Operations system for home-appliance sales, rental and maintenance

Jake Tae-moon Jo · Engineering lead · https://jake-jo-portfolio.vercel.app/en/work/field-service-ops/

Built an office web app, a technician app and a customer portal; real use began 123 days after the first commit, and 26 days later the data from the client's old management program was moved into production.

3 areas in live use
office, technician, customer

123 days
first commit to real use

About 8 min
to load old data into production

1,263
automated tests, all passing in CI

Results

Real use began 123 days after the first commit, and 26 days later the old management program's data was moved into production.

Every change request from the client was shipped within 24 hours.

Built an office web app, a technician app (PWA) and a customer portal in 3 languages, and all three are in live use. One domain and one app serve the three areas, each with its own sign-in.

On Sep 25 the test data was cleared from the production database and real use started from zero, 123 days after the first commit.

On Oct 21 the customer, contract, equipment, per-appliance consumable and payment history moved over from the old management program. The run had 0 pre-check errors across 7 files and took about 8 minutes, with no customer notifications during it.

Built 13 PDF templates that carry 2 languages side by side (5 contract variants, receipt, work confirmation, delivery note, scheduled inspection sheet and more) to replace 9 paper forms. Wrote 4 manuals (office, technician, customer, catalog upload) in 2 languages, with screenshots captured automatically from a seed database.

All 1,263 automated tests passed in the Nov 6 CI run.

Period

May – Nov 2026

Role

Only human on the build team; designed and ran the AI agent team

Team

Me plus an 11-role AI agent team

Platforms

Office web app, technician app (PWA), customer portal (3 languages)

Context

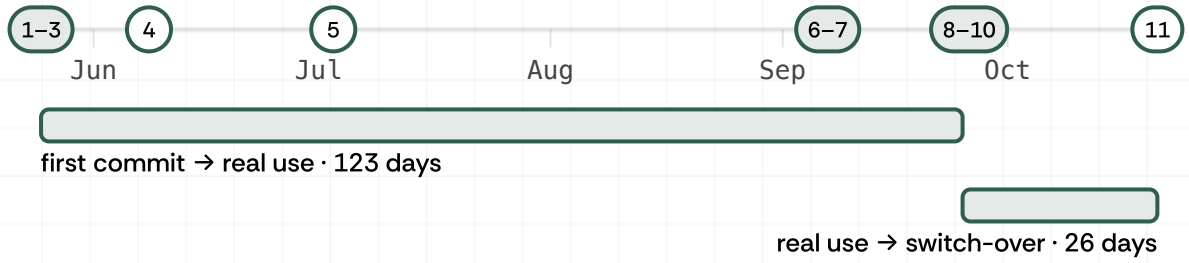
A company in Vietnam that sells, rents out and maintains home appliances, running an old management program with one row per appliance

From first commit to replacing the old program

The client answered all 50 scoping questions in a day; real use began 123 days after the first commit, and the old management program's data was moved into production 26 days later.

123 days
first commit to real use

26 days
real use to the old program's data in production



1. May 25, 2026 — **Bootstrap**
   Agent team and build setup carried over from an earlier project; spec and client questions written
2. May 26, 2026 — **Scope fixed by the client's answers**
   All 50 questions, each with options and a recommended default, answered
3. May 31, 2026 — **Auto-deploy to an in-country VPS begins**
   The client chose in-country hosting
4. Jun 1, 2026 — **Separate sign-in for office, technicians, customers**
   Security review fixes on Jun 2
5. Jul 3, 2026 — **Client change requests**
   Requests #1–#4 shipped (12 PRs), then a four-step bulk equipment registration added
6. Sep 7, 2026 — **Equipment code rule settled**
   Per-model sequence, issued by the server only

7	Sep 12, 2026	**Live SMS sending**
8	Sep 22, 2026	**Client domain connected**
9	Sep 24, 2026	**Migration screen** Two steps: check (no save), then commit
10	Sep 25, 2026	**Real use begins** Test data cleared from the production database; started from zero
11	Oct 21, 2026	**Old program's data loaded into production** 7 files, 0 check errors, about 8 minutes, no customer notifications during the run

Screens

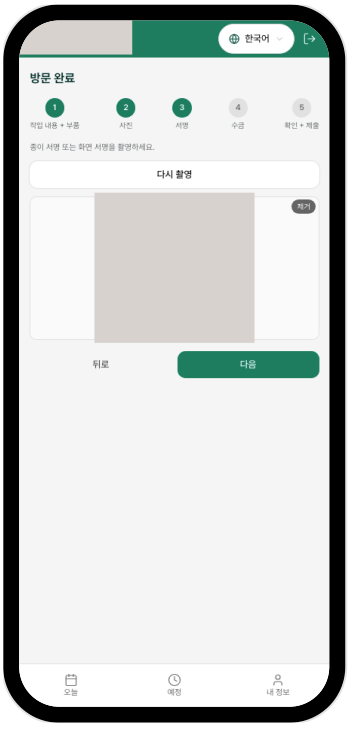
Technician app "Today's visits" (Korean UI)

Each visit card shows service type, date and time, and the documents that need a signature. Seed data; customer names, customer codes, models and logo masked, colours shifted.



Visit completion, step 3 of 5 (signature) in Korean UI

The screen uploads a photo of the paper signature. Photo and logo masked.



Problem and context

The job was to replace an old program built around single appliances with a system built around the customer, shared by the office, the technicians and the customers themselves.

The client is a company in Vietnam that sells and rents out home appliances and maintains them on a schedule. An appliance starts with a sale or rental contract, and a technician installs it. After that, scheduled visits at a set interval replace consumables and run inspections. On-demand service such as repairs, relocations and paid parts comes in between, and payment arrives as cash on a visit, a bank transfer, or a tax invoice for business customers. When a rental ends, ownership passes to the customer, who can continue on a maintenance contract.

The client ran all this on an old dBase-format management program. The old program kept one row per appliance. A business site with several appliances meant finding, editing and printing the same customer once per appliance. Visit confirmations were also entered appliance by appliance, and consumable due dates were worked out by hand. The goal was: look up a customer once, see everything, update it once.

Head-office staff (three roles: admin, manager, staff), field technicians and customers all handle the same appliances. On the customer side there is one contract owner and several operating contacts. Customers include households and companies with several sites, which called for a three-level hierarchy: customer > site > appliance.

The constraints were clear. The client chose to keep its data on servers inside the country. Technicians had to record work, photos, signatures and payments even where there is no signal, such as an underground car park. The screens had to come in 3 languages, notifications had to go out in each recipient's language, and documents such as contracts and receipts were agreed with the client to carry 2 languages side by side. The history built up in the old program had to move to the new system as well.

The life of one appliance

From contract to the end of the rental, one appliance cycles through scheduled visits and payments, passing between the office, the technicians and the customer.

1. **Sale or rental contract** ○Person
   Customer > site > appliance; homes and businesses
2. **Installation** ○Person
   Installation history stays with the appliance
3. **Next due date** □System
   Last service date + cycle; an email to the main contact within 14 days
4. **Scheduled visit** ○Person
   Office picks from the top-3 suggestions; SMS the day before; five-step completion in the app
5. **On-demand service** ○Person
   Portal request: free types go straight to a visit, paid ones to office review
6. **Payment** ○Person
   Cash on visit, transfer or tax invoice; technician cash tracked until handed to the office
7. **End of rental** □System
   Ownership passes to the customer; renewed as a maintenance contract

Loops back

↻ ⑥ Payment → ③ Next due date: next cycle

My role

I did the planning, designing the questions with their options and recommended defaults, and the client set the scope by answering them. I also ran the agent team.

The agent team has 11 roles: orchestrator, designer, TDD guide, frontend, backend, reviewer, API docs, QA, manuals, git flow and DevOps. The pipeline runs: start a branch → design → failing tests first → implement → tests pass → review and API docs → QA → manuals → PR. The team, the build setup and the rules came over from an earlier project (the construction PMIS); the domain code is new. I am the only git author, and 213 of the 254 commits (84%) carry an AI co-author trailer, or 213 of 227 (94%) leaving out merge commits.

The scope came from questions. On day one (May 25) I wrote 50 questions for the client, each with options, a recommended default, and what stays blocked until it is answered. The client answered all 50 the next day; with questions added later there are 57 now. Seven of the client's answers were major changes against the recommendation (a site

level in the hierarchy, several technicians per visit, a preferred technician per customer, tax invoices for every business customer, and others). The decision card numbers (C.3 and so on) are the numbers in that question document. The project keeps no numbered decision register; decisions live in that question document, in the decision tables of the design documents, in commit messages and in the project rules file.

I opened all 142 PRs and merged 139.

Who decided what

The client set the scope by answering the questions, one person designed the questions, and code changes were deployed only when CI was green.

Client

Answered all 50 questions in a day (May 26); 57 now

7 major choices that differ from the recommended default (site hierarchy, several technicians per visit, …)

In-country hosting (H.1), a PWA for technicians (C.3) and full-history migration (J.1) were the client's picks

Confirmed operating rules four times (May 26 and 28, Sep 25, Oct 20)

Me

Every question with options, a recommended default and what it blocks

Ran an 11-role agent team carried over from an earlier project; 213 of 254 commits carry the AI co-author mark

Gates

CI on every PR and push to main: migrations, type check, tests, build

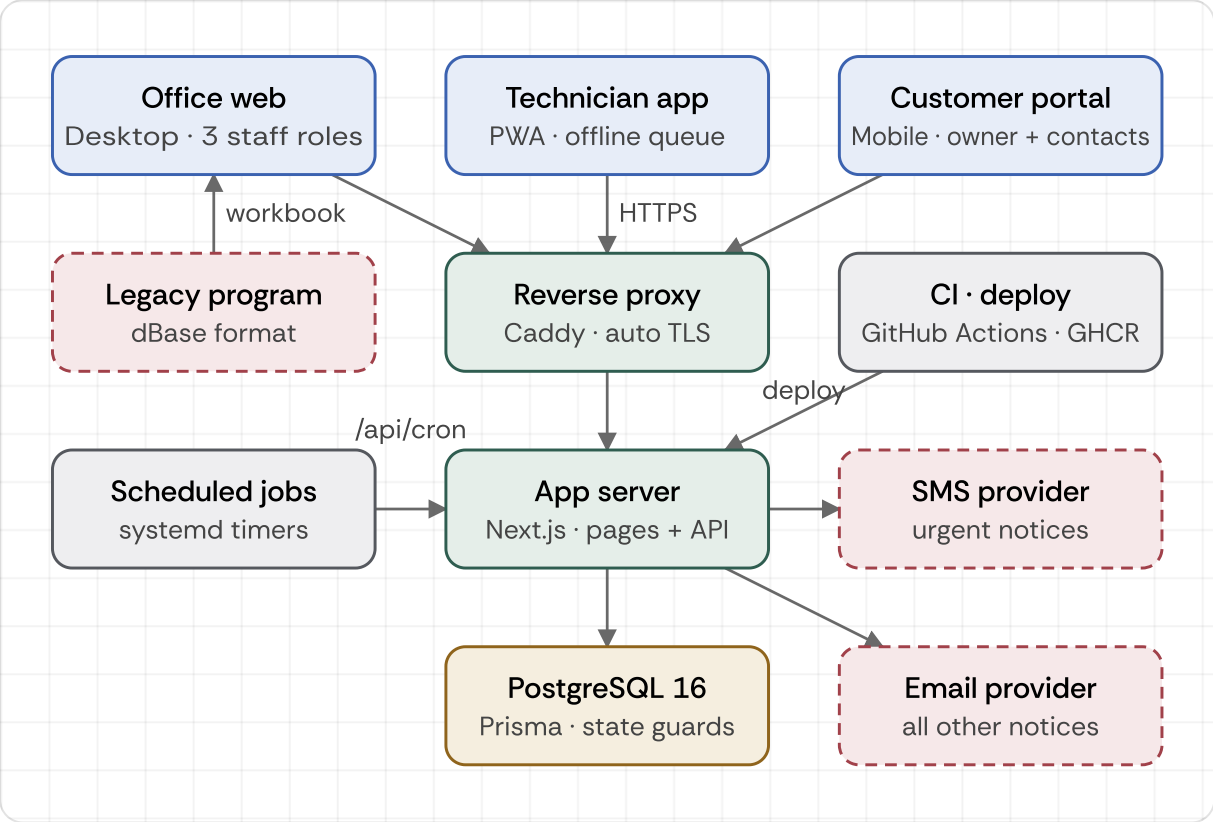
Image build, deploy and health check only after CI is green

Production purges and migrations need a typed confirmation, a fresh dump and one transaction

System architecture

System architecture

The three areas share one domain and one app but sign in separately, and even the old program's data came in through the same migration screen the staff use.



Office web

Customers, sites and contacts; contracts (new, amend, renew, annex); equipment (single and bulk registration); visits (calendar, unassigned, batch printing); service requests, payments, tax invoices, reports and admin screens. Tax invoices are PDFs made elsewhere and uploaded by staff (client answer D.1).

Technician app

A mobile-first PWA: today, upcoming, visit detail and a five-step completion. Customer phone numbers are hidden; there is only a "call head office" button. With no connection, or if it drops mid-send, the completion is put in an IndexedDB queue.

Customer portal

Equipment, visit history, payments, tax invoices, service requests (with a message thread) and contacts. One contract owner and several operating contacts each have their own account.

Legacy program

The dBase-format management program the client used (one row per appliance). The converter never writes to the database: it produces the migration workbook and feeds it through the office app's check-then-commit path. Re-uploading skips whatever is already in.

Reverse proxy

Caddy 2 attaches Let's Encrypt certificates automatically and serves all three areas on one domain. Each area uses its own JWT.

CI · deploy

On every PR and push to main it runs migrations, the type check, tests and the build; when green it builds an image, deploys over SSH and runs a health check.

Scheduled jobs

Timers hold the cash-handover warning, overdue reminders, consumable due dates, rental renewal and completion, day-before visit notice, monthly billing and a nightly backup. Every run writes a CRON_RUN audit row.

App server

Pages, API and middleware are one Next.js app. The three areas use separate JWT audiences, cookies and session stores, so one browser can be signed in to all three at once. Contracts, visits, payments and service requests move through pure-function transition tables, and every update pins the previous state in the WHERE clause to stop concurrent edits.

SMS provider

Only urgent notices, such as the day-before visit reminder, go by SMS. Development uses a mock adapter.

PostgreSQL 16

PostgreSQL 16 with Prisma 7. Appliances are never deleted; only their status changes (replaced, relocated, deactivated, terminated).

Email provider

All other notices go by email, in the recipient's language (the contract owner or the main operating contact).

`src/lib/equipment/asset-code.ts` · A per-model lock keeps concurrent registrations of the same model from taking the same number, and the next number comes from the largest code compared as a number. (comments condensed from the source; prefix changed)

```
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` · The validation plan is a pure function with no database access, so preview and commit run the same code, and applying a workbook is one transaction. (the headers of the two modules)

```
// 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.
 */
```

Key decisions

Of the five decisions, four (technician screen, technician assignment, where the data is kept, migration scope) were set by the client answering my questions, and all five are written up with their options and costs.

A PWA for technicians, with an offline queue `C.3` Client side

Problem

Technicians have to record work, photos, signatures and payments even where there is no signal, such as an underground car park.

Options considered

✓ PWA: runs in the browser, no app store, faster to launch

Native iOS and Android apps: smoother photos, signatures and offline use, but 3–6 more months to launch

Decision

The client chose the PWA (C.3) and, for offline, "online first, queue later" (C.4). In practice the offline queue went in anyway, in the last phase of the v1 scope. Pending work is stored in IndexedDB, the service worker is hand-written, and an offline badge shows on screen. Completing a visit takes five steps: work and parts → photos → signature → payment → confirm. Following the client's answer (E.1), the signature is a photo of a signed paper form.

Result

If the device is offline when a completion is submitted, or the connection drops while sending, the app puts the completion in the queue and returns to the screen.

Cost & remaining risk

As the question stated, photos, signatures and offline handling may be less smooth than in a native app. Maps and route optimization were deferred (C.5).

Technician assignment: the system suggests, the office confirms with one click C.1 Client side

Problem

The office was choosing by hand which technician to send on each visit.

Options considered

Manual assignment by the office (the current way)

✓ System suggests by area, workload and skill; the office confirms (recommended default)

Fully automatic assignment

Decision

The client chose suggest-and-confirm and asked for a "preferred technician" field per customer (C.2). A visit has one lead technician and any number of assisting technicians (K.3). Only the lead can complete, collect the signature and take payment; assistants add photos and notes only. The score is +100 for the preferred technician, +30 for a matching area and -5 for each visit already booked that day, and an admin screen adjusts the weights.

Result

The assignment screen shows the top 3 by score, and the office confirms the lead and assisting technicians. Confirming moves the visit from "suggested" to "scheduled".

Cost & remaining risk

Area only adds to a technician's score as a preference (client answer C.2). There is no map-based routing; the map screen is only a placeholder (C.5).

Data stays in the country: a self-hosted local VPS H.1 Client side

Problem

The first plan was a managed cloud. Where the data had to live decided how it would be hosted.

Options considered

✓ Must be in the country: local hosting

Regions abroad allowed: managed cloud (recommended default, faster launch)

Decision

The client chose in-country storage. On a local VPS, Docker Compose runs Caddy (automatic TLS), the app and PostgreSQL 16, and systemd timers hold the scheduled jobs and the nightly backup.

Result

Merging to main builds an image only when CI is green, then deploys it over SSH and runs a health check. One domain serves all three areas.

Cost & remaining risk

The servers, certificates, scheduled jobs and backups that a managed cloud would look after have to be run in-house.

Equipment codes: three rules, ending at a per-model sequence issued only by the server A.3.1

Problem

Appliances come in through several paths, such as single and bulk registration. Codes could not collide, even when the same model was registered from several paths at once.

Options considered

A number based on the customer code (first plan)

Model code + install date + sequence, unique across the table (Sep 4)

✓ Fixed prefix + 6 digits, sequence per model (Sep 7)

Decision

A fixed prefix plus 6 digits, with the sequence counted separately for each model. What is unique is the (model, code) pair. A code cannot be edited or reused, and only the server issues codes, on every registration path. The bulk-registration design had an "automatic or manual entry" choice; it was dropped, leaving server-issued codes only. So that concurrent registrations of the same model never take the same number, each model

gets an advisory lock, and the next number comes from the largest code compared as a number.

Result

On Sep 7 a migration re-issued the codes of existing appliances under the new rule. On Nov 2 the serial number field was removed, and appliances are now told apart by equipment code alone.

Cost & remaining risk

A code alone does not identify an appliance; it has to be read together with the model. PostgreSQL 16's default unique index treats NULLs as distinct, so it cannot stop duplicates among appliances with no model; a code comment records that the lock keeps those unique.

Migration: full history, check before commit, one workbook format `J.1`

Client side

Problem

The customer, contract, equipment, consumable and payment history in the old program had to move over. The old data had one row per appliance, so it had to be grouped by customer.

Options considered

✓ Migrate the full history | Start with new data; look up history in the old sheets | Active customers only (recommended default)

Decision

The client chose the full history. The migration screen came first, on Sep 24: a template, an export of current data, and two steps, "check (no save) → commit". The validation plan is a pure function that never touches the database, so preview and commit run the same code, and applying a workbook is one transaction. On Oct 20 the converter for the old program was built not to write to the database at all: it produces the same migration workbook and sends it down that check-and-commit path. Files are split per customer because of an upload size limit, re-uploads skip what is already in, and every corrected value is reported with a count per kind and examples. The switch date was set to Oct 21.

Result

The Oct 21 production run had 0 pre-check errors across 7 files and took about 8 minutes, and no notifications went to customers during it. The flow is shown below. Three kinds of item for staff to check (operating contact, contracts without a monthly amount, appliances without consumable-replacement records) are gathered at the top of the customer screen.

Cost & remaining risk

The full consumable and inspection history, outstanding balances and accounting records were not moved; they are looked up in the old program.

Switching over from the old program

The old program's data never went straight into the database; it passed the same check-then-commit screen the staff use, and the production run took about 8 minutes with zero check errors.

1. **Scope decision** Person
   Full history (the default was active customers only); duplicates auto-matched, then reviewed
2. **Migration screen** System
   Template, export, check then commit; preview and commit share the code; one transaction per workbook
3. **Converter** System
   Writes the same workbook, never the DB; split per customer; re-uploads skip what is already in
4. **Rehearsal** System
   Seven files into an empty DB through the real API: applied in 51 seconds, 0 errors
5. **Production run** System
   Merge, auto-deploy, backup, 7 files checked with 0 errors, about 8 minutes to apply
6. **Items to check** Person
   Three kinds gathered at the top of the customer screen: operating contact, contracts without a monthly amount, appliances without consumable records

Quality and verification

Stateful records move through state machines, production data jobs sit behind layered safeguards, and all 1,263 automated tests passed in the Nov 6 CI run.

1,263 automated tests (168 Vitest files) all passed in the Nov 6 CI run on main, along with the type check and the build. Per the work log, the count grew from 418 on May 27 to 656 on Jul 10 and 736 on Jul 21. By file there are 95 unit, 41 integration (against the PostgreSQL service in CI) and 32 component test files; that split is a static count of files.

Anything with a status moves through a state machine. Contracts, visits, payments and service requests use pure-function transition tables, and every update pins the

previous state in the WHERE clause. If two people change the same record at once, one of them gets an error.

Deleting or loading production data has layered safeguards. It runs only by hand, needs a typed confirmation, takes a pg_dump just before, and runs as one transaction. Seeding is refused on the production database. A migration upload is applied only after everything has been checked and COMMIT is typed.

The July bulk-registration work left sub-agent development records. Across 28 tasks, review returned "approved" 17 times and "needs fixes" 3 times (re-reviewed after the fix); of 4 end-of-phase reviews, 1 was conditional and passed after fixes. In the switch of cycles to days, review caught a bug that added months to a day value and made cycles 30 times too long.

One feature, start to finish

This follows one story from a client question to the production run: the switch from the old management program.

1. **Question and decision — J.1, J.2 (May 26)**

   The client chose to migrate the full history (the recommended default was active customers only). Duplicates are matched automatically and then reviewed by a person (J.2), and verification is shared between the client's office admin and the development side.

2. **Real use begins — production data reset (Sep 24–25)**

   A workflow clears the test data from the production database. It has a typed confirmation, a dump just before, a single transaction, and gives the same result however many times it runs. After a local rehearsal it ran in production on Sep 25.

3. **Migration screen (Sep 24)**

   A workbook template, an export of current data, and the two steps check → commit. The same day the screen was narrowed to admins (ADMIN) only.

4 **Converter for the old program (Oct 20)**

The mapping of the old tables was settled by comparing the program's own CSV export against the source rows. A contract debit-day column and "needs checking" warnings were added. The same day the migration rules were confirmed with the client. When grouping appliance rows into customers, companies are matched by name and households by name plus phone or address. Terminated customers move too, and small old amounts are treated as USD, with the amount left blank and a note added.

5 **Rehearsal (Oct 20)**

One catalog file and seven migration files went into an empty database through the real API in about 1 minute in total, 0 errors. Uploading the same files again skipped everything. Assuming a switch on Nov 1, the scheduled jobs were run for that date, confirming 0 overdue items and 0 reminders on switch day.

6 **Production run (Oct 21)**

The PR was merged in the morning, which auto-deployed and added the new column. After a production database backup and an integrity check, the run went export → convert → check 7 files (0 errors) → apply. Applying took about 8 minutes. No notifications reached customers during the run, and no portal accounts were opened automatically.

Retrospective

The equipment-code rule changed three times, and the deploy chain failed often in its first four days.

The equipment-code rule changed three times. It began as a number based on the customer code; on Sep 4 it became model code plus install date plus sequence, unique across the whole table; on Sep 7 it was settled as a per-model sequence. On Nov 2 the serial number field was removed, and appliances are now told apart by equipment code alone.

The deploy chain failed most often in its first four days (May 31 to Jun 3). Problems such as CI wait permissions, Docker runtime dependencies and Caddy TLS were fixed one PR at a time.

How I work

This project was also built with a role-based AI agent team. The team and its gates differ from project to project; this one's are under "My role" above. Below is the method as written up while building the EWA platform.

See how I set up and ran the work on the 2026 EWA platform →

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

Contact

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