

자보히르 (네드)

백엔드 개발자 & PostgreSQL DBA | 경력기술서

+82 10-8864-3848 | nediro883@gmail.com | github.com/kicheol83 | KakaoTalk open.kakao.com/o/sa2bEaRi | javohir.dev | 서울 서대문구

경력

백엔드 개발자 (원격, 타슈켄트)

2024.09 - 2025.08

Sigma Software

우크라이나에서 설립된 글로벌 IT 서비스 기업(전문가 2,000명 이상, 12개국). 6명 팀(백엔드 3, 프론트엔드 2, QA 1)에서 백엔드 API와 DB 스키마를 담당했습니다. Java, Kotlin, Spring Boot, PostgreSQL, Redis, Docker, GitHub Actions, AWS, Kubernetes.

- 주문 목록 API p95 1.4초 → 210ms: 복합 인덱스를 추가하고 N+1 쿼리를 JOIN 하나로 합침 (Grafana p95 그래프 전후 비교)
- Redis 캐시 도입 후 주 DB로 오는 반복 읽기 쿼리 약 45% 감소 (pg_stat_statements calls 전후 비교)
- Streaming Replication 기반 Primary-Standby HA 구성에 참여, Primary 장애 시 5~10초 안에 Standby로 failover
- 유럽 고객사의 주문·청구서 API를 Java/Kotlin, Spring Boot로 설계하고 Flyway 기반 버전 관리 마이그레이션 도입

백엔드 개발자 & PostgreSQL DBA (원격, 타슈켄트)

2022.10 - 2024.05

LeverX Group

SAP 플랫폼 기반 소프트웨어와 웹, 모바일, 클라우드 솔루션을 만드는 글로벌 IT 기업(직원 1,500명 이상). 8명 팀에서 연동 서비스, 리포트 API, DB를 담당했습니다. Node.js, NestJS, PostgreSQL, REST, Docker, Redis, AWS.

- PostgreSQL 13 → 15 메이저 업그레이드를 logical replication으로 서비스 중단 없이 완료
- 내부 시스템 간 데이터 교환 연동 서비스를 만들어 수작업 연동 업무를 없앴
- 대용량 리포트에 페이지네이션과 배치 내보내기를 적용해 서버 부하 없이 분할 처리
- 단위·통합 테스트를 작성해 CI에 연결, 느린 DB 쿼리 최적화

백엔드 & 데이터베이스 개발자 (원격)

2021.11 - 2025.09

프리랜서

프로젝트 단위로 해외 고객사의 DB 성능, 보안, 확장 문제를 해결했습니다 (총 16개월).

UAE, 제조 · 2021.11 - 2022.04 (6개월)

해결하려는 문제: 데이터가 늘면서 일일 생산 리포트가 느려져 관리자들이 오래 기다렸습니다.

주요 작업: EXPLAIN ANALYZE로 느린 쿼리를 분석해 다시 쓰고, 인덱스를 추가하고, 자주 쓰는 복잡한 리포트에 Materialized View를 도입했습니다.

결과: 리포트 로딩 12초 → 0.8초 (약 15배), DB 전체 쿼리 부하 감소

홍콩, 레스토랑 / F&B · 2023.01 - 2023.04 (4개월)

해결하려는 문제: 주문 시스템의 일부 DB 사용자가 업무에 필요 없는 과도한 권한을 갖고 있어 보안 위험이 컸습니다.

주요 작업: RBAC로 권한을 다시 설계하고 Row-Level Security로 테이블 접근을 제한했으며, 일일 백업과 복구 절차를 도입했습니다.

결과: 불필요한 superuser 접속 7 → 0, 백업 복구 테스트에서 15분 이내 복원

방글라데시, AI 서비스 · WellDev Bangladesh Ltd. · 2025.03 - 2025.09 (6개월)

해결하려는 문제: AI 모델 결과를 저장하는 테이블이 빠르게 커지면서 INSERT가 느려졌습니다.

주요 작업: 월 단위 range partitioning으로 테이블을 나누고 오래된 데이터의 아카이빙 전략을 도입해, 활성 부하가 필요한 파티션만 다루게 했습니다.

결과: INSERT p95 320ms → 45ms (약 7배), 활성 데이터 180GB → 22GB

PG Insight

누구나 가입해 자신의 PostgreSQL을 등록하고, 본인이 등록한 DB만 보는 오픈 멀티테넌트 모니터링 서비스

2026.08 - 2026.10

역할: 단독 개발: 설계, 백엔드, 프론트엔드, 보안, 테스트, 배포와 운영 (커밋 81개)

NestJS, TypeScript, Prisma, PostgreSQL, TimescaleDB, Socket.IO, React, Vite, Tailwind CSS, Recharts, Jest, Vitest, Docker, Caddy

증명하는 역량

- 테넌트 격리: 전역 가드가 모든 :targetId 라우트의 소유자를 확인하고, 남의 대상에는 403이 아니라 404를 돌려줍니다. 보호되지 않은 변경 라우트가 추가되면 테스트가 실패합니다
- SSRF 방어: DNS 응답의 모든 IP를 검사하고 연결을 검증한 IP에 고정합니다(TLS SNI 유지). 프로덕션에서 platform-db, 169.254.169.254, 127.0.0.1을 거부했습니다
- 임의 SQL 제한: EXPLAIN을 6단계로 막고 READ ONLY와 statement_timeout 5초로 실행합니다. pg_sleep은 0.05초, generate_series(1, 1000000000)는 5.4초에 400으로 끊겼습니다
- 모니터링 대상에 주는 부하를 측정했습니다: 10분 동안 CPU 1코어의 0.06%, 컨테이너 CPU +0.3%
- 운영: SHA 태그 배포, health 확인 뒤에만 기록, 양방향 rollback 훈련

해결하려는 문제

운영 중인 PostgreSQL의 느린 쿼리, 락 대기, 블로트, 복제 지연은 여러 도구와 SQL을 오가며 따로 확인해야 합니다. 이를 하나의 서비스로 모으되, 외부인이 남의 DB에 접속하는 공개 서비스인 만큼 테넌트 격리, SSRF 방어, 임의 SQL 제한을 처음부터 갖춰야 했습니다.

아키텍처

수집기가 각 DB에서 3초에서 5분 주기로 메트릭을 모아 TimescaleDB(hypertable 10개)에 기록하고, 실시간 화면은 대상 DB를 직접 조회합니다. 사용자, 대상, 알림, 감사 로그는 Prisma 7로 관리하는 플랫폼 DB(PostgreSQL 16)에 있습니다. NestJS 11의 전역 가드(Throttler → JWT → TargetAccess)가 모든 :targetId 라우트의 소유권을 확인하고, 대상 DB 쿼리

는 대상별 pool(최대 5)에서 raw SQL로 실행합니다. 역할은 admin(첫 계정)과 user(본인 대상 최대 3개, 공인 주소만) 두 가지이며, UI는 한국어, 영어, 우즈베크어를 지원합니다.

핵심 설계 결정

- 데모가 아닌 오픈 멀티테넌트 서비스: 가입을 열고, 사용자마다 본인이 등록한 대상만 보이게 했습니다. / 트레이드오프: 테넌트 격리, SSRF 방어, 할당량, 레이트 리미트가 필수가 되었습니다.
- 소유권 검사는 컨트롤러가 아니라 전역 가드에서: Throttler → JWT → TargetAccess 순서의 전역 가드가 모든 : targetId 라우트를 확인합니다. / 트레이드오프: 다른 ID를 쓰는 라우트에는 @TargetAccess 데코레이터가 필요합니다.
- 남의 대상에는 403이 아니라 404: 권한이 없으면 존재하지 않는 대상과 똑같이 404를 돌려줍니다. / 트레이드오프: admin에 대해서도 존재 여부를 따로 확인해야 합니다.
- 호스트를 resolve해 IP에 고정: DNS 응답의 모든 IP를 net.BlockList로 검사하고, 검증한 IP로 직접 연결합니다. / 트레이드오프: TLS SNI를 직접 넘겨야 하고, 셀프 호스팅용으로 TARGET_ALLOW_PRIVATE_HOSTS 옵션을 뒀습니다.
- EXPLAIN은 남기되 READ ONLY + 5초 + 함수 차단: 6단계 검사 뒤 READ ONLY 트랜잭션에서 statement_timeout과 lock_timeout을 걸고 실행합니다. / 트레이드오프: 시간과 함수 제한 때문에 일부 정상 쿼리도 거절될 수 있습니다.
- 새 대상의 기본값은 sslmode=require: 새 대상과 DB 기본값을 require로 바꿨습니다 (47287b4). / 트레이드오프: TLS가 없는 DB는 사용자가 직접 disable을 골라야 합니다.
- 마이그레이션은 되돌리지 않는다: 모든 마이그레이션을 추가형으로 작성합니다(먼저 추가하고 나중에 정리). / 트레이드오프: 모든 마이그레이션이 이전 코드와도 호환되어야 합니다.

핵심 수치

- 10분 동안 대상 DB에 보낸 쿼리: 2,430
- 대상 DB 부하 (CPU 1코어 기준): 0.06 %
- 쿼리 평균 실행 시간: 0.14 ms
- 수집 데이터 지연: 1.4 s
- EXPLAIN 안의 pg_sleep 차단 시간: 0.05 s
- 자동 테스트 (백엔드 188 + 프론트엔드 49): 237

장애 대응

API를 재시작한 뒤 어떤 대상도 모니터링되지 않았습니다. 부하 측정에서 PG Insight 쿼리가 0건으로 나왔습니다.

원인: 로그에 Starting collection for 0 active target(s)가 찍혔습니다. 오케스트레이터가 onModuleInit에서 pool이 연결되기 전에 시작됐고, 14:23-17:25 UTC 데이터가 사라졌습니다.

해결: 시작 시점을 onApplicationBootstrap으로 옮겼습니다 (6246ce2). 재시작 직후 대상 2개 모두 수집됩니다.

재발 방지: 오류를 재현하는 lifecycle 테스트를 추가했고, 일시정지한 대상을 건너뛰는지도 함께 검사합니다. 새 테스트는 이전 코드에서 실패하는 것을 확인했습니다.

테스트와 품질

- 테스트 수: 237
- CI: GitHub Actions (워크플로 준비, billing 문제로 현재 중단)
- Jest, Supertest, Vitest, 실제 PostgreSQL 16

결과

수집이 대상 DB에 주는 부하는 10분 동안 쿼리 2,430건, 실행 시간 합계 353.4ms로 CPU 1코어의 약 0.06%였습니다. 실제 프로덕션 DB(LedgerCore, FlashDrop)에서 5건, 코드 리뷰와 테스트에서 3건의 버그를 찾았고, 모두 먼저 테스트로 재현한 뒤 고쳤습니다. 다른 사례: 실시간 브로드캐스트 오류가 수집기를 멈추던 문제(b9825e4), lock 수집 쿼리의 RENAME 문법 오류(b9825e4), 연결 메트릭 시각이 세션 시작 시각으로 기록되던 문제(d587ce6, 잘못된 행 891개 삭제), 일시정지한 대상이 재시작 뒤 다시 켜지던 문제(41569c7), 비밀번호 변경 시 SSL 설정이 사라지고 prefer가 실제로는 암호화하지 않던 문제(41569c7, 47287b4), 첫 rollback 훈련에서 이전 이미지가 없어 스크립트가 안전하게 멈춘 일.

pginsight.javohir.dev | github.com/kicheol83/pg-insight

기 원장

역할: 단독 개발: 설계, 백엔드, DB, 테스트, 부하 테스트, 배포 (커밋 53개)

NestJS, TypeScript, PostgreSQL, node-pg-migrate, zod, React, Vite, Vitest, k6, Docker, Caddy, GitHub Actions

증명하는 역량

- 동시성 정확성: 순서가 정해진 행 락으로 부하 테스트 12/12에서 deadlock 0, fault 0. reversal race를 찾아 테스트로 고쳤습니다
- 불변식은 DB에: 차변 = 대변 규칙을 COMMIT 시점 트리거로 검사하고, 부하 중에도 백업에서 복원한 DB에서도 imbalance 0
- 측정 기반 최적화: 잔액 조회를 단일 statement로 줄여 고정 속도 A/B에서 p95 24.9 → 13.4ms
- 증거로 원인 찾기: 부하 중 10초 정지를 Postgres 로그와 API 로그의 시각을 맞춰 추적했고, 원인은 동기 로그 쓰기였습니다
- 운영: SHA 태그 배포, 약 1초 rollback, 매일 백업과 복원 검증

해결하려는 문제

계좌 간 자금 이동을 기록하는 시스템의 유일한 요구사항은 절대 틀리지 않는 것입니다. 요청이 같은 순간에 도착하든, 클라이언트가 재시도하든, 처리 도중 프로세스가 죽든 돈이 새로 생기거나 사라지면 안 되고, 같은 이체가 두 번 적용되어서도 안 됩니다.

아키텍처

API(NestJS 10 모듈러 모놀리스)가 사용자와 계좌, 입금과 출금, 이체, 취소(reversal), 잔액, 내역, 대사(reconciliation)를 처리합니다. 같은 코드베이스의 별도 워커 프로세스가 outbox 이벤트를 FOR UPDATE SKIP LOCKED로 가져와 HMAC 서명 웹훅으로 보냅니다. 불변식은 PostgreSQL 16의 deferred constraint trigger가 검사하므로 애플리케이션 코드의 실수로 우회할 수 없습니다. 운영 콘솔(React + Vite)은 계좌, 이체, outbox 전달 상태를 보여 줍니다.

핵심 설계 결정

- ADR-0001 금액은 BIGINT 최소 단위, API에서는 문자열: BIGINT 최소 단위로 저장하고 bigint 기반 Money로 감싸, 모든 경계에서 문자열로 직렬화합니다. / 트레이드오프: 클라이언트가 금액을 문자열로 읽어야 하지만, 부동소수점 오류가 없습니다.
- ADR-0002 잔액 컬럼 없음, 엔트리에서 계산: 잔액은 항상 ledger_entries 합계로 계산하고 covering index로 비용을 제한합니다. / 트레이드오프: 읽기 비용이 내역과 함께 늘어납니다. 측정이 필요해지면 스냅샷을 추가합니다.
- ADR-0003 ORM 대신 raw SQL: pg 드라이버와 raw SQL, 마이그레이션은 node-pg-migrate. / 트레이드오프: 타입 안전성을 컴파일러가 아니라 통합 테스트가 보장합니다.
- ADR-0004 모듈러 모놀리스: accounts, ledger, idempotency, outbox 모듈을 한 프로세스와 한 DB에 둡니다. / 트레이드오프: 모듈은 다른 모듈의 테이블에 직접 접근하지 않습니다.
- ADR-0005 불변식은 deferred constraint trigger로: COMMIT 시점 트리거가 엔트리 2개 이상, 차변 = 대변, 합계 = 거래 금액, 통화 일치를 검사합니다. / 트레이드오프: 오류가 statement가 아니라 COMMIT에서 나옵니다.
- ADR-0006 READ COMMITTED + id 순서 FOR UPDATE: 계좌 행 락을 항상 id 오름차순으로 잡습니다. / 트레이드오프: 새로운 쓰기 경로도 반드시 이 순서를 지켜야 합니다.
- ADR-0007 outbox만 월별 파티션, 원장은 파티션 없음: created_at 기준 월별 파티션 + DEFAULT 파티션을 outbox_events에만 적용했습니다. / 트레이드오프: 파티션을 미리 만드는 것이 운영 작업이 됩니다.

핵심 수치

- 부하 테스트 유효 실행 (deadlock 0, fault 0): 12 / 12
- 일반 이체 처리량 (50 VU): 249 req/s
- 일반 이체 p95: 228 ms
- hot 계좌 이체 p95 (40 VU → 계좌 1개, 409-deadlock 0): 896 ms

- 혼합 부하 처리량 (읽기 30 + 쓰기 10): 884 req/s
- 잔액 조회 p95 (고정 속도 A/B, 개선 후): 13.4 ms
- 백업에서 복원 (14/14 테이블, imbalance 0): 0.68 s
- 자동 테스트: 392

장애 대응

전체 벤치마크 중 500 3건과 409 3건이 나왔습니다. 원장은 균형을 유지했습니다.

원인: Postgres는 04:00:52.7에 lock_timeout을 냈는데 API는 그 응답을 10초 뒤에 기록했고, 같은 순간 40ms 안에 500이 몰려 나왔습니다. Node 프로세스 자체가 멈춰 있었던 것입니다. 원인은 모든 요청을 info 수준으로 로깅한 것으로, Windows에서 stdout 파이프 쓰기는 동기이고 PowerShell 리다이렉션이 따라가지 못했습니다.

해결: LOG_LEVEL=warn으로 같은 시나리오에서 fault 0, 409 0, 처리량 +37%. lock_timeout 3s, statement_timeout 5s, idle_in_transaction_session_timeout 10s가 락을 풀어 주었고, 그때 500 INTERNAL_ERROR를 돌려주던 timeout을 503 DATABASE_TIMEOUT(retryable: true)으로 바꿨습니다 (8c30dba).

재발 방지: 실제 Postgres와 pg-pool로 timeout → 503 테스트 3개를 작성했습니다. 프로덕션에는 비동기 logger나 sampling이 필요하다는 점을 제약에 기록했습니다.

테스트와 품질

- 테스트 수: 392
- CI: GitHub Actions (커밋 메시지, 포맷, lint, typecheck, Postgres 서비스로 테스트, 빌드; billing 문제로 현재 중단)
- Vitest, supertest, 실제 PostgreSQL 16 (mock 없음), k6 + bench.ps1

결과

최종 코드의 부하 테스트 12회가 모두 유효했고 fault 0, deadlock 0, contention 0, 매 실행 뒤 원장 균형이 맞았습니다. 잔액 조회를 단일 statement로 줄여 고정 속도 A/B에서 p95를 24.9ms에서 13.4ms로 낮췄습니다. 그 밖에 동시 reversal 두 건 중 두 번째가 409 대신 422를 돌려주던 race(ccee39b), .env.test가 gitignore에 있어 깨끗한 클론에서 테스트가 아예 돌지 않던 문제(e0e6884), k6 스크립트의 __ITER와 teardown 오류(0f76291)를 숫자를 믿기 전에 찾아 고쳤습니다.

ledger.javohir.dev | github.com/kicheol83/ledger_core

Smart Farm

ESP32 센서 → MQTT → 파이프라인 → 규칙 → 펌프 → 다시 기기로 닫히는 온실 실시간 2026.01 - 2026.10

IoT 모니터링과 자동 관수

역할: 단독 개발: 백엔드, 프론트엔드, ESP32 펌웨어, 배포와 운영 (커밋 171개, 2026-10-04부터 프로덕션)

NestJS, TypeScript, GraphQL, MongoDB, Redis, BullMQ, MQTT, Socket.IO, React, Vite, Leaflet, ESP32, MicroPython, Jest, Docker, Caddy

증명하는 역량

- IoT 데이터가 사라지지 않음을 측정으로 증명: MongoDB 132초 중단에 24/24, 백엔드 20초 중단에 7/7. 수정 전 코드의 유실(12건 중 1건, 배포 때 1건)도 먼저 측정했습니다
- 보안 감사와 수정: 가입 시 ADMIN 상승, 임의 필드 쓰기, 타 회원 리소스 접근, OTP 무제한 추측, 하드코딩된 JWT 키 등 12개 이상을 프로덕션 악용 여부부터 확인한 뒤 달았습니다
- 기기와 백엔드 사이의 루프를 닫음: RELAY_ON/OFF를 MQTT로 보내고, 펌프 시간을 서버(30분)와 펌웨어(1800초) 양 쪽에서 제한하며, 자동 규칙이 동시에 두 번 실행되지 않습니다
- 엔지니어링 규율: 새 테스트 89개는 모두 수정 전 코드에서 실패하는 것을 확인했고, 틀린 가설과 테스트가 부족했던 경우도 기록했습니다

해결하려는 문제

온실 센서 값은 끊기거나 튀고, 기기 시간이 틀리기도 하며, DB나 백엔드가 잠시 멈춰도 측정값을 잃으면 안 됩니다. 흙이 마르면 사람이 화면을 보기 전에 펌프가 켜져야 하고, 네트워크가 끊겨도 펌프가 계속 돌아 물이 넘치면 안 됩니다. 남의 농장 데이터에 접근할 수 없어야 하는 것도 기본입니다.

아키텍처

실제 ESP32(DHT22, 정전식 토양 수분, BH1750 조도, 수위, 릴레이 + 12V 펌프)와 프로덕션 데모용 시뮬레이터가 같은 경로로 Mosquitto(TLS 8883, QoS 1)에 보냅니다. NestJS 11 IoT 파이프라인이 기기 API 키, payload 형태, topic과 deviceId 일치, 센서 소유권, 보정을 확인하고 Redis 버퍼(재시도, DLQ)를 거쳐 MongoDB에 원시 값과 90일 TTL time-series로 저장한 뒤 시간·일·월 집계를 만듭니다. 센서별 Z-score 이상 탐지(≥ 2 경고, ≥ 3 심각), 식물 건강 지수, Open-Meteo 날씨(10분 캐시)를 계산하고, 자동화 규칙이 액추에이터 상태를 바꾸면 MQTT로 RELAY_ON/OFF를 기기에 보냅니다. 대시보드는 GraphQL(resolver 51개)과 Socket.IO로 실시간 갱신되며 한국어, 영어, 우즈베크어를 지원합니다.

핵심 설계 결정

- 메시지 버퍼: Redis list 대신 hash + zset 2개: `iot:buffer:messages(hash)`, `iot:buffer:inflight(zset, 처리 시작 시각)`, `iot:buffer:retry(zset, 다음 시도 시각)`, `iot:buffer:dead(DLQ)`. ack는 id 기준 O(1) `hdel + zrem`. / 트레이드오프: 재시도 로직을 직접 관리해야 해서 테스트 10개와 실제 redis-server 검증을 붙였습니다.
- 읽기값마다 `done[]` 기록: `Promise.allSettled` 결과에서 성공한 센서를 `done[]`에 적고 재시도 때 건너뛩니다. 재시도는 원래 `receivedAt`을 유지합니다.
- 권한 오류와 DB 오류 구분: `UnauthorizedException`만 버리고 나머지 오류는 모두 재시도로 보냅니다.
- MQTT 지속 세션 + `onApplicationBootstrap` 연결: `clean:false`와 고정 `clientId(MQTT_CLIENT_ID로 두 번째 인스턴스 구분)`를 쓰고 모든 `onModuleInit` 뒤인 `onApplicationBootstrap`에서 연결합니다.
- 오류 로그 기록은 best-effort: 로그 기록을 try/catch로 감싸고 실패하면 warn만 남겨 재시도 횟수에 영향을 주지 않습니다.
- 소유권 검사: 전역 인터셉터 + `@OwnedArgs`: 전역 `OwnershipInterceptor`가 인자 이름으로 리소스를 찾아 `farm.memberId`까지 체인을 올라가고, 여러 의미의 인자는 `@OwnedArgs({ 'input.fieldId': 'fieldMap' })`로 지정합니다. / 트레이드오프: 요청마다 소유권 조회가 추가됩니다.
- 펌프 제한을 양쪽에: 서버는 `autoOffAt`을 최대 30분으로 두고, 펌웨어는 `ticks_ms` 기준 1800초 + 60초 유예 뒤 스스로 끕니다.
- 자동 규칙: `cooldown`을 원자적으로 점유: `findOneAndUpdate` 필터에 `cooldown` 조건을 넣어 하나만 규칙을 차지합니다.
- `recordedAt` 대체값: 기기 시각이 24시간보다 오래됐거나 5분보다 미래면 서버 수신 시각을 쓰고, 초 단위 시각도 자동 인식합니다. 펌웨어는 NTP를 매시간 다시 시도합니다.

핵심 수치

- MongoDB 132초 중단 중 저장된 메시지 (DLQ 0): 24 / 24
- 백엔드 20초 중단 중 도착한 메시지: 7 / 7
- 배포 후 10분간 저장된 행 (거부 0): 320
- 보안 probe 거부: 5 / 5
- 새 테스트 (모두 수정 전 코드에서 실패 확인): 89
- 펌프 최대 가동 시간 (서버): 30 min
- 최대 재시도 간격 (30 / 60 / 120초, 3회 뒤 DLQ): 120 s
- MongoDB 컬렉션: 40

장애 대응

가입 mutation에 `memberRole: "ADMIN"`을 보내면 ADMIN 회원이 만들어졌습니다(probe before).

원인: `CreateMemberInput`에 `memberRole` 필드가 있었고 `memberModel.create(input)`이 입력 전체를 저장했습니다. 프로필 수정의 `MemberUpdateInput`도 역할, 상태, 이메일, 비밀번호를 받았습니다. 같은 감사에서 `@Roles(MANAGER)`는 있지만 `@UseGuards(RolesGuard)`가 없어 회원 목록이 토큰 없이 열려 있었고(`{ "total": 7 }`), `guard`가 클래스 수준 `@Roles`를 읽지 않아 버퍼 관리 resolver도 열려 있었습니다.

해결: 입력에서 필드를 없애고 create와 \$set에 허용된 필드만 넘깁니다(22a760bd). guard를 붙이고 getAllAndOverride로 handler와 클래스를 함께 읽어, 잘못된 토큰은 401입니다.

재발 방지: 고치기 전 프로덕션의 악용 흔적을 먼저 확인했습니다(회원 7명 모두 WORKER/ACTIVE, 권한 계정 0, 의심 호출 0). member-security.spec.ts 13개가 수정 전 코드에서 12개 실패하는 것을 확인했고, 배포 후 외부 probe 5개가 모두 거부됐습니다.

테스트와 품질

- 테스트 수: 89
- Jest 30, ts-jest, 실제 redis-server, Mosquitto 2.0.18, MicroPython stub harness

결과

2026-10-10-11 감사에서 수정 커밋 9개를 냈고, 모두 전후 측정이 있습니다. MongoDB를 132초 멈춰도 메시지 24/24가 저장됐고(재시도 6회, DLQ 0), 백엔드를 20초 멈춰도 7/7이 도착했으며 그중 백엔드가 꺼져 있던 동안 보낸 메시지는 broker 큐에서 왔습니다. 수정 전 코드에서는 58초 중단에 12건 중 1건, 배포 재시작에 1건이 사라졌습니다. 보안 점검에서 12개 이상의 취약점을 찾아 닫았습니다: 가입 때 memberRole로 ADMIN 되기(22a760bd), 프로필 수정으로 임의 필드 쓰기, 남의 농장·섹터·알림 접근(c2c5ca26), OTP 무제한 추측과 Math.random OTP, 코드에 박힌 JWT 키(a5a1b62e), 프로덕션에 열린 GraphQL playground. 고치기 전에 프로덕션의 악용 흔적부터 확인했습니다(회원 7명, 권한 계정 0, 의심 호출 0). 또 백엔드가 DB에서만 펌프를 켜고 기기에는 아무 명령도 보내지 않던 문제(3f3eacb2), NTP 동기화 전 ESP32가 1970년 시각을 보내던 문제도 고쳤습니다. 처음 세운 가설(오류가 heartbeat에서 나온다)이 틀렸을 때는 로그로 실제 원인(/sensors)을 찾았고, 45초 중단은 드라이버의 30초 대기 넘지 못해 오류 경로를 검증하지 못한다는 점도 기록했습니다.

farm.javohir.dev | github.com/kicheol83/smart-farm

FlashDrop

4가지 동시성 전략으로 초과 발급 없는 선착순 쿠폰 발급을 구현하고 k6로 측정해 비교

2026.09 - 2026.10

역할: 단독 개발: 설계, 백엔드, 테스트, 부하 테스트, 프론트엔드, 배포와 운영 (커밋 21개)

Java, Spring Boot, Spring Data JPA, PostgreSQL, Redis, Lua, Flyway, Bucket4j, JUnit 5, Testcontainers, k6, Docker, Caddy, Cloudflare

증명하는 역량

- 문제를 숫자로 증명: 락 없음 방식이 50장 대신 300장을 발급했고, remaining_quantity는 0이 아니라 46에 머물렀습니다
- 올바른 해법 3가지: 나머지 세 방식은 300건 중 정확히 50건만 통과시켰고 DB에도 정확히 50행
- 결과를 설명: 이 부하에서 낙관적 락이 FOR UPDATE보다 빨랐던 이유를 찾았습니다. FOR UPDATE는 거절될 250건까지 줄 세웁니다
- 측정 규율: warm-up, 3회 중앙값, TCP backlog로 일부 거부된 실행은 자동 무효. 재측정이 결론을 뒤집자 README를 고쳤습니다
- 운영: SHA 태그 배포와 rollback, 매일 백업과 복원 시험, 충돌 표시가 배포된 사고 뒤 넣은 배포 보호 장치

해결하려는 문제

쿠폰 50장에 동시 요청 300건이 몰려도 한 장도 초과 발급되면 안 됩니다. 단순한 '읽기 → 확인 → 쓰기'는 lost update 때문에 50장 대신 300장을 발급합니다. 같은 비즈니스 규칙을 4가지 방식으로 구현해, 실제 부하에서 정확성, 처리량, 지연 시간의 균형을 측정해야 했습니다.

아키텍처

Caddy가 정적 프론트엔드를 제공하고 /api/* 요청을 Spring Boot로 넘깁니다. 요청은 먼저 Bucket4j 레이트 리미터(상태는 Redis)를 통과합니다. 전략마다 엔드포인트, 엔티티, 테이블이 따로 있습니다. Redis 방식은 남은 수량과 '받은 사용자' 집합을 Lua 스크립트 하나에서 원자적으로 확인하고, PostgreSQL 기록은 @Async로 백그라운드에서 처리합니다.

핵심 설계 결정

- Redis: Lua에서 DECR + SADD, DB 기록은 @Async: SISMEMBER → GET → DECR + SADD를 claim.lua 하나에서 실행해 hot path에 Postgres도, check-then-act 톰도 없습니다. / 트레이드오프: Redis와 DB 사이에서 프로세스가 죽으면 쿠폰이 DB에서 사라질 수 있습니다.
- 낙관적 락의 재시도를 두 번으로 분리: OptimisticClaimService(재시도, 최대 20회) → OptimisticClaimAttempt (@Transactional, @Version) / 트레이드오프: 하나의 로직이 두 클래스로 나뉩니다.
- 전략마다 별도 테이블과 엔드포인트: 4가지 전략이 각자의 campaign / claim 테이블, 엔티티, 엔드포인트를 갖습니다. / 트레이드오프: 코드와 마이그레이션이 중복됩니다(테이블 8개).
- 락 없음 방식을 일부러 남김: lost update가 그대로 일어나는 엔드포인트를 유지하고, 해당 테이블에는 CHECK 제약도 두지 않았습니다. / 트레이드오프: 프로덕션에서는 이런 엔드포인트를 두지 않습니다.
- 멍등성과 재고 하한을 DB에서: UNIQUE(campaign_id, user_id), CHECK (remaining_quantity >= 0): 모든 claim 테이블에 UNIQUE를, 세 개의 올바른 전략에 V9 CHECK를 걸었습니다. 프로덕션에서 remaining_quantity = -1 갱신이 거부된 것을 확인했습니다. / 트레이드오프: Redis 전략은 Redis와 DB 두 곳에서 검사합니다.
- 레이트 리밋 키는 userId: Bucket4j + Redis(Lettuce) 토큰 버킷을 userId 기준으로 적용해, 인스턴스가 여러 개여도 한도를 공유합니다. / 트레이드오프: 클라이언트가 바꿀 수 있는 값이라 프로덕션에서는 IP와 함께 써야 합니다.

핵심 수치

- Redis Lua 처리량: 873 req/s
- Redis Lua p95: 178 ms
- SELECT FOR UPDATE 처리량: 188 req/s
- SELECT FOR UPDATE p95: 1,465 ms
- 낙관적 락 처리량: 331 req/s
- 낙관적 락 p95: 831 ms
- 락 없음: 쿠폰 50장 중 실제 발급: 300
- rollback 명령 시간: 1.2 s
- 백업에서 DB 복원 (11/11 테이블): 0.22 s

장애 대응

2026-10-09 배포 후 flashdrop.javohir.dev UI가 깨지고 페이지에 <<<<<<< HEAD 문자열이 그대로 나왔습니다.

원인: 로컬 저장소가 GitHub보다 뒤쳐져 push가 거부됐고, git pull 병합에서 4개 파일에 충돌이 났는데 표시를 지우지 않은 채 커밋했습니다. 프론트엔드는 Caddy가 git 작업 디렉터리에서 바로 제공했기 때문에 git pull 즉시 프로덕션에 반영됐고, 백엔드 rollback은 프론트엔드를 되돌리지 못했습니다.

해결: 마지막으로 정상 동작한 커밋에서 frontend/index.html을 복구하고 README를 다시 쓴 뒤, 실수로 커밋된 IDE 빌드 산출물(bin/)을 지우고 .gitignore에 추가해 다시 배포했습니다 (7ac0754).

재발 방지: deploy.sh가 git grep으로 upstream의 충돌 표시를 검사하고, 발견하면 배포를 멈춥니다. 병합은 --ff-only로만 합니다 (a2dd204). 숨은 후유증도 찾았습니다: 충돌을 고칠 때 README(ko/en/uz)가 예전 차가운 실행 표로 돌아가 있었고, 포트폴리오 문서를 repo와 대조하다 발견해 재측정 결과로 복구했습니다 (a29c567).

테스트와 품질

- 테스트 수: 4
- JUnit 5, Testcontainers (PostgreSQL 17, Redis 7), virtual threads, k6, bench.ps1

결과

락 없음 방식은 50장 대신 300장을 발급했고(remaining_quantity은 0이 아니라 46), 나머지 세 방식은 300건 중 정확히 50건만 통과시켜 DB에도 정확히 50행이 남았습니다. Redis 방식은 FOR UPDATE보다 처리량 4.6배, p95 8.2배 좋았습니다. 수치는 warm-up 1회 뒤 유효한 3회의 중앙값이며, 성공 + 품질 합계가 300보다 적은 실행(TCP backlog로 일부 거부)은 자동으로 무효 처리하고 다시 실행했습니다. 레이트 리밋은 같은 userId로 12건 중 6건 통과, 6건 429(토큰 5개 + greedy refill)였습니다.

flashdrop.javohir.dev | github.com/kicheol83/flashdrop

라이브 방송 중 한정 수량 특가 처리하는 플랫폼: 재고 100개에 구매 시도 1,000건, 초과 판매 0건

2026.09 - 2026.10

역할: 단독 개발: 아키텍처, 서비스 7개, 프론트엔드, 테스트, 부하 테스트, 배포와 운영 (커밋 127개)

Kotlin, Spring Boot, Spring Cloud Gateway, Spring Data JPA, PostgreSQL, Apache Kafka, WebSocket (STOMP), Flyway, Toss Payments, OpenTelemetry, Jaeger, Next.js, TypeScript, Tailwind CSS, JUnit 5, k6, Docker, Caddy

증명하는 역량

- 정확성: 재고 100개에 구매 시도 1,000건, 3회 반복에서 초과 판매 0, 결제·판매 불일치 0, HTTP 오류 0. 매 실행 끝에 불변식 5개 확인
- 장애 내성: Kafka나 결제 DB를 실제로 멈춰도 이벤트 유실 0, 이중 결제 0 (카오스 테스트)
- 측정으로 개선: 판매 완료 1분 27초 → 약 23초, 주문 생성 p95 5.72초 → 2.68초. 커넥션 풀 가설은 측정으로 기각
- 프로덕션의 비즈니스 위험을 증거로 차단: 한 계정이 미결제 주문으로 재고 전체를 잡던 문제를 partial unique index로 막음(15 → 0에서 15 → 5)
- 경계 방어: 로그인, 가입, 주문 레이트 리밋. 신뢰할 수 있는 프록시 체인에서 클라이언트 IP를 뽑아 위조한 X-Forwarded-For로는 429를 피할 수 없음(프로덕션 확인)
- 운영: SHA 태그 배포, health 기반 rollback, 프로덕션 양방향 훈련, 재시작 뒤에도 토큰이 유효한 JWT 키

해결하려는 문제

라이브 방송 중 한정 수량 판매는 몇 초 안에 주문이 몰리고, 재고, 결제, 배송이 서로 다른 서비스에 있습니다. 플래시 세일에 상품절은 일상이라 결제를 먼저 받으면 품절될 때마다 환불과 PG 수수료가 생기고, 서비스 하나가 멈추면 이벤트 유실이나 이중 결제가 생길 수 있습니다.

아키텍처

api-gateway(Spring Cloud Gateway)가 JWT를 검증하고 신원 헤더를 다시 쓰며 레이트 리밋과 WebSocket 프록시를 맡습니다. order-service의 Saga 오케스트레이터가 재고를 먼저 10분 잡고 결제를 받습니다(AWAITING_STOCK → AWAITING_PAYMENT → PAYMENT_CONFIRMING → CONFIRMING_STOCK → COMPLETED, 결제 창 5분). inventory(선점, 확정, 반환, 만료 정리), payment(Toss 승인과 취소, Idempotency-Key), live(STOMP 채팅, 실시간 재고, 시청자 수), auth(RS256 JWT, JWKS, 리프레시 토큰 교체), delivery(배송, HMAC 택배 웹훅)가 각자의 PostgreSQL을 쓰고, 서로의 데이터는 API나 Kafka 이벤트로만 읽습니다. 이벤트는 각 서비스의 transactional outbox에서 나가며 traceparent를 함께 저장해 Kafka를 거쳐도 Saga 전체가 한 Jaeger 트race로 이어집니다.

핵심 설계 결정

- 재고를 먼저 잡고 결제 (hold-then-pay): 재고를 10분 선점한 뒤 5분 결제 창을 열고, 결제가 확정되면 선점을 확정합니다. / 트레이드오프: 재고가 10분 동안 묶이므로 구매자당 열린 주문 하나 규칙이 필요해졌습니다.
- 코레오그래피가 아닌 오케스트레이션: order-service가 Saga를 조율하고 멈춘 단계를 복구 스케줄러가 5초마다 다시 진행합니다. / 트레이드오프: order-service가 중심 지점이 됩니다.
- Transactional outbox, 브로커 확인 뒤에만 PUBLISHED: 상태 변경과 이벤트를 한 트랜잭션에 기록하고, 릴레이가 병렬로 보낸 뒤 연속으로 확인된 앞부분만 PUBLISHED로 표시합니다. / 트레이드오프: 중복이 생길 수 있어 소비자를 모두 먹등하게 만들었습니다.
- Toss 호출은 트랜잭션 밖, 결과는 outbox와 같은 트랜잭션: TransactionTemplate로 Toss 승인은 밖에서 호출하고, 결과와 outbox 이벤트를 한 트랜잭션에 기록합니다. / 트레이드오프: 결과가 기록되지 않으면 복구 스케줄러가 같은 Idempotency-Key로 다시 시도합니다.
- 품절은 락 없는 스칼라 조회로 먼저 거절: 락 전에 숫자 하나만 읽어 0이면 거절합니다. 엔티티를 먼저 읽으면 Hibernate 1차 캐시가 다음 FOR UPDATE에서 오래된 값을 돌려줘 초과 판매가 날 수 있어 엔티티는 읽지 않습니다. / 트레이드오프: 재고가 반환되는 순간 품절 응답을 받을 수 있지만 초과 판매는 일어나지 않습니다.
- 열린 주문 하나: partial unique index: 열린 상태에 대해 (member_id, product_id) partial unique index(V5)를 두고, 위반하면 409 OPEN_ORDER_EXISTS와 기존 orderId를 돌려줍니다. / 트레이드오프: 구매자는 새 주문 전에 기존 주문을 결제하거나 취소해야 하며, UI가 기존 주문으로 안내합니다.

- 레이트 리밋은 gateway 메모리, 주문은 사용자 기준: 로그인 10/분, 가입 5/시간은 IP 기준, 주문 20/분은 사용자 기준. IP는 XFF 체인의 가장 오른쪽 공인 주소를 씁니다. / 트레이드오프: gateway가 하나일 때만 정확하고, IP 기준 로그인 제한은 여러 IP에서 한 계정을 노리는 공격을 막지 못합니다.
- JWT 키는 .env, kid는 RFC 7638 thumbprint: JWT_SIGNING_KEY를 서버 .env(chmod 600)에서 읽고, 같은 키는 항상 같은 kid를 냅니다. 키가 없으면 compose가 시작되지 않습니다. / 트레이드오프: 키를 교체하면 이전 토큰이 잠시 거부됩니다.

핵심 수치

- 초과 판매 (재고 100, 시도 1,000, 3회 모두): 0
- 판매 = 결제: 100 / 100
- 주문 생성 p95: 2.68 s
- 결제 요청 p95: 0.63 s
- 판매 완료 시간: 23 s
- 재고 알림 도달률 (시청자 500명): 99.2 %
- 시청자 수 메시지 감소: 98 %
- 카오스 테스트: 이벤트 유실 / 이중 결제: 0

장애 대응

2026-10-09 프로덕션에서 lc-boards-check.sh로 테스트 계정 하나가 미결제 주문을 연달아 만들자 2x201, 재고 15 → 0 이 되었습니다. 한 사용자가 10분 hold 동안 방송 상품 전체를 다른 사람에게서 막을 수 있었습니다.

원인: Saga가 POST /api/orders마다 별도의 hold를 열었고, '구매자 한 명에 열린 주문 하나' 규칙이 어디에도 없었습니다.

해결: V5 마이그레이션: 열린 상태(AWAITING_STOCK, AWAITING_PAYMENT, PAYMENT_CONFIRMING, CONFIRMING_STOCK)에 대한 (member_id, product_id) partial unique index. 컨트롤러가 먼저 확인하고, 경합 시에는 DataIntegrityViolationException을 인덱스 이름으로 구분해 409 OPEN_ORDER_EXISTS와 기존 orderId를 돌려주며, 프론트엔드는 그 주문을 이어 갑니다.

재발 방지: 로컬 Postgres에서 동시 트랜잭션 두 개 중 두 번째가 1.5초 기다린 뒤 거부되는 것을 확인했고, 프로덕션 재검사에서 1x201 + 409, 재고 5에서 멈췄습니다. 동시 요청 10개 중 정확히 1개만 201인지 확인하는 통합 테스트를 넣었고, k6 flash-sale.js는 VU마다 자기 사용자를 쓰도록 바꿨습니다(USERS ≥ VUS 검사).

테스트와 품질

- 테스트 수: 46
- JUnit 5 (통합 24, 카오스 3, gateway 단위 14, SigningKeys 5), 실행 중인 전체 스택에 실제 HTTP, k6 3종

결과

재고 100개에 구매 시도 1,000건(200 VU)을 3회 실행해 매번 초과 판매 0건, 결제와 판매 수량 불일치 0건, HTTP 오류 0 건이었고, 매 실행 끝에 불변식 5개를 admin API로 확인했습니다. 측정으로 병목을 찾아 판매 완료 시간을 1분 27초에서 약 23초로 줄였고, 커넥션 풀 가설은 측정으로 기각했습니다. 프로덕션에서는 계정 하나가 미결제 주문 2건으로 재고 전체 (15 → 0)를 잡을 수 있던 문제를 partial unique index로 막았고(15 → 5), auth 재시작 뒤 같은 토큰에 서비스마다 다른 응답(401/200)이 나던 JWT 키 문제(0eeaf2c)와 로그인·주문에 제한이 없던 문제(085ded0)를 고쳤습니다. 그 밖에 브로커가 없을 때 outbox 트랜잭션이 몇 분씩 락을 잡던 문제(881afda), 파티션 증설 뒤 컨슈머가 새 파티션을 못 보던 문제, 이전 실행의 hold가 만료돼 판매가 101로 보이던 k6 불변식 오류, 다른 프로젝트 컨테이너가 8080 포트를 잡아 통합 테스트 14 개가 실패한 일도 원인을 찾아 정리했습니다.

live.javohir.dev | github.com/kicheol83/livecommerce-saga