길현준

Backend Engineer / Platform Part Leader

kboxstar@gmail.com GitHub ↗ LinkedIn ↗ 웹 포트폴리오 ↗

개발 운영 하네스 프레임워크

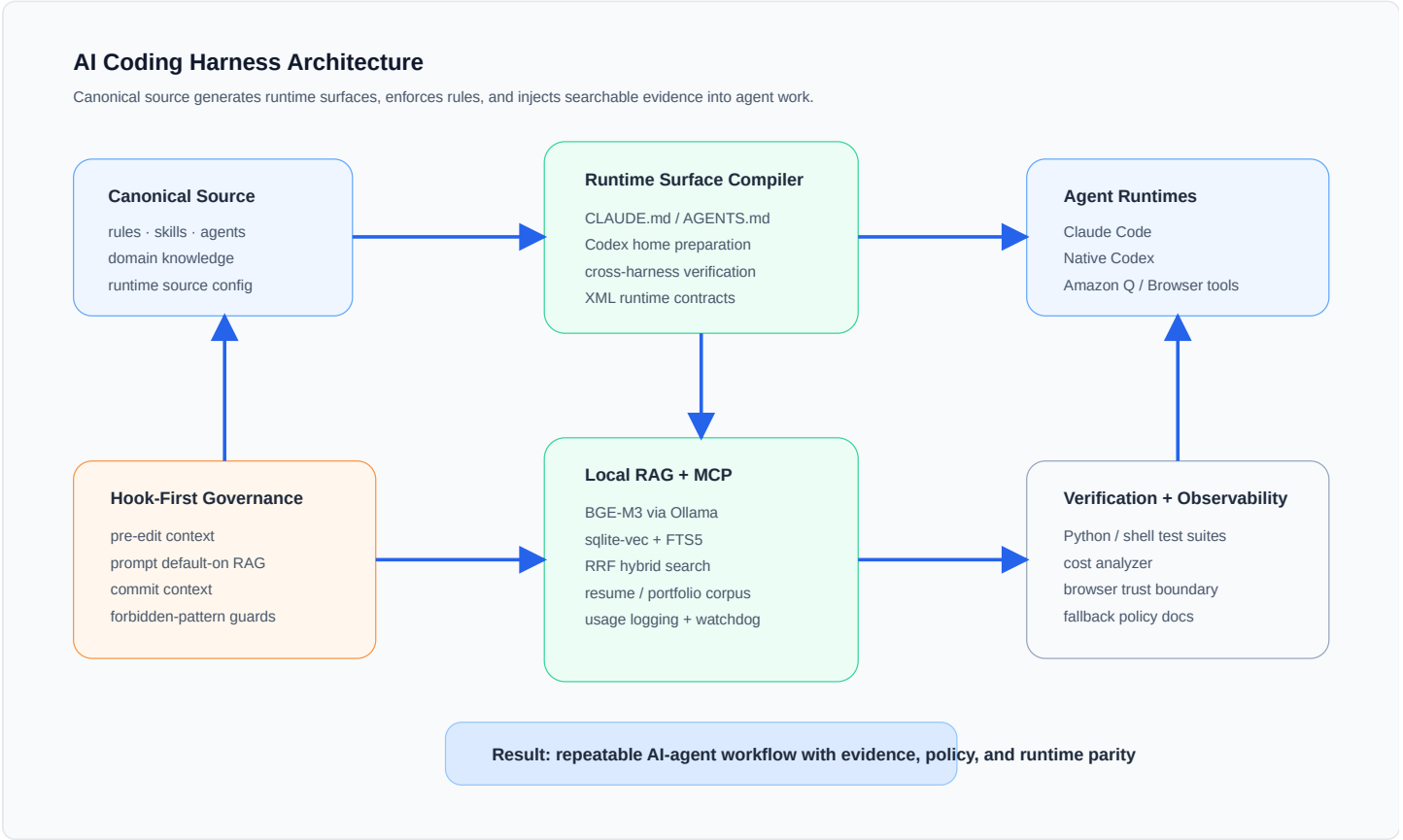
2026.04 — 진행 중 · 개인

GitHub ↗ 상세 ↗

3개 하네스 · RAG MVP · JD feedback loop

개발 운영과 이력서/JD 큐레이션을 재현 가능하게 만든 Git 기반 개인 하네스 — 로컬 BGE-M3 RAG, MCP 검색, Git 훅 품질 게이트, JD-custom PDF 생성, 직행/Zighang 외부 피드백 루프와 안전한 T1/T2/T3 반영 체계를 통합

TypeScript Python Bash MCP sqlite-vec FTS5 Ollama BGE-M3 Vector RAG Git Hooks Runtime Surface Compiler Quality Gates



성과 지표	이전	이후
RAG MVP 인덱스	manual context search	**548 files / 3,347 chunks / 20.7 MB** (0.7s warm query)
RAG default-on 검증	opt-in trigger	**15/15 tests across 3 harnesses** (prompt >=12 chars auto-search)
Resume RAG 코퍼스	0 indexed chunks	**109 chunks** (32 tests passed)
Work-data 증거 아카이브	manual evidence curation / 44% metric coverage	**279 incremental work items / schema v1.2** (50% metric coverage)
JD 피드백 루프	manual JD feedback capture	**JD PDF + Zighang recruitment-roast + T1/T2/T3 triage + form-start guard** (Juvis pass range / Nexon low-fit / Hanwha 82% improve-to-pass)

문제 해결 과정

개발 런타임별 컨텍스트와 규칙이 문서·설정·로컬 훔에 흩어져 실행 환경마다 드리프트 발생

하네스 소스를 canonical source로 두고 런타임 서피스를 자동 생성, XML runtime contract와 cross-harness diff 검증 추가

→ **kh/gp/gd 3개 하네스의 생성 문서 및 런타임 컨트랙트 정합성 확보**

마크다운 지식 베이스가 커져 에이전트가 관련 컨텍스트를 매번 수동 탐색

sqlite-vec + FTS5 하이브리드 RAG, BGE-M3 로컬 임베딩, RRF fusion, MCP 검색 서버, 마크다운 변경 후 자동 재색인 구현

→ **548 파일·3,347 청크·20.7 MB 인덱스, warm 쿼리 0.7초, 골드 쿼리 5건 top-2/3 적중**

RAG가 opt-in 참고 도구라 실제 조사·검토 프롬프트에서 누락

UserPromptSubmit 기본 트리거로 전환하고 프롬프트 12자 이상이면 search_harness가 자동 호출되도록 gp/gd까지 포팅

→ **3개 하네스에서 15/15 default-on 테스트 통과, 실 프롬프트 smoke test 적중**

이력서·포트폴리오 JSON과 JD별 외부 피드백이 서로 분리되어, 공고별 개선이 raw feedback에 의존

resume/portfolio JSON chunker, JD-custom PDF 생성기, 직행/Zighang recruitment-roast 수집기, T1/T2/T3 triage 템플릿을 연결

→ 쥬비스 NestJS JD 통과권, 넥슨 정산 JD 35% low-fit, 한화생명 AI Backend 82% 보완-통과권 피드백을 각각 raw feedback→T1/T2/T3 triage→검증 루프로 분류

이력서·JD 커스터마이징 근거가 세션 로그와 작업 기록에 흩어져 최신 성과 선별과 정량화가 매번 수작업으로 반복됨

private work-data 아카이브를 schema v1.2로 정리하고 outcome_metrics 필드를 도입, Jira·세션 evidence를 resumeProject 단위로 분류해 resume/portfolio/RAG 큐레이션 경로에 연결

→ 279개 작업 항목 증분 수집, 메트릭 보유율 44%→50% 개선, work-data→이력서 후보 선별→검증 루프 정착

기술 선택 근거

- 클라우드 벡터 DB 대신 로컬 BGE-M3 + sqlite-vec 선택: 마크다운 변경 직후 자동 재색인과 비용 통제를 우선
- 문서 규칙 대신 hook-first 강제 선택: 에이전트가 필요 시점에 문서를 읽지 않는 문제를 런타임 차단으로 보완
- 런타임별 수동 설정 대신 runtime surface compiler 선택: 생성물 드리프트를 줄이고 3개 하네스 정합성 검증을 자동화
- 검색 opt-in 대신 default-on 선택: 조사·리뷰·편집 같은 실제 프롬프트에서 근거 검색 누락을 줄이기 위해 UserPromptSubmit에 연결

주요 내용

- 런타임 서피스 자동 생성 + XML runtime contract로 3개 하네스 정합성 확보
- 로컬 BGE-M3 RAG로 548 파일·3,347 청크 색인, warm 쿼리 0.7초 달성
- RAG default-on 전환으로 프롬프트 12자 이상 search_harness 자동 호출, 3개 하네스 15/15 테스트 통과
- resume/portfolio JSON을 RAG 코퍼스로 통합, 109 청크 색인 + 32개 테스트 통과
- 브라우저·REPL 런타임 신뢰 경계와 fallback 순서 문서화, 3개 하네스 설정 반영
- work-data schema v1.2와 outcome_metrics 기반 증거 큐레이션으로 279개 작업 항목 증분 수집·메트릭 보유율 44%→50% 개선
- JD-custom PDF + 직행/Zighang 피드백 루프 구축, 쥬비스 92%→88% 통과권·넥슨 35% low-fit 분류로 과최적화 방지
- 직행/Zighang recruitment-roast 헬퍼를 main-scope 결과 저장·Puppeteer 실제 click·upload-form guard로 보강해 sidebar 과거 결과 오탐 방지

깨달은 점

- AI 지원 개발 생산성은 프롬프트 품질만으로 유지되지 않고, 런타임 컨텍스트·규칙 강제·증거 검색·비용 관측성을 같은 운영 경로로 묶어야 재현 가능하다는 점을 체득
- 문서 규칙은 필요 시점에 읽히지 않으면 정책이 아니라 참고 자료에 머물기 때문에, 중요한 규칙은 훅·테스트·컴파일러 출력으로 강제해야 한다는 기준 정립

콘서트 예약 서비스

2024.10 — 2024.11 · 개인

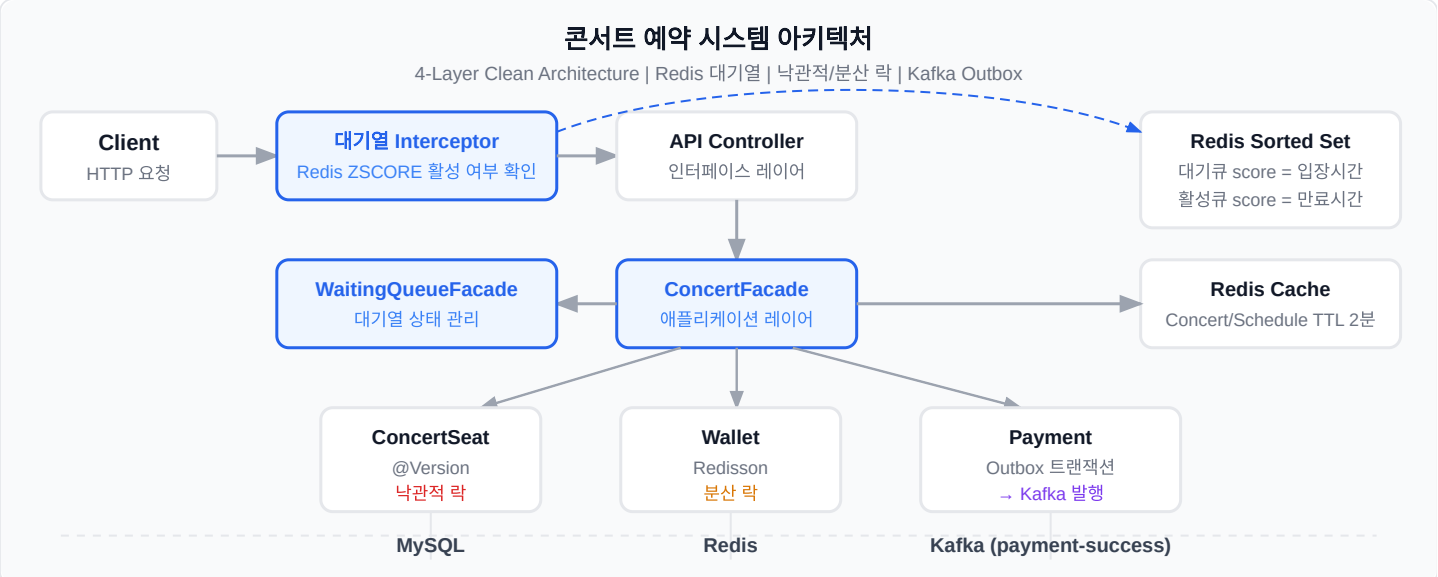
GitHub ↗ 상세 ↗ JPA 비관적 락과 낙관적 락 및 재시도 ↗ Spring Boot 콘서트 예약 시나리오 동시성 문제 분석 ↗ Spring Boot Redis를 활용한 분산 락 구현 ↗

캐시(Cache)와 캐싱 전략(Caching Strategy) ↗

1만 동시 요청 · K6 3,000 TPS · Kafka 3-broker · Prometheus+Grafana

1만 동시 요청 환경에서 좌석 선점·결제·포인트 충전의 동시성 문제를 시나리오별 하이브리드 락 전략으로 해결한 콘서트 좌석 예약 서비스 — 낙관적 락 + Redisson 분산 락, Redis 캐싱 TPS 15배 향상, Kafka 이벤트 드리븐 아키텍처, K6 부하 테스트 기반 병목 개선

Java | Spring Boot | JPA | Redis | MySQL | Kafka | Docker | K6



성과 지표	이전	이후
좌석 예약 응답시간	1,678ms	**835ms** (-50%)
좌석 조회 TPS	100	**1,500** (15x)
DB 쿼리 비율	100%	**6%** (-94%)

문제 해결 과정

1만 동시 요청 환경에서 좌석 선점 시 동일 좌석 이중 배정 위험
비관적/낙관적/분산 락 3종을 K6로 비교 검증 후, 단일 좌석 경합에 최적인 @Version 낙관적 락 선택
→ **좌석 예약 응답시간 50% 개선 (1,678ms → 835ms), 이중 배정 0건**

포인트 충전과 결제의 동시 실행 시 잔액 정합성 붕괴
재시도 기반 낙관적 락의 무한 루프 위험을 분석하고, Redisson 분산 락(userWalletLock:{userId})으로 사용자 단위 직렬화
→ **동시 충전/결제 시나리오에서 잔액 정합성 100% 보장**

콘서트/좌석 조회 TPS가 100에 불과하여 예약 오픈 시 DB 병목
Redis 캐시 레이어 적용 + DB 인덱스 최적화를 병행하여 계층적 캐싱 전략 구현
→ **TPS 100 → 1,500 (15배 향상), DB 쿼리 94% 감소**

예약 완료 후 결제/알림 처리가 동기 호출로 묶여 도메인 간 강결합
Kafka Transactional Outbox 패턴으로 예약 TX 내 outbox_event 삽입 후 스케줄러가 비동기 발행
→ **도메인 간 결합도 제거, 메시지 유실 방지, 독립 확장 가능**

기술 선택 근거
• 좌석 선점에 @Version 낙관적 락: 단일 좌석 경합에서 비관적 락 대비 처리량 우위, K6 1만 VU 검증
• 결제·충전에 Redisson 분산 락(userWalletLock:{userId}): 잔액 정합성 필수 트랜잭션에서 재시도 기반 낙관적 락의 무한 루프 위험 회피
• Redis Sorted Set 대기열: 스코어=Unix timestamp로 입장 순서+만료 시간 이중 활용, DB 대기열 대비 O(log N)
• Transactional Outbox 패턴: 결제 TX 내 outbox_event 행 삽입 → 스케줄러가 Kafka 발행, 메시지 유실 방지

주요 내용
• 낙관적 락으로 좌석 예약 응답 시간 50% 개선 (1,678ms → 835ms)
• 시나리오별 하이브리드 락 전략 설계 (낙관적 락 + Redisson 분산 락)
• Redis 캐싱으로 TPS 15배 향상 (100 → 1,500), DB 쿼리 94% 감소
• Kafka 이벤트 드리븐 아키텍처로 도메인 간 결합도 제거
• K6 부하 테스트 기반 성능 병목 식별 및 개선

깨달은 점
• 비관적/낙관적/분산 락의 트레이드오프를 비교하며 동시성 제어의 본질적 차이를 이해
• 시나리오별 하이브리드 락 전략 설계로 단일 솔루션이 아닌 맥락 기반 선택의 중요성 학습

Ledgerly

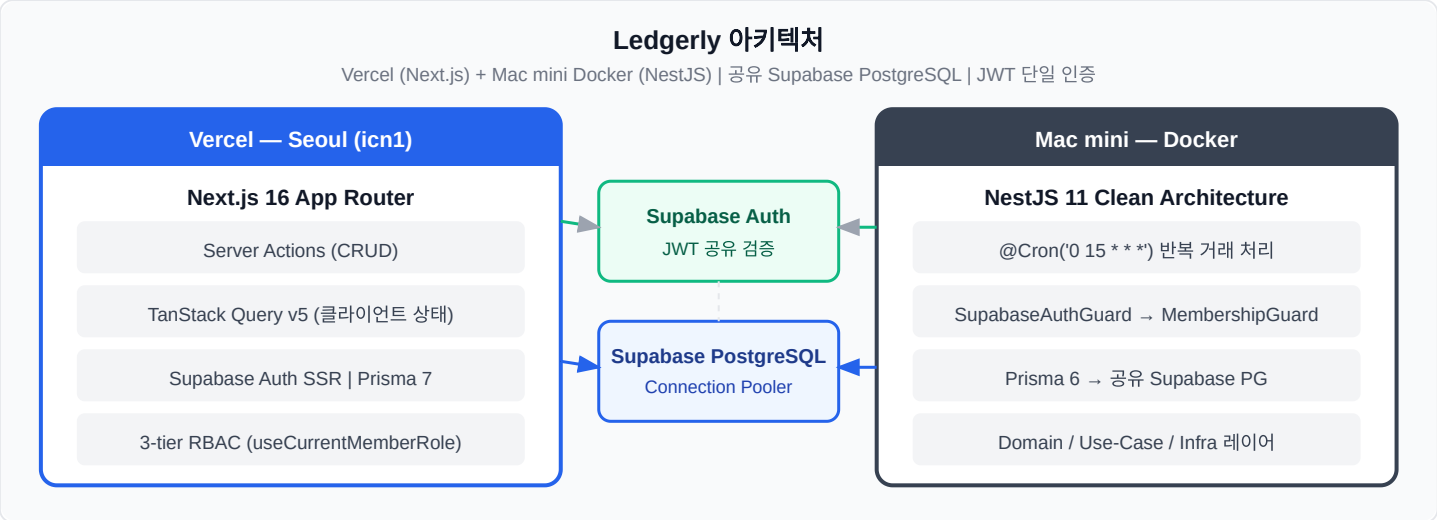
2025.08 — 진행 중 · 개인

상세 ↗

가족/조직 단위 멀티테넌시 · 3단계 RBAC · FE+BE 분리 배포

가족·조직 단위 공유 가계부 앱 — Next.js 16 풀스택, Supabase PostgreSQL, 3단계 역할 기반 접근 제어(OWNER/ADMIN/MEMBER), NestJS Cron 정기거래 자동 처리, 99.45% 라인 커버리지 단위 테스트 + 300건 이상 E2E 통합 테스트

Next.js | React | TypeScript | Supabase | PostgreSQL | Prisma | NestJS | Tailwind CSS | TanStack Query | Zod | Vitest | Playwright | Docker | Vercel



성과 지표	이전	이후
Vitest 단위 테스트	-	**397건 (BE 25 + FE 31 suites)** (99.45% 커버리지)
Playwright E2E	-	**300건+ (24 specs · 5 projects)** (역할별 시나리오)

문제 해결 과정

가족/조직 내 역할별 권한 분리 없이 공유 가계부 운영 시 데이터 보안 위험
OWNER/ADMIN/MEMBER 3단계 RBAC 설계, SupabaseAuthGuard → MembershipGuard → SuperAdminGuard 3단계 Guard 체인으로 인증+인가 이중화
→ **조직별 독립 데이터 관리 + 역할별 세분화된 접근 제어 실현**

Vercel Hobby 플랜 Cron 제한(1일 1회)으로 정기거래 자동 처리 불가
NestJS 기반 별도 백엔드를 Mac mini에서 Docker로 분리 운영, @nestjs/schedule로 매일 15:00 KST 실행
→ **정기거래 매일 자동 처리 실현, 플랫폼 제한 우회**

SSR 환경에서 서버/클라이언트 컴포넌트 간 인증 상태 불일치
Next.js 16 App Router + Supabase Auth SSR 조합으로 쿠키 기반 세션 복원 + 인증 컨텍스트 전파 패턴 설계
→ **서버/클라이언트 컴포넌트 간 원활한 인증 상태 동기화**

복잡한 RBAC + 비동기 로직의 높은 결함 위험을 수동 테스트로 커버 불가
Vitest 단위 테스트 + Playwright E2E를 역할별(Admin/Member) 시나리오로 구성, 실제 BE + 로컬 Supabase 연동 풀스택 통합 테스트 환경 구축
→ **라인 커버리지 99.45%, 역할별 시나리오 자동 검증 체계 구축**

기술 선택 근거

- NestJS Docker 분리: Vercel Hobby 플랜 Cron 제한(1일 1회) 우회, 정기거래 매일 15:00 KST 실행 보장을 위해 Mac mini에서 독립 운영
- Supabase Auth + 앱 레벨 RBAC 이중화: SupabaseAuthGuard(JWT 검증) → MembershipGuard(역할 확인) → SuperAdminGuard(관리자 전용) 3단계 Guard 체인
- Prisma + Supabase PG: Prisma의 타입 안전성과 마이그레이션 도구 활용, Supabase 커넥션 풀러 경유로 서버리스 환경 커넥션 관리

주요 내용

- Vitest 397건 단위 테스트, 라인 커버리지 99.45% 달성
- Playwright 300건+ E2E 테스트, 역할별 시나리오 분리
- OWNER/ADMIN/MEMBER 3단계 RBAC + Supabase RLS 적용
- NestJS Cron 정기거래 자동 처리 (Docker 배포)
- 풀스택 통합 테스트 환경 구축 (FE+BE+Supabase)

깨달은 점

- Next.js 16 App Router와 Supabase Auth를 조합하여 SSR 환경에서의 인증 상태 관리와 Row Level Security 기반 데이터 접근 제어를 구현하며, 서버·클라이언트 컴포넌트 간 인증 컨텍스트 전파 패턴을 학습
- OWNER/ADMIN/MEMBER 3단계 역할 체계를 설계하며 조직 단위 멀티테넌시에서의 권한 분리와 데이터 격리 전략을 경험

대신물류 배차현황 챗봇

2026.01 — 진행 중 · 개인

GitHub ↗ 상세 ↗

월~토 06:00~20:00 매시 크롤링 · Blue-Green 무중단 배포 · SQLite 단일 서버

대신물류 배차현황 데이터를 Cheerio로 크롤링하고 카카오톡 챗봇 스킬서버와 Next.js 모바일웹으로 조회할 수 있는 서비스 — Clean Architecture + TSyringe DI, Express 5, Prisma SQLite, Traefik Blue-Green 무중단 배포

TypeScript Express Next.js React Prisma SQLite Cheerio TSyringe TanStack Query Recharts Docker Traefik Tailwind CSS Vitest

성과 지표	이전	이후
크롤링 주기	수동 조회	**일 15회 자동** (자동화)
배포 다운타임	수동 재시작	**0초** (Blue-Green)

문제 해결 과정

물류 회사 배차현황을 웹사이트에서 수동 확인해야 하는 운영 비효율
Cheerio + Axios로 EUC-KR 인코딩 배차 데이터 크롤링, node-cron으로 월~토 06:00~20:00 매시 자동 동기화
→ **수동 조회 → 일 15회 자동 크롤링, 실시간 데이터 제공**

현장 직원이 PC 없이 모바일로 배차 정보를 조회할 수단 부재

카카오 i 오픈빌더 스킬서버 프로토콜 직접 구현 + Next.js 반응형 모바일 웹 + Recharts 통계 대시보드 병행 제공

→ **카카오톡 챗봇 + 모바일 웹 이중 채널로 현장 즉시 조회 가능**

크롤러/카카오 API/DB 등 외부 의존성 변경 시 비즈니스 로직까지 수정 필요

Clean Architecture로 도메인/애플리케이션/인프라 계층 분리, TSyringe DI 토큰으로 인터페이스 바인딩, Value Object 패턴으로 도메인 규칙 타입 수준 강제

→ **외부 의존성 교체 시 인프라 계층만 수정, 비즈니스 로직 무변경**

수동 재시작 방식의 배포로 서비스 다운타임 발생

Traefik 파일 프로바이더 기반 Blue-Green 배포 설계, deploy.sh로 빌드 → 헬스체크 → YAML 재작성 → 트래픽 전환 자동화

→ **배포 다운타임 0초 달성, Docker 소켓 노출 없이 보안성 확보**

기술 선택 근거

- Clean Architecture + TSyringe DI: 크롤러·카카오 API·DB 등 외부 의존성 교체 가능성을 고려한 계층 분리, DI 토큰으로 인터페이스 바인딩
- SQLite 선택: 단일 서버·읽기 위주 워크로드에서 PostgreSQL 대비 운영 오버헤드 제거, 파일 기반 볼륨 마운트로 Docker 이식성 확보
- Traefik 파일 프로바이더: Docker 소켓 노출 없이 YAML 재작성으로 트래픽 전환, 보안성과 롤백 단순화 동시 달성

주요 내용

- Clean Architecture + TSyringe DI로 계층 분리 설계
- Cheerio 크롤링 + node-cron 자동 동기화 (월~토 매시)
- 카카오톡 챗봇 스킬서버 연동 (노선·차량·도착지 검색)
- Traefik Blue-Green 무중단 배포 + 자동 배포 스크립트
- Next.js 모바일웹 + Recharts 통계 대시보드

깨달은 점

- Clean Architecture의 도메인·애플리케이션·인프라 계층 분리와 TSyringe DI 컨테이너를 적용하며, 비즈니스 로직과 외부 의존성(크롤러, DB, 카카오 API)의 결합도를 제거하는 설계를 학습
- 카카오 i 오픈빌더 스킬서버 프로토콜을 직접 구현하며 챗봇 플랫폼의 요청·응답 규격과 시나리오 블록 연동 방식을 이해

기타 프로젝트

냠냠위듀 — 구내식당 식단 알림 봇 (개인) — TypeScript, Playwright, Slack Bot API, Express 상세 ↗

병원 채용공고 알림 프로젝트 (개인) — NestJS, TypeScript, PostgreSQL, TypeORM 상세 ↗

스타트업풀 (팀) — NestJS, TypeScript, PostgreSQL, JavaScript 상세 ↗