KAZAKH-BRITISH TECHNICAL UNIVERSITY

KBTU

Kazakh-British Technical University

Assignment 3

Web Application Development

Done by : Sarsengaliyev Zhaisan

Checked by: Serek Azamat

Almaty, 2024

CONTENT

Introduction 3

1. Django Models 4

2. Django Views 7

3. Django Templates.......... 11

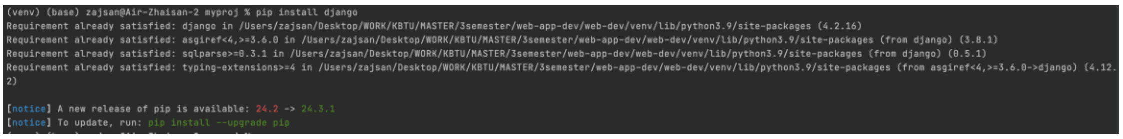
Conclusion 16

INTRODUCTION

In this lab assignment I developed basic blog application using the Django framework. Project includes creating, configuring models, views, templates and forms to manage blog posts. I also added styling with CSS and also added image uploads for posts. Throughout this work I explored using class-based views, template inheritance and handling static and media files. This project allows users to create, edit and view posts, and also applying basic design elements to improve the user interface.

1. Django Models

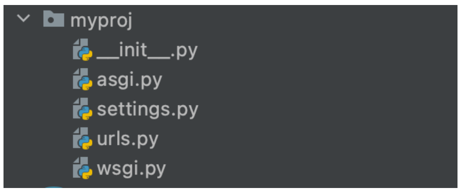
Exercise 1: Creating a Basic Model



```
(venv) (base) zajsan@Air-Zhaisan-2 myproj % pip install django
Requirement already satisfied: django in /Users/zajsan/Desktop/WORK/KBTU/MASTER/3semester/web-app-dev/web-dev/venv/lib/python3.9/site-packages (4.2.16)
Requirement already satisfied: asgiref<4,>=3.6.0 in /Users/zajsan/Desktop/WORK/KBTU/MASTER/3semester/web-app-dev/web-dev/venv/lib/python3.9/site-packages (from django) (3.8.1)
Requirement already satisfied: sqlparse>=0.3.1 in /Users/zajsan/Desktop/WORK/KBTU/MASTER/3semester/web-app-dev/web-dev/venv/lib/python3.9/site-packages (from django) (0.5.1)
Requirement already satisfied: typing-extensions>=4 in /Users/zajsan/Desktop/WORK/KBTU/MASTER/3semester/web-app-dev/web-dev/venv/lib/python3.9/site-packages (from asgiref<4,>=3.6.0->django) (4.12.
2)

[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: pip install --upgrade pip
```

Figure 1.1: Installing Django

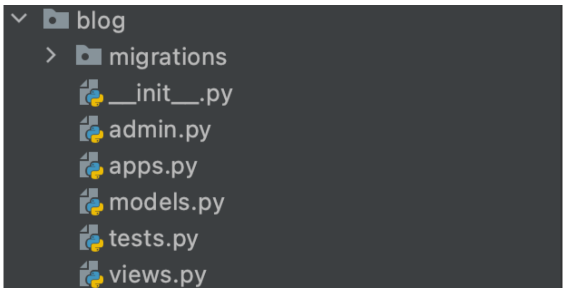
In figure 1.1 I make sure that Django is installed using *‘pip install django’* command.



```
myproj
    __init__.py
    asgi.py
    settings.py
    urls.py
    wsgi.py
```

Figure 1.2: Creating Django project

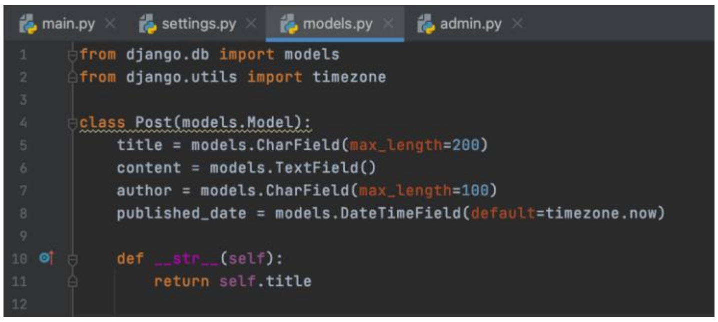
Next, I created Django project using *‘django-admin startproject myproj’* command.



```
blog
  > migrations
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    views.py
```

Figure 1.3: Creating app inside project

After I created app called ‘blog’ in project folder.



main.py | settings.py | models.py | admin.py

```python
from django.db import models
from django.utils import timezone

class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    author = models.CharField(max_length=100)
    published_date = models.DateTimeField(default=timezone.now)

    def __str__(self):
        return self.title

```

Figure 1.4: Models

Inside ‘*blog/models.py’* I created models for posts with the fields:

- Title: post title, string with 200 characters long.
- Content: post content, field for storing text.
- Author: author name, string with 100 characters long.
- Published_date: post publication date and time, current date and time by default.

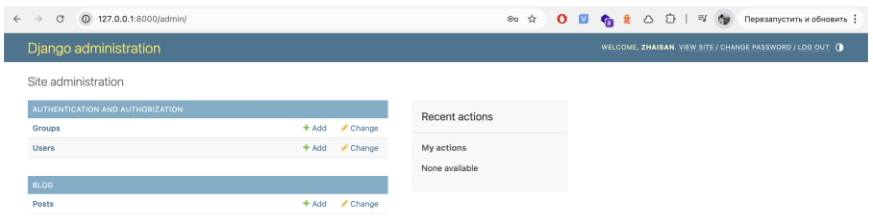
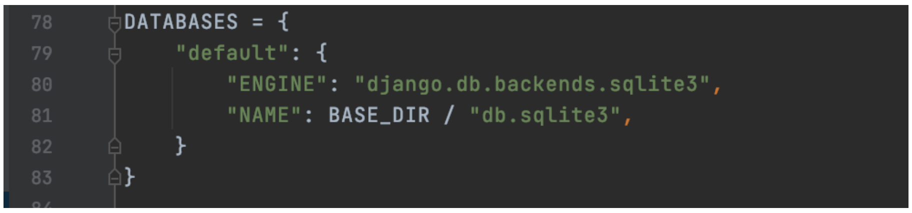


Figure 1.5: Admin panel

In admin panel figure 1.5 I made sure that our models are created.

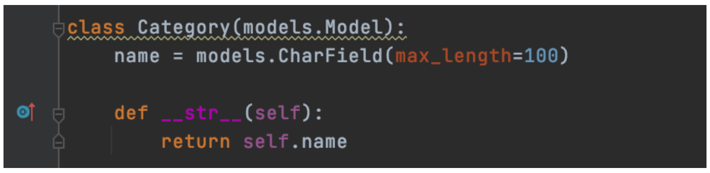


```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.sqlite3",
        "NAME": BASE_DIR / "db.sqlite3",
    }
}
```

Figure 1.6: SQLite3

By the way, for this project I use SQLite3. I use the default database which is quite enough to cover all our lab work needs.

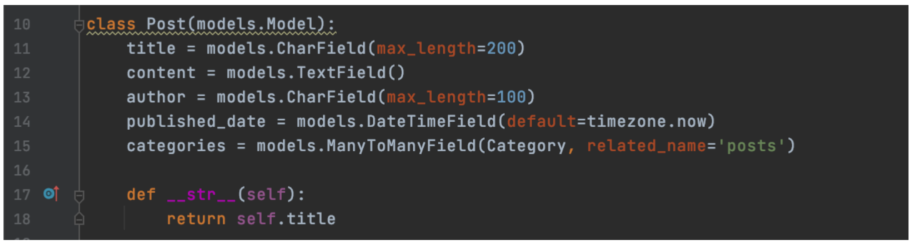
Exercise 2: Model Relationships



```
class Category(models.Model):
    name = models.CharField(max_length=100)

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

Figure 1.7: Category model

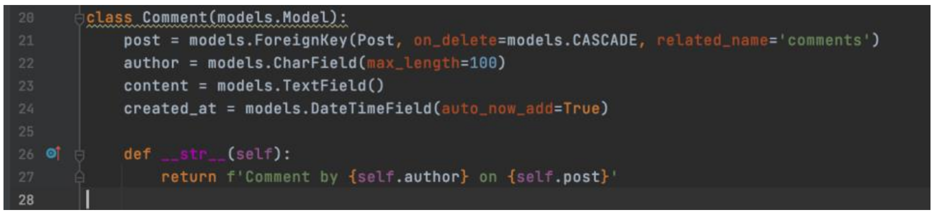
In figure 1.7 you see that I included Category model with many-to-many relationship.



```
class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    author = models.CharField(max_length=100)
    published_date = models.DateTimeField(default=timezone.now)
    categories = models.ManyToManyField(Category, related_name='posts')

    def __str__(self):
        return self.title
```

Figure 1.8: Category field

Here I added '*categories'* field to model '*Post'* and I define it as ManyToManyField indicating to model '*Category'*.

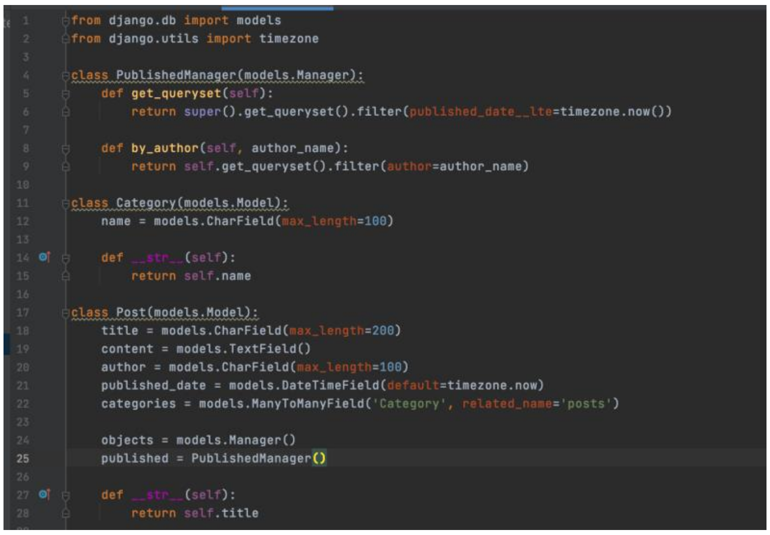


```
class Comment(models.Model):
    post = models.ForeignKey(Post, on_delete=models.CASCADE, related_name='comments')
    author = models.CharField(max_length=100)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f'Comment by {self.author} on {self.post}'
```

Figure 1.9: Comment model

In this figure 1.9 I added '*Comment'* model with oneToMany relationship to '*Post'*. One post can have many comments, but each comment can be related to only one post. 'on_delete=models.CASCADE' means that if the post is deleted, the all comments which are related to the post also will be deleted.

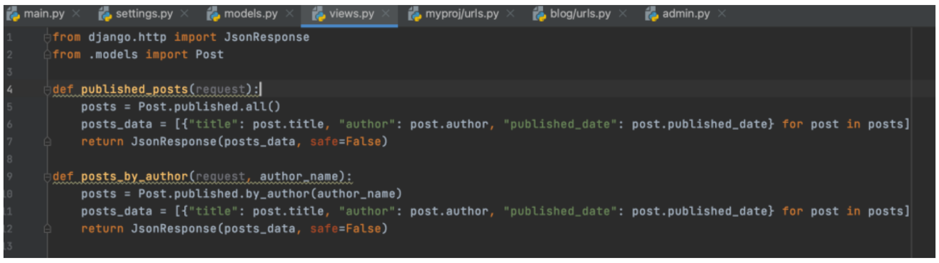
Exercise 3: Custom Manager



```
from django.db import models
from django.utils import timezone

class PublishedManager(models.Manager):
    def get_queryset(self):
        return super().get_queryset().filter(published_date__lte=timezone.now())

    def by_author(self, author_name):
        return self.get_queryset().filter(author=author_name)

class Category(models.Model):
    name = models.CharField(max_length=100)

    def __str__(self):
        return self.name

class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    author = models.CharField(max_length=100)
    published_date = models.DateTimeField(default=timezone.now)
    categories = models.ManyToManyField('Category', related_name='posts')

    objects = models.Manager()
    published = PublishedManager()

    def __str__(self):
        return self.title
```

Figure 1.10: Manager for Post

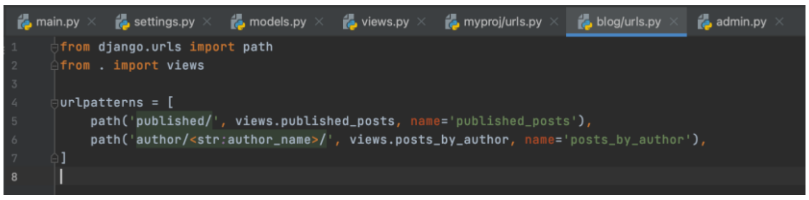
I created custom manager which will manage collection of published posts and filtering them by the author. This manager will be inherited from models.Manager and in it I will define methods for filtering posts.



```python
from django.http import JsonResponse
from .models import Post

def published_posts(request):
    posts = Post.published.all()
    posts_data = [{"title": post.title, "author": post.author, "published_date": post.published_date} for post in posts]
    return JsonResponse(posts_data, safe=False)

def posts_by_author(request, author_name):
    posts = Post.published.by_author(author_name)
    posts_data = [{"title": post.title, "author": post.author, "published_date": post.published_date} for post in posts]
    return JsonResponse(posts_data, safe=False)
```

Figure 1.11: Views

In this *views.py* I added views for posts using the *'titile', 'author', 'published_date'* fields. I use JsonResponse to respod with JSON data.



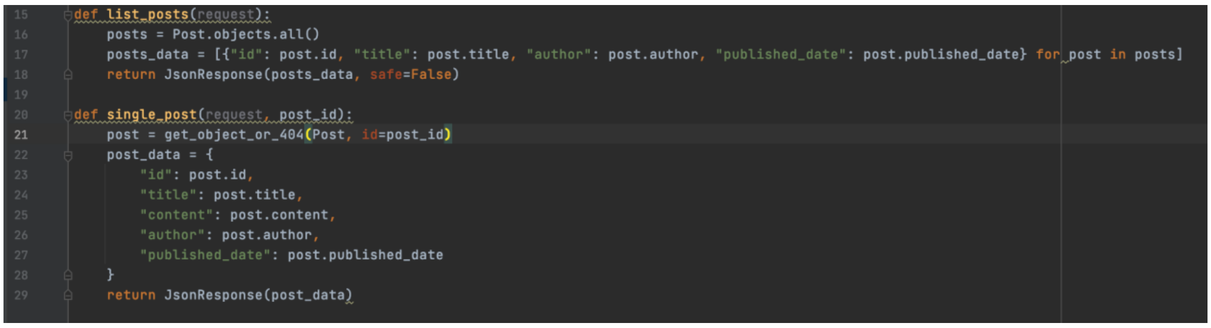
```python
from django.urls import path
from . import views

urlpatterns = [
    path('published/', views.published_posts, name='published_posts'),
    path('author/<str:author_name>/', views.posts_by_author, name='posts_by_author'),
]
```

Figure 1.12: Blog URLs

To connect views with URL addresses I added routes.

2. Django Views

Exercise 4: Function-Based Views



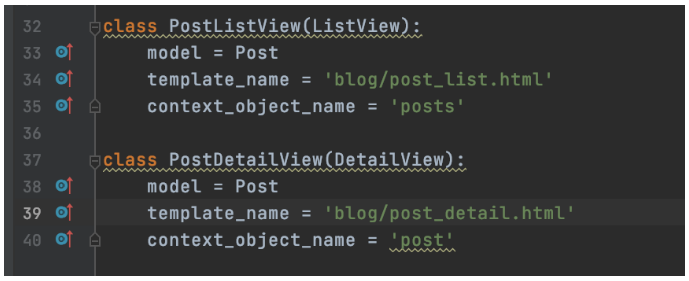
```python
def list_posts(request):
    posts = Post.objects.all()
    posts_data = [{"id": post.id, "title": post.title, "author": post.author, "published_date": post.published_date} for post in posts]
    return JsonResponse(posts_data, safe=False)

def single_post(request, post_id):
    post = get_object_or_404(Post, id=post_id)
    post_data = {
        "id": post.id,
        "title": post.title,
        "content": post.content,
        "author": post.author,
        "published_date": post.published_date
    }
    return JsonResponse(post_data)
```

Figure 2.1: Views

Here I added two functions. First one is '*list_posts*' which returns all posts in json format. The '*Post.objects.all()*' is used to get all posts. Also, the single post function is implemented. It returns single posts by using its id. '*get_object_or_404(Post, id=post_id)*' tries to get post by id, but if the post is not found, I return 404 error.

Exercise 5: Class-Based Views

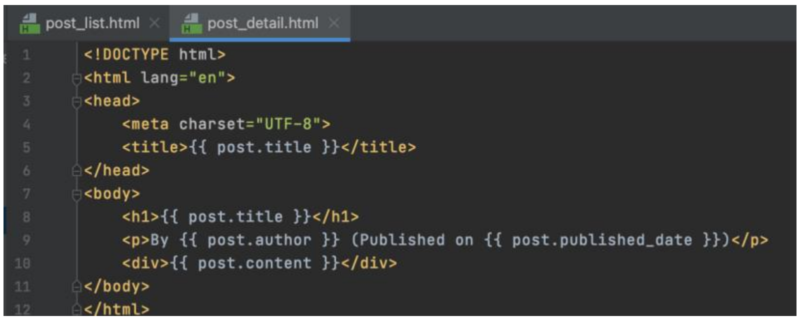
In views file I added ‘*PostListView’* class which uses ‘*ListView’* to return list of posts.



```python
class PostListView(ListView):
    model = Post
    template_name = 'blog/post_list.html'
    context_object_name = 'posts'

class PostDetailView(DetailView):
    model = Post
    template_name = 'blog/post_detail.html'
    context_object_name = 'post'
```

Figure 2.2: Views

In this class I indicated the path to rendering template. Also in the second class ‘*PostDetailView’* for returning details of one specific post.

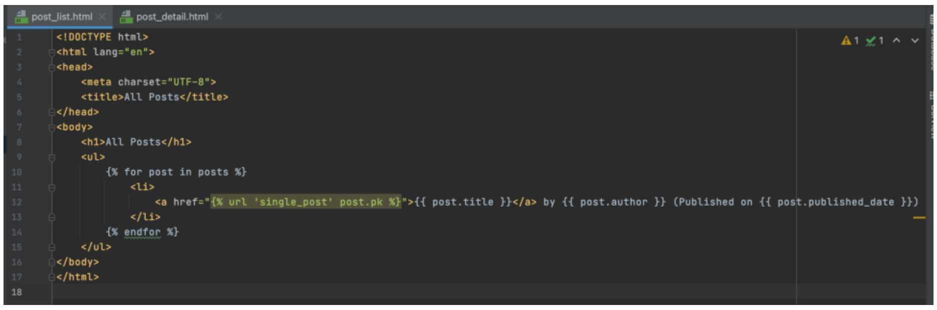


post_list.html post_detail.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{{ post.title }}</title>
</head>
<body>
    <h1>{{ post.title }}</h1>
    <p>By {{ post.author }} (Published on {{ post.published_date }})</p>
    <div>{{ post.content }}</div>
</body>
</html>
```

Figure 2.3: HTML file for post details

This html file illustrates the details of post after clicking it.

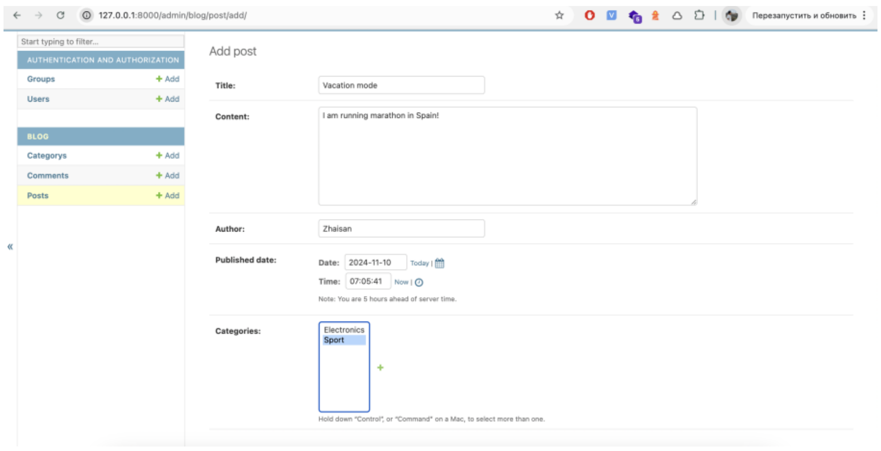


post_list.html post_detail.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>All Posts</title>
</head>
<body>
    <h1>All Posts</h1>
    <ul>
        {% for post in posts %}
            <li>
                <a href="{% url 'single_post' post.pk %}">{{ post.title }}</a> by {{ post.author }} (Published on {{ post.published_date }})
            </li>
        {% endfor %}
    </ul>
</body>
</html>
```

Figure 2.4: HTML file for listing posts

This html file is to illustrate the list of posts.


127.0.0.1:8000/admin/blog/post/add/
Перезапустить и обновить
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
BLOG
Categorys + Add
Comments + Add
Posts + Add

Add post

Title: Vacation mode

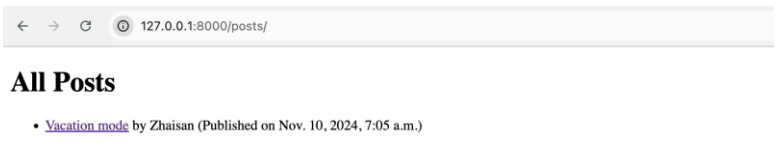
Content: I am running marathon in Spain!

Author: Zhaisan

Published date: Date: 2024-11-10 Today | Time: 07:05:41 Now |
Note: You are 5 hours ahead of server time.

Categories: Electronics, Sport
Hold down "Control", or "Command" on a Mac, to select more than one.

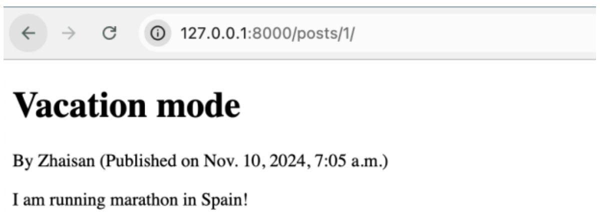
Figure 2.5: Creating post from admin panel



All Posts

- Vacation mode by Zhaisan (Published on Nov. 10, 2024, 7:05 a.m.)

Figure 2.6: Listing posts



Vacation mode

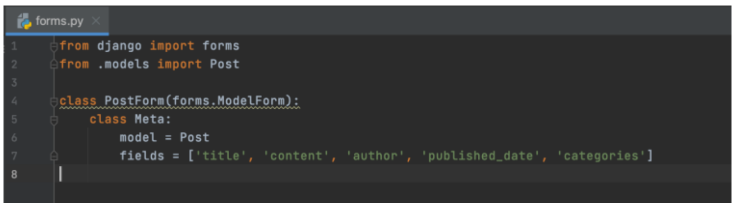
By Zhaisan (Published on Nov. 10, 2024, 7:05 a.m.)

I am running marathon in Spain!

Figure 2.7 Post detail

In figures 2.5, 2.6, 2.7 I check that operations work well in simple HTML.

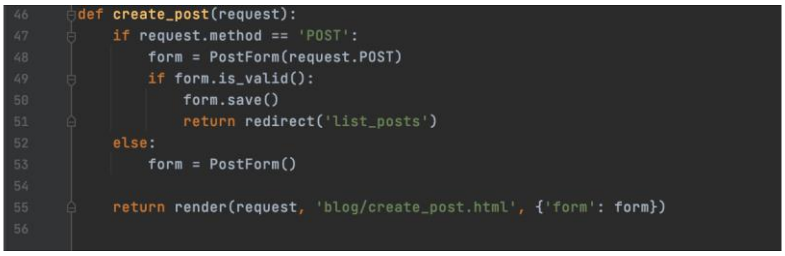
Exercise 6: Handling Forms



```python
from django import forms
from .models import Post

class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ['title', 'content', 'author', 'published_date', 'categories']
```

Figure 2.8: Creating form PostForm

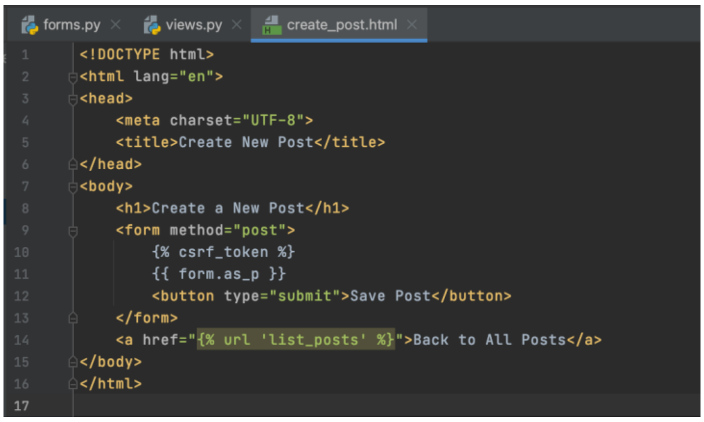
In this figure I declared class which PostForm inherits from forms.ModelForm which simplifies form creation by automatically adding Post model fields.



```
def create_post(request):
    if request.method == 'POST':
        form = PostForm(request.POST)
        if form.is_valid():
            form.save()
            return redirect('list_posts')
    else:
        form = PostForm()

    return render(request, 'blog/create_post.html', {'form': form})
```

Figure 2.9: View create_post

Here the view *create_post* which shows the form for creating new post and handles its sending.

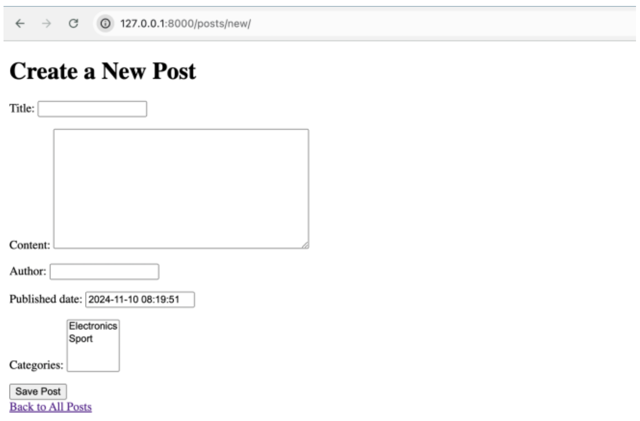


forms.py views.py create_post.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Create New Post</title>
</head>
<body>
    <h1>Create a New Post</h1>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Save Post</button>
    </form>
    <a href="{% url 'list_posts' %}">Back to All Posts</a>
</body>
</html>
```

Figure 2.10: HTML for creating post

'{% csrf_token %}' adds CSRF protection which is required for POST requests in Django. *'{{ form.as_p }}'* automatically renders the form with fields wrapped in <p> tags for easier rendering.



127.0.0.1:8000/posts/new/

Create a New Post

Title:

Content:

Author:

Published date: 2024-11-10 08:19:51

Categories: Electronics Sport

Save Post

Back to All Posts

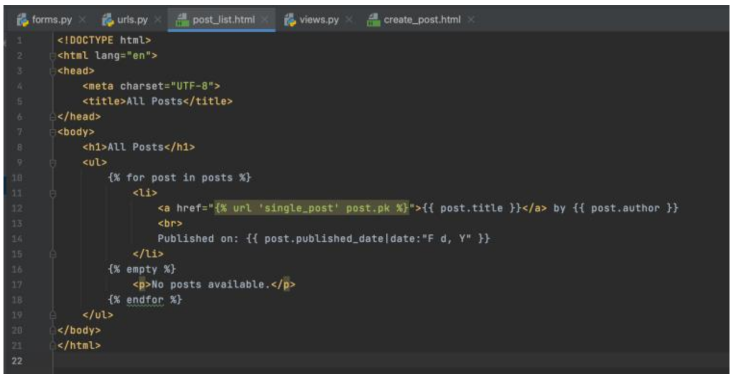
Figure 2.11: Rendered page for creating post

The new created forms for creating post is illustrated in this figure 2.11.

3. Django Templates

Exercise 7: Basic Template Rendering

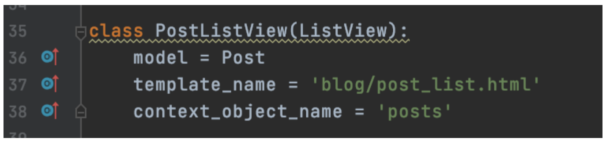
Now I create templates for reflecting the list of posts. In template will be shown titles, authors, and dates of publication. For formatting dates of publication I use built-in template tags of Django.



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>All Posts</title>
</head>
<body>
    <h1>All Posts</h1>
    <ul>
        {% for post in posts %}
            <li>
                <a href="{% url 'single_post' post.pk %}">{{ post.title }}</a> by {{ post.author }}
                <br>
                Published on: {{ post.published_date|date:"F d, Y" }}
            </li>
        {% empty %}
            <p>No posts available.</p>
        {% endfor %}
    </ul>
</body>
</html>
```

Figure 3.1: List HTML

In this figure I used *‘{{ post.published_date|date:"F d, Y" }}’* built-in date template tag that formats publication date in Month Day, Year format (e.g. November 10, 2024). This formatting makes the date more readable.



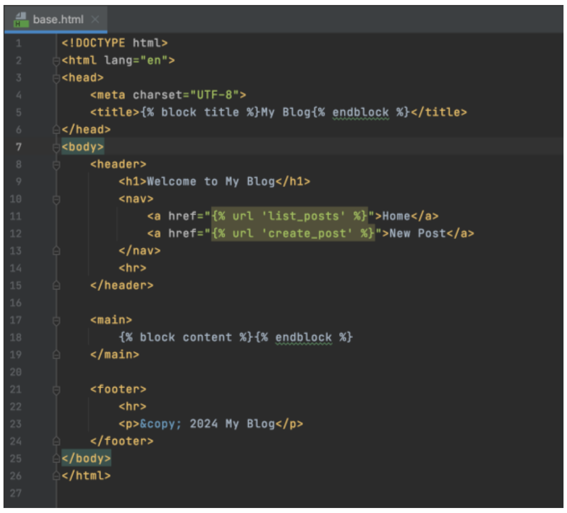
```python
class PostListView(ListView):
    model = Post
    template_name = 'blog/post_list.html'
    context_object_name = 'posts'
```

Figure 3.2: View which uses template

In this figure I created the view which indicates the template.

Exercise 8: Template Inheritance

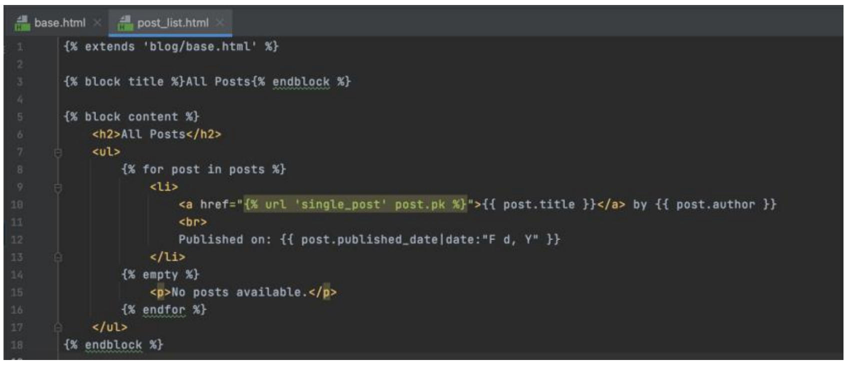
Now I create base template which includes header, footer, and I configure for listing and getting specific posts to inherit this base template. This helps to avoid code duplication and simplifies structure of templates.



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}My Blog{% endblock %}</title>
</head>
<body>
    <header>
        <h1>Welcome to My Blog</h1>
        <nav>
            <a href="{% url 'list_posts' %}">Home</a>
            <a href="{% url 'create_post' %}">New Post</a>
        </nav>
        <hr>
    </header>

    <main>
        {% block content %}{% endblock %}
    </main>

    <footer>
        <hr>
        <p>&copy; 2024 My Blog</p>
    </footer>
</body>
</html>
```

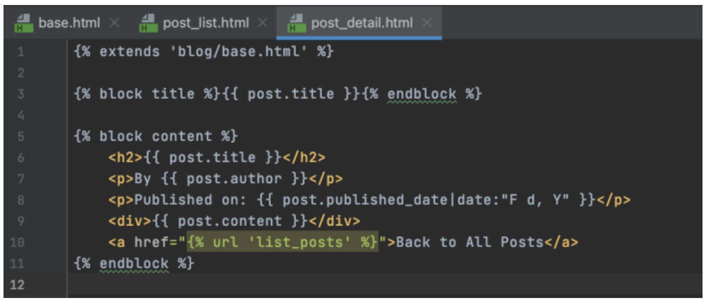
Figure 3.3: Base template

This template has header and footer which are common for all pages of our app.



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

{% block title %}All Posts{% endblock %}

{% block content %}
    <h2>All Posts</h2>
    <ul>
        {% for post in posts %}
            <li>
                <a href="{% url 'single_post' post.pk %}">{{ post.title }}</a> by {{ post.author }}
                <br>
                Published on: {{ post.published_date|date:"F d, Y" }}
            </li>
        {% empty %}
            <p>No posts available.</p>
        {% endfor %}
    </ul>
{% endblock %}
```

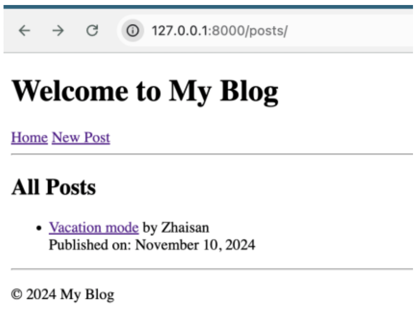
Figure 3.4: List template with base HTML



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

{% block title %}{{ post.title }}{% endblock %}

{% block content %}
    <h2>{{ post.title }}</h2>
    <p>By {{ post.author }}</p>
    <p>Published on: {{ post.published_date|date:"F d, Y" }}</p>
    <div>{{ post.content }}</div>
    <a href="{% url 'list_posts' %}">Back to All Posts</a>
{% endblock %}
```

Figure 3.5: Post Detail template with base HTML

In Figures 3.4, 3.5 I added base HTML parts. *'{% extends 'blog/base.html' %}'* specifies that the post_list.html template extends the base template base.html.


127.0.0.1:8000/posts/

Welcome to My Blog

Home New Post

All Posts

- Vacation mode by Zhaisan
  Published on: November 10, 2024

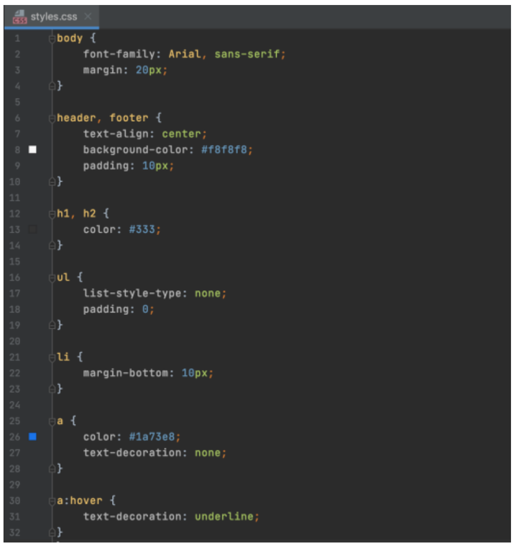
© 2024 My Blog

Figure 3.6: Page with header and footer

In this figure I make sure that header and footer are illustrated correctly.

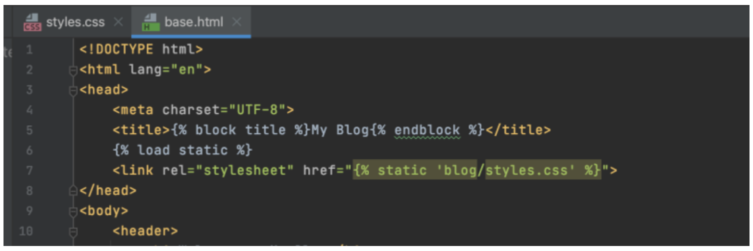
Exercise 9: Static Files and Media

Here I add CSS file to our templates using static files, and also I configure folder for media files to make users to upload images for posts.



```css
body {
    font-family: Arial, sans-serif;
    margin: 20px;
}

header, footer {
    text-align: center;
    background-color: #f8f8f8;
    padding: 10px;
}

h1, h2 {
    color: #333;
}

ul {
    list-style-type: none;
    padding: 0;
}

li {
    margin-bottom: 10px;
}

a {
    color: #1a73e8;
    text-decoration: none;
}

a:hover {
    text-decoration: underline;
}
```

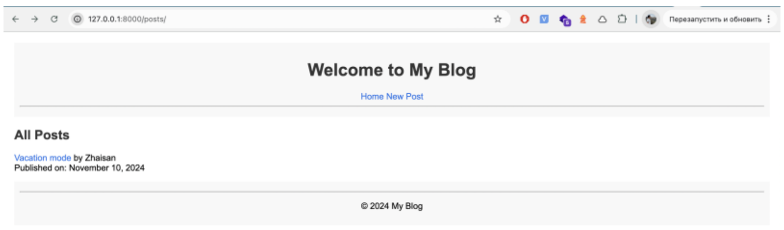
Figure 3.7: Styles.css file



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}My Blog{% endblock %}</title>
    {% load static %}
    <link rel="stylesheet" href="{% static 'blog/styles.css' %}">
</head>
<body>
    <header>
```

Figure 3.8: HTML with styles

In figure 3.7 I created css file for styling our app. And in figure 3.8 in base template I linked css file.


127.0.0.1:8000/posts/

Welcome to My Blog

Home New Post

All Posts

Vacation mode by Zhaisan
Published on: November 10, 2024

© 2024 My Blog

Figure 3.9: Styled page

This figure illustrates styled page using basic css features.

Here I start adding settings for media uploading.

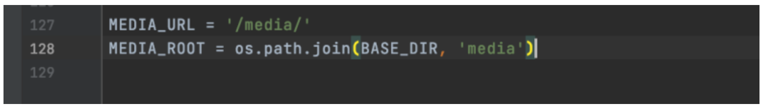

```python
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

Figure 3.10: Settings

MEDIA_URL is the URL to access media files. MEDIA_ROOT is the path to the directory where the uploaded files will be stored.

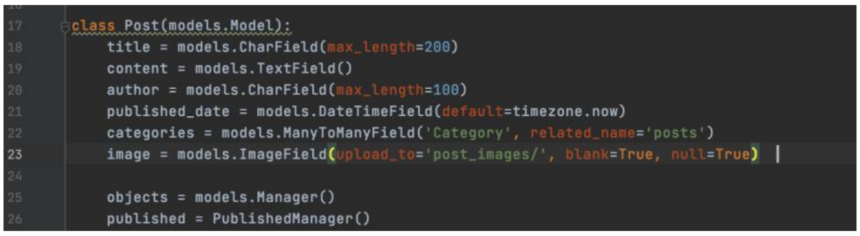
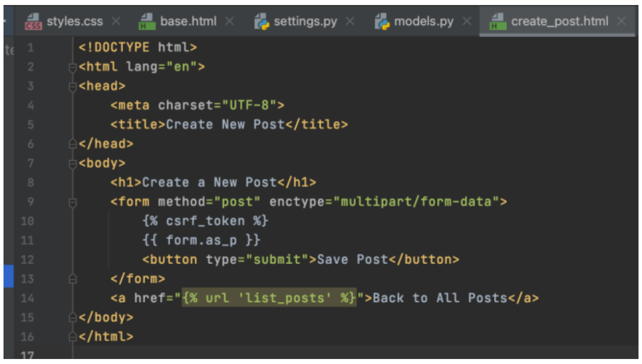

```python
class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    author = models.CharField(max_length=100)
    published_date = models.DateTimeField(default=timezone.now)
    categories = models.ManyToManyField('Category', related_name='posts')
    image = models.ImageField(upload_to='post_images/', blank=True, null=True)

    objects = models.Manager()
    published = PublishedManager()
```

Figure 3.11: New field for image

Here I added in figure 3.11 the field for image in Post model to make users to upload images for each post.



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Create New Post</title>
</head>
<body>
    <h1>Create a New Post</h1>
    <form method="post" enctype="multipart/form-data">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Save Post</button>
    </form>
    <a href="{% url 'list_posts' %}">Back to All Posts</a>
</body>
</html>
```

Figure 3.12: New attribute

Adding the feature to upload a file to the form by adding the enctype" multipart/form-data" attribute in figure 3.12.

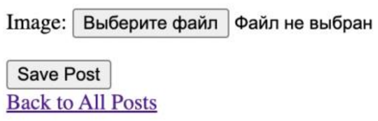


Figure 3.13: The upload button

Figure 3.13 shows updated template page to upload image for post.

CONCLUSION

This lab assignment provided practical experience in building Django application covering basic aspects of web development including models, views, templates, and forms. By implementing features like post creation, image uploads, and basic CSS styling, I improved the application's functionality and visuality.