안녕하세요. 김승진입니다.
Server Developer

Phone.
Email. ohksj77@naver.com
GitHub. https://github.com/ohksj77
Blog. https://ohksj77.tistory.com

**한국공학대학교** 학사 [졸업] 구) 한국산업기술대학교 컴퓨터공학부 소프트웨어학과 2020.03 ~ 2024.02

# 소개

- 안정적으로 데이터를 처리할 수 있는 백엔드 엔지니어입니다.
  - MQ를 통해 비동기로 데이터를 처리하며 데드레터와 재시도 전략을 수립했습니다.
- 문제 해결과 개선 방안을 꾸준히 고민합니다.
  - 실시간 양방향 위치 공유 시스템 설계를 위해 기술들을 비교 분석하며 선정했습니다.
  - 여러 단계에 걸쳐 Full Text 인덱스를 탐구하며 성능을 개선했습니다.
- 주도적으로 문제를 해결하기 위해 노력합니다.
  - rabbitmq-java-client, quartz 오픈소스에 기여하며 직접 문제를 해결한 경험이 있습니다.

# 기술 스택

Java Spring, JPA, MySQL, Redis, Docker, aws ec2

# 프로젝트

* 항목별 **파란** 글꼴은 블로그 포스트로 연결됩니다.

## 이길저길 길치들을 위한 경로 제공 및 만남 관리 서비스

GitHub 저장소: https://github.com/HongDam-org/TWTW

*#STOMP #OpenAPI #FCM*

**주요 기여 기술 : Java Spring, JPA, MySQL, RabbitMQ, Redis, Docker, GitHub Actions**

백엔드(70%), 인프라(100%): [위치 공유, 알림, 친구, 약속장소, 이미지, 데드레터] 도메인 / Jib 및 cicd 기반 aws ec2 배포

1. **실시간 양방향 위치 공유 시스템 설계 및 지속적으로 갱신되는 위치 좌표 저장 고민**
   - **Websocket, API 폴링, SSE 비교 분석과 STOMP 선택**
     - [실시간성 / 양방향 지원 / 업데이트 빈도 적합성(각 다른 시점, 3초 주기)] 로 요구사항 기반 기준 비교
     - 위 비교와 **성능 테스트 수행 결과**를 함께 고려해 STOMP 선택
   - **queue 활용 전략 고민과 Message Broker 선택**
     - 사용자 그룹당 평균 1.3/s 의 요청이 오기에 여러 개의 queue로 분산하는 것이 좋을 것이라 판단
     - 그룹별로만 데이터를 공유하면 되기에 그룹별로 Binding 하여 queue 사용하는 방법 고려
     - 동적으로 queue 생성/삭제 가능하며 그룹별 라우팅을 쉽게 구성 가능한 RabbitMQ 선택
   - **지속적으로 갱신되는 위치 좌표 저장 고민과 Redis Geo 선택**
     - O(log(n)) 의 거리 연산이 가능한 Redis Geo를 DB 활용에 비해 성능적 개선을 기대하며 선택
   - **Redis, RabbitMQ aws 아키텍처 선택**
     - Topic당 2만, queue당 2천 이상의 tps로 설계 요구사항을 상회하는 처리량을 제공하는 Amazon MQ, 장애 대응을 위한 스냅샷 등을 제공하는 AWS Elasticache를 선택하며 결합도 낮은 아키텍처 구성

2. **FCM 알림 발송 비동기 처리를 위한 MQ 활용과 데드레터 전략 수립**
    - **비동기 처리 고려**
        - FCM의 응답은 사용하지 않으며 성능을 고려해 아키텍처의 RabbitMQ로 비동기 처리하기로 결정
    - **데드레터와 재시도 전략 수립**
        - DB에 진행 상태를 저장하였으며 에러를 파악하기 위해 데드레터 발생 시 Slack 알림 발송
        - 관리자 권한 API를 통해 데드레터 DB PK를 받아 DB 조회 후 큐로 재전송 가능하도록 구성
        - 일시적인 에러의 경우에는 재시도로 처리하고자 최소 10초 간격으로 지수 백오프 재시도

3. **OpenAPI RateLimit 이슈 대응과 호출 비용 절감을 위한 캐시 적용 및 만료 전략**
   [Latency 91.88ms -> 22.26ms 4.12배 개선]
    - **불필요한 OpenAPI 호출 고민과 캐싱 도입**
        - 데이터가 자주 변경되지 않는 OpenAPI를 지속적으로 호출, 간헐적으로 RateLimit 이슈 발생
        - 기존 평균 91.88ms 소요되었던 OpenAPI 활용 부분 캐싱으로 이슈 해결 기대
    - **Cache Aside 패턴 기반 캐싱 도입과 갱신/만료 전략**
        - 동일 로직의 @Cacheable를 사용하는 API와 @CachePut을 사용하는 API로 분리
        - 평균 22.26ms로 개선 및 다음과 같은 갱신/만료 전략 수립
        - 새로고침 요청 시 @CachePut 사용 API 호출하여 OpenAPI 직접 호출 이후 캐시 갱신
        - 일정 시간이 지나 새 데이터 확인이 필요한 경우를 고려해 TTL 설정으로 캐시 만료

4. **무분별한 Mock 사용 제거를 위한 Strategy 패턴을 적용한 테스트 더블**
    - **단위 테스트 중 Mock 문제와 코드 구조 고민**
        - Service Layer 테스트 시 Mock 사용이 많았으며, 저장 이후 목록 조회 등의 로직 동적 테스트 불가
    - **Fake객체와 Strategy 패턴으로 구조 개선**
        - JpaRepository를 상속받는 인터페이스와 Fake객체의 공통 인터페이스를 만들어 상속받도록 수정
        - @TestConfiguration 클래스에 @Primary와 함께 공통 인터페이스 타입으로 Fake 구현체 빈 등록
        - 운영 시와 테스트 시 서로 다른 구현체가 주입되었으며 무분별한 Mock 사용 제거

5. **OpenAPI 서버 장애 이슈 대응을 위한 서킷 브레이커와 모니터링**
    - **OpenAPI 서버 장애에 따른 대응 고민**
        - Kakao, Naver, Tmap 의 OpenAPI 서버의 점검이나 장애 상황에도 호출 및 대기 후 에러 발생 반복
        - OpenAPI 호출을 동적으로 처리하고 DB 데이터는 응답이 가능하도록 Resilience4J 서킷브레이커 도입하여 호출 중 실패율이 30% 도달 시 Open 되어 fallback 메서드로 기본 응답 반환
    - **모니터링의 도입과 알림 시스템**
        - OpenAPI, FCM를 사용하며 RabbitMQ와 같은 기술과 상호작용하는 등 시스템에 변수가 많은 점 또한 고려해 모니터링 도입 및 Spring Actuator 중 Prometheus와 Resilience4J 사용
        - Grafana로 개선 작업 시 API Latency, 운영 시 API Status Code 분포 등 확인
        - 장애 상황을 빠르게 파악하고 대응하고자 AlertManager로 관리자 Slack으로 알림 발송하도록 구성

6. **Like 쿼리 개선을 위한 Full Text 인덱스 도입과 n-gram parser 분석**
   [쿼리 2.63s -> 0.53s 4.96배 개선]
    - **구현 초반 사용한 Like 쿼리와 문제점**
        - 닉네임의 Like 기반 검색 쿼리가 500만 row의 데이터 기준으로 평균 2.63s 소요
        - 문자열 검색을 지원하는 MySQL의 Full Text 인덱스로 Like 기반 쿼리를 대체하고자 함
    - **match against 쿼리 에러 발생**
        - n-gram parser와 함께 boolean mode로 match 연산 시 쿼리 캐시 공간 부족 에러 발생
    - **첫 번째 시도: 닉네임 길이 제한**
        - 쿼리시 고려할 경우의 수를 줄이고자 닉네임 최대 길이 8로 제한하니 29s 이상 걸리는 롱 쿼리 발생
        - query profiling 확인하여 FULLTEXT initialization 과정에서 29s 소요됨을 확인
    - **두 번째 시도: n-gram parser 특징을 고려한 쿼리 수정**
        - [ngram_token_size = 2] 임을 확인하며 검색어의 길이가 2보다 긴 것이 원인이라 추측
        - 검색어 string을 길이 2씩 분할하여 쿼리하도록 수정해 평균 0.53s로 개선

## GitRank 블록체인 기반 깃허브 활용도를 통한 랭킹 서비스

GitHub 저장소: https://github.com/tukcom2023CD/DragonGuard-JinJin

*#크롤링 #블록체인 연동 #OpenAPI #이메일*

**주요 기여 기술 : Java Spring, JPA, MySQL, Redis, Python**

백엔드(100%), 인프라(100%): 프로젝트 리더를 맡은 한이음 ICT 멘토링 공모전 입선작으로 Kotlin으로 리팩토링 중입니다.

1. **GitHub OpenAPI의 느린 응답 이슈 개선을 위한 스케줄링을 통한 DB 업데이트**
   [Latency 7.72s -> 49.26ms 15.6배 개선]
   - **기존 구현과 OpenAPI의 응답 속도 이슈**
     - 5s ~ 10s를 기다려야 응답을 받는 OpenAPI를 매번 호출하며 평균 Latency 7.72s로 응답
   - **첫 번째 시도: DB 우선 조회**
     - 결과 데이터를 DB에 저장하여 DB 데이터 반환하며 Latency 평균 49.26ms로 개선
   - **두 번째 시도: 스케줄링으로 업데이트**
     - GitHub 서버와의 데이터 Sync를 위해 스케줄링으로 2시간마다 전체 데이터 업데이트
   - **GitHub OpenAPI의 RateLimit과 히트율을 고려한 개선할 사항**
     - Star 수가 많은 Repository만 스케줄링으로 업데이트 및 개인 Repository는 로그인 시 일괄 업데이트

2. **DB 동시성 이슈 개선을 위한 Lock 전략 수립과 Lock 해제 시점 고민**
   - **요구사항과 동시성 이슈**
     - 한 번의 요청에 각각 다른 4종류의 데이터를 가져와 하나의 테이블에 update, 한 테이블에는 insert
     - 동시 여러 번 호출 시 랭킹이 잘못 산정되는 문제 발생
   - **첫 번째 시도: 낙관적 락**
     - 성능을 생각해 낙관적 락을 우선 적용하였지만, 충돌로 인한 롤백 자주 발생
   - **두 번째 시도: 베타락**
     - 베타락을 도입했으나, 데드락 발생으로 timeout 설정 필요 및 tps 영향 우려
   - **Redis 락 적용과 Lock 해제 시점**
     - 이후 아키텍처에 Redis가 추가되어 INCR, Lists, SETNX 활용과 비교해 Redisson RLock으로 수정
     - 독립된 처리를 위한 전파 레벨 설정, 정합성을 위해 별도 생성 트랜잭션이 끝나면 Lock 해제되도록 구현

3. **DB 페이징 및 정렬 기반 랭킹 시스템 Redis 자료구조를 통한 개선**
   [Latency 120.44ms -> 26.26ms 4.58배 개선]
   - **랭킹 페이징 쿼리 성능 이슈**
     - 5만 row 더미 데이터 삽입 후 테스트 시 페이징 및 정렬 기반 랭킹 조회 API Latency 120.44ms 소요
   - **커서 기반 페이징 도입 불가로 Redis의 Sorted Set 고민과 도입**
     - Member의 DB PK와 포인트를 Redis Sorted Set에 저장하여 랭킹을 산정하도록 재구현
     - 부하 테스트 기준 랭킹 API Latency 평균 26.26ms로 개선

# 오픈소스 기여

**rabbitmq/rabbitmq-java-client** [2024.11]
PR#1469, PR#1476
requeue 메트릭 추가 및 해당 메트릭 수집 기능 추가

**quartz-scheduler/quartz** [2024.11]
PR#1260, PR#1261
다중 misfired trigger를 retrieve 중 예외 시 롤백 및 재처리로 인한 무한 실패 이슈를 에러 핸들링으로 해결

# 활동

**한이음 ICT 멘토링**
2023.04 ~ 2023.11

**한국공학대학교 UMC 4기 운영진**
2023.03 ~ 2023.08 * 대학 IT 연합 동아리

**교내 프로그래밍 동아리 씨부엉 운영진**
2022.03 ~ 2022.12

# 수상내역

**TUKOREA SW-PowerUp**
[2023.12] 한국공학대학교 컴퓨터공학부
최우수상

**한이음 ICT 멘토링 공모전**
[2023.12] 한국정보산업연합회
한국정보산업연합회장상(입선)

**SW 캡스톤 디자인 콘테스트**
[2022.10] 한국공학대학교 컴퓨터공학부
Pre-캡스톤 디자인 부문 동상