Professional Summary
Java/Spring 생태계에 대한 깊은 이해를 바탕으로 대규모 트래픽을 안정적으로 처리하는 백엔드 시스템을 설계하고 구현해온 3년차 엔지니어입니다.
단순한 기능 개발을 넘어, 시스템의 확장성(Scalability)과 가용성(Availability)을 고려한 아키텍처 설계를 지향합니다.
특히 모놀리식 아키텍처의 한계를 경험하고 이를 MSA(Microservice Architecture)로 성공적으로 전환하여 배포 유연성을 확보한 경험이 있습니다.
Kafka를 활용한 이벤트 기반 아키텍처(Event-Driven Architecture)를 도입하여 서비스 간 결합도를 낮추고 데이터 처리량을 극대화하는 데 강점이 있습니다.
코드 품질을 위한 TDD 실천과 Clean Architecture 적용을 통해 유지보수 비용을 낮추는 팀 문화를 주도한 경험이 있습니다.

Work Experience

1. 핀테크 결제 플랫폼 MSA 전환 프로젝트
- **기간:** 2023.01 ~ 2024.12 (24개월)
- **역할:** 백엔드 코어 개발자 및 아키텍처 설계 (팀원 5명)
- **Tech Stack:** Java 17, Spring Boot 3.0, Spring Cloud, Kafka, Redis, PostgreSQL, Docker, AWS EKS

Context & Problem
- **[배경]** 기존 레거시 시스템은 거대한 모놀리식 구조로, 결제 로직 수정 시 전체 시스템을 재배포해야 했으며, 블랙 프라이데이 등 트래픽 폭주 시 전체 서비스 장애로 이어지는 구조적 취약점이 있었습니다.
- **[문제]** 특정 서비스(쿠폰)의 장애가 메인 결제 로직까지 전파되어 매출 손실이 발생했으며, 배포 시간이 1시간 이상 소요되어 긴급 대응이 불가능했습니다.

Solution (Key Actions)
- **[서비스 분리]** 도메인 주도 설계(DDD) 워크샵을 주도하여 '주문', '결제', '쿠폰', '배송'의 Bounded Context를 정의하고, 점진적으로 서비스를 분리했습니다. Strangler Fig 패턴을 적용하여 운영 중단 없이 이관을 진행했습니다.
- **[분산 트랜잭션 관리]** 서비스 간 데이터 정합성을 보장하기 위해 기존의 2PC(2-Phase Commit) 대신, 가용성이 높은 Saga Pattern(Chureography 방식)을 도입했습니다. Kafka를 통해 결제 성공 이벤트를 발행하면, 쿠폰 서비스와 배송 서비스가 이를 구독하여 후속 처리를 하도록 구현했습니다.
- **[회복 탄력성 확보]** 외부 PG사 연동 구간에 Resilience4j 라이브러리의 Circuit Breaker를 적용했습니다. 특정 PG사 장애 발생 시 자동으로 백업 라인으로 트래픽을 우회하거나 Fail-fast 처리하도록 하여, 전체 시스템의 응답 지연을 방지했습니다.

Results (Quantitative)
- 배포 프로세스 개선: 전체 빌드/배포 시간 **60분 -> 5분**으로 90% 단축.
- 시스템 안정성: 피크 타임 TPS 3,000 상황에서도 에러율 0.01% 미만 유지 (SLA 99.99% 달성).
- 결제 처리 성능: 비동기 처리 도입으로 평균 응답 시간 **500ms -> 150ms**로 개선.

2. 사내 포인트 시스템 대규모 리팩토링
- **기간:** 2022.06 ~ 2022.12 (6개월)
- **Tech Stack:** Java 11, Spring Boot, MySQL, Redis

Context & Problem
- **[문제]** 동시 접속자가 몰릴 때 포인트 차감 로직에서 'Lost Update' 문제가 발생하여 포인트 잔액이 마이너스가 되거나 중복 차감되는 데이터 정합성 이슈가 빈번했습니다.

Solution

- **[동시성 제어]** DB 레벨의 비관적 락(MySQL `SELECT ... FOR UPDATE`)을 도입하여 원자성을 보장했습니다. 하지만 이로 인한 데드락 가능성과 성능 저하를 우려하여, Redis 기반의 분산 락(Redisson)으로 고도화했습니다.
- **[캐싱 전략]** 포인트 잔액 조회 트래픽이 전체 DB 부하의 60%를 차지함을 확인하고, Redis에 포인트 정보를 캐싱(Look-aside 패턴) 적용했습니다. 캐시 일관성(Cache Consistency) 유지를 위해 포인트 변동 시 캐시를 즉시 무효화(Evict)하는 전략을 세웠습니다.

Results
- 동시성 테스트(JMeter) 1,000 VUser 환경에서 데이터 오차 **0건** 달성.
- DB CPU 사용률 **80% -> 20%**로 안정화.

Personal Projects

1. 대용량 트래픽 처리를 위한 티켓 예매 시스템 (Toy Project)
- **목적:** 선착순 이벤트와 같은 극단적인 트래픽 상황을 가정하고, 대기열 시스템을 직접 구현해보기 위함.
- **Tech Stack:** Spring WebFlux, Redis Sorted Set, Kafka

Trial & Error
- **[시도]** 처음에는 RDB에 대기열 테이블을 만들어 폴링(Polling) 방식으로 구현했으나, DB 커넥션 폭주로 서버가 다운됨.
- **[개선]** Redis의 `Sorted Set` 자료구조를 활용하여 인메모리 대기열 시스템을 구현. 타임스탬프(Score) 기준으로 순서를 보장하고, `rank` 명령어로 대기 순번을 O(logN) 시간복잡도로 조회하도록 개선.
- **[학습]** 전통적인 RDB 중심 설계에서 벗어나, 데이터 특성에 맞는 NoSQL 활용의 중요성을 체감함. Non-blocking I/O 모델인 WebFlux의 이점을 학습함.

Skills
- **Proficient (능숙):** Java, Spring Boot, Spring Data JPA, MySQL, Git
- **Experienced (경험):** Kafka, Redis, Docker, Kubernetes, AWS (EC2, RDS, S3), JUnit5
- **Familiar (이해):** Kotlin, React, Elasticsearch, Hadoop