SUA KANG

Software Enginner

+82 1045921134 rkdtndk09@gmail.com

https://github.com/suakang17

EDUCATION

Bachelor of Oceanography

Pusan National University

2019 - 2023 GPA 3.55/4.5

- Related subjects
  - Understanding UNIX
  - Python Data Analyzing
  - Fintech

SKILLS

- Java/Spring
- Python/FastAPI
- MySQL/MongoDB/MSSQL
- Docker
- Redis
- Kafka
- Git/Github

LANGUAGE

English

- TOEIC 975
- TOEIC Speaking AL

Hi there!
I have been focusing on backend development using Java and Spring Boot since 2022.

Participating in a project to transform a legacy ERP system into microservices in Smilegate Holdings. When performing tasks, I consider it important to design code structures that are easy to maintain and expand.

Recently interested in object-oriented programming and functional programming. While I mainly develop following object-oriented principles, I also strive to actively utilize the advantages of functional programming when appropriate.
My goal is to choose the optimal approach for each situation, considering the characteristics of both paradigms.

I aim to grow into a developer who can design efficient and stable systems through practical experience, not just implementing functions, but constantly striving for better solutions.

WORK EXPERIENCE

Smilegate Holdings — 2023.04 -

Server Enginner in the MSA ERP renewal project

OTHER EXPERIENCES

Smilegate DevCamp 5th — Dec.2023 - Feb.2024

MSA Based Music Streaming Social Network Service Project - Server Developer

Operating System Seminar, Kafka Seminar

Samsung Software Accademy — Aug.2023 ~ Dec.2023

Algorithm, Java/Spring, Vue.js, MySQL study

Desktop Exercise Reminder Application Project - Server and Infrastructure Implementation

UMC PNU Java/Spring Seminar — Mar.2022 ~

Project Busan Algorithm Seminar, Side Project (Java/Spring) — Jan.2022 ~

GDSC PNU GDSC Ideathon, Side Project (Vue.js), Seminar — Mar.2021 ~

PROJECTS

Work Experience - Summary

Smilegate Holdings

Project to reconstruct the monolithic structure of a legacy ERP system into a microservice-based architecture

- Development of backend logic for distributed system infrastructure, integrated authorization management, general affairs, and attendance systems
- Organized mini refactoring seminars within the team to improve code reusability and maintainability
- Removed single server dependency and built a Kafka KRaft Mode Cluster, creating an environment capable of processing approximately 170 data transactions per second
- Built a CDC pipeline using Kafka Connect that processes over 100 DB I/O events per second
- Implemented CI/CD pipelines based on Jenkins Webhook Events for development, staging, and production environments of microservices, automating deployment

Work Experience - Detail : Enterprise Authorization Management

Apr.2024 - Jul.2024

Comprehensive Backend Implementation, Infrastructure Setup, and RESTful API Development

- Participated in the early stages of the project to establish development standards and create common functionalities (e.g., common validation, multilingual support).
- Previously, user-specific permissions were hardcoded, leading to repetitive development across multiple screens and a lack of management for underlying domains, resulting in an inability to accurately assess permission statuses.
- Revamped by separating into an independent integrated permission management system and redesigning the system architecture to resolve the existing issues.

Java 17
SpringBoot 3.1.2
Redis MSSQL
Rocky Linux 8.9
Harbor
Jenkins
Kafka

Work Experience - Detail : Attendance System Reconstruction Project

Apr.2024 -

Participation in Backend Implementation, Infrastructure Setup, and RESTful API Development

- Implemented backend logic for the back-office based on permissions set by the integrated permission management system.
- Developed backend logic for the attendance service used across the organization.
- Improved the existing attendance system’s structure and performance through Kafka event blocking.
- Wrote Jenkins CI/CD scripts.

Java 17
SpringBoot 3.1.2
Redis MSSQL
Rocky Linux 8.9
Harbor
Jenkins
Kafka

LALALA - Music Streaming Social Service

Smilegate DevCamp Dec.2023 - Feb.2024

Responsible for Server Backend Design and Implementation, Infrastructure Setup, and RESTful API Development

Details:

1. Chat Server
   - Implemented a group chat feature allowing users to listen to the same playlist and share their thoughts in real-time via streaming.
   - Developed the chat API.
   - Designed and implemented the system to handle message loss issues and server scale-out using Kafka.
   - Conducted STOMP performance tests with JMeter, achieving a 0% error rate with 1000 threads.
   - Managed traffic with Nginx to maintain a stable connection for active users by limiting connections when requests exceeded the number of threads specified in Spring.

Java 17
SpringBoot
KafkaProducer/Consumer
MongoDB
Websocket STOMP

2. Playlist Server
   - Developed the playlist API.
   - Improved performance.
   - Introduced caching with Redis to handle a high volume of write operations (adding, deleting, and reordering tracks) efficiently.

Java 17
SpringBoot
Redis
MongoDB

3. Search Server
   - Developed indexing and search functionalities using the Elasticsearch Search API for target domains.

Python 3.11
FastAPI
Elasticsearch 8.12.0

LALALA - SYSTEM DESIGN AND PROBLEM-SOLVING JOURNEY

Melon (Music Streaming) Clone MSA Team Project: Backend / Infrastructure

In our team project of creating a Melon (music streaming) clone with a microservices architecture (MSA), I was responsible for the backend servers (chat server, playlist server, search server) and the infrastructure. Here, I will share my experience in building the chat server and playlist server.

Chat Server

The chat server allows users to listen to the same playlist synchronized by song and chat with each other. From architecture design to performance testing using JMeter, I was able to apply simple algorithms and knowledge in the field of networking.

1-1. Chat Implementation

Based on a Java/SpringBoot server, I implemented real-time group chat using WebSocket STOMP. During this process, I encountered two main issues:

1. Limitation on the size of sessions that can be accommodated
2. Possibility of message loss in case of failure

To address these issues, I introduced Kafka as an external message broker. This approach was chosen because it allows for scaling out, making it flexible to handle large-scale traffic. Additionally, messages are first stored and then delivered to subscribers, ensuring data consistency as users can rejoin the chat rooms.

1-2. Performance Testing

As we were testing the load for large group chats, we designed scenarios to connect, subscribe, and send messages simultaneously while maintaining the connections. In our JMeter STOMP load test, we achieved a 0% error rate in an environment with 1000 threads.

1-3. Performance Issues and Improvement Considerations

Errors began occurring when users did not receive responses in environments with 3000 threads. Initially, we thought it was solely a server issue, but further study revealed that it was due to the read timeout value set in JMeter. Requests were delayed, leading to read failures. Increasing the read timeout to 20000ms resolved the read errors. Users maintain a SUBSCRIBE state while connected and wait to receive messages until they UNSUBSCRIBE, which solved the response error. For performance improvement, we considered introducing Nginx to manage traffic. Since Spring can experience heap memory overflow if more requests arrive than the configured number of threads, Nginx could help maintain a stable connection count, providing more reliable service to currently connected users.

1-4. Song Synchronization Algorithm

An algorithm was applied to synchronize songs based on the creation time of the chat room, the playtime of songs in the playlist, and the user's entry time. Consequently, when a user joins or rejoins, the response body sent to the client includes the ID of the song that should be played at that moment.

By sharing this experience, I hope to convey the technical challenges and solutions encountered during the development of the chat server for our music streaming service clone.

Playlist Server

The playlist server handles the playlist functionality of the music streaming service, utilizing knowledge in networking and computer architecture. This server mainly operates CRUD (Create, Read, Update, Delete) operations for playlists, and retrieves songs through communication with a separate music server using OpenFeign.

Considering that write operations (such as adding, deleting, and rearranging songs) would dominate, leading to potential performance degradation, we introduced caching with Redis to mitigate this issue.

Caching Strategy

Given the high volume of write operations, our caching strategy focused on two key points:

1. Reducing resource consumption by combining both writing to and reading from the cache first.
2. Implementing a Look-Aside and Write-Back approach, considering that even if the cache fails, it is not critical to the domain.

Additionally, the Write-Back process was implemented asynchronously to enhance performance by allowing the processor to handle other operations concurrently.

By sharing these experiences, I aim to highlight the technical challenges and solutions involved in developing the playlist server for our music streaming service clone.