Workshop: Petstagram

This document contains the third part of the Petstagram Workshop. Today, we will **create the pet forms** for our model for the project and we will implement them in our templates. After that, we will **add the image functionality** in the photo model, we will **create the form**, and inject it into the templates. Finally, we will **create a comment form** to add comments and a **search form** to search images by a name of a tagged pet.

Note: we will NOT work with the profile/ user form in the Python Web Basics Course.

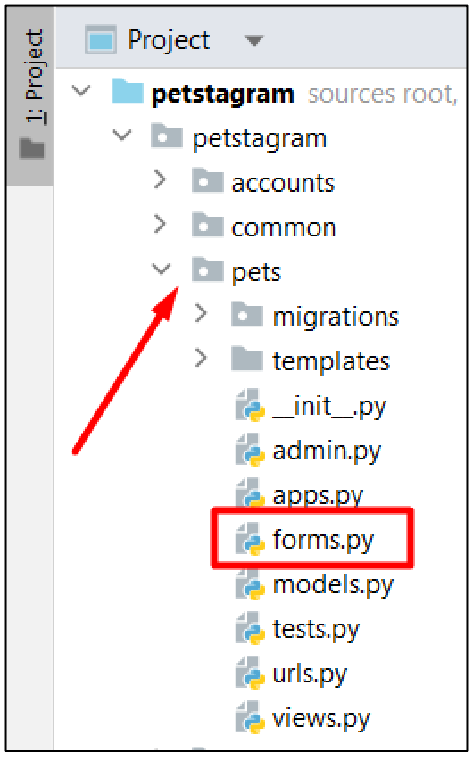
The full project description of the project can be found in the **Workshop Description Document**.

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

1. Workshop - Part 3.1

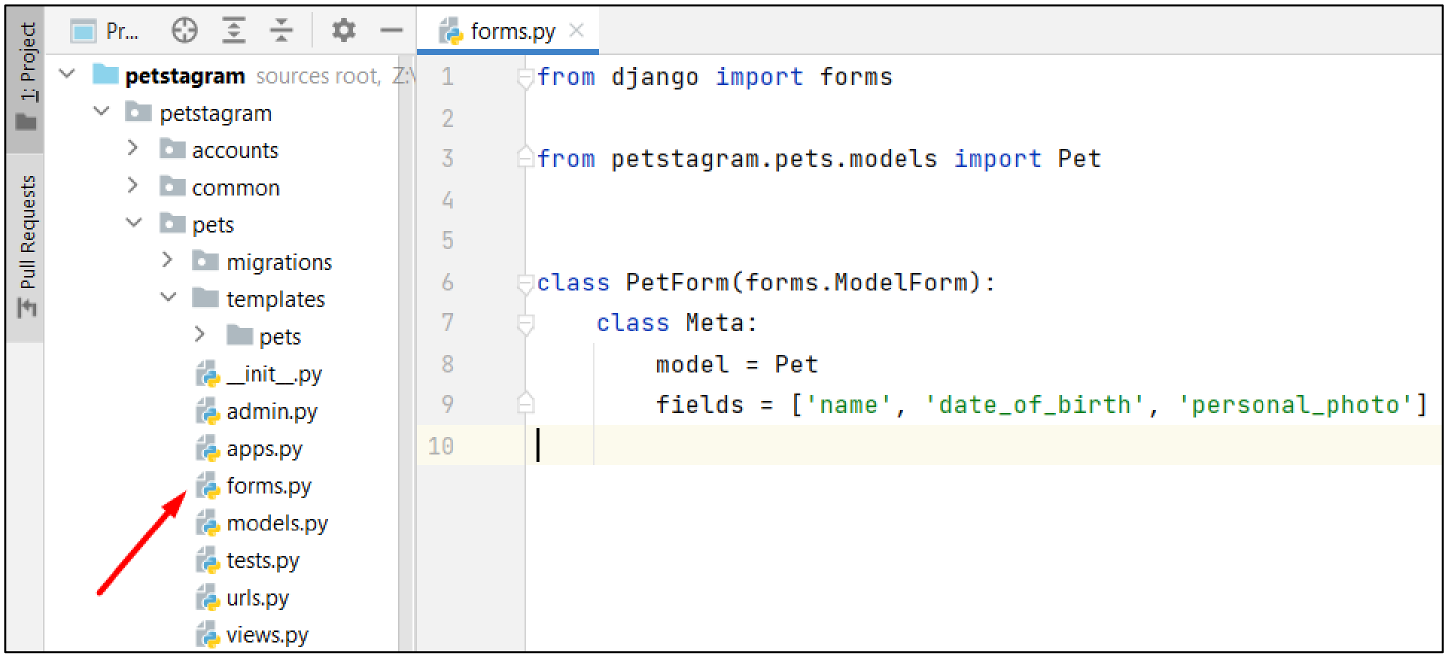
Creating a Pet Form

In a Django project, there is **NOT** a forms file in the prebuilt structure. We need to add a new **`forms.py`** file inside the **`pet`** application directory:



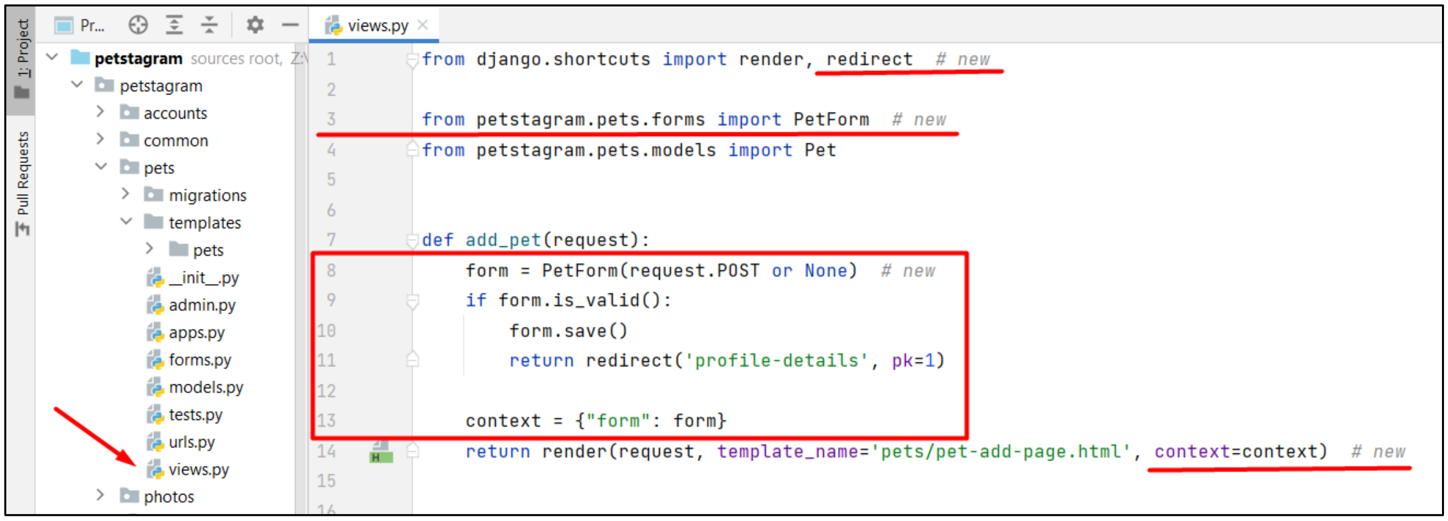
In **`forms.py`** we can implement the pet form. Because the pet already has a model in our app, we do **NOT need to create the form field by field** - Django can do it for us with the **`ModelForm` class**. Let us open the **`forms.py`** file, **import**

the Pet model and **create a simple Pet form** from the Pet model:



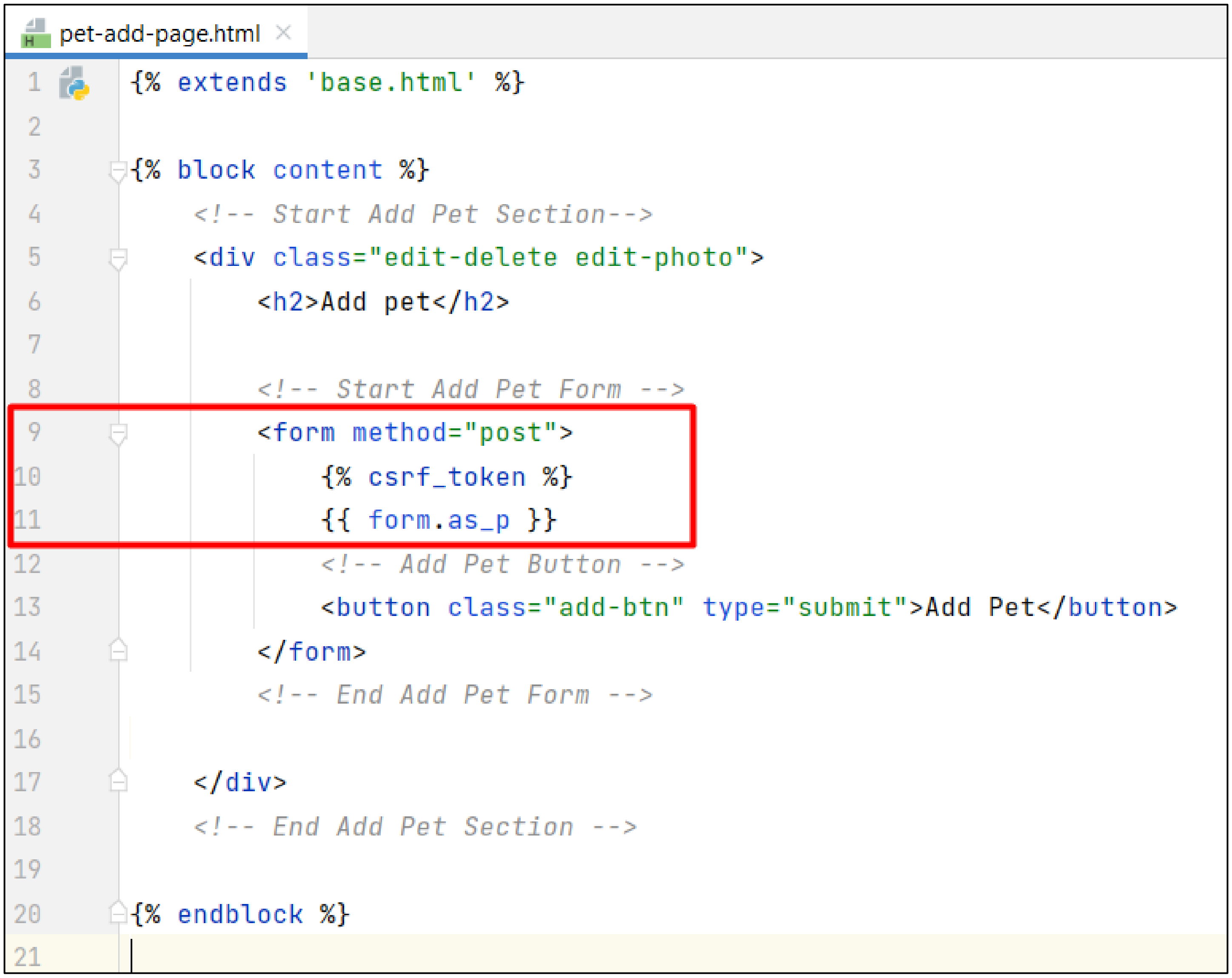
```python
from django import forms

from petstagram.pets.models import Pet


class PetForm(forms.ModelForm):
    class Meta:
        model = Pet
        fields = ['name', 'date_of_birth', 'personal_photo']
```

Next, let us **create the form functionality** in the **`pets/view.py`**. First, we will import the **`PetForm`**. Next, we will **create a form** that should be filled with information (**request.POST** is a dictionary-like object that lets you access submitted data) **or should be blank**. Next, we will **check if the form is valid** and if so - we will **save the information** in the database. When the information is saved, we want to **redirect the user to the profile details page** (for now we will set a random number for the pk). Finally, if the **form is NOT valid** (it is blank, or the validation fails) we want to **add it to the context to be shown in the template**:



```python
from django.shortcuts import render, redirect  # new

from petstagram.pets.forms import PetForm  # new
from petstagram.pets.models import Pet


def add_pet(request):
    form = PetForm(request.POST or None)  # new
    if form.is_valid():
        form.save()
        return redirect('profile-details', pk=1)

    context = {"form": form}
    return render(request, template_name='pets/pet-add-page.html', context=context)  # new
```

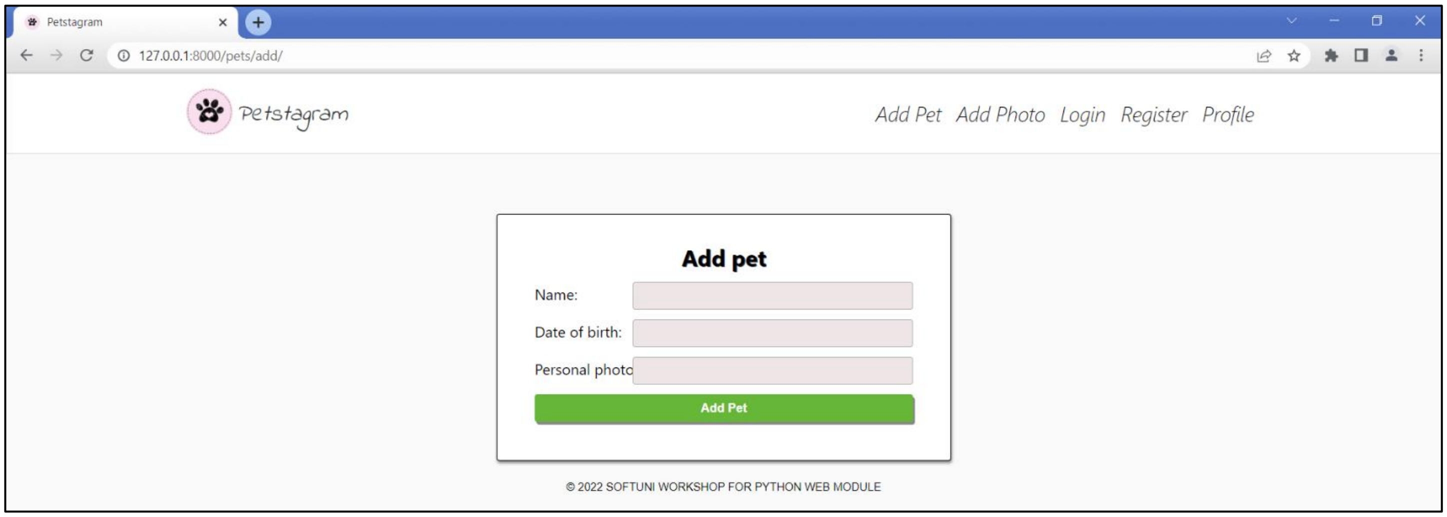
Now, it is time to **add the form** to the **`pet-add-page.html`** template. First, we will **delete the html form** and we will **inject the Django form** from the context - we want to **show the form on separate lines**, so will use the shortcut method **`"as_p"`** to show each input field on a new line (new paragraph). Next, we will **set a post method** to the form, and finally - we will **add the CSRF token**:



pet-add-page.html

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

{% block content %}
    <!-- Start Add Pet Section-->
    <div class="edit-delete edit-photo">
        <h2>Add pet</h2>

        <!-- Start Add Pet Form -->
        <form method="post">
            {% csrf_token %}
            {{ form.as_p }}
            <!-- Add Pet Button -->
            <button class="add-btn" type="submit">Add Pet</button>
        </form>
        <!-- End Add Pet Form -->

    </div>
    <!-- End Add Pet Section -->

{% endblock %}

```

Let us **check** what we have done by now:



Petstagram

127.0.0.1:8000/pets/add/

Add pet

Name:

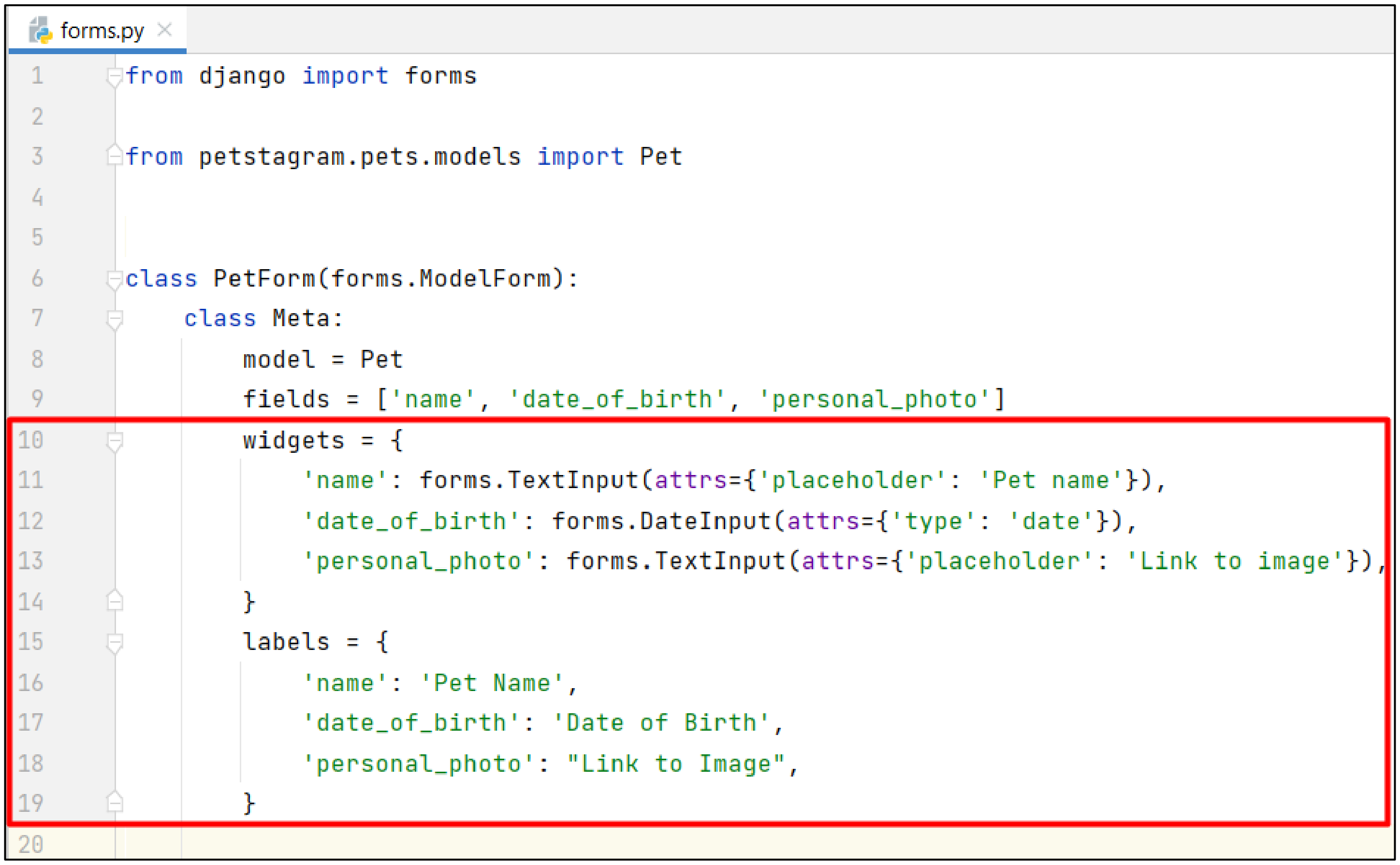
Date of birth:

Personal photo

© 2022 SOFTUNI WORKSHOP FOR PYTHON WEB MODULE

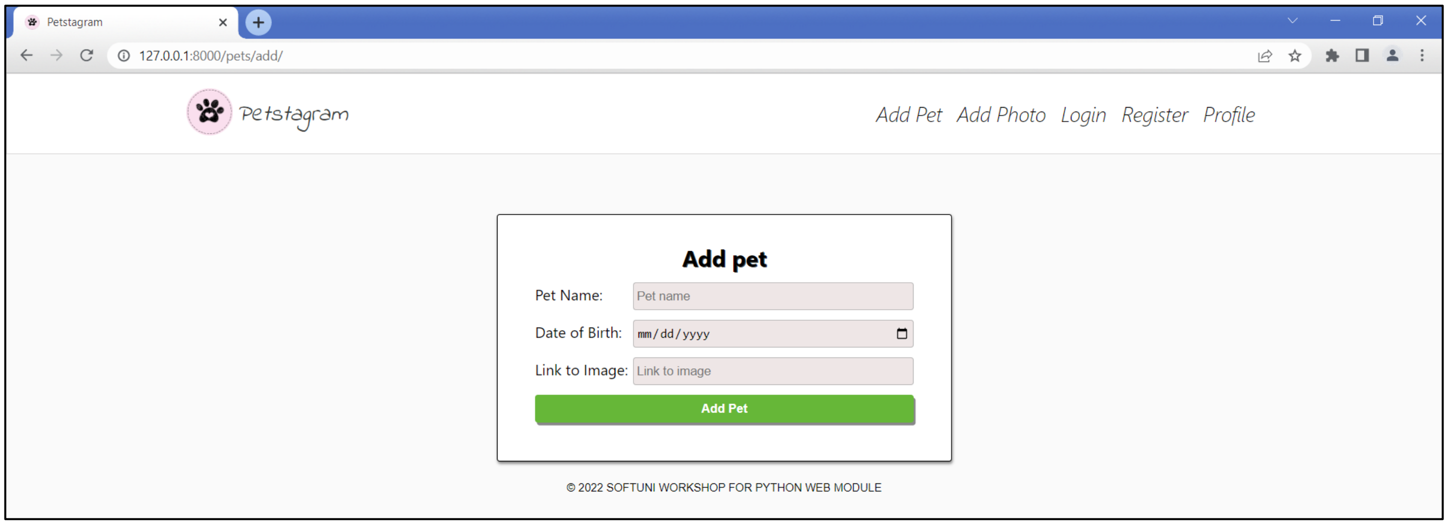
The pet form is **generated** and **work corettly**, but it can look much better. To **improve the UI** we can change some **labels**, and some **placeholders** and **make the date field** visualized with a generated calendar. Let us again **open the**

pet/forms.py and write some code:



```python
from django import forms

from petstagram.pets.models import Pet


class PetForm(forms.ModelForm):
    class Meta:
        model = Pet
        fields = ['name', 'date_of_birth', 'personal_photo']
        widgets = {
            'name': forms.TextInput(attrs={'placeholder': 'Pet name'}),
            'date_of_birth': forms.DateInput(attrs={'type': 'date'}),
            'personal_photo': forms.TextInput(attrs={'placeholder': 'Link to image'}),
        }
        labels = {
            'name': 'Pet Name',
            'date_of_birth': 'Date of Birth',
            'personal_photo': "Link to Image",
        }
```

Now, the pet add page looks like that and the form works correctly:



Petstagram
127.0.0.1:8000/pets/add/
Petstagram
Add Pet Add Photo Login Register Profile

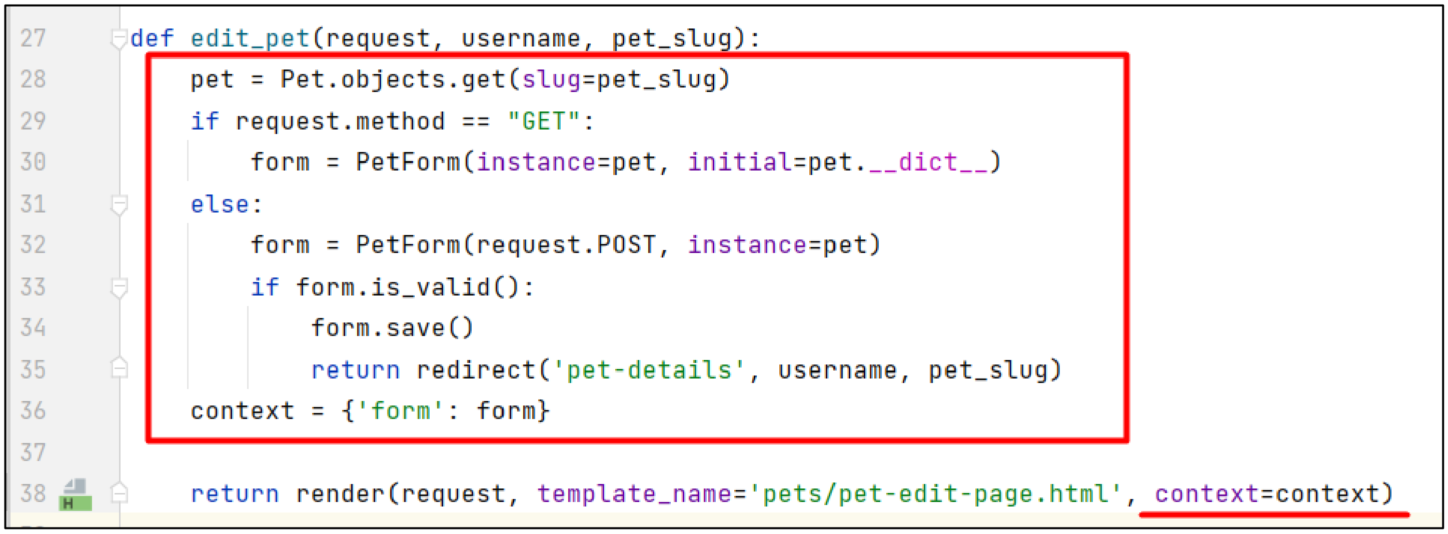
Add pet

Pet Name:	Pet name
Date of Birth:	mm/dd/yyyy
Link to Image:	Link to image

Add Pet
© 2022 SOFTUNI WORKSHOP FOR PYTHON WEB MODULE

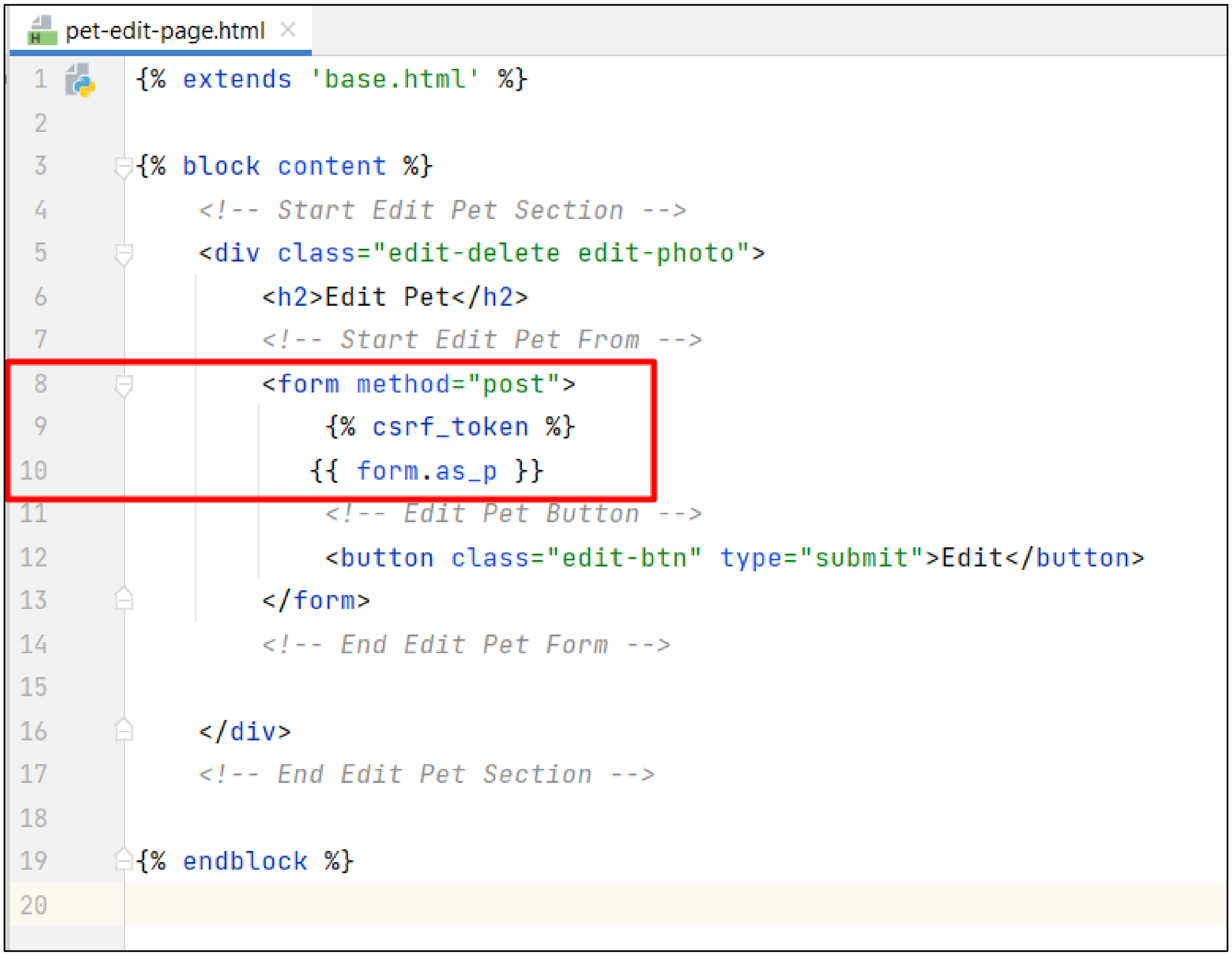
Creating a Pet Edit Form

We should **add a pet edit form** functionality. We should **use the same fields and same formatting** as in the pet creation form. So, we can **use the already generated `PetForm`** and **prepopulate it with the data from the current pet** we want to edit. Let us open the **`pets/views.py`** file and create the pet edit functionality. When the method is **GET** we will fill the form with the **initial pet data**, and if the method is **POST** - we will **update the data** in the concrete pet instance and will **save it in the database**:



```python
def edit_pet(request, username, pet_slug):
    pet = Pet.objects.get(slug=pet_slug)
    if request.method == "GET":
        form = PetForm(instance=pet, initial=pet.__dict__)
    else:
        form = PetForm(request.POST, instance=pet)
        if form.is_valid():
            form.save()
            return redirect('pet-details', username, pet_slug)
    context = {'form': form}

    return render(request, template_name='pets/pet-edit-page.html', context=context)
```

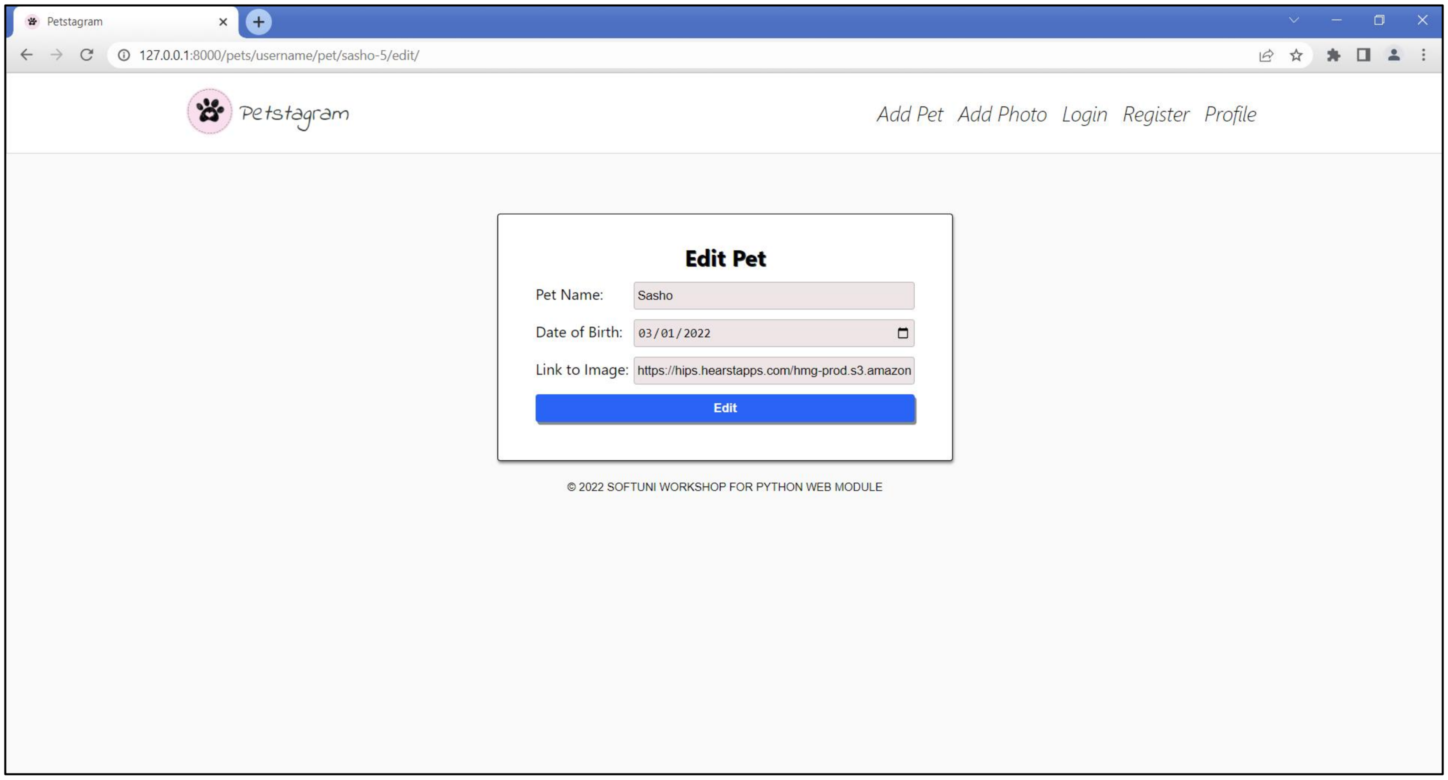
Refactor the **`pet-edit-page.html`** template:



pet-edit-page.html

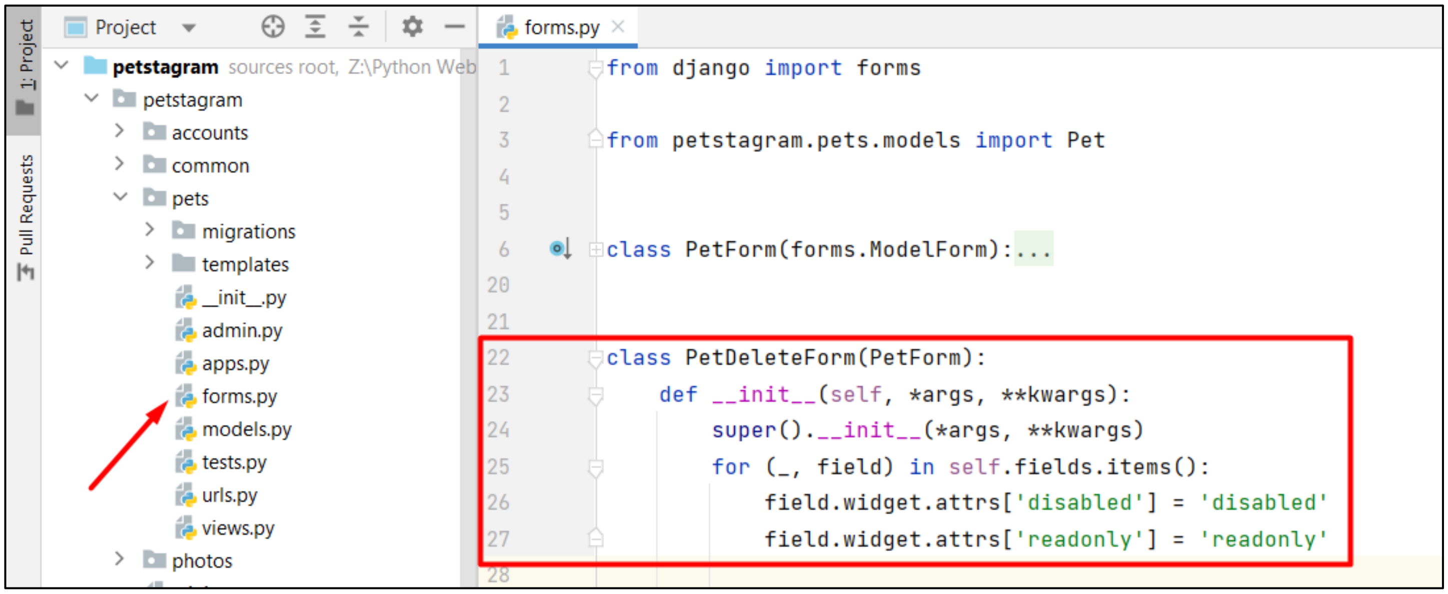
```html
{% extends 'base.html' %}

{% block content %}
    <!-- Start Edit Pet Section -->
    <div class="edit-delete edit-photo">
        <h2>Edit Pet</h2>
        <!-- Start Edit Pet From -->
        <form method="post">
            {% csrf_token %}
           {{ form.as_p }}
            <!-- Edit Pet Button -->
            <button class="edit-btn" type="submit">Edit</button>
        </form>
        <!-- End Edit Pet Form -->

    </div>
    <!-- End Edit Pet Section -->

{% endblock %}
```

Check if the pet edit functionality works correctly:


Petstagram
127.0.0.1:8000/pets/username/pet/sasho-5/edit/
Petstagram
Add Pet
Add Photo
Login
Register
Profile
Edit Pet
Pet Name: Sasho
Date of Birth: 03/01/2022
Link to Image: https://hips.hearstapps.com/hmg-prod.s3.amazon
Edit
© 2022 SOFTUNI WORKSHOP FOR PYTHON WEB MODULE

Creating a Pet Delete Form

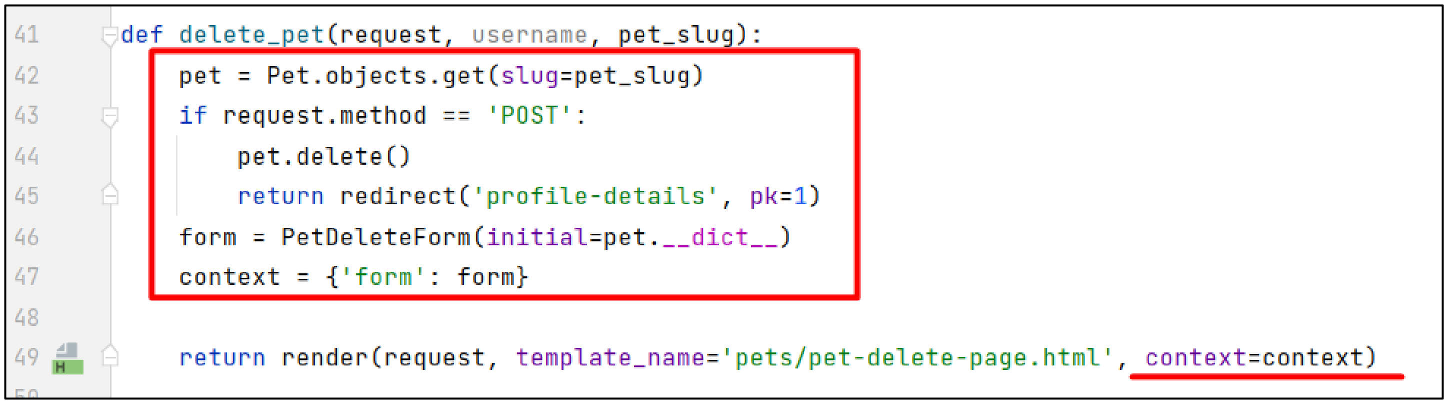
Last for the pet form section, we will **create a pet delete from** and functionality. First, open the **`pets/forms.py`** file and we will **add the delete form** that **inherits from the `PetForm`** and **disables all fields**, and set them to be **read-only**:



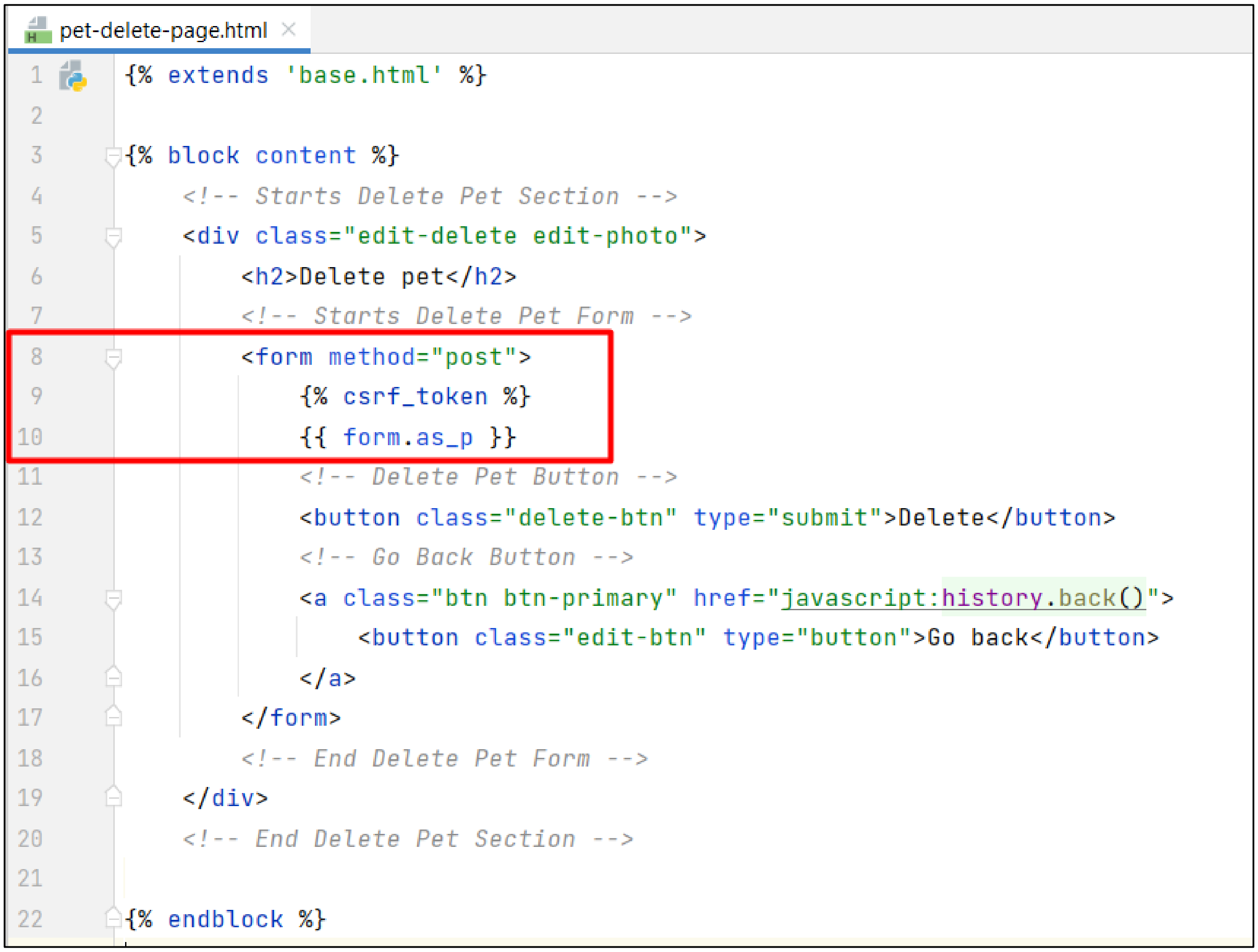
```python
from django import forms

from petstagram.pets.models import Pet


class PetForm(forms.ModelForm):...


class PetDeleteForm(PetForm):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        for (_, field) in self.fields.items():
            field.widget.attrs['disabled'] = 'disabled'
            field.widget.attrs['readonly'] = 'readonly'
```

Then, we will **write the view functionality**. First, we will try to **get the pet object** we want to delete. Next, if the request method is **POST** we will **delete it** and we will **redirect to the profile details page**. If the method is **GET** we will **generate a**

form with the initial pet data:


```python
def delete_pet(request, username, pet_slug):
    pet = Pet.objects.get(slug=pet_slug)
    if request.method == 'POST':
        pet.delete()
        return redirect('profile-details', pk=1)
    form = PetDeleteForm(initial=pet.__dict__)
    context = {'form': form}

    return render(request, template_name='pets/pet-delete-page.html', context=context)
```

Finally, we will refactor the **`pet-delete-page.html`** template:



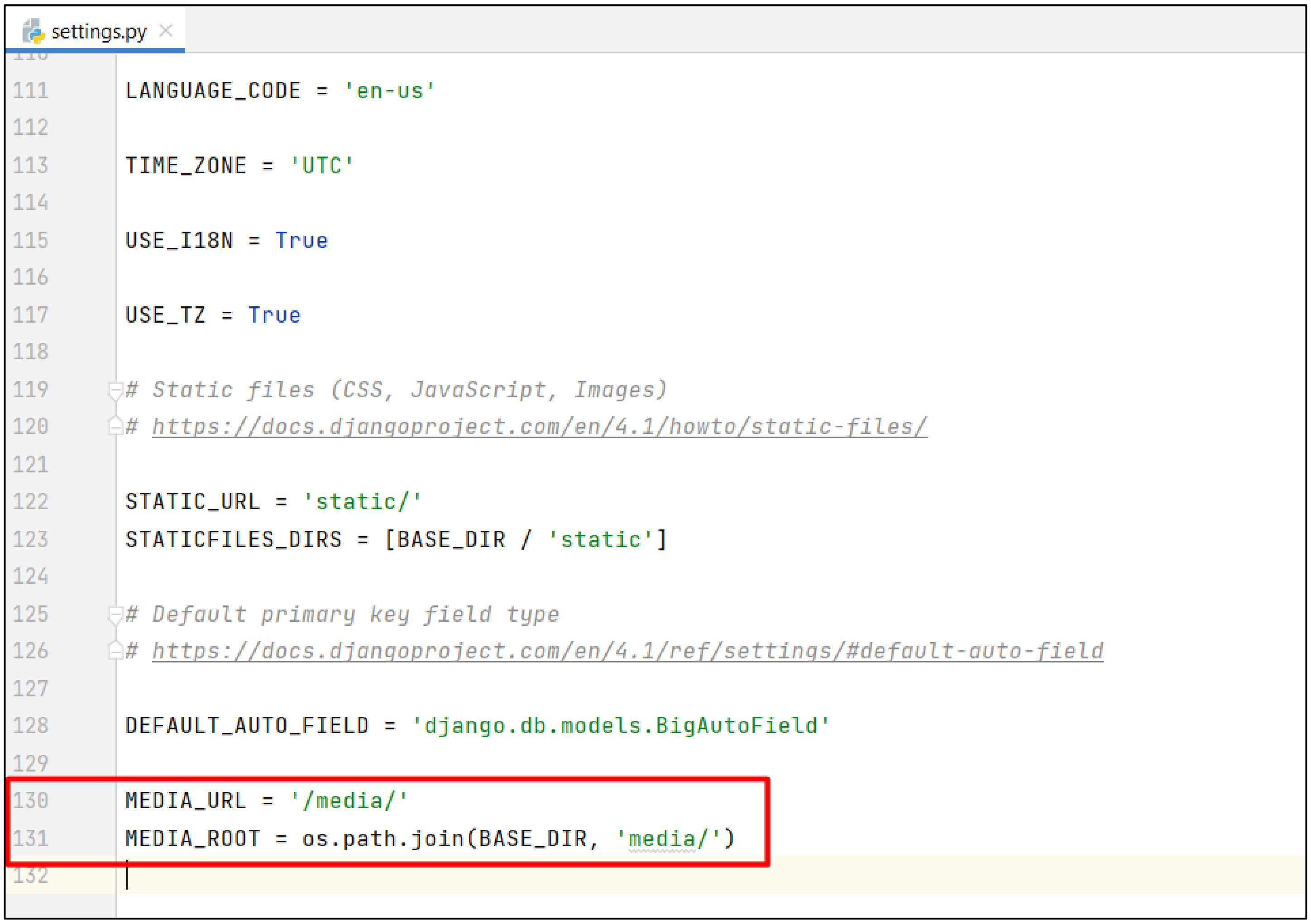
```html
{% extends 'base.html' %}

{% block content %}
    <!-- Starts Delete Pet Section -->
    <div class="edit-delete edit-photo">
        <h2>Delete pet</h2>
        <!-- Starts Delete Pet Form -->
        <form method="post">
            {% csrf_token %}
            {{ form.as_p }}
            <!-- Delete Pet Button -->
            <button class="delete-btn" type="submit">Delete</button>
            <!-- Go Back Button -->
            <a class="btn btn-primary" href="javascript:history.back()">
                <button class="edit-btn" type="button">Go back</button>
            </a>
        </form>
        <!-- End Delete Pet Form -->
    </div>
    <!-- End Delete Pet Section -->

{% endblock %}
```

Check if the form works correctly.

2. Workshop - Part 3.2

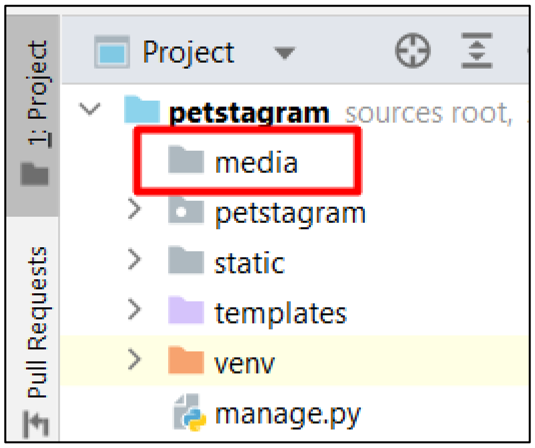
Working with Media Files

Next, we want to **create a photo creation and edition forms**. However, first, we need to make some changes to our project to work with media files. Let us **open the `settings.py` file** and **add the following settings**:



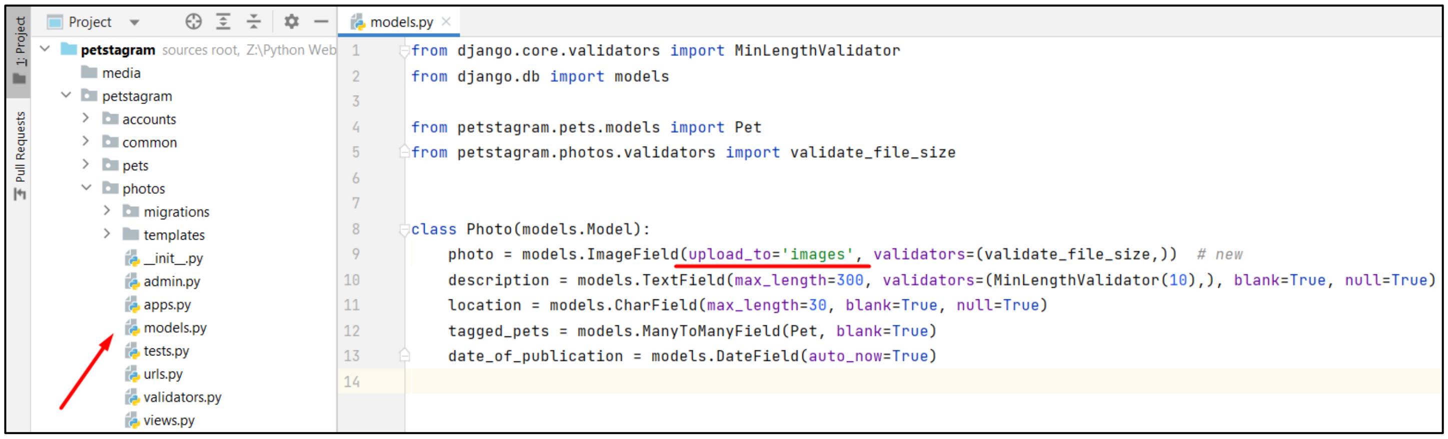
```python
LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.1/howto/static-files/

STATIC_URL = 'static/'
STATICFILES_DIRS = [BASE_DIR / 'static']

# 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/')
```

Now, we need to **create a media directory** on the **`manage.py`** level:



```
petstagram  sources root,
    media
    petstagram
    static
    templates
    venv
    manage.py
```

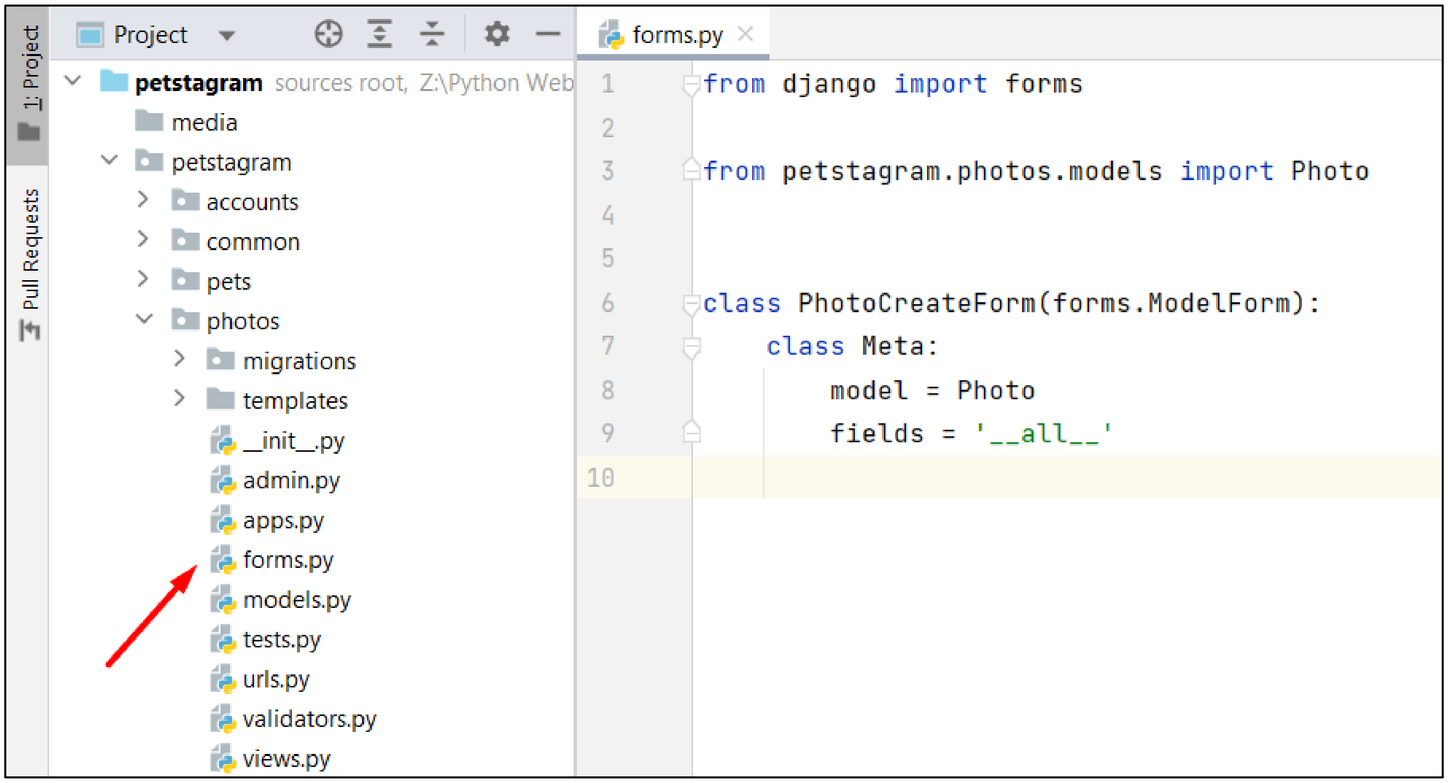
Next, let us open the **`photos/models.py`** file and **add an "`upload_to`" argument** in our photo field that will create an **"images" directory in the "media" folder** and will **save the uploaded photos** there:


```
from django.core.validators import MinLengthValidator
from django.db import models

from petstagram.pets.models import Pet
from petstagram.photos.validators import validate_file_size


class Photo(models.Model):
    photo = models.ImageField(upload_to='images', validators=(validate_file_size,))  # new
    description = models.TextField(max_length=300, validators=(MinLengthValidator(10),), blank=True, null=True)
    location = models.CharField(max_length=30, blank=True, null=True)
    tagged_pets = models.ManyToManyField(Pet, blank=True)
    date_of_publication = models.DateField(auto_now=True)
```

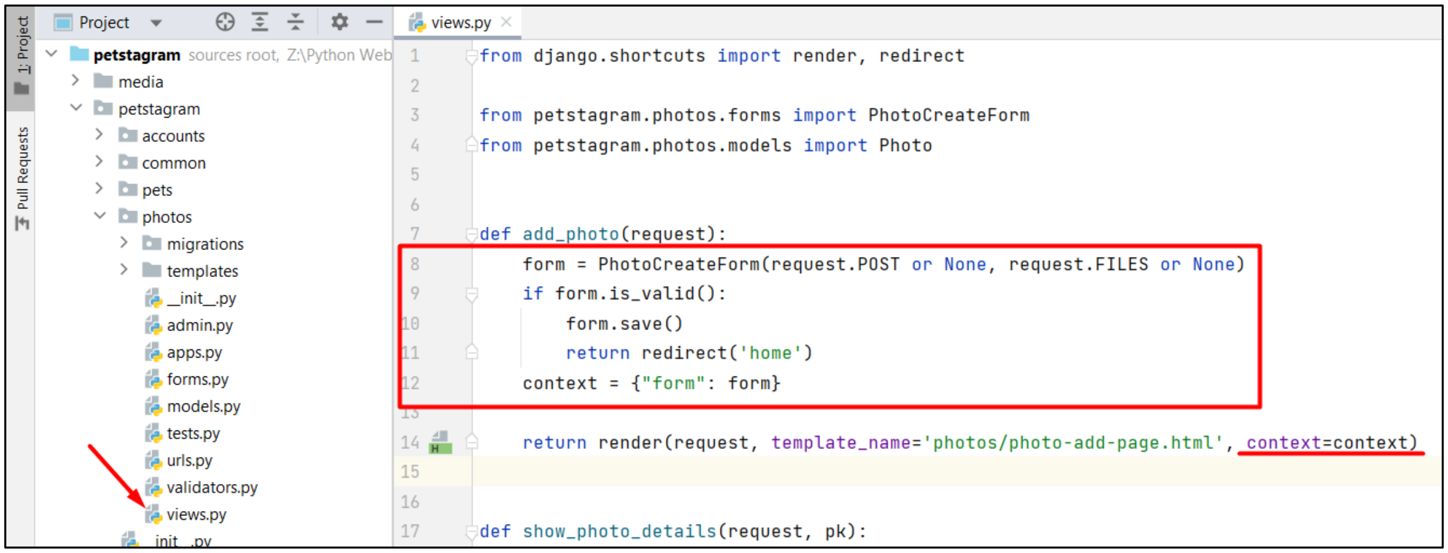
Make migrations and **migrate the changes** to the model.

Creating a Photo Creation Form

Let us start **adding the photo creation form**. **Create a new `forms.py`** file in the **`photos`** app and implement the form:

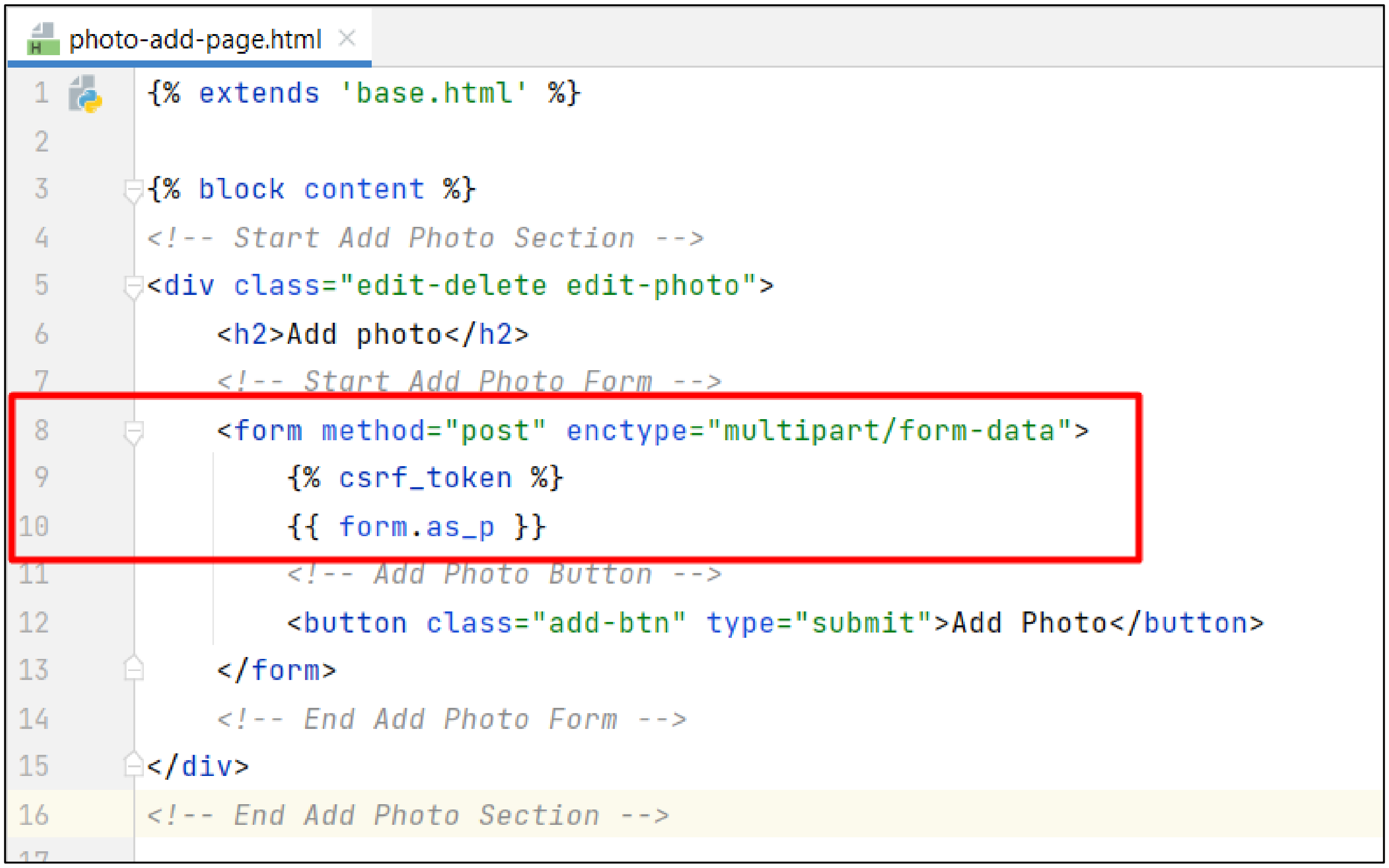

```
from django import forms

from petstagram.photos.models import Photo


class PhotoCreateForm(forms.ModelForm):
    class Meta:
        model = Photo
        fields = '__all__'
```

Next, **add the photo** form functionality in the **photos/views.py** file. Do not forget to add the **request.FILES** (it is a dictionary-like object containing all uploaded files) :



```python
from django.shortcuts import render, redirect

from petstagram.photos.forms import PhotoCreateForm
from petstagram.photos.models import Photo


def add_photo(request):
    form = PhotoCreateForm(request.POST or None, request.FILES or None)
    if form.is_valid():
        form.save()
        return redirect('home')
    context = {"form": form}

    return render(request, template_name='photos/photo-add-page.html', context=context)


def show_photo_details(request, pk):
```

And finally - refactor the **photo-add-page.html** template. Note: **request.FILES will only contain data if the request method is POST and the <form> has enctype="multipart/form-data"**:



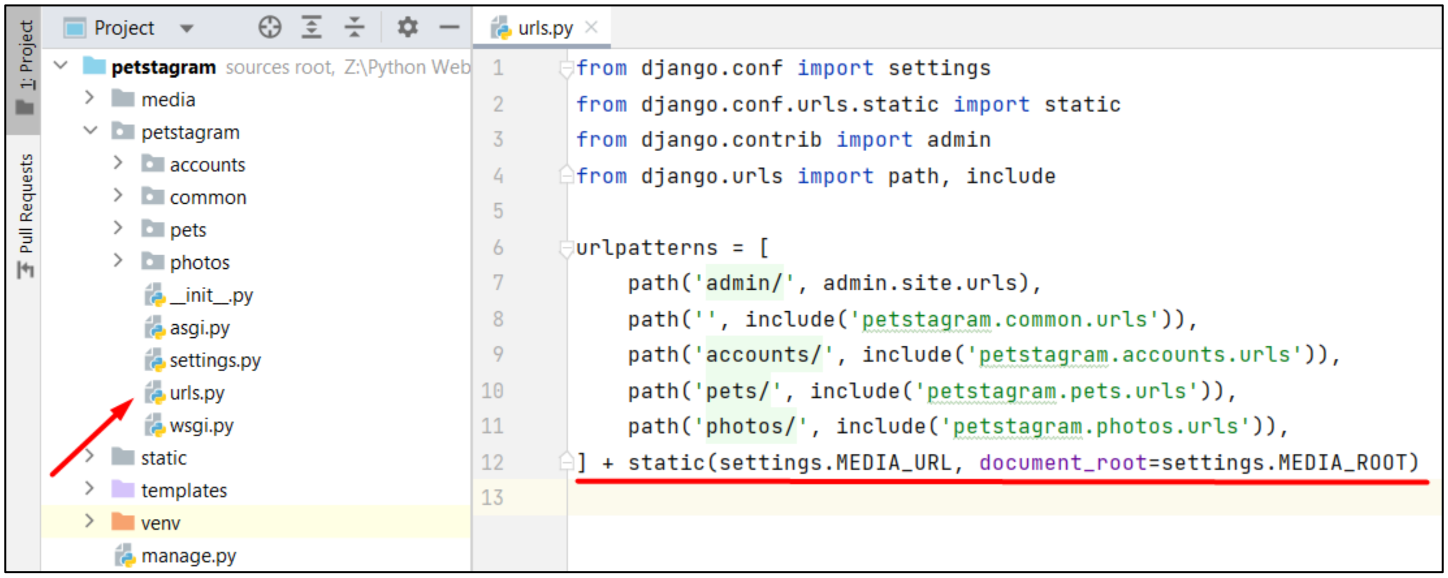
```html
{% extends 'base.html' %}

{% block content %}
<!-- Start Add Photo Section -->
<div class="edit-delete edit-photo">
    <h2>Add photo</h2>
    <!-- Start Add Photo Form -->
    <form method="post" enctype="multipart/form-data">
        {% csrf_token %}
        {{ form.as_p }}
        <!-- Add Photo Button -->
        <button class="add-btn" type="submit">Add Photo</button>
    </form>
    <!-- End Add Photo Form -->
</div>
<!-- End Add Photo Section -->
```

Check if the form works correctly.

Add Photo to Templates

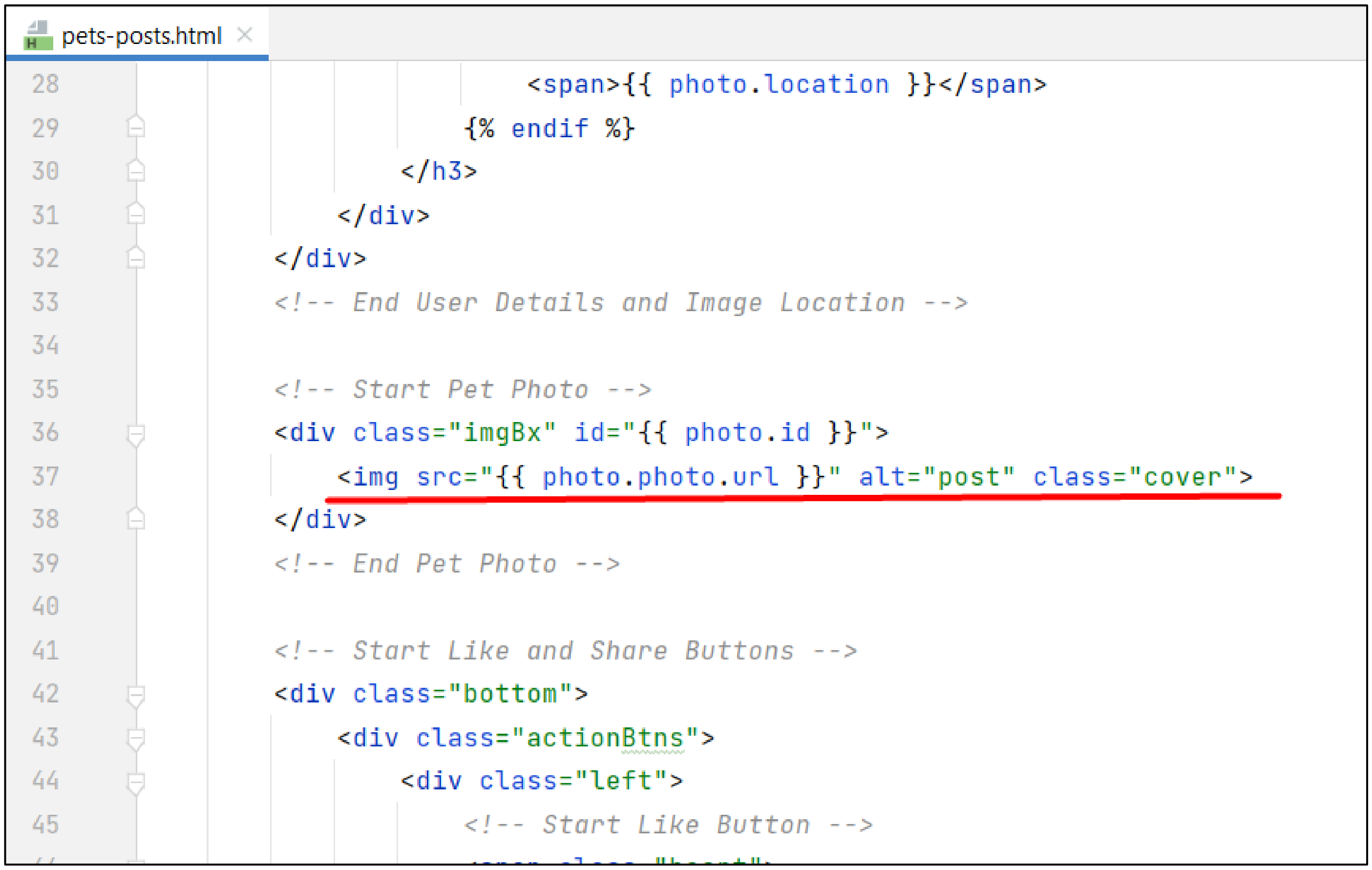
In the **settings.py** file we created the implementation of the media url - it loads a url like this one: 127.0.0.1:8000/media/images/image.jpeg. However, **it is not enough** - to visualize media files in Django it is needed to

add a special path that will find the file in the media folder and connect it to a media URL. Let us **open the `project/urls.py`** file and **add the functionality**:



```python
from django.conf import settings
from django.conf.urls.static import static
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')),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Now, it is time to **refactor the template**, so it visualizes the uploaded image. Let us open the **common** app **`pets-posts.html`** template and **find the "Start Pet Photo" comment**. Then, we will **add only the URL** of the uploaded image (Django will find it and will generate it):

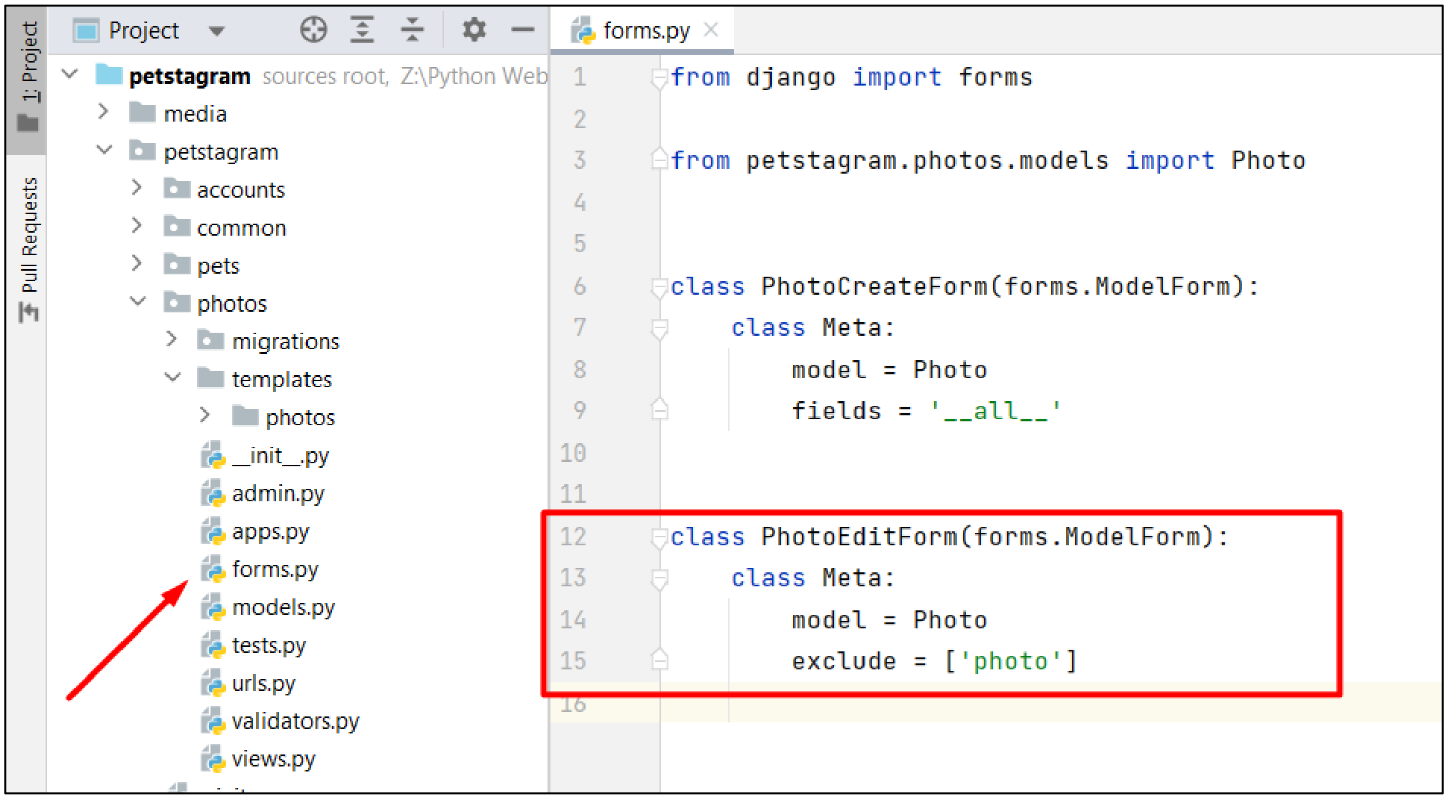


```html
                    <span>{{ photo.location }}</span>
                {% endif %}
            </h3>
        </div>
    </div>
    <!-- End User Details and Image Location -->

    <!-- Start Pet Photo -->
    <div class="imgBx" id="{{ photo.id }}">
        <img src="{{ photo.photo.url }}" alt="post" class="cover">
    </div>
    <!-- End Pet Photo -->

    <!-- Start Like and Share Buttons -->
    <div class="bottom">
        <div class="actionBtns">
            <div class="left">
                <!-- Start Like Button -->
```

Next, open the **`photo-details-page.html`** template and **add the photo URL**.

Create a Photo Edit Form

Next, let us **implement the photo edition form**. We do **NOT** want to edit the **photo**, so we will **exclude it from the form**:

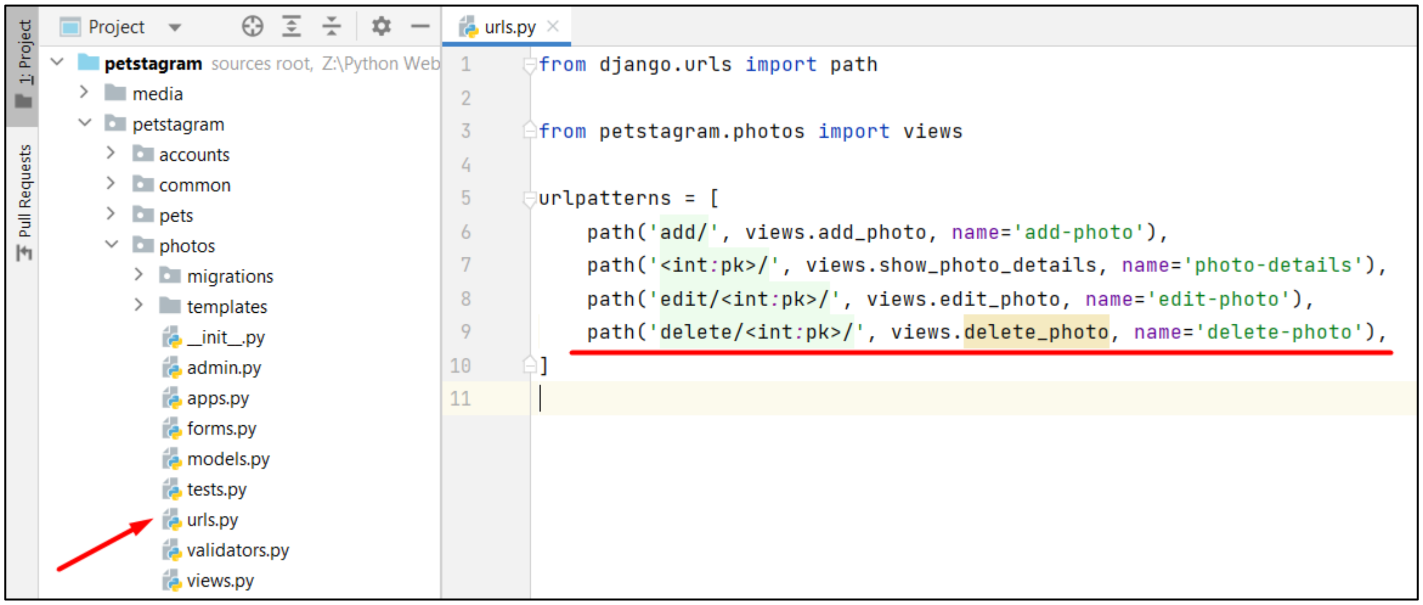


```python
from django import forms

from petstagram.photos.models import Photo


class PhotoCreateForm(forms.ModelForm):
    class Meta:
        model = Photo
        fields = '__all__'


class PhotoEditForm(forms.ModelForm):
    class Meta:
        model = Photo
        exclude = ['photo']
```

Open the **photos/views.py** file and **add the photo edit functionality** and refactor the template. When the photo is edited, the **user should be redirected to the photo details page**. Check if the functionality works correctly.

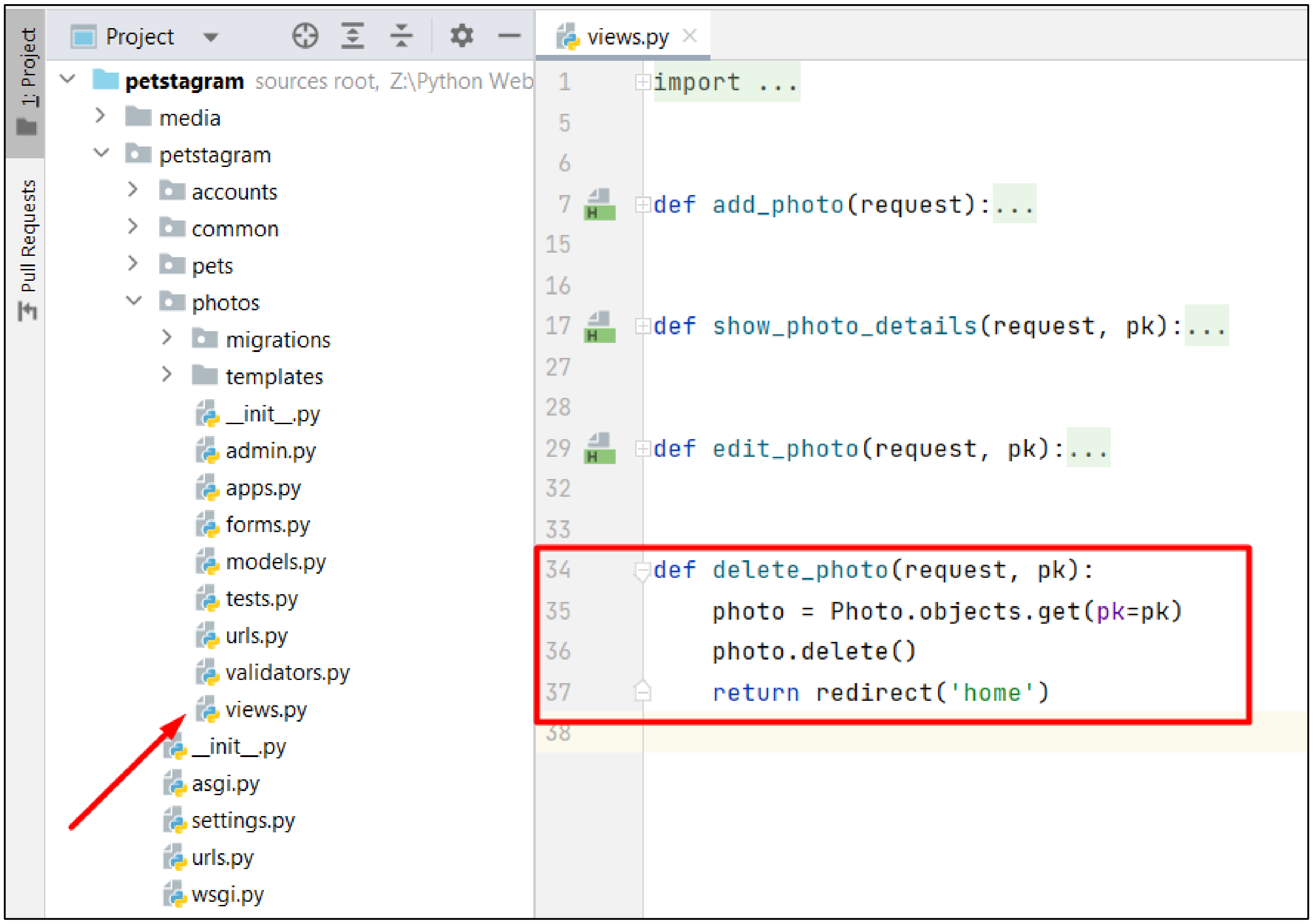
Deleting a Photo

Last for the photo, we will **implement the photo deletion functionality**. The photo is **directly deleted** after clicking on the **delete button** (on the photo details page). To start implementing the functionality, we need to **create a path with a corresponding view**:



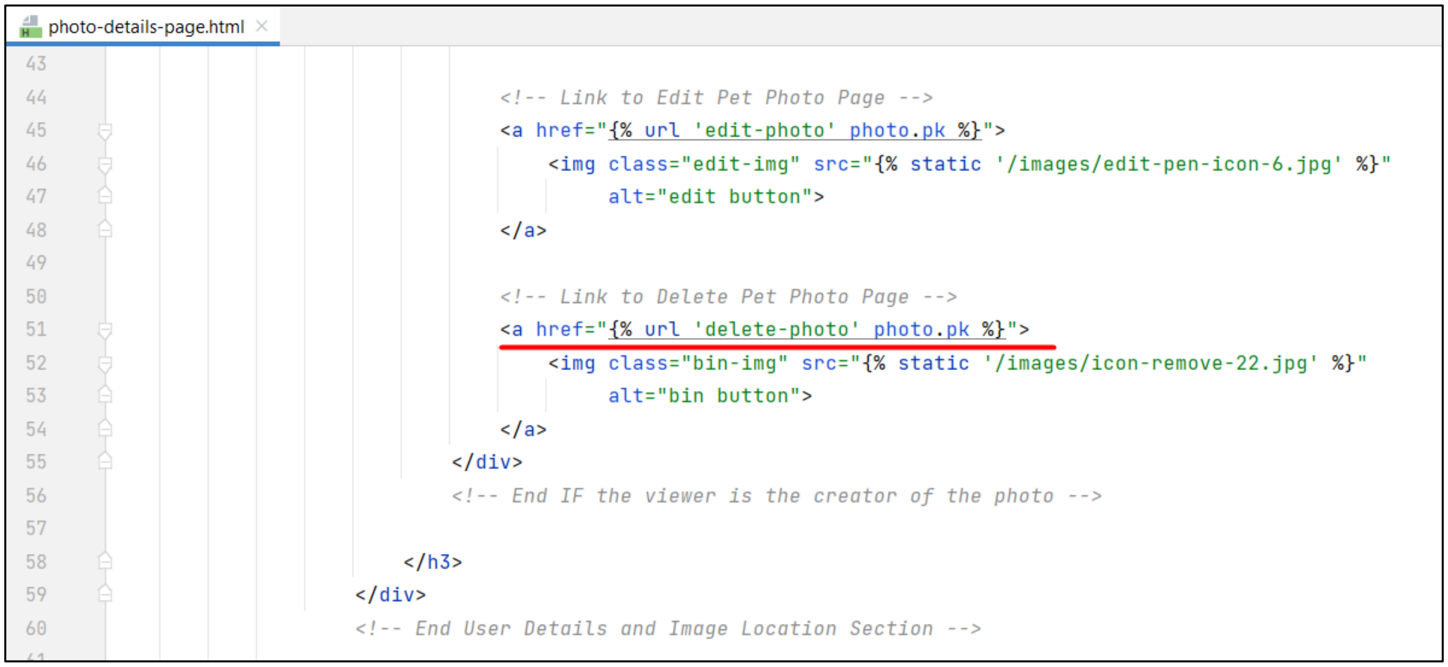
```python
from django.urls import path

from petstagram.photos import views

urlpatterns = [
    path('add/', views.add_photo, name='add-photo'),
    path('<int:pk>/', views.show_photo_details, name='photo-details'),
    path('edit/<int:pk>/', views.edit_photo, name='edit-photo'),
    path('delete/<int:pk>/', views.delete_photo, name='delete-photo'),
]
```

Now, let us create the **`delete_photo`** view. It **gets the pk of the photo**, **finds the photo object**, **deletes it,** and **redirects to the home page**:



```python
import ...


def add_photo(request):...


def show_photo_details(request, pk):...


def edit_photo(request, pk):...


def delete_photo(request, pk):
    photo = Photo.objects.get(pk=pk)
    photo.delete()
    return redirect('home')
```

Last, let us **refactor the template** by **adding the delete path to the delete button** in the **`photo-details-page.html`**. Find the "Link to Delete Pet Photo Page" comment and add the path:

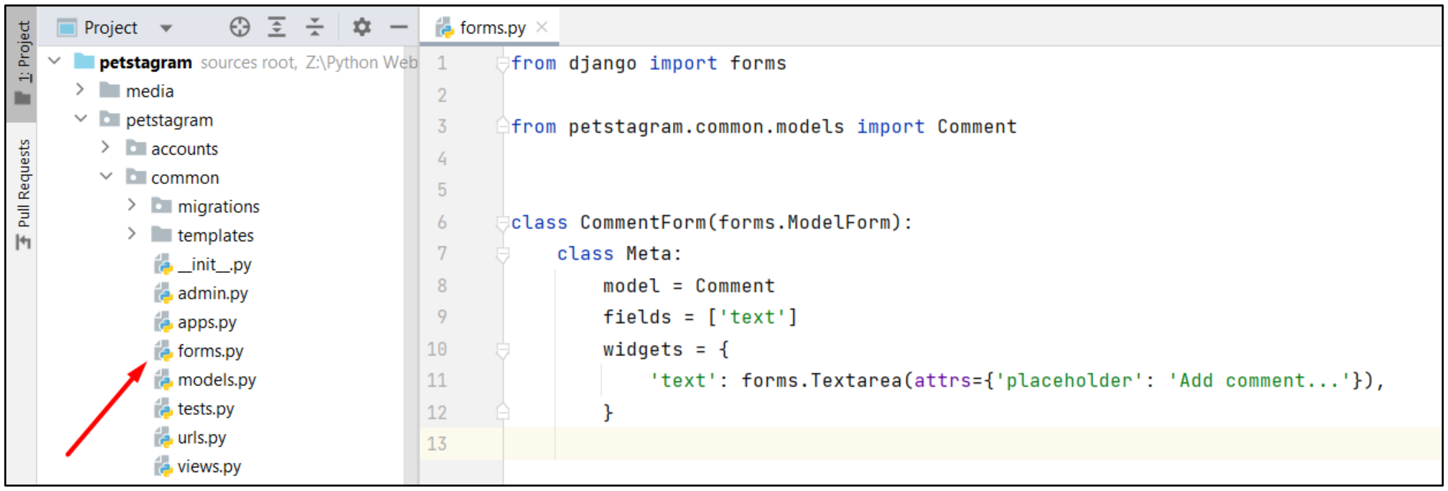


```html

                        <!-- Link to Edit Pet Photo Page -->
                        <a href="{% url 'edit-photo' photo.pk %}">
                            <img class="edit-img" src="{% static '/images/edit-pen-icon-6.jpg' %}"
                                 alt="edit button">
                        </a>

                        <!-- Link to Delete Pet Photo Page -->
                        <a href="{% url 'delete-photo' photo.pk %}">
                            <img class="bin-img" src="{% static '/images/icon-remove-22.jpg' %}"
                                 alt="bin button">
                        </a>
                    </div>
                    <!-- End IF the viewer is the creator of the photo -->

                </h3>
            </div>
            <!-- End User Details and Image Location Section -->
```

3. Workshop - Part 3.3

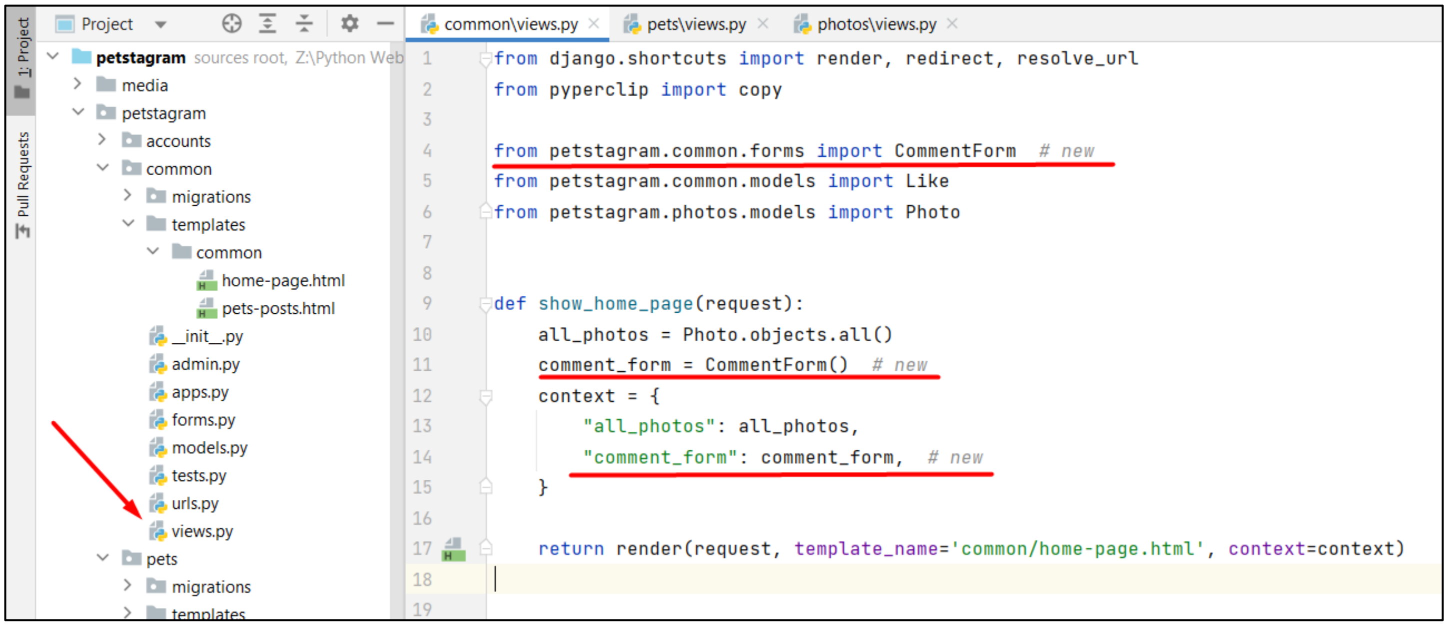
Creating a Comment Form

It is time to **start implementing the comment form**. In this project, our users can **NOT edit or delete their comments** once they post them on the app - so, the only thing needed is to **add a comment creation form**. Let us **add a `forms.py` file in the common app** and **create the form**. The only **visible field** we want to add is **the text field**. Also, it would be great to **add a placeholder** ("Add comment...") that will guide the user:



```
from django import forms

from petstagram.common.models import Comment


class CommentForm(forms.ModelForm):
    class Meta:
        model = Comment
        fields = ['text']
        widgets = {
            'text': forms.Textarea(attrs={'placeholder': 'Add comment...'}),
        }
```

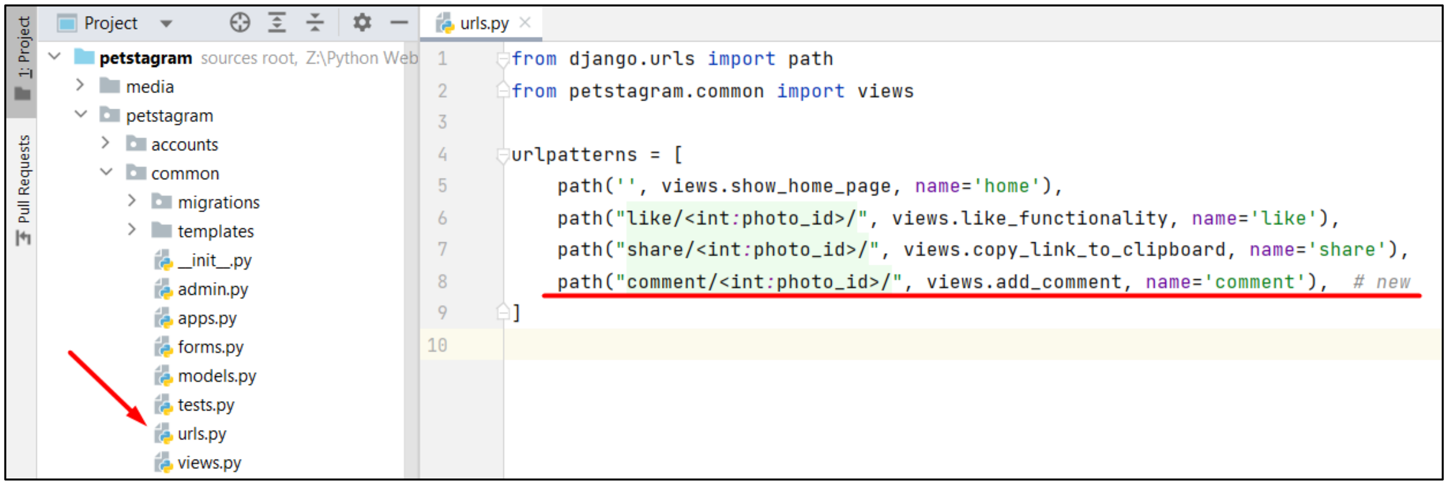
We want to **generate the comment form** on the **Home Page**, **Pet Details Page**, and on **Photo Details Page**. It means that we should add the form to **3 views** (**`show_home_page`**, **`show_pet_details`**, and **`show_photo_details`** views) and **2 templates** (**`pets-posts.html`** and **`photo-details-page.html`**). First, let us **add it to the Home page**. Open the **`common/views.py`** file and **add the form**:



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

from petstagram.common.forms import CommentForm  # new
from petstagram.common.models import Like
from petstagram.photos.models import Photo


def show_home_page(request):
    all_photos = Photo.objects.all()
    comment_form = CommentForm()  # new
    context = {
        "all_photos": all_photos,
        "comment_form": comment_form,  # new
    }

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

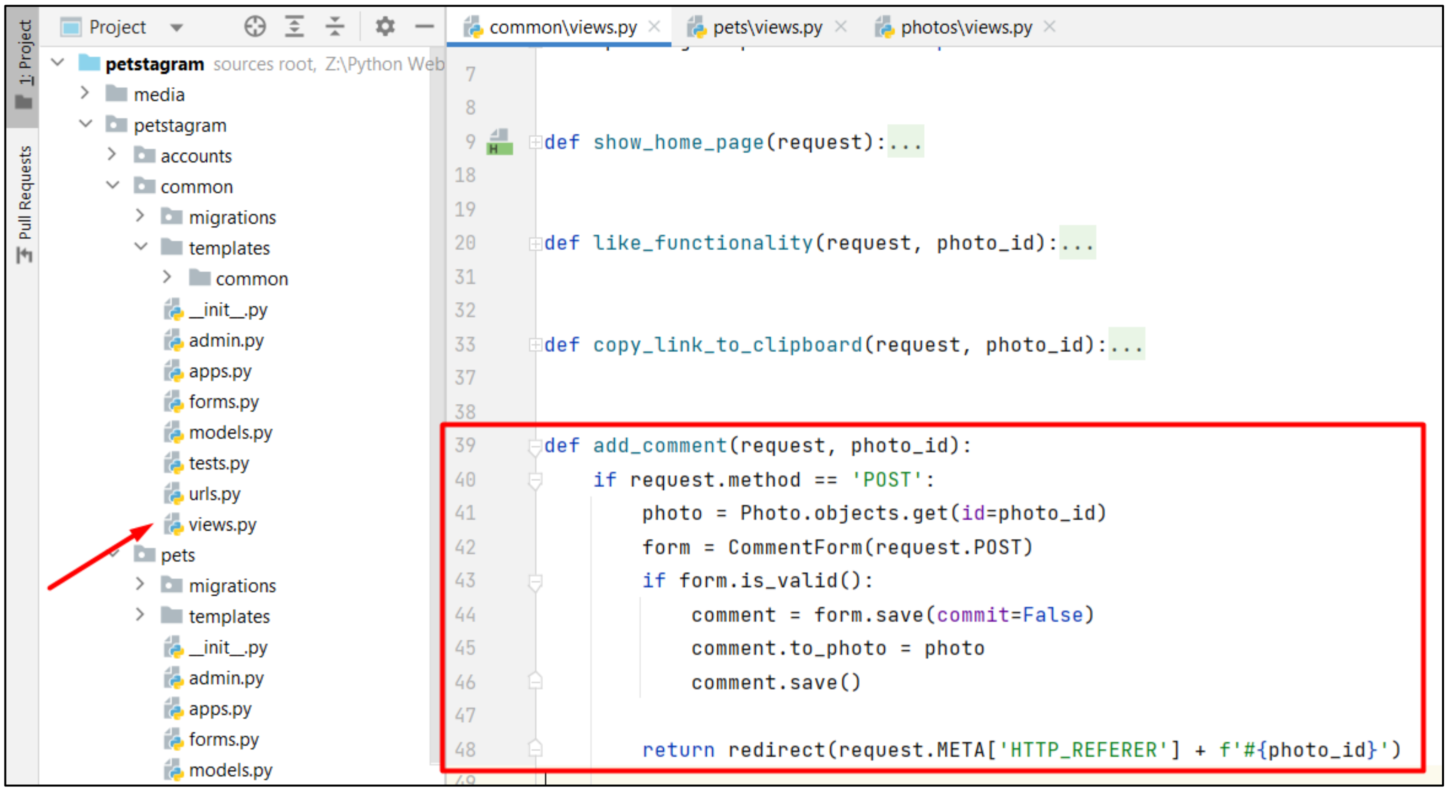
Next, we should **add the comment form functionality for saving the form**. The best way to do so is by **creating a new view** that will handle the business logic each time a user posts a comment (no matter on which of the 3 pages). First,

create a new path that will accept the photo id:



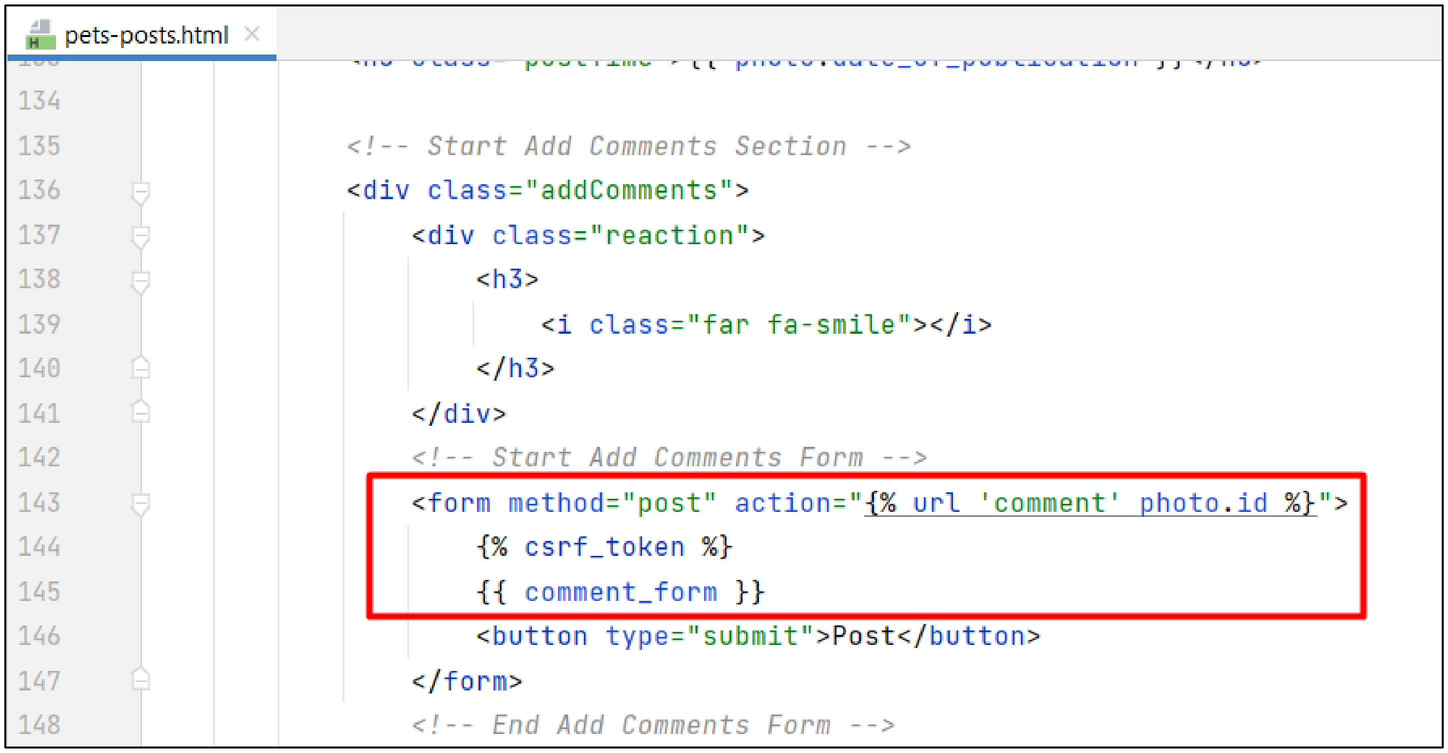
```python
from django.urls import path
from petstagram.common import views

urlpatterns = [
    path('', views.show_home_page, name='home'),
    path("like/<int:photo_id>/", views.like_functionality, name='like'),
    path("share/<int:photo_id>/", views.copy_link_to_clipboard, name='share'),
    path("comment/<int:photo_id>/", views.add_comment, name='comment'),  # new
]
```

Next, **create a view** that accepts the comment text from the user, finds the photo by the given photo id, and injects it into the form. Then, the **form is saved**, and the **user is redirected to the same place they last were**:



```python
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.save()

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

Now, we should **refactor the template**. Let us open the **`pets-posts.html`** template and find the "Start Add Comments Form" comment. In a difference from the other templates, here we **must add an action to the form**. When the form is submitted the **user must be redirected to the add_comment view**:

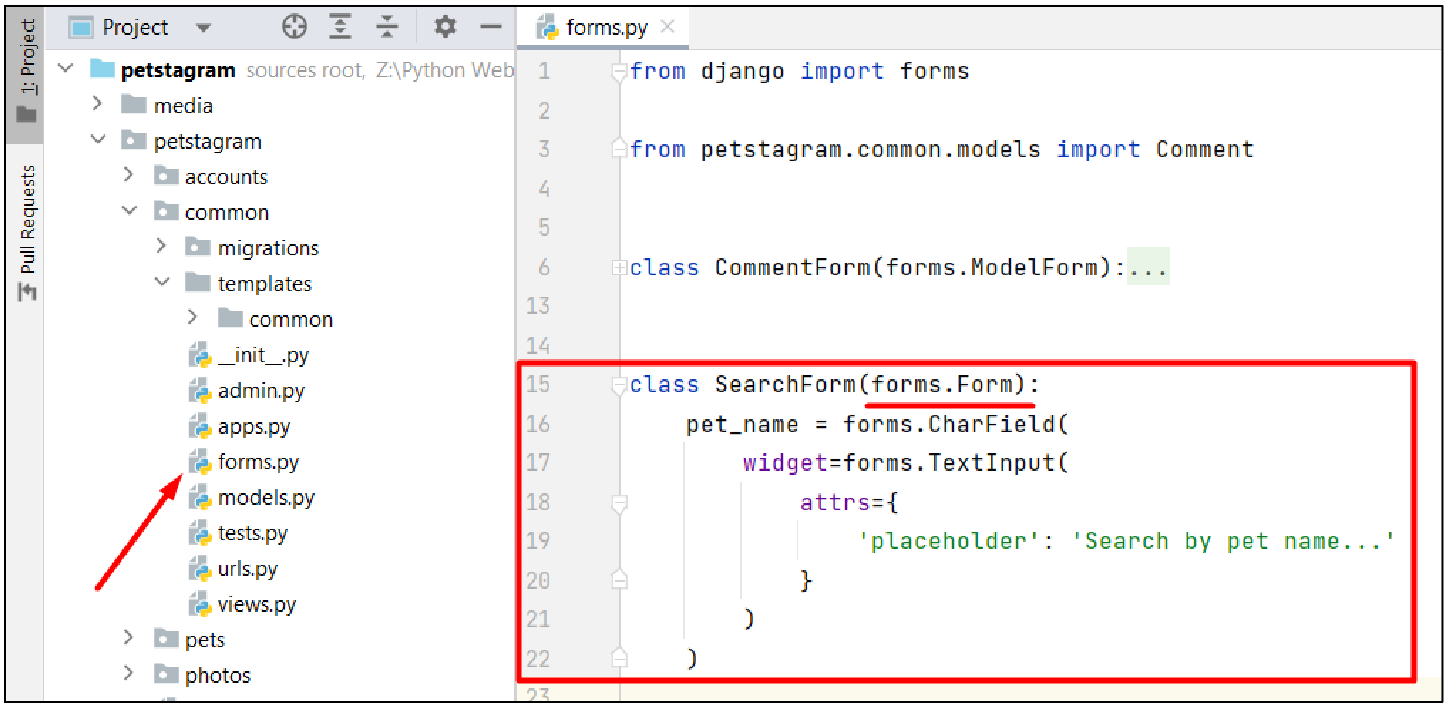


```
    <!-- Start Add Comments Section -->
    <div class="addComments">
        <div class="reaction">
            <h3>
                <i class="far fa-smile"></i>
            </h3>
        </div>
        <!-- Start Add Comments Form -->
        <form method="post" action="{% url 'comment' photo.id %}">
            {% csrf_token %}
            {{ comment_form }}
            <button type="submit">Post</button>
        </form>
        <!-- End Add Comments Form -->
```

We should **add the comment form to the pet details view and the photo details view with the photo details template**. **Check** if the form works correctly on the 3 pages.

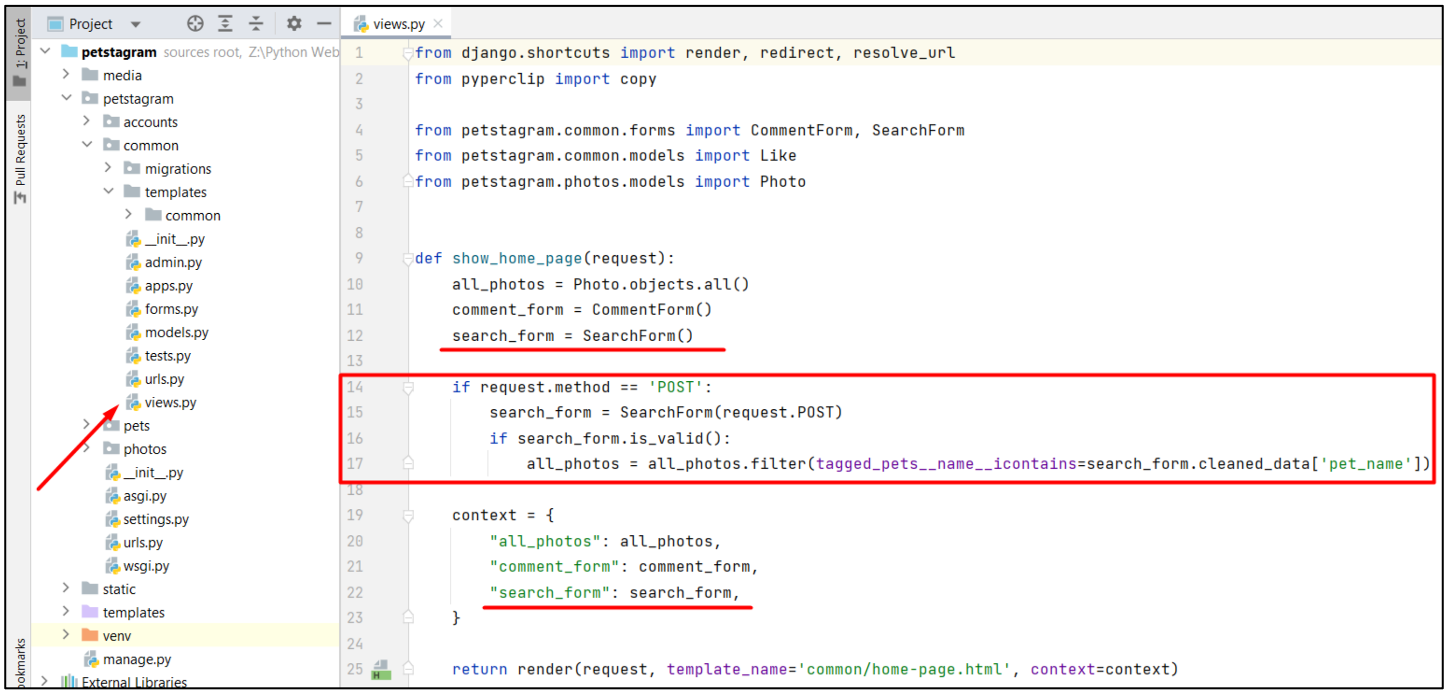
Creating a Search Form

Let us do something additional for our project. Let us **add a search bar functionality**. As you know, not every form in Django needs to be connected to a Model. In this case, we want to create a search form, but we do not want each search of a user to be saved to a database. So, we can **start directly by creating a form**. Our search bar will **receive a string that will search for a pet by its name**:



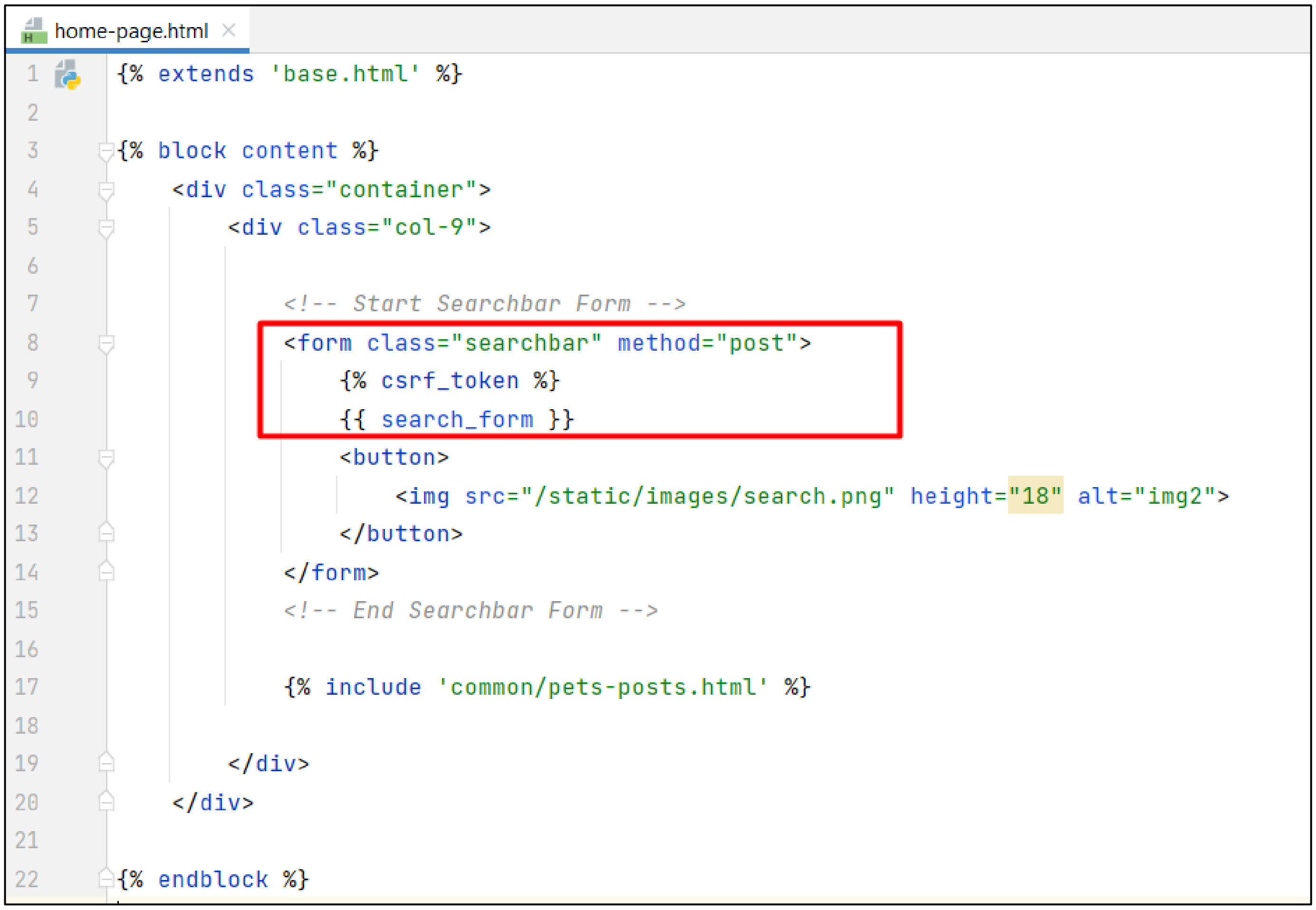
```python
from django import forms

from petstagram.common.models import Comment


class CommentForm(forms.ModelForm):...


class SearchForm(forms.Form):
    pet_name = forms.CharField(
        widget=forms.TextInput(
            attrs={
                'placeholder': 'Search by pet name...'
            }
        )
    )
```

Our search form is **positioned on the home page**, so we do not need to create an additional path. We can directly add the search form functionality in the **show_home_page** view. The search form is generated in the context. And if it is filled and the method is POST, we will **filter the photos** to find **all of them containing a tagged pet with the given name**. We **make the search case insensitive** by filtering with the **`icontains`** lookup:



```
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):
    all_photos = Photo.objects.all()
    comment_form = CommentForm()
    search_form = SearchForm()

    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,
    }

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

Last for this workshop, we will **add the form in the template**. Open the **`home-page.html`** template and **add the form**:



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

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

            <!-- Start Searchbar Form -->
            <form class="searchbar" method="post">
                {% csrf_token %}
                {{ search_form }}
                <button>
                    <img src="/static/images/search.png" height="18" alt="img2">
                </button>
            </form>
            <!-- End Searchbar Form -->

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

        </div>
    </div>

{% endblock %}
```