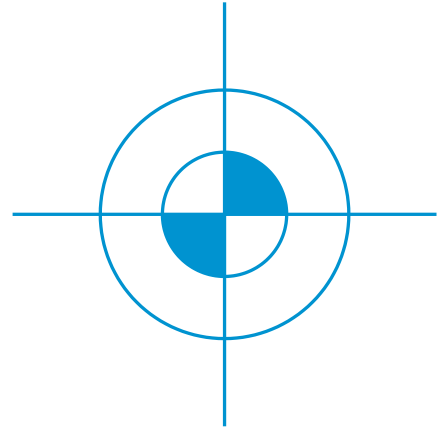




데이터가 어긋나지 않게, 끝까지 운영하는 개발자

공공 도시재생 정보시스템을 12개월간 유지보수하면서, 매년 반복되는 약 3,000만 건 규모의 데이터 갱신을 중간에 멈춰도 이어서 처리하고 다시 실행해도 같은 결과가 나오도록 다시 설계했습니다. 개인·팀 프로젝트에서도 수집부터 적재, 배포, 운영까지 직접 맡아 왔고, 문제가 생기면 원인을 기록한 뒤 다시 생기지 않는 구조로 고칩니다.

대량 데이터	건축물대장 약 3,000만 건 갱신 작업을 필요한 컬럼만 읽고, 실패 원인을 남기고, 멈춘 지점부터 이어서 처리하는 구조로 재설계했습니다. 개인 프로젝트에서는 하루 7,500롤 한도 안에서 경기 13,724건의 과거 데이터를 체크포인트 방식으로 무인 적재했습니다.
정합성	<code>INSERT ... ON CONFLICT</code> 역등 적재, 조건부 <code>UPDATE</code> 로 라이브 스코어 역행 차단, 발주 행 잠금 (<code>FOR UPDATE</code>)으로 동시 입고 시 초과 입고 차단.
운영	GitHub Actions 블루/그린 무중단 배포, 매일 자동 백업과 행 수 대조 복원 리허설, 헬스체크를 통과한 버전만 롤백본으로 남기는 배포 스크립트.
협업	5인 팀에서 DB 스키마와 서버 공통 기반을 맡고, 16개 모듈 중 15개의 구현 가이드를 작성했습니다. 3인 팀에서는 팀장으로 이슈와 PR 흐름을 운영했습니다.
연락처	rjsgmlq33@gmail.com github.com/donasman

기술과 경력

기술

언어	Java, TypeScript, SQL, Python
백엔드	Spring Boot, Spring Security, MyBatis, JPA, Spring Batch 공공 포털 유지보수(Spring Legacy), LessonTree, PitchLog 첫 버전 NestJS, Prisma PitchLog League, FantaSteel ERP
데이터베이스	PostgreSQL(PostGIS, Supabase), MySQL 트랜잭션 경계, 행 잠금과 advisory lock, 부분 unique 인덱스, 윈도우 함수, MyBatis 동적 SQL
인프라와 배포	AWS EC2와 S3, Docker, Nginx, systemd, GitHub Actions, Vercel 블루/그린 배포, 헬스체크 기반 롤백, pg_dump 백업과 복원 리허설
테스트	Vitest, Jest, Supertest, JUnit 운영과 같은 PostgreSQL에서 돌리는 통합 테스트, 동시성 시나리오 테스트
데이터와 GIS	Python ETL, QGIS, ArcGIS, 좌표계 변환과 레이어 통합
AI 활용	Claude Code로 개발하되 pre-commit 훅, 브랜치 보호, CI 필수 검사로 결과물을 검증합니다. Gemini function calling 기반 어시스턴트와 MCP 서버를 직접 구현했습니다.

경력과 학습

2026.09 – 2026.10	FantaSteel ERP K-뉴딜 아카데미 팀 프로젝트(5인). 백엔드 기반 설계, 구매·MRP·조직·알림 모듈
2026.08 – 2026.10	PitchLog League 개인 프로젝트. 유럽 19개 대회 축구 데이터 서비스 설계부터 운영까지
2026.08 – 현재	K-뉴딜 아카데미 수강 중
2026 – 현재	AI-Byte-Club 스터디 『주니어 백엔드 개발자가 반드시 알아야 할 실무 지식』 챕터별 토론. DB 인덱스, 동시성, 인증과 암호화
2024.09 – 2025.08	도시재생 종합정보체계 유지관리 사업 주택도시보증공사 발주, 아이티커넥 소속 프리랜서(주입), 12개월
2024.12 – 2025.05	백준 알고리즘 스터디 같은 문제를 Java와 JavaScript로 풀며 주 단위 마감 유지
2024.05 – 2024.08	Table Maid QR 비대면 주문·결제 팀 프로젝트(4인). 계정 찾기, 임시 비밀번호 메일, 서버 헬스체크 API
2024.04 – 2024.05	과외나무 LessonTree 팀 프로젝트(3인) 팀장
2023.12 – 2024.05	코리아IT아카데미 빅데이터·AI 기반 시니어 디지털 헬스케어 웰니스 플랫폼 개발 과정. Java, Spring Boot, React, MySQL, AWS
2019.03 – 2023.02	동의과학대학교 졸업

실무

2024.09 - 2025.08

12개월

아이티커넥 소속

프리랜서(주임)

응용SW개발

발주

주택도시보증공사

Java(Spring Legacy),

PostgreSQL, Python, QGIS,

ArcGIS

도시재생 종합정보체계 유지관리

국토교통부의 도시재생 종합정보체계를 주택도시보증공사가 발주한 유지관리 사업에서, 수행사 소속으로 전국 도시재생 사업지와 토지·건축물 데이터를 다루는 공공 포털의 백엔드 유지보수와 연 단위 데이터 갱신을 맡았습니다.

city.go.kr

맡은 일

유지보수	수년간 운영된 Spring Legacy와 PostgreSQL 구조를 분석해 오류 원인을 추적하고 수정했습니다. 데이터를 갱신한 뒤 통계와 화면에서 드러나는 오류도 확인해 고쳤습니다.
데이터 갱신	국가중점데이터로 개방되는 도시재생 지표를 매년 갱신하고, 공공데이터와 GIS 데이터를 정기적으로 처리했습니다.
공간정보	지자체가 제출한 사업경계 레이어를 하나로 통합하고, 좌표계 불일치와 위상 오류를 QGIS 검증 도구로 확인해 보정했습니다. 처음 다룬 GIS는 사내 문서와 QGIS 실무자 커뮤니티에 질문하며 익혔습니다.

대량 데이터 처리

건축물대장 3,000만 건 갱신 작업 재설계

문제	전국 건축물대장(약 3,000만 건 CSV)으로 노후건축물 비율 같은 지표와 용도·구조·연도별 GIS 레이어를 매년 새로 만들어야 했습니다. 전년도 코드를 다시 쓰려 했지만 그대로는 돌지 않았고, 실패하면 어디서 왜 멈췄는지 알기 어려웠습니다.
원인	<ul style="list-style-type: none">- 파일이 커서 한 번에 읽어 처리하기 어려웠습니다.- 건축물대장 데이터셋 구성이 바뀌어 전년도 코드와 맞지 않았습니다.- 지표마다 코드가 따로 실행되고 실패 기록이 남지 않아, 멈추면 처음부터 다시 돌려야 했습니다.
해결	<ul style="list-style-type: none">- 지표별로 필요한 컬럼만 골라 읽는 전처리 단계를 만들어 읽는 데이터 양을 줄였습니다.- 지표마다 흩어져 있던 실행 코드를 하나의 실행 흐름으로 합쳤습니다.- 처리 단위를 나누고 적재 전에 검증 단계를 두어, 같은 작업을 다시 실행해도 결과가 같게 했습니다.- 실패하면 원인을 로그로 남기고, 마지막 저장 시점부터 이어서 처리하게 했습니다.



결과

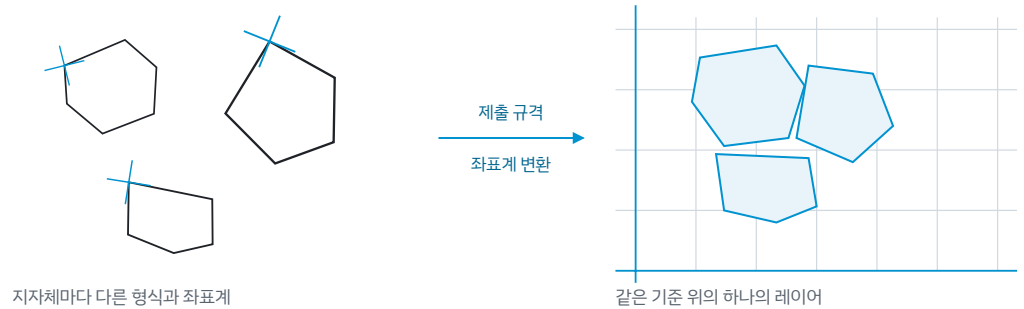
필요한 지표와 레이어를 산출했고, 중간에 멈춰도 3,000만 건을 처음부터 다시 돌리지 않게 되었습니다. 다음 갱신에서도 같은 코드를 이어서 썼습니다.

문제 지자체마다 사업경계 파일의 형식과 좌표계가 달라, 자료를 모을 때마다 변환하고 확인하는 데 시간이 걸렸습니다.

원인 제출 기준이 없어서 지자체마다 쓰던 형식 그대로 제출했습니다.

해결

- 입력 데이터 표준을 정의한 제출 규격과 작성 가이드를 만들어 지자체에 배포했습니다.
- 이미 받은 파일은 좌표계를 하나로 변환하고 데이터 형식을 맞춰 하나의 레이어로 통합했습니다.



결과 지자체마다 다르게 들어오던 자료를 같은 기준으로 관리할 수 있게 되었습니다.

문제 지표를 매년 갱신하는데, 기존 행정구역이 통합되거나 분리되면 전년도 값과 비교할 수 없었습니다.

해결 통합, 분리 같은 변경 유형별로 처리 기준을 세워 갱신 작업에 적용했습니다.

이 일에서 배운 것

기준을 먼저 반복되는 데이터 작업일수록 처리 기준을 먼저 정해 두어야 매년 같은 결과가 나온다는 것을 배웠습니다. 제출 규격과 처리 기준을 문서로 남겨 두었기 때문에 다음 갱신에서도 같은 방식으로 작업할 수 있었습니다.

레거시 읽기 국비 과정에서 DB 스키마를 직접 설계해 본 경험이, 남이 만든 레거시 구조를 빠르게 읽는 데 도움이 되었습니다. 오류는 화면에서 시작해 쿼리와 테이블까지 거슬러 올라가며 원인을 찾았습니다.

모르는 분야 GIS처럼 처음 보는 분야도 문서와 커뮤니티로 익혀 실무에 적용했습니다. 이 경험이 이후 축구 데이터, 철강 공정처럼 낯선 도메인의 프로젝트를 시작할 때 기준이 되었습니다.

개인 프로젝트

2026.08 – 2026.10

6주

1인

기획, 백엔드, 프론트엔드, 운영

NestJS 12, TypeScript,
Prisma 7,
PostgreSQL(Supabase),
React 18, AWS EC2와 S3,
Vercel

PitchLog League

유럽 5대 리그와 UEFA 대회, 국내 컵까지 19개 대회의 최근 5시즌 경기·선수 데이터를 외부 API에서 모아 보여 주는 축구 데이터 서비스입니다. 하루 7,500콜 한도 안에서 과거 데이터를 무인으로 채웠고, 라이브 스코어 수집과 AI 어시스턴트도 구현했습니다.

pitchlog-league.vercel.app github.com/donasman/pitchlog-league

규모

2026.09.17 기준

13,724 경기

2,110 팀

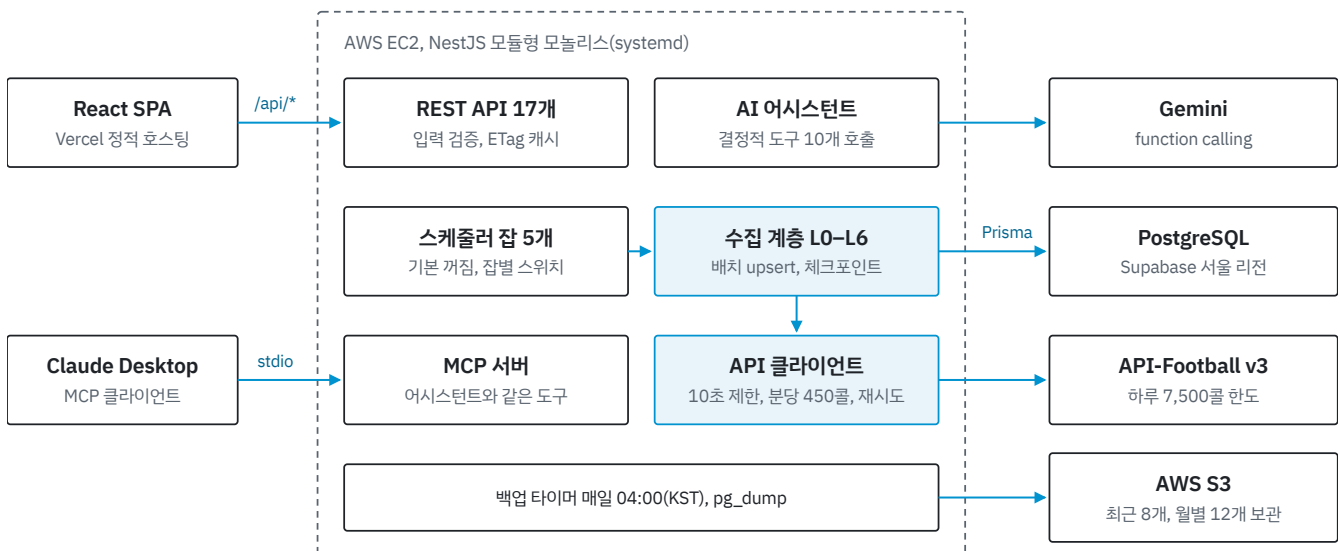
30,925 선수 시즌 통계

480 백엔드 테스트

136 PR

첫 버전에서

2026 월드컵 기간에 Spring Boot와 Spring Batch로 첫 버전을 만들어 운영했습니다. 회고에서 검증 장치 부재, 0건 수집을 성공으로 처리한 '조용한 실패', 배치와 스케줄러의 상호 호출을 문제로 정리했고, 범위를 넓히면서 NestJS와 Prisma로 다시 설계했습니다. 전환 근거는 ADR 문서로 남겼습니다.



핵심 설계 결정

외부 API 보호

외부 호출을 한 클래스(`ApiFootballClient`)로 모으고 10초 제한 시간, 분당 450콜 자체 제한, 네트워크 오류·429·5xx에만 1, 2, 4초 간격 재시도를 걸었습니다. 응답이 200이어도 오류 필드가 있으면 실패로 처리하고, 하루 한도에 닿으면 즉시 멈춰 '조용한 실패'를 막았습니다.

큐 없는 백필

약 4만 3천 콜 규모의 경기 상세 백필을 메시지 큐 대신 `backfill_jobs` 테이블의 커서와 단일 루프로 처리했습니다. 하루 자율 상한(5,700콜)과 중단 사유를 기록해 멈춘 커서부터 이어 가고, 라이브 수집은 백필이 끝난 뒤 커서 같은 한도를 두고 다투지 않게 했습니다.

DB 설계

"대회당 현재 시즌은 하나" 같은 규칙은 부분 unique 인덱스로 DB가 지키게 했습니다. 대량 적재 속도를 위해 외래 키를 두지 않는 대신 참조 컬럼 인덱스 69개와 고아 행 검사(CI, 매일)로 무결성을 확인하고, 테스트는 운영과 같은 PostgreSQL에서 돌립니다.

행 단위 upsert가 원격 DB 왕복에 막힌 문제

문제	행마다 upsert하자 원격 DB 왕복 때문에 5초 트랜잭션 한도에 걸렸습니다. 20개 팀을 저장하는 데만 약 4초가 걸려, 수십만 행 규모의 백필은 불가능했습니다.
원인	행마다 약 60ms의 네트워크 왕복이 생겼고, ORM의 upsert는 경우에 따라 조회와 저장 두 번의 쿼리로 나갔습니다.
해결	<ul style="list-style-type: none">- 테이블당 한 문장의 <code>INSERT ... ON CONFLICT DO UPDATE</code>로 묶는 <code>batchUpsert</code>를 만들고, 충돌 키 중복 제거, 명시적 타입 변환, 파라미터 65,535개 한도에 맞춘 분할, <code>COALESCE</code>로 기존 값 보호까지 처리했습니다.- DB 리전을 앱 서버와 같은 서울로 옮겼습니다.
결과	대화·시즌 하나당 쿼리가 3문장으로 줄었고, 기초 데이터 적재 시간이 200초에서 71초로 줄었습니다(리전 이전 효과 포함). 경기 13,724건과 상세 백필을 무인으로 끝냈습니다.

라이브 스코어 역행과 동시 저장 충돌

문제	15초 주기 라이브 폴러와 매일 도는 일정 수집 잡이 같은 경기 상태를 덮어써, 진행 중인 스코어가 이전 상태로 되돌아갈 수 있었습니다. 즐겨찾기를 동시에 저장하면 unique 충돌로 500 오류가 났습니다.
해결	<ul style="list-style-type: none">- 상태 순위와 경과 시간을 <code>WHERE</code> 절에서 비교하는 조건부 <code>UPDATE</code> 한 문장으로 바꿔, 읽고 쓰는 사이의 경합 없이 역행하는 쓰기를 막았습니다. 쓸 때마다 <code>data_version</code>을 올립니다.- 잡마다 <code>matches</code> 테이블에서 쓸 수 있는 컬럼을 나눠 서로 덮어쓰지 않게 했습니다.- 즐겨찾기 저장은 트랜잭션 안에서 사용자 단위 <code>pg_advisory_xact_lock</code>으로 직렬화했습니다.
결과	막힌 쓰기는 <code>/health</code> 에 차단 건수로 집계되고, 동시 저장 시나리오의 테스트로 고정했습니다.

운영

백업과 복원	매일 04:00(KST)에 <code>pg_dump</code> 로 백업해 S3에 올리고, 최근 8개와 월별 12개를 보관합니다. 일회용 컨테이너에 복원한 뒤 원본과 행 수를 대조하는 리허설을 두 번 통과했습니다.
배포	필수 환경변수가 빠진 배포로 서비스가 재시작을 반복했고, 이어서 재배포한 깨진 빌드가 롤백본을 덮어쓴 일이 있었습니다. 이후 배포 스크립트가 헬스체크를 통과한 버전만 롤백본(<code>dist.prev</code>)으로 남기도록 고쳤습니다.
CI	빈 PostgreSQL 16에 마이그레이션을 적용하고, 부분 인덱스 확인, 타입 검사와 린트, 단위 테스트 289개와 E2E 테스트 191개, 고아 행 검사를 통과해야 병합됩니다. 작업 브랜치에 보호 규칙을 걸고 PR 136개로 병합했습니다.
AI 활용과 검증	Claude Code로 개발하되 pre-commit 혹(널 바이트, 인코딩, API 키 유출 차단), 브랜치 보호, CI 필수 검사로 결과물을 검증했습니다. 어시스턴트는 숫자를 만들지 않고 서버의 결정적 도구만 호출하며, 답에 기준 시각과 근거를 붙입니다.
다음에 할 일	라이브 쓰기 모드의 실제 경기 관측 검증, 외부 알림 연동, 배포 자동화가 남아 있습니다.

팀 프로젝트

K-뉴딜 아카데미

2026.09 – 2026.10

5인

백엔드 기반 설계 담당

NestJS 12, Prisma 7,

PostgreSQL, Next.js 16, Jest

FantaSteel ERP

원료 입고부터 제선, 제강, 연주, 열연, 출하까지 일관제철 공정을 다루는 철강 제조 협업 ERP입니다. 팀의 백엔드 기반(스키마, 공통 모듈, 모듈별 구현 가이드)을 맡았고, 구매, MRP, 조직, 업무 알림 모듈을 직접 구현했습니다.

github.com/SteelFlowLLM/fantasteel-erp

맡은 일

기반 설계

ERD를 Prisma 스키마(44개 모델)와 초기 마이그레이션으로 옮기고, CHECK 제약과 부분 unique 인덱스를 정의했습니다. 인증·권한 가드, 공통 예외 필터, 채번, 작업 로그 기록기를 만들었습니다.

구현 가이드

16개 모듈 중 15개의 구현 가이드를 썼습니다. 테이블, API, 업무 규칙, 동시성 조건, 필수 테스트를 정의해 팀원들이 같은 기준으로 구현하게 했습니다.

직접 구현

구매(요청, 승인, 발주, 입고), MRP, 조직, 업무·알림 4개 모듈의 API 31개와 서버 테스트 약 120개.

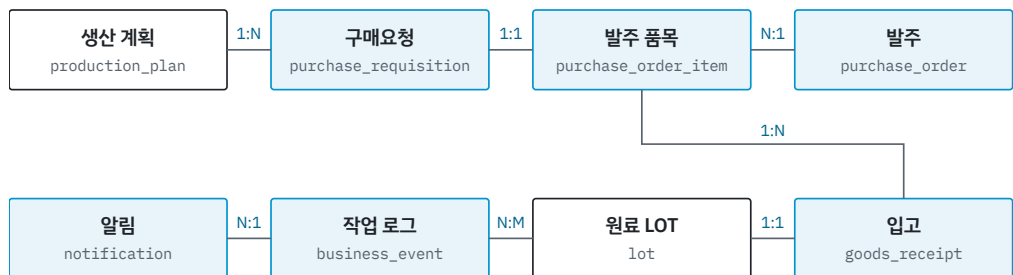
협업

팀 PR 216개 중 124개를 병합하고, 팀 공용 DB 반영을 맡았습니다.

설계 포인트

구매 데이터 모델

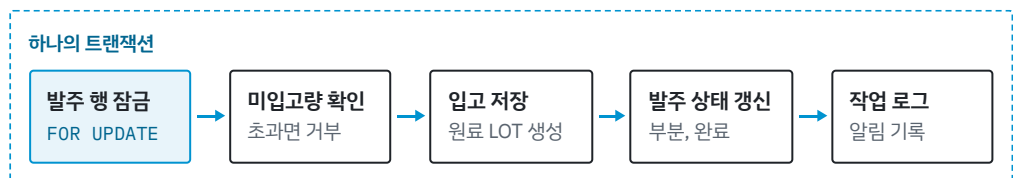
구매요청부터 발주, 입고, 원료 LOT, 작업 로그까지 이어지는 흐름을 테이블로 나눴습니다. 입고 한 건은 원료 LOT 하나를 만들고, 작업 로그는 LOT와 N:M으로 연결됩니다.



색칠한 테이블은 직접 구현한 구매·알림 모듈과 공통 작업 로그입니다.

초과 입고 방지

입고를 확정할 때 발주 품목이 아니라 상위 발주 행을 **FOR UPDATE**로 잠급니다. 발주 상태(부분 입고, 입고 완료)는 같은 발주의 모든 품목 입고 합계로 정해지기 때문입니다. 60t씩 동시에 두 번 입고하면 하나만 성공하고 원료 LOT도 하나만 생기는 것을 테스트로 확인했습니다.



상태 전이	승인, 반려, 재요청은 "현재 상태가 기대한 상태일 때만" 바꾸는 조건부 UPDATE로 처리해, 동시에 들어와도 한쪽만 반영됩니다. 여러 구매요청으로 발주를 만들 때는 id 순서로 잠가 교착을 피했습니다.
알림의 원자성	작업 로그와 알림을 본 거래와 같은 트랜잭션에서 기록해, 거래가 롤백되면 알림도 남지 않게 했습니다. 중복 알림은 부분 unique 인덱스로 막았습니다.
MRP 계산	생산 계획의 히트 톤, 제강 수율, 원단위로 원료 소요량을 구하고, 원료 LOT 잔량과 입고 예정을 빼 순소요량을 계산합니다. 같은 공급을 두 계획에서 중복으로 빼지 않도록 공급 사용 순서를 정했고, Decimal로 계산한 뒤 마지막에 한 번만 반올림합니다.
권한	승인은 권한 코드가 아니라 요청자 부서의 부서장인지로 판단하고, 본인 요청은 승인할 수 없게 했습니다. 권한은 요청마다 DB에서 다시 읽어 변경이 바로 반영됩니다.

트랜잭션 트랜잭션 안에서 쿼리가 겹치는 문제

문제	테스트에서 pg 드라이버 경고 <code>client is already executing a query</code> 가 나왔습니다. 다음 메이저 버전에서는 오류로 바뀌는 경고였습니다.
원인	Prisma 트랜잭션 안에서 include로 여러 관계를 한 번에 읽으면, 하나의 연결에 쿼리가 동시에 들어갔습니다. 쿼리 호출을 가로채 대기열을 세어 보니 경고는 5건이었지만 실제로 겹친 횟수는 789회였습니다.
해결	트랜잭션 안에서는 관계를 하나씩 읽고, 응답 모양을 만드는 조회는 커밋 뒤로 옮겼습니다.
결과	구매요청 등록부터 입고까지 전 구간에서 겹침 0회를 테스트로 고정했고, 같은 원인이 있는 다른 모듈의 위치를 PR에 정리해 팀에 공유했습니다.

팀 개발 환경 병합 전 변경이 공용 DB에 들어갈 위험

문제	개발 서버 실행 명령이 <code>DATABASE_URL</code> 이 가리키는 DB에 마이그레이션과 시드를 자동으로 실행해, 병합 전 브랜치의 변경이 팀 공용 DB에 들어갈 위험이 있었습니다.
해결	대상 DB가 로컬인지 판정하는 스크립트로 마이그레이션과 시드를 감싸고, 공용 DB 반영은 develop 브랜치, 깨끗한 작업 트리, 원격과의 일치를 확인하는 명령 하나로만 하게 했습니다.
결과	원격 주소로 실행하면 막히는 것을 확인했고, 그 과정에서 로컬(PostgreSQL 18)과 공용 DB(17)의 버전 차이를 발견해 리뷰 체크리스트에 넣었습니다.

팀 프로젝트

코리아IT아카데미

2024.04 – 2024.05

3인, 팀장

Java 11, Spring Boot 2.6,
MyBatis, MySQL, React 18,
Docker, Nginx, GitHub
Actions, AWS EC2와 S3

과외나무 LessonTree

학생과 선생님을 이어 주는 과외 매칭 웹 서비스입니다. 팀장으로 DB를 설계하고
이슈와 PR 흐름을 운영했으며, 배포 파이프라인을 직접 만들었습니다.

github.com/test-2024-0402-team4

담당

DB 설계, 학생 공고와 선생님 프로필 조회·검색, 과외 신청과 프로필 메일, 관리자·신고·계정 비활성화, CI/CD

협업

이슈 → 브랜치 → PR → 팀장 병합 흐름을 운영했습니다. 이슈 187개, PR 172개 중 65개를 작성하고 152개를 병합했습니다.

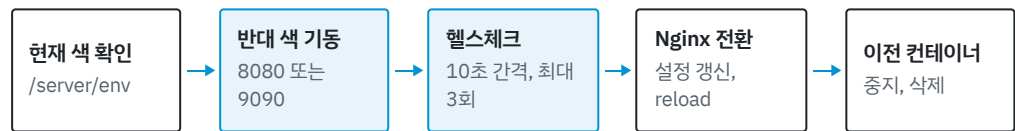
무중단 배포

main에 반영되면 GitHub Actions가 빌드한 이미지를 운영 중이 아닌 쪽 포트에 띄우고, 헬스체크를 통과하면 Nginx 대상을 바꿉니다. 적용 하루 전에 별도 저장소에서 같은 구조를 먼저 검증했습니다.

GitHub Actions에서 빌드



AWS EC2에 배포



다중 조건 검색

지역, 과목, 요일, 학생 유형 조건을 MyBatis 동적 SQL로 조합해 공고 ID만 먼저 뽑고, 그 ID로 과목·요일 같은 1:N 상세를 한 번에 조회했습니다. 조건 필터가 상세 목록을 잘라내지 않고, 비활성 회원의 공고는 빠지게 했습니다.

리팩토링

학생과 선생님 기능이 몰려 커진 `AccountService`를 `StudentService`와 `TeacherService`로 나눴습니다(14개 파일, 506줄 삭제, 265줄 추가).

메일

처음에는 메일 본문에 HTML 문자열을 직접 이어 붙였는데, 코드가 길어지고 데이터가 제대로 전달되지 않아 Thymeleaf 템플릿 4종(인증, 결과, 프로필, 과외 신청)으로 바꿨습니다.

돌아보면

신고와 계정 비활성화를 기획 단계에서 빠뜨려 개발 중에 테이블과 컬럼을 추가하고 계정 기능을 고쳐야 했습니다. 그 뒤로는 기능 목록을 먼저 확정하고 DB를 설계합니다. 배포 워크플로는 PR 이벤트에서도 배포 단계가 돌도록 되어 있었는데, 지금이라면 push 이벤트로만 제한하겠습니다.