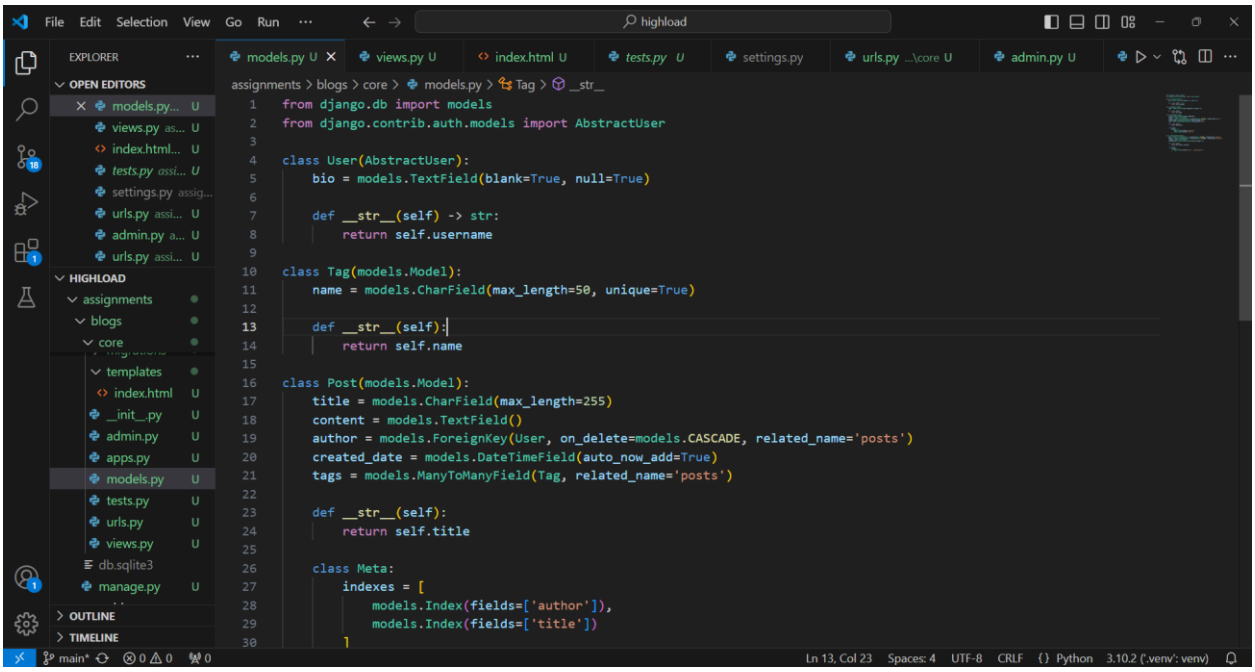
Assignment 2, backend for high load

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.



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

```python
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.

```python

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

    def __str__(self):
        return self.author.username

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

3. **Query Optimization:**
    - Write a Django ORM query to fetch all posts with their related comments in a single query.

```python
from django.shortcuts import render
from .models import Post

def post_list(request):
    posts = Post.objects.all().prefetch_related('comments__author', 'tags')
    print(posts.query)
    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:

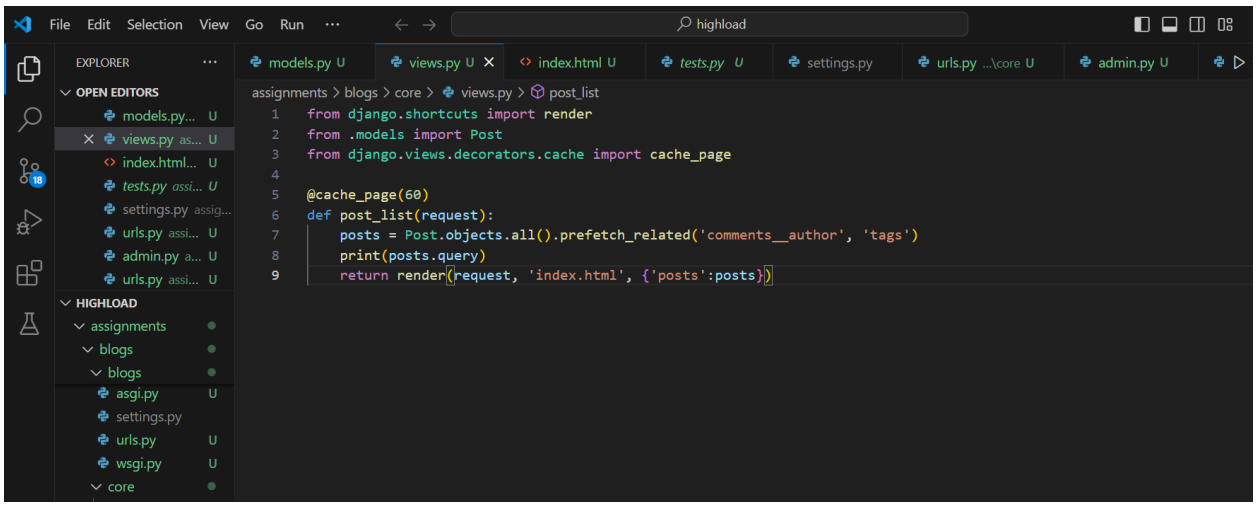
- Post model indexes: Accelerates queries filtering posts by a specific author, such as retrieving all posts by a user and enhances performance for search operations based on post titles.
- Comment model composite index: Optimizes retrieval of comments from specific post.

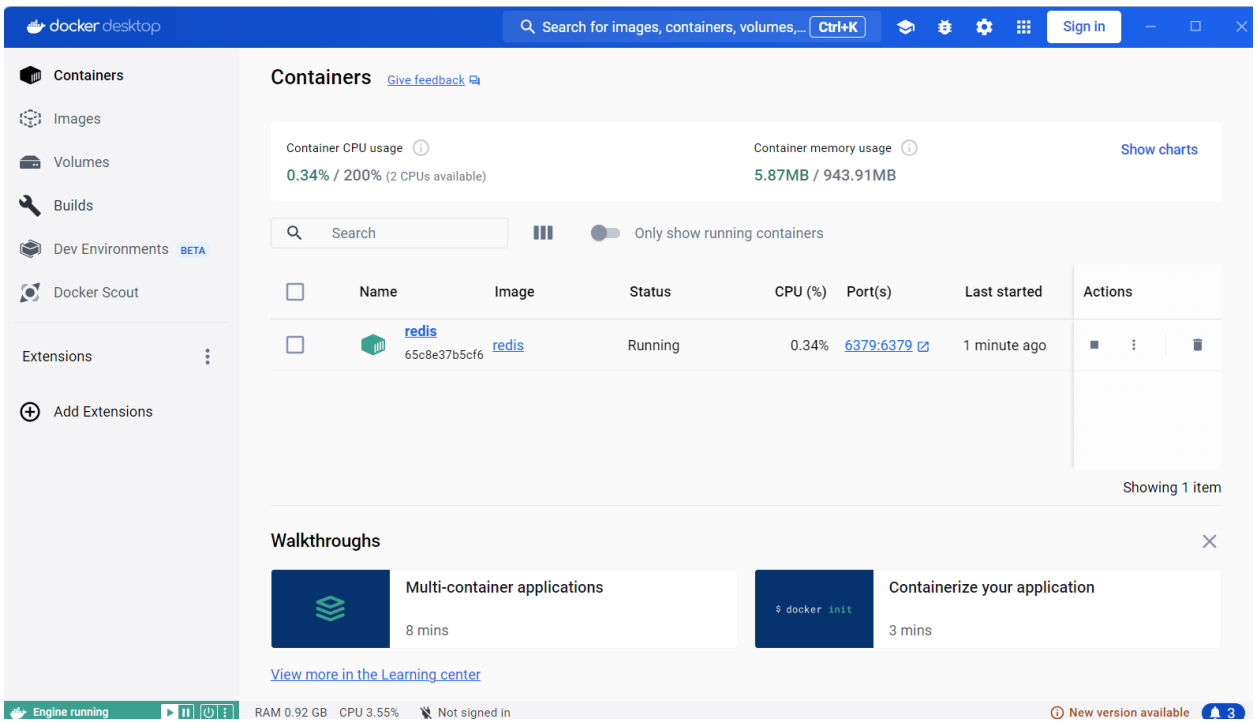
Exercise 2: Caching Strategies

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.






```python
def post_detail(request, post_id):
    post = get_object_or_404(Post, id=post_id)
    recent_comments_cache_key = f'recent_comments_post_{post.id}'

    recent_comments = cache.get(recent_comments_cache_key)
    if not recent_comments:
        recent_comments = Comment.objects.filter(post=post).select_related('author').order_by('-created_date')[:10]
        cache.set(recent_comments_cache_key, recent_comments, timeout=60)

    context = {
        'post': post,
        'recent_comments': recent_comments,
        'recent_comments_cache_key': recent_comments_cache_key,
    }
    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 ('.ve

2. **Template Fragment Caching:**

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

```html
{% load cache %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Post</title>
</head>
<body>
    <h1>{{post.title}}</h1>
    <p>{{post.content}}</p>

    <h2>Comments</h2>
        {% cache 60 recent_comments_cache_key %}
            {% for comment in recent_comments %}
                <div class="comment">
                    <p><strong>{{ comment.author.username }}</strong>: {{ comment.content }}</p>
                    <p><em>{{ comment.created_date }}</em></p>
                </div>
            {% empty %}
                <p>No comments yet.</p>
            {% endfor %}
        {% endcache %}
</body>
</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.

```python
def post_detail(request, post_id):
    post = get_object_or_404(Post, id=post_id)
    recent_comments_cache_key = f'recent_comments_post_{post.id}'

    recent_comments = cache.get(recent_comments_cache_key)
    if not recent_comments:
        recent_comments = Comment.objects.filter(post=post).select_related('author').order_by('-created_date')
        cache.set(recent_comments_cache_key, recent_comments, timeout=60)

    comment_count_cache_key = f'comment_count_post_{post.id}'
    comment_count = cache.get(comment_count_cache_key)
    if comment_count is None:
        comment_count = Comment.objects.filter(post=post).count()
        cache.set(comment_count_cache_key, comment_count, timeout=300)

    context = {
        'post': post,
        'recent_comments': recent_comments,
        'recent_comments_cache_key': recent_comments_cache_key,
        'comment_count': comment_count,
    }
    return render(request, 'post_detail.html', context)

@login_required
def add_comment(request, post_id):
```

```python
def post_detail(request, post_id):
        'recent_comments_cache_key': recent_comments_cache_key,
        'comment_count': comment_count,
    }
    return render(request, 'post_detail.html', context)

@login_required
def add_comment(request, post_id):
    post = get_object_or_404(Post, id=post_id)
    content = request.POST.get('content')
    author = request.user

    if content:
        Comment.objects.create(post=post, author=author, content=content)

        comment_count_cache_key = f'comment_count_post_{post.id}'
        cache.delete(comment_count_cache_key)

        recent_comments_cache_key = f'recent_comments_post_{post.id}'
        cache.delete(recent_comments_cache_key)

    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.

```python
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/1",
        "OPTIONS": {
            "CLIENT_CLASS": "django_redis.client.DefaultClient",
            # "PASSWORD": "",
        }
    }
}
```

```
[11/Oct/2024 12:40:56] "GET /1 HTTP/1.1" 200 1250
(.venv) PS C:\Users\wwwis\Documents\highload\assignments\blogs>
```

```python
@cache_page(60)
def post_list(request):
    posts = Post.objects.all().prefetch_related('comments__author', 'tags')
    print(posts.query)
    return render(request, 'index.html', {'posts':posts})

def post_detail(request, post_id):
    post = get_object_or_404(Post, id=post_id)
    recent_comments_cache_key = f'recent_comments_post_{post.id}'

    recent_comments = cache.get(recent_comments_cache_key)
    if not recent_comments:
        recent_comments = Comment.objects.filter(post=post).select_related('author').order_by('-created_date')
        cache.set(recent_comments_cache_key, recent_comments, timeout=60)

    comment_count_cache_key = f'comment_count_post_{post.id}'
    comment_count = cache.get(comment_count_cache_key)
    if comment_count is None:
        comment_count = Comment.objects.filter(post=post).count()
        cache.set(comment_count_cache_key, comment_count, timeout=300)

    context = {
        'post': post,
        'recent_comments': recent_comments,
        'recent_comments_cache_key': recent_comments_cache_key,
        'comment_count': comment_count,
    }
    return render(request, 'post_detail.html', context)

```

```python
def post_detail(request, post_id):
        'comment_count': comment_count,
    }
    return render(request, 'post_detail.html', context)

@login_required
def add_comment(request, post_id):
    post = get_object_or_404(Post, id=post_id)
    content = request.POST.get('content')
    author = request.user

    if content:
        Comment.objects.create(post=post, author=author, content=content)

        comment_count_cache_key = f'comment_count_post_{post.id}'
        cache.delete(comment_count_cache_key)

        recent_comments_cache_key = f'recent_comments_post_{post.id}'
        cache.delete(recent_comments_cache_key)

    return HttpResponseRedirect(reverse('post_detail', args=[post.id]))
```

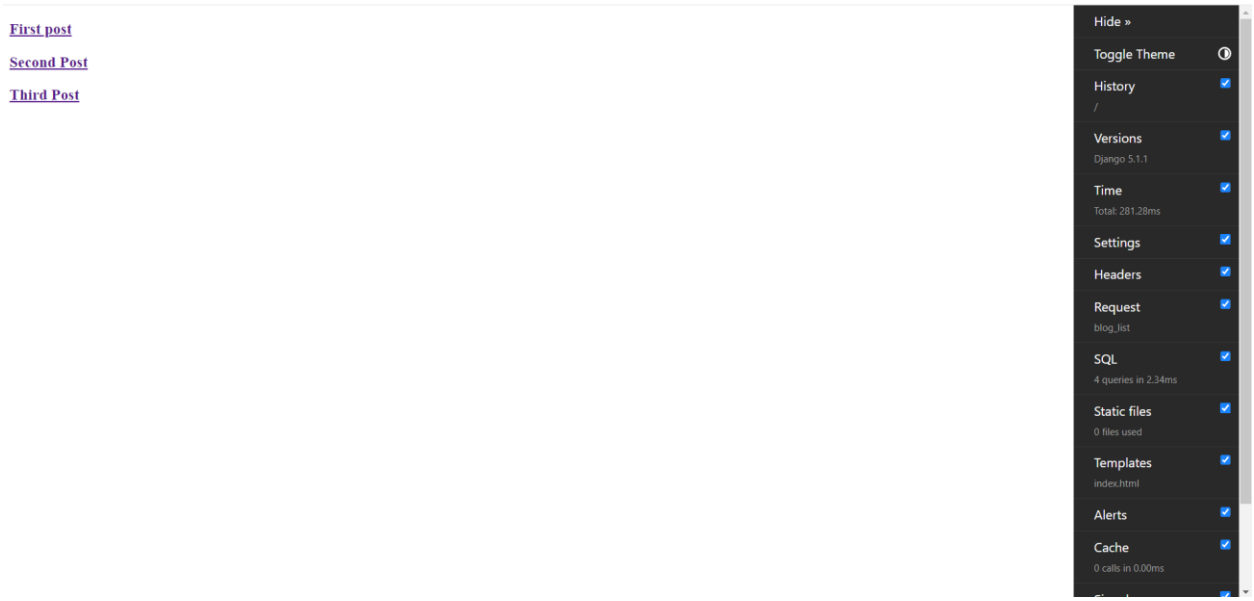
```html
{% load cache %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Post</title>
</head>
<body>
    <h1>{{post.title}}</h1>
    <p>{{post.content}}</p>

    <h2>Comments: {{comment_count}}</h2>
        {% cache 60 recent_comments_cache_key %}
            {% for comment in recent_comments %}
                <div class="comment">
                    <p><strong>{{ comment.author.username }}</strong>: {{ comment.content }}</p>
                    <p><em>{{ comment.created_date }}</em></p>
                </div>
            {% empty %}
                <p>No comments yet.</p>
            {% endfor %}
        {% endcache %}
        <h2>Add a Comment</h2>
        <form method="post" action="{% url 'add_comment' post.id %}">
            {% csrf_token %}
            <textarea name="content" required></textarea>
            <button type="submit">Submit</button>
        </form>
</body>
```

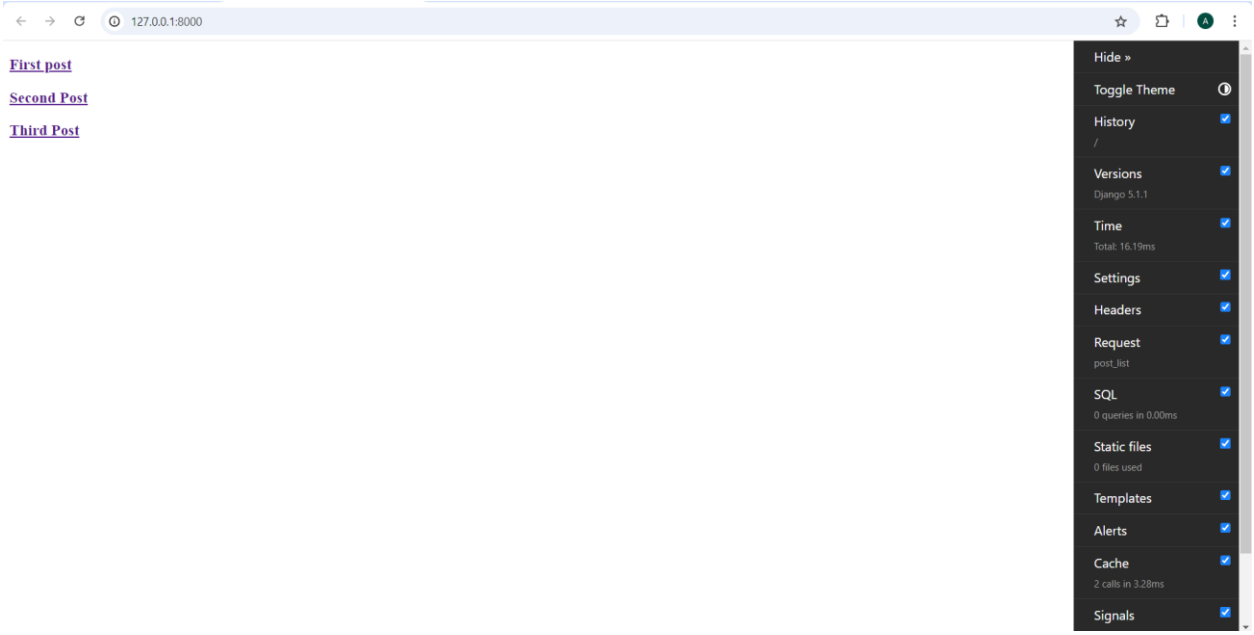
5. **Performance Analysis:**
    - Measure the performance of the application before and after implementing caching.
    - Write a report comparing the load times and resource usage.

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.