

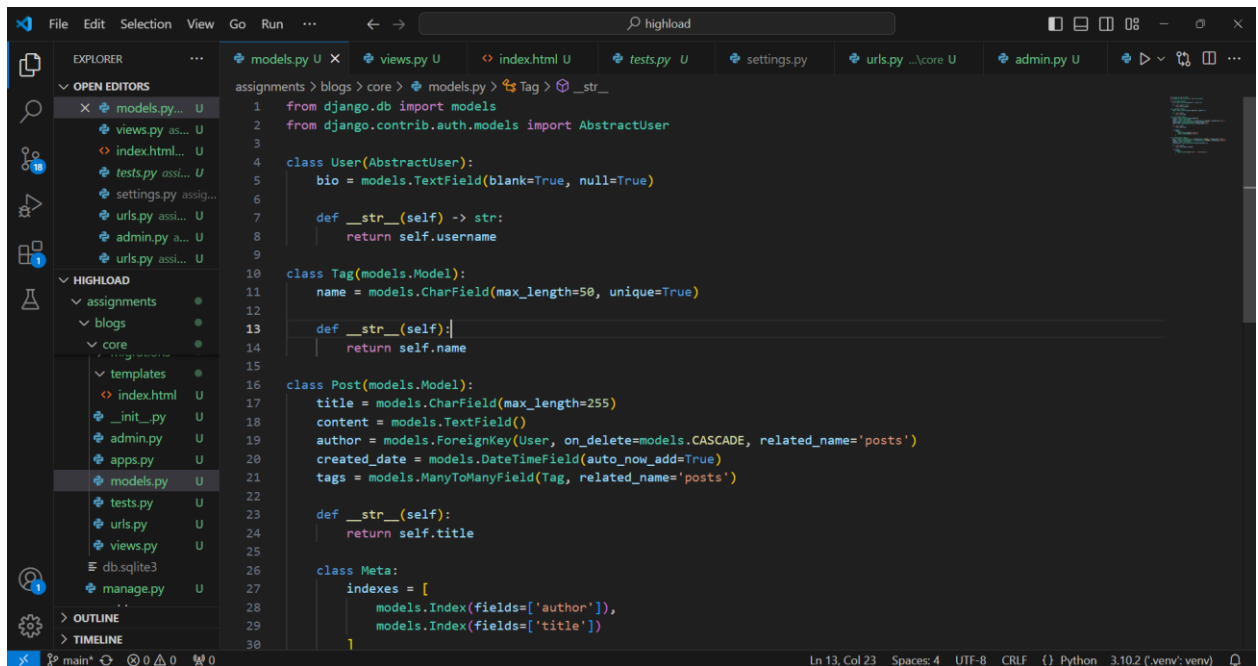
## Assignment 2, backend for high load

### Exercise 1: Database Design and Optimization

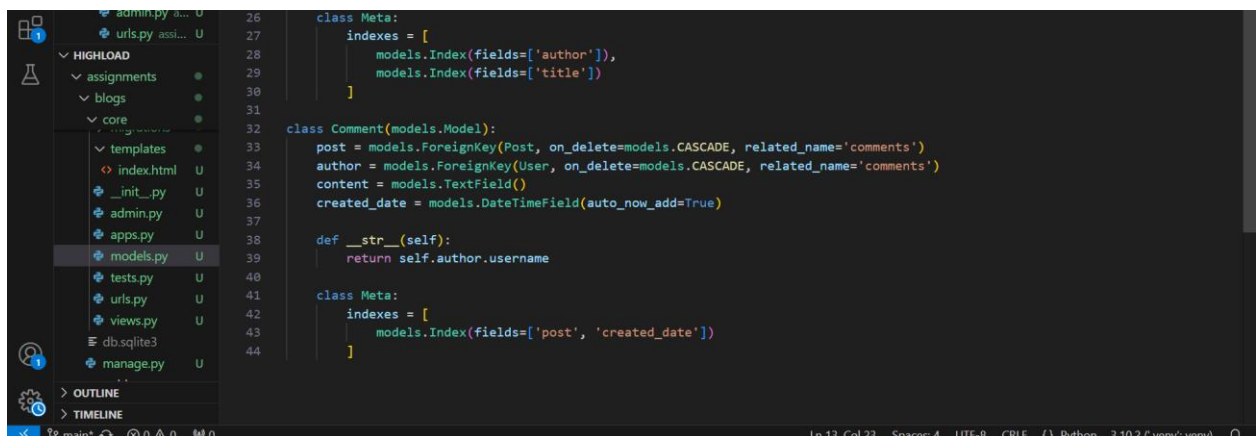
**Objective:** Design an efficient database schema and optimize queries in a Django application.

#### Task:

1. **Schema Design:** Create a Django model for a simple blog application with the following entities:
  - User: Username, Email, Password, Bio.
  - Post: Title, Content, Author (ForeignKey to User), Created Date, Tags (ManyToManyField).
  - Comment: Post (ForeignKey to Post), Author (ForeignKey to User), Content, Created Date.



```
1 from django.db import models
2 from django.contrib.auth.models import AbstractUser
3
4 class User(AbstractUser):
5     bio = models.TextField(blank=True, null=True)
6
7     def __str__(self) -> str:
8         return self.username
9
10 class Tag(models.Model):
11     name = models.CharField(max_length=50, unique=True)
12
13     def __str__(self):
14         return self.name
15
16 class Post(models.Model):
17     title = models.CharField(max_length=255)
18     content = models.TextField()
19     author = models.ForeignKey(User, on_delete=models.CASCADE, related_name='posts')
20     created_date = models.DateTimeField(auto_now_add=True)
21     tags = models.ManyToManyField(Tag, related_name='posts')
22
23     def __str__(self):
24         return self.title
25
26 class Meta:
27     indexes = [
28         models.Index(fields=['author']),
29         models.Index(fields=['title'])
30     ]
```



```
26 class Meta:
27     indexes = [
28         models.Index(fields=['author']),
29         models.Index(fields=['title'])
30     ]
31
32 class Comment(models.Model):
33     post = models.ForeignKey(Post, on_delete=models.CASCADE, related_name='comments')
34     author = models.ForeignKey(User, on_delete=models.CASCADE, related_name='comments')
35     content = models.TextField()
36     created_date = models.DateTimeField(auto_now_add=True)
37
38     def __str__(self):
39         return self.author.username
40
41 class Meta:
42     indexes = [
43         models.Index(fields=['post', 'created_date'])
44     ]
```

## 2. Indexing:

- Add indexes to the Post model to optimize query performance for filtering by Author and Tags.

```
class Post(models.Model):
    title = models.CharField(max_length=255)
    content = models.TextField()
    author = models.ForeignKey(User, on_delete=models.CASCADE, related_name='posts')
    created_date = models.DateTimeField(auto_now_add=True)
    tags = models.ManyToManyField(Tag, related_name='posts')

    def __str__(self):
        return self.title

    class Meta:
        indexes = [
            models.Index(fields=['author']),
            models.Index(fields=['title'])
        ]
```

- Add a composite index to the Comment model for Post and Created Date.

```
31
32 class Comment(models.Model):
33     post = models.ForeignKey(Post, on_delete=models.CASCADE, related_name='comments')
34     author = models.ForeignKey(User, on_delete=models.CASCADE, related_name='comments')
35     content = models.TextField()
36     created_date = models.DateTimeField(auto_now_add=True)
37
38     def __str__(self):
39         return self.author.username
40
41     class Meta:
42         indexes = [
43             models.Index(fields=['post', 'created_date'])
44         ]
```

## 3. Query Optimization:

- Write a Django ORM query to fetch all posts with their related comments in a single query.

```
assignments > blogs > core > views.py > post_list
1 from django.shortcuts import render
2 from .models import Post
3
4 def post_list(request):
5     posts = Post.objects.all().prefetch_related('comments__author', 'tags')
6     print(posts.query)
7     return render(request, 'index.html', {'posts':posts})
```

- Analyze the SQL generated by the Django ORM and suggest improvements if necessary.

```
SELECT "core_post"."id", "core_post"."title", "core_post"."content", "core_post"."author_id", "core_post"."created_date" FROM "core_post"
[09/Oct/2024 00:07:14] "GET / HTTP/1.1" 200 303
```

## 4. Optimization Report:

Impact of chosen indexes:

- Post model indexes: Accelerates queries filtering posts by a specific author, such as retrieving all posts by a user and enhances performance for search operations based on post titles.
- Comment model composite index: Optimizes retrieval of comments from specific post.

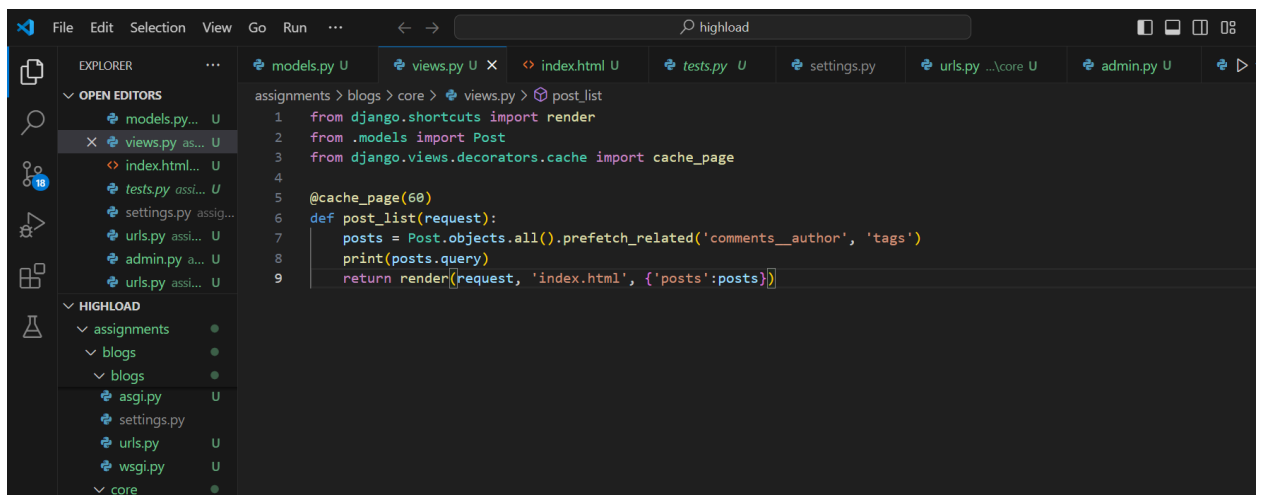
## Exercise 2: Caching Strategies

**Objective:** Implement caching to improve the performance of a Django application.

**Task:**

### 1. Basic Caching:

- Implement view-level caching for a page that displays a list of blog posts.
- Set the cache timeout to 60 seconds.



The screenshot shows a VS Code editor interface with a Django project. The Explorer panel on the left shows the project structure with folders for 'assignments', 'blogs', and 'core'. The 'blogs' folder is expanded, showing 'asgi.py', 'settings.py', 'urls.py', and 'wsgi.py'. The 'core' folder is also expanded, showing 'models.py', 'views.py', 'index.html', 'tests.py', 'settings.py', 'urls.py', and 'admin.py'. The 'views.py' file is open in the editor, showing the following code:

```
1 from django.shortcuts import render
2 from .models import Post
3 from django.views.decorators.cache import cache_page
4
5 @cache_page(60)
6 def post_list(request):
7     posts = Post.objects.all().prefetch_related('comments__author', 'tags')
8     print(posts.query)
9     return render(request, 'index.html', {'posts': posts})
```

Containers

Images

Volumes

Builds

Dev Environments BETA

Docker Scout

Extensions

Add Extensions

Containers [Give feedback](#)

Container CPU usage ⓘ  
0.34% / 200% (2 CPUs available)

Container memory usage ⓘ  
5.87MB / 943.91MB

Show charts

Search

Only show running containers

	Name	Image	Status	CPU (%)	Port(s)	Last started	Actions
<input type="checkbox"/>	<a href="#">redis</a>	<a href="#">redis</a>	Running	0.34%	<a href="#">6379:6379</a>	1 minute ago	<div><div></div><div></div><div></div></div>

Showing 1 item

Walkthroughs

Multi-container applications  
8 mins

Containerize your application  
3 mins

[View more in the Learning center](#)

Engine running

RAM 0.92 GB CPU 3.55% Not signed in

New version available

3

13 

def post\_detail(request, post\_id):

14 post = get\_object\_or\_404(Post, id=post\_id)

15 recent\_comments\_cache\_key = f'recent\_comments\_post\_{post.id}'

16

17 recent\_comments = cache.get(recent\_comments\_cache\_key)

18 

if not recent\_comments:

19 recent\_comments = Comment.objects.filter(post=post).select\_related('author').order\_by('-created\_date')[:10]

20 cache.set(recent\_comments\_cache\_key, recent\_comments, timeout=60)

21

22 

context = {

23 'post': post,

24 'recent\_comments': recent\_comments,

25 'recent\_comments\_cache\_key': recent\_comments\_cache\_key,

26 }

27 return render(request, 'post\_detail.html', context)

TERMINAL

PROBLEMS

DEBUG CONSOLE

OUTPUT

PORTS

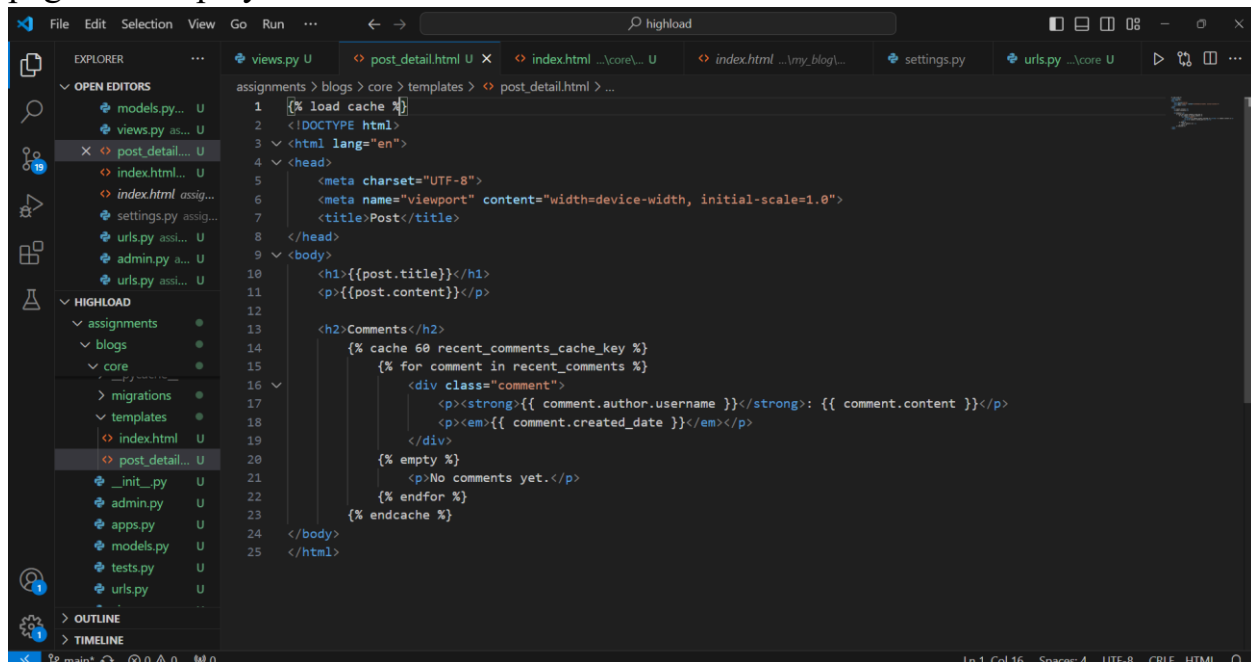
py - blogs

System check identified no issues (0 silenced).  
October 11, 2024 - 12:21:00  
Django version 5.1.1, using settings 'blogs.settings'  
Starting development server at http://127.0.0.1:8000/  
Quit the server with CTRL-BREAK.  
  
[11/Oct/2024 12:21:01] "GET /1 HTTP/1.1" 200 517

Ln 27, Col 56 Spaces: 4 UTF-8 CRLF {} Python 3.10.2 (v

## 2. Template Fragment Caching:

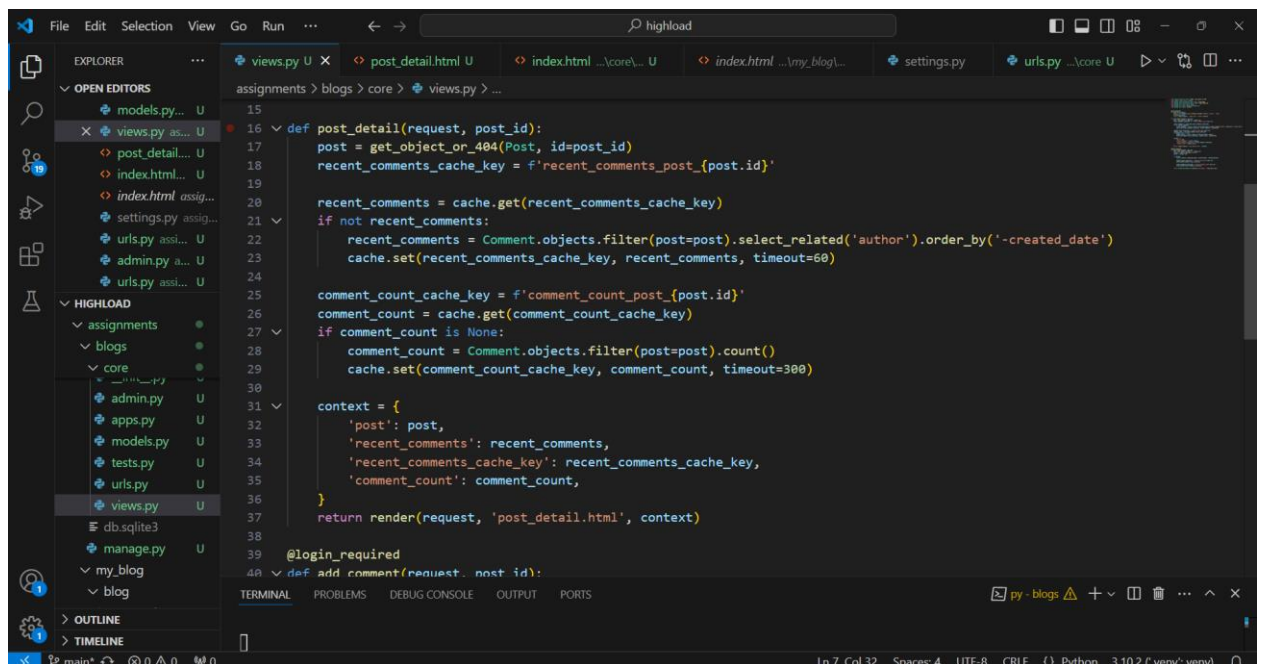
Implement template fragment caching for a section of the blog post detail page that displays the most recent comments



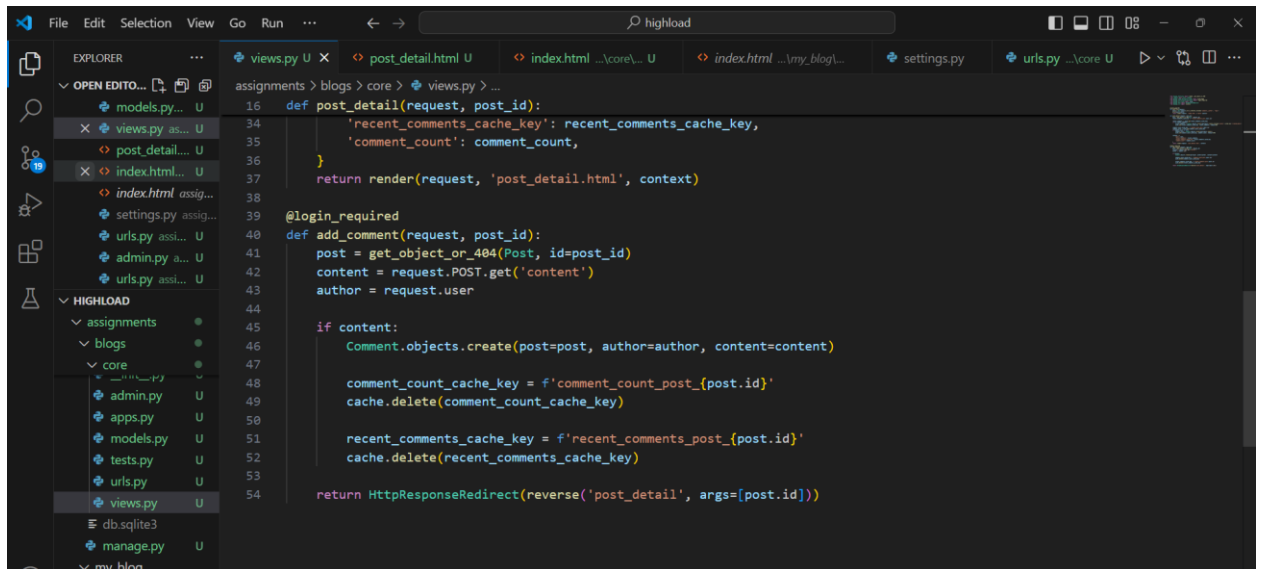
```
1 {% load cache %}
2 <!DOCTYPE html>
3 <html lang="en">
4 <head>
5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>Post</title>
8 </head>
9 <body>
10     <h1>{{post.title}}</h1>
11     <p>{{post.content}}</p>
12
13     <h2>Comments</h2>
14     {% cache 60 recent_comments_cache_key %}
15     {% for comment in recent_comments %}
16         <div class="comment">
17             <p><strong>{{ comment.author.username }}</strong>: {{ comment.content }}</p>
18             <p><em>{{ comment.created_date }}</em></p>
19         </div>
20     {% empty %}
21     <p>No comments yet.</p>
22     {% endfor %}
23     {% endcache %}
24 </body>
25 </html>
```

## 3. Low-Level Caching:

- Implement low-level caching using Django's cache framework to store the result of an expensive database query (e.g., counting the number of comments for a post).
- Set a timeout for the cache and handle cache invalidation when new comments are added.



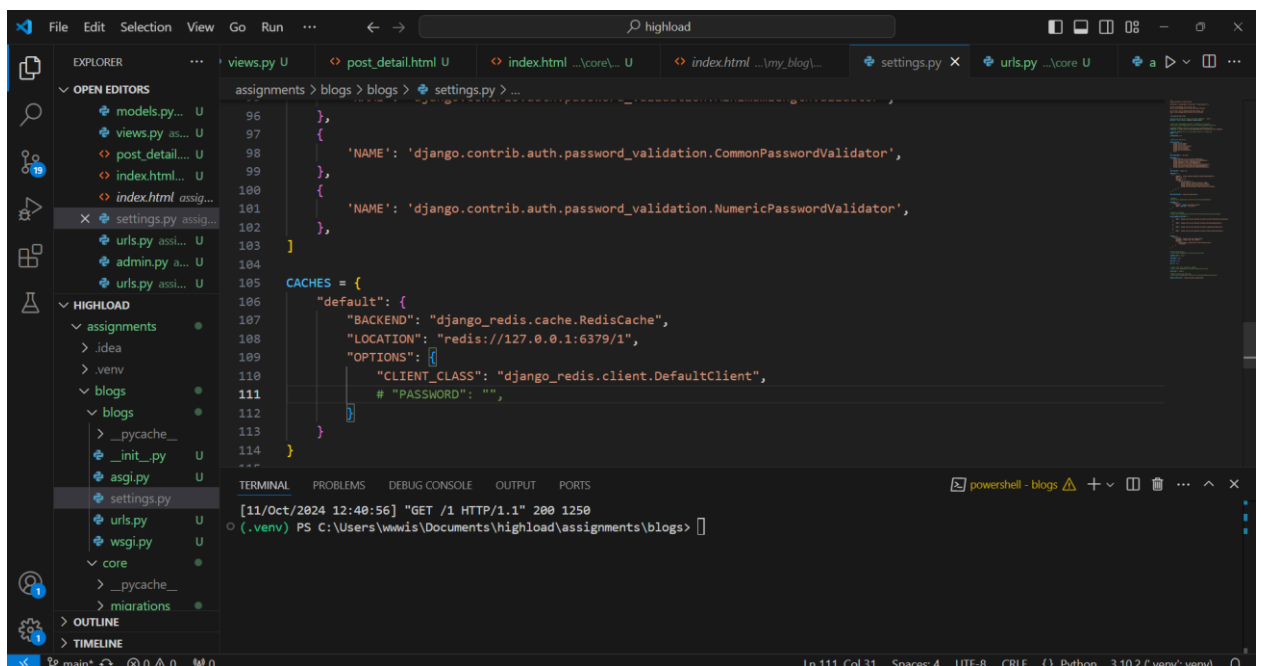
```
15
16 def post_detail(request, post_id):
17     post = get_object_or_404(Post, id=post_id)
18     recent_comments_cache_key = f'recent_comments_post_{post.id}'
19
20     recent_comments = cache.get(recent_comments_cache_key)
21     if not recent_comments:
22         recent_comments = Comment.objects.filter(post=post).select_related('author').order_by('-created_date')
23         cache.set(recent_comments_cache_key, recent_comments, timeout=60)
24
25     comment_count_cache_key = f'comment_count_post_{post.id}'
26     comment_count = cache.get(comment_count_cache_key)
27     if comment_count is None:
28         comment_count = Comment.objects.filter(post=post).count()
29         cache.set(comment_count_cache_key, comment_count, timeout=300)
30
31     context = {
32         'post': post,
33         'recent_comments': recent_comments,
34         'recent_comments_cache_key': recent_comments_cache_key,
35         'comment_count': comment_count,
36     }
37     return render(request, 'post_detail.html', context)
38
39 @login_required
40 def add_comment(request, post_id):
```



```
16 def post_detail(request, post_id):
34     'recent_comments_cache_key': recent_comments_cache_key,
35     'comment_count': comment_count,
36 }
37 return render(request, 'post_detail.html', context)
38
39 @login_required
40 def add_comment(request, post_id):
41     post = get_object_or_404(Post, id=post_id)
42     content = request.POST.get('content')
43     author = request.user
44
45     if content:
46         Comment.objects.create(post=post, author=author, content=content)
47
48         comment_count_cache_key = f'comment_count_post_{post.id}'
49         cache.delete(comment_count_cache_key)
50
51         recent_comments_cache_key = f'recent_comments_post_{post.id}'
52         cache.delete(recent_comments_cache_key)
53
54     return HttpResponseRedirect(reverse('post_detail', args=[post.id]))
```

#### 4. Cache Backend:

- Configure Django to use Redis as the cache backend.
- Implement a caching strategy that combines view-level, template fragment, and low-level caching.



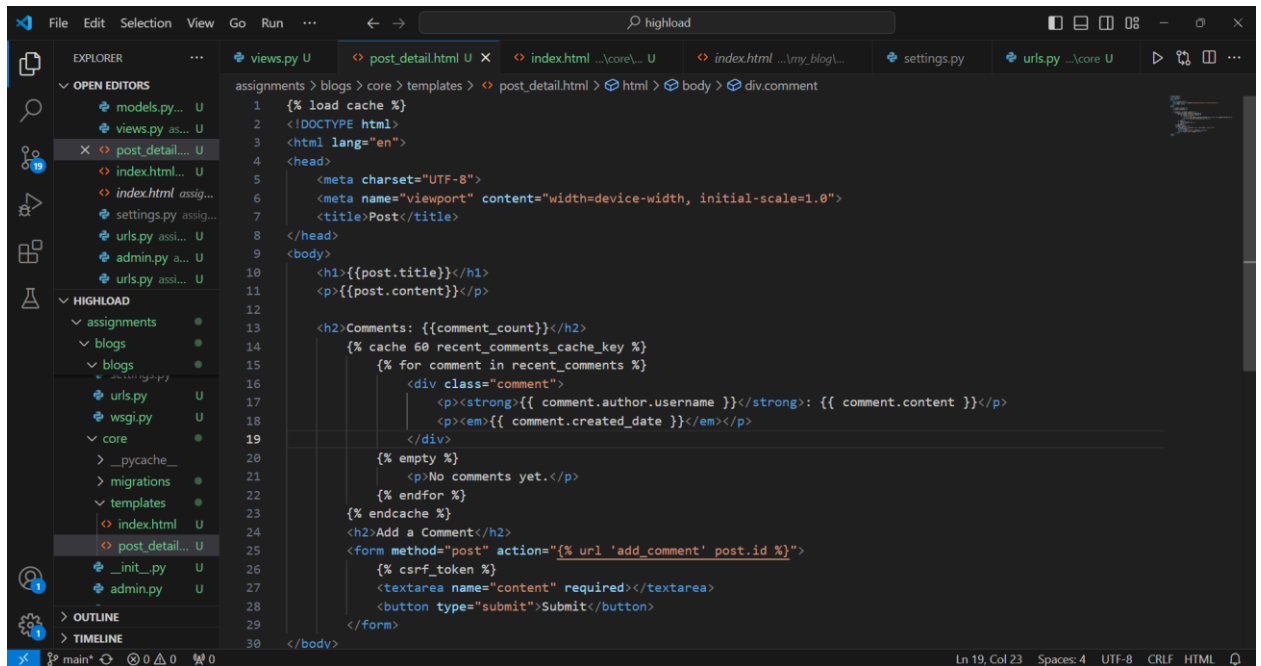
```
106 CACHES = {
107     "default": {
108         "BACKEND": "django_redis.cache.RedisCache",
109         "LOCATION": "redis://127.0.0.1:6379/1",
110         "OPTIONS": {
111             "CLIENT_CLASS": "django_redis.client.DefaultClient",
112             # "PASSWORD": "",
113         }
114     }
115 }
```

Terminal: [11/Oct/2024 12:40:56] "GET /1 HTTP/1.1" 200 1250

```
9
10 @cache_page(60)
11 def post_list(request):
12     posts = Post.objects.all().prefetch_related('comments__author', 'tags')
13     print(posts.query)
14     return render(request, 'index.html', {'posts': posts})
15
16 def post_detail(request, post_id):
17     post = get_object_or_404(Post, id=post_id)
18     recent_comments_cache_key = f'recent_comments_post_{post.id}'
19
20     recent_comments = cache.get(recent_comments_cache_key)
21     if not recent_comments:
22         recent_comments = Comment.objects.filter(post=post).select_related('author').order_by('-created_date')
23         cache.set(recent_comments_cache_key, recent_comments, timeout=60)
24
25     comment_count_cache_key = f'comment_count_post_{post.id}'
26     comment_count = cache.get(comment_count_cache_key)
27     if comment_count is None:
28         comment_count = Comment.objects.filter(post=post).count()
29         cache.set(comment_count_cache_key, comment_count, timeout=300)
30
31     context = {
32         'post': post,
33         'recent_comments': recent_comments,
34         'recent_comments_cache_key': recent_comments_cache_key,
35         'comment_count': comment_count,
36     }
37     return render(request, 'post_detail.html', context)
38
```

```
35 def post_detail(request, post_id):
36     'comment_count': comment_count,
37     return render(request, 'post_detail.html', context)
38
39 @login_required
40 def add_comment(request, post_id):
41     post = get_object_or_404(Post, id=post_id)
42     content = request.POST.get('content')
43     author = request.user
44
45     if content:
46         Comment.objects.create(post=post, author=author, content=content)
47
48         comment_count_cache_key = f'comment_count_post_{post.id}'
49         cache.delete(comment_count_cache_key)
50
51         recent_comments_cache_key = f'recent_comments_post_{post.id}'
52         cache.delete(recent_comments_cache_key)
53
54     return HttpResponseRedirect(reverse('post_detail', args=[post.id]))
```





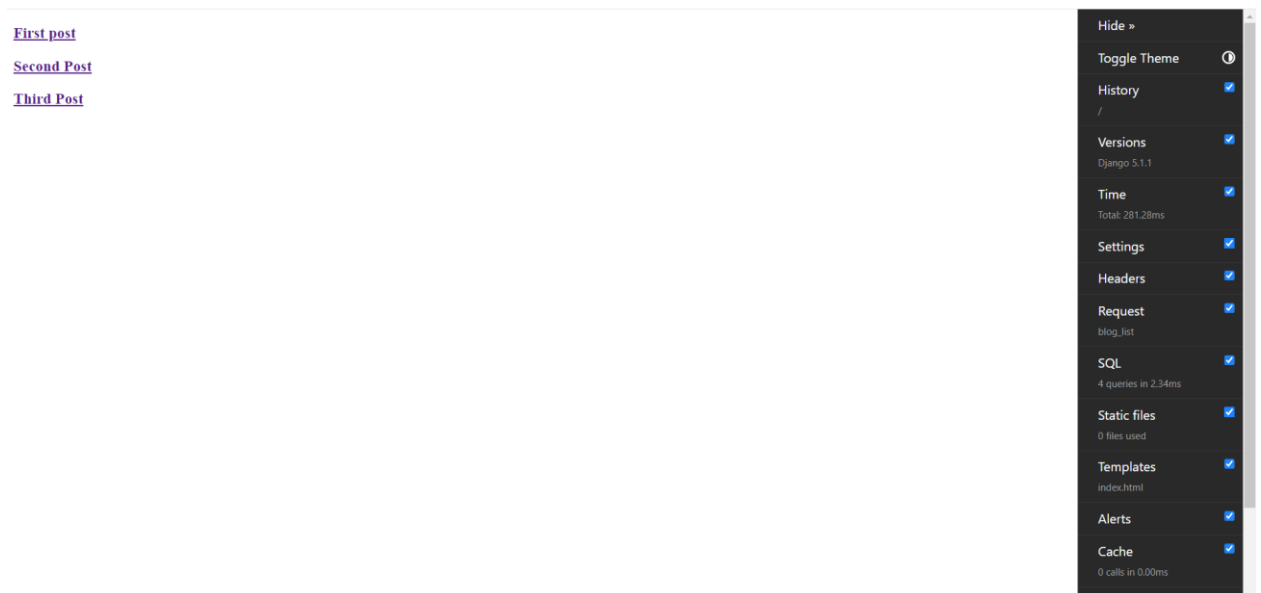
## 5. Performance Analysis:

- Measure the performance of the application before and after implementing caching.
- Write a report comparing the load times and resource usage.

To measure the performance of our application we can use Django Debug Toolbar.

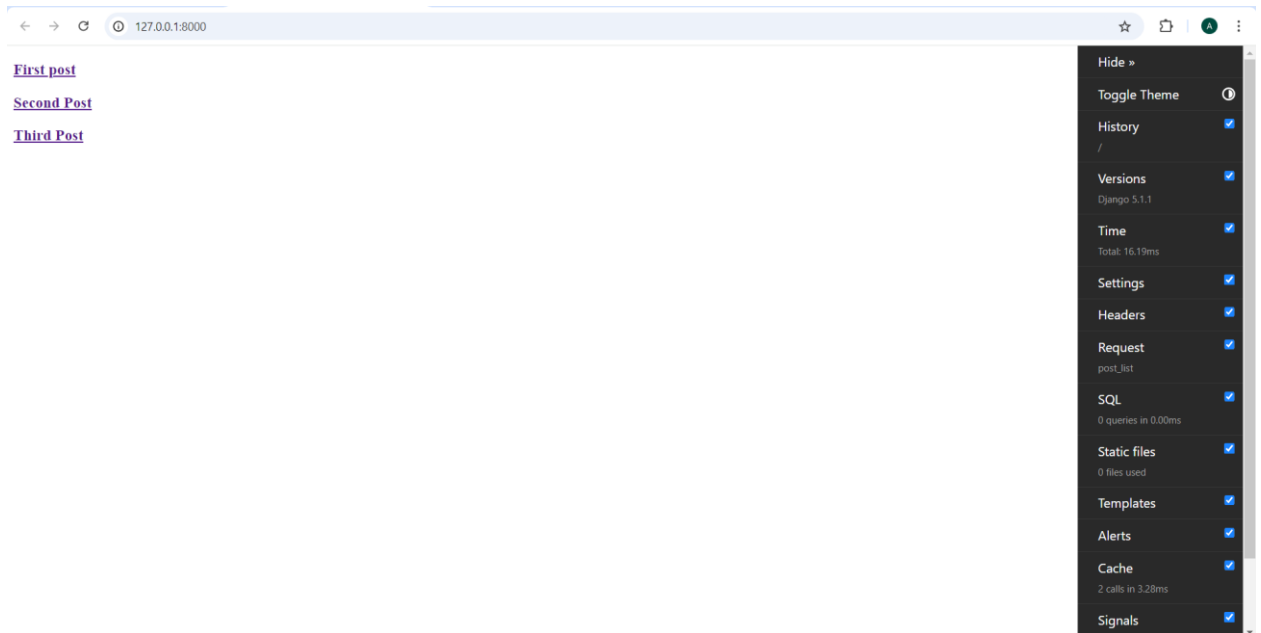
Example of usage:

Before caching:



After caching:





Metric	Before Caching	After Caching	Improvement
Page Load Time	281.28ms	16.19ms	faster
Number of SQL Queries	4	0-4	reduction
Cache Calls	N/A	2	Caching implemented

Metric	Before Caching	After Caching	Improvement
Post detail load time	307.69ms	29.87ms	faster
Number of SQL Queries	2	1	reduction
Cache Calls	N/A	3	Caching implemented

Overall performance of our app improved.