CHIA YI QUAN

Student Number: 210220223

aDVANCED WEB DEVELOPMENT COURSEWORK 2

SOCIAL NETWORK

Table of Contents

[Running Of The Project 2](#_Toc129461194)

[Installation And Setup 2](#_Toc129461195)

[Creating A New Environment 3](#_Toc129461196)

[Setting Up Redis(skip this if it is installed and running) 4](#_Toc129461198)

[Running The Test Case 5](#_Toc129461199)

[Running The Project 5](#_Toc129461200)

[Project Functionality Requirements 6](#_Toc129461201)

[Conclusion 43](#_Toc129461202)

[References 45](#_Toc129461203)

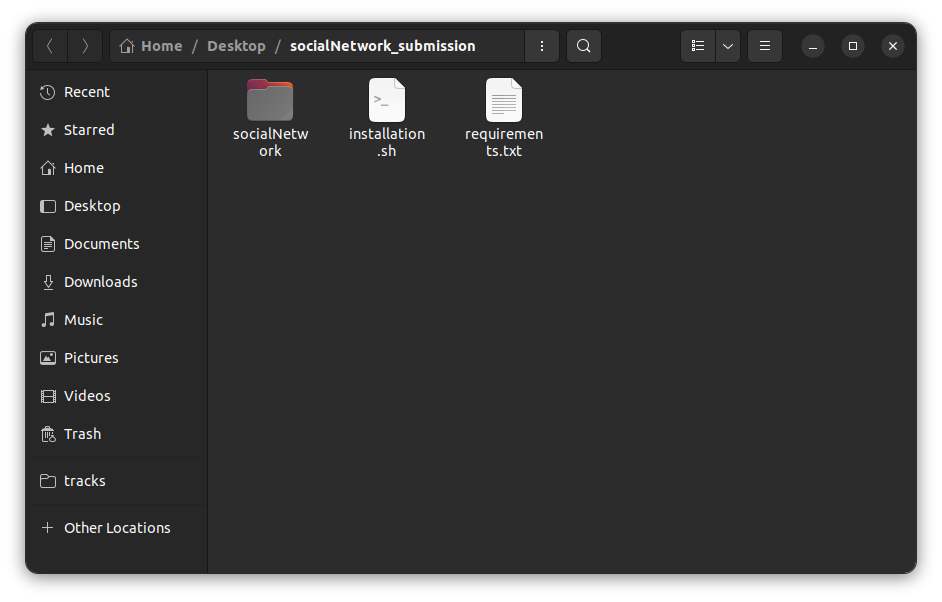
Introduction

This report will cover building a social network coursework. The first portion of the report will have the instructions on installation and running the code, the second portion will cover the explanation of the code and the functionalities that were met for the criteria of the project.

# Running Of The Project

## Installation And Setup

From the submitted project folder, you will find a requirements.txt file and the socialNetwork folder. Before we start, this code was developed and tested on ubuntu system, due to redis doesn’t have direct support on windows I would recommend you running this code on ubuntu but if you uses windows operating system, the redis can be installed through windows subsystem for linux(wsl) but it won’t work perfectly due to windows firewall are able to block the data from wsl.



The submission should consist of 2 files and 1 folder. The socialNetwork folder is where the django code is located, installation.sh is a script that creates an environment and installs all the dependencies for the project and change directory to the project folder. The requirements.txt contains all the packages needed for this project. This script only works in the unix operating system(tested with Ubuntu only).

### Creating A New Environment

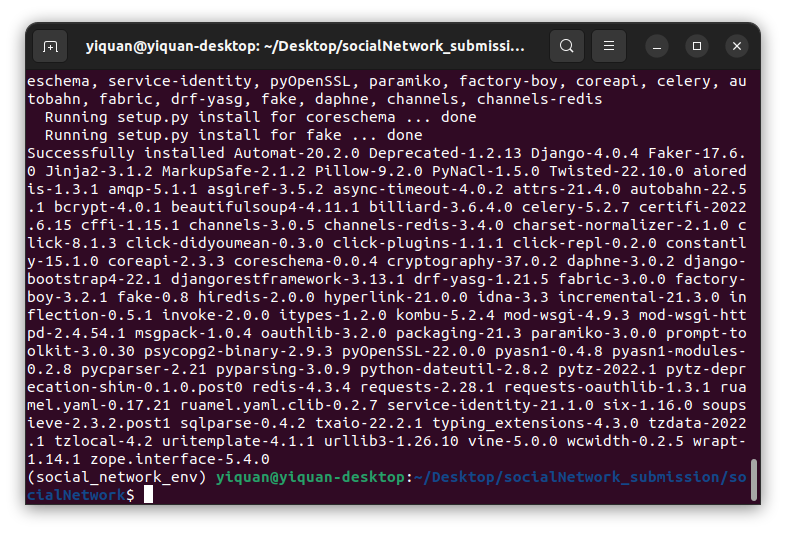
Right click in the submission folder(where installation.sh and requirements.txt is located) and click on open in terminal.

### 

Type the following command to create new environment and install the dependencies:

**. ./installation.sh**

Using **.** will run the script in the current terminal because we want the environment to be activated in our current terminal.



The packages should be installed and social\_network\_env will be activated.

### Setting Up Redis(skip this if it is installed and running)

In the terminal run the following command (installation instruction can be found [here](https://redis.io/docs/getting-started/installation/install-redis-on-linux/)):

**curl -fsSL https://packages.redis.io/gpg | sudo gpg --dearmor -o /usr/share/keyrings/redis-archive-keyring.gpg**

**echo "deb [signed-by=/usr/share/keyrings/redis-archive-keyring.gpg] https://packages.redis.io/deb $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/redis.list**

**sudo apt-get update**

**sudo apt-get install redis**

**sudo systemctl start redis-server**

**redis-cli ping**

If you get pong as a response, your redis is up and running, proceed to the next section.

### Running The Test Case

In the terminal make sure the env(created in Creating A New Environment section) is already activated. Run the following commands:

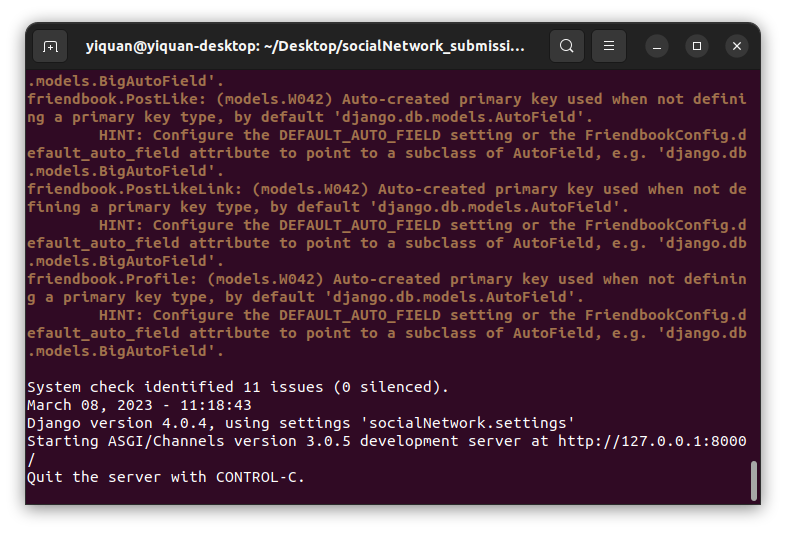
**python manage.py test**

### Running The Project

In the terminal make sure the env(created in Creating A New Environment section) is already activated. Run the following commands:

**python manage.py runserver**

You should see the same result in the terminal as the screenshot.

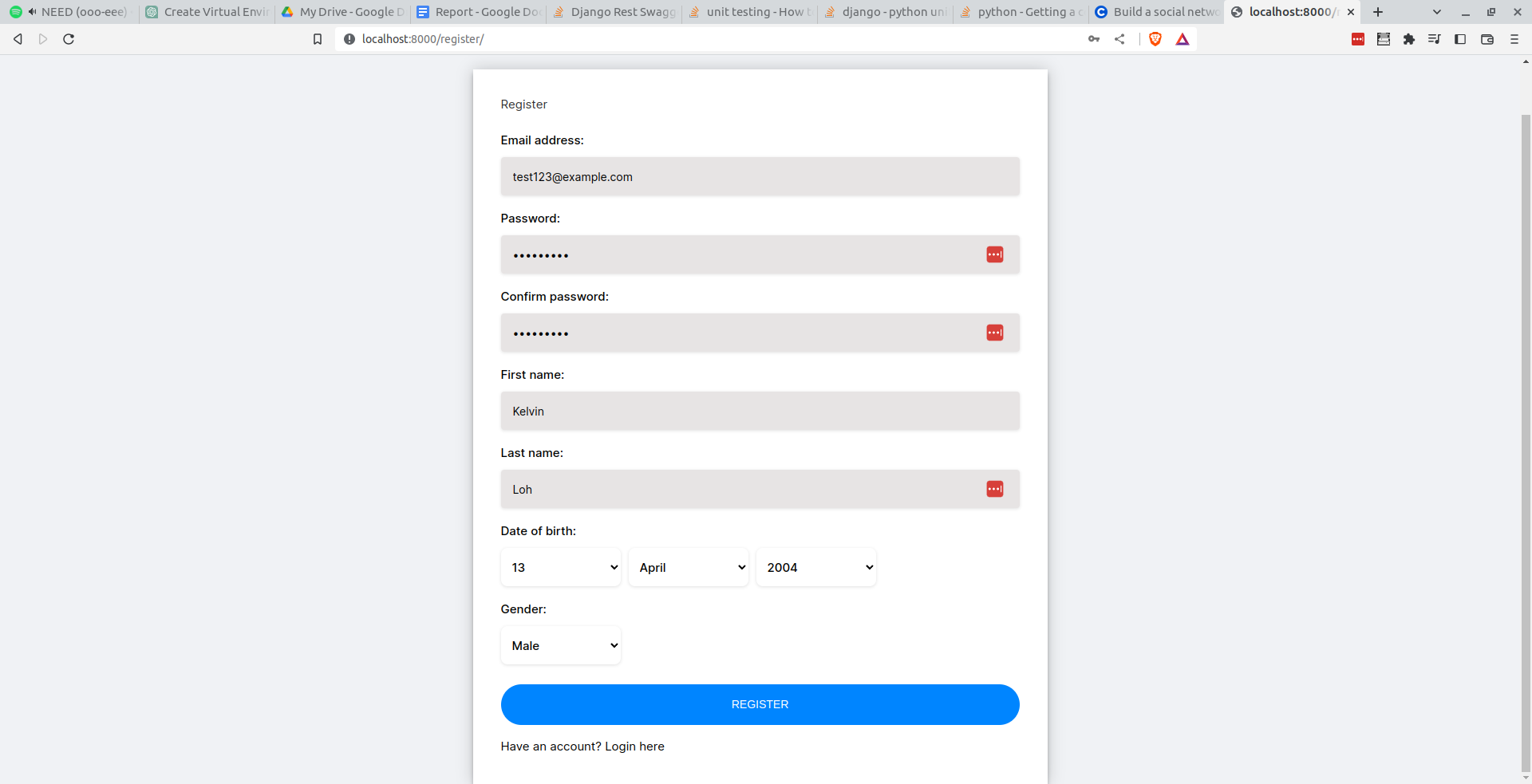


Ensure that the ASGI/Channels is running which is used for the chat websocket. Once you get the same result as the screenshot above, open up your browser and navigate to [localhost:8000/](http://localhost:8000/)

# Project Functionality Requirements

Before we go into the project functionality, I would like to say that I used some code from this [repository that provided](https://github.com/tomitokko/django-social-media-template) an empty template which includes the css and styling of a page. So the code that I used are the navigation bar(header) code in index.html, the post(status) styling, all the stylesheets that are in assets/css and the javascript in assets/js. I used the template file to speed up the designing of the project as they are logicless which means that implementation must be done on my own and changing the code to adapt to the django template(e.g. navigation bar is in nav.html so that base.html can consume it to display the navigation bar in every page, making post(status) to display dynamic data instead of hardcoded data). The javascript that is in the template is for the navigation bar and the logic for the image clicked from the post(status) to be displayed and navigate to a different image for that post.

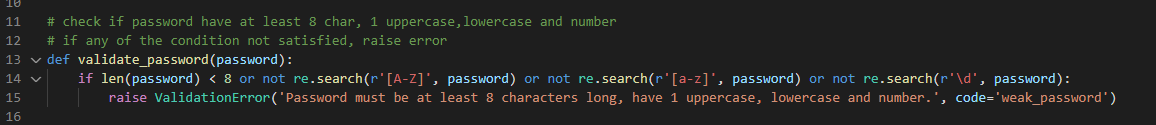
### R1a. Users can create accounts



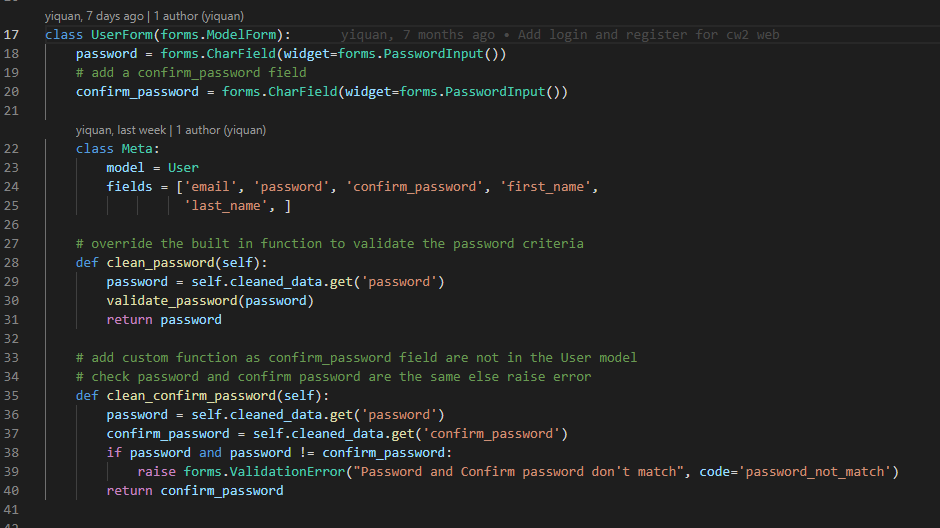
This is the registration page for my application.

##### How this was achieved

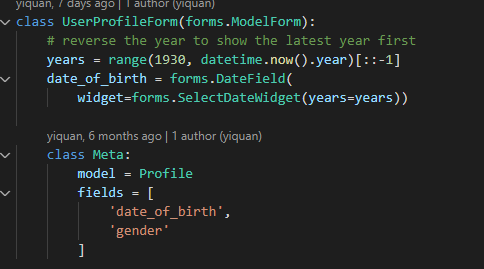
**forms.py**

****

validate\_password function takes in a string password and checks if the password is 8 characters long, has at least 1 uppercase and lowercase character and a number in the password. This is to ensure that user create a “strong” password although simple and common password can be used in this project.

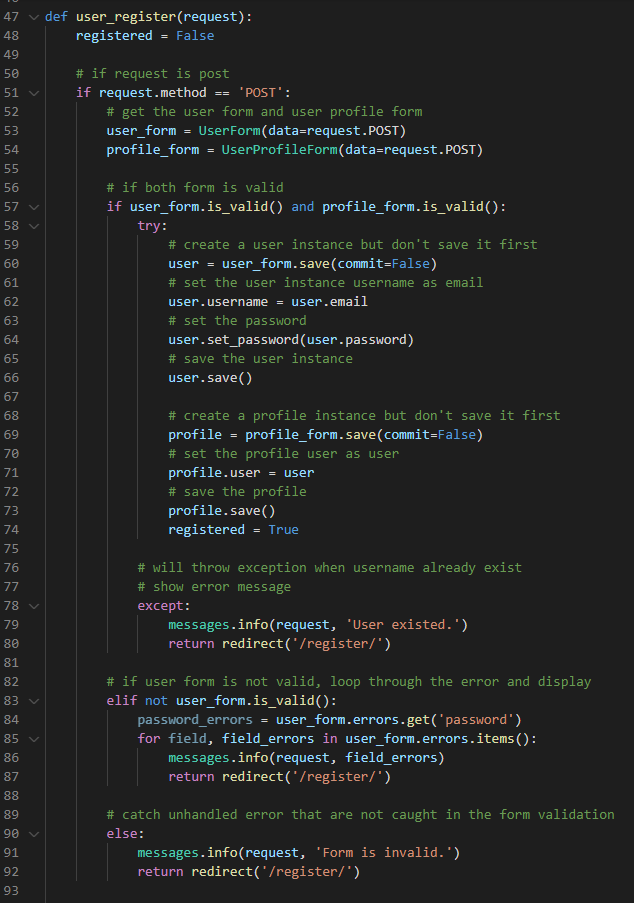
****

The above screenshot is user form, what this form does is generate email, password, confirm password, first name and last name field from the User model. The clean\_password function is to override the built in function so we can validate the password whether it met our password requirement. The clean\_confirm\_password is to declare a custom function to check for the confirm\_password. For django forms, field that are in the models have all this clean function inbuilt but for custom field that are not in the model we have to manually declare it else there will be an error. In this function, we just compare the password and confirm password to be the same if not it will raise a validation error which will be caught by the caller in views.py.

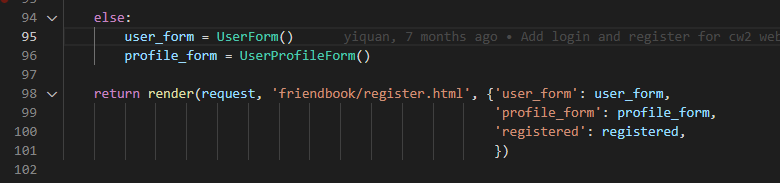
****

The above screenshot is user profile form, this form generates the date of birth and gender field from the Profile model. For the date of birth, I want it to display from the latest year to the earlier year so what i did was to range the year value from 1930 to current year then reverse it with [::-1]. After that in the date widget, replace the years with the years we generated from the range.

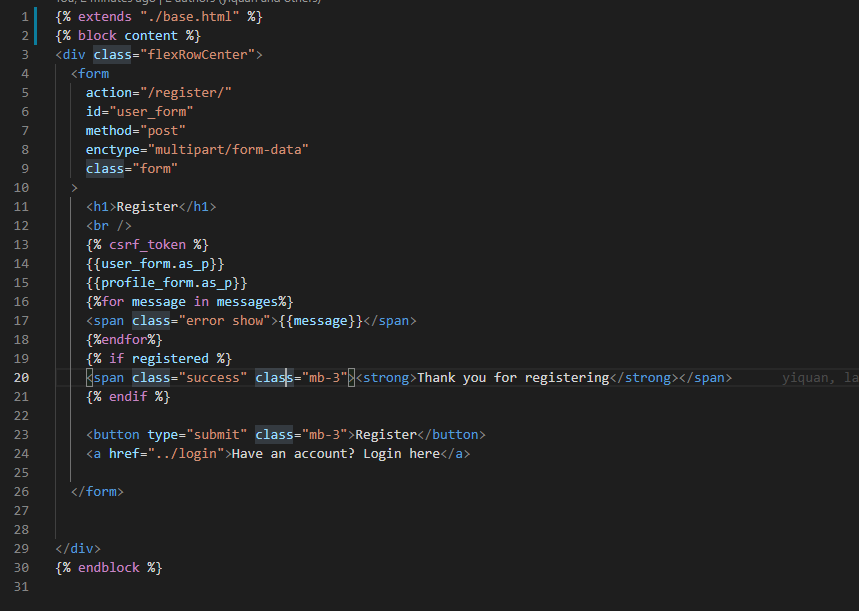
**views.py**



The above screenshot is the user\_register function that is in the views.py file. If the request is a post method, put the data into the UserForm and UserProfileForm and check if both are valid. If it is valid, save the user\_form but don’t commit because we want to set the username as the user email and set the password. After which we save the user and do the same for profile\_form by setting the profile user as the user before saving(committing to database) it. After which we set the registered variable to True which will render the success message in the html page. If there is an error for inserting the current user into the User table, this means that the username has duplicates as the username is unique so catch the exception and return an error for the html to render. Next if the user\_form is not valid, loop through the errors in user\_form and display it in the html page. If user\_form and/or profile\_form is invalid which shouldn’t be, will just display an error “Form is invalid.” in the html page in the registration page.



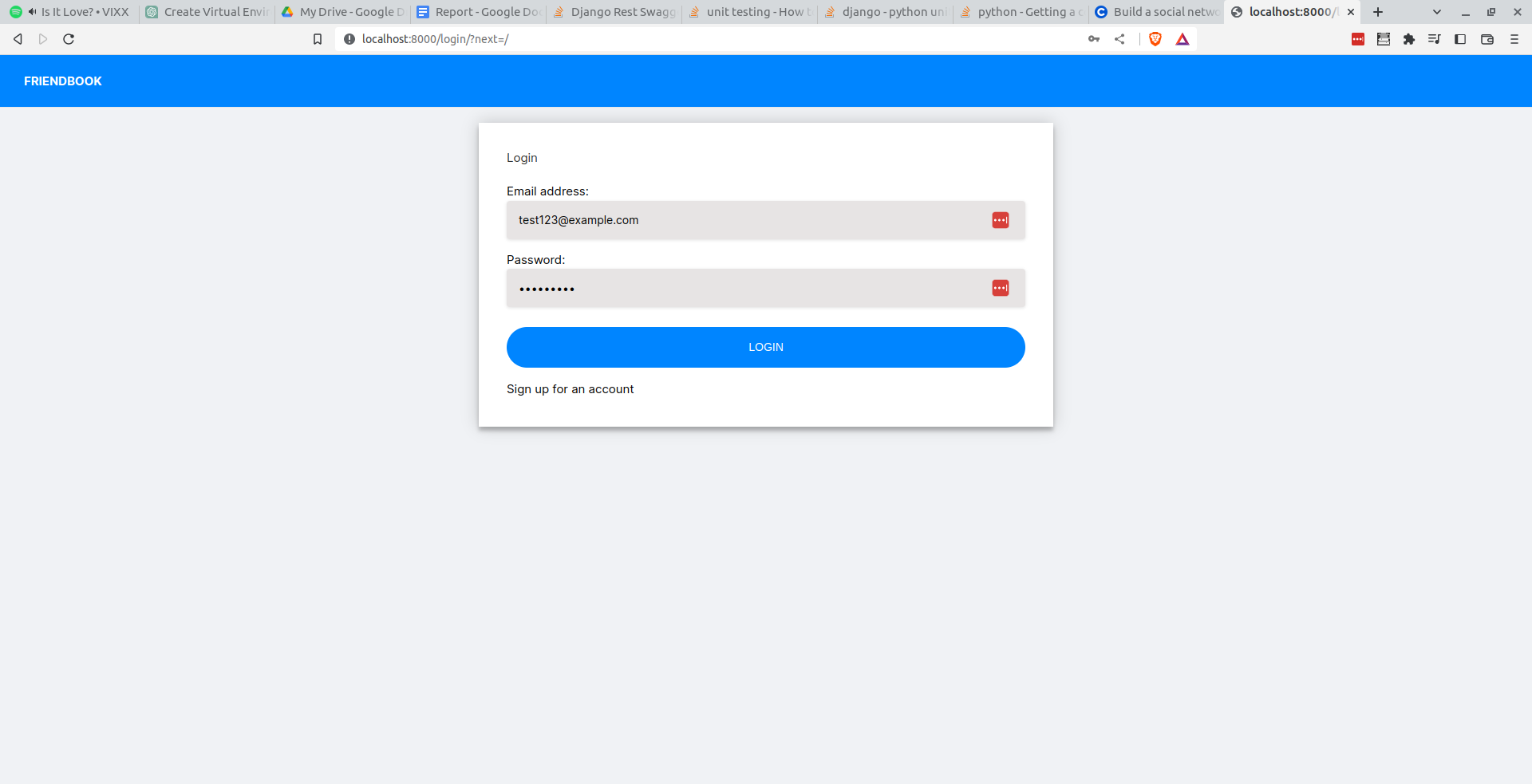
This code is a continuation of the user\_register function, if it is not a post function, pass in user\_form, profile\_form and registered into the register.html to render.



This is the register.html, what it does is mostly just displaying the user\_form and profile\_form as\_p will render the field to be in <p> tag and display any error message or registration success message.

### R1b. Users can log in and log out

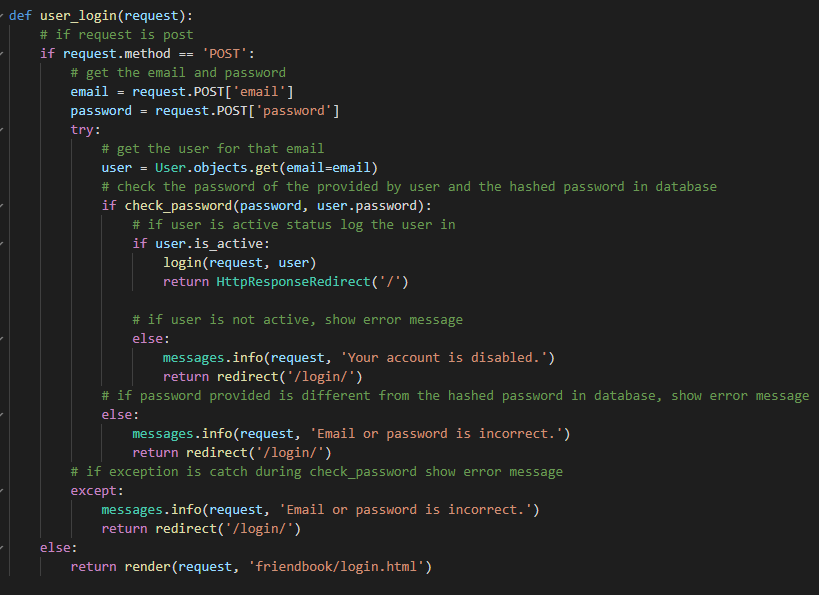
**Users can log in**



This is the login page for my application. User can login their account in this page to post status, images, see other people's status, add friends and chat with friends.

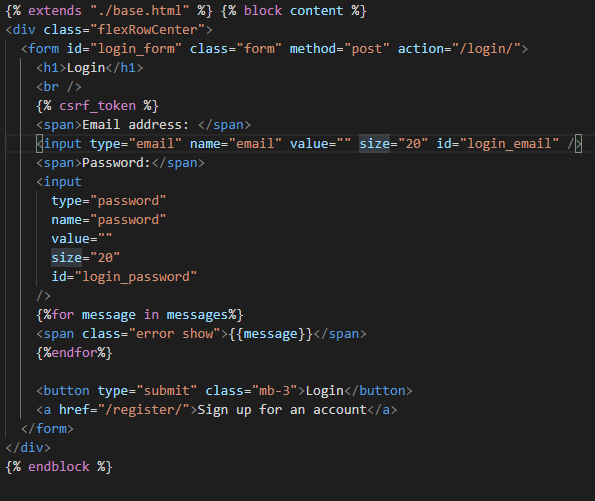
##### How this was achieved

**views.py**



The user\_login function can be found in the views.py file. So what this function does is if the request method is not POST, it will render the login.html else it will get the email and password from the request. After that we will get the user from the User table, this will give us access to the password hash of the user. We will use the check\_password function which is from django.contrib.auth.hashers and pass in the raw password(from the request) and the hashed password(from the database). If the password is correct and user status is active, log the user into the request and redirect them to the homepage, else we will return the error as “Email or password is incorrect”. This error message is returned when the password is wrong or username doesn’t exist, reason being in real life scenario we do not want anyone to guess if an email has an account in the platform. Just something out of this module is that there is this [website](https://haveibeenpwned.com/) that checks if your email or number has been compromised or not. When we register on other websites with our email, we do not know how they secure their backend(password can be plain text, server access leak etc..) so if they get compromised, hackers can use that information and try to access important services like email and other main important services. So always use a password manager(yes I know lastpass got hacked but if the master password is strong it will take years for them to crack) generates a random password for any website credential and avoids reusing the same password.

**login.html**

****

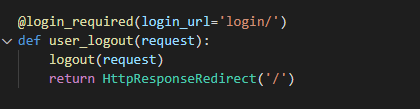
This code is in the login.html, this html contains 1 email field, password field, a for loop to display the error message, a login button and a hyperlink to register page. When the form is submitted, it will post a request to login/ alongside with the email and password.

**User can log out**

TODO INSERT LOGOUT SS

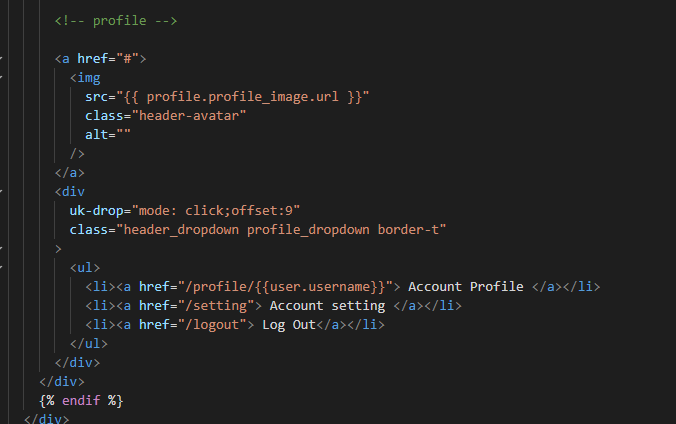
##### How this was achieved

**views.py**



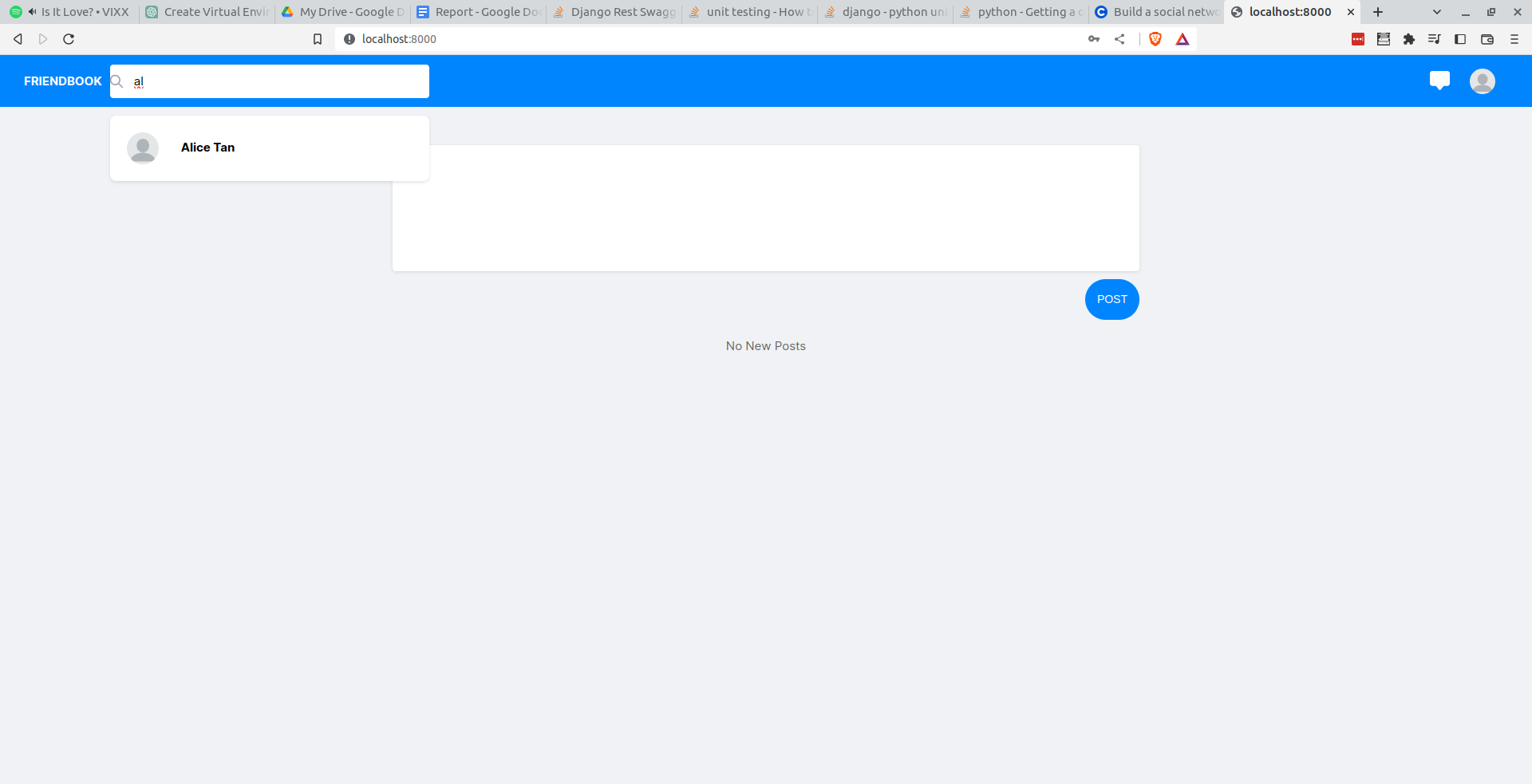
The user\_logout function can be found in the views.py. This function just log user out from the current session with the requirement to be logged in.

**nav.html**



This code can be found in nav.html, this top navigation bar item will only appear if the user is logged in. The log out is nested under the profile avatar, user can logout by clicking the logout link.

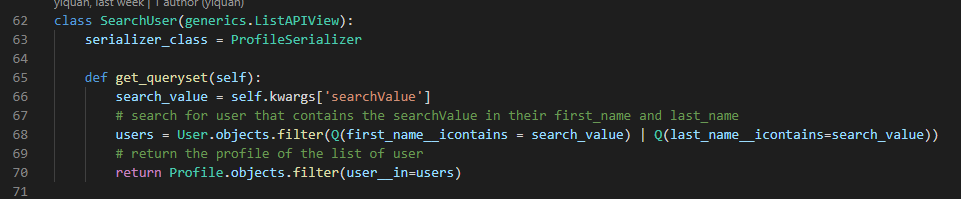
### R1c. Users can search for other users



This is the search bar where user can search for users that are on the platform.

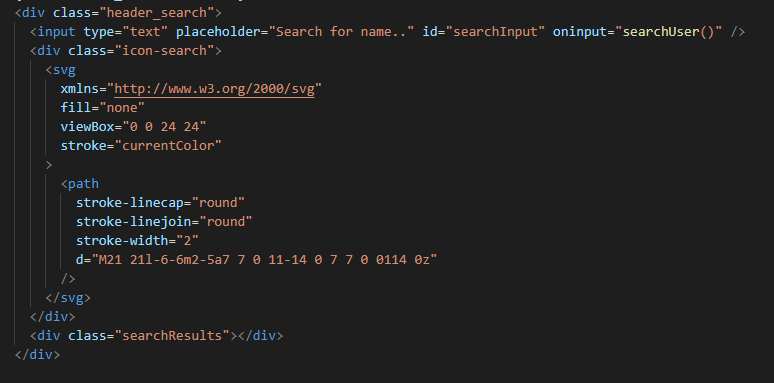
##### How this was achieved

**api.py**

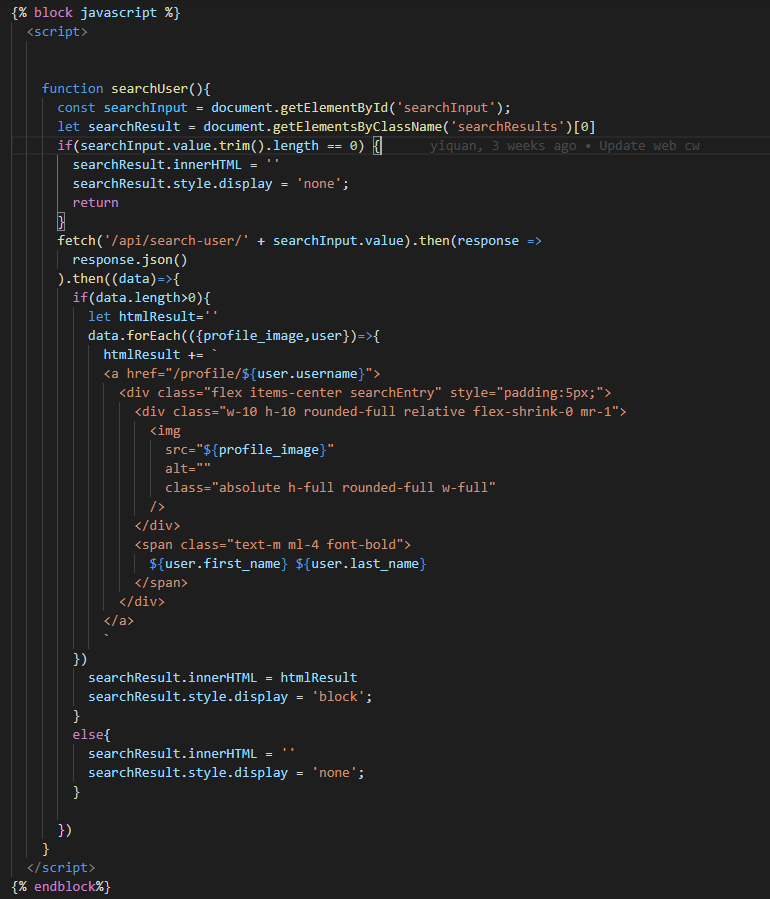


This SearchUser class can be found in the api.py file, basically the class search user uses the ProfileSerializer and returns a list. The get\_queryset allows us to query anything we want, so I took out the search value first from the request which is in self.kwargs and filter it by first\_name or last\_name. This means that if the first\_name or last\_name contains the search value, it will be returned. I used icontains because it will ignore case sensitive and if any part of the first\_name or last\_name contains that search value it will be returned. The url for this api is api/search-user/<str:searchValue> in the urls.py.

**nav.html**

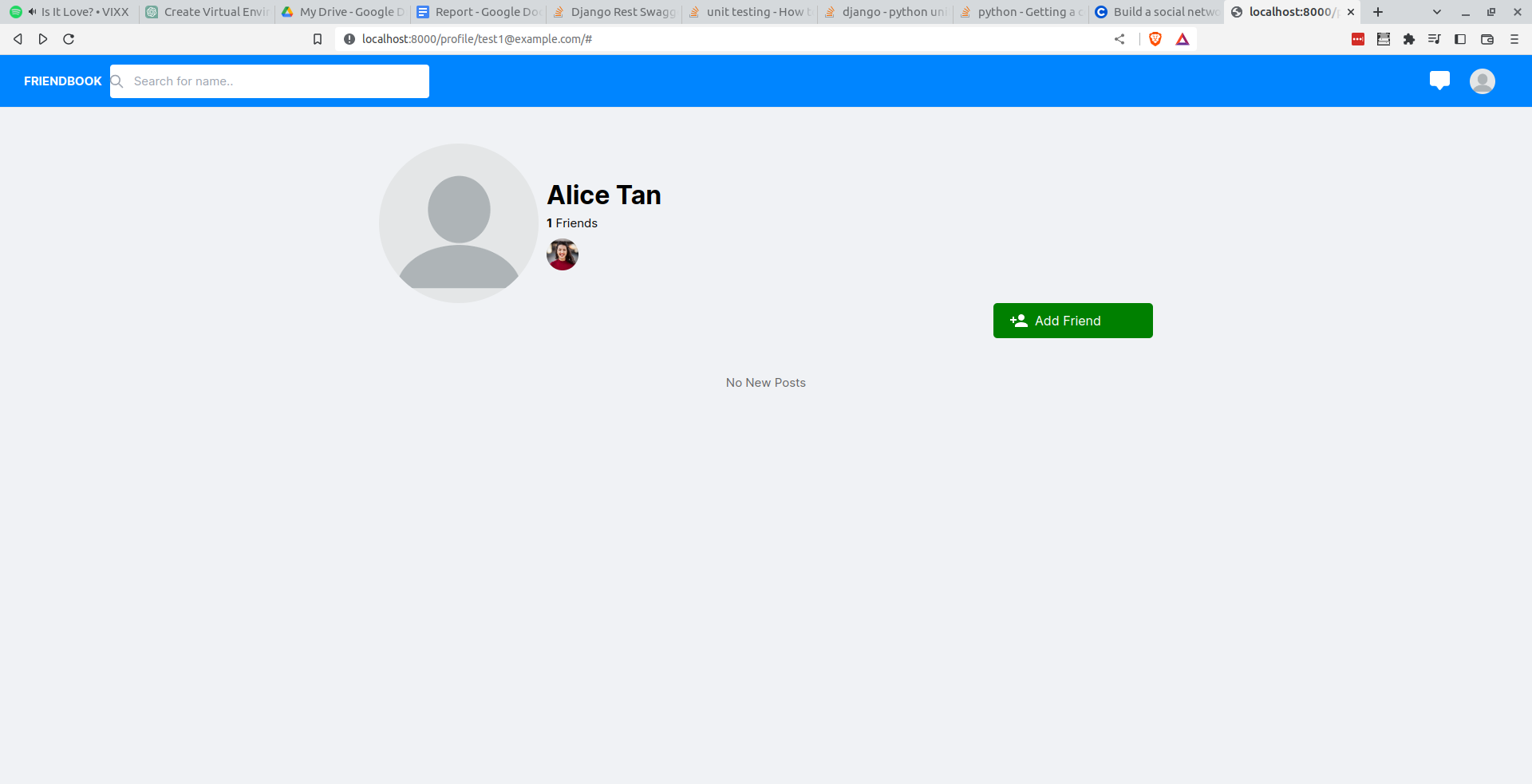


This is the html for the search bar which is located in the nav.html file. The search bar is made up of a text input, input and a searchResults div. When the user type a letter, the input will trigger the searchUser function which is the next screenshot.

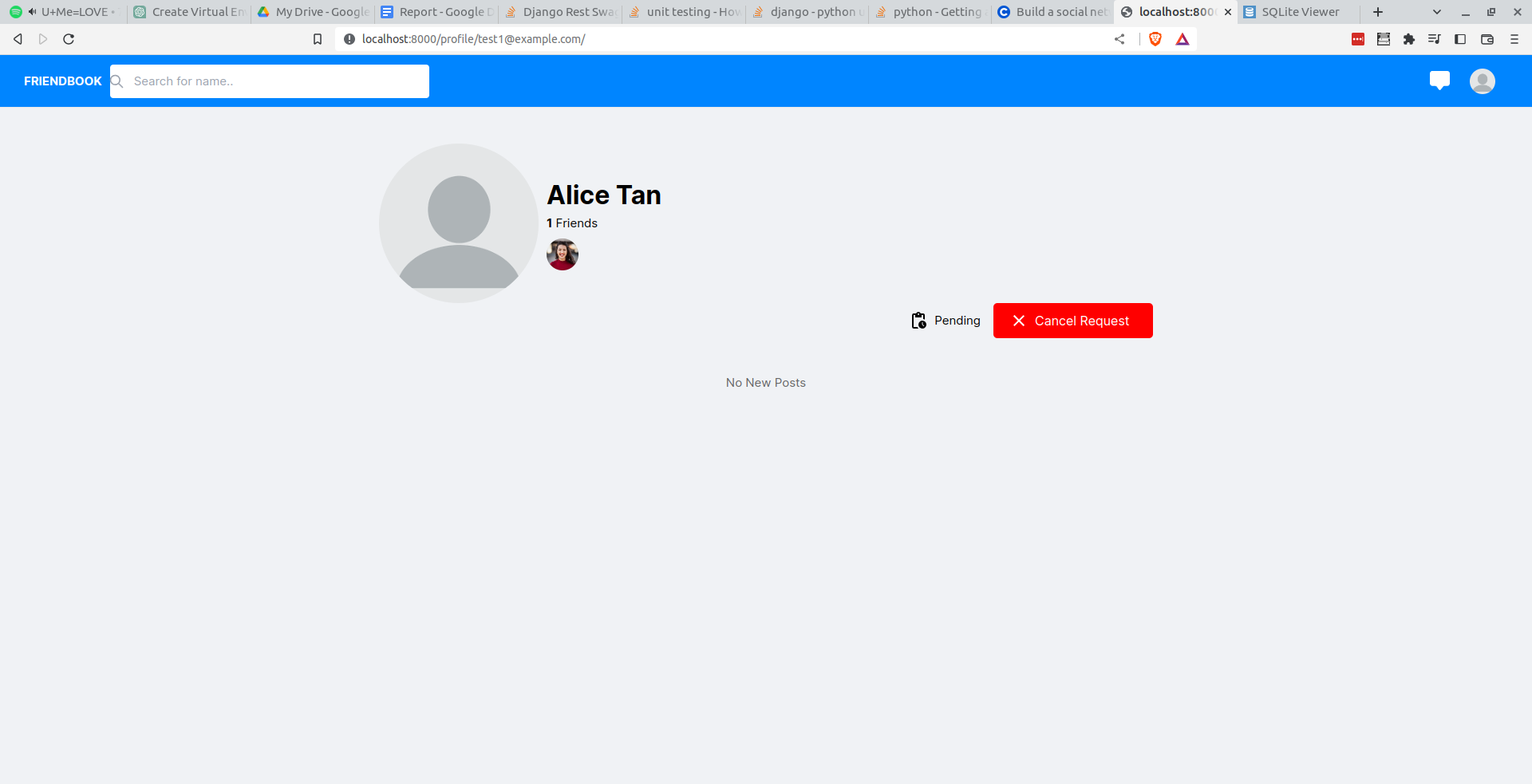


The following code can be found in nav.html, what this function does is that when the user types something in the input, we get the searchInput and searchResult element. Next we check if the searchInput value is empty, I have used the trim function to remove the white space from the front and end of the string to make sure the user keyed in something and not just whitespace. If the value is empty, the searchResult inner html will be set to empty string, set the display of searchResult div to be empty and return the function. If the searchInput value is not empty, we call the /api/search-user/ endpoint and pass in the searchInput value into the URI parameter and do a fetch. Because the fetch method is asynchronous so we have to use .then function to get the result. In reactjs we can just declare the function as async and use await for fetch to wait for the response but in this javascript we cannot. There is one way to make the function async which is to use iife(Immediately Invoked Function Expression). Basically is wrapping parenthesis and calling the function immediately (e.g. (async function searchUser(){})() however, if I remembered correctly when I tried implementing this my input is unable to call the function which I decided not to use iife for this. So after getting a response from the api, we convert it to json() but this function is also asynchronous so we have to .then again. If we have used an async function, we don’t have to have so many .then as the await keyword will wait for the asynchronous function to finish before executing the next line of code. After we get the json result, we can then check if the result has at least 1 record or not, if it doesn’t have any record we will just set the searchResult inner html to empty string and the display to none. The reason we do this is that the user might have received results from their previous search but with 1 more character in the search there might be no result returned so we have to remove those previous results. So if there are results from the response, we will loop each record and append(add) a html string into the htmlResult string. I used html string instead of using the javascript way where you create the element(document.createElement) and add each attribute and inner html which is very painful and a lot of lines. So each entry is a hyperlink which has href value of /profile/user.username. I have used `` instead of “” or ‘’ because it is easier to do string interpolation rather than using the plus sign. Inside the hyperlink there is a div which contains the profile image and the name of the user. After all the entry has been added to the htmlResult variable, we just have to set the searchResult inner html to htmlResult and set the display to block so user can see the result.

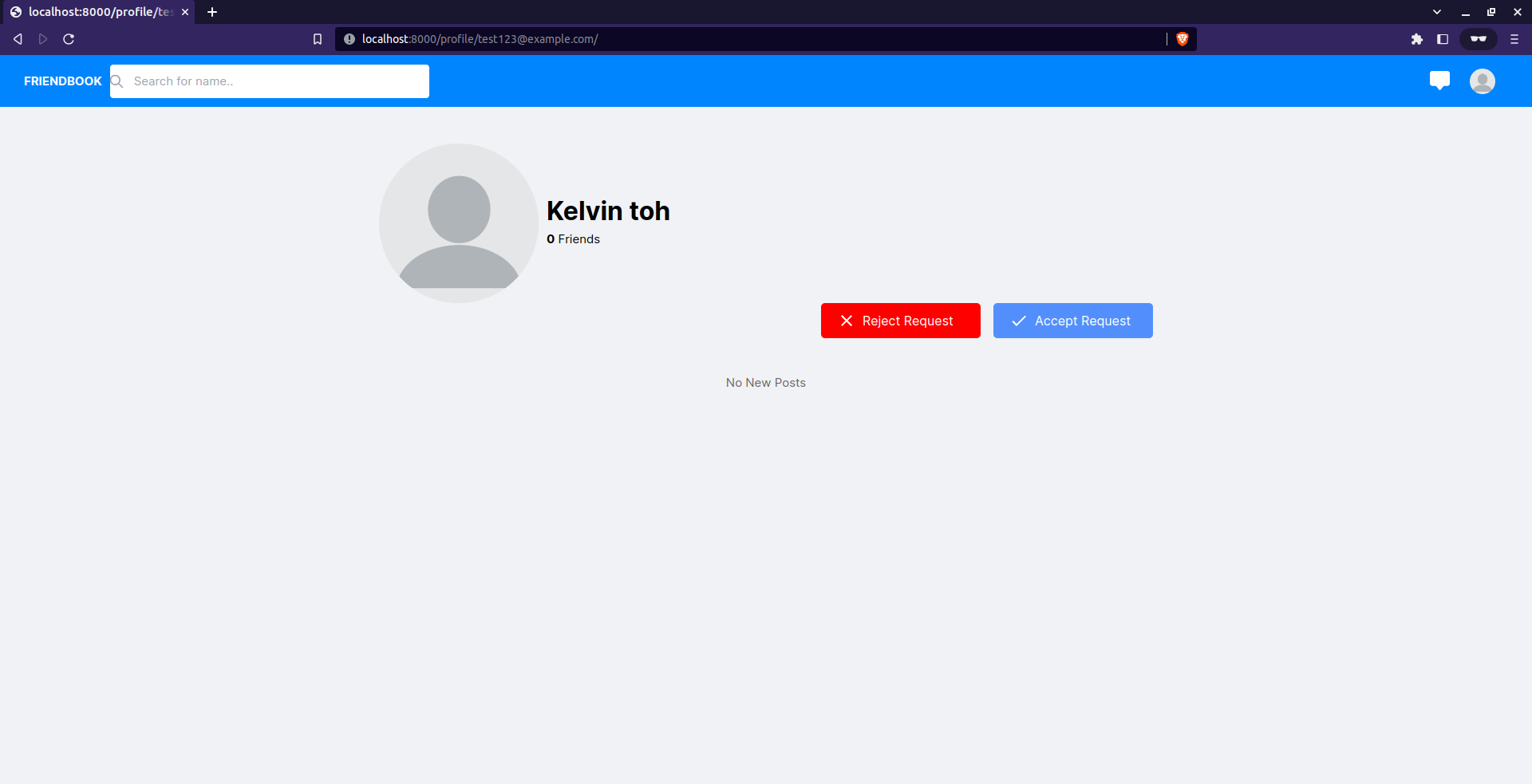
### R1d. Users can add other users as friends



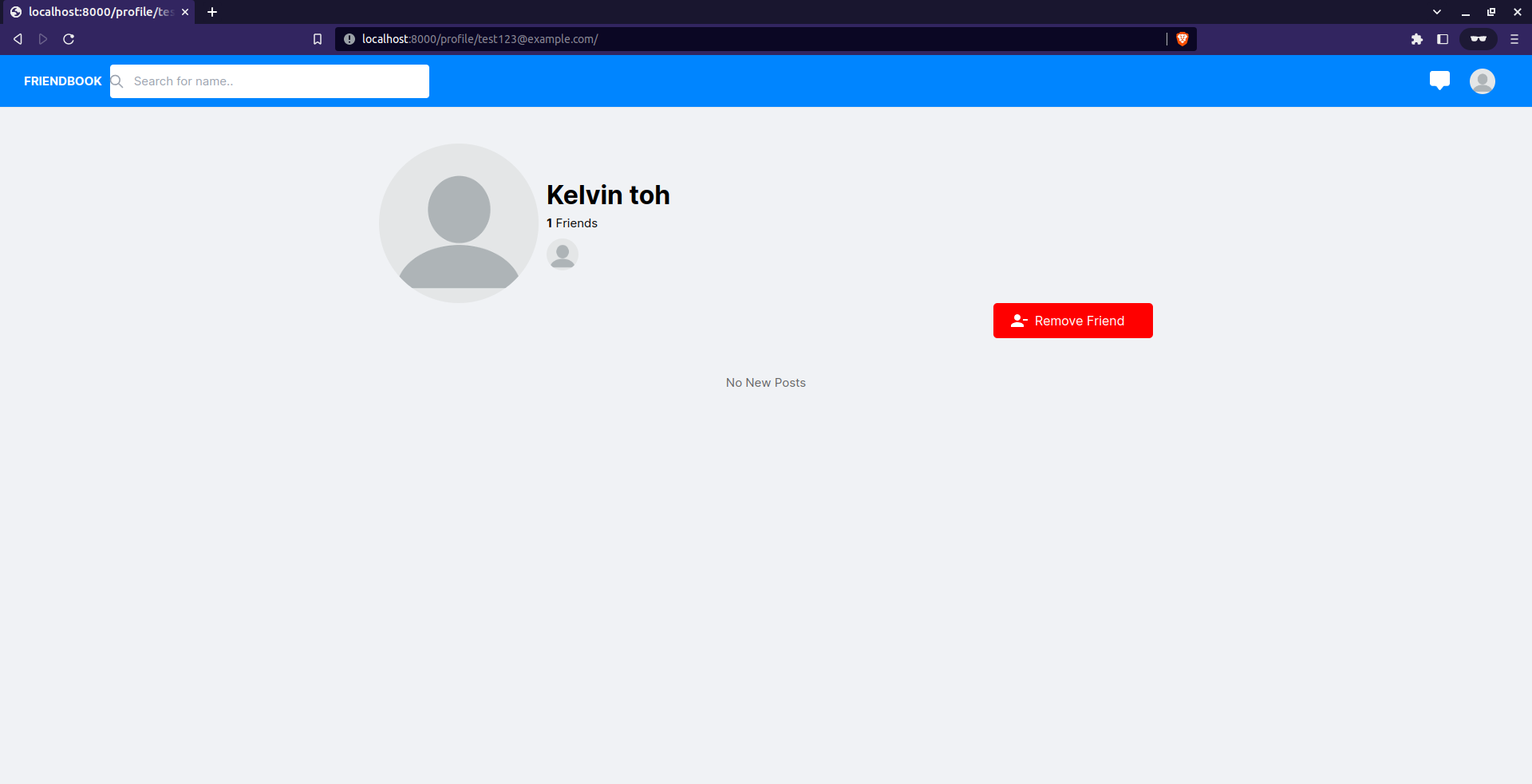
In the user profile page, the current user can send friend requests to a user. In this screenshot, I am Kelvin toh and I visited Alice Tan page.



After I click on the “Add Friend” button, I will see the “Cancel Request” button and a “Pending” text beside the button instead of “Add Friend”. This indicated that I have successfully send the friend request to Alice Tan.



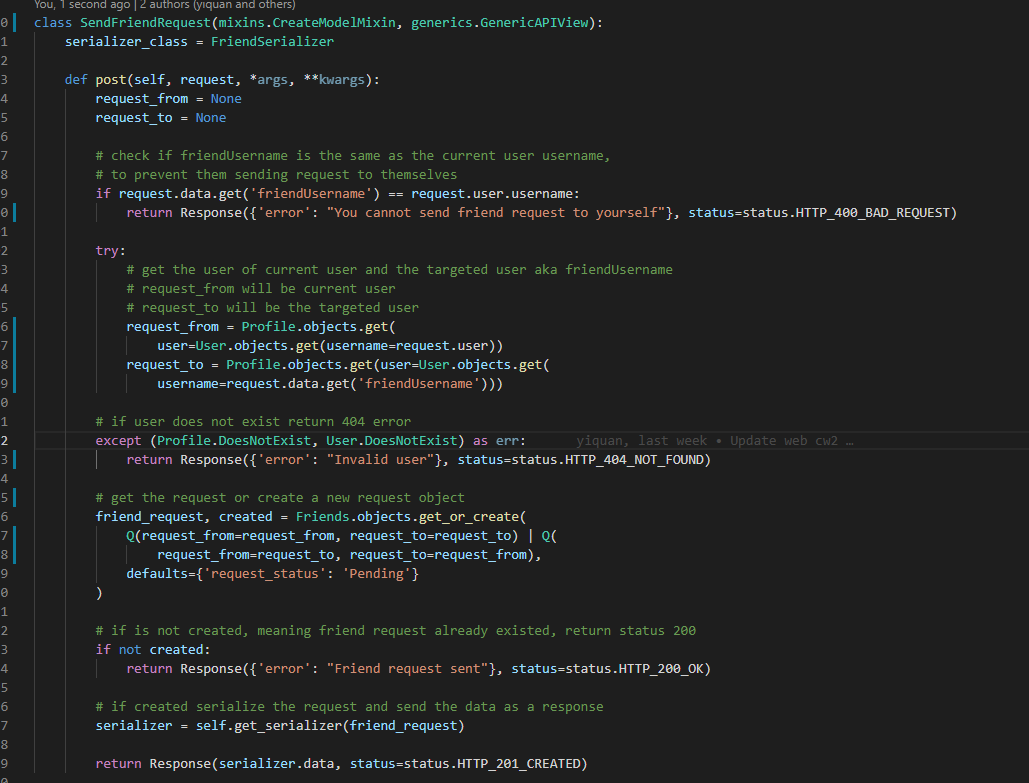
The current user I am using in incognito is Alice Tan, when I visit Kelvin toh profile page, I will see 2 buttons which are “Accept Friend” and “Reject Request” button.



When I click on “Accept Request”, I would only see the “Remove Friend” button which indicated that I have successfully added Kelvin toh as friend.

##### How this was achieved

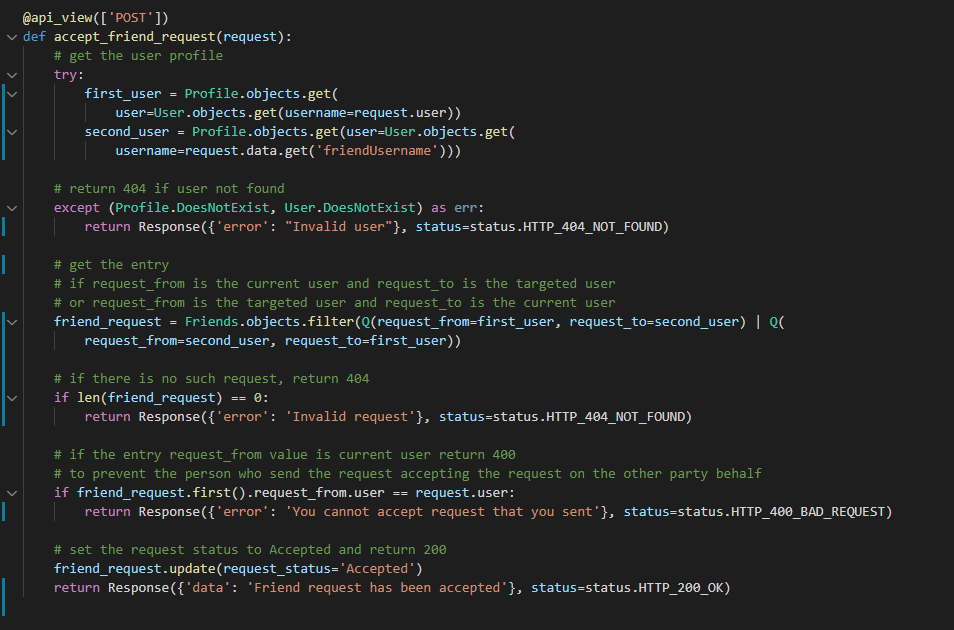
**api.py**

****

This is the sendFriendRequest class which can be found in the api.py file. This api uses the FriendSerializer and it only accepts post requests. First, I declared a request\_from and request\_to variable outside of the try except. This is because I came from a javascript background where variables declared in try catch cannot be accessed outside of the try catch as this is called block scope and thinking that try except it the same. However, towards the end I accidentally discovered that variables that are declared in try except can be accessed outside of the try except scope, I was shocked. Reason being it is a bad practice to be accessing variable like this as it will become like javascript var which is bad as es6 have introduced the let and const variable which are block scope. Why it matters because multiple try catch can mutate variables that are declared in a try catch which can cause bugs and be hard to spot. I apologize for my complaint back to this, inside the try except scope we will retrieve the current user and the recipient user data. If any of the 2 users are not in the database, return a status 404 error as the response. If both of the 2 users exist, we will use the get\_or\_create function here, if an entry of either the current user is request\_from or request\_to and the recipient user is request\_from or request\_to exist, it will get the entry as friend\_request else it will create a new object with the current user being the request\_from and the recipient user being the request\_to with a default value of “Pending” for request\_status. The created variable will return False if there is entry found in the Friends table and will return True if the object is created. If the entry existed(if created is false) we will just return a status 200 response but if the object is created, we will serialize the object and return the data with status 201 response.

****

This is the reject\_friend\_request function which can be found in the api.py. Basically this function will delete the request(rejecting the friend request), so the first thing to do is retrieve the current user and the recipient user. If either one is not found, return a status 404 response. If both are valid user, retrieve the request either the current user is request\_from or request\_to and the recipient user is request\_from or request\_to. After retrieving the request we use the delete function to delete the record and return a status 200 response. I have used the filter function instead of get, the difference is that using a filter function and delete will not raise error when there are no record whereas using a get and delete will raise error if the record doesn’t exist. I kept the except but the code shouldn’t be able to reach that except from what I know.

****

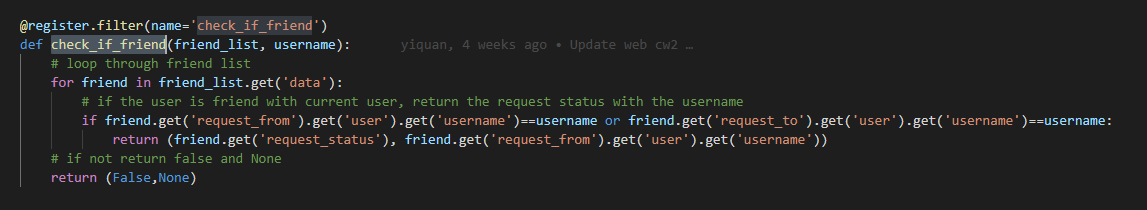
This is the accept\_friend\_request function that can be found in the api.py file. This is where I discovered that the try except declared variable can be used outside of the scope but we will retrieve the current user and recipient user. If either user doesn’t exist, return a status 404 response. If both user exist, we will get the friends object using the filter function where the current user is the request\_from or request\_to and the recipient user is request\_from or request\_to. Filter will return a list so if there is no record, return a status 404 response. In our scenario, we assume that there are only 1 record of the current user and the recipient user however, 2 records can exist if both users send a request at the same exact time which I will cover more in the later report on how to handle this. So assuming the list only return us 1 record, we will check the first record only where the request\_from user is not equal to the current user. Reason is that if this clause is not here, the person who send this request can use postman or other ways to post to this api and accept the request on behalf of the recipient user which is a flaw. So if the current user is the request\_from user we will return a status 400 error else we will update the request\_status to “Accepted” and return a status 200 response.

**views.py**



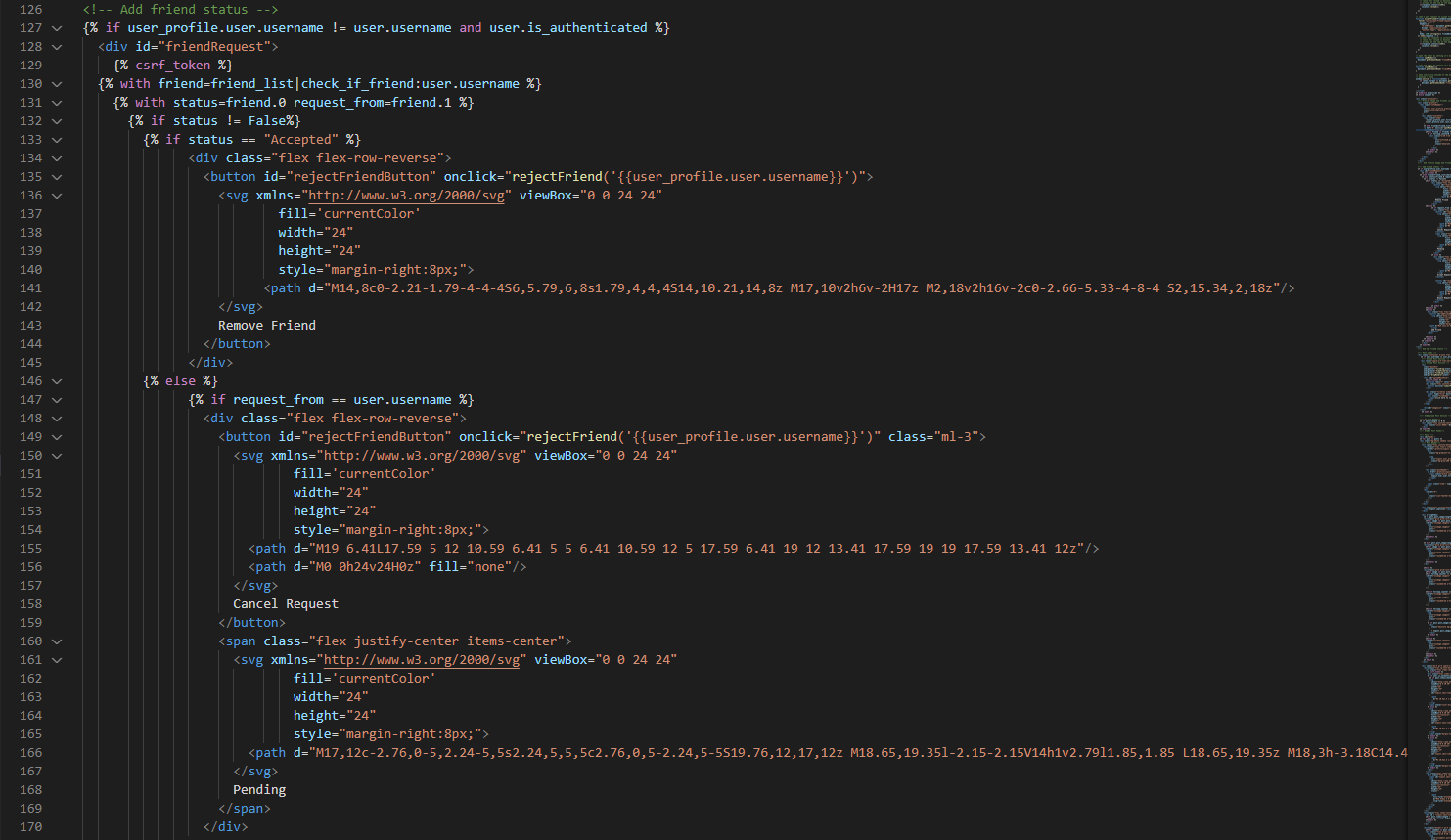
This is the user\_profile function in the views.py file. What this function does is to retrieve from getPost, getUser and getFriendList api and render to the profile.html. All those api I will cover in the later part of the report.

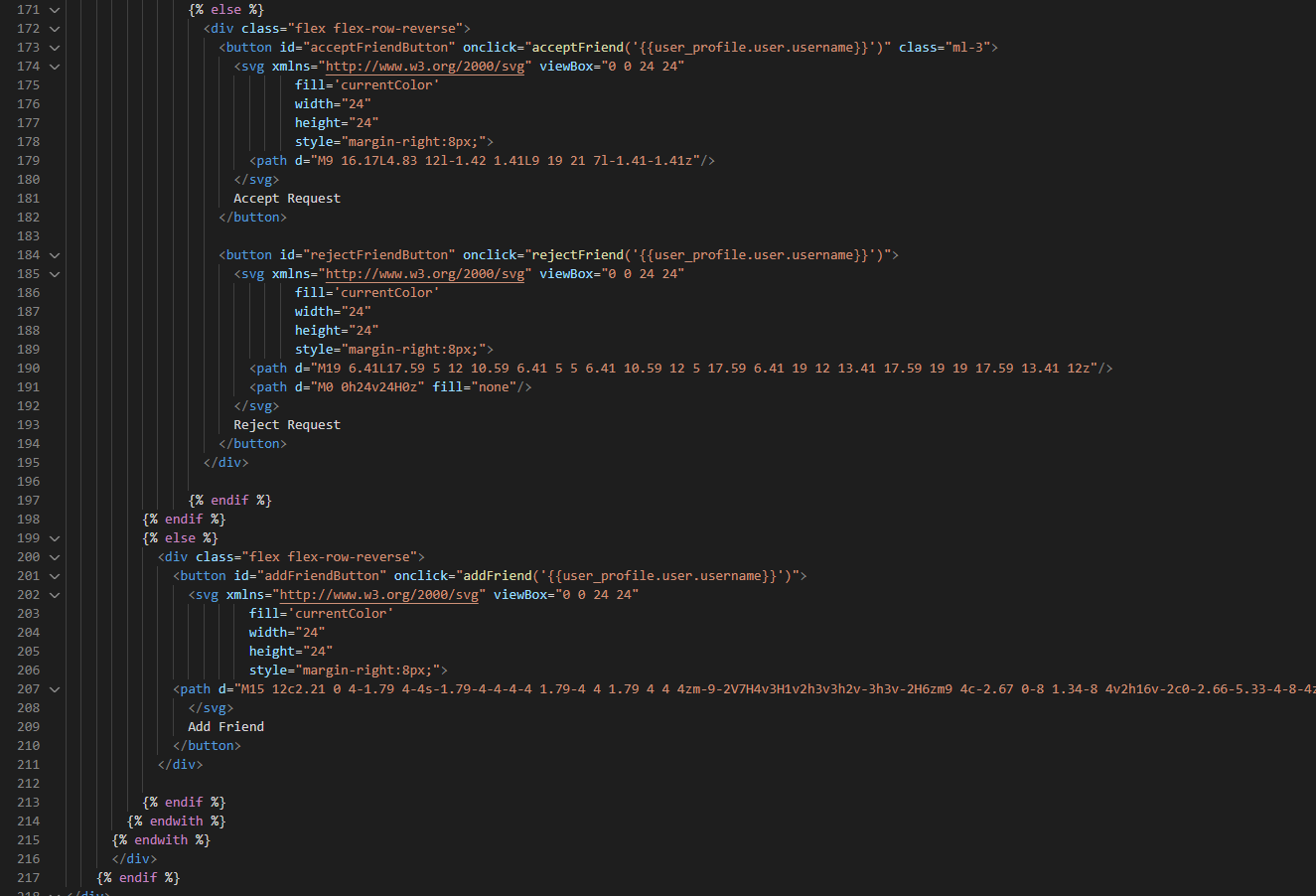
**custom\_tags.py**



This is the check\_if\_friend function which can be found in the custom\_tags.py file. It takes in a friend list and a username, what this function does is to see if the username passed in is a friend in the friend list. So we loop through the friend\_list and check if the username is a request\_from or request\_to and if it exists, return a tuple of request\_status and the request\_from user value. Else, return False and None.

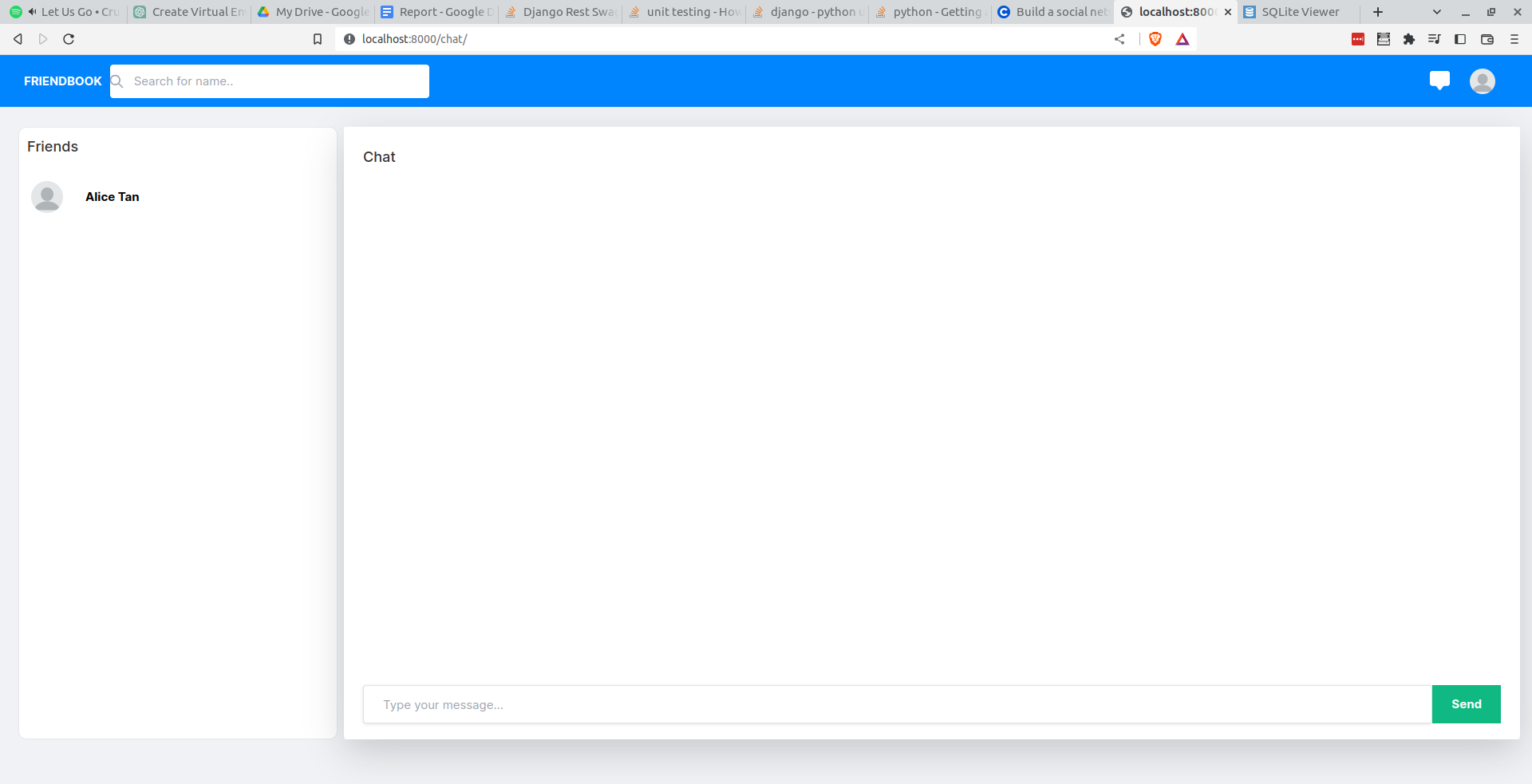
**profile.html**



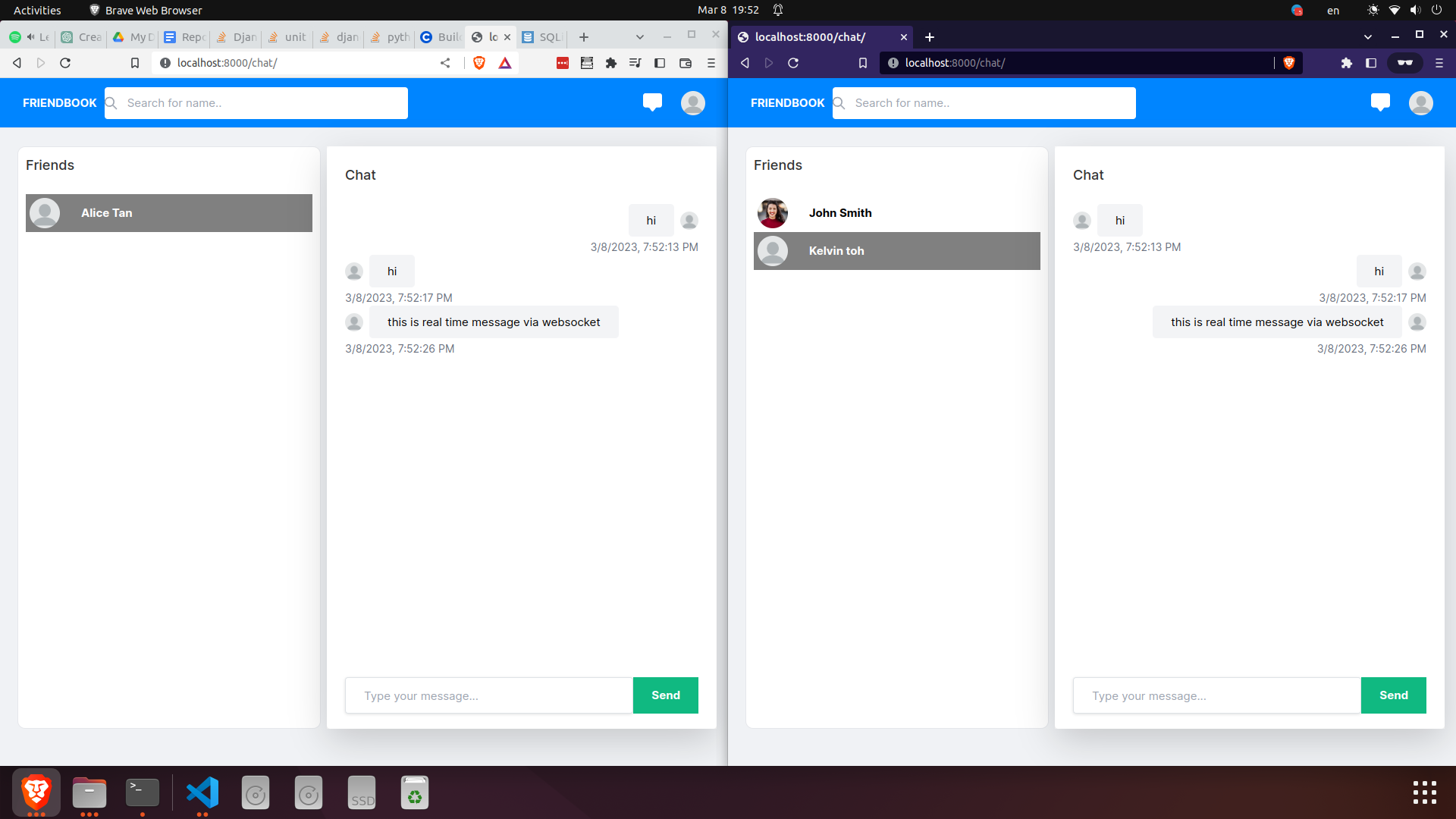


This is the part where buttons are being rendered, so we only render the buttons if the current user is not the profile user and they are logged in. After which we call the check\_if\_friend function passing in the friend\_list(current user) as the first parameter and the current user username.

### R1e. Users can chat in realtime with friends



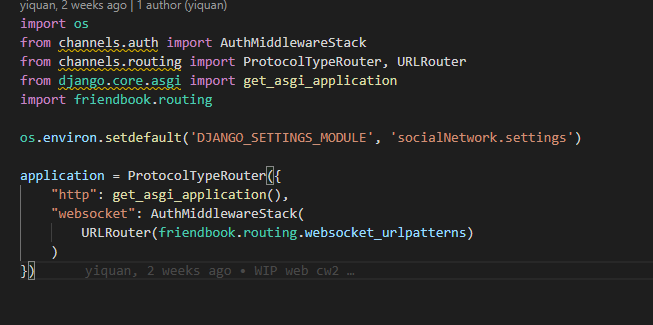
For the next few requirements I will keep it brief because after writing finished R1d, my word count is at 3200 words which is not enough as I have yet to reach the 50% of the report. This is the chat page where user can chat real time with their friends. It can be accessed by clicking on the chat icon beside the profile icon. Note that friend refers to recipient who have accepted their friend request.



When the user clicks on a friend in the chat page, it will retrieve all the past chat messages and when the user sends a message to the friend, it will be realtime via websocket.

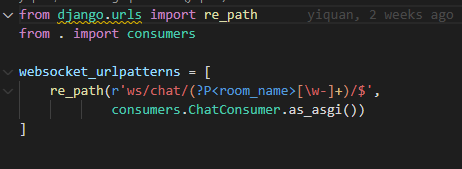
##### How this was achieved

**socialNetwork/asgi.py**

****

This code is found in asgi.py, this will create an instance of a websocket with the websocket URI.

**routing.py**

****

This is the routing.py which is the websocket path for the chat. It uses regular expressions where it can be a word or hyphen(-) with many occurrences. The hyphen(-) is here because the room\_name i save into the database is a uuid4 so we will have to match the hyphen. When the path matched, it will call the consumers.ChatConsumer and convert it to ASGI Application.

**consumers.py**

I will not screenshot the consumers.py but the file can be found in the friendbook folder. The code in that file would basically create a connection for that room name and when there is a message being sent to the websocket URI, it will triggers the receive function which will then send the message, room\_name and user to the chat\_message function. In the chat\_message function we will get the chat object with the room\_name and create a ChatMessage object with the message and the user. After which add it to the chat object, save it and broadcast the message data(serialize the ChatMessage object) to all the users that are in the room.

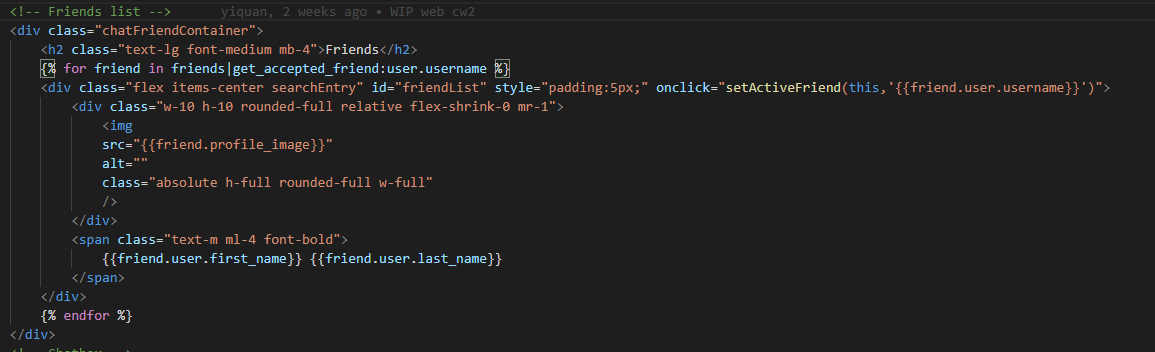
**api.py**

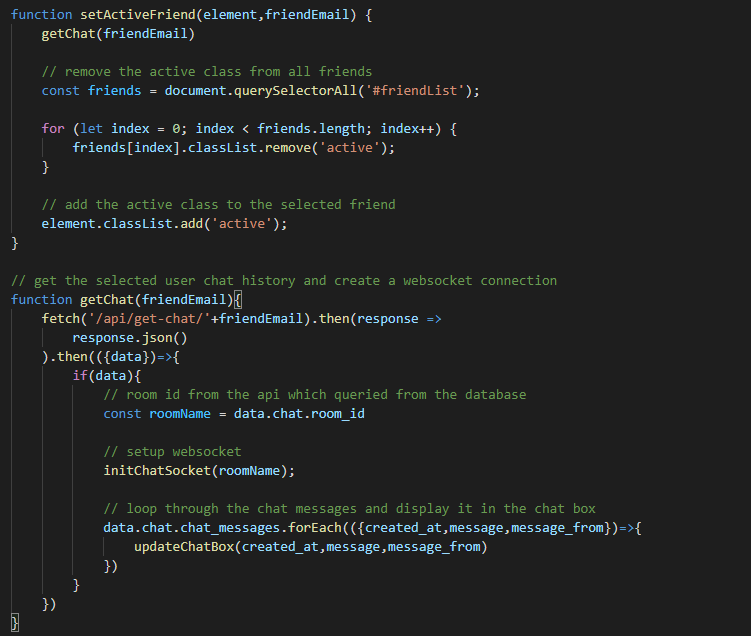
****

This is the get\_chat function which can be found in the api.py file. What this function does is to get the chat messages of a chat. I will not explain the code as there are comments there but if there is no chat between 2 users, this function will create a chat object for the 2 users alongside with an auto generated uuid4.

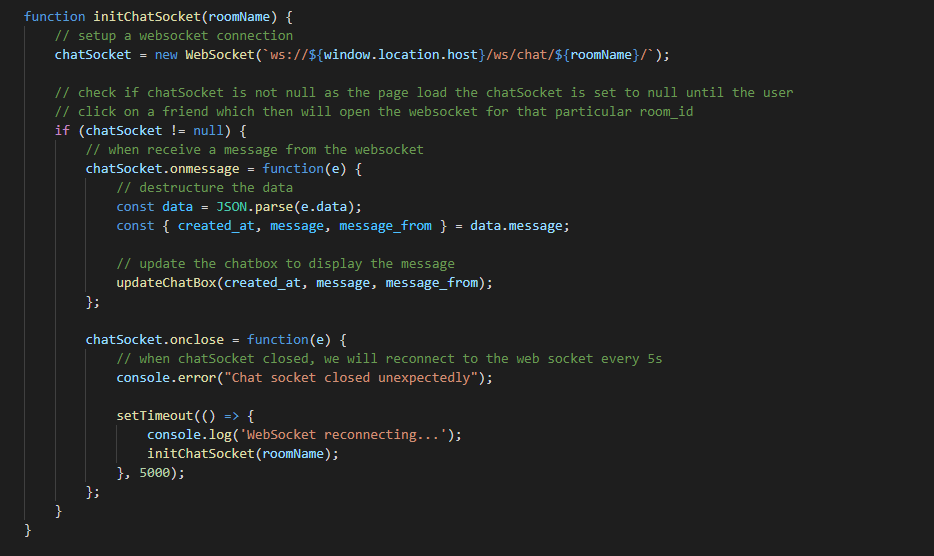
**chat.html**

All this code can be found in the chat.html. I will keep the explanation short so in the html, this is the portion where all the friends are displayed.

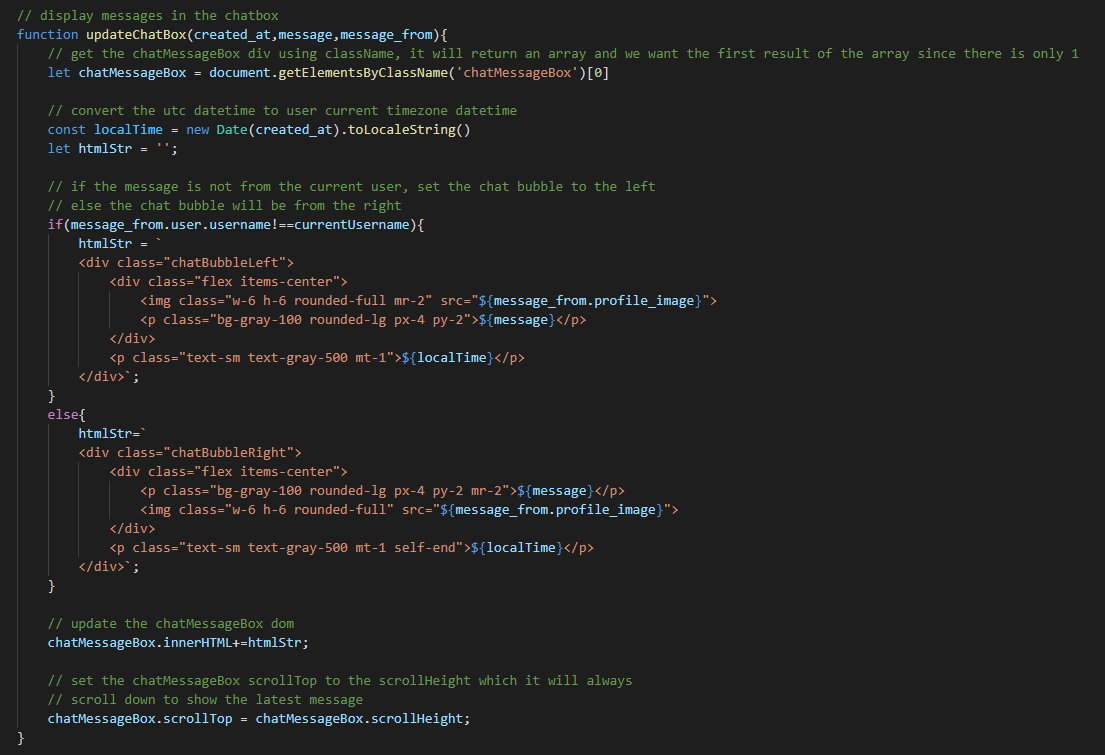
 The list will display each friend's profile image and their full name with onclick function binded to the div. When the user clicks on a friend, it will call the setActiveFriend function.



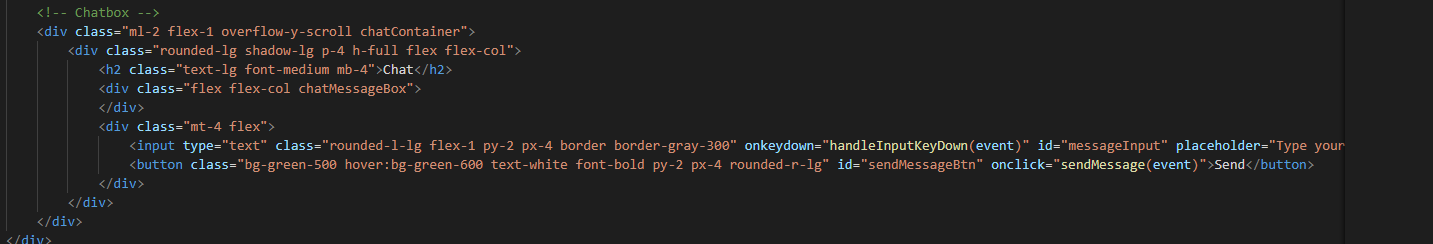
The setActiveFriend function basically call the getChat function and sets the selected friend to active with css(setting the background color). The getChat function will the /api/get-chat URI with the selected username. The api will retrieve all the chat messages and the room id, so we initialize the websocket and call the updateChatBox function.



The initChatSocket function will initialize a web socket with the retrieved room name and add in all the listener functions.



The updateChatBox function will get the chatMessageBox and display the message according to who sent the message(message\_from). If the message\_from is the current user, the message will display at the right side, else it will be on the left side.



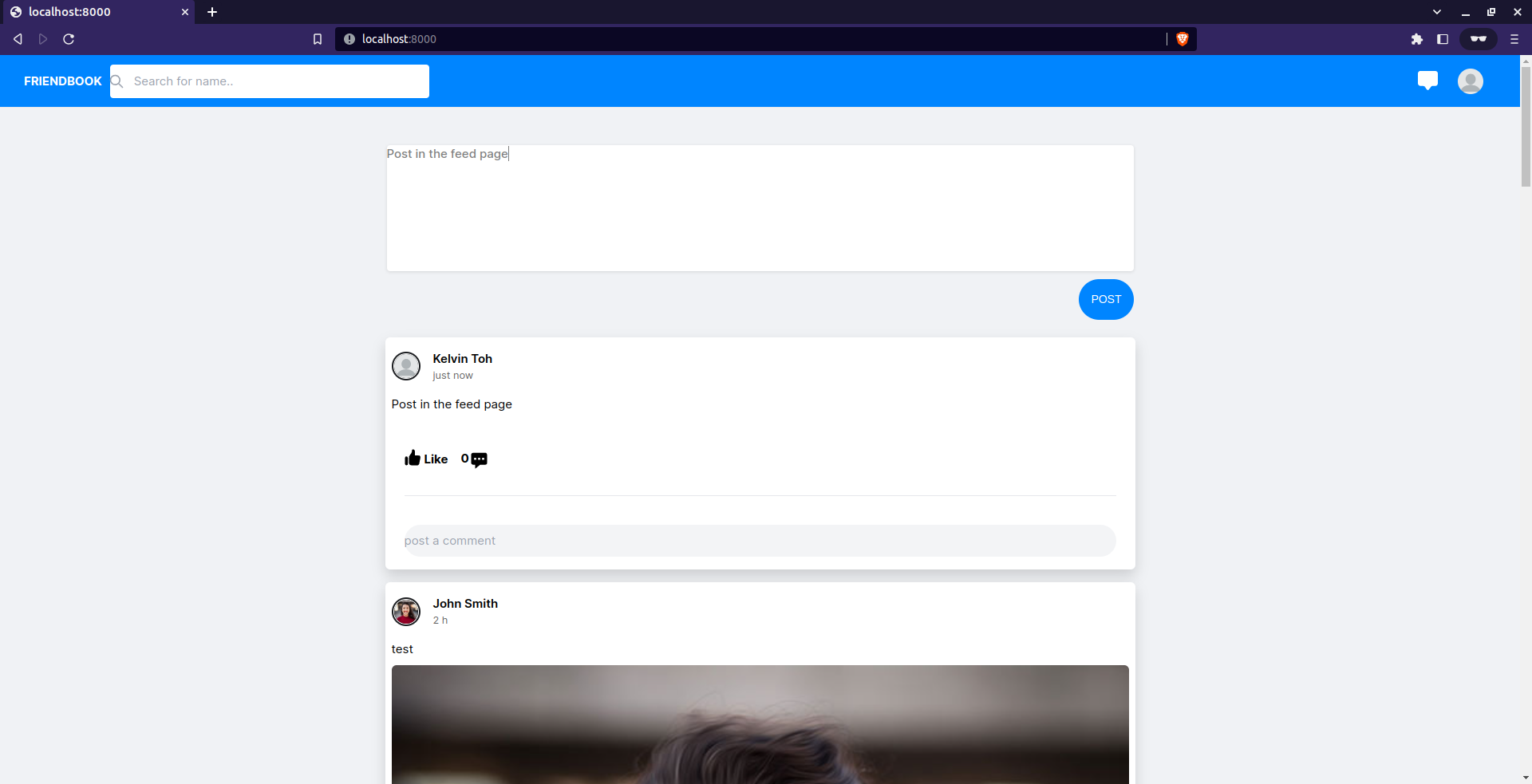
This is the chatbox html, it consists of an input and button, the input has the onKeyDown attribute to check if the user has pressed the “enter” key. The button have the onclick function that trigger the sendMessage function

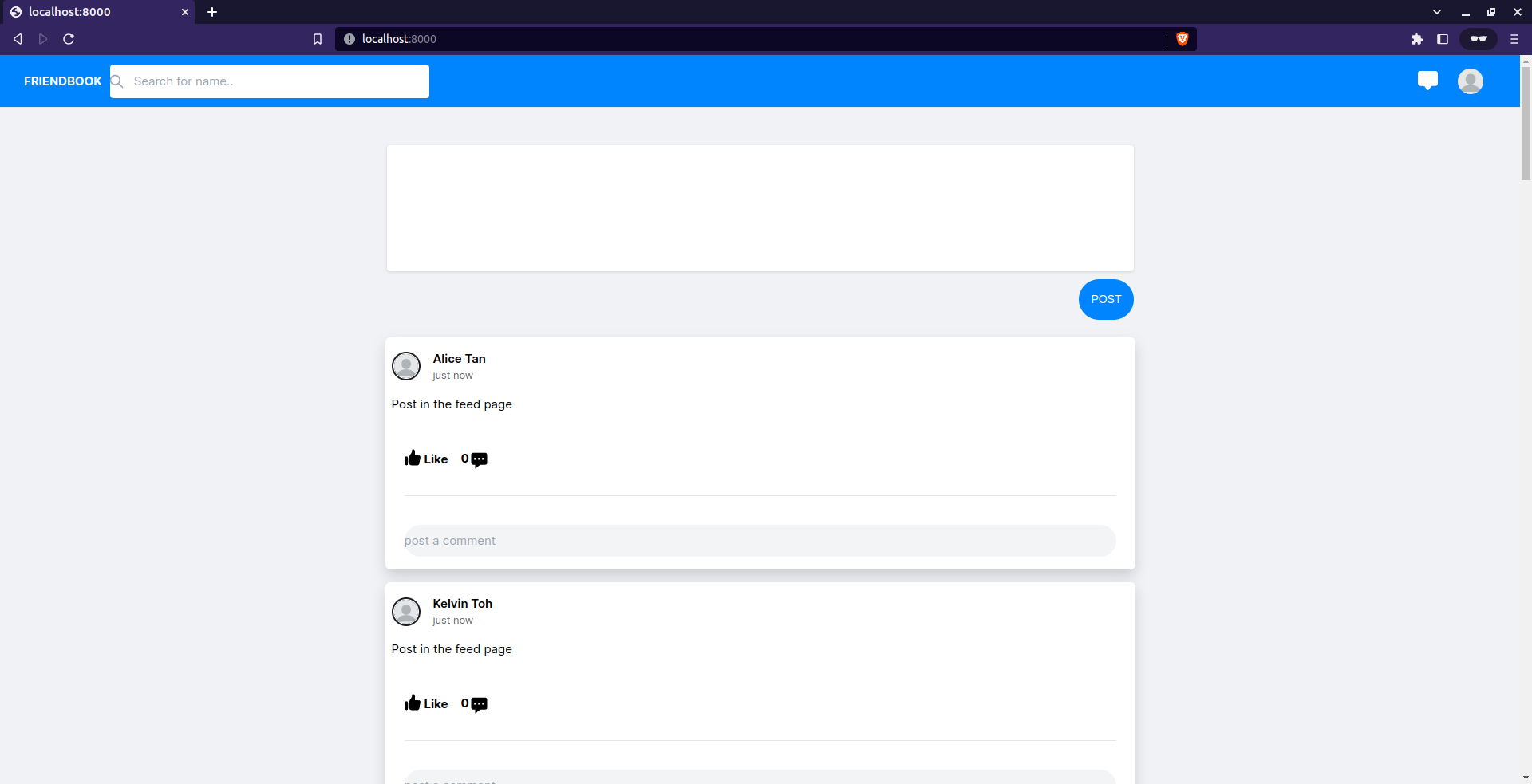


The handleInputKeyDown is to check if the user has pressed the “enter” key, if “enter” triggers the click function of the button. The sendMessage function will send the message to the websocket and set the input value to an empty string.

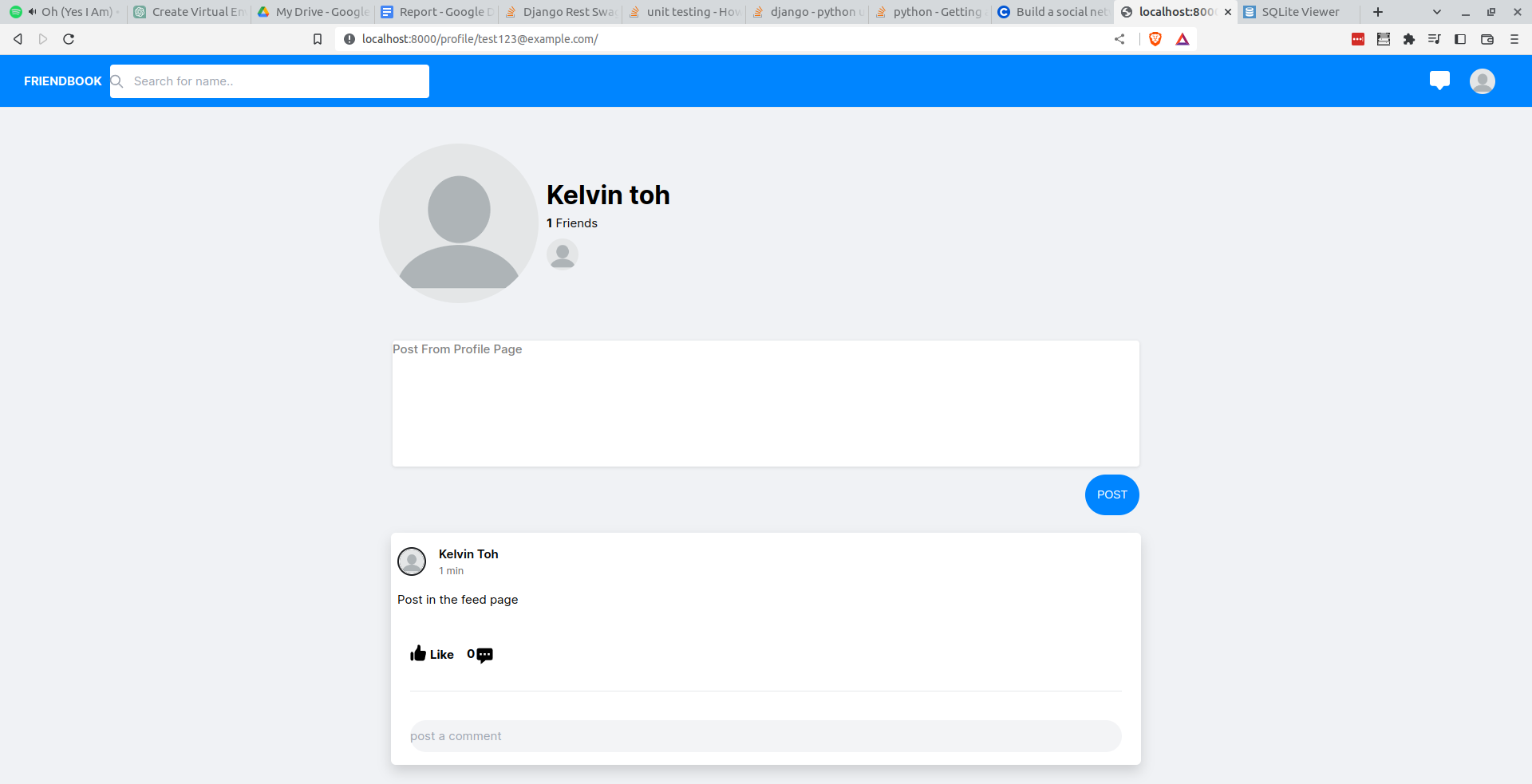
### R1f & R1g.Users can add status updates to their home page and Users can add media (such as images to their account and these are accessible via their home page

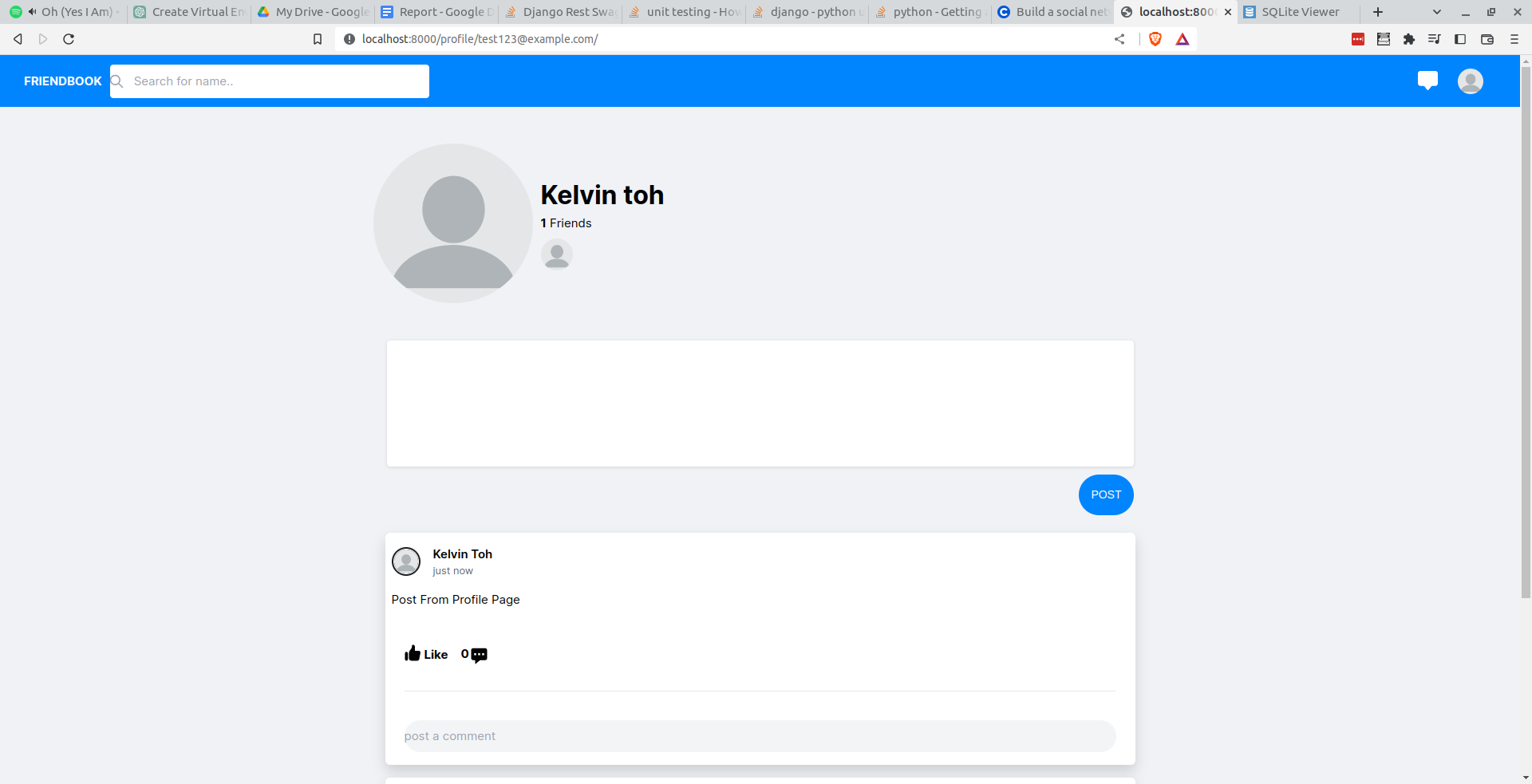
1. User can post status from the feed page



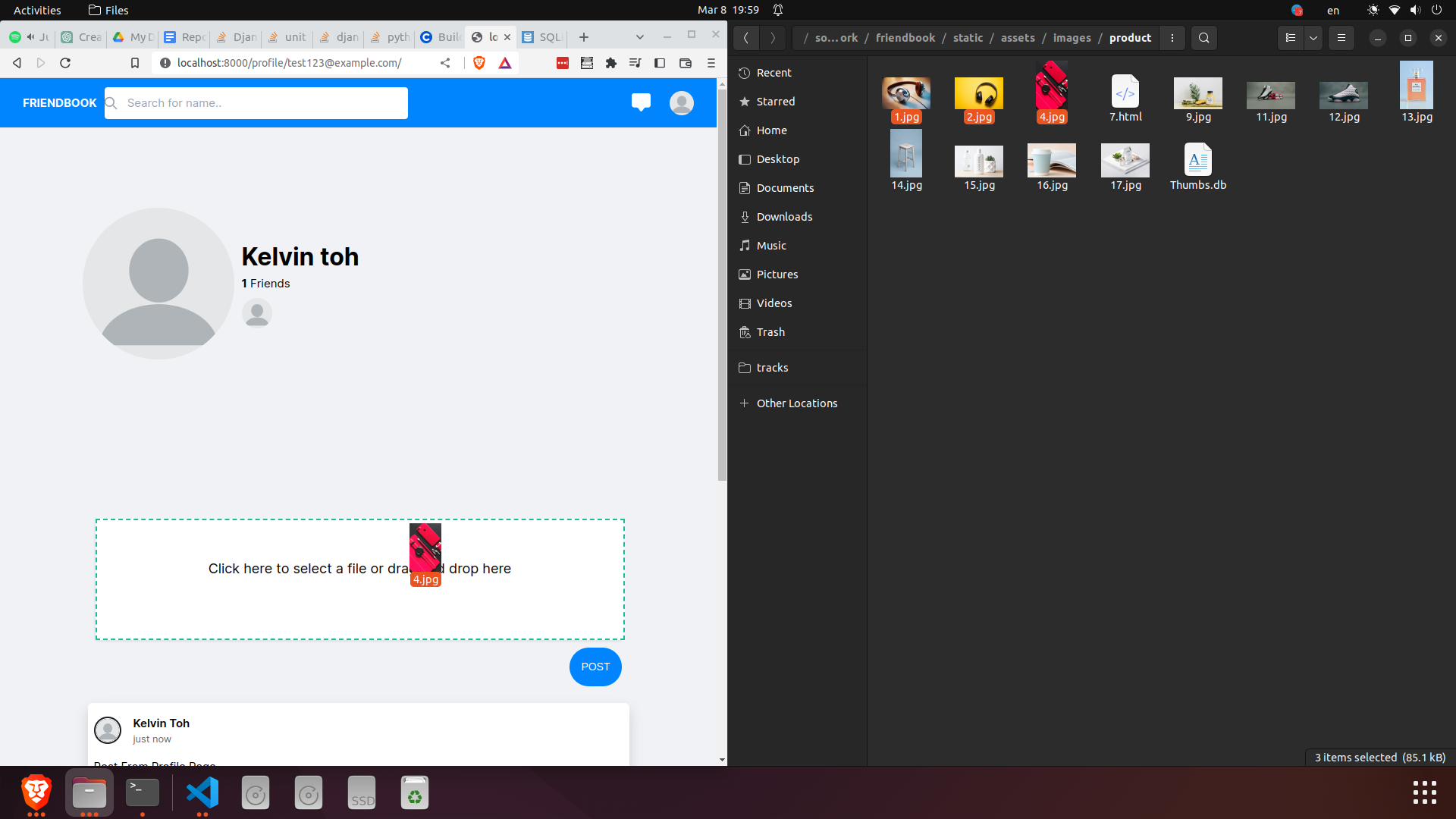


1. User can post status on their profile page





1. User can upload image as status(post)

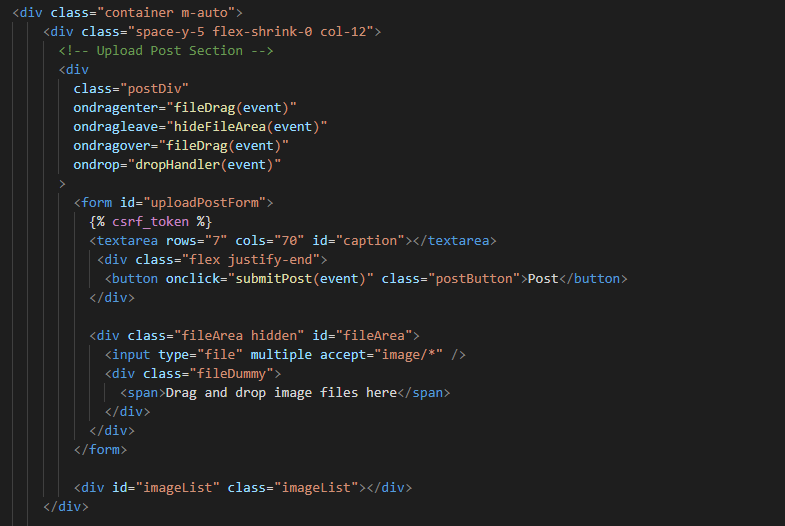


#### 

#### 

Users can post status(post) in the main page or in their profile page with or without images.

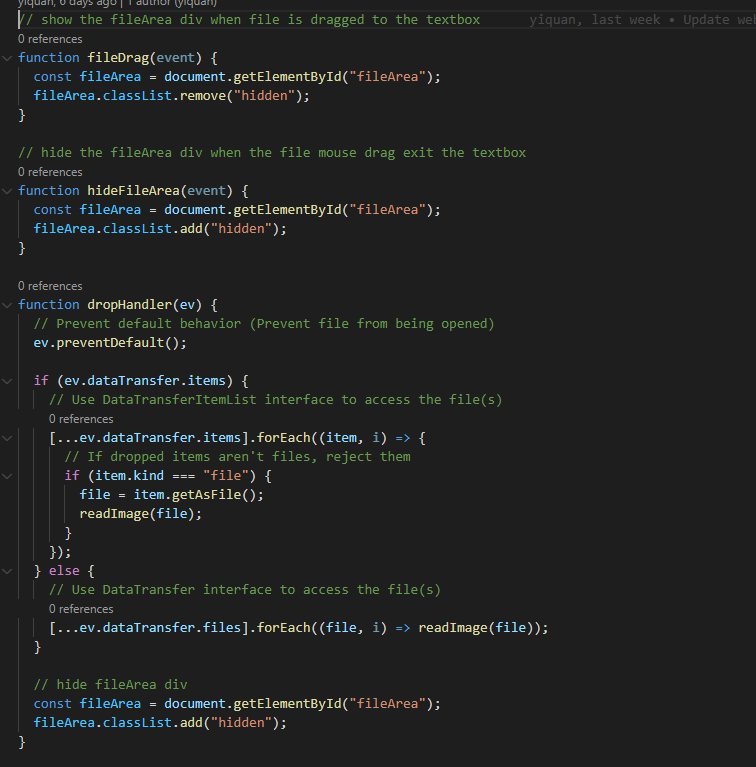
**index.html, profile.html**



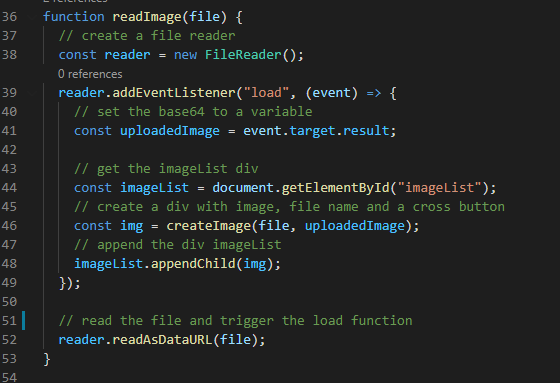
This is the drag and drop image text box which can be found in index.html and profile.html. The entire div listens to ondragenter, ondragleave, ondragover and ondrop. These are listeners for the file drag and drop. Inside the form, there are 2 divs which is 1 textarea with a post button and another div is for the overlaying the message above the textarea. The last div is the imageList which shows all the images that are dropped into the textarea.

**post.js**

All the following code can be found in the post.js which is located in friendbook/static/js/post.js.



The fileDrag function will display the drag and drop text div and the hideFileArea will hide the drag and drop text div. The dropHandler function will handle all the images dropped into the textarea and get the file and pass it to the readImage function. After reading all the image and displaying it, hide the drag and drop text div.



The readImage function takes in a file and uses the FileReader to read the file as string and create it as an image object via the createImage function. When the image object is created, append the object to the imageList div.



The createImage function takes in the file object and the base64 image. An image div consists of the image, the file name and a cross button. The cross button(deleteSpan) has an onclick function which removes an element via id with the file name. The entire image div has an id of the file name which we can use to identify which image to remove from the list.

#### 

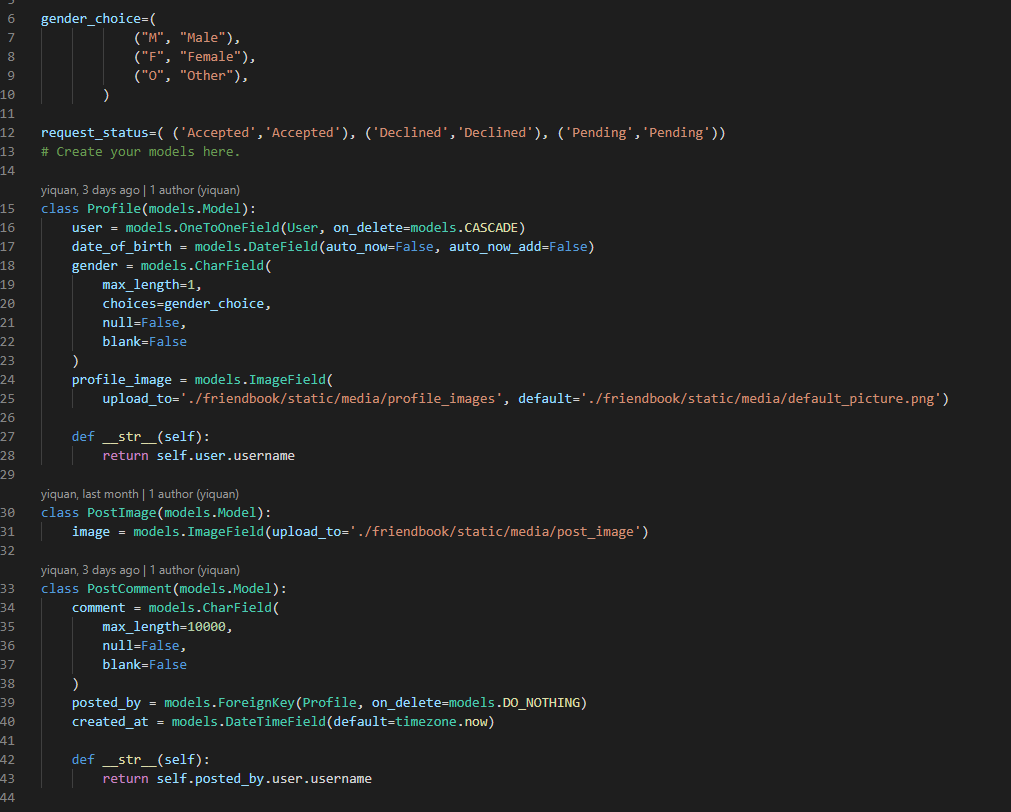
This is the submitPost function which will be triggered when the post button is clicked. The function will first get the caption values and image list. Django doesn’t support multi-part form out of the box(meaning if caption and images are part of the same form, value cannot be retrieved) and I am not able to find another work around to make it work so if you know how to make it work in django please put it in the feedback as I am quite keen to learn it. For nodejs, it works out of the box. So in my case, I get the base64 and push it to an array and post it together in the body with the caption. The reason why base64 should not be sent in a post body is because they are large sizes which can affect performances and if the image size is too big it will be an error.

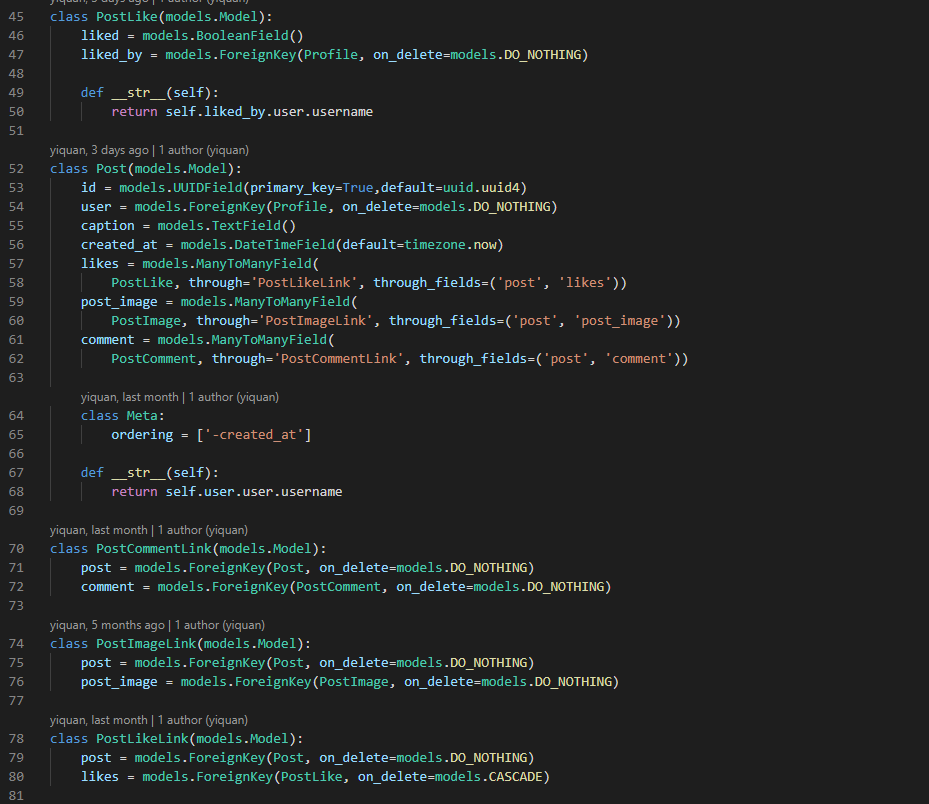
**api.py**

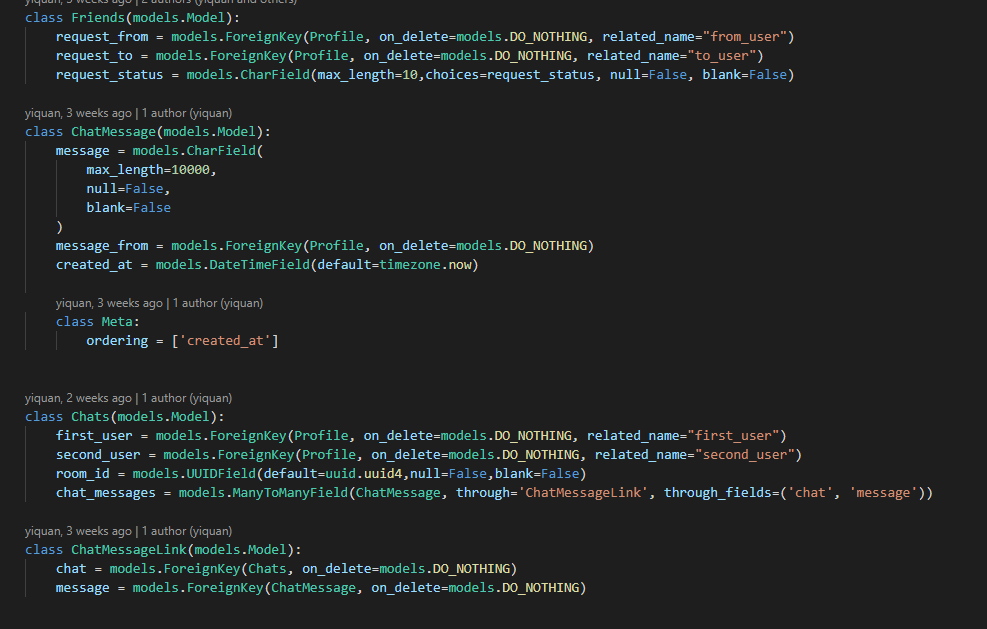
#### 

This is the upload\_post function which can be found in the api.py file. What this function does is get the user, if the user does not exist, return status 404 response else create a Post object with the user and caption. After that, loop through the base64 image, get the image data by doing a split and save the file to local disk using ContentFile(django.core.files.base) function and save the file with an uuid4 file name. If there is an error in saving the image, delete the new post and return status 400 response. Because I used Post.objects.create which inserted the data into the database, I must remove the post from the database if the image fails to save. I can use Post(user=user,caption=caption) as it doesn't save to the database but this will throw foreign key error as this post doesn’t exist in the Post table when inserting to the PostImageLink table. The best practice would be using a transaction which can do a rollback if anything fails. If the image is saved to the disk, create a PostImage object with the file and add it to the new post. After all the image has been saved and all the PostImage has been added to the new post, save the new post and return status 200 response.

### R1h. correct use of models and migrations







These are the models I used for this project and can be found in the models.py. So this is how I designed my models, starting from Profile, a user can have username, first and last name, password, date of birth, gender and a profile image. Username, password, first and last name already existed in the User model(django auth model) so I created a field user in Profile and set the User model as foreign key. The date of birth, gender and profile image will be the field in Profile. In my case, the profile image is being uploaded to friendbook/static/media/profile\_images and the default profile image for all users is a person silhouette image which is located at friendbook/static/media. For status(post), there is PostImage, PostComment, PostLike and Post. A post has an id, user(user who created the post), caption, created\_at, likes, post\_images and comment. The id is a uuid4, the user is a foreign key of Profile, caption is a string, created\_at is a utc datetime, likes, post\_images and comment is a ManyToMany field. A post is created by 1 user but a post can have more than 1 like, more than 1 image and more than 1 comment. So in order to fulfill the criteria, a link table must be created which are PostCommentLink, PostImageLink, PostLikeLink. This will ensure that a post can have more than 1 person who likes it, have more than 1 comment and have more than 1 image. The Friends model has the field of request\_from, request\_to and created\_at. The request\_from and request\_to is foreign key of Profile and created\_at is a datetime field which is set to utc. A chat will have 2 users, room id and multiple chat messages. So in the Chats model, there are first\_user, second\_user, room\_id and chat\_messages. The first\_user and second\_user both are foreign key of Profile, related\_name is added because it will raise an error because the same model was used as foreign key multiple times. The room\_id is a uuid4 and chat\_messages is a ManyToMany field. Same as the post, a chat can have multiple messages so a link table is needed which is ChatMessageLink model. The rest of the field for ChatMessages, PostImage, PostComment, PostLike I would not explain in this report. I used uuid for some of the id because using the default incremental id which is not secure whereas using a uuid are safer from people retrieving the data from api. I have used Profile as foreign key for those user field rather than User model because I want to retrieve the profile\_image of the user so using Profile is a much better option.

# Conclusion

Although I did not explain very in-depth about a lot of functions, models, api, serializers and test cases due to the word limit, I have written comments in the code. Overall my implementation works well for this project as it meets the requirements but it’s only sufficient for individual use/prototyping. If it is open for real world users to use, this implementation won’t be able to scale and there are pages that lack features. Why this implementation don’t scale well because if there are lots of users there will be a lot of edge cases like concurrency issues(like chat object being created twice, friend request being created twice), performance will slow down because retrieval of data retrieve everything(no pagination) like posts, friends, chat messages. So this led to what could be done better, using transaction would solve the concurrency issue and meet the ACID(atomicity, consistency, isolation and durability) properties. However, using transaction for database might affect performance but at least it fixed the concurrency issue. For retrieving of data, the api endpoint can take in a limit and offset parameter which can be used when retrieving the data(add [offset:offset+limit] for the queryset) however, this would require all the .get function to use .filter or .all function because .get function doesn’t support this.

The changes I would made if I attempt this project again is to refactor the code to follow DRY(don’t repeat yourself), example like the try except for retrieving user and make it to a function as most function would retrieve the user with try except so a helper function can make the code DRY. I would also like to add pagination to my api. Feature wise I would like to implement a websocket to send notifications(for chat, status, comment, likes) to the user and the ability to send images into the chat message. There are features that I did not cover in the report but there are features that I have written here is a list of it. Setting page able to upload profile picture and update information. Homepage able to retrieve status(posted by friends and ownself). Status(post) can be liked and commented.

To wrap up the report, I would like to share my thoughts based on my experience and this project. As a SPA background developer, I find that django templating is not as flexible compared to framework like reactjs or elm, variables doesn’t have a global state to update the html value(maybe there is but I have yet to find it) and having the need to code function in 2 language(I have to code the format\_date function in custom\_tags which is consumed by the django template but in my javascript I would need to write the same function in javascript because javascript code cannot use python function). For the backend if compares to nodejs, I would say Django might have some benefits over nodejs but those benefits nodejs can overcome easily. Example are Django can do database migration with models which is really convenient, the setup to get the server running is very minimum, no sql like and test case are built in so can write and test without the need to have a lot of setup which make Django very development friendly. But nodejs can do the same but needs to install packages like knexjs(no sql database query and database migration), jest(unit testing) and the need to write the code for index.js(all the ports, bodyparser etc…). The downside of Django is that I find it confusing and very easy to make mistake for example request.user this return a user model object but if you do a print, it returns the username because during development when I checked the username given with request.user which give me false which I went around google searching for the answer. However, this is a good experience for me because I don’t believe there is a framework that “dominate” the coding world but rather different framework will provide a learning opportunity. I have learnt how to use tailwind css(is much faster then declaring styling in css file), learnt how to build a drag and drop file area(in reactjs I always use package) by learning how to use ondragenter, ondragleave, ondragover, ondrag listener brushed up my python and javascript skill and finally websocket which I have never touch during the time I work.

**Word count: 5810**

# References

Html template: <https://github.com/tomitokko/django-social-media-template>

File drag and drop: <https://developer.mozilla.org/en-US/docs/Web/API/HTML_Drag_and_Drop_API/File_drag_and_drop>

Html code taken from repository:

<https://github.com/tomitokko/django-social-media-template>