Final project, backend for high load

It is INDIVIDUAL work, make sure to submit report pdf and .zip of project in time according to deadline in teams, and be ready to defend during scheduled final time.

Write report in pdf and code in .zip.
Attach to google form according to teams deadline.
Google form:
https://docs.google.com/forms/d/e/1FAIpQLSeLLtzrCXURO2Qf9iNIyT3JjYGeK4m_ClukxhAv0eEtl7vlHw/viewform?usp=sf_link
Be prepared for defense.

High-Load E-Commerce Web Platform Project

Project Overview

The goal of this project is to build a scalable, high-performance e-commerce web platform capable of handling high user traffic while ensuring a smooth user experience. This will involve backend development, database optimization, caching, load balancing, security, and continuous deployment.

Table Entities

1. **User**
   - Fields: `id`, `username`, `email`, `password`, `first_name`, `last_name`, `created_at`, `updated_at`
2. **Product**
   - Fields: `id`, `name`, `description`, `price`, `stock_quantity`, `category_id`, `created_at`, `updated_at`
3. **Category**
   - Fields: `id`, `name`, `parent_id`, `created_at`, `updated_at`
4. **Order**
   - Fields: `id`, `user_id`, `order_status`, `total_amount`, `created_at`, `updated_at`
5. **OrderItem**
   - Fields: `id`, `order_id`, `product_id`, `quantity`, `price`, `created_at`, `updated_at`

6. **ShoppingCart**
    - Fields: `id`, `user_id`, `created_at`, `updated_at`
7. **CartItem**
    - Fields: `id`, `cart_id`, `product_id`, `quantity`, `created_at`, `updated_at`
8. **Payment**
    - Fields: `id`, `order_id`, `payment_method`, `amount`, `status`, `created_at`, `updated_at`
9. **Review**
    - Fields: `id`, `product_id`, `user_id`, `rating`, `comment`, `created_at`, `updated_at`
10. **Wishlist**
    - Fields: `id`, `user_id`, `created_at`, `updated_at`
11. **WishlistItem**
    - Fields: `id`, `wishlist_id`, `product_id`, `created_at`

Project Structure

1. Introduction to High-Load Systems

- **Tasks:**
    - Research and compile characteristics of high-load systems.
    - Document principles of high-load system design.

2. Backend Fundamentals

- **Tasks:**
    - Set up the Django project structure and environment.
    - Implement RESTful API endpoints for each model:

3. Database Design and Optimization

- **Tasks:**
    - Design and implement a normalized database schema in PostgreSQL.
    - Optimize the database through indexing and query optimization.
- **Deliverables:**
    - Migrations files in the `migrations` directory and a document outlining optimization strategies (`docs/db_optimization.md`).

4. Caching Strategies

- **Tasks:**
    - Set up Redis or Memcached for caching.

    - Implement caching for the following:
        - `GET /api/products/` – Cache product listings.
        - `GET /api/products/<id>/` – Cache individual product details.
- **Deliverables:**
    - Caching logic implemented in API views.

5. Load Balancing Techniques

- **Tasks:**
    - Research and choose a load balancer (Nginx or HAProxy).
    - Configure the load balancer to distribute traffic among application servers.
    - Document load balancing configuration and strategies.
- **Deliverables:**
    - Nginx configuration files in the repository (`nginx.conf`).

6. Distributed Systems and Data Consistency

- **Tasks:**
    - Explore distributed database options (e.g., DynamoDB).
    - Implement data replication and consistency models.
- **Deliverables:**
    - Code snippets and configurations documenting distributed architecture in `docs/distributed_systems.md`.

7. Scaling Backend Systems

- **Tasks:**
    - Identify opportunities for sharding or transitioning to microservices.
    - Plan and document implementation of microservices for key features, such as product management and order processing.
- **Deliverables:**
    - Architecture diagrams and code examples in `docs/scaling.md`.

8. Monitoring and Observability

- **Tasks:**
    - Set up monitoring tools (Prometheus and Grafana).
    - Implement structured logging for application monitoring.
    - Create performance dashboards to visualize API response times.
- **Deliverables:**
    - Configuration files for Prometheus and Grafana, and example logging implementation in `settings.py`.

9. Performance Tuning and Optimization

- **Tasks:**
  - Conduct load testing using tools like Apache JMeter.
  - Analyze bottlenecks in the following API endpoints:
    - `GET /api/products/`
    - `GET /api/orders/`
  - Implement optimizations based on analysis.
- **Deliverables:**
  - Load testing scripts and performance reports in `docs/performance_tuning.md`.

10. Message Queues and Asynchronous Processing

- **Tasks:**
  - Set up a message broker (RabbitMQ or Celery).
  - Implement background processing for tasks such as:
    - Sending order confirmation emails.
    - Processing payments.
- **Deliverables:**
  - Celery tasks defined in `tasks.py` and example usage documented in `docs/asynchronous_processing.md`.

11. Security in High-Load Systems

- **Tasks:**
  - Conduct a comprehensive security audit.
  - Implement security measures including:
    - `POST /api/auth/login/` – Secure user login.
    - `POST /api/auth/register/` – Secure user registration.
- **Deliverables:**
  - Security middleware in `middleware.py` and example implementations documented in `docs/security.md`.

12. Fault Tolerance and Resilience

- **Tasks:**
  - Design and implement redundancy in critical components (e.g., database replicas).
  - Create a disaster recovery plan that includes backup strategies for user data and orders.
- **Deliverables:**
  - Documentation of fault tolerance strategies in `docs/fault_tolerance.md`.

13. Testing and Continuous Integration/Continuous Deployment (CI/CD)

- **Tasks:**
  - Set up a CI/CD pipeline with automated testing and deployment.
  - Write unit tests and integration tests for key components, including:
    - User authentication.
    - Order processing.
  - Automate deployment processes for staging and production.
- **Deliverables:**
  - CI/CD configuration files (e.g., GitHub Actions workflow) in the repository and tests in `tests/`.

Project Report Structure

1. Title Page

- **Project Title**
- **Date**
- **Institution/Organization Name**

2. Executive Summary

- Brief overview of the project's goals, significance, and outcomes.
- Highlight key findings and recommendations.

3. Table of Contents

- List all sections and subsections with corresponding page numbers.

4. Introduction

- **Background**: Describe the context of the project and the problem it aims to solve.
- **Objectives**: Clearly state the project goals and objectives.
- **Scope**: Define what is included and excluded from the project.

5. Project Structure

- Outline the architecture of the system, including:
  - High-level architecture diagram.
  - Description of key components (e.g., frontend, backend, database).

6. Table Entities

- List and describe all table entities used in the database:
  - Name of the entity.
  - Attributes and their data types.
  - Relationships between entities.

7. Development Process

- **Technologies Used**: List all technologies, frameworks, and tools.
- **Implementation**:
  - Describe how each component was implemented.
  - Include sample code snippets for critical functionalities (APIs, models, etc.).

8. API Endpoints

- List all implemented API endpoints:
    - Method (GET, POST, etc.)
    - Endpoint URL
    - Description of functionality
    - Example requests and responses.

9. Database Design and Optimization

- Explain the database schema.
- Discuss indexing, normalization, and any optimization techniques used.

10. Caching Strategies

- Describe the caching mechanisms implemented.
- Explain how caching improves performance.

11. Load Balancing

- Detail the load balancing strategies used.
- Include configuration examples.

12. Security Measures

- Outline the security measures implemented.
- Discuss vulnerabilities addressed and their solutions.

13. Monitoring and Performance

- Describe monitoring tools and metrics tracked.
- Present performance analysis and optimization efforts.

14. Challenges and Solutions

- Discuss any significant challenges faced during the project.
- Explain how these challenges were addressed.

15. Conclusion

- Summarize key findings.
- Discuss future work and improvements.

16. References

- List all sources cited in the report, formatted in a consistent citation style (e.g., APA, MLA).

17. Appendices

- Include additional documentation, diagrams, or supporting material that is relevant but not essential to the main report.

Rules for Writing the Report

1. **Clarity and Precision**:
   - Use clear, concise language.
   - Avoid jargon unless it is defined.
2. **Structure and Flow**:
   - Follow the outlined structure to maintain a logical flow.
   - Use headings and subheadings for clarity.
3. **Formatting**:
   - Use consistent font styles and sizes (e.g., 12-point Times New Roman).
   - Maintain consistent spacing and margins (1-inch margins, double-spaced).
4. **Visuals**:
   - Use diagrams, charts, and tables where necessary to illustrate concepts.
   - Ensure visuals are labeled and referenced in the text.
5. **Code Formatting**:
   - Use code blocks for all code snippets.
   - Comment code appropriately to explain its purpose.
6. **Citations**:
   - Cite all external sources and references.
   - Use a consistent citation style throughout the report.
7. **Proofreading**:
   - Check for spelling, grammar, and punctuation errors.
   - Ensure technical accuracy and correctness of all information presented.
8. **Collaborative Contributions**:
   - If multiple team members contributed, indicate who was responsible for each section.
   - Ensure the report reflects a cohesive team effort.
9. **Length**:
   - Aim for a concise report; typically 15-25 pages, depending on project complexity.
10. **Submission Guidelines**:

    - Adhere to any specific submission requirements provided by your institution (file format, electronic submission, etc.).