ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ

УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Кафедра «Програмної інженерії»

Звіт з Лабораторної роботи №1

з дисципліни «Високорівневі мови програмування та фреймворки»

Виконав:

ст. гр. ПЗПІ-20-4 Прудіус В. Ю. Перевірила:

ас. каф. ПІ

Зибіна К. В.

2023

**Тема:**

Ознайомлення з основами Python та фреймворку Django.

**Мета:**

Ознайомитись з основами Python та навчитись створювати прості веб додатки з використанням фреймворку Django.

**Хід роботи:**

До початку виконання завдання, було використано скрипт, який автоматично створює проект на Django. Цей файл було взято на сторінці курсу на ресурсі dl.nure. Після виконання скрипту, ми отримали проект з наступним змістом:

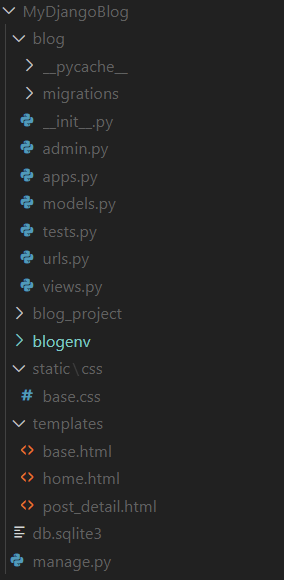
****

Рис 1 - Вміст створеного проекту

Також було отримано завдання, модифікувати створений проект. Для цього було обрано наступну функціональність:

* Реалізація пошуку за назвою посту;
* Реалізація функціональності з тегами;
* Реалізація фільтрації постів по тегу;

Для реалізації цього, було модифіковано код системи. По перше додано новий клас моделі, це клас Tag який відображує тег, а саме його назву та колір. А по друге додано зв’язок багато до багатьох між постом та тегом. Таким чином ми отримали наступний вміст файлу models.py:

**from django.db import models**

**class Tag(models.Model):**

**name = models.CharField(max\_length=50)**

**color = models.CharField(max\_length=10)**

**def \_\_str\_\_(self):**

**return self.name**

**class Post(models.Model):**

**title = models.CharField(max\_length=200)**

**author = models.ForeignKey(**

**'auth.User',**

**on\_delete=models.CASCADE,**

**)**

**body = models.TextField()**

**tags = models.ManyToManyField(Tag)**

**def \_\_str\_\_(self):**

**return self.title**

Наступним кроком, є налаштування класів відображень (View) задля того, щоб було можливо підвантажувати та відображати теги для постів. Тут основним моментом є реалізація у відкритому вигляді методів GET для обох вью, де зусереджено нову просунуту функціональність. Для відображення списку постів було додано параметри запиту, звідки беруться аргументи для фільтрацій постів. А також додано підвантаження тегів, до вью детального перегляду посту.

**from django.views.generic import ListView, DetailView # нове**

**from django.db.models import Q**

**from django.shortcuts import render**

**from .models import Post, Tag**

**class BlogListView(ListView) :**

**model = Post**

**template\_name = 'home.html'**

**def get(self, request):**

**search = request.GET.get("search", False)**

**tag = request.GET.get("tag", False)**

**posts = Post.objects.all()**

**response = []**

**if search :**

**query = Q(title\_\_icontains=search)**

**posts = posts.filter(query).select\_related()**

**if tag :**

**query = Q(tags\_\_name\_\_icontains=tag)**

**posts = posts.filter(query).select\_related()**

**for post in posts:**

**tags = post.tags.all()**

**response.append({'post': post, 'tags': tags})**

**context = {'data' : response, 'search': search, 'tag': tag}**

**return render(request, self.template\_name, context)**

**class BlogDetailView(DetailView): # нове**

**model = Post**

**template\_name = 'post\_detail.html'**

**def get(self, request, pk) :**

**selected = Post.objects.get(id=pk)**

**tags = Tag.objects.filter(post=selected)**

**context = { 'post': selected, 'tags': tags }**

**return render(request, self.template\_name, context)**

І останній крок – модифікація темплейтів, задля можливості відображення нових даних та впровадження нового функціоналу:

Код темплейту головної сторінки, наприклад отримав нову форму, яка надсилає запити на поточну адресу з аргументами для пошуку, а також в код генерації було додано вкладений цикл для відображення тегів:

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

**{% block content %}**

**<form>**

**<input**

**type="search"**

**placeholder="Search..."**

**name="search"**

**{% if search %} value="{{ search }}" {% endif %}>**

**<input**

**type="search"**

**placeholder="Tag..."**

**name="tag"**

**{% if tag %} value="{{ tag }}" {% endif %}>**

**<button type="submit">Search</button>**

**</form>**

**{% for item in data %}**

**<div class="post-entry">**

**<h2><a href="{% url 'post\_detail' item.post.pk %}">{{ item.post.title }}</a></h2>**

**<p>{{ item.post.body }}</p>**

**{% for tag in item.tags %}**

**<p class="tag" style="background: {{ tag.color }}">{{ tag.name }}</p>**

**{% endfor %}**

**</div>**

**{% endfor %}**

**{% endblock content %}**

В результаті ми отримали такий веб додаток:

Зображення, що містить текст, знімок екрана, Шрифт, програмне забезпечення

Автоматично згенерований опис

Рис 2 – зовнішній вигляд головної сторінки

Зображення, що містить текст, знімок екрана, Шрифт, ряд

Автоматично згенерований опис

Рис 3 – зовнішній вигляд головної сторінки після пошуку і фільтрації

Зображення, що містить текст, знімок екрана, Шрифт, дизайн

Автоматично згенерований опис

Рис 4 – зовнішній вигляд сторінки детального перегляду посту

Також можливе адміністрування даних, за допомогою вбудованої в джанго функціональності. Адмін-панель має наступний вигляд:

Зображення, що містить текст, знімок екрана, програмне забезпечення, Мультимедійне програмне забезпечення

Автоматично згенерований опис

Рис 5 – зовнішній вигляд сторінки адміністрування

**Висновки:** В ходілабораторної роботи ми ознайомились з основами Python та навчились створювати прості веб додатки з використанням фреймворку Django.