Workshop: Petstagram

This document contains the fourth part of the Petstagram Workshop. Today, we will add a user to our project. First, we will start by extending the user model. Then, we will implement the register, login, and logout functionalities. We will work with the profile templates. We will also refactor the code in all apps to add the user object where it is needed.

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

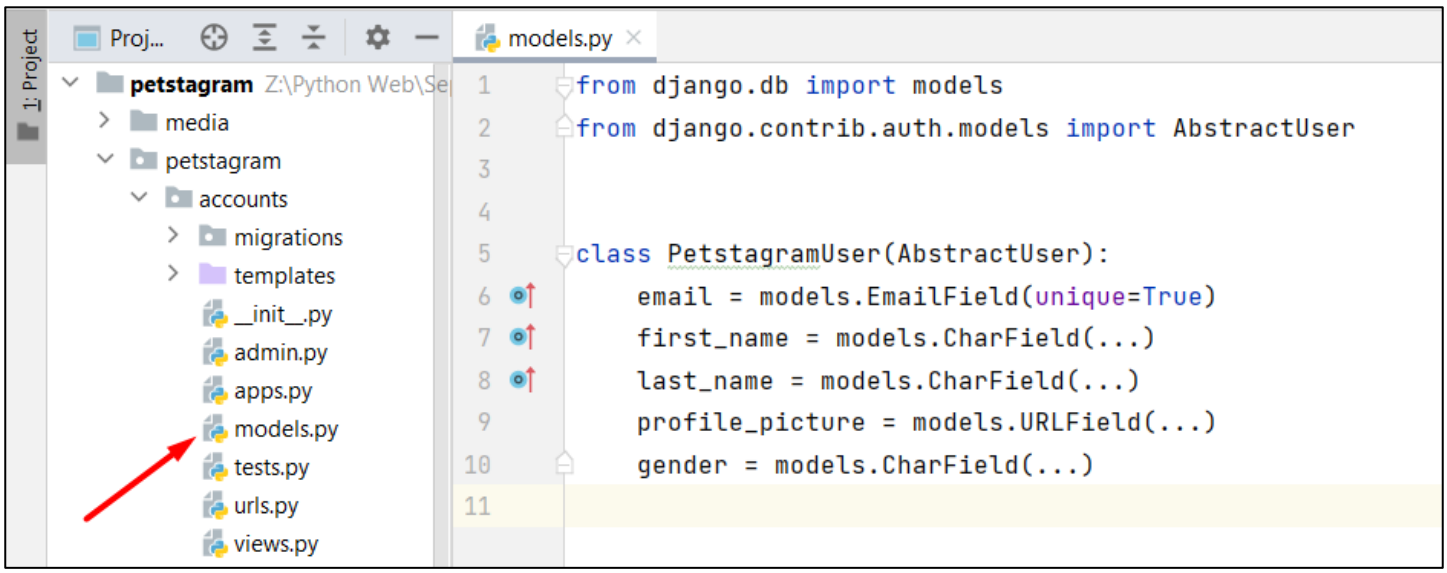
You can directly dive into the app here: https://softuni-petstagram.azurewebsites.net/

1. Workshop - Part 4.1

Extending the User Model

There are several ways that we can choose from to implement our user model. The way Django handles authorization is a perfect fit for our project but still, we want to **add extra attributes without having to create a separate Model**. So, we are going to **create a custom user model inheriting from the `AbstractUser`** Django class.

- The **first** and **last names** of each user should have a **minimum length of 2**, and a **maximum length of 30**. Also, they must contain **only alphabetical letters**.
- Each registered **email** in the app **must be unique**.
- The **gender** is a **choice field** where the user can choose between "**Male**", "**Female**" and "**Do not show**" options:



© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

Then, we will need to update the **`settings.py`** defining the **`AUTH_USER_MODEL`** property to our custom user model:

```
# Default primary key field type
# https://docs.djangoproject.com/en/4.1/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

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

AUTH_USER_MODEL = 'accounts.PetstagramUser'
```

Note: if you are building an app with user authentication, you should consider creating a custom user model at the beginning of the work process.

As this is a workshop project, for the needs of this course, we will need to **drop the database** and create it all over again. Then, we can **delete all migration files**, and **make the migration** for all apps once again.

Next, we will **register the created model in the administration**, and we can **test** if the implementation works **correctly**.

Adding User Registration

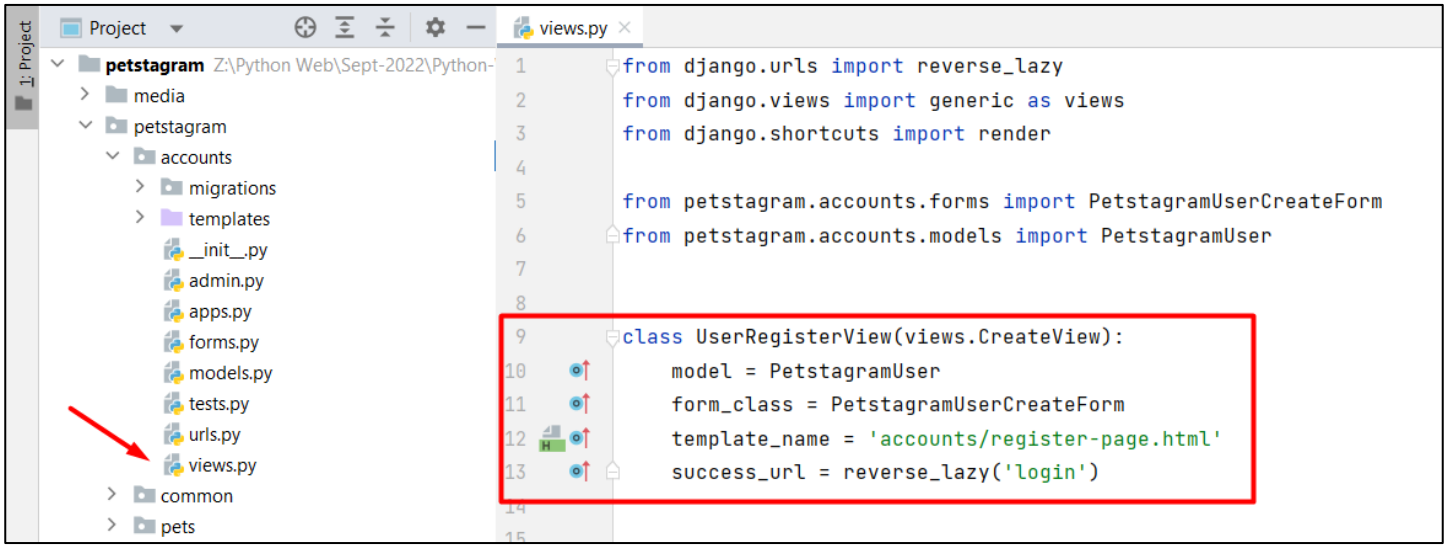
To create a user registration form, we can **extend the Django built-in `UserCreationForm`**. Let us add a **`forms.py`** file in our **`accounts` app** and inherit from the built-in Django form:

```
from django.contrib.auth.forms import UserCreationForm

from petstagram.accounts.models import PetstagramUser


class PetstagramUserCreateForm(UserCreationForm):
    class Meta(UserCreationForm.Meta):
        model = PetstagramUser
        fields = ('username', 'email')
```

Let us now **add the form functionality in the register view** (when the user is successfully registered, they should be **redirected to the login page**). As we know more about the CBVs in Django we will try to implement the register

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

functionality using **`CreateView`**. There we should **add the model**, the **form**, the **template**, and the **success URL**:

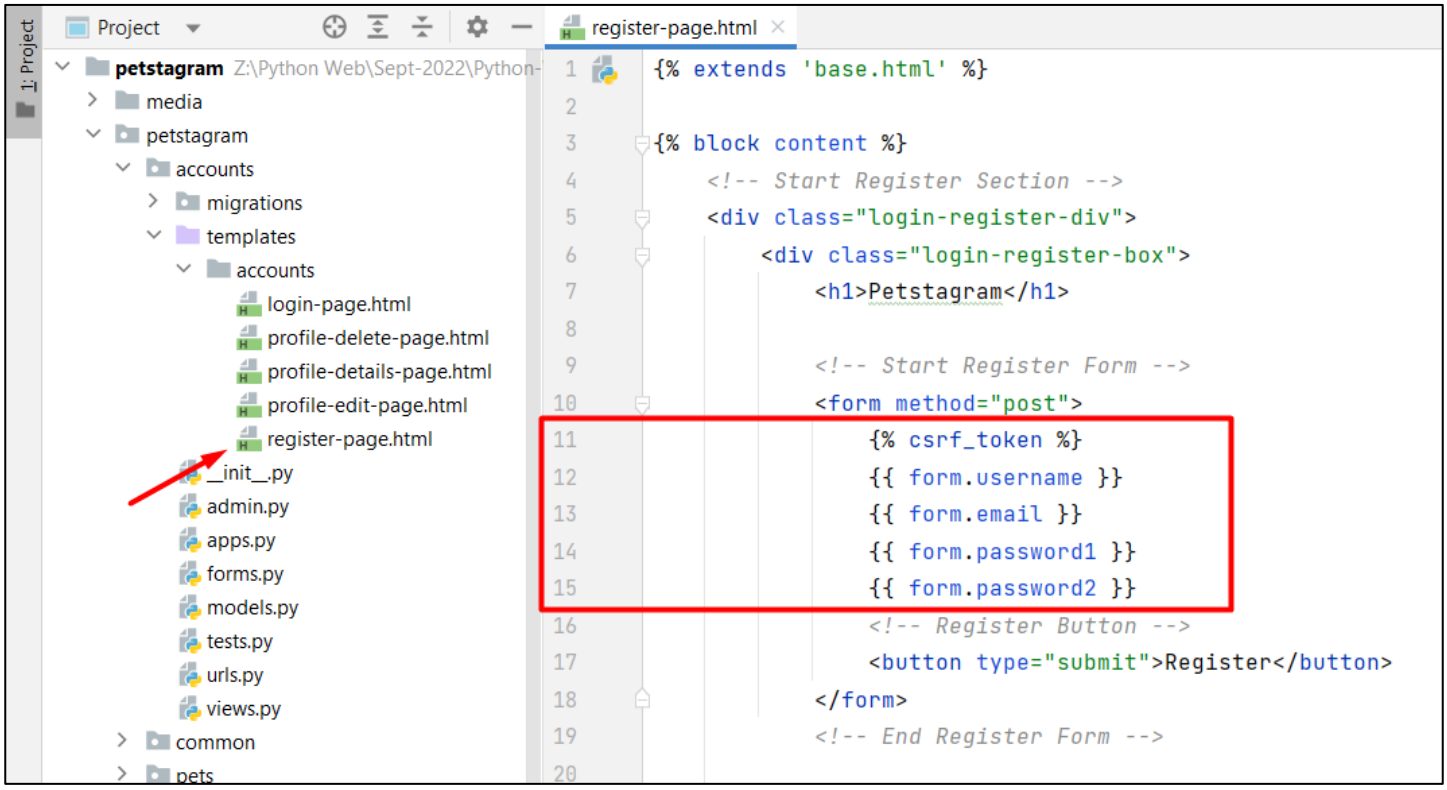


We should **change the view** in the path function on **`accounts/urls.py`** file:

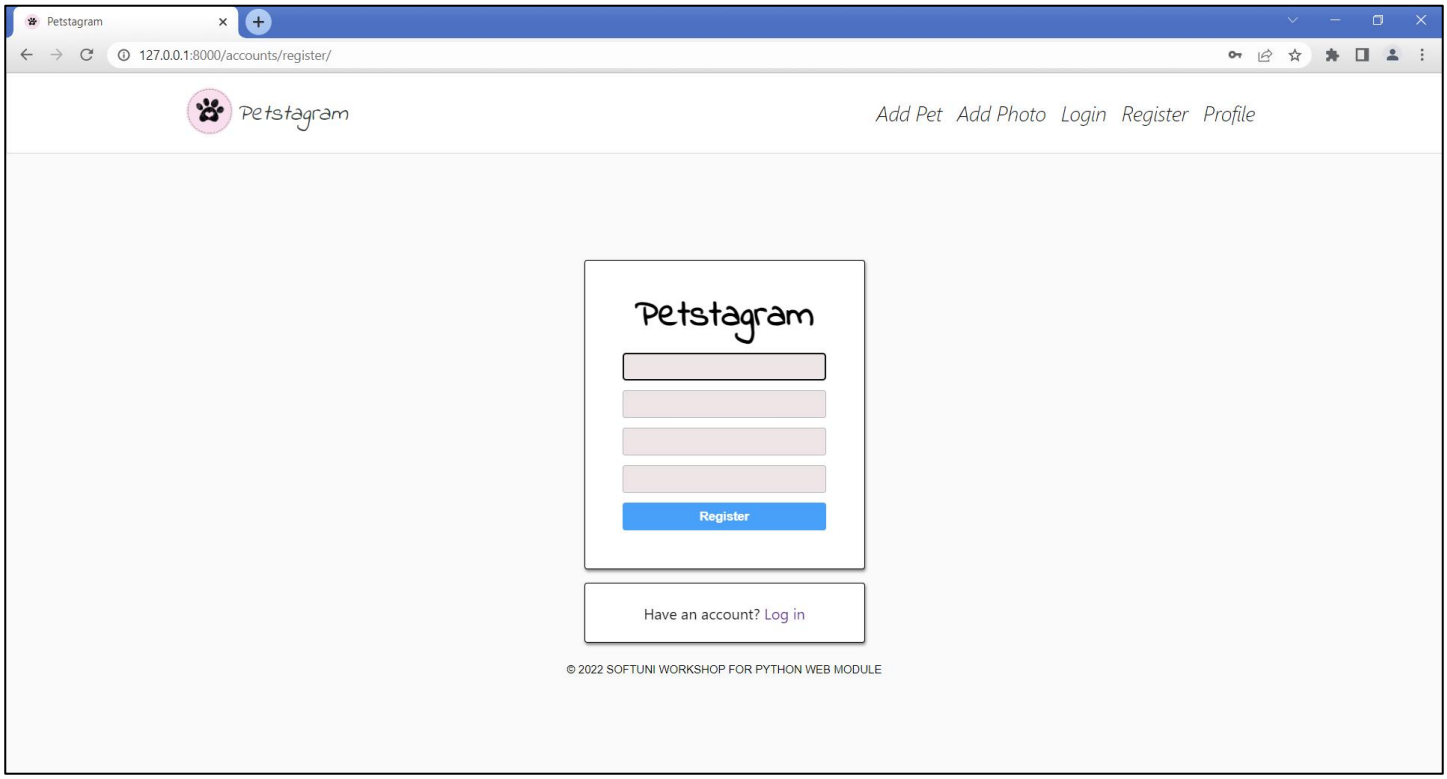
Next, we will add the form to the **`register-page.html`** template. We can escape showing the help texts on the web page, by simply adding the concrete fields we want our users to fill. **Note: Do not forget to add a "href" on the Login**

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

link:



However, it is **not enough** - it is missing the placeholders:



© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

To add them, we can easily **create a template filter**. Add a **`templatetags`** package in the **`accounts`** app and create a **`custom_filters.py`** file. Then, add the implementation of the placeholder and register it:

```
from django import template

register = template.Library()


@register.filter
def placeholder(value, token):
    value.field.widget.attrs["placeholder"] = token
    return value
```

Now, we can **add the template filter** with the value we want to show on the page:

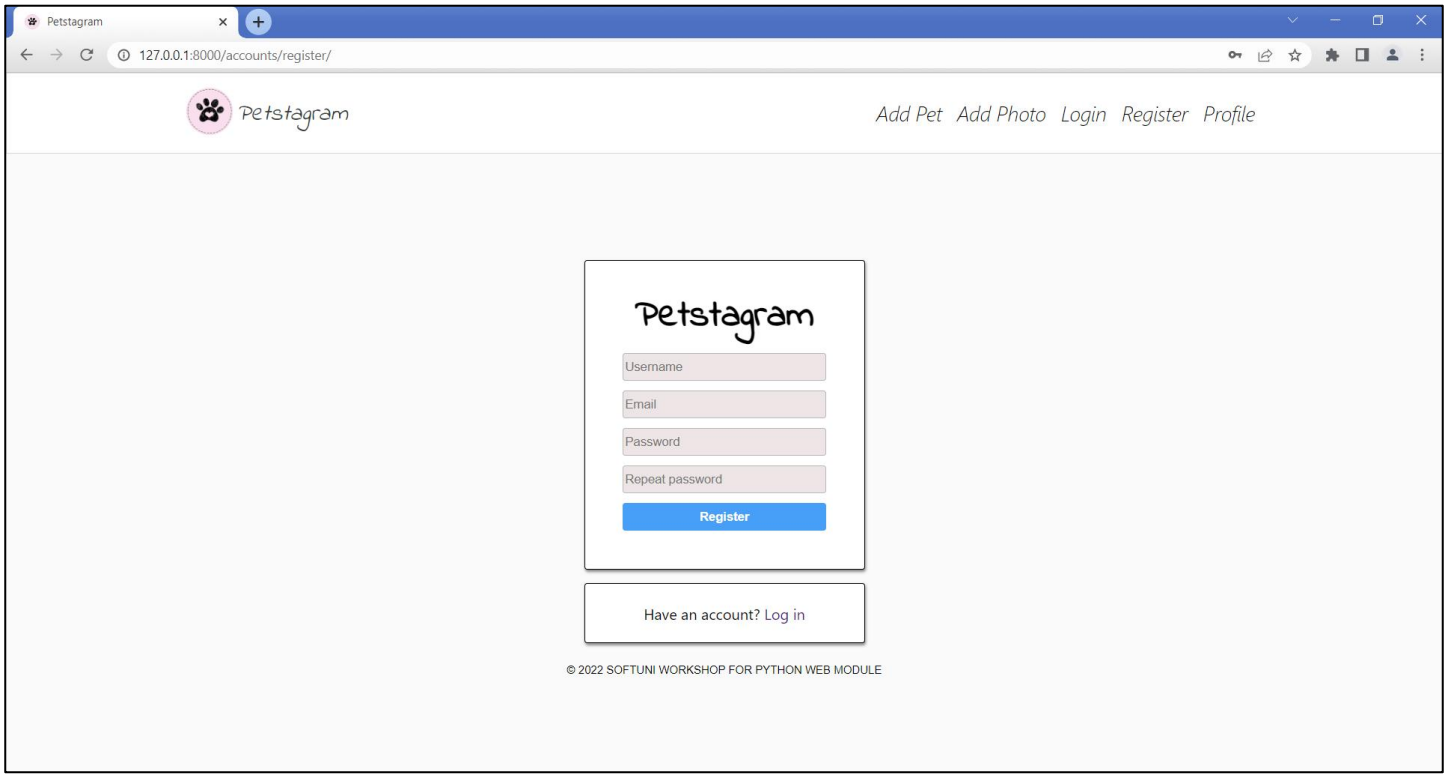
```
{% extends 'base.html' %}
{% load custom_filters %}

{% block content %}
    <!-- Start Register Section -->
    <div class="login-register-div">
        <div class="login-register-box">
            <h1>Petstagram</h1>

            <!-- Start Register Form -->
            <form method="post">
                {% csrf_token %}
                {{ form.username|placeholder:'Username' }}
                {{ form.email|placeholder:'Email' }}
                {{ form.password1|placeholder:'Password' }}
                {{ form.password2|placeholder:'Repeat password' }}
                <!-- Register Button -->
                <button type="submit">Register</button>
            </form>
            <!-- End Register Form -->
```

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

Also, we want to **show errors** when they occur. We can decide if they are going to stay above or below the field:

```
{% block content %}
    <!-- Start Register Section -->
    <div class="login-register-div">
        <div class="login-register-box">
            <h1>Petstagram</h1>

            <!-- Start Register Form -->
            <form method="post">
                {% csrf_token %}
                {{ form.username.errors }}
                {{ form.username|placeholder:'Username' }}
                {{ form.email.errors }}
                {{ form.email|placeholder:'Email' }}
                {{ form.password1.errors }}
                {{ form.password2.errors }}
                {{ form.password1|placeholder:'Password' }}
                {{ form.password2|placeholder:'Repeat password' }}
                <!-- Register Button -->
                <button type="submit">Register</button>
            </form>
            <!-- End Register Form -->

```

Restart the development server and **check** if the implementation **works correctly**:



© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

Adding User Login

Now, it is time to add a user login functionality. First, we will override the **`AuthenticationForm`** so that we can add placeholders in the fields "**username**" and "**password**":

```
from django import forms
from django.contrib.auth.forms import UserCreationForm, AuthenticationForm, UsernameField

from petstagram.accounts.models import PetstagramUser


class PetstagramUserCreateForm(UserCreationForm):...


class LoginForm(AuthenticationForm):
    username = UsernameField(widget=forms.TextInput(attrs={"autofocus": True, "placeholder": "Username"}))
    password = forms.CharField(
        strip=False,
        widget=forms.PasswordInput(attrs={"autocomplete": "current-password", "placeholder": "Password"}),
    )
```

We can use the built-in view **`LoginView`** from the **`contrib.auth.views`** module. We will **add** the overridden **form**, the **template** we want to use, and the **URL** where requests are **redirected after login**:

```
from django.urls import reverse_lazy
from django.views import generic as views
from django.shortcuts import render
from django.contrib.auth import views as auth_views

from petstagram.accounts.forms import PetstagramUserCreateForm, LoginForm
from petstagram.accounts.models import PetstagramUser


class UserRegisterView(views.CreateView):...


class UserLoginView(auth_views.LoginView):
    form_class = LoginForm
    template_name = 'accounts/login-page.html'
    next_page = reverse_lazy('home')
```

Let us refactor the **`login-page.html`** template by adding the "**`form`**" attribute. **Note: Do not forget to add the "href" on the Register link.** Then, **start** the development server and **check** if the form is created **correctly**.

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

Adding User Logout

Let us create the user logout functionality. First, we will **add the path** in the **`account/urls.py`** file:

```
from django.urls import path, include

from petstagram.accounts import views

urlpatterns = [
    path('register/', views.UserRegisterView.as_view(), name='register'),
    path('login/', views.UserLoginView.as_view(), name='login'),
    path('logout/', views.UserLogoutView.as_view(), name='logout'),  # new
    path('profile/<int:pk>/', include([
        path('', views.show_profile_details, name='profile-details'),
        path('edit/', views.edit_profile, name='profile-edit'),
        path('delete/', views.delete_profile, name='profile-delete'),
    ])),
]
```

Next, we will use the Django built-in **`LogoutView`** to reuse the logout functionality implementation and only will **overwrite the `next_page`** attribute, so it redirects to the **login page** after the user successfully logs out:

```
import ...


class UserRegisterView(views.CreateView):...


class UserLoginView(auth_views.LoginView):...


class UserLogoutView(auth_views.LogoutView):
    next_page = reverse_lazy('login')
```

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

Last, we will **add a button** that the users will use to log out from the app. Let us open the **`base.html`** file and use a HTML code to **implement a new navigation bar hyperlink**:

```html
base.html
                <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="{% url 'profile-details' 1 %}"><i>Profile</i></a></li>
                <li class="nav-item"><a href="{% url 'logout' %}"><i>Logout</i></a></li>  <!-- new -->
            </ul>
        </div>
    </div>
  </nav>
</header>
```

Start the development server and check if the implementation works correctly.

2. Workshop - Part 4.2

Add User Editing

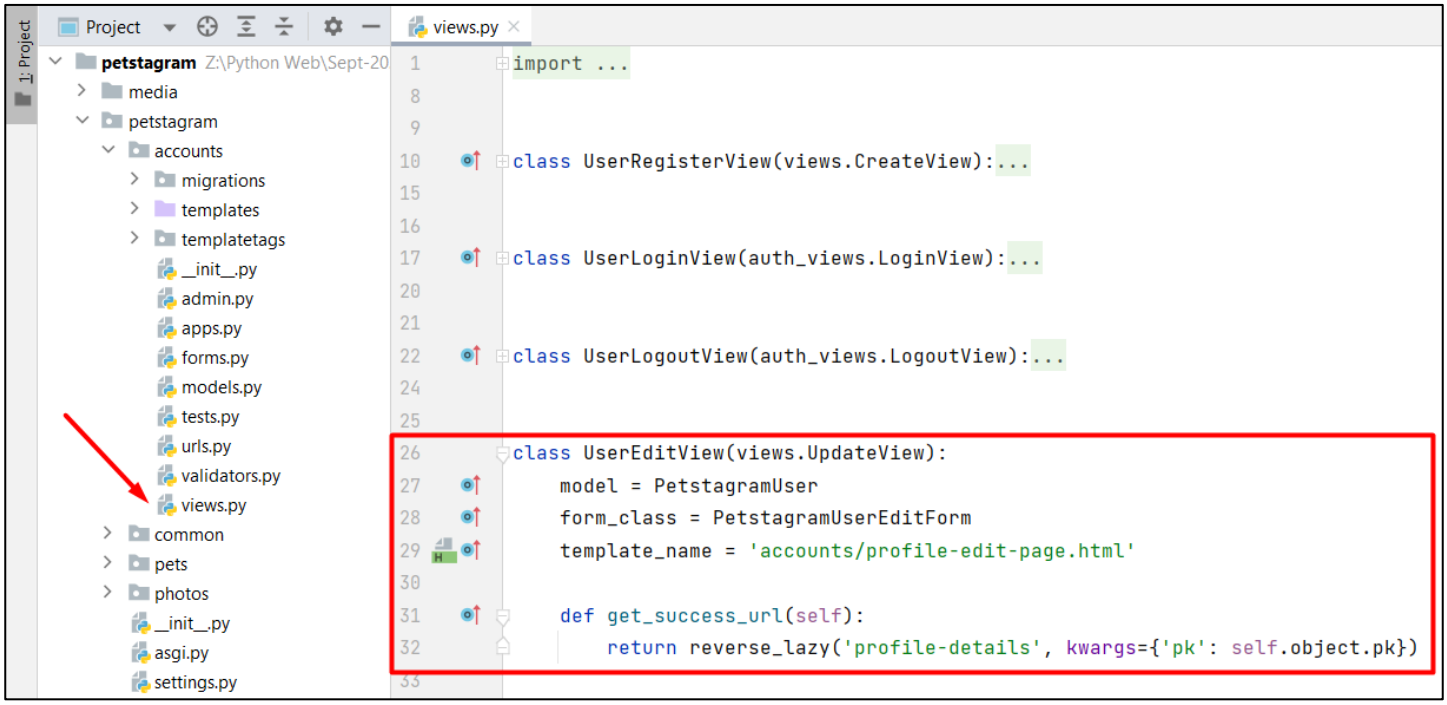
We will continue to implement the user functionality by adding a user edit form. We want the form to **contain** the **username**, **first name**, **last name**, **email**, **profile picture**, and **gender**:

```
Project
petstagram Z:\Python Web\Sept-20
  media
  petstagram
    accounts
      migrations
      templates
      templatetags
      __init__.py
      admin.py
      apps.py
      forms.py
      models.py
      tests.py
      urls.py
      validators.py
      views.py
    common
    pets
    photos
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
```

```python
forms.py
from django import forms
from django.contrib.auth.forms import UserCreationForm

from petstagram.accounts.models import PetstagramUser


class PetstagramUserCreateForm(UserCreationForm):...


class PetstagramUserEditForm(forms.ModelForm):
    class Meta():
        model = PetstagramUser
        fields = ('username', 'first_name', 'last_name', 'email', 'profile_picture', 'gender')
        exclude = ('password',)
        labels = {'username': 'Username',
                  'first_name': 'First Name:',
                  'last_name': 'Last Name:',
                  'email': 'Email:',
                  'profile_picture': 'Image:',
                  'gender': 'Gender:'
                  }
```

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

Then, we will **implement the profile edit view** that will inherit from the **`UpdateView`** class:



© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

Next, we will **refactor the code** in the **`profile-edit-page.html`** template to implement the user form using the Django template language:

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

{% block content %}
    <!-- Start Edit Profile Section -->
    <div class="edit-delete">
        <h2>Edit Profile</h2>
        <!-- Start Edit Profile From -->
        <form method="post">
            {% csrf_token %}

            {% for field in form %}
                <p class="error"> {{ field.errors }} </p>
                <div class="label-input">
                    <label>{{ field.label }}</label>

                    {% if not field.label == "Gender:" %}
                        {{ field }}

                    {% else %}
                        <div class="list-choice">
                            <div class="list-choice-title">Gender</div>
                            <div class="list-choice-objects">

                                {% for type, value in form.fields.gender.choices %}
                                    <label>
                                        <input type="radio" value="{{ value }}"
name="gender"
                                                {% if form.gender.value == value %}
                                               checked
                                                {% endif %}/>
                                        <span>{{ value }}</span>
                                    </label>
                                {% endfor %}

                            </div>
                        </div>
                    {% endif %}

                </div>
                <br>
            {% endfor %}

            <!-- Edit Profile Button -->
            <button class="edit-btn" type="submit">Edit</button>
        </form>
        <!-- End Edit Profile Form -->
    </div>
    <!-- End Edit Profile Section -->
{% endblock %}
```

3. Workshop - Part 4.3

Refactor Navigation Bar

Now, as we have a user, we need to make some changes to the app. Let us start by refactoring the navigation bar. We want to **show the links "Profile" and "Logout" to authenticated users only**, and the links **"Register" and "Login" to**

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

unauthenticated users only. Open the **`base.html`** template and add the condition:

```html
<!-- Our Header section Starts from here -->
<header>
    <nav class="navbar">
        <div class="container">
            <div class="logo">
                <a href="{% url 'home' %}">
                    <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>
                    {% if not request.user.is_authenticated %}
                        <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>
                    {% else %}
                        <li class="nav-item"><a href="{% url 'profile-details' 1 %}"><i>Profile</i></a></li>
                        <li class="nav-item"><a href="{% url 'logout' %}"><i>Logout</i></a></li>
                    {% endif %}
                </ul>
            </div>
        </div>
```

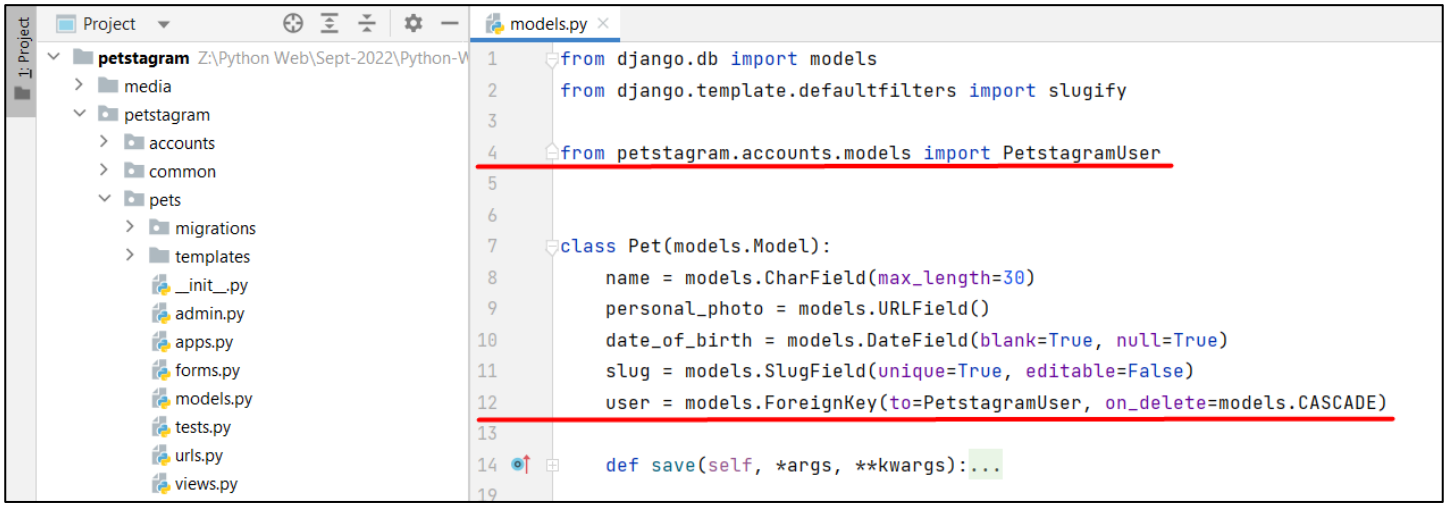
Next, we can **implement the Profile Page URL**:

```html
            <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>
                    {% if not request.user.is_authenticated %}
                        <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>
                    {% else %}
                        <li class="nav-item"><a href="{% url 'profile-details' request.user.pk %}"><i>Profile</i></a></li>
                        <li class="nav-item"><a href="{% url 'logout' %}"><i>Logout</i></a></li>
                    {% endif %}
                </ul>
            </div>
        </div>
    </nav>
</header>
<main>
```

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

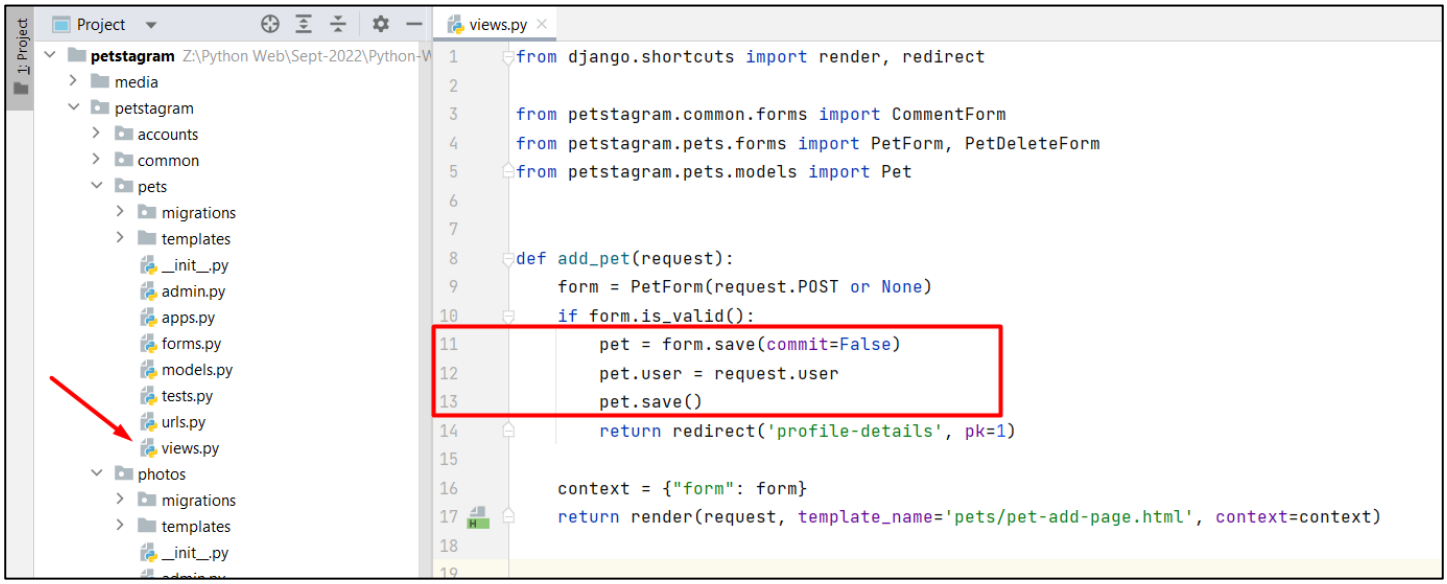
Refactor Models

We should **connect the project models to the user**. Let us start by **adding the user to the pet's model**:



Next, we can connect the user to the **photos, the comments, and the likes**. Make the **migration** files and **migrate** the changes to the database.

When a user creates a pet or a photo they should be **automatically added to the model**. Let us implement it:



Implement the functionality to the **add_photo** view, too. **Note**: **when our form includes many-to-many fields, we should also call `form.save_m2m()` after saving the model instance**. Make changes to the forms if needed.

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

We should also **add the user to the like and comment views**. Let us start by refactoring the code in the **`like_functionality`** view:

```
from django.shortcuts import render, redirect, resolve_url
from pyperclip import copy

from petstagram.common.forms import CommentForm, SearchForm
from petstagram.common.models import Like
from petstagram.photos.models import Photo


def show_home_page(request):...


def like_functionality(request, photo_id):
    photo = Photo.objects.get(id=photo_id)
    liked_object = Like.objects.filter(to_photo_id=photo_id, user=request.user).first()

    if liked_object:
        liked_object.delete()
    else:
        like = Like(to_photo=photo, user=request.user)
        like.save()

    return redirect(request.META['HTTP_REFERER'] + f'#{photo_id}')
```

Next, **add the user to the comment** they post:

```
from petstagram.photos.models import Photo


def show_home_page(request):...


def like_functionality(request, photo_id):...


def copy_link_to_clipboard(request, photo_id):...


def add_comment(request, photo_id):
    if request.method == 'POST':
        photo = Photo.objects.get(id=photo_id)
        form = CommentForm(request.POST)
        if form.is_valid():
            comment = form.save(commit=False)
            comment.to_photo = photo
            comment.user = request.user
            comment.save()

        return redirect(request.META['HTTP_REFERER'] + f'#{photo_id}')
```

Refactor Pet Photo Posts

Let us now work to **improve the interface of the home page**. First, we should check the **`show_home_page`** view. We will need to add sorting for the **likes made by the current user using the app**, so the app shows the user which photos

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

are liked by them:

```
from petstagram.photos.models import Photo


def show_home_page(request):
    all_photos = Photo.objects.all()
    comment_form = CommentForm()
    search_form = SearchForm()
    user = request.user
    all_liked_photos_by_request_user = [like.to_photo_id for like in user.like_set.all()]

    if request.method == 'POST':
        search_form = SearchForm(request.POST)
        if search_form.is_valid():
            all_photos = all_photos.filter(tagged_pets__name__icontains=search_form.cleaned_data['pet_name'])

    context = {
        "all_photos": all_photos,
        "comment_form": comment_form,
        "search_form": search_form,
        "all_liked_photos_by_request_user": all_liked_photos_by_request_user,
    }

    return render(request, template_name='common/home-page.html', context=context)
```

Now, we are ready to refactor the **`pets-posts.html`** template. We want for **each picture to show to the user who uploaded it**. Let us open the file and start implementing the user information:

```
{% load static %}
{% for photo in all_photos %}
    <div class="card">
        <!-- Start User Details and Image Location -->
        <div class="top">
            <div class="userDetails">
                <!-- Link to User Profile Page -->
                <a href="{% url 'profile-details' photo.user.pk %}">
                    <div class="profilepic">
                        <div class="profile_img">
                            <div class="image">
                                {% if photo.user.profile_picture %}
                                    <img src="{{ photo.user.profile_picture }}" alt="img8">
                                {% else %}
                                    <img src="{% static 'images/person.png' %}" alt="img8">
                                {% endif %}
                            </div>
                        </div>
                    </div>
                </a>
                <h3>
                    <!-- Link to User Profile Page -->
                    <a href="{% url 'profile-details' photo.user.pk %}">
                        <!-- if user has first name and last name -->
```

On the next rows we can see that we should **show the first and/or the last name of the user if they exist**. Otherwise, we should **show the user's username**. To do that, we can escape writing code in the template by **adding an additional**

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

method to the user model:

```
Project
petstagram  Z:\Python Web\Sept-2022\Python-W
  media
  petstagram
    accounts
      migrations
      templates
      templatetags
      __init__.py
      admin.py
      apps.py
      forms.py
      models.py
      tests.py
      urls.py
      validators.py
      views.py
    common
      migrations
      templates
        common
          home-page.html
          pets-posts.html
      __init__.py
      admin.py
      apps.py
      forms.py
      models.py
      tests.py
```

```python
    last_name = models.CharField(
        max_length=30,
        validators=(MinLengthValidator(2), validate_isalpha),
        null=True,
        blank=True)

    profile_picture = models.URLField(null=True, blank=True)

    gender = models.CharField(
        max_length=11,
        choices=GENDERS,
        null=True,
        blank=True,
        default=DO_NOT_SHOW
    )

    def get_user_name(self):
        if self.first_name and self.last_name:
            return self.first_name + ' ' + self.last_name
        elif self.first_name or self.last_name:
            return self.first_name or self.last_name
        else:
            return self.username
```

Then, we can **use the method in the template**:

```html
<!-- Start User Details and Image Location -->
<div class="top">
    <div class="userDetails">
        <!-- Link to User Profile Page -->
        <a href="{% url 'profile-details' photo.user.pk %}">
            <div class="profilepic">
                <div class="profile_img">
                    <div class="image">
                        {% if photo.user.profile_picture %}
                            <img src="{{ photo.user.profile_picture }}" alt="img8">
                        {% else %}
                            <img src="{% static 'images/person.png' %}" alt="img8">
                        {% endif %}
                    </div>
                </div>
            </div>
        </a>
        <h3>
            <!-- Link to User Profile Page -->
            <a href="{% url 'profile-details' photo.user.pk %}">
                <!-- if user has first name and last name -->
                {{ photo.user.get_user_name }}
                <!-- else -->
                <!-- show user username -->
            </a>
            <br>
```

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

We will change the visualtion of the **like button**, so it shows **which pictures are liked by the user**:

pets-posts.html

```
<div class="bottom">
    <div class="actionBtns">
        <div class="left">
            <!-- Start Like Button -->
            <span class="heart">
                <a href="{% url 'like' photo.id %}">
                <!-- if user has liked the photo -->
                {% if photo.id in all_liked_photos_by_request_user %}
                    <svg style="..."
                        xmlns="http://www.w3.org/2000/svg"
                        width="24"
                        height="24"
                        fill="currentColor"
                        class="bi bi-heart-fill"
                        viewBox="0 0 16 16">
                    <!-- Coordinate path -->
                    <path fill-rule="evenodd"
                         d="M8 1.314C12.438-3.248 23.534 4.735
                         8 15-7.534 4.736 3.562-3.248 8 1.314z"
                         fill="red">
                    </path>
                </svg>
                    <!-- else -->
                {% else %}
                    <svg aria-label="Like"
                        color="#262626"
                        fill="#262626"
                        height="24"
                        role="img"
```

We should **refactor the pet details URL**, too:

pets-posts.html

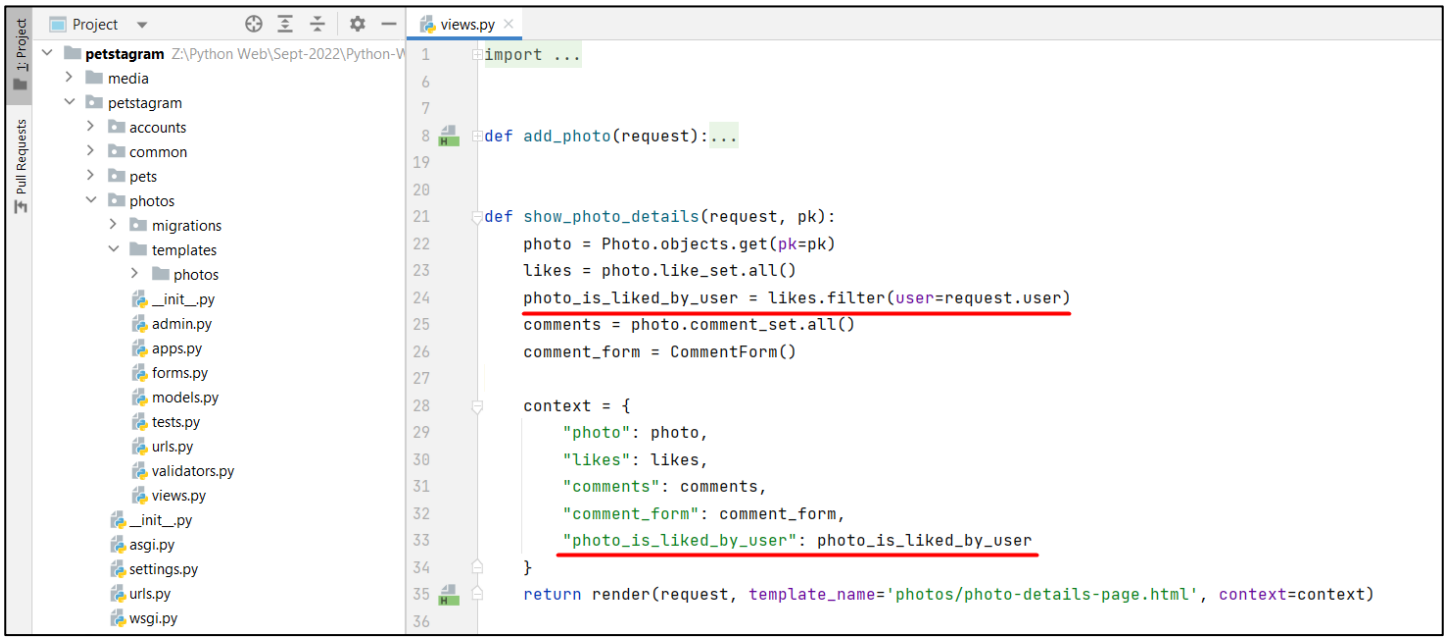
```

<!-- Start Tagged Pets -->
{% for pet in photo.tagged_pets.all %}
    <!-- Link to First Tagged Pet Details Page-->
    <a href="{% url 'pet-details' pet.user.username pet.slug %}">
        <p class="message">
            <b>{{ pet.name }}</b>
        </p>
    </a>
{% endfor %}
<!-- End Tagged Pets -->
```

Start the development server and check the visualization.

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

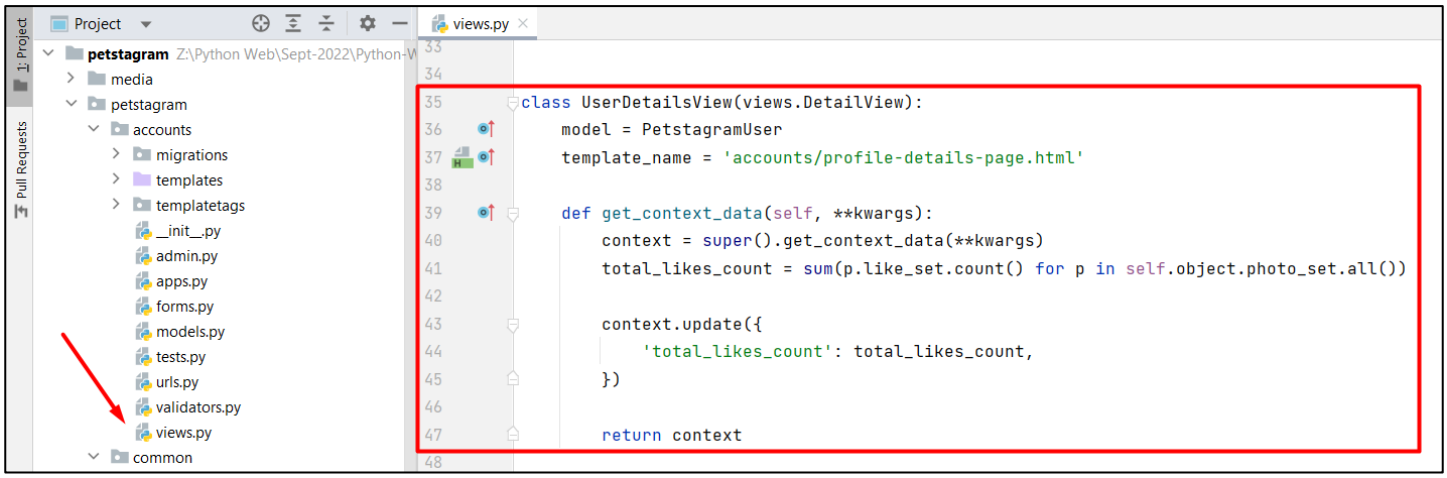
Refactor Photo Details Page

Add the **photo owner's data**, and **the comment user data** and **implement the like button visualization**, so the current user sees if they liked the photo or not. The **buttons edit** and **delete should be only visible to the owner** of the photo.
Hint: you can check if the user liked the photo by filtering them in the likes set:



Refactor Profile Details Page

It is time to **add the user data to the profile**. We will use a CBV to generate the **user model** and the **template**, and we will additionally **add the total count of likes in the context**:



In the **`profile-details-page.html`** template we will **add the user data**, the **count of all user photos**, the **count of all user pets**, and **the count of all likes** that the user has collected, **all user pets**, and **all user photos**.

Check for Additional Changes

Do not forget to check all functionality on the app to see if everything works correctly:

- Check if the **user object** is needed elsewhere.
- Check all **links** and **buttons**.
- **Login is required** for all pages and buttons, **except** for the **home page**, and **share post button**.

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

Example: in the **pet app**, the **`show_pet_details`** view should **find the owner of the pet** and use it to **visualize the edit/delete button to the owner** of the pet **only**:

```
Project
petstagram Z:\Python Web\Sept-2022\Python-W
  media
  petstagram
    accounts
    common
    pets
      migrations
      templates
      __init__.py
      admin.py
      apps.py
      forms.py
      models.py
      tests.py
      urls.py
      views.py
```

```python
def show_pet_details(request, username, pet_slug):
    pet = Pet.objects.get(slug=pet_slug)
    owner = PetstagramUser.objects.get(username=username)
    all_photos = pet.photo_set.all()
    comment_form = CommentForm()
    context = {
        "pet": pet,
        "all_photos": all_photos,
        "comment_form": comment_form,
        "owner": owner,
    }
    return render(request, template_name='pets/pet-details-page.html', context=context)
```

Add the **implementation of the URL** in the **`pet-details-page.html`** template, too:

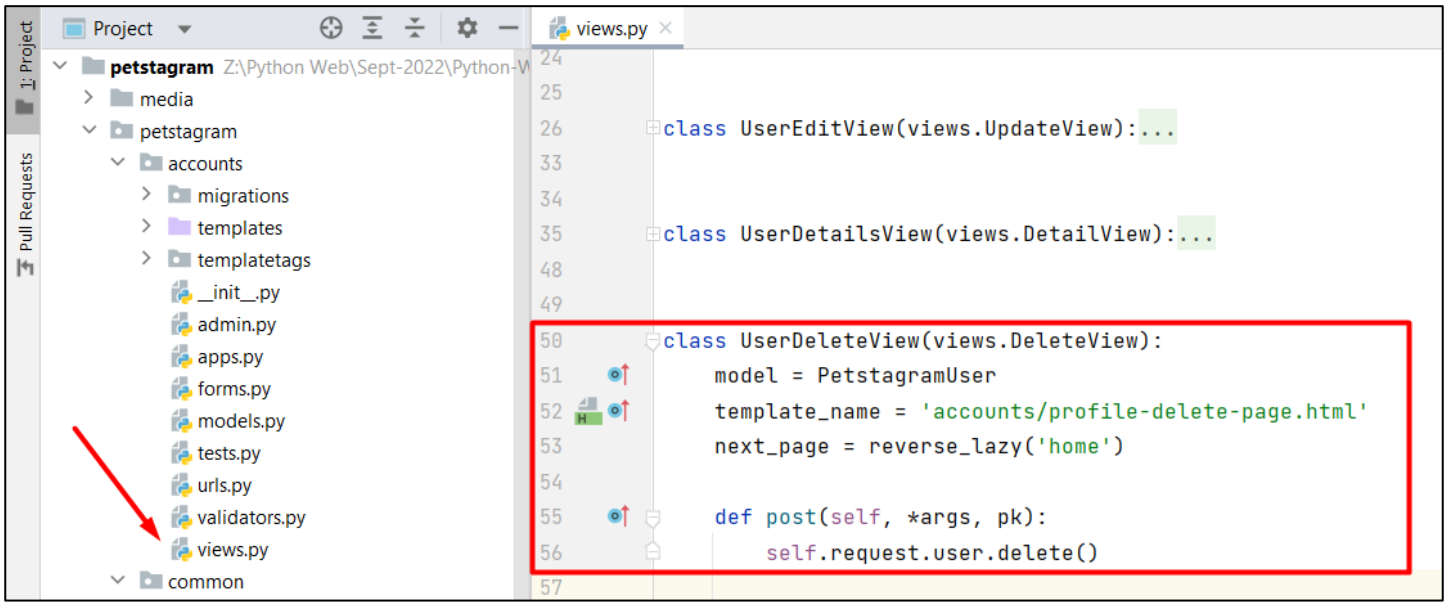
```html
<div class="personal">
    <div class="edit">
        <!-- Pet Name -->
        <p>{{ pet.name }}</p>
        <!-- Pet Edit Button -->
        {% if owner == request.user %}
        <a href="{% url 'edit-pet' owner.username pet.slug %}">
            <img class="edit-img" src="/static/images/edit-pen-icon-6.jpg" alt="edit button">
        </a>
        <!-- Pet Delete Button -->
        <a href="{% url 'delete-pet' owner.username pet.slug %}">
            <img class="bin-img" src="/static/images/icon-remove-22.jpg" alt="bin button">
        </a>
        {% endif %}
    </div>
    <div class="data">
        <!-- Pet Total Photos -->
        <span>{{ all_photos.count }}</span>
        <p>photos</p>
```

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

4. Workshop - Part 4.4

Add user delete functionality

We will add the final user functionality for this workshop. When a user is deleted, all their **photos**, **pets**, **likes**, and **comments should be deleted** too. Then, the app **redirects to the home page** with no authentication:



We should refactor the **`profile-delete-page.html`** template:

```
profile-delete-page.html
{% extends 'base.html' %}

{% block content %}
    <!-- Start Delete Profile Section -->
        <div class="delete-profile">
            <h1>
                Are you sure you want to delete your profile?
            </h1>
            <form method="post" action="{% url 'profile-delete' request.user.pk %}">
                {% csrf_token %}
            <div class="sure-buttons">
                <!-- Yes (Delete Profile) Button -->
                <button>Yes</button>
                <!-- Go Back Button -->
                <a class="edit-btn" href="javascript:history.back()">Go Back</a>
            </div>
            </form>
            <img src="/static/images/are_you_sure_cat.gif" alt="are you sure cat git">
        </div>
    <!-- End Delete Profile Section -->
```

© SoftUni – about.softuni.bg. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us: