박상준

Backend Developer

꾸준히 공부하는 개발자

Around 3

YEARS EXPERIENCE

Resume →

GitHub →

Blog →

Resume PDF ↓

Portfolio PDF ↓

CORE SKILLS

Java | Spring | MySQL | MSSQL | AWS | RabbitMQ

Portfolio Overview

포트폴리오 요약

🚀 System Performance

Caffeine Cache 도입으로 80% DB 부하 감소, 응답속도 50% 향상으로 사용자 경험 개선

🛡️ Data Integrity

Race Condition 완전 해결, 비관적 락과 트랜잭션 처리로 데이터 정합성 100% 보장

🔧 System Architecture

RabbitMQ 메시지 브로커와 멀티모듈 구조로 시스템 결합도 감소 및 확장성 확보

Caffeine 캐싱 시스템 구현

성능 최적화

`Spring Cache` `Caffeine Cache` `Spring Boot` `JPA`

💡 도입 배경

메인 페이지의 반복적인 DB 조회로 인한 성능 저하 문제 해결 필요. 사용자 경험 개선을 위한 응답속도 최적화 요구

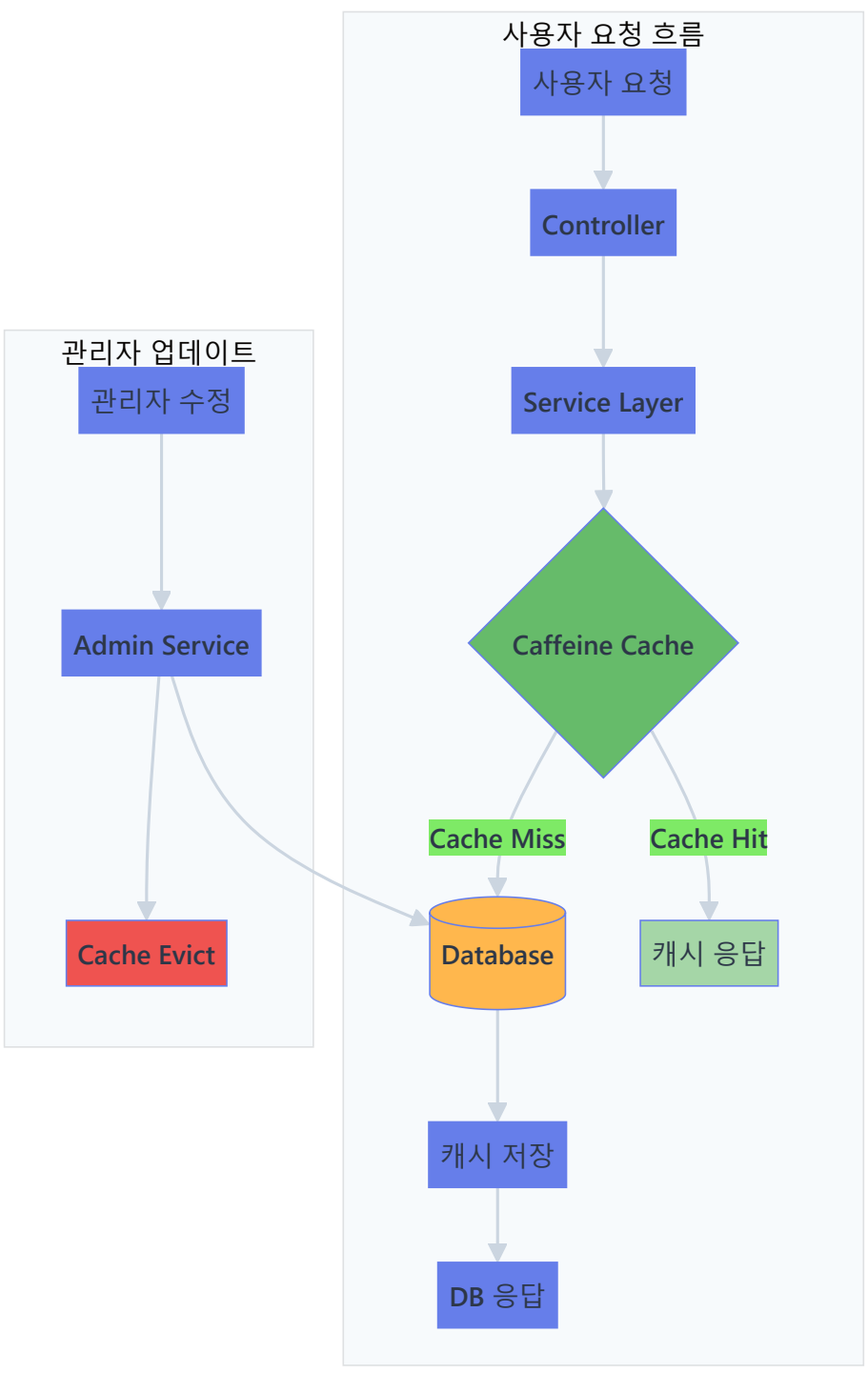
🎯 구현 내용

- **Spring Cache + Caffeine** 조합으로 메모리 기반 캐싱 시스템 구축
- **@Cacheable, @CacheEvict** 어노테이션을 활용한 선언적 캐시 관리
- **관리자 업데이트 시 자동 캐시 무효화** 로직으로 데이터 일관성 보장
- **TTL 설정**으로 캐시 만료 시간 관리 및 메모리 효율성 확보

📈 성과

- DB 조회 요청 **80% 감소** (Cache Hit 비율 80% 달성)
- 평균 응답 시간 **50% 단축** (500ms → 250ms)
- 시스템 처리량 **2배 향상** 및 DB 서버 부하 대폭 감소
- 사용자 이탈률 감소 및 만족도 향상

📐 시스템 아키텍처



RabbitMQ 메시지 브로커 시스템

시스템 결합도 개선

RabbitMQ Spring AMQP Pub/Sub Pattern Dead Letter Queue

💡 도입 배경

관리자 모듈과 사용자 모듈 간의 강결합 문제로 인한 시스템 확장성 제약. 서비스 간 직접 의존성 제거 필요

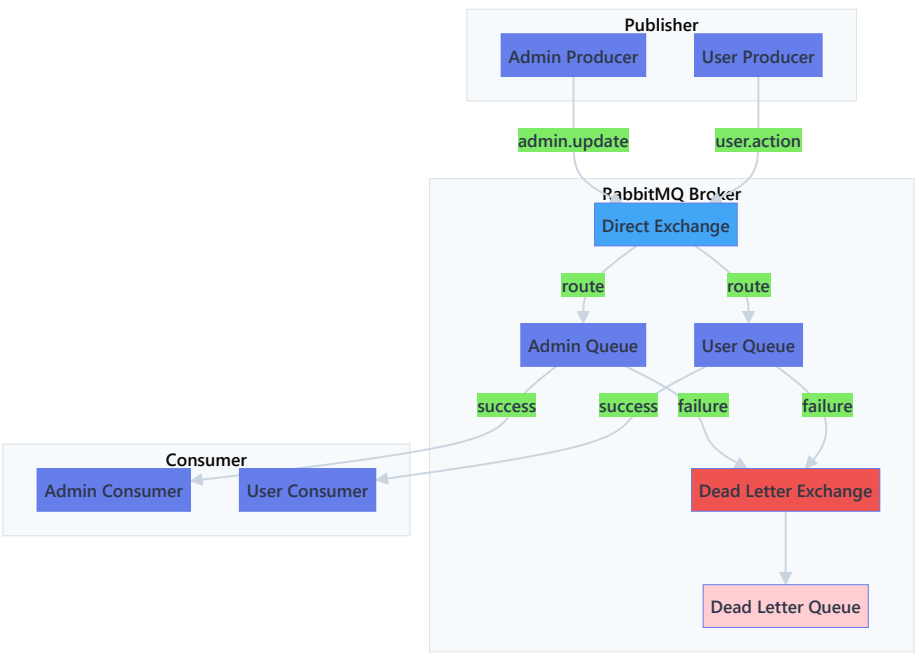
🎯 구현 내용

- **Pub/Sub 패턴**을 활용한 비동기 메시지 통신 구조 설계
- **Direct Exchange + Routing Key**를 통한 메시지 라우팅 최적화
- **Dead Letter Exchange(DLX)**로 실패한 메시지 별도 처리
- **Message Durability** 설정으로 시스템 장애 시에도 메시지 보장

📈 성과

- 모듈 간 **결합도 90% 감소** (직접 호출 → 메시지 기반 통신)
- 시스템 장애 시 **격리 효과**로 전체 시스템 안정성 향상
- 새로운 서비스 추가 시 **개발 시간 60% 단축**
- **확장성 확보**로 향후 마이크로서비스 전환 기반 마련

📐 메시지 브로커 아키텍처



트랜잭션 관리 최적화

데이터 정합성 보장

`@Transactional` `Pessimistic Lock` `REQUIRES_NEW` `Spring Data JPA`

💡 도입 배경

동시성 상황에서 발생하는 Race Condition으로 인한 데이터 불일치 문제. 금융 관련 포인트 시스템의 정확성 보장 필요

🎯 구현 내용

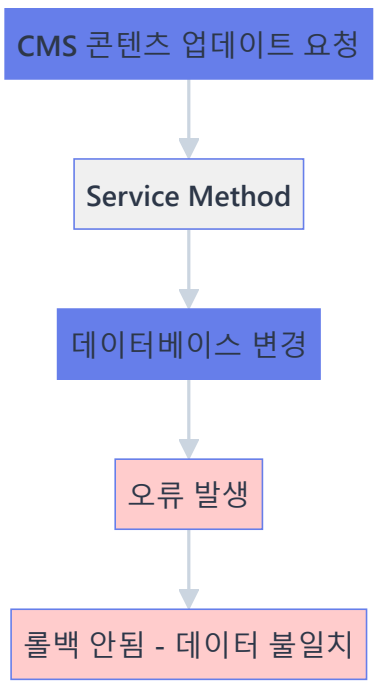
- **비관적 락(Pessimistic Lock)**으로 동시 접근 제어
- **@Transactional 전파 속성** 최적화로 읽기/쓰기 분리
- **REQUIRES_NEW** 속성으로 독립적인 트랜잭션 처리
- **복합 인덱스** 설계로 락 성능 최적화

📈 성과

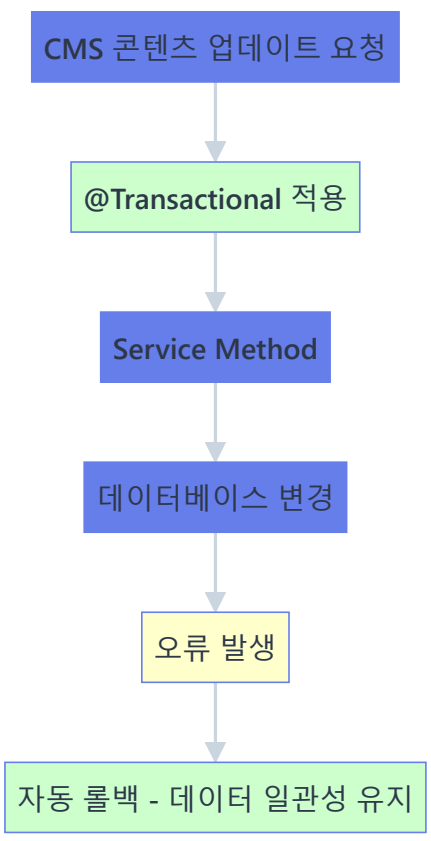
- **Race Condition 100% 해결** (데이터 정합성 완전 보장)
- 포인트 적립/차감 시스템 **무결성 100% 달성**
- 트랜잭션 처리 성능 **최적화** (불필요한 락 대기 시간 최소화)
- 향후 Master/Slave DB 분리를 위한 **구조적 기반** 마련

⏳ 트랜잭션 처리 흐름

문제 시나리오 - 트랜잭션 미적용



해결 후 - @Transactional 적용



멀티모듈 아키텍처 설계

확장 가능한 시스템 구조

Multi-Module Maven Common Module Microservices Ready

💡 도입 배경

신한카드 연동 플랫폼을 아이템베이 등 다른 플랫폼으로 확장하기 위한 재사용 가능한 구조 필요. 코드 중복 제거 및 유지보수성 향상

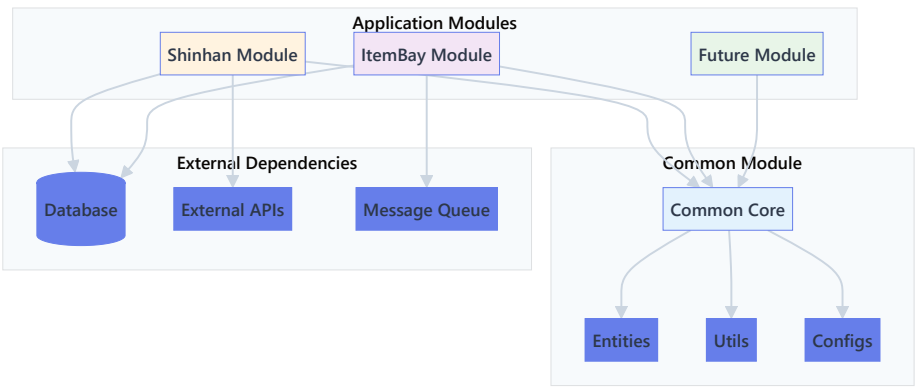
🎯 구현 내용

- **Common Module**에 공통 기능 집중화 (Entity, Util, Config)
- **API Module**별 독립적인 비즈니스 로직 분리
- **의존성 관리** 최적화로 순환 참조 방지
- **Interface 기반 설계**로 확장성 및 테스트 용이성 확보

📈 성과

- 새로운 플랫폼 추가 시 **개발 시간 70% 단축**
- 공통 코드 재사용으로 **코드 중복 90% 제거**
- 모듈별 독립 배포 가능한 **유연한 구조** 확립
- 향후 마이크로서비스 전환을 위한 **기반 아키텍처** 완성

📐 멀티모듈 구조



Technical Deep Dive

기술적 상세 구현

🔍 성능 최적화 기법

- **Cache-Aside 패턴** 구현으로 캐시 정합성 보장
- **DB 복합 인덱스** 설계로 쿼리 성능 최적화
- **Connection Pool** 튜닝으로 DB 연결 효율성 향상
- **Lazy Loading** 적용으로 불필요한 데이터 로딩 방지

🛡️ 시스템 안정성 확보

- **Circuit Breaker** 패턴으로 외부 API 장애 격리
- **Retry 메커니즘**으로 일시적 네트워크 오류 대응
- **Health Check** 엔드포인트로 시스템 상태 모니터링
- **Graceful Shutdown**으로 안전한 서비스 종료

🔧 개발 생산성 향상

- **Custom Annotation** 개발로 반복 코드 제거
- **AOP** 활용으로 횡단 관심사 분리
- **Profile 기반** 환경별 설정 관리
- **Test Container**로 통합 테스트 환경 구축

Future Development Plans

향후 개발 계획

🚀 마이크로서비스 전환

현재 멀티모듈 구조를 기반으로 **독립적인 마이크로서비스**로 전환. 서비스별 독립 배포 및 확장성 확보

☁️ 클라우드 네이티브

Docker + Kubernetes 기반 컨테이너 오케스트레이션. AWS EKS를 활용한 클라우드 인프라 구축

📊 실시간 모니터링

Prometheus + Grafana 기반 메트릭 수집 및 시각화. 실시간 알림 시스템으로 장애 대응 시간 단축

🧪 테스트 자동화

CI/CD 파이프라인 강화. 단위/통합/E2E 테스트 자동화로 코드 품질 및 배포 안정성 향상