약 4년 경력 · JAVA/KOTLIN·SPRING BACKEND ENGINEER

백엔드 개발자

홍철민

상태가 복잡한 백엔드 문제를 구현과 검증으로 해결합니다.

약 4년간 Java/Kotlin·Spring으로 결제, 비동기 메시징과 AWS 배포를 구현했습니다. 현재는 성능 측정과 Python·RAG·MCP 기반 AI 서비스 백엔드로 경험을 확장하고 있습니다.

Java · Kotlin | Spring | MySQL · Redis | AWS | Python · RAG · MCP

01	**호두랩스**	미완료 PURCHASED 주문을 Redis TTL 안에서 식별·재처리하고, 여러 WAS에서 복구 스케줄러가 동시에 실행되는 것을 ShedLock으로 제한했습니다.
02	**KExcel — Kotlin DSL 기반 대용량 엑셀 생성 라이브러리**	FastExcel DSL 오버헤드를 16.3%에서 3.1%로 줄이고 512MB heap에서 100만 행 생성을 완료했으며, JitPack 0.1.0으로 공개했습니다.
03	**WorkShield Web — 실패 경계를 설계한 AI 서비스 백엔드**	검증된 근거의 실제 ID를 백엔드가 결합하도록 바꾸고, 고정 합성 fixture 16회 최종 gate를 통과한 모델을 답변·Suggestions 후보로 선정했습니다.

github.com/Hong1008 velog.io/@hong1008/posts https://hong1008.github.io/portfolio/

호두랩스 통합 결제

외부 결제 승인과 내부 재화 제공이 한 번에 끝나지 않을 수 있어, 승인 이후 내부 처리가 완료되지 않은 주문을 구분할 필요가 있었습니다.

2021.02–2021.08 | 결제 API 및 복구 배치 개발 | Java · Spring Boot · MySQL · Redis · ShedLock

REQUESTED 주문·주문키 생성 → 외부 승인 → **PURCHASED** 승인 DB 반영 → 재화 제공 → **COMPLETED** 내부 처리 완료

핵심 판단

주문을 REQUESTED·PURCHASED·COMPLETED로 나누고, PURCHASED 상태와 Redis 정보가 남은 주문만 복구 배치가 재처리하도록 했습니다.

구현 결과

미완료 PURCHASED 주문을 Redis TTL 안에서 식별·재처리하고, 여러 WAS에서 복구 스케줄러가 동시에 실행되는 것을 ShedLock으로 제한했습니다.

역할 경계

애플리케이션 결제 흐름을 구현했으며, Stored Procedure와 DB 처리 구조는 DBA와 협업했습니다.

직접 담당

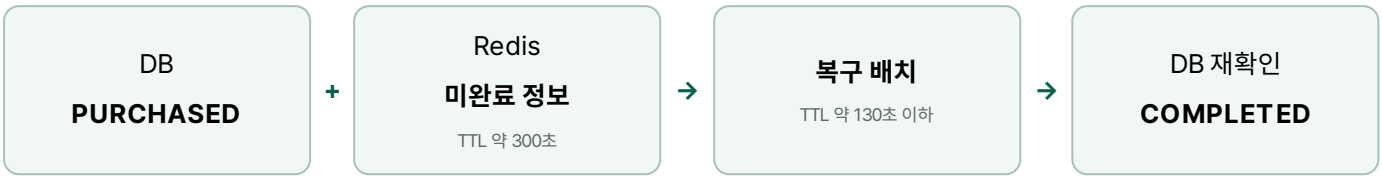
- PG 결제 API 구현
- Google Play·Apple 인앱 결제 구현
- 결제 상태 흐름 구현
- 미완료 주문 복구 배치 구현

검증 범위

주문 상태, Redis TTL과 DB 제약이 담당하는 범위를 대조해 정상 처리와 재처리 조건, 처리하지 못하는 실패를 구분했습니다.

복구 흐름과 책임 범위

외부 승인이 끝났다는 사실과 내부 재화 제공이 끝났다는 사실을 분리해, 재처리가 가능한 실패만 복구 대상으로 삼았습니다.



복구 가능한 범위

- PURCHASED 저장 후 재화 제공이 실패한 주문
- Redis 미완료 정보가 TTL 안에 남은 주문
- DB 상태 재확인 후 완료 처리가 가능한 주문

이 흐름으로 복구하지 못하는 범위

- PG 승인 후 DB의 PURCHASED 저장 자체가 실패하면 Redis와 DB 상태가 REQUESTED에 남을 수 있어 이 흐름으로 복구할 수 없습니다.
- 서비스 중단이 Redis TTL보다 길거나 캐시가 유실되면 복구 대상 정보를 찾을 수 없습니다.
- 외부 결제 상태와 내부 DB를 정기적으로 대조하는 reconciliation은 구현 범위에 없었습니다.

기술적 판단

스케줄러 중복 실행과 주문 멱등성을 같은 문제로 다루지 않았습니다.

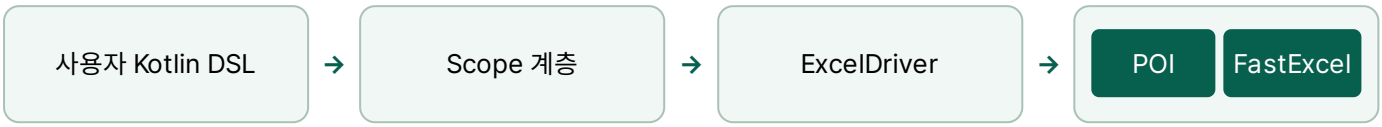
ShedLock은 여러 WAS의 복구 스케줄러가 동시에 실행되는 범위만 제한합니다. 전체 주문 처리나 외부 승인의 Exactly-once를 보장하는 장치로 확대하지 않습니다.

확인한 한계

PG 승인 후 PURCHASED 저장 자체가 실패하거나 Redis 정보가 유실·만료되면 이 복구 흐름만으로 처리할 수 없습니다.

KExcel — Kotlin DSL 기반 대용량 엑셀 생성 라이브러리

Apache POI의 저수준 API를 직접 사용하면 객체와 좌표, 스타일 생명주기를 반복해서 관리해야 했고, 엔진을 바꾸면 보고서 코드도 함께 변경해야 했습니다.



구조와 안전 제약

- 동일 DSL과 엔진 구현을 ExcelDriver로 분리
- 행 단위 스트리밍과 처리 완료 행 재접근 차단
- 동일 시트 동시 쓰기 오용을 Fail-Fast로 감지
- 엔진 고유 기능은 Native Extension으로 노출

측정 환경

OpenJDK 21, 512MB heap, G1GC, JMH 1.36에서 Native API와 DSL을 같은 데이터 규모로 비교했습니다.

> Thread-safe 쓰기를 지원한다는 뜻이 아니라 데이터 손상 전에 오용을 드러내는 정책입니다.

FastExcel DSL 오버헤드
3.1%
Native API 대비

100만 행 생성
3.396s
FastExcel · 512MB heap

POI 병합 처리량
3.36 → 17.42
ops/s · 병합 시나리오

트레이드오프

병합 중복 검증을 우회한 최적화는 잘못된 병합 요청을 사전에 방지해야 하며, 병합 메타데이터의 메모리 비용은 남습니다.

WorkShield Web — 실패 경계를 설계한 AI 서비스 백엔드

계약서 비교 MCP를 FastAPI 웹 API와 LLM 설명 계층, AWS 배포 환경에 연결했습니다. 검토 상태와 질문 대상·출처를 백엔드가 관리하고, API와 배포 구조의 자동 검증을 구성했습니다.

결정론적 상태 경로

FastAPI

멱등 요청 → Review 상태 → 근거·출처 결합

↔

외부 실행 경계

MCP · LLM

검색·분류와 근거 범위의 자연어 생성

중복과 상태 경쟁

검토 상태 규칙을 한곳에 모으고, 같은 요청의 재전송·중복 생성·동시 변경을 각각 다른 저장 규칙으로 막았습니다.

> 단일 실행 프로세스·SQLite 범위이며 분산 환경에서 한 번만 실행됨을 보장하지 않습니다.

모델 책임 경계

질문 대상과 사용할 근거를 백엔드가 먼저 확정하고, 모델은 출처 종류만 선택하게 했습니다. 실제 출처 ID는 백엔드가 검증한 근거에 결합합니다.

합성 데이터 최종 통과

16 / 16

8개 × 2회 · 보정 포함

모델 응답 시간 p50

7.102초

동시 요청 1개

모델 응답 시간 p95

16.416초

고정 합성 검증 데이터

구현 결과

검증된 근거의 실제 ID를 백엔드가 결합하도록 바꾸고, 고정 합성 fixture 16회 최종 gate를 통과한 모델을 답변·Suggestions 후보로 선정했습니다.

> 제한 조건 · 파일형 데이터베이스와 단일 실행 프로세스·단일 서버를 사용하는 출시 전 구조로, 여러 서버의 동시 실행과 중단 지점 자동 재개를 지원하지 않습니다.

경력 요약

Java·Kotlin과 Spring 기반 애플리케이션 개발에서 시작해 결제, 비동기 메시징, 도메인 분리와 배포 환경을 다뤘습니다.

2022.11–2023.08
샤플앤컴퍼니
백엔드 개발

배치 서버와 메일 서버를 SQS로 분리하고 발송 요청과 메시지 소비 단계에 서로 다른 DB UNIQUE 제약을 적용했습니다.

온보딩 메일 구조를 인보이스 발송에도 재사용했고, 점검 보고서의 라이브러리 종속 코드를 Kotlin DSL 공통 빌더로 분리했습니다.

2021.09–2022.10
휴니크
개발 리드·백엔드·인프라

기존 구조를 그대로 신규 시스템에 복제하지 않고 도메인별 책임을 분리한 별도 Spring 시스템을 개발했습니다. 기존 회원은 기존 인증 서비스와 연결했습니다.

주요 어드민·키오스크 API와 S3·CloudFront 정적 제공 환경, Jenkins에서 ECR·ECS Fargate로 이어지는 백엔드 배포 흐름을 구현했습니다.

2020.02–2021.02
지투이
애플리케이션·백엔드 개발

화면별로 반복하지 않도록 인증·세션·전역 오류·감사 로그를 공통 기능으로 분리하고 Spring Boot API와 연결했습니다.

인증·세션·전역 오류·감사 로그를 화면 공통 경로로 재사용하고, 포탈 사용자의 메뉴별 접근 범위를 권한에 따라 구분했습니다.

> **경력 사례 공개 원칙**
> 회사 내부 코드와 실제 사용자 데이터는 포함하지 않았으며, 운영·출시 상태와 개인·팀의 책임 범위를 확인 가능한 수준으로 제한했습니다.

추가 프로젝트

AI 보조 개발의 범위 통제

WorkShield — 표준계약서 비교 MCP

AI 제안으로 검색 도구가 검증하기 어려운 상세 비교까지 커진 뒤, 실제 실행 기록과 평가 가능성을 기준으로 상세 비교를 제외하고 목표·비목표를 다시 정했습니다.

판단

실제 판정 후보와 점수를 보존하고, 실험 전에 가설·채택·보류·중단 조건과 되돌릴 기준선을 기록했습니다.

역할과 한계

PM·MCP 구현을 담당했습니다. 이 과정은 도구 사용 자체의 성과가 아니라 문제 정의·검증·중단 판단의 책임을 사람이 소유한 사례입니다.

SK Networks Family AI 캠프 팀 프로젝트

영화 흥행 예측 및 배급 시뮬레이터

개봉일 기준 시간 분할 평가에서 Stage 1 R^2 0.5638, Stage 2 R^2 0.5517을 기록하고 두 단계 예측 결과를 Streamlit 데모로 연결했습니다.

역할

PM·2-Stage 모델 설계

검증

2010–2025년 2,489편·28개 컬럼 데이터를 개봉일 기준으로 시간 분할해 각 Stage의 R^2와 RMSE를 기록했습니다.

SK Networks Family AI 캠프 팀 프로젝트

전기차 충전 인프라 SOS 대시보드

17개 시·도의 전기차 등록 대수와 충전기 수를 한 흐름에서 검증·적재하고, 지역별 비교 결과를 지도와 차트로 확인할 수 있게 했습니다.

역할

백엔드·대시보드 구현

검증

17개 시·도 단위로 전기차 등록 대수와 충전기 수를 바탕으로 계산한 불편지수를 지도와 차트로 확인했습니다.

상세 자료

각 사례의 구조, 측정 조건, 근거와 전체 한계는 HTML 포트폴리오에서 확인할 수 있습니다.

포트폴리오
https://hong1008.github.io/portfolio/

GitHub
https://github.com/Hong1008

기술 블로그
https://velog.io/@hong1008/posts

이메일
zxcbnm6404@gmail.com