



# ADVANCED WEB DEVELOPMENT COURSEWORK 2

SOCIAL NETWORK

CHIA YI QUAN  
Student Number: 210220223

## Table of Contents

Running Of The Project	2
Installation And Setup	2
Creating A New Environment	3
Setting Up Redis(skip this if it is installed and running)	5
Running The Test Case	6
Running The Project	6
Project Functionality Requirements	7
R1a. Users can create accounts	7
How this was achieved	7
R1b. Users can log in and log out	11
How this was achieved	11
How this was achieved	13
R1c. Users can search for other users	14
How this was achieved	15
R1d. Users can add other users as friends	18
How this was achieved	19
R1e. Users can chat in realtime with friends	25
How this was achieved	26
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	32
R1h. correct use of models and migrations	41
Conclusion	44
References	45

# 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. This are the list of account seeded in the sqlite database.

## **Username**

alice@example.com  
ben@example.com  
derrick@example.com  
casey@example.com  
darren@example.com  
zen@example.com  
collin@example.com

## **Password**

QWEasd123

## **Admin Panel**

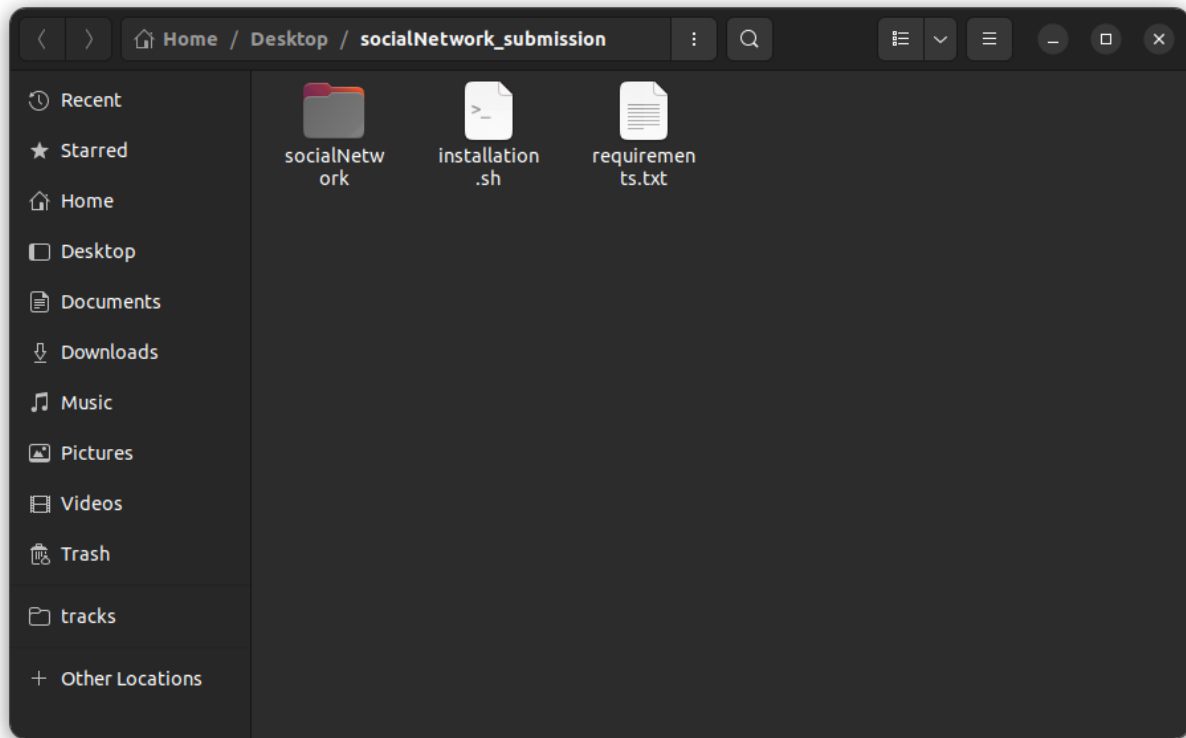
Username: admin  
Password: 123456

**Demo url:** [https://youtu.be/bvzX\\_TUilKQ](https://youtu.be/bvzX_TUilKQ)

# Running Of The Project

## Installation And Setup

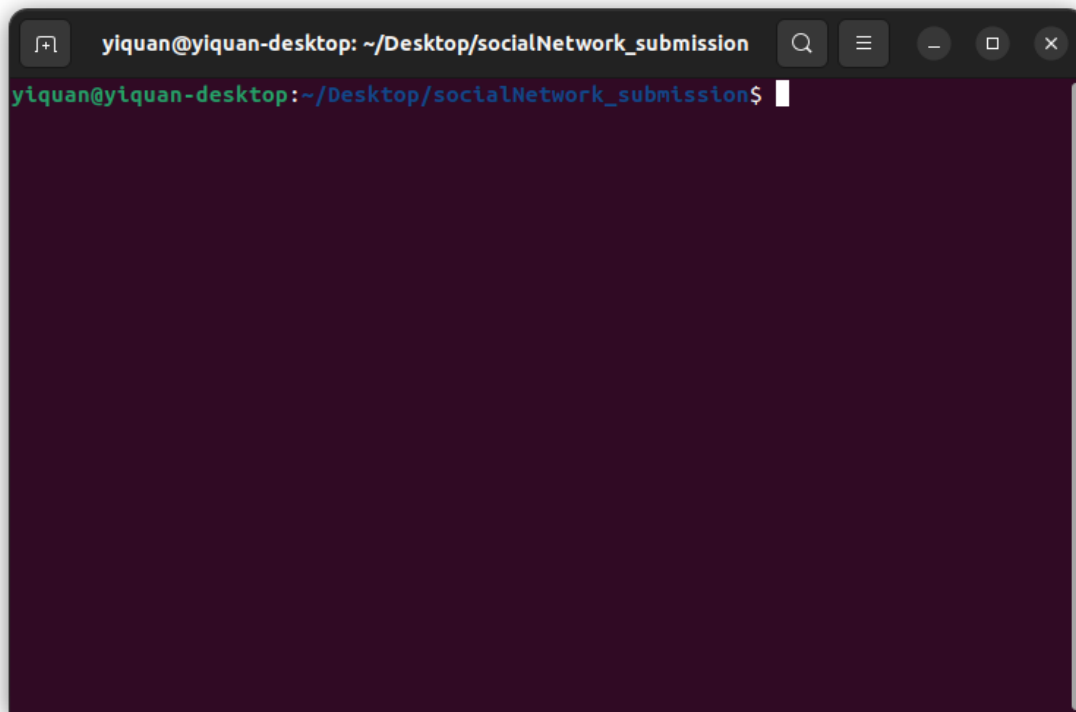
From the submitted project folder, you will find a requirements.txt file and the socialNetwork folder. Before we start, this code was developed and tested on ubuntu system, due to redis doesn't have direct support on windows I would recommend you running this code on ubuntu but if you use windows operating 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.

```
yiquan@yiquan-desktop: ~/Desktop/socialNetwork_submissi...
eschema, service-identity, pyOpenSSL, paramiko, factory-boy, coreapi, celery, au
tobahn, fabric, drf-yasg, fake, daphne, channels, channels-redis
Running setup.py install for coreschema ... done
Running setup.py install for fake ... done
Successfully installed Automat-20.2.0 Deprecated-1.2.13 Django-4.0.4 Faker-17.6.
0 Jinja2-3.1.2 MarkupSafe-2.1.2 Pillow-9.2.0 PyNaCl-1.5.0 Twisted-22.10.0 aiored
is-1.3.1 amqp-5.1.1 asgiref-3.5.2 async-timeout-4.0.2 attrs-21.4.0 autobahn-22.5
.1 bcrypt-4.0.1 beautifulsoup4-4.11.1 billiard-3.6.4.0 celery-5.2.7 certifi-2022
.6.15 cffi-1.15.1 channels-3.0.5 channels-redis-3.4.0 charset-normalizer-2.1.0 c
lick-8.1.3 click-didyoumean-0.3.0 click-plugins-1.1.1 click-repl-0.2.0 constantl
y-15.1.0 coreapi-2.3.3 coreschema-0.0.4 cryptography-37.0.2 daphne-3.0.2 django
bootstrap4-22.1.djangorestframework-3.13.1 drf-yasg-1.21.5 fabric-3.0.0 factory
boy-3.2.1 fake-0.8 hiredis-2.0.0 hyperlink-21.0.0 idna-3.3 incremental-21.3.0 in
flection-0.5.1 invoke-2.0.0 itypes-1.2.0 kombu-5.2.4 mod-wsgi-4.9.3 mod-wsgi-htt
pd-2.4.54.1 msgpack-1.0.4 oauthlib-3.2.0 packaging-21.3 paramiko-3.0.0 prompt-to
olkkit-3.0.30 pycpg2-binary-2.9.3 pyOpenSSL-22.0.0 pyasn1-0.4.8 pyasn1-modules-
0.2.8 pycparser-2.21 pyparsing-3.0.9 python-dateutil-2.8.2 pytz-2022.1 pytz-depr
ecation-shim-0.1.0.post0 redis-4.3.4 requests-2.28.1 requests-oauthlib-1.3.1 rua
mel.yaml-0.17.21 ruamel.yaml.clib-0.2.7 service-identity-21.1.0 six-1.16.0 soups
ieve-2.3.2.post1 sqlparse-0.4.2 txaio-22.2.1 typing_extensions-4.3.0 tzdata-2022
.1 tzlocal-4.2 uritemplate-4.1.1 urllib3-1.26.10 vine-5.0.0 wcwidth-0.2.5 wrapt-
1.14.1 zope.interface-5.4.0
(social_network_env) yiquan@yiquan-desktop:~/Desktop/socialNetwork_submission/so
cialNetwork$
```

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](#)):

```
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:

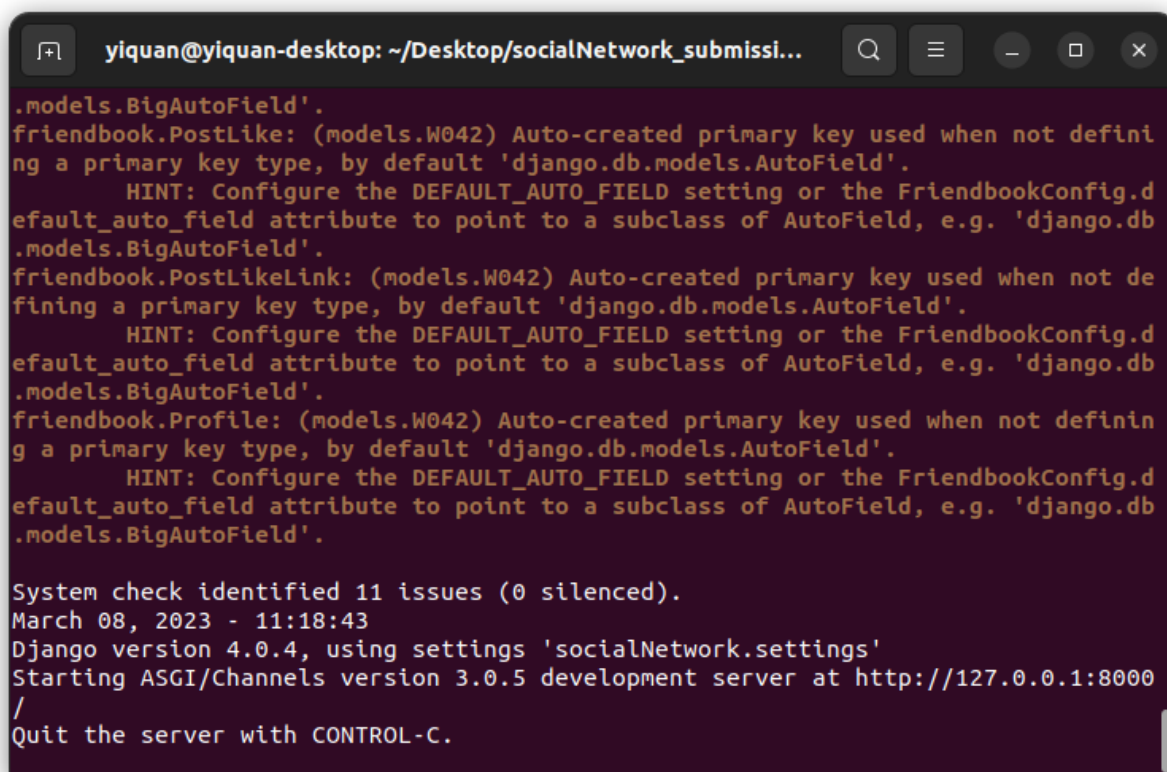
```
python manage.py test
```

## 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.



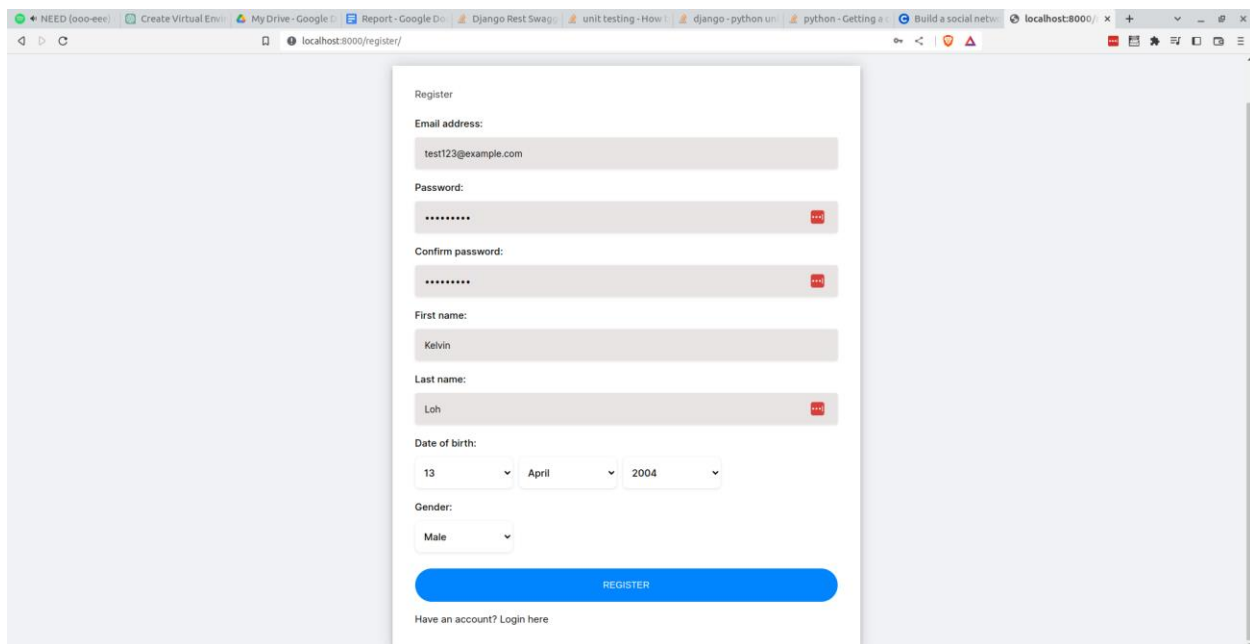
```
yiquan@yiquan-desktop: ~/Desktop/socialNetwork_submissi...  
models.BigAutoField'.  
friendbook.PostLike: (models.W042) Auto-created primary key used when not defini  
ng a primary key type, by default 'django.db.models.AutoField'.  
  HINT: Configure the DEFAULT_AUTO_FIELD setting or the FriendbookConfig.d  
efault_auto_field attribute to point to a subclass of AutoField, e.g. 'django.db  
.models.BigAutoField'.  
friendbook.PostLikeLink: (models.W042) Auto-created primary key used when not de  
fining a primary key type, by default 'django.db.models.AutoField'.  
  HINT: Configure the DEFAULT_AUTO_FIELD setting or the FriendbookConfig.d  
efault_auto_field attribute to point to a subclass of AutoField, e.g. 'django.db  
.models.BigAutoField'.  
friendbook.Profile: (models.W042) Auto-created primary key used when not definin  
g a primary key type, by default 'django.db.models.AutoField'.  
  HINT: Configure the DEFAULT_AUTO_FIELD setting or the FriendbookConfig.d  
efault_auto_field attribute to point to a subclass of AutoField, e.g. 'django.db  
.models.BigAutoField'.  
  
System check identified 11 issues (0 silenced).  
March 08, 2023 - 11:18:43  
Django version 4.0.4, using settings 'socialNetwork.settings'  
Starting ASGI/Channels version 3.0.5 development server at http://127.0.0.1:8000/  
Quit the server with CONTROL-C.
```

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 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](#) an empty template which includes the css and styling of a page. 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

```
10
11 # check if password have at least 8 char, 1 uppercase, lowercase and number
12 # if any of the condition not satisfied, raise error
13 def validate_password(password):
14     if len(password) < 8 or not re.search(r'[A-Z]', password) or not re.search(r'[a-z]', password) or not re.search(r'\d', password):
15         raise ValidationError('Password must be at least 8 characters long, have 1 uppercase, lowercase and number.', code='weak_password')
16
```

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.



```

yiquan, 7 days ago | 1 author (yiquan)
17 class UserForm(forms.ModelForm):
18     password = forms.CharField(widget=forms.PasswordInput())
19     # add a confirm_password field
20     confirm_password = forms.CharField(widget=forms.PasswordInput())
21
yiquan, last week | 1 author (yiquan)
22 class Meta:
23     model = User
24     fields = ['email', 'password', 'confirm_password', 'first_name',
25             'last_name', ]
26
27 # override the built in function to validate the password criteria
28 def clean_password(self):
29     password = self.cleaned_data.get('password')
30     validate_password(password)
31     return password
32
33 # add custom function as confirm_password field are not in the User model
34 # check password and confirm password are the same else raise error
35 def clean_confirm_password(self):
36     password = self.cleaned_data.get('password')
37     confirm_password = self.cleaned_data.get('confirm_password')
38     if password and password != confirm_password:
39         raise forms.ValidationError("Password and Confirm password don't match", code='password_not_match')
40     return confirm_password
41
42

```

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.

```

yiquan, 7 days ago | 1 author (yiquan)
✓ class UserProfileForm(forms.ModelForm):
    # reverse the year to show the latest year first
    years = range(1930, datetime.now().year)[::-1]
    ✓ date_of_birth = forms.DateField(
        widget=forms.SelectDateWidget(years=years))

yiquan, 6 months ago | 1 author (yiquan)
✓ class Meta:
    ✓ model = Profile
    fields = [
        'date_of_birth',
        'gender'
    ]

```

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

```
47 def user_register(request):
48     registered = False
49
50     # if request is post
51     if request.method == 'POST':
52         # get the user form and user profile form
53         user_form = UserForm(data=request.POST)
54         profile_form = UserProfileForm(data=request.POST)
55
56         # if both form is valid
57         if user_form.is_valid() and profile_form.is_valid():
58             try:
59                 # create a user instance but don't save it first
60                 user = user_form.save(commit=False)
61                 # set the user instance username as email
62                 user.username = user.email
63                 # set the password
64                 user.set_password(user.password)
65                 # save the user instance
66                 user.save()
67
68                 # create a profile instance but don't save it first
69                 profile = profile_form.save(commit=False)
70                 # set the profile user as user
71                 profile.user = user
72                 # save the profile
73                 profile.save()
74                 registered = True
75
76                 # will throw exception when username already exist
77                 # show error message
78             except:
79                 messages.info(request, 'User existed.')
80                 return redirect('/register/')
81
82             # if user form is not valid, loop through the error and display
83             elif not user_form.is_valid():
84                 password_errors = user_form.errors.get('password')
85                 for field, field_errors in user_form.errors.items():
86                     messages.info(request, field_errors)
87                 return redirect('/register/')
88
89             # catch unhandled error that are not caught in the form validation
90             else:
91                 messages.info(request, 'Form is invalid.')
92                 return redirect('/register/')
93
```

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.

```
94     else:
95         user_form = UserForm()
96         profile_form = UserProfileForm()
97
98     return render(request, 'friendbook/register.html', {'user_form': user_form,
99                                                         'profile_form': profile_form,
100                                                         'registered': registered,
101                                                         })
102
```

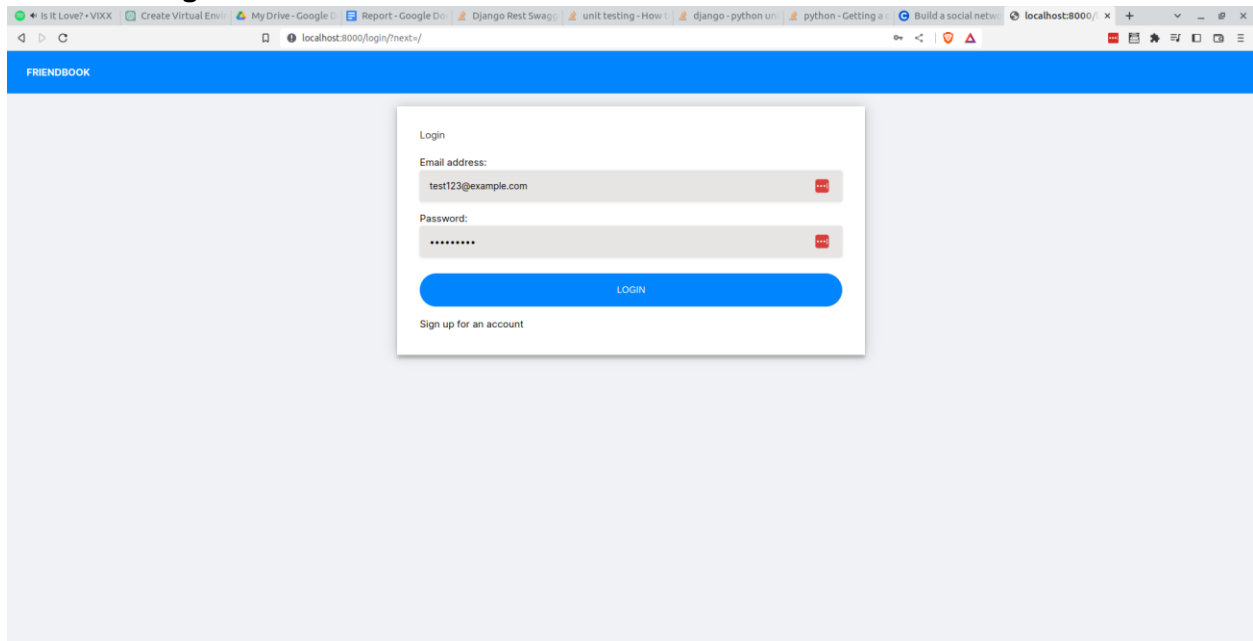
This code is a continuation of the user\_register function, if it is not a post function, pass in user\_form, profile\_form and registered into the register.html to render.

```
1  {% extends "../base.html" %}
2  {% block content %}
3  <div class="flexRowCenter">
4      <form
5          action="/register/"
6          id="user_form"
7          method="post"
8          enctype="multipart/form-data"
9          class="form"
10     >
11         <h1>Register</h1>
12         <br />
13         {% csrf_token %}
14         {{user_form.as_p}}
15         {{profile_form.as_p}}
16         {%for message in messages%}
17         <span class="error show">{{message}}</span>
18         {%endfor%}
19         {% if registered %}
20         <span class="success" class="mb-3"><strong>Thank you for registering</strong></span>
21         {% endif %}
22
23         <button type="submit" class="mb-3">Register</button>
24         <a href="../login">Have an account? Login here</a>
25
26     </form>
27
28 </div>
29 {% endblock %}
30
31
```

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

```
def user_login(request):
    # if request is post
    if request.method == 'POST':
        # get the email and password
        email = request.POST['email']
        password = request.POST['password']
        try:
            # get the user for that email
            user = User.objects.get(email=email)
            # check the password of the provided by user and the hashed password in database
            if check_password(password, user.password):
                # if user is active status log the user in
                if user.is_active:
                    login(request, user)
                    return HttpResponseRedirect('/')

                # if user is not active, show error message
                else:
                    messages.info(request, 'Your account is disabled.')
                    return redirect('/login/')

            # if password provided is different from the hashed password in database, show error message
            else:
                messages.info(request, 'Email or password is incorrect.')
                return redirect('/login/')

            # if exception is catch during check_password show error message
        except:
            messages.info(request, 'Email or password is incorrect.')
            return redirect('/login/')
    else:
        return render(request, 'friendbook/login.html')
```

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](#) 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.

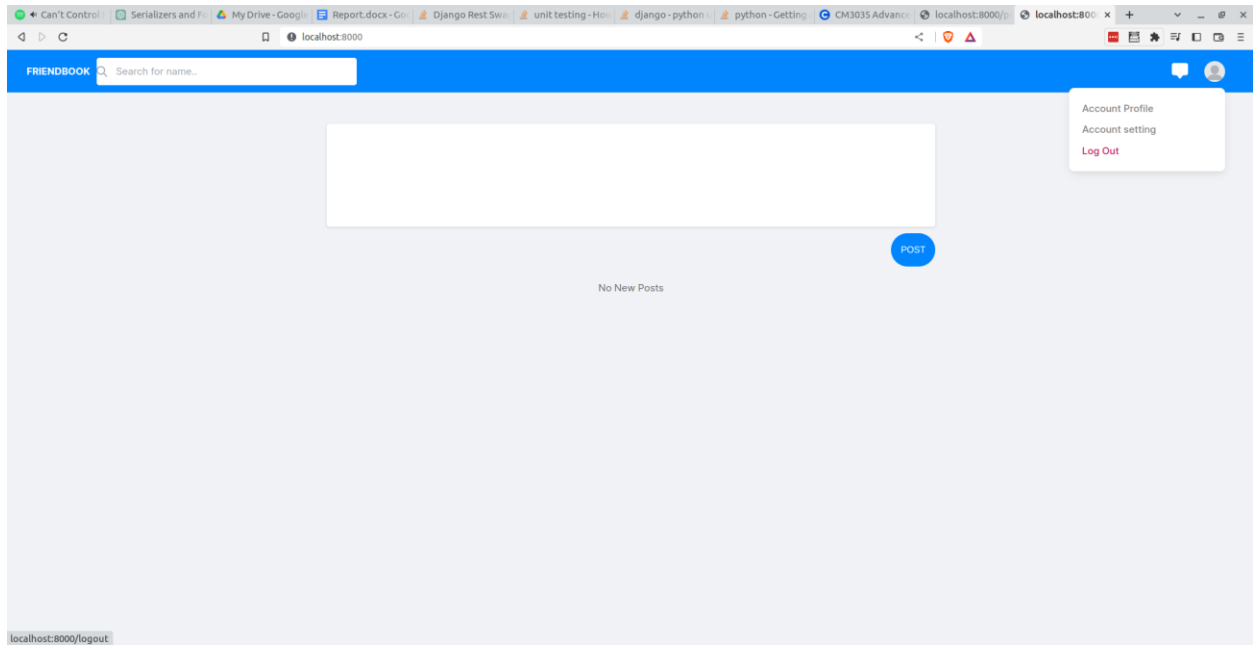
### login.html

```
{% extends "../base.html" %} {% block content %}
<div class="flexRowCenter">
  <form id="login_form" class="form" method="post" action="/login/">
    <h1>Login</h1>
    <br />
    {% csrf_token %}
    <span>Email address: </span>
    <input type="email" name="email" value="" size="20" id="login_email" />
    <span>Password:</span>
    <input
      type="password"
      name="password"
      value=""
      size="20"
      id="login_password"
    />
    {%for message in messages%}
    <span class="error show">{{message}}</span>
    {%endfor%}

    <button type="submit" class="mb-3">Login</button>
    <a href="/register/">Sign up for an account</a>
  </form>
</div>
{% endblock %}
```

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



How this was achieved

### views.py

```
@login_required(login_url='login/')
def user_logout(request):
    logout(request)
    return HttpResponseRedirect('/')
```

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

```

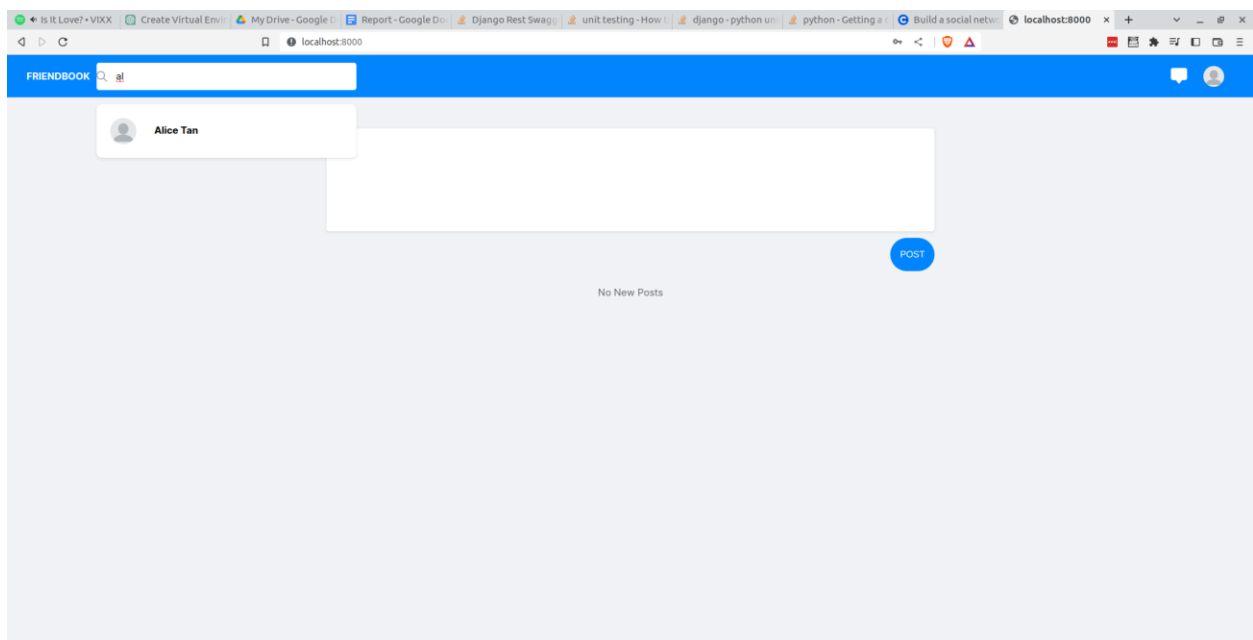
<!-- profile -->

<a href="#">
  
</a>
<div
  uk-drop="mode: click;offset:9"
  class="header_dropdown profile_dropdown border-t"
>
  <ul>
    <li><a href="/profile/{{user.username}}"> Account Profile </a></li>
    <li><a href="/setting"> Account setting </a></li>
    <li><a href="/logout"> Log Out</a></li>
  </ul>
</div>
</div>
{% endif %}
</div>

```

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

```
62 class SearchUser(generics.ListAPIView):
63     serializer_class = ProfileSerializer
64
65     def get_queryset(self):
66         search_value = self.kwargs['searchValue']
67         # search for user that contains the searchValue in their first_name and last_name
68         users = User.objects.filter(Q(first_name__icontains = search_value) | Q(last_name__icontains=search_value))
69         # return the profile of the list of user
70         return Profile.objects.filter(user__in=users)
71
```

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

```
<div class="header_search">
  <input type="text" placeholder="Search for name.." id="searchInput" oninput="searchUser()" />
  <div class="icon-search">
    <svg
      xmlns="http://www.w3.org/2000/svg"
      fill="none"
      viewBox="0 0 24 24"
      stroke="currentColor"
    >
      <path
        stroke-linecap="round"
        stroke-linejoin="round"
        stroke-width="2"
        d="M21 21l-6-6m2-5a7 7 0 11-14 0 7 7 0 0114 0z"
      />
    </svg>
  </div>
  <div class="searchResults"></div>
</div>
```

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



```

{% block javascript %}
<script>

function searchUser(){
  const searchInput = document.getElementById('searchInput');
  let searchResult = document.getElementsByClassName('searchResults')[0]
  if(searchInput.value.trim().length == 0) {
    searchResult.innerHTML = '';
    searchResult.style.display = 'none';
    return
  }
  fetch('/api/search-user/' + searchInput.value).then(response =>
    response.json()
  ).then((data)=>{
    if(data.length>0){
      let htmlResult=''
      data.forEach(({profile_image,user})=>{
        htmlResult += `
        <a href="/profile/${user.username}">
          <div class="flex items-center searchEntry" style="padding:5px;">
            <div class="w-10 h-10 rounded-full relative flex-shrink-0 mr-1">
              
            </div>
            <span class="text-m ml-4 font-bold">
              ${user.first_name} ${user.last_name}
            </span>
          </div>
        </a>
        `
      })
      searchResult.innerHTML = htmlResult
      searchResult.style.display = 'block';
    }
    else{
      searchResult.innerHTML = ''
      searchResult.style.display = 'none';
    }
  })
}

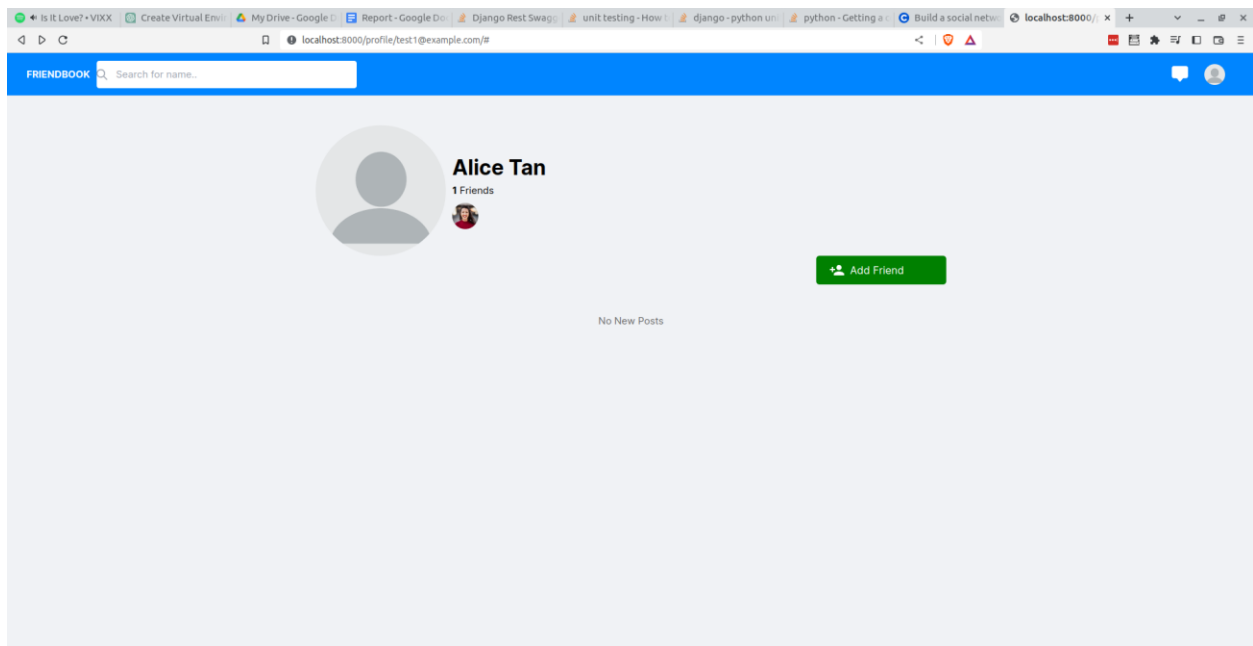
</script>
{% endblock%}

```

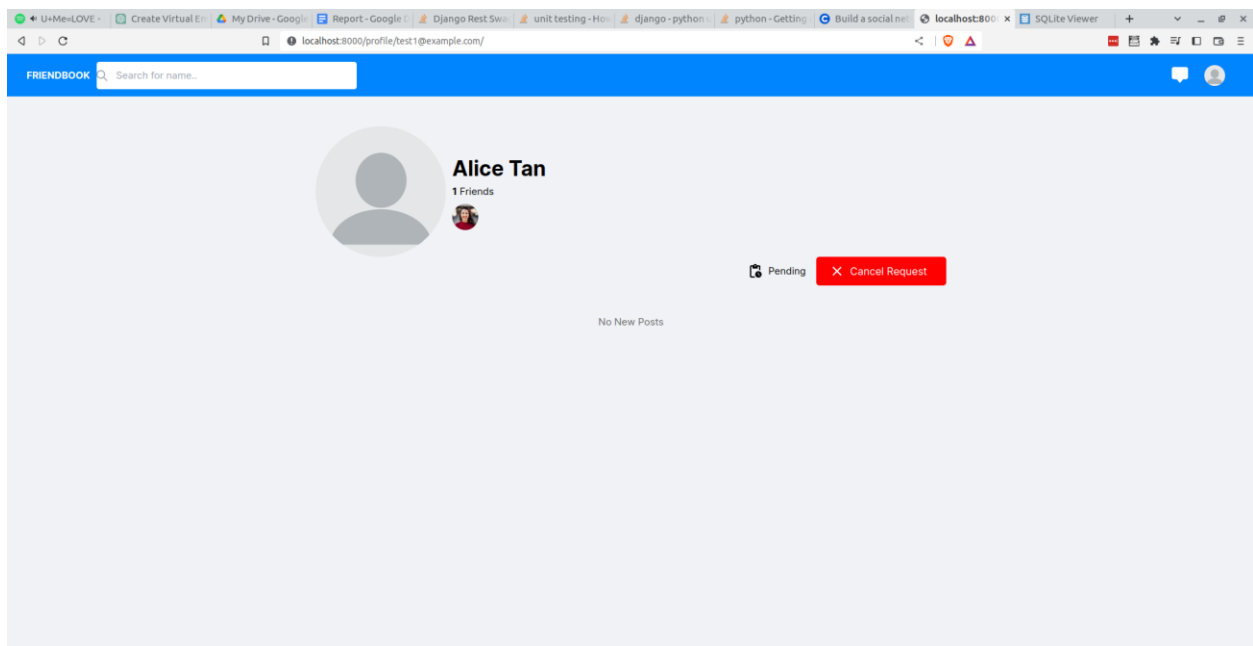
The following code can be found in nav.html, what this function does is that when the user types something in the input, we get the searchInput and searchResult element. Next, we check if the searchInput value is empty, I have used the trim function to remove the white space from the front and end of 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 must remove those previous results. 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. 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 the user can see the result.

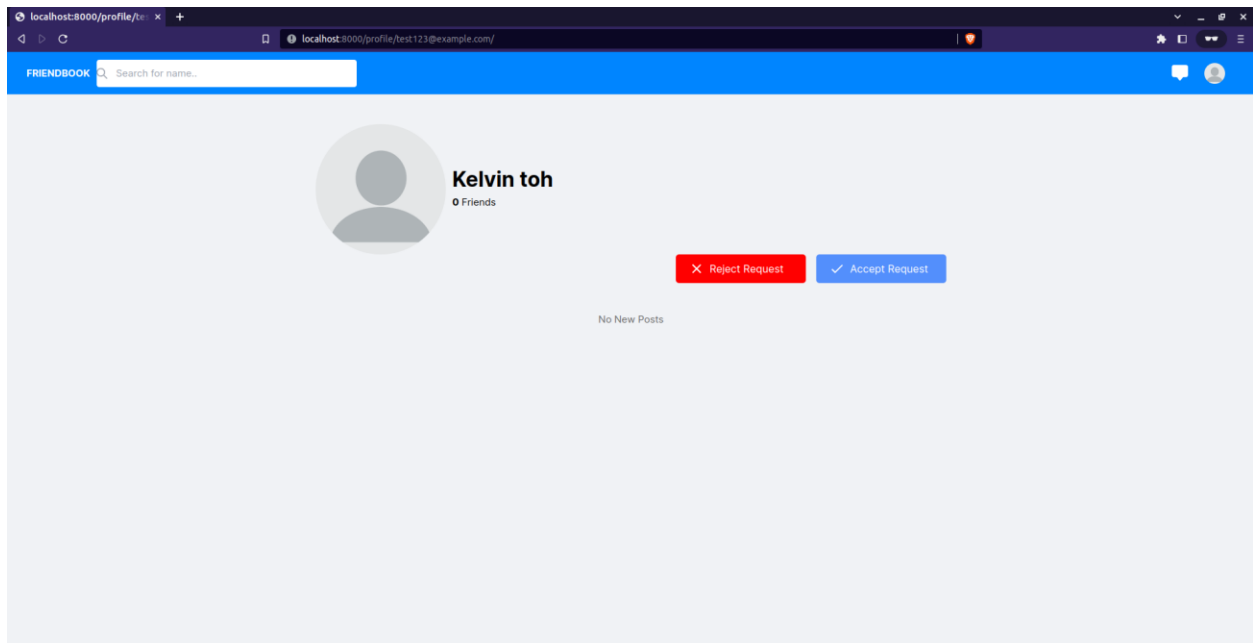
## R1d. Users can add other users as friends



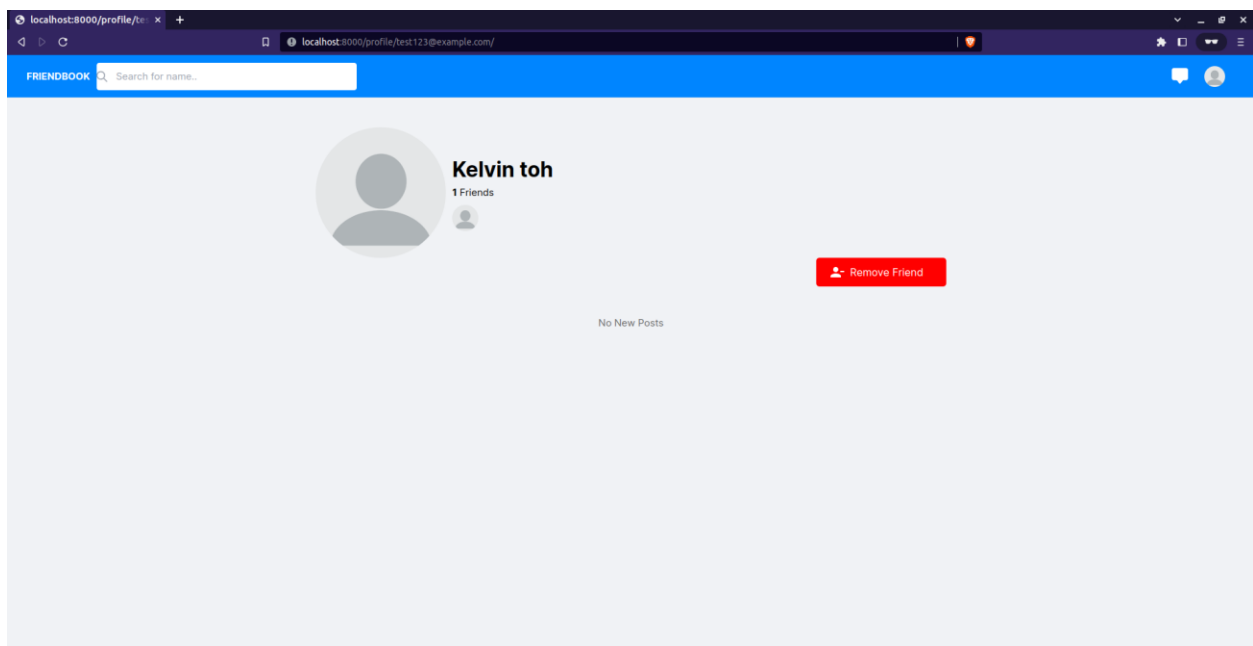
In the user profile page, the current user can send friend requests to a user. In this screenshot, I am Kelvin toh and I visited Alice Tan page.



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 sent 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**

```

yiquan, 4 hours ago | 2 authors (yiquan and others)
class SendFriendRequest(mixins.CreateModelMixin, generics.GenericAPIView):
    serializer_class = FriendSerializer

    def post(self, request, *args, **kwargs):
        request_from = None
        request_to = None

        # check if friendUsername is the same as the current user username,
        # to prevent them sending request to themselves
        if request.data.get('friendUsername') == request.user.username:
            return Response({'error': 'You cannot send friend request to yourself'}, status=status.HTTP_400_BAD_REQUEST)

        try:
            # get the user of current user and the targeted user aka friendUsername
            # request_from will be current user
            # request_to will be the targeted user
            request_from = Profile.objects.get(
                user=User.objects.get(username=request.user))
            request_to = Profile.objects.get(user=User.objects.get(
                username=request.data.get('friendUsername')))

            # if user does not exist return 404 error
            except (Profile.DoesNotExist, User.DoesNotExist) as err:
                return Response({'error': "Invalid user"}, status=status.HTTP_404_NOT_FOUND)

            # query where request_from or request_to is current user
            # and request_from or request_to is the recipient user
            query_filter = Q(request_from=request_from, request_to=request_to) | Q(
                request_from=request_to, request_to=request_from)

            # get the request or create a new request object
            friend_request, created = Friends.objects.filter(query_filter).get_or_create(
                defaults={'request_from': request_from, 'request_to': request_to, 'request_status': 'Pending'})

            # if is not created, meaning friend request already existed, return status 200
            if not created:
                return Response({'error': "Friend request sent"}, status=status.HTTP_200_OK)

            # if created serialize the request and send the data as a response
            serializer = self.get_serializer(friend_request)

            return Response(serializer.data, status=status.HTTP_201_CREATED)

```

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 do a filter 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` then use the `get_or_create` function here, if the filter returns

nothing 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 else the record will be returned. 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.

```
@api_view(['POST'])
def reject_friend_request(request):
    request_from = None
    request_to = None

    # get the user profile
    try:
        request_from = Profile.objects.get(user=User.objects.get(username=request.user))
        request_to = Profile.objects.get(user=User.objects.get(username=request.data.get('friendUsername')))

    # return 404 if user not found
    except (Profile.DoesNotExist, User.DoesNotExist) as err:
        return Response({'error': "Invalid user"}, status=status.HTTP_404_NOT_FOUND)

    try:
        # get the entry
        # if request_from is the current user and request_to is the targeted user
        # or request_from is the targeted user and request_to is the current user
        # delete the record and return 200
        request_list = Friends.objects.filter(Q(request_from=request_from, request_to=request_to) | Q(request_from=request_to, request_to=request_from))
        request_list.delete()

        return Response({'data': 'Friend has been removed'}, status=status.HTTP_200_OK)
    # filter most probably won't throw exception error when it return a empty list and did a delete()
    # but put this just in case
    except Friends.DoesNotExist:
        return Response({'error': "Invalid request"}, status=status.HTTP_404_NOT_FOUND)
```

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.

```

@api_view(['POST'])
def accept_friend_request(request):
    # get the user profile
    try:
        first_user = Profile.objects.get(
            user=User.objects.get(username=request.user))
        second_user = Profile.objects.get(user=User.objects.get(
            username=request.data.get('friendUsername')))

        # return 404 if user not found
    except (Profile.DoesNotExist, User.DoesNotExist) as err:
        return Response({'error': "Invalid user"}, status=status.HTTP_404_NOT_FOUND)

    # get the entry
    # if request_from is the current user and request_to is the targeted user
    # or request_from is the targeted user and request_to is the current user
    friend_request = Friends.objects.filter(Q(request_from=first_user, request_to=second_user) | Q(
        request_from=second_user, request_to=first_user))

    # if there is no such request, return 404
    if len(friend_request) == 0:
        return Response({'error': 'Invalid request'}, status=status.HTTP_404_NOT_FOUND)

    # if the entry request_from value is current user return 400
    # to prevent the person who send the request accepting the request on the other party behalf
    if friend_request.first().request_from.user == request.user:
        return Response({'error': 'You cannot accept request that you sent'}, status=status.HTTP_400_BAD_REQUEST)

    # set the request status to Accepted and return 200
    friend_request.update(request_status='Accepted')
    return Response({'data': 'Friend request has been accepted'}, status=status.HTTP_200_OK)

```

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 users 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. Assuming the list only returns 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 sends 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. 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**

```
def user_profile(request, email):
    # get the user post from getPost api
    post_url = reverse('getPost', args=[email])
    post_result = requests.get(
        request.build_absolute_uri(post_url), params=request.GET)

    # get the user data with email
    user_url = reverse('getUser', args=[email])
    user_result = requests.get(
        request.build_absolute_uri(user_url), params=request.GET)

    # get the friend list of the user with email
    get_friends_url = reverse('getFriendList', args=[email])
    friend_list = requests.get(request.build_absolute_uri(
        get_friends_url), params=request.GET)

    # display the information on the profile page
    return render(request, 'friendbook/profile.html', {'posts': post_result.json(), 'user_profile': user_result.json(), 'friend_list': friend_list.json()})
```

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

```
@register.filter(name='check_if_friend')
def check_if_friend(friend_list, username):
    # loop through friend list
    for friend in friend_list.get('data'):
        # if the user is friend with current user, return the request status with the username
        if friend.get('request_from').get('user').get('username')==username or friend.get('request_to').get('user').get('username')==username:
            return (friend.get('request_status'), friend.get('request_from').get('user').get('username'))
    # if not return false and None
    return (False, None)
```

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. 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



```

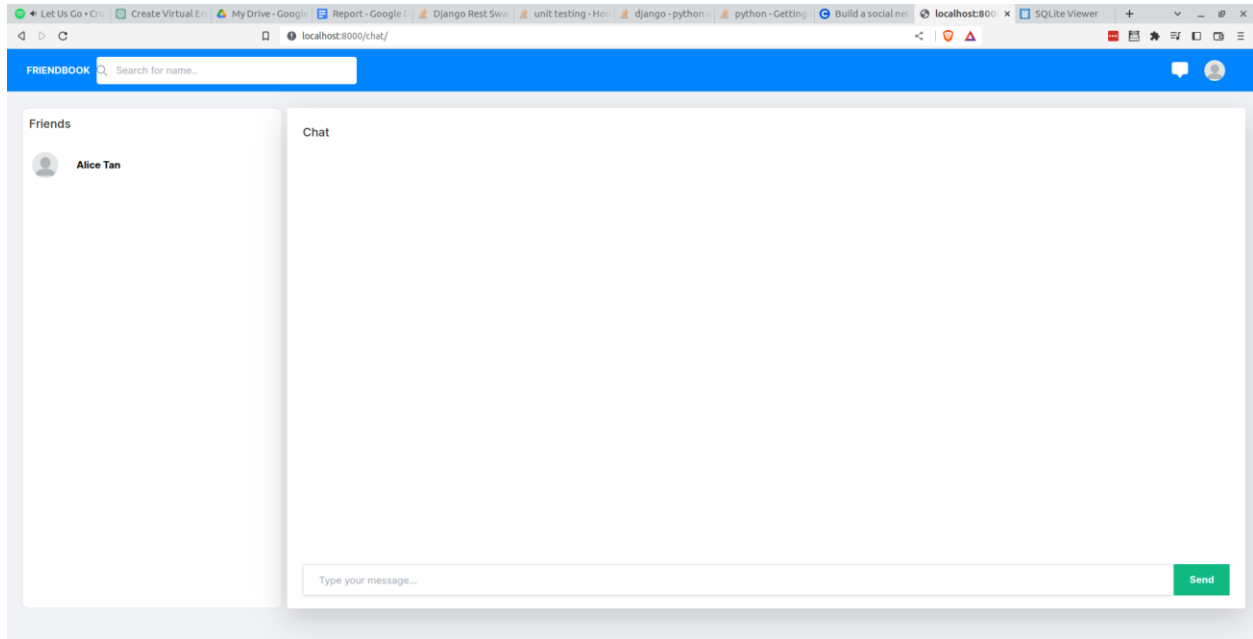
126 <!-- Add friend status -->
127 {% if user_profile.user.username != user.username and user.is_authenticated %}
128 <div id="friendRequest">
129   {% csrf_token %}
130   {% with friend=friend_list|check_if_friend:user.username %}
131   {% with status=friend.0 request_from=friend.1 %}
132   {% if status != False %}
133     {% if status == "Accepted" %}
134       <div class="flex flex-row-reverse">
135         <button id="rejectFriendButton" onclick="rejectFriend('{{user_profile.user.username}}')">
136           <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24">
137             fill="currentColor"
138             width="24"
139             height="24"
140             style="margin-right:8px;"
141             <path d="M14,8c0-2.21-1.79-4.4-4.56,5.79,6.85,1.79,4.4,4.514,10.21,14.8z M17,10v2h6v-2H17z M2,18v2h16v-2c0-2.66-5.33-4-8-4 S2,15.34,2,18z"/>
142           </svg>
143           Remove Friend
144         </button>
145       </div>
146     {% else %}
147       {% if request_from == user.username %}
148         <div class="flex flex-row-reverse">
149           <button id="rejectFriendButton" onclick="rejectFriend('{{user_profile.user.username}}')" class="ml-3">
150             <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24">
151               fill="currentColor"
152               width="24"
153               height="24"
154               style="margin-right:8px;"
155               <path d="M19 6.41L17.59 5 12 10.59 6.41 5 5 6.41 10.59 12 5 17.59 6.41 19 12 13.41 17.59 19 17.59 13.41 12z"/>
156               <path d="M0 0h24v24H0z" fill="none"/>
157             </svg>
158             Cancel Request
159           </button>
160           <span class="flex justify-center items-center">
161             <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24">
162               fill="currentColor"
163               width="24"
164               height="24"
165               style="margin-right:8px;"
166               <path d="M17,12c-2.76,0-5,2.24-5,5s2.24,5,5,5s2.76,0,5,2.24,5,5s19.76,12,12z M18.65,19.35l-2.15-2.15V14h1v2.79l1.85,1.85 L18.65,19.35z M18,3h-3.18C14.4
167             </svg>
168             Pending
169           </span>
170         </div>
171       {% else %}
172         <div class="flex flex-row-reverse">
173           <button id="acceptFriendButton" onclick="acceptFriend('{{user_profile.user.username}}')" class="ml-3">
174             <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24">
175               fill="currentColor"
176               width="24"
177               height="24"
178               style="margin-right:8px;"
179               <path d="M9 16.17L4.83 12l-1.42 1.41L9 21 7l-1.41-1.41z"/>
180             </svg>
181             Accept Request
182           </button>
183           <button id="rejectFriendButton" onclick="rejectFriend('{{user_profile.user.username}}')">
184             <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24">
185               fill="currentColor"
186               width="24"
187               height="24"
188               style="margin-right:8px;"
189               <path d="M19 6.41L17.59 5 12 10.59 6.41 5 5 6.41 10.59 12 5 17.59 6.41 19 12 13.41 17.59 19 17.59 13.41 12z"/>
190               <path d="M0 0h24v24H0z" fill="none"/>
191             </svg>
192             Reject Request
193           </button>
194         </div>
195       {% endif %}
196     {% endif %}
197   {% else %}
198     <div class="flex flex-row-reverse">
199       <button id="addFriendButton" onclick="addFriend('{{user_profile.user.username}}')">
200         <svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 24 24">
201           fill="currentColor"
202           width="24"
203           height="24"
204           style="margin-right:8px;"
205           <path d="M15 12c2.21 0 4-1.79 4-4s-1.79-4-4-4-4 1.79-4 4-4 4zm-9-2V7H4v3H1v2h3v3h2v-3h3v-2H6zm9 4c-2.67 0-8 1.34-8 4v2h16v-2c0-2.66-5.33-4-8-4"
206           </path>
207         </svg>
208         Add Friend
209       </button>
210     </div>
211   {% endif %}
212   {% endif %}
213   {% endif %}
214   {% endif %}
215   {% endif %}
216 </div>
217 {% endif %}

```

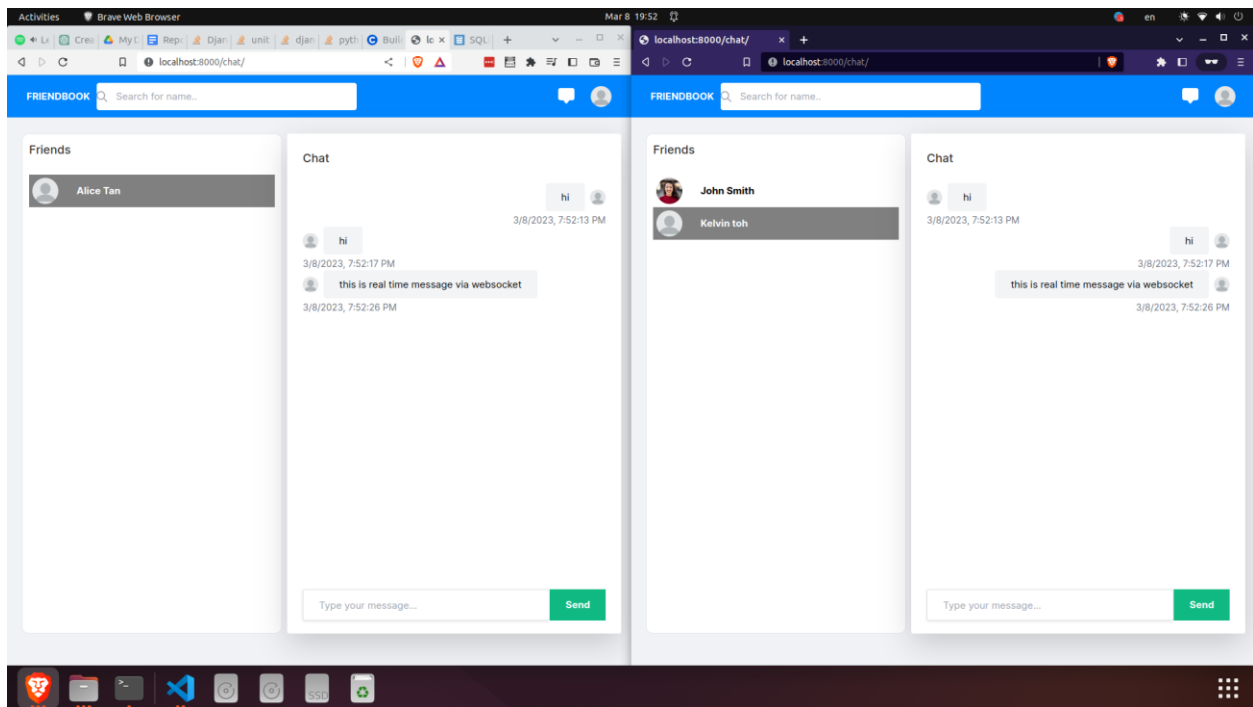
This is the part where buttons are being rendered, so we only render the buttons if the current user is not the profile user and they are logged in. After which we call the `check_if_friend`

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 real-time via websocket.

How this was achieved

### **socialNetwork/asgi.py**

```
yiquan, 2 weeks ago | 1 author (yiquan)
import os
from channels.auth import AuthMiddlewareStack
from channels.routing import ProtocolTypeRouter, URLRouter
from django.core.asgi import get_asgi_application
import friendbook.routing

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'socialNetwork.settings')

application = ProtocolTypeRouter({
    "http": get_asgi_application(),
    "websocket": AuthMiddlewareStack(
        URLRouter(friendbook.routing.websocket_urlpatterns)
    )
})
yiquan, 2 weeks ago • WIP web cw2 ...
```

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

### routing.py

```
✓ from django.urls import re_path      yiquan, 2 weeks ago
  from . import consumers

✓ websocket_urlpatterns = [
  |     re_path(r'ws/chat/(?P<room_name>[\w-]+)/$',
  |             consumers.ChatConsumer.as_asgi())
  | ]
```

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, 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 it will triggers the receive function which will then send the message data(serialize the ChatMessage object) to the chat\_message function. In the chat\_message function broadcast the message to all the users that are in the room.

### api.py

```

@api_view(['GET'])
def get_chat(request, chatUser):
    # get the user profile
    try:
        first_user = Profile.objects.get(
            user=User.objects.get(username=request.user))
        second_user = Profile.objects.get(
            user=User.objects.get(username=chatUser))

        # return 404 if user not found
    except (Profile.DoesNotExist, User.DoesNotExist) as err:
        return Response({'error': "Invalid user"}, status=status.HTTP_404_NOT_FOUND)

    # get the chat
    # if first_user is the current user and second_user is the targeted user
    # or first_user is the targeted user and second_user is the current user
    chat = Chats.objects.filter(Q(first_user=first_user, second_user=second_user) | Q(
        first_user=second_user, second_user=first_user))

    # if there are no chat record, create a chat object, save it and return 200
    # messages are saved from the websocket so saving a entry in the database here will
    # make websocket code to be neater
    if len(chat) == 0:
        chat_obj = Chats.objects.create(
            first_user=first_user, second_user=second_user)
        chat_obj.save()
        return Response({'data': {'chat': ChatSerializer(chat_obj).data}}, status=status.HTTP_200_OK)

    # return all the chat plus all the messages in the response
    return Response({'data': {'chat': ChatSerializer(chat.first()).data}}, status=status.HTTP_200_OK)

```

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.

```

<!-- Friends list --> yiquan, 2 weeks ago • WIP web cw2
<div class="chatFriendContainer">
  <h2 class="text-lg font-medium mb-4">Friends</h2>
  {% for friend in friends|get_accepted_friend:user.username %}
  <div class="flex items-center searchEntry" id="friendList" style="padding:5px;" onclick="setActiveFriend(this, '{{friend.user.username}}')">
    <div class="w-10 h-10 rounded-full relative flex-shrink-0 mr-1">
      
    </div>
    <span class="text-m ml-4 font-bold">
      {{friend.user.first_name}} {{friend.user.last_name}}
    </span>
  </div>
  {% endfor %}
</div>

```

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.

```

function setActiveFriend(element,friendEmail) {
  getChat(friendEmail)

  // remove the active class from all friends
  const friends = document.querySelectorAll('#friendList');

  for (let index = 0; index < friends.length; index++) {
    friends[index].classList.remove('active');
  }

  // add the active class to the selected friend
  element.classList.add('active');
}

// get the selected user chat history and create a websocket connection
function getChat(friendEmail){
  fetch('/api/get-chat/'+friendEmail).then(response =>
    response.json()
  ).then(({data})=>{
    if(data){
      // room id from the api which queried from the database
      const roomName = data.chat.room_id

      // setup websocket
      initChatSocket(roomName);

      // remove all existing chat messages
      let chatMessageBox = document.getElementsByClassName('chatMessageBox')[0]
      chatMessageBox.innerHTML = ''

      // loop through the chat messages and display it in the chat box
      data.chat.chat_messages.forEach(({created_at,message,message_from})=>{
        updateChatBox(created_at,message,message_from)
      })
    }
  })
}

```

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.

```

function initChatSocket(roomName) {
  // setup a websocket connection
  chatSocket = new WebSocket(`ws://${window.location.host}/ws/chat/${roomName}/`);

  // check if chatSocket is not null as the page load the chatSocket is set to null until the user
  // click on a friend which then will open the websocket for that particular room_id
  if (chatSocket != null) {
    // when receive a message from the websocket
    chatSocket.onmessage = function(e) {
      // destructure the data
      const data = JSON.parse(e.data);
      const { created_at, message, message_from } = data.message;

      // update the chatbox to display the message
      updateChatBox(created_at, message, message_from);
    };

    chatSocket.onclose = function(e) {
      // when chatSocket closed, we will reconnect to the web socket every 5s
      console.error("Chat socket closed unexpectedly");

      setTimeout(() => {
        console.log('WebSocket reconnecting...');
        initChatSocket(roomName);
      }, 5000);
    };
  }
}

```

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

```

// display messages in the chatbox
function updateChatBox(created_at,message,message_from){
  // get the chatMessageBox div using className, it will return an array and we want the first result of the array since there is only 1
  let chatMessageBox = document.getElementsByClassName('chatMessageBox')[0]

  // convert the utc datetime to user current timezone datetime
  const localTime = new Date(created_at).toLocaleString()
  let htmlStr = '';

  // if the message is not from the current user, set the chat bubble to the left
  // else the chat bubble will be from the right
  if(message_from.user.username!==currentUsername){
    htmlStr = `
    <div class="chatBubbleLeft">
      <div class="flex items-center">
        
        <p class="bg-gray-100 rounded-lg px-4 py-2">${message}</p>
      </div>
      <p class="text-sm text-gray-500 mt-1">${localTime}</p>
    </div>`;
  }
  else{
    htmlStr=`
    <div class="chatBubbleRight">
      <div class="flex items-center">
        <p class="bg-gray-100 rounded-lg px-4 py-2 mr-2">${message}</p>
        
      </div>
      <p class="text-sm text-gray-500 mt-1 self-end">${localTime}</p>
    </div>`;
  }

  // update the chatMessageBox dom
  chatMessageBox.innerHTML+=htmlStr;

  // set the chatMessageBox scrollTop to the scrollHeight which it will always
  // scroll down to show the latest message
  chatMessageBox.scrollTop = chatMessageBox.scrollHeight;
}

```

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.

```
<!-- Chatbox -->
<div class="ml-2 flex-1 overflow-y-scroll chatContainer">
  <div class="rounded-lg shadow-lg p-4 h-full flex flex-col">
    <h2 class="text-lg font-medium mb-4">Chat</h2>
    <div class="flex flex-col chatMessageBox">
    </div>
    <div class="mt-4 flex">
      <input type="text" class="rounded-l-lg flex-1 py-2 px-4 border border-gray-300" onkeydown="handleInputKeyDown(event)" id="messageInput" placeholder="Type your message" />
      <button class="bg-green-500 hover:bg-green-600 text-white font-bold py-2 px-4 rounded-r-lg" id="sendMessageBtn" onclick="sendMessage(event)">Send</button>
    </div>
  </div>
</div>
```

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

```
// trigger when keydown for input field
function handleInputKeyDown(event){
  // when enter is pressed
  if (event.keyCode === 13) {
    // get the sendMessageBtn and trigger it's click function
    const sendMessageBtn = document.getElementById('sendMessageBtn');
    sendMessageBtn.click();
  }
}

// send message to websocket
function sendMessage(event){
  // get the messageInput and it's value
  const messageInput = document.getElementById('messageInput');
  const message = messageInput.value;

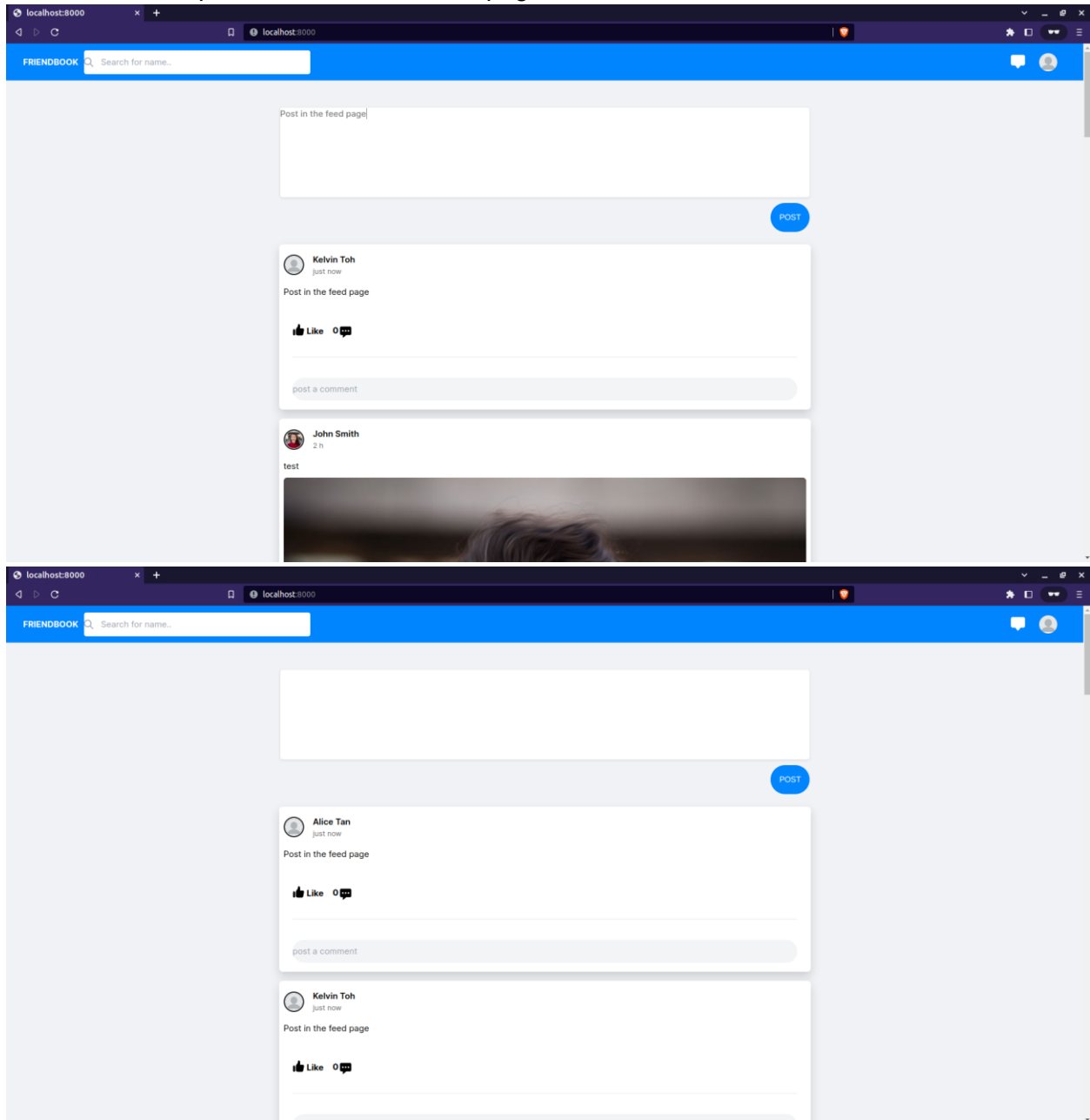
  // send the message to websocket if it's length is bigger than 0
  if(message.trim().length>0){
    if(chatSocket!=null){
      chatSocket.send(JSON.stringify({
        'message': message,
      }));
      // set the messageInput to be empty string which "reset" the value of the input
      messageInput.value = '';
    }
  }
}
```

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.

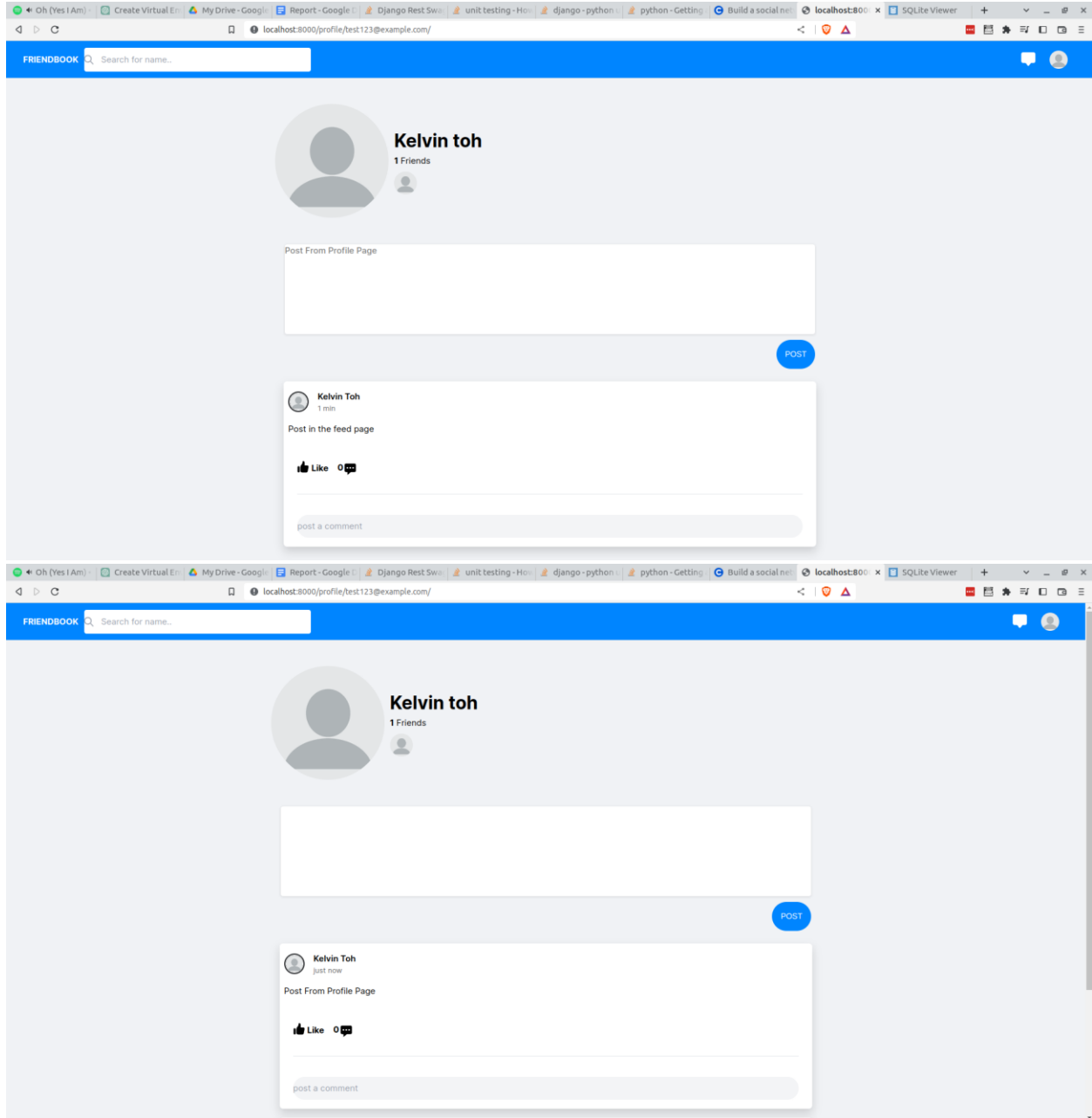


R1f & R1g. Users can add status updates to their home page and Users can add media (such as images to their account and these are accessible via their home page

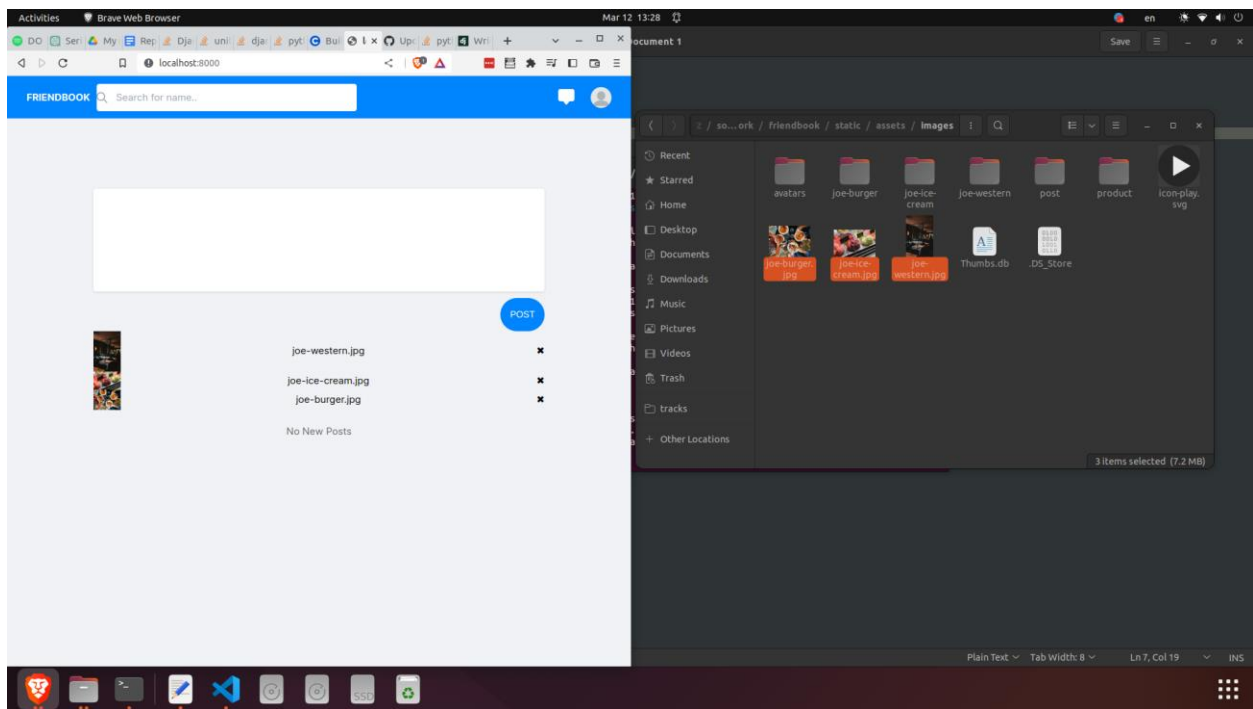
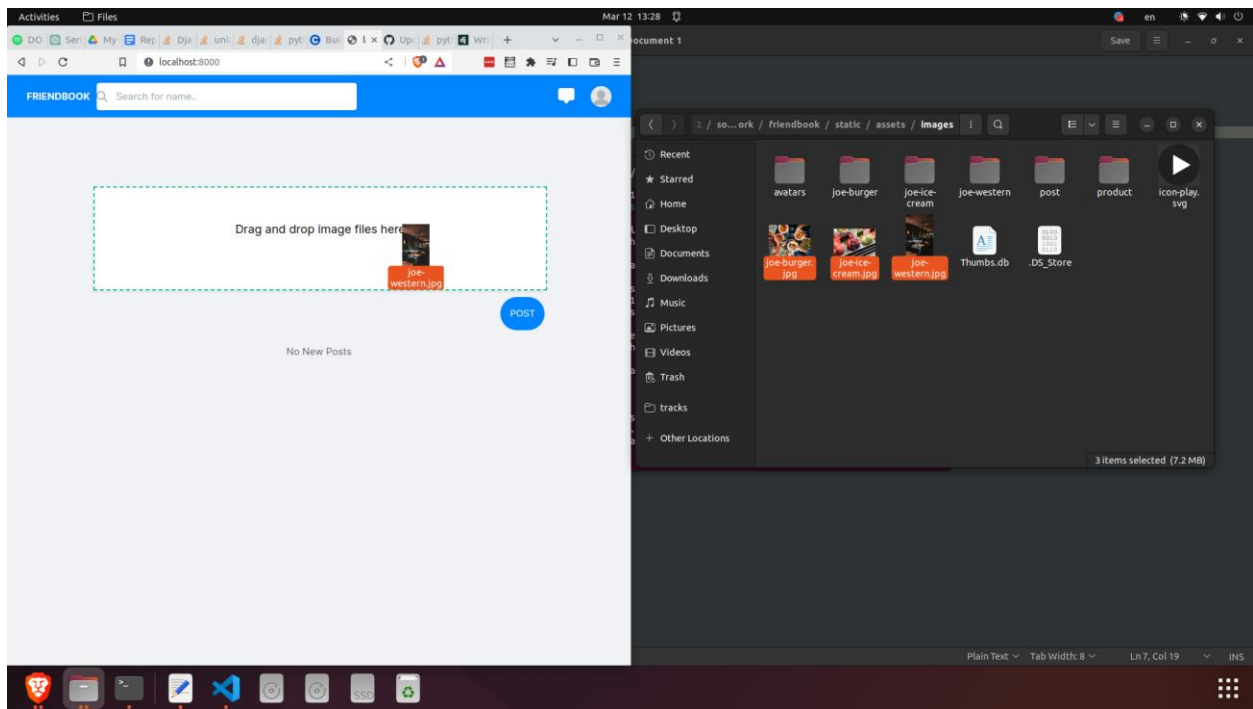
### 1. User can post status from the feed page

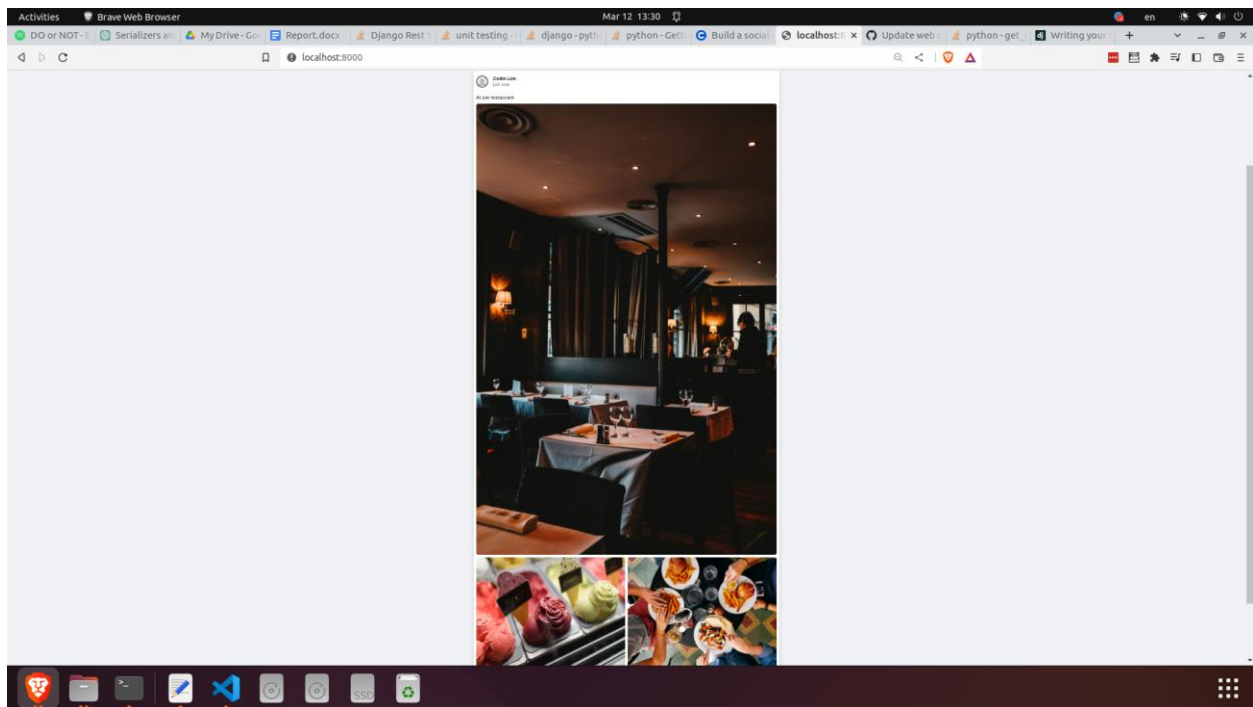


## 2. User can post status on their profile page



## 3. 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

```
<div class="container m-auto">
  <div class="space-y-5 flex-shrink-0 col-12">
    <!-- Upload Post Section -->
    <div
      class="postDiv"
      ondragenter="fileDrag(event)"
      ondragleave="hideFileArea(event)"
      ondragover="fileDrag(event)"
      ondrop="dropHandler(event)"
    >
      <form id="uploadPostForm">
        {% csrf_token %}
        <textarea rows="7" cols="70" id="caption"></textarea>
        <div class="flex justify-end">
          <button onclick="submitPost(event)" class="postButton">Post</button>
        </div>

        <div class="fileArea hidden" id="fileArea">
          <input type="file" multiple accept="image/*" />
          <div class="fileDummy">
            <span>Drag and drop image files here</span>
          </div>
        </div>
      </form>

      <div id="imageList" class="imageList"></div>
    </div>
  </div>
</div>
```

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.

```
yiquan, 6 days ago | 1 author (yiquan)
// show the fileArea div when file is dragged to the textbox yiquan, last week • Update wel
0 references
function fileDrag(event) {
  const fileArea = document.getElementById("fileArea");
  fileArea.classList.remove("hidden");
}

// hide the fileArea div when the file mouse drag exit the textbox
0 references
function hideFileArea(event) {
  const fileArea = document.getElementById("fileArea");
  fileArea.classList.add("hidden");
}

0 references
function dropHandler(ev) {
  // Prevent default behavior (Prevent file from being opened)
  ev.preventDefault();

  if (ev.dataTransfer.items) {
    // Use DataTransferItemList interface to access the file(s)
    0 references
    [...ev.dataTransfer.items].forEach((item, i) => {
      // If dropped items aren't files, reject them
      if (item.kind === "file") {
        file = item.getAsFile();
        readImage(file);
      }
    });
  } else {
    // Use DataTransfer interface to access the file(s)
    0 references
    [...ev.dataTransfer.files].forEach((file, i) => readImage(file));
  }

  // hide fileArea div
  const fileArea = document.getElementById("fileArea");
  fileArea.classList.add("hidden");
}
```

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.

```

36 function readImage(file) {
37     // create a file reader
38     const reader = new FileReader();
39     // 0 references
40     reader.addEventListener("load", (event) => {
41         // set the base64 to a variable
42         const uploadedImage = event.target.result;
43
44         // get the imageList div
45         const imageList = document.getElementById("imageList");
46         // create a div with image, file name and a cross button
47         const img = createImage(file, uploadedImage);
48         // append the div imageList
49         imageList.appendChild(img);
50     });
51     // read the file and trigger the load function
52     reader.readAsDataURL(file);
53 }
54

```

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.

```

54 1 reference
55 ✓ function createImage(file, uploadedImage) {
56     // create a div
57     const div = document.createElement("div");
58
59     // create a img tag with the base64 loaded
60     const img = new Image();
61     img.src = uploadedImage;
62
63     // create a span with the file name
64     const nameSpan = document.createElement("span");
65     nameSpan.id = "title";
66     nameSpan.innerText = file.name;
67
68     // create a cross button
69     const deleteSpan = document.createElement("span");
70     deleteSpan.innerHTML = "&#10006;";
71     deleteSpan.classList.add("buttonCursor");
72     1 reference
73     deleteSpan.onclick = () => document.getElementById(file.name).remove();
74
75     // append all the element into a div
76     div.id = file.name;
77     div.appendChild(img);
78     div.appendChild(nameSpan);
79     div.appendChild(deleteSpan);
80
81     // return the div
82     return div;
83 }

```

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.

```

84 function submitPost(event) {
85     event.preventDefault();
86     // get the caption value
87     const caption = document.getElementById("caption").value;
88     // get the imageList div
89     const images = document.getElementById("imageList");
90     // get all the element with img tag
91     const imgList = images.getElementsByTagName("img");
92
93     const imageData = [];
94
95     // loop through all the img element and get it's src(base64)
96     for (img of imgList) {
97         imageData.push(img.getAttribute("src"));
98     }
99
100    // send a post request to upload the post
101    fetch("/api/upload-post/", {
102        method: "POST",
103        headers: {
104            "X-CSRFToken": document.getElementsByName("csrfmiddlewaretoken")[0].value,
105            Accept: "application/json",
106            "Content-Type": "application/json",
107        },
108        body: JSON.stringify({ caption, images: imageData }),
109    }).then((data) => {
110        if (data.status === 200) {
111            // reset the value of caption and imageList
112            document.getElementById("caption").value = "";
113            document.getElementById("imageList").innerHTML = "";
114
115            location.reload();
116        }
117    });
118 }
119

```

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

```
@api_view(['POST'])
def upload_post(request):
    user = None

    # get the current user
    try:
        user = Profile.objects.get(
            user=User.objects.get(username=request.user))

    # if user doesn't exist return 404 error
    except (Profile.DoesNotExist, User.DoesNotExist) as err:
        return Response({'error': "Invalid user"}, status=status.HTTP_404_NOT_FOUND)

    # get the caption and images(list of base64) from the request
    caption = request.data.get('caption')
    images = request.data.get('images')

    # create a Post object with the user and caption
    new_post = Post.objects.create(user=user, caption=caption)

    # loop through the images
    for image in images:
        # split the base64 into format and the base64 data
        format, imgstr = image.split(';base64,')
        # get the extension type of the file
        ext = format.split('/')[1]

        # try to store the base64 image into a jpeg/jpg/png file on local disk
        try:
            file = ContentFile(base64.b64decode(imgstr),
                               name=str(uuid.uuid4()) + "." + ext)
            # if file cannot be save/base64 is invalid, return error 400
        except binascii.Error:
            new_post.delete()
            return Response({'error': 'Incorrect padding in base64 string'}, status=status.HTTP_400_BAD_REQUEST)

        # once the file is saved to the local disk, create a PostImage object and set the file as image field value
        post_image = PostImage.objects.create(image=file)

        # add the post_image into the post
        new_post.post_image.add(post_image)

    # once all the image are saved and added to the post, save the post object
    new_post.save()

    # serialize the new post and send it back as response
    return Response({'data': PostSerializer(new_post).data}, status=status.HTTP_200_OK)
```

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

```
6 gender_choice=(
7     ("M", "Male"),
8     ("F", "Female"),
9     ("O", "Other"),
10 )
11
12 request_status=( ('Accepted','Accepted'), ('Declined','Declined'), ('Pending','Pending'))
13 # Create your models here.
14
15 yiquan, 3 days ago | 1 author (yiquan)
16 class Profile(models.Model):
17     user = models.OneToOneField(User, on_delete=models.CASCADE)
18     date_of_birth = models.DateField(auto_now=False, auto_now_add=False)
19     gender = models.CharField(
20         max_length=1,
21         choices=gender_choice,
22         null=False,
23         blank=False
24     )
25     profile_image = models.ImageField(
26         upload_to='./friendbook/static/media/profile_images', default='./friendbook/static/media/default_picture.png')
27
28     def __str__(self):
29         return self.user.username
30
31 yiquan, last month | 1 author (yiquan)
32 class PostImage(models.Model):
33     image = models.ImageField(upload_to='./friendbook/static/media/post_image')
34
35 yiquan, 3 days ago | 1 author (yiquan)
36 class PostComment(models.Model):
37     comment = models.CharField(
38         max_length=10000,
39         null=False,
40         blank=False
41     )
42     posted_by = models.ForeignKey(Profile, on_delete=models.DO_NOTHING)
43     created_at = models.DateTimeField(default=timezone.now)
44
45     def __str__(self):
46         return self.posted_by.user.username
```

```

45 class PostLike(models.Model):
46     liked = models.BooleanField()
47     liked_by = models.ForeignKey(Profile, on_delete=models.DO_NOTHING)
48
49     def __str__(self):
50         return self.liked_by.user.username
51
yi quan, 3 days ago | 1 author (yiquan)
52 class Post(models.Model):
53     id = models.UUIDField(primary_key=True, default=uuid.uuid4)
54     user = models.ForeignKey(Profile, on_delete=models.DO_NOTHING)
55     caption = models.TextField()
56     created_at = models.DateTimeField(default=timezone.now)
57     likes = models.ManyToManyField(
58         PostLike, through='PostLikeLink', through_fields=('post', 'likes'))
59     post_image = models.ManyToManyField(
60         PostImage, through='PostImageLink', through_fields=('post', 'post_image'))
61     comment = models.ManyToManyField(
62         PostComment, through='PostCommentLink', through_fields=('post', 'comment'))
63
yi quan, last month | 1 author (yiquan)
64 class Meta:
65     ordering = ['-created_at']
66
67     def __str__(self):
68         return self.user.user.username
69
yi quan, last month | 1 author (yiquan)
70 class PostCommentLink(models.Model):
71     post = models.ForeignKey(Post, on_delete=models.DO_NOTHING)
72     comment = models.ForeignKey(PostComment, on_delete=models.DO_NOTHING)
73
yi quan, 5 months ago | 1 author (yiquan)
74 class PostImageLink(models.Model):
75     post = models.ForeignKey(Post, on_delete=models.DO_NOTHING)
76     post_image = models.ForeignKey(PostImage, on_delete=models.DO_NOTHING)
77
yi quan, last month | 1 author (yiquan)
78 class PostLikeLink(models.Model):
79     post = models.ForeignKey(Post, on_delete=models.DO_NOTHING)
80     likes = models.ForeignKey(PostLike, on_delete=models.CASCADE)
81

```

```

yi quan, 3 weeks ago | 1 author (yiquan) and 1 other
class Friends(models.Model):
    request_from = models.ForeignKey(Profile, on_delete=models.DO_NOTHING, related_name="from_user")
    request_to = models.ForeignKey(Profile, on_delete=models.DO_NOTHING, related_name="to_user")
    request_status = models.CharField(max_length=10, choices=request_status, null=False, blank=False)

yi quan, 3 weeks ago | 1 author (yiquan)
class ChatMessage(models.Model):
    message = models.CharField(
        max_length=10000,
        null=False,
        blank=False
    )
    message_from = models.ForeignKey(Profile, on_delete=models.DO_NOTHING)
    created_at = models.DateTimeField(default=timezone.now)

yi quan, 3 weeks ago | 1 author (yiquan)
class Meta:
    ordering = ['created_at']

yi quan, 2 weeks ago | 1 author (yiquan)
class Chats(models.Model):
    first_user = models.ForeignKey(Profile, on_delete=models.DO_NOTHING, related_name="first_user")
    second_user = models.ForeignKey(Profile, on_delete=models.DO_NOTHING, related_name="second_user")
    room_id = models.UUIDField(default=uuid.uuid4, null=False, blank=False)
    chat_messages = models.ManyToManyField(ChatMessage, through='ChatMessageLink', through_fields=('chat', 'message'))

yi quan, 3 weeks ago | 1 author (yiquan)
class ChatMessageLink(models.Model):
    chat = models.ForeignKey(Chats, on_delete=models.DO_NOTHING)
    message = models.ForeignKey(ChatMessage, on_delete=models.DO_NOTHING)

```



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. The api should use a JWT(json web token) for the Authorization field in the header which I don't have the time to research and implement for my api therefore, there is an api that uses Authorization field with email which is bad practice. 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 and also proper JWT authentication for 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 on.

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 compared to nodejs, I would say Django might have some benefits over nodejs but those benefits nodejs can overcome easily. Examples are Django can do database migration with models which is really

convenient, the setup to get the server running is very minimal, 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 powerful side of Django is that queryset can query foreign key tables without writing inner join query and paired with serializers, it can create a good and maintainable backend. 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 best framework but rather a 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, exposed to another alternative backend framework other than nodejs and finally websocket which I have never touch during the time I work.

**Word count: 5934**

## 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](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>