Front End Implementation

Part1- Login and Sign-up

6 components(roughly)

1. Signup
2. Login
3. Myaccount
    a. View profile
    b. Edit profile
    c. Change password and delete user

Functionalities from the backend:

1. Signup

   Send POST request to http://127.0.0.1:8000/api/signup with data in json format as:

```
{
"email":"ram@gmail.com",
"Password":"123456",
"profile": {
      "first_name": "ram",
      "last_name": "kumar",
      "phone_number": "1234567891",
      "age": 11,
      "gender": "M"
}
}
```

   Upon Successful submission the response will be as follows:

```
        {
"success": "True",
"status code": 201,
"message": "User registered  successfully"
}
```

   Else if email or phone number is not unique, something like this will be sent:

```
{
  "email": [
     "user with this email address already exists."
  ],
  "profile": {
     "phone_number": [
        "user profile with this phone number already exists."
     ]
  }
}
```

And in any other case of unsuccessful signup error will be thrown.

2. Login
   Send a POST request to http://127.0.0.1:8000/api/login with data in json format:
   {
   "email":"ram@gmail.com",
   "password":"123456",
   }

   Upon successful login the response from the server will be something like:
   {
   "success": "True",
   "status code": 200,
   "message": "User logged in successfully",
   "token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoiODY3Y2E3YjAtZDhjNC00ZTdkLWE1NmYtOWRhYWJkOTAwNmQ1IiwidXNlcm5hbWUiOiJyYW1AZ21haWwuY29tIiwiZXhwIjoxNTgwMTA0MDc4LCJlbWFpbCI6InJhbUBnbWFpbC5jb20ifQ.0cCgOlKrYHrouVJEeIEt6TdGyza2C78J5swXFEaLLFM"
   }
   You need to store the token obtained in the frontend. See this https://stackoverflow.com/questions/63141982/how-to-store-and-handle-jwt-tokens-on-front-end-using-fetch-and-how-to-store-it

   Wrong login credentials will land up in a response something like this:
   {
     "non_field_errors": [
       "A user with this email and password is not found."
     ]
   }

   And in any other case error will be sent.

3. My Account
   a. View profile
      Send a GET request to http://127.0.0.1:8000/api/profile with empty body obviously and whose request header contains the token obtained on login like this:
      Authorization: Bearer <token>
      (Read how to set headers in requests)

      Upon success the data received will of following format:
      {
        "success": "true",

```
  "status code": 200,
  "message": "User profile fetched successfully",
  "data": [
     {
        "first_name": "shyam",
        "last_name": "kumar",
        "phone_number": "1234523491",
        "age": 22,
        "gender": "F"
     }
  ]
}
```


Note that the actual data is contained in the data field.
Upon sending invalid token the response will be something like this:


```
{
  "detail": "Error decoding signature."
}
```


Upon sending a valid token but the user not existing in the database following response will be sent:


```
{
            “success”: “false”,
            “status code”: 404,
            “message”: “User does not exists”,
            “error”: “Some error info here”
 }
```


This case can be distinguished from the others by the presence of error field.

b. Edit profile: make a form with the same fields as signup except email and password

Send a PUT request to http://127.0.0.1:8000/api/profile with JST token in the request header as above and body contains data in json format like:


```
{
  "first_name": "ramesh",
  "last_name": "kumar",
  "phone_number": "1234567891",
  "age": 11,
  "gender": "M"
}
```


In case of success the same response will be sent back as:

{
"first_name": "ramesh",
"last_name": "kumar",
"phone_number": "1234567891",
"age": 11,
"gender": "M"
}
Upon sending invalid token the response will be something like this:
{
"detail": "Error decoding signature."
}
And in any other case 404 status with serializer errors will be sent

Delete user:
Send a DELETE request to http://127.0.0.1:8000/api/delete_user with token in the request header and empty body obviously.

Upon success 204 “no content” response status will be sent and it is advisable to redirect users to signup page after this.
And other cases include not sending a valid token(same response as above) or User corresponding to token not existing in database(404 status)

Change password:
Send a PUT request to http://127.0.0.1:8000/api/change_pass with token in the request header and body contains data in json format as follows:
{
"old_password":"123456",
"new_password":"123457"
}

Note that there should be three fields on the front end where the user is supposed to type the new password twice and they both are the same or not should be checked in the frontend itself and any one can be sent in the “new_password” field. No need to verify anything abt old_password, that will be handled in the backend.

You can have change_password and delete user functionality in the same app component if you wish.

- The task of logout is under development. Just create a button for logout in the frontend as of now.

- Another task is of sessions, I have set session timings in the backend say one hour when the front end is supposed to ask the user if he/she wants to extend the session, if pressed yes, you need to send a refresh token request to backend where another token will be sent that is supposed to used in the header for subsequent requests. After the expiry time of the first token(say 5 hours) no matter even if the user refreshed the token he will have to login again to continue.

We can discuss the implementation of this feature after you are done with the rest.

Part 2 - Plagiarism Checker Implementation

The main file for detecting plagiarism is similarity.py

How to run it :

python3 similarity.py <folder_name>

It MIGHT work with python3 similarity.py <folder_path>. Please try to test using this format.

What does it do?

1. Prints correlation matrix on stdout
2. Makes 2 folders in the folder to be tested - Graphs and CSV
3. Graphs contains 2 graphs -
   I. Histogram
   II. Heat map
4. CSV file contains the similarities in the format
   file_1,file_2,similarity
   Some points to be noted on this -
   A. Similarity is a number between 0 and 1 (both inclusive in case it is not obvious).
   B. Only one of (file_1,file_2,similarity), (file_2,file_1,similarity) will exist in the CSV file.
   C. CSV file will not contain (file_1,file_1,similarity) as this is trivial and the similarity will be 1.

D. The similarities are arranged in no order. Use sqlite3 or something to sort according to your requirements.