Project 4: Social Network part 2

Will start by initiate project and required apps:

Step1: Create a Django Project and App

- Create a project named social

```
django-admin startproject social
```

- create an app named dwitter

```
python manage.py startapp dwitter
```

- Don’t forget to install the app in settings

```
# social/settings.py

# ...

INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "dwitter",
]
```

Step2: Migrate and create super user

- Create database

```
python manage.py migrate
```

- Create superuser

```
python manage.py createsuperuser
```

- Finally, run the server

```
python manage.py runserver
```

Django Admin Interface

It’s a powerful tool for managing your application, will apply some customizations

Step3: Open admin panel and investigate default models

By default, default model entries for *Groups* and *Users* which come from Django’s built-in authentication and user management apps

But we don’t want Groups, so we have to eliminate it from admin view which makes us focus on what is needed. What is **Groups?**

- means of categorizing users
- This allows for granting permissions to a specific group.
- It also saves a lot of time, as compared to granting permissions to each individual user.

A user may belong to any number of groups and automatically has all the permissions granted to that group.

Step4: Unregistering the Group model

- First, Group model is related to `django.contrib.auth.models`

- To unregister from admin, control should be in admin file
- Go to dwitter admin file
- Add the following code:

```
# dwitter/admin.py

from django.contrib import admin
from django.contrib.auth.models import Group

admin.site.unregister(Group)
```

- first import it from `django.contrib.auth.models`
- Then, use `.unregister()` to remove it from admin display
- Recheck admin panel

Now, regarding users, will only display the username, gonna customize the admin panel for this.

To do this, need to:

- Need to first unregister it since the model comes registered by default.
- Then, you can re-register the default `User` model to limit which fields the Django admin should display

Step4: Displaying the User model

- Go to admin file in Dwitter
- Add the following code

```
from django.contrib import admin
from django.contrib.auth.models import User, Group

class UserAdmin(admin.ModelAdmin):
    model = User
    # Only display the "username" field
    fields = ["username"]

admin.site.unregister(User)
admin.site.register(User, UserAdmin)
admin.site.unregister(Group)
```

Explanation:

- import the built-in `User`
- create `UserAdmin` a custom class based on the imported `User` model
- limit the fields that the admin interface displays to only `username` which is enough to create a test user
- unregister the `User` model that comes registered by default in the admin interface
- register the `User` model again, additionally passing the custom `UserAdmin` class we created, which applies the changes that we want

Recheck admin panel

Now, we need a way to hold information about the users of our app:

- started from scratch, you'd have to build an entirely new user model for that
- Instead, going to use the built-in Django `User` model to rely on Django's well-tested implementation
- which can avoid reinventing the authentication wheel

However, we need additional functionality that the default `User` model doesn't cover which is:

How can one user follow another user? need a way to connect users to other users.

The built in User focuses on the minimum setup necessary for authentication, will apply specific customizations, with extend the `User` model

Profile Model

- Will extend Django's built-in `User` model by using a one-to-one relationship with a small and focused new model, `Profile`
- Will build this `Profile` from scratch.
- This `Profile` model will keep track of the additional information that want to collect about each user

What do we need in addition to the user information??

Analysis:

- The `Profile` model only contains information that your users create *after* they already have a user account
- This allows you to let Django handle the sign-up and authentication process
- `Profile` model should record the user's connections to other user profiles
  - `Profile` model will be setting it up to record who follows a profile and, from the other direction, whom a profile follows

So need to create one field to model both of these connections

Step5: Creating profile model

- Open models file in dwitter
- Add the following code

```
# dwitter/models.py

from django.db import models
from django.contrib.auth.models import User
```

```
class Profile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    follows = models.ManyToManyField(
        "self",
        related_name="followed_by",
        symmetrical=False,
        blank=True
    )
```

Explanation:

- import the built-in `User`
- define a `OneToOneField` object called `user`
    - representing the profile's connection to the user that was created with Django's built-in user management app
    - define that any profile will get deleted if the associated user gets deleted
- define a `ManyToManyField` object with the field name `follows` which can hold connections to other user profiles
    - `related_name` keyword on your `follows` field, which allows you to access data entries from the other end of that relationship through the descriptive name `"followed_by"`
- set `symmetrical` to `False` so that your users can follow someone without them following back
- `blank=True` which means your users don't *need* to follow anyone. The `follows` field can remain empty

Step6: Update database

- Run the following commands

```
python manage.py makemigrations
python manage.py migrate
```

Step7: Register the Profile model to admin interface

- Go to admin file in dwitter
- Register as follows:

```
# dwitter/admin.py

# ...
from .models import Profile

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

- Run the server