# Workshop: Petstagram

In today's session, we will be delving into **Django** **template** **inheritance**. Additionally, we'll be implementing comprehensive **CRUD** functionality for our pet application by incorporating **views** and **forms**. Our agenda includes working with **media** **files** and integrating features such as **liking**, **sharing**, and **commenting**.

**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**](https://softuni.bg/downloads/svn/python-web/Sept-2024/Django-Basics/05-Workshop-Part-I/05-Workshop-Description.docx).

## 1. Workshop - Part 2.1

### Creating Template Inheritance

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

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

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

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

{% load static %}  
  
<!DOCTYPE html>  
<html lang="en">  
*<!-- Starts Head Section -->*<head>  
 <link rel="stylesheet" href="{% static 'css/styles.css' %}">  
 <link rel="icon" type="image/x-icon" href="{% static 'images/free-30-instagram-stories-icons23\_122570.png' %}">  
 <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.5.1/css/all.min.css"  
 integrity="sha512-DTOQO9RWCH3ppGqcWaEA1BIZOC6xxalwEsw9c2QQeAIftl+Vegovlnee1c9QX4TctnWMn13TZye+giMm8e2LwA=="  
 crossorigin="anonymous" referrerpolicy="no-referrer"/>  
 <title>Petstagram</title>  
</head>  
*<!-- End Head Section -->  
  
<!-- Starts Body Section -->*<body>  
  
*<!-- Starts Header Section with Navigation Bar -->*<header>  
 <nav class="navbar">  
 <div class="container">  
 <div class="logo">  
 *<!-- Link to Home Page -->* <a href="#">  
 <img width="50px" src="{% static 'images/free-30-instagram-stories-icons23\_122570.png' %}"  
 alt="img1">  
 </a>  
  
 *<!-- Link to Home Page -->* <a class="home" href="{% url 'home' %}">  
 <i>Petstagram</i>  
 </a>  
 </div>  
  
 <div class="nav-links">  
 <ul class="nav-group">  
 <li class="nav-item">  
  
 *<!-- Link to Add Pet Page -->* <a href="{% url 'add-pet' %}">  
 <i>Add Pet</i>  
 </a>  
 </li>  
 <li class="nav-item">  
  
 *<!-- Link to Add Photo Page -->* <a href="{% url 'add-photo' %}">  
 <i>Add Photo</i>  
 </a>  
 </li>

...

...

<li class="nav-item">  
  
 *<!-- Link to Login Page -->* <a href="{% url 'login' %}">  
 <i>Login</i>  
 </a>  
 </li>  
 <li class="nav-item">  
  
 *<!-- Link to Register Page -->* <a href="{% url 'register' %}">  
 <i>Register</i>  
 </a>  
 </li>  
 <li class="nav-item">  
  
 *<!-- Link to Profile Page -->* <a href="#">  
 <i>Profile</i>  
 </a>  
 </li>  
 <li class="nav-item">  
 <a href="#">  
 <i>Logout</i>  
 </a>  
 </li>  
 </ul>  
 </div>  
 </div>  
 </nav>  
</header>  
*<!-- End Header Section with Navigation Bar -->  
  
<!-- Starts Main Section -->*<main>  
  
 {% block content %}  
 {% endblock %}  
  
</main>  
*<!-- End Main Section -->  
  
<!-- Start Footer Section -->*<div class="footer">  
 <span class="footer-section">  
 © 2024 SOFTUNI WORKSHOP FOR PYTHON WEB MODULE  
 </span>  
</div>  
*<!-- End Footer Section -->*</body>  
</html>

Next, we should connect the **base** template with all the other templates. In the **base** template **mark the place where the code should be extended** - it is only the main part. We should **delete the common part of each template** and **add the appropriate tags**. For example, the **register-page** template should look like this:

{% extends 'base.html' %}  
{% block content %}

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

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

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

*<!-- End Register Form -->*  
 </div>  
 <div class="second-option"><p>Have an account?

*<!-- Link to Login Page-->*

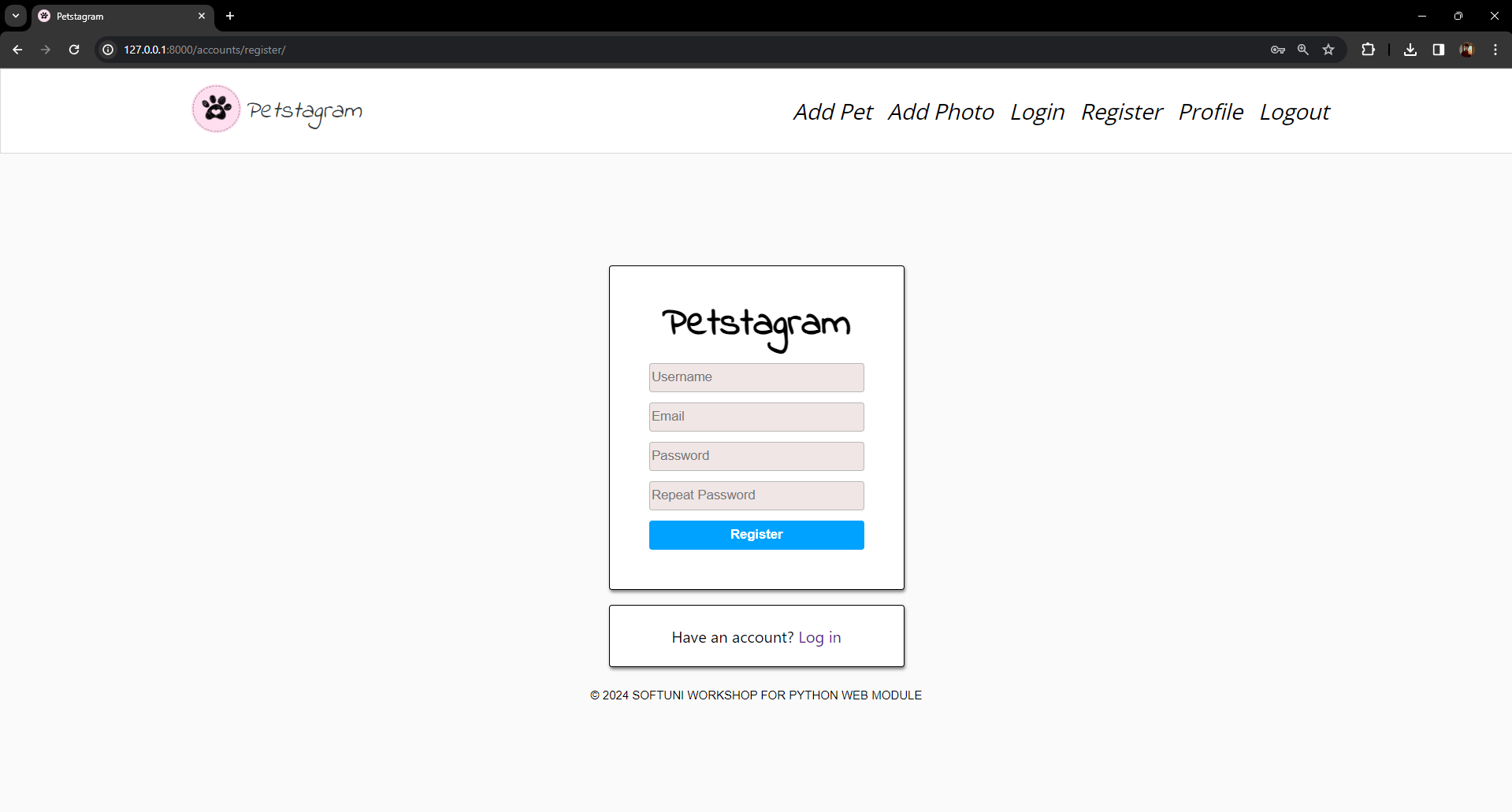
<a href="#">Log in</a></p>

</div>  
 </div>

*<!-- End Register Section-->*

{% endblock %}

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



### Adding Hyperlinks in Templates

In the base template is positioned the **navigation bar**. It navigates the users to different parts of our app. To be fully functional, **hyperlinks should be added** to it. Using Django Template Language, it is not a difficult task to be done. We will use the **url template tag** and map it to the view names that we wrote in the **urlpatterns** list:

*...*

*<!-- Starts Header Section with Navigation Bar -->*<header>  
 <nav class="navbar">  
 <div class="container">  
 <div class="logo">  
 *<!-- Link to Home Page -->* <a href="#">  
 <img width="50px" src="{% static 'images/free-30-instagram-stories-icons23\_122570.png' %}"  
 alt="img1">  
 </a>  
  
 *<!-- Link to Home Page -->* <a class="home" href="{% url 'home' %}">  
 <i>Petstagram</i>  
 </a>  
 </div>  
  
 <div class="nav-links">  
 <ul class="nav-group">  
 <li class="nav-item">  
  
 *<!-- Link to Add Pet Page -->* <a href="{% url 'add-pet' %}">  
 <i>Add Pet</i>  
 </a>  
 </li>  
 <li class="nav-item">  
  
 *<!-- Link to Add Photo Page -->* <a href="{% url 'add-photo' %}">  
 <i>Add Photo</i>  
 </a>  
 </li>  
 <li class="nav-item">  
  
 *<!-- Link to Login Page -->* <a href="{% url 'login' %}">  
 <i>Login</i>  
 </a>  
 </li>  
 <li class="nav-item">  
  
 *<!-- Link to Register Page -->* <a href="{% url 'register' %}">  
 <i>Register</i>  
 </a>  
 </li>

...

...

<li class="nav-item">  
 *<!-- Link to Profile Page -->* <a href="#">  
 <i>Profile</i>  
 </a>  
 </li>  
 <li class="nav-item">  
 <a href="#">  
 <i>Logout</i>  
 </a>  
 </li>  
 </ul>  
 </div>  
 </div>  
 </nav>  
</header>  
*<!-- End Header Section with Navigation Bar -->*

...

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

### Separating Common Parts

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

Картина, която съдържа дъждовник

Описанието е генерирано автоматично

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

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

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

{% extends 'base.html' %}  
{% block content %}  
{% load static %}  
  
 <div class="container">  
 <div class="col-9">  
  
 *<!-- Start Searchbar Form -->* <form class="searchbar" method="post">  
 <input type="text" placeholder="Search by pet name...">  
 <button>  
 <img src="{% static 'images/search.png' %}" height="18" alt="img2">  
 </button>  
 </form>  
 *<!-- End Searchbar Form -->  
  
 <!-- Start Pet Photos Post Section -->*

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

*<!-- End Pet Photos Post Section -->* </div>  
 </div>  
  
{% endblock %}

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

{% extends 'base.html' %}  
{% block content %}  
{% load static %}  
  
 <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="#"  
 alt="img8">  
 </div>  
 </div>  
 <div class="personal">  
 <div class="edit">  
 *<!-- Pet Name -->* <p>Rex</p>  
 *<!-- Pet Edit Button -->* <a href="#">  
 <img class="edit-img" src="{% static 'images/edit-pen-icon-6.jpg' %}" alt="edit button">  
 </a>  
 *<!-- Pet Delete Button -->* <a href="#">  
 <img class="bin-img" src="{% static 'images/icon-remove-22.jpg' %}" alt="bin button">  
 </a>  
 </div>  
 <div class="data">  
 *<!-- Pet Total Photos -->* <span>125</span>  
 <p>photos</p>  
 </div>  
 </div>  
 </div>  
 </div>  
 *<!-- End Pet Personal Data Section -->* <div class="pet-posts">

{% include 'common/pet-posts.html' %}  
<img class="no-posts" src="{% static '/images/no\_posts.png' %}" alt="no posts image">  
 </div>  
 </div>  
  
{% endblock %}

One more thing - you see that there are **3 posts with pet pictures**, so we can **delete two of them** and leave just one that we will work with in the next sessions.

## 2. Workshop - Part 2.2

### 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:

Картина, която съдържа текст, екранна снимка, софтуер, Шрифт

Описанието е генерирано автоматично

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):

{% 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 %}

### 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:

Картина, която съдържа текст, Шрифт, линия, екранна снимка

Описанието е генерирано автоматично

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}'**):

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

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 %}  
 <i class="fa-solid fa-heart" style="color: #ed4040;"></i>

*<!-- else -->*  
 {% else %}  
 <i class="fa-regular fa-heart"></i>  
 {% endif %} ...  
 *<!-- 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 -->*

*...*

*...*

### 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**:

Картина, която съдържа текст, Шрифт, линия, екранна снимка

Описанието е генерирано автоматично

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

Картина, която съдържа текст, Шрифт, линия, екранна снимка

Описанието е генерирано автоматично

**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**:

Картина, която съдържа текст, екранна снимка, софтуер, Уеб страница

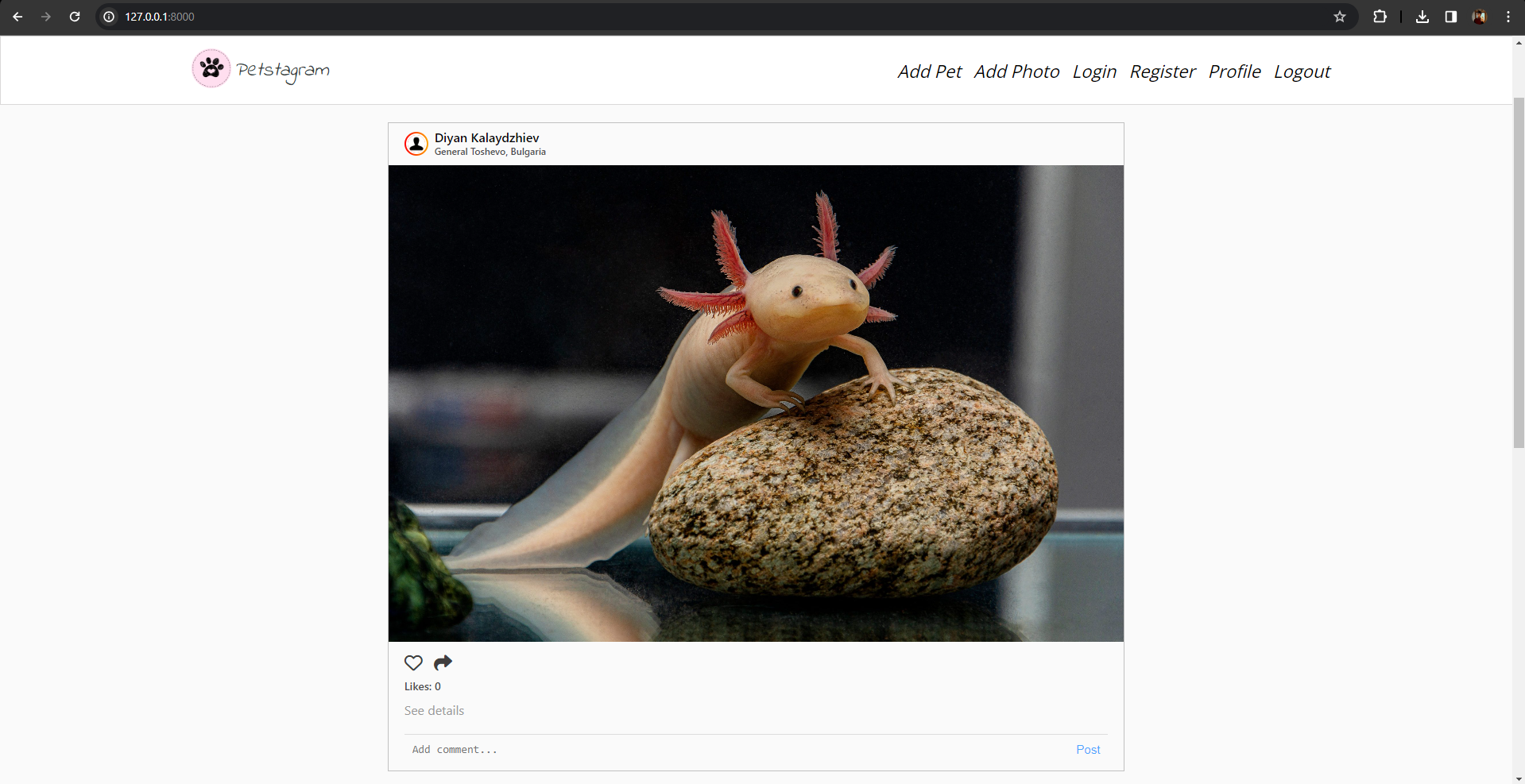
Описанието е генерирано автоматично

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

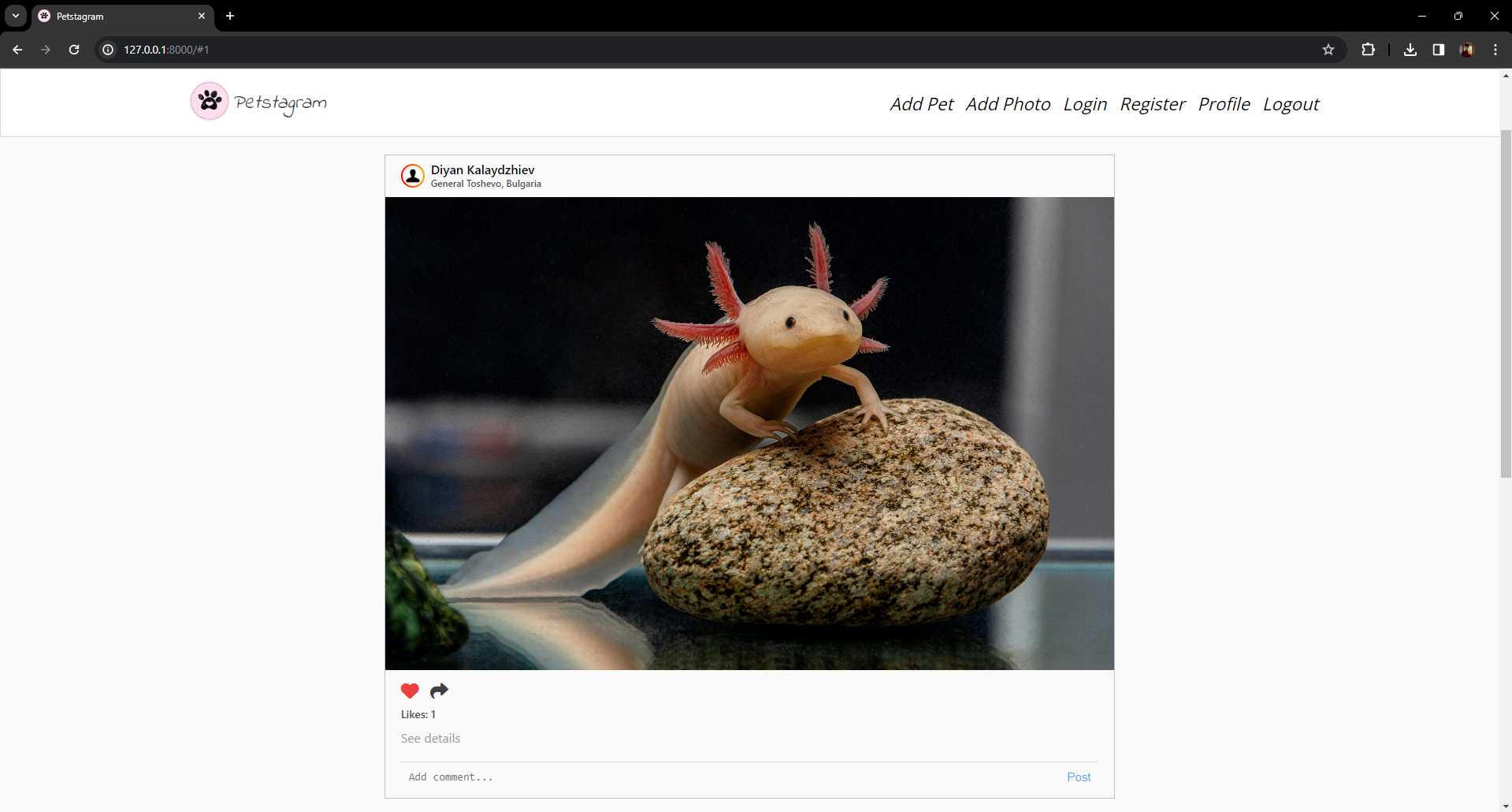
*...*

*<!-- Start Share Button -->*<a href="{% url 'share' photo.id %}">  
 <i class="fa-solid fa-share"></i>  
</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**.

### 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**:

Картина, която съдържа текст, екранна снимка, софтуер, Шрифт

Описанието е генерирано автоматично

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**.

### 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**:

Картина, която съдържа текст, екранна снимка, софтуер

Описанието е генерирано автоматично

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' %}  
{% load static %}  
  
{% block content %}  
 <div class="container">  
 <div class="col-9">  
  
 *<!-- Start Pet Photo Post Section -->* <div class="card">  
 <div class="top">  
  
 *<!-- Start User Details and Image Location Section -->* <div class="userDetails">

...

...

*<!-- Link to User Profile Details Page-->* <a href="#">  
 <div class="profilepic">  
 <div class="profile\_img">  
 <div class="image">  
 <img src="{% static '/images/person.png' %}" alt="img8">  
 </div>  
 </div>  
 </div>  
 </a>  
 <h3>  
  
 *<!-- Link to User Profile Details Page -->* <a href="#">  
 *<!-- IF user has first name and last name -->* Diyan Kalaydzhiev  
 *<!-- else -->  
 <!-- show user username -->* </a>  
 <br>  
  
 *<!-- 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 -->* <a href="#">  
 <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 -->*</div>  
  
*<!-- Start Pet Photo -->*<div class="imgBx" id="place the id of the photo here">  
 *<!-- Pet Photo -->* <img src="{% static '/images/axolotl.jpeg' %}" alt="post" class="cover">  
</div>  
*<!-- End Pet Photo -->*

...

...

<div class="bottom">  
  
 *<!-- 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 photo.like\_set.all %}  
 <i class="fa-solid fa-heart" style="color: #ed4040;"></i>  
 *<!-- else -->* {% else %}  
 <i class="fa-regular fa-heart"></i>  
 {% endif %}  
 </a>  
 </span>  
 *<!-- End Like Button -->  
  
 <!-- Start Share Button -->  
  
 <!-- Link to Share Path -->* <a href="{% url 'share' photo.id %}">  
 <i class="fa-solid fa-share"></i>  
 </a>  
 </div>  
 </div>  
 *<!-- 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 %}  
  
 *<!-- 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 %}  
 </div>  
  
 *<!-- Start Add Comments Section -->* <div class="addComments">  
 *<!-- Start Add Comments Form -->* <form method="post" action="#">  
 <label for="id\_body">Body:</label>  
 <textarea name="body" cols="40" rows="10" placeholder="Add comment..." maxlength="300"  
 required="" id="id\_body"></textarea>  
 <button type="submit">Post</button>  
 </form>  
 *<!-- End Add Comments Form -->* </div>  
 *<!-- End Add Comments Section -->* </div>  
 *<!-- End Pet Photo Post Section -->* </div>  
 </div>  
  
{% endblock %}

### 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**:

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

## 3. Workshop - Part 2.3

### 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:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

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:

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

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**:

Картина, която съдържа текст, екранна снимка, софтуер, Компютърна икона

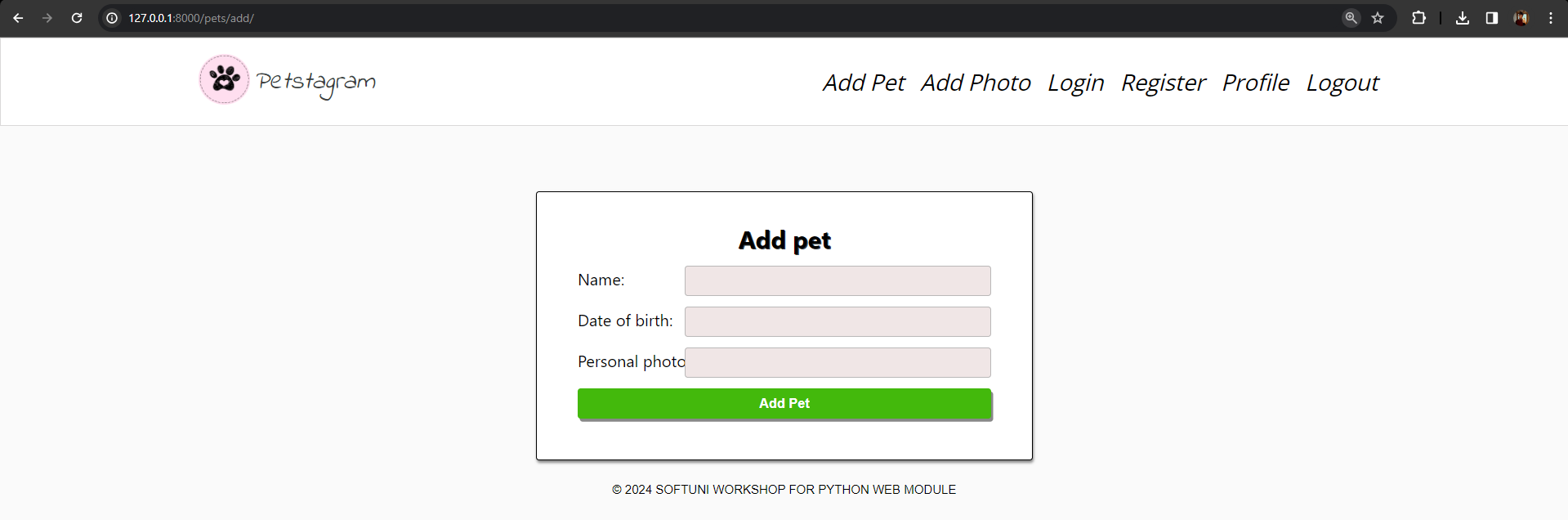
Описанието е генерирано автоматично

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**:

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

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

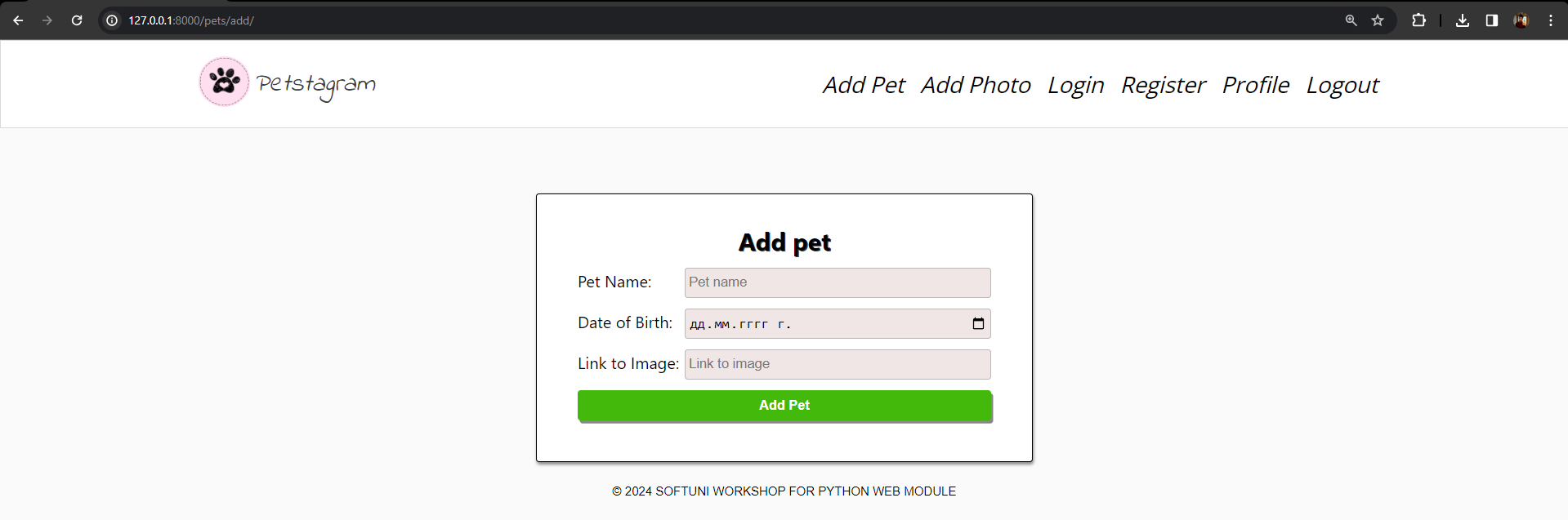


The pet form is **generated** and **work corettly**, but it can look much better. To **improve the UI** we can change some **labels**, and some **placeholders** and **make the date field** visualized with a generated calendar. Let us again **open the** **pet/forms.py** and write some code:

Картина, която съдържа текст, екранна снимка, софтуер, Шрифт

Описанието е генерирано автоматично

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



### 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**:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

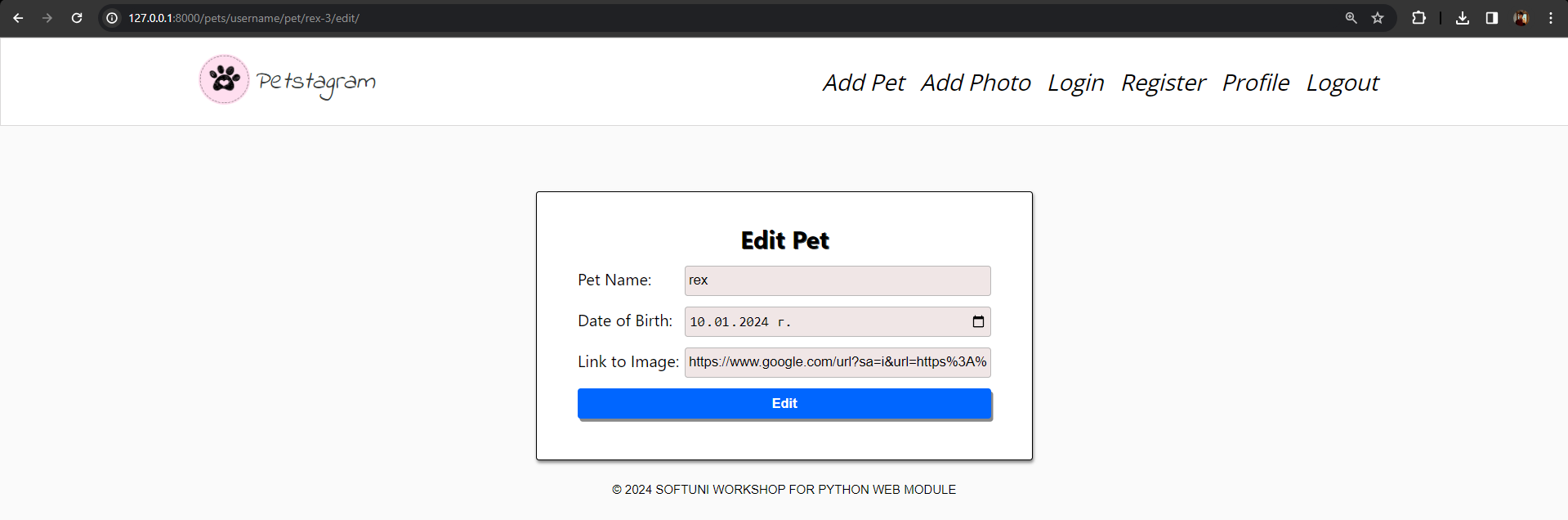
Описанието е генерирано автоматично

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

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

**Check** if the pet edit functionality works correctly:



### 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**:

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

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**:

Картина, която съдържа текст, Шрифт, екранна снимка

Описанието е генерирано автоматично

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

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

Check if the form works correctly.

## 4. Workshop - Part 2.4

### 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**:

Картина, която съдържа текст, екранна снимка, софтуер, Шрифт

Описанието е генерирано автоматично

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

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

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:

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

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

### 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:

Картина, която съдържа текст, номер, Шрифт, софтуер

Описанието е генерирано автоматично

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) :

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

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"**:

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

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**:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

Now, it is time to **refactor the template**, so it visualizes the uploaded image. Let us open the **common** app **pet-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):

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

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

Картина, която съдържа текст, екранна снимка, софтуер, Компютърна икона

Описанието е генерирано автоматично

### 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**:

Картина, която съдържа текст, екранна снимка, номер, софтуер

Описанието е генерирано автоматично

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**:

Картина, която съдържа текст, екранна снимка, Шрифт, номер

Описанието е генерирано автоматично

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**:

Картина, която съдържа текст, екранна снимка, софтуер, дисплей

Описанието е генерирано автоматично

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:

Картина, която съдържа текст, софтуер, Шрифт, номер

Описанието е генерирано автоматично

## 5. Workshop - Part 2.5

### 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 greatto **add a placeholder** ("Add comment...") that will guide the user: Картина, която съдържа текст, софтуер, екранна снимка, номер

Описанието е генерирано автоматично

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**:

Картина, която съдържа текст, екранна снимка, софтуер, Уеб страница

Описанието е генерирано автоматично

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:

Картина, която съдържа текст, екранна снимка, Шрифт

Описанието е генерирано автоматично

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**:

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

Now, we should **refactor the template**. Let us open the **pet-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**:

Картина, която съдържа текст, екранна снимка, софтуер, номер

Описанието е генерирано автоматично

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.

### 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**:

Картина, която съдържа текст, екранна снимка, номер, софтуер

Описанието е генерирано автоматично

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 GET, 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:

Картина, която съдържа текст, екранна снимка, софтуер, Уеб страница

Описанието е генерирано автоматично

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

Картина, която съдържа текст, екранна снимка, софтуер, дисплей

Описанието е генерирано автоматично

### Search Form Styles

For some **devices** the search bar may have issues:

Картина, която съдържа текст, екранна снимка

Описанието е генерирано автоматично

We can fix it by only modifying one **css** property. Open the **css/home** and change the value of the **margin-top to 2.45rem**:

Картина, която съдържа текст, екранна снимка, Шрифт, софтуер

Описанието е генерирано автоматично

Now the **UI** is working as it should be:

Картина, която съдържа Мустаци, Малки и средни по размер котки, текст, екранна снимка

Описанието е генерирано автоматично