Highload Backend Assignment 4 Report

Prepared by:

Bogdatov Chingis

ID: 21B030792

Prepared for:

Highload Backend class

Date:

November 16 , 2024

Executive Sumary

This report documents the implementation of three key tasks for securing and optimizing a Django application: asynchronous

processing with Celery and Redis and implementing robust security measures in high-load systems.

Key Findings:

1. **Asynchronous Processing**: Implemented Celery with Redis as the message broker to handle background tasks such as email sending. The system achieved improved performance and user experience through offloaded tasks.
2. **Security Measures**: Enhanced user authentication with two-factor authentication (2FA), implemented secure API endpoints, and applied field-level encryption for sensitive data. Rate limiting and input validation ensured robustness under high traffic.

Recommendations:

- Use monitoring tools like Flower and load-testing frameworks to continuously evaluate system performance.
- Regularly audit and update security protocols to counter evolving threats.

Table of Contents

Introduction 3

Background 3

Purpose and Scope 3

Outline 3

Methodology 3

Asynchronous Processing 3

Security Measures 4

Findings/Results 4
Task 1: Asynchronous Processing 4
Task 2: Security Measures 4
Discussion/Analysis 5
Conclusions 5

Introduction

Background

Modern web applications demand both performance and security. Efficient handling of background tasks and resilience under high traffic are critical for meeting user expectations and ensuring data integrity.

Purpose and Scope

This report focuses on implementing:

1. **Asynchronous Processing** using Celery and Redis for task delegation and system optimization.
2. **Security Measures** to fortify the application against threats while ensuring high performance during peak loads.

Outline

The report details methodologies, results, and practical recommendations for each task, along with insights into the tools and technologies used.

Methodology

Asynchronous Processing

- **Tools Used**: Celery, Redis, Django Email Backend.

- **Steps**:
  - Configured Redis as the Celery broker.
  - Created a Celery task for sending emails.
  - Integrated the task into a Django view.
  - Monitored task execution using Flower.

Security Measures

- **Tools Used**: Django REST Framework (DRF), django-otp, Locust.
- **Steps**:
  - Configured Django's built-in authentication system.
  - Added 2FA with django-otp and TOTP-based authentication.
  - Created secure API endpoints with custom permissions and token-based authentication.
  - Simulated high traffic using Locust for load testing.

Findings/Results

Task 1: Asynchronous Processing

- Successfully implemented Celery tasks for sending emails asynchronously.
- The Redis broker efficiently managed task queues, reducing API response time for users.
- **Placeholder for Flower Monitoring Screenshot**

Task 2: Security Measures

- **User Authentication**:
  - Implemented secure login and registration with hashed passwords and 2FA.
  - **Placeholder for 2FA Workflow Diagram**
- **API Security**:

    - Restricted API access using JWT-based authentication and custom permission classes.
    - Rate limiting applied differentially based on user roles.
    - **Placeholder for API Endpoint Access Chart**
- **Load Testing**:
    - Application handled 500 concurrent requests with minimal latency.
    - **Placeholder for Locust Load Test Graph**

Discussion/Analysis

- **Asynchronous Processing**:
    - Offloading email tasks would improve user experience but requires careful monitoring of Redis and Celery worker health.
    - Proper error handling ensured retries for failed tasks without user intervention.
- **Security Measures**:
    - The combination of JWT and custom permissions provided robust API security.
    - Rate limiting effectively mitigated the risk of denial-of-service attacks.
    - Field-level encryption ensured compliance with data protection standards.

Conclusions

- Asynchronous processing with Celery significantly improved the responsiveness of the application.
- Implemented security measures made the system resilient to threats and capable of handling high traffic without compromising performance.