CASE STUDY · CTO · PLATFORM REBUILD

Rebuilding HUNDO: website, iOS app, API, admin panel and AI, as CTO.

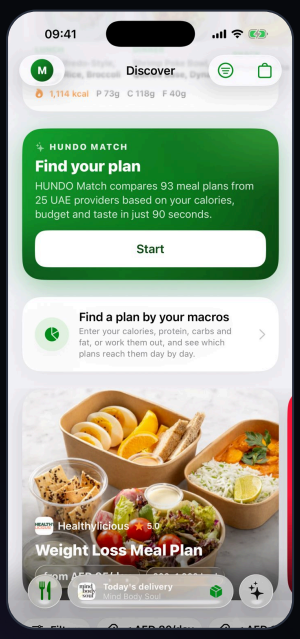
HUNDO is a meal-plan marketplace in the UAE. In September 2026 its owner asked me to replace a system a previous agency had built, and to stop partner kitchens' menus being typed in by hand. I led discovery, made the technical decisions and built the platform with AI coding agents.

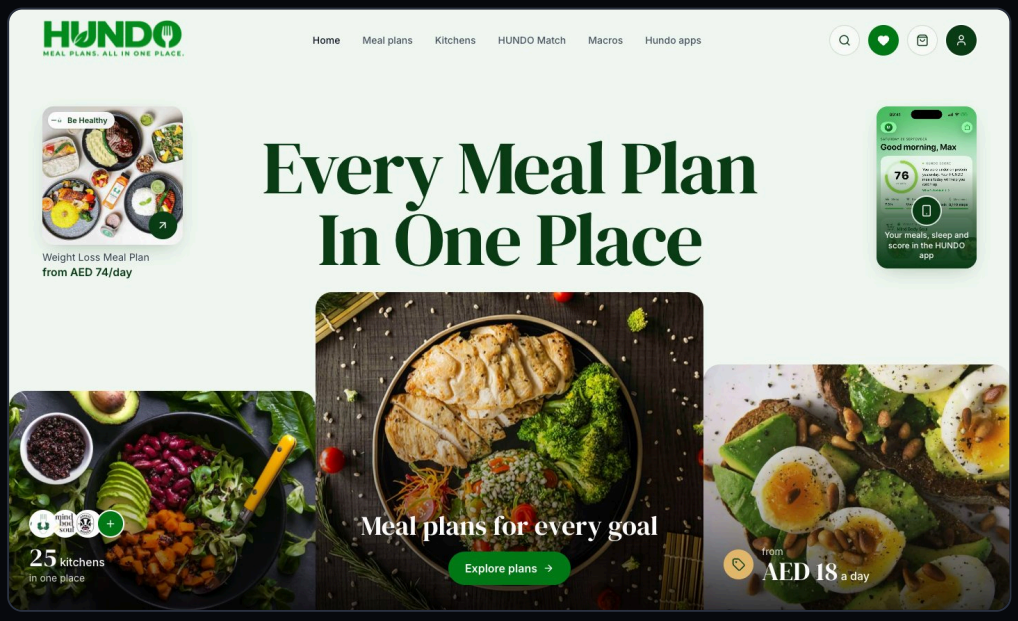
Day 1
first commit, plus a read-only extractor for the old live API

11 days
to a working admin panel demo with AI menu extraction

25 days
to the new iOS app in TestFlight

123 tables
PostgreSQL 17, 16 versioned migrations





DISCOVERY AND DECISIONS

From a handed-over system to a clear plan

What the client brought

A live system from a previous agency (a Laravel API, admin portal and Next.js site on Heroku, plus iOS and Android apps), a request to recreate and improve it, and a goal: menus arriving automatically from partner kitchens.

What I clarified

The old agency had finished and there was no database handover. On day one I built a read-only, resumable extractor (GET requests only) against the live API, then mapped what the system really did: how kitchens, menus, meal plans, orders and payments fit together.

What I decided

- A new Next.js and PostgreSQL backend, with an operations panel for staff and kitchens.
- One customer API shared by the iOS app and the website, which also serves the old App Store app through compatibility routes during the switch.
- AI menu extraction from the menus kitchens already have, with human review and hard rules.
- The new iOS app ships as version 2.0.0 of the existing App Store listing, so customers and ratings carry over.
- One rehearsed switch-over night: freeze, final import, reconciliation and a go/no-go, production settings, smoke tests, DNS, with the app approved in advance and held for manual release.

ARCHITECTURE

iOS app
SwiftUI, Swift 6, iOS 18; Stripe PaymentSheet

Website
React 19, Vite 8, react-router; Stripe Payment Element

Old App Store app
v1.0.55, served through compatibility routes

Operations panel
Next.js 16 App Router, VVS Panel, four roles

↓ bearer tokens · `/api/app/v1` (thin route handlers → `server/app-api`) · server actions for the panel

Next.js API and panel
TypeScript, zod, Drizzle ORM, jose, sharp, poppler; extraction runs after the response

PostgreSQL 17
123 tables in 17 schema files; advisory locks for orders

Cron container
publish-due and club every minute, menu-fill every 15 min, retention hourly

↕ integrations

Claude
menu extraction, nutrition estimates, photo checks

Stripe
orders placed on the webhook

Resend
through a recorded outbox

Apple and Google
sign-in

Kitchen websites
collector, robots.txt respected

TIMELINE

24 SEP
Brief, first commit, extractor for the old live API

25 SEP
Old data imported into the new database

5 OCT
Panel demo live, extracting menus with Claude

13 OCT
Written go-live plan

19 OCT
iOS app 2.0.0 uploaded to TestFlight; customer API merged for production

Weeks of work, done in hours

I build with AI coding agents under a strict review-and-test process, so things that take most developers weeks take me hours. At HUNDO: first commit on 24 September, a working admin panel demo by 5 October, the go-live plan by 13 October, and the new iOS app in TestFlight by 19 October.

CUSTOMER API AND DATA

Built for real traffic, real money and a clean switch-over

```
# /api/app/v1
auth   code/request · code/verify · apple · google
· logout · DELETE account
public catalog · content · meta
me     profile · questionnaire · addresses ·
devices
orders checkout/quote · orders · per-day menu ·
skip/unskip
       pause/resume · cancel · checkout/by-key
more   wallet · notifications · stripe webhook ·
club
```

Sign-in	6-digit e-mail code (10 minutes, 5 tries), Apple or Google. Rate limits per address and per network
Tokens	Opaque bearer tokens; only a SHA-256 hash is stored. Sliding expiry: 90 days idle, 1 year at most
Orders	Order changes take a `pg_advisory_xact_lock` with an 8-second lock timeout and answer `409 order_busy` instead of piling up; capped at 12 per 5 minutes and 40 per hour; `POST /orders` needs an Idempotency-Key
Payments	The server prices the basket; Stripe Payment Element on the web, PaymentSheet on iOS (card, Apple Pay, Google Pay); orders are placed when Stripe's webhook confirms
Roles	superadmin, kitchen, kitchen assistant (read-only), personal trainer
Enforcement	Three layers: a signed session cookie checked in the proxy, `requireRole()` on every page and action, and every query wrapped in `scope()`, so a restricted role with no matching column gets no rows. Role tests run against a real database
E-mail	Every message is written to an outbox first; it is sent through Resend only when live mode, keys, sender, public URL and an allow-list all agree
Photos	One pipeline for every source: format from the bytes, sRGB, metadata stripped, blur and brightness checks, focal-point crops into square, card, hero and thumb in WebP and JPEG; AI-generated photos are never used
Exports	Streaming CSV with a BOM for Excel and a formula-injection guard; columns withheld by role; every export audited

Data migration

- The importer reloads imported tables in foreign-key order and never touches panel-owned ones.
- Demo data is anonymised deterministically; the real-people mode refuses to run against the demo database.
- New-system ids start at 10,000,000, so they can never collide with an old id on a re-import, and the importer refuses an extract that reaches that range.

124
checks in the customer API smoke script

Burst-safe
13 simultaneous changes to one order used to exhaust the pool; now they queue or get a clean 409

4 roles
enforced on the server, not by hiding buttons

AI MENU INGESTION

Any menu, in any format, into structured data

Input
PDF, image, XLSX/CSV, text or a public URL (private addresses refused), up to 25 MB

Prepare
PDF → document block, image → image block, sheets → text tables; text over 400k chars refused, never truncated

Extract
Claude (claude-opus-5), streaming, adaptive thinking, structured output from a zod JSON schema, cached system prompt

Validate
weeks, days and ranges checked; every problem reported with its JSON path

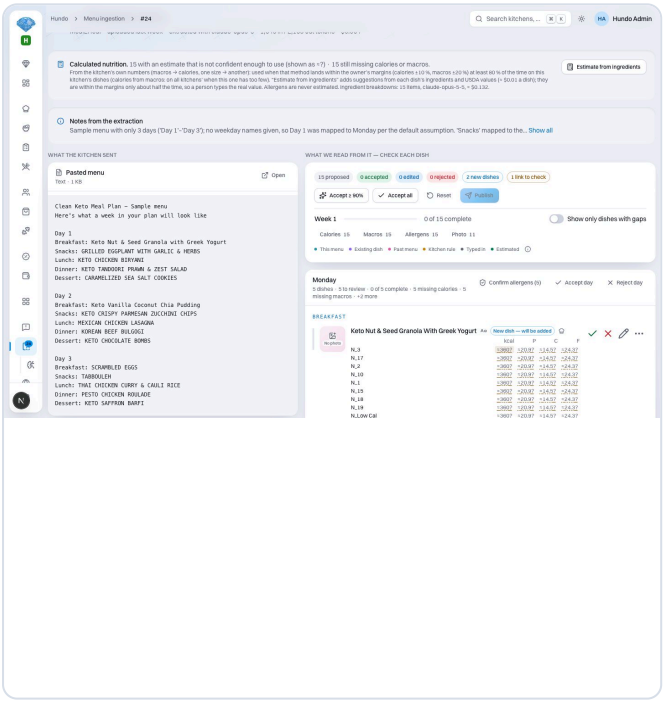
↓

Match
exact names, else trigram Dice + Jaro-Winkler (threshold 0.72); meal types by alias or ≥ 0.85

Confidence
model score calibrated, multiplied by the match; a new dish never reaches 0.9

Review
source beside proposal; accept, edit or reject per item, day or week; "accept ≥ 90%"

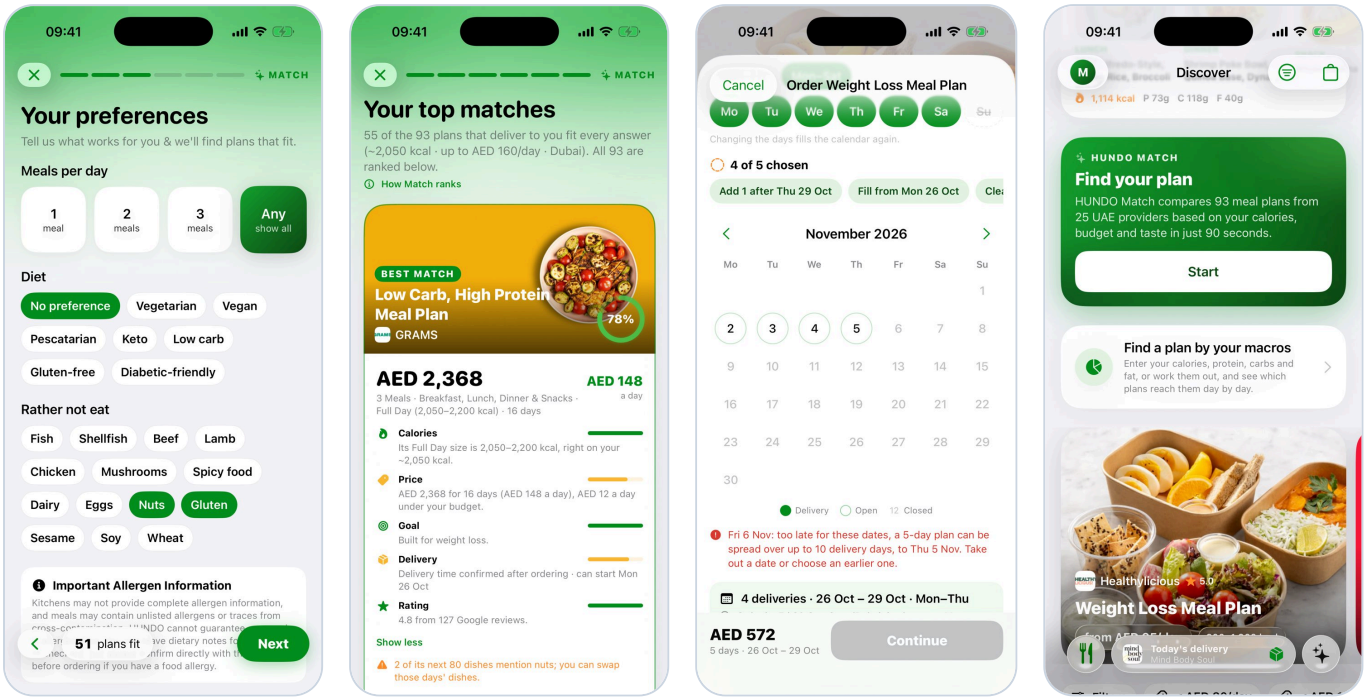
Publish
one transaction under row locks; auto-publish only through a 6-rule gate



- **Learns from reviewers:** every edit is stored per kitchen; the newest 12 go into the prompt.
- **Costs known up front:** about 1,800 tokens per PDF page, 1,500 per image and 90 output tokens per dish are estimated before a run; actual usage is shown after.
- **No lost updates:** publish, enrichment and review take row locks in the same order as the menu builder, and a fingerprint plus item version refuse writes made from stale data.
- **Allergens are never guessed:** inferred allergens need a person's confirmation. Nutrition calculated from ingredients (USDA SR Legacy, Claude) goes live only marked "estimated", and only where the method proves accurate on the kitchen's own dishes (targets: ±10% calories, ±20% macros).
- **Menu sources:** a collector respects robots.txt, reads Shopify product feeds, renders JavaScript pages in headless Chromium, skips unchanged pages by content hash and stops at a monthly budget.

APPS, DELIVERY AND QUALITY

Customer app, website, deployment and tests



iOS app

- SwiftUI, Swift 6 with main-actor isolation by default, iOS 18, generated by XcodeGen.
- `APIClient` on URLSession with typed errors; token in the Keychain.
- A bundled catalogue is the first paint and offline fallback, refreshed by ETag at most every 5 minutes; an empty server answer never replaces a working catalogue.
- HUNDO Match ranks plans by weighted score (calories 0.22, price 0.20, taste 0.18, goal 0.12, meals 0.10, delivery 0.10, rating 0.08); the top three come from three different kitchens, and commission plays no part.
- Delivery calendar rules ported from the website's code: an N-day plan within its first 2N delivery days, kitchen start days, cut-offs and holidays.

Deployment

- Production VM with four containers: PostgreSQL, the Next.js API, a cron runner and Caddy serving the website and forwarding only the API and media paths; firewall open to Cloudflare only.
- Deploys build only from `origin/main`, refuse if local main is ahead, run migrations before the new API starts, and record the running commit in a REVISION file.
- Hourly backups with a tested restore; the TestFlight script refuses uncommitted or unpushed work.

Tests and reviews

- Panel: 81 test files, about 990 tests (Node's test runner; role tests against PostgreSQL). Website: 25 Vitest files, about 507 tests. iOS: 220 Swift Testing tests.
- Independent review passes: one confirmed 36 issues in the menu system; a nine-reviewer pass on the app raised 84 findings.
- 357 of 368 commits co-written with Claude Code.

Size: panel 111,374 lines of TypeScript in 894 files · website 29,388 · iOS 22,405 lines of Swift in 96 files · design system 9,067 · documentation 9,849 lines.