Eötvös Loránd University

Faculty of Informatics

Department of Programming Languages and Compiler

# 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.

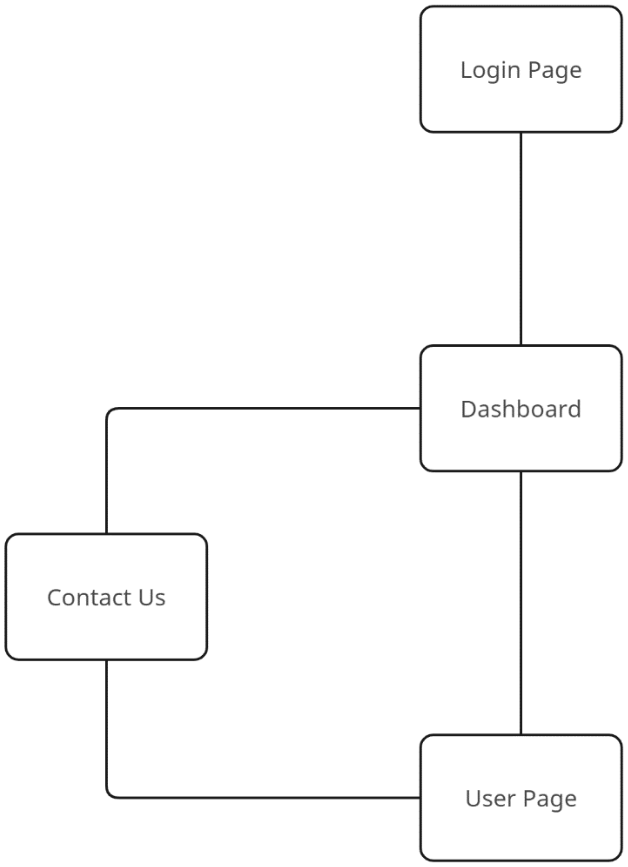
You will need to fork the repository of this project for that. Follow the instructions below:

- 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.

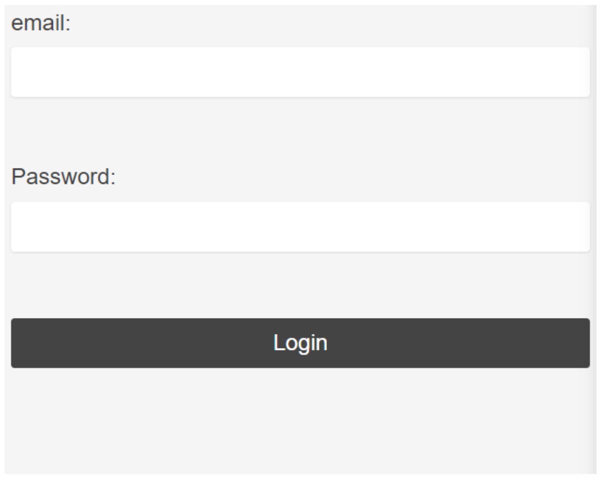
## Navigation Map for the Admin Website



### 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.

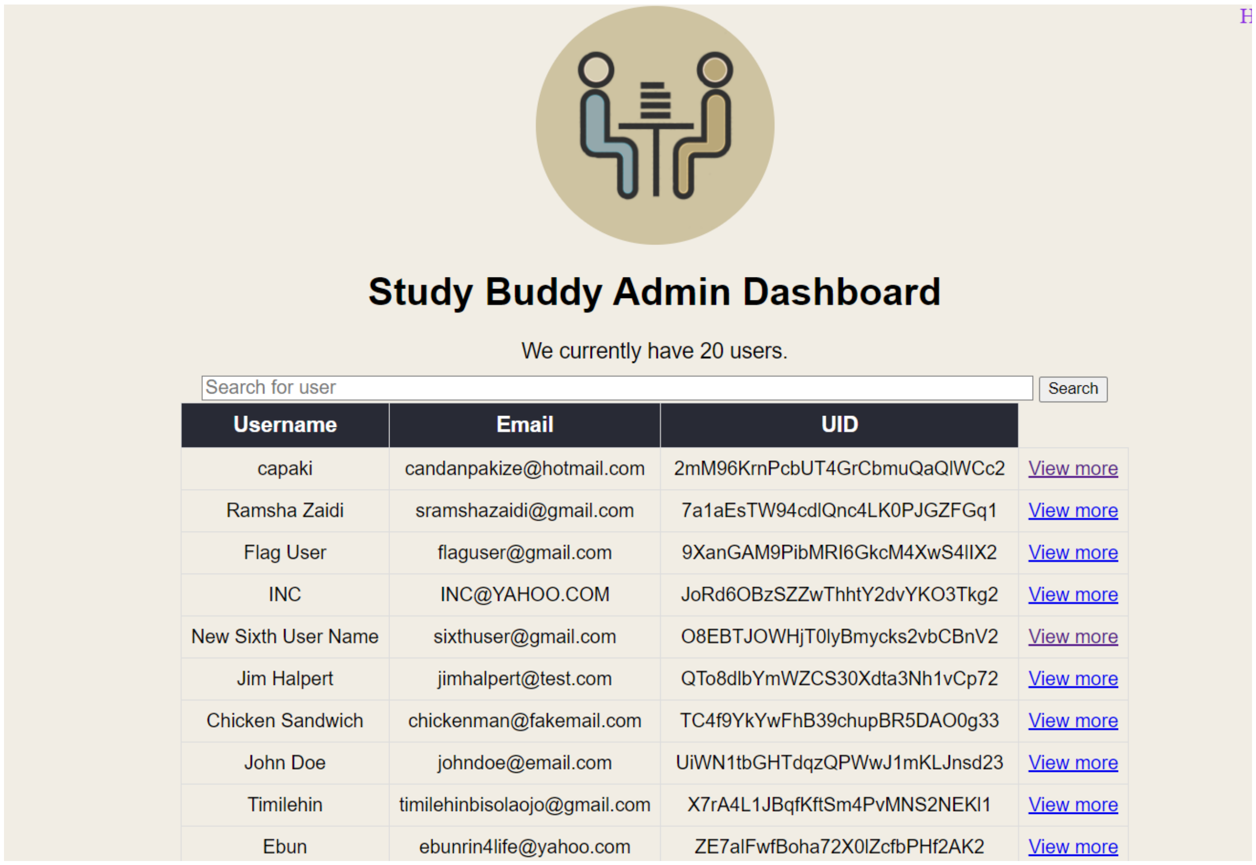
**Login Page Screenshot**



## 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



**Study Buddy Admin Dashboard**

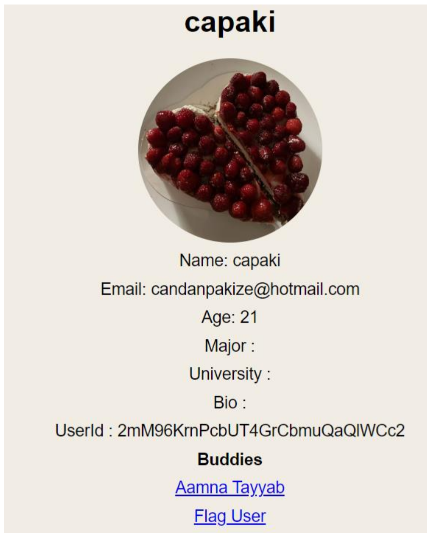
We currently have 20 users.

| Username | Email | UID | |
|---|---|---|---|
| capaki | candanpakize@hotmail.com | 2mM96KrnPcbUT4GrCbmuQaQlWCc2 | View more |
| Ramsha Zaidi | sramshazaidi@gmail.com | 7a1aEsTW94cdlQnc4LK0PJGZFGq1 | View more |
| Flag User | flaguser@gmail.com | 9XanGAM9PibMRI6GkcM4XwS4lIX2 | View more |
| INC | INC@YAHOO.COM | JoRd6OBzSZZwThhtY2dvYKO3Tkg2 | View more |
| New Sixth User Name | sixthuser@gmail.com | O8EBTJOWHjT0lyBmycks2vbCBnV2 | View more |
| Jim Halpert | jimhalpert@test.com | QTo8dlbYmWZCS30Xdta3Nh1vCp72 | View more |
| Chicken Sandwich | chickenman@fakemail.com | TC4f9YkYwFhB39chupBR5DAO0g33 | View more |
| John Doe | johndoe@email.com | UiWN1tbGHTdqzQPWwJ1mKLJnsd23 | View more |
| Timilehin | timilehinbisolaojo@gmail.com | X7rA4L1JBqfKftSm4PvMNS2NEKl1 | View more |
| Ebun | ebunrin4life@yahoo.com | ZE7alFwfBoha72X0lZcfbPHf2AK2 | View more |

## 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



**capaki**

Name: capaki

Email: candanpakize@hotmail.com

Age: 21

Major :

University :

Bio :

UserId : 2mM96KrnPcbUT4GrCbmuQaQlWCc2

**Buddies**

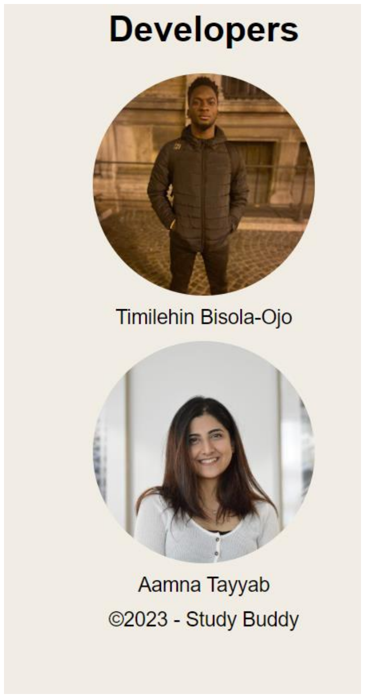
Aamna Tayyab

Flag User

## 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


Developers

Timilehin Bisola-Ojo

Aamna Tayyab

©2023 - Study Buddy

## 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.eyJpc3M
iOiJodHRwczovL3NlY3VyZXRva2VuLmdvb2dsZS5jb20vc3R1ZHktYnVkZHktYmFja2VuZC1kZTA4YSIsImF1ZCI6InN0dWR5LWJ1ZGR5LWJhY2
tlbmQtZGUwOGEiLCJhdXRoX3RpbWUiOjE2ODQ2NzE2OTksInVzZXJfaWQiOiJTR25vU2s0TFNzUjJjVDRKcGt1ekVvczNZMjMzIiwic3ViIjoiU
0dub1NrNExTc1IyY1Q0SnBrdXpFb3MzWTIzMyIsImlhdCI6MTY4NDY3MTY5OSwiZXhwIjoxNjg0Njc1Mjk5LCJlbWFpbCI6InNlY29uZHVzZXJA
Z21haWwuY29tIiwiZW1haWxfdmVyaWZpZWQiOmZhbHNlLCJmaXJlYmFzZSI6eyJpZGVudGl0aWVzIjp7ImVtYWlsIjpbInNlY29uZHVzZXJAZ21
haWwuY29tIl19LCJzaWduX2luX3Byb3ZpZGVyIjoicGFzc3dvcmQifX0.JbwCe23Ybbtf4T_evWsgD09zQqB0NgOu_CCD1DwMRyVf2SOaZZqmgY
pKJSUgaA8PmNl5bkyPG52Xe4dnGMVYu5BdpitNi8toHTz62PPc1HqFpAcvtY2ObqIozPLeoXdMMgJLy0cd6Vl8C_fI7d9ZppiqQ_ePKv6k7oDqW
8r5wLznIOpdi1lny7V8b9DgMfas4GfkkeS8KC0KeDq0UIV4eTz1-93CSIDLG5sGVmTVKpg0as9EUtJFseAAKwgLrJWn9t-
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.

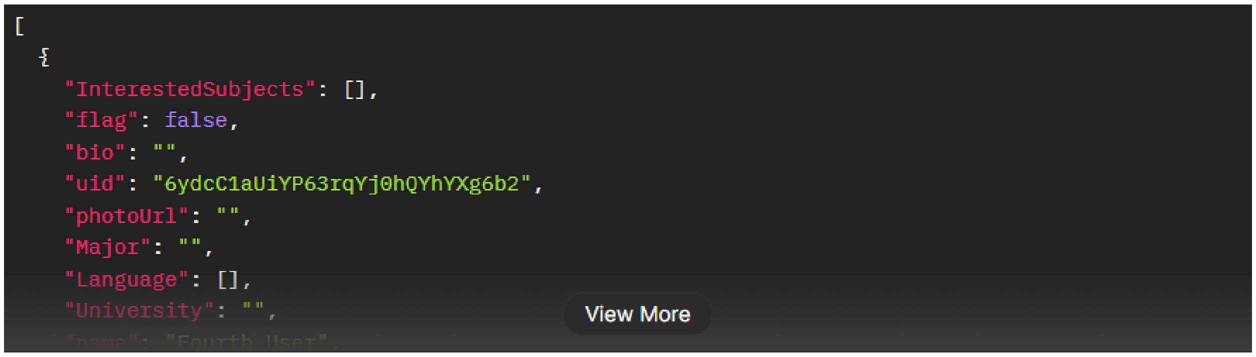
### Example Request

```
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**



```
[
  {
    "InterestedSubjects": [],
    "flag": false,
    "bio": "",
    "uid": "6ydcC1aUiYP63rqYj0hQYhYXg6b2",
    "photoUrl": "",
    "Major": "",
    "Language": [],
    "University": "",
    "name": "Fourth User"
```

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

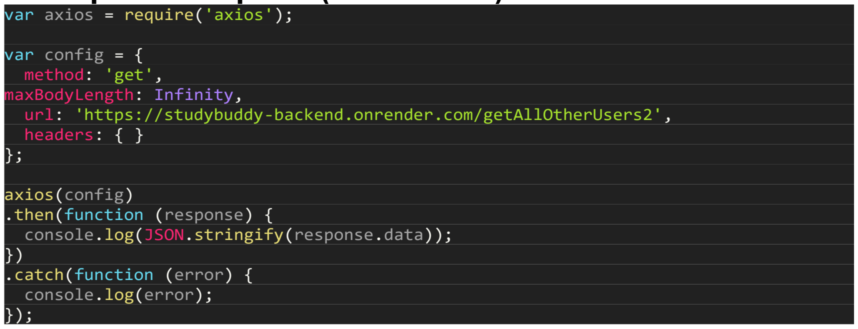
https://studybuddy-backend.onrender.com/getAllOtherUsers2

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)**

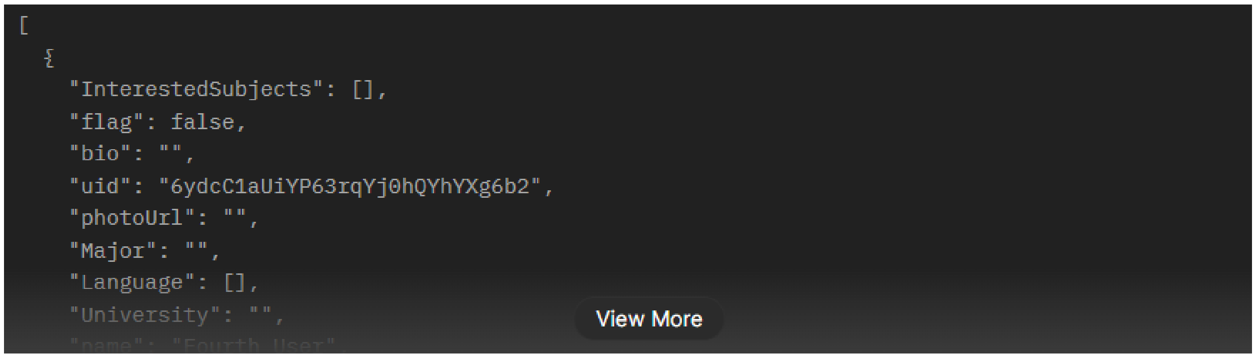


```
var axios = require('axios');

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)**



```
[
  {
    "InterestedSubjects": [],
    "flag": false,
    "bio": "",
    "uid": "6ydcC1aUiYP63rqYj0hQYhYXg6b2",
    "photoUrl": "",
    "Major": "",
    "Language": [],
    "University": "",
    "name": "Fourth User",
```

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_
_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);
});
```

## 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
oiYlRrTjlGeGZPcVhXTTdGV20wYlVjVTU1N3MwMyIsImlhdCI6MTY4NDY4NTMwOSwiZXhwIjoxNjg0Njg4OTA
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.'
    }"
```

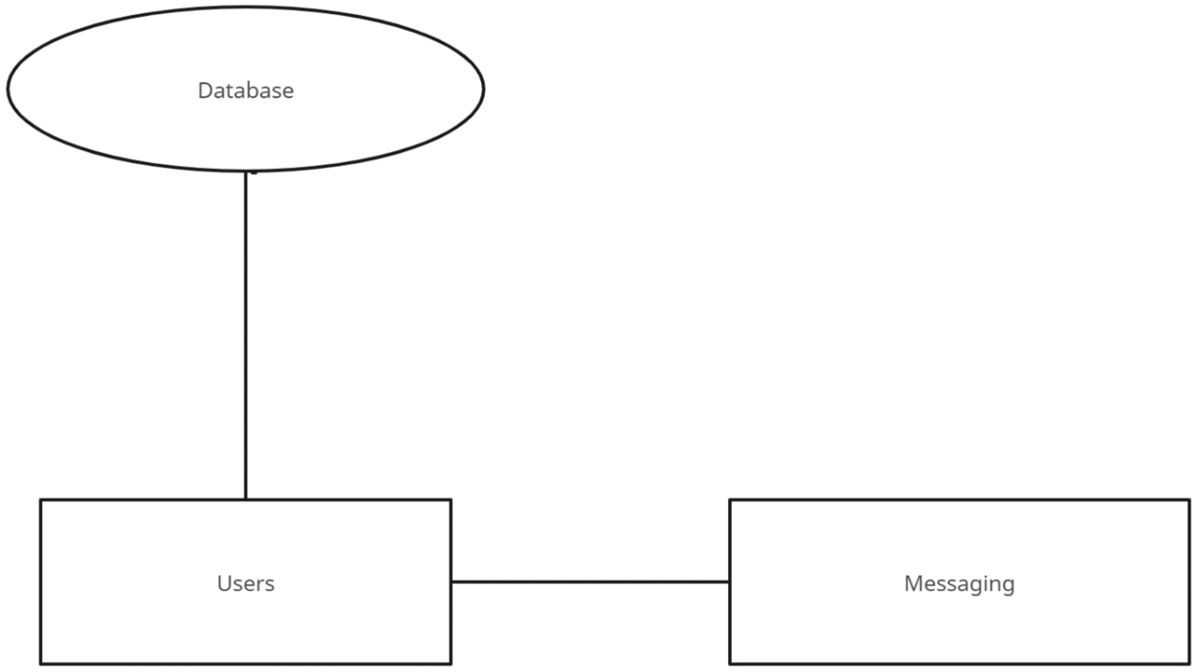
# Developers Documentation

## 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.

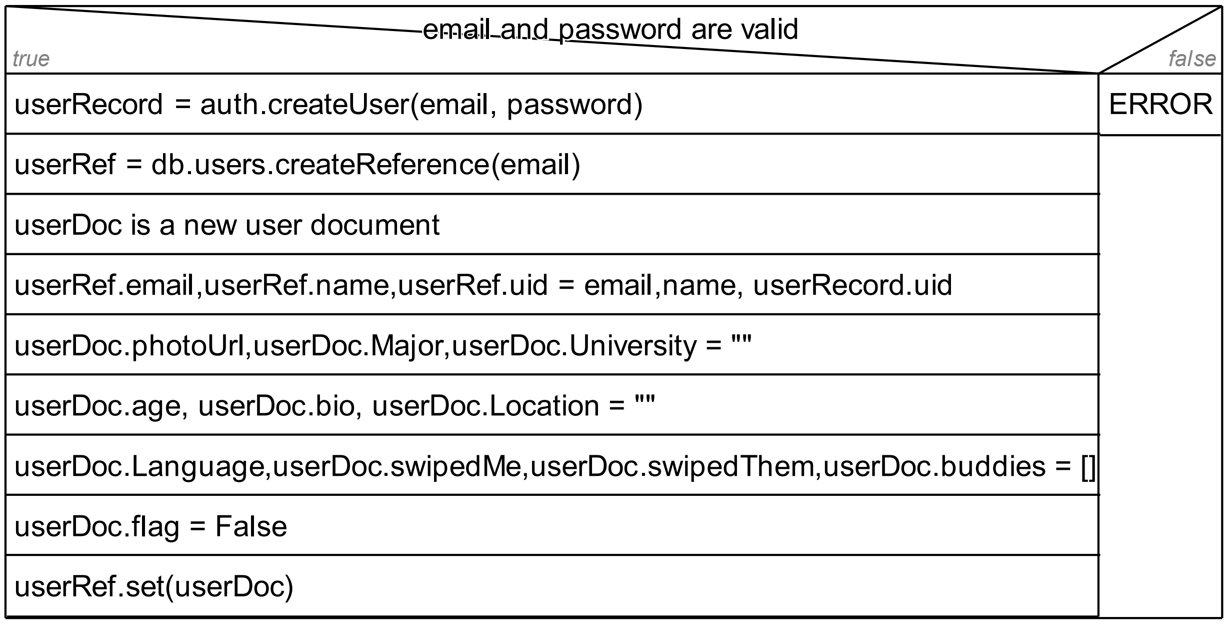
By utilizing the power of the GPT model and providing a well-structured prompt, we can leverage its natural language processing capabilities to generate accurate, relevant, and personalized recommendations for our users.

## 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)**

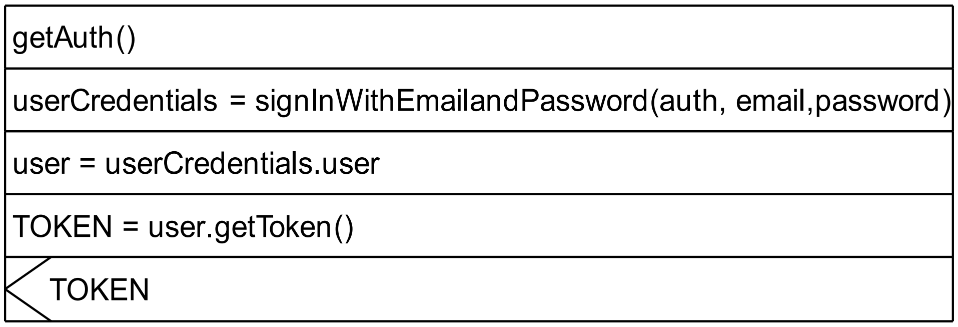


```
email and password are valid
true:
    userRecord = auth.createUser(email, password)
    userRef = db.users.createReference(email)
    userDoc is a new user document
    userRef.email,userRef.name,userRef.uid = email,name, userRecord.uid
    userDoc.photoUrl,userDoc.Major,userDoc.University = ""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

### 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)**

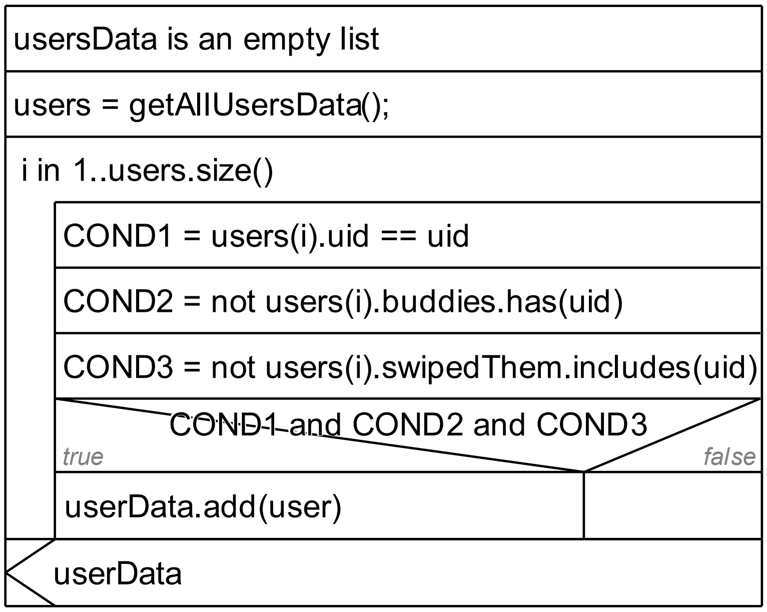


```
getAuth()
userCredentials = signInWithEmailandPassword(auth, email,password)
user = userCredentials.user
TOKEN = user.getToken()
return TOKEN
```

## 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)



```
usersData is an empty list
users = getAllUsersData();
i in 1..users.size()
    COND1 = users(i).uid == uid
    COND2 = not users(i).buddies.has(uid)
    COND3 = not users(i).swipedThem.includes(uid)
    COND1 and COND2 and COND3
        true: userData.add(user)
        false:
return userData
```

## 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.

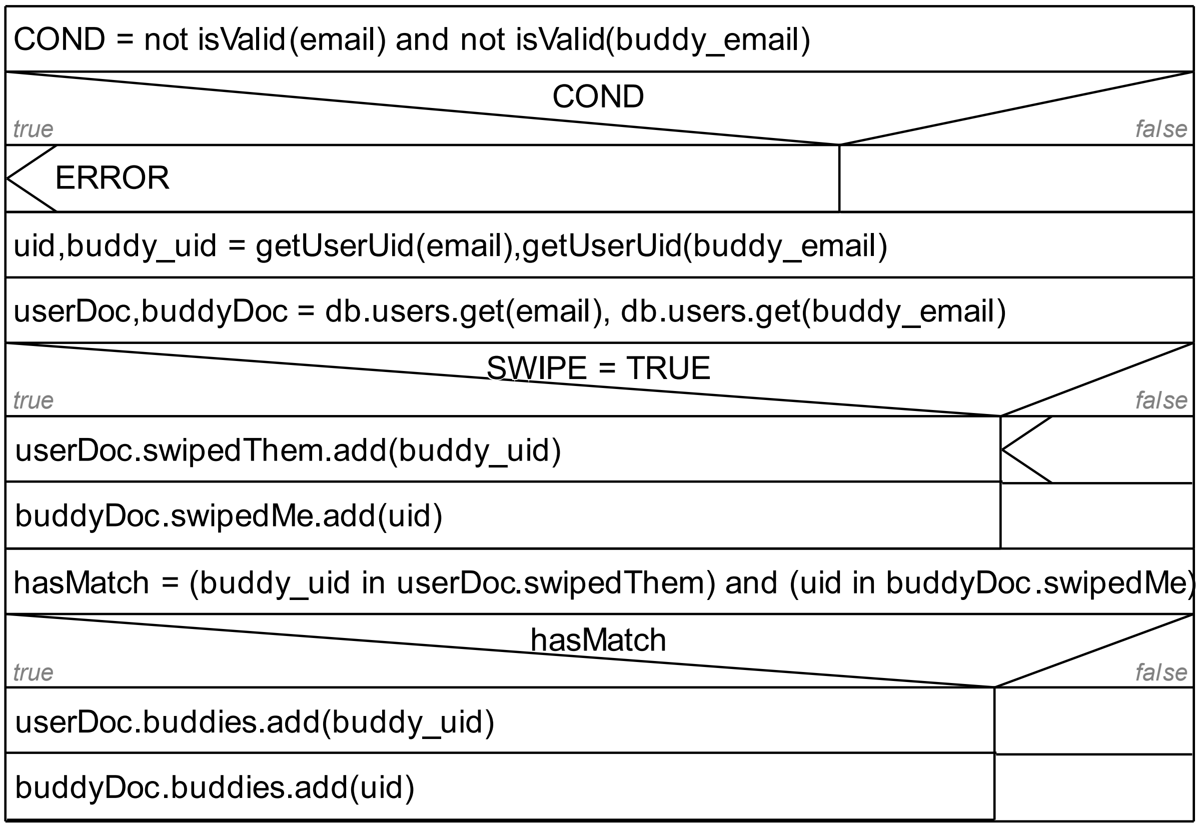
### getMessages(chatId)

```
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)**



```
COND = not isValid(email) and not isValid(buddy_email)
COND
  true:
    ERROR
  false:
uid,buddy_uid = getUserUid(email),getUserUid(buddy_email)
userDoc,buddyDoc = db.users.get(email), db.users.get(buddy_email)
SWIPE = TRUE
  true:
    userDoc.swipedThem.add(buddy_uid)
    buddyDoc.swipedMe.add(uid)
  false:
    (exit)
hasMatch = (buddy_uid in userDoc.swipedThem) and (uid in buddyDoc.swipedMe)
hasMatch
  true:
    userDoc.buddies.add(buddy_uid)
    buddyDoc.buddies.add(uid)
  false:
```

## Getting All Users Data

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

**getAllUserData()**

```
userData is an empty list
AuthUsers = getAllUsers();
user in AuthUsers
    user_data = db.users.get(user.email)
    usersData.add(user_data);
UsersData;
```

## 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.

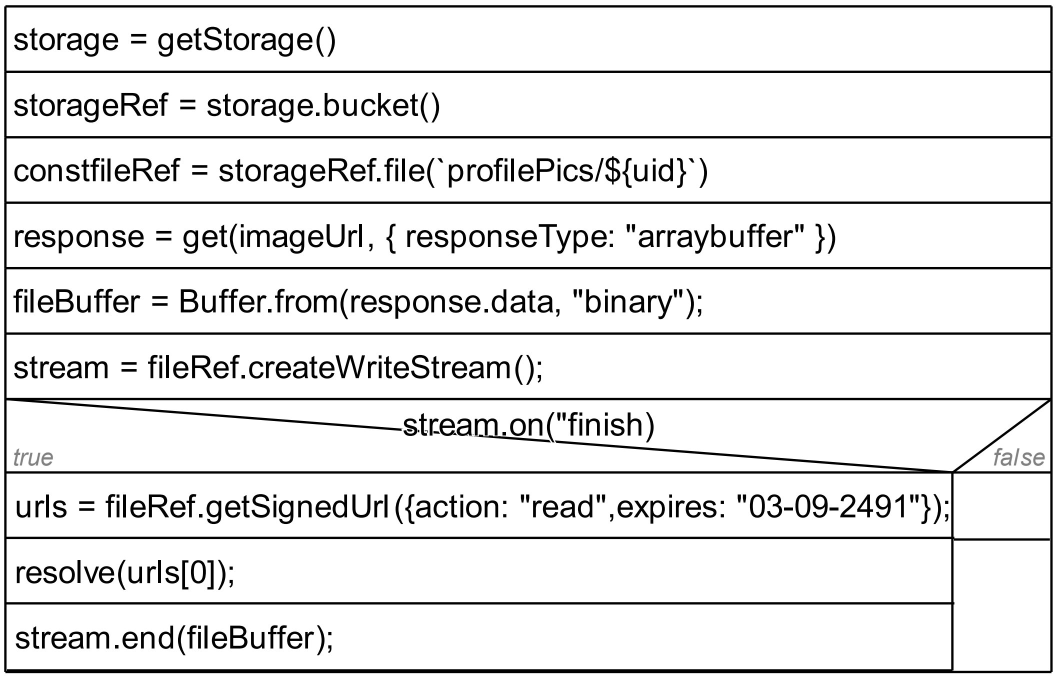
### Getting Users UID

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)**



```
storage = getStorage()
storageRef = storage.bucket()
constfileRef = storageRef.file(`profilePics/${uid}`)
response = get(imageUrl, { responseType: "arraybuffer" })
fileBuffer = Buffer.from(response.data, "binary");
stream = fileRef.createWriteStream();
stream.on("finish)
    true
        urls = fileRef.getSignedUrl({action: "read",expires: "03-09-2491"});
        resolve(urls[0]);
        stream.end(fileBuffer);
    false
```

### 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.

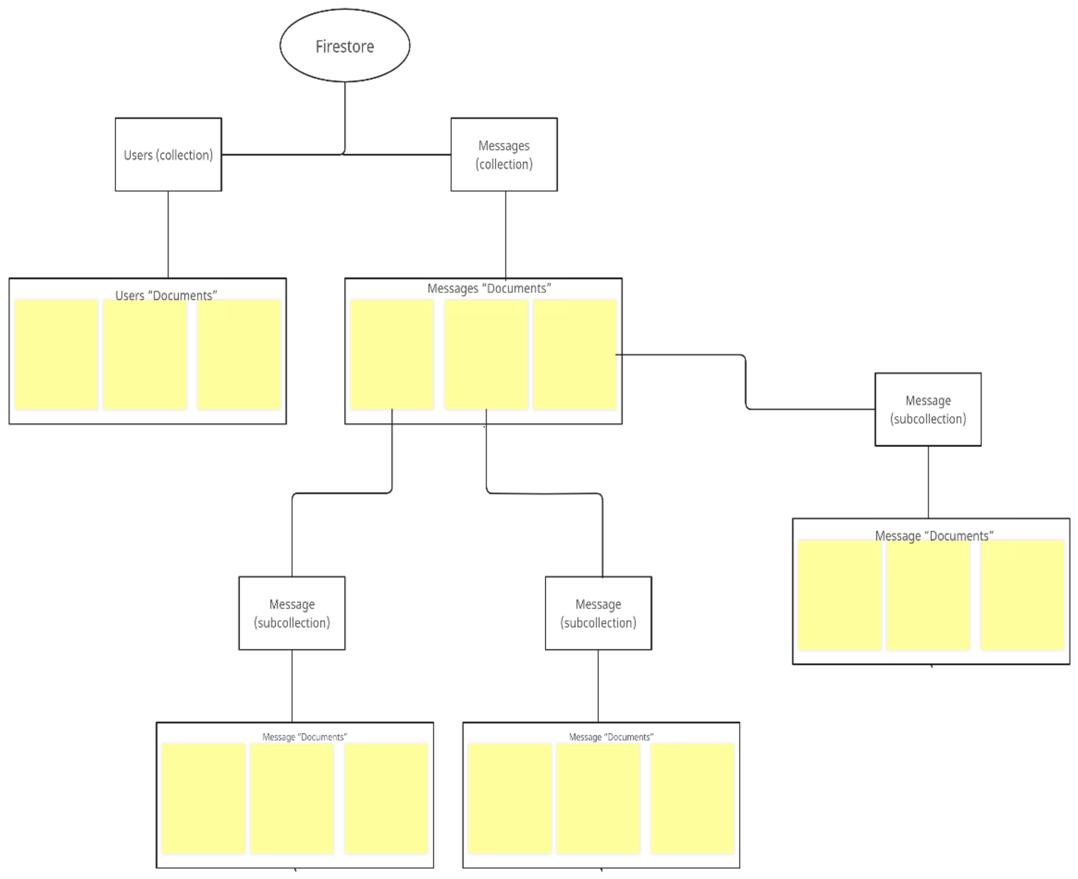
## The Database

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

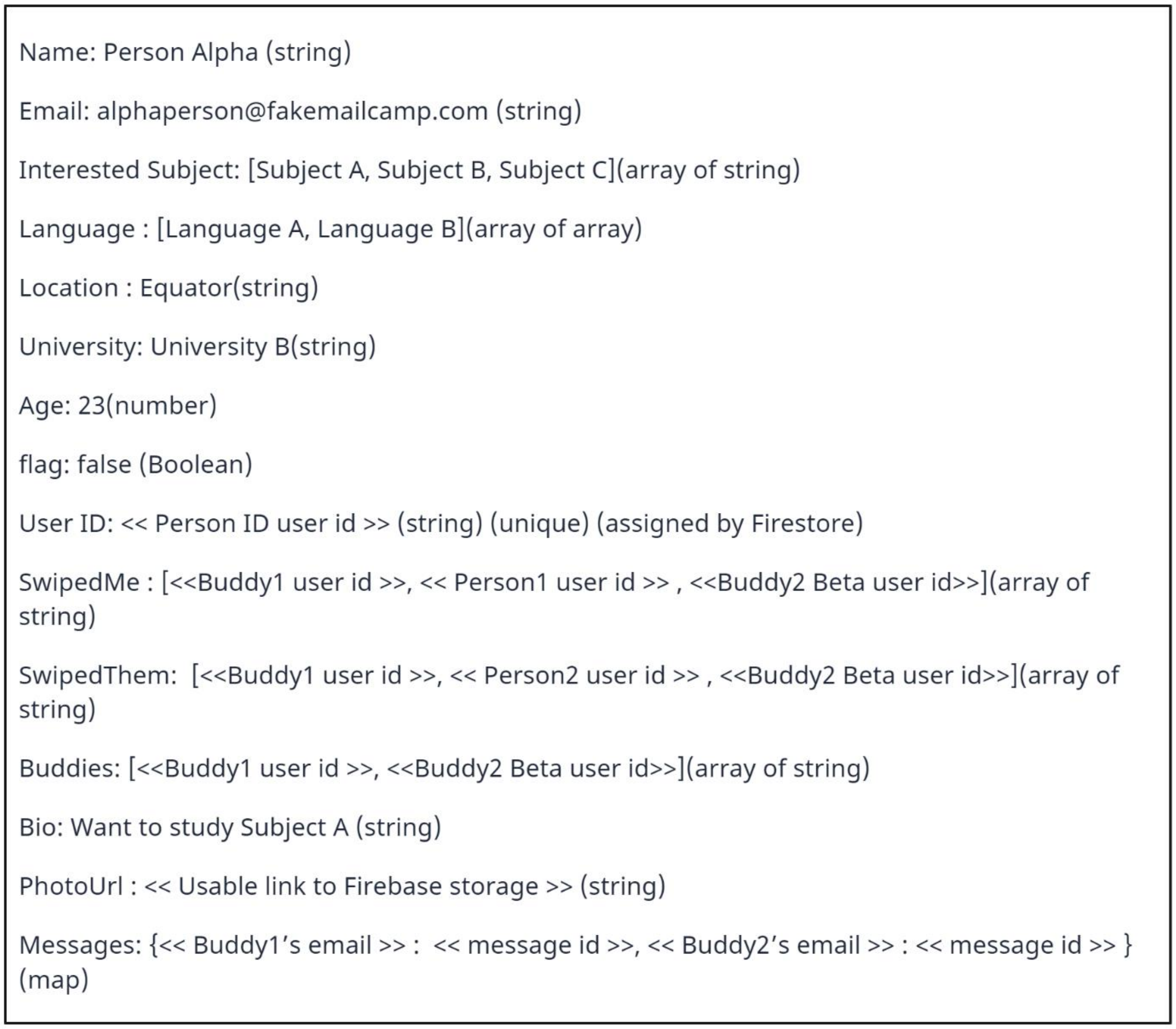


User collection is where we store information about each user. Each user's data is stored in a document named after their email address in the User collection. This includes their personal details like their name, age, email address, location, the languages they speak, the university they go to, university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

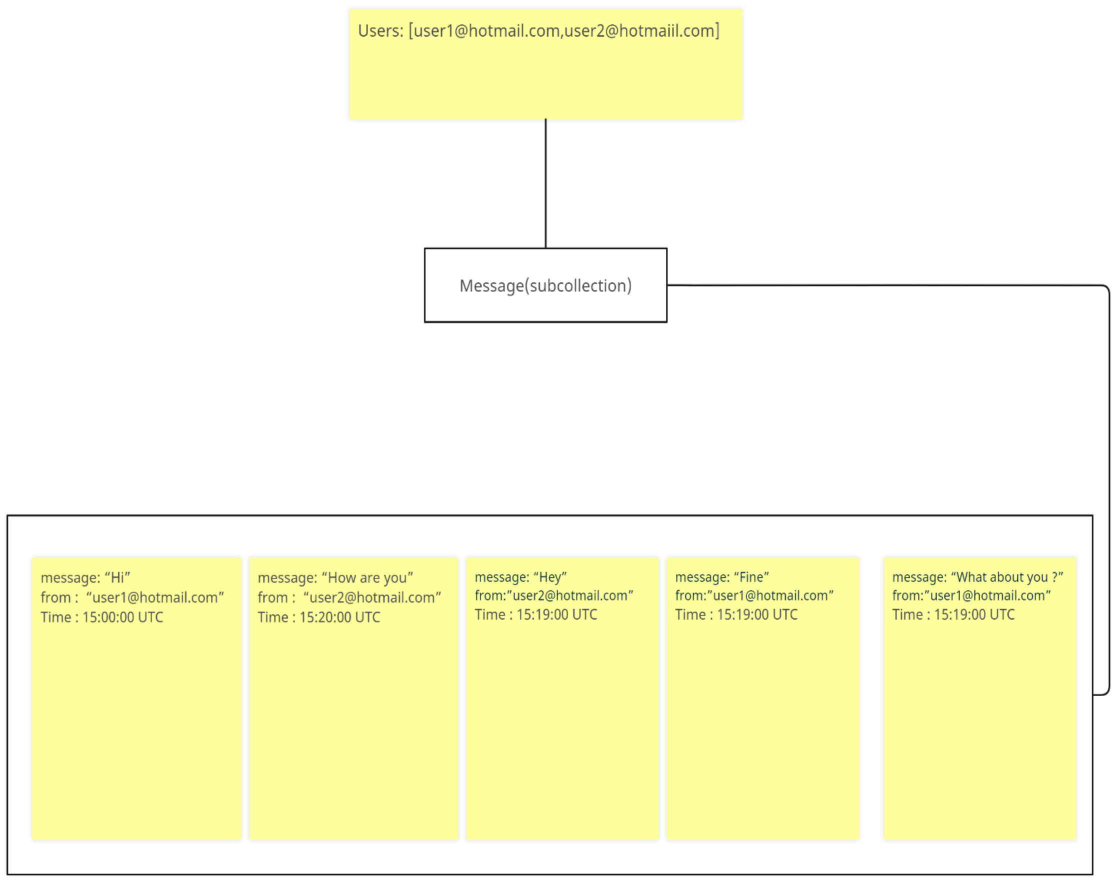
PhotoUrl : << Usable link to Firebase storage >> (string)

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


Users: [user1@hotmail.com,user2@hotmaiil.com]
Message(subcollection)
message: “Hi” from : “user1@hotmail.com” Time : 15:00:00 UTC
message: “How are you” from : “user2@hotmail.com” Time : 15:20:00 UTC
message: “Hey” from:”user2@hotmail.com” Time : 15:19:00 UTC
message: “Fine” from:”user1@hotmail.com” Time : 15:19:00 UTC
message: “What about you ?” from:”user1@hotmail.com” Time : 15:19:00 UTC

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