

Adding Applications: User App

Introduction

We'll continue the development of our Django project by adding other applications besides the projects app that we already finalized styling during the last guide.

You'll see that the process of adding new applications to a Django project tends to be very repetitive and many of the steps in designing a new application are the same or at least they have many commonalities among them:

- Create the application using the command

```
python manage.py startapp <application_name>
```

- Set up our routing links to the new app
- Create our new app model
- Do the database migrations

```
python manage.py createmigration  
python manage.py migrate
```

- Create our forms and route them
- Do the corresponding styling by modifying our templates and setting the input fields accordingly to our needs in the model form.

During this guide we will be creating an application for the users (the developers in our site). Our project already has three tables available:

1. Project table,
2. Review table, and
3. Tag table

all present within our "Projects" app.

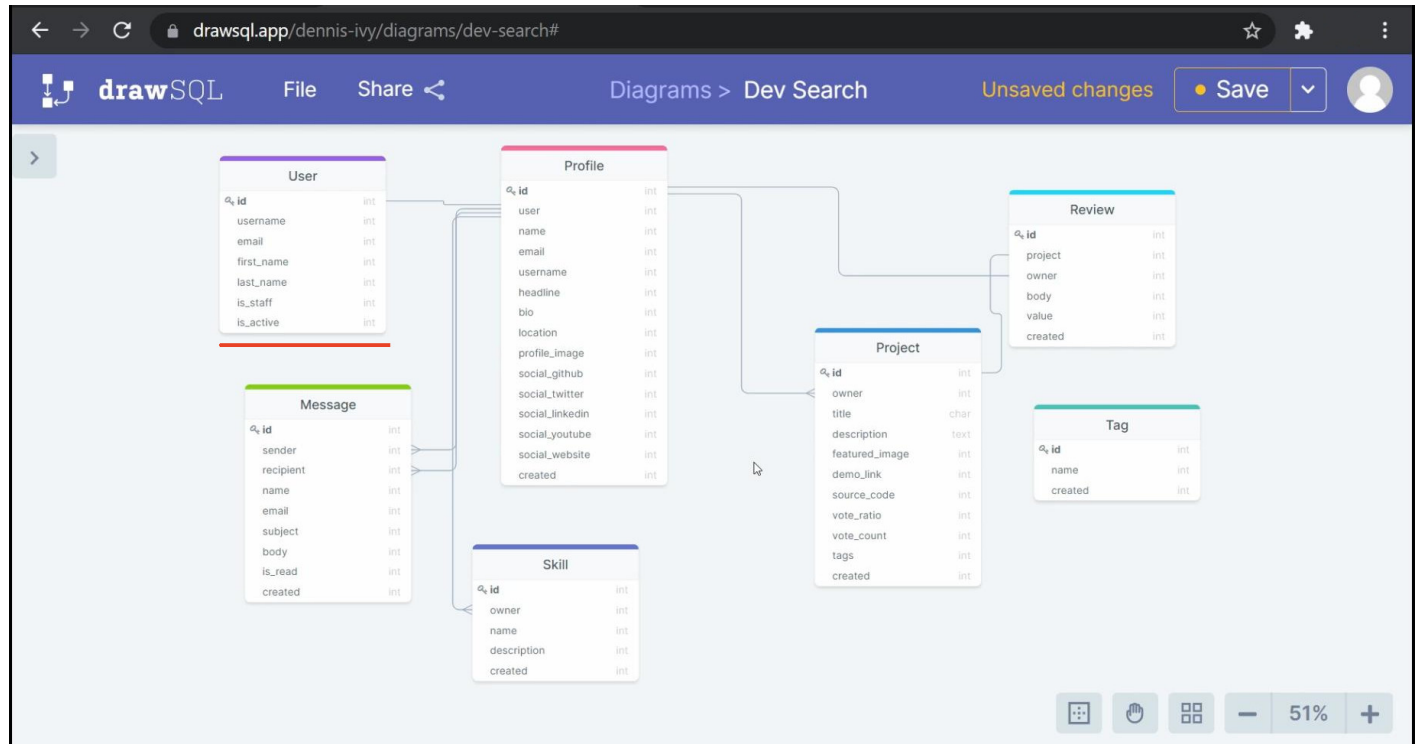
The new application will add three new tables:

1. Messages table,
2. Profile table, and
3. Skills table.

Remember that the User table is already built-in into the Django project. This is the table that assigns access privileges to all users according to whatever convention is decided to follow. However, we will need to connect into it for the Profile table.

Our users will have access to CRUD methods that allow them to add new projects, set up tags, and vote on their favourite projects without revealing what kind of vote they place on a particular project. The Profile table is responsible for assigning the user privileges to the site and keep more detailed information about the user since the default User table provided by Django is quite basic. We could also extend and assign those privileges into the User table, but for didactical purposes we will maintain both separate tables.

This is the model we will follow for the user table



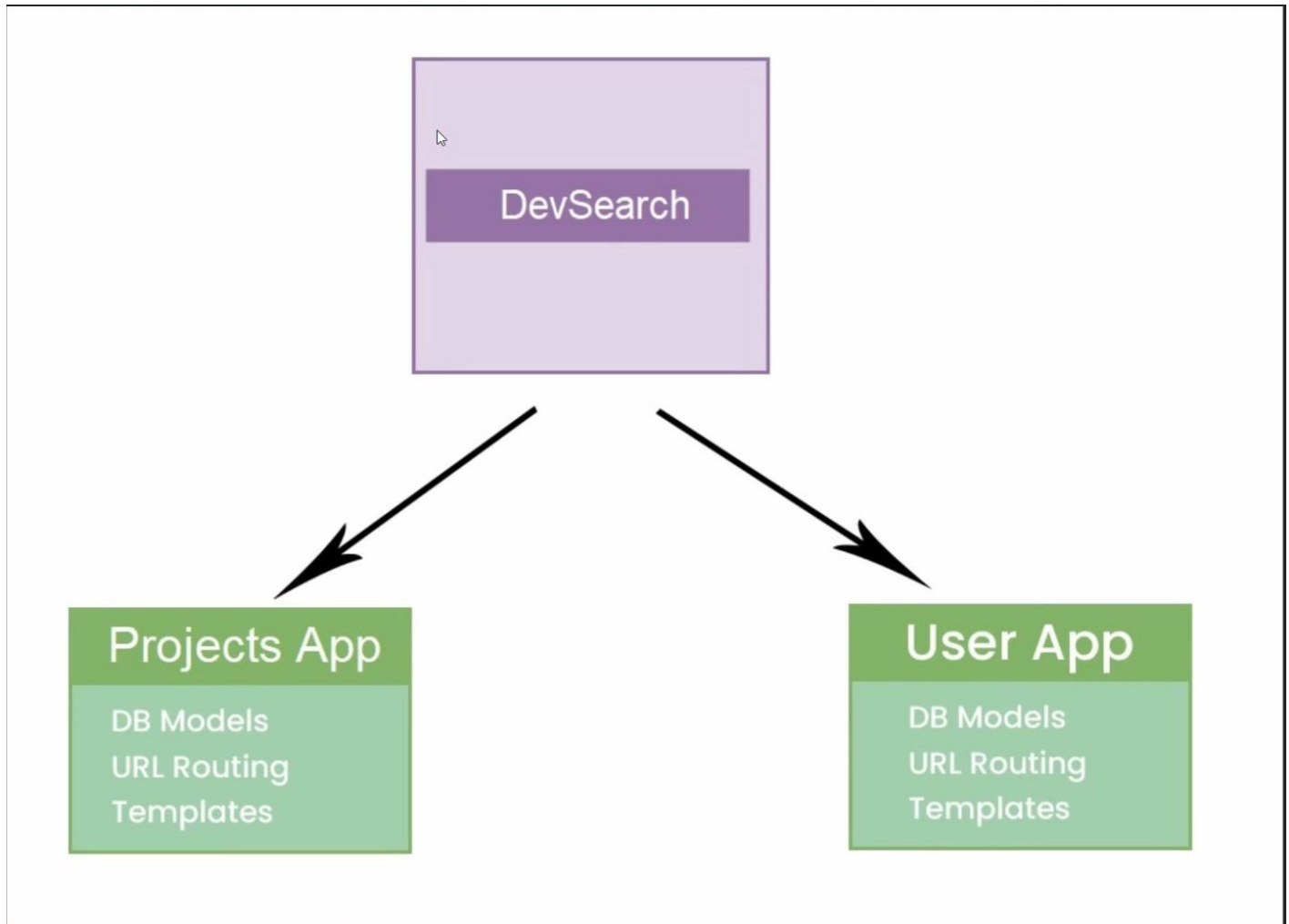
while the user table holds basic information, each user will have a "Profile" record which provides more detailed information about the user and its contact info.

What's important about separating a project into different projects is that the development process is done in incremental steps and during each step different functionalities can be tested without creating conflicts that are prevalent in monolithic projects. It allows flexibility since our models are separated with very specific responsibilities (This is one of most important SOLID principles in software development).

Starting Our New Django App

This is the basic structure of our Django project now. It's split into two basic apps:

1. Projects App, and
2. User App.



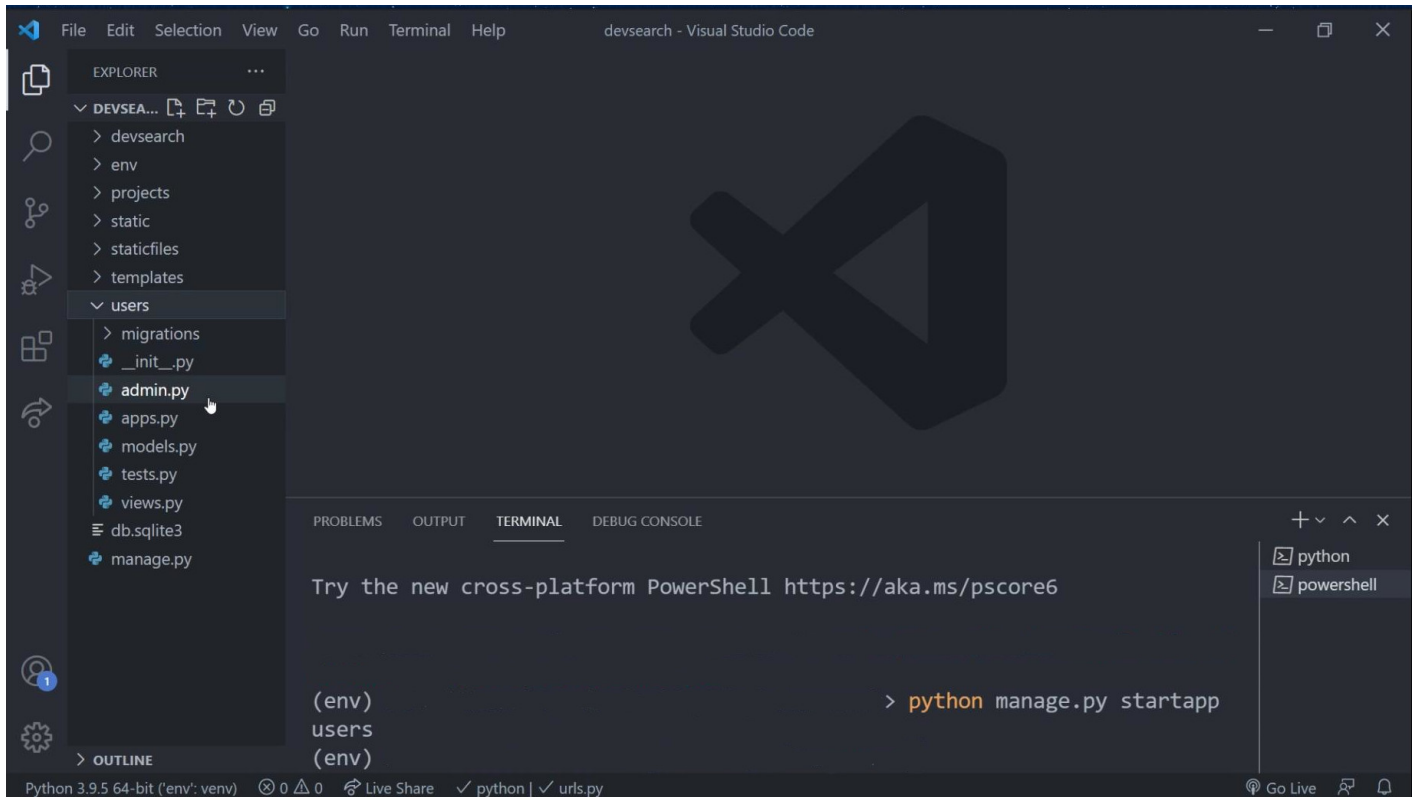
each app contains a DB model, URL Routing, and Page templates. Our concern is to set up all the application elements so they work in tandem.

Bulding the User App

With in your IDE, open the TERMINAL console at the bottom of the IDE and type the command:

```
python manage.py startapp users
```

This will create a new folder at the Django project root level:



If you open the newly created folder and the files, you'll find that they only contain the basic boiler plate for the users app and that the migrations folder is empty since we still haven't defined any model in the models.py file.

The first step is do the app connections and configuration to the Django project.

1. Register the new application within the */devsite/settings.py*. In the **INSTALLED_APPS** list add:

'users.apps.UsersConfig'

```

34 'django.contrib.admin',
35 'django.contrib.auth',
36 'django.contrib.contenttypes',
37 'django.contrib.sessions',
38 'django.contrib.messages',
39 'django.contrib.staticfiles',
40
41 'projects.apps.ProjectsConfig',
42 'users.apps.UsersConfig'
43
44
45 MIDDLEWARE = [

```

UsersConfig is a class inside the `/users/apps.py` file.

```

1 from django.apps import AppConfig
2
3
4 class UsersConfig(AppConfig):
5     default_auto_field = 'django.db.models.BigAutoField'
6     name = 'users'
7

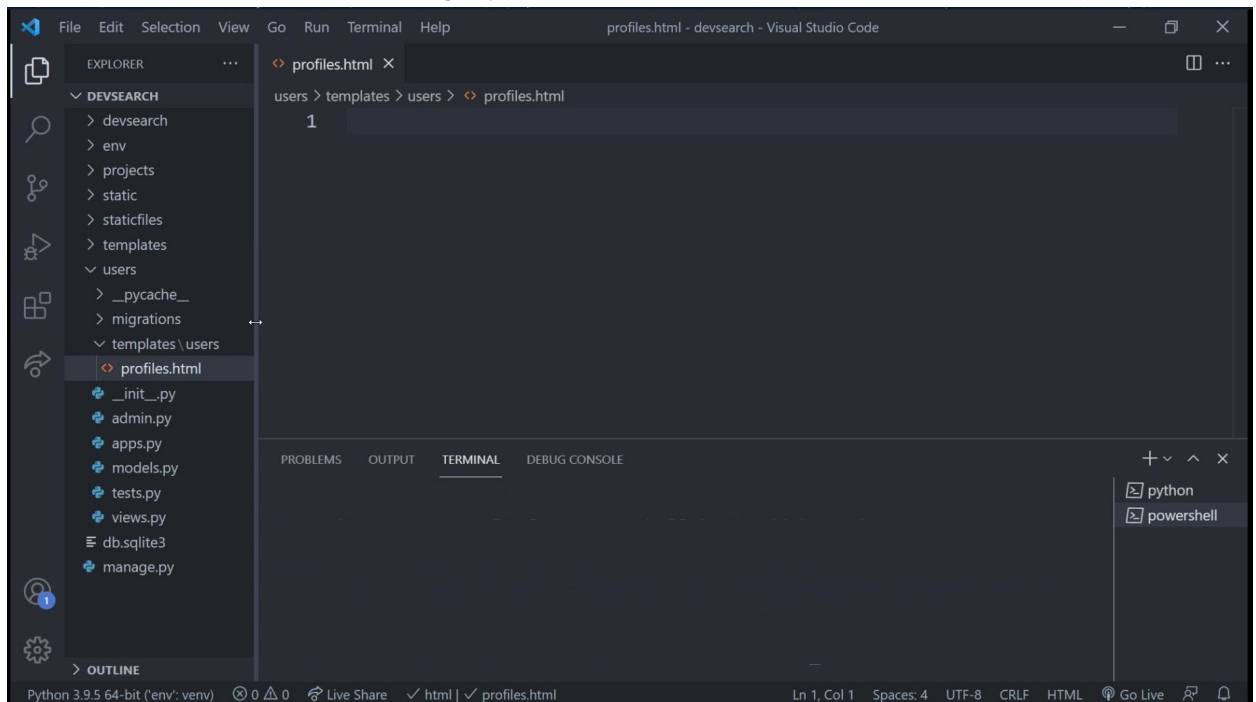
```

This step is vital because it informs to the Django project from where to run the model migrations, find page templates, and URL dynamic routings.

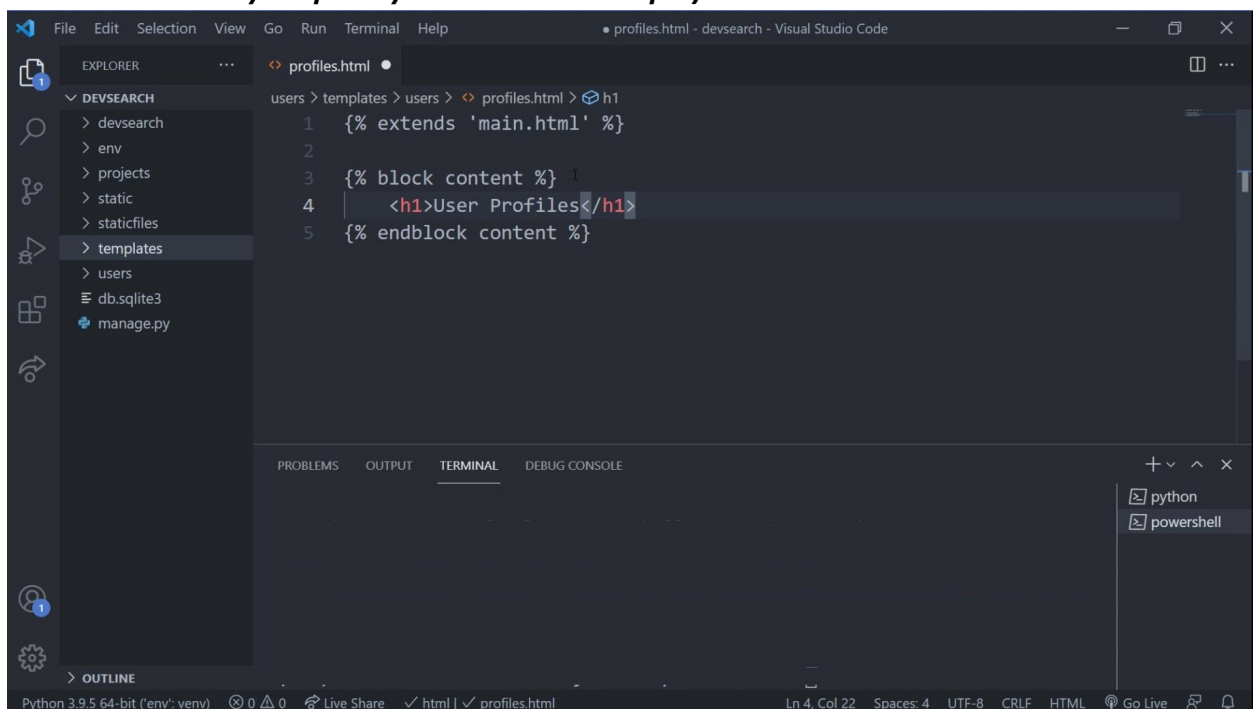
2. Create some templates to route and do their routing definitions.

- In your IDE create a new folder inside your users app: `/users/templates/users` very much like we did for the projects app and its templates.

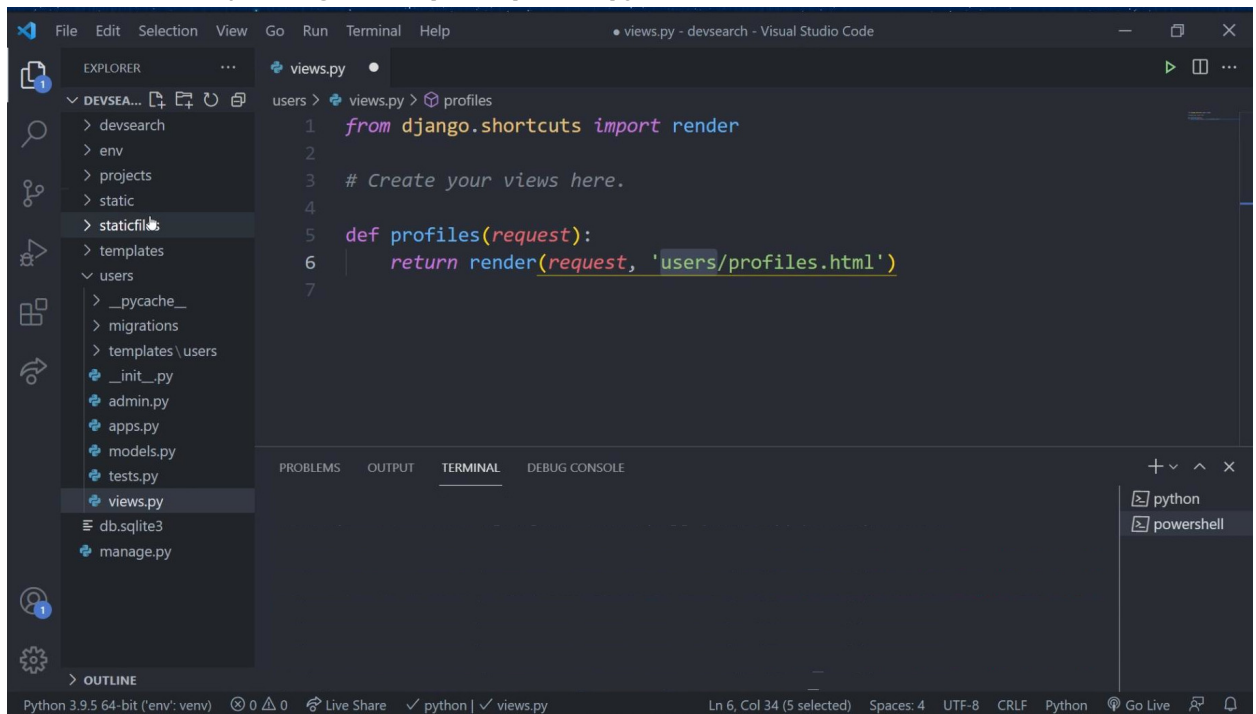
- Inside the folder, create a new file ***profiles.html***



- We will extend the ***/templates/main.html*** into our ***profiles.html***



- Create the corresponding view in **/users/views.py** with function-based views:

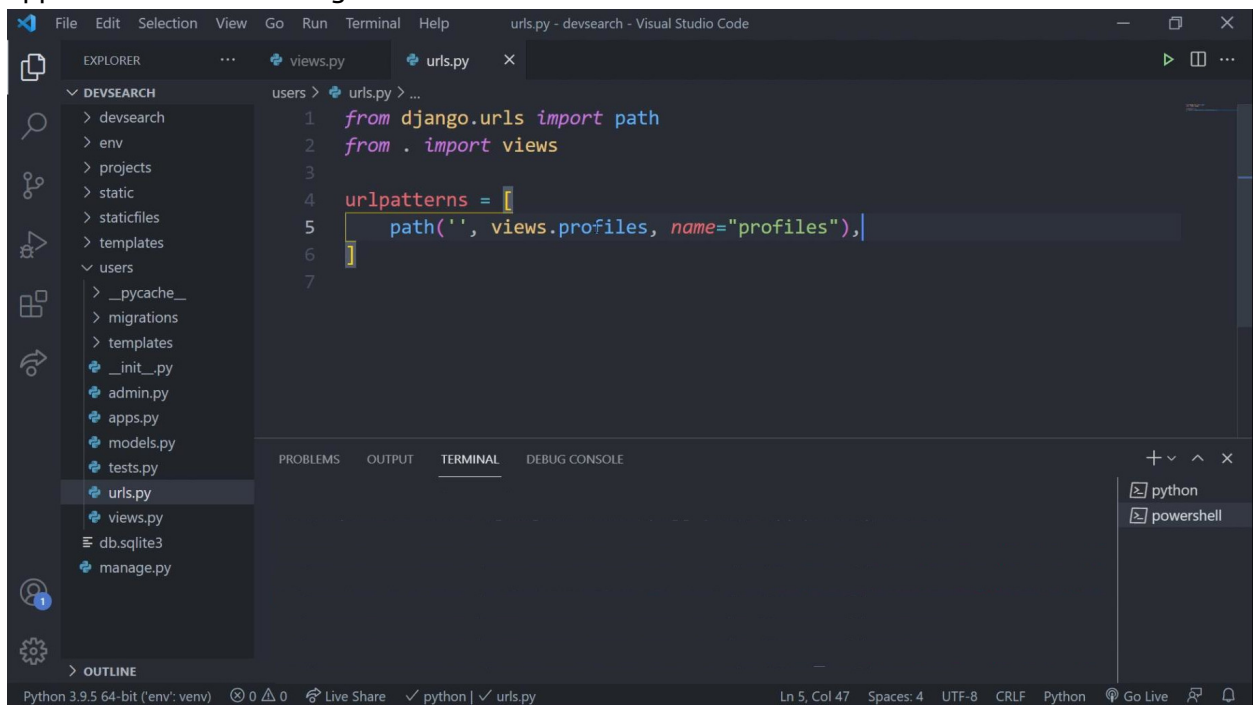


The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left. The file explorer is expanded to show the 'users' directory, which contains files like __init__.py, admin.py, apps.py, models.py, tests.py, and views.py. The 'views.py' file is selected and open in the editor. The code in the editor is as follows:

```
1 from django.shortcuts import render
2
3 # Create your views here.
4
5 def profiles(request):
6     return render(request, 'users/profiles.html')
7
```

The status bar at the bottom indicates the file is 'views.py' in the 'users' directory, using Python 3.9.5 64-bit ('env': venv).

- Create the **/users/urls.py**. By default, Django doesn't create this file, we have to add it to the application. Create it using the IDE with the IDE's tree view at the left of the IDE



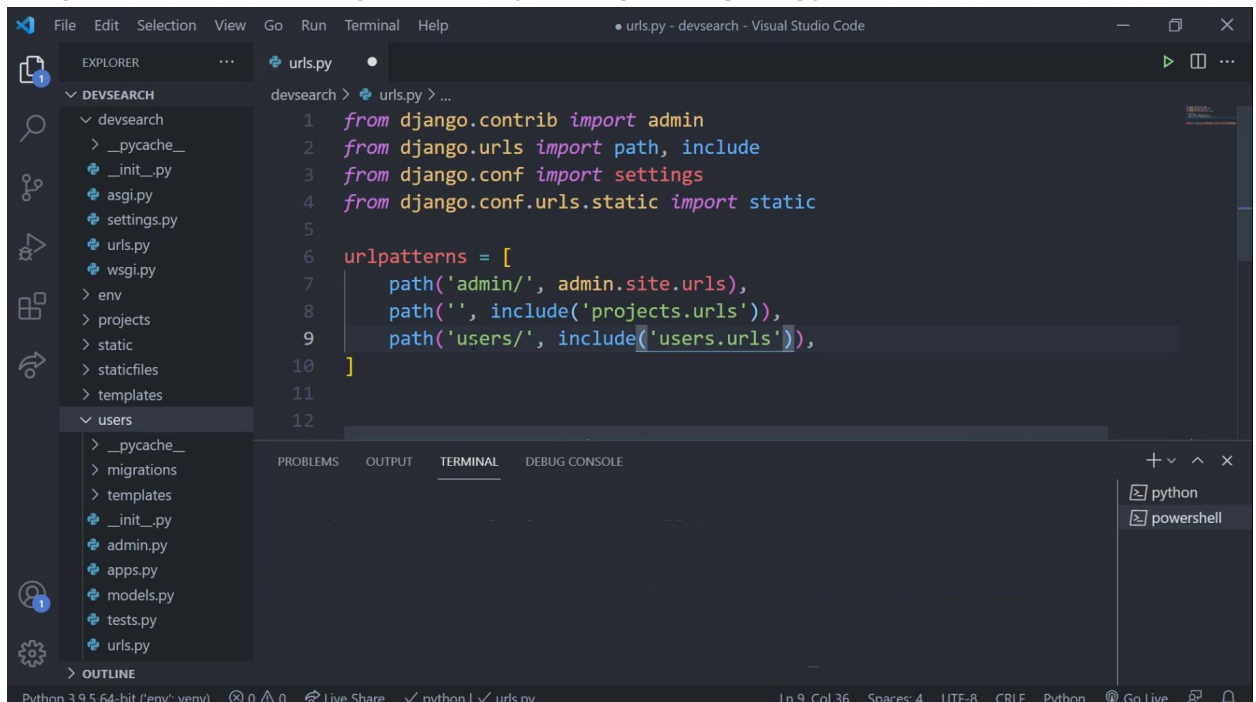
The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left. The file explorer is expanded to show the 'users' directory, which now includes 'urls.py' along with the other files. The 'urls.py' file is selected and open in the editor. The code in the editor is as follows:

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('', views.profiles, name="profiles"),
6 ]
7
```

The status bar at the bottom indicates the file is 'urls.py' in the 'users' directory, using Python 3.9.5 64-bit ('env': venv).

This will render the view using the template at the assign dynamic URL name='profiles'

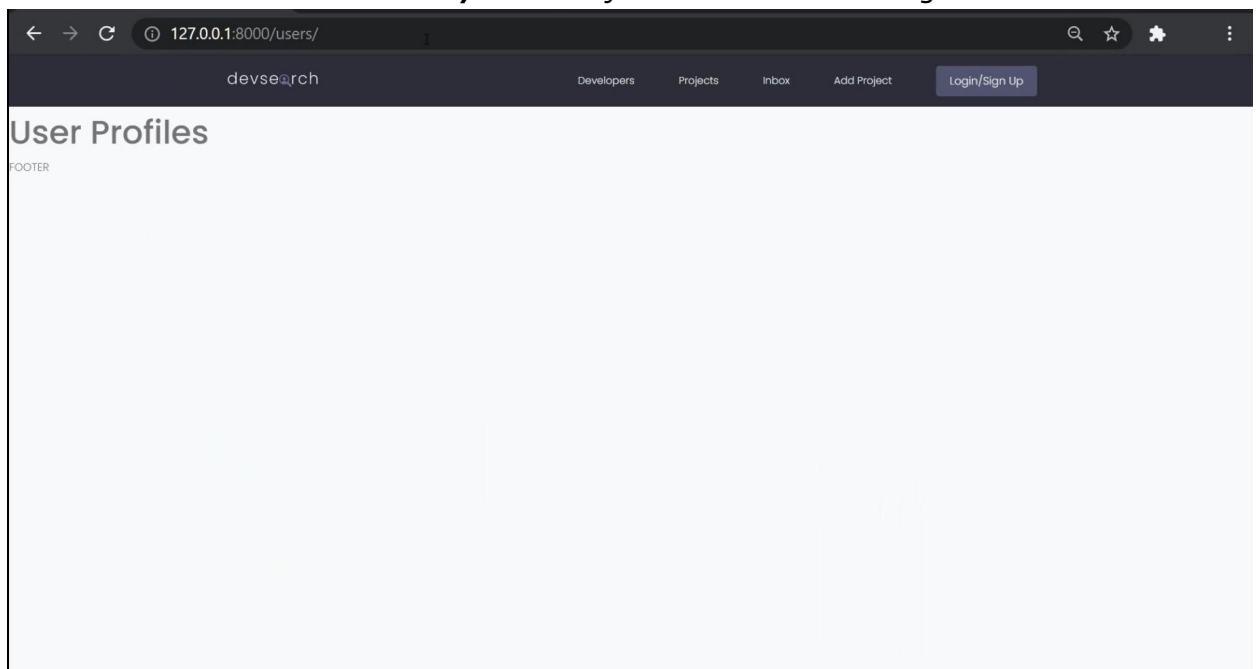
- Assign the view URL to the general URL paths at ***/devsite/urls.py*** file



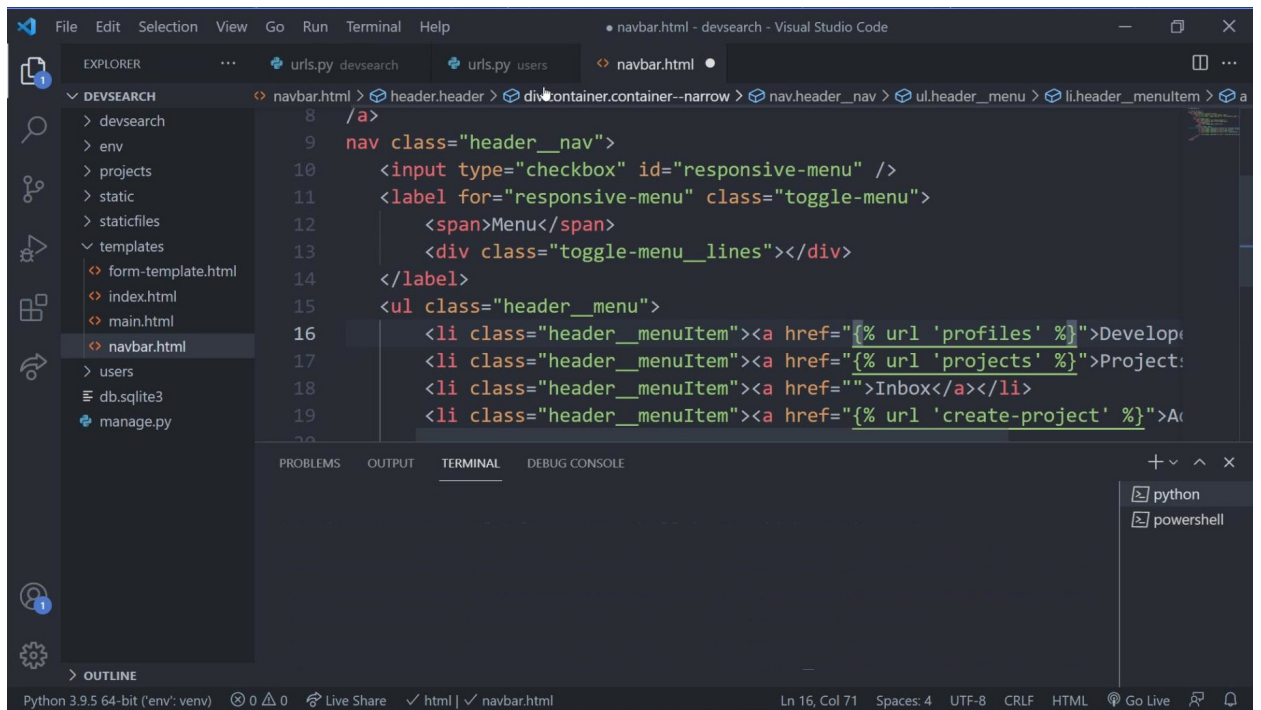
With all these steps we should have a functional page view that allow us to test the new application routing up to this point. Run your server:

```
python manage.py runserver
```

and access the route ***127.0.0.1:8000/users*** and you should see something similar to this:

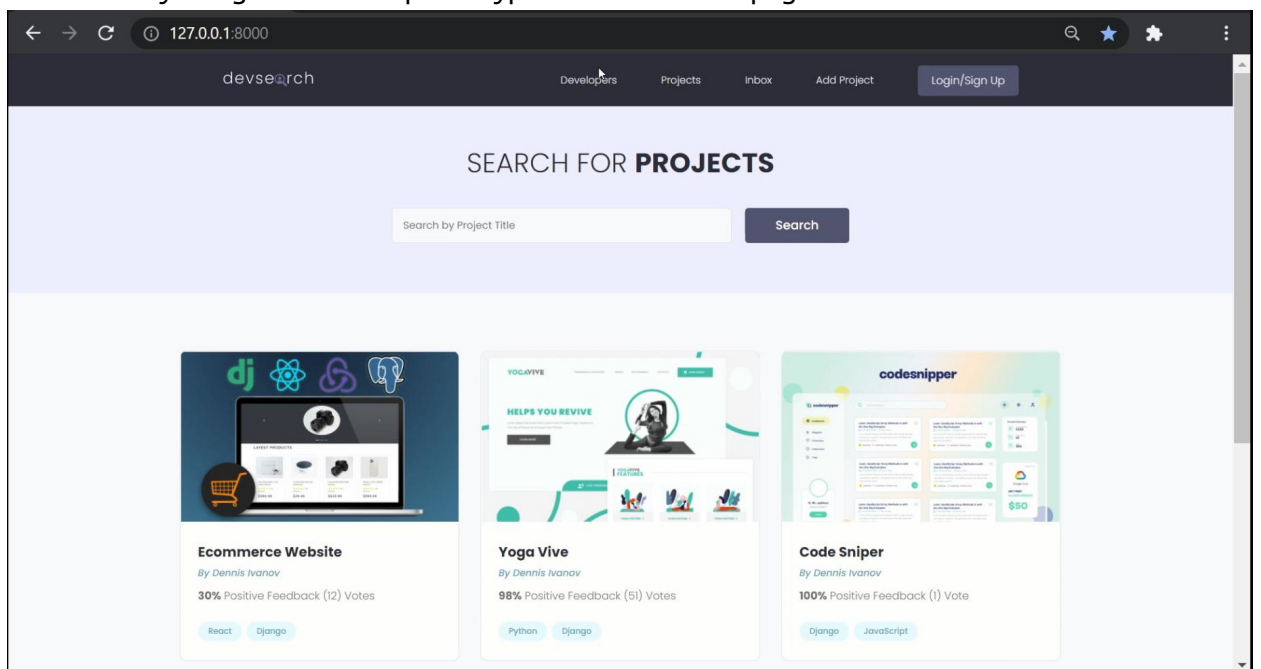


- Next, connect the user profile to the navbar. Open up the ***/templates/navbar.html*** and use the reference to the "Developers" anchor to include the profiles page:

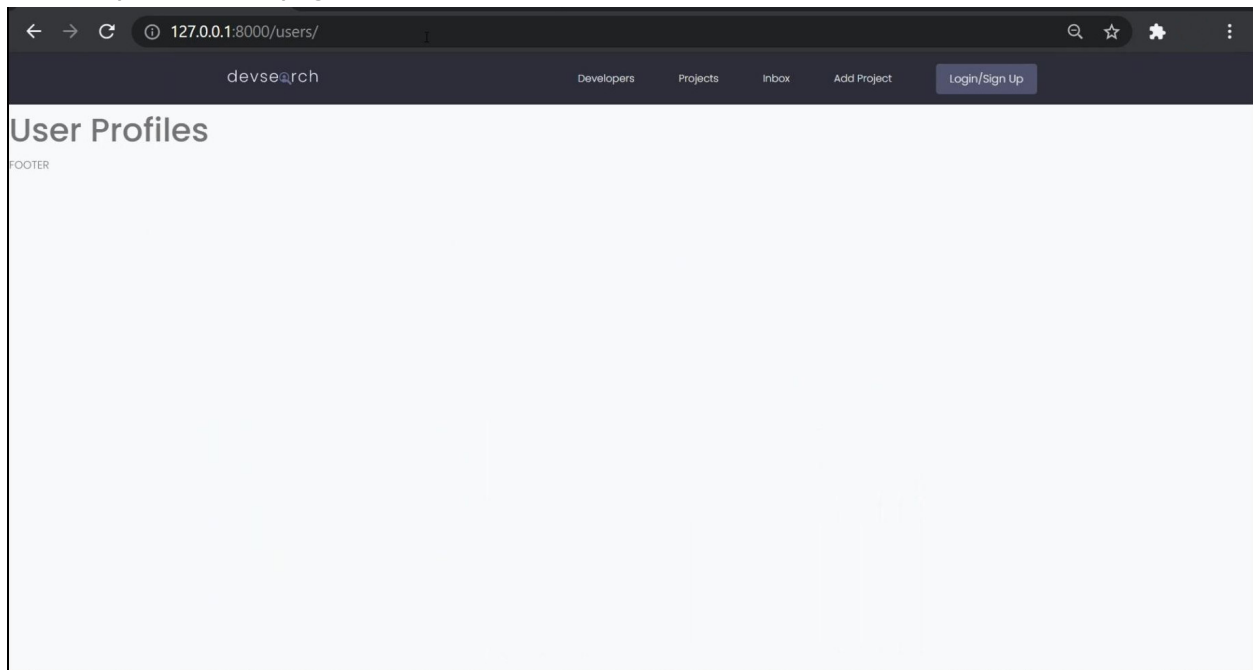


```
8 /a>
9 nav class="header__nav">
10 <input type="checkbox" id="responsive-menu" />
11 <label for="responsive-menu" class="toggle-menu">
12   <span>Menu</span>
13   <div class="toggle-menu__lines"></div>
14 </label>
15 <ul class="header__menu">
16   <li class="header__menuItem"><a href="{% url 'profiles' %}">Developers
17   <li class="header__menuItem"><a href="{% url 'projects' %}">Projects
18   <li class="header__menuItem"><a href="">Inbox</a></li>
19   <li class="header__menuItem"><a href="{% url 'create-project' %}">Add Project</li>
20 </ul>
21 </nav>
```

and test it by using the "Developers" hyperlink in the main page

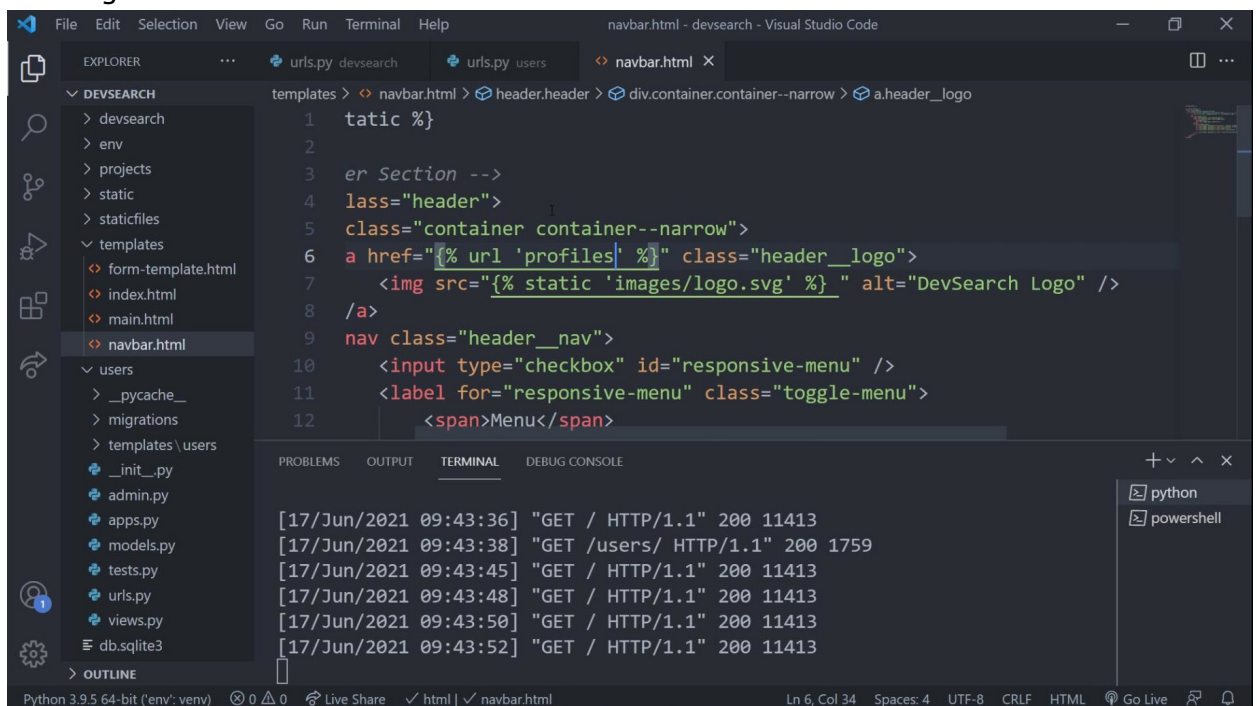


should open the new page



3. Re-structure the URL Routing. We're going to change the main page from the "Projects" App to the "Users" App. This requires to make some changes to the routing we assign to the site for the first time.

- As you may recall, there are several places that define "Projects" as the main page: The logo, and the "Projects" in the navbar point their links towards the "Projects" app as the main page. We have to change those definitions:



and modify the ***/devsite/urls.py*** file:

```

1  from django.contrib import admin
2  from django.urls import path, include
3  from django.conf import settings
4  from django.conf.urls.static import static
5
6  urlpatterns = [
7      path('admin/', admin.site.urls),
8      path('projects/', include('projects.urls')),
9      path('', include('users.urls')),
10 ]
11
12

```

```

[17/Jun/2021 09:43:52] "GET / HTTP/1.1" 200 11413
[17/Jun/2021 09:44:08] "GET / HTTP/1.1" 200 11419
[17/Jun/2021 09:44:09] "GET / HTTP/1.1" 200 11419
[17/Jun/2021 09:44:10] "GET /users/ HTTP/1.1" 200 1765
[17/Jun/2021 09:44:15] "GET / HTTP/1.1" 200 11419
[17/Jun/2021 09:44:17] "GET /users/ HTTP/1.1" 200 1765

```

later on we will modify the `/projects/project/<id>` route to adjust the length of the url route.

4. Create our models for the user, the message, and the skill tables.

Before we start, let me clarify a specific point about the default User table that Django uses for authentication. This table is highly integrated into Django's functionality. On specific terms, it means that if something breaks or goes wrong with the User table, the whole project becomes compromised. For this reason alone, we prefer to construct the Profile table in order to handle the user's profile since this approach doesn't compromise Django's functionality. The relationship between the User table and the Profile table is a One-to-One relationship.

- Open `/users/models.py` file to create our initial models. The code for the Profile model is as follows:

```

from django.db import models
from django.contrib.auth.models import User
import uuid
# Create your models here

class Profile(models.Model):
    user = models.OneToOneField(user, on_delete=models.CASCADE,
null=True, blank=True)
    name = models.CharField(max_length=200, blank=True, null=True)
    email = models.EmailField(max_length=500, blank=True,
null=True)
    username = models.CharField(max_length=200, blank=True,
null=True)
    short_intro = models.CharField(max_length=200, blank=True,
null=True)

```

```
bio = models.TextField(blank=True, null=True)
profile_picture = models.ImageField(blank=True, null=True,
upload_to='profiles/', default="profiles/user-default.png")  # **
social_github = models.CharField(max_length=200, blank=True,
null=True)
social_twitter = models.CharField(max_length=200, blank=True,
null=True)
social_linkedin = models.CharField(max_length=200, blank=True,
null=True)
social_youtube = models.CharField(max_length=200, blank=True,
null=True)
social_website = models.CharField(max_length=200, blank=True,
null=True)
created = models.DateTimeField(auto_now_add=True)
id = models.UUIDField(default=uuid.uuid4, unique=True,
primary_key=True, editable=False)

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

** We have to create a folder: ***/static/images/profiles*** since the argument ***upload_to*** in the ImageField allow us to define the folder in which the images are uploaded to. Previously, all our images (the thumbnails for the project template are uploaded into the ***/static/images*** folder).

Also, go to the resources folder you downloaded from the repository for this project and copy the ***user-default.png*** image into ***/static/images/profiles/*** folder, so the default image can be found by the model when no image is provided.

Run the necessary migrations:

```
python manager.py createmigrations
```

check that the migrations files at ***/users/migrations/*** are present. If not there's an issue or an error blocking the code from compiling.

If everything checks out, run the command:

```
python manager.py migrate
```

Register the new models into ***/users/admin.py*** file:

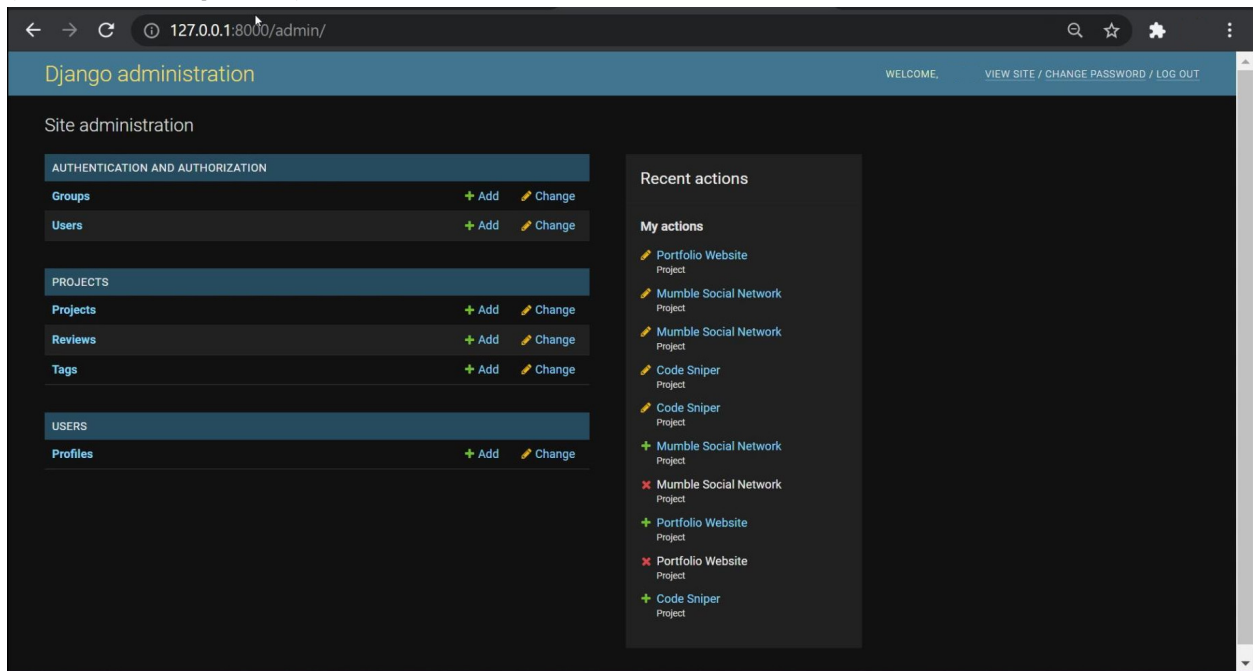
```
from django.contrib import admin

# Register your models here

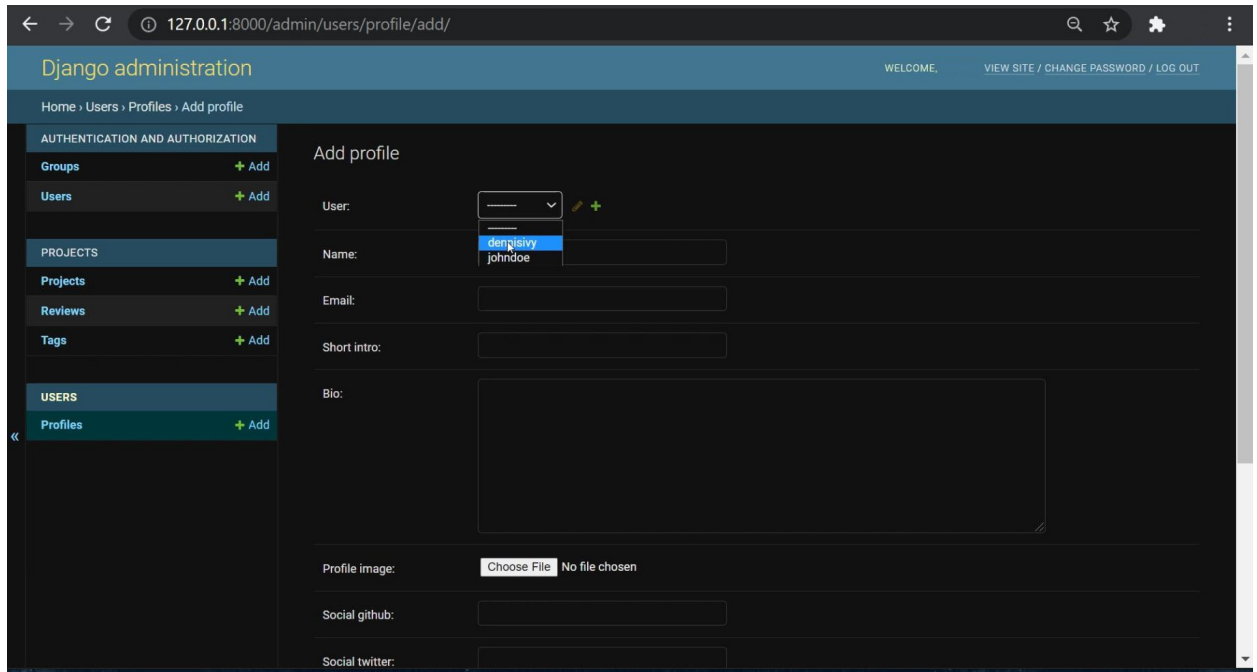
from .models import Profile

admin.site.register(Profile)
```

If everything is OK, then you should be able to check the new table at the admin panel
(**127.0.0.1:8000/admin**)



Test the new model by connecting to a user. Click on Add button for Profiles and a form will open in the adminstration panel:



and fill the requested fields. You should notice that adding a user is linked up to the User table and that's the reason it shows a drop-down menu. The relationship is explicitly set in the user field from our model.

Check that when you select a profile image for the user, the image is loaded into the ***/static/image/profile/*** folder.

5. We're going to connect the "Project" model to the user profile model on a ManyToOne relationship using a ForeignKey Field. Open the ***/projects/models.py*** file and modify the **class Project** as follows:

```
from django.db import models
import uuid
from users.models import Profile    # New
# Create your models here.

class Project(models.Model):
    owner = models.ForeignKey(Profile, null=True, blank=True,
on_delete=models.SET_NULL)
    ...
    ...
```

Run the migration:

```
python manage.py makemigration
python manage.py migrate
```


And as final step, we need to change the "Projects" template (`/templates/projects/projects.html`) to show the owner of the project:

```

32
33
34 project.id %}" class="project">
35 thumbnail" src="{{project.featured_image.url}}"
36 rail" />
37 ">
38 _title">{{project.title}}</h3>
39 ct_author" href="{% url 'project' project.id %}">By {{project.owner.name}}
40
41 -rating">
42 ont-weight: bold;">{{project.vote_ratio}}%</span> Positive
43 oject.vote_total}}) Vote{{project.vote_total|pluralize:"s"}}
44
45 t_tags">
46 project.tags.all %}

```

Running migrations:
 Applying users.0002_profile_username... OK
 Applying projects.0004_project_owner... OK
 (env) PS C:\Users\Dennis Ivy\Desktop\devsearch>

To show any changes, go to the admin panel and change some owners in for each project. You should see the changes in the Projects page (`127.0.0.1:8000/projects`)

