Eötvös Loránd University

Faculty of Informatics

Dept. of Software Technology

and

Methodology

StudyBuddy Frontend

Supervisor:
Dr. Rudolf Szendrei
Associate Professor

Author:
Aamna Tayyab
Computer Science BSc.

Budapest, 2023

EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

Thesis Registration Form

Attention! The deadline for amendment is 1 February in case of final exam in June and 31 August in case of final exam in January.

Student's Data:

Student's Name: Tayyab Aamna

Student's Neptun code: UOXP5R

Course Data:

Student's Major: Computer Science BSc

I have an internal supervisor

Internal Supervisor's Name: *Szendrei, Rudolf*

Supervisor's Home Institution: ***Department of Software Technology and Methodology***

Address of Supervisor's Home Institution: ***1117, Budapest, Pázmány Péter sétány 1/C.***

Supervisor's Position and Degree: *Associate Professor, Computer Science*

Supervisor's Position and Degree:

Thesis Title: Study Buddy

Topic of the Thesis:

(Upon consulting with your supervisor, give a 150-300-word-long synopsis os your planned thesis.)

Year after year, students struggle to find people of similar subjects to study with. This problem was made worse by the COVID-19 pandemic which affected the social interactions and made many students unable to study with their peers.

My project will be to write the front-end of a cross platform application that allows people who are looking for other people to study with called Study Buddy to match people according to Subject/ school. Users can study with individuals or in groups. Users can create or join study groups in the app. They can discuss about different Education Topics with other users on different Topic Forums. The target of user demographics of this application are university students.

It will be cross-platform, the user interface will be written in a cross-platform framework to ensure that it works on multiple/popular platforms which is why the app will be written in cross platform programming language. I will also use web technologies the design of pages. This app is developed in teamwork with another student, Timilehin Bisola-Ojo who will work with the backend. We will have extended features of the application divided among the team members.

Automated tests will be written using a testing library (preferably in the language used to write majority of the application). I will do some manual testing of the different components and buttons to ensure their functionality works correctly and to ensure all pages are linked correctly.

Budapest, 2022. 12. 01.

Acknowledgements

Prima facie, I would like to express my deepest gratitude to Allah, my family, my friends and roommates who have contributed to the completion of my Bachelor's thesis in Computer Science. Their constant support and encouragement have been invaluable throughout this journey.

I would like to thank my thesis supervisor, Dr. Rudolf Szendrei, for his unwavering guidance. His expertise, patience and insightful feedback has been crucial to the entire process.

I extend my sincere thanks to the faculty members at the Faculty of Informatics at Eötvös Loránd University. Their dedication to teaching has encouraged me to be a better student.

I am thankful to my friends and fellow students who provided me with valuable insights during the course of my research. Their feedback and suggestions were instrumental in shaping my ideas and improving the overall quality of this work.

I had the pleasure of collaborating with Timilehin Bisola-Ojo. Working alongside him has been an enriching experience, and I am thankful for his dedication. Together, we have navigated challenges, brainstormed ideas, and supported each other, creating a dynamic and fruitful partnership.

Although it is not possible to name everyone who has contributed to this thesis individually, I am sincerely grateful for the support and assistance provided by all those involved.

Contents

Chapter 1

Introduction

1.1 Motivation

The COVID-19 pandemic has dramatically transformed the landscape of education, necessitating the adoption of distance learning as the new norm. As a result, students around the world faced the challenges of navigating online education without the traditional support system of physical classrooms and study groups. Recognizing the need for a solution to facilitate effective remote learning, the concept of Study Buddy was conceived.

Study Buddy, a mobile application developed as part of this Bachelor's thesis, aims to address the difficulties encountered by students in finding study groups and study partners in the digital learning environment. The idea originated during a business fundamentals class in 2020, where the task was to create an application, that would assist students during the pandemic.

Working collaboratively with another student from the same business fundamentals group, we envisioned Study Buddy as a platform, where students could connect with others studying similar courses or pursuing similar majors within their respective universities. By creating a profile that includes university, language, major, and selected courses, students can easily find study buddies who share common academic interests.

The underlying motivation for developing Study Buddy stemmed from the realization that study buddies play a pivotal role in the learning process, fostering

collaboration, knowledge exchange, and academic support. However, amidst the challenges of online education, finding suitable study partners became increasingly difficult for our cohort, which commenced their degree program during the pandemic.

The motivation to pursue Study Buddy as a Bachelor's thesis project further intensified after achieving recognition in an online UI hackathon, where the concept garnered third place. This acknowledgment bolstered our belief in the application's potential impact on enhancing the learning experience for students.

By formalizing the concept of Study Buddy within this thesis documentation, we aspire to delve deeper into the development process, including the design, implementation, and evaluation of the application. Through this endeavor, we aim to contribute to the broader discourse on digital education and empower students to establish meaningful connections and collaborative learning environments in the context of distance learning.

In summary, this thesis aims to explore the development of Study Buddy, a mobile application designed to alleviate the challenges of online education by facilitating the formation of study groups and partnerships. By fostering collaboration and knowledge-sharing, Study Buddy endeavors to enhance the overall learning experience for students navigating the world of digital education.

1.2 Collaboration

The Study Buddy application was developed as a collaborative effort between myself and my colleague, Timilehin Bisola-Ojo. As part of a team of two, we divided the responsibilities based on our individual skills and expertise to deliver a comprehensive solution for the challenges posed by remote learning.

In this two-pronged approach, my role primarily revolved around frontend development. I focused on building a user-centric interface, ensuring the application was intuitive and engaging for students navigating the digital learning environment.

Timilehin, on the other hand, was responsible for backend development. His tasks involved constructing the underlying systems and structures that ensured the app's performance and scalability. In addition to this, he developed an admin website, providing a crucial tool for managing the application and monitoring its usage.

This clear division of roles enabled us to effectively progress in our development process, with each contributing unique strengths to the project. The result of our collaboration is Study Buddy - a practical and user-friendly application designed to address the unique challenges of distance learning.

1.3 Thesis Structure

The thesis consists of 4 main chapters including introduction. In addition to these chapters, the thesis includes a bibliography, which lists all the relevant sources and references used throughout the project development, a list of figures, a list of codes and an appendix highlighting the work on backend development and development of the web app used for this application's administration, in a separate documentation.

Chapter 2: User Documentation
This chapter covers the installation process and provides step-by-step instructions on how to install the app. The aim of this chapter is to ensure that users can easily navigate and utilize the app.

Chapter 3: Developer Documentation
This chapter offers an in-depth exploration of the implementation structure.It provides developers with detailed insight into the architecture and design choices made during the development process, enabling a deeper understanding of the project's underlying technology.

Chapter 4: Conclusion
The concluding chapter serves as the summary of the entire project. It presents an overview of the project's future direction, identifying potential areas for further exploration and improvement.

By following this structure, the thesis aims to provide a comprehensive understanding of the application, its development process, and its potential for future growth and enhancement.

Chapter 2

User documentation

2.1 Project Description

StudyBuddy is a mobile application designed to facilitate partnerships among students in remote education, addressing challenges faced during the COVID-19 transition.

By creating a profile, users select their university, preferred languages, study major, and selected courses they want to have a study session about. Based on this information, users can connect with peers who share common academic interests. StudyBuddy serves as a virtual platform where students can arrange study sessions, engage in discussions and support each other academically.

Key Features:

- **Profile creation:** Users can create a personalized profile that includes uploading profile picture, adding a comprehensive bio, age, preferred languages, university, major and selected courses. Selecting language, university, major and courses is mandatory to make profile.
- **Discover and Connect:** Students can explore and connect with like-minded peers based on shared courses, majors, and university choices, facilitating one-on-one communication in a user-driven approach.
- **Peer Connections:** After becoming "buddies" students can then message each other, enabling direct communication for meeting, discussing academic queries, and fostering collaborative learning experiences.

2.2 System Requirements

StudyBuddy is an Android application built with Expo SDK Version 48.0.10 which means it supports Android Version 5+ (SDK version 21) for Android which means that all versions after 21 can compile the application. Recommended version is Android 10.0 or higher (API Level 29 or higher). Minimal storage space required for the APK is 32.54 MB. The application installation minimal storage is 72.29 MB.

2.3 Installation Guide

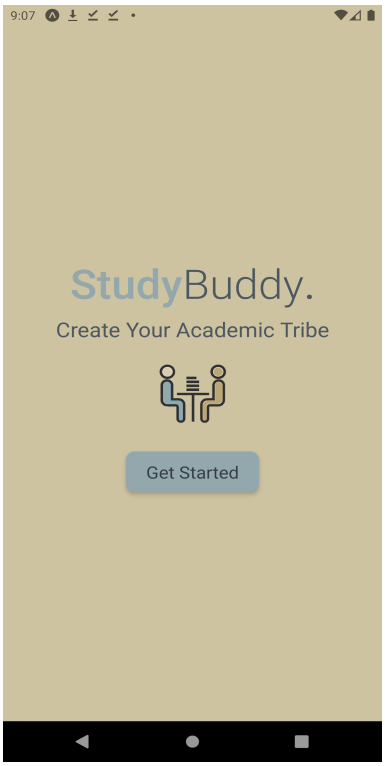
StudyBuddy Application is currently an Android application, but can be made cross platform as well. It is currently available as an APK (available here: StudyBuddy.apk). Ensure that your device meets the minimum specifications required for the APK. Check that your device is running a compatible version of the Android operating system and you have enough storage space as mentioned in Section 2.2. Find the downloaded APK file on your mobile and tap on it to initiate the installation process. During installation, you may encounter security warnings related to app permissions or compatibility. If you have Google Play Protect enabled, you may encounter problems during installation. You can follow the instructions on Google Account Help page to resolve any related issues. Once the installation is complete, locate the app in your device and tap on the app icon to launch the APK and start using it.

2.4 Basic Usage and User Interface

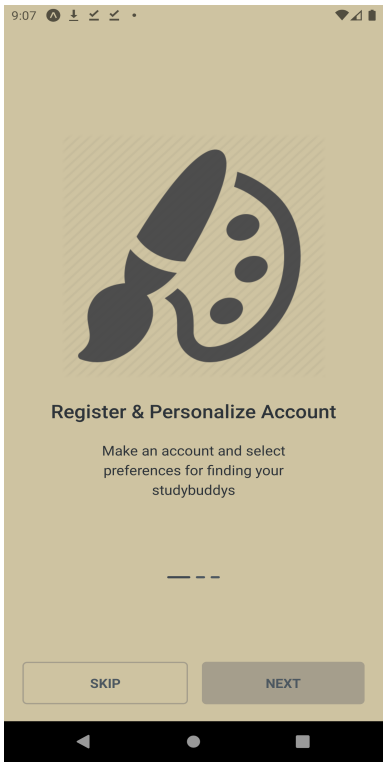
The app offers a seamless user experience with different screens, each serving a specific purpose. Below is a detailed description of the screens which mimic the workflow of the app.

2.4.1 Get Started

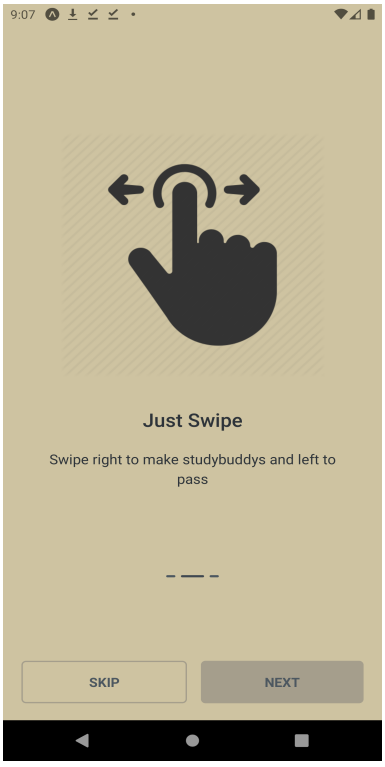
Upon downloading the app, users are greeted with the Get Started screen. By tapping the button, they proceed to an informative Onboarding slider, consisting of three screens that explain how to use the app. Users can skip or navigate through the screens using slide gestures or next buttons as shown in Figure 2.1



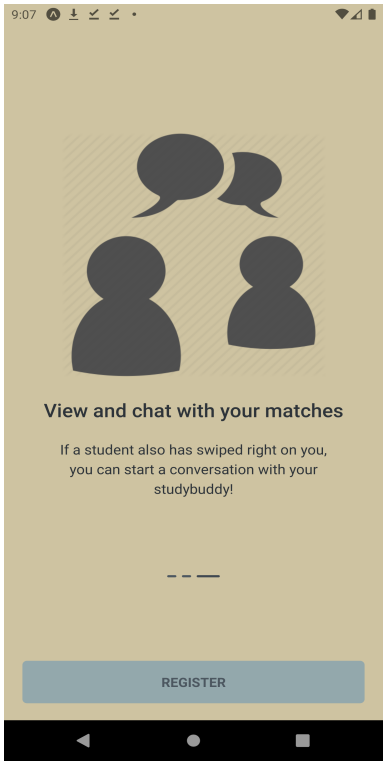
(a) Get Started Screen



(b) Onboarding Slider-1



(c) Onboarding Slider-2

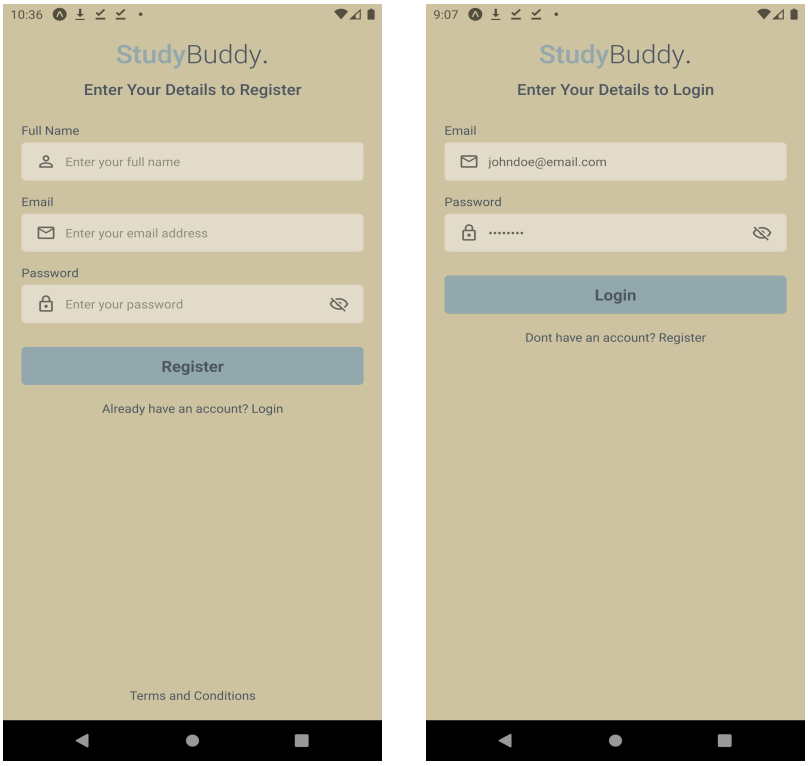


(d) Onboarding Slider-3

Figure 2.1: Getting Started

2.4.2 Registration and Login

Towards the end of the Onboarding slider, users find a register button, which takes them to the registration screen. As shown in Figure 2.2, if users are already registered, they can navigate to the login page. New users are required to provide a valid email, password, and full name to complete the registration process. After registration, they can log into the app. At the registration page, users can also view terms and conditions which are mentioned in detail in section 2.5.

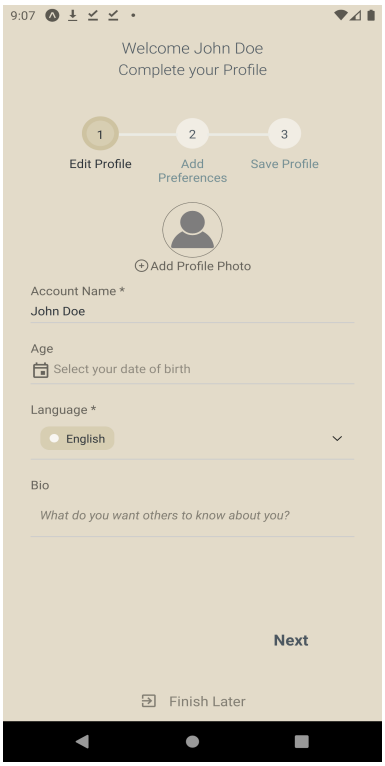


(a) Registration Screen (b) Login Screen

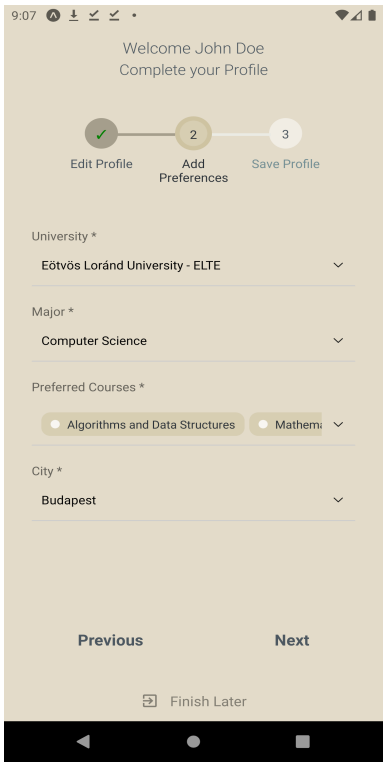
Figure 2.2: Authentication

2.4.3 Complete Profile

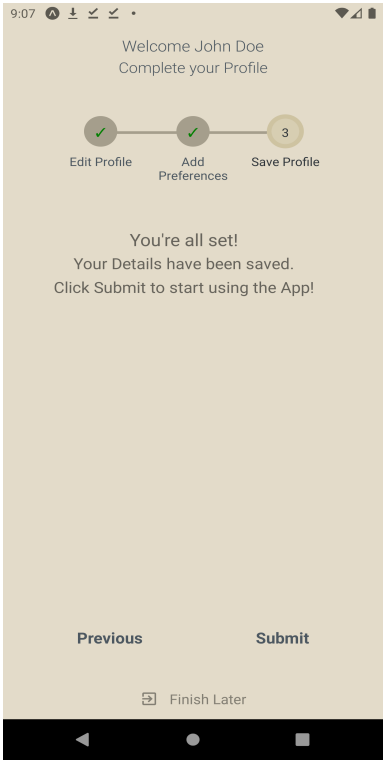
After login, users are directed to the complete profile section. They need to provide specific information, such as selecting up to four languages and courses, their university, major and city. These selections are mandatory for moving forward and are mentioned with asterisk as shown in Figure 2.3. Upon submission, users are navigated to the home screen.



(a) Edit Profile



(b) Add Preferences

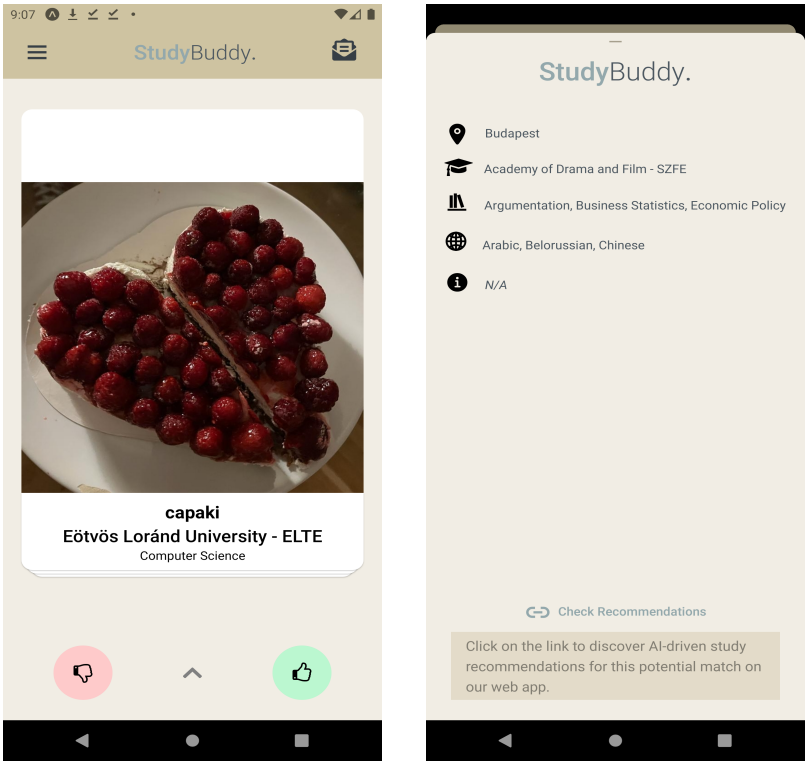


(c) Submit Details

Figure 2.3: Complete Profile Screen

2.4.4 Home Screen and Swipe Functionality

The home screen is the central hub of the app. Users can view other app users as cards that can be swiped left or right. Each card is accompanied by a user info modal at the bottom, displaying additional information for each user as shown in Figure 2.4. Users can also check recommendation scores for each other that is implemented by OpenAI API (see section 2.4.5). Users can swipe left or right (see figure 2.5) on the cards or tap the like/dislike buttons represented by thumbs up or thumbs down icons. Once all users have been swiped, the buttons are disabled, and no more cards are shown.



(a) Home Screen (b) User Info Modal

Figure 2.4: Home Screen

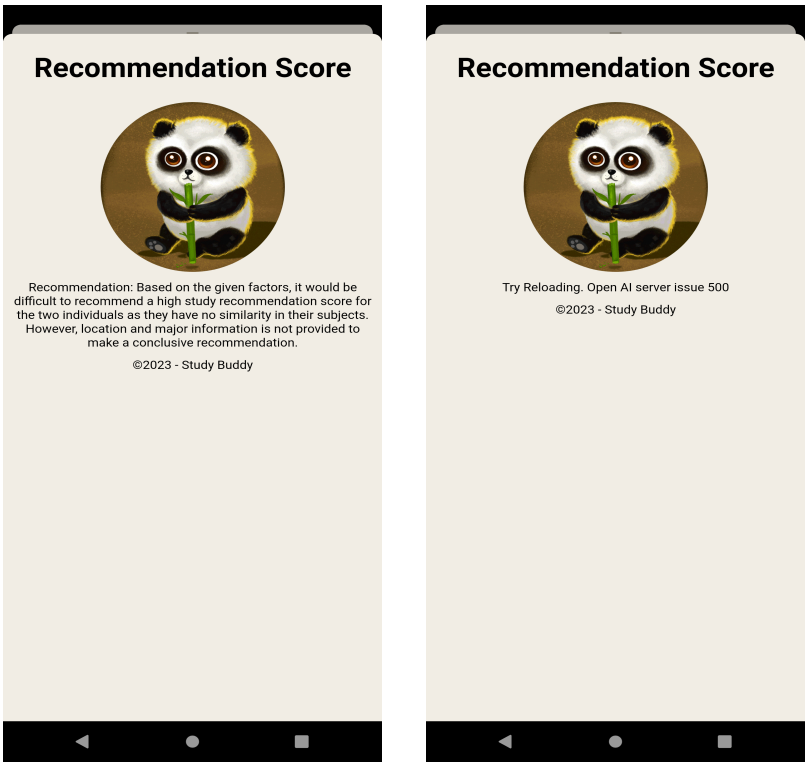
(a) Swiping

Figure 2.5: Home Screen

2.4.5 Recommendations

Users can conveniently check their recommendation scores by accessing the admin web app of our mobile application. The recommendation scores, provided by the OpenAI API, are displayed on a web page accessible through a unique link. The scores are given on a scale of ten, allowing users to gauge the relevance and suitability of recommendations for their specific needs.

In the event of any unforeseen slowdown or error in the OpenAI API, our web page incorporates an error handler as shown in figure 2.6. This ensures a seamless user experience by promptly addressing and communicating any temporary disruptions. We strive to minimize any impact on the accessibility and accuracy of the recommendation scores, providing users with reliable and valuable insights through our admin web app.

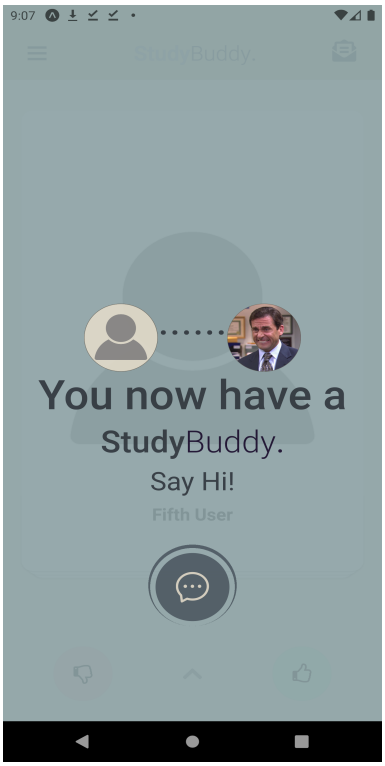


(a) Recommendation Score (b) Error

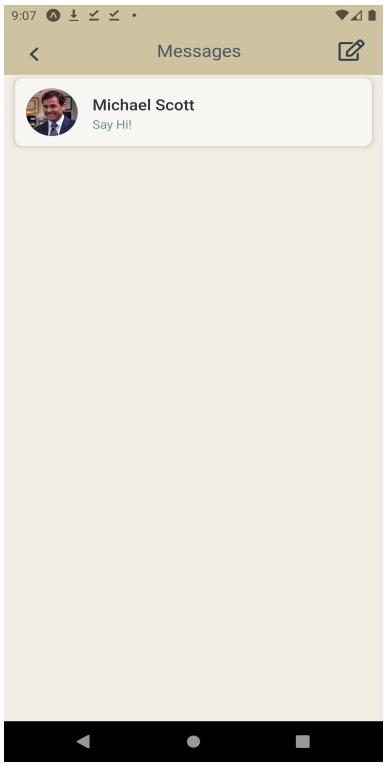
Figure 2.6: Recommendation

2.4.6 Matches and Messaging

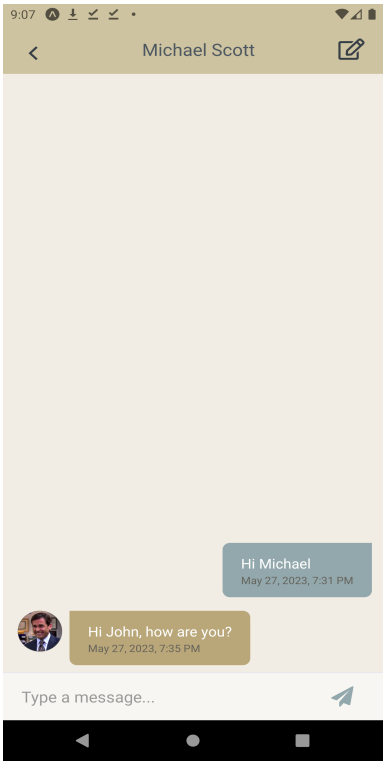
When two users mutually swipe right on each other's cards, they are instantly connected as "buddies". The user who swipes right for the second time will be shown a Match notification screen as shown in Figure 2.7. From there, they can effortlessly navigate to the messages screen and initiate a conversation with their new buddy. The messaging system operates on an asynchronous model, where messages are fetched from the server only when the chat screen is mounted or when a new message is sent. This approach optimizes resource usage by minimizing unnecessary API calls and ensures that messages are retrieved and displayed efficiently, enhancing the overall user experience.



(a) Match Modal Notification



(b) Messages Screen



(c) Messages Screen

Figure 2.7: Matching

2.4.7 Navigation Options

The home screen provides several navigation options as shown in Figure 2.8. A header with a messages icon allows users to access the messages screen from anywhere in the app. The top-left corner features a navicon that opens a drawer, which can also be accessed by sliding right on the screen. The drawer displays the logged-in user's profile image and number of buddies. Navigational options within the drawer include Home screen, Edit Profile screen, Matches screen, Contact Us screen, and Logout button.

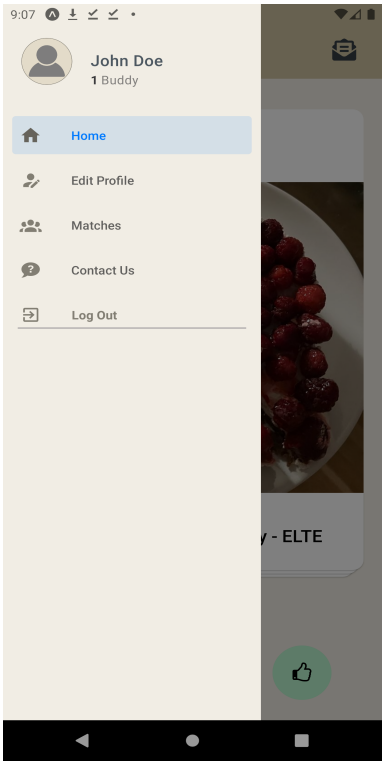
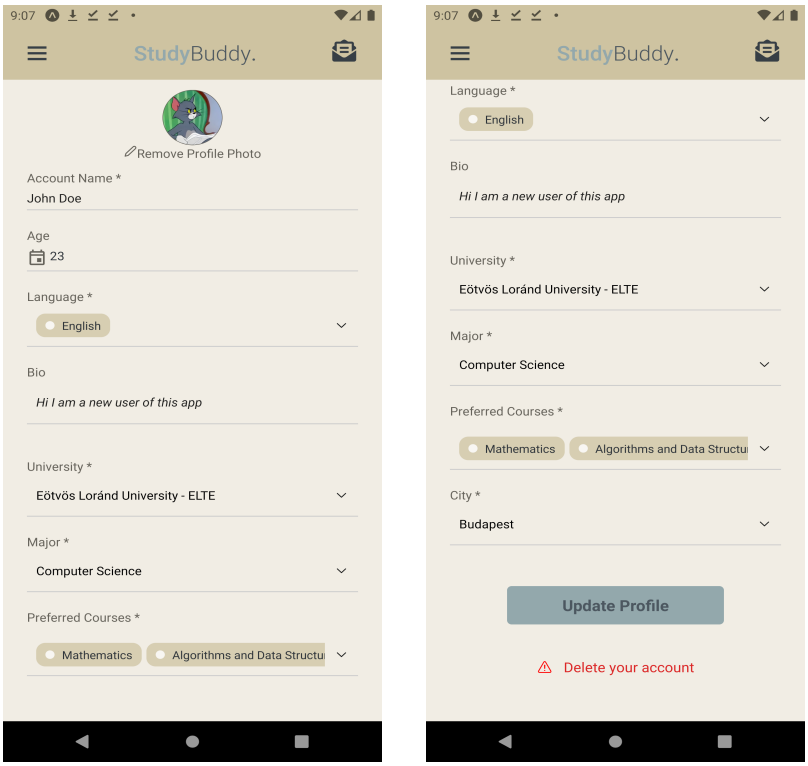


Figure 2.8: Drawer

2.4.8 Edit Profile

On this screen, users can make changes to their profile picture, bio, age, and other mandatory information. To update the profile, users have to scroll to the bottom and click the update profile button. They can also delete their accounts here if needed as shown in the Figure 2.9.



(a) Edit (b) Update

Figure 2.9: Edit Profile

2.4.9 Matches Screen

The Matches screen displays a list of buddies if available. If there are no buddies yet, a button is shown to navigate back to the home screen. In the list of buddies, users can click on a buddy to open a buddy info modal similar to 2.4, providing detailed information about the buddy. Each buddy listing includes message and delete buttons. Tapping the message button navigates the user to write a message, while the delete button removes the buddy from the list as shown in Figure 2.10

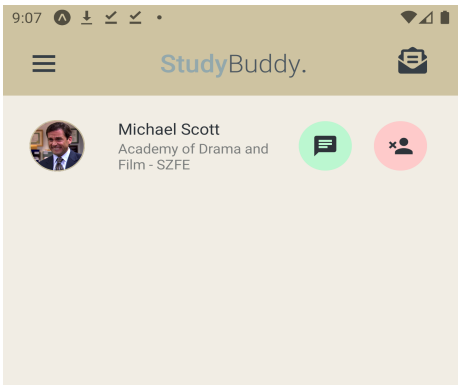


Figure 2.10: Matches Screen

2.4.10 Contact Us and Logout

The Contact Us screen (figure 2.11) provides developers' contact information for users who need assistance or wish to report problems. The Logout option, which is available in the drawer as shown in figure 2.8, triggers a prompt to confirm if the user wants to log out. Upon confirmation, users are navigated back to the login screen.

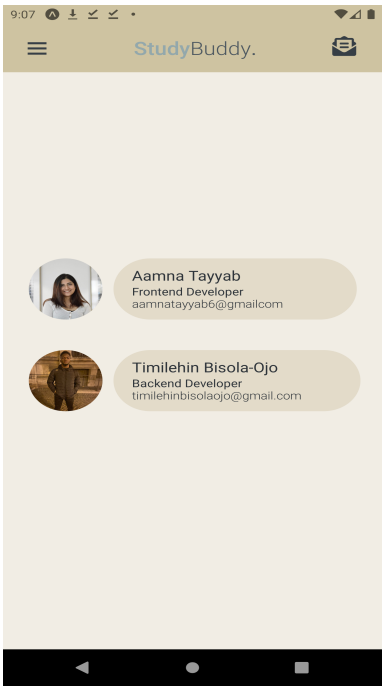


Figure 2.11: Contact Us Screen

To manage daily app usage and server load, users are automatically logged out after 50 minutes of session. The swipe functionality ensures that unwanted interactions are minimized, making the implementation of a topic forum or group messaging less relevant. (Refer to Section 2.8)

2.5 Terms and Conditions

During Registration process, users are required to accept Terms and Conditions which are stated in this section. Users are responsible for maintaining the confidentiality of their login data, including email address and password.
By using this application, users agree to strictly comply with the terms and agreements and undertake to refrain from engaging in the following activities:

1. Discrimination: Users must not discriminate based on age, ethnicity, color, religion, gender identity, sexual orientation, appearance, personal and political beliefs, or engage in any similar activities.
2. Copyright Violations: Users must not engage in posting or sharing any content that violates copyright rules.
3. Illegal Activities and Prohibited Content: Users must not engage in illegal activities or promote anything that violates community guidelines, such as selling or promoting drugs, pornography, or adult content.
4. Personal Information: Users must not disseminate their personal and sensitive information to other users or third parties without proper consent.
5. False or Misleading Statements: Users must not spread false, misleading, or deceptive statements or representations about the App, the Websites, or the Services.
6. Not a Dating or Matrimonial Platform: Users should not treat this application as a dating or matrimonial platform.

Each user is required to report any misconduct or abuse by other users to the creators of the app. In cases of hate speech, abusive behavior, or content dissemination, the creators reserves the right to permanently remove the user's account and take strict legal actions.

The terms and conditions, as well as the privacy policy, may be updated by the creators as the application evolves. Users are responsible for complying with the latest terms and conditions, and if they do not agree, they should cease using the application.

2.5.1 Messaging System and Privacy Policy

Our messaging system operates on an asynchronous model, where messages are fetched from the server only when the chat screen is mounted or when a new message is sent. This approach optimizes resource usage by minimizing unnecessary API calls and ensures that messages are retrieved and displayed efficiently, enhancing the overall user experience. The creators reserve the right to change the message exchanges to real-time chats in future updates.

2.5.2 Privacy Policy Statement

We collect necessary private information, such as name, email address, and location, to facilitate user interaction within our service. Optional sensitive information, like a profile picture and personal profile, can be provided to enhance profile visibility for potential study partners.

2.5.3 Collection of User Data

During registration, users will be asked to provide their personal email address. Additionally, users can select preferred courses and subjects to find suitable study partners. To improve the matching process, we may request additional information such as geolocation, university affiliation, and languages spoken fluently. However to uphold fairness and respect user privacy, we do not mandate the provision of age-related information during registration. Additionally, including a profile picture is optional, allowing users the choice to personalize their profiles at their discretion.

2.5.4 How User Information is Used

The application utilizes private and optional information to perform analysis and provide users with study buddy recommendation scores based on different preferences using OpenAI API (see section 3.5). All user-provided information is treated confidentially and used solely for managing accounts, finding suitable study partners, and providing customer support.

2.5.5 Data Protection

To safeguard user data from unauthorized access or disclosure, we employ security precautions. While we take measures to protect information, no system can guarantee 100 percent security. Users should also exercise caution when sharing personal information.

2.5.6 Contact Us

For any queries regarding our privacy policy or terms and conditions, please contact us via emails (aamnatayyab6@gmail.com or timilehinbisolaojo@gmail.com).

2.6 Network

The app operates on a network-dependent model, necessitating an active internet connection for full functionality. Users are required to have a stable Wi-Fi connection or cellular data connectivity to access and interact with the app's features. Similar to other messenger and social apps, StudyBuddy relies on network connectivity to send and receive data to and from the backend server.

2.7 Quick Start Guide

2.7.1 Video Demonstration

In addition to the tutorial, a video demonstration showcasing the key functionalities of StudyBuddy is available. The videos provide a visual walk-through of the app's features, highlighting its user-friendly interface and collaborative capabilities. The videos for registration of user is available at this link. The app usage video is available here.

2.8 Limitations and Constraints

While using the Study Buddy app, it's important to be aware of certain limitations and constraints. These constraints are outlined below to provide users with a clear understanding of the app's functionalities and restrictions:

1. **Platform Limitation:** The app was initially intended to be cross-platform; however, due to design considerations not aligning with iOS build guidelines, it is currently available only for Android devices. The Expo framework, although capable of building for iOS and web platforms, is optimized primarily for Android.
2. **Swipe Functionality:** The app incorporates swipe functionality to minimize unwanted interactions and focus on individual matching between users. As a result, the implementation of features such as topic forums or group messaging was deemed less relevant and therefore not included in the app.
3. **Geographic Restriction:** StudyBuddy is designed specifically for users in Hungary. Therefore, when selecting cities and universities within the app, only

options from Hungary will be available in the respective drop-down menus.

By understanding these constraints, users can align their expectations with the app's current capabilities and intended usage. The StudyBuddy app aims to provide a streamlined and focused study partner finding experience, tailored to the Hungarian context.

Chapter 3

Developer documentation

This chapter will offer insight into the project's development and implementation structure, design and extensibility. It will provide developers with a comprehensive understanding of the technology stack and functionality. This resource enables contributions, enhancements, and future development leveraging the project's components.

3.1 Development Tools

The frontend and backend repositories of the project can be forked, built and tested on GitHub. This application is developed using React Native (0.71.7) and Expo (SDK Version 48.0.10). The following tools provide a comprehensive overview of the development environment:

- Android Version 5+: Expo SDK Version 48.0.10 supports minimum SDK version 21 for Android which means that all versions after 21 can compile the application.
- Target Version: the compileSdkVersion is 33 for Expo SDK Version 48.0.0+ [1]
- Expo Command Line Interface (CLI) that serves as the primary interface between developer and other Expo tools.
- To use Expo CLI one needs to have the following tools on the developer machine:
  - Node.js

  - Git

- Visual Studio Code (or any preferred code editor).
- Expo Go Client App installed on your emulator or physical device (only for testing because it is not needed to build anything locally with Expo Go).

The development tools specific to the development of this app and the reasons of these choices are mentioned in the next Section 3.2.

3.2 Development Guidelines

3.2.1 Strategy

The front-end development strategy for this application focused on creating a cross platform application and an intuitive user experience built using windows operating system which turned out to be limitation during development later as mentioned in Section 2.8. To achieve this, a combination of carefully chosen tools and technology was employed. These tools played a vital role in shaping the app's design, functionality, and overall front-end development process. The following tools were utilized during the front-end development, chosen for their specific capabilities and benefits also mentioned below:

- React Native (0.71.7) and Expo (SDK Version 48.0.10): React Native is a cross-platform library that builds native mobile apps. It inherits all the functionalities from the popular web framework ReactJS. Expo is a framework that is used to build React Native apps. This comprehensive toolkit consists of various tools and services specifically designed for React Native. Expo was selected for several compelling reasons, including its ability to facilitate rapid development, handle native code complexities under the hood, eliminate the need for direct Android Studio involvement, provide Over-The-Air (OTA) updates, and offer the advantages of being a free and open-source framework. [2]
- Expo Command Line Interface (CLI) version 6.3.2.
- Node.js version 16.19.1 installed with npm version 8.19.3.
- Git version 2.33.1.windows.1.

- JDK 17.0.6 for generating apk.
- Visual Studio Code editor with useful extensions such as Tabnine, Expo Tools, and TailwindCSS.
- Expo Go Client App installed on Android 11 (API 30) emulator.

3.2.2 Codebase Organization

There are organized directories and files in the codebase to enhance readability, maintainability, and collaboration. Here is a description of the structure of the codebase for the front-end of the project:

1. **Root Directory:**
   - App.js: The main entry point of the application.
   - Config files: Various configuration files such as app.json, babel.config.js, package-lock.json, package.json, and tailwind.config.json. These files define project settings, dependencies, and build configurations.
2. **Src Folder:**
   - Assets: Contains the images folder, where images used in the application are stored.
   - Const: Includes files that hold reusable data that remains constant throughout the project, such as styles.js for custom styling, slides.js for onboarding slider information, colors.js for defining color schemes, and placeholder_image_uri.js for placeholder image URL.
   - api: Contains UserContext.js, a custom hook for managing context API state and functionality. It also contains Storage.js for managing Async Storage used by the app to decide whether app was launched before.
   - Navigation: Holds the DrawerNavigator and StackNavigator files responsible for managing the navigation flow within the application.
   - View: Consists of the components and screens folders.
     - Components: Contains custom reusable components that can be used across different screens.
     - Screens: Includes stack screens and drawer screens, which represent different screens of the application.

This is much simpler to understand from the codebase tree in figure 3.1.

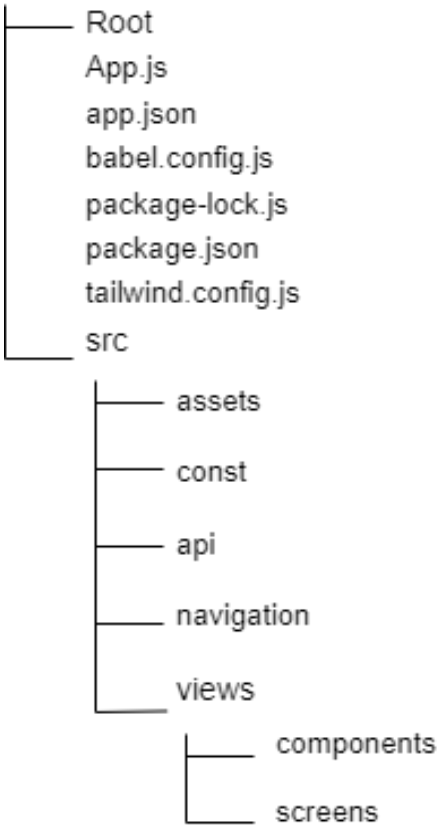


Figure 3.1: Folder Structure

3.2.3 Components and Modules

The configuration files in the project play a crucial role in defining and customizing the behavior of the application. In this section we will delve into the details of each file and custom components.

app.json: This file contains configuration options specific to the Expo project. These options change the behavior of the project while developing, building, submitting and updating app. It includes options such as the app's name, version, orientation, icon, splash screen, supported platforms, and more. By configuring this file, you can tailor the app's settings to meet specific requirements. The important aspects of the file are mentioned in snippet 3.1. The package is how Expo creates binary and APK for the app (more on this in section 3.2.4).

```
{
  "expo": {
    "name": "Study Buddy",
    "slug": "Study_Buddy",
    "platforms": [
      "ios",
      "android",
      "web"
    ],
    "version": "1.0.0",
    "orientation": "portrait",
    "icon": "./src/assets/images/icon.png",
    "userInterfaceStyle": "automatic",
    "splash": {
      "image": "./src/assets/images/splash.png",
      "resizeMode": "contain",
      "backgroundColor": "#CEC3A1"
    },
    "assetBundlePatterns": [
      "**/*",
      "src/assets/*",
      "src/assets/images/*"
    ],
    "android": {
      "adaptiveIcon": {
        "foregroundImage": "./src/assets/images/adaptive-icon.png",
        "backgroundColor": "#CEC3A1"
      },
      "package": "com.Study_Buddy.studybuddy",
    },
    "plugins": [
      [
        "expo-image-picker",
        {
          "photosPermission": "The app accesses your photos to let
↪  you share them with your friends."
        }
      ]
    ],
  }
}
```

Code 3.1: app.json

babel.config.js: The babel.config.js file (shown in 3.2) is used to configure Babel, a tool used for transpiling and transforming JavaScript code. In this

file, presets and plugins can be defined to specify how the code is transformed. In this specific configuration, the presets "babel-preset-expo" and plugins such as "nativewind/babel" and "react-native-reanimated/plugin" are utilized. These configurations ensure that the code is compatible with the desired platform and includes necessary transformations. For instance, "babel-preset-expo" is used for extending default React Native preset and adds support for optional native dependencies if they are installed, nativewind plugin is used for compiling Tailwind CSS styles and the plugin for reanimated provides usage of the API that simplifies the process of maintainable and smooth animations.

```
module.exports = function(api) {
  api.cache(true);
  return {
    presets: ['babel-preset-expo'],
    plugins: ["nativewind/babel", "react-native-reanimated/plugin"],
  };
};
```

Code 3.2: babel.config.js

package.json: The package.json file lists all the npm packages that the app depends on. It includes the project's name, version, main entry point, scripts, and the dependencies and devDependencies required for the app to run. By managing the packages through this file, one can easily install, update, and track the dependencies of this project. For instance, after cloning the project, we can run the following code to install all packages used in this app:

```
npm install
```

Then for running the expo app we just need to run the following command and select the relevant option. Metro (installed with Expo-CLI) takes care of the rest.

```
npx expo start
```

tailwind.config.js: This file is used with Tailwind CSS. It allows customizing the configuration of Tailwind CSS, such as defining custom colors, spacing, typography,

and more. It proved very beneficial as the styles are generated at build time so it is quite fast. The snippet in figure 3.2 shows how tailwind is used in a component for styling.

```
<View className="justify-center items-center">
  <Text className="text-xl text-outer-space font-normal">{title}</Text>
</View>
```

Figure 3.2: Tailwind styled View

Styling in the app is primarily achieved using NativeWind which uses tailwind CSS as scripting language to create a universal style system for React Native. NativeWind components can be shared between platforms and will output their styles as CSS StyleSheet on web and StyleSheet.create for native. It provides utility classes that simplify the styling process. However, custom styles are also defined in the styles.js file, which allows for additional customization and flexibility in styling the app's components. [3]

The codebase follows a modular approach, organizing components into separate files for reusability throughout the app. These custom components serve specific purposes and are imported into files using 'import' statements at the top. Additionally, the app utilizes external resources, such as image storage, which are further explained in the API section (refer to Section 3.3) for more details. This modular structure enhances code organization and promotes code reusability, making it easier to maintain and expand the application.
Components used in this app are functional components and are reusable, lightweight and use custom styling. To use the component, import from /src/views/components and include it in your desired screens or components. If there are any additional dependencies required, they are mentioned. Most items use styling from tailwind CSS in the component file, otherwise default styles are extended.

1. ButtonComponent:
    - API Documentation: It accepts props such as title, onPress to customize its appearance and behavior. It is used in Register Screen, Login Screen, Matches Screen etc.

- Interactions: It triggers the onPress event when pressed, allowing developers to define the desired functionality.

2. TextInputComponent:
   - Overview: The TextInputComponent provides an input field for users to enter text.
   - API Documentation: The TextInputComponent accepts props such as label, icon, error, password and onFocus function to manage the input field's content and appearance.
   - Dependencies: This component also uses MaterialCommunityIcons from @expo/vector-icons package.
   - Interactions: This component is used to render input fields on Register and Login Screen. It shows different styling to handle errors such as empty input field errors and also implements password field's secure text entry by enabling user to hide or show password by pressing on an icon. It sets the outlines of the text input boxes as darker or red in case of focus or errors respectively.
3. ChatList:
   - API Documentation: The ChatList component uses FlatList from react native, user and buddies from Context API (more on Context API in Section 3.3).
   - Interactions: The ChatList component is the only component other than header that is rendered on Messages screen. It shows list of chats.
4. ChatRow:
   - API Documentation: The ChatRow component accepts props which is data of the buddy with whom the chat is, such as email, name, messageId and photo to display chat information on the messages screen.
   - Dependencies: It uses useNavigation hook from react-navigation/native to navigate to chat screen.
   - Interactions: The component triggers the onPress event when clicked to be taken to Chat Screen where the chat of the logged in user with the buddy is displayed.
5. ReceiverMessage and SenderMessage:
   - Overview: The ReceiverMessage and SenderMessage components represent message bubbles for received and sent messages, respectively.

- API Documentation: Both components accept props such as message and time, ReceiverMessage component also accepts photourl as prop so it can load the receiver's profile picture with the message bubble.
- Interactions: The ReceiverMessage and SenderMessage components do not handle any specific interactions but can be combined with other components for messaging functionality.

6. CustomDrawerContent:
   - Overview: The CustomDrawerContent component provides a customized drawer menu for navigation through drawer navigator.
   - API Documentation: Usually it is a self contained component however, it displays logged in user's name, profile picture and buddy count therefore it requires props from drawer navigator which uses context API.
   - Dependencies: The component uses several packages like react-native-paper for use of Avatar, text styles, sections. It also uses packages like @react-navigation/drawer for building custom drawer by using its scroll view and drawer items list components. Other than that, @expo/vector-icons and @react-navigation/native packages are used for icons and handling navigation respectively.
   - Interactions: It can be opened on any drawer screens either by tapping the navicon from the header on top left corner or by sliding the screen towards right. One can navigate to Home, Edit Profile, Matches and Contact Us screens or can tap the logout button for logging out of the app.
7. Header:
   - API Documentation: The Header component accepts props such as title to customize its content.
   - Dependencies: Header also uses icon and navigation from @expo/vector-icons and @react-navigation/native packages.
   - Interactions: This particular header component is actually used in navigation from messages screen to chat screen to display name of the 'buddy' the user is engaging in messages with.
8. ImageComponent:
   - API Documentation: The ImageComponent only accepts a prop called 'card' to check if users being displayed on home screen have valid profile pictures or not. It displays image if a user has the image otherwise it

handles the error by setting the image to a placehold url.

9. Loader:
    - Overview: Loader displays react native's ActivityIndicator and a custom styled loading container with text during data fetching or processing.
    - API Documentation: The Loader component normally manages its own internal state, but it requires a boolean called visible to make handling its visibility in different API calls, easier.

3.2.4 Building

The tools needed for development are already mentioned in Section 3.1. Here are mentioned the tools needed to generate APK for fast packaging and distribution. The packaging and building is handled by Expo through EAS (Expo Application Services) which are the cloud services for the React Native Expo apps. EAS Build is a hosted service for building app binaries for Expo and React Native projects. [4]

Install the EAS CLI

EAS CLI is the dedicated command line for Expo Application Services.

```
npm install -g eas-cli
```

Log in to your Expo account

An Expo account is needed to use Expo Services which offers 30 lower-priority builds (of those, up to 15 iOS) per month with build timeout upto 45 minutes.

```
eas login
```

Create a configuration file

Configure EAS as shown:

```
eas build:configure
```

Select the platforms for which you would like to configure EAS Build. In this case it will be Android. This will generate an "eas.json" file. Modify the build settings as per needed. For APK, just edit the "preview" option as shown in the snippet 3.3.

```
{
"preview": {
      "android": {
        "buildType": "apk"
      }
    },
}
```

Code 3.3: eas.json

Run a build

Before the build can start, you'll need to generate or provide app signing credentials. [5]

```
eas build -p android --profile preview
```

3.2.5 Persistent Data Storage and State Management

In the app, AsyncStorage is used to store items related to the "user" entity. AsyncStorage is a storage system provided by React Native that allows key-value pairs to be persisted locally on the device. By utilizing AsyncStorage, data related to the user can be saved and retrieved across different app sessions.

To simplify the handling of AsyncStorage, a single file is dedicated to the implementation of the Context API present in the api folder as shown by the folder structure in Figure 3.1. The Context API allows for the creation and management of global application state. In this case, the main navigator in App.js is wrapped with the UserProvider from the Context API, ensuring that the data stored in the context is accessible throughout the app. This approach eliminates the need for making costly API calls on each screen and helps to optimize performance.

One drawback of the Context API is that it can potentially load all values even if only a specific object is needed. To address this, the useMemo hook and memoized values are utilized within the Context API. By specifying dependencies in the dependency array, only the necessary values are updated when the Context API is called, preventing unnecessary re-rendering and recreating of context values.

By managing all states in the Context API, the need to use useEffect on every screen multiple times is eliminated. Instead, useFocusEffect and useCallback are employed on screens where data needs to be refreshed after specific processes. The useFocusEffect hook ensures that the defined refresh methods are called when the screen comes into focus, while useCallback optimizes the performance of the refresh methods by memoizing their values and preventing unnecessary re-renders.

Most APIs responsible for fetching data from the back-end, login functionality and signout is being handled in context API. On signout function call, several actions are performed to ensure proper cleanup and logout.

1. The "user" item stored in AsyncStorage is removed, effectively clearing the user's data from storage.
2. All relevant data stored in the state variables is cleared, ensuring that any user-related information is reset.
3. Additionally, the timer responsible for automatically logging out the user is cleared to prevent any further automatic logout actions from occurring.

These steps collectively ensure that the user's data is securely cleared, state variables are reset, and any automatic logout mechanisms are halted upon sign-out. Overall, the combination of AsyncStorage, the Context API, memoized values in useMemo, useFocusEffect, and useCallback helps to efficiently manage data, reduce API calls, and optimize performance throughout the app.
For backend data storage, the app utilizes Firestore, a real-time NoSQL database. Firestore maintains collections of documents, and in this app, two main documents are used: "user" and "messages". The "user" document contains a collection of user profiles, while the "messages" document stores a map of messages exchanged between users.

To ensure data security, most API calls in the app are POST requests that include a token specific to the logged-in user. This token is then verified using Firebase JWT tokens, which helps authenticate and authorize the user on the back-end, thereby ensuring app security.

A more detailed description of the database structure and functionality can be found in the appendix A dedicated to back-end and database structure.

3.2.6 Backend and Third-Party Services

The application utilizes a Node.js backend with Firebase integration. The backend is responsible for handling various functionalities of the app, including user authentication and data management. Additionally, images are uploaded to the app through a third-party application called Cloudinary.

The backend of the application is hosted on OnRender, providing a reliable hosting environment for the server-side components. The database used for storing app data is Firestore, Firebase's NoSQL database, which offers flexibility and scalability for managing and organizing data.

There are several reasons for incorporating a backend with Firebase in the Study Buddy app. Firstly, the mobile app is controlled and managed through an admin web application developed using Flask. This allows for efficient administration and monitoring of the app's operations.

Additionally, the backend implementation includes the AI integration in the form of a recommendation algorithm model using OpenAI API. This algorithm enhances the app's functionality by providing personalized study partner recommendations to users based on their preferences.

To ensure secure user authentication, Firebase JWT authentication is implemented in the app. This means that token validation is performed through the backend, providing an extra layer of security by validating user access through the backend server rather than directly accessing Firestore documents.

These backend components and integration contribute to the overall functionality and user experience of the Study Buddy app, enabling seamless communication, personalized recommendations, and secure user authentication. More on this is mentioned in the documentation dedicated to back-end and storage.

3.3 API Documentation

This section provides an overview of the API calls used on the client-side to retrieve and submit data. The endpoints described here have been thoroughly tested by the backend developer using Postman (more detailed description see appendix A), ensuring their reliability and functionality. By following the instructions and

examples provided, you will be able to effectively make API calls from the client-side of your application to the server hosted on Render hosting service.

3.3.1 Registration

/createUser		
Method	**Parameters**	**Location**
POST	name, email, password	RegisterScreen.js

Table 3.1: API Description for Registration

The API is being called to register user to the hosted server. The call is posting data stored in 'inputs' object state as shown in the code snippet 3.4.

```
const signUpFunction = async () => {
  const url =
  ↪ "https://studybuddy-backend.onrender.com/createUser";
  setLoading(true);
  setTimeout(async () => {
    setLoading(false);
    try {
      const response = await axios.post(url, inputs);
      if (response.status === 200) {
        try {
          await AsyncStorage.setItem("user",
          ↪ JSON.stringify(inputs));
          navigation.navigate("Login");
        } catch (error) {
          Alert.alert("Error", "Something went wrong with app's
          ↪ storage.");
        }
      } else {
        Alert.alert("Error", "Something went wrong");
      }
    } catch (e) {
      Alert.alert("Error", "Email already exists, choose a
      ↪ different email.");
    }
  }, 3000);
};
```

Code 3.4: Client-side API Call

3.3.2 Login

/signIn		
Method	**Parameters**	**Location**
POST	email, password	UserContext.js

Table 3.2: API Description for Login

The API is being called to login user by posting user information 3.5. As seen from 3.2 that the function is located inside Context API and is being called on the login screen by using useContext() from React. This is how user is available to the entire app with UserContext Provider acting as a wrapper, and hence making it easier to handle routing as explained in section 3.2.5. This function also handles whether the user has completed profile or not by using 'flag'.

```
const loginFunction = async (email, password, navigation) => {
  const url = "https://studybuddy-backend.onrender.com/signIn";
  try {
    const response = await axios.post(url, {
      email: email,
      password: password,
    });
    if (response.status === 200) {
      if (response?.data?.token) {
        setToken(response?.data?.token);
        const flag = response?.data?.flag;
        let userData = {
          token: response?.data?.token,
          email: email,
          flag: flag,
        };
        if (flag) {
          AsyncStorage.setItem(
            "user",
            JSON.stringify({ ...userData, loggedIn: true })
          );
          setIsLoggedIn(true);
          navigation.navigate("Drawer");
        } else {
          AsyncStorage.setItem(
            "user",
            JSON.stringify({ ...userData, loggedIn: false })
          );
          navigation.navigate("CompleteProfile");
        }
      } else {
        Alert.alert("Error", "Invalid Login!");
      }
    } else {
      setLoading(false);
      Alert.alert("Request failed", "Something went wrong!");
    }
  }
};
```

Code 3.5: Client-side API Call

3.3.3 Complete Profile

/setUserData , image upload		
Method	**Parameters**	**Location**
POST	token, name, age, Language, Major, InterestedSubjects, Location, University, bio, photoUrl, flag	CompleteProfile.js
POST	uri, photo name, photo type	CompleteProfile.js

Table 3.3: API calls on Complete Profile Screen

The setUserData is being called to post user data to the server. This API is also used on Edit Profile page.

```
const setUserData = async () => {
  const url =
  ↪  "https://studybuddy-backend.onrender.com/setUserData";
  try {
    const postData = {
      token: token,
      name: inputs.account_name,
      age: age,
      Language: selectedLanguages,
      Major: selectedMajor,
      InterestedSubjects: selectedCourses,
      Location: selectedLocation,
      University: selectedUniversity,
      bio: inputs.bio,
      photoUrl: uploadedImageUrl || image,
      flag: true,
    };
    const response = await axios.post(url, postData);
    if (response.status === 200) {
      setUploadedImageUrl("");
      setImage("");
      setTimeout(() => {
        navigation.navigate("Drawer");
      }, 3000);
    } else {
      Alert.alert("Request Failed!", "Something went wrong!");
    }
  } catch (e) {
    Alert.alert("Error", "Something went wrong!");
  }
};
```

Code 3.6: Client-side API Call for setting user data

The is the API call made to Cloudinary (see section 3.5) in case user uploads a profile picture. The response then generates a url that can be sent to server to store as user's photoUrl.

```
const handleUpload = async (image) => {
  const data = new FormData();
  let uriParts = image.split(".");
  let fileType = uriParts[uriParts.length - 1];
  data.append("file", {
    uri: image,
    name: `photo.${fileType}`,
    type: `image/${fileType}`,
  });
  data.append("upload_preset", "study_buddy");
  const url =
  ↪  "https://api.cloudinary.com/v1_1/dg8969jxs/image/upload";
  const config = {
    headers: {
      "Content-Type": "multipart/form-data",
    },
  };
  try {
    setLoading(true);
    const response = await axios.post(url, data, config);
    setUploadedImageUrl(response.data.secure_url);
    setLoading(false);
  } catch (error) {
    setLoading(false);
    if (error.response) {
      console.log(error.response.data);
      console.log(error.response.status);
      console.log(error.response.headers);
    } else if (error.request) {
      console.log(error.request);
    } else {
      Alert.alert(
        "Error",
        "Something happened in setting up the request that
        ↪  triggered an Error"
      );
    }
    console.log(error.config);
    Alert.alert(
      "Error",
      "Something happened in setting up the request that triggered
      ↪  an Error"
    );
  }
};
```

Code 3.7: API call to Cloudinary (Image Upload)

3.3.4 Home

/getAllOtherUsers , /swipe		
Method	**Parameters**	**Location**
GET	token (Bearer)	UserContext.js
POST	email, buddy_email, swipe	UserContext.js

Table 3.4: API calls on home screen

The first API is being called to fetch all the other authenticated users that are not the currently logged in user or their buddies. This way user can still swipe on them.

```
const fetchAllOtherUsers = async (token) => {
  if (!user || !token) return;
  setLoading(true);
  try {
    const url =
    "https://studybuddy-backend.onrender.com/getAllOtherUsers";
    axios
      .get(url, { headers: { Authorization: `Bearer ${token}` } })
      .then((res) => {
        if (res.data) {
          setAllOtherUsers(res.data);
          setSwipedData(res.data[0]);
          setLoading(false);
        } else {
          Alert.alert("Error", "An error occurred, please try
          ↪  again later.");
          setLoading(false);
        }
      });
  } catch (error) {
    setLoading(false);
    Alert.alert("Error", "An error occurred, please try again
    ↪  later.");
  }
};
```

Code 3.8: Client-side API Call for getting users

The /swipe API posts the swipe information to the server to handle users' connections. Here the navigation to "MatchModal" also mimics the effect of a notification.

```
const swipe = async (direction, navigation) => {
  const url = "https://studybuddy-backend.onrender.com/swipe";
  try {
    const response = await axios.post(url, {
      email: user.email,
      buddy_email: swipedData?.email,
      swipe: direction,
    });
    if (response.status === 200) {
      refreshBuddies(); // refresh buddies if swipe happens
      if (response.data.isMatch && direction === true) {
        navigation.navigate("MatchModal", {
          newBuddy: swipedData,
          loggedInUser: user,
        });
      }
    }
  } catch (e) {
    Alert.alert("Error " + e.message);
  }
};
```

Code 3.9: Client-side API Call for Swipe

3.3.5 Matches

/getBuddies, /getUserData/:uid, /removeBuddy		
Method	**Parameters**	**Location**
POST	token (Bearer)	UserContext.js
GET	-	UserContext.js
POST	token, buddy_email, swipe	Matches.js

Table 3.5: API calls on home screen

The getBuddies api does two jobs of getting all buddies and then being able to select one from all based on uid.

```
const fetchBuddies = async (token) => {
  try {
    const response = await axios.post(
    "https://studybuddy-backend.onrender.com/getBuddies",{token:
    ↪  token,});
      if (response.status === 200) {
      const url =
    `https://studybuddy-backend.onrender.com/getUserData/${buddyId}`
        const buddyData = await Promise.all(
        response.data.map(async (buddyId) => {
            const buddyResponse = await axios.get(url);
            return buddyResponse.data;
        })
        );
        setBuddies(buddyData);
    } else {
        Alert.alert("Error", "An error occurred,
        please try again later.");
    }
} catch (error) {
    Alert.alert("Error", "An error occurred, please try again
    ↪  later.");
    }
};
```

Code 3.10: Client-side API Call for getting buddies

The /removeBuddy API call deletes the buddy.

```
const removeBuddy = async (token, buddy_email) => {
    try {
      const response = await axios.post(
    "https://studybuddy-backend.onrender.com/removeBuddy",
        {
          token: token,
          buddy_email: buddy_email,
        }
      );
      if (response.status === 200) {
        refreshBuddies();
      } else {
        Alert.alert("Error", "Something went wrong");
      }
    } catch (err) {
      Alert.alert("Error", "Something went wrong");
    }
  };
```

Code 3.11: Client-side API Call for Deleting Buddy

3.3.6 Messaging

/sendMessage, /getMessages2		
Method	**Parameters**	**Location**
POST	token, buddy_email, message	ChatScreen.js
POST	messageID	ChatScreen.js

Table 3.6: API calls on chat screen

The send message sets input to empty afterwards and also gets message as its not real-time messaging.

```
const sendMessage = async () => {
    const url =
    ↪  "https://studybuddy-backend.onrender.com/sendMessage";
    try {
      const postData = {
        token: token,
        buddy_email: email,
        message: input,
      };
      const response = await axios.post(url, postData);
      if (response.status === 200) {
        setInput("");
        getMessages();
      } else {
        Alert.alert("Error", "Something went wrong!");
      }
    } catch (e) {
      Alert.alert("Error", "Something went wrong!");
    }
  };
```

Code 3.12: Client-side API Call for sending messages

The API call for get message only happens when the screen is mounted or when user sends a message.

```
const getMessages = async () => {
    const url =
    ↪  "https://studybuddy-backend.onrender.com/getMessages2";
    try {
      const postData = {
        chatId: messageDetails,
      };
      const response = await axios.post(url, postData);
      if (response.status === 200) {
        const sortedMessages = response.data.sort((a, b) => {
          return new Date(b.time) - new Date(a.time);
        });
        setMessages(sortedMessages);
        await refreshUser();
      } else {
        Alert.alert("Error", "Something went wrong!");
      }
    } catch (e) {
      Alert.alert("Error", "Something went wrong!");
    }
  };
```

Code 3.13: Client-side API Call for Getting Messages

3.3.7 Delete Account

/deleteProfile		
Method	**Parameters**	**Location**
POST	token (Bearer)	MyProfile.js

Table 3.7: API Description for Deleting Account

The API is being called on Edit Profile page and deletes user's account and redirects user to Login Page.

```
const deleteAccount = async () => {
    const url =
    ↪  "https://studybuddy-backend.onrender.com/deleteProfile";
    try {
      const postData = {
        token: token,
      };
      const response = await axios.post(url, postData);
      if (response.status === 200) {
        signOut(false);
        Alert.alert("Success", "Your account was deleted!");
      } else {
        Alert.alert("Request Failed", "Something went wrong, try
        ↪  again later!");
      }
    } catch (e) {
      Alert.alert("Error", "Something went wrong!");
    }
  };
```

Code 3.14: Client-side API Call

3.4 Architecture

The navigational directions based on the app's architecture given by material design[6] are as follows:

- **Lateral Navigation:** The primary navigation component of the app, the Drawer Navigator, provides access to all top-level destinations. Users can seamlessly switch between screens such as Home, My Profile, Matches, and Contact Us, allowing them to explore different sections of the app without losing context.
- **Forward Navigation:** Forward navigation involves moving between screens at consecutive levels of hierarchy or steps in a flow. Users can navigate forward by interacting with various components, such as buttons or links, to access different sections of the app. For example, on the Home screen, users can swipe cards right or left using the React Native Deck Swiper component, which opens the User Info Modal screen for each card, providing detailed information about the user. Similarly, on the Matches screen, users can open the Buddy Info Modal to view additional details about a specific match. This

forward navigation functionality in Study Buddy ensures smooth transitions and empowers users to explore and engage with the app's features effortlessly.

- **Reverse Navigation:** Reverse navigation allows users to move backward through screens either chronologically or hierarchically. Chronological reverse navigation refers to moving backward through the app's screens in the order in which they were accessed. For instance, users can go back from the Chat screen to the Messages screen to revisit their conversations. Hierarchical reverse navigation follows platform conventions and allows users to navigate back within the app's navigation hierarchy. Users can, for example, return from the My Profile screen to the Home screen by using the back button or react native gesture handler. By incorporating lateral, forward, and reverse navigation within the StudyBuddy app, users can easily explore and interact with different screens, smoothly progress through different levels of hierarchy, and effortlessly navigate backward to revisit previous screens or steps in their user journey.

3.4.1 Navigational Hierarchy

The navigational hierarchy demonstrates the structure of the app's screens and how they are organized within the Navigation Container and Stack.Navigator. The Stack.Navigator manages the screens and their navigation options within the app. Based on the isLoggedIn state, the Stack.Navigator renders different screen groups. When isLoggedIn is true, the app renders screens related to the authenticated user. These screens include a Drawer.Navigator, which serves as the main navigation for the app. It contains screens such as HomeScreen, MyProfile, Matches, and ContactUs. Additionally, there are screens like MessagesScreen and ChatScreen for managing messages and conversations between users. The app also includes modal screens, UserInfoModal and BuddyInfoModal, for displaying user information. Another modal screen, MatchModal, is used to show matching results. When isLoggedIn is false, indicating that the user is not authenticated, the app renders screens for the user authentication process. These screens include GetStartedScreen, OnboardingSlider, RegisterScreen, LoginScreen, and CompleteProfile.
The application's navigational hierarchy consists of a NavigationContainer component which holds one Stack Navigator only which holds different screens. The hierarchy is as follows:

- Stack.Group (isLoggedIn)
  - Stack.Screen (name: "Drawer", component: DrawerNavigator)
    - Drawer.Navigator
      - Drawer.Screen (name: "Home", component: HomeScreen)
      - Drawer.Screen (name: "MyProfile", component: MyProfile)
      - Drawer.Screen (name: "Matches", component: Matches)
      - Drawer.Screen (name: "ContactUs", component: ContactUs)
  - Stack.Screen (name: "Messages", component: MessagesScreen)
  - Stack.Screen (name: "Chat", component: ChatScreen)
  - Stack.Group (Modal Screens)
    - Stack.Screen (name: "UserInfoModal", component: UserInfoModal)
    - Stack.Screen (name: "BuddyInfoModal", component: BuddyInfoModal)
  - Stack.Group (transparent Modal Screen)
    - Stack.Screen (name: "MatchModal", component: MatchModal)
- Stack.Group (!isLoggedIn)
  - Stack.Screen (name: "GetStarted", component: GetStartedScreen)
  - Stack.Screen (name: "OnboardingSlider", component: OnboardingSlider)
  - Stack.Screen (name: "Register", component: RegisterScreen)
  - Stack.Screen (name: "Login", component: LoginScreen)
  - Stack.Screen (name: "CompleteProfile", component: CompleteProfile)

3.4.2 Navigation Map

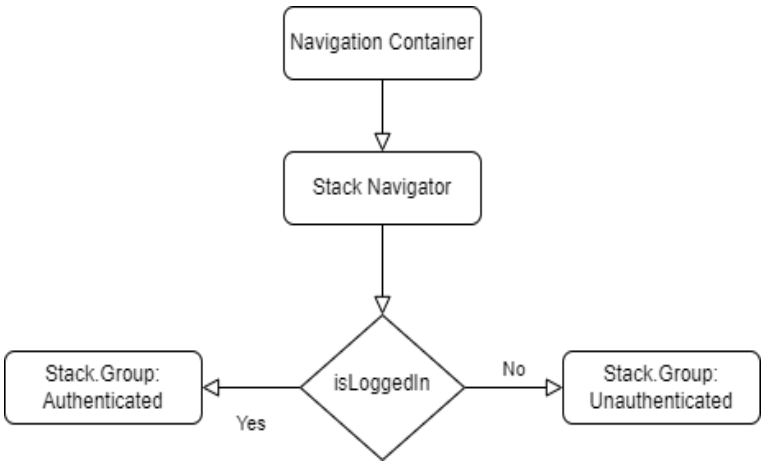


Figure 3.3: Navigation Map

Based on the hierarchy represented in the subsection 3.4.1, we have constructed a navigation map in Figure 3.3 which provides a visual representation of the navigational flow of the app's screens. It showcases the conditional rendering based on the isLoggedIn state, highlighting the screens available to authenticated and unauthenticated users shown in Figure 3.4 and Figure 3.5 respectively.

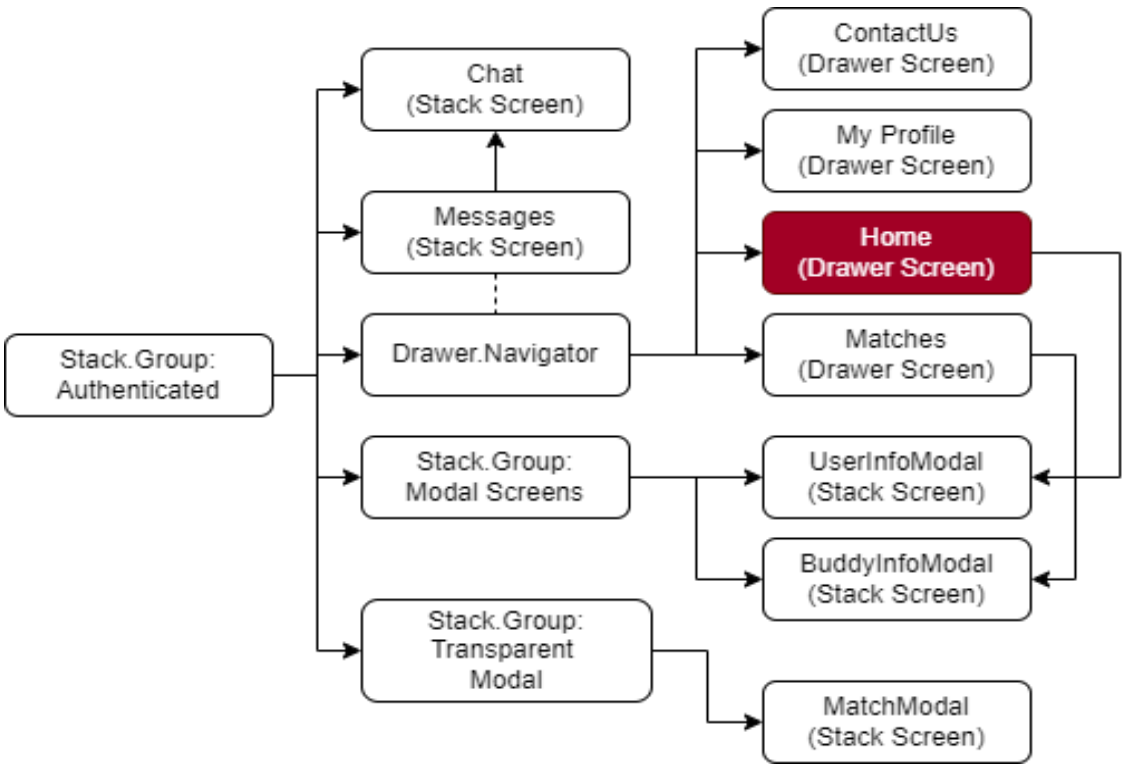


Figure 3.4: Navigation Map (if user is logged in)

In case where isLoggedIn state is set to *true* the initial route is set to Home Screen as shown in Figure 3.4. The Messages screen is available through header section of the app from Drawer Screens. The diagram shows the inclusion in Stack if user is autheticated. For instance, authenticated user can only reach to Chat screen through Messages screen but Chat is still a part of the given stack group. The user flow of the app will be shown in Section 2.4.

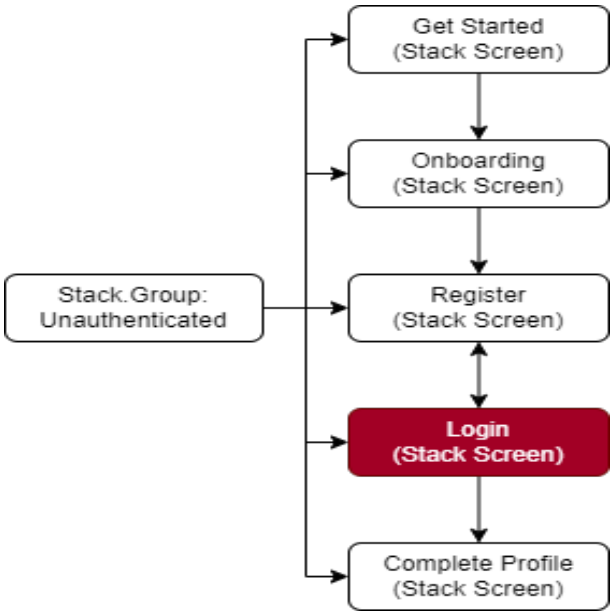


Figure 3.5: Navigation Map (if user is not logged in)

Similarly, in the case where isLoggedIn state is set to *false* the initial route is set to Login Screen as shown in Figure 3.5. The user can switch between Registration and Login screens. In order to navigate to authenticated stack, user needs to complete profile from Complete Profile screen.

3.5 Libraries and Sources

The application utilizes various third-party libraries and services to enhance frontend design and streamline development. Notable libraries include expo/vector-icons and expo-image for creating an appealing frontend design. The decision to incorporate expo-image was made during the later stages of development due to caching issues discovered during testing, which will be discussed further in Section 3.7.3. Additionally, the following external sources were utilized:

- **Figma:** Used for creating icons, splash screens, and wireframes.

- **Expo Metro Bundler:** Employed as a reliable bundling tool within the Expo ecosystem. (Used by Expo-CLI) [7]
- **Coolors:** Utilized to generate a color palette for the application's design. [8]
- **Tailwind CSS:** Adopted for simplified styling capabilities. [9]
- **Cloudinary:** Integrated as an image storage service. [10]
- **Alphabetical Sorter:** VS code extention used in sorting JSON data used in dropdowns.
- **Icon-Library:** Leveraged as a free icon library to generate logos, icons, and images for various screens, including the app icon. [11]
- **gifgit.com:** Provided free online tools for editing the said icons and images.
- **OpenAI API:** Users can conveniently check their recommendation scores by accessing the admin web app of our mobile application. More on this is detailed in section 2.4.5.

These third-party resources were instrumental in achieving the desired functionality and aesthetics of the application.

3.6 Testing Plan

In the case of testing frontend, mostly unit testing is implemented with jest version 29.2.1 and jest-expo 48.02.2. The entire user-flow is tested by end-to-end (e2e) testing. The unit testing with jest is both white box and black box in this case, here is how:

1. White Box Testing:
    - The test cases will access the internal implementation details of components. They render the component using render() from the @testing-library/react-native package and make assertions based on the component's internal state or properties.
    - Example would be testing the presence of specific elements or properties like getByTestId, getByText, and accessing props.style.opacity as shown in the following code snippet 3.15

```
import React from "react";
import { fireEvent, render } from "@testing-library/react-native";
import ButtonComponent from
 ↪  "../../../../src/views/components/ButtonComponent";

describe("Button Component", () => {
  afterEach(() => {
    jest.clearAllMocks();
  });

  test("1. Render Button", () => {
    const { getByTestId } = render(<ButtonComponent title="Test
     ↪  Title" />);
    expect(getByTestId("button")).toBeTruthy();
  });

  test("2. Title and disabled props", () => {
    const { getByText, getByTestId } = render(<ButtonComponent
     ↪  title="Test Title" disabled={true} />);
    expect(getByText("Test Title")).toBeTruthy();
    expect(getByTestId('pressButton').props.style.opacity)
    .toBe(0.3);
  });

  test("3. Button onPress function", () => {
    const mockOnPress = jest.fn();
    const { getByTestId } = render(<ButtonComponent title="Test
     ↪  Title" onPress={mockOnPress} />);

    fireEvent.press(getByTestId('pressButton'));
    expect(mockOnPress).toHaveBeenCalledTimes(1);
  });
});
```

Code 3.15: Testing ButtonComponent

2. Black Box Testing:
   - The test cases also focus on the external behavior of the components. They interact with the rendered component as a black box without considering the internal implementation details.
   - Examples of black box testing in the app are simulating button presses using fireEvent.press and checking the expected behavior or side effects, such as verifying that the mockOnPress function is called as shown in the code snippet 3.15.

3.7 Testing Execution

3.7.1 Unit Testing

For unit testing with Jest, tests are written using the Jest testing syntax and can be organized into test suites and test cases. For testing its the same working tree as that of the src directory but in tests folder. Tests are typically placed in separate files within the test directory, following a naming convention (e.g. '*test.js').

Mocking Dependencies: Jest allows for easy mocking of dependencies, such as API calls or external libraries, to isolate and control the behavior of the code being tested. Mocking is useful for focusing on specific code paths, simulating different scenarios, and avoiding unwanted side effects during testing. This mocking is mainly done in the mocks folder as shown in Figure 3.6.

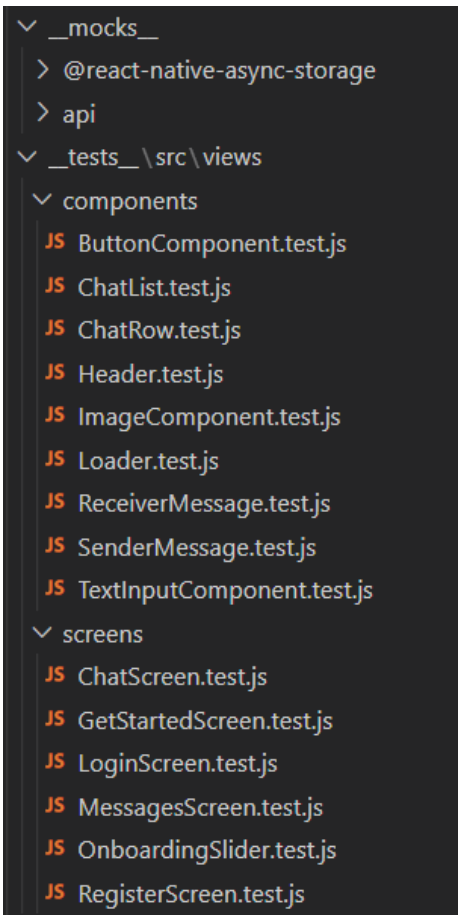


Figure 3.6: Folder Structure (Tests)

All the major code components are tested with unit testing by Jest. There are 15 test suites and 32 test cases as shown in figure 3.7.

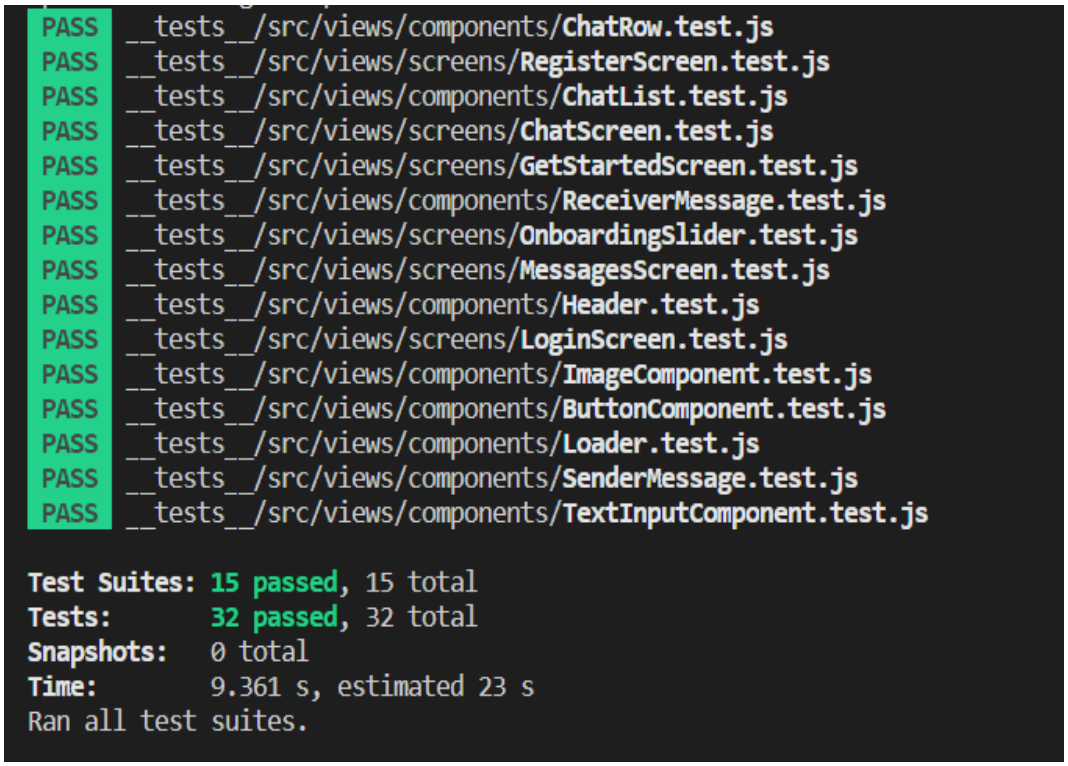


Figure 3.7: Passed Test Cases

3.7.2 Manual Testing

The following user stories outline the end-to-end (e2e) testing of the user-flow within the application. These user stories cover various scenarios and interactions that users may encounter while navigating through the app. Each user story represents a specific task or action that a user would perform, allowing us to test the functionality and user experience of the app across different screens and features. The user stories are presented in a simplified format, using the "Given," "When," "Then," and "And" statements to describe the desired behavior and outcomes of each scenario.

Title: Onboarding Introduction

Scenario:

- Given that I am a new user
- When I launch the app for the first time
- Then I expect to see an introduction screen with the app-icon, tagline and an option to get started.

Title: Navigating to Onboarding Slider

Scenario:

- Given that I have opened the app and I am on the get started screen
- When I tap on the "Get Started" button
- Then I expect to be navigated to the onboarding slider screen.

Title: Viewing Onboarding Slides

Scenario:

- Given that I am on the onboarding slider screen
- When I swipe or tap 'Next' to proceed through the slides
- Then I expect to see informative slides that showcase the app's key features and benefits.

Title: Skipping Onboarding

Scenario:

- Given that I am on the onboarding slider screen
- When I choose to skip the onboarding process by tapping 'Skip'
- Then I expect to be directed to the last slider screen which shows the 'Register' button.

Title: Registering with Full Name, Email, and Password

Scenario:

- Given that I am on the registration screen
- When I enter my full name, email, and password
- And I tap on the "Register" button
- Then I expect my registration information to be validated and stored securely.
- And I should be redirected to the login screen.

Title: Logging In with Email and Password

Scenario:

- Given that I am on the login screen
- When I enter my registered email and password
- And I tap on the "Login" button
- Then I expect my login credentials to be authenticated.
- And I should be directed to the complete profile screen.

Title: Navigating to Login or Registration from Registration Screen
Scenario:

- Given that I am on the registration screen
- When I view the options below the registration button
- Then I expect to see a text prompting me to navigate to the login screen if I already have an account.

Title: Navigating to Login or Registration from Login Screen
Scenario:

- Given that I am on the login screen
- When I enter my registered email and password
- And I tap on the "Login" button
- Then I expect my login credentials to be authenticated.
- And I should be directed to the complete profile screen.

Title: Logging In with Email and Password
Scenario:

- Given that I am on the login screen
- When I view the options below the login button
- Then I expect to see a text prompting me to navigate to the registration screen if I don't have an account.

Title: Skip Completing Profile on Login
Scenario:

- Given that I already have an account
- When I log in with my credentials
- Then I should be directed straight to the home screen, bypassing the profile completion process.

Title: Providing Mandatory Information on Complete Profile
Scenario:

- Given that I have successfully logged in and directed to the complete profile screen

- When I enter my account name and select up to four preferred languages from the dropdown menu
- And I click on "Next" to proceed to the next step
- Then I should be able to navigate back to the previous step by clicking "Previous."

Title: Choosing to Finish Profile Later

Scenario:

- Given that I am on the complete profile screen
- When I decide to press the "Finish Later" button
- Then I should be prompted with an alert message informing me that all entered information will be lost if I proceed.
- If I confirm the prompt, I should be taken back to the login screen.
- If I cancel the prompt, I should remain on the complete profile screen to continue entering the information.

Title: Submitting Complete Profile Information

Scenario:

- Given that I have completed all progress steps of the profile completion process
- When I reach the final step and review my information
- And I click on the "Submit" button
- Then I expect my profile details to be saved
- And I should be directed to the home screen to access the full app experience.

Title: View on Home Screen

Scenario:

- Given that I am on the home screen
- When I access the main screen
- Then I should be able to see a swiper component that displays user profile cards. Each card should contain the user's profile picture, name, university, and major information. At the bottom of the screen, there should be two buttons for "Like" and "Dislike" actions. Between the like and dislike buttons,

there should be an up arrow button that, when clicked, opens a modal screen displaying additional information about the user.

Title: Like and Dislike Buttons with Card Swipe Actions
Scenario:

- Given that I am on the home screen
- When I click the "Like" button or perform a swipe right gesture on a user's profile card
- Then the user's profile card should be swiped to the right, indicating a positive interaction

Title: Notification of Mutual Swipe Right and Buddies Connection
Scenario:

- Given that I have swiped right on a user's profile card on the home screen
- When the user I swiped right on also swipes right on my profile card
- Then I should receive a transparent modal notification stating this user is now my StudyBuddy with a button to message the new buddy and initiate a conversation.

Title: Dislike Button with Card Swipe Action
Scenario:

- Given that I am on the home screen with user profile cards and like/dislike buttons
- When I click the "Dislike" button or perform a swipe left gesture on a user's profile card
- Then the user's profile card should be swiped to the left, indicating a negative interaction and no notification or action is triggered if the user I disliked swipes left or dislikes my profile card.

Title: Modal Screen with User Information on Arrow Button Press
Scenario:

- Given that I am on the home screen with user profile cards and an arrow button at the bottom center

- When I tap the arrow button
- Then a modal screen should appear, displaying detailed information about the current user on the profile card

Title: Navigating to Messages and Sending/Receiving Messages
Scenario:

- Given that I am on the home screen
- When I tap on the messages icon in the top-right corner of the screen
- Then I should be navigated to the messages screen, where I can view my conversations and select a user to message.

Title: Selecting a User to Message
Scenario:

- Given that I am on the messages screen
- When I choose a user from the list of conversations
- Then I should be directed to a chat interface where I can send and receive messages with that user.

Title: Sending and Receiving Messages
Scenario:

- Given that I am in a chat interface with another user
- When I type a message and press the send button
- Then my message should be sent to the other user, and I should see it in the chat.
- If the other user sends a message, I should receive it through an asynchronous API call. To view the received message, I need to navigate back to the messages screen or send another message.

Title: Testing Edit Profile Screen and Account Deletion
Scenario:

- Given that I am on the home screen and have access to the side drawer
- When I select the "Edit Profile" option from the side drawer
- Then I should be navigated to the edit profile screen.

Title: Editing Profile Details on Edit Profile Screen

Scenario:

- Given that I am on the edit profile screen
- When I make changes to my profile, such as editing or removing my profile picture, updating personal information, or modifying any other editable fields
- And I have filled out all mandatory fields
- Then I need to click the "Update Profile" button to save the changes.

Title: Deleting My Account on Edit Profile Screen

Scenario:

- Given that I am on the edit profile screen
- When I choose the option to delete my account
- And I confirm the account deletion
- Then I should be taken to the login screen, indicating that my account has been successfully deleted.

Title: Testing Matches Screen and Buddy Interaction

Scenario:

- Given that I am on the home screen and have access to the side drawer
- When I select the "Matches" option from the side drawer
- Then I should be navigated to the matches screen.

Title: Viewing Buddies List on Matches Screen

Scenario:

- Given that I am on the matches screen
- When I view the list of buddies
- Then I should see each buddy's name, along with a message button and a delete button in front of their name.

Title: Viewing Buddy Information on Buddy Selection

Scenario:

- Given that I am on the matches screen
- When I click on a buddy from the list

- Then a modal should open, displaying detailed information about the selected buddy.

Title: Navigating to Chat Screen from Matches

Scenario:

- Given that I am on the matches screen
- When I click on the message button associated with a buddy
- Then I should be directed to the chat screen with that specific buddy, allowing me to initiate a conversation.

Title: Removing Buddy on Confirmation

Scenario:

- Given that I am on the matches screen
- When I click on the delete button associated with a buddy
- And I confirm the deletion
- Then the buddy should be removed from my buddies list, indicating a successful deletion.

Title: Viewing Contact Us Page and Static Information

Scenario:

- Given that I am on any drawer screen with have access to the side drawer
- When I select the "Contact Us" option from the side drawer
- Then I should be navigated to the contact us page.

Title: Viewing Static Information

Scenario:

- Given that I am on the contact us page
- When I view the page's content
- Then I should see static information about the developers of the app.

Title: Logging Out and Confirmation

Scenario:

- Given that I am on any drawer screen with have access to the side drawer

- When I select the "Logout" option from the side drawer
- Then a confirmation prompt should appear, asking me to confirm the logout.

Title: Confirming Logout and Navigating to Login Screen
Scenario:

- Given that I have been prompted to confirm the logout
- When I confirm the logout action
- Then I should be signed out of the app and taken to the login screen.

3.7.3 Findings

During the testing phase of our React Native Expo application, several critical findings emerged that could significantly improve the app's functionality and user experience.

Firstly, it was observed that images weren't updating in real time due to caching issues. To resolve this, I opted for the 'expo-image' package which effectively handled image caching and was compatible with our app's swipe functionality.

Secondly, I identified superfluous API calls originating from the usage of Cloudinary, an external cloud service for image upload. Each image upload generated a new link, even when an image was removed from the user's profile, leading to a lack of comparison capabilities. As a result, the removed image would still be updated.

Upon user testing, it was revealed that the app's messaging system wasn't real-time, a feature the users weren't aware of. Consequently, we've decided to include this detail in our terms and conditions to set the right user expectations, underlining that messaging is asynchronous. However, we are planning to integrate real-time messaging in later versions to enhance the overall user experience.

Another notable finding was related to the app's logout functionality. The current sign-out mechanism triggers re-rendering due to multiple navigations, which could be better addressed using a logout listener. This adjustment will improve the application's performance and provide a smoother user experience.

An additional issue was detected with the loading indicator, which sometimes rendered twice due to overlapping API calls. This will be rectified to ensure a consistent user interface.

Finally, I employed a module, 'Storage.js', to manage async storage exclusively for tracking whether the app has been launched previously. If this tool had been implemented earlier in the development process, it would have also elegantly managed the async storage of user information.

In conclusion, these observations will guide our future updates, improving the app's performance and delivering a more seamless user experience.

Chapter 4

Conclusion

The development of the Study Buddy application has been a rich, learning-filled journey. Motivated by the challenges students face in online education due to COVID-19, our goal was to provide a platform that eases the process of forming study groups.

From its initial conception to its current state, the process has been both demanding and rewarding. Recognition at the online UI hackathon was a motivating factor, reaffirming our belief in the app's potential.

Through testing, we identified various improvements, such as real-time image updates, reducing unnecessary API calls, and the need for real-time messaging. We also learned the importance of incorporating certain tools, like 'Storage.js', earlier in development.

As we move forward, user feedback and testing outcomes will guide our enhancements to the app. We plan to introduce real-time messaging, improve logout functionality, and refine user data handling with async storage.

In summary, the experience of building Study Buddy has deepened our understanding of app development and highlighted the importance of continuous improvement. Our journey is a testament to how motivated individuals can create meaningful solutions to real-world challenges, and we hope our work contributes to the evolving discourse on digital education.

Bibliography

[1] *Expo SDK Support for Android and iOS versions*. Accessed January, 2023. 2023. URL: https://docs.expo.dev/versions/latest/#support-for-android-and-ios-versions.

[2] Hugo Hutri. "Comparison of React Native and Expo". In: (2023).

[3] *NativeWind*. 2023. URL: https://www.nativewind.dev/.

[4] *EAS Build*. 2023. URL: https://docs.expo.dev/build/introduction/.

[5] *Android App Signing Credentials*. 2023. URL: https://docs.expo.dev/build/setup/#android-app-signing-credentials.

[6] *Understanding navigation*. Accessed January, 2023. URL: https://m2.material.io/design/navigation/understanding-navigation.html.

[7] *Metro Bundler*. 2023. URL: https://docs.expo.dev/guides/customizing-metro/.

[8] *Coolors Palette*. 2023. URL: https://coolors.co/4c5760-93a8ac-5e5131-a59e8c-66635b.

[9] *Tailwind*. 2023. URL: https://tailwindcss.com/docs/installation.

[10] *Cloudinary*. 2023. URL: https://cloudinary.com/documentation.

[11] *Icon Library*. 2023. URL: https://icon-library.com/.

List of Figures

List of Tables

List of Codes

Appendix A

Backend Documentation

This appendix serves as a comprehensive documentation resource for the backend of the mobile app and the web app created for the administration of the mobile app. It provides detailed information about the architecture, functionality, and implementation of the backend systems supporting the mobile app's core features all implemented by my colleague, Timilehin Bisola-Ojo. Additionally, you will find complete API documentation, outlining the available endpoints, request/response formats, and authentication methods for seamless integration with the app. Furthermore, this appendix includes an overview of the database structure, schemas, and relationships, offering insights into the data management and storage aspects of the application. This documentation serves as a valuable reference for developers and users involved in understanding the backend infrastructure of the mobile app.

Eötvös Loránd University

Faculty of Informatics

Department of Programming Languages and Compilers

StudyBuddy Backend

Supervisor:
Gregory Morse
PHD Student

Author:
Timilehin Bisola-Ojo
Computer Science BSc

Budapest, 2023

EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

Thesis Registration Form

Student's Data:

Student's Name: Bisola-Ojo Timilehin Oyedeji

Student's Neptun code: AIE3BR

Course Data:

Student's Major: Computer Science BSc

I have an internal supervisor

Internal Supervisor's Name: *Gregory Morse*

Supervisor's Home Institution: ***Department of Programming Languages and Compilers***

Address of Supervisor's Home Institution: ***1117, Budapest, Pázmány Péter sétány 1/C.***

Supervisor's Position and Degree: *PhD Student with MSc Computer Science from ELTE*

Thesis Title: Study Buddy

Topic of the Thesis:

(Upon consulting with your supervisor, give a 150-300-word-long synopsis os your planned thesis.)

Year after year, students struggle to find people of similar subjects to study with. This problem was made worse by the COVID-19 pandemic which affected social interactions and made many students unable to study with their peers.
My project will be to write the back end of a cross-platform application that allows people who are looking for other people to study with called Study Buddy using Machine Learning to match people according to Subject/ school. Users can study with individuals or in groups. Users can create or join study groups in the app. They can discuss different Education Topics with other users on different Topic Forums. The target user demographics of this application are university students.
It will be cross-platform. The users will be able to create and delete accounts. I will be responsible for data storage, security, and other server-side functions. This app is developed in teamwork with another student, Aamna Tayyab who will work with the front end. We will have extended features of the application divided among the team members.

Budapest, 2022. 12. 01.

Table of Contents

USER DOCUMENTATION

Introduction

This backend is part of a larger project called StudyBuddy. It is a collaboration project between I and Aamna Tayyab. The documentation of the frontend is attached to this as an Appendix. The application is aimed at helping students find others to study with them.

Project Overview

The StudyBuddy Backend is a server-side application built using Flask, Node.js and Express.js. It provides a range of API endpoints and an admin website for managing user accounts, authentication, user data, messaging, and other functionalities related to a StudyBuddy matching system. The project aims to facilitate communication and collaboration among students to find suitable study partners.

Features of the API:

1. User Management:
    - Create a user account by providing name, email, and password.
    - Authenticate users with email and password for signing in.
    - Sign out a user from the system.
    - Update user email and password.
2. User Data:
    - Set and retrieve user-specific data such as name, age, language, major, interested subjects, location, university, bio, and profile picture.
    - Get all users' data, including their personal information.
    - Retrieve data for a specific user based on their unique identifier (uid).
    - Retrieve data for other users except the current user.
3. Buddy Matching:
    - Swipe feature for matching users based on preferences.
    - Send messages to buddies and retrieve message history.
    - Retrieve a user's buddies list.
    - Remove a buddy from the user's list.
    - User recommendations
4. Miscellaneous:
    - Retrieve the total number of registered users.

- Upload and display profile pictures.
- Delete user profiles.

System Requirements

You won't need to install anything to use the StudyBuddy backend API. The API can be accessed using a modern browser (developer's tools), the command line of an operating system and API testing tools (there are online tools that can be accessed through a browser, or you could download on your computer). You can also be accessed using the programming of your choice. This way, it can be integrated with a mobile or web application. Internet access is important to be able to make API calls.

The admin website is a basic Flask website which means it doesn't have robust system requirement. Although, the following device requirements are recommended:

1. Desktop Browsers:
    - Google Chrome (latest version)
    - Mozilla Firefox (latest version)
    - Safari (latest version)
    - Microsoft Edge (latest version)
2. Mobile Browsers:
    - Safari for iOS (latest version)
    - Google Chrome for Android (latest version)
3. Screen Resolution:
    - The website is responsive and adapt to various screen resolutions, including but not limited to:
        - Desktop: Minimum 1024x768 pixels resolution
        - Tablet: Minimum 768x1024 pixels resolution
        - Mobile: Minimum 320x480 pixels resolution
4. Internet Connection:
    - The website requires an active internet connection for data retrieval and interaction with the Flask application.
5. Browser Compatibility:
    - The website should be compatible with the latest versions of popular browsers mentioned above. It should also provide a consistent and usable

experience on older browser versions, although some visual enhancements may be limited.

6. Git and Github(Optional):
    - If you want to run the admin app locally on your computer, you need to have a Git on your computer. This will allow you to fork and clone to repository to your computer.

Installation Guide:

No installation is required to use the API or the admin website, you just need a browser, your CLI or an API testing tool.

The API testing tool recommend for the API is Postman. Postman can be used to write functional tests, integration tests, regression tests, and more. Postman's Node.js-based runtime contains support for common patterns and libraries that you can use to build tests quickly (Referenced here from Postman website).

The admin website can be accessed from the local host as well. You will need to have a minimum of Python 3.10 on your computer.

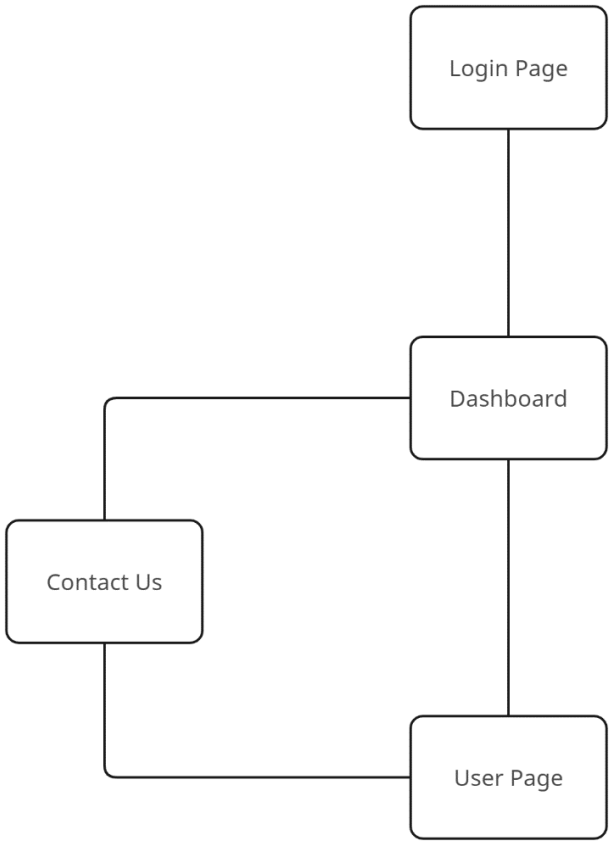
You will need to fork the repository of this project for that. Follow the instructions below:

- On GitHub.com, navigate to the mastertimisensei/StudyBuddyBE repository.
- In the top-right corner of the page, click **Fork**. (Fork a Repo, 2023)
- Under "Owner," select the dropdown menu and click an owner for the forked repository.
- By default, forks are named the same as their upstream repositories. Optionally, to further distinguish your fork, in the "Repository name" field, type a name.
- Click Create Fork.

After forking the repository, you will have to clone it to your local device. You can follow the instructions below to clone the repository.

- On GitHub.com, navigate to **your fork** of the StudyBuddyBE repository.
- Above the list of files, click **Code**. You will see a dropdown with a URL. Make sure you select the HTTPS or SSH option. Copy the URL.
- If you haven't already, install Git on your local device. You can download it from the official Git website and follow the installation instructions for your operating system.
- Open the terminal and navigate to the directory of your choice.
- Clone the github repo into that repository by running “git clone <<github URL >>
- Navigate to the the folder called Adminweb in the directory through terminal.
- Type “pip install -r requirement.txt” into your terminal.
- Type “python app.py” into your terminal.
- Click on the port you see. In this case it is 127.0.0.1/5000 on my browser.

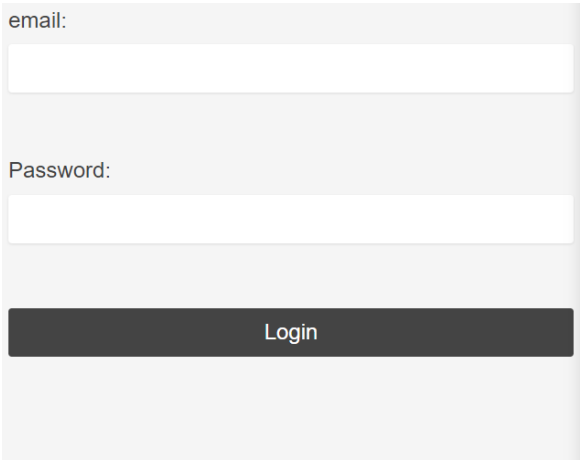
Navigation Map for the Admin Website



Login Page

Unfortunately, not every user can login into the admin website. Only selected users who are trusted and have admin status would be able to login. For the sake of this thesis, all users would be able to login to the admin application.

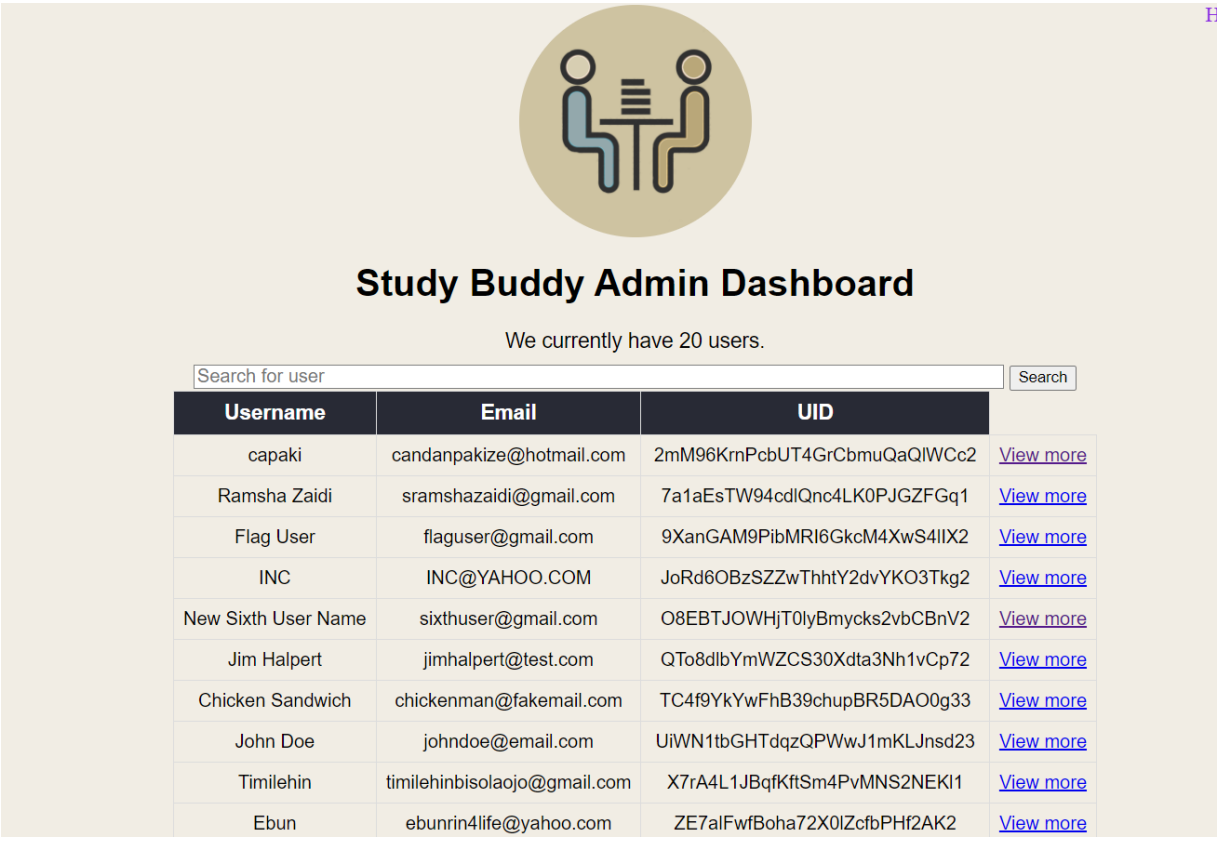
Login Page Screenshot



Dashboard

This shows us all the users in our application at that moment. We can see all the name, email and user ID of all the registered users of StudyBuddy. If “View More” is clicked, it would take us to that user page.

Dashboard Screenshot



User Page

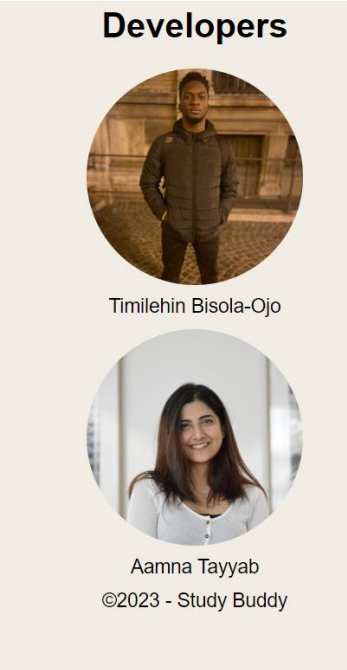
This page contains the users informations and profile picture. We can also see who their buddies are on this page.

User Page Screenshot

Contact Us

This page contains information about the creators of this project and how you can get in contact with them using various platforms.

Contact Us Screenshot



Endpoints

An API endpoint is a point at which an API -- the code that allows two software programs to communicate with each other -- connects with the software program. APIs work by sending *requests* for information from a web application or web server and receiving a *response*.

In other words, API endpoints are the specific digital location where requests for information are sent by one program to retrieve the digital resource that exists there. Endpoints specify where APIs can access resources and help guarantee the proper functioning of the incorporated software. An API's performance depends on its capacity to successfully communicate with API endpoints. (Contributor, 2023)

Endpoint Documentation can be found here:
https://documenter.getpostman.com/view/27253356/2s93m1bQoy

StudyBuddy Endpoint

POST Create User

https://studybuddy-backend.onrender.com/createUser

This endpoint creates a new user with their name, email, and password.

Parameters

email:string
name: string
password: string

Example Request

```
var axios = require('axios');
var data = JSON.stringify({
  "name": "Chicken Sandwich",
  "email": "chickenman@fakemail.com",
  "password": "password"
});

var config = {
  method: 'post',
maxBodyLength: Infinity,
  url: 'https://studybuddy-backend.onrender.com/createUser',
  headers: {
    'Content-Type': 'application/json'
  },
  data : data
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Response of the Example

```
User created successfully
```

POST Login

https://studybuddy-backend.onrender.com/signIn

This API allows a user to sign into their StudyBuddy account using the users email and password.

Parameters

username: string
password: string

Expected Response

Token and Flag in a json format

Example Request

```
var axios = require('axios');
var data = '{"email":"seconduser@gmail.com","password":"password"}';

var config = {
  method: 'post',
maxBodyLength: Infinity,
  url: 'https://studybuddy-backend.onrender.com/signIn',
  headers: { },
  data : data
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Response of Example Request


```
{
 "token":
"eyJhbGciOiJSUzI1NiIsImtpZCI6ImQwZTFkMjM5MDllNzZmZjRhNzJlZTA4ODUxOWM5M2JiOTg4ZjE4NDUiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vc3R1ZHktYnVkZHktYmFja2VuZC1kZTA4YSIsImF1ZCI6InN0dWR5LWJ1ZGR5LWJhY2tlbmQtZGUwOGEiLCJhdXRoX3RpbWUiOjE2ODQ2NzE2OTksInVzZXJfaWQiOiJTR25vU2s0TFNzUjJjVDRKcGt1ekVvczNZMjMzIiwic3ViIjoiU0dub1NrNExTc1IyY1Q0SnBrdXpFb3MzWTIzMyIsImlhdCI6MTY4NDY3MTY5OSwiZXhwIjoxNjg0Njc1Mjk5LCJlbWFpbCI6InNlY29uZHVzZXJAZ21haWwuY29tIiwiZW1haWxfdmVyaWZpZWQiOmZhbHNlLCJmaXJlYmFzZSI6eyJpZGVudGl0aWVzIjp7ImVtYWlsIjpbInNlY29uZHVzZXJAZ21haWwuY29tIl19LCJzaWduX2luX3Byb3ZpZGVyIjoicGFzc3dvcmQifX0.JbwCe23Ybbtf4T_evWsgD09zQqB0NgOu_CCD1DwMRyVf2SOaZZqmgYpKJSUgaA8PmNl5bkyPG52Xe4dnGMVYu5BdpitNi8toHTz62PPc1HqFpAcvtY2ObqIozPLeoXdMMgJLy0cd6V18C_fI7d9ZppiqQ_ePKv6k7oDqW8r5wLznIOpdi1lny7V8b9DgMfas4GfkkeS8KC0KeDq0UIV4eTz1-93CSIDLG5sGVmTVKpg0as9EUtJFseAAKwgLrJWn9t-yOqhdOh60A36SMMf9OlqJRp3I8p3hlUAVUjETfxNqjvMlvGa9XNwf3JEUHkYp1sIHrDJZNC6ZIvuyuEGXHQ",
  "flag": true
}
```


GET Count Users

https://studybuddy-backend.onrender.com/countUsers

This endpoint returns the number of registered users we have.

Example Request

```
var axios = require('axios');

var config = {
  method: 'get',
maxBodyLength: Infinity,
  url: 'https://studybuddy-backend.onrender.com/countUsers',
  headers: { }
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Example Response

```
{
  "users": 8
}
```

GET Getting All Users Data

https://studybuddy-backend.onrender.com/getAllUsersData

This endpoint returns all the registered users database information.

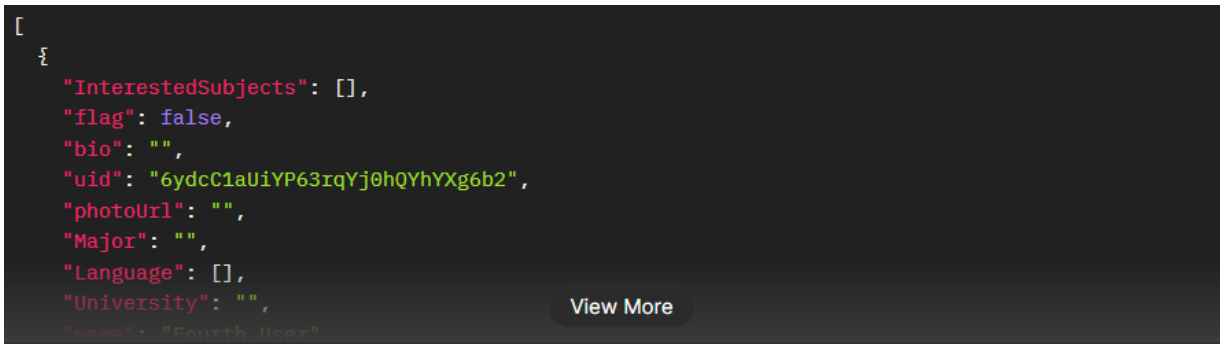
Example Request

```
var axios = require('axios');

var config = {
  method: 'get',
maxBodyLength: Infinity,
  url: 'https://studybuddy-backend.onrender.com/getAllUsersData',
  headers: { }
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Example Response



You can see the full output on the API documentation.

POST Get Buddies

https://studybuddy-backend.onrender.com/getBuddies

Gets the buddies of a particular user which we determine based on the token given to us.

Parameters

token: string

Response

Array of UID

Example Request

```
var axios = require('axios');
var data = '{"token":
"eyJhbGciOiJSUzI1NiIsImtpZCI6ImQwZTFkMjM5MDllNzZmZjRhNzJlZTA4ODUxOWM5M2JiOTg4ZjE4NDUi
LCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vc3R1ZHktYnVkZH
ktYmFja2VuZC1kZTA4YSIsImF1ZCI6InN0dWR5LWJ1ZGR5LWJhY2tlbmQtZGUwOGEiLCJhdXRoX3RpbWUiOjE
2ODQ2NzQwMDIsInVzZXJfaWQiOiJTR25vU2s0TFNzUjJjVDRKcGt1ekVvczNZMjMzIiwic3ViIjoiU0dub1Nr
NExTc1IyY1Q0SnBrdXpFb3MzWTIzMyIsImlhdCI6MTY4NDY3NDAwMiwiZXhwIjoxNjg0Njc3NjAyLCJlbWFpb
CI6InNlY29uZHVzZXJAZ21haWwuY29tIiwiZW1haWxfdmVyaWZpZWQiOmZhbHNlLCJmaXJlYmFzZSI6eyJpZG
VudGl0aWVzIjp7ImVtYWlsIjpbInNlY29uZHVzZXJAZ21haWwuY29tIl19LCJzaWduX2luX3Byb3ZpZGVyIjo
icGFzc3dvcmQifX0.MLleshtqt9aGfZPrRQgJxqEfsAYycIa-
nCH4G5oqKKd5JRLpFMtH0IrPnCRrtovz1J4DgkKNvIb5XfaHXRtYAHhDMkl3QD5jLB6JAz7KA11R_LvclHL5l
V6txN-OXKvmxU2Wx0wyAHRxZbG7bysXs63k78bYN18WY7gxW4KU-
DtLr83KfMEMTe1CMhTxc_TO1FfxDJ5E0AnFnJg_6hzow6bZwiDQWbzVJbSmciM8EYe1M24YmmQm9oJbM4RFDb
1ZhXV3F8l24B2eVfBfU0Zk9x7eT7BWR6zEIsidc1Gky0OOydPE2f18NhSW1ly_gq9N4pHTNSvHuD_7tNZQ2Xo
Ycw"}\r\n';

var config = {
  method: 'post',
maxBodyLength: Infinity,
  url: 'https://studybuddy-backend.onrender.com/getBuddies',
  headers: { },
  data : data
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Example Response

```
[
  "O8EBTJOWHjT0lyBmycks2vbCBnV2",
  "p5HdzdiUhAfXXtaTbgvI92Rx5Tp1"
]
```

Example of Request with bad/expired token

```
var axios = require('axios');
var data = '{"token":
"eyJhbGciOiJSUzI1NiIsImtpZCI6ImQwZTFkMjM5MDllNzZmZjRhNzJlZTA4ODUxOWM5M2JiOTg4ZjE4NDUi
LCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vc3R1ZHktYnVkZH
ktYmFja2VuZC1kZTA4YSIsImF1ZCI6InN0dWR5LWJ1ZGR5LWJhY2tlbmQtZGUwOGEiLCJhdXRoX3RpbWUiOjE
2ODQ2NzQwMDIsInVzZXJfaWQiOiJTR25vU2s0TFNzUjJjVDRKcGt1ekVvczNZMjMzIiwic3ViIjoiU0dub1Nr
NExTc1IyY1Q0SnBrdXpFb3MzWTIzMyIsImlhdCI6MTY4NDY3NDAwMiwiZXhwIjoxNjg0Njc3NjAyLCJlbWFpb
CI6InNlY29uZHVzZXJAZ21haWwuY29tIiwiZW1haWxfdmVyaWZpZWQiOmZhbHNlLCJmaXJlYmFzZSI6eyJpZG
VudGl0aWVzIjp7ImVtYWlsIjpbInNlY29uZHVzZXJAZ21haWwuY29tIl19LCJzaWduX2luX3Byb3ZpZGVyIjo
icGFzc3dvcmQifX0.MLleshtqt9aGfZPrRQgJxqEfsAYycIa-
nCH4G5oqKKd5JRLpFMtH0IrPnCRrtovz1J4DgkKNvIb5XfaHXRtYAHhDMkl3QD5jLB6JAz7KA11R_LvclHL5l
V6txN-
OXKvmxU2Wx0wyAHRxZbG7bysXs63k78bYN18WY7gxW4KUDtLr83KfMEMTe1CMhTxc_TO1FfxDJ5E0AnFnJg_6
hzow6bZwiDQWbzVJbSmciM8EYe1M24YmmQm9oJbM4RFDb1ZhXV3F8l24B2eVfBfU0Zk9x7eT7BWR6zEIsidc1
Gky0OOydPE2f18NhSW1ly_gq9N4pHTNSvHuD_7tNZQ2XoYcw"}\r\n';

var config = {
  method: 'post',
maxBodyLength: Infinity,
  url: 'https://studybuddy-backend.onrender.com/getBuddies',
  headers: { },
  data : data
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Response

```
Error getting buddies
```

GET Getting Other Buddies

https://studybuddy-backend.onrender.com/getAllOtherUsers2

This endpoint takes a token as an authorization, and provides all users whom the current user(represented as the token) has not swiped.

AUTHORIZATION Bearer Token

Token <token>

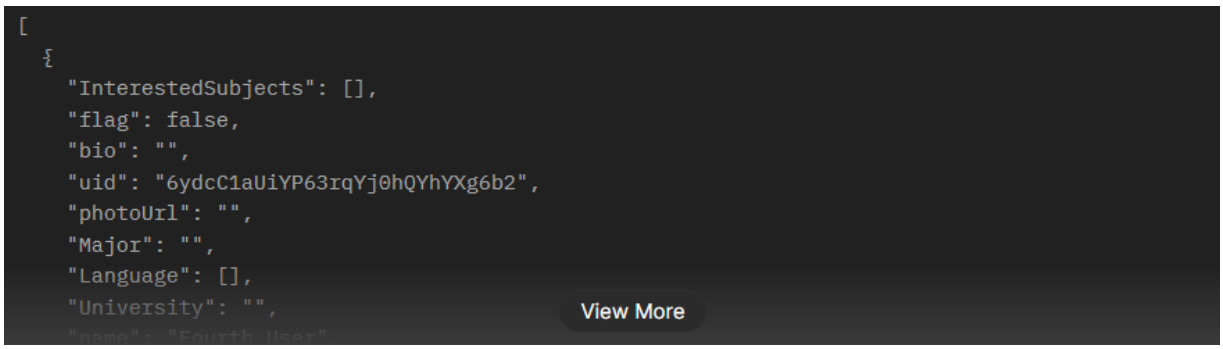
Example of Request (with Token)

```
var axios = require('axios');

var config = {
  method: 'get',
maxBodyLength: Infinity,
  url: 'https://studybuddy-backend.onrender.com/getAllOtherUsers2',
  headers: { }
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Example of Response (with Token)



To see the full response, you can check the API documentation.

POST Getting Users Data

https://studybuddy-backend.onrender.com/getUserData

This endpoint takes a user token as a request and returns the users data as a response.

token: string

Response with Status 200

User data in JSON format

Example Requests

```
var axios = require('axios');
var data =
'{"token":"eyJhbGciOiJSUzI1NiIsImtpZCI6ImQwZTFkMjM5MDllNzZmZjRhNzJlZTA4ODUxOWM5M2JiOT
g4ZjE4NDUiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vc3R1
ZHktYnVkZHktYmFja2VuZC1kZTA4YSIsImF1ZCI6InN0dWR5LWJ1ZGR5LWJhY2tlbmQtZGUwOGEiLCJhdXRoX
3RpbWUiOjE2ODQ2ODUzMDksInVzZXJfaWQiOiJiVGtOOUZ4Zk9xWFdNN0ZXbTBiVWNVNTU3czAzIiwic3ViIj
oiYlRrTjlGeGZPcVhXTTdGV20wYlVjVTU1N3MwMyIsImlhdCI6MTY4NDY4NTMwOSwiZXhwIjoxNjg0Njg4OTA
5LCJlbWFpbCI6ImphY2tzb25raW5nQGZha2VtYWlsLmNvbSIsImVtYWlsX3ZlcmlmaWVkIjpmYWxzZSwiZmly
ZWJhc2UiOnsiaWRlbnRpdGllcyI6eyJlbWFpbCI6WyJqYWNrc29ua2luZ0BmYWtlbWFpbC5jb20iXX0sInNpZ
25faW5fcHJvdmlkZXIiOiJwYXNzd29yZCJ9fQ.dsJZgCWBSYVa6QCbdlP2y0KMNYnDThMU_zy5xQcwF4HZ0Yq
xUOA_tAiE8QZU-omrTxuOaQMZxEr-
RKzG4QRTDITrhuL_uaaYbWtSGrnfWpL2Zt1IjKXu0Y6fNNnBG5FIOcV1XqzhrCaMo76_wIzK93Ai3YeXCVxn_
_pqzA4L1JYAsOWChJgZFU4ukgu0GHztzP3Hp-pkd6eYPWYJgWGRk8nQKdXamXlOQEq37bBvJyYi-
iXL6STMsAf6ftzyALeYdxclcp4lh7LkEdR4ERikULGkMxYePAFc-
J4XdH_7zqqrwkIsB3aiIXFeJeyNnWy3B7pnjxxAGjqKTUKXuyD4rQ"}';

var config = {
  method: 'post',
maxBodyLength: Infinity,
  url: 'https://studybuddy-backend.onrender.com/getUserData',
  headers: { },
  data : data
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Example Response


```
{
  "swipedThem": [],
  "InterestedSubjects": [],
  "flag": false,
  "bio": "",
  "swipedMe": [],
  "uid": "bTkN9FxfOqXWM7FWm0bUcU557s03",
  "University": "",
  "buddies": [],
  "email": "jacksonking@fakemail.com",
  "photoUrl": [
    "https://storage.googleapis.com/study-buddy-backend-
de08a.appspot.com/profilePics/bTkN9FxfOqXWM7FWm0bUcU557s03?GoogleAccessId=firebase-
adminsdk-gjxpe%40study-buddy-backend-
de08a.iam.gserviceaccount.com&Expires=16447017600&Signature=FCDD7eRpgkAWDFB%2Fm8EPzqz
0BhAmanVpdb%2B5SiVjKCxPyUCivwyCONGPJZpnTLRufxCzHvZ26AjdpoFvU7aCuaGSQRqGwzRM66TxjgYiOk
tbC2nJTMi0kScAJdAK%2FhweLo2FkrStNb6Xb5%2FASEFHnCjkL6zQrNBNfBp6XZky25YMEWvrHPb%2BLHFPW
pdsdS6t95Uwa%2Bw4wViFfiJbYZqjcHpLUxdizZRpRlyiAbguKPA6ysGx0RrBYD7I3cHJ03k35eMGnEkt8GO1
xQwJr0olxj0NhF%2BCfSDmE4W0M6BPkeuWX9ZCofL92ebfOXgIz6%2BsHmsFakN34rD4eS5Hd75Hug%3D%3D"
  ],
  "Major": "Computer Engineering",
  "Language": [
    "French"
  ],
  "name": "Jackson Kinf",
  "age": 19,
  "Location": "Budapest"
}
```

Request with Invalid Token

```
var axios = require('axios');
var data =
'{"token":"eyJhbGciOiJSUzI1NiIsImtpZCI6ImQwZTFkMjM5MDllNzZmZjRhNzJlZTA4ODUxOWM5M2JiOT
g4ZjE4NDUiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vc3R1
ZHktYnVkZHktYmFja2VuZC1kZTA4YSIsImF1ZCI6InN0dWR5LWJ1ZGR5LWJhY2tlbmQtZdsJZgCWBSYVa6QCb
dlP2y0KMNYnDThMU_zy5xQcwF4HZ0YqxUOA_tAiE8QZU-omrTxuOaQMZxEr-
RKzG4QRTDITrhuL_uaaYbWtSGrnfWpL2Zt1IjKXu0Y6fNNnBG5FIOcV1XqzhrCaMo76_wIzK93Ai3YeXCVxn_
_pqzA4L1JYAsOWChJgZFU4ukgu0GHtztP3Hp-pkd6eYPWYJgWGRk8nQKdXamXlOQEq37bBvJyYi-
iXL6STMsAf6ftzyALeYdxclcp4lh7LkEdR4ERikULGkMxYePAFc-
J4XdH_7zqqrwkIsB3aiIXFeJeyNnWy3B7pnjxxAGjqKTUKXuyD4rQ"}';

var config = {
  method: 'post',
maxBodyLength: Infinity,
  url: 'https://studybuddy-backend.onrender.com/getUserData',
  headers: { },
  data : data
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Response

```
Error getting user data
```

POST Setting Users Data

https://studybuddy-backend.onrender.com/setUserData

This POST request is used to update information about the user. It takes a token and the relevant information about the user in the request body

Parameters

token: string
name: string
Language: List/Array
age: Integer
University: string
Major: string
Location: string

University: string
bio: string
flag: Boolean
photoUrl: string(HTTP Link to an Image)

Example Request with Valid Token

```
var axios = require('axios');
var data = ' {\r\n
"token":"eyJhbGciOiJSUzI1NiIsImtpZCI6ImQwZTFkMjM5MDllNzZmZjRhNzJlZTA4ODUxOWM5M2JiOTg4
ZjE4NDUiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vc3R1ZH
ktYnVkZHktYmFja2VuZC1kZTA4YSIsImF1ZCI6InN0dWR5LWJ1ZGR5LWJhY2tlbmQtZGUwOGEiLCJhdXRoX3R
pbWUiOjE2ODQ2ODUzMDksInVzZXJfaWQiOiJiVGtOOUZ4Zk9xWFdNN0ZXbTBiVWNVNTU3czAzIiwic3ViIjoi
YlRrTjlGeGZPcVhXTTdGV20wYlVjVTU1N3MwMyIsImlhdCI6MTY4NDY4NTMwOSwiZXhwIjoxNjg0Njg4OTA5L
CJlbWFpbCI6ImphY2tzb25raW5nQGZha2VtYWlsLmNvbSIsImVtYWlsX3ZlcmlmaWVkIjpmYWxzZSwiZmlyZW
Jhc2UiOnsiaWRlbnRpdGllcyI6eyJlbWFpbCI6WyJqYWNrc29ua2luZ0BmYWtlbWFpbC5jb20iXX0sInNpZ25
faW5fcHJvdmlkZXIiOiJwYXNzd29yZCJ9fQ.dsJZgCWBSYVa6QCbdlP2y0KMNYnDThMU_zy5xQcwF4HZ0YqxU
OA_tAiE8QZU-omrTxuOaQMZxEr-
RKzG4QRTDITrhuL_uaaYbWtSGrnfWpL2Zt1IjKXu0Y6fNNnBG5FIOcV1XqzhrCaMo76_wIzK93Ai3YeXCVxn_
_pqzA4L1JYAsOWChJgZFU4ukgu0GHtztP3Hp-pkd6eYPWYJgWGRk8nQKdXamXlOQEq37bBvJyYi-
iXL6STMsAf6ftzyALeYdxclcp4lh7LkEdR4ERikULGkMxYePAFc-
J4XdH_7zqqrwkIsB3aiIXFeJeyNnWy3B7pnjxxAGjqKTUKXuyD4rQ",\r\n    "name":"Jackson
Kinf",\r\n    "age":19,\r\n    "Language":["French"],\r\n    "Major":"Computer
Engineering",\r\n    "InterestedSubjects":[],\r\n    "Location":"Budapest",\r\n
"University":"",\r\n    "bio":"",\r\n
"photoUrl":"https://source.unsplash.com/user/c_v_r",\r\n    "flag":false\r\n  }';

var config = {
  method: 'post',
maxBodyLength: Infinity,
  url: 'https://studybuddy-backend.onrender.com/setUserData',
  headers: { },
  data : data
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Response

```
User data updated successfully
```

POST Getting Users Data

https://studybuddy-backend.onrender.com/getUserData

This endpoint takes a user token as a request and returns the users data as a response.

token: string

Response

User data in JSON format

Example Request with Valid Token

```
var axios = require('axios');
var data =
'{"token":"eyJhbGciOiJSUzI1NiIsImtpZCI6ImQwZTFkMjM5MDllNzZmZjRhNzJlZTA4ODUxOWM5M2JiOT
g4ZjE4NDUiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vc3R1
ZHktYnVkZHktYmFja2VuZC1kZTA4YSIsImF1ZCI6InN0dWR5LWJ1ZGR5LWJhY2tlbmQtZGUwOGEiLCJhdXRoX
3RpbWUiOjE2ODQ2ODUzMDksInVzZXJfaWQiOiJiVGtOOUZ4Zk9xWFdNN0ZXbTBiVWNVNTU3czAzIiwic3ViIj
oiY1RrTjlGeGZPcVhXTTdGV20wYlVjVTU1N3MwMyIsImlhdCI6MTY4NDY4NTMwOSwiZXhwIjoxNjg0Njg4OTA
5LCJlbWFpbCI6ImphY2tzb25raW5nQGZha2VtYWlsLmNvbSIsImVtYWlsX3ZlcmlmaWVkIjpmYWxzZSwiZmly
ZWJhc2UiOnsiaWRlbnRpdGllcyI6eyJlbWFpbCI6WyJqYWNrc29ua2luZ0BmYWtlbWFpbC5jb20iXX0sInNpZ
25faW5fcHJvdmlkZXIiOiJwYXNzd29yZCJ9fQ.dsJZgCWBSYVa6QCbdlP2y0KMNYnDThMU_zy5xQcwF4HZ0Yq
xUOA_tAiE8QZU-omrTxuOaQMZxEr-
RKzG4QRTDITrhuL_uaaYbWtSGrnfWpL2Zt1IjKXu0Y6fNNnBG5FIOcV1XqzhrCaMo76_wIzK93Ai3YeXCVxn_
_pqzA4L1JYAsOWChJgZFU4ukgu0GHtztP3Hp-pkd6eYPWYJgWGRk8nQKdXamXlOQEq37bBvJyYi-
iXL6STMsAf6ftzyALeYdxclcp4lh7LkEdR4ERikULGkMxYePAFc-
J4XdH_7zqqrwkIsB3aiIXFeJeyNnWy3B7pnjxxAGjqKTUKXuyD4rQ"}';

var config = {
  method: 'post',
maxBodyLength: Infinity,
  url: 'https://studybuddy-backend.onrender.com/getUserData',
  headers: { },
  data : data
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Response


```
{
  "swipedThem": [],
  "InterestedSubjects": [],
  "flag": false,
  "bio": "",
  "swipedMe": [],
  "uid": "bTkN9FxfOqXWM7FWm0bUcU557s03",
  "University": "",
  "buddies": [],
  "email": "jacksonking@fakemail.com",
  "photoUrl": [
    "https://storage.googleapis.com/study-buddy-backend-de08a.appspot.com/profilePics/bTkN9FxfOqXWM7FWm0bUcU5
  ],
  "Major": "Computer Engineering",
  "Language": [
    "French"
  ],
  "name": "Jackson Kinf",
  "age": 19,
  "Location": "Budapest"
}
```


POST Sending a Message

https://studybuddy-backend.onrender.com/sendMessage

This endpoint is used to send messages between users who are buddies. We need the token of the sender and the email of the buddy they are sending the message to.

Parameters

token: string
buddy_email: string

Example of Request

```
var axios = require('axios');
var data =
'{"token":"eyJhbGciOiJSUzI1NiIsImtpZCI6ImQwZTFkMjM5MDllNzZmZjRhNzJlZTA4ODUxOWM5M2JiOT
g4ZjE4NDUiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vc3R1
ZHktYnVkZHktYmFja2VuZC1kZTA4YSIsImF1ZCI6InN0dWR5LWJ1ZGR5LWJhY2tlbmQtZGUwOGEiLCJhdXRoX
3RpbWUiOjE2ODQ3MDkzNDgsInVzZXJfaWQiOiJUQzRmOVlrWXdGaEIzOWNodXBCUjVEQU8wZzMzIiwic3ViIj
oiVEM0ZjlZa1l3RmhCMzljaHVwQlI1REFPMGczMyIsImlhdCI6MTY4NDcwOTM0OCwiZXhwIjoxNjg0NzEyOTQ
4LCJlbWFpbCI6ImNoaWNrZW5tYW5AZmFrZW1haWwuY29tIiwiZW1haWxfdmVyaWZpZWQiOmZhbHNlLCJmaXJl
YmFzZSI6eyJpZGVudGl0aWVzIjp7ImVtYWlsIjpbImNoaWNrZW5tYW5AZmFrZW1haWwuY29tIl19LCJzaWduX
2luX3Byb3ZpZGVyIjoicGFzc3dvcmQifX0.iSyhhSJKo6YG8QnUOh0dAfEiAvxtSIWmuzz33NkAMvyh7FS594
8TFP7C-
rec_f0GkcBbyW92Tepq8zSHlwMmTGaoyH2jKp2uYFIH8R5_g5k0chlv9k7_pozrhb026kkR9NkhHbGQV7B0qx
-
DeFSSDCj43whX0EuASgFOWCxFBBrhBk6J9gbvy9BQvqx2LRXgRHbB0dkr_N8ZVxAajenT2qJMIu0gYm6vVNsU
2RsL0VCG3wMErTNtYDQ29eUF-
cH_l0PV6dBmPYo1QurWdlDB1SElmcTiATOleHFis3UyfxeOkl41Vzzi75HZpez_JB4Ds8hSRC5Lo6m91LE5gR
-9ow", "buddy_email":"newuser@gmail.com","message":"Hey"}';

var config = {
  method: 'post',
maxBodyLength: Infinity,
  url: 'https://studybuddy-backend.onrender.com/sendMessage',
  headers: { },
  data : data
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Response

```
Message sent successfully
```

POST Get Messages

https://studybuddy-backend.onrender.com/getMessages2

This endpoint gives us the messages of a particular chatId. The chatId is a unique identifier for messages between two users.

Parameters

chatId: string

Example of Request

```
var axios = require('axios');
var data = '{"chatId":"Y5vQRrXceuccL3y9EZcf"}';

var config = {
  method: 'post',
maxBodyLength: Infinity,
  url: 'https://studybuddy-backend.onrender.com/createUser',
  headers: { },
  data : data
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Response

```
[
  {
    "time": "2023-05-16 02:21:48",
    "message": "First Message to fifth user should go through",
    "sender": "seconduser@gmail.com"
  }
]
```

POST Swiping a user

https://studybuddy-backend.onrender.com/swipe

This endpoint is used to represent when a user swipes on a user.

Parameters

token: string
buddy_email: string
swipe: Boolean

```
var axios = require('axios');
var data = '{"token":
"eyJhbGciOiJSUzI1NiIsImtpZCI6ImQwZTFkMjM5MDllNzZmZjRhNzJlZTA4ODUxOWM5M2JiOTg4ZjE4NDUi
LCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vc3R1ZHktYnVkZH
ktYmFja2VuZC1kZTA4YSIsImF1ZCI6InN0dWR5LWJ1ZGR5LWJhY2tlbmQtZGUwOGEiLCJhdXRoX3RpbWUiOjE
2ODQ3MDkzNDgsInVzZXJfaWQiOiJUQzRmOVlrWXdGaEIzOWNodXBCUjVEQU8wZzMzIiwic3ViIjoiVEM0ZjlZ
a1l3RmhCMzljaHVwQlI1REFPMGczMyIsImlhdCI6MTY4NDcwOTM0OCwiZXhwIjoxNjg0NzEyOTQ4LCJlbWFpb
CI6ImNoaWNrZW5tYW5AZmFrZW1haWwuY29tIiwiZW1haWxfdmVyaWZpZWQiOmZhbHNlLCJmaXJlYmFzZSI6ey
JpZGVudGl0aWVzIjp7ImVtYWlsIjpbImNoaWNrZW5tYW5AZmFrZW1haWwuY29tIl19LCJzaWduX2luX3Byb3Z
pZGVyIjoicGFzc3dvcmQifX0.iSyhhSJKo6YG8QnUOh0dAfEiAvxtSIWmuzz33NkAMvyh7FS5948TFP7C-
rec_f0GkcBbyW92Tepq8zSHlwMmTGaoyH2jKp2uYFIH8R5_g5k0chlv9k7_pozrhb026kkR9NkhHbGQV7B0qx
-
DeFSSDCj43whX0EuASgFOWCxFBBrhBk6J9gbvy9BQvqx2LRXgRHbB0dkr_N8ZVxAajenT2qJMIu0gYm6vVNsU
2RsL0VCG3wMErTNtYDQ29eUF-
cH_l0PV6dBmPYo1QurWdlDB1SElmcTiATOleHFis3UyfxeOkl41Vzzi75HZpez_JB4Ds8hSRC5Lo6m91LE5gR
-9ow","buddy_email": "newuser@gmail.com", "swipe": true }';

var config = {
  method: 'post',
maxBodyLength: Infinity,
  url: 'https://studybuddy-backend.onrender.com/swipe',
  headers: { },
  data : data
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Response

```
Swipe sent successfully
```

POST Delete User

https://studybuddy-backend.onrender.com/deleteProfile

This endpoint deletes a particular user from the StudyBuddy. This is the equivalence of some deleting their account.

Parameters

token: string

```
var axios = require('axios');
var data =
'{"token":"eyJhbGciOiJSUzI1NiIsImtpZCI6ImQwZTFkMjM5MDllNzZmZjRhNzJlZTA4ODUxOWM5M2JiOT
g4ZjE4NDUiLCJ0eXAiOiJKV1QifQ.eyJpc3MiOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vc3R1
ZHktYnVkZHktYmFja2VuZC1kZTA4YSIsImF1ZCI6InN0dWR5LWJ1ZGR5LWJhY2tlbmQtZGUwOGEiLCJhdXRoX
3RpbWUiOjE2ODQ2ODk2NzUsInVzZXJfaWQiOiJiVGtOOUZ4Zk9xWFdNN0ZXbTBiVWNVNTU3czAzIiwic3ViIj
oiYlRrTjlGeGZPcVhXTTdGV20wYlVjVTU1N3MwMyIsImlhdCI6MTY4NDY4OTY3NSwiZXhwIjoxNjg0NjkzMjc
1LCJlbWFpbCI6ImphY2tzb25raW5nQGZha2VtYWlsLmNvbSIsImVtYWlsX3Zlcml"}\r\n';

var config = {
  method: 'post',
maxBodyLength: Infinity,
  url: 'https://studybuddy-backend.onrender.com/deleteProfile',
  headers: { },
  data : data
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Response

```
Profile deleted successfully
```

POST Show Recommendation Score

https://admin-web-6a8l.onrender.com/showRecommendationScore

This endpoint takes gives us the recommendation score for two users studying together.

Parameters

uid: string
buddy_uid: string

Example of a Request

```
var axios = require('axios');
var data = '{"uid":"TC4f9YkYwFhB39chupBR5DAO0g33",
"buddy_uid":"SGnoSk4LSsR2cT4JpkuzEos3Y233"}';

var config = {
  method: 'post',
maxBodyLength: Infinity,
  url: 'https://admin-web-6a8l.onrender.com/showRecommendationScore',
  headers: { },
  data : data
};

axios(config)
.then(function (response) {
  console.log(JSON.stringify(response.data));
})
.catch(function (error) {
  console.log(error);
});
```

Response

```
"{
    score: 3.5, reason: 'Although the subjects similarity is important, the cosine similarity between
Person 1 and Person 2 is very low, which suggests that they have very different study habits and
preferences. In addition, their location is not specified, which makes it difficult to determine whether
they would be able to meet up to study. Finally, we don't have information on their universities and
majors, which could be important factors in determining whether they would be a good match for studying
together.'
    }"
```

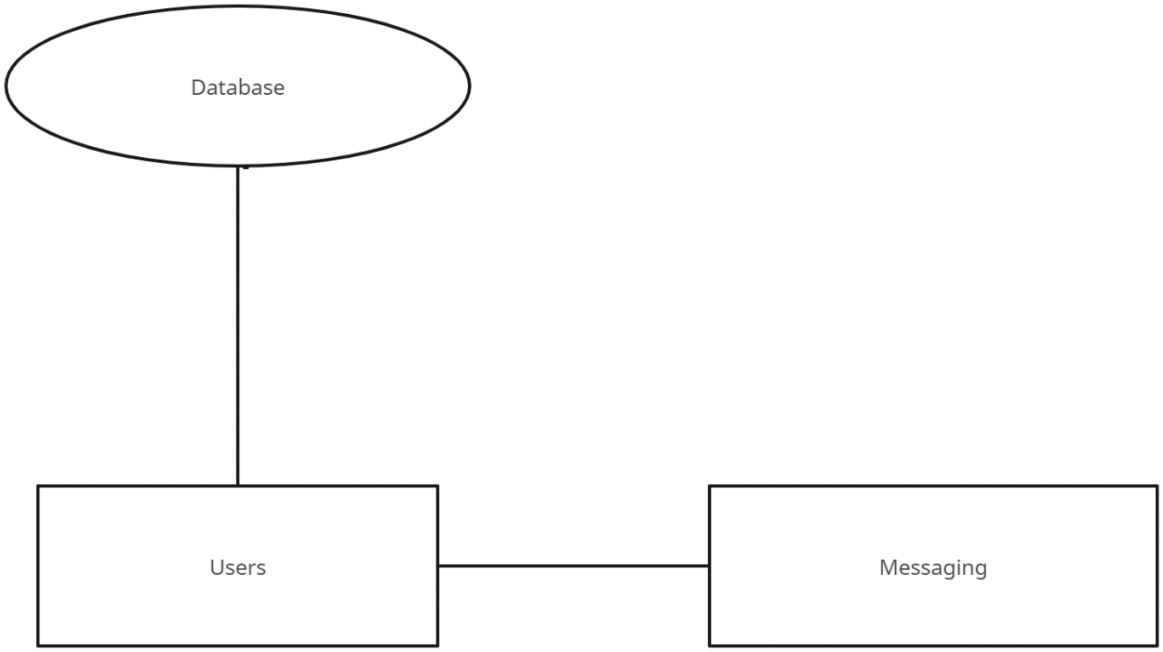
Developers Documentation

Design Plan

Firstly, we are going to need a way to database to store user's data. The database doesn't need to be very complicated because the user's actions within the application are not going to be very complex.

The database should also be able to store messages between users. The users would store a reference to the messages or should be linked to the messages in some kind of way. A simple visualization of what the database would look like the figure below.

Design Diagram



I am also planning to create a simple Admin website for simple management of users. This should not be available to everyone, only people with certain permission/privileges should have access to it. It will also contain CSS styling.

For the recommendation, we are going to collect some data from the users, and we are going to use an AI model to check if the two users are a match or not. We are not going to use all the user's data as some of it may not be useful in making recommendations.

There is a frontend to StudyBuddy and we will need a way to connect to it to the backend. We will also need to ensure that the backend will be able to stand on its own and changes to the frontend will not affect it.

Required Developers Tools

The backend, admin website and Machine Learning can be developed using any text editor or IDE. Visual Studio Code is the preferred text editor because it is lightweight and has a large variety of installable extensions in its marketplace which can be helpful in developing the backend and the admin website. All required packages can be installed using NPM.

For hosting and deploying our website and backend, we are going to use “Render.com”. It is easy to use, easy to configure and very low cost. You don’t have to download anything to use Render.com, all you just need to do is to connect your Github project to it.

For testing and retrieval purposes, we are going to use Postman. “Postman is an API platform for building and using APIs. Postman simplifies each phase of the API lifecycle, allowing you to design better APIs faster” (Postman, 2023). You can use either the online version or download the software on your computer. For the development of this, I used the software version as the online version can’t be used for testing the api on the Local Host.

Github or Gitlab is also required in the development of this project. Render.com requires you to connect your Github(or Gitlab) project to it. For this project, I am using Github and I have also downloaded Github Desktop which is software offered by Github. It drastically reduces the need to use CLI.

Backend Server

I had to create one backend server and an admin app. For creating the backend server, I used Express.js which is a backend web application framework popular used for creating RESTful with Node.js. It has a large community of developers, and it is completely open source.

You can choose any the library or module to make HTTP requests to the backend. This will depend on your needs and the language used as well as other factors like speed and complexity of the project. The backend will use firebase authentication for the authentication of the users and Cloud Firestore for storing and retrieving user data. Firebase also offers direct services to client-side applications. However, Firebase alongside Node.js becomes a powerful tool. This gives you access. You may harness the full capabilities of Node.js and construct additional server-side functionality to augment your Firebase services by using it alongside Node.js. Node.js would give us access additional libraries and modules.

Other reasons to choose Node JS alongside Firebase include:

• Support of real-time operations, instant synchronization of the data in the application.

• Firebase Storage

• It offers Analytics tools

• It is free to use

• Custom Security rules

Firebase has two types of databases. The Realtime Database and the Cloud Firestore. We are going to use the Cloud Firestore.

Cloud Firestore provides a more flexible and scalable data model, allowing for more complex and nested data structures, making it easier to express and query data. Cloud Firestore provides powerful querying capabilities to retrieve data based on multiple conditions, perform compound queries, and order and limit results without compromising performance. Most importantly, Cloud Firestore scales automatically to handle high read/write loads and distributes data across multiple servers and regions for high availability and low latency making it suitable for small/medium applications.

Using Node.js, we will be able to retrieve and edit users' data, using both POST and GET requests in the backend. User data in the Firestore would include personal data given to us by the user(including messages), public data from Firebase Authentication and their actions within the application.

For the authentication, we are going to use Firebase Authentication. Firebase Authentication provides a simple and easy-to-use API to integrate popular authentication methods such as email/password, social sign-in, phone number authentication, and more. It handles the authentication process securely, provides features for managing user identities, integrates with third-party authentication providers, provides user management and permissions, and integrates with other Firebase services. It is built on Google's infrastructure, providing scalability and high availability, and can handle authentication requests from millions of users simultaneously. Although we aren't going to use most of these Authentication services, it still benefits us to use it as it is faster than retrieving data from the database. We don't have to store the password of our users (or an encrypted version) in the database as Firebase Authentication takes care of that for us.

The Admin Website

This is a relatively simple web application to make using the Flask app, we are going to have 4 major pages.

The Login page is meant to be able to sign in authorized users. It should be able to handle cases of wrong email/password and handles cases of unauthorized users trying to login with their details.

The Dashboard page should provide basic information about the users. Through this page we should be able navigate each user's data page. It would also contain a search bar which can be used to search for users based on their name or email address.

The User page should provide more in-depth information about the user. We can see more information about the user like when they created their account, their age, and their current friends(buddies).

The Contact page should provide information about the developers and the development process of the web application.

The admin website would use the Node.js backend that we developed for retrieving and editing user data. The website will also contain CSS designs to make the pages look better. I will use Render.com for the hosting and deployment of the website as it has options for hosting Flask apps.

Recommendations

We should be able to compare two users and determine whether they would be a good fit studying together. It should do this by giving a recommendation score as well as the reasoning behind the recommendation. We will have to use an external Natural Language Processor model to decide this based on the users' data.

Some of the user data would be gotten straight out of the database without a change, while some require some form of transformation to make the recommendation more efficient.

We should also have a way for people access this function either through either a POST request to an API or through the Admin website. Using wouldn't require any form of login or user authentication. We will just require the users' data for processing.

Prompting with the GBT Model

We are going to use an existing Machine Learning model for our recommendations. We are going to use Open AI's GBT mdel for this. OpenAI's GPT 3.5 Turbo model is ML model designed to generate human-like responses based on given prompts and instructions.

A prompt is a specific input provided to a language model that sets the context and guides the model's response. To utilize the model effectively, we need to structure our prompt with clear instructions, provide any necessary context, and specify the desired format or output. We can also provide any supplementary information or guidelines that help the model make informed recommendations.

By utilizing the power of the GPT model and providing a well-structured prompt, we can leverage its natural language processing capabilities to generate accurate, relevant, and personalized recommendations for our users.

The detailed description of the used methods

Creating users

Our function to create a new user takes in a name, email, and password as input parameter. To create a user, we are first going to ensure that email and passwords are in a valid format. If this is true, create a user in the authentication and then add him to the User database. We set the user's name and email to what was given to us as a parameter. We also get the user's uid and store it in the database.

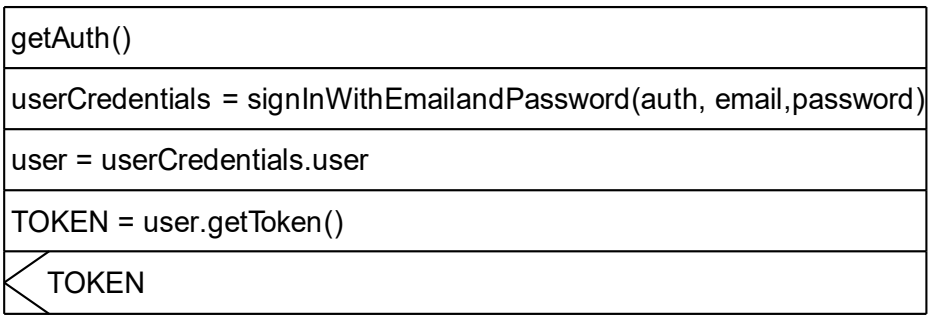
createUser(name, email, password)

email and password are valid: *true*	*false*
userRecord = auth.createUser(email, password)	ERROR
userRef = db.users.createReference(email)	
userDoc is a new user document	
userRef.email,userRef.name,userRef.uid = email,name, userRecord.uid	
userDoc.photoUrl,userDoc.Major,userDoc.University = ""	
userDoc.age, userDoc.bio, userDoc.Location = ""	
userDoc.Language,userDoc.swipedMe,userDoc.swipedThem,userDoc.buddies = []	
userDoc.flag = False	
userRef.set(userDoc)	

Signing in a User

To sign in a user, we are going to get authentication object (it will be given to us) from Firebase. We are then going to accept the email and password from the input parameters. If the login is successful, Firebase authentication will give us the user's credentials. We are then able to get a token from the user's credentials using the getToken() method. This token is useful as it is used to verify if the request is sent from the user or not.

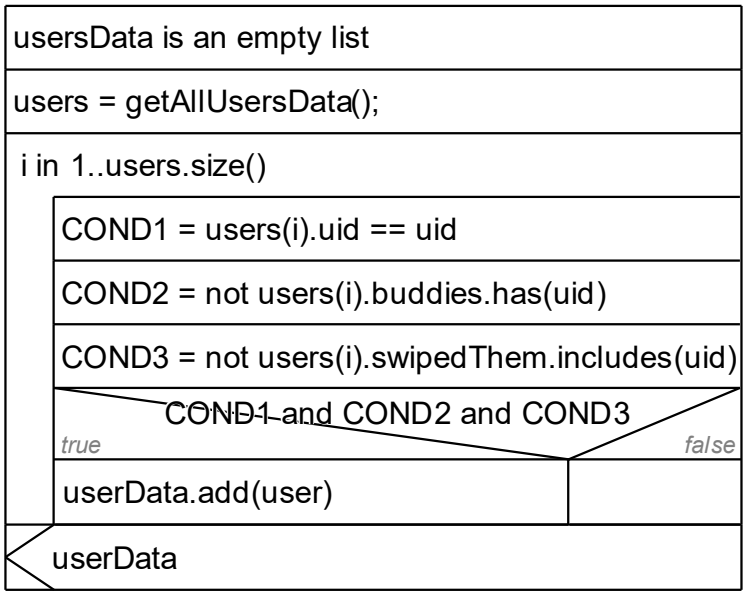
SignIn(email,password)



Getting users to Swipe

This function returns a list of users that a particular user can “swipe”. The list of users returned would not include this user, other users who the user has swiped before and his buddies. It only takes one parameter which is the uid of the user that we want to generate this list for.

getOtherUsers(user_uid)



Messaging between Buddies

Send Messages

This function allows users to send messages to one of their buddies. As input parameters, the function takes their email, their friends' emails, and the message they

want to send. We begin by locating the message's reference within the user's data (since the user's data is a map, we can locate the reference using our buddy's email address). After we find the messageId, we locate it within the message collection and create a new message reference inside the subcollection. The sender of the message, the current time, and the message text are retrieved and copied into a new document and placed within the reference.

messageBuddy (userEmail, buddyEmail, text)

```
userDoc = db.users.get(userEmail)
buddyDoc = db.users.get(buddyEmail)
messageId = userDoc.messages[buddyEmail]
messageDocRef = db.message.get(messageId).createReference()
newMessage is a new Message
newMessage.sender = userEmail
newMessage.time = Time.now()
newMessage.message = text
messageDocRef.set(newMessage)
```

Getting Messages between users

This function retrieves the messages between two users based on the chatId as a JSON String. It also ensures that messages sent are sorted by time.

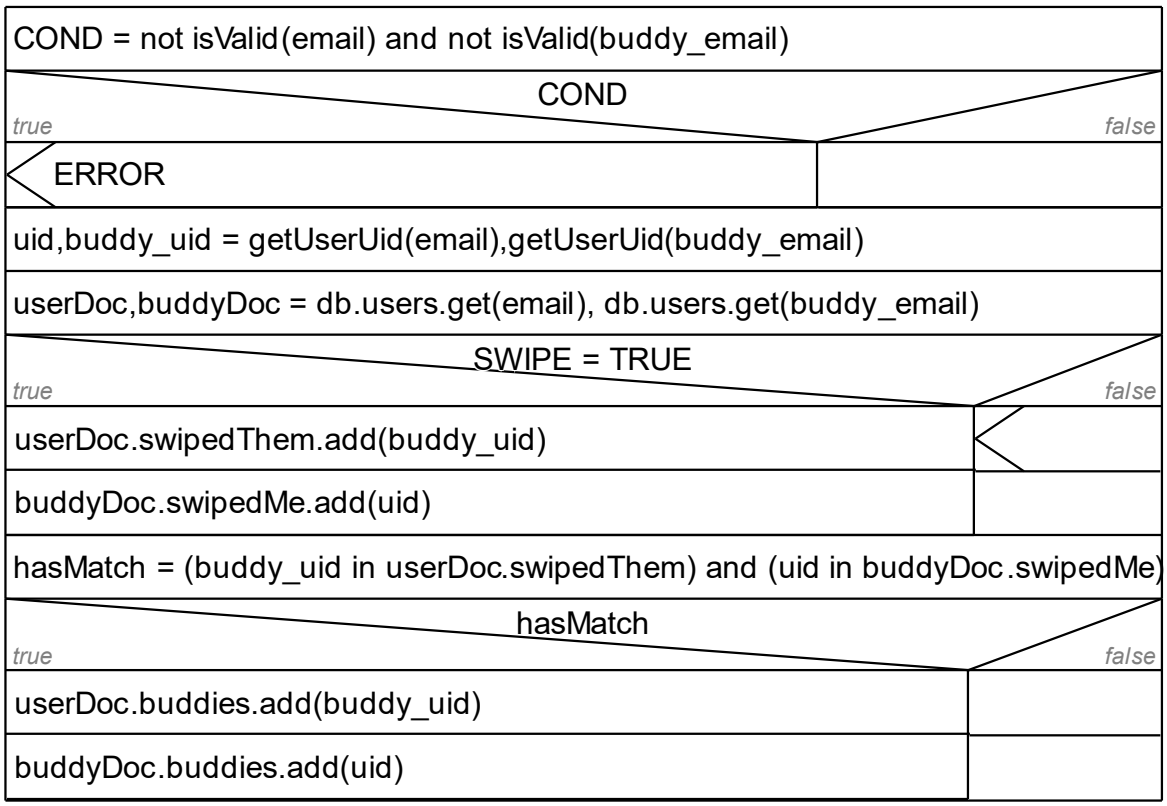
getMessages(chatId)

```
MessageDoc = TIME_SORT(db.messages.get(chatId))
messages = "["
 message in MessageDoc
    messages = CONCAT(messages,message)
CONCAT(messages,"]")
 messages
```

Swiping and Matching Users

This function simulates the swiping functionality of an app. If a user “swipes right” on another user, the other user is added to the user’s swipedThem list and the user is added to the other user’s swipedMe list. If the user has been swiped by the other user, they are a match. This means that they would both be added to their respective buddy lists.

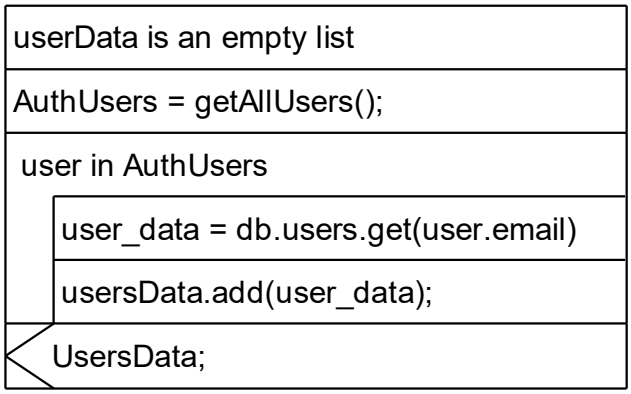
swipe(email, buddy_email)



Getting All Users Data

This function returns all the users’ data in our database. It is only used within the admin website.

getAllUserData()



Other Methods/Functions

Getting All Users Authentication Data

This function is similar to getAllUsersData because it returns information about the all the users. The major difference between them is that this functio returns authentication data about the user while getUsersData returns each user's data from the Firestore database.

Count Users

This function simply returns the number of Users in the authentication and the database.

Update Email Address

This function changes the email of a particular user in both the authentication as well as the database.

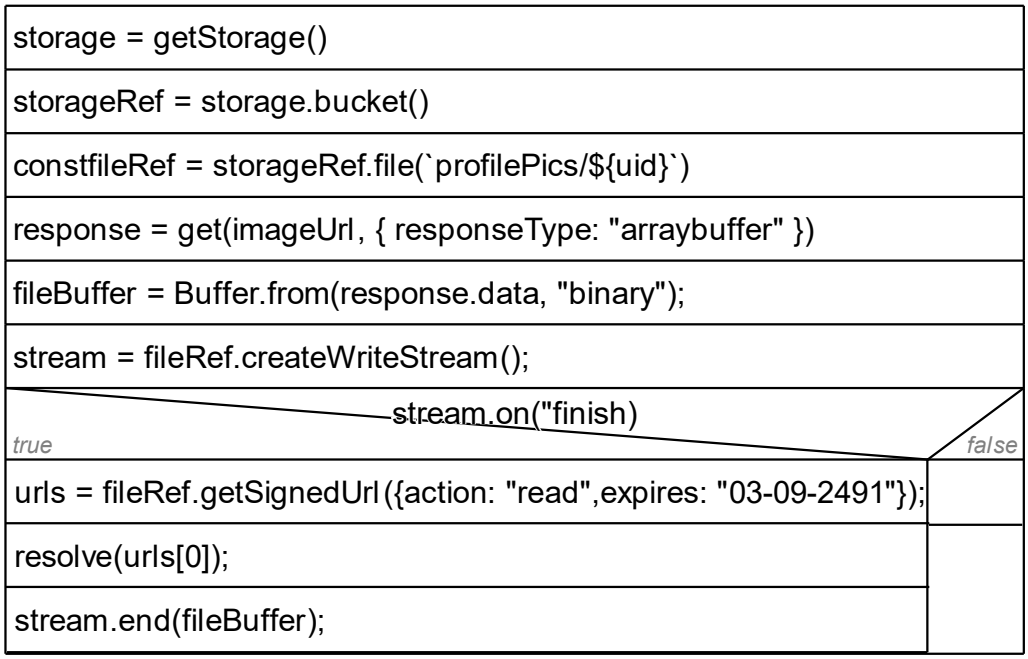
Getting Users UID

This function gets the users.

Upload Profile Picture

This can be used to upload pictures to the cloud Storage, it takes a URL and then takes the image in the URL into the storage.

uploadProfilePicture(uid)



Showing the Profile Picture

There is a function to show the user's data based on their UID. It shows the uploaded profile picture that the user chooses while filing in their data.

Removing a Buddy

This function can unmatch two users who are already buddies. It does this by removing the users from each other's buddy list. It takes the user's email address and the buddy's email address that he wants to remove.

Recommendation Algorithm

Getting the Cosine similarity of the bios

Bios are usually long texts that are hard to work with. This function calculates the cosine similarity between two user's bios. Firstly, we clean the text by removing punctuation. Then, we obtain the word frequencies for both bios. We then calculate the dot product and magnitudes of the word frequencies to determine the cosine similarity.

Generating Prompt

This function takes two users and generates a prompt for providing a study recommendation score between 1 and 10. It includes instructions, considerations, the individuals' information, cosine similarity, and a response format. The prompt is returned as the output of the function.

Generate the response

This function is a Python function that uses the OpenAI model to generate a response based on our generated prompt. It takes a prompt parameter and constructs a system message that introduces the context of the conversation, including the instruction for the user to provide a study recommendation score. The response from the API is extracted and returned as the output of the function.

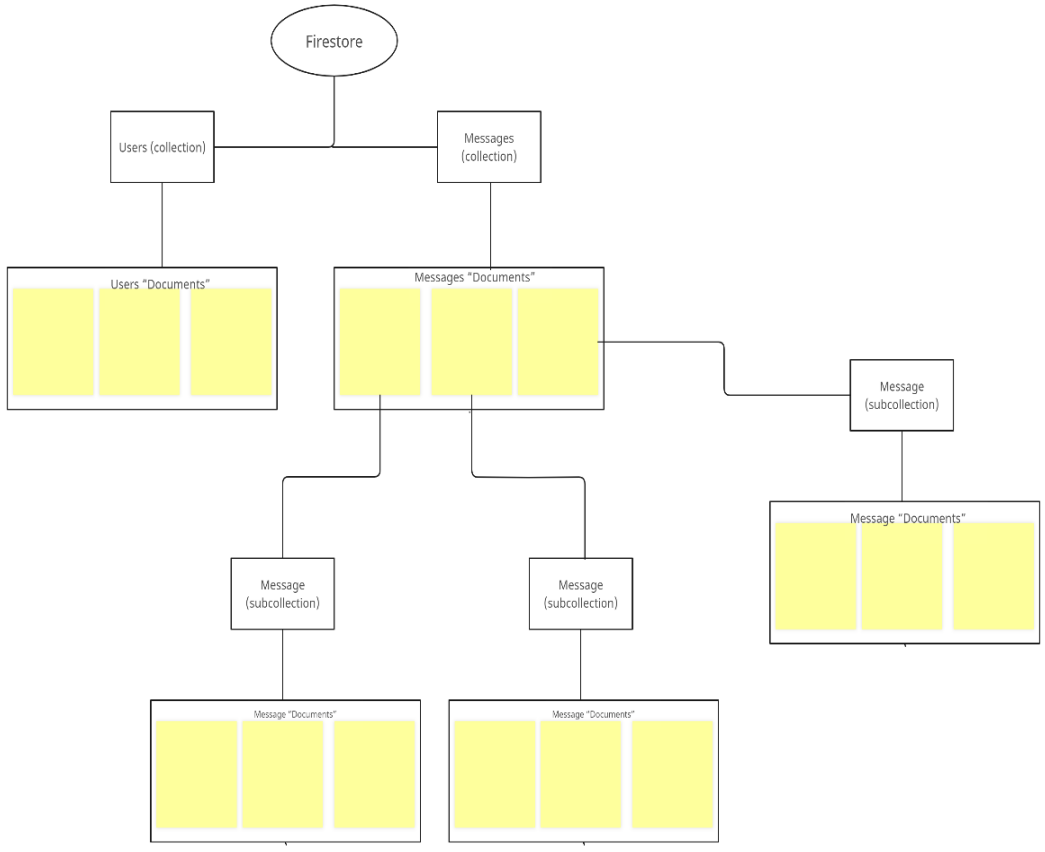
The Database

We are using Firestore to store most of the data in the backend. Firestore is a NoSQL database hence it doesn't store its data in tables but in collections. We are going to create two different collections:

- Users
- Messages

The diagram below shows how the database structure.

Database Diagram



User collection is where we store information about each user. Each user's data is stored in a document named after their email address in the User collection. This includes their personal details like their name, age, email address, location, the languages they speak, the university they go to, university major and the subjects they want to study.

In addition to that, we also store the other users that our user wants to become "buddies" with and those who want to become "buddies" with our user. We would also store their current buddies and a reference to the messages with their "buddies." Users can also decide to add a Bio to their profile.

Flag is used to determine whether the document is sufficiently filled or not.

Structure of sample user document.

Name: Person Alpha (string)

Email: alphaperson@fakemailcamp.com (string)

Interested Subject: [Subject A, Subject B, Subject C](array of string)

Language : [Language A, Language B](array of array)

Location : Equator(string)

University: University B(string)

Age: 23(number)

flag: false (Boolean)

User ID: << Person ID user id >> (string) (unique) (assigned by Firestore)

SwipedMe : [<<Buddy1 user id >>, << Person1 user id >> , <<Buddy2 Beta user id>>](array of string)

SwipedThem: [<<Buddy1 user id >>, << Person2 user id >> , <<Buddy2 Beta user id>>](array of string)

Buddies: [<<Buddy1 user id >>, <<Buddy2 Beta user id>>](array of string)

Bio: Want to study Subject A (string)

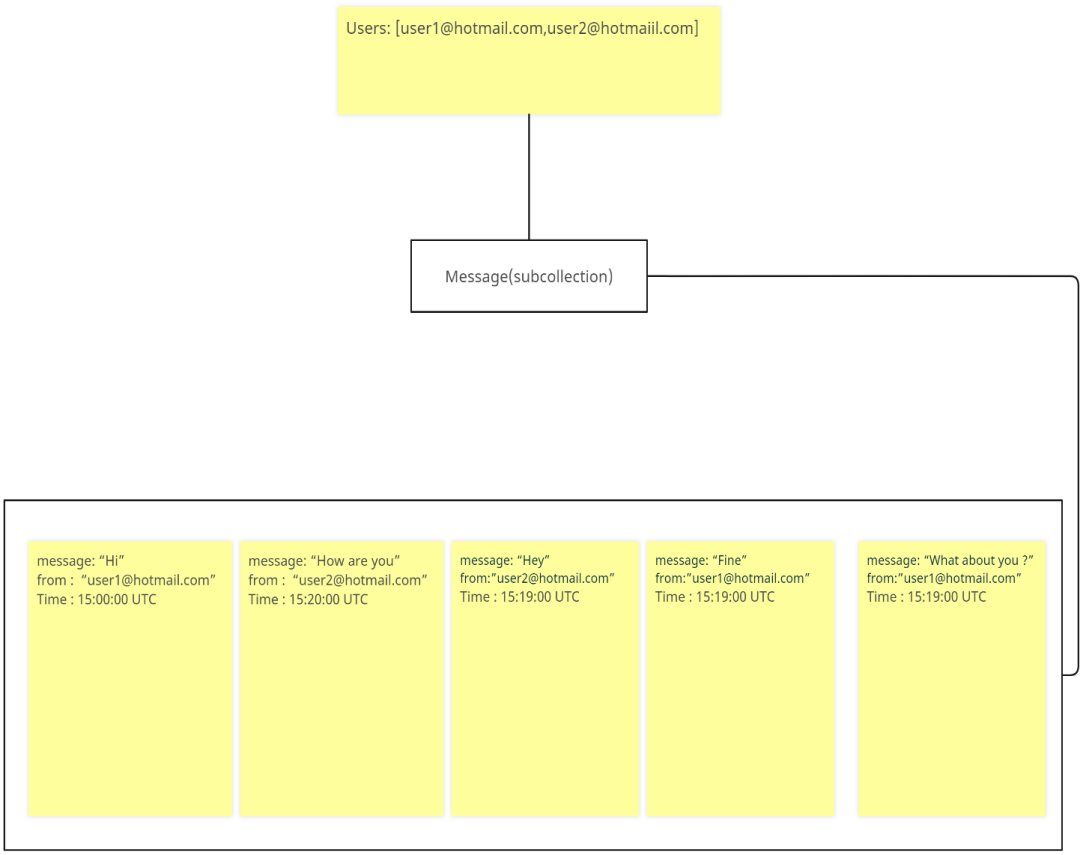
PhotoUrl : << Usable link to Firebase storage >> (string)

Messages: {<< Buddy1's email >> : << message id >>, << Buddy2's email >> : << message id >> } (map)

Messages collection stores the users and the messages of two users. Each document has a unique id (provided by the database). Firestore allows us to store subcollections within documents. In each document, we store the users in a list and a message subcollection.

The subcollection stores each message sent between the two users(their email address). Each message is stored as a document in the subcollection. Each document has a unique identifier as its name, which is automatically assigned to them by the database. Each document contains the sender of the message, the message sent and the time that the message was sent.

Sample diagrams of a document in Message collection



Note that the documents in the Message subcollection are not sorted.

The Authentication

Firebase Authentication is used to handle user management functions in the backend of the application. It ensures that users' identities are verified and authenticated before granting access to certain features and functionalities. It also works with Firestore, a NoSQL database, to link user authentication information with their corresponding data stored in Firestore. This combination of services empowers the application to deliver consistent and personalized data to users while maintaining the highest standards of security and data integrity.

Basically, with Firebase Authentication, we are going to create new user accounts, allow users to sign in securely, facilitate the process of signing out, and even delete user accounts when necessary. (Firebase, 2023)

Storage

Firebase provides a powerful cloud storage solution that can securely store and manages profile pictures of users. When a user uploads their profile picture, we use one of their identifiers called the "uid" (user ID) and store it in Firebase Storage. To display the user's profile picture on the frontend, they retrieve the image URL from Firebase Storage using the user's uid. The user uid is stored in the user's documents.

By utilizing Firebase's cloud storage solution, users can store, manage, and display profile pictures without worrying about managing image files locally. If a user wants to change their existing profile picture, the frontend can just send their new photo URL and the Storage replaces the old profile with the new one automatically. (Firebase, 2023)

API/Modules I used.

1. Firebase Web SDK (API key required)
2. Firebase Admin SDK (API key required)
3. OpenAI (API key required)
4. Flask (No API key required)

Configuring Firebase

For our project, we will need to configure Firebase. The following steps can be used to install Firebase into our project. This is assuming that we are using a Windows system:

1. (Firebase Website ref) Firstly, we need to install Firebase and Firebase admin module using NPM.

```
npm install firebase firebase-admin
```

2. Get your Firebase Admin SDK from the Firebase Console. Firebase Admin SDK allows you to use Firebase services in your backend or server. You can get the Admin SDK using the following steps (Firebase, 2023):

a. Go to the Firebase Console (https://console.firebase.google.com/) and sign in with your Google account.
b. Create a new Firebase project.
c. When you are inside the project, click on the gear icon.
d. Go to “Service Accounts” tab.
e. Click the "Generate New Private Key" button under the "Firebase Admin SDK" section which will generate a JSON file containing your private key.
f. Download and save the keys on your computer.

3. We also need to configure Firebase authentication.
    a. In the Firebase Console, go to your project's settings.
    b. Select the "Project settings" tab.
    c. Add a new web app and provide a nickname.
    d. Firebase will generate a configuration object containing your Firebase project's credentials.

Once you have completed these steps, you can proceed with initializing Firebase and using the Admin SDK.

Getting your OpenAI Key and Configuring OpenAI (Raf, 2023)

- Login to platform.openai.com
- Go to Settings and click on View API keys.
- You can generate an API key by clicking on “Create new Secret Key”.

Importance of Hiding API Keys

For safety reasons, we would need to hide our API keys from GitHub. Uploading your key in Github in a public repository can cause a security breach and unauthorized usage of your keys compromising the integrity of the backend.

Strategy for Hiding API Keys.

Firstly, I added the JSON file containing our private key into the gitignore file. This will prevent Git/Github from tracking the file (i.e the file wouldn’t show on Github).

Using the JSON file and the configuration object generated for authentication, I created a Firebase configuration JavaScript file. This file would also be added to the gitignore file, so it won’t be tracked (it also contains sensitive information).

We also need to create a python script which would store our OpenAI API key. This script will also be added to our gitignore.

Ensure that all files containing the keys is not tracked before uploading your code to Github/Gitlab.

Running the Backend in a remote server (Chavan, 2022).

- Signup/Login to Render.com.
- Click on the “New” Button and select “Web Services”.
- Connect your Github account and your Github repository to Render.com.

- Select the ‘main’ branch.
- Enter “npm install” into the “build command” box. This will install all necessary node modules.
- Enter “node server.js” into the “Start command" box.
- Add the firebase configuration files as secret files.
- Click on “Create Web Service”.

Running the Admin Website in a remote server. (Render.com, 2023)

- Signup/Login to Render.com.
- Click on the “New” Button and select “Web Services”.
- Connect your Github account and your Github repository to Render.com.
- Select the ‘adminweb’ branch.
- Enter “pip install -r requirements.txt” into the “build command” box. This will install all necessary python libraries and modules.
- Enter “gunicorn app:app” into the “Start command" box.
- Add the file containing the OpenAI API key as a “Secret file”.
- Click on “Create Web Service”.

TESTINGS

Types of Tests we are going to do. We are going to do 3 types of testing.

1. Unit testing.
2. Manual testing.
3. Database testing.

Unit testing

We are going to test some of our functions. The functions we are testing only retrieve data from the database that rarely changes.

In our unit testing, we are going to test the following methods/functions.

- Getting User Data
- Get User UID
- Count User

User Story 1: Testing User UID Function

- As a developer, I want to confirm that the **getUserUid** function correctly returns the user UID for a given email.
- I want to ensure that the user UID returned is accurate and matches the expected UID.
- Additionally, I want to validate that the user UID does not match an incorrect UID for a different email.

User Story 2: Testing Count Users Function

- As a developer, I want to verify that the **countUsers** function accurately counts the number of users in the database.
- I expect the function to return the correct number of users and validate that it matches the expected count.
- I want to compare the number of users counted with the number of user IDs retrieved from the database to ensure consistency.

User Story 3: Testing Get Users Data Function

- As a developer, I aim to test the function to ensure it retrieves the correct data for a given user UID.
- I want to confirm that the user's email, name, flag, and age are retrieved accurately.
- I also want to validate that the function successfully retrieves the user's buddies and messages.

Manual Testing and Database testing

Manual Testing with Postman

I want to ensure the successful execution of requests and the accuracy of responses by using Postman. This will allow me to validate the functionality of the backend methods and endpoints.

Acceptance Criteria:

- I will send requests using Postman to the appropriate endpoints.
- I receive the expected HTTP response codes, such as 200 for successful requests or appropriate error codes for failed requests.
- The response payloads match the defined API specifications, including the expected data and response structure.
- The response time of the requests falls within acceptable limits.

Database Testing

I need to verify that the backend operations are modifying the data correctly in the database. By conducting database testing, I can ensure the integrity and consistency of the data.

Acceptance Criteria:

- I perform operations through the endpoints that interact with the Firestore database.
- I examine the relevant database tables or collections to ensure the expected modifications have been made.
- By executing appropriate queries or retrieval operations, I validate that the data modifications through the endpoints are accurately reflected in the database.
- The data in the database aligns with the expected results or predefined test scenarios.

We are going to combine manual testing with database testing for the other methods and endpoints. We are using Postman to confirm whether the request was successful, and we are going to check the database to ensure that the data was modified accordingly.

Scenario 1: User sends a request to create a new user with incorrectly formatted email and password.

- Given that a user wants to create a new account.
- When the user provides a email or password in an incorrect format.
- No new user is created in the Firestore Database and Firebase Authentication.
- The system should respond with an appropriate error message indicating the incorrect format of the name and/or email.
- The response should include an appropriate status code to indicate that a new user wasn't created.

Scenario 2: User sends a request to create a new user with correctly formatted email and password.

- Given that a user wants to create a new account.
- When the user provides a name, email, and password in the correct format.
- Then the system should create a new user in the Firestore Database and Firebase Authentication
- The system should respond with a status code 200 to indicate a successful request.
- The response confirms the successful user creation.

By ensuring the handling of user creation with both correct and incorrect formats for the name and email, we can provide a user-friendly experience by preventing the creation of users with invalid data while allowing the successful creation of users when the correct format is used.

Scenario 3: User sends a request to sign in with the wrong password or wrong email address.

- Response is sent with appropriate error messages and appropriate status code.
- The User is not signed into the Firebase authentication.

Scenario 4: User sends a request to sign in with the right password and right email.

- Response with status code 200 with the right message is sent.
- The message is a JSON with a token and a flag.
- The User is signed into the Authentication.
- The token can be used to verify the user.

Scenario 5: User sends a request to set their data in the wrong format.

- Response is sent with appropriate error message.
- User Data in the database is not updated in the User Document.

Scenario 6: A request to set their data with the invalid token.

- Response is sent with appropriate error message with appropriate error message.
- User Data in the database is not updated.

Scenario 7: User sends a request to set their data in the right format and with the right token.

- Response with status code 200 is sent with a success message.
- User Data in the database is updated.

Scenario 8: User sends a request to get their data with the wrong token.

- Response is sent with appropriate error message with appropriate error message.

Scenario 9: User sends a request to get their data with the right token.

- Response with status code 200 is sent with a success message containing that Users data.

Scenario 10: User sends a request to swipe another user with the wrong format.

- Response is sent with appropriate error message with appropriate error message.
- No data is added to our database.

Scenario 11a: User sends a request to positively swipe another user with the right format.

- Response with status code 200 is sent with a success message.
- User's data is updated in the database. The User's swipedThem array is updated in the database with the user that they swiped.
- The other user's data is updated in the database. This User's swipedMe array is updated in the database with the user that swiped them.

Scenario 11b: User sends a request to negatively swipe another user with the right format.

- Response with status code 200 is sent with a success message.
- No new data is added to the database.

Scenario 11c: User sends a request to positively swipe another user with the right format, the other user had already swiped positively on the current user before.

- Same as Scenario 11a.
- The other user is added to the user's buddies list and the user is added to the other user's buddies list.

Scenario 12: User sends a request to message one of their buddies in the wrong format.

- Response is sent with appropriate error message with appropriate error message.
- No message is added to the database.

Scenario 13: User sends a request to message one of their buddies in the right format.

- Response with status code 200 is sent with a success message.
- Message is added to the database.

Scenario 14: User sends a request to delete their account.

- User is removed from the database and the authentication.
- Response with status code 200 is sent with a success message.

Bibliography

Chavan, Y. (2022, September 1). *How to Deploy Your Node.js Application for Free with Render*. From FreeCodeCamp: https://www.freecodecamp.org/news/how-to-deploy-nodejs-application-with-render/

Contributor, T. (2023). *What is an API endpoint?* From Tech Target Website: https://www.techtarget.com/searchapparchitecture/definition/API-endpoint

Firebase. (2023). *Add the Firebase Admin SDK to your server*. From Firebase Website: https://firebase.google.com/docs/admin/setup

Firebase. (2023). *Cloud Storage for Firebase*. From Firebase Website: https://firebase.google.com/docs/storage

Fork a Repo. (2023). From Github Documentation Website: https://docs.github.com/en/get-started/quickstart/fork-a-repo

Postman. (2023). *What is Postman? Postman API Platform*. From Postman Website: https://www.postman.com/product/what-is-postman/

Raf. (2023). *Where do I find my Secret API Key?* From OpenAI website: https://help.openai.com/en/articles/4936850-where-do-i-find-my-secret-api-key

Render.com. (2023). *Deploy a Flask App*. From Render.com Website: https://render.com/docs/deploy-flask