KAZAKH-BRITISH TECHNICAL UNIVERSITY

Prepared by: Serzhan Yerasil

Assignment 3

Web Development

09.11.2024

Table of Contents

Introduction

In this project, we are building a simple blog using Django. Our goal is to create a site where you can publish articles, view them, and leave comments. We will work with a database to store information, use templates to design pages, and add some styles to make the blog look nice. This project will help you understand how to build web applications with Django, starting with the basics.

Exercise 1, 2, 3

In this exercise, we will create a simple model for blog posts. The model is called Post. This model will allow us to store posts with information about each one. I will also create a method that returns the post title as a string to make it easier to display the posts. We created a new model called Category so that we can add different categories to our posts. Categories will be related to posts through a many-to-many relationship, meaning that one post can have multiple categories, and one category can have many posts. In Exercise 3, I added a custom manager to our Post model to make it easier to find published posts and posts by a specific author.

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

    def __str__(self):
        return self.name


class PublishedManager(models.Manager):
    def published(self):
        return self.filter(published_date__isnull=False)

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



class Post(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    published_date = models.DateTimeField(blank=True, null=True)
    categories = models.ManyToManyField('Category', related_name='posts')

    objects = PublishedManager()

    def __str__(self):
        return self.title

class Comment(models.Model):
    post = models.ForeignKey(Post, on_delete=models.CASCADE, related_name="comments")
    content = models.TextField()
    author = models.ForeignKey(User, on_delete=models.CASCADE)
    created_date = models.DateTimeField(auto_now_add=True)

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

Pic1. First Fragment

The Category model is used to store categories that posts can be assigned to.
The name field is the name of the category, limited to 100 characters.
The __str__ method returns the category name as a string for easy display.
PublishedManager is a special manager that adds methods to make searching for posts easier.
The published() method returns only those posts that have a publication date set (that is, they are published).
The by_author(author) method returns posts written by a specific author (it must be passed to author). The Comment model stores comments on posts.
The post field indicates the post to which the comment belongs. Associated with the Post model. If a post is deleted, all its comments are deleted as well.
The content field is the text of the comment.
The author field indicates the user who wrote the comment.
The created_date field automatically saves the creation date of the comment when it is created (auto_now_add=True).

Exercise 4, 5, 6

In ex 4 I created a function that displays all blog posts. This function will get all the posts from the database and send them to the template, where they will be displayed as a list. In Exercise 5, I created the same pages, but using class based view instead of functions. Django gives special classes that make it easier to work with such pages. Created a class that will display a form on the page. If the user fills it out and submits it, the function will check the data and add the new post to the database. After successfully adding, the user will be redirected to the page with the list of posts.

```python
def post_list(request):
    posts = Post.objects.published()
    return render(request, 'blog/post_list.html', {'posts': posts})

def post_detail(request, pk):
    post = get_object_or_404(Post, pk=pk)
    return render(request, 'blog/post_detail.html', {'post': post})

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

    def get_queryset(self):
        return Post.objects.published()

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

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

The first function Gets all published posts from the database and passes them to the post_list.html template for display. It calls the published() method to select only those posts that have a publication date. It then uses render to display a template with those posts. The second function Finds and displays one specific post by its ID (pk). The second class specifies that the Post model should be used and that posts should be passed under the name posts. The get_queryset() method selects only published posts, as in the post_list function.

Exercise 7, 8, 9

In Exercise 7, create an HTML template to display a list of posts. The template will show the titles, authors, and publication dates of each post. Add a special template tag to make the publication date look nice. In Exercise 8, create a main template with common elements, such as a header and footer, for the site. This will help avoid duplicating code on different pages. Use the base template as a basis for other templates (for example, for a list of posts and a page with post details). These templates will add their own content, while maintaining a single style. In Exercise 9, create a CSS file that will set the styles for the pages. Include it in templates to change the appearance of text, colors, and the arrangement of elements.

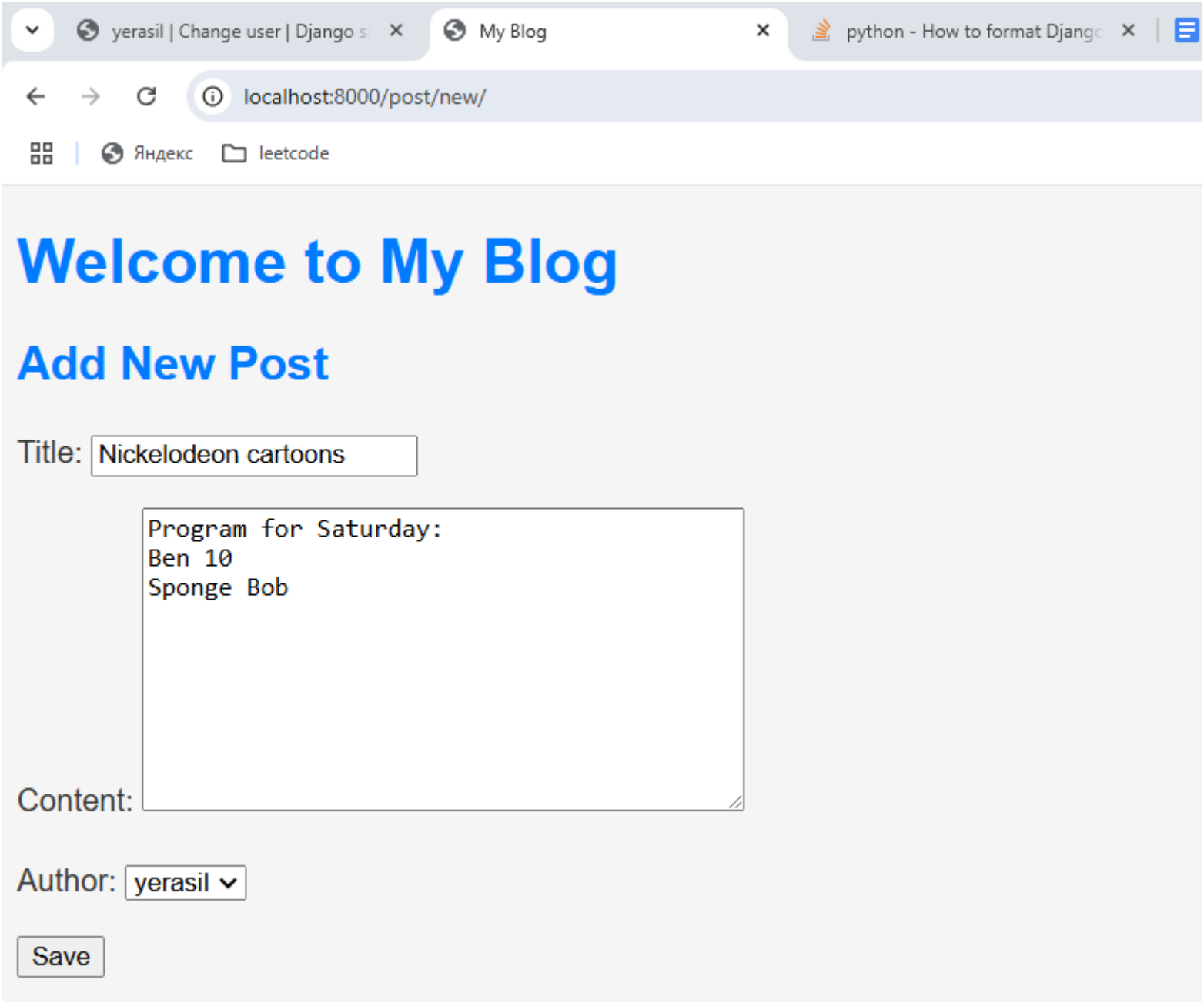
assignment3 > blog > static > blog > css > # style.css > ...

```css
body {
    font-family: Arial, sans-serif;
    background-color: #f5f5f5;
    color: #333;
}

h1, h2 {
    color: #007BFF;
}

p {
    line-height: 1.6;
}
```

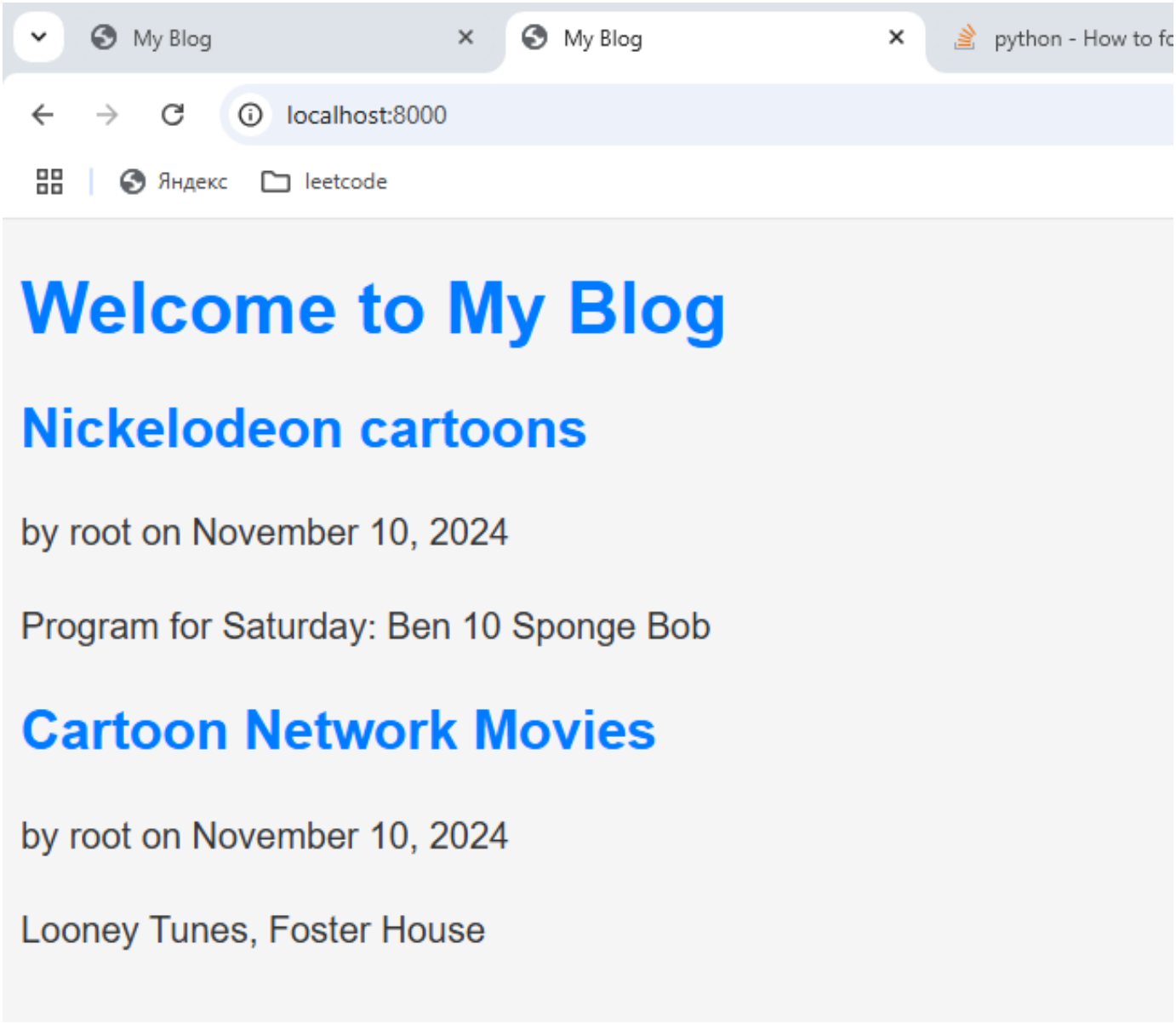
assignment3 > blog > templates > blog > post_list.html > ...

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

{% block content %}

{% for post in posts %}
    <h2>{{ post.title }}</h2>
    <p>by {{ post.author }} on {{ post.published_date|date:"F d, Y" }}</p>
    <p>{{ post.content|truncatewords:30 }}</p>
{% endfor %}

{% endblock %}
```

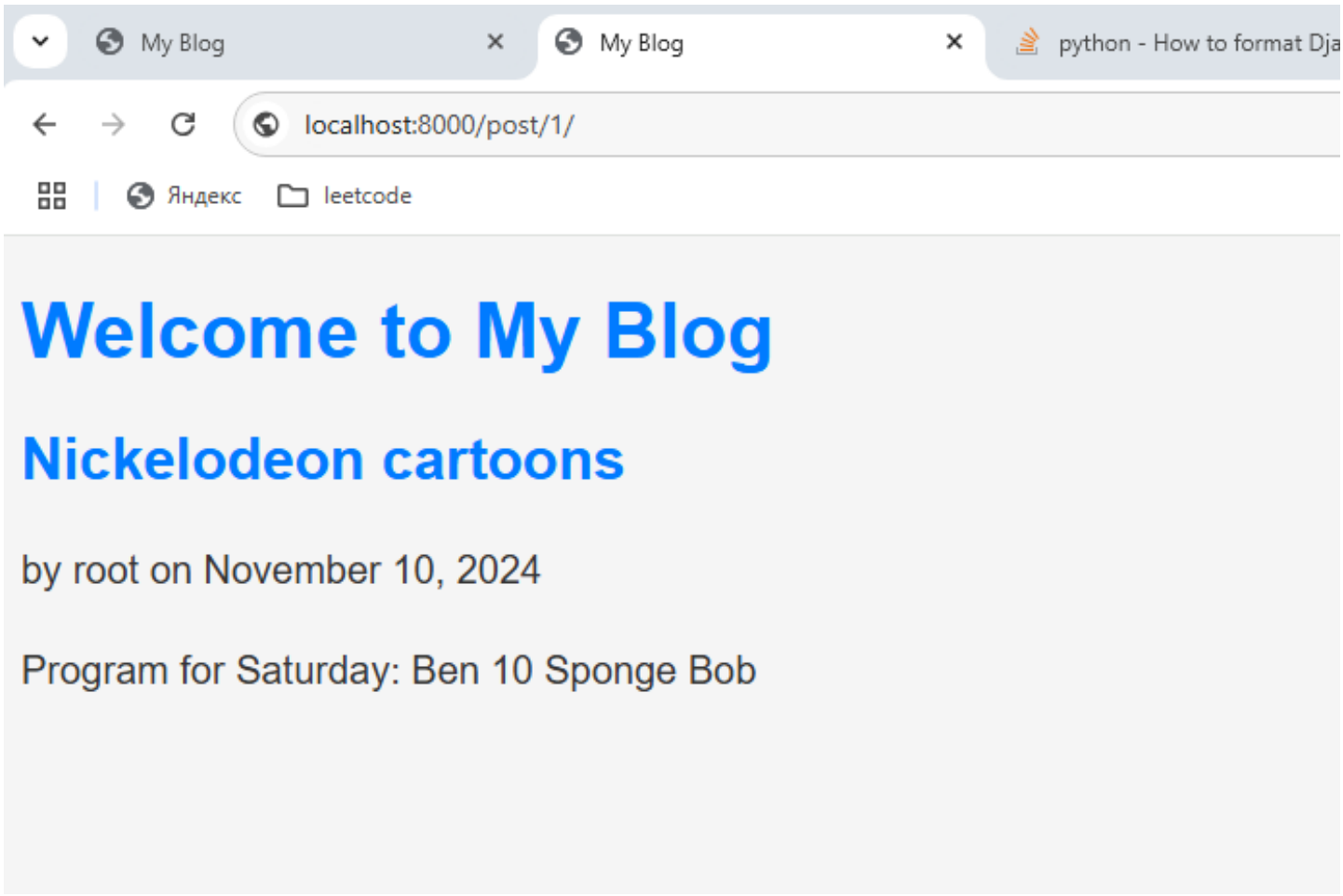
The current template uses base.html as a base. This means that it will add its content to the special blocks of the base template.
This template goes through each post, outputs its title, author, date, and the first 30 words of text, neatly inserting this information into the base template. This CSS code sets the styles for the elements on the web page. This code makes the page background light gray, sets Arial as the primary font, sets the color blue for the headings, and increases the space between lines in paragraphs.



Pic1. Post creation



Pic2. List of posts



Pic3. Post by id

Conclusion

In the end, we created a working blog on Django. Now we have the ability to add articles, view them, and leave comments. We also customized the appearance using CSS and added categories to organize posts. This project showed the basic principles of working with Django and gave basic skills for creating web applications. In the future, you can expand the functionality of the blog, adding, for example, user registration or search by articles.