Workshop: Petstagram

We will be creating a complete Django project called **"Petstagram"** throughout this module. The project will cover the following **functionalities**: user **registration**, **login**, and **logout**; each user can **add pets** to their profile and **upload pet photos**; a user can **view all photos** of pets, open **details**, where can **like** and **comment** on a photo. Each user can **edit** and **delete their photos** and **pet** information.

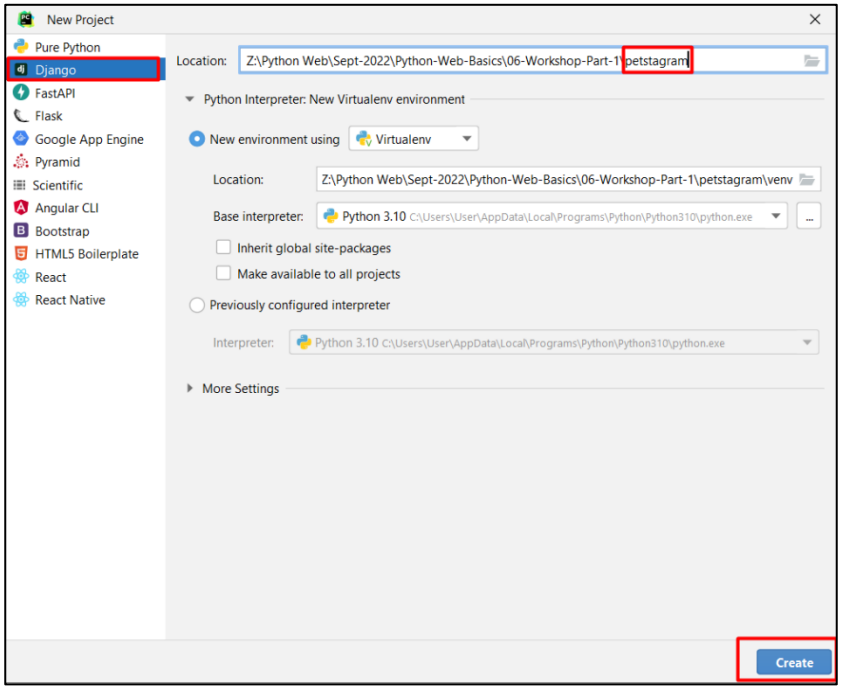
The full project description can be found in the **Workshop Description Document**.

You can directly dive into the app here: https://softuni-petstagram.azurewebsites.net/

1. Workshop - Part 1.1

Setup

Let us start by **creating the project**:

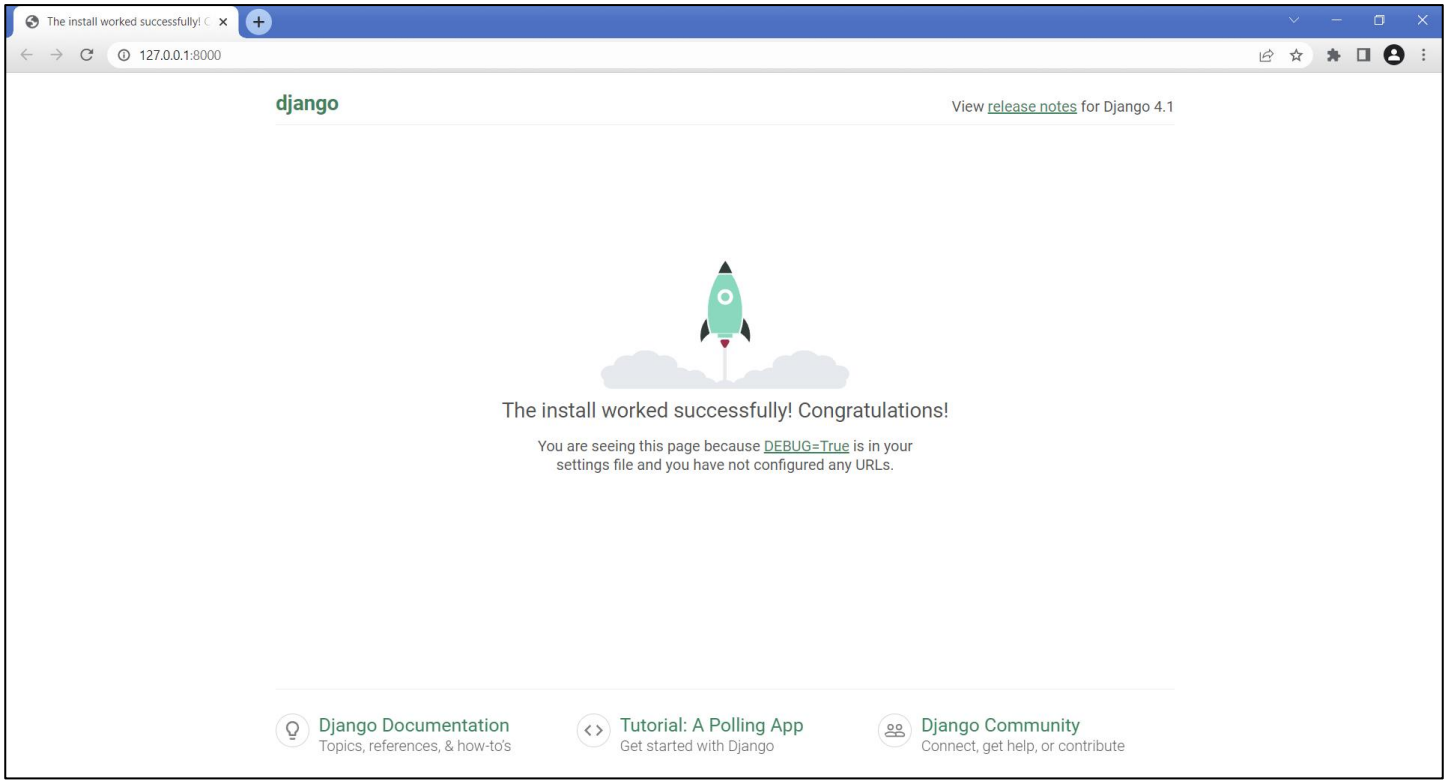


To check if everything works correctly, we can start the development server. One way to do it is to use the PyCharm Toolbar:

Another way to do this is to write the command **`python manage.py runserver`** in the Terminal and click on the provided link:

```
(venv) PS Z:\Python Web\Sept-2022\Python-Web-Basics\06-Workshop-Part-1\petstagram> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

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.
September 12, 2022 - 16:44:22
Django version 4.1.1, using settings 'petstagram.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

You should see the autogenerated Django "Congratulations" page:

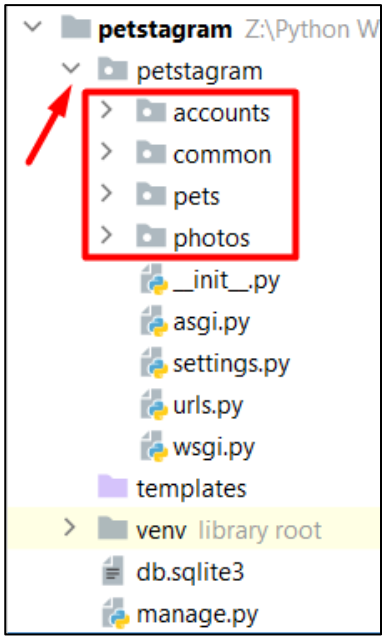


Creating the Apps

Now, let us create the **apps** we will work with. They are called **`'photos'`**, **`'pets'`**, **`'accounts'`**, and **`'common'`** and they will contain all **parts** of our **project**:

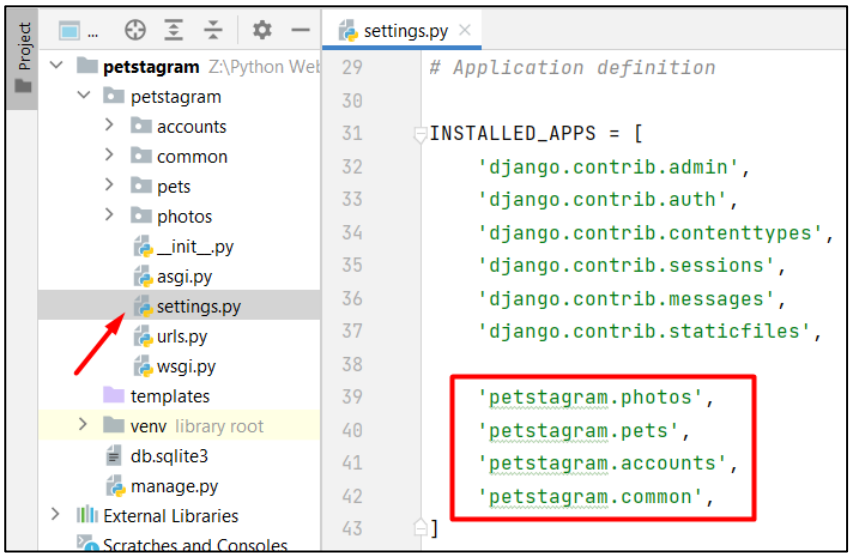
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

(venv) PS Z:\Python Web\Sept-2022\Python-Web-Basics\06-Workshop-Part-1\petstagram> django-admin startapp photos
(venv) PS Z:\Python Web\Sept-2022\Python-Web-Basics\06-Workshop-Part-1\petstagram> django-admin startapp pets
(venv) PS Z:\Python Web\Sept-2022\Python-Web-Basics\06-Workshop-Part-1\petstagram> django-admin startapp accounts
(venv) PS Z:\Python Web\Sept-2022\Python-Web-Basics\06-Workshop-Part-1\petstagram> django-admin startapp common
(venv) PS Z:\Python Web\Sept-2022\Python-Web-Basics\06-Workshop-Part-1\petstagram>
```

For clarification, **move the created apps inside** the project:



Configurations

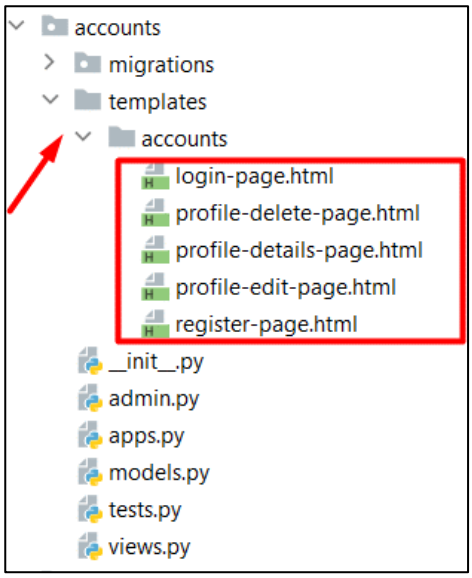
We need to **add the apps** we just created in the **`INSTALLED_APPS`** setting:



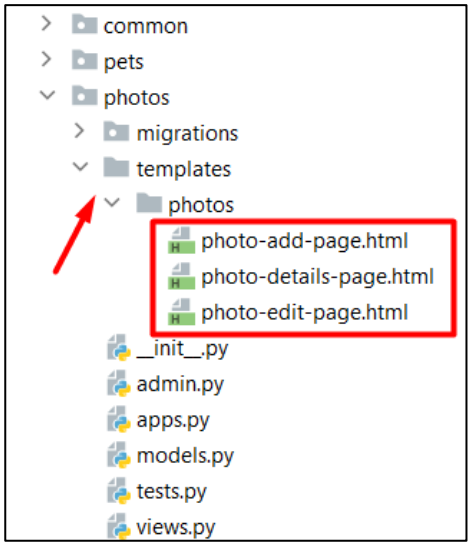
2. Workshop - Part 1.2

Adding the Templates

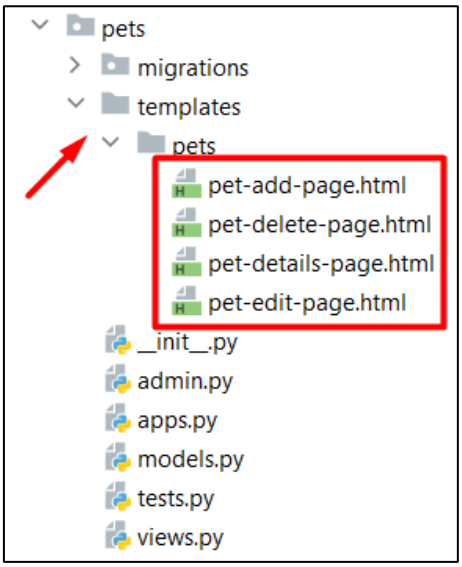
The next step is to **create** the **templates folder inside each app directory and add the given templates to it**. The templates associated with the account should be added to the **`accounts`** app:



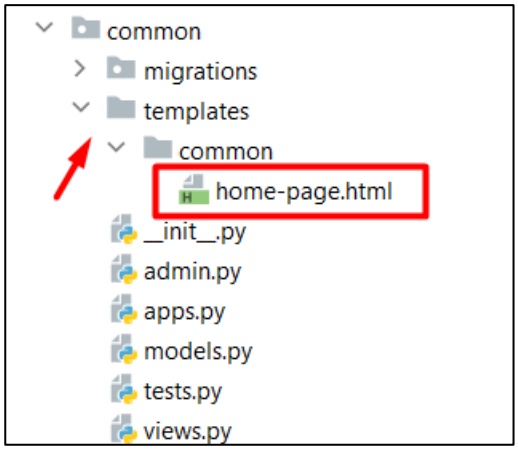
Respectively, the templates associated with the photos should be placed into the **`photos`** app:



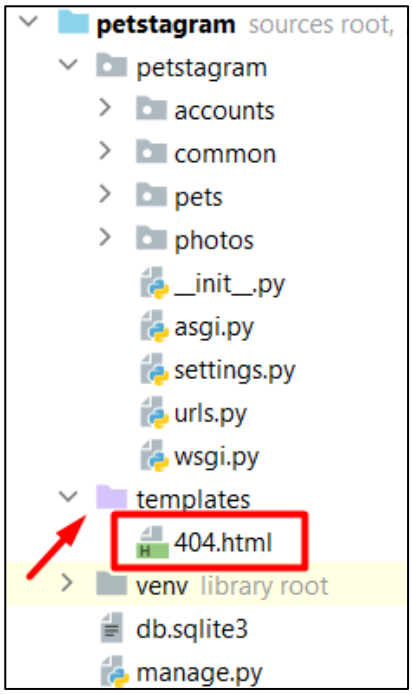
The templates associated with the pets should be placed into the **`pets`** app:



We can position the home-page template in the **`common`** app:

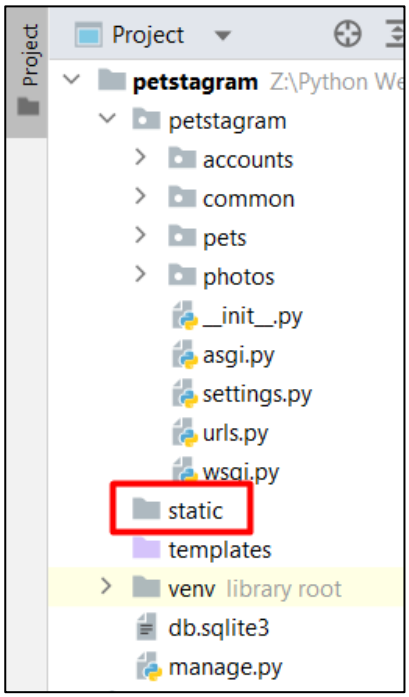


The only template left is the **`404.html`**. By default, it should be put inside the project **`templates`** directory on the **`manage.py`** level:

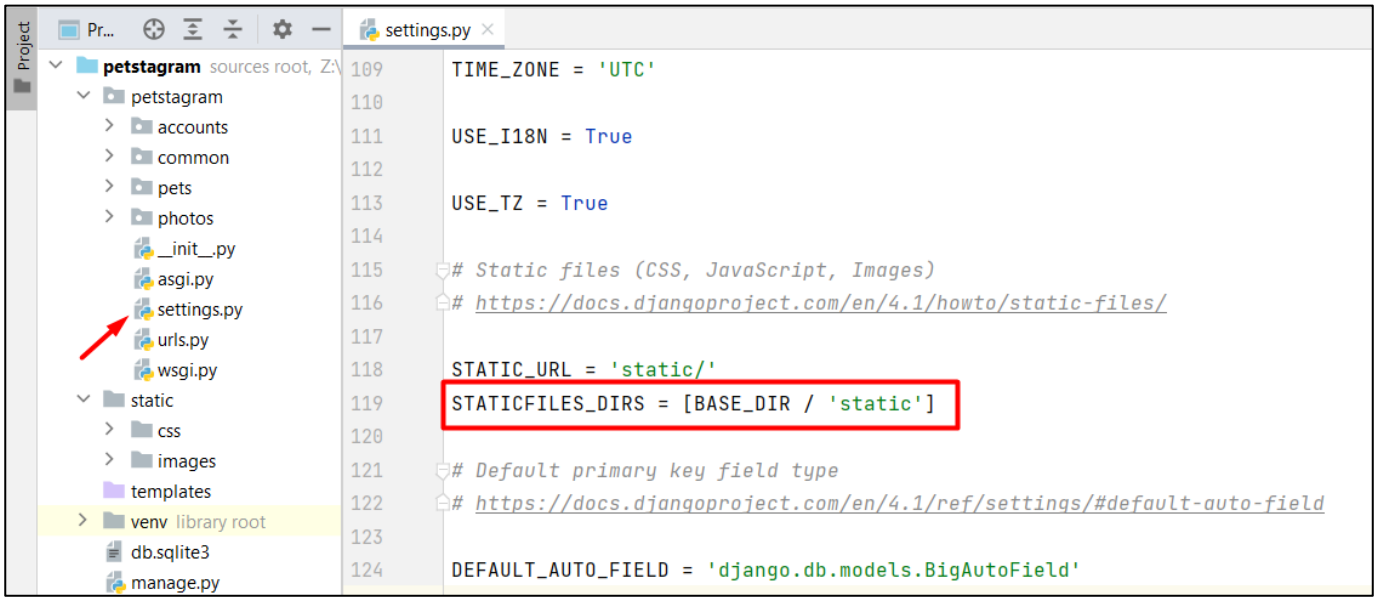


Adding the Static Files

Add a directory called **`static`** on the **`manage.py`** level:



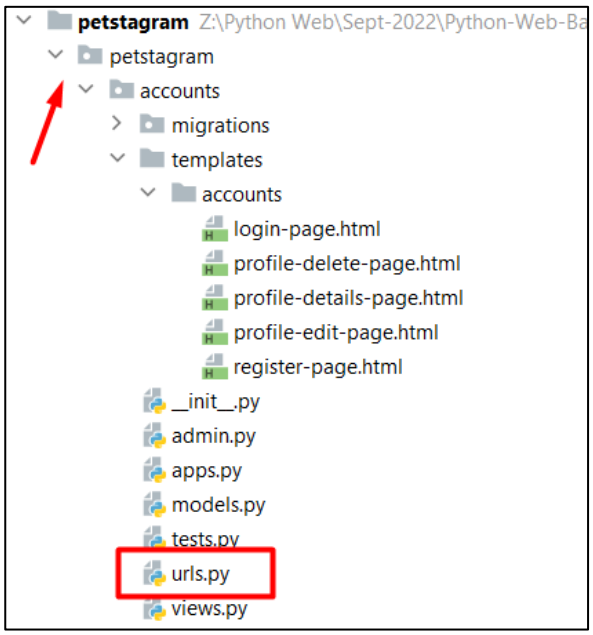
Add the provided folders ("css" and "images") to the directory. Next, Django should find the static files when loading web pages, so write the setting in the **`settings.py`** file:



Adding the URLs (paths)

We want to load each template in the browser using a concrete path - each app should load its templates.

To do that, we should add **`urls.py`** files in each app:



Then, we can start including them in the main project **`urls.py`** file. We should import the **`include()`** function from the Django **`urls`** module, then we can use the **`path()`** function to **construct a path**, which will **lead to each app** **`urlpatterns`**:

```python
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('petstagram.common.urls')),
    path('accounts/', include('petstagram.accounts.urls')),
    path('pets/', include('petstagram.pets.urls')),
    path('photos/', include('petstagram.photos.urls')),
]
```

Next, in **each app create an `urls.py`** file. Then, create paths connected to each view we will configure. Let us start by adding the following paths in the **`accounts`** app:

Registration Page: http://127.0.0.1:8000/accounts/register/

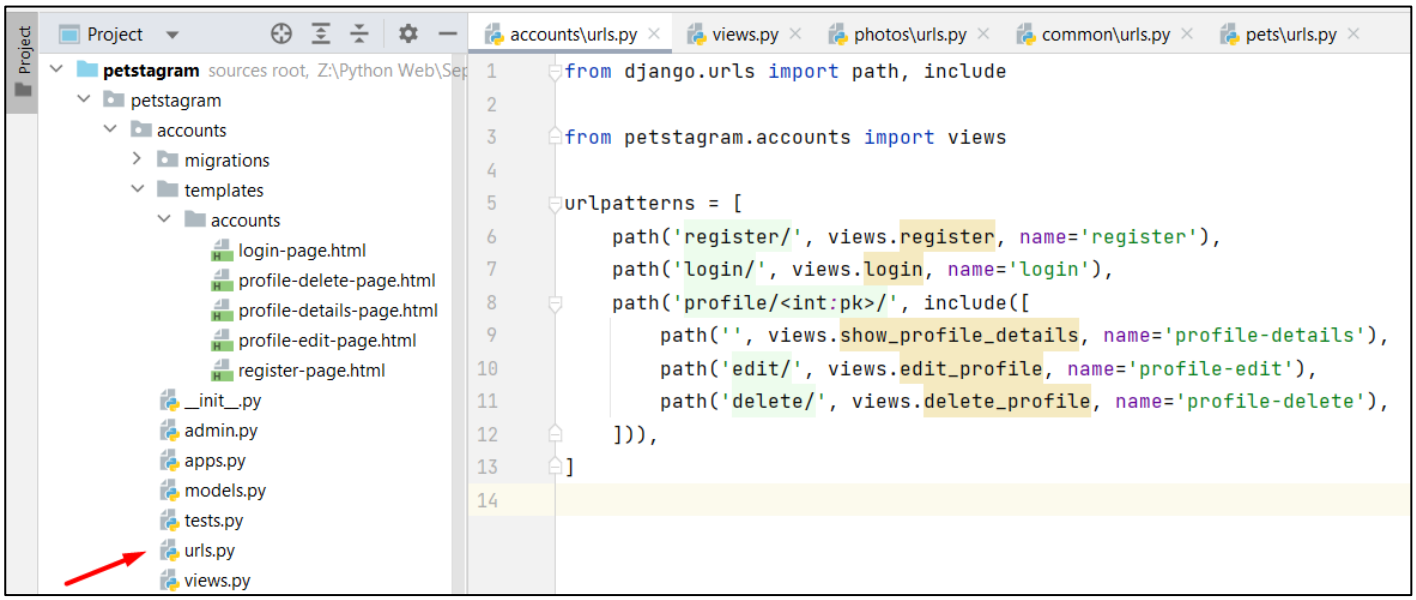
Login Page**:** http://127.0.0.1:8000/accounts/login/

Profile Details Page: http://127.0.0.1:8000/accounts/profile/<int:pk>/

Profile Edit Page: http://127.0.0.1:8000/accounts/profile/<int:pk>/edit/

Profile Delete Page: http://127.0.0.1:8000/accounts/profile/<int:pk>/delete/

In order to do that, we should create a **`urlspatterns`** list in the **`accounts/urls.py`** file:



The same configuration should be added to the other apps for the following URLs:

Home Page: http://127.0.0.1:8000/

Photo Add Page: http://127.0.0.1:8000/photos/add/

Photo Details Page: http://127.0.0.1:8000/photos/<int:pk>/

Photo Edit Page: http://127.0.0.1:8000/photos/<int:pk>/edit/

Pet Add Page: http://127.0.0.1:8000/pets/add/

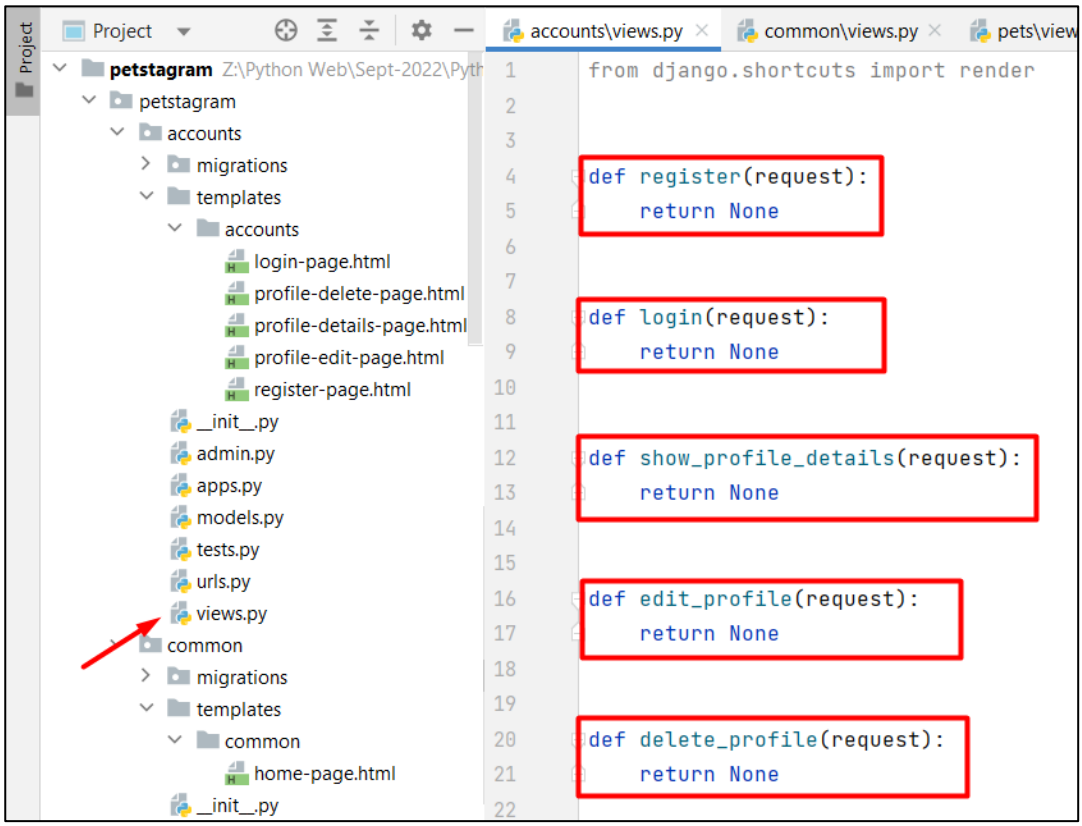
Pet Details Page: http://127.0.0.1:8000/pets/<str:username>/pet/<slug:pet_name>/

Pet Edit Page: http://127.0.0.1:8000/pets/<str:username>/pet/<slug:pet_name>/edit/

Pet Delete Page: http://127.0.0.1:8000/pets/<str:username>/pet/<slug:pet_name>/delete/

Adding Views

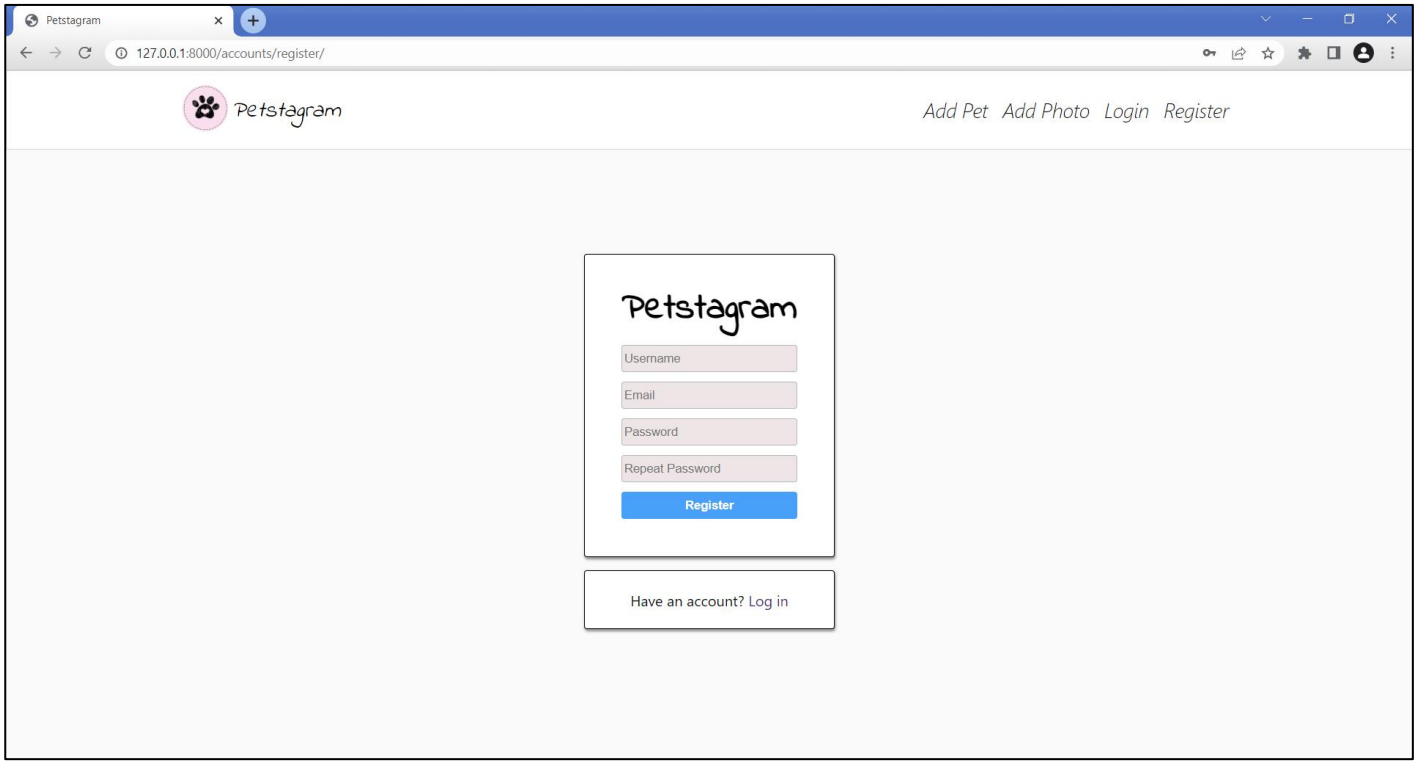
As you see, we have not created the views so far, and they appear to be missing in the **`urls.py`** file. Now, we will start adding them to the **`views.py`** file for each app. For example, we will use the names of the views created in the **`accounts/urls.py`** file to create the views in the **`accounts/views.py`** file:



Each view should receive a request and return a response. The response we want to return is a rendered HTML page we already added to our **`templates/accounts`** directory. Let us write a render function for each view to return:

The first parameter the **`render()`** function accepts is the given request. Next, we could pass the template we want to be shown when loading the specific path. In this case, when we load http://127.0.0.1:8000/accounts/register/ we

should see the register-page template:



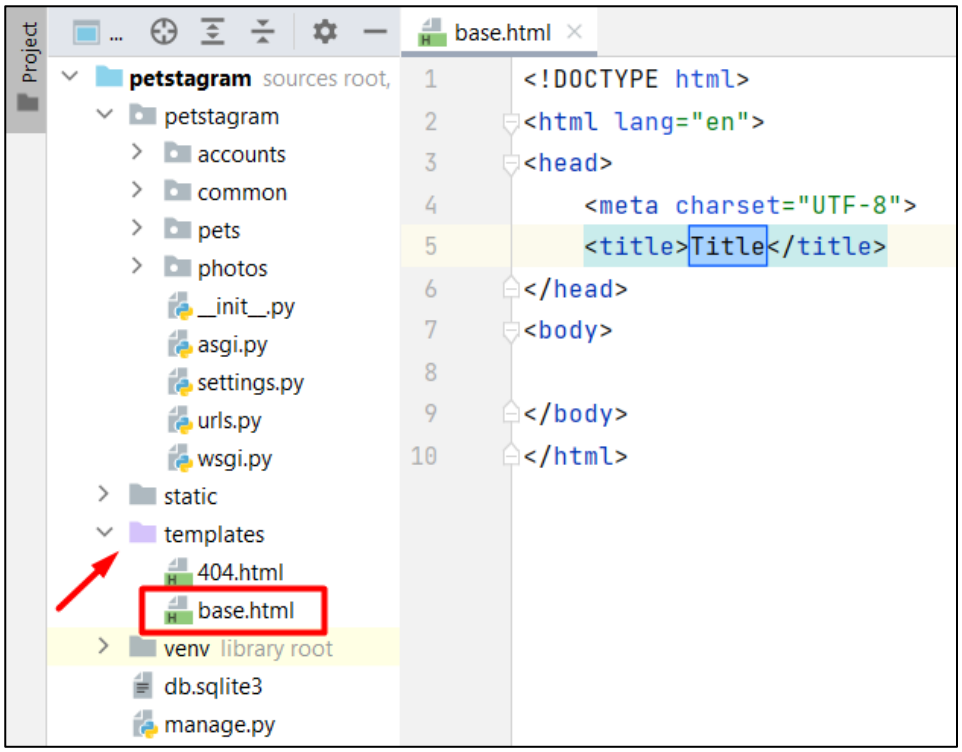
In the same way we can **create all views** in the project.

3. Workshop - Part 1.3

Creating Template Inheritance

If we look closely at each template, we can see that there are many common parts. The head, the header with the navigation bar, and the footer are the same for all templates. We can export them in a separate **`.html`** file in the project's **`template`** directory.

Let us create a **`base.html`** template in the **`templates`** directory on the **`manage.py`** level. We will position it there because the code is common for all apps:



Now, we can add **all common parts** that will structure the **base** template:

```
<!DOCTYPE html>
<html lang="en">
<!-- Starts Head Section -->
<head>
    <link rel="stylesheet"
          href="/static/css/styles.css">
    <link rel="icon" type="image/x-icon" href="/static/images/free-30-instagram-
stories-icons23_122570.png">
    <title>Petstagram</title>
</head>
<!-- End Head Section -->
<!-- Starts Body Section -->
<body>
<!-- Starts Header Section with Navigation Bar -->
<header>
    <nav class="navbar">
        <div class="container">
            <div class="logo">
                <!-- Link to Home Page -->
                <a href="#"><img width="50px" src="/static/images/free-30-instagram-
stories-icons23_122570.png" alt="img1"></a>
                <!-- Link to Home Page -->
                <a class="home" href="#"><i>Petstagram</i></a>
            </div>
            <div class="nav-links">
                <ul class="nav-group">
                    <li class="nav-item">
                        <!-- Link to Add Pet Page -->
                        <a href="#"><i>Add Pet</i></a>
                    </li>
                    <li class="nav-item">
                        <!-- Link to Add Photo Page -->
                        <a href="#"><i>Add Photo</i></a>
                    </li>
                    <li class="nav-item">
                        <!-- Link to Login Page -->
                        <a href="#"><i>Login</i></a>
                    </li>
                    <li class="nav-item">
                        <!-- Link to Register Page -->
                        <a href="#"><i>Register</i></a>
                    </li>
                    <li class="nav-item">
                        <!-- Link to Profile Page -->
                        <a href="#"><i>Profile</i></a>
                    </li>
                </ul>
            </div>
        </div>
    </nav>
</header>
<!-- End Header Section with Navigation Bar -->
<!-- Starts Main Section -->
<main>
</main>
<!-- End Main Section -->
...
```

```
...
<!-- Start Footer Section -->
<div class="footer">
      <span class="footer-section">© 2022 SOFTUNI WORKSHOP FOR PYTHON WEB MODULE</span>
</div>
<!-- End Footer Section -->
</body>
</html>
```

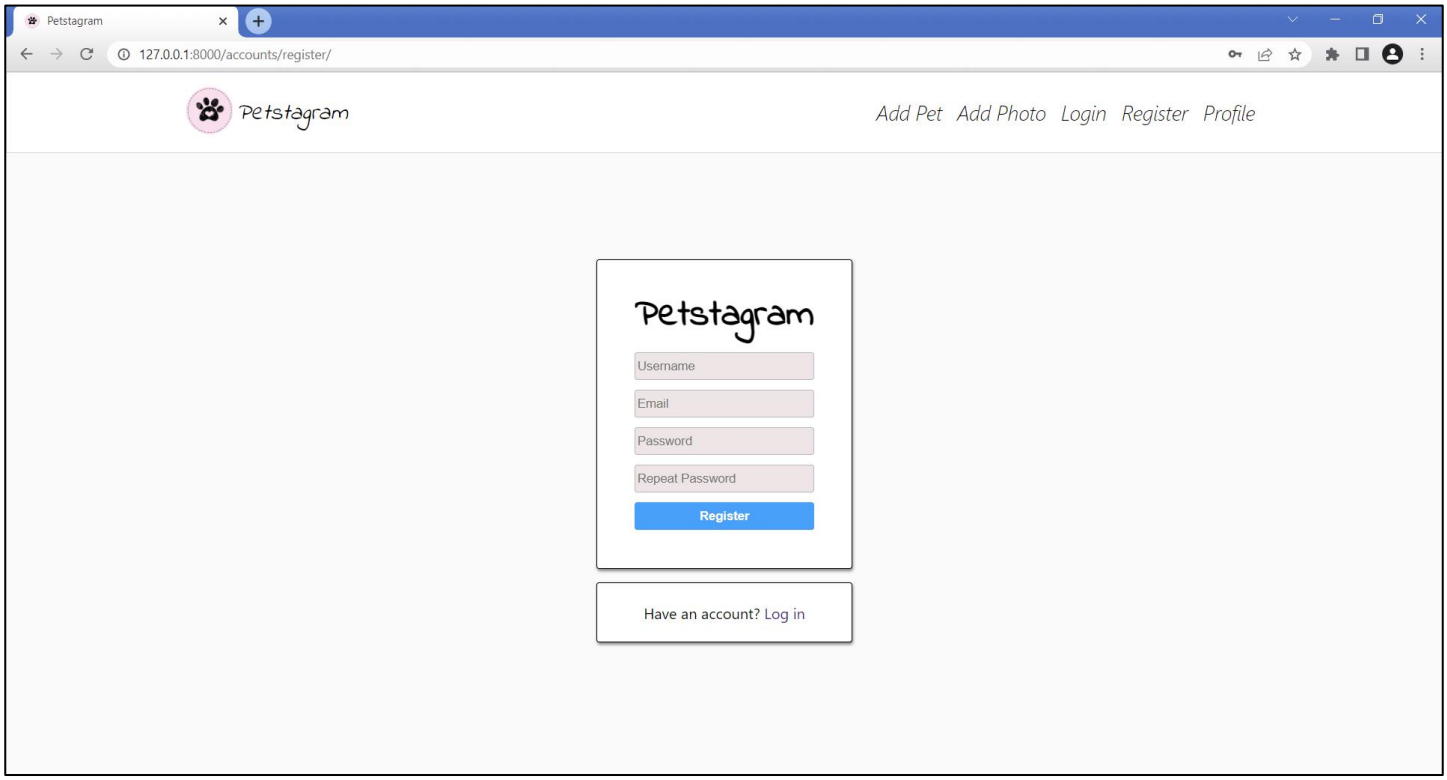
Next, we should connect the **base** template with all the other templates. In the **base** template **mark the place where the code should be extended** - it is only the main part:

```
<!DOCTYPE html>
<html lang="en">
...
    <!-- Starts Main Section -->
    <main>
    {% block content %}
    {% endblock %}
    </main>
    <!-- End Main Section -->
...
</html>
```

We should **delete the common part of each template** and **add the appropriate tags**. For example, the **register-page** template should look like this:

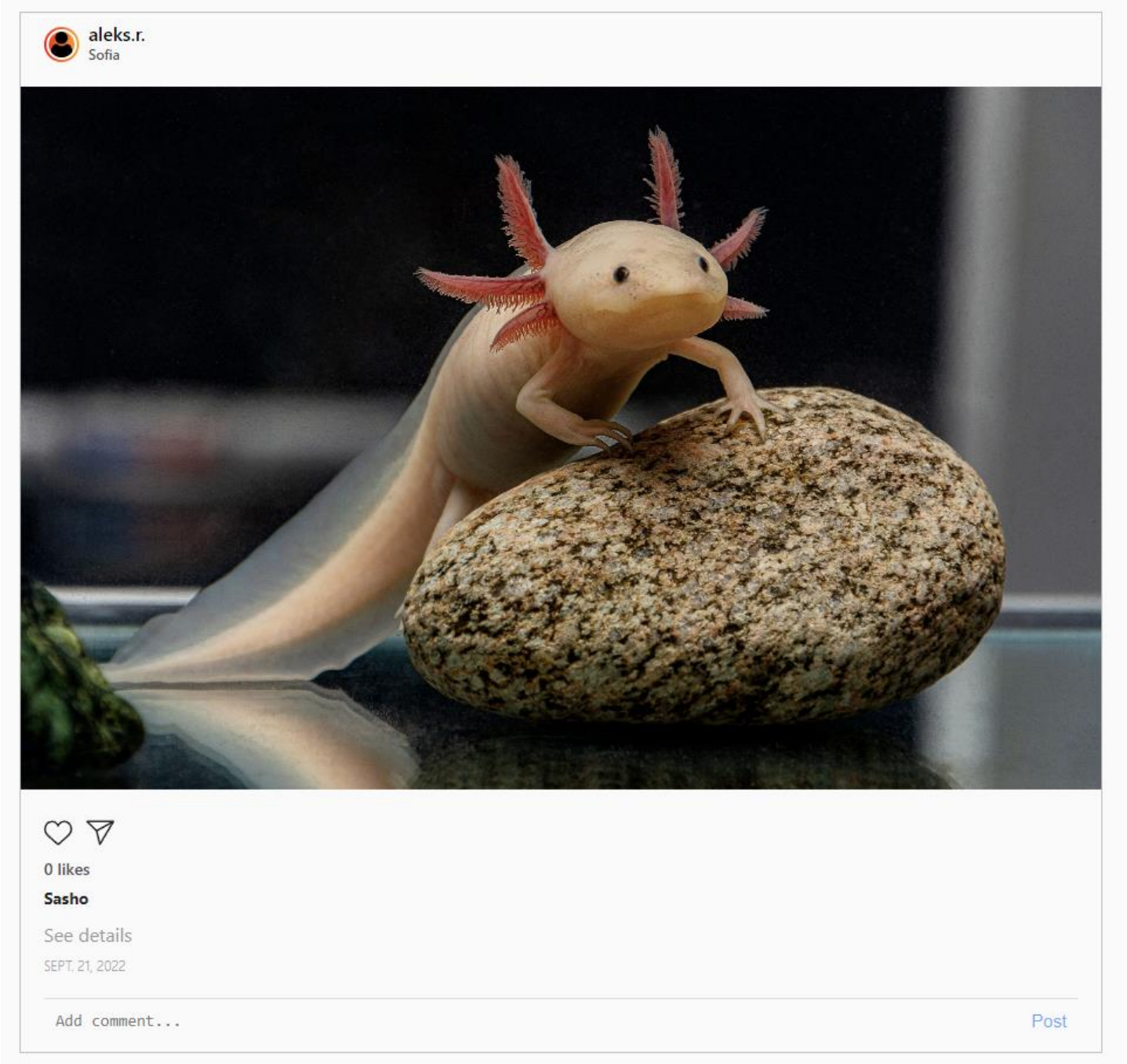
```
{% extends 'base.html' %}

{% block content %}

        <!-- Start Register Section-->
        <div class="login-register-div">
            <div class="login-register-box">
                <h1>Petstagram</h1>

                <!-- Start Register Form-->
                <form action="">
                    <input type="text" placeholder="Username"><br>
                    <input type="email" placeholder="Email"><br>
                    <input type="password" placeholder="Password"><br>
                    <input type="password" placeholder="Repeat Password"><br>

                    <!-- Register Button-->
                    <button type="submit">Register</button>
                </form>
                <!-- End Register Form -->

            </div>
            <div class="second-option"><p>Have an account?

                <!-- Link to Login Page-->
                <a href="#">Log in</a></p>
            </div>
        </div>
        <!-- End Register Section-->

{% endblock %}
```

If you **start the development server** and **load the register page**, you should see **no difference** between the style of the page now and the one before:

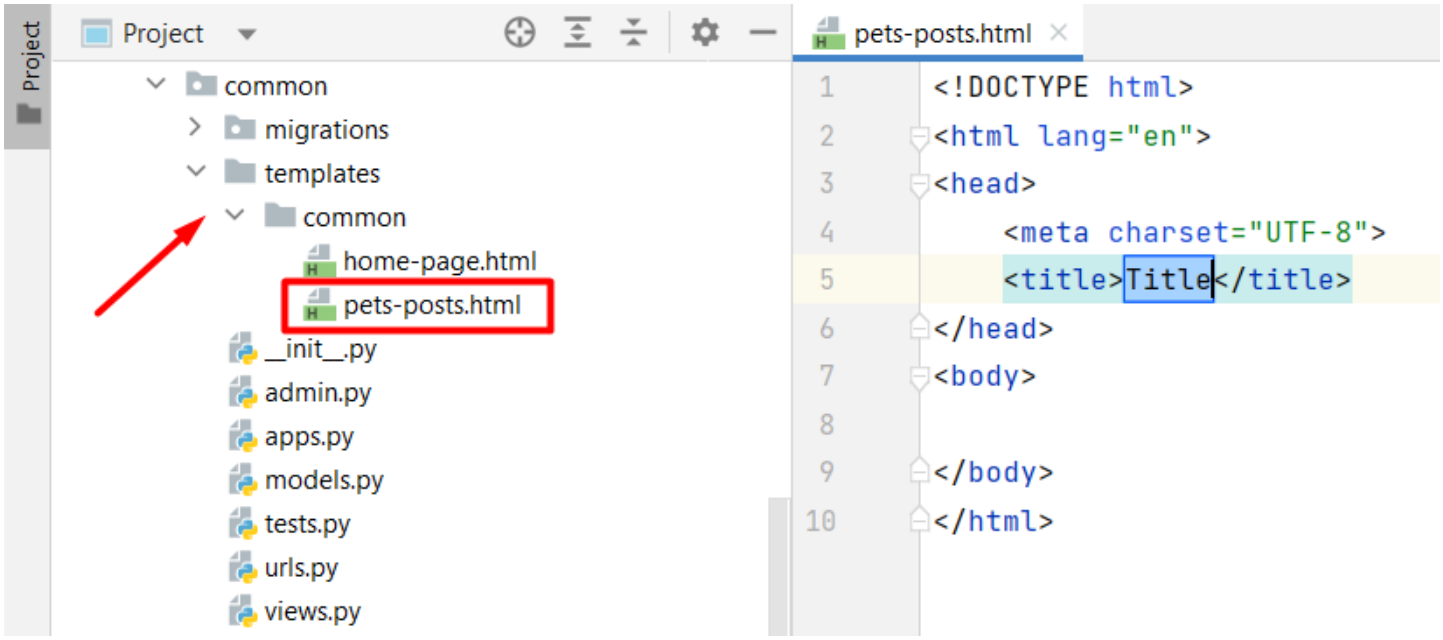


Separating Common Parts

We can see that there are **common parts for couple of pages**. Let us start by checking the templates **`home-page.html`** and **`pet-details-page.html`** - both templates have pet photos (posts) part:



We can move it to separate template and then include the template in **`home-page.html`** and **`pet-details-page.html`**. So, let us **create a new template** called **`pets-photos.html`** in the **common** app:



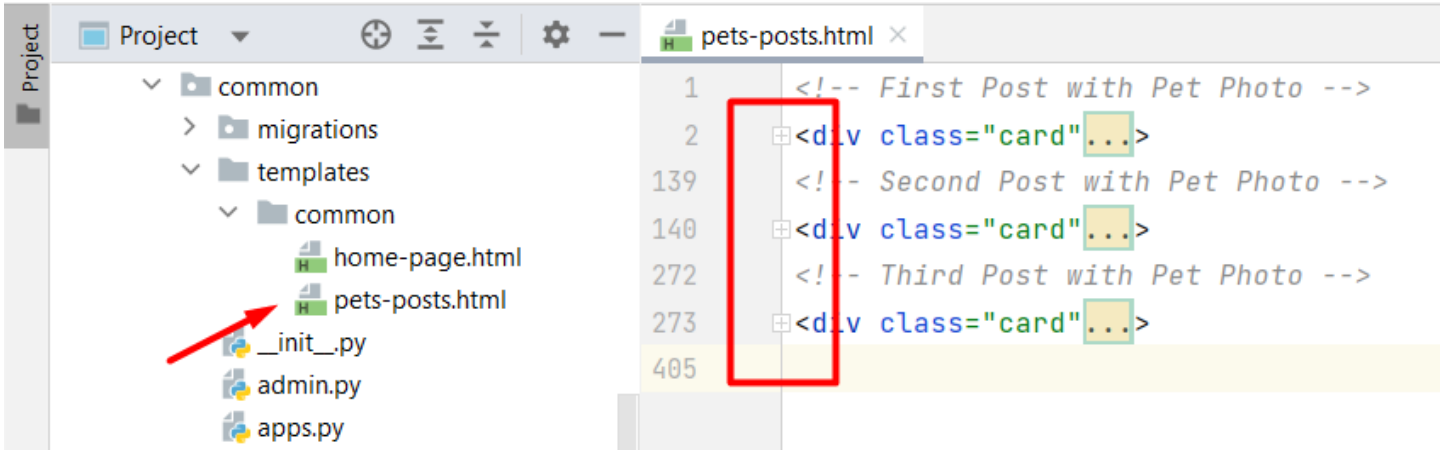
Then, **move all the posts with photos** to the **`pets-posts.html`** and include the template using the **`include`** template tag in the **`home-page.html`**:

```
{% extends 'base.html' %}

<!-- Main Code -->
{% block content %}
    <div class="container">
        <div class="col-9">

            <!-- Searchbar -->
            <form class="searchbar" method="post">
                <input type="text" placeholder="Search by pet name...">
                <button>
                    <img src="/static/images/search.png" height="18" alt="img2">
                </button>
            </form>

            {% include 'common/pets-posts.html' %}

        </div>
    </div>
{% endblock %}
```

Next, delete the pets' photos in the **`pet-details-page.html`** template and include the **`pets-post.html` template**:

```
{% extends 'base.html' %}
{% block content %}
    <div class="pet-profile">
        <div class="profile">
            <div class="profile-data">
                <div class="profile_img">
                    <div class="image">
                        <img src="https://encrypted-
tbn0.gstatic.com/images?q=tbn:ANd9GcTgoIHq82qLYnvomfz5ZkW5CLQ8VNxCyI-hUw&usqp=CAU"
                             alt="img8">
                    </div>
                </div>
                <div class="personal">
                    <div class="edit">
                        <p>Sasho</p>
                        <a href="">
                            <img class="edit-img" src="/static/images/edit-pen-icon-
6.jpg" alt="edit button">
                        </a>
                        <a href="">
                            <img class="bin-img" src="/static/images/icon-remove-
22.jpg" alt="bin button">
                        </a>
                    </div>
                    <div class="data">
                        <span>4</span>
                        <p>photos</p>
                    </div>
                </div>
            </div>
        </div>
        <div class="pet-posts">
            <!-- Start if pet photos -->
            {% include 'common/pets-posts.html' %}
            <!-- End if pet photos -->

            <!-- Start if not pet photos -->
            <img class="no-posts" src="/static/images/no_posts.png" alt="no posts
image">
            <!-- End if not pet photos -->
        </div>
    </div>

{% endblock %}
```

One more thing - you see that there are **3 posts with pet pictures**, so we can **delete two of them** and leave just one that we will work with in the next sessions. **Hint**: it is easy to see the blocks of code when we **collapse the code sections**:



4. Workshop - Part 1.4

Adding Static Files in Templates

Django uses special **template tag** to tell the template engine to **use the files from the static folder**. In each template where we will use static files this tag should be added. In the **base** template it will look like this:

```
{% load static %}

<!DOCTYPE html>
<html lang="en">

<!-- Head -->
<head>
    <link rel="stylesheet" href="{% static 'css/styles.css' %}">
    <link rel="icon" type="image/x-icon" href="{% static 'images/free-30-instagram-
stories-icons23_122570.png' %}">
    <title>Petstagram</title>
</head>

<!-- Body -->
<body>

    <!-- Header -->
    <header>
        <nav class="navbar">
            <div class="container">
                <div class="logo">
                    <a href="#"><img width="50px" src="{% static 'images/free-30-
instagram-stories-icons23_122570.png' %}" alt="img1"></a>

...
```

Adding Hyperlinks in Templates

In the base template is positioned the **navigation bar**. It navigates the users to different parts of our app. To be fully functional, **hyperlinks should be added** to it. Using Django Template Language, it is not a difficult task to be done. We

will use the **url template tag** and map it to the view names that we wrote in the **`urlpatterns`** list:

```html
base.html
<!-- Header -->
<header>
    <nav class="navbar">
        <div class="container">
            <div class="logo">
                <a href="#">
                    <img width="50px" src="{% static 'images/free-30-instagram-stories-icons23_122570.png' %}" alt="img1">
                </a>
                <a class="home" href="{% url 'home' %}"><i>Petstagram</i></a>
            </div>
            <div class="nav-links">
                <ul class="nav-group">
                    <li class="nav-item"><a href="{% url 'add-pet' %}"><i>Add Pet</i></a></li>
                    <li class="nav-item"><a href="{% url 'add-photo' %}"><i>Add Photo</i></a></li>
                    <li class="nav-item"><a href="{% url 'login' %}"><i>Login</i></a></li>
                    <li class="nav-item"><a href="{% url 'register' %}"><i>Register</i></a></li>
                    <li class="nav-item"><a href="#"><i>Profile</i></a></li>
                </ul>
            </div>
        </div>
    </nav>
</header>
<!-- End Header -->
```

For now, we **CANNOT add the profile hyperlink implementation**. We will handle it in the **next Web course**.

A thing we could do is to "handle" it by **adding a number for the "pk" parameter**. Any number will work, as there is no implementation for user so far (Note: remember to change it in the future):

```html
<li class="nav-item"><a href="{% url 'profile-details' 1 %}"><i>Profile</i></a></li>
```

404 Template

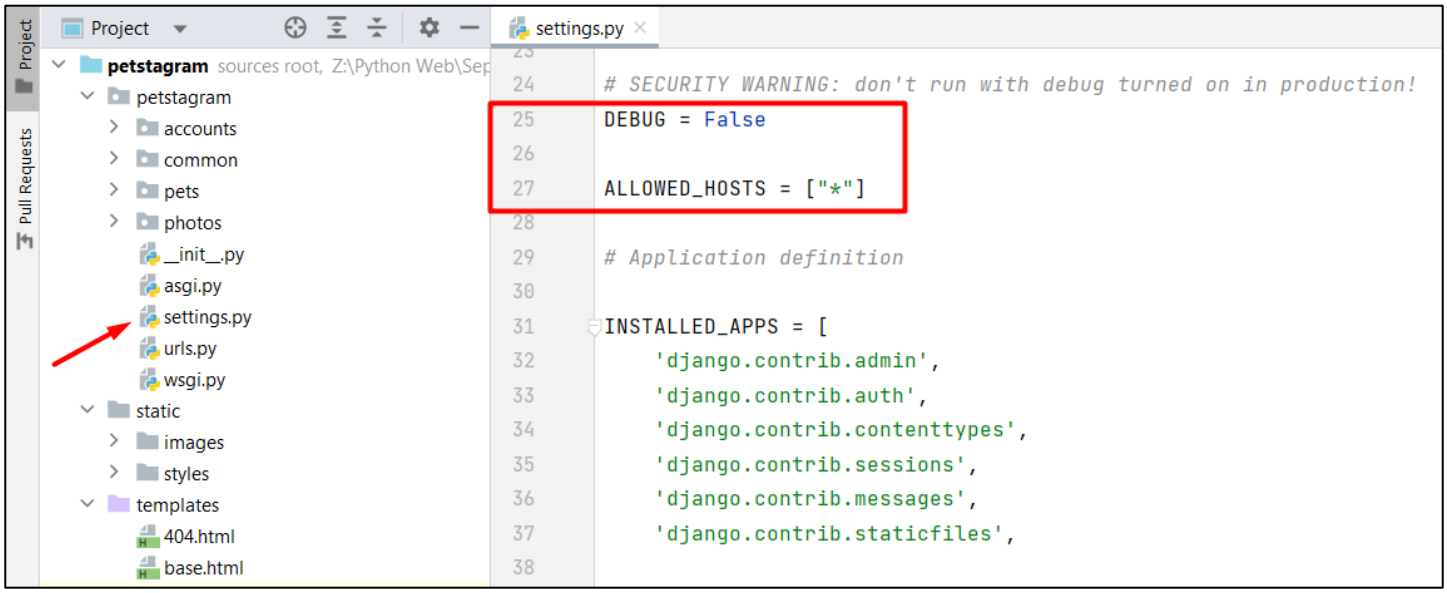
When we **implement the inheritance** and **load the static files** in the template, it should look like this:

```html
404.html
{% extends 'base.html' %}
{% load static %}

{% block content %}
    <div class="not-found-img">
        <img src="{% static 'images/zillow-404-page-5f3ed6a6abaa5.gif' %}" alt="404 not found gif">
    </div>

    <div class="not-found">
        <h1>Uh oh,<br> something broke.</h1>
        <h3>Error 404 - page not found.</h3>
        <a href="#"><button>Return to Petstagram</button></a>
    </div>
{% endblock %}
```

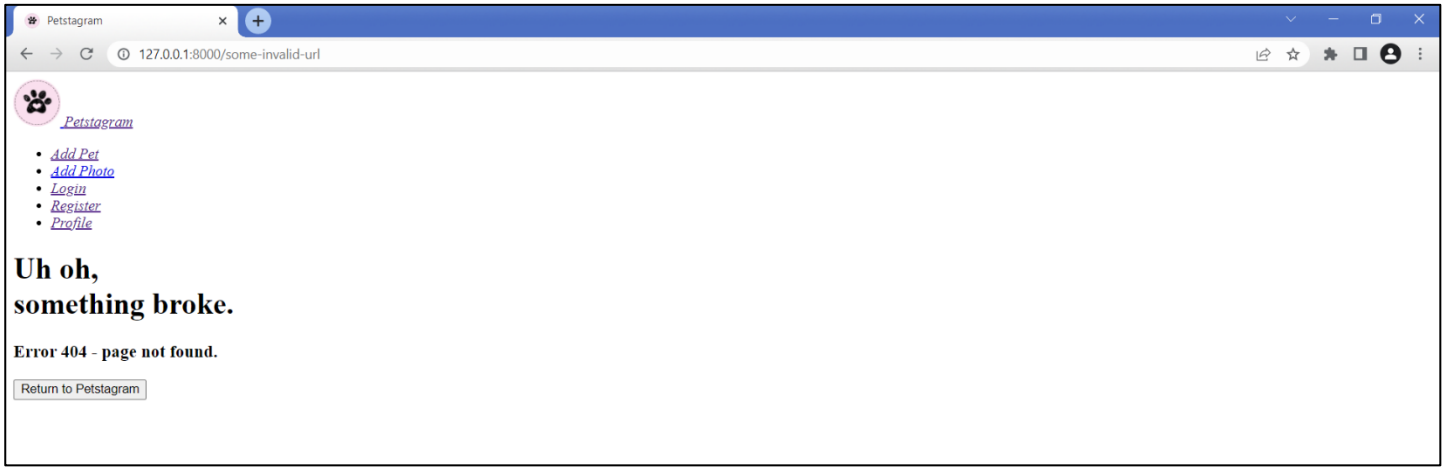
The only thing needed to be done here is **adding a hyperlink** which will lead to the **`Petstagram` home** page. We can **use the url tag**:

```
12        <a href="{% url 'home' %}"><button>Return to Petstagram</button></a>
```

The page is only visible when **DEBUG** setting is set to False. Let us use **`"*"`** syntax to allow all hosts to reach the app:



When you **start the development server** and try to reach **an invalid page** you should see the **404** Template:



Notice that the HTML is loaded, but the **CSS is NOT loaded**. With debug turned off Django will not handle static files for us. The meaning of **DEBUG** set to **`False`** is that the app is in production, so the production web server should take care of the style. To test in **DEBUG `False`** mode locally, we can **run the development server with a specific command**:

"python manage.py runserver --insecure":