생활가전 판매·렌탈·유지보수 운영 시스템

Jake Tae-moon Jo · 개발 리드 · https://jake-jo-portfolio.vercel.app/ko/work/field-service-ops/

사무실 웹, 기사용 앱, 고객 포털을 만들어 첫 커밋 123일 만에 실사용을 시작했고, 26일 뒤 고객사가 쓰던 옛 관리 프로그램의 데이터를 운영에 옮겼습니다.

3개 영역 운영 중	123일	약 8분	1,263
사무실 웹·기사 앱·고객 포털	첫 커밋 → 실사용 시작	옛 프로그램 데이터 운영 반영	자동 테스트, CI에서 모두 통과

결과

첫 커밋 123일 만에 실사용을 시작했고, 26일 뒤 옛 관리 프로그램의 데이터를 운영에 옮겼습니다.

고객의 수정 요청은 모두 24시간 안에 반영했습니다.

사무실 웹, 기사용 앱(PWA), 고객 포털을 3개 언어로 만들었고, 세 영역 모두 운영 중입니다. 한 도메인과 한 앱이 세 영역을 서비스하고, 인증은 영역마다 따로입니다.

9월 25일 운영 DB의 시험 데이터를 비우고 0에서 실사용을 시작했습니다. 첫 커밋 123일 만입니다.

10월 21일 옛 관리 프로그램의 고객·계약·장비·장비별 소모품·수금 이력을 옮겼습니다. 7개 파일 사전 검사 오류 0, 반영 약 8분이었고, 이관 중 고객 알림은 0건이었습니다.

종이 양식 9종을 대신할, 2개 언어를 함께 적은 PDF 템플릿 13종(계약 5종, 영수증, 작업확인서, 납품서, 정기점검표 등)을 만들었습니다. 사무실·기사·고객·카탈로그 업로드 매뉴얼 4종을 2개 언어로 만들었고, 스크린샷은 시드 DB에서 자동으로 찍습니다.

자동 테스트 1,263개가 11월 6일 CI 실행에서 모두 통과했습니다.

기간
2026.05 – 2026.11

역할
사람은 본인 1명 · AI 에이전트 팀 설계·운영

팀
본인 1명 + AI 에이전트 11개 역할

플랫폼
사무실 웹, 기사용 앱(PWA), 고객 포털 (3개 언어)

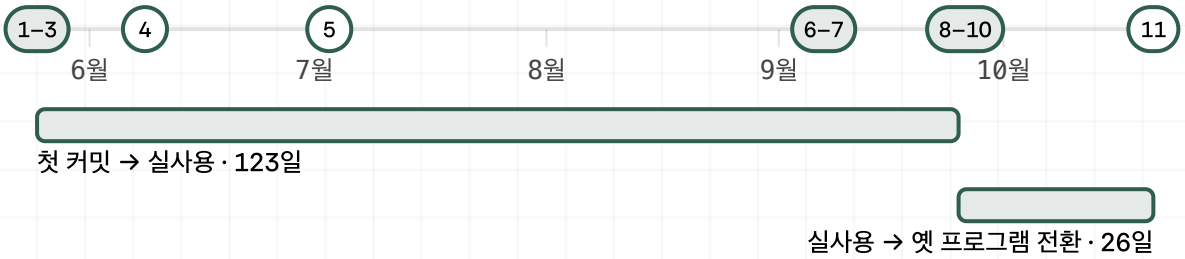
배경
베트남의 생활가전 판매·렌탈·유지보수 회사. 장비 한 대가 한 행인 옛 관리 프로그램을 쓰고 있었음

첫 커밋에서 옛 프로그램 전환까지

고객이 질문 50개에 하루 만에 답해 범위를 정했고, 첫 커밋 123일 만에 실사용을, 그 26일 뒤 옛 관리 프로그램의 데이터를 운영에 옮겼다.

123일
첫 커밋 → 실사용

26일
실사용 → 옛 프로그램 데이터 이전



1	2026년 5월 25일	**부트스트랩**	앞선 프로젝트의 에이전트 팀·빌드 틀을 옮기고 명세·고객 질문 작성
2	2026년 5월 26일	**고객 답변으로 범위 확정**	선택지·추천 기본값을 붙인 질문 50개에 모두 답
3	2026년 5월 31일	**국내 VPS에 자동 배포 시작**	고객이 국내 보관을 골랐다
4	2026년 6월 1일	**사무실·기사·고객 인증 분리**	6월 2일 보안 리뷰 반영
5	2026년 7월 3일	**고객 개선 요청 반영**	요청 #1~#4 반영(PR 12개), 이어서 장비 대량 등록 4단계 추가
6	2026년 9월 7일	**장비코드 규칙 확정**	모델별 순번, 서버만 발행
7	2026년 9월 12일	**SMS 실발송 시작**	
8	2026년 9월 22일	**고객 도메인 연결**	
9	2026년 9월 24일	**이관 화면**	검사(저장 안 함) → 확정 2단계
10	2026년 9월 25일	**실사용 시작**	운영 DB의 시험 데이터를 비우고 0에서 시작
11	2026년 10월 21일	**옛 관리 프로그램 데이터 운영 반영**	7개 파일 검사 오류 0, 반영 약 8분, 이관 중 고객 알림 0건

화면

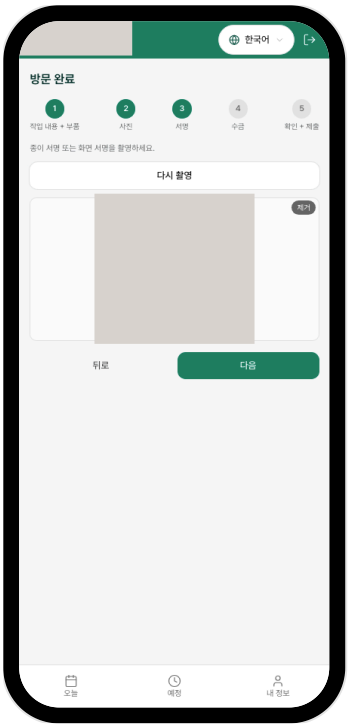
기사 앱「오늘의 방문」

방문 카드마다 서비스 유형·일시·서명이 필요한 서류가 있다. 시드 데이터이며 고객명·고객 코드·모델과 로고는 가리고 색을 바꿨다.



방문 완료 5단계 중 3단계(서명)

종이 서명을 사진으로 올리는 화면이다. 사진과 로고는 가렸다.



문제와 배경

장비 한 대를 중심으로 짠 옛 프로그램을, 고객 한 명을 중심으로 사무실·기사·고객이 함께 쓰는 시스템으로 바꾸는 일이었습니다.

고객사는 베트남에서 생활가전을 팔고 임대하며 정기적으로 유지보수하는 회사입니다. 장비 한 대는 판매나 임대 계약으로 시작하고, 기사가 설치합니다. 그 뒤 정해진 주기의 정기 방문으로 소모품을 교체하고 점검합니다. 그 사이에 수리·이전·유상 부품 같은 수시 서비스가 끼고, 수금은 방문 현금·계좌이체·회사 고객의 세금계산서로 받습니다. 임대가 끝나면 소유권이 고객에게 넘어가고, 원하면 유지관리 계약으로 이어집니다.

고객사는 dBase 형식의 옛 관리 프로그램으로 이 일을 관리했습니다. 옛 프로그램은 장비 한 대가 한 행이었습니다. 그래서 장비가 여러 대인 사업장은 같은 고객을 장비 수만큼 찾아 고치고 인쇄해야 했습니다. 방문 확인도 장비마다 따로 입력했고, 소모품 교체 예정일은 손으로 계산했습니다. 목표는「고객을 한 번 검색해 전부 보고, 한 번에 갱신한다」였습니다.

같은 장비를 본사 직원(관리자·매니저·직원 3단계), 현장 기사, 고객이 함께 다룹니다. 고객 쪽은 계약 주체 1명과 운영 담당자 여러 명으로 나뉩니다. 가정집과, 사업장이 여러 곳인 회사가 모두 고객이어서 고객 > 사업장 > 장비 3단계 계층이 필요했습니다.

제약도 분명했습니다. 고객사는 데이터를 국내 서버에 두기로 했습니다. 기사는 지하 주차장처럼 신호가 없는 곳에서도 작업 내용·사진·서명·수금을 남겨야 했습니다. 화면은 3개 언어로 만들고 알림은 받는 사람의 언어로 보내야 했으며, 계약서·영수증 같은 문서는 2개 언어를 함께 적기로 고객과 정했습니다. 옛 프로그램에 쌓인 이력도 새 시스템으로 옮겨야 했습니다.

장비 한 대의 일생

장비 한 대는 계약부터 임대 종료까지 정기 방문과 수금을 되풀이하며 사무실·기사·고객 세 영역을 오간다.

① **판매·임대 계약** ○사람
고객 > 사업장 > 장비 3단계, 가정과 회사 모두

② **기사 설치** ○사람
설치 이력이 장비에 남는다

③ **다음 예정일 계산** □시스템
최근 수행일 + 주기, 14일 안이면 대표 운영 담당자에게 이메일

④ **정기 방문** ○사람
추천 상위 3명 중 사무실이 기사 확정, 전날 SMS, 기사 앱 완료 5단계

⑤ **수시 서비스** ○사람
고객 포털 요청 → 무료 유형은 바로 방문, 유료 유형은 사무실 검토

⑥ **수금** ○사람
방문 현금·계좌이체·세금계산서, 기사 현금은 사무실 인계까지 추적

⑦ **임대 종료** □시스템
소유권을 고객에게 넘기고 유지관리 계약으로 갱신

되돌아가는 길
↻ ⑥ 수금 → ③ 다음 예정일 계산: 다음 주기

제 역할

기획은 제가 맡아 선택지와 추천 기본값을 붙인 질문을 설계했고, 범위는 고객이 그 질문에 답해 정했습니다. 에이전트 팀 운영도 제가 맡았습니다.

에이전트 팀은 11개 역할입니다. 오케스트레이터, 디자이너, TDD 가이드, 프런트엔드, 백엔드, 리뷰어, API 문서, QA, 매뉴얼, git 흐름, 데브옵스입니다. 파이프라인은 브랜치 시작 → 설계 → 실패하는 테스트 먼저 → 구현 → 테스트 통과 → 리뷰·API 문서 → QA → 매뉴얼 → PR 순서입니다. 팀과 빌드 틀, 규칙은 앞선 프로젝트(건설사 PMIS)에서 옮겨 왔고, 도메인 코드는 새로 썼습니다. git 작성자는 저 한 사람이고, 커밋 254개 중 213개(84%)에 AI 공동 작성 표시가 있습니다. 병합 커밋을 빼면 227개 중 213개(94%)입니다.

범위는 질문으로 정했습니다. 첫날(5월 25일) 고객에게 보낼 질문 50개를 썼고, 질문마다 선택지와 추천 기본값, 답이 없으면 막히는 단계를 적었습니다. 고객은 다음 날 50개 모두에 답했고, 이후 더한 질문까지 지금은 57개입니다. 고객이 추천과 다르게 고른 주요 변경은 7건이었습니다(사업장 계층, 한 방문에 기사 여러 명, 고객별 선호 기사, 회사 고객은 모두 세금계산서 등). 결정 카드의 번호(C.3 등)는 이 질문 문서의 번호입니다. 번호 붙은 결정 등록부는 따로 없고, 결정은 이 질문 문서, 설계 문서의 결정표, 커밋 본문, 프로젝트 규칙 파일에 남았습니다.

PR 142개를 모두 제가 열었고 139개를 병합했습니다.

누가 무엇을 정했나

범위는 고객이 질문에 답해 정했고, 질문 설계는 한 사람이 맡았으며, 코드 변경은 CI가 녹색일 때만 배포됐다.

고객

질문 50개에 하루 만에 모두 답했다(5월 26일), 이후 더해 지금 57개
추천 기본값과 다르게 고른 주요 변경 7건(사업장 계층, 한 방문에 기사 여러 명 등)
국내 보관(H.1), 기사 화면 PWA(C.3), 전체 이력 이관(J.1)은 고객이 고른 선택지
운영 규칙을 네 번 확인(5월 26일·28일, 9월 25일, 10월 20일)

본인

질문마다 선택지·추천 기본값·막히는 단계 표기
11개 역할 에이전트 팀 운영(앞선 프로젝트에서 이식), 커밋 254개 중 213개에 AI 공동 작성 표시

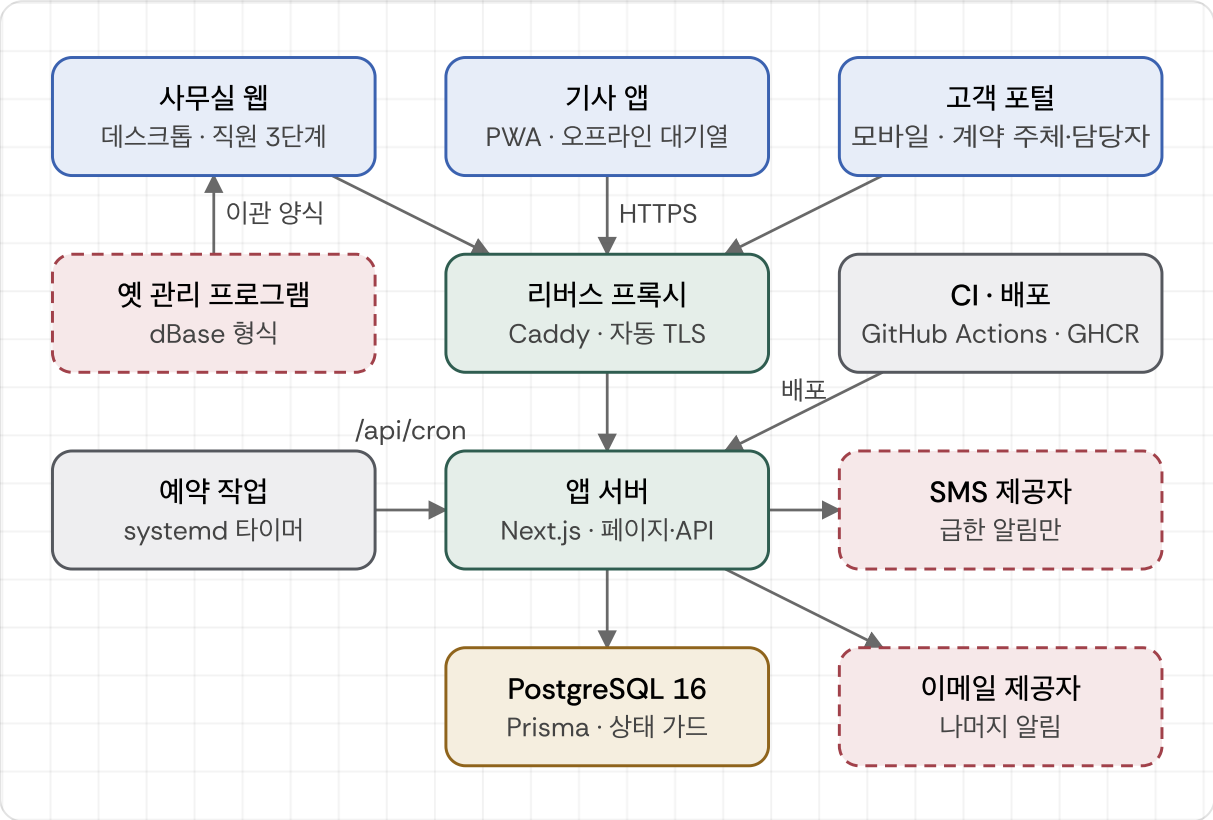
게이트

PR·main push마다 CI: 마이그레이션 적용·타입 검사·테스트·빌드
CI가 녹색일 때만 이미지 빌드 → 배포 → 헬스 체크
운영 데이터 삭제·이관은 확인 문구 + 직전 덤프 + 단일 트랜잭션

시스템 구조

시스템 구성

세 영역이 한 도메인과 한 앱을 함께 쓰되 인증은 영역마다 따로이고, 옛 프로그램의 데이터도 직원이 쓰는 이관 화면을 거쳐 들어왔다.



사무실 웹

고객·사업장·연락처, 계약(신규·수정·갱신·부록), 장비(단건·대량 등록), 방문(캘린더·미배정·일괄 인쇄), 서비스 요청, 수금, 세금계산서, 리포트, 관리 화면을 담는다. 세금계산서는 외부에서 만든 PDF를 직원이 올린다(고객 답변 D.1).

기사 앱

모바일 우선 PWA. 오늘·예정·방문 상세·완료 5단계. 고객 전화번호는 보이지 않고 「본사로 전화」만 있다. 연결이 없거나 전송 중 끊기면 완료 제출을 IndexedDB 대기열에 넣는다.

고객 포털

장비·방문 이력·수금·세금계산서·서비스 요청(메시지 스레드)·연락처를 본다. 계약 주체 1명과 운영 담당자 여러 명이 각자 계정을 쓴다.

옛 관리 프로그램

고객이 쓰던 dBase 형식 관리 프로그램(장비 1대 = 1행). 변환기는 DB에 직접 쓰지 않고 이관 양식(워크북)을 만들어 사무실 웹의 검사 → 확정 경로로 넣는다. 다시 올려도 이미 들어간 것은 건너뛴다.

리버스 프록시

Caddy 2가 Let's Encrypt 인증서를 자동으로 붙이고, 한 도메인으로 세 영역을 받는다. 영역마다 JWT를 따로 쓴다.

CI · 배포

PR과 main push마다 마이그레이션·타입 검사·테스트·빌드를 돌리고, 녹색이면 이미지를 만들어 SSH로 배포한 뒤 헬스 체크한다.

예약 작업

현금 인계 경고, 연체 독촉, 소모품 예정, 임대 갱신·종료, 방문 전날 알림, 월 청구, 야간 백업을 타이머로 둔다. 실행마다 CRON_RUN 감사 행을 남긴다.

앱 서버

페이지·API·미들웨어가 한 Next.js 앱이다. 세 영역은 JWT audience·쿠키·세션 저장소를 따로 써서 한 브라우저에서 동시에 로그인할 수 있다. 계약·방문·수금·서비스 요청은 순수 함수 전이표로 움직이고, 갱신은 이전 상태를 WHERE에 고정해 동시 수정을 막는다.

SMS 제공자

방문 전날처럼 급한 알림만 SMS로 보낸다. 개발 환경은 목 어댑터.

PostgreSQL 16

PostgreSQL 16 + Prisma 7. 장비는 지우지 않고 상태만 바꾼다(교체·이전·비활성·해지).

이메일 제공자

나머지 알림은 이메일로, 받는 사람(계약 주체 또는 대표 운영 담당자)의 언어로 보낸다.

`src/lib/equipment/asset-code.ts` · 같은 모델을 동시에 등록해도 번호가 겹치지 않게 모델별 잠금을 걸고, 코드를 숫자로 비교한 최댓값으로 다음 순번을 정한다. (주석은 원문을 줄여 옮김, 접두사는 바꿈)

```
async function nextSequence(
  tx: Prisma.TransactionClient,
  modelId: string | null,
): Promise<number> {
  // Serialize allocations for this model until the transaction commits.
  // Without it two parallel registrations of the same model read the
  // same max and both INSERT `EQ-000001`; Postgres aborts the whole
  // transaction on that unique violation, so retrying inside it is out.
  const key = modelId ?? NO_MODEL_BUCKET;
  await tx.$executeRaw`SELECT pg_advisory_xact_lock(hashtext(${key}))`;
  const taken = await tx.equipment.findMany({
    where: { modelId, assetCode: { startsWith: ASSET_CODE_PREFIX } },
    select: { assetCode: true },
  });
  // Take the numeric max, not ORDER BY assetCode DESC LIMIT 1:
  // lexical order breaks past 999999 and on legacy suffixes.
  let max = 0;
  for (const row of taken) {
    const n = Number(row.assetCode?.slice(ASSET_CODE_PREFIX.length));
    if (Number.isInteger(n) && n > max) max = n;
  }
  return max + 1;
}
```

`src/lib/migration/{plan,apply}.ts` · 검증 계획은 DB를 쓰지 않는 순수 함수라 미리보기와 확정이 같은 코드를 지나고, 반영은 워크북 전체가 한 트랜잭션이다. (두 모듈의 머리 주석)

```
// src/lib/migration/plan.ts
/**
 * Migration workbook → import plan.
 *
 * Pure: takes the parsed sheets plus a snapshot of what already
 * exists, and returns the rows to create together with every problem
 * found. No database access, so the whole validation surface is
 * testable without a schema, and the preview step and the commit
 * step run the exact same code — a preview that says "12 customers,
 * no errors" cannot disagree with what commit does.
 */

// src/lib/migration/apply.ts
/**
 * Writes a validated migration plan.
 *
 * One transaction for the whole workbook: a migration that half-lands
 * leaves customers without their contracts and equipment without its
 * consumables, and there is no sensible way to resume from that.
 * Either the file is in or it is not.
 *
 * What it deliberately does NOT do:
 *   - send anything. These customers are already being served; a
 *     portal invitation or a visit reminder going out during a data
 *     load would be a visible mistake on the customer's phone.
 */
```

핵심 결정

다섯 결정 중 기사 화면, 기사 배정, 데이터 보관 위치, 이관 범위의 넷은 고객이 제 질문에 답해 정했고, 다섯 모두 선택지와 대가까지 적었습니다.

기사 화면은 PWA, 오프라인 대기열까지 `C.3` 고객 쪽 결정

문제

기사는 지하 주차장처럼 신호가 없는 곳에서도 작업 내용·사진·서명·수금을 남겨야 합니다.

검토한 선택지

✓ PWA: 브라우저에서 돌고 앱스토어가 필요 없어 빨리 출시

iOS·Android 네이티브 앱: 사진·서명·오프라인이 더 매끄럽지만 출시까지 3~6개월 추가

결정

고객이 PWA(C.3)를 골랐고, 오프라인은 「온라인 우선, 대기열은 나중에」(C.4)를 골랐습니다. 실제로는 v1 범위의 마지막 단계에 오프라인 대기열까지 넣었습니다. 제출할 작업은 IndexedDB에 저장하고, 서비스 워커는 직접 작성했으며, 화면에 오프라인 배지를 띄웁니다. 방문 완료는 작업 내용·부품 → 사진 → 서명 → 수금 → 확인 5단계입니다. 서명은 고객 답변(E.1)에 따라 종이 서명을 찍은 사진으로 받습니다.

결과

완료를 제출할 때 오프라인이거나 전송 중에 연결이 끊기면, 앱은 완료 요청을 대기열에 넣고 화면으로 돌아갑니다.

대가·남은 위험

질문에 적은 대로, 네이티브 앱보다 사진·서명·오프라인 처리가 덜 매끄러울 수 있습니다. 지도와 경로 최적화는 뒤로 미뤘습니다(C.5).

기사 배정은 시스템 추천 + 사무실 한 번 클릭 확정 C.1 고객 쪽 결정

문제

방문마다 어느 기사를 보낼지 사무실이 손으로 정하고 있었습니다.

검토한 선택지

사무실 수동 배정(현재 방식) ✓ 시스템이 지역·부하·전문으로 추천하고 사무실이 확정(추천 기본값) 완전 자동 배정

결정

고객이 추천 + 확정을 골랐고, 고객별 「선호 기사」 필드를 더해 달라고 했습니다(C.2). 한 방문에는 주관 기사 1명과 협업 기사 여러 명을 둡니다(K.3). 완료·서명·수금은 주관 기사만 할 수 있고, 협업 기사는 사진과 메모만 남깁니다. 점수는 선호 기사 +100, 지역 일치 +30, 그날 이미 잡힌 방문 1건당 −5이고, 가중치는 관리자 화면에서 바꿉니다.

결과

배정 화면은 점수 상위 3명을 보여 주고, 사무실이 주관·협업 기사를 확정합니다. 확정하면 방문이 「제안됨」에서 「예정」으로 넘어갑니다.

대가·남은 위험

지역은 점수에 선호로만 더해집니다(고객 답변 C.2). 지도 기반 동선은 없고, 지도 화면은 자리만 잡아 두었습니다(C.5).

데이터는 국내에: 현지 VPS 자체 호스팅 H.1 고객 쪽 결정

문제

초기 계획은 관리형 클라우드였습니다. 데이터를 어느 나라에 둘지에 따라 호스팅 방식이 갈렸습니다.

검토한 선택지

✓ 국내 필수: 현지 호스팅 해외 리전 허용: 관리형 클라우드(추천 기본값, 빠른 출시)

결정

고객이 국내 보관을 골랐습니다. 현지 VPS에서 Docker Compose로 Caddy(자동 TLS)·앱·PostgreSQL 16을 돌리고, 예약 작업과 야간 백업은 systemd 타이머로 둡니다.

결과

main에 병합하면 CI가 녹색일 때만 이미지를 만들어 SSH로 배포하고 헬스 체크까지 합니다. 한 도메인이 세 영역을 모두 서비스합니다.

대가·남은 위험

관리형 클라우드가 맡아 주던 서버·인증서·예약 작업·백업을 직접 운영해야 합니다.

장비코드: 세 번 바꾼 끝에「모델별 순번, 서버만 발행」 A.3.1

문제

장비는 단건 등록, 대량 등록 등 여러 경로로 들어옵니다. 같은 모델을 여러 경로에서 동시에 등록해도 코드가 겹치면 안 됐습니다.

검토한 선택지

고객 코드 기반 번호(초기 계획) | 모델코드 + 설치일 + 순번, 테이블 전체에서 유일(9월 4일) | ✓ 고정 접두사 + 6자리, 순번은 모델별(9월 7일)

결정

고정 접두사와 6자리 숫자로 하고, 순번은 모델마다 따로 셉니다. 유일한 것은 (모델, 코드) 쌍입니다. 코드는 수정할 수 없고 다시 쓰지 않으며, 모든 등록 경로에서 서버만 발행합니다. 대량 등록 설계에 있던「자동/직접 입력」선택은 빼고 서버 자동 발행만 남겼습니다. 같은 모델을 동시에 등록해도 같은 번호를 잡지 않게 모델별 advisory lock을 걸고, 다음 순번은 코드를 숫자로 비교한 최댓값에서 정합니다.

결과

9월 7일 기존 장비의 코드를 새 규칙으로 다시 발행하는 마이그레이션을 넣었습니다. 11월 2일 시리얼 번호 필드를 없애고 장비 구분을 장비코드 하나로 통일했습니다.

대가·남은 위험

코드만으로는 장비가 정해지지 않고 모델과 함께 봐야 합니다. PostgreSQL 16의 기본 유일 제약은 NULL끼리를 서로 다른 값으로 보므로 모델이 없는 장비의 중복은 막지 못하고, 그 장비들의 유일성은 잠금이 맡는다고 코드 주석에 적었습니다.

이관은 전체 이력, 검사 후 확정, 하나의 양식 J.1 고객 쪽 결정

문제

옛 프로그램에 쌓인 고객·계약·장비·소모품·수금 이력을 옮겨야 했습니다. 옛 데이터는 장비 한 대가 한 행이라 고객 단위로 묶어야 했습니다.

검토한 선택지

✓ 전체 이력 이관 | 새 데이터부터, 이력은 옛 시트로 조회 | 활성 고객만(추천 기본값)

결정

고객이 전체 이력을 골랐습니다. 9월 24일 이관 화면을 먼저 만들었습니다. 템플릿, 현재 데이터 내보내기, 「검사(저장 안 함) → 확정」 두 단계입니다. 검증 계획은 DB를 쓰지 않는 순수 함수라 미리보기와 확정이 같은 코드를 지나고, 반영은 워크북 전체를 한 트랜잭션으로 합니다. 10월 20일 옛 프로그램 변환기는 DB에 직접 쓰지 않고 같은 이관 양식을 만들어 이 검사·확정 경로로 넣게 했습니다. 업로드 크기 제한 때문에 고객 단위로 파일을 나누고, 다시 올리면 이미 들어간 것은 건너뛰며, 보정한 값은 종류별 건수와 예시를 보고서로 남깁니다. 전환일은 10월 21일로 정했습니다.

결과

10월 21일 운영 반영은 7개 파일 사전 검사 오류 0, 반영 약 8분이었고, 이관 중 고객에게 간 알림은 0건이었습니다. 흐름은 아래 도표와 같습니다. 직원이 확인할 항목 3종(운영 담당자, 월 금액 없는 계약, 소모품 교체 기록 없는 장비)은 고객 화면 위에 모았습니다.

대가·남은 위험

소모품·점검의 전체 이력, 미수금 잔액, 회계 자료는 옮기지 않았고 옛 프로그램에서 조회합니다.

옛 프로그램에서 전환한 날

옛 프로그램의 데이터는 DB로 바로 쓰지 않고 직원이 쓰는 이관 화면의 「검사 → 확정」을 거쳤고, 운영 반영은 검사 오류 0으로 약 8분에 끝났다.

① **범위 결정** ○사람
전체 이력 이관(추천 기본값은 활성 고객만), 중복은 자동 매칭 + 사람 검토

② **이관 화면** □시스템
템플릿·내보내기·검사 → 확정, 미리보기와 확정이 같은 코드, 워크북 전체가 한 트랜잭션

③ **변환기** □시스템
DB에 쓰지 않고 같은 이관 양식을 만든다. 고객 단위 분할, 다시 올리면 들어간 것은 건너뜀

④ **리허설** □시스템
빈 DB에 실제 API로 파일 7개, 반영 51초·오류 0

⑤ **운영 반영** □시스템
PR 병합 → 자동 배포 → 운영 DB 백업 → 7개 파일 검사 오류 0 → 반영 약 8분

⑥ **확인 필요 항목** ○사람
운영 담당자·월 금액 없는 계약·소모품 기록 없는 장비 3종을 고객 화면 위에 모은다

품질과 검증

상태가 있는 것은 상태 기계로, 운영 데이터 작업은 겹친 안전장치로 지켰고, 자동 테스트 1,263개가 11월 6일 CI 실행에서 모두 통과했습니다.

자동 테스트 1,263개(Vitest 파일 168개)가 11월 6일 main CI 실행에서 모두 통과했습니다. 같은 실행에서 타입 검사와 빌드도 통과했습니다. 테스트 수는 작업 기록 기준으로 5월 27일 418개, 7월 10일 656개, 7월 21일 736개를 거쳐 늘었습니다. 파일로 나누면 단위 95개, CI의 PostgreSQL에 붙는 통합 41개, 컴포넌트 32개이고, 이 구분은 파일을 정적으로 센 값입니다.

상태가 있는 것은 상태 기계로 움직입니다. 계약·방문·수금·서비스 요청은 순수 함수 전이표를 쓰고, 갱신할 때 이전 상태를 WHERE 조건에 고정합니다. 두 사람이 같은 건을 동시에 바꾸면 한쪽은 오류를 받습니다.

운영 데이터를 지우거나 넣는 작업에는 안전장치를 겹쳐 두었습니다. 수동으로만 실행하고, 확인 문구를 입력해야 하며, 직전에 pg_dump를 뜨고, 한 트랜잭션으로 돌립니다. 운영 DB에서는 시드를 거부합니다. 이관 업로드는 전부 검사한 뒤 COMMIT을 입력해야 반영됩니다.
7월 장비 대량 등록 작업에는 하위 에이전트 개발 기록이 남아 있습니다. 작업 28개에서 리뷰 「승인」 17회, 「수정 필요」 3회(수정 후 재리뷰)가 나왔고, 단계별 최종 리뷰 4회 중 1회는 조건부였다가 수정 후 통과했습니다. 주기를 일 단위로 바꾸던 작업에서는 리뷰가 일 값에 월 더하기를 써서 30배가 되는 버그를 잡았습니다.

기능 하나, 처음부터 끝까지

스토리 하나가 고객 질문에서 운영 반영까지 어떻게 이어지는지 보여 드립니다. 옛 관리 프로그램에서 전환한 일입니다.

1 **질문과 결정 — J.1·J.2 (5월 26일)**

고객이 전체 이력 이관을 골랐습니다(추천 기본값은 활성 고객만). 중복은 자동으로 짝을 짓고 사람이 검토하며(J.2), 검증은 고객사 사무실 관리자와 개발 쪽이 함께 맡기로 했습니다.

2 **실사용 시작 — 운영 데이터 초기화 (9월 24~25일)**

운영 DB의 시험 데이터를 비우는 워크플로를 만들었습니다. 확인 문구, 직전 덤프, 단일 트랜잭션을 갖췄고, 여러 번 돌려도 결과가 같습니다. 로컬에서 리허설한 뒤 9월 25일 운영에서 실행했습니다.

3 **이관 화면 (9월 24일)**

워크북 템플릿, 현재 데이터 내보내기, 「검사 → 확정」 두 단계를 넣었습니다. 같은 날 이 화면을 관리자(ADMIN) 전용으로 좁혔습니다.

4 **옛 프로그램 변환기 (10월 20일)**

옛 테이블의 매핑은 프로그램 자체의 CSV 내보내기와 원본 행을 대조해 확정했습니다. 계약 출금일 열을 더하고 「확인 필요」 경고를 넣었습니다. 같은 날 고객과 이관 규칙을 확인했습니다. 장비 행을 고객으로 묶을 때 회사는 이름으로, 가정집은 이름과 전화 또는 주소로 묶습니다. 해지 고객도 옮기고, 오래된 소액은 USD로 보고 금액 칸을 비운 채 메모로 남깁니다.

5 **리허설 (10월 20일)**

빈 DB에 실제 API로 카탈로그 1개와 이관 파일 7개를 올리는 데 합계 약 1분, 오류 0이었습니다. 같은 파일을 다시 올리면 전부 건너뛰었습니다. 11월 1일 전환을 가정하고 예약 작업을 날짜 지정으로 돌려, 전환일에 연체와 독촉이 0건인 것을 확인했습니다.

6 **운영 반영 (10월 21일)**

오전에 PR을 병합해 자동 배포와 열 추가를 마쳤습니다. 운영 DB를 백업하고 무결성을 확인한 뒤, 내보내기 → 변환 → 7개 파일 검사(오류 0) → 반영 순서로 진행했습니다. 반영은 약 8분 걸렸습니다. 이관 중 고객에게 간 알림은 0건, 자동으로 열린 포털 계정도 0개였습니다.

회고

장비코드 규칙은 세 번 바꿨고, 배포 체인은 첫 나흘 동안 자주 실패했습니다.

장비코드 규칙은 세 번 바꿨습니다. 처음에는 고객 코드 기반 번호였고, 9월 4일에는 모델코드와 설치일과 순번으로 테이블 전체에서 유일하게 했으며, 9월 7일에 모델별 순번으로 정했습니다. 11월 2일에는 시리얼 번호 필드를 없애고 장비 구분을 장비코드 하나로 통일했습니다.

배포 체인은 첫 나흘(5월 31일~6월 3일)에 가장 많이 실패했습니다. CI 대기 권한, Docker 런타임 의존성, Caddy TLS 같은 문제를 PR로 하나씩 고쳤습니다.

일하는 방식

이 프로젝트도 역할별 AI 에이전트 팀으로 만들었습니다. 팀 구성과 게이트는 프로젝트마다 다르고, 이 프로젝트의 것은 위 「제 역할」에 적었습니다. 아래는 EWA 플랫폼을 만들며 정리한 방식입니다.

2026년 EWA 플랫폼을 만들며 세우고 운영한 방식 보기 →

https://jake-jo-portfolio.vercel.app/ko/process/

연락처

whxoans@gmail.com
+82 10-9233-7440