Assignment 2, backend for high load

Put all deliverables into github repository in your profile. Defend by explaining deliverables and answering questions.
Deliverables: code, report (pdf)
Google form:
https://docs.google.com/forms/d/e/1FAIpQLSe0GyNdOYIvM1tX_I_CtlPod5jBf-ACLGdHYZq1gVZbUeBzIg/viewform?usp=sf_link

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`.
    - Add a composite index to the `Comment` model for `Post` and `Created Date`.
3. **Query Optimization:**
    - Write a Django ORM query to fetch all posts with their related comments in a single query.
    - Analyze the SQL generated by the Django ORM and suggest improvements if necessary.
4. **Optimization Report:**
    - Explain how the chosen indexes improve query performance.
    - Suggest additional optimization techniques, such as denormalization or using select_related/prefetch_related.

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.
2. **Template Fragment Caching:**
    - Implement template fragment caching for a section of the blog post detail page that displays the most recent comments.
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.
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.
5. **Performance Analysis:**
    - Measure the performance of the application before and after implementing caching.
    - Write a report comparing the load times and resource usage.

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. **Scaling:**
    - Simulate a traffic surge by generating concurrent requests to the application.
    - Monitor the load distribution and server performance.
4. **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.