CITY_SIGNAL · JAVA/SPRING BACKEND

도시 교통신호 최적화 시스템의 Java/Spring 중간 계층 설계·구현

강화학습 기반 교통신호 최적화 시스템에서 메타데이터 영속화, 시뮬레이션 오케스트레이션, 실시간 스트리밍, 분석 리포팅을 담당했습니다.

도메인	담당 영역
도시 교통신호 강화학습 최적화	**Java/Spring 백엔드**
기간	팀 구성
2025.09 ~ 2025.12	**Java 백엔드 / Python 연산 서버 / Unity 클라이언트 / 관리자 웹**

프로젝트 개요

고처리량 연산인 강화학습·교통 시뮬레이션은 Python 서버가 담당하고, 영속화·외부 인터페이스·운영 도구는 Java/Spring 백엔드가 담당하도록 역할을 분리했습니다.

이 구조를 통해 학습이 수십 분에서 수 시간 걸리더라도 운영 화면이 멈추지 않고, 연산 서버와 운영 서버가 서로의 장애로부터 격리될 수 있도록 했습니다.

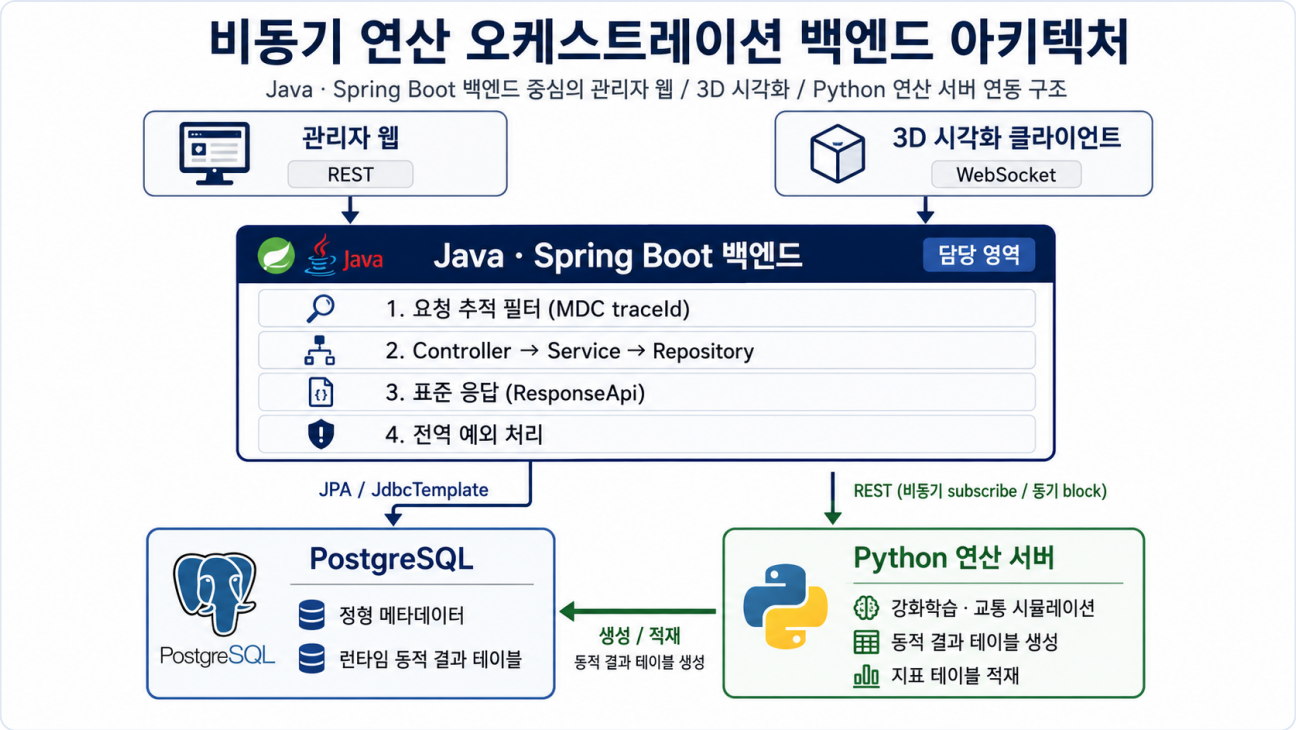
담당 역할 & 기여 요약

영역	기여
모델/최적화 요청 관리	학습·최적화 요청 오케스트레이션, AI/Custom 분리, JPA Specification 동적 필터
동적 테이블 연동	연산 서버가 런타임에 생성하는 결과 테이블을 JdbcTemplate로 조회·정리
실시간 스트리밍	Unity 시각화 클라이언트에 시뮬레이션 데이터를 고정 주기로 WebSocket 푸시
신호 도메인 변환	Ring × Phase 신호 계획 매핑, Net 녹색시간 계산
분석 리포팅	전/후 지표 집계 JOIN 쿼리와 Apache POI 기반 다중 시트 Excel 생성
공통 인프라	표준 응답 포맷, 전역 예외 처리, MDC traceId, WebClient/WebSocket/CORS 설정

기술 스택

Java 21 **Spring Boot 2.7** **Spring WebFlux(WebClient)** **Spring WebSocket**
JPA / Hibernate **JdbcTemplate** **PostgreSQL** **Apache POI** **Gradle**
log4j2

시스템 아키텍처



관리자 웹은 REST로 Java/Spring 백엔드와 통신하고, 3D 시각화 클라이언트는 WebSocket으로 데이터를 수신합니다. Java 백엔드는 요청 추적, 표준 응답, 예외 처리, 영속화를 담당하고 Python 연산 서버와 PostgreSQL을 연동합니다.

핵심 기술 의사결정

1. 장시간 작업은 비동기, 삭제는 동기

문제

학습·최적화는 분~시간 단위 작업이라 블로킹 호출 시 서버 스레드가 오래 점유되고, 동시 요청에서 스레드풀이 고갈될 수 있었습니다. 반면 삭제는 순서가 틀어지면 데이터 정합성이 깨질 수 있었습니다.

선택

쓰기 요청은 **`WebClient.subscribe()`** 로 즉시 반환하고 콜백에서 DB를 반영했습니다. 삭제 요청은 **`block()`** 으로 외부 정리 → 종속 데이터 → 메타 삭제 순서를 보장했습니다.

근거

프론트엔드 응답성과 데이터 정합성을 요청 성격에 따라 분리했습니다. 한 단계라도 실패하면 후속 단계를 중단해 부분 삭제 상태를 방지했습니다.

2. 런타임 동적 테이블은 JPA 대신 JdbcTemplate

문제

연산 서버가 시뮬레이션마다 이름이 다른 결과 테이블을 생성했습니다. JPA Entity는 테이블명이 컴파일 타임에 고정되어야 해서 표현이 어려웠습니다.

선택

동적 결과 테이블 조회·삭제 영역만 `JdbcTemplate` 으로 처리하고, 정형 메타데이터는 JPA를 유지했습니다.

근거

ORM 범위를 벗어나는 영역만 선택적으로 JDBC로 내려가고, 나머지 영역의 ORM 안정성과 생산성은 유지했습니다.

3. 실시간 시각화는 150ms 고정 주기 스케줄링

문제

시각화 클라이언트는 부드러운 애니메이션이 필요했지만, 매 프레임 DB 조회는 부하가 컸습니다. 원본 시뮬레이션 데이터는 1초 단위였습니다.

선택

`ScheduledExecutorService` 로 150ms 고정 주기 WebSocket 푸시를 적용했습니다.

근거

33ms는 DB 쿼리가 과도하고, 500ms 이상은 끊겨 보이기 쉬웠습니다. 150ms는 6~7fps 수준으로 부하와 체감의 균형을 잡은 선택이었습니다.

4. MDC traceId 표준화

문제

클라이언트 응답만으로 서버 로그에서 해당 요청 흐름을 찾기 어려웠습니다.

선택

Servlet Filter에서 요청마다 UUID traceId를 MDC에 주입하고, 표준 응답 객체에 traceId와 timestamp를 자동 포함했습니다.

근거

응답의 traceId로 서버 로그를 추적할 수 있고, try-finally에서 MDC를 제거해 스레드 재사용 시 이전 요청 ID 오염을 방지했습니다.

5. N+1 제거와 동적 필터

문제

목록 조회 시 행마다 연관 데이터를 조회하면 N+1 문제가 발생하고, 사용자 조합 필터를 문자열 JPQL로 분기하면 가독성과 안전성이 떨어졌습니다.

선택

연관 키를 모아 IN 절로 배치 조회 후 Map으로 룩업했습니다. 검색 조건은 JPA Specification으로 값이 있을 때만 동적으로 추가했습니다.

근거

조회 성능 문제를 구조적으로 줄이고, 동적 쿼리는 타입 안전하게 조립했습니다.

공통 계층 표준화

통합 응답 포맷

모든 REST 응답을 record 기반 **ResponseApi<T>** 로 통일하고, 생성 시점에 traceId와 timestamp를 자동 주입했습니다.

비즈니스 코드와 HTTP 코드 분리

같은 HTTP 400이라도 파라미터 형식 오류와 검증 실패를 별도 비즈니스 코드로 구분해 프론트엔드가 다르게 처리할 수 있도록 했습니다.

전역 예외 처리

@RestControllerAdvice 를 적용해 비즈니스 예외는 메시지 그대로, 시스템 예외는 고정 메시지와 500으로 변환했습니다.

외부 오류 노출 차단

내부 오류 상세는 외부에 노출하지 않고 서버 로그에만 전체 스택을 남겨 공격 표면을 줄였습니다.

트러블슈팅

신호 데이터 충돌 - 조합 키로 해결

시범 운영 데이터에서 서로 다른 Ring이 동일한 phase 번호를 가져 **Collectors.toMap** 이 충돌하는 문제가 발생했습니다. 키를 **ringType + "-" + phaseNo** 조합으로 만들고, merge 함수를 추가해 중복 데이터가 있어도 첫 값을 유지하도록 방어했습니다.

```
.collect(Collectors.toMap(
    p -> p.getRingType() + "-" + p.getPhaseNo(),
    p -> p.getOprTime(),
    (a, b) -> a));
```

리포팅 - 다중 시트 Excel

전/후 지표를 화면용 JSON과 보고서용 Excel 두 형태로 제공했습니다. Apache POI로 시간대별 지표, 지점별 효과, 신호 변경, 안전 시간 등 관점별 다중 시트를 생성하고, 시트별 헤더 색상을 도메인 의미와 일치시켜 비개발 검토자도 즉시 이해할 수 있도록 구성했습니다.

한계 인식 & 개선 로드맵

우선순위	항목	개선 방향
높음	인증·인가 미구현	JWT/세션 기반 인증 도입
높음	외부 서버 주소·자격증명 외부화 필요	@ConfigurationProperties + 프로필별 설정
중	동적 테이블명 식별자 검증	UUID 정규식 검증 추가
중	예외 처리 방식 이원화	전역 핸들러로 일원화
낮음	단일 세션 스트리밍	세션별 Map + 태스크 분리