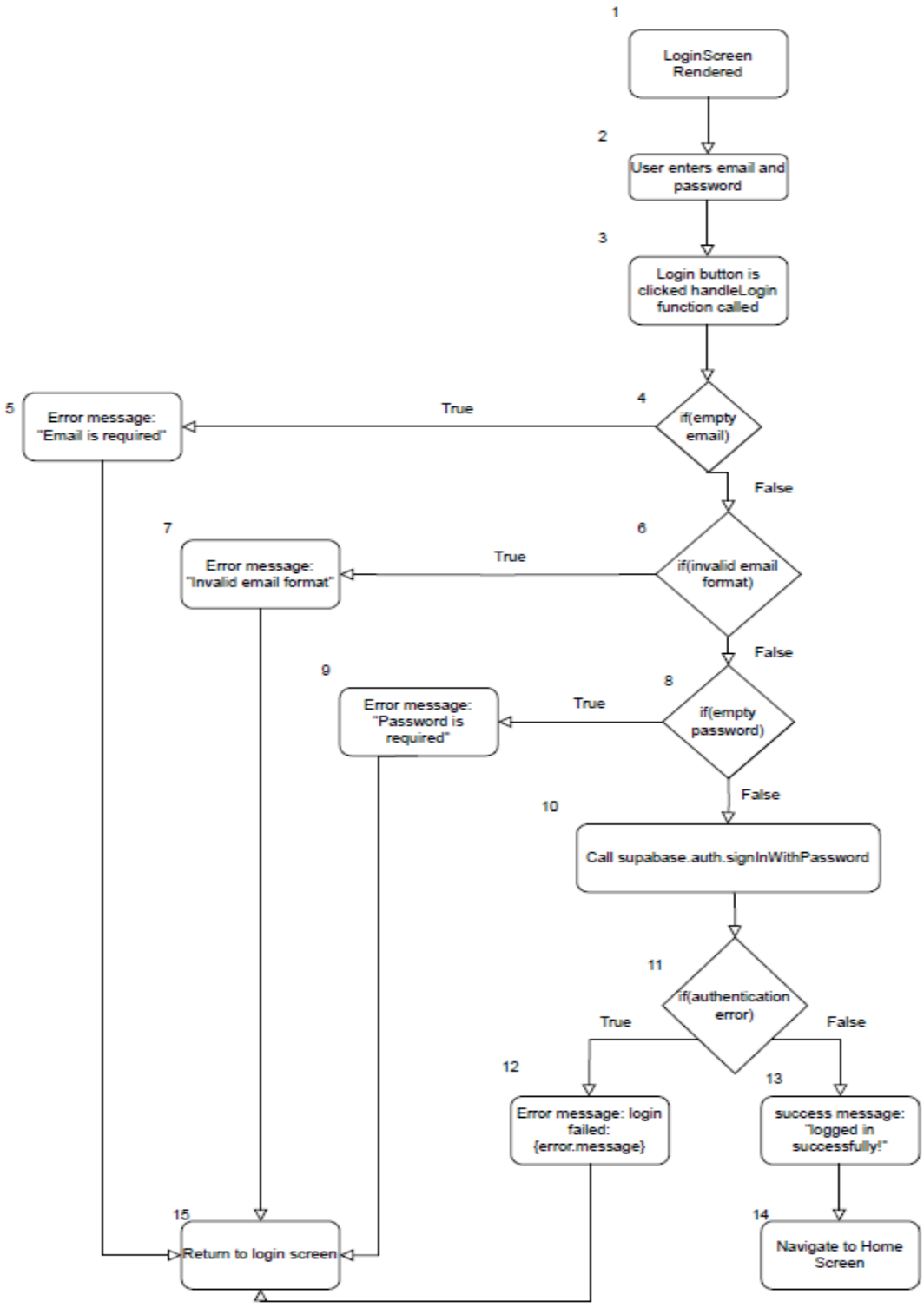
No.	Postal Code input	Expected Output	Actual Output	Test Result
1	Any invalid postal code	Returns default library location: “National Library / Lee Kong Chian Reference Library”	Returns default library location: “National Library / Lee Kong Chian Reference Library”	Pass

Test Case 2

Test Case	Empty Postal Code		
Test Scenario	Search Nearby Library	**Test Case ID**	T-SE-2
Test Description	Empty Postal Code	**Test Priority**	Medium
Prerequisite	Stable Internet Connection	**Post-Requisite**	N/A

No.	Postal Code input	Expected Output	Actual Output	Test Result
1	[empty field]	Error Message: “Please enter a postal code.”	Error Message: “Please enter a postal code.”	Pass

6.9.2 White Box Testing



Basic Path Testing

Table of Basis Paths

No	Basis Path	Path Description
1.	1, 2, 3, 4, 5, 15	Empty Email
2.	1, 2, 3, 4, 6, 7, 15	Invalid Email Format
3.	1, 2, 3, 4, 6, 8, 9, 15	Empty Password
4.	1, 2, 3, 4, 6, 8, 10, 11, 12, 15	Authentication Error
5.	1, 2, 3, 4, 6, 8, 10, 11, 13, 14	Login Successful

Test Cases & Results

Test Input	Expected Output	Actual Output
email = "example@gmail.com” password = "123123123"	Navigation to Main screen	Navigation to Main screen
email = "" password = "123123123”	“Login failed: missing email or password”	“Login failed: missing email or password”
email = "example.com" password = "123123123"	“Login failed: Invalid login credentials”	“Login failed: Invalid login credentials”
email = "example@gmail.com” password = ""	“Login failed: Invalid login credentials”	“Login failed: Invalid login credentials”
email = "example@gmail.com" password = "321"	“Login failed: Invalid login credentials”	“Login failed: Invalid login credentials”

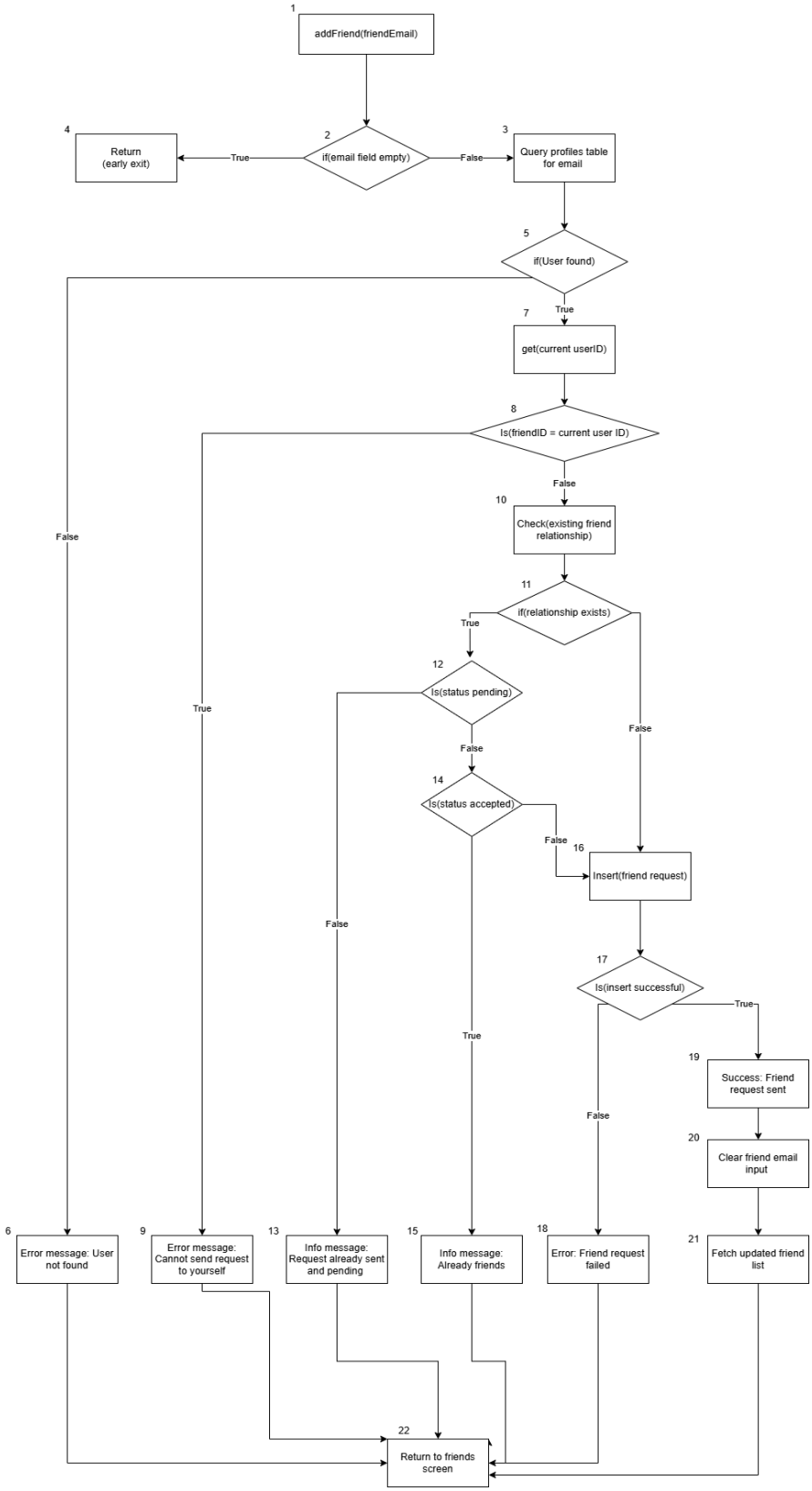

1
addFriend(friendEmail)
2
if(email field empty)
True
4
Return
(early exit)
False
3
Query profiles table for email
5
if(User found)
False
6
Error message: User not found
True
7
get(current userID)
8
Is(friendID = current user ID)
True
9
Error message: Cannot send request to yourself
False
10
Check(existing friend relationship)
11
if(relationship exists)
True
12
Is(status pending)
True
13
Info message: Request already sent and pending
False
14
Is(status accepted)
True
15
Info message: Already friends
False
16
Insert(friend request)
17
Is(insert successful)
False
18
Error: Friend request failed
True
19
Success: Friend request sent
20
Clear friend email input
21
Fetch updated friend list
22
Return to friends screen

Table of Basis Paths

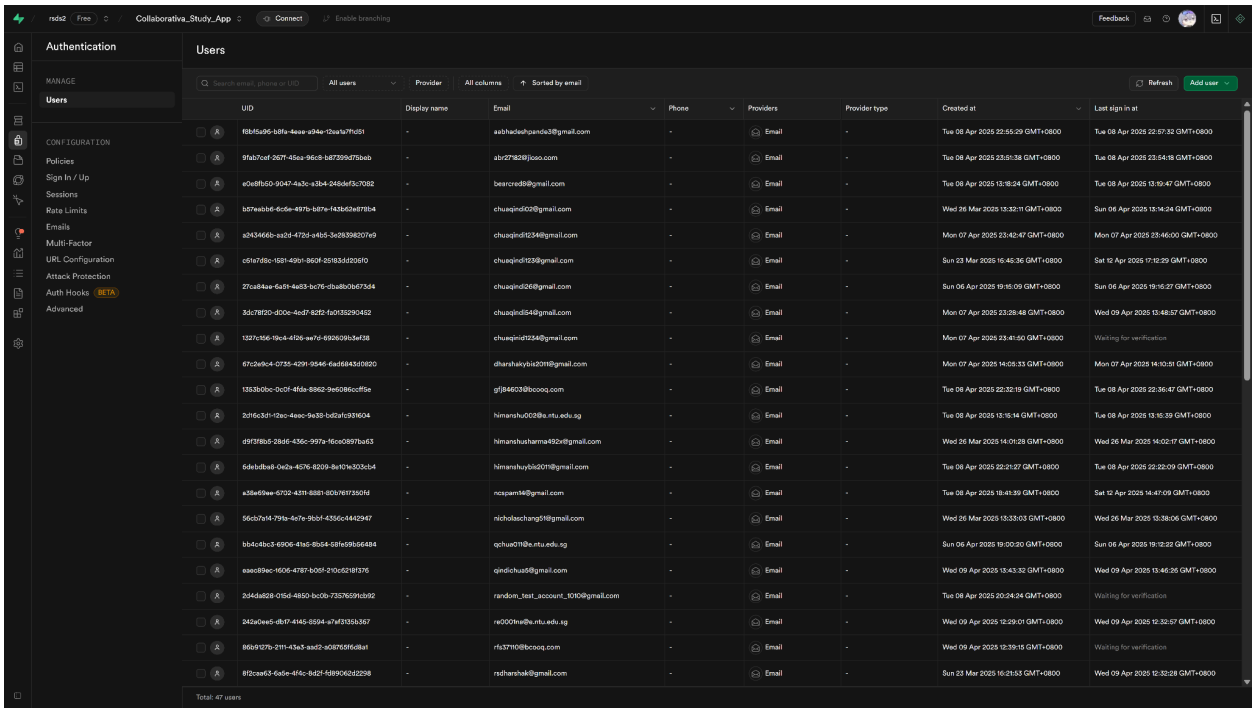
No	Basis Path	Path Description
1.	1, 2, 4	Empty Email field
2.	1, 2, 3, 5, 6, 22	User not found
3.	1, 2, 3, 5, 7, 8, 9, 22	Send request to self
4.	1, 2, 3, 5, 7, 8,10, 11,12, 13, 22	Request already pending
5.	1, 2, 3, 5, 7, 8, 10, 11, 12, 14, 15, 22	Already friends
6.	1, 2, 3, 5, 7, 8, 10, 11, 12, 14, 16, 17, 18, 22	Insert failed
7.	1, 2, 3, 5, 7, 8, 10, 11, 12, 14, 16, 17, 19, 20, 21, 22	Request sent successfully

Test case 1

Test Input	Expected output	Actual output
Email: cys@gmail.com (not existent user)	Error message: User not found	Error message: User not found
Email : ncspam14@gmail.com (before accepting friend request)	Info Message: Friend request already sent and pending.	Info Message: Friend request already sent and pending.
Email : ncspam14@gmail.com (after accepting friend request)	Info Message : You are already friends.	Info Message : You are already friends.

6.10 Database

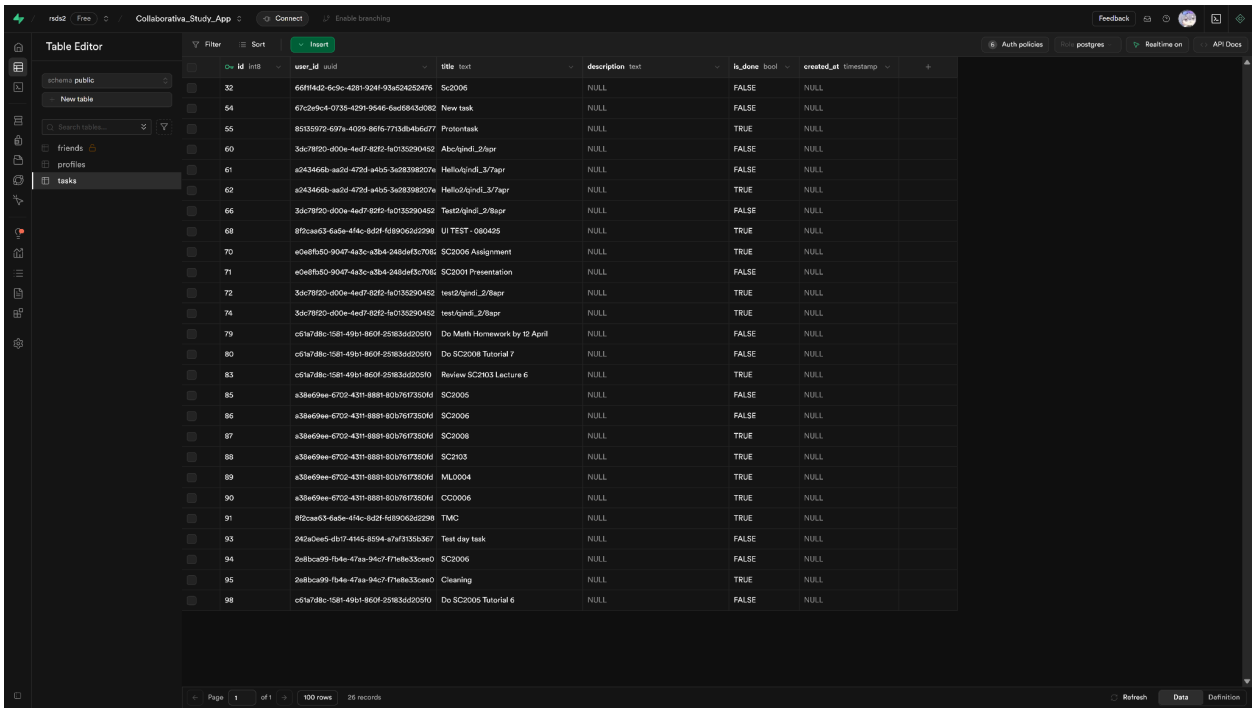
6.10.1 User Authentication



The application uses Supabase Authentication to manage user identity, access, and login sessions. This module is essential to ensure secure, role-based interaction with Study Buddy's core features such as task creation, progress tracking, and peer collaboration.

Authentication is email-based and powered by JWT (JSON Web Tokens). Supabase provides built-in functionality for user registration, login, password reset, and session management through its hosted authentication service.

6.10.2 Task Management



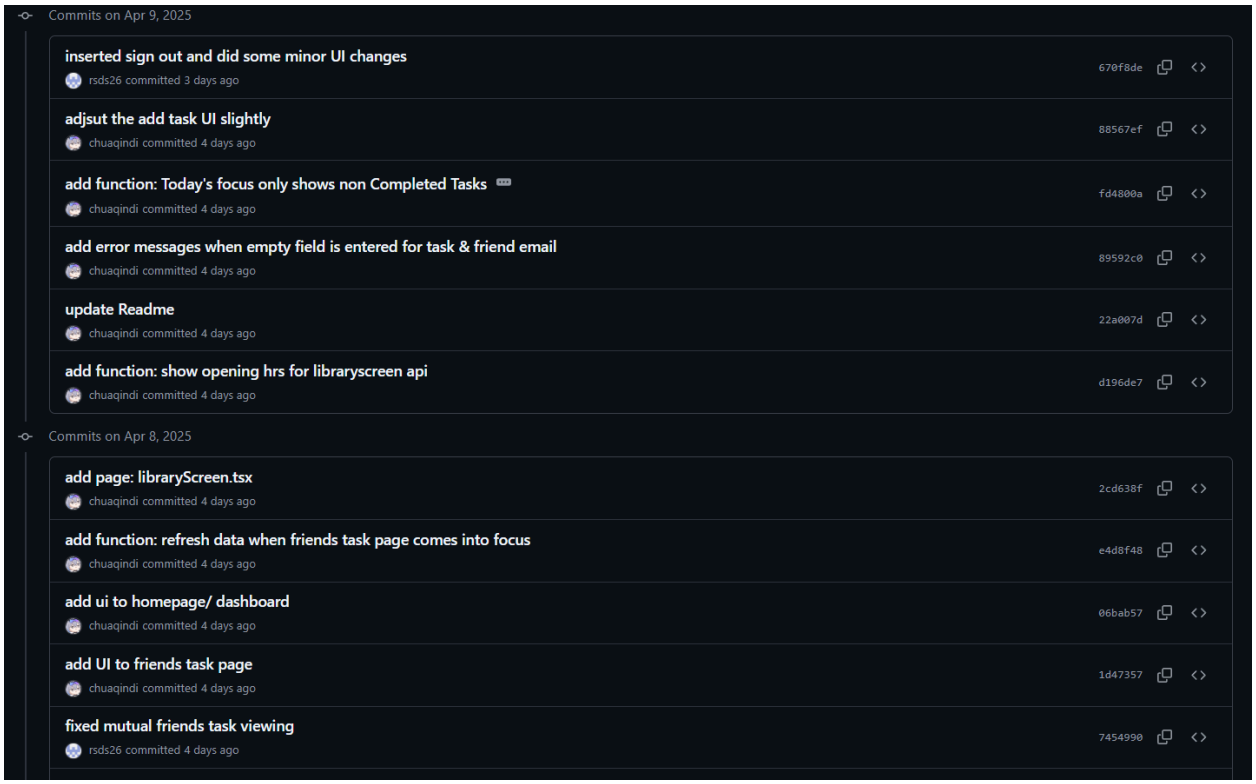
The tasks table is a core component of the application's data model, responsible for storing all task-related information created by users. Each row in the table represents a single task, uniquely identified by an auto-incremented ID. Tasks are linked to specific users through a foreign key (user_id), allowing the system to retrieve and manage personalized task lists.

This table supports key functionalities such as task creation, completion tracking, collaboration, and performance analytics. It is designed to scale efficiently and support real-time updates across devices, ensuring smooth and responsive user experiences.

6.11 Software Engineering Practices

6.11.1 Documentation

1. GitHub served as the central platform for version control, issue tracking, and project documentation
2. All code contributions were managed through pull requests which included descriptive commit messages
3. **README:** Our main repository README provides an overview of the project, setup instructions, and links to relevant resources.
4. **Pull Requests:** All code changes were pushed via pull requests, with descriptive commit messages.
5. We practiced peer reviews to ensure that every pull request was reviewed by at least one other team member before merging, in order to maintain high code quality and shared understanding.



Github commit history for version control and documentation

6.11.2 Scrum

To ensure timely delivery and maintain steady progress, the team adopted the **Scrum methodology**, a flexible Agile framework suitable for iterative development.

- **Sprint Structure**:
  The project was divided into weekly sprints, each with clear objectives and defined deliverables. This allowed the team to continuously deliver incremental progress and adapt to changing priorities.

- **Sprint Planning**:
  At the start of each sprint, the team conducted a sprint planning session to:

  - Define sprint goals
  - Break down high-level features into manageable development tasks
  - Estimate task difficulty and assign priorities

- **Task Tracking**:
  A sprint backlog was maintained to monitor task progress. Tasks were categorized under three columns:

  - **To Do**: Tasks not yet started
  - **In Progress**: Tasks actively being worked on
  - **Completed**: Finished and verified tasks

6.11.3 Good Coding Practices

To ensure a high-quality, maintainable, and collaborative codebase, the development team adhered to a set of consistent coding standards and practices throughout the Study Buddy project.

6.5.1 Simplicity

1. Code was written to be concise and purpose-driven.
2. Functions were kept short, with a single responsibility, avoiding unnecessary complexity or deeply nested logic.

6.5.2 Readability

1. The team followed consistent naming conventions for files, variables, and functions.
2. Indentation and file structure were standardized to improve navigation and onboarding for all developers.
3. Code was commented where necessary to explain logic or decisions, especially in shared modules.

6.5.3 Pair Programming

1. Developers regularly engaged in pair programming, promoting active knowledge sharing and collaborative debugging.
2. This practice enhanced code quality by enabling real-time code review, early issue detection, and mutual understanding of functionality.
3. It also reinforced shared ownership of code, minimizing silos and bottlenecks.

6.5.4 DEEP Method

The team applied the **DEEP method** for adaptive task management:

- **D – Detailed**: Tasks were clearly scoped and well-defined before implementation.
- **E – Emergent**: New tasks were identified and added iteratively as the project evolved.
- **E – Estimated**: Task complexity was estimated to facilitate sprint planning and workload balancing.
- **P – Prioritized**: Tasks were prioritized based on impact, urgency, and dependencies.

These practices supported the project's goals of code maintainability, developer efficiency, and teamwide accountability.