MySocial App-> a social media application using Django Python Web Framework.

Functionalities/Demo:

- Users will be able to post feeds(upload pictures, texts) on their profiles.
- Feature that allows users to like and unlike other users' posts.
- Being redirected to their profiles once clicking their usernames.
- Displaying the amount of following and followers on users' profiles.
- Enabling users to follow and unfollow other users.
- Providing following suggestions on the Home page sidebar.
- Account Setting feature to change profile image, bio and location.
- Search bar to search for specific usernames.
- Login section with authentication after logout.

Developing Tools:

Django 4.1 https://www.djangoproject.com
MacBook OS Monterey Version 12.6 https://www.apple.com/macos/monterey/
Python 3.10.7 https://www.python.org/downloads/
Anaconda Python Environment 3.9
https://www.anaconda.com/products/distribution?gclid=Cj0KCQjw-fmZBhDtARIsAH6H8qhJsFCSk21Gg-mqnKGMFDP49R0JPQE3eDuNTV0N81AcoJquKcbk2ocaAvAHEALw_wcB

Prerequisites & Setup:

Create a new folder named `my_social_app`. Obtain its local directory.

Install Django:

In Mac Terminal:

```
cd <my_social_app DIRECTORY>
 pip install django
```

Create a new Django project in `my_social_app`:

```
 django-admin startproject my_social_app
```

Locate to the new folder generated:

```
 cd my_social_app
```

Create `Core` app to handle various functionalities:

```
 django-admin startapp core
```

Initial run to make sure the project works:

```
python3 manage.py runserver
```

If we get the following message, the project setup is done:

```
System check identified no issues (0 silenced).
You have 18 unapplied migration(s). Your project may not work properly
until you apply the migrations for app(s): admin, auth, contenttypes,
sessions.
Run 'python manage.py migrate' to apply them.
October 03, 2022 - 00:03:26
Django version 4.1.1, using settings 'my_social_app.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Where the url `http://127.0.0.1:8000/` is our Localhost for future performance.

Open the Localhost, if we see The install worked successfully! Congratulations!
You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs. Then all setups are done.

Synchronous Developing Notes:

URL Routing:

Import and open the project with Visual Studio Code IDE.

Create a new file named `urls.py` under `Core`.

In `urls.py`:

```
from django.urls import path
 urlpatterns = [
   path('', views.index, name = "index")]
```

use `views.index` to access `views.py` here.

Create a simple HTTP Response request in `views.py`:

```
from django.http import HttpResponse
```

Import views: `from . import views`

To get urls reflected on the main, under `my_social_app` folder, modify `urls.py`:

```
from django.urls import path, include
urlpatterns = [
   path("admin/", admin.site.urls),
   path('', include('core.urls'))]
```

Refresh localhost `http://127.0.0.1:8000`:

welcome testing white page.PNG

Template Setup:

Configurations setup:

in `settings.py`:

```
import os
```

Create a new folder named `templates` to hold all html files. Need to tell Django:

```
 "DIRS": [os.path.join(BASE_DIR, "templates")]
```

Downloadable essential html files https://github.com/tomitokko/django-social-media-website

Drag all html files (`index.html, setting.html, signin.html, signup.html, profile.html`) into templates.

In `views.py`:

```
return render(request, "index.html")
```

Now all templates are cluttered on localhost:

all templates are uploaded.PNG

Static Files:

Refresh Vscode IDE, create a new folder named `static`:

Set up static files roots in `settings.py`:

```
STATIC_URL = "static/"
STATIC_ROOT = os.path.join(BASE_DIR, "staticfiles")
STATICFILES_DIRS = (os.path.join(BASE_DIR, "static"), )
```

Move and copy all static files into the static folder.
Now in `index.html`, we have all static properties loaded:

```
    <link rel="stylesheet" href="{% static 'assets/css/icons.css' %}">
    <link rel="stylesheet" href="{% static 'assets/css/uikit.css' %}">
    <link rel="stylesheet" href="{% static 'assets/css/style.css' %}">
    <link rel="stylesheet" href="{% static
'assets/css/tailwind.css'%}">
```

Change the configuration of the JavaScript link to the favicon.PNG image:

```
<link href="{% static 'favicon.png' %}" rel="icon" type="image/png">
```

Refresh the localhost, we get **basic all JavaScript worked template.PNG**

Creating a Profile Model:

Create authentication with the website->Register and login to the social media:
we need to create a customized profile model, so in `models.py`, extend database, create a separate model to the profile and link it to the user model using **foreign key**
https://en.wikipedia.org/wiki/Foreign_key :
import contrib.auth to get user model and initialize it:

```
from django.contrib.auth import get_user_model
User = get_user_model()
```

Create all necessary attributes to specify profile model:

```
 class Profile(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    id_user = models.IntegerField()
    bio = models.TextField(blank=True)
    profileimg = models.ImageField(upload_to='profile_images',
default='blank-profile-picture.png')
    location = models.CharField(max_length=100, blank=True)
```

Here, create a new `media` folder to hold the default avatar, which is `blank-profile-picture.PNG` here, and configure media url and root in `settings.py`:

```
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

Also configure the url pattern in `urls.py`:

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

Now return the username:

```
    def __str__(self):
        return self.user.username
```

Open the Terminal panel:
Now we got the following admin access for the static files:

```
"GET /static/admin/css/fonts.css HTTP/1.1" 304 0
"GET /static/admin/fonts/Roboto-Regular-webfont.woff HTTP/1.1" 304 0
"GET /static/admin/fonts/Roboto-Light-webfont.woff HTTP/1.1" 304 0
"GET /static/admin/fonts/Roboto-Bold-webfont.woff HTTP/1.1" 304 0
```

`You have 18 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.`

Now we need to make migrations:

```
python3 manage.py makemigrations
```

Migrations for core app are made:

```
Migrations for 'core':
  core/migrations/0001_initial.py
    - Create model Profile
```

Now migrate:

```
python3 manage.py migrate
```

The migrations operations are all successful:

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, core, 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 core.0001_initial... OK
  Applying sessions.0001_initial... OK
```

As admin, we need to create a super user, which is the administrator as ourselves:

```
python3 manage.py createsuperuser
```

Create `username, email address` and `password` for admin login.

Then to make the profile models visible on **admin portal,** in `admin.py`:

```
from .models import Profile
admin.site.register(Profile)
```

Now open and login to http://127.0.0.1:8000/admin/ , we have:

admin portal initial view.PNG

Signup:

Add a new signup path in `urls.py(Core):`

```
path('signup', views.signup, name = 'signup' )
```

Create the signup view in `views.py:`

```
def signup(request):
    return render(request, "signup.html")
```

Type http://127.0.0.1:8000/signup we got: **signup initial page.PNG**

Temporary Error:

```
raise ValueError(
ValueError: The view core.views.signup didn't return an HttpResponse
object. It returned None instead.
[06/Oct/2022 10:56:36] "POST /signup HTTP/1.1" 500 61151
```

DEBUGGING: In `views.py`, import messages and authentication of user model:

```
from django.contrib.auth.models import User, auth
from django.contrib import messages
```

format document->CONTROL+C-> python3 manage.py runserver

Since passwords are confidential values, we need to use POST method in `signup.html:`

```
<form action= "" method = "POST">
```

Also we use a CSRF token to prevent CSRF attacks:

```
{% csrf_token %}
```

Implement the **Signup View:**

If use POST method, load all user attributes:

```
def signup(request):
    if request.method == "POST":
        username = request.POST['username']
        email = request.POST['email']
        password = request.POST['password']
        password2 = request.POST['password2']
```

If two passwords matched and input email exists, redirect user back to signup page:

```
if password == password2:
            if User.objects.filter(email=email).exists():
                messages.info(request, 'Email Taken')
                return redirect('signup')
```

If two passwords matched and input username exists, redirect back to signup page:

```
elif User.objects.filter(username=username).exists():
                messages.info(request, 'Username Taken')
                return redirect('signup')
```

If two passwords matched and new user, create a new user with all the infos and save:

```
  else: user = User.objects.create_user(username=username,
email=email, password=password)
```

```
                user.save()
```
Render the request:
```
        return render(request, "signup.html")
```
Now in `signup.html`, create styles to highlight red the duplicate warning message:
```
   <div> <style>
           h5 {color: red; }
                </style>
                {% for message in messages %}
             <h5>{{message}}</h5>
                 {% endfor %}  </div>
```
Now test the signup page with existing username, email and unmatched PWs should get:
user taken.PNG **email taken.PNG** **unmatched passwords.PNG**
Now with no error, we want the system to automatically create a new user with creating a new profile that we can view in our **Django Administration** page. So in `views.py`:
```
 #log user in and redirect them to settings page
 #create a Profile object for the new user
      user_model = User.objects.get(username=username)
      new_profile = Profile.objects.create(user=user_model,
id_user=user.id)
      new_profile.save()
      return redirect('signup')
```
Also import Profile Model to make profile creation work:
```
from .models import Profile
```
Now register new user on Signup portal, and refresh from the admin portal, we can see that the user we just registered has their username and profile successfully created:
user's username and their profile created.PNG
Signin and Logout:
Noticeable Error:
10k+ tracking forks on the sidebar Source Control every time press `CONTROL+C` to re-run the localhost server, unable to remove the accidentally fetched **.gitignore.**
DEBUGGING:
METHOD 1:
Right-click main in Source Control and select `Close Repositories.` DO NOT use python shell virtual environment in Terminal, if activated by accident, deactivate **virtualenv** with:
`conda deactivate`
METHOD 2:
Delete the entire .Git folder to temporarily disable Source Control:
In Vscode: **`Preference-> Settings-> Git Enabled(CLICK OFF)`**
Now we successfully disable SCM from auto-fetching the .GIt root repositories.

Add a new path in `urls.py`:

```
path('signin', views.signin, name = 'signin' )
```

In `views.py`, create a new sign in request:

```
def signin(request):
    return render(request, "signin.html")
```

type in http://127.0.0.1:8000/signin we get:

signin page.PNG

Configure POST Method for login authentication:

In `views.py`, if user exists after input username and password, log the user in, if the user does not exist or either password OR username is incorrect, “credential invalid”:

```
def signin(request):
        if request.method == "POST":
        username = request.POST['username']
        password = request.POST['password’]
        user = auth.authenticate(username=username, password =
password)
        if user is not None:
            auth.login(request, user)
            return redirect('/')
        else:
            messages.info(request, 'Credentials Invalid')
            return redirect('signin')
        else:
        return render(request, "signin.html")
```

Credential invalid message.PNG

To achieve logout, start by creating a new logout path in `urls.py`:

```
path('logout', views.logout, name = 'logout' ),
```

Then define Logout functionality in `views.py`:

```
def logout(request):
    auth.logout(request)
    return redirect('signin')
```

Now if the user click the sidebar logout, they will be redirected to sign in page.

Also we need to import login_required:

```
from django.contrib.auth.decorators import login_required
```

which for security purposes will redirect the user to the login page if detected user not logged in.

Along with the signature above index request:

```
@login_required(login_url='signin')
```

So that when we refresh the home page, sign in page will show with the following url:

redirect to signin url.PNG

http://127.0.0.1:8000/signin?next=/

So if ONLY the registered user sign in here they will be redirected to the home page.

Account Settings:

Start by creating a new Settings path in `urls.py`:

```
path('settings', views.settings, name = 'settings'),
```

Also render the request from `setting.html` in `views.py`:

```
@login_required(login_url='signin')
def settings(request):
    return render(request, "setting.html")
```

Now if we type in http://127.0.0.1:8000/settings

Account Settings page should appear available for user to edit Basic and Privacy Infos:

Account Setting initial page.PNG

Log the new registered user automatically to Account Setting page:

```
   #log user in and redirect them to settings page
              user_login = auth.authenticate(username=username,
password=password)
                auth.login(request, user_login)

                #create a Profile object for the new user
                user_model = User.objects.get(username=username)
                new_profile = Profile.objects.create(user=user_model,
id_user=user.id)
                new_profile.save()
                return redirect('settings')
```

Now we want specific username to show up on Account Setting page:

In `setting.html`:

```
<h1 class="text-2xl leading-none text-gray-900 tracking-tight mt-3"><a
href="/">Home</a> / Account Setting for <b>{{user.username}}</b></h1>
```

Home/ Account Setting for username.PNG

Remove Privacy, the overview of the General left only setting page:

general left only account setting.PNG

Remove Working at and Relationship:

simplified setting.PNG

Give user option to upload profile image:

```
  <div class="col-span-2">
            <label for=""> Profile Image</label>
            <input type="file" name="image" placeholder=""
class="shadow-none bg-gray-100">
                    </div>
```

profile image upload.PNG

ERROR:
Django return "Profile matching query does not exist" when refresh Account Setting page.
DEBUGGING: Request new url link for Profile Image-> profileimg in `setting.html:`

```
<div class="col-span-2">
                    <label for=""> Profile Image</label>
                    <img width="100" height="100"
src="{{user_profile.profileimg.url}}" />
                   <input type="file" name="image" value=""
class="shadow-none bg-gray-100">
                   </div>
```

Now besides allowing user to upload their own profile image, the default avatar shows:
default profile image shows/upload pfp.PNG
Save all uploads and settings changes:
In `views.py:`
If user upload their own profile image, save all settings infos in their profile:

```
    if request.FILES.get('image') != None:
            image = request.FILES.get('image')
            bio = request.POST['bio']
            location = request.POST['location']

            user_profile.profileimg = image
            user_profile.bio = bio
            user_profile.location = location
            user_profile.save()
        return redirect('settings')
```

If user did not upload their own profile image, save all settings infos in their profile:

```
    if request.FILES.get('image') == None:
            image = user_profile.profileimg
            bio = request.POST['bio']
            location = request.POST['location']

            user_profile.profileimg = image
            user_profile.bio = bio
            user_profile.location = location
            user_profile.save()
```

Being redirected to Account Settings page when user clicks `Account setting` on sidebar:

```
    <li><a href="/settings"> Account setting </a> </li>
```

Now if I upload **krystal profile image.PNG** as my own profile image and edit the settings as:
all settings are saved.PNG
After saved, on admin portal we can see the profile for krystal as updated successfully:
krystal profile on admin side.PNG

Uploading Posts:

In `models.py`, to make every post with an unique ID:

```
import uuid
from datetime import datetime
...
id= models.UUIDField(primary_key = True,default=uuid.uuid4)
```

The rest of the attributes initialization for posting posts:

```
    user = models.CharField(max_length = 100)
    image = models.ImageField(upload_to='post_images')
    caption = models.TextField()
    created_at = models.DateTimeField(default= datetime.now)
    no_of_likes = models.IntegerField(default=0)
```

Then we make migrations for this newly created model in Terminal:

```
 python3 manage.py makemigrations
...
 Migrations for 'core':
  core/migrations/0002_post.py
    - Create model Post
```

Now migrate:

```
 python3 manage.py migrate
...
 Operations to perform:
  Apply all migrations: admin, auth, contenttypes, core, sessions
Running migrations:
  Applying core.0002_post... OK
```

Now in admin portal, we can see `Posts` appears as a Core:

posts as core appears.PNG

Create a new upload view in `views.py`:

```
 @login_required(login_url='signin')
def upload(request):
    return HttpResponse('<h1>Upload View</h1>')
```

Upload initial button.PNG

We want the right top profile image to show our own profile image. So in `index.html`:

```
  <a href="#">
<img src="{{user_profile.profileimg.url}}" class="header-avatar"
alt=""> </a>
```

right sidebar show our own profile image.PNG

In `views.py`, add new image upload view:

```
 @login_required(login_url='signin')
def upload(request):
      if request.method == "POST":
        user = request.user.username
```

```
        image = request.FILES.get('image_upload')
        caption = request.POST['caption']
        new_post = Post.objects.create(user=user, image=image,
caption=caption)
        new_post.save()
        return redirect('/')
    else:
        return redirect('/')
```

Error when uploading new image:
Forbidden (403)
CSRF verification failed. Request aborted.
Help
Reason given for failure:
  CSRF token missing.
DEBUGGING:
add `{% csrf_token %}` after `<form action...>` in `index.html`.
Now post a new image **new post image1.PNG**
And from admin portal we can see: **new post from krystal admin.PNG**
Which the new post has its unique ID.
Post Feed:
Change the post username to our own username in `index.html`:

```
<span class="block font-semibold ">@{{post.user}}</span>
```

Change the post image to the logged in user uploaded:
In `index.html` change the image source url:

```
<img src="{{post.image.url}}" alt="">
```

Now **my first post showing.PNG**
And click the image we got the zoom in: **zoom in first image.PNG**
Now reduce unnecessary features and add:

```
<p>
 <a>{{post.user}} {{post.caption}}</a></p>
```

Refresh and the caption description shows: **first post with caption showing.PNG**
Bolden the username prefix to caption:

```
<a><strong>{{post.user}}</strong></a> {{post.caption}}
```

Add a hash link as:

```
<span class="block font-semibold "><a href =
"#">@{{post.user}}</a></span>
```

So when clicking on username above post, go to http://127.0.0.1:8000/#
Now let's upload another picture:
new post image2.PNG
second post.PNG
To get the latest posts always shows ABOVE the older posts:

```
{% for post in posts reversed %}
```

Like Posts:

Create a new LikePost model in `models.py`:

```
class LikePost(models.Model):
    post_id = models.CharField(max_length = 500)
    username = models.CharField(max_length=100)
    def __str__(self):
        return self.username
```

Make migrations in Terminal:

```
 python3 manage.py makemigrations
...
Migrations for 'core':
  core/migrations/0003_likepost.py
    - Create model LikePost
```

Migrate:

```
 python3 manage.py migrate
...
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, core, sessions
Running migrations:
  Applying core.0003_likepost... OK
```

Create a new view for LikePost in `views.py`:

```
@login_required(login_url='signin')
def like_post(request):
    username = request.user.username
    post_id = request.GET.get('post_id')
    post = Post.objects.get(id=post_id)
```

Check if a registered user already liked or not liked a specific post to determine if like/unlike:

```
 like_filter = LikePost.objects.get(post_id=post_id, username =
username).first()
```

If a user has not liked this post, their click will become a new like that save to the post:

```
   if like_filter == None:
        new_like = LikePost.objects.create(post_id=post_id,
username=username)
        new_like.save()
        post.no_of_likes = post.no_of_likes+1
        post.save()
        return redirect('/')
```

Otherwise, unlike the post:

```
     else: like_filter.delete()
       post.no_of_likes = post.no_of_likes-1
       post.save()
       return redirect(‘/’)
```

We want the number of likes to show underneath the post, so in `index.html`:

```
<p>Liked by {{post.no_of_likes}} person</p>
```

Now if we click like button: **1 person liked the post.PNG**

To differentiate from liked by 0, 1 and 2+ people, we need if-statement:

```
  </svg>
  {% if post.no_of_likes == 0 %}
   <p>No likes</p>
  {% elif post.no_of_likes == 1 %}
   <p>Liked by {{post.no_of_likes}} person</p>
   {% else %}
   <p>Liked by {{post.no_of_likes}} people</p>
   {% endif %}
  </div>
```

Profile Page:

To customize our own profile page, create a new url in `urls.py`:

```
path('profile/<str:pk>', views.profile, name = 'profile'),
```

Also create new view in `views.py`:

```
@login_required(login_url='signin')
def profile(request, pk):
    return render(request, 'profile.html')
```

Go to http://127.0.0.1:8000/profile/krystalzhang612

and can observe our own profile: http://127.0.0.1:8000/profile/krystalzhang612

user's own profile.PNG

To customize profile home page, in `profile.html`:

To make the posted images visible on their profile:

```
      {% for post in user_posts %} <li>
<a class="strip" href="{{post.image.url}}" title=""
data-strip-group="mygroup" data-strip-group-options="loop: false">
<img src="{{post.image.url}}" style="height: 250px; width: 300px;"
alt=""></a></li>
      {% endfor %}
```

To make the number of the posts reflected:

```
      {% if user_post_length == 0%}
<span style="color: white; font-size: 27px;"><b>No Post</b></span>
      {% elif user_post_length == 1%}
<span style="color: white; font-size: 27px;"><b>{{user_post_length}}
Post</b></span>
      {% else %}
<span style="color: white; font-size: 27px;"><b>{{user_post_length}}
Posts</b></span>
      {% endif%}
```

krystal's profile overview.PNG

Profile Page Posts Display:

Make profile images scrollable:

```
 <script data-cfasync="false" src="{% static
'../../cdn-cgi/scripts/5c5dd728/cloudflare-static/email-decode.min.js'
%}"></script><script src="{% static 'js/main.min.js' %} "></script>
```

Follow and Unfollow Users:

Make the following button work. Create a new user @mckennagrace. Then in `models.py`:

```
 class FollowersCount(models.Model):
    follower = models.CharField(max_length=100)
    user = models.CharField(max_length=100)
    def __str__(self):
        return self.user
```

Similar to prior, make migrations in Terminal:

```
 python3 manage.py makemigrations
...
Migrations for 'core':
  core/migrations/0004_followerscount.py
    - Create model FollowersCount
```

Migrate:

```
 python3 manage.py migrate
...
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, core, sessions
Running migrations:
  Applying core.0004_followerscount... OK
```

Then create a new url to send the FollowerCount request. In `urls.py`:

```
 path('follow', views.follow, name = 'follow'),
```

Then we can have a new view in `views.py` corresponds to the url:

```
@login_required(login_url='signin')
def follow(request):
if request.method == "POST":
        follower = request.POST['follower']
        user = request.POST['user']
    else:
        return redirect('/')
```

Add two fields for the Follow button in `profile.html`:

```
<input type= "hidden" value= "{{user.username}}" name = "follower" />
<input type= "hidden" value= "{{user_object.username}}" name = "user"
```

In `views.py`, check if a person already followed a user, if followed, delete follower(unfollow):

```
   if FollowersCount.objects.filter(follower=follower,
user=user).first():
```

```
            delete_follower =
FollowersCount.objects.get(follower=follower, user=user)
            delete_follower.delete()
            return redirect('/profile/'+user)
```

Else, if not followed yet, add the new follower:

```
           else:
            new_follower =
FollowersCount.objects.create(follower=follower, user= user)
            new_follower.save()
            return redirect('/profile/'+user)
```

Now hit Follow button: **follower count admin.PNG**

Make Follow button disappear if a user is viewing their own profiles. So in `profile.html`:

```
 {% if user_object.username == user.username %}
<a href = "/settings" data-ripple="">Account Settings</a>
 {% else %}
<a data-ripple=""><button type = "submit" style="background-color:
#ffc0cb; border: #ffc0cb;">Follow</button></a>
 {% endif %}
```

So at logged in user's own profile page: **not follow button but account setting.PNG**

To switch Unfollow/Follow based on following status, in `views.py`:

```
    follower = request.user.username
    user = pk
    if FollowersCount.objects.filter(follower = follower, user =
user).first():
        button_text = 'Unfollow'
    else:
        button_text = 'Follow'
```

unfollow button works.PNG

Now in order to get correct amounts of followers and following showing, in `views.py`:

```
    user_followers = len(FollowersCount.objects.filter(user=pk))
    user_following = len(FollowersCount.objects.filter(follower=pk))
```

Also in `profile.html`:

```
 {% if user_followers == 0 or user_followers == 1%}
     <span style="color: white; font-size:
27px;"><b>{{user_followers}} follower</b></span>
     {% else %}
<span style="color: white; font-size: 27px;"><b>{{user_followers}}
followers</b></span>
     {% endif %}
<span style="color: white; font-size: 27px;"><b>{{user_following}}
following</b></span>
```

Now we get: **correct amount of following and followers showing.PNG**

Post Feed Updated:

To get to see the following users' feed, we need to use iteration and append in `views.py`:

```
  user_following =
FollowersCount.objects.filter(follower=request.user.username)
 for users in user_following:
 user_following_list.append(users.user)
 for usernames in user_following_list:
   feed_list = Post.objects.filter(user=usernames)
   feed.append(feed_list)
   feed_list = list(chain(*feed))
   posts = Post.objects.all()
   return render(request, "index.html", {'user_profile': user_profile,
'posts': feed_list})
```

Now krystal's posts shows on mckenna's feed since she follows her:

following users' feed visible.PNG

Download Post Images:

In `index.html`:

```
<a href="{{post.image.url}}" class="flex items-center space-x-2 flex-1
justify-end" download >
```

Click the bottom square-shaped button and the download feature works:

download feature works.PNG

Search User:

Make search bar searchable in `index.html`:

```
<input type="text" name = "username" placeholder="Search for
username..">   
<button type ="submit"><i class="fa fa-search fa-1x"></i></button>
```

Now the search bar has proper spacing and placeholder:

username search bar initial look.PNG

Then create new url request as search in `views.py`:

```
@login_required(login_url='signin')
def search(request):
    user_object = User.objects.get(username=request.user.username)
    user_profile = Profile.objects.get(user= user_object)
   if request.method == 'POST':
        username = request.POST['username']
  username_object = User.objects.filter(username__icontains =
username)
        username_profile= []
        username_profile_list = []
        for users in username_object:
        username_profile.append(users.id)
```

```
        for ids in username_profile:
            profile_lists = Profile.objects.filter(id_user = ids)
            username_profile_list.append(profile_lists)
        username_profile_list = list(chain(*username_profile_list))
    return render(request, 'search.html', {'user_profile':
user_profile, 'username_profile_list': username_profile_list})
```

Then loop the search list in `search.html`:

```
  {% for users in username_profile_list %}
        <section class="search-result-item">
              <a class="image-link"
href="/profile/{{users.user}}"><img class="image"
src="{{users.profileimg.url}}">
                  </a>
        <div class="search-result-item-body">
                <div class="row">
         <div class="col-sm-9">
   <h4 class="search-result-item-heading"><a
href="/profile/{{users.user}}"><b>@{{users.user}}</b></a></h4>
   <p class="info">{{users.location}}</p>
   <p class="description">{{users.bio}}</p>
```

Now if search `krystalzhang612` then we get the search feature working:

username search feature works.PNG

User Suggestions:

Looping through all the lists of following users in `views.py`:

```
  # user suggestion starts
    all_users = User.objects.all()
    user_following_all = []
    for user in user_following:
        user_list = User.objects.get(username = user.user)
        user_following_all.append(user_list)
    new_suggestions_list = [x for x in list(all_users) if (x not in
list(user_following_all()))]
    current_user = User.objects.filter(username=request.user.username)
    final_suggestions_list = [x for x in list(new_suggestions_list) if
(x not in list(current_user))]
    random.shuffle(final_suggestions_list)
```

Now create at least 4 more new accounts.

registered accounts:

@krystalzhang612 @admin @mckennagrace

@danreynolds @barmstrong @margotrobbie @chesterb

Now refresh the logged in page and we can see on sidebar all suggested users showed:

user suggestions showed.PNG