Workshop: Petstagram

This document contains the second part of the Petstagram Workshop. Today, we will create the **models** for the project. Then, we will connect **PostgreSQL** and migrate them. After that, we will work with the **Django admin site** to make CRUD operations with the models. And finally, we will **read** (select and filter) them **using python code**, and we will **present** the information for each model on the **"details" web pages**.

Note: we will NOT work with the profile/ user model 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 2.1

Creating the Pet Model

Let us start by **creating the Pet model**.

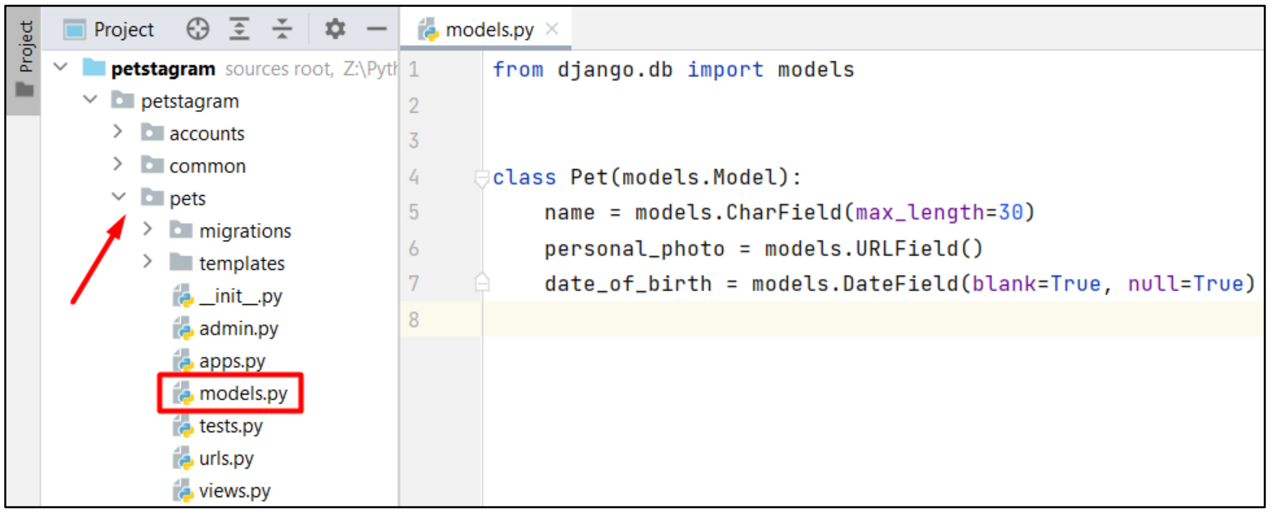
The fields **Name** and **Pet Photo** are **required**:

- **Name** - it should consist of a **maximum of 30 characters**.
- **Personal Pet Photo** - the user can **link a picture** using a URL

The field **date of birth** is **optional**:

- **Date of Birth** - pet's day, month, and year of birth

Open the **`pets/models.py`** file and let us create the model:

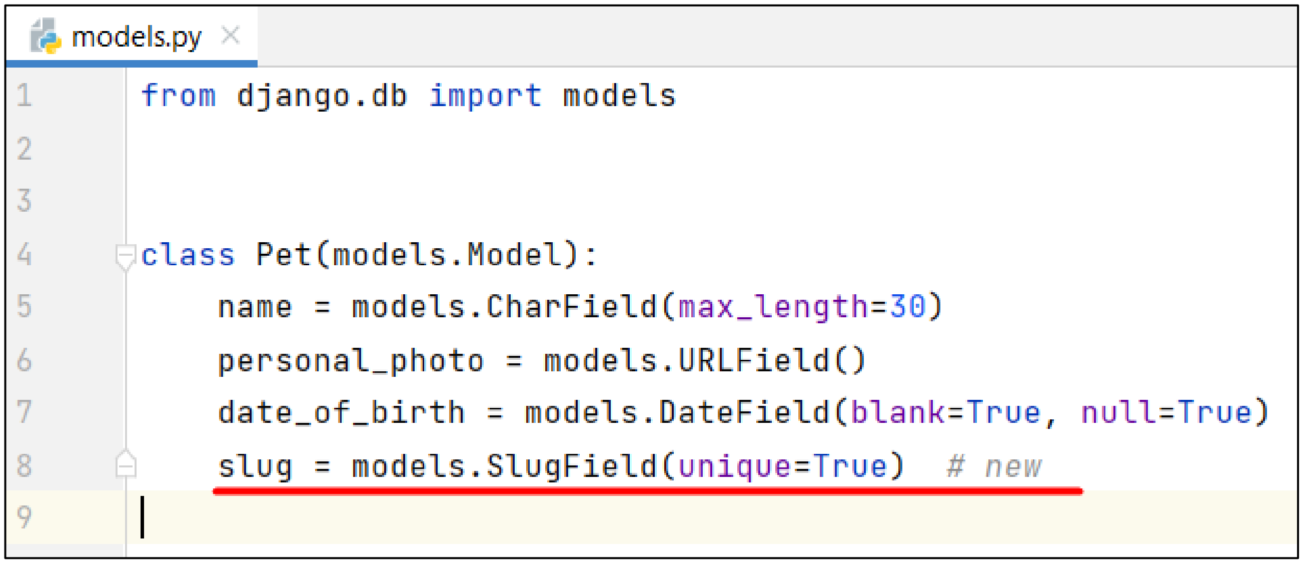


```python
from django.db import models


class Pet(models.Model):
    name = models.CharField(max_length=30)
    personal_photo = models.URLField()
    date_of_birth = models.DateField(blank=True, null=True)
```

There should be created **one more field** that will be **auto-populated** with the following information:

- **Slug** - a slug automatically generated using the **pet's name and the pet's id, separated by a "-"** (dash).

The **slug is part of the URL** and as you know **each URL should be unique**:

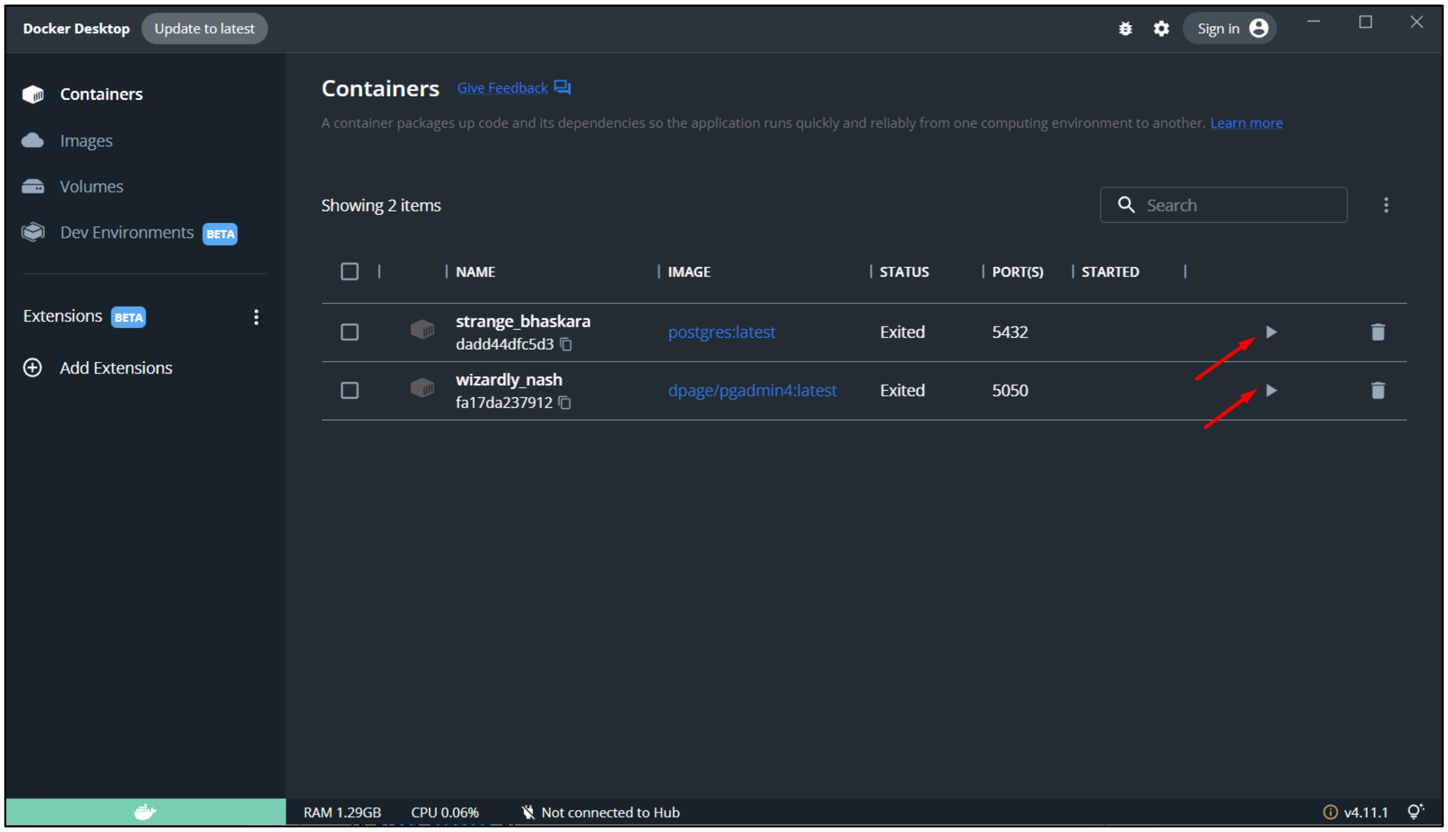


models.py

```python
from django.db import models


class Pet(models.Model):
    name = models.CharField(max_length=30)
    personal_photo = models.URLField()
    date_of_birth = models.DateField(blank=True, null=True)
    slug = models.SlugField(unique=True)  # new

```

Setting up the Database

Up to this moment, our Pet model is created and now we need to **migrate it to the database**. First, **start** the **PostgreSQL** container and the **PgAdmin** container (or you can create new ones):



Docker Desktop · Update to latest · Sign in

Containers · Images · Volumes · Dev Environments BETA · Extensions BETA · Add Extensions

Containers Give Feedback

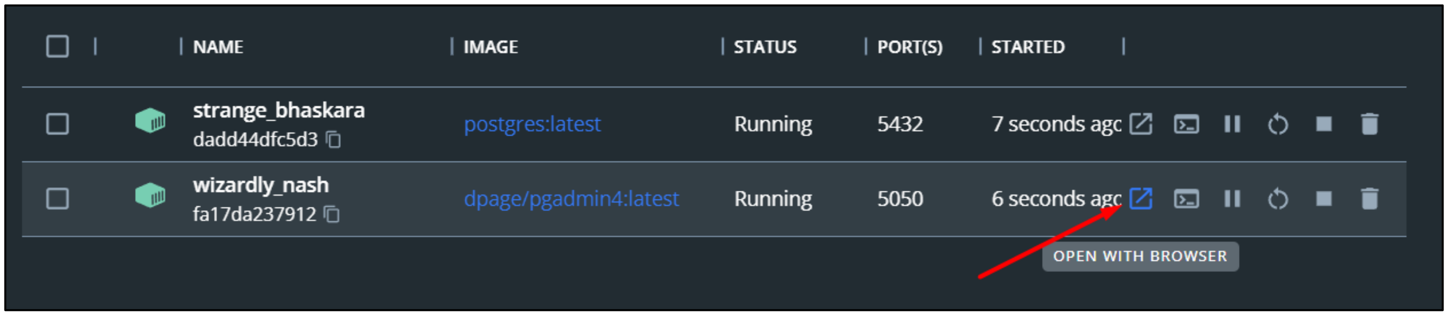
A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Showing 2 items

NAME	IMAGE	STATUS	PORT(S)	STARTED
strange_bhaskara dadd44dfc5d3	postgres:latest	Exited	5432	
wizardly_nash fa17da237912	dpage/pgadmin4:latest	Exited	5050	

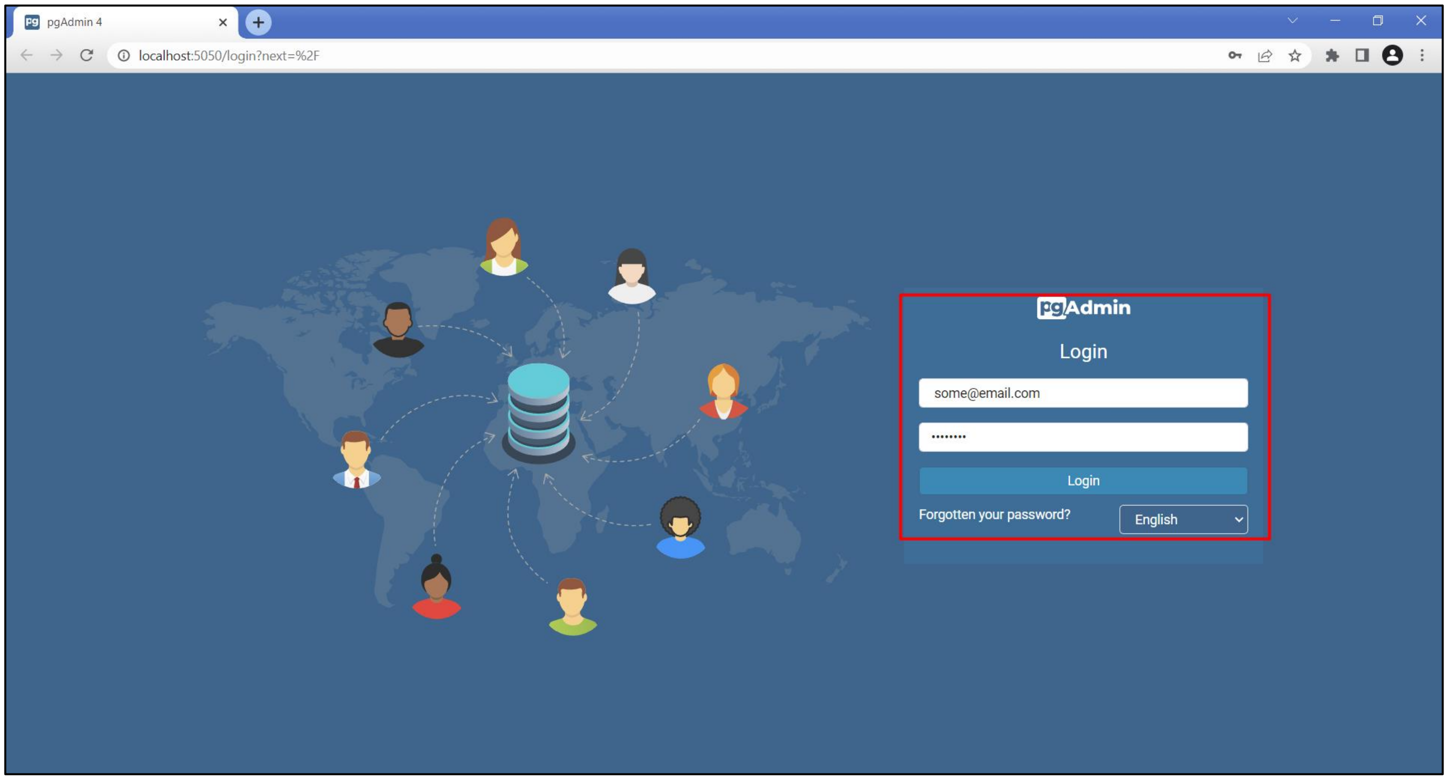
RAM 1.29GB · CPU 0.06% · Not connected to Hub · v4.11.1

Wait a few seconds and **open the pgAdmin using the browser**:

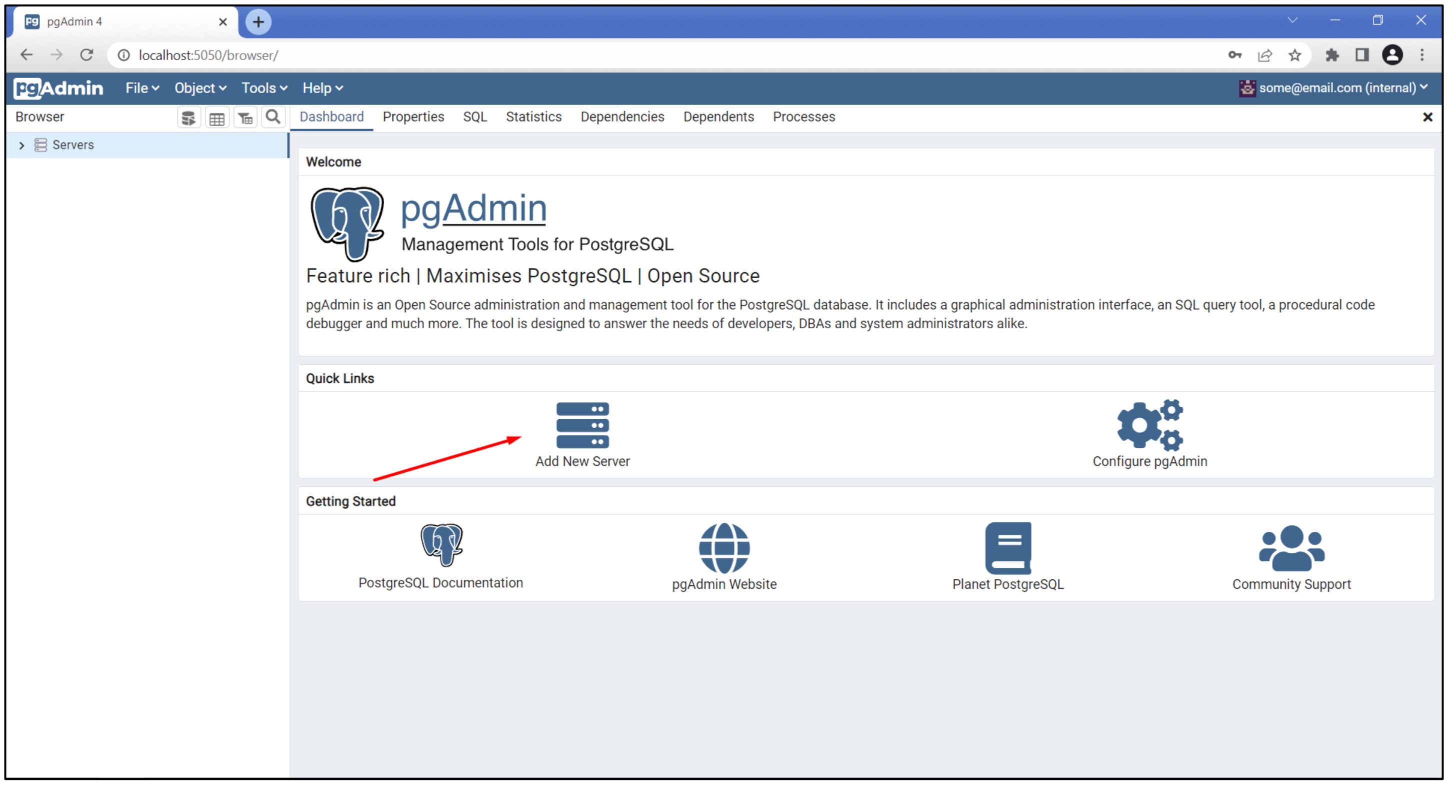


NAME	IMAGE	STATUS	PORT(S)	STARTED
strange_bhaskara dadd44dfc5d3	postgres:latest	Running	5432	7 seconds ago
wizardly_nash fa17da237912	dpage/pgadmin4:latest	Running	5050	6 seconds ago

Then, log in with the **email** and **password you configure** (when creating the pgAdmin container):

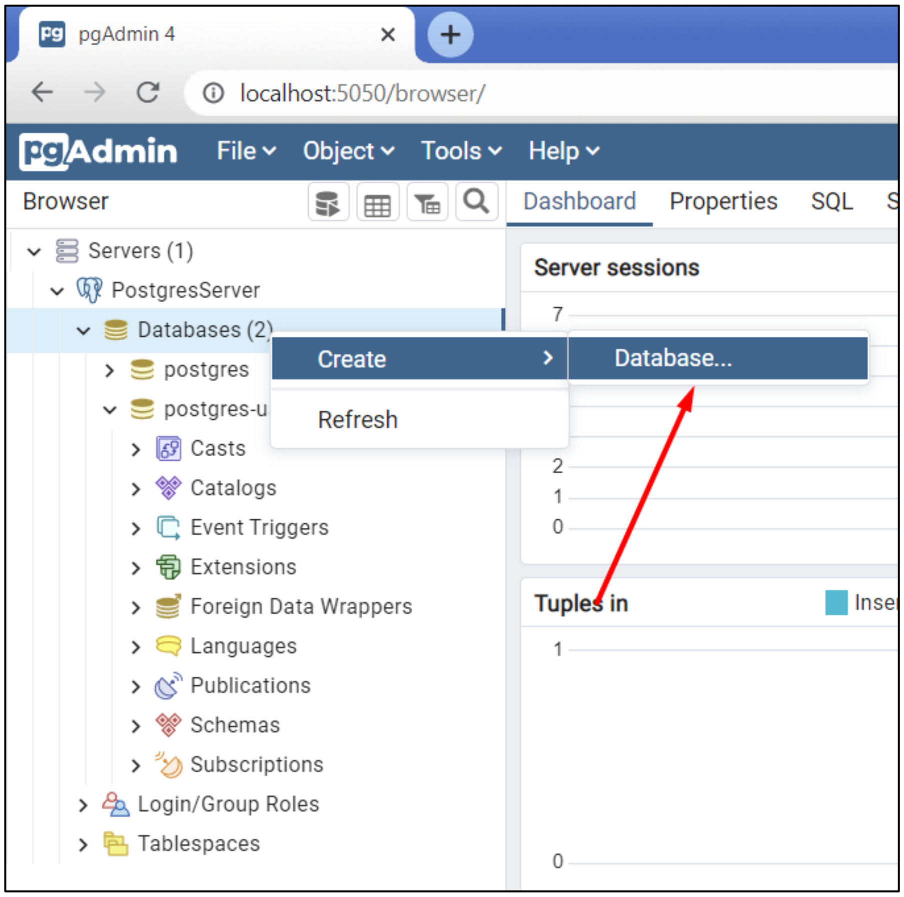


You can **add a new server** to work with **or use the created one**:

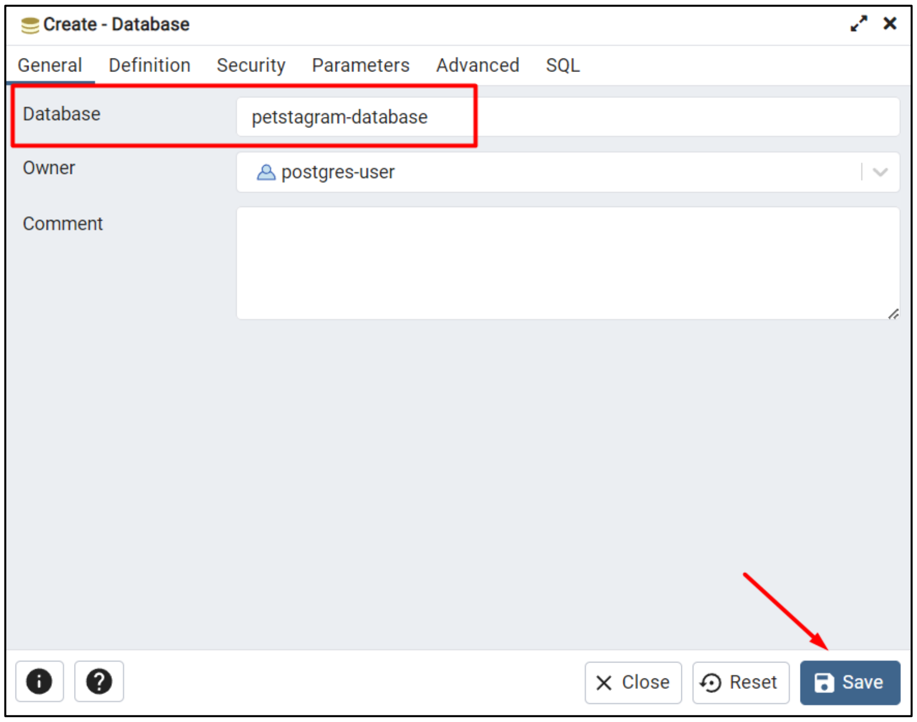

pgAdmin 4
localhost:5050/browser/
pgAdmin File Object Tools Help
some@email.com (internal)
Browser
Servers
Dashboard Properties SQL Statistics Dependencies Dependents Processes
Welcome
pgAdmin
Management Tools for PostgreSQL
Feature rich | Maximises PostgreSQL | Open Source
Quick Links
Add New Server
Configure pgAdmin
Getting Started
PostgreSQL Documentation
pgAdmin Website
Planet PostgreSQL
Community Support

pgAdmin is an Open Source administration and management tool for the PostgreSQL database. It includes a graphical administration interface, an SQL query tool, a procedural code debugger and much more. The tool is designed to answer the needs of developers, DBAs and system administrators alike.

Next, on the server, we will **create the database we will work with**. Right-click on the "Database" field and choose to create a database:

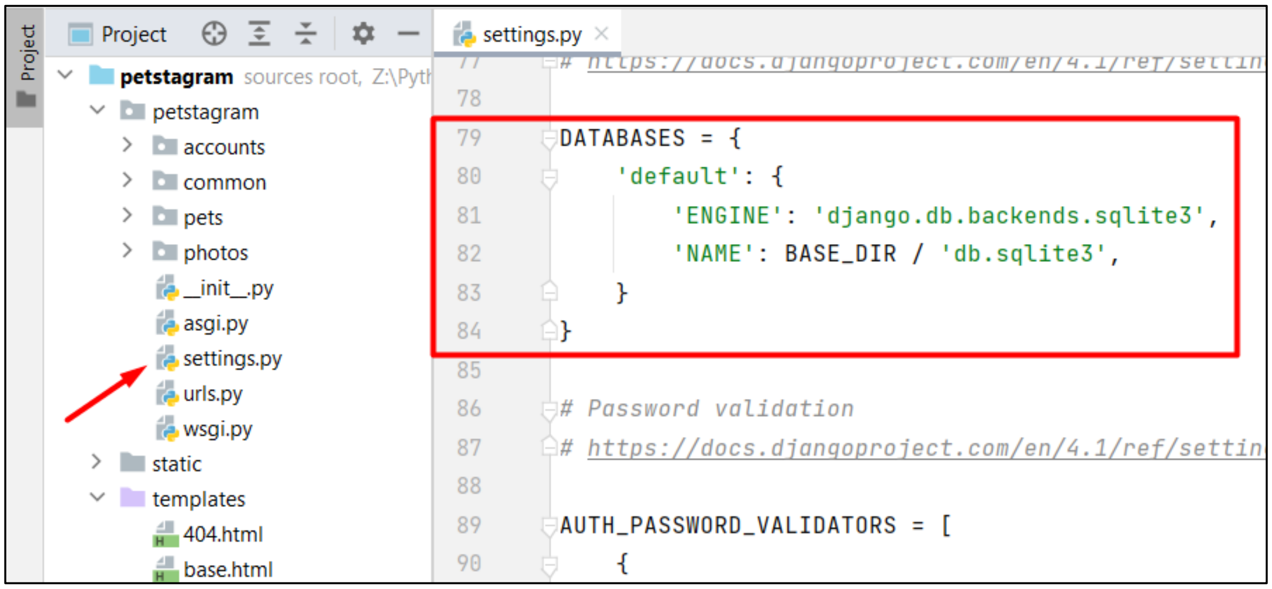


Let us use the name "**pestagram-database**" for the project and **save it**:



Database: petstagram-database
Owner: postgres-user
Comment:

Now, it is time to **configure it in the project**. Let us open the **`settings.py`** file and find the **DATABASES** setting:

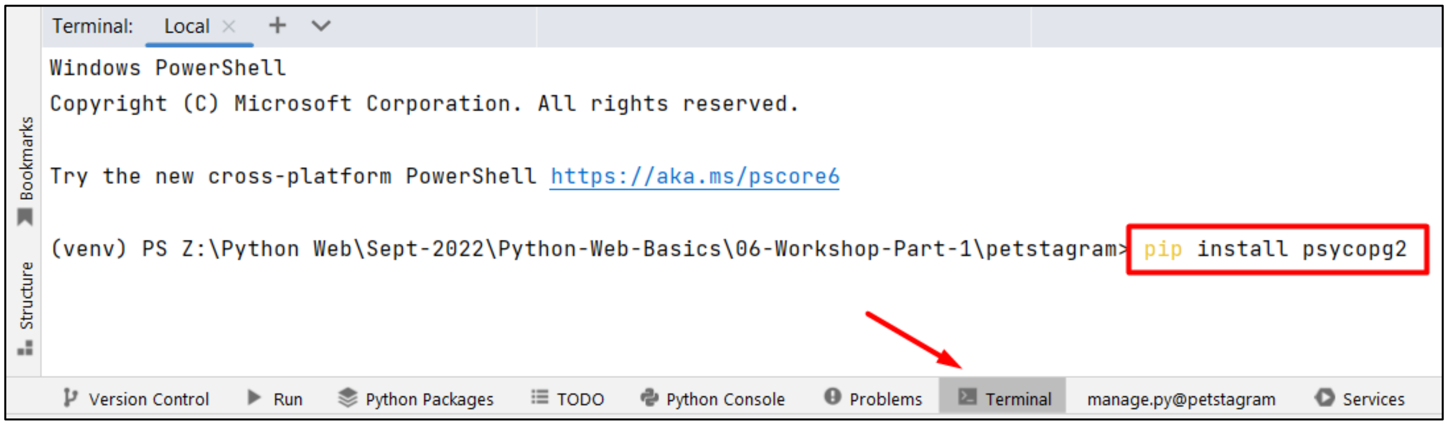


```
# https://docs.djangoproject.com/en/4.1/ref/settin

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

# Password validation
# https://docs.djangoproject.com/en/4.1/ref/settin

AUTH_PASSWORD_VALIDATORS = [
    {
```

Up until now, the project uses the default engine - SQLite. It is time to **write the configuration for the PostgreSQL**:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'petstagram-database',   # database name
        'USER': 'postgres-user',   # postgres user
        'PASSWORD': 'password',   # postgres password
        'HOST': '127.0.0.1',   # postgres host
        'PORT': '5432',   # postgres port
    }
}
```

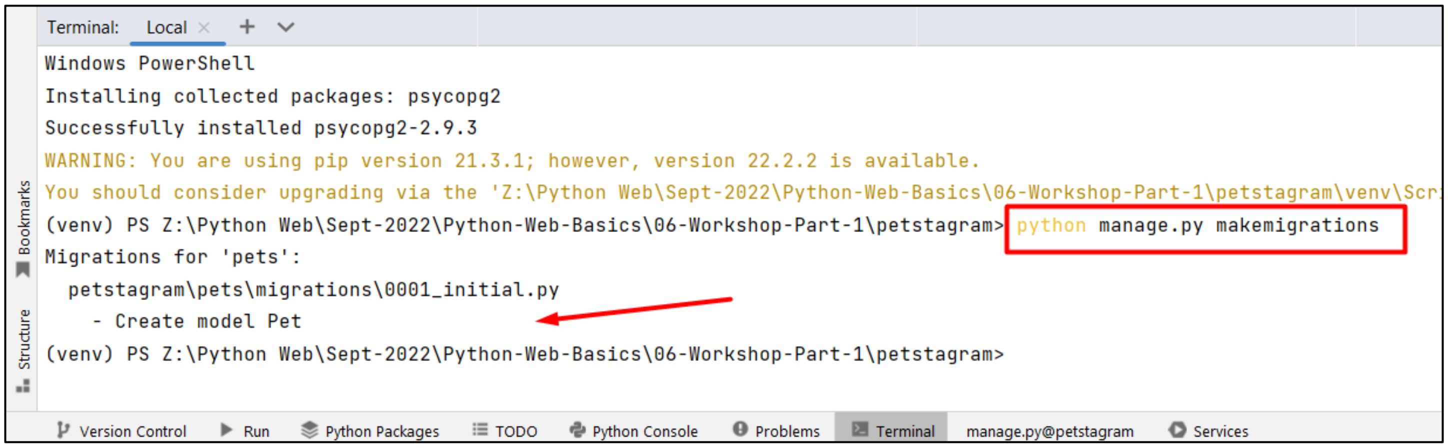
The next required step is to **install psycopg2**. Open the Terminal and write the command **"`pip install psycopg2`"**:



```
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> pip install psycopg2
```

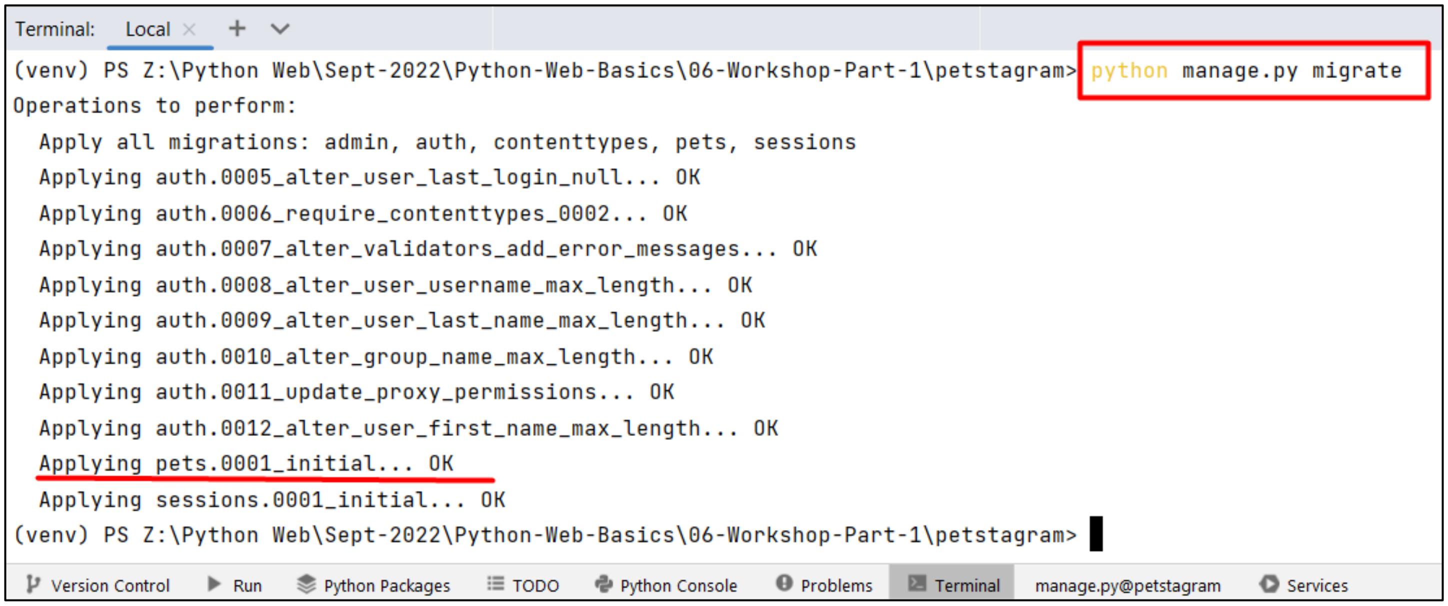
Migrate the Pet Model

When the installation is done, we can now **make the migration files** with the command "**`python manage.py makemigrations`**" and check if the migration file is successfully created:



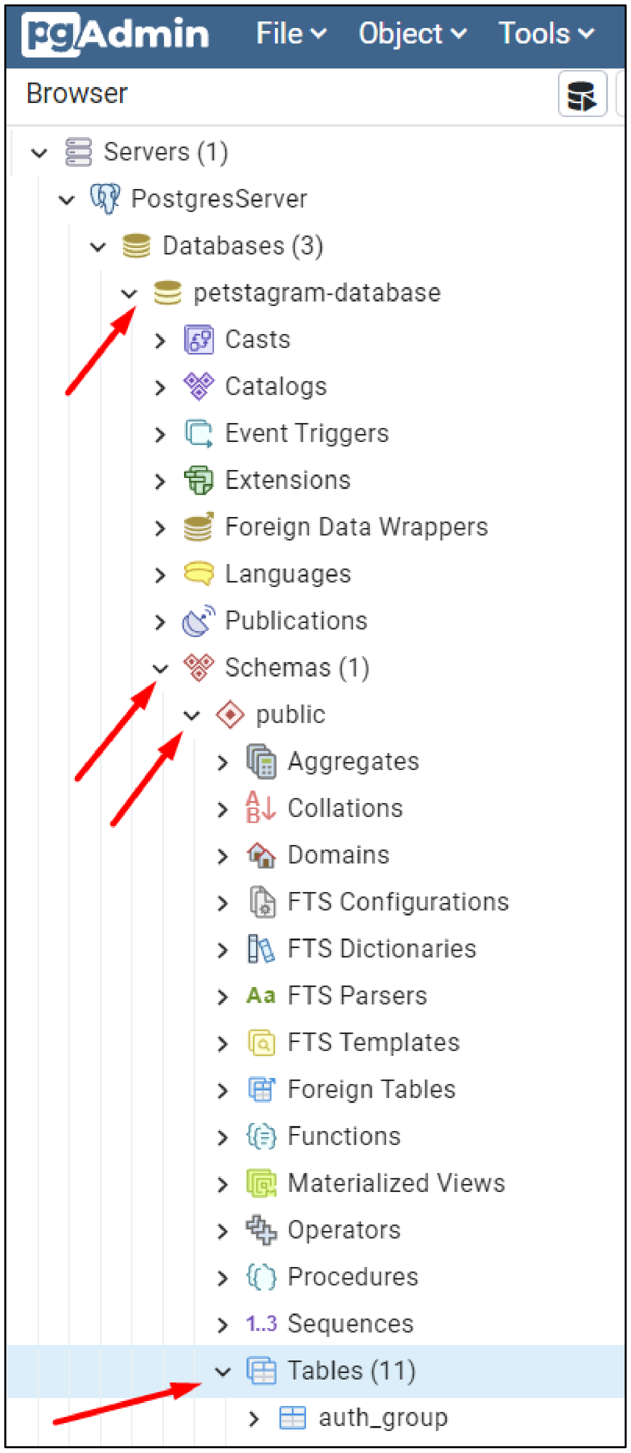
```
Windows PowerShell
Installing collected packages: psycopg2
Successfully installed psycopg2-2.9.3
WARNING: You are using pip version 21.3.1; however, version 22.2.2 is available.
You should consider upgrading via the 'Z:\Python Web\Sept-2022\Python-Web-Basics\06-Workshop-Part-1\petstagram\venv\Scr
(venv) PS Z:\Python Web\Sept-2022\Python-Web-Basics\06-Workshop-Part-1\petstagram> python manage.py makemigrations
Migrations for 'pets':
  petstagram\pets\migrations\0001_initial.py
    - Create model Pet
(venv) PS Z:\Python Web\Sept-2022\Python-Web-Basics\06-Workshop-Part-1\petstagram>
```

Then, we can **migrate the changes** to the database using the command "**`python manage.py migrate`**":

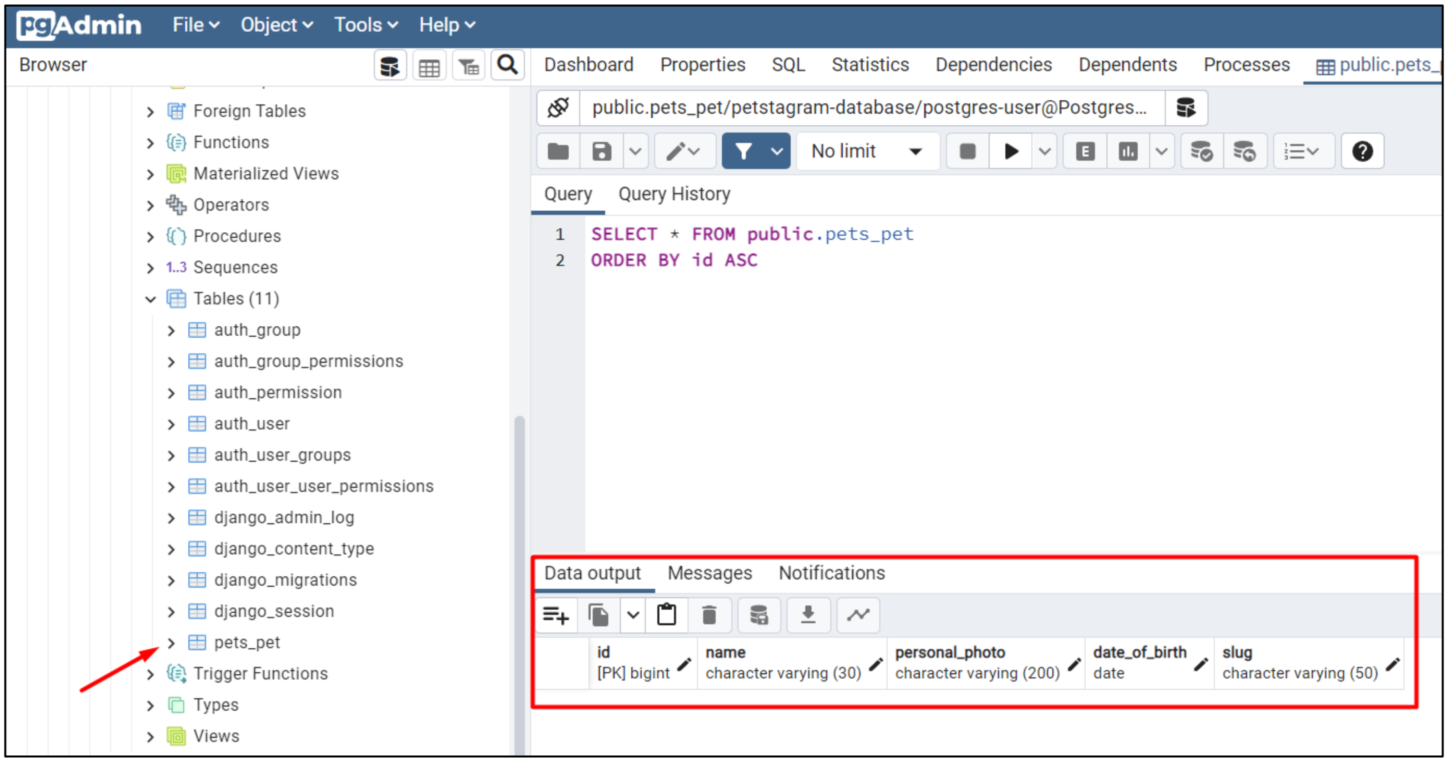


```
(venv) PS Z:\Python Web\Sept-2022\Python-Web-Basics\06-Workshop-Part-1\petstagram> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, pets, sessions
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying pets.0001_initial... OK
  Applying sessions.0001_initial... OK
(venv) PS Z:\Python Web\Sept-2022\Python-Web-Basics\06-Workshop-Part-1\petstagram>
```

We can see that **not only our model was migrated** but some additional models are prebuilt in Django. Let us **check if our database is updated**. Follow the path **`petstagram-database`** **→** **`Schemas`** **→** **`public`** **→** **`Tables`**:



When we open the created table **`pets_pet`** from our **`Pet`** model, **we can see all columns** we added when defining it. (Note: to see the table right-click on the **`pets_pet`** and choose "View/Edit Data" -> "All Rows"):

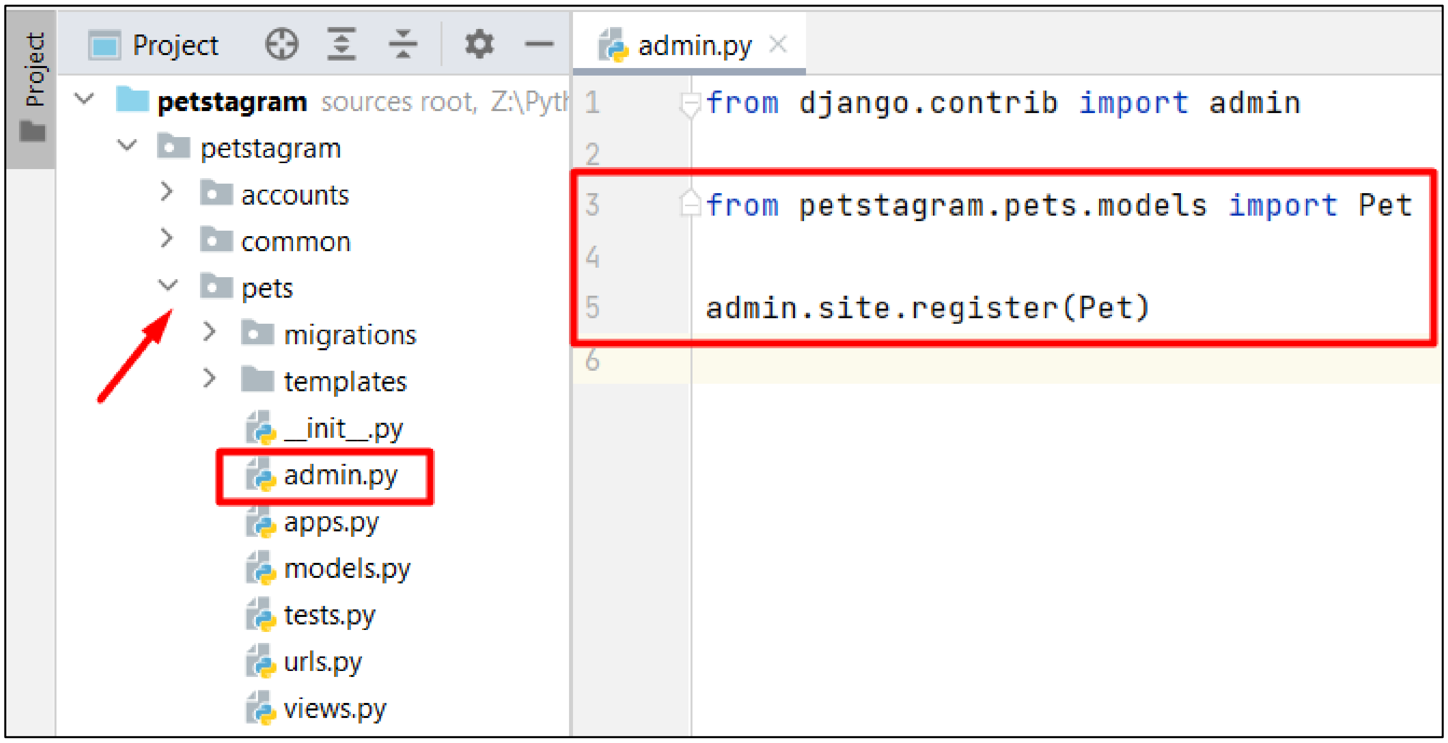


```
SELECT * FROM public.pets_pet
ORDER BY id ASC
```

id [PK] bigint	name character varying (30)	personal_photo character varying (200)	date_of_birth date	slug character varying (50)

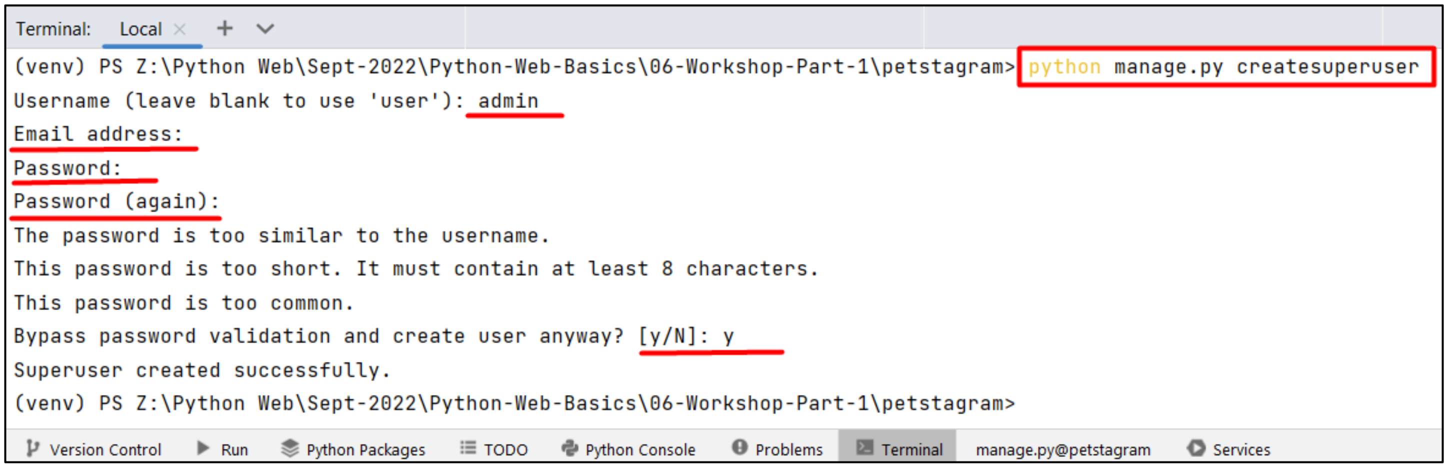
Work with the Django Admin Site

Let us now work more with the model in **the Django admin interface**. First, open the **`pets/admin.py file`** and register the model on the admin site:



```python
from django.contrib import admin

from petstagram.pets.models import Pet

admin.site.register(Pet)
```

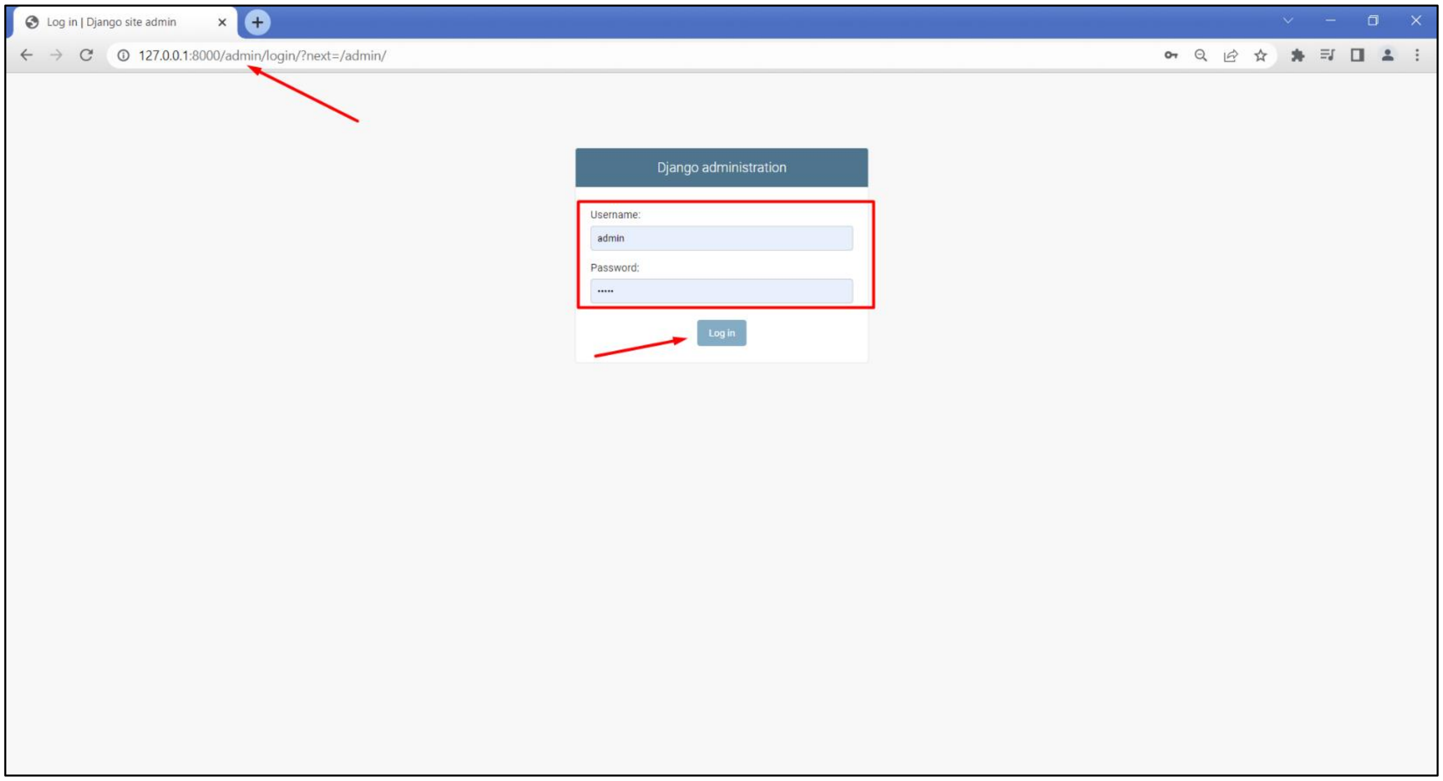
Next, to be able to **login to the admin site** (accessible only by admins) we must register as an administrator - it means that we should **create a "superuser" account**. Open the Terminal once again and write the command "**python** `manage.py createsuperuser`":



```
(venv) PS Z:\Python Web\Sept-2022\Python-Web-Basics\06-Workshop-Part-1\petstagram> python manage.py createsuperuser
Username (leave blank to use 'user'): admin
Email address:
Password:
Password (again):
The password is too similar to the username.
This password is too short. It must contain at least 8 characters.
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
(venv) PS Z:\Python Web\Sept-2022\Python-Web-Basics\06-Workshop-Part-1\petstagram>
```

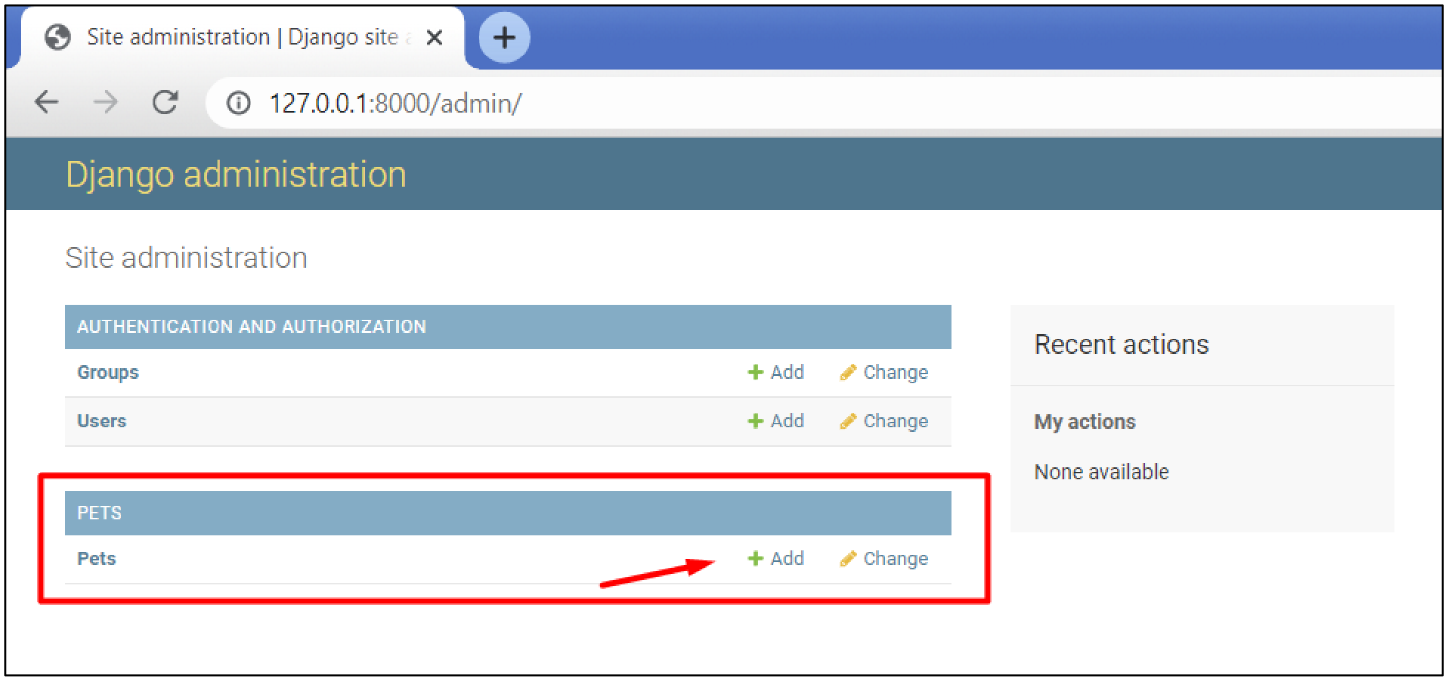
Right after we execute the command, Django asks us to **create a username** (in this case the username is "admin"), an **email address** (we can leave it blank just by clicking Enter), and a **password** (in this case the password is "admin"). (**Note**: In our case, Django asks us if we are sure we want to create an admin profile with a non-secure password. Let us type down "**y**" (for yes) as this is a personal project.)

Now, **start the development server**, go to the admin site at http://127.0.0.1:8000/admin/, and **log in with the credentials**:

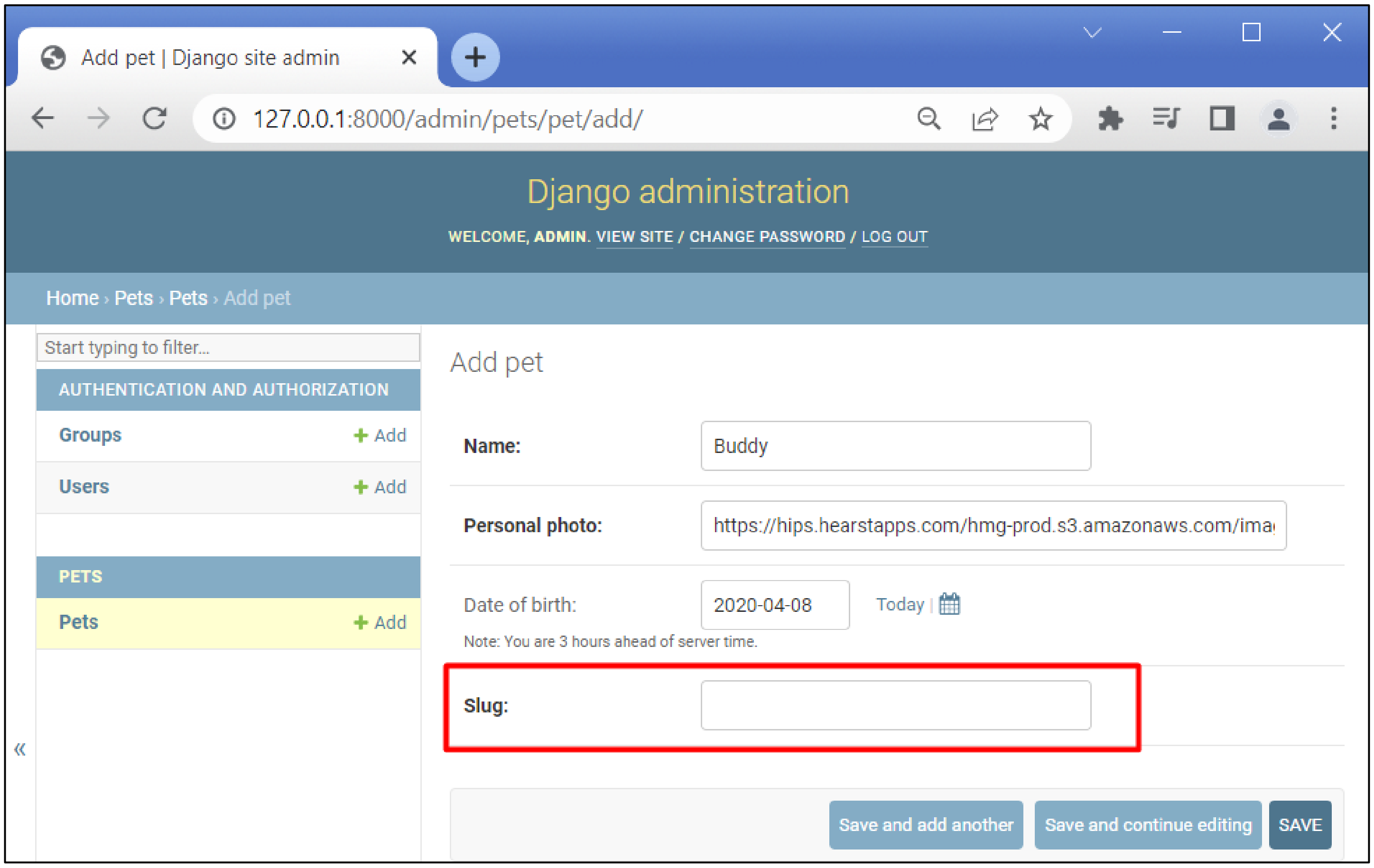


Log in | Django site admin
127.0.0.1:8000/admin/login/?next=/admin/
Django administration
Username:
admin
Password:
•••••
Log in

As we log in, we could see all registered models. The one we need is Pets. Let us **add a new pet**:



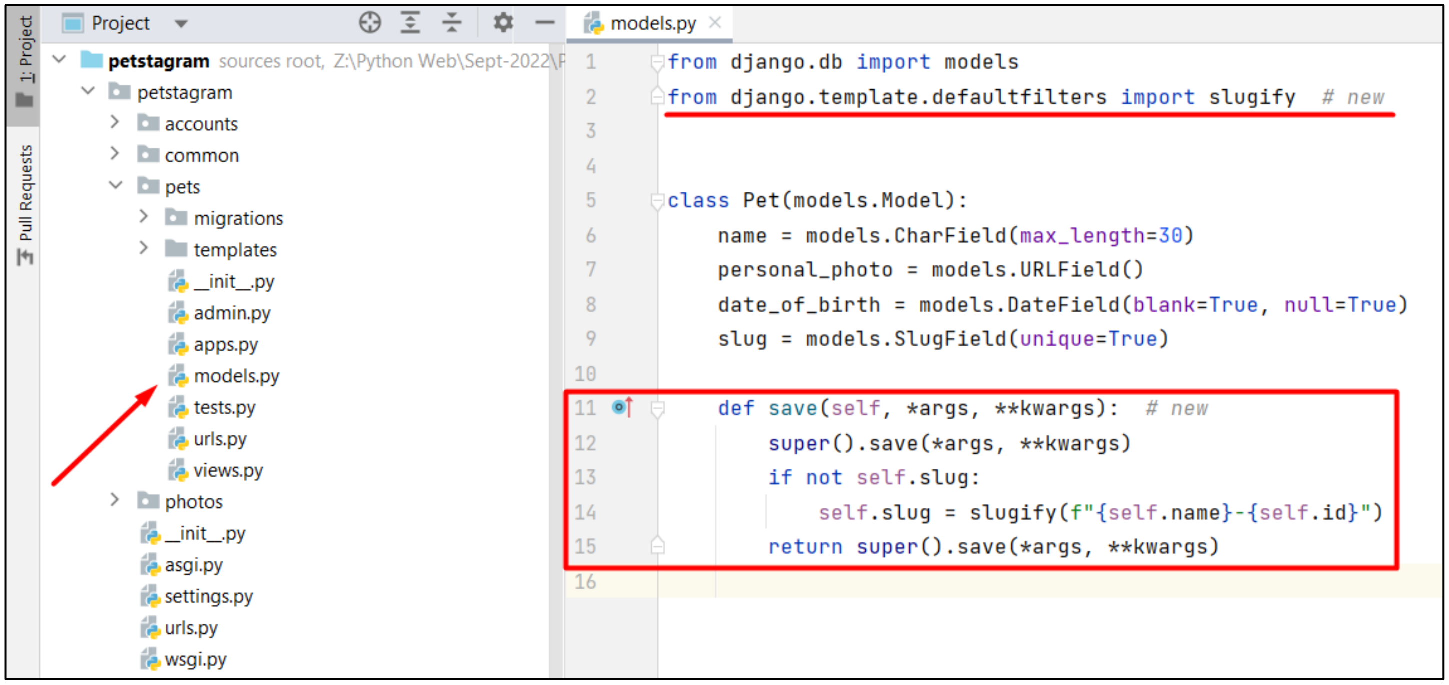
We can create a pet successfully. However, the **slug field is not auto-populated**:


Add pet | Django site admin
127.0.0.1:8000/admin/pets/pet/add/
Django administration
WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Pets › Pets › Add pet
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups
Add
Users
Add
PETS
Pets
Add
Add pet
Today
Save and add another
Save and continue editing
SAVE

Name: Buddy
Personal photo: https://hips.hearstapps.com/hmg-prod.s3.amazonaws.com/ima
Date of birth: 2020-04-08
Note: You are 3 hours ahead of server time.
Slug:

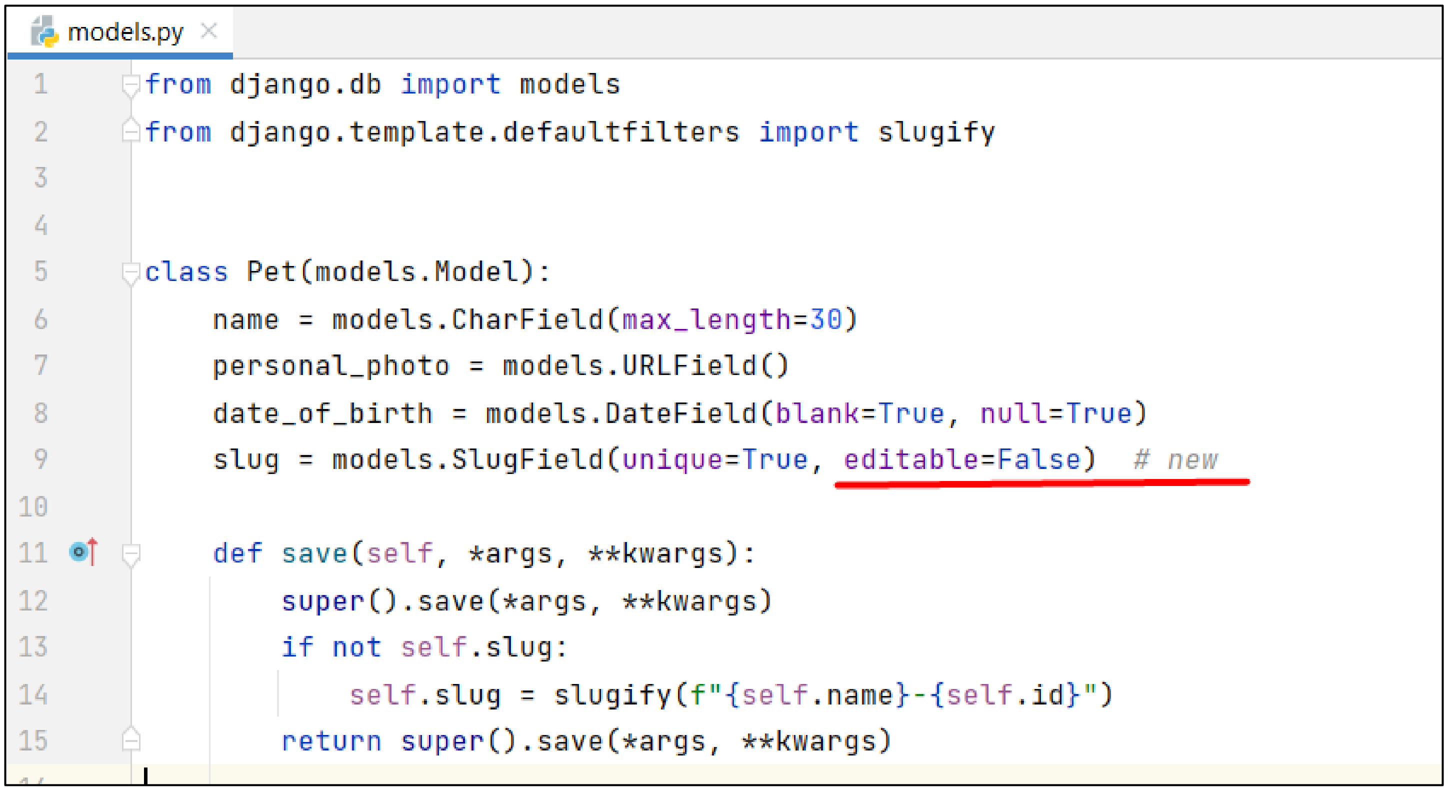
To do that, we will **override the Pet model `save()` method** using a special function called **`slugify()`** which helps us structure a slug from a given value. The if-statement stands to say that the **slug field will NOT be changed when the**

name of the pet is changed:



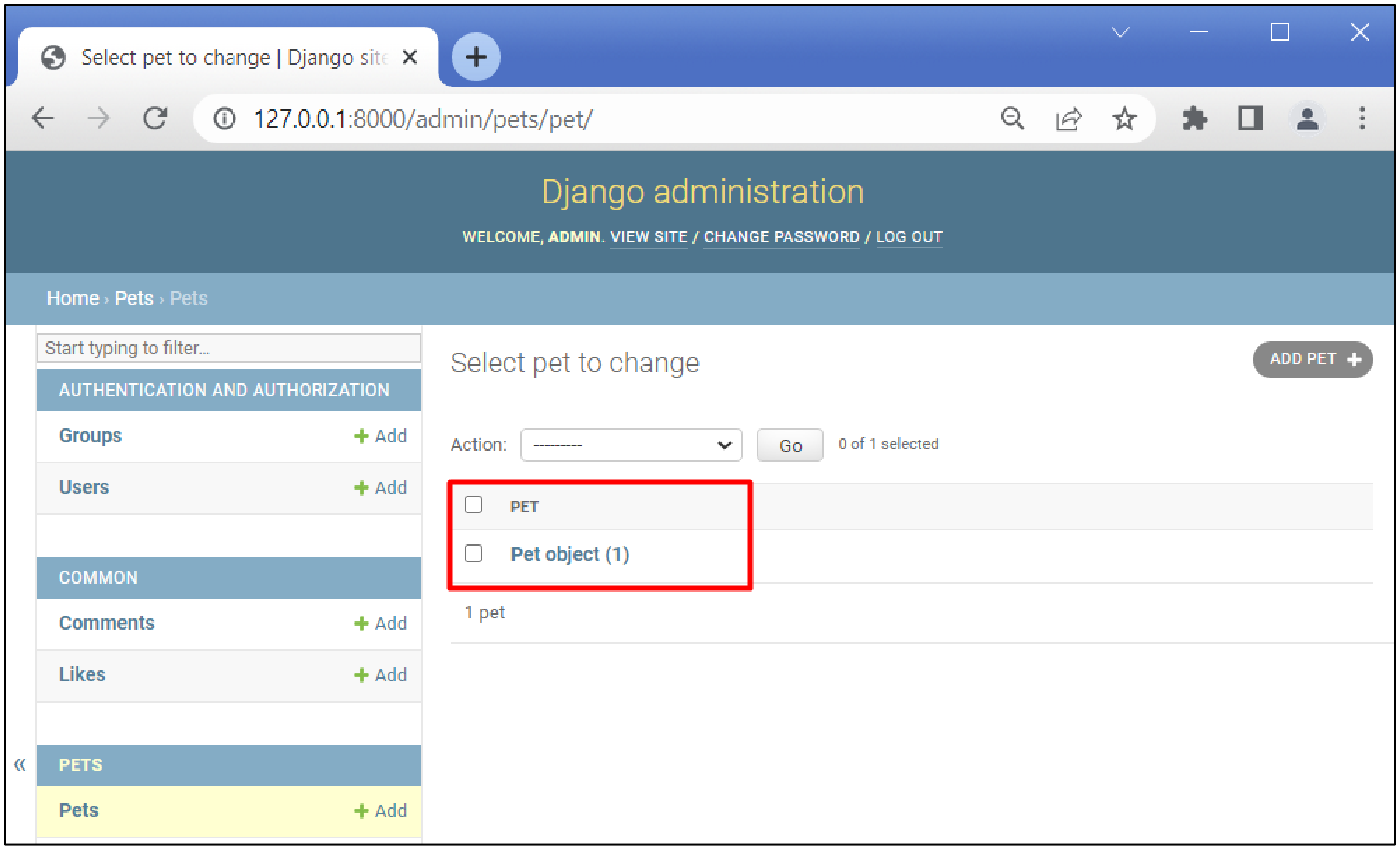
```python
from django.db import models
from django.template.defaultfilters import slugify  # new


class Pet(models.Model):
    name = models.CharField(max_length=30)
    personal_photo = models.URLField()
    date_of_birth = models.DateField(blank=True, null=True)
    slug = models.SlugField(unique=True)

    def save(self, *args, **kwargs):  # new
        super().save(*args, **kwargs)
        if not self.slug:
            self.slug = slugify(f"{self.name}-{self.id}")
        return super().save(*args, **kwargs)
```

We can go further and **change the slug field to be non-editable**. This way we ensure that the field will NOT be changed either in the form or the admin site:

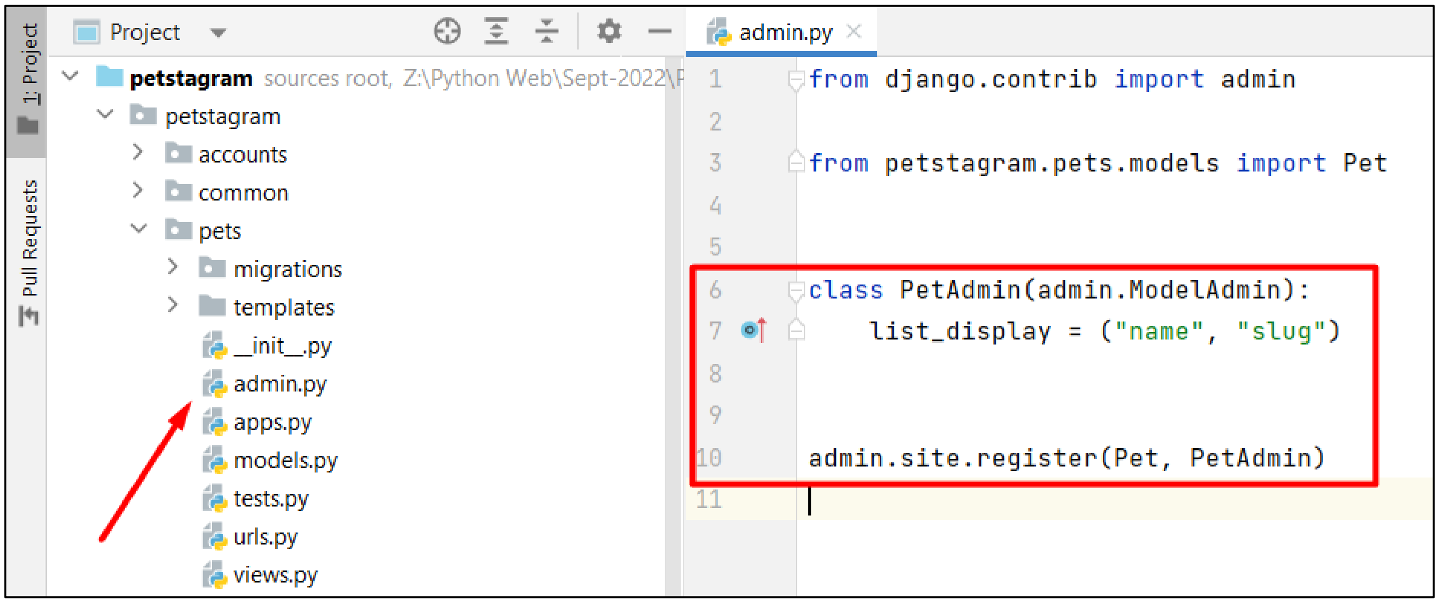


```python
from django.db import models
from django.template.defaultfilters import slugify


class Pet(models.Model):
    name = models.CharField(max_length=30)
    personal_photo = models.URLField()
    date_of_birth = models.DateField(blank=True, null=True)
    slug = models.SlugField(unique=True, editable=False)  # new

    def save(self, *args, **kwargs):
        super().save(*args, **kwargs)
        if not self.slug:
            self.slug = slugify(f"{self.name}-{self.id}")
        return super().save(*args, **kwargs)
```

Now, we can improve the admin site interface by **visualizing the pet models in a human-readable way**:

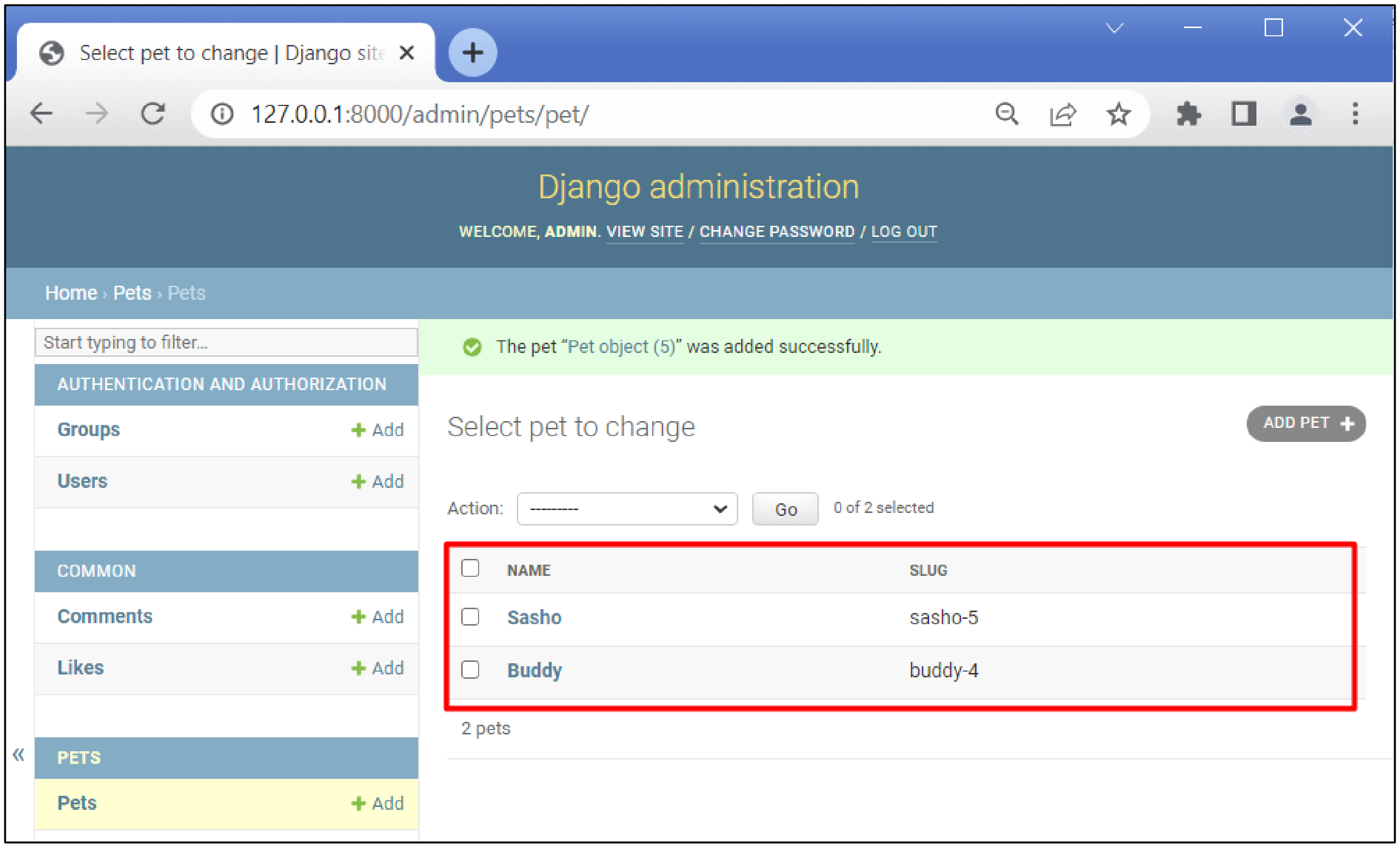

Select pet to change | Django site
127.0.0.1:8000/admin/pets/pet/
Django administration
WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Pets › Pets
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
COMMON
Comments + Add
Likes + Add
PETS
Pets + Add
Select pet to change
ADD PET +
Action: --------- Go 0 of 1 selected
PET
Pet object (1)
1 pet

Up until now, each pet looks like a "**Pet object**" with an **id**. A thing we could do to ease the work of the administrators of the app is to **show each pet by its name**. Let us open the **`pets/admin.py`** file again and add a **`list_display`** option:



```python
from django.contrib import admin

from petstagram.pets.models import Pet


class PetAdmin(admin.ModelAdmin):
    list_display = ("name", "slug")


admin.site.register(Pet, PetAdmin)
```

The admin interface changed like this:



Select pet to change | Django site

127.0.0.1:8000/admin/pets/pet/

Django administration

WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Pets › Pets

The pet "Pet object (5)" was added successfully.

Select pet to change

Action: --------- Go 0 of 2 selected

NAME	SLUG
Sasho	sasho-5
Buddy	buddy-4

2 pets

Creating the Photo Model

It is time to create the second model for the pet's photo.

The field **Photo** is **required**:

- **Photo** - the user can **upload** a picture **from storage**, the **maximum size** of the photo can be **5MB**

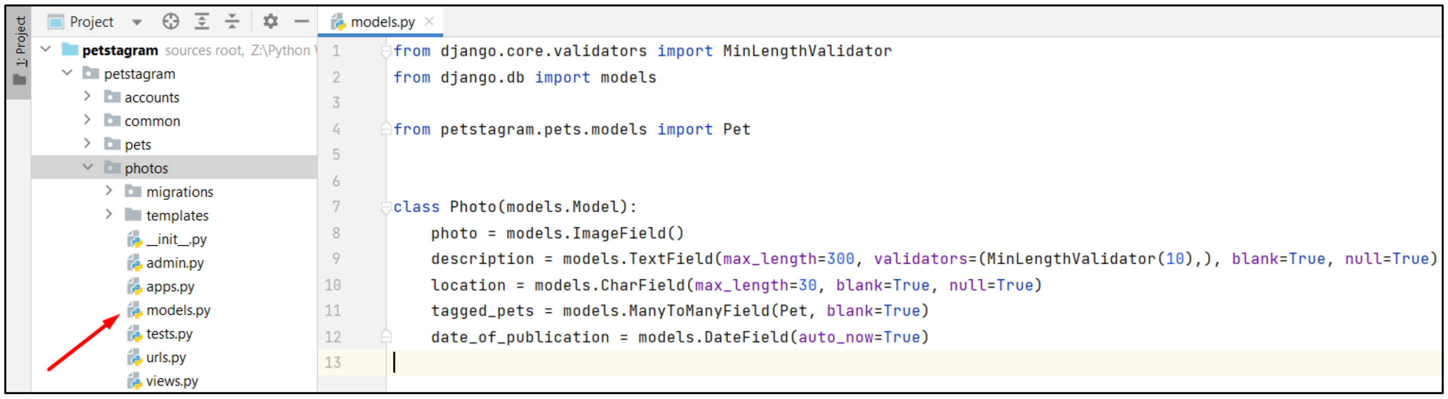
The fields **description and tagged pets** are **optional**:

- **Description** - a user can write any description of the photo; it should consist of a **maximum of 300 characters** and a **minimum of 10 characters**
- **Location** - it should consist of a **maximum of 30 characters**
- **Tagged Pets** - the user can tag **none, one, or many of all pets**. There is **no limit** on the number of tagged pets

There should be created **one more field** that will be **automatically generated**:

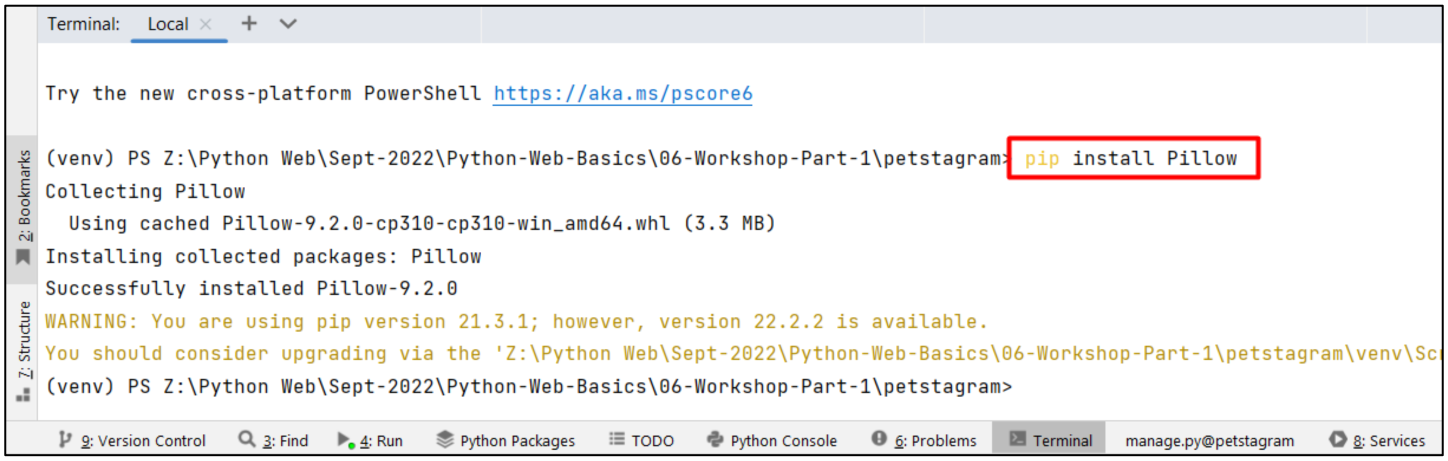
- **Date of publication** - when a picture is **added or edited**, the date of publication is **automatically generated**

Open the **`photos/models.py`** file and let us create the model:



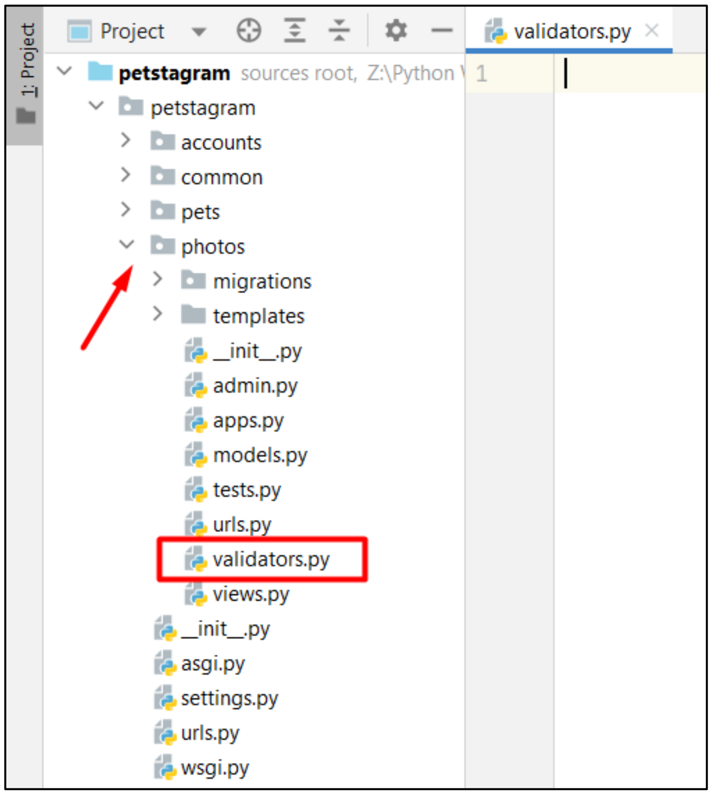
```python
from django.core.validators import MinLengthValidator
from django.db import models

from petstagram.pets.models import Pet


class Photo(models.Model):
    photo = models.ImageField()
    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)
```

To **work with an image field**, we should **install a library called Pillow**:

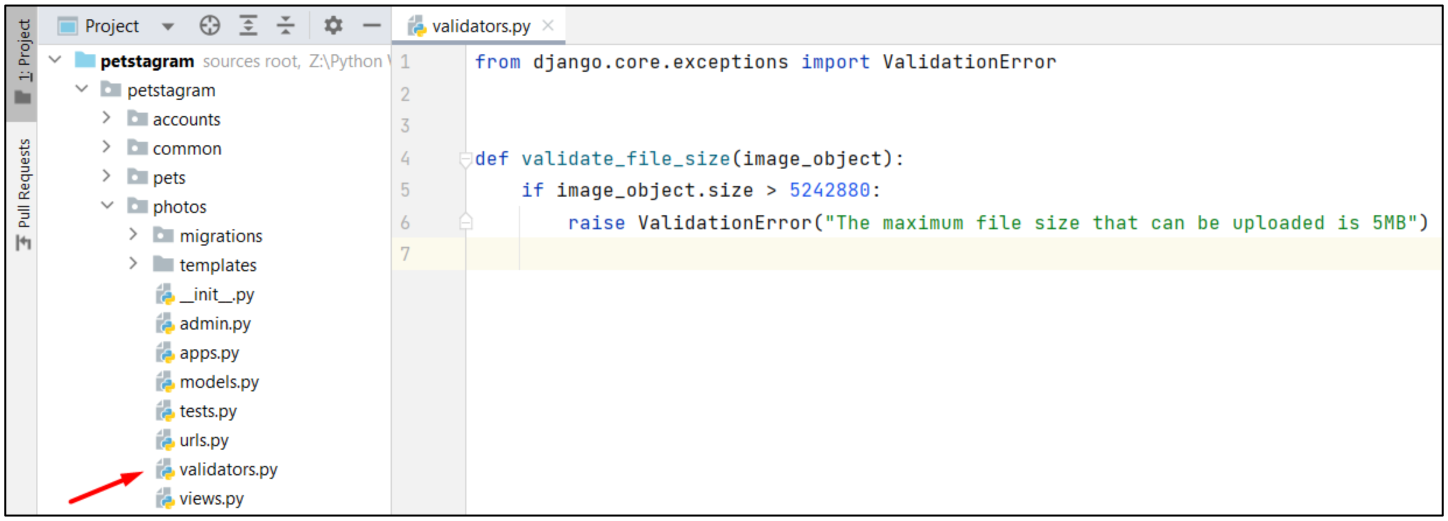


```
Try the new cross-platform PowerShell https://aka.ms/pscore6

(venv) PS Z:\Python Web\Sept-2022\Python-Web-Basics\06-Workshop-Part-1\petstagram> pip install Pillow
Collecting Pillow
  Using cached Pillow-9.2.0-cp310-cp310-win_amd64.whl (3.3 MB)
Installing collected packages: Pillow
Successfully installed Pillow-9.2.0
WARNING: You are using pip version 21.3.1; however, version 22.2.2 is available.
You should consider upgrading via the 'Z:\Python Web\Sept-2022\Python-Web-Basics\06-Workshop-Part-1\petstagram\venv\Sc
(venv) PS Z:\Python Web\Sept-2022\Python-Web-Basics\06-Workshop-Part-1\petstagram>
```

Note that the **photo field** has **additional validation for a maximum size of 5MB**. We should **create a custom validator** to validate the requirement. Let us create a new **`validators.py`** file in the **`photos`** app:

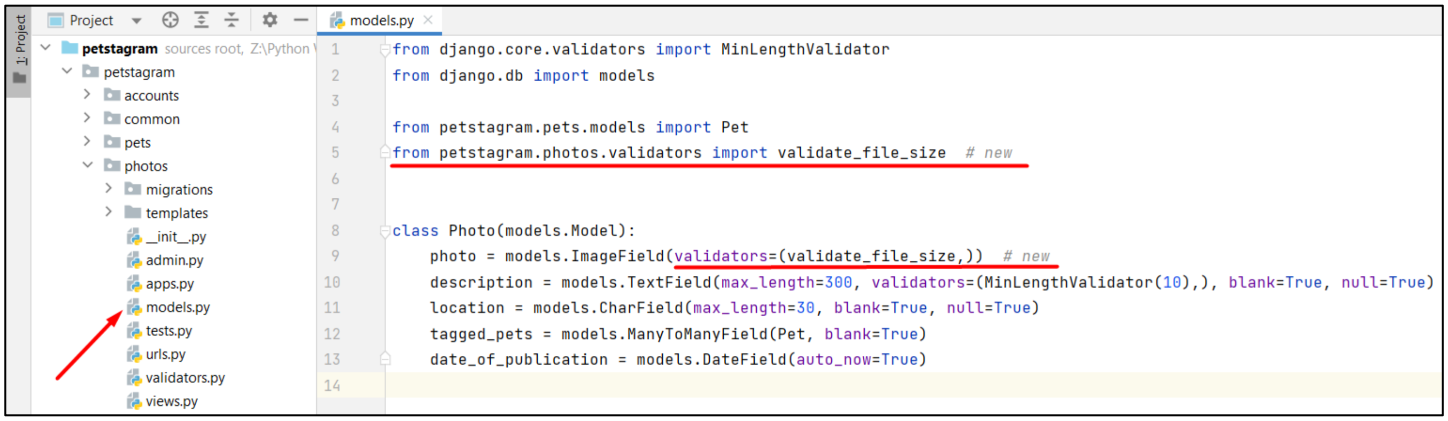


Open the file and **write the validation function** that will check if the photo size is above 5MB. In this case, it will raise a **`ValidationError`**:



```python
from django.core.exceptions import ValidationError


def validate_file_size(image_object):
    if image_object.size > 5242880:
        raise ValidationError("The maximum file size that can be uploaded is 5MB")
```

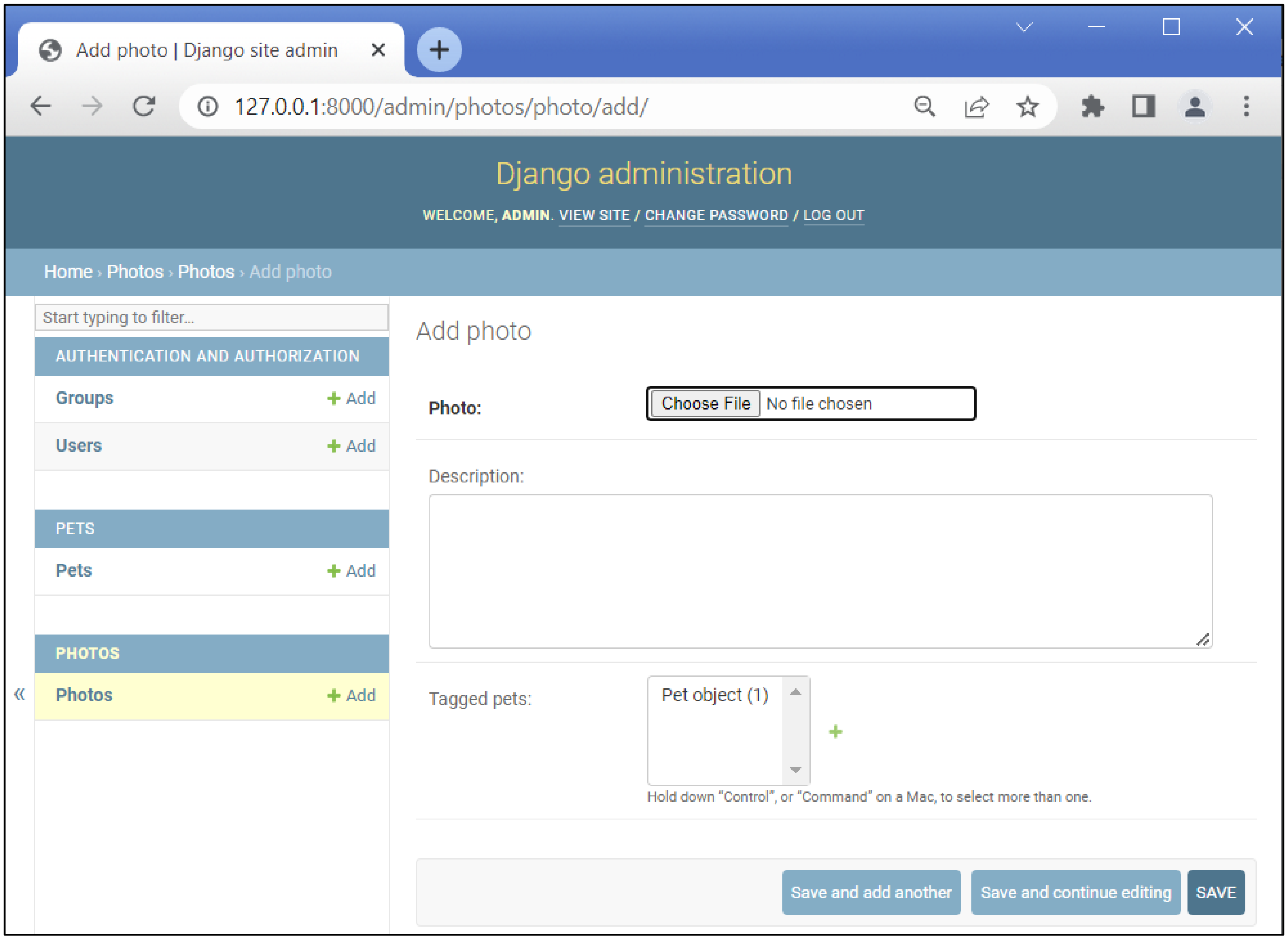
Then, we will **add the validator** to our **`validators`** list in the **photo** field:



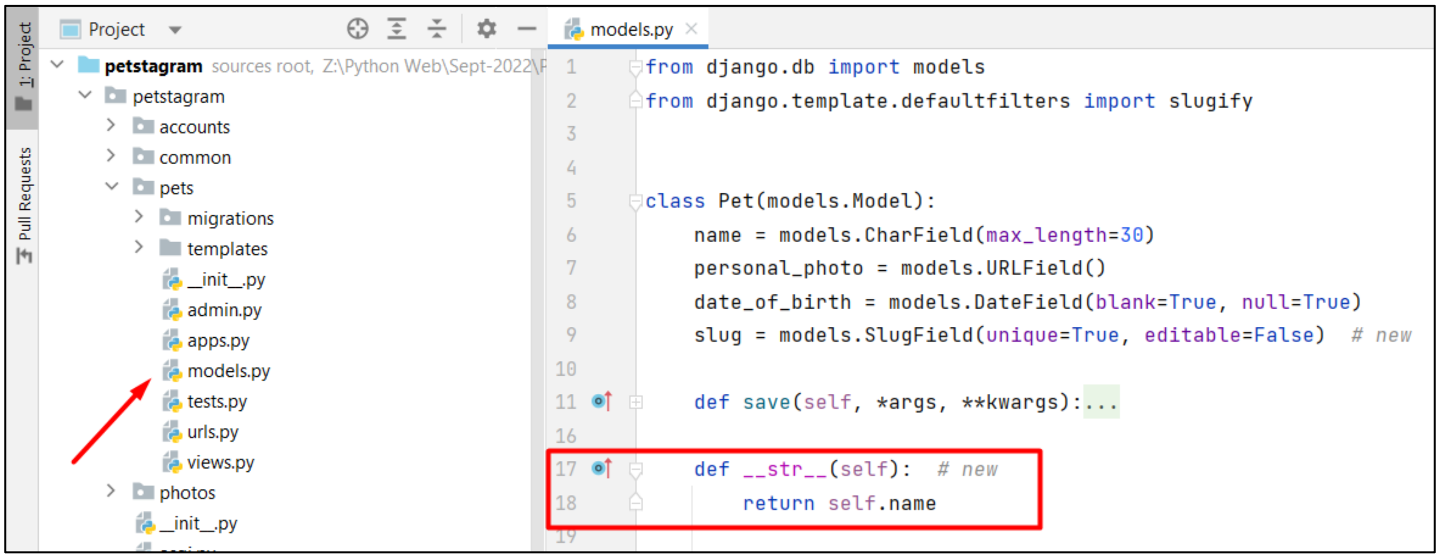
```python
from django.core.validators import MinLengthValidator
from django.db import models

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


class Photo(models.Model):
    photo = models.ImageField(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 **migration files** and **migrate the changes** to the database. **Register the model in the admin**.

Then, **start the development server** and check if everything **works correctly** in the admin panel:

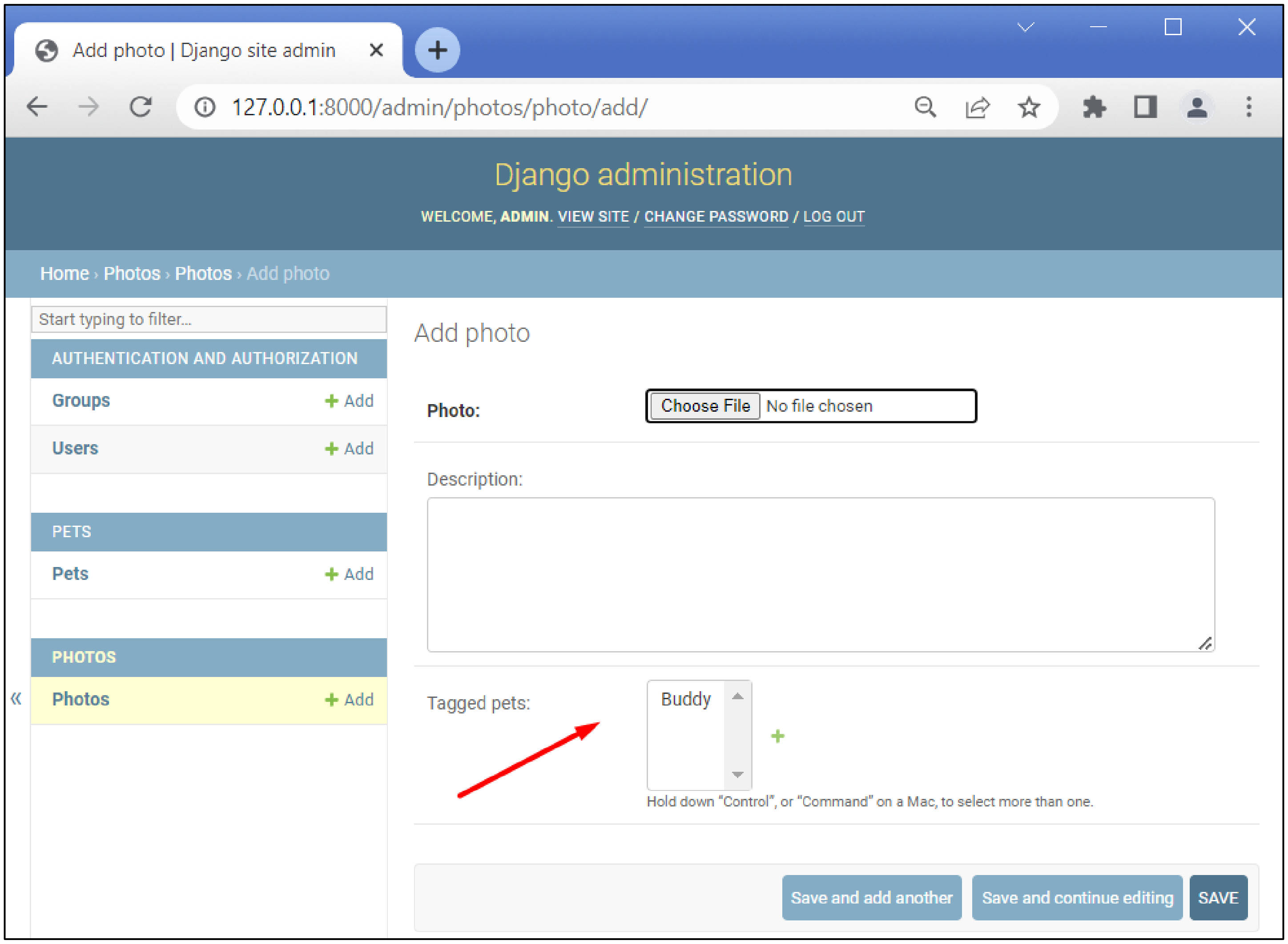

Add photo | Django site admin
127.0.0.1:8000/admin/photos/photo/add/
Django administration
WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Photos › Photos › Add photo
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
PETS
Pets + Add
PHOTOS
Photos + Add
Add photo
Photo: Choose File No file chosen
Description:
Tagged pets: Pet object (1)
Hold down "Control", or "Command" on a Mac, to select more than one.
Save and add another
Save and continue editing
SAVE

We can see that the pets in the tagged pets section **visualize as pet objects with an id**. We can change this by **overriding the `__str__` method in the `Pet` model**:

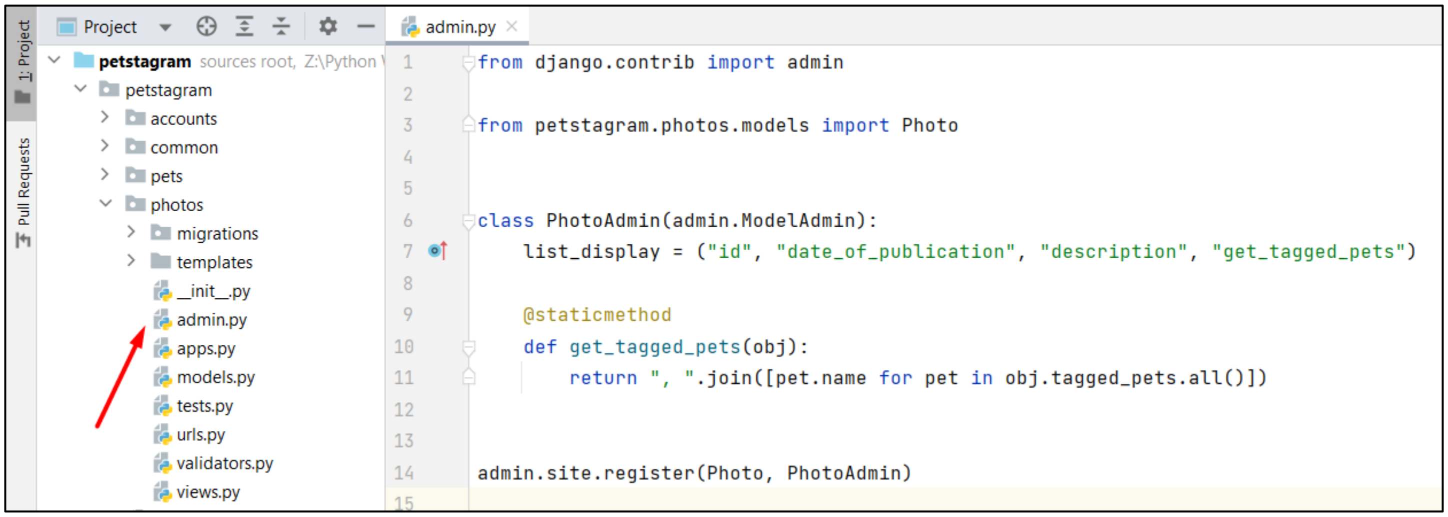


```python
from django.db import models
from django.template.defaultfilters import slugify


class Pet(models.Model):
    name = models.CharField(max_length=30)
    personal_photo = models.URLField()
    date_of_birth = models.DateField(blank=True, null=True)
    slug = models.SlugField(unique=True, editable=False)  # new

    def save(self, *args, **kwargs):...

    def __str__(self):  # new
        return self.name
```

When we **reload the add photo page in the admin site** we will see the difference:

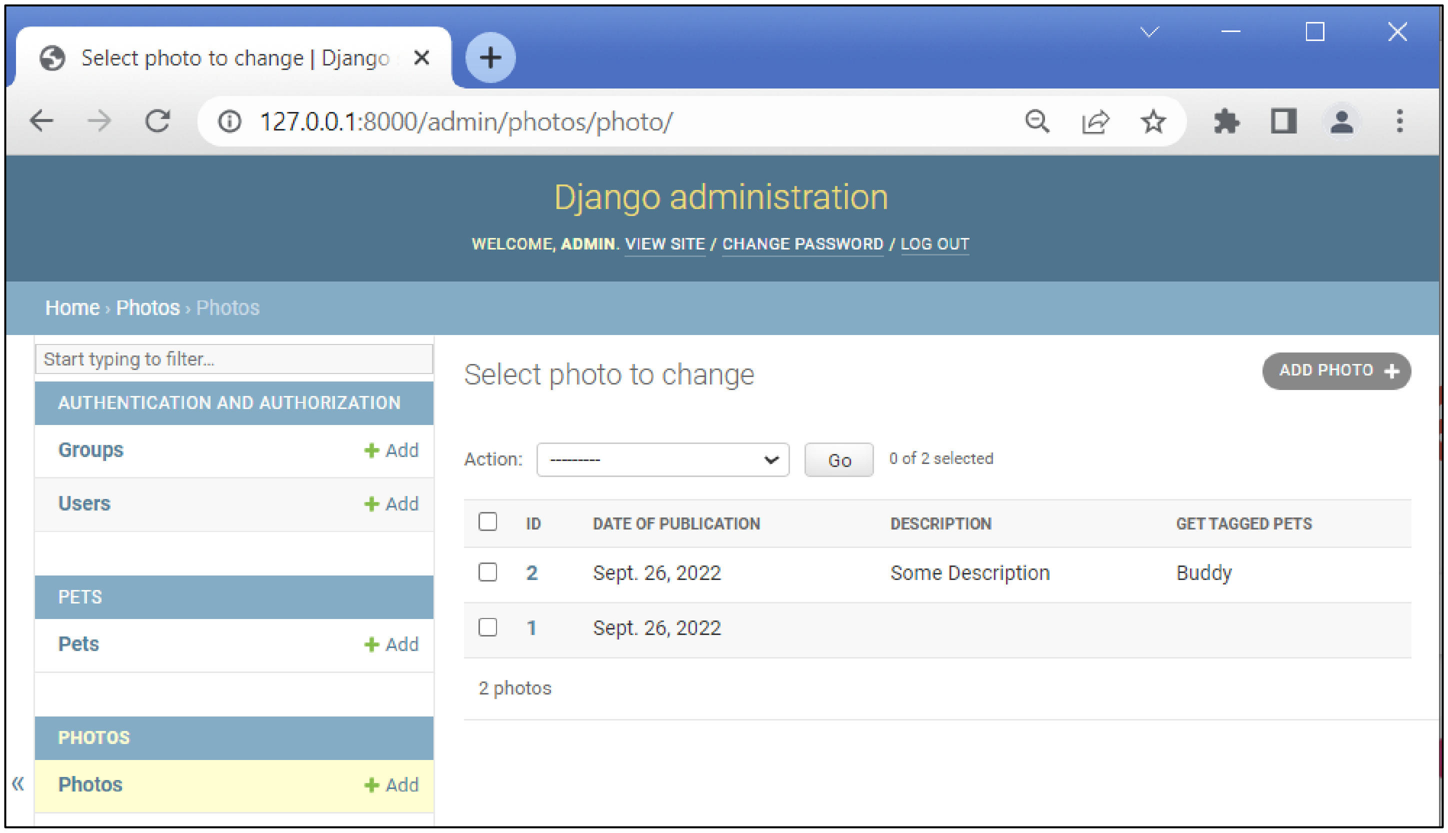

Add photo | Django site admin
127.0.0.1:8000/admin/photos/photo/add/
Django administration
WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Photos › Photos › Add photo
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
PETS
Pets + Add
PHOTOS
Photos + Add
Add photo
Photo: Choose File No file chosen
Description:
Tagged pets: Buddy
Hold down "Control", or "Command" on a Mac, to select more than one.
Save and add another
Save and continue editing
SAVE

Now, it is time to **customize the admin site interface** on the photos model page. Let us add a list of fields to be displayed on the photo model's page. The fields are the **id of the photo**, **date of publication**, **description**, and **names of all tagged pets**. We **cannot list a Many-to-Many field**, but we can **list the result of a function** that gets all objects from a Many-to-Many field and concatenate their names in a string:



```
from django.contrib import admin

from petstagram.photos.models import Photo


class PhotoAdmin(admin.ModelAdmin):
    list_display = ("id", "date_of_publication", "description", "get_tagged_pets")

    @staticmethod
    def get_tagged_pets(obj):
        return ", ".join([pet.name for pet in obj.tagged_pets.all()])


admin.site.register(Photo, PhotoAdmin)
```

Now the **interface looks like that**:


Select photo to change | Django
127.0.0.1:8000/admin/photos/photo/
Django administration
WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Photos › Photos
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
PETS
Pets + Add
PHOTOS
Photos + Add
ADD PHOTO +
Action: --------- Go 0 of 2 selected

Select photo to change

ID	DATE OF PUBLICATION	DESCRIPTION	GET TAGGED PETS
2	Sept. 26, 2022	Some Description	Buddy
1	Sept. 26, 2022		

2 photos

Creating the Comment Model

It is time to create the comment model.

The field **Comment Text** is **required**:

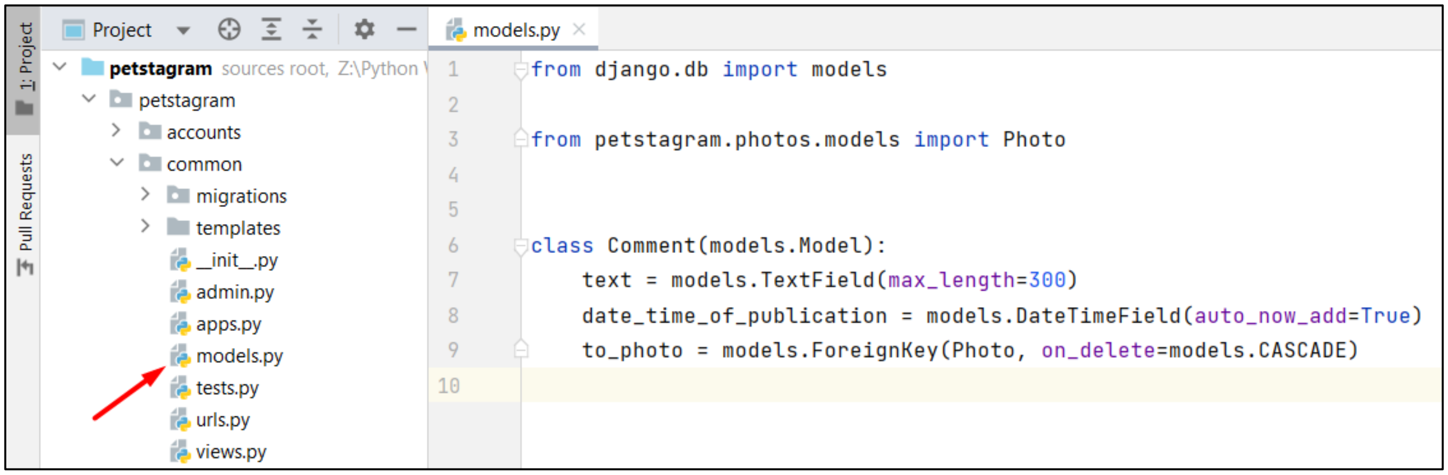
- **Comment Text** - it should consist of a **maximum of 300 characters**

An additional field should be created:

- **Date and Time of Publication** - when a comment is created (only), the date of publication is **automatically generated**

One more thing we should keep in mind is that **the comment should relate to the photo** (as in social apps users comment on a specific photo/post, i.e., the comment object is always connected to the photo object).

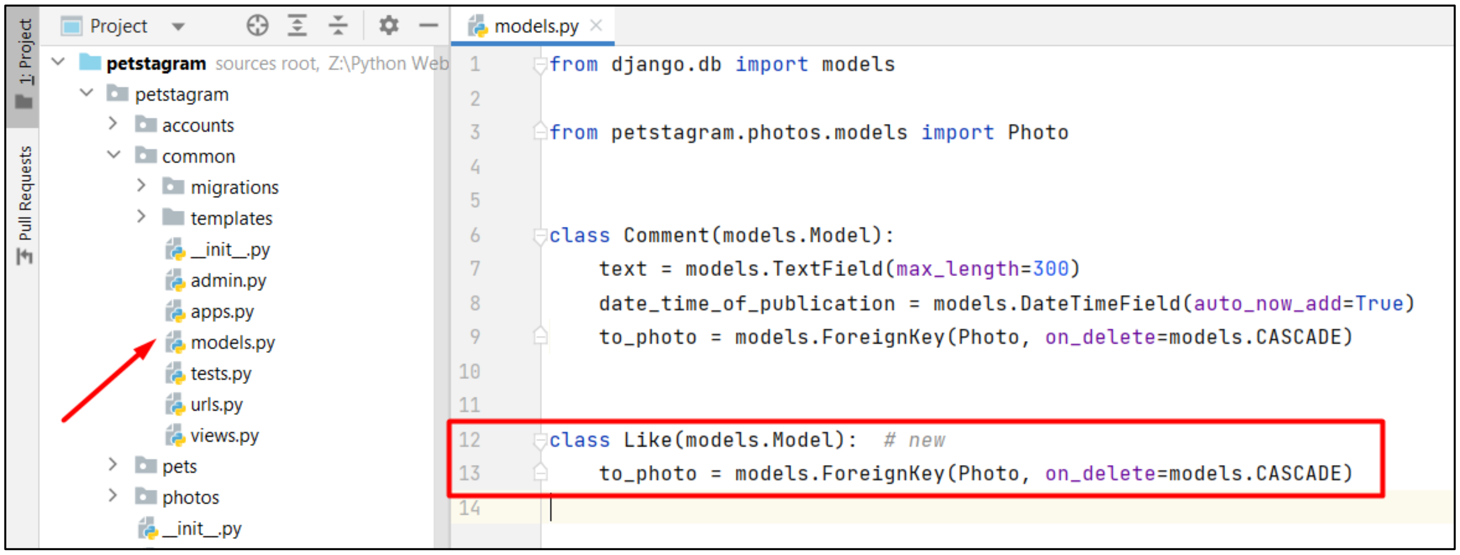
Open the **`common/models.py` file** and let us **create the model**:



```python
from django.db import models

from petstagram.photos.models import Photo


class Comment(models.Model):
    text = models.TextField(max_length=300)
    date_time_of_publication = models.DateTimeField(auto_now_add=True)
    to_photo = models.ForeignKey(Photo, on_delete=models.CASCADE)
```

Creating the Like Model

Finally, create the **Like model** which should connect one photo to one user. However, **we do not have a user object**, so we will **just create the model and add the photo relation**:



```python
from django.db import models

from petstagram.photos.models import Photo


class Comment(models.Model):
    text = models.TextField(max_length=300)
    date_time_of_publication = models.DateTimeField(auto_now_add=True)
    to_photo = models.ForeignKey(Photo, on_delete=models.CASCADE)


class Like(models.Model):  # new
    to_photo = models.ForeignKey(Photo, on_delete=models.CASCADE)
```

Make the migration files and **migrate the changes to the database**. We can now **register the models in the Django admin** site and **check if they work correctly**.

2. Workshop - Part 2.2

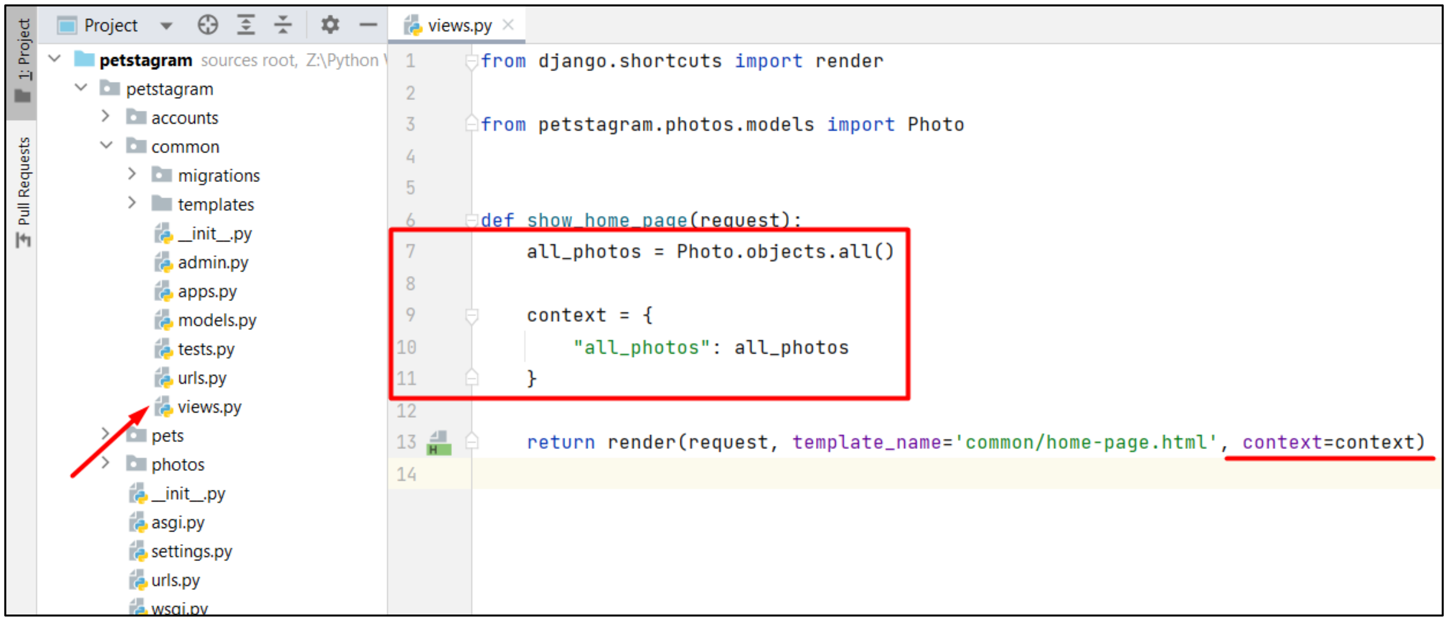
Add models to Home Page

We are ready to add some functionality to our **Home page**.

The Home page consists of **pet posts.** First, we will configure:

- The **location** (if one is added)
- The **tagged pets** (if any are added) - if there is **more than one pet** tagged, they must be shown on **different lines**
- The **link to the photo details** page
- **Date** of publication or edition of the photo

Let us open the **`common/views.py`** file. We will **read all photo objects from the database** and **add them to a context** dictionary:



```
from django.shortcuts import render

from petstagram.photos.models import Photo


def show_home_page(request):
    all_photos = Photo.objects.all()

    context = {
        "all_photos": all_photos
    }

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

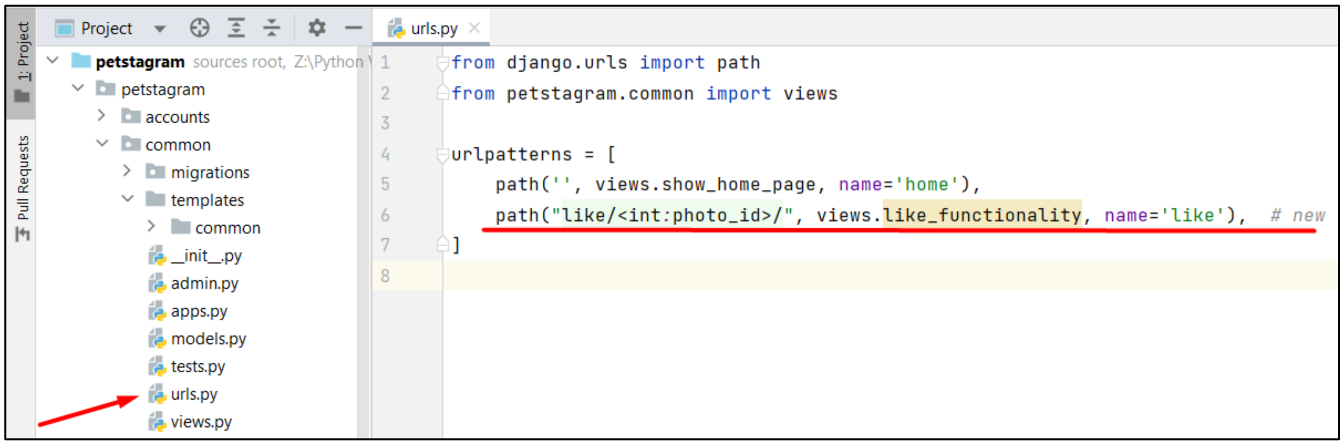
Now, we can **inject the information into the `pets-posts.html` template**. (Note: we will **use the string "username" in the pet details URL** to bypass the user implementation):

```
{% load static %}
{% for photo in all_photos %}
...

        <!-- Start User Details and Image Location -->
...
                    <!-- if the photo has location -->
                    {% if photo.location %}
                        <span>{{ photo.location }}</span>
                    {% endif %}
...
        <!-- End User Details and Image Location -->
...
            <!-- Start Tagged Pets -->
            {% for pet in photo.tagged_pets.all %}
                <!-- Link to First Tagged Pet Details Page-->
                <a href="{% url 'pet-details' "username" pet.slug %}">
                    <p class="message">
                        <b>{{ pet.name }}</b>
                    </p>
                </a>
            {% endfor %}
            <!-- End Tagged Pets -->

            <!-- Link to Photo Details Page -->
            <a href="{% url 'photo-details' photo.pk %}">
                <h4 class="details">See details</h4>
            </a>

            <!-- Date of Publication -->
            <h5 class="postTime">{{ photo.date_of_publication }}</h5>
...
{% endfor %}
```

Implement Like Button Functionality

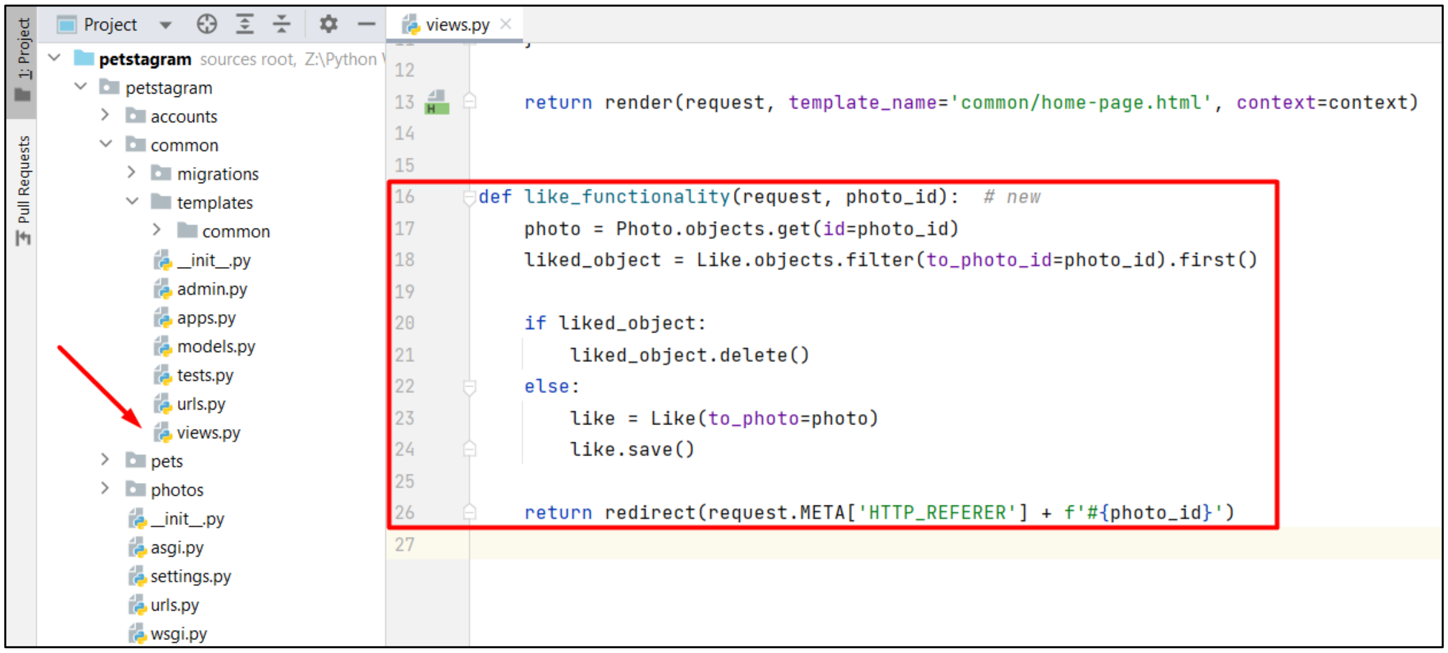
Next, we will **implement the like button** and the **number of likes per photo**.

Let us start by creating a like button functionality - to work with the like button we should **create a view with the specific functionality**. First, create a **like button path** in the **`common/urls.py` `urlspatterns`** list:



```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'),  # new
]
```

Now, **create a `like_functionality` view** in the **`common/views.py`**. The view will **receive the id of the current photo** and will **get the photo by the given id**. Then, the **view tries to filter the Like objects by the photo id** - if it finds an object, it means that the photo is liked. Based on that**, if the object is liked the view will delete the like** (and the object will be unliked). Otherwise, **the view will create a new Like object related to the photo** and **will save it to the database** (and the object will be liked). In the end, we will **write a redirect function** that will redirect to the **last visited page** (**`request.META['HTTP_REFERER']`**) and will **stop exactly at the photo we liked/unliked** (**`f'#{photo_id}'`**):



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


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

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

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

Let us **refactor the template**. We will **implement the path** where the user should reach **when the heart button is clicked**. Then, the template will **check if the photo is connected to some of the Like objects**. Django uses "**`like_set`**" to **reverse the search** - the Photo model is related to the Like model via One-to-Many relation; so we can get all like objects that are connected to the Photo model using the syntax "**`like_set.all`**". In the same way, we can **count all**

likes for the photo, this time using the method **count** in the template:

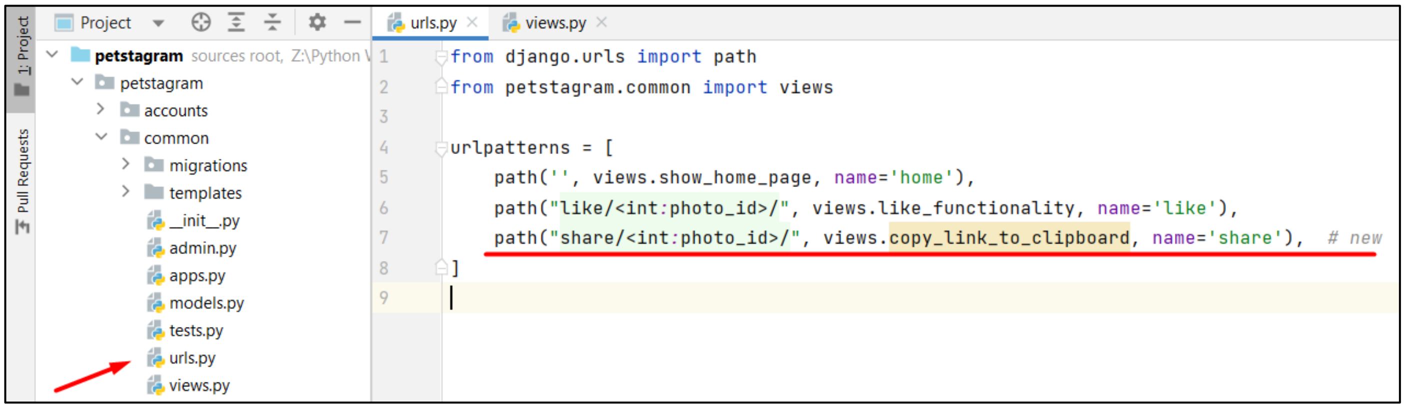
```
...
<!-- Start Like and Share Buttons -->
<div class="bottom">
    <div class="actionBtns">
        <div class="left">
            <!-- Start Like Button -->
            <span class="heart">
                 <a href="{% url 'like' photo.id %}">

                     <!-- if user has liked the photo -->
                      {% if photo.like_set.all %}
                          <svg style="color: red"
                              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 -->
                          ...
                      <!-- else -->
                       {% else %}
                           <svg aria-label="Like"
                               color="#262626"
                               fill="#262626"
                               height="24"
                               role="img"
                               viewBox="0 0 48 48"
                               width="24">
                       {% endif %}
                           <!-- Coordinate path -->
                           ...
              <!-- End Like Button -->
               ...
<!-- End Like and Share Buttons -->

<!-- Number of Likes per Photo -->
<p class="likes">{{ photo.like_set.count }} likes</p>

...
```

One more thing we should do is to **add the photo id** to the template **in the photo div**. It is needed, so the **redirection works properly**:

```
...
<!-- Start Pet Photo -->
<div class="imgBx" id="{{ photo.id }}">
    <img src="{% static 'images/axolotl.jpeg' %}" alt="post" class="cover">
</div>
<!-- End Pet Photo -->
...
```

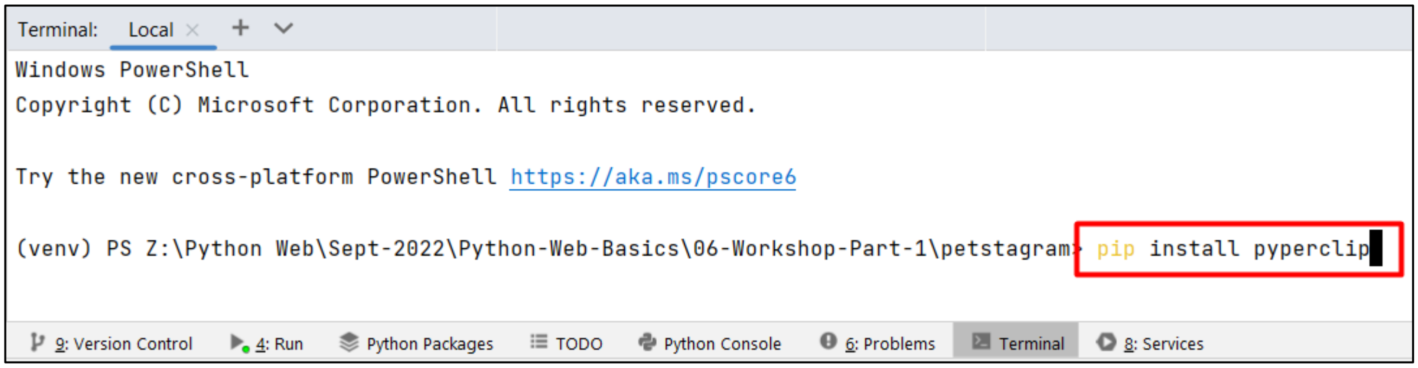
Implement Share Button Functionality

The **share button copies the photo details page URL in the clipboard**. To make the functionality, first, **add a path to a share view**:



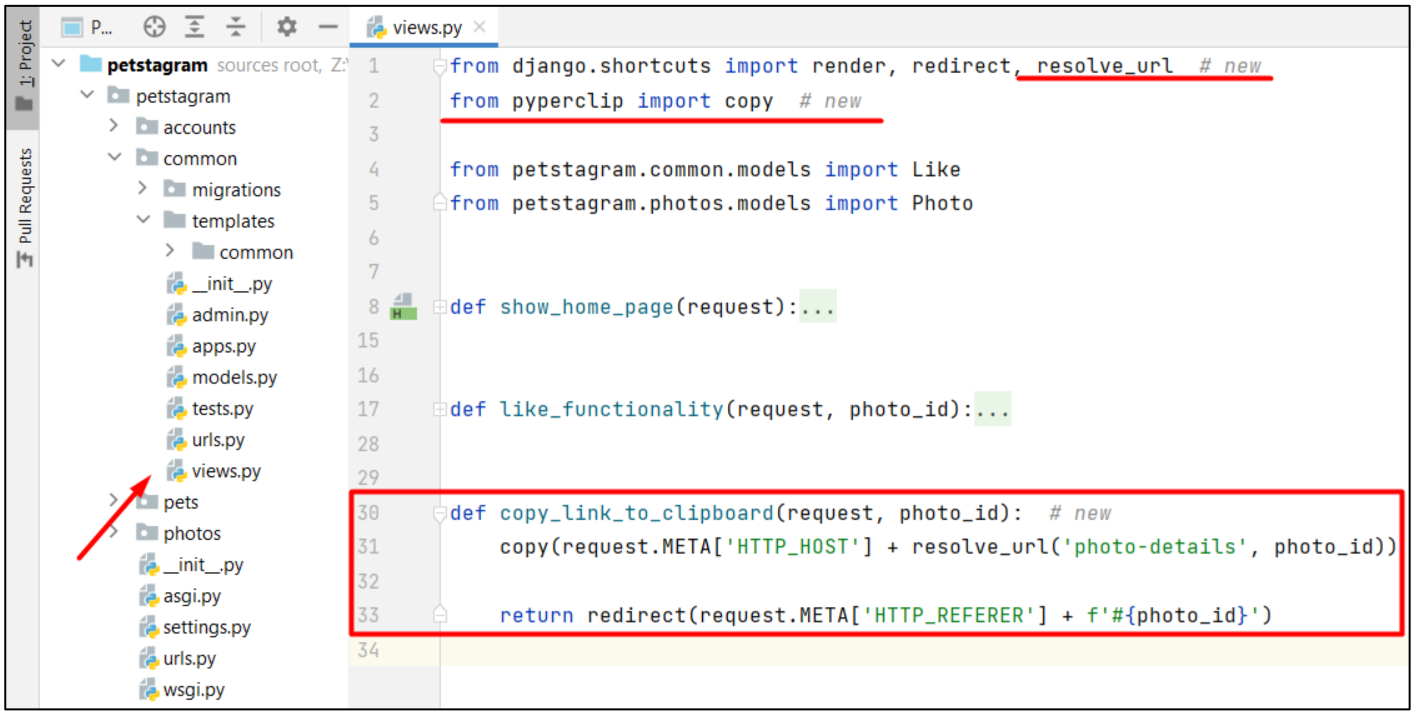
```python
from django.urls import path
from petstagram.common import views

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'),  # new
]
```

There is an **additional module called `pyperclip` that we need to install**:



```
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> pip install pyperclip
```

Import the `copy()` function from this module in the **`common/views.py` file.** Then, we will **create a link to be copied** - the first half contains the **domain** (**`request.META['HTTP_HOST']`**) and the second half - the **path to the photo details page** (**`resolve_url('photo-details', photo_pk)`**). Finally, as in the **`like_functionality`** view, we will **redirect the user to the last page visited on the exact photo they clicked**:

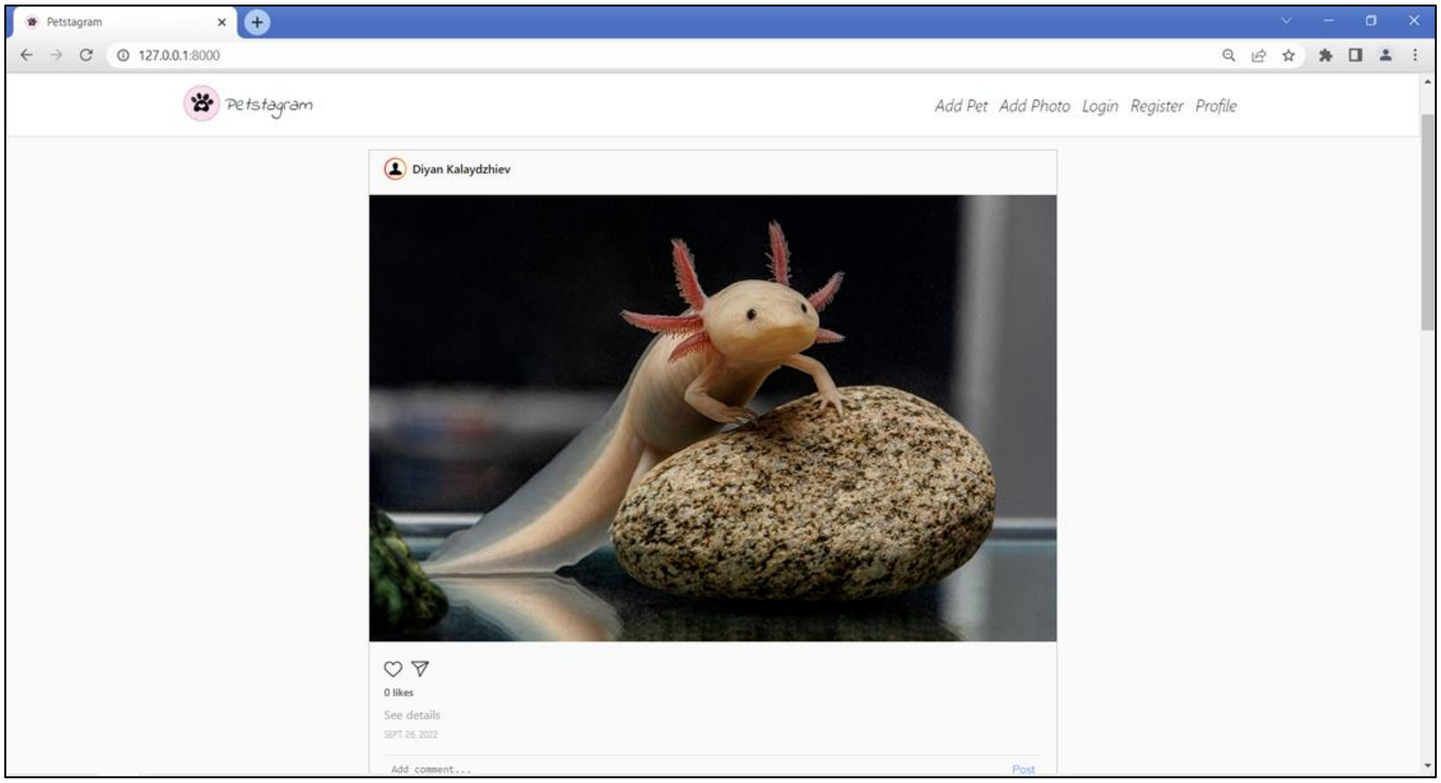


```python
from django.shortcuts import render, redirect, resolve_url  # new
from pyperclip import copy  # new

from petstagram.common.models import Like
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):  # new
    copy(request.META['HTTP_HOST'] + resolve_url('photo-details', photo_id))

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

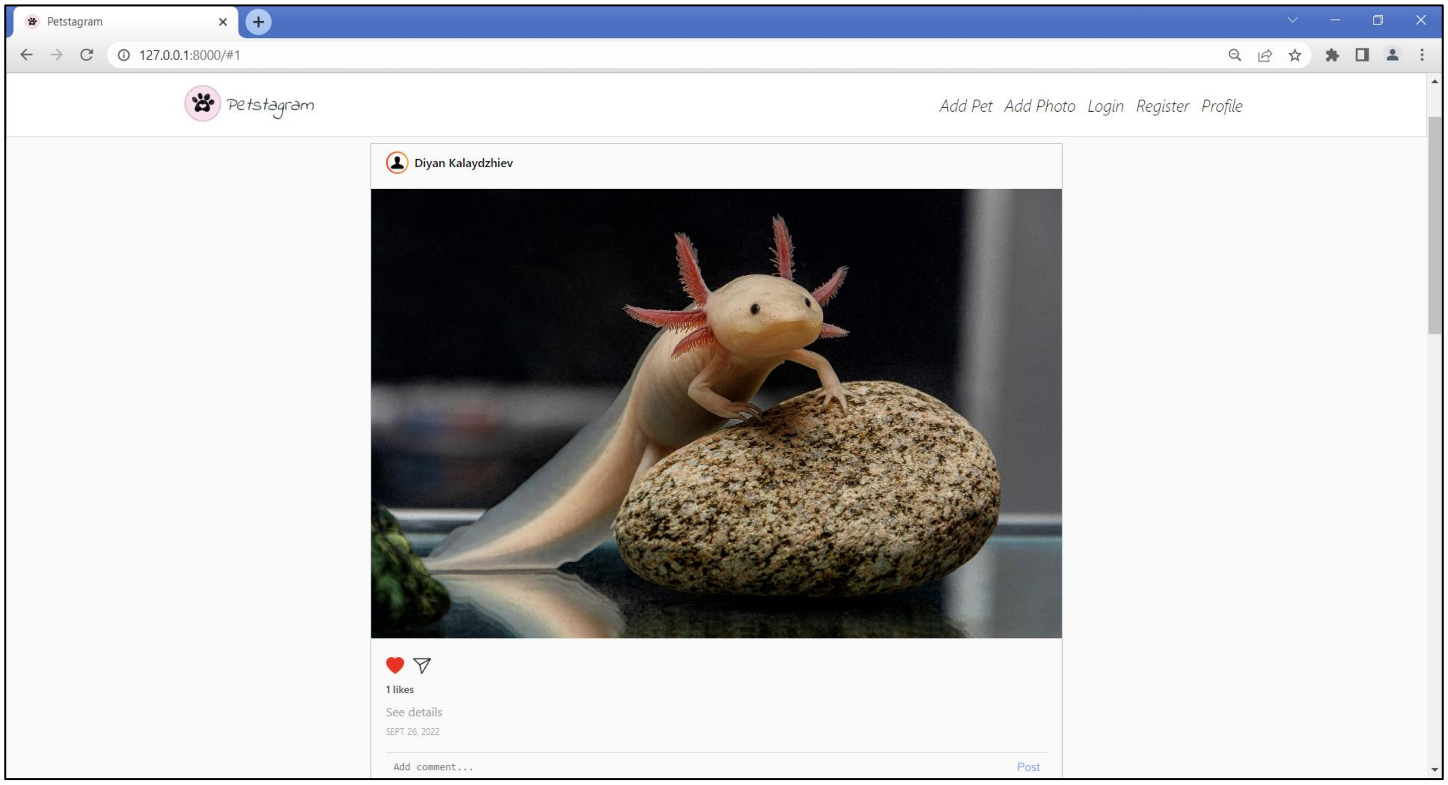
Now, let us **refactor the `pets-post.html`** template. The only needed thing to do here is to **add the URL path**:

```
...
<!-- Start Share Button -->
<a href="{% url 'share' photo.id %}">
    <svg aria-label="Share Post" class="_8-yf5 " color="#262626" fill="#262626"
        height="24" role="img" viewBox="0 0 48 48" width="24">
        <path d="M47.8 3.8c-.3-.5-.8-.8-1.3-.8h-45C.9 3.1.3
                  3.5.1 4S0 5.2.4 5.7l15.9 15.6 5.5 22.6c.1.6.6
                  1 1.2 1.1h.2c.5 0 1-.3
                  1.3-.7l23.2-39c.4-.4.4-1 .1-1.5zM5.2
                  6.1h35.5L18 18.7 5.2 6.1zm18.7
                  33.6l-4.4-18.4L42.4 8.6 23.9 39.7z">
        </path>
    </svg>
</a>
 <!-- End Share Button -->
```

Let us **test the functionality**. Start the development server and **open the home page**. We should see a page like this:



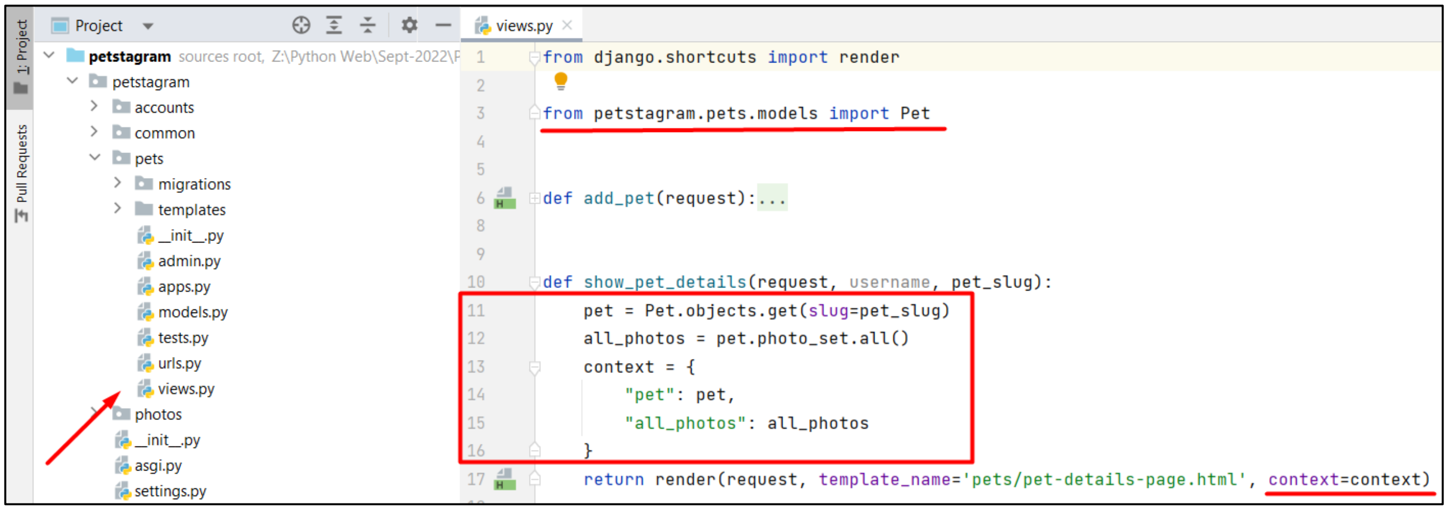
Now, we **can click on the like button**, and it **should turn red**, and the **URL should change**. Now we have **1 like**:



Next, let us click on the **share button**. Again, the page is reloaded, and if we **paste the URL**, it should look like this: **"`127.0.0.1:8000/photos/1/`"** and should **lead to the photo details page**.

Add models to Pet Details Page

The **pet details page contains 2 main parts** - **pet personal data** and **pet photos**. It means that we should add the **`Pet`** object and all its photos to the **view's `context`**:



```python
from django.shortcuts import render

from petstagram.pets.models import Pet


def add_pet(request):...


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

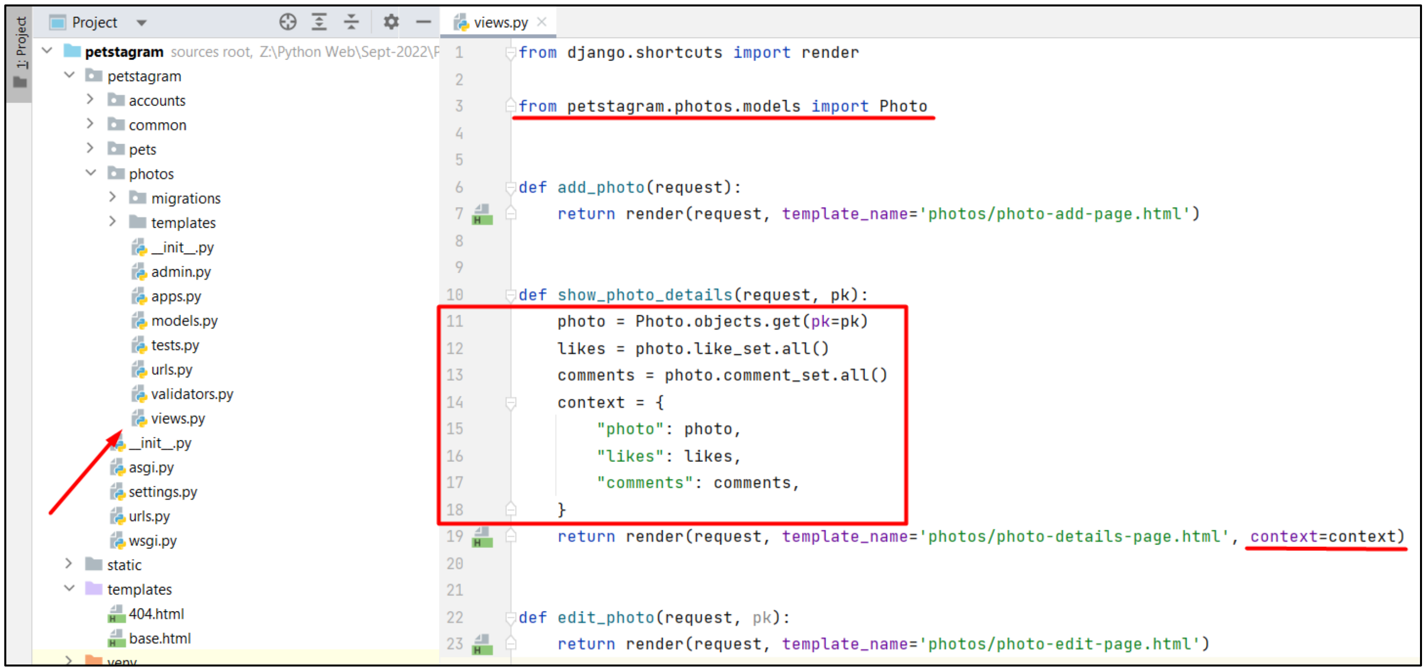
Next, let us **refactor the `pet-details-page.html` template**. We can **add the URL of the pet photo**, the **pet's name**, the **edit** and **delete buttons** paths, and **the total photos count**. We will add the **if statement** that checks if there are photos and **shows all photos of the pet**; otherwise, shows the **default no photos image**:

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

{% block content %}
    <div class="pet-profile">
        <!-- Start Pet Personal Data Section -->
        <div class="profile">
            <div class="profile-data">
                <div class="profile_img">
                    <div class="image">
                        <!-- Pet URL Image -->
                        <img src="{{ pet.personal_photo }}"
                             alt="img8">
                    </div>
                </div>
                <div class="personal">
                    <div class="edit">
                        <!-- Pet Name -->
                        <p>{{ pet.name }}</p>
                        <!-- Pet Edit Button -->
                        <a href="{% url 'edit-pet' "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' "username" pet.slug %}">
                            <img class="bin-img" src="/static/images/icon-remove-
22.jpg" alt="bin button">
                        </a>
                    </div>
                    <div class="data">
                        <!-- Pet Total Photos -->
                        <span>{{ all_photos.count }}</span>
                        <p>photos</p>
                    </div>
                </div>
            </div>
        </div>
        <!-- End Pet Personal Data Section -->
        <div class="pet-posts">

            {% if all_photos %}
                {% include 'common/pets-posts.html' %}
                <!-- IF Photos End Pet Photos Post Section -->
            {% else %}
                <!-- IF NOT Photos Show No Post Image -->
                <img class="no-posts" src="{% static '/images/no_posts.png' %}"
alt="no posts image">
            {% endif %}
        </div>
    </div>

{% endblock %}
```

We **do not need to implement the photo posts context again** - it is already done. We just need to **use the same variable name** for all pet photos - **`all_photos`**.

Add models to Photo Details Page

Last for this workshop, we will **implement the models on the photo details page**. It consists of **`Photo` object** information, **photo likes,** and **comments** - so we need to **get the specific photo from the database**, **all its likes**, and **all its comments**, and **add it to the `context`**:



```python
from django.shortcuts import render

from petstagram.photos.models import Photo


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


def show_photo_details(request, pk):
    photo = Photo.objects.get(pk=pk)
    likes = photo.like_set.all()
    comments = photo.comment_set.all()
    context = {
        "photo": photo,
        "likes": likes,
        "comments": comments,
    }
    return render(request, template_name='photos/photo-details-page.html', context=context)


def edit_photo(request, pk):
    return render(request, template_name='photos/photo-edit-page.html')
```

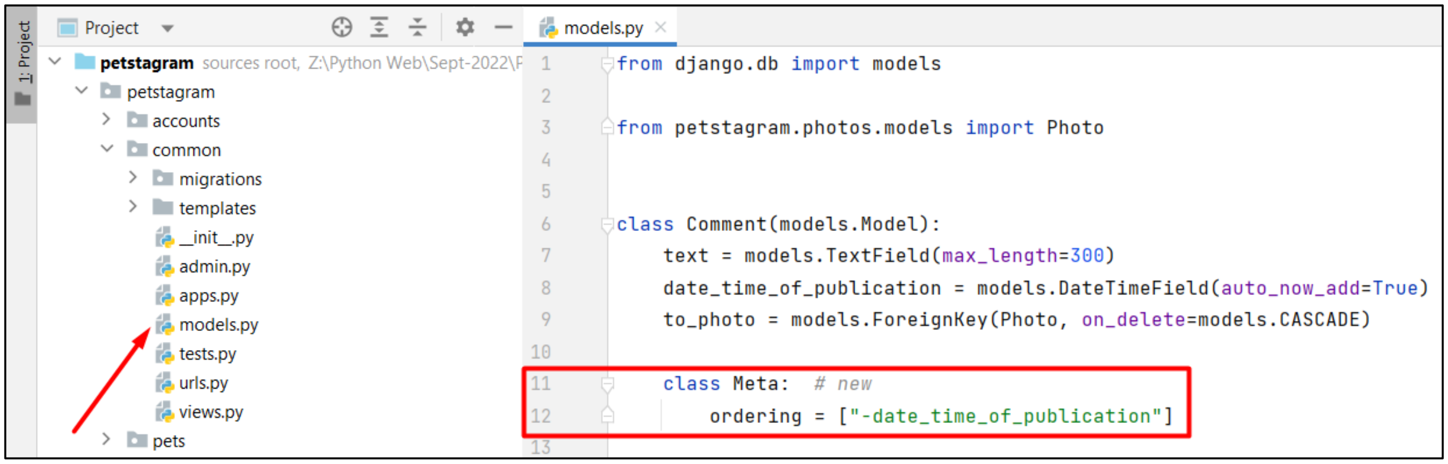
Then, we **open the `photo-details-page.html` template** and we will **add the photo information**, **implement the like and share functionality**, and **add the number of likes for that photo**, **specify the tagged pets**, the photo **description**, and the **date of publication**. And in the end, we will **add the comment object**, containing the **text**, and the

date and time of publication:

```
{% extends 'base.html' %}
...
            <!-- Start Pet Photo Post Section -->
            ...
                            <!-- IF the photo has location -->
                            {% if photo.location %}
                                <span>{{ photo.location }}</span>
                            {% endif %}

                            <!-- IF the viewer is the creator of the photo -->
                            <div class="edit-delete-btns">

                                <!-- 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 -->
                                ...
                    <!-- End User Details and Image Location Section -->
                    ...
                    <!-- Start Like and Share Buttons Section -->
                    <div class="actionBtns">
                        <div class="left">

                            <!-- Start Like Button -->
                            <span class="heart">

                                    <!-- Link to Like Path -->
                           <a href="{% url 'like' photo.id %}">

                              <!-- IF user has liked the photo -->
                                        {% if likes %}
                                            <svg style="color: red"
                                                 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 -->
                                            ...
                                            <!-- IF NOT user has liked the photo -->
                                        {% else %}
                                            <svg aria-label="Like"
                                                 color="#262626"
                                                 fill="#262626"
                                                 height="24"
                                                 role="img"
                                                 viewBox="0 0 48 48"
                                                 width="24">
                                        {% endif %}
                                        <!-- Coordinate path -->
                                        ...
                            <!-- Start Share  Button -->

                            <!-- Link to Share Path -->
                            <a href="{% url 'share' photo.id %}">
                                <svg...>
                    ...
                    <!-- End Like and Share Buttons Section -->

                    <!-- Number of Likes for the Photo -->
                    <p class="likes">{{ likes.count }} likes</p>

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

                <!-- Photo Description -->
                <p class="pet-details">{{ photo.description }}</p>

                <!-- Date of Publication or edit of the Photo -->
                <h5 class="postTime">{{ photo.date_of_publication }}</h5>

                <!-- Start Comments Section -->
               {% for comment in comments %}
                 <div class="comments">
                    <div class="top">
                        <div class="userDetails">
                            <div class="comment-data">
                                <div class="profilepic">
                                    <div class="profile_img">
                                        <div class="image">
                                            <!-- User Profile Image -->
                                            <img src="{% static 'images/person.png'
%}" alt="img8">

                                        </div>
                                    </div>
                                </div>
                                <p>
                                    <!-- Link to User Profile Details Page-->
                                    <!-- User First and/or Last Name or username-->
                                    <a href="">Steven Ivanov</a>
                                    <!-- User Comment -->
                                    {{ comment.text }}

                                </p>
                            </div>
                            <span>{{ comment.date_time_of_publication }}</span>
                        </div>
                    </div>
                 </div>
                <!-- End Comments Section -->
                {% endfor %}
            ...
```

Order Comments

The comments do not appear to be in the order we want. The **last comment** published should **appear first** in the comment section. To do that we can use the model's `class Meta` option "`ordering`" to order the comments in **descending order by the date and time of publication**:



```python
from django.db import models

from petstagram.photos.models import Photo


class Comment(models.Model):
    text = models.TextField(max_length=300)
    date_time_of_publication = models.DateTimeField(auto_now_add=True)
    to_photo = models.ForeignKey(Photo, on_delete=models.CASCADE)

    class Meta:  # new
        ordering = ["-date_time_of_publication"]
```