



박상준

Backend Developer

Portfolio

3 Years

EXPERIENCE

### NAVIGATION

 Resume

 GitHub

 Blog

 Resume PDF

 Portfolio PDF


### PROJECT TECH STACK


 Java 17

 Spring Boot

 MySQL

 Redis

 RabbitMQ

 OpenFeign

### PROJECT TIMELINE

2024 스포츠크ard 플랫폼 진행중

2025 신한카드 연동 API 완료

2022 소셜 로그인 업그레이드 완료

## Portfolio Overview

포트폴리오 개요

3년간의 실무 경험을 통해 구축한 주요 프로젝트들의 상세 기술 문서입니다.

SI 회사에서의 신규 개발과 서비스 회사에서의 레거시 시스템 운영 경험을 바탕으로, 적

절한 트래픽 처리, 시스템 아키텍처 설계, 성능 최적화 등의 기술적 도전을 해결해온 과정을 담았습니다.

### 🕒 시스템 아키텍처

DDD 기반 계층형 아키텍처와 RabbitMQ를 활용한 **마이크로서비스 간 느슨한 결합** 구현

### ⚡ 성능 최적화

Caffeine Cache를 통한 **DB 부하 80% 감소** 및 응답속도 50% 향상

### 🔒 데이터 정합성

비관적 락과 트랜잭션 분기를 통한 **금융 시스템급 데이터 무결성** 보장

## 스포츠카드 거래 플랫폼

2024.12 ~ 진행중 | 팀 프로젝트

Spring Boot 2.7

Java 17

RabbitMQ

OpenFeign

MySQL

### 📋 프로젝트 목적

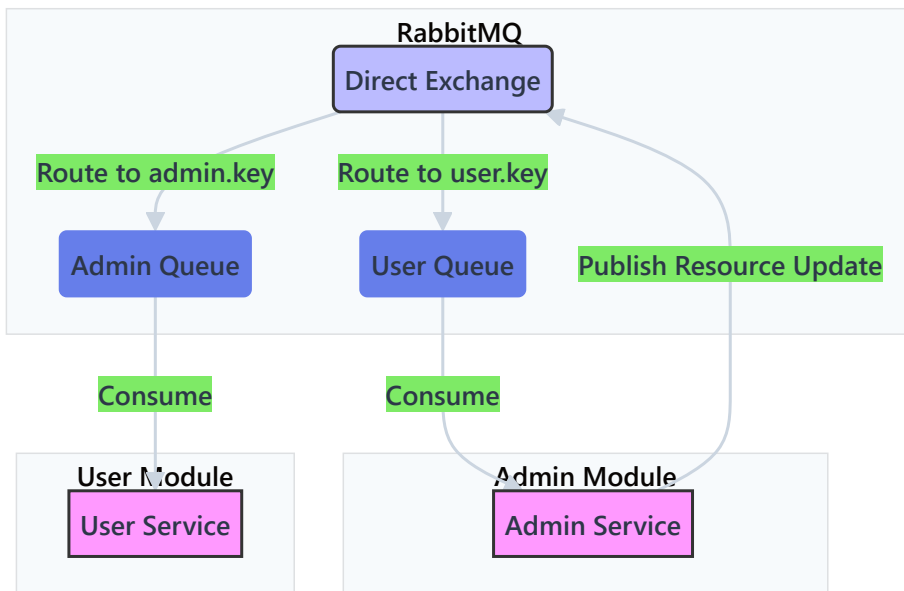
스포츠카드 중개 사이트를 통해 사용자들이 스포츠카드를 거래하고, 관련 정보를 공유하며, 실시간으로 상호작용할 수 있는 플랫폼 제공

### 🕒 인프라 아키텍처 설계

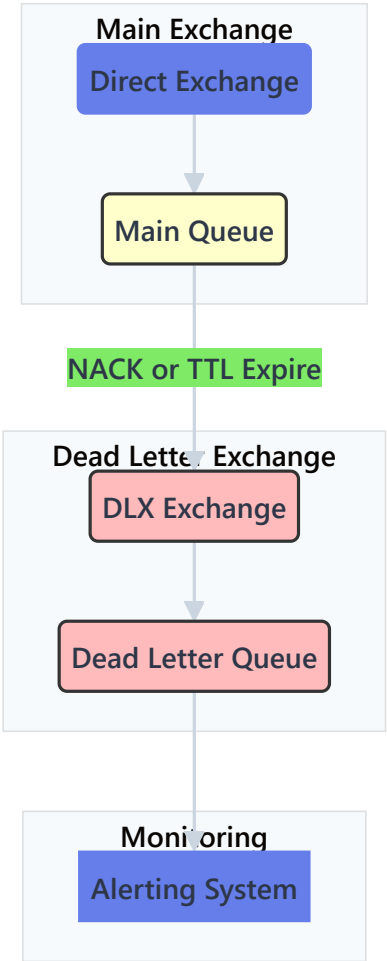
#### 메시지 브로커 선정 기준

- **강결합 문제 해결:** 관리자와 사용자 모듈 간 시큐리티 정보 동기화에서 Pub/Sub 모델을 통한 모듈 독립성 확보
- **설정 복잡도:** Kafka 대비 RabbitMQ의 간단한 설정과 Spring AMQP의 우수한 통합성
- **Spring Boot 통합성:** Spring AMQP 라이브러리를 통한 메시지 전송/수신 추상화

#### 기본 Pub/Sub 흐름



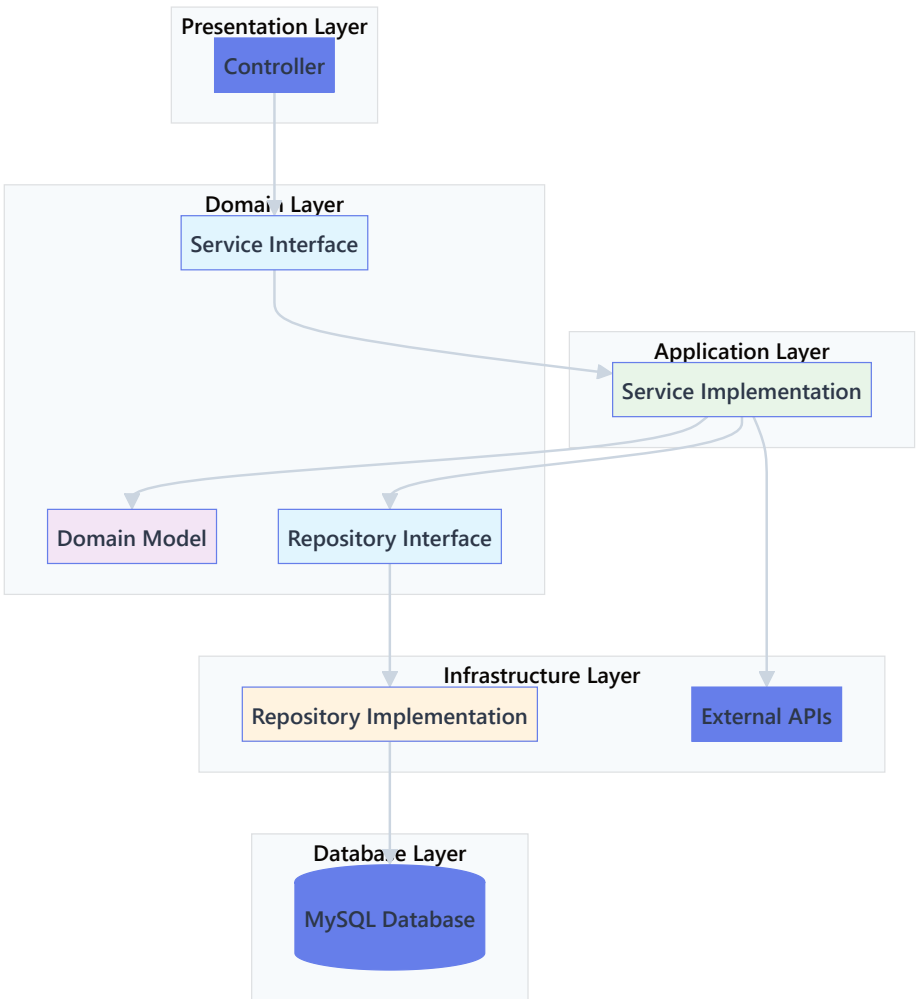
#### Dead Letter Exchange(DLX)를 이용한 실패 처리



구현한 메시지 브로커 구조

- **Exchange 구성:** Direct Exchange로 라우팅 키 기반 정확한 메시지 전달
- **DLX 설정:** 실패한 메시지의 Dead Letter Queue 처리 (TTL: 1주일)
- **모니터링:** RabbitMQ 관리자 페이지를 통한 실시간 메시지 상태 확인

💻 애플리케이션 아키텍처 설계



설계 기준

- **비즈니스 중심 설계:** 도메인 계층을 중심으로 한 일관된 의존성 방향
- **의존성 역전 원칙:** Controller가 Service 인터페이스에 의존하여 느슨한 결합 유지

- **확장성:** JPA 기반 Infrastructure 변경 시에도 도메인 계층에 최소 영향
- **팀 도입:** DDD 기반 아키텍처 가이드 문서화 및 팀 내부 교육 진행

## 신한카드 연동 광고 플랫폼

2025.01 ~ 2025.05 | 단독 개발

Spring Boot 2.7

Java 17

JPA

QueryDSL

MySQL 8.0

### 프로젝트 목적

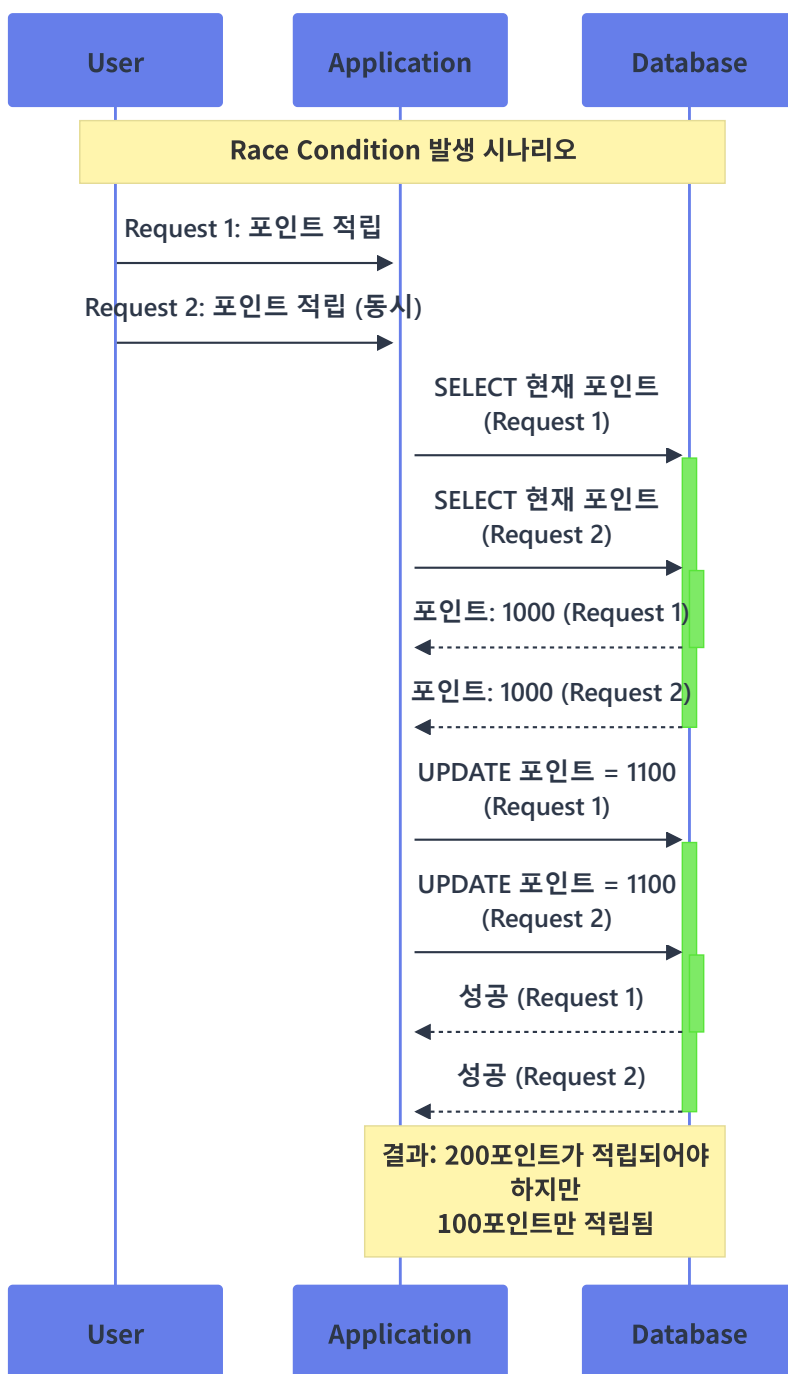
약 **90 QPS** 처리하는 신한카드 API 연동 광고 플랫폼으로, 관리자 광고 등록 → 신한카드앱 웹앱 노출 → 포인트 적립 서비스

- **사용자 광고 참여 유도:** 포인트 적립 보상을 통한 서비스 이용 증대
- **외부 API 연동:** 외부 은행 API 활용한 포인트 적립 시스템으로 사용자 혜택 확대

### Race Condition 해결

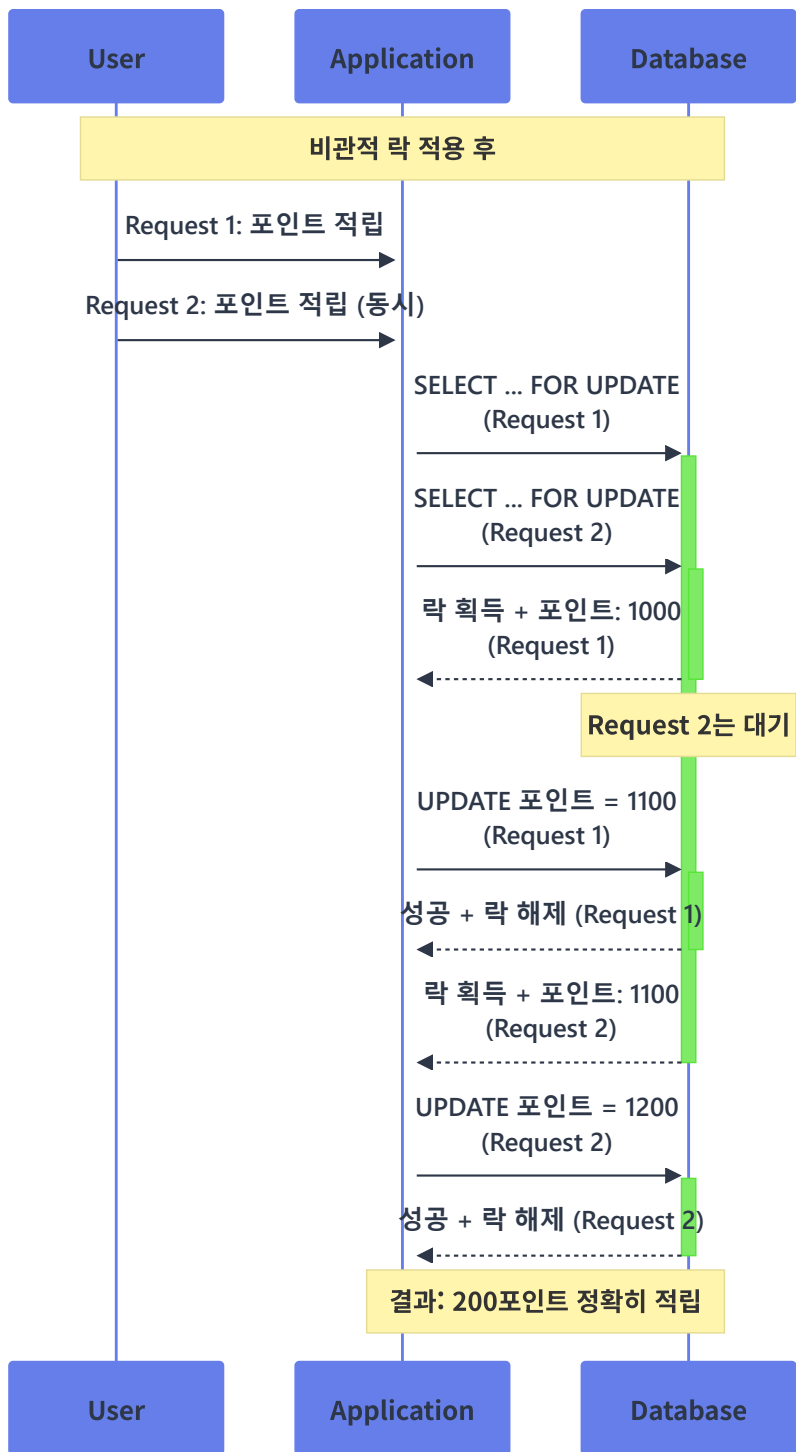
#### 문제 상황

광고 시청 후 포인트 적립에서 동일 사용자의 동시 요청으로 인한 **포인트 중복 적립** 발생



#### 해결 방안: 비관적 락(Pessimistic Lock)

- **선택 이유:** 금융 시스템의 강력한 데이터 정합성 요구사항
- **구현 방식:** SELECT ... FOR UPDATE를 통한 레코드 락 획득
- **프로세스:** 트랜잭션 시작 → 광고 시청 기록 → 비관적 락 획득 → 포인트 적립 → 트랜잭션 종료



결과

Race Condition으로 인한 포인트 중복 적립 문제를 완전히 해결하고, **데이터 정합성 100% 보장**하는 안정적인 포인트 시스템 구축

⚡ OpenFeign 재시도 최적화

문제 상황

외부 은행 API 호출 시 일시적 네트워크 오류(503)로 인한 요청 실패 → 회원 정보 호출 실패로 UX 저하

해결 방안

- **Retryer 설정:** 일시적 오류에 대한 최대 3번 재시도
- **ErrorDecoder 설정:** 503 Service Unavailable 오류 감지 및 재시도 로직

결과

외부 API 호출 안정성 개선으로 **성공 확률 증대** 및 회원정보 누락 문제 해결로 UX 향상

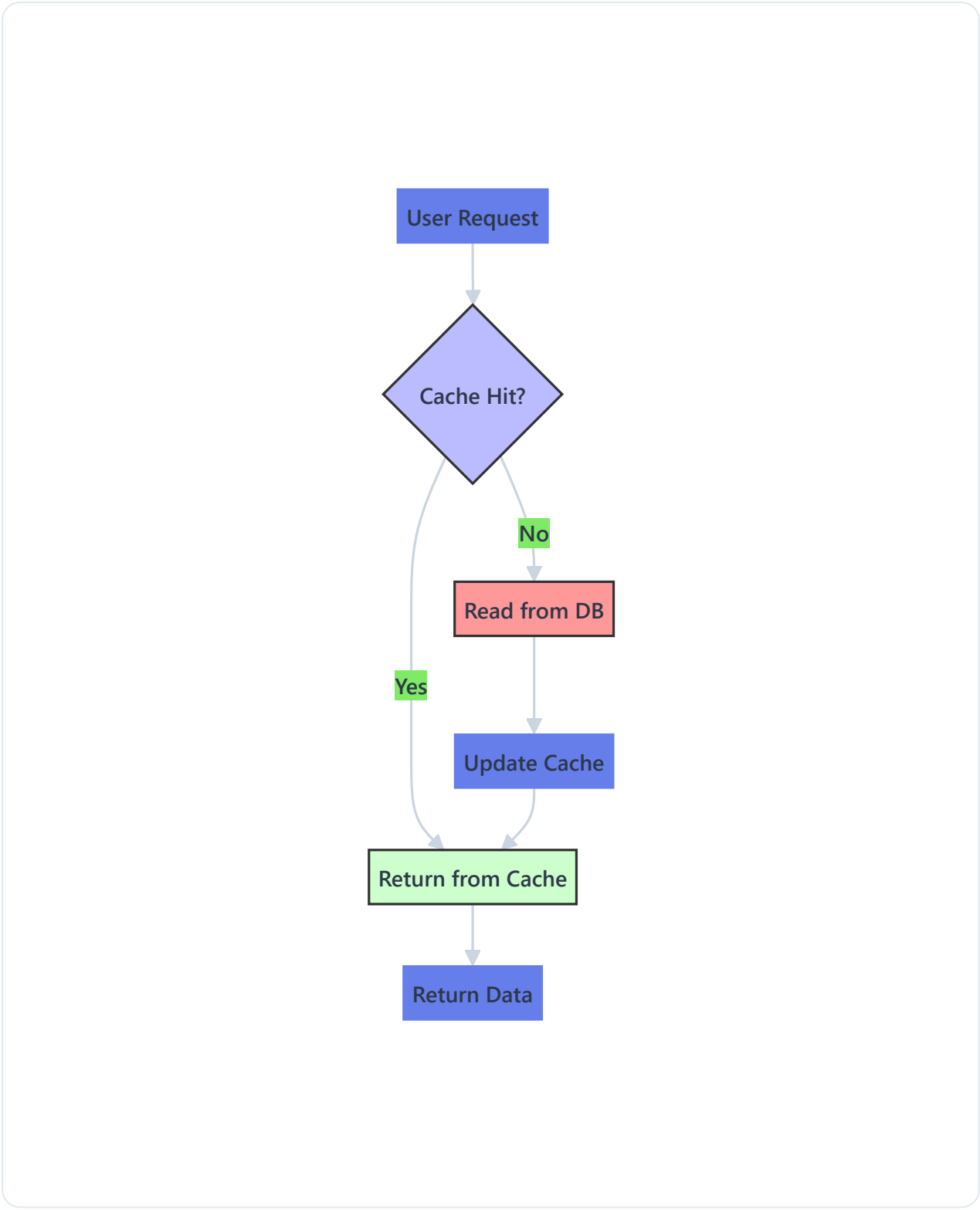
Caffeine 캐싱 시스템 구현

아이템베이 메인배너 최적화

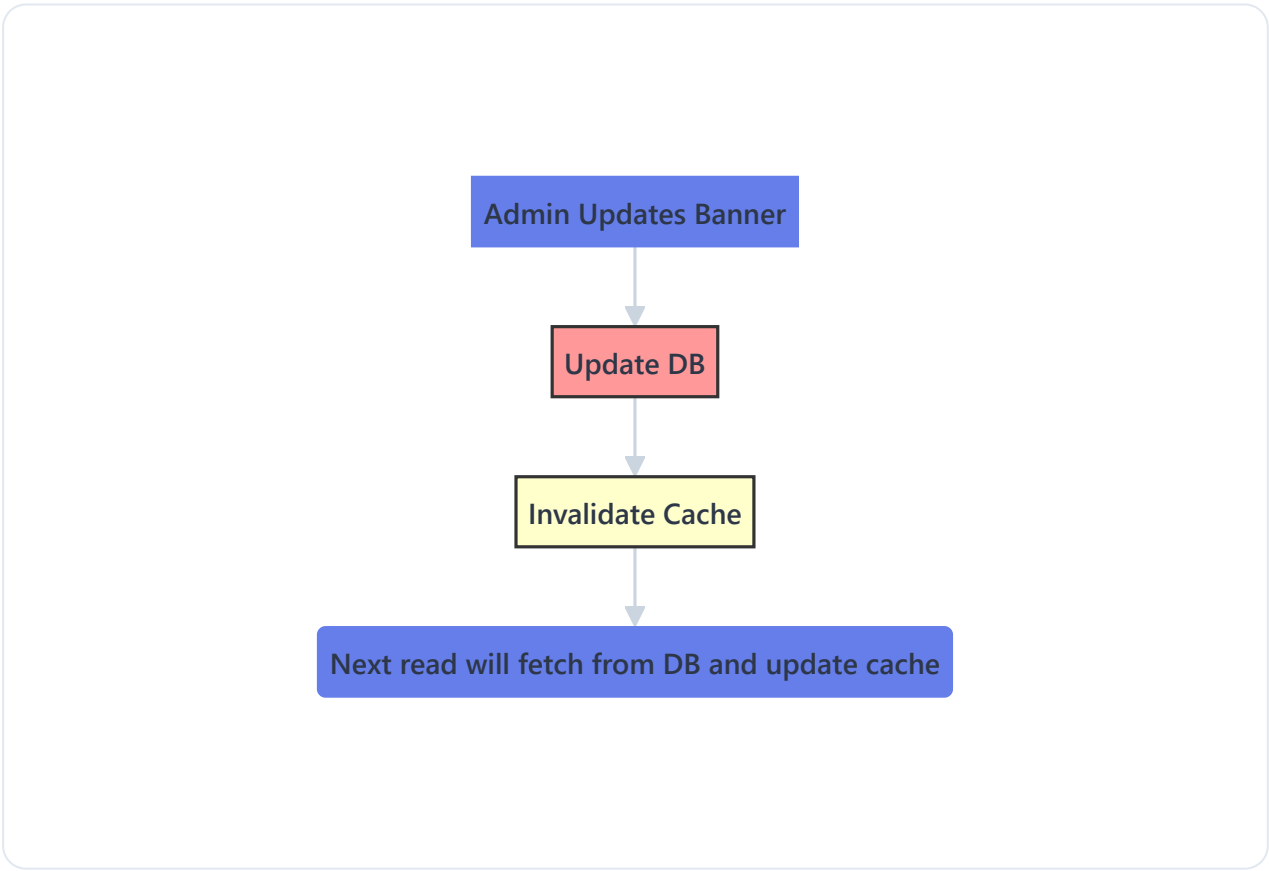
문제 상황

아이템베이 메인페이지 배너 데이터의 **매번 DB 직접 조회**로 인한 메인페이지 DB 부하

Cache Read 흐름



Cache Write/Update 흐름



구현 사항

- **캐싱 전략:** DB 직접 호출 → Caffeine Cache 조회 방식 전환
- **캐시 갱신:** 관리자 배너 수정/갱신 시 캐시 무효화 후 다음 요청에서 재조회
- **TTL 설정:** 관리자 설정 게시기간에 맞춘 만료시간 적용

결과

메인 배너 영역 **DB 부하 대폭 감소** 및 사용자 응답속도 향상

Spring Boot 2.4

Java 8

MyBatis

MySQL 8.0

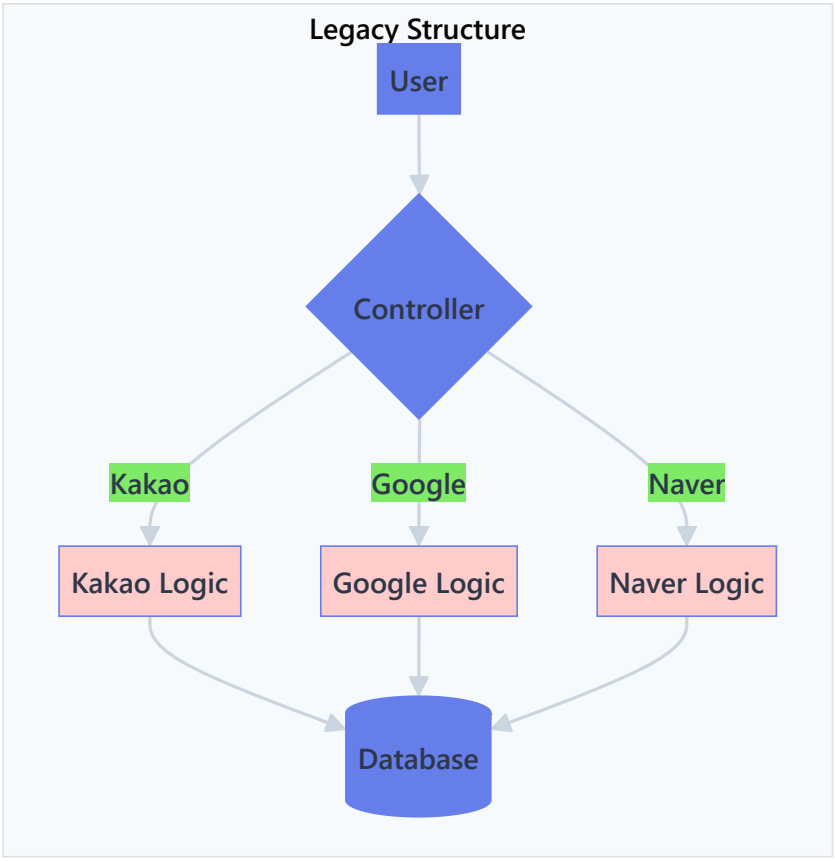
Spring Security OAuth2

## Legacy → Modern Architecture

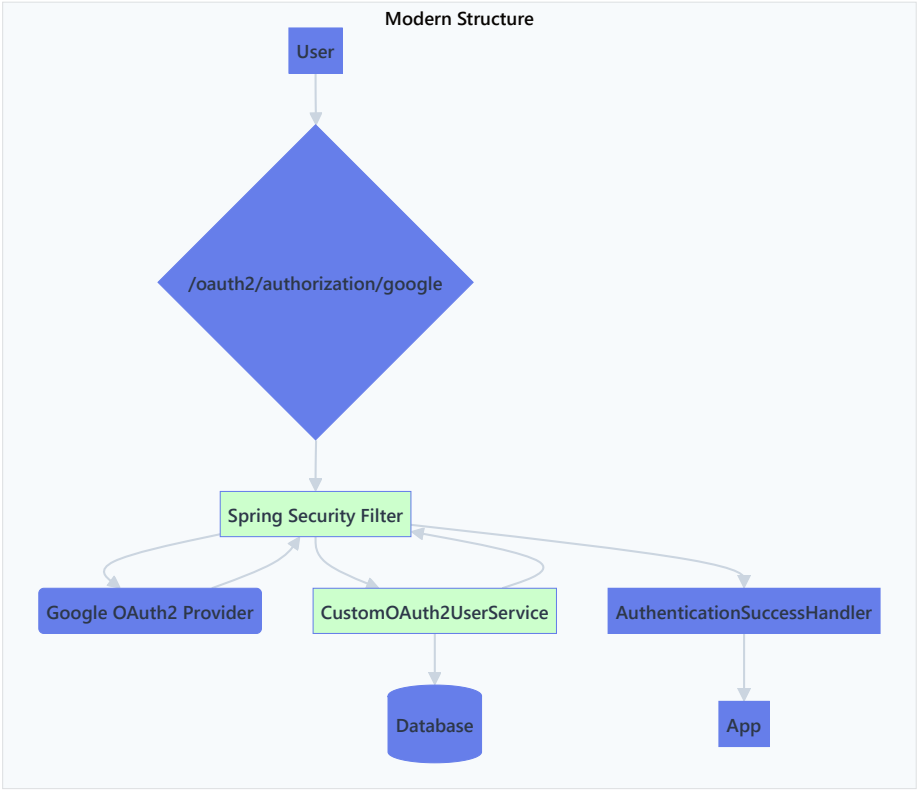
### 문제 상황

- **하드코딩된 플랫폼별 로직:** 카카오, 구글 등 각 플랫폼별 별도 로그인 로직
- **일관되지 않은 인증/인가:** 유지보수 어려움 및 사용자 경험 저하
- **확장성 부족:** 신규 소셜 플랫폼 추가의 높은 비용

### 레거시 소셜 로그인 구조



### Spring Security OAuth2 기반 통합 구조



### 해결 방안

- **Spring Security OAuth2 도입:** 통합된 소셜 로그인 지원
- **설정 중앙화:** 클라이언트 ID/Secret 환경 설정 파일 관리

- **OAuth2UserService 커스터마이징:** 통합 사용자 정보 처리
- **사용자 매핑 로직:** 기존 사용자 매핑 또는 신규 사용자 등록

결과

통합된 인증/인가 시스템 구축으로 신규 소셜 플랫폼 추가 용이성 확보 및 확장성 향상

## 트랜잭션 관리 시스템 구축

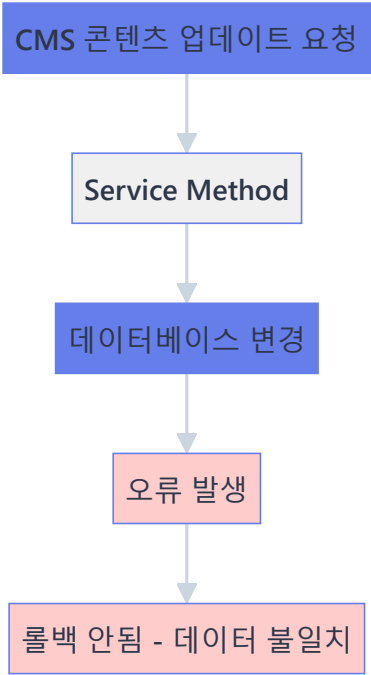
레거시 마이그레이션 중 트랜잭션 최적화

### 트랜잭션 관리 최적화

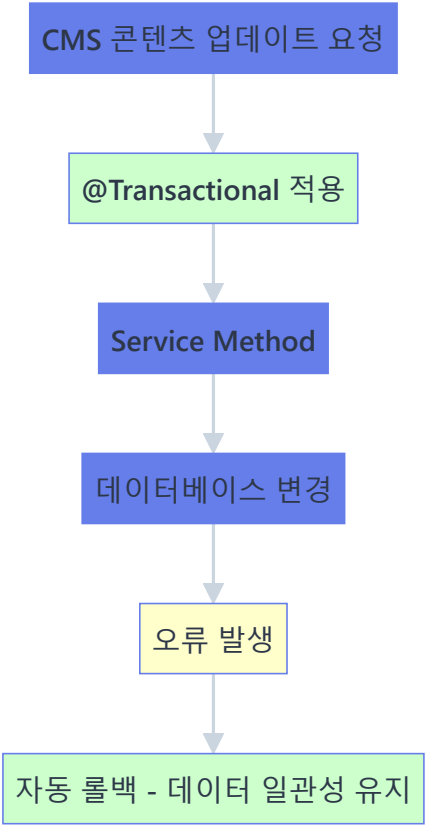
문제 상황

레거시 → Spring Boot CMS 마이그레이션 시 **글로벌 트랜잭션 설정 누락**으로 인한 롤백 미동작

문제 시나리오 - 트랜잭션 미적용



해결 후 - @Transactional 적용

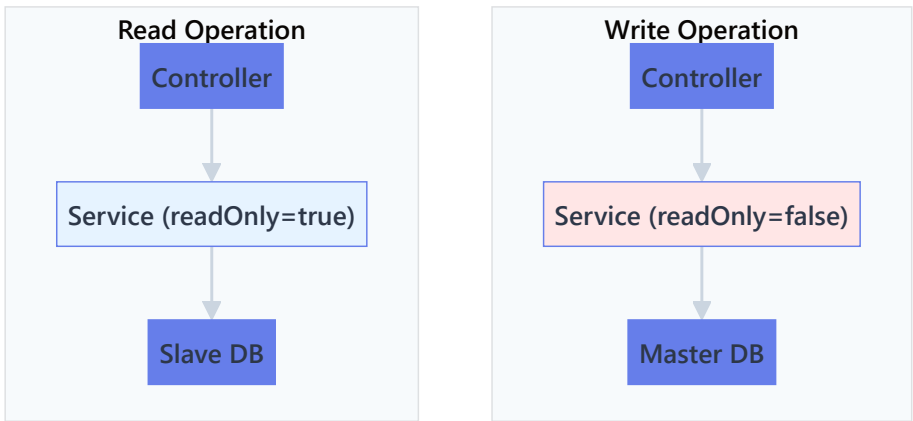




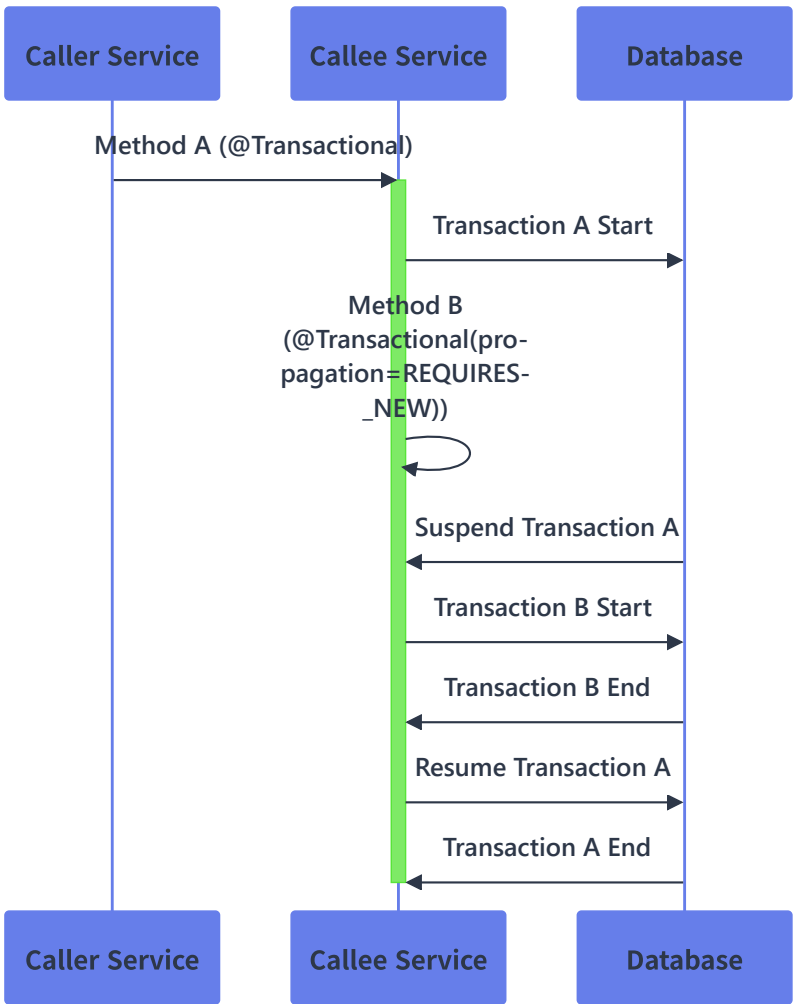
트랜잭션 분기 처리 - 읽기/쓰기 DB 분리 대비

향후 Master/Slave DB 분리를 대비한 트랜잭션 전략 수립 및 구현

읽기/쓰기 트랜잭션 분리



REQUIRES\_NEW 전파 속성을 이용한 독립 트랜잭션



구현 상세

- **읽기 전용 서비스:**
  - @Transactional(readOnly=true, propagation=REQUIRES\_NEW)
  - 별도의 트랜잭션으로 분리하여 Slave DB 연결 대비
  - REQUIRES\_NEW로 기존 쓰기 트랜잭션과 독립적 실행
- **쓰기 서비스:**
  - @Transactional(readOnly=false)
  - Master DB 연결 대비 일반 트랜잭션
  - 롤백 시 데이터 일관성 보장
- **전파 속성 분리:** 읽기/쓰기 작업의 명확한 트랜잭션 경계 설정

기본 해결 방안

- **@EnableTransactionManagement:** 글로벌 트랜잭션 관리 활성화
- **@Transactional 명시적 적용:** 주요 서비스 메서드에 트랜잭션 경계 설정
- **트랜잭션 매니저 검증:** 데이터베이스 작업 시 적절한 트랜잭션 경계 확인
- **롤백 테스트:** 의도적 오류 발생으로 롤백 동작 검증

## 결과

CMS 콘텐츠 업데이트 중 오류 발생 시 **변경사항 자동 롤백**으로 데이터 일관성 유지. 향후 **읽기/쓰기 DB 분리** 시에도 **최적화된 트랜잭션 처리** 가능한 구조 확보

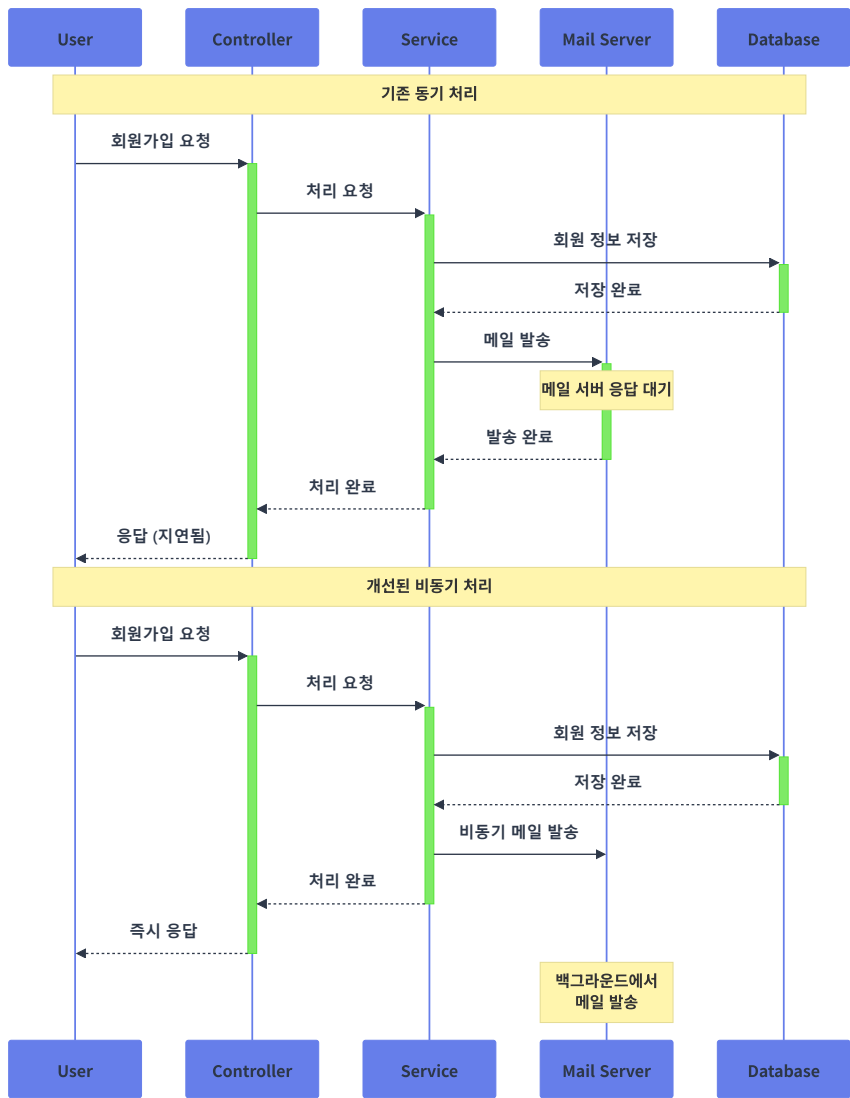
## 비동기 메일 발송 시스템

동기 → 비동기 리팩토링

### 비동기 처리 아키텍처

#### 문제 상황

동기적 메일 발송으로 인한 **사용자 응답 시간 지연** 및 메일 서버 응답 대기로 인한 전체 프로세스 영향



#### 기술적 해결 과정

##### 1. 비동기 메서드 호출 실패 해결

- **문제:** 동일 클래스 내 `@Async` 메서드 호출 시 프록시 미적용
- **해결:** 메일 발송 로직을 별도 서비스 클래스로 분리 + `@Async` 적용

##### 2. 트랜잭션 분리 문제 해결

- **문제:** 메일 발송 성공 후 로그 저장 실패 시 데이터 불일치
- **해결:** 로그 저장 트랜잭션 전파 설정을 `REQUIRES_NEW`로 분리

## 결과

JMeter 성능 테스트 결과 **응답 시간 50% 단축** 달성 및 사용자 경험 개선

## 기술적 성과 요약

시스템 아키텍처

DDD 기반 계층형 아키텍처

도메인 중심 설계로 비즈니스 로직 보호

RabbitMQ 메시지 브로커

모듈 간 강결합 해결 및 시스템 독립성 확보

마이크로서비스 설계

확장 가능한 멀티모듈 구조 구현

성능 최적화

Caffeine 캐싱 시스템

DB 부하 80% 감소

Caffeine Cache 도입

응답속도 50% 향상

비동기 처리

@Async를 통한 메일 발송 최적화

데이터 안정성

비관적 락

Race Condition 해결, 정합성 100% 보장

트랜잭션 분기

읽기/쓰기 DB 분리 대비 최적화

DB 복합 인덱스

쿼리 성능 최적화 및 유니크 제약