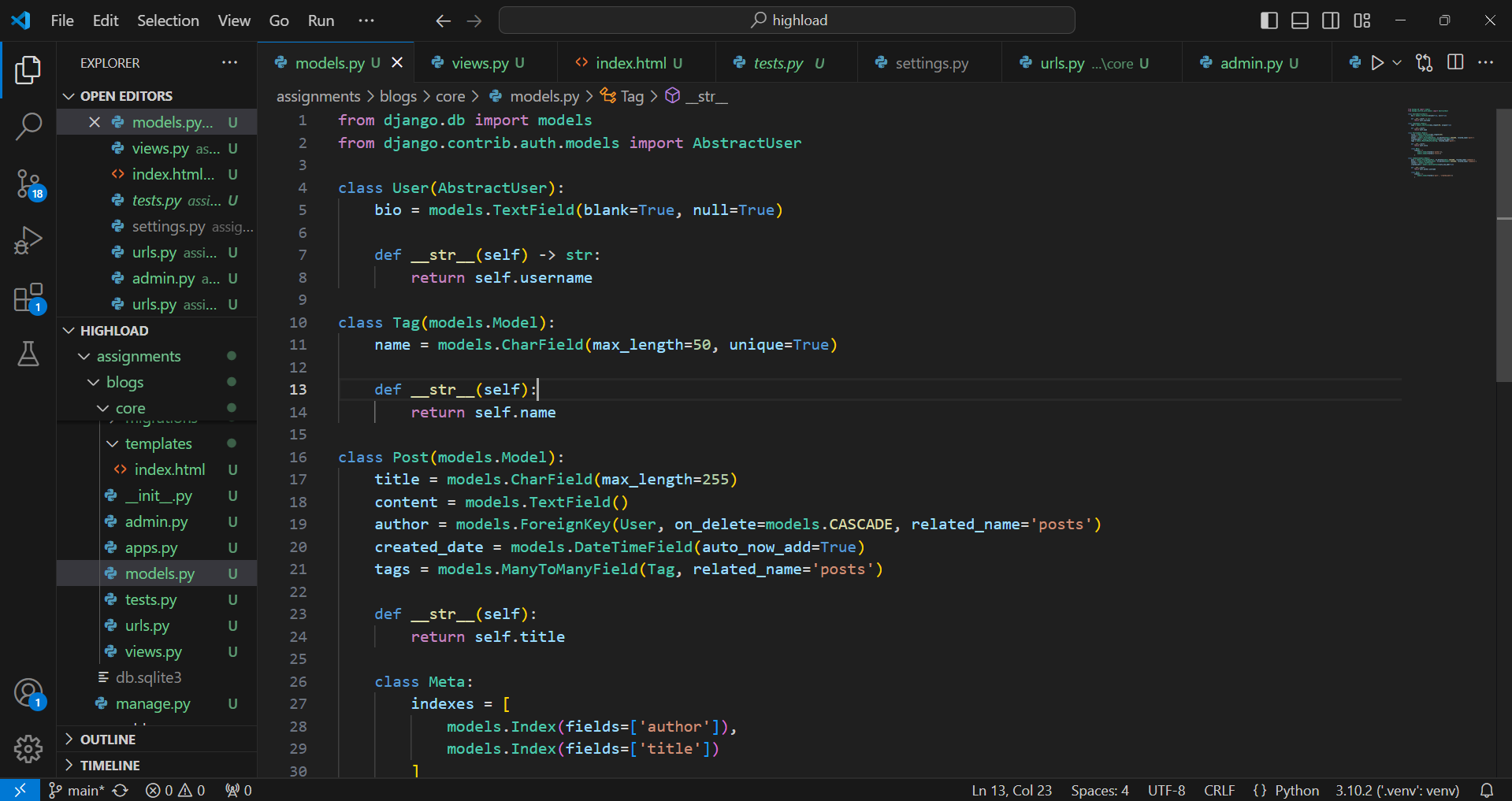
**Assignment 2, backend for high load**

**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.



Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

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



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

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

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

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

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



1. **Optimization Report:**

Impact of chosen indexes:

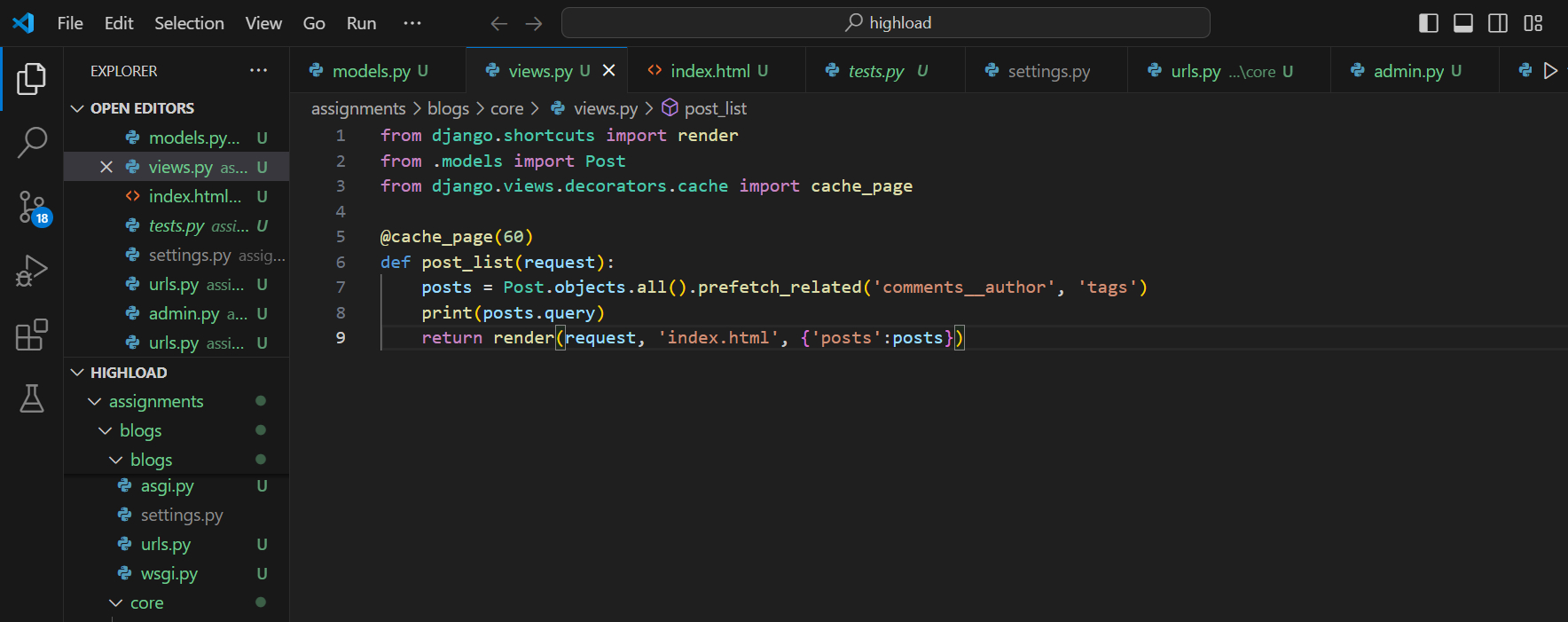
* Post model indexes: Accelerates queries filtering posts by a specific author, such as retrieving all posts by a user and enhances performance for search operations based on post titles.
* Comment model composite index: Optimizes retrieval of comments from specific post.

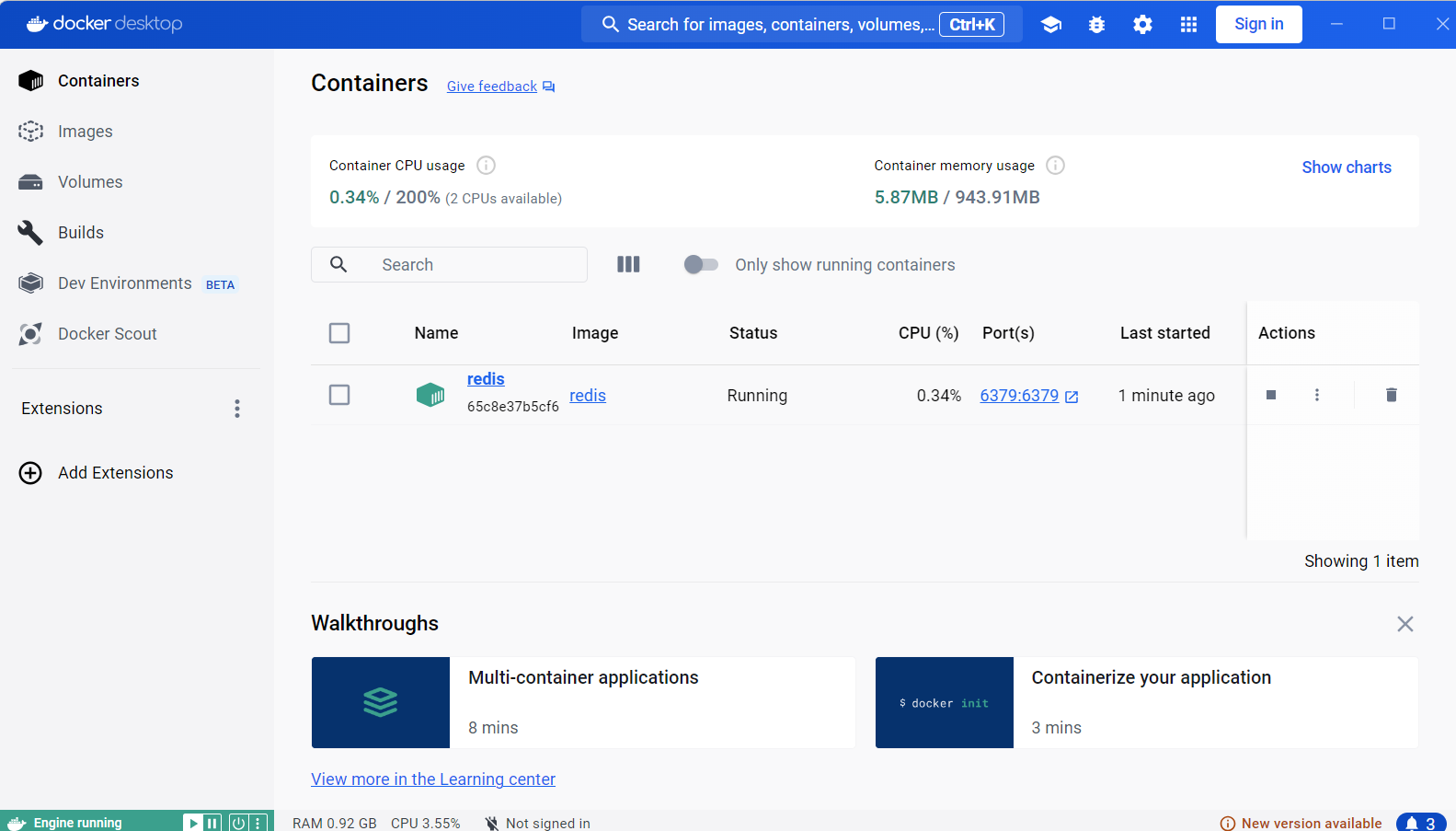
**Exercise 2: Caching Strategies**

**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.





Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

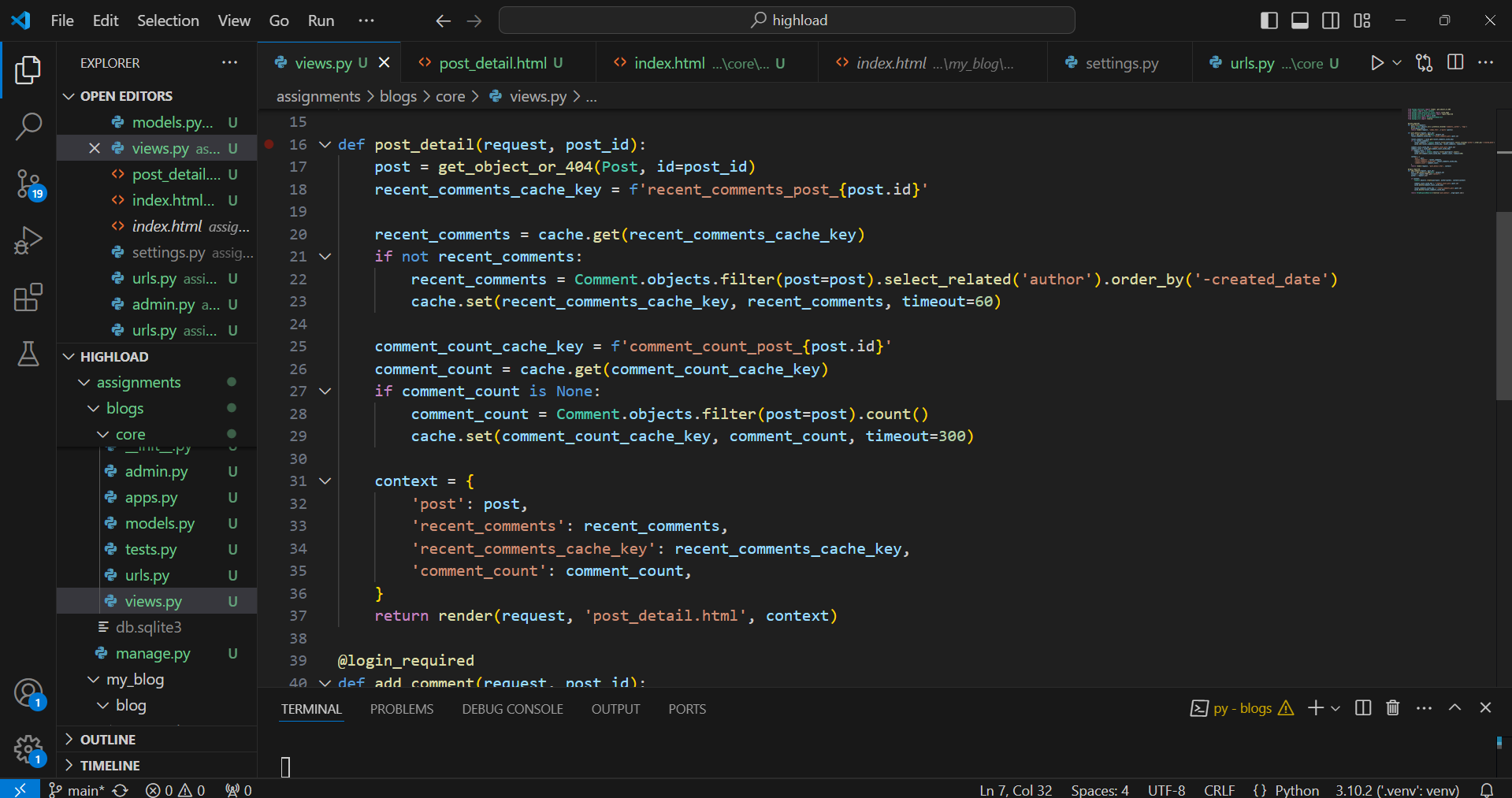
Автоматически созданное описание

1. **Template Fragment Caching:**

Implement template fragment caching for a section of the blog post detail page that displays the most recent commentsИзображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

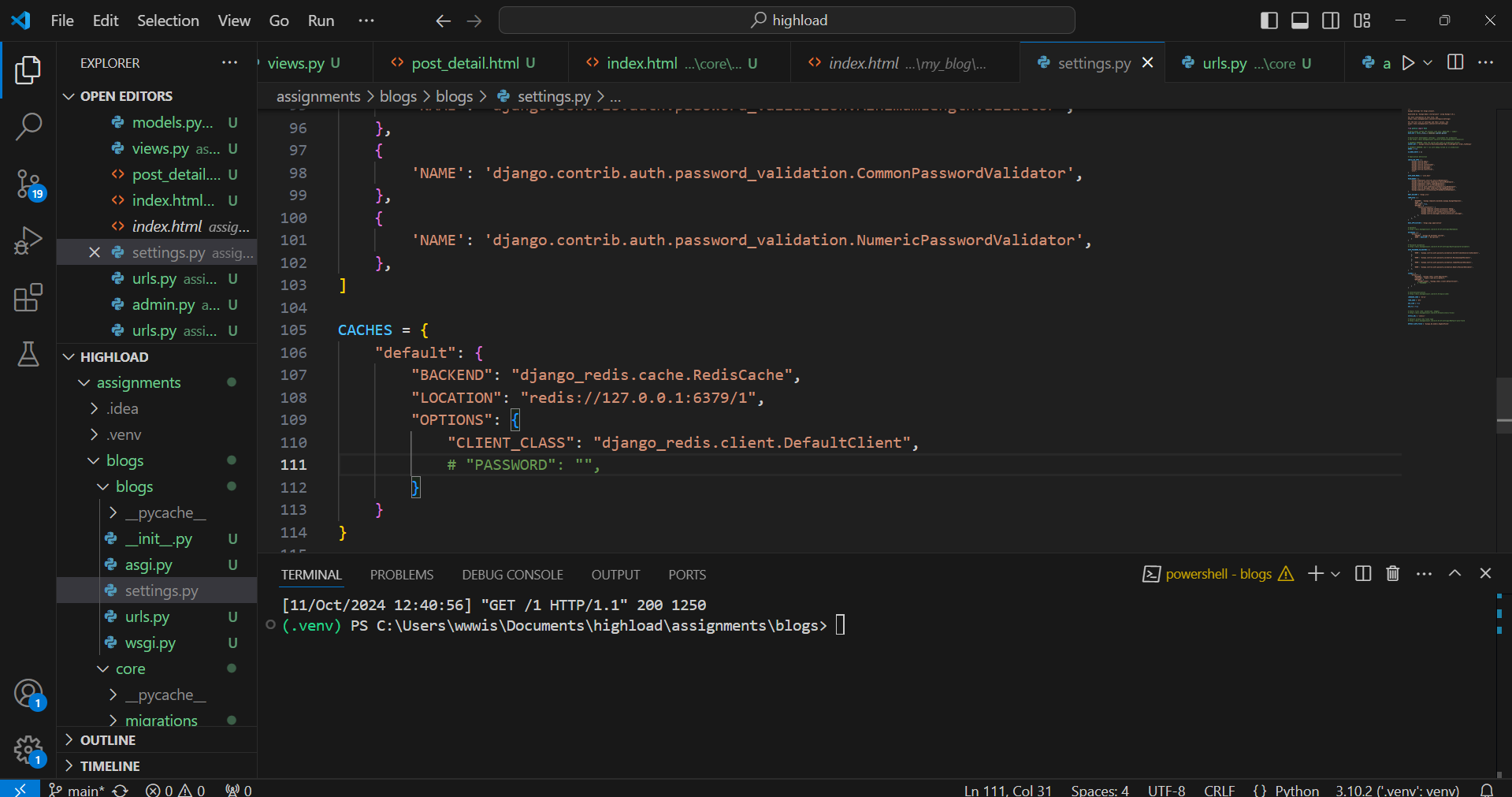
1. **Low-Level Caching:**
   * Implement low-level caching using Django's cache framework to store the result of an expensive database query (e.g., counting the number of comments for a post).
   * Set a timeout for the cache and handle cache invalidation when new comments are added.



Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

1. **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.



Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

1. **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:

Изображение выглядит как снимок экрана, программное обеспечение

Автоматически созданное описание

After caching:

Изображение выглядит как снимок экрана, программное обеспечение, текст, Мультимедийное программное обеспечение

Автоматически созданное описание

| **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.

**Exercise 3: Load Balancing Techniques**

**Objective:** Implement load balancing in a Django application to distribute traffic and ensure high availability.

**Task:**

1. **Set Up a Basic Load Balancer:**
   * Set up a load balancer (e.g., using NGINX) to distribute traffic between two Django application servers.
   * Configure round-robin load balancing.
2. **Session Management:**
   * Implement sticky sessions to ensure that a user's session data is consistent across requests routed to different servers.
3. **Health Checks:**
   * Configure the load balancer to perform health checks on the Django application servers.
   * Ensure that if one server goes down, the load balancer redirects traffic to the healthy server.
4. **Scaling:**
   * Simulate a traffic surge by generating concurrent requests to the application.
   * Monitor the load distribution and server performance.
5. **Report:**
   * Analyze the effectiveness of the load balancer in distributing traffic.
   * Discuss any challenges faced in setting up the load balancer and how they were resolved.