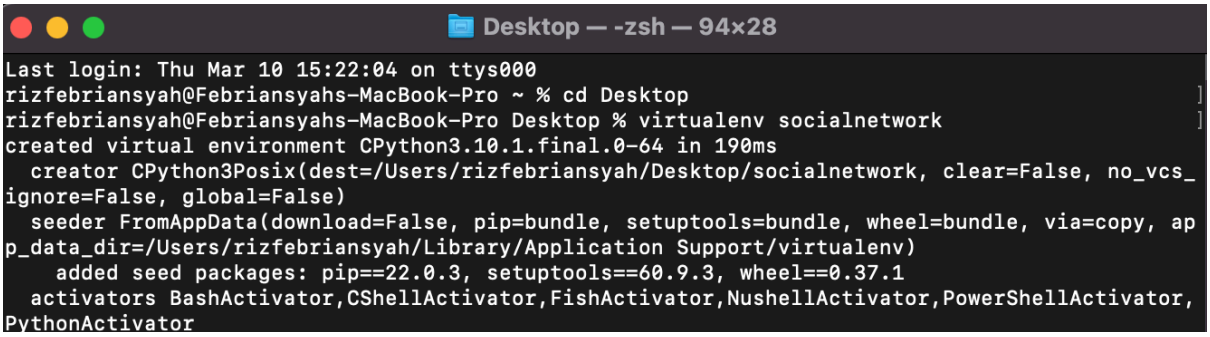
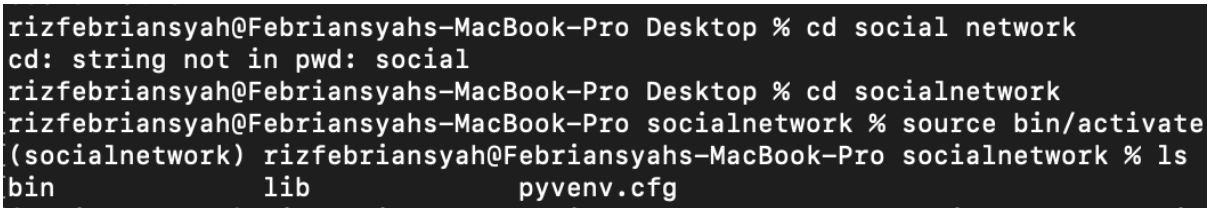
I will be going through on how I created my social networking app, called Friztalk.

admin credentials
e-mail: riz@gmail.com
username: riz
password: test12345

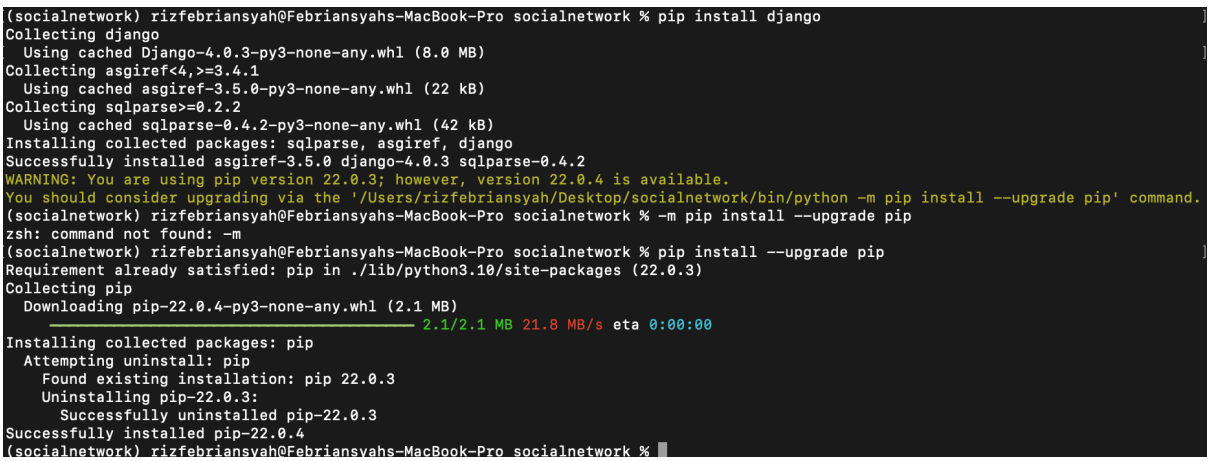
On terminal, I would like to create a virtual environment called socialnetwork. I have decided to save it at my desktop. So, we will go to my Desktop directory (cd Desktop), and afterwards type: virtualenv socialnetwork



We need to enter the folder with virtual environment: cd socialnetwork
Activate the virtual environment: source bin/activate



Afterwards, we need to install Django: pip install Django. I have also updated to the latest version of 22.0.4



Start a new Django project: django-admin startproject friztalk
In the terminal, rename project file to src: mv friztalk src
Go to the source folder: cd src

```
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro socialnetwork % django-admin startproject friztalk
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro socialnetwork % ls
bin             friztalk        lib             pyvenv.cfg
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro socialnetwork % cd friztalk
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro friztalk % ls
friztalk        manage.py
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro friztalk % pwd
/Users/rizfebriansyah/Desktop/socialnetwork/friztalk
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro friztalk % cd ..
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro socialnetwork % mv friztalk src
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro socialnetwork % cd src
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro src % ls
friztalk        manage.py
```

Afterwards, do the migration: python manage.py migrate (this command is responsible for applying migrations. It creates the tables in the database file)

```
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro src % python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
```

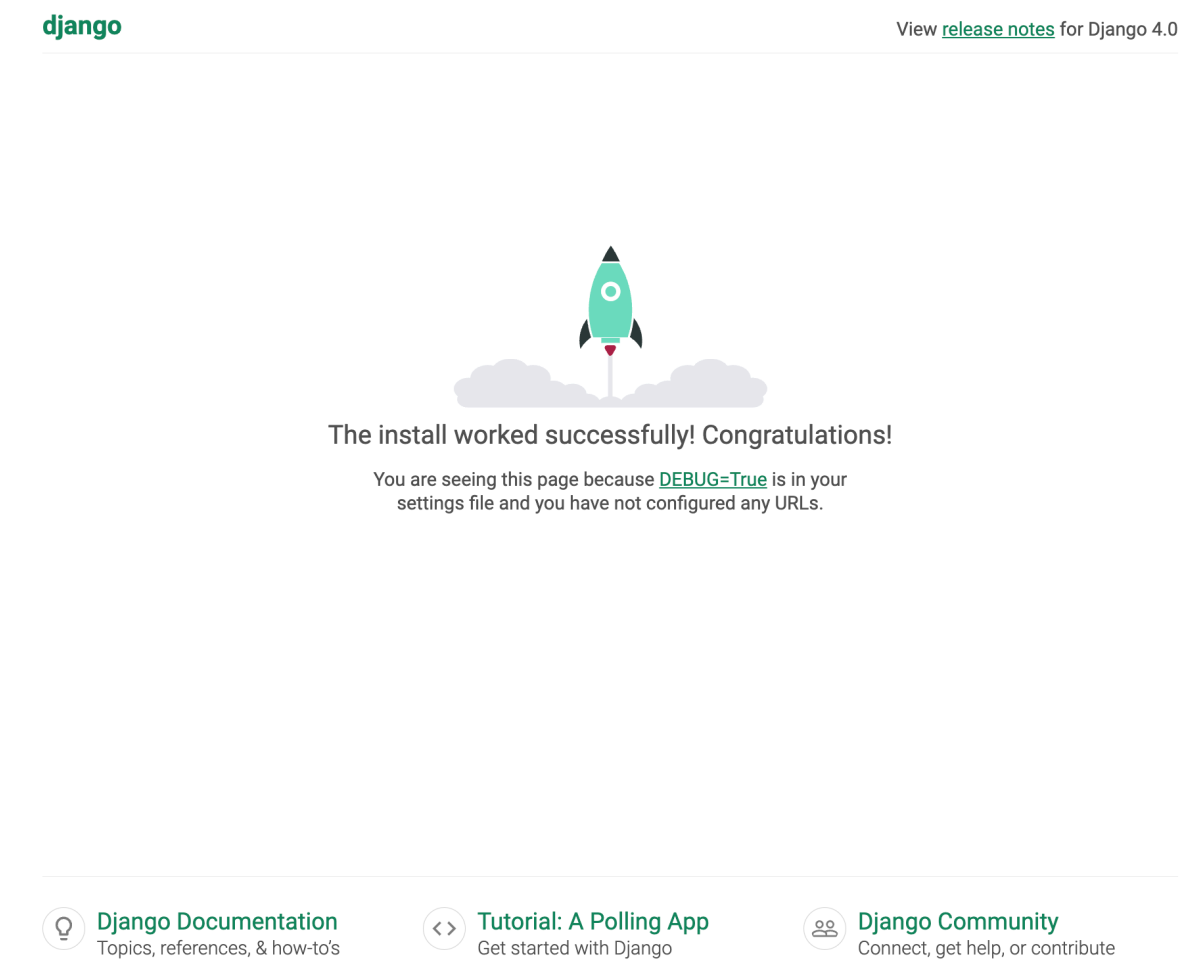
Now, let's create a super user: python manage.py createsuperuser
password: test12345

```
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro src % python manage.py createsuperuser
Username (leave blank to use 'rizfebriansyah'):
Email address:
Password:
Password (again):
Superuser created successfully.
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro src %
```

Testing the server by: python manage.py runserver

```
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro src % python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
March 10, 2022 - 07:37:36
Django version 4.0.3, using settings 'friztalk.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Login to admin -> in the browser type: http://127.0.0.1:8000/



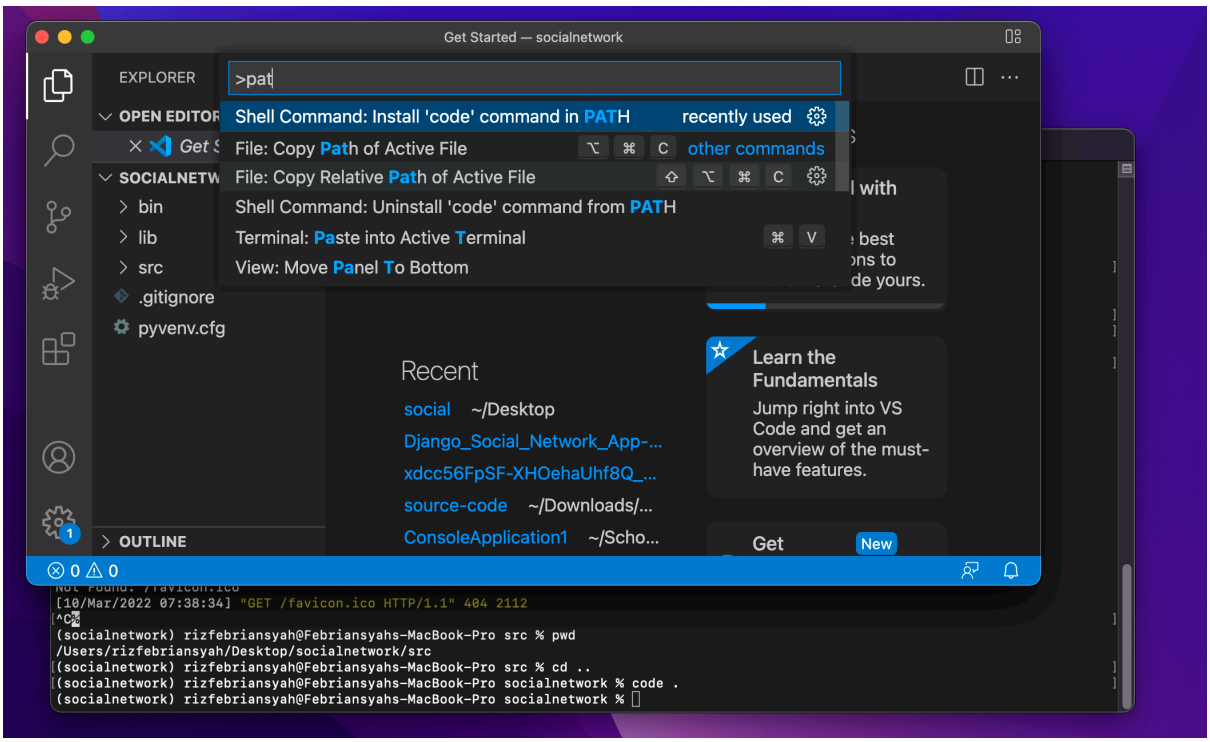
manage.py is a file which is added to each Django project, and it is created after starting a project. It simply helps with the management of the site.

python manage.py makemigrations – this generates the sql commands to create the table of. each class defined in the models.py

python manage.py migrate – this command is responsible for applying migrations. It creates the tables in the database file.

Quit the server by typing control C.

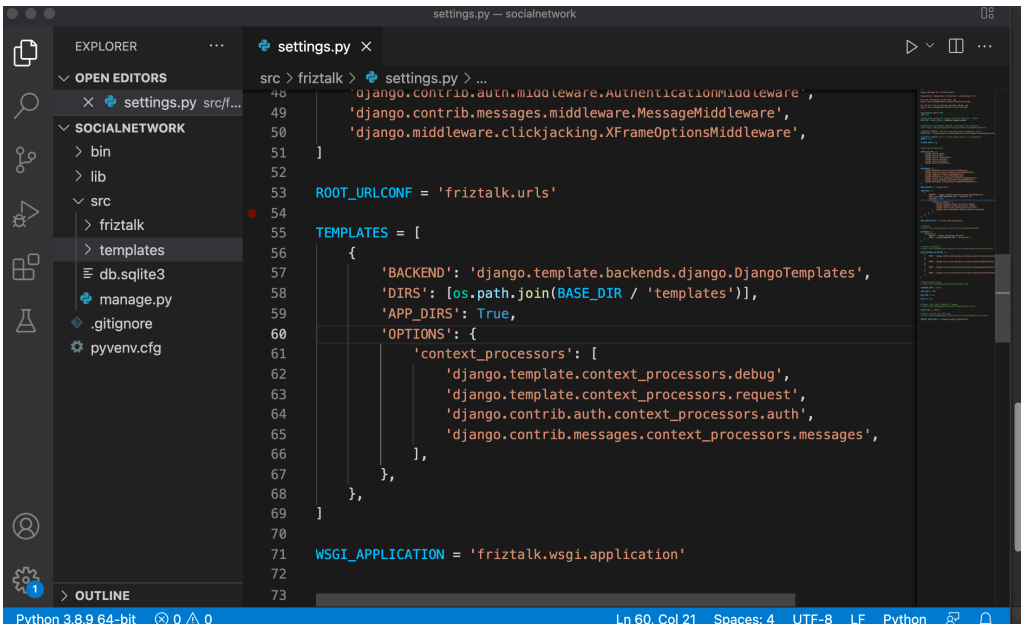
Then proceed to the social directory by typing: cd ..
Open visual studio code by typing: code .



os.path.dirname(path): this returns the name of the file from the given path
os.path.basename(path): this returns the directory of the file with the name of the file
os.path.join(path): this returns a path merged out of multiple path components

On visual studio code,
Under src > friztalk, in settings.py.
We need to add this under ‘DIRS’:

```
[os.path.join(BASE_DIR / 'templates')]
```



Under settings.py, we need to add:

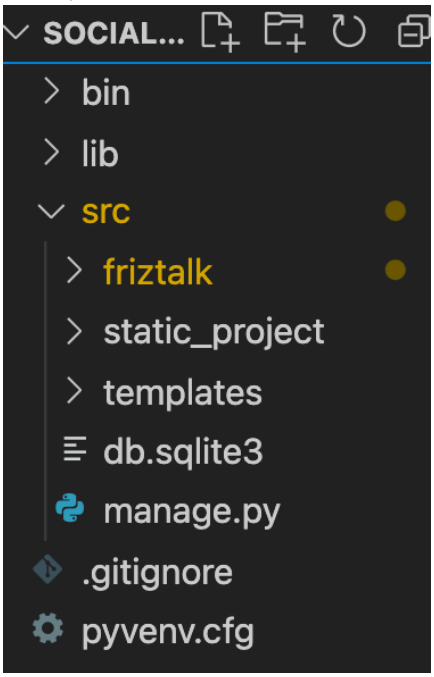
```
STATIC_URL = '/static/'

# Default primary key field type
# https://docs.djangoproject.com/en/4.0/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static_project')
]

STATIC_ROOT = os.path.join(os.path.dirname(BASE_DIR), "static_cdn", "static_root")

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(os.path.dirname(BASE_DIR), "static_cdn", "media_root")
```

Under urls.py, we need to add:

```
from django.contrib import admin
from django.urls import path
from django.conf import settings
from django.conf.urls.static import static
```

```
urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

Now, we need to create two new folders under src, templates and static_project



Once done, go to the terminal, go to the src folder (cd src) and type:
python manage.py collectstatic

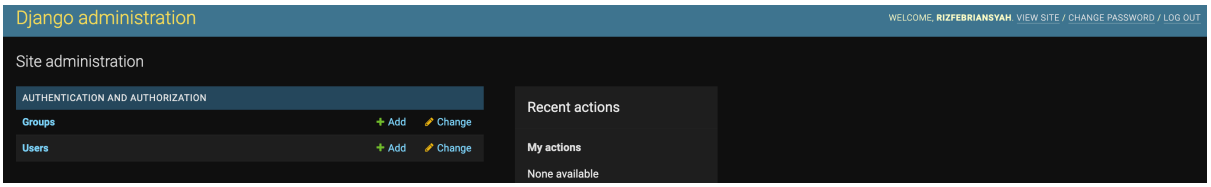
128 static files are copied

```
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro socialnetwork % code .
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro socialnetwork % cd src
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro src % python manage.py collectstatic

128 static files copied to '/Users/rizfebriansyah/Desktop/socialnetwork/static_cdn/static_root'.
```

Now, it is time to run the server:
python manage.py runserver

```
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro src % python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
March 10, 2022 - 08:07:18
Django version 4.0.3, using settings 'friztalk.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

In the browser type : http://127.0.0.1:8000/admin
Login (username: rizfebriansyah, password: test12345)



Remarks:
Static files are dedicated to javascript, css files, images while media files are meant for content that the user uploads
We distinguish MEDIA_ROOT and STATIC_ROOT as separate directories within the static_cdn folder
To the STATIC_ROOT will be copied to this directory after running collectstatic command with the purpose to serve all the static files from a single location
MEDIA_ROOT will be updated each time a user uploads something to the server

STATICFILES_DIRS will inform Django where to look for additional static files outside of the static folder in the application directory. If the applications do not contain static folders, it will be the place to store static files.

//////////////////

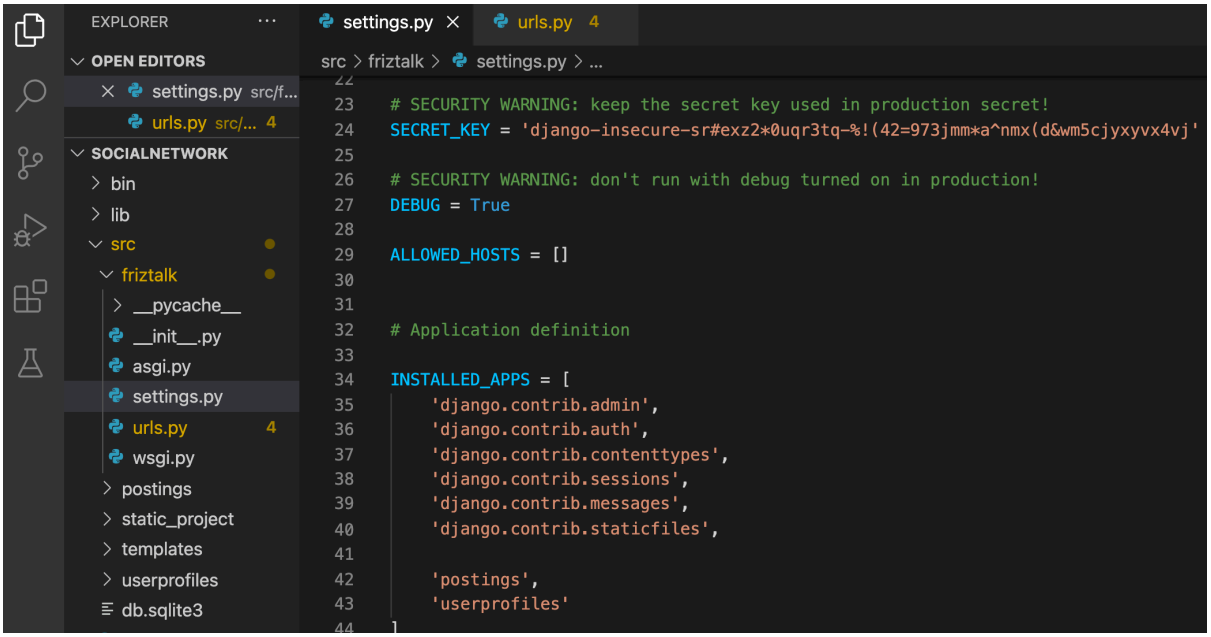
I will create 3 applications in total.
userprofiles, postings and allauth(authentication)

Type this code to create the two apps and two folders (profiles and posts) under src:

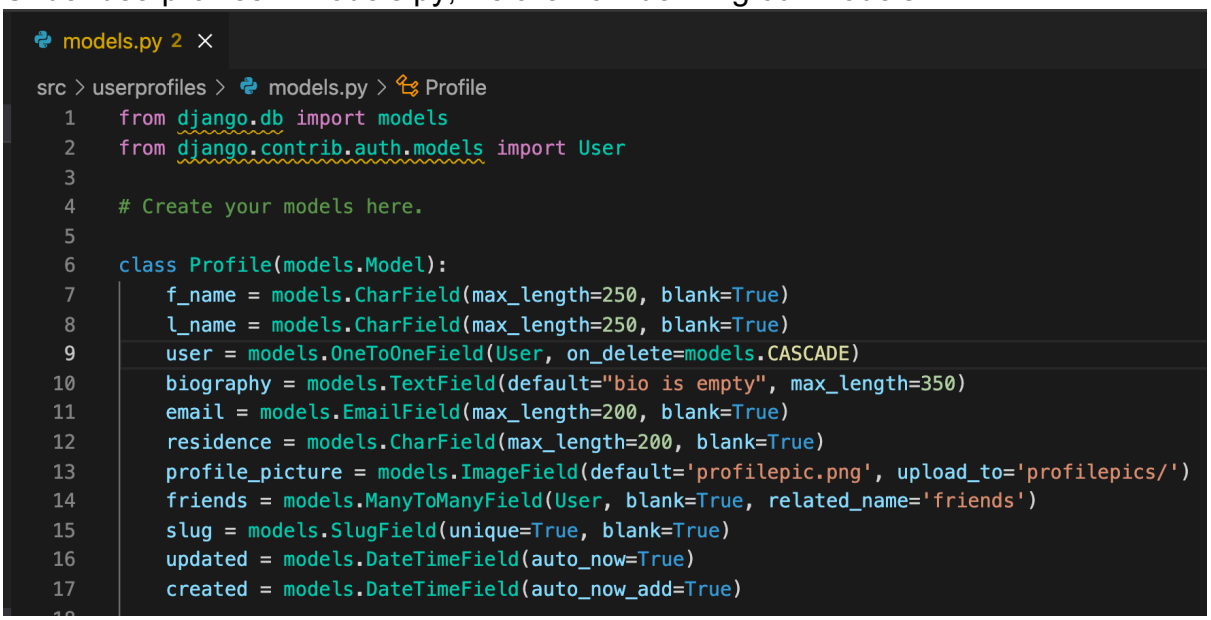
```
python manage.py startapp postings
python manage.py startapp userprofiles
```

```
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro src % python manage.py startapp postings
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro src % python manage.py startapp userprofiles
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro src %
```

Add ‘postings’, ‘userprofiles’, under INSTALLED_APPS on settings.py:



Under userprofiles > models.py, we are now defining our models:



Now I add a string representation method:

```
    def __str__(self):
        return f"{self.user.username}-{self.created.strftime('%d-%m-%Y')}"
```

Next, I need to type:
pip install pillow
python manage.py makemigrations
python manage.py migrate

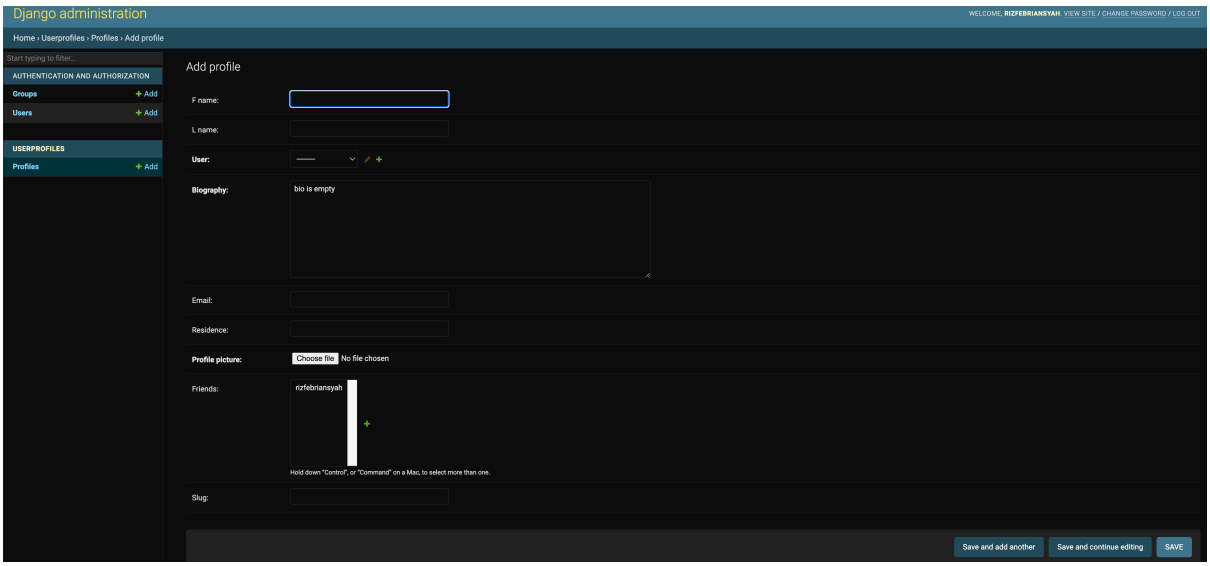
```
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro src % pip install pillow
Collecting pillow
  Using cached Pillow-9.0.1-1-cp310-cp310-macosx_11_0_arm64.whl (2.7 MB)
Installing collected packages: pillow
Successfully installed pillow-9.0.1
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro src % python manage.py makemigrations
Migrations for 'userprofiles':
  userprofiles/migrations/0001_initial.py
    - Create model Profile
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro src % python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, userprofiles
Running migrations:
  Applying userprofiles.0001_initial... OK
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro src %
```

Then run the server by typing:
python manage.py runserver

I need to register Profile in the admin. Therefore, I need to head into the admin.py file and import our models. Add this into the admin.py file:

```
from django.contrib import admin
from .models import Profile
# Register your models here.

admin.site.register(Profile)
```

Afterwards, I refresh http://127.0.0.1:8000/admin and I can now add a profile.



We need to think of a situation where we will have 2 users which has the same first and last name. So, what I plan to generate a random character under slug. For example, the first profile with first and last name ‘Tom’ and ‘Jerry’ respectively will have a slug ‘Tom-Jerry’. If there is another user who has the same name, the slug will be ‘Tom-Jerry-2423jwde’.

We create a helper function and store it under utils.py file (uuid is universal unique identifier)

```
import uuid

def generate_random_character():
    randomcode = str(uuid.uuid4())[:9].replace('-','').lower()
    return randomcode
```

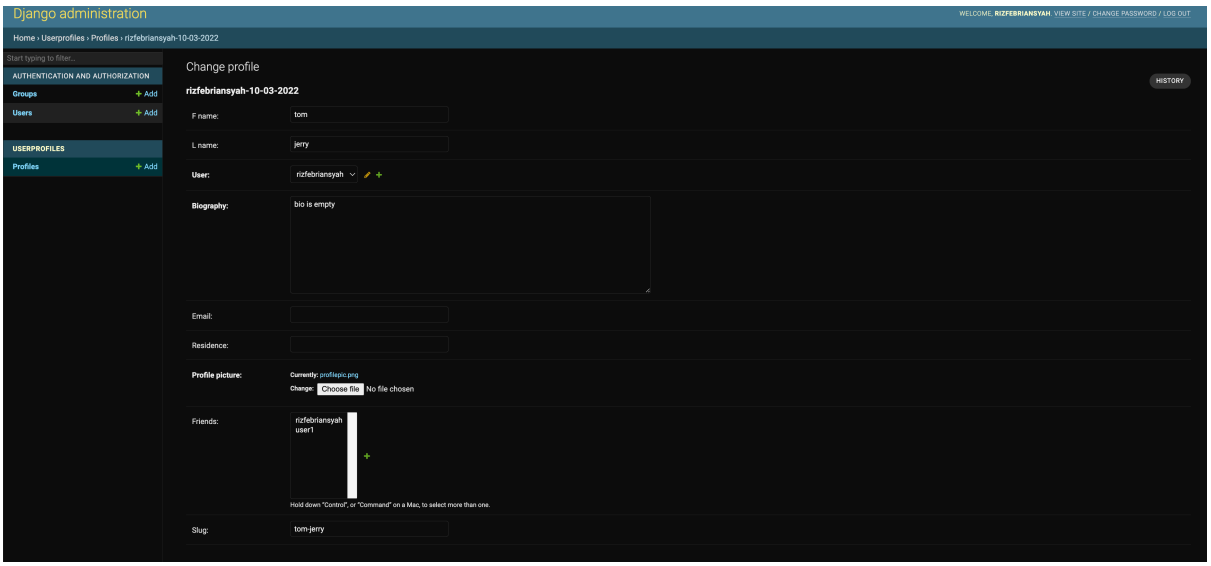
We now add these codes at models.py to import the packages

```
from .utils import generate_random_character
from django.template.defaultfilters import slugify
```

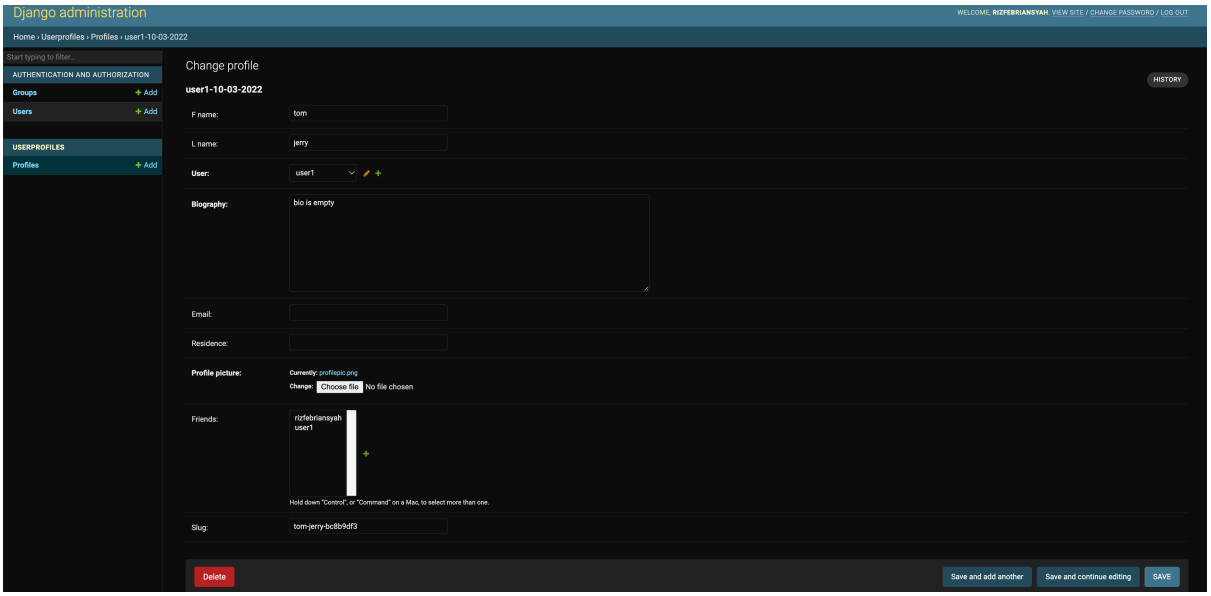
Now I need to check if the first name and last name exists. If they exist, I will create a slug out of their first name and last name. In other cases, if someone has the same first and last name, the generate random character will be implemented and slug will be e.g. ‘Tom-Jerry-m323jhgj’. Therefore, this ensures that the slug will be unique.

```
    def __str__(self):
        return f"{self.user.username}-{self.created.strftime('%d-%m-%Y')}"

    def save(self, *args, **kwargs):
        excheck = False
        if self.f_name and self.l_name:
            to_slug = slugify(str(self.f_name) + " " + str(self.l_name))
            excheck = Profile.objects.filter(slug=to_slug).exists()
            while excheck:
                    to_slug = slugify(to_slug + " " +
str(generate_random_character()))
                    excheck = Profile.objects.filter(slug=to_slug).exists()
        else:
            to_slug = str(self.user)
        self.slug = to_slug
        super().save(*args, **kwargs)
```

To get my point across, under the user rizfebriansyah, I inserted a first name ‘tom’ and last name ‘jerry’ and the slug generated was ‘tom-jerry’.



Created a new user, username: user1 and password: testinguser1
Under another user, called “user1”, I inserted the first and last name, ‘tom’ and ‘jerry’ respectively and the slug generated was ‘tom-jerry- bc8b9df3. This ensures that the slug is unique.



//////////////////////////////

Signals in Django allow us to send information to some specific application regarding an event that took place. Often, we use it while creating or modifying one of the models to simply execute some action.
Therefore, I am going to create a new file ‘signals.py’ under userprofiles.

Find apps.py, and I will import the signal files inside the ready method.

```
from django.apps import AppConfig


class UserprofilesConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'userprofiles'

    def ready(self):
        import userprofiles.signals
```

Inside the __init__.py file, I need to indicate the default app config:

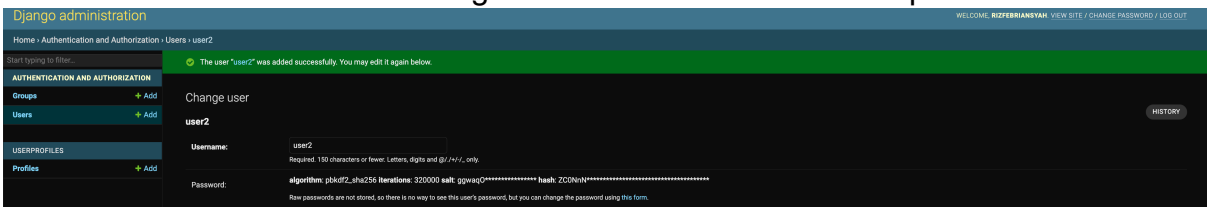
```
default_app_config = 'userprofiles.apps.UserProfilesConfig'
```

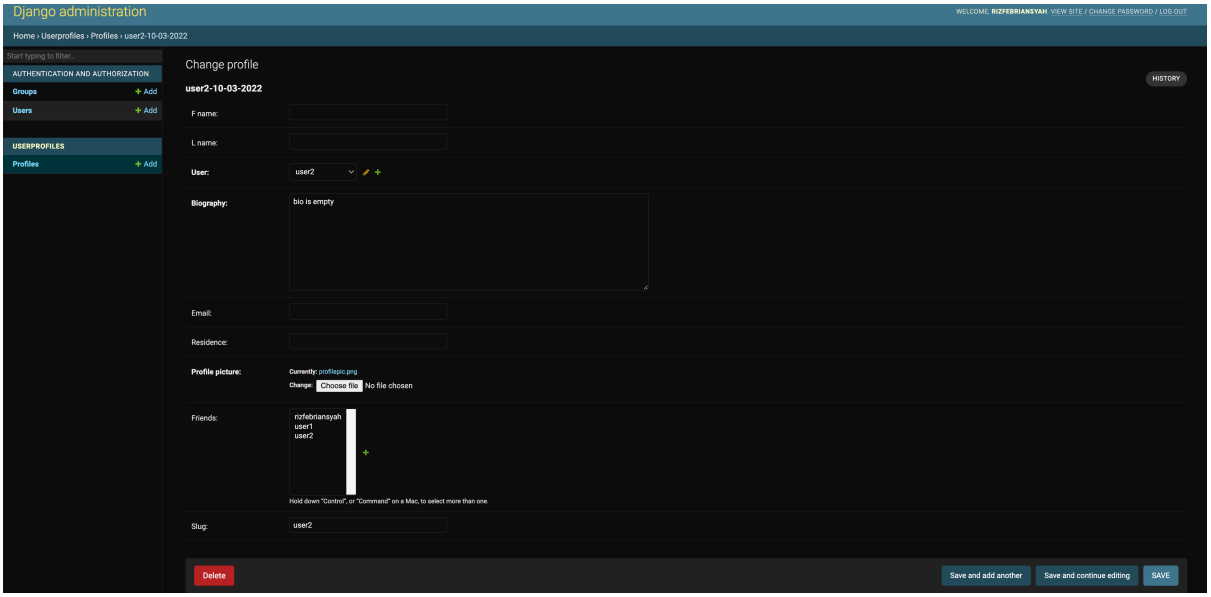
Now I can and start working on the signals.py
The information will be sent at the end of the save method.
I need the user model, and in order to register signal, I will be using receiver decorator.
And finally, I need the profile model.

Under signals.py, add these lines of code:

```
from django.db.models.signals import post_save
from django.contrib.auth.models import User
from django.dispatch import receiver
from .models import Profile

@receiver(post_save, sender=User)
def post_save_create_profile(sender, instance, created, **kwargs):
    if created:
        Profile.objects.create(user=instance)
```

I created a new user, username: user2 with password: testinguser2
We can see from here that Adding a new user will create a new profile as well.





Moving on, we are going to create a Connection model.
We will need 3 important fields.
sender -> who sends the invitation
receiver -> who receives the invi
status -> the status of this connection/relationship (is it sent or accepted)

On models.py, we create a new model:

```
class Connection(models.Model):
    sender = models.ForeignKey(Profile, on_delete=models.CASCADE,
related_name='sender')
    receiver = models.ForeignKey(Profile, on_delete=models.CASCADE,
related_name='receiver')
    status = models.CharField(max_length=8, choices=STATUS_CHOICES)
    updated = models.DateTimeField(auto_now=True)
    created = models.DateTimeField(auto_now_add=True)
```

Afterwards, we add a string representation method.

```
    def __str__(self):
        return f"{self.sender}-{self.receiver}-{self.status}"
```

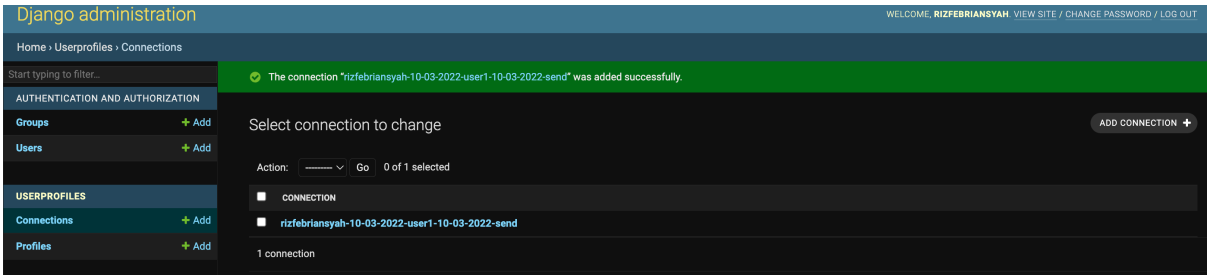
We can now head over to the admin.py file add register the new Connection model:

```
from django.contrib import admin
from .models import Profile, Connection
# Register your models here.

admin.site.register(Profile)
admin.site.register(Connection)
```

Now, we need to do migrations and run the server.
Go over to your terminal and type:
python manage.py makemigrations
python manage.py migrate
python manage.py runserver

```
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro src % python manage.py makemigrations
Migrations for 'userprofiles':
  userprofiles/migrations/0002_connection.py
    - Create model Connection
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro src % python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions, userprofiles
Running migrations:
  Applying userprofiles.0002_connection... OK
(socialnetwork) rizfebriansyah@Febriansyahs-MacBook-Pro src % python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
March 10, 2022 - 09:18:50
Django version 4.0.3, using settings 'friztalk.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Now, we can go to the website and add new connections

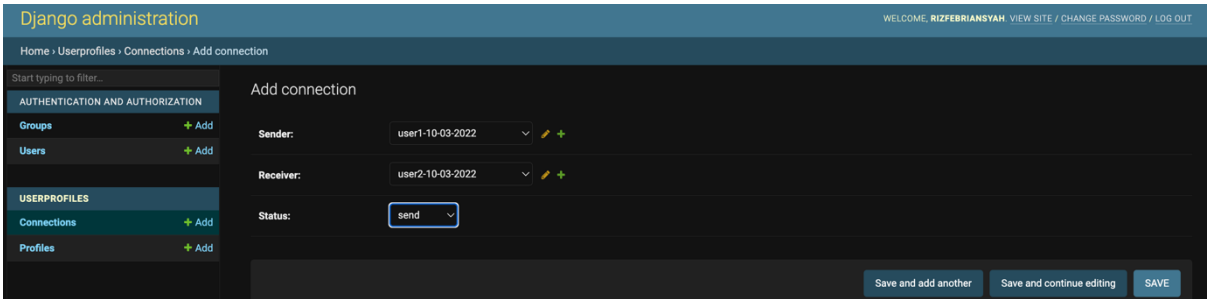


We can now confirm that we are able to send and receive new connections between users. However, once a connection has been accepted, it does not show in the profile as one of the friends. We are now going to fix that.

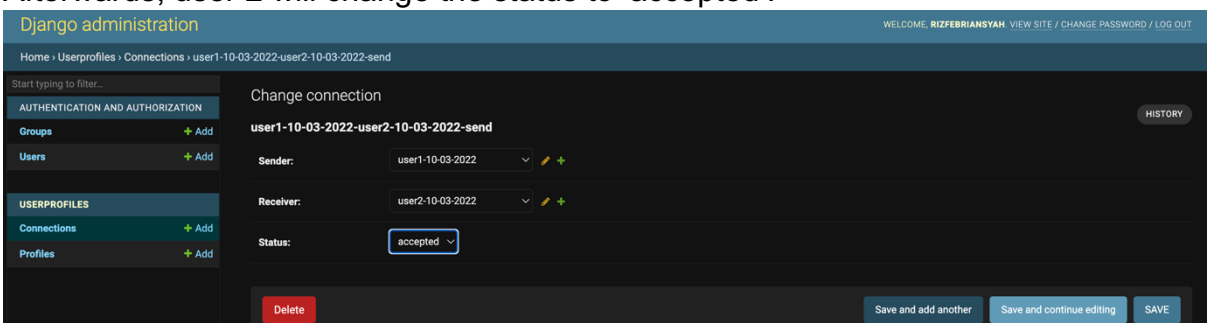
We need to add this on signals.py:

```python
from django.db.models.signals import post_save
from django.contrib.auth.models import User
from django.dispatch import receiver
from .models import Profile, Connection

@receiver(post_save, sender=User)
def post_save_create_profile(sender, instance, created, **kwargs):
    if created:
        Profile.objects.create(user=instance)

@receiver(post_save, sender=Connection)
def post_save_add_to_friends(sender, instance, created, **kwargs):
    sender_ = instance.sender
    receiver_ = instance.receiver
    if instance.status == 'accepted':
        sender_.friends.add(receiver_.user)
        receiver_.friends.add(sender_.user)
        sender_.save()
        receiver_.save()
```

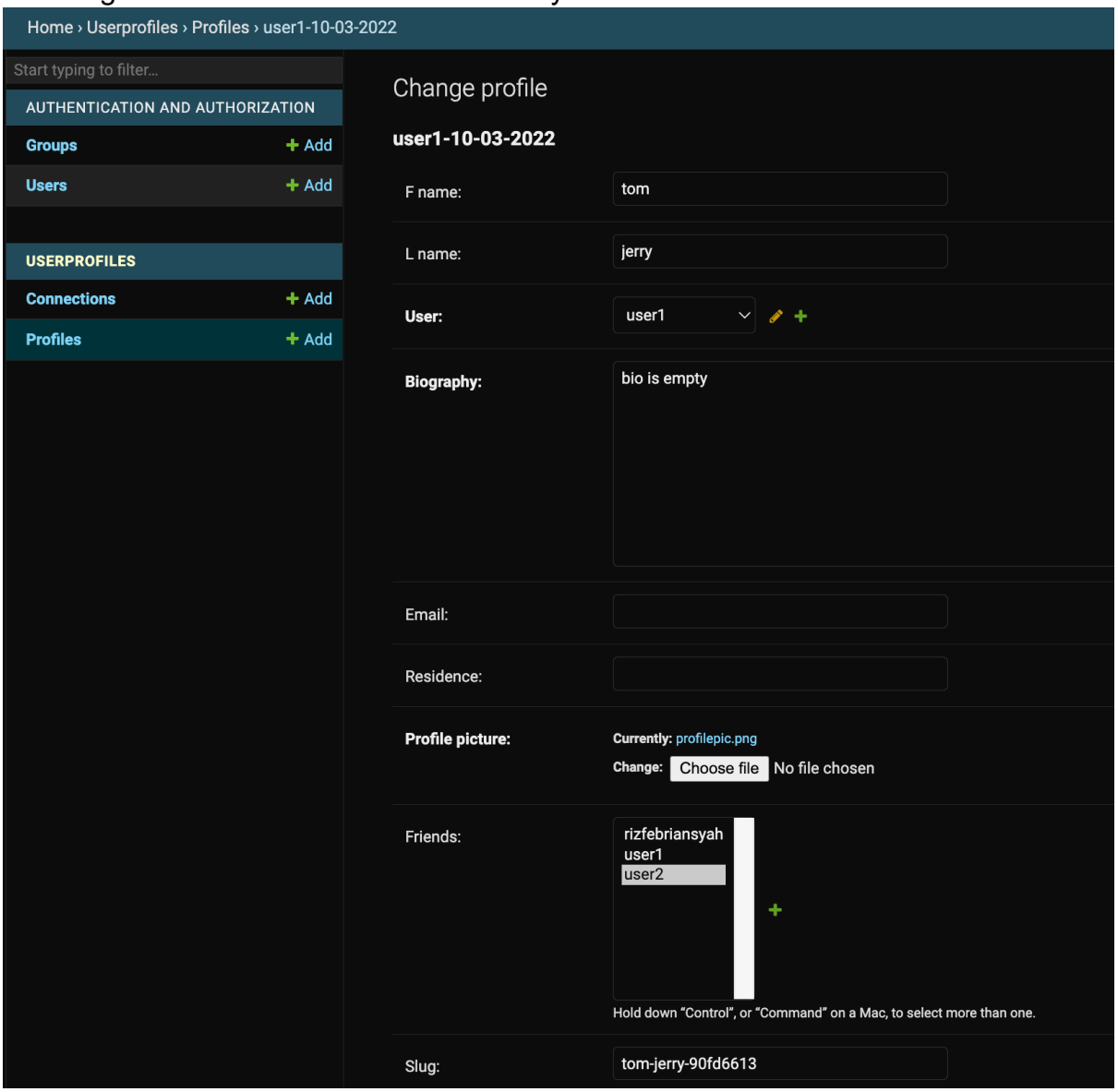
As we see here, we will now add a new connection. The sender will be user1 and the receiver will be user2. It is a ‘send’ status.



Afterwards, user 2 will change the status to ‘accepted’.



Now, going to user1 profile, we can see under Friends that user2 is highlighted, meaning now user2 and user1 are officially friends.



////////////////////////////////////

Moving on, we are going to start designing our homepage.
We will be working on the views and the template.

We are going to create a new file, views.py under friztalk.

Under views.py, we need to add:

```
from django.http import HttpResponse

def home_page(request):
    return HttpResponse('Hello World. Welcome to Friztalk')
```

We will now a new PATH on urls.py:

```
from django.contrib import admin
from django.urls import path
from django.conf import settings
from django.conf.urls.static import static
from .views import home_page

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', home_page, name='home-page'),
]
```

And now we can run the browser, and enter http://127.0.0.1:8000/
and this is the output:

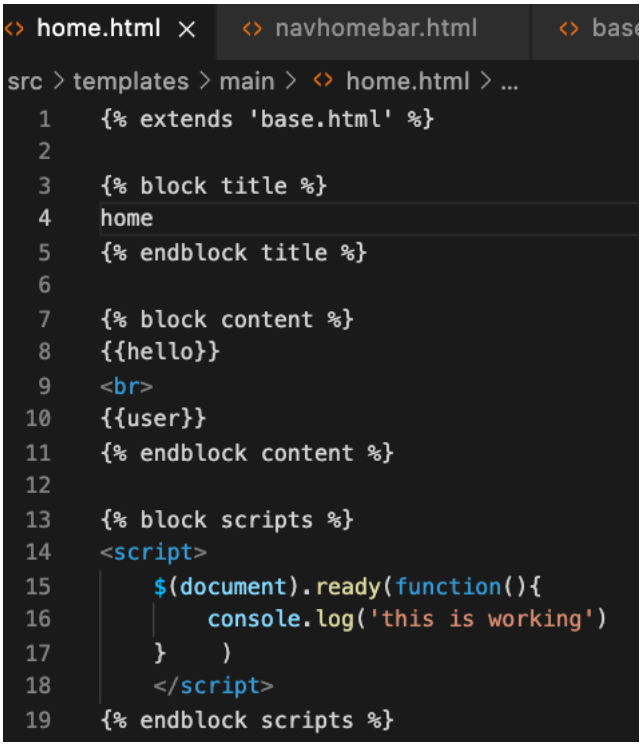
Hello World. Welcome to Friztalk

We will now define the key on views.py:

```
from django.http import HttpResponse
from django.shortcuts import render

def home_page(request):
    user = request.user
    hello = 'Hello World'

    context = {
        'user': user,
        'hello' : hello,
    }
    return render(request, 'main/home.html', context)
    # return HttpResponse('Hello World. Welcome to Friztalk')
```

On the ‘templates’ folder, we create a new file called “base.html”.
On the same ‘templates’ folder, we create a folder called ‘main’.
Under the ‘main’ folder, we will create “home.html” and “navhomebar.html”

We add this code to home.html:



Add this code to homenavbar.html:

home.html | navhomebar.html | base.html | views.py 2

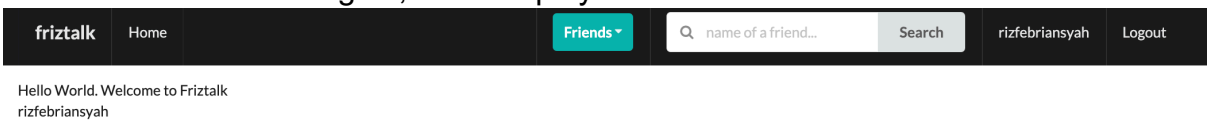
src > templates > main > navhomebar.html > div.ui.inverted.menu > div.ui.container > div.right.menu > div

```
<div class="ui inverted menu">
    <div class="ui container">
      <a href="" class="item">
          <h3>friztalk</h3>
      </a>
      {% if request.user.is_authenticated %}
      <a href="" class="item">
          Home
      </a>
      {% endif %}

      <div class="right menu">
          {% if request.user.is_authenticated %}
          <div class="item">
              <div class="ui floating dropdown icon button teal"> Friends
                  <i class="dropdown icon"></i>
                  <div class="menu">
                      <a href="" class="item">
                          All Profiles
                      </a>
                      <a href="" class="item">
                          Send Invites
                      </a>
                      <a href="" class="item">
                          Received Invites
                      </a>
                  </div>
              </div>
          </div>
               <div class="item">
                <form action="" method="GET">
                  <div class="ui action left icon input">
                  <input class="prompt" type="text" name='q' placeholder="name of a friend...">
                  <i class="search icon"></i>
                  <button type='submit' class="ui button">Search</button>
                  </div>
              </form>
              </div>

              <a href="" class="item">
                  {{ request.user }}
              </a>

              <a href="" class="ui item">
                  Logout
              </a>
          {% else %}
          <a href="" class="ui item">
                  Login
              </a>
          {% endif %}
      </div>
    </div>
  </div>
```

Add this code to base.html:

```
<!-- jquery -->
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<!--Custom css ADD HERE-->

<!-- semantic UI -->
<link rel="stylesheet" type='text/css' href="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.2.14/semantic.min.css">
<script src="https://cdnjs.cloudflare.com/ajax/libs/semantic-ui/2.4.1/semantic.min.js"></script>


<title>friztalk - {% block title %}{% endblock title %}</title>
</head>
<body>
  {% include 'main/navhomebar.html' %}
<div class="ui container">
  {% block content %}
  {% endblock content %}
</div>

{% block scripts %}
{% endblock scripts %}

<!--Custom js ADD HERE-->

</body>
</html>
```

If we load the website again, it will display this:



Hello World. Welcome to Friztalk
rizfebriansyah

We will now create two files under static_project folder, main.js and style.css

Under base.html, add these linse of code for css and js to work.

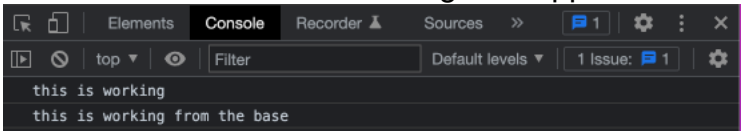
```
<!--Custom css ADD HERE-->
<link rel="stylesheet" type="text/css" href={% static "style.css" %}
```

```
<!--Custom js ADD HERE-->
<script type="text/javascript" src={% static 'main.js' %}></script>
```

Now, we add this code to main.js file to check if it is working, by inspecting the console:

```
$(document).ready(function(){
    console.log('this is working from the base')
})
```

And this shows that it is working as it appears on the console.



Now we are going to customize and style our homepage, by editing the style.css file. We will create a homepage with a grey background and a maroon navigation bar.

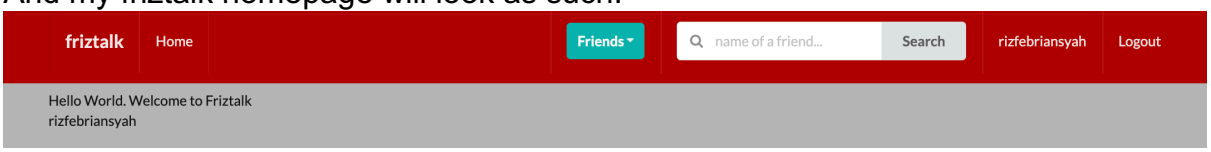
```
body {
    background: #b6b6b6 !important;
}

.nav {
    border-radius: 5px !important;
    padding: 10px 0 10px 0 !important;
    background-color: #b10000 !important;
}

.mb-8 {
    margin-bottom: 7px !important;
}
```

We add this line of code to navhomebar.html:

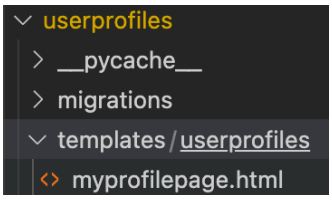
```
<div class="ui inverted menu nav mb-8">
```

```
<div class="ui inverted menu nav mb-8">
    <div class="ui container">
      <a href="" class="item">
          <h3>friztalk</h3>
      </a>
      {% if request.user.is_authenticated %}
      <a href="" class="item">
          Home
      </a>
      {% endif %}
```

And my friztalk homepage will look as such:

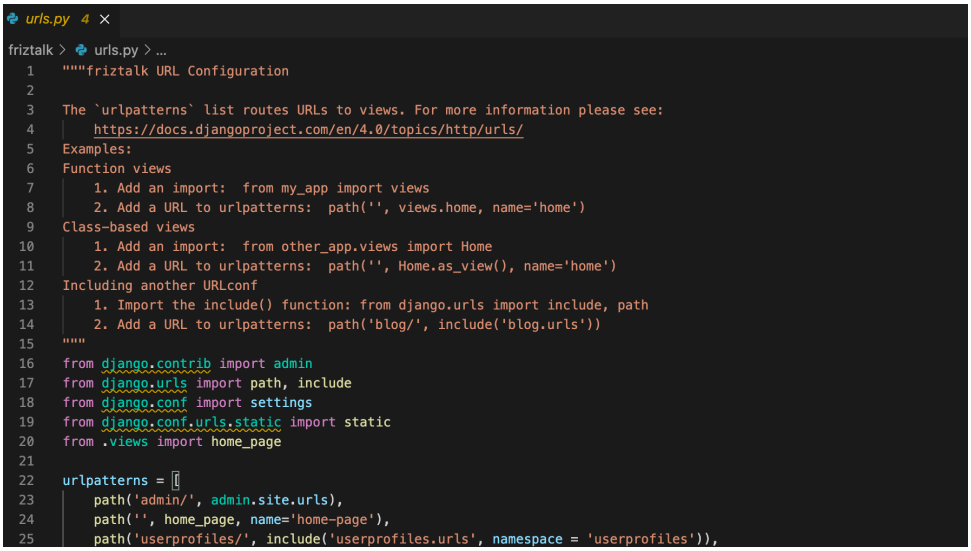


Now we create a new folder, template, under userprofiles.
Under template, create a new folder called userprofiles.
Inside userprofiles, create a file myprofilepage.html. This myprofilepage.html will be used to utilize and edit the layout and overall appearance of the users' profile page.

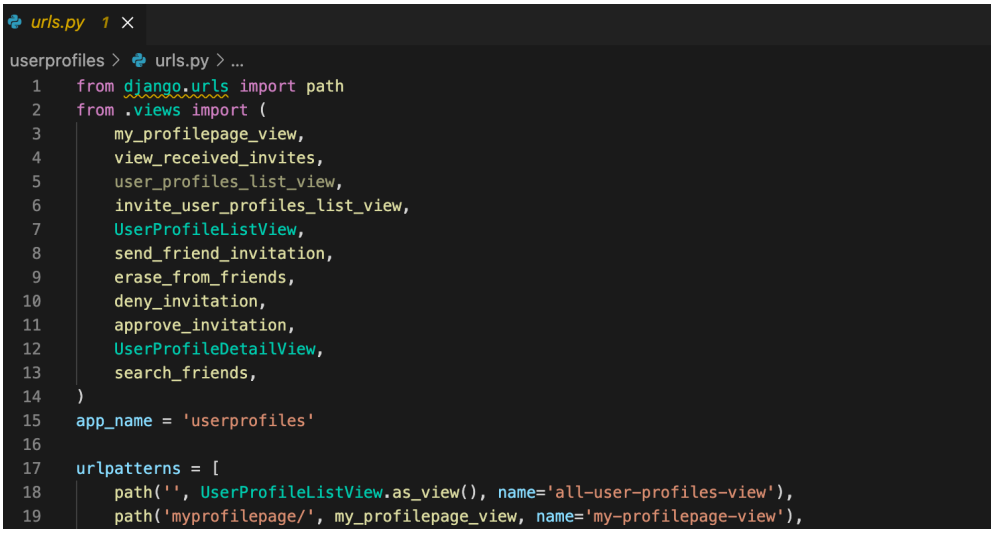


We need to add this urls on friztalk > urls.py:

```
path('userprofiles/', include('userprofiles.urls', namespace = 'userprofiles')),
```

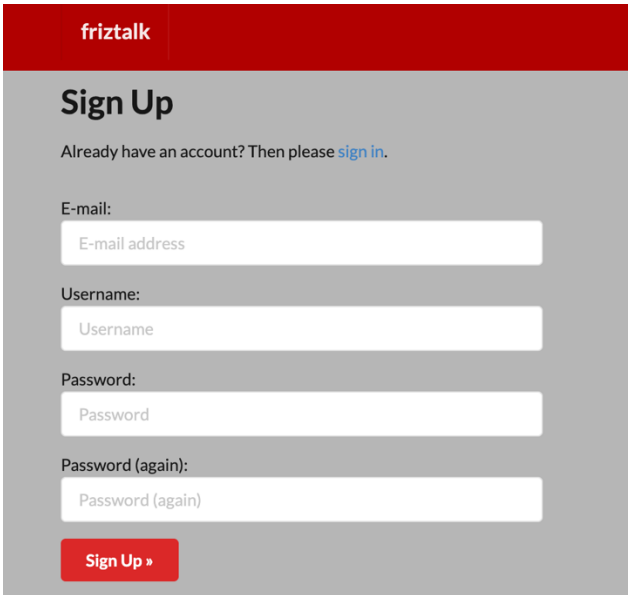


This is inherited from the userprofiles > urls.py. This is why you need to import include and the line writes "include('userprofiles.urls', namespace = 'userprofiles')). Meaning it will be inherited from the urls.py under userprofiles. The namespace will be the app_name which is userprofiles.



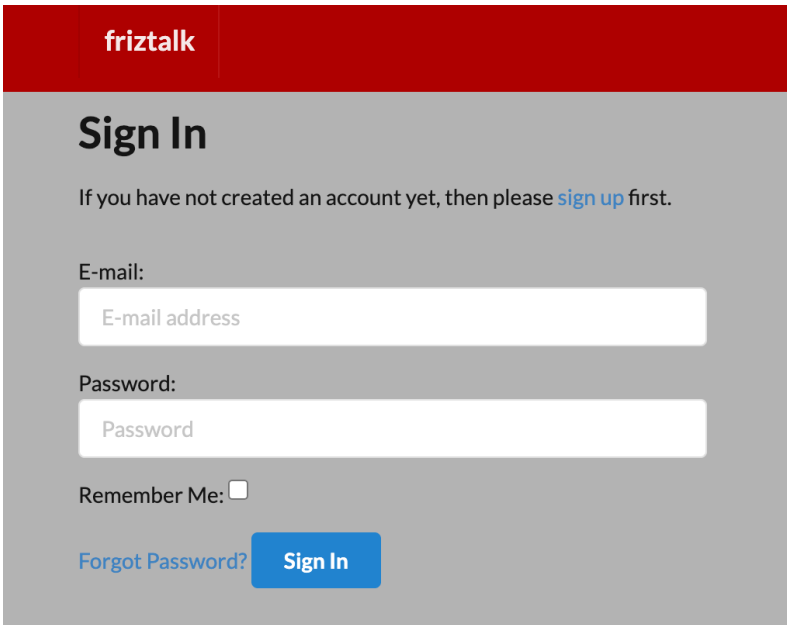
Users can create accounts:

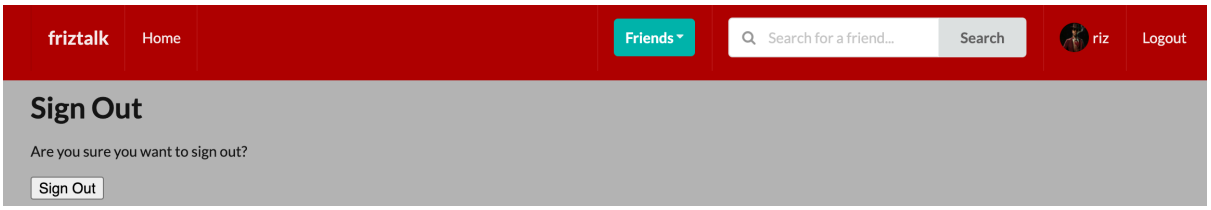
Users are able to create a new account. They need to provide an e-mail address, username, password to sign up.



Users can log in and log out:

Once a user has created their account, they are able to login via their e-mail and password. Once logged in, they have the option to log out as well.



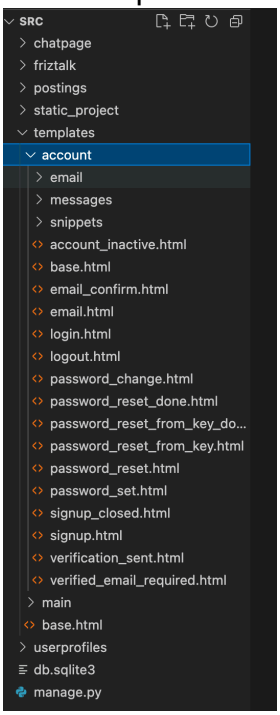


For the authentication and registration, I will be using Django allauth.
I have followed the installation step by step from this document itself:
https://django-allauth.readthedocs.io/en/latest/installation.html

Afterwards, download the zip file of Django-allauth from github:
https://github.com/pennersr/django-allauth

Unzip the downloaded file, open it.
Go to allauth > templates and you will see the accounts folder.

Copy this account folder and paste in on your projects folder under socialnetwork > src > templates.



Then I will edit the html to inherit my own base.html. Make modifications so every page will run smoothly.

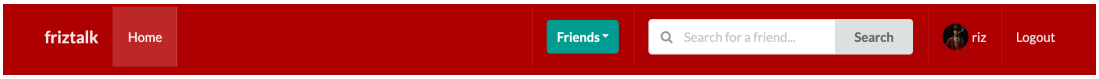
Add this on friztalk > settings.py:

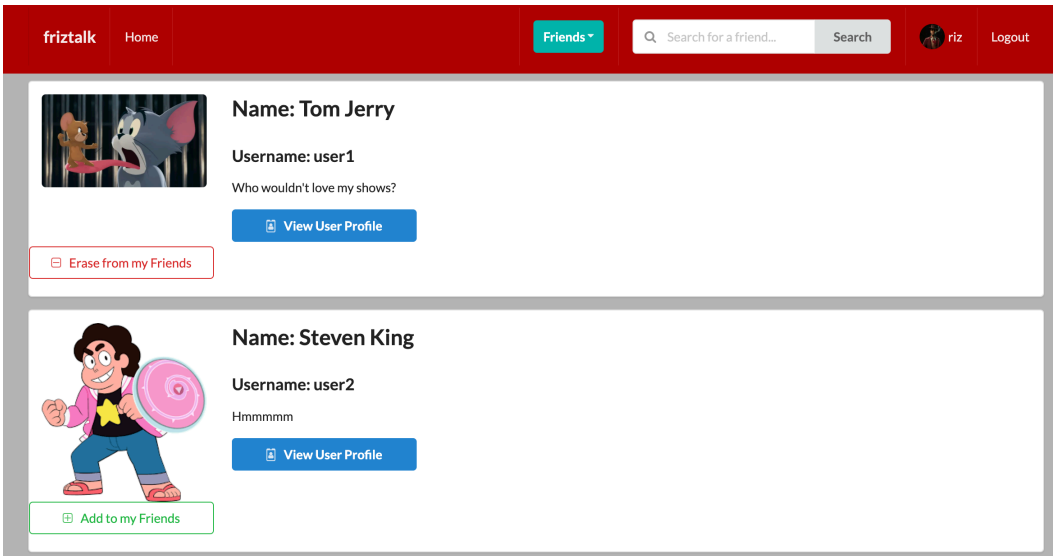
```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.sites',

    'postings',
    'userprofiles',
    'chatpage',

    # django all-auth apps
    'allauth',
    'allauth.account',
    'allauth.socialaccount',
]
```

```
SITE_ID = 1

# LOGIN_URL = '/admin/'
LOGIN_REDIRECT_URL = '/postings'

ACCOUNT_AUTHENTICATION_METHOD = 'email'
ACCOUNT_EMAIL_REQUIRED = True
ACCOUNT_EMAIL_UNIQUE = True
# ACCOUNT_EMAIL_VERIFICATION = 'mandatory'

if DEBUG:
    EMAIL_BACKEND = 'django.core.mail.backends.dummy.EmailBackend'

# EMAIL_BACKEND= 'django.core.mail.backends.smtp.EmailBackend'
```

User can search for others:

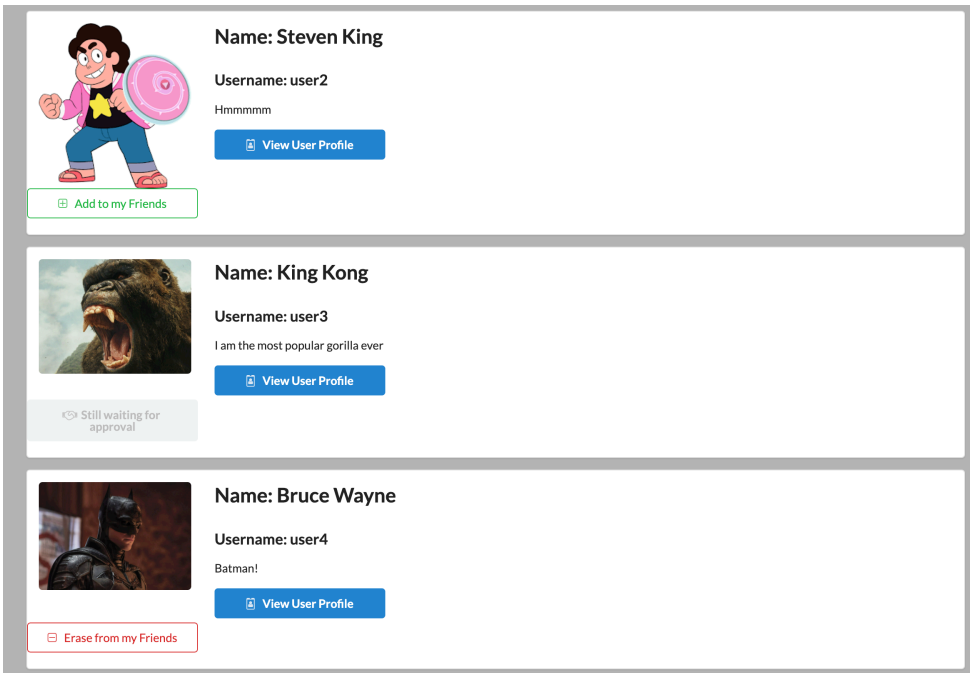
Once logged in, user can go to the search bar, type a name, and a list of other users will be displayed.





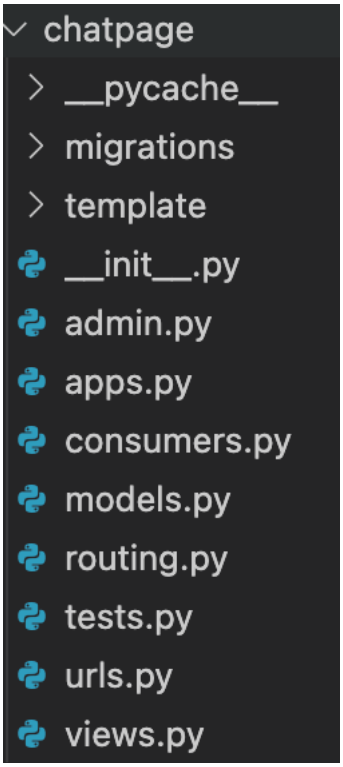
Users can add other users as friends:

User are able to add others as friends, or even remove them from your friend list. It will show three status, first one being “Add to My Friends”, meaning that the user would want to add him/her as friend. Second being “Still waiting for approval”, meaning the other user has yet to accept/deny the user’s friend request. Lastly will be “Erase from my Friends”, where the user would like to remove him/her from his list of friends.



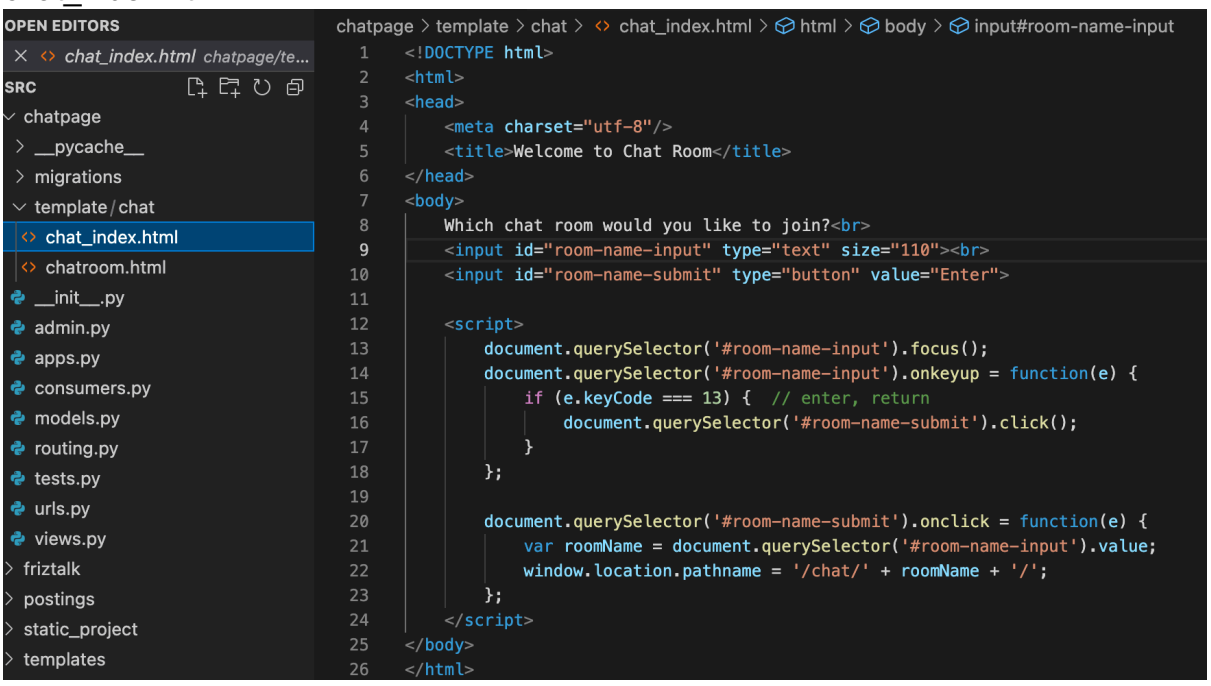
Users can chat in real-time with friends:

I have created a new app called ‘chatpage’. This is to implement the chat function. However, this time round I wasn’t able to implement it fully.

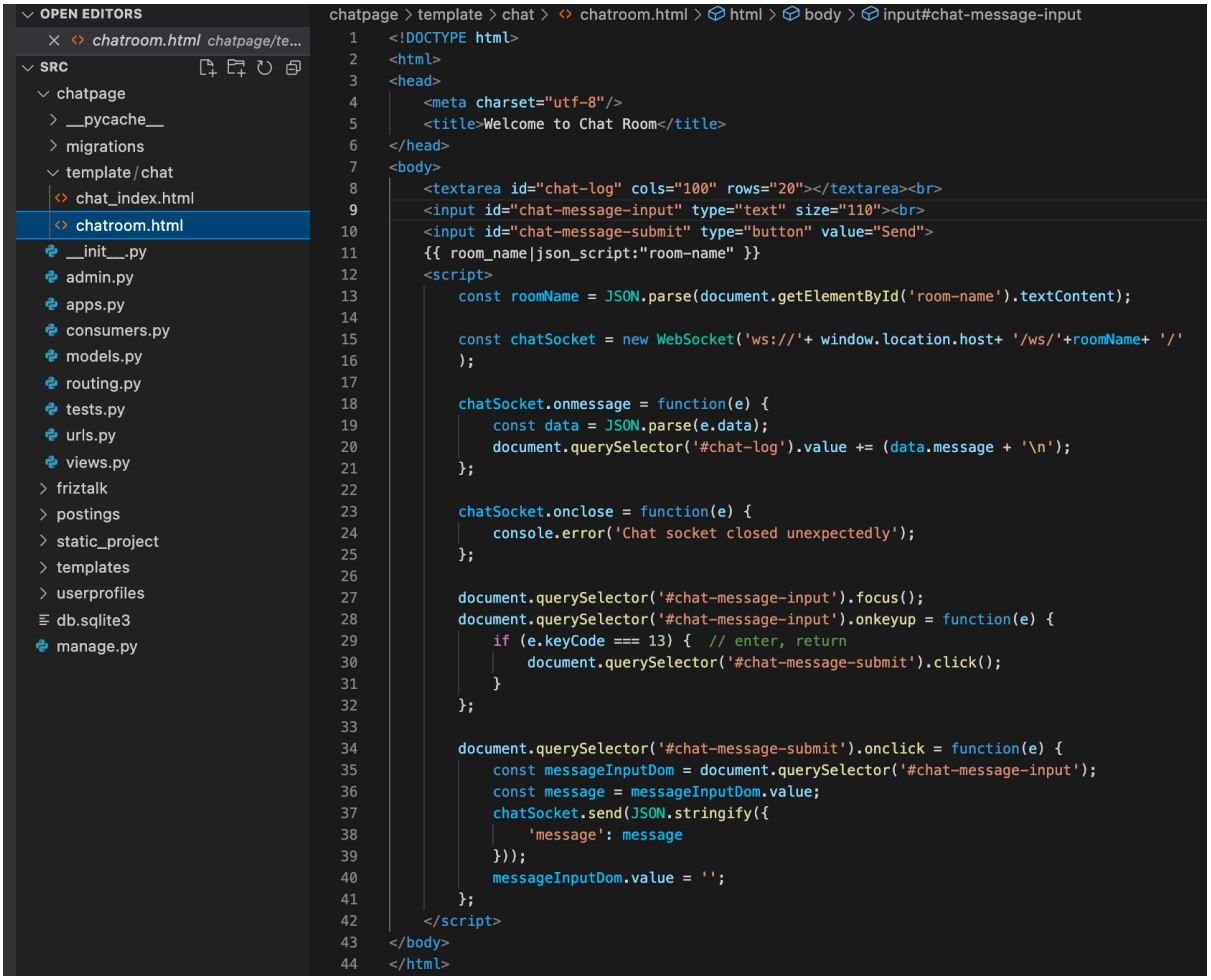


I have created a templates folder, which stores chat_index.html and chatroom.html

chat_index.html:

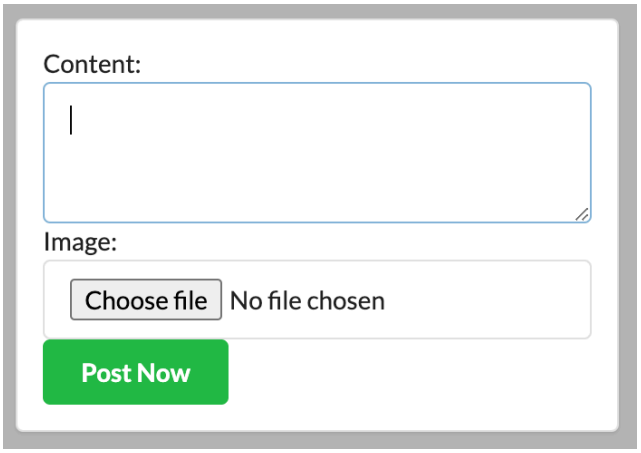


chatroom.html:

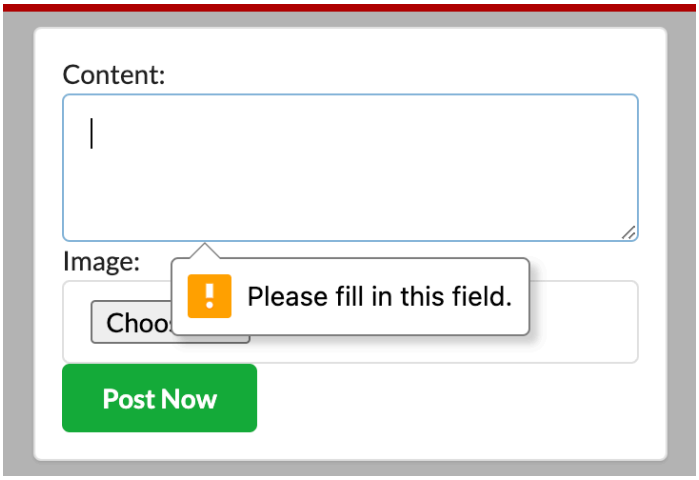


Users can add status updates to their homepage:

Users have the freedom to update their status. They are able to do so in the homepage. On the right-hand corner, they are able to see this box (shown below). They can type whatever they like under content, upload an image if they would love to, and once they are satisfied, they can click ‘Post Now’. Their new status will now appear in the homepage.

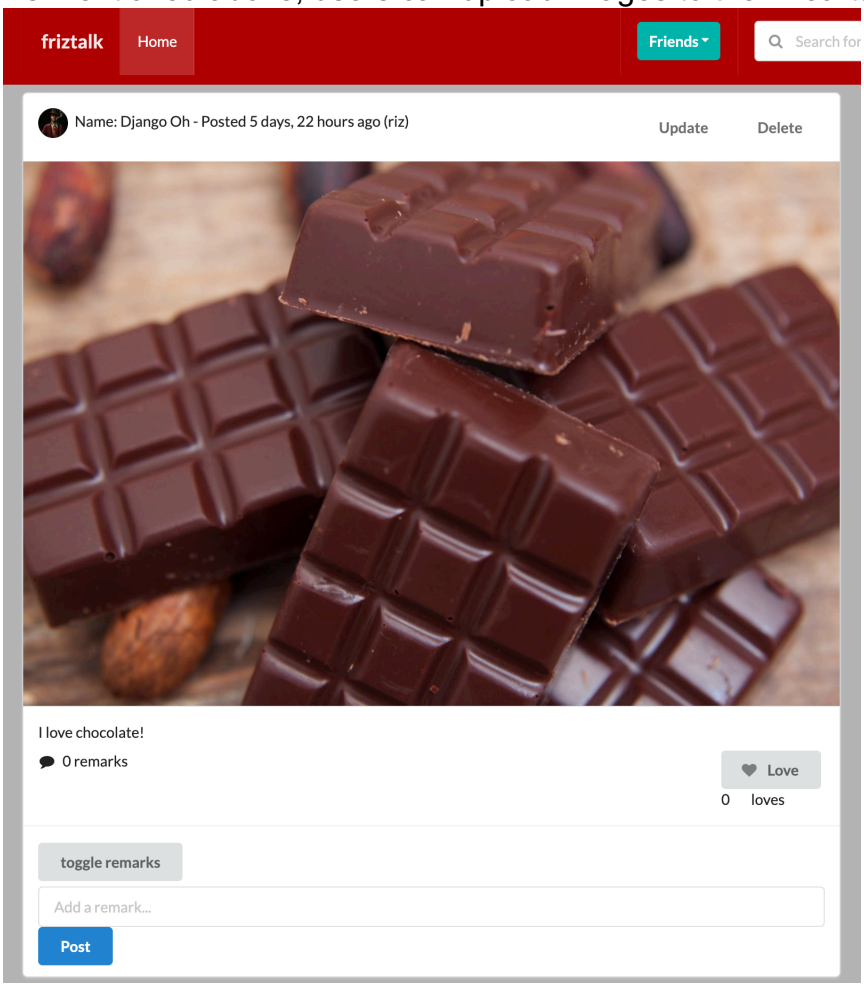


Users also have to type in something before clicking the ‘Post Now’. If not, a message will appear, prompting them to fill in the box.



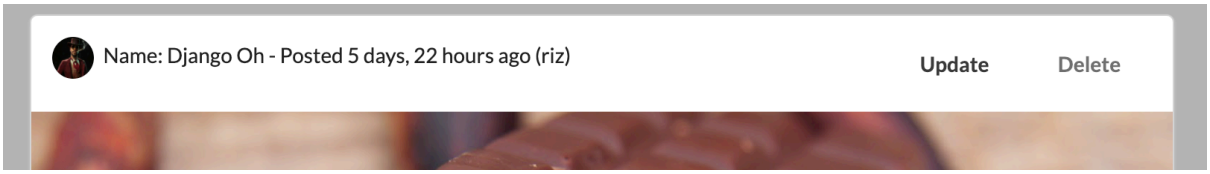
Users can add media (such as images to their account and these are accessible via their home page:

As mentioned above, users can upload images to their hearts' content.

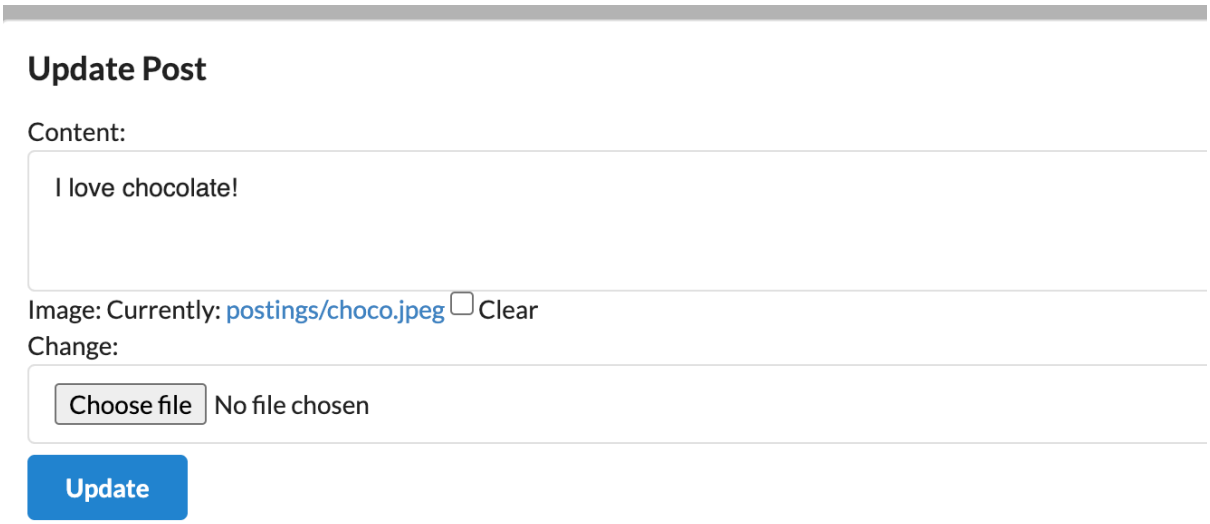


An appropriate method for storing and displaying media files is given:

The original user who posts a status also have the choice to update or delete their post.

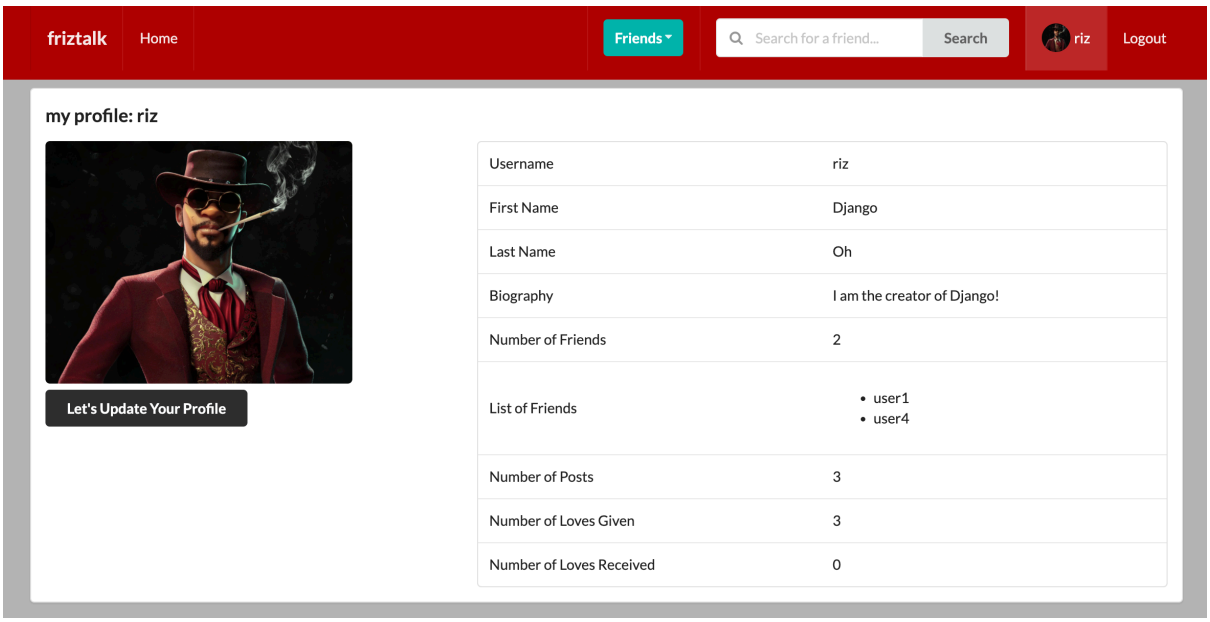


Clicking update, the user can edit their post and change the image they have uploaded.

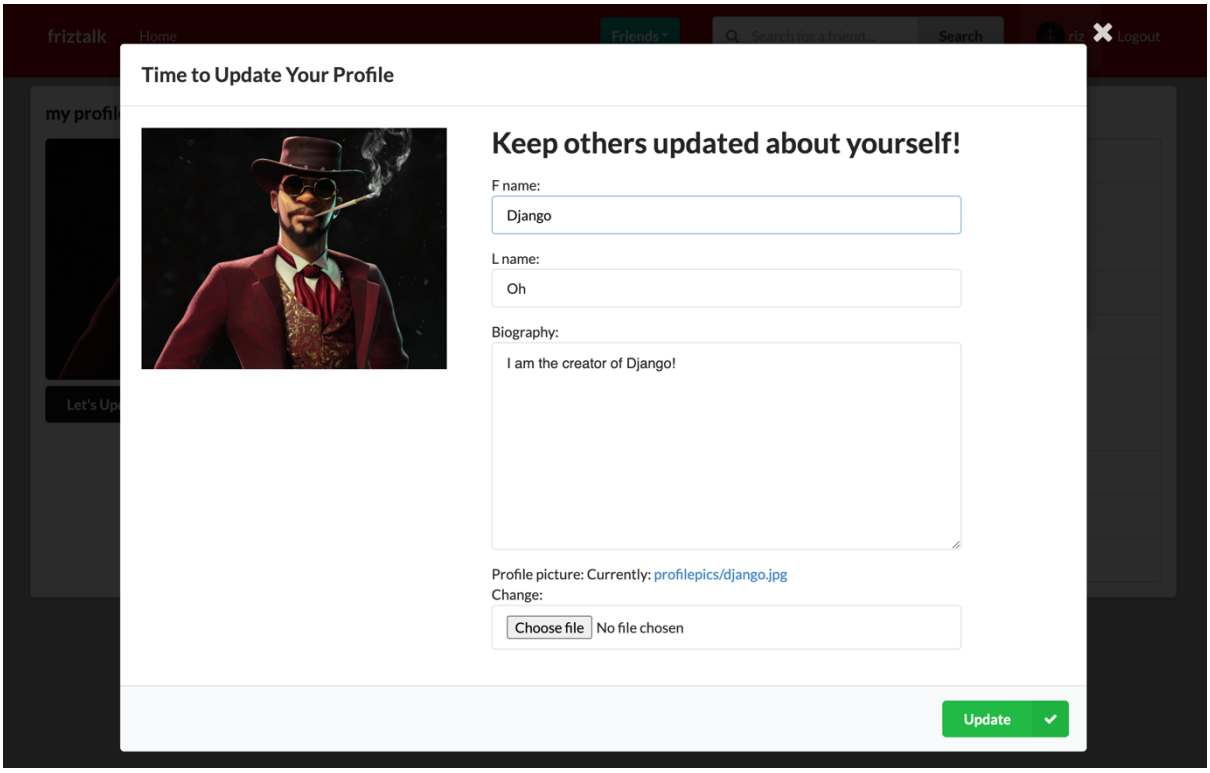


Meanwhile, clicking Delete, the user will be prompted by a message whether they are sure to delete their post.

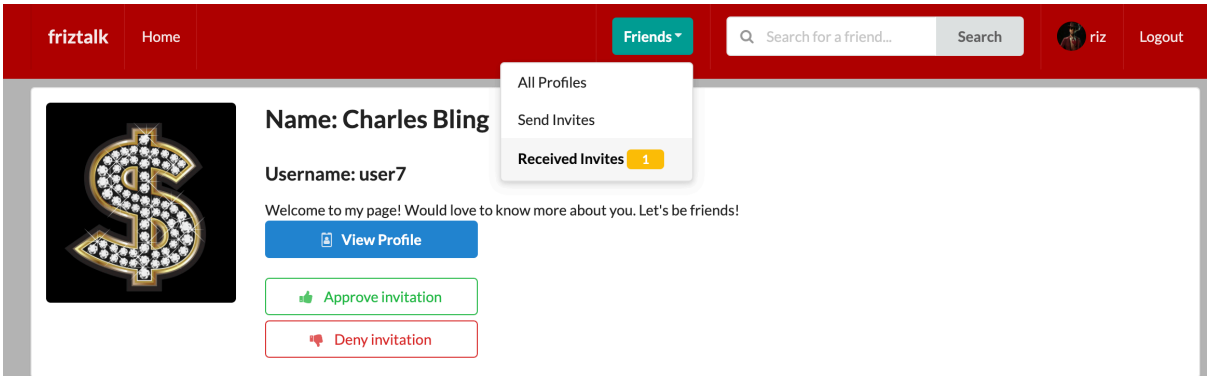
Users are also able to view their own profile information by clicking on their profile picture on the navigation bar, just beside the search bar.



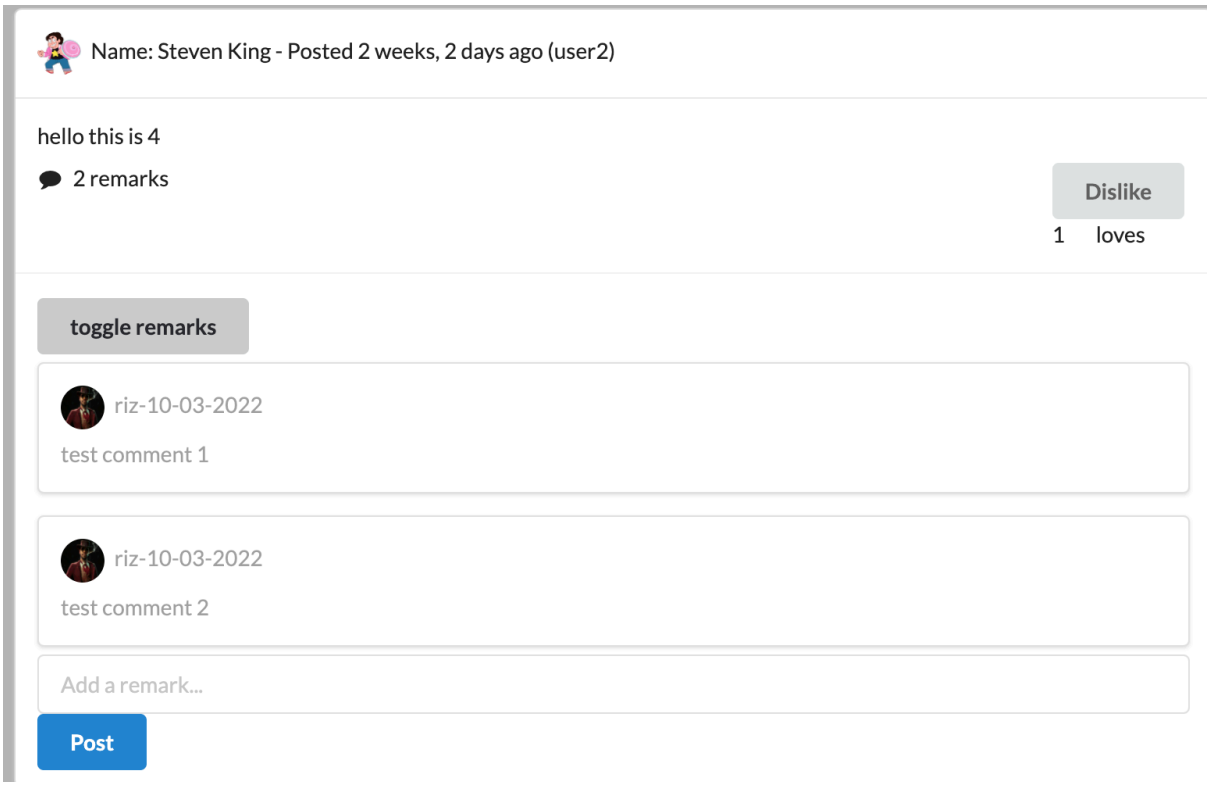
Users can also update their profile in real time. Changing their first name, last name, biography, and profile picture is so easy and seamless now. Once done, do click the Update button.



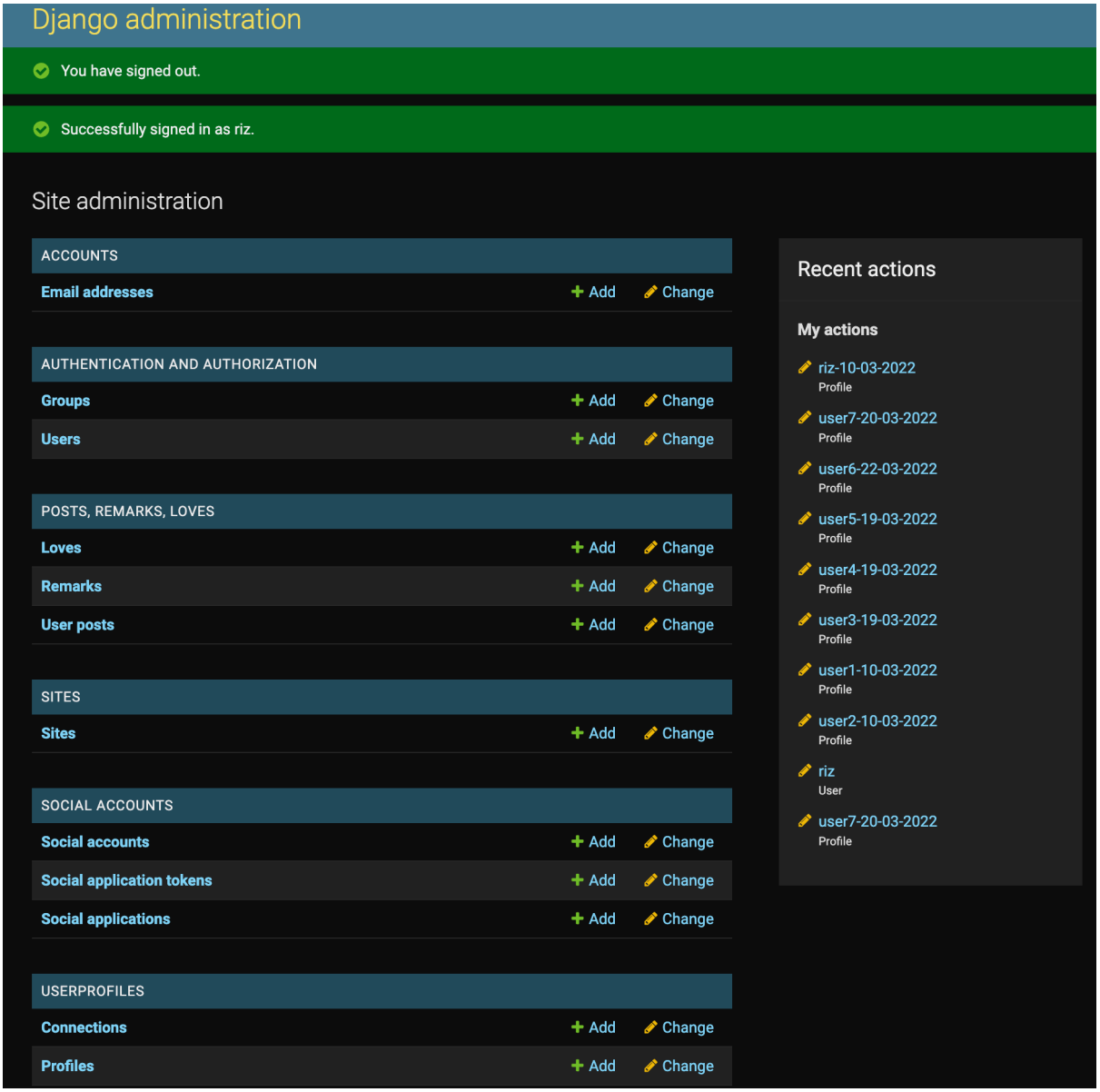
Also, on the navigation bar, there's a dropdown menu. Users are able to see who sent them a friend request and the number of friend requests they have. Clicking the 'Received Invites' will display the list of friend invitations and the user logged in is able to view their profile first, before deciding to accept or reject the friend invitation.



In addition to that, users can see status updates from other users on their homepage, as long as they are friends with each other. Users are able to Love or Dislike the status, and I have implemented a counter function. Clicking Love will update the 0 love to 1 love. Other than that, users are also able to add a remark on their friends' posts too.



Running http://127.0.0.1:8000/admin/ on the browser will lead us to the Django administration. This shows the relationships between the accounts. They are all interconnected to one another.



under src, I have also created a folder static_project, which consists of main.js and style.css. The style.css file is responsible for the overall layout and appearance of my social network app.

correct use of models and migrations:

I have used and implemented the correct use of models and migrations. This can be seen here below:

postings > models.py:

models.py 3

postings > models.py > ...

```python
from django.db import models
from userprofiles.models import Profile
from django.core.validators import FileExtensionValidator

# Create your models here.
class UserPost(models.Model):
    content = models.TextField()
    image = models.ImageField(upload_to='postings', validators=[FileExtensionValidator(['jpg', 'jpeg', 'png'])], blank=True)
    loved = models.ManyToManyField(Profile, blank=True, related_name='loves')
    updated = models.DateTimeField(auto_now=True)
    created = models.DateTimeField(auto_now_add=True)
    author = models.ForeignKey(Profile, on_delete=models.CASCADE, related_name='posts')

    def __str__(self):
        return str(self.content[:20])

    def num_loves(self):
        return self.loved.all().count()

    def num_remarks(self):
        return self.remark_set.all().count()

    class Meta:
        ordering = ('-created',)

class Remark(models.Model):
    user = models.ForeignKey(Profile, on_delete=models.CASCADE)
    post = models.ForeignKey(UserPost, on_delete=models.CASCADE)
    body = models.TextField(max_length=400)
    updated = models.DateTimeField(auto_now=True)
    created = models.DateTimeField(auto_now_add=True)

LOVE_CHOICES = (
    ('Love', 'Love'),
    ('Dislike', 'Dislike'),
)

class Love(models.Model):
    user = models.ForeignKey(Profile, on_delete=models.CASCADE)
    post = models.ForeignKey(UserPost, on_delete=models.CASCADE)
    value = models.CharField(choices=LOVE_CHOICES, max_length=9)
    updated = models.DateTimeField(auto_now=True)
    created = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"{self.user}-{self.post}-{self.value}"

```

userprofiles > models.py:

models.py 6

userprofiles > models.py > ProfileManager > attain_all_profiles_to_invite

```python
from django.db import models
from django.contrib.auth.models import User
from .utils import generate_random_character
from django.db.models import Q
from django.template.defaultfilters import slugify
from django.shortcuts import reverse

# Create your models here.
class ProfileManager(models.Manager):

    def attain_all_profiles_to_invite(self, sender):
        profiles = Profile.objects.all().exclude(user=sender)
        profile = Profile.objects.get(user=sender)
        qs = Connection.objects.filter(Q(sender=profile) | Q(receiver=profile))
        print(qs)
        print("############################")

        accepted = set([])
        for rel in qs:
            if rel.status == 'accepted':
                accepted.add(rel.receiver)
                accepted.add(rel.sender)
        print(accepted)
        print("############################")

        available = [profile for profile in profiles if profile not in accepted]
        print(available)
        print("############################")
        return available


    def attain_all_profiles(self, me):
        profiles = Profile.objects.all().exclude(user=me)
        return profiles

class Profile(models.Model):
    f_name = models.CharField(max_length=250, blank=True)
    l_name = models.CharField(max_length=250, blank=True)
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    biography = models.TextField(default="bio is empty", max_length=350)
    email = models.EmailField(max_length=200, blank=True)
    residence = models.CharField(max_length=200, blank=True)
    profile_picture = models.ImageField(default='profilepic.png', upload_to='profilepics/')
    friends = models.ManyToManyField(User, blank=True, related_name='friends')
    slug = models.SlugField(unique=True, blank=True)
    updated = models.DateTimeField(auto_now=True)
    created = models.DateTimeField(auto_now_add=True)

    objects = ProfileManager()

    def __str__(self):
        return f"{self.user.username}-{self.created.strftime('%d-%m-%Y')}"

    def attain_absolute_url(self):
        return reverse("userprofiles:user-profile-detail-view", kwargs={"slug": self.slug})

    def attain_friends(self):
        return self.friends.all()

    def attain_friends_number(self):
        return self.friends.all().count()
```

correct use of form, validators, and serialisation:

I have used and implemented the correct use of form, validators and serialisation. This can be seen here below:

userprofiles > forms.py

forms.py 2

userprofiles > forms.py > ProfilePageModelForm > Meta

```python
from django import forms
from .models import Profile

class ProfilePageModelForm(forms.ModelForm):
    class Meta:
        model = Profile
        fields = ('f_name', 'l_name', 'biography', 'profile_picture')
```

postings > forms.py

forms.py 2

postings > forms.py > RemarkModelForm > Meta

```python
from django import forms
from .models import UserPost, Remark

class UserPostModelForm(forms.ModelForm):
    content = forms.CharField(widget=forms.Textarea(attrs={'rows':3}))
    class Meta:
        model = UserPost
        fields = ('content', 'image')

class RemarkModelForm(forms.ModelForm):
    body = forms.CharField(label='',
                           widget=forms.TextInput(attrs={'placeholder': 'Add a remark...'}))
    class Meta:
        model = Remark
        fields = ('body',)
```

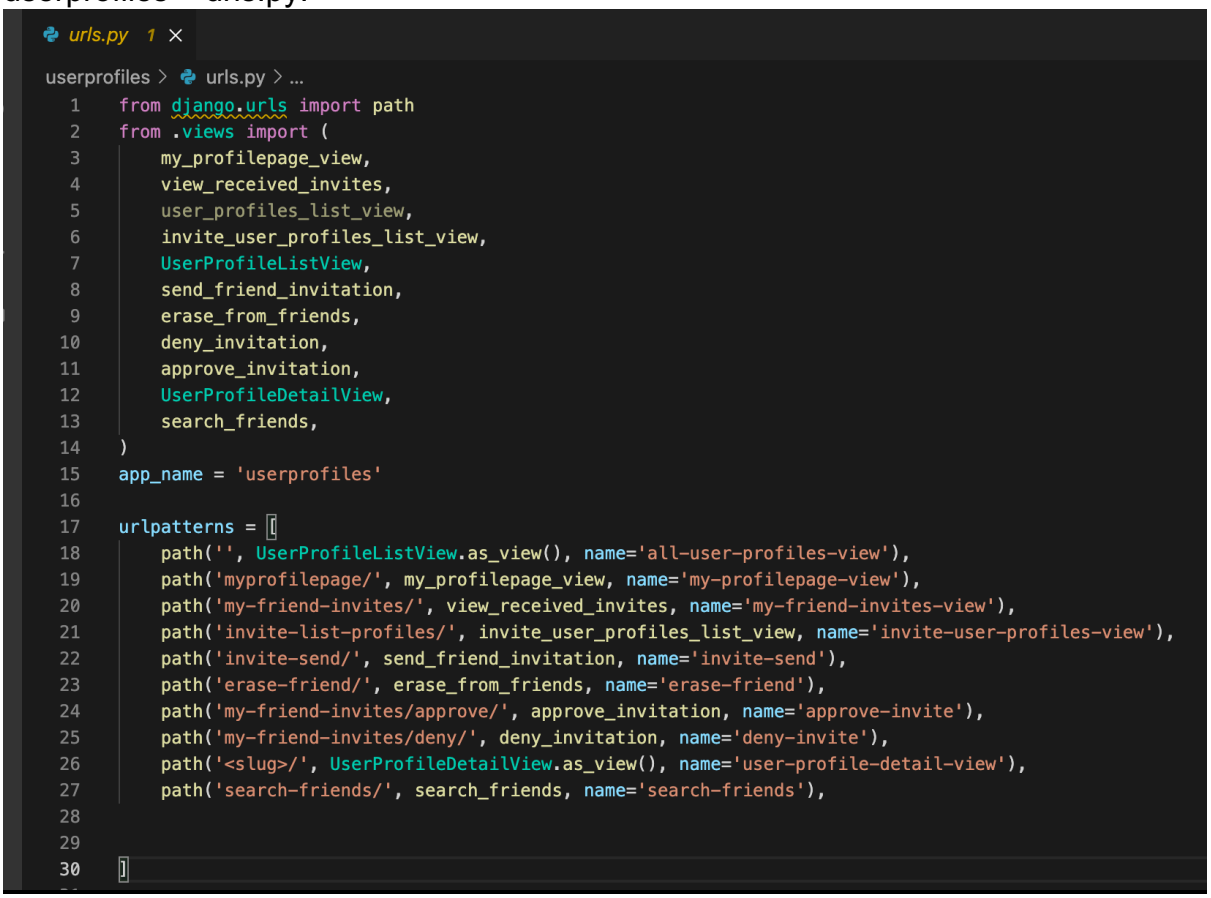
correct use of URL routing:

I have used and implemented the correct use of URL routing. This can be seen here below:

postings > urls.py:

```python
from django.urls import path
from .views import post_remark_create_and_list_view, love_dislike_post, UserPostDeleteView, UserPostUpdateView

app_name = 'postings'

urlpatterns = [
    path('', post_remark_create_and_list_view, name='main-post-view'),
    path('loved/', love_dislike_post, name='love-post-view'),
    path('<pk>/delete/', UserPostDeleteView.as_view(), name='post-delete'),
    path('<pk>/update/', UserPostUpdateView.as_view(), name='post-update'),
]
```

userprofiles > urls.py:



```python
from django.urls import path
from .views import (
    my_profilepage_view,
    view_received_invites,
    user_profiles_list_view,
    invite_user_profiles_list_view,
    UserProfileListView,
    send_friend_invitation,
    erase_from_friends,
    deny_invitation,
    approve_invitation,
    UserProfileDetailView,
    search_friends,
)
app_name = 'userprofiles'

urlpatterns = [
    path('', UserProfileListView.as_view(), name='all-user-profiles-view'),
    path('myprofilepage/', my_profilepage_view, name='my-profilepage-view'),
    path('my-friend-invites/', view_received_invites, name='my-friend-invites-view'),
    path('invite-list-profiles/', invite_user_profiles_list_view, name='invite-user-profiles-view'),
    path('invite-send/', send_friend_invitation, name='invite-send'),
    path('erase-friend/', erase_from_friends, name='erase-friend'),
    path('my-friend-invites/approve/', approve_invitation, name='approve-invite'),
    path('my-friend-invites/deny/', deny_invitation, name='deny-invite'),
    path('<slug>/', UserProfileDetailView.as_view(), name='user-profile-detail-view'),
    path('search-friends/', search_friends, name='search-friends'),


]
```

postings > urls.py:

urls.py 5

friztalk > urls.py > ...

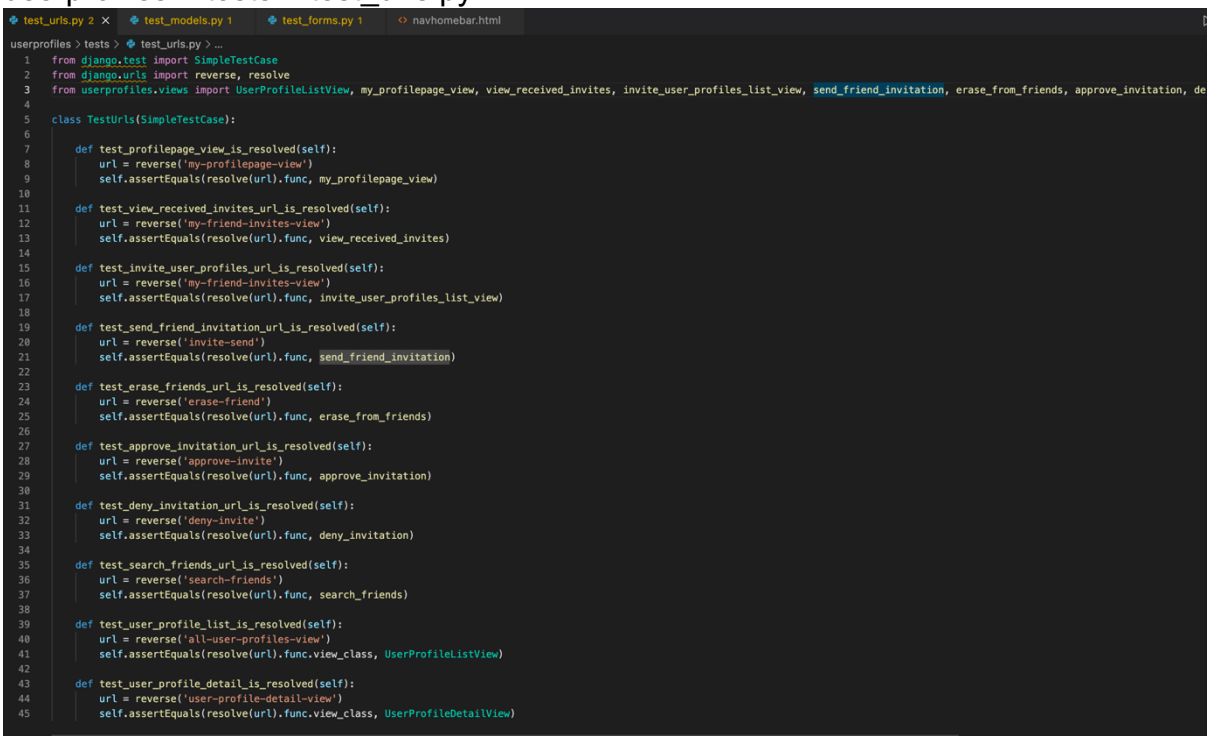
```python
"""friztalk URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.0/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static
from .views import home_page

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', home_page, name='home-page'),
    path('userprofiles/', include('userprofiles.urls', namespace = 'userprofiles')),
    path('postings/', include('postings.urls', namespace = 'postings')),
    path('accounts/', include('allauth.urls')),

]

urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)
```

Appropriate use of unit testing:

I have implemented unit testings on both apps, userprofiles and postings.

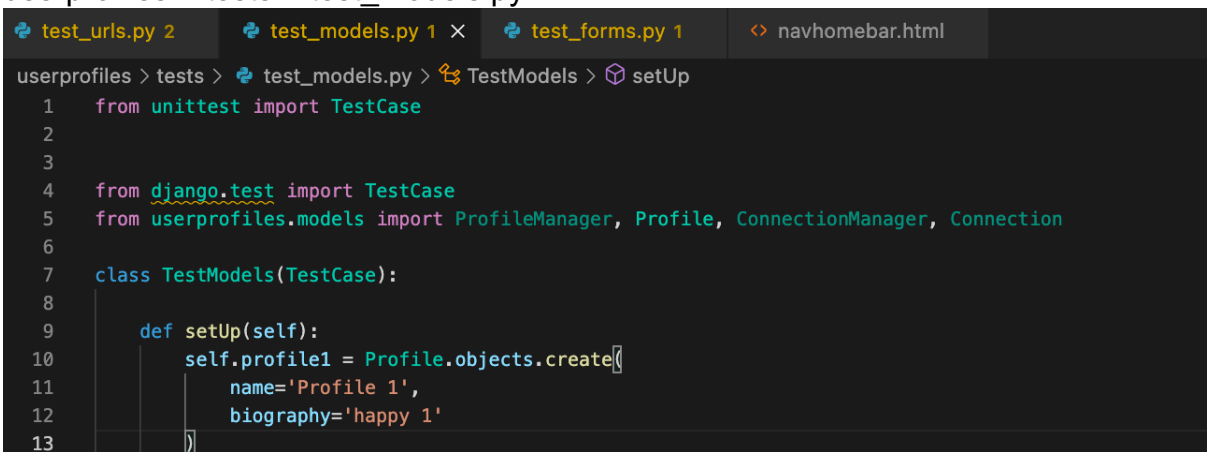
Under userprofiles, I created a tests folder, and I will be testing the forms, models and urls.

userprofiles > tests > test_urls.py:



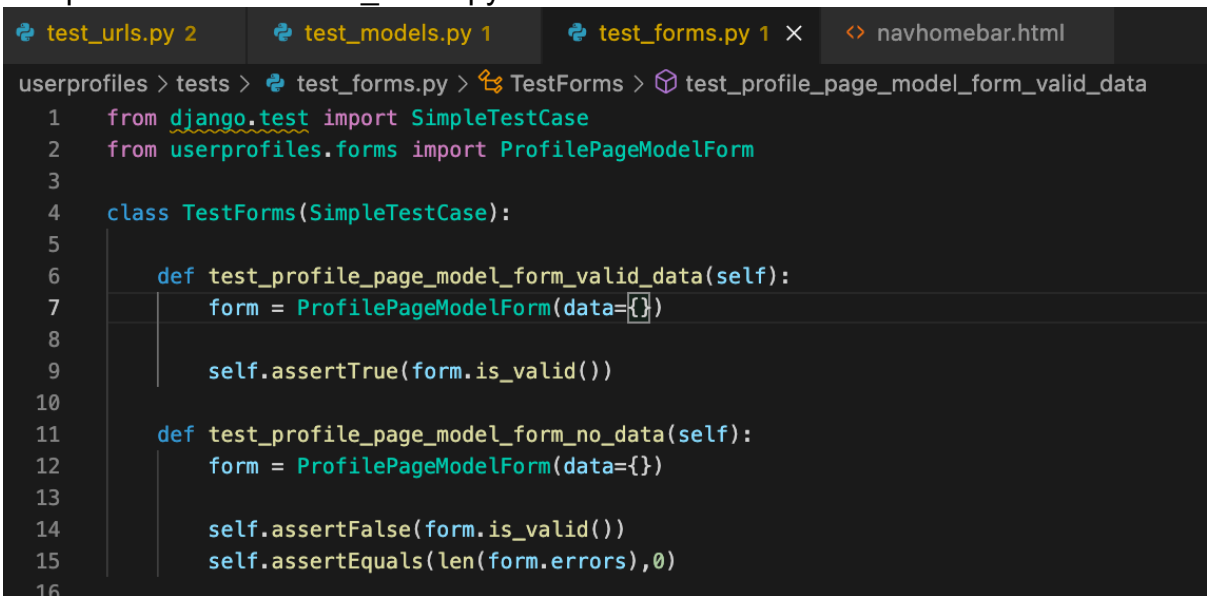
```python
from django.test import SimpleTestCase
from django.urls import reverse, resolve
from userprofiles.views import UserProfileListView, my_profilepage_view, view_received_invites, invite_user_profiles_list_view, send_friend_invitation, erase_from_friends, approve_invitation, de

class TestUrls(SimpleTestCase):

    def test_profilepage_view_is_resolved(self):
        url = reverse('my-profilepage-view')
        self.assertEquals(resolve(url).func, my_profilepage_view)

    def test_view_received_invites_url_is_resolved(self):
        url = reverse('my-friend-invites-view')
        self.assertEquals(resolve(url).func, view_received_invites)

    def test_invite_user_profiles_url_is_resolved(self):
        url = reverse('my-friend-invites-view')
        self.assertEquals(resolve(url).func, invite_user_profiles_list_view)

    def test_send_friend_invitation_url_is_resolved(self):
        url = reverse('invite-send')
        self.assertEquals(resolve(url).func, send_friend_invitation)

    def test_erase_friends_url_is_resolved(self):
        url = reverse('erase-friend')
        self.assertEquals(resolve(url).func, erase_from_friends)

    def test_approve_invitation_url_is_resolved(self):
        url = reverse('approve-invite')
        self.assertEquals(resolve(url).func, approve_invitation)

    def test_deny_invitation_url_is_resolved(self):
        url = reverse('deny-invite')
        self.assertEquals(resolve(url).func, deny_invitation)

    def test_search_friends_url_is_resolved(self):
        url = reverse('search-friends')
        self.assertEquals(resolve(url).func, search_friends)

    def test_user_profile_list_is_resolved(self):
        url = reverse('all-user-profiles-view')
        self.assertEquals(resolve(url).func.view_class, UserProfileListView)

    def test_user_profile_detail_is_resolved(self):
        url = reverse('user-profile-detail-view')
        self.assertEquals(resolve(url).func.view_class, UserProfileDetailView)
```

userprofiles > tests > test_models.py:



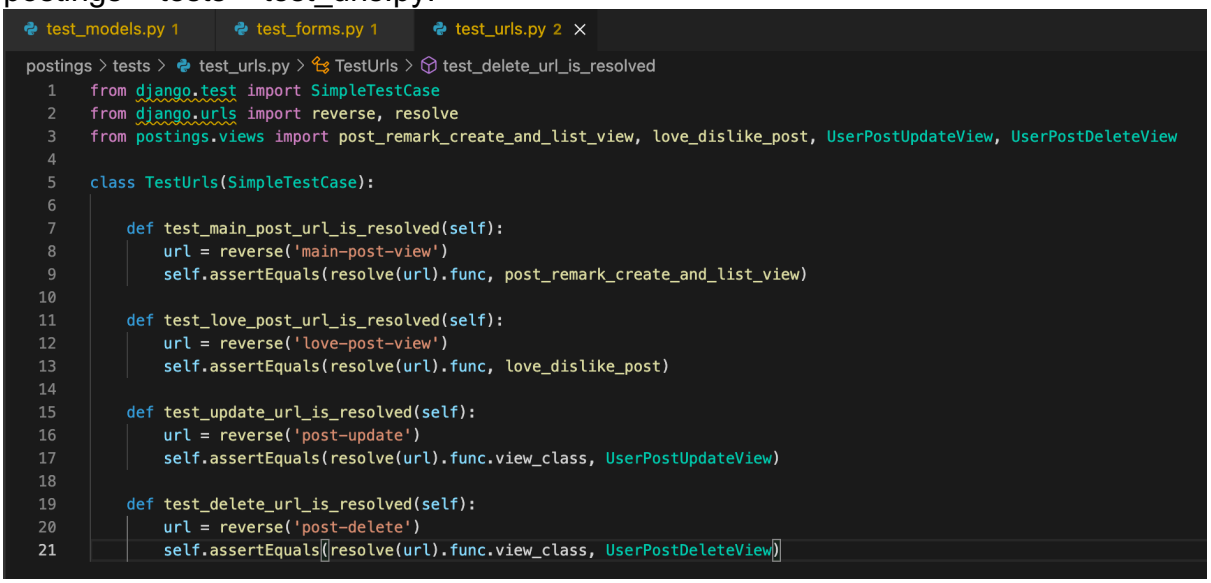
```python
from unittest import TestCase


from django.test import TestCase
from userprofiles.models import ProfileManager, Profile, ConnectionManager, Connection

class TestModels(TestCase):

    def setUp(self):
        self.profile1 = Profile.objects.create(
            name='Profile 1',
            biography='happy 1'
        )
```

userprofiles > tests > test_forms.py:

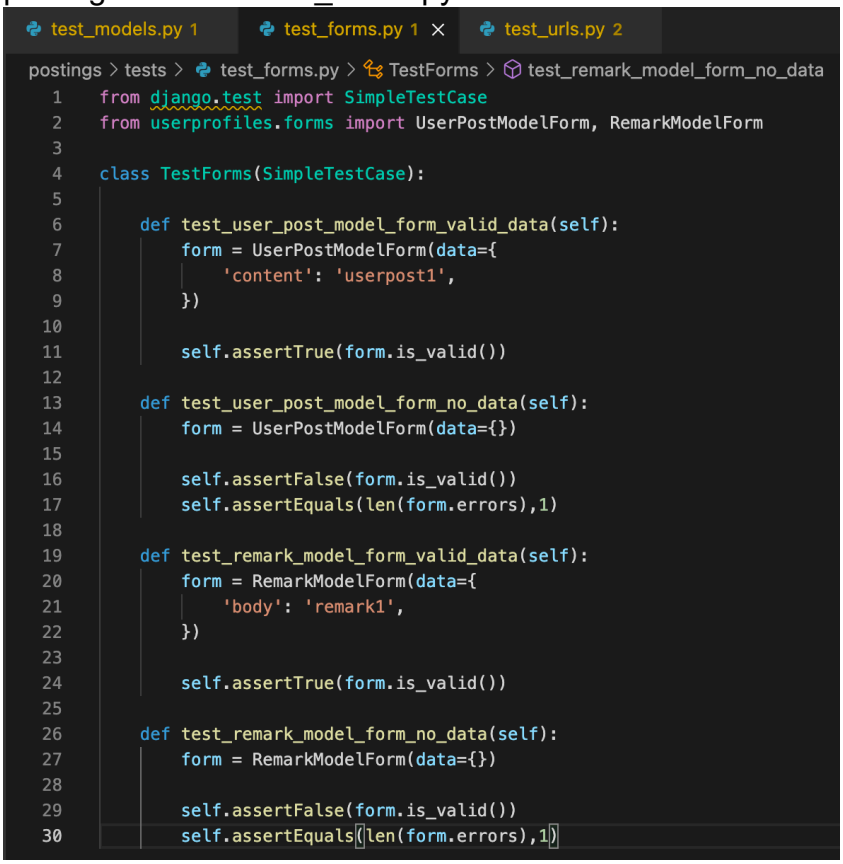


Likewise, under postings, I created a tests folder, and I will be testing the forms, models and urls.

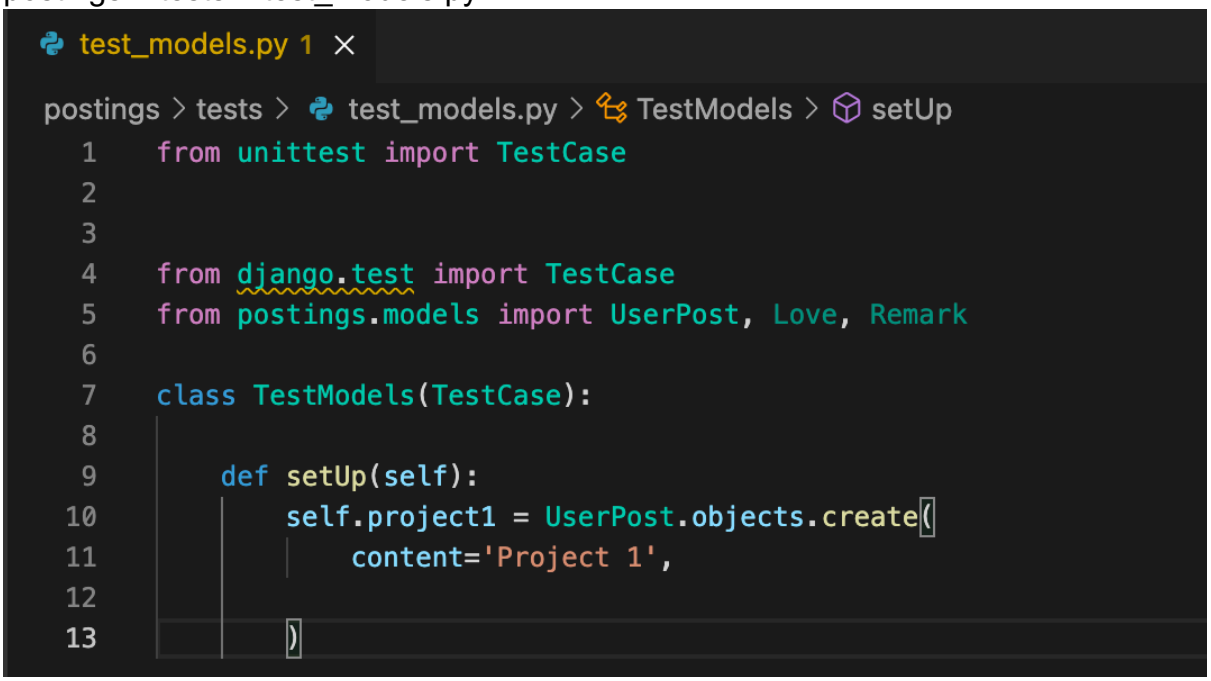
postings > tests > test_urls.py:



postings > tests > test_forms.py:



postings > tests > test_models.py:



This project has been an eye-opening and enjoyable to begin with. I have many takeaways from this final term assessment.
It is best to have a strategy and goal in mind. I tend to be more effective and efficient when doing so. The first stage in any online project should always be determining the ultimate objective so that you always know what you're working towards and have clear targets to keep in mind throughout the process.

Another thing would be the user experience. I can do hours and hours of coding, but it will be useless if I do not think about the overall design and experience of my social networking app. Always keep the user in mind, considering how they will interact with my social networking applications and ensuring that everyone has the greatest possible experience. This is what will make my app and all of my hard work worthwhile and valuable.

Another point is to keep my code useful and concise. I try to avoid writing complex codes. They are typically significantly riskier than simpler options, as they introduce more potential for unforeseen problems and concerns. As a result, I strive to prevent this. I try to keep my code short and make sure that every line is necessary.

Another point will be not to mix html with CSS or Java as it can get very messy, and you I have a hard time making out my code. I have also used semantic HTML. Semantic tags help browsers and search engines understand your on-page communication. They provide further customisation with the use of well-applied CSS.

I have arranged my code as it is as it easy to work with and I am able to import within one another. I have also designed my social network application as such, as I find it very intuitive and easy to follow. The home page provides all users to view all the posts of their friends, users can post a status accompanied with a photo. Users are also able to leave a remark on their own status or on other people’s status. Users can choose to Love or Dislike the status. Users of the original post have the freedom to edit their status or even delete it. Everything is in one page and at one glance, users are able to navigate through the page easily and seamlessly. I find that the overall design and appearance sleek and sexy, the maroon navigation bar stands out and it is the main colour code of my app. I have made the overall background light grey in colour so it will be easier for users to view, and to have an overall contrast. As mentioned, users are also able to search their friends, send them friend requests, and users are also able to edit their own profiles. This gives users the flexibility and platform to express themselves.

One part of the application which could have been done better would be the real time chat between friends. I have partially failed to implement it this time round. This is due to the fact that I am unable to link the friends to the chatpage.

As for me, I had a fantastic time doing it, and it is a great feeling to be able to implement my first ever social networking app, and not only about coding 24/7, but also having to think of the design of my app and also the overall user experience it has to offer.