

KNOWLEDGE TRANSFER / HANDOVER

Prima Fresh Mart Scanner

app-pfm-ocr-v2

On-premise, GPU-accelerated OCR that turns a store staff member's phone photo of a Delivery Order into structured, database-backed data. Flutter mobile client + Dockerized Next.js / PaddleOCR / vLLM / Postgres backend.

PREPARED BY

[TO FILL IN] (outgoing engineer)

HANDOVER DATE

[TO FILL IN]

How to use this deck

Everything in here is drawn from what is actually committed in the repo — no invented features. Where only the outgoing engineer knows something (accounts, secrets, contacts), the slide says [TO FILL IN] instead of guessing.

Read in this order

1. README.md — architecture, accuracy, install, gotchas
2. This deck — the guided tour of the above
3. CLAUDE.md + AGENTS.md — how work is done in this repo
4. backend/CLAUDE.md — before touching anything in backend/
5. docs/workflow_detail_aplikasi.md — field-by-field rules
6. plans/next-enhancements.md (x2) — the live backlog

The deck is in three parts

Slides 3–16 WHAT EXISTS — architecture, code, data, accuracy

Slides 17–19 WHAT WILL BITE YOU — gotchas + access checklist

Slides 20–25 WHAT'S NEXT — the written-down backlog, scope

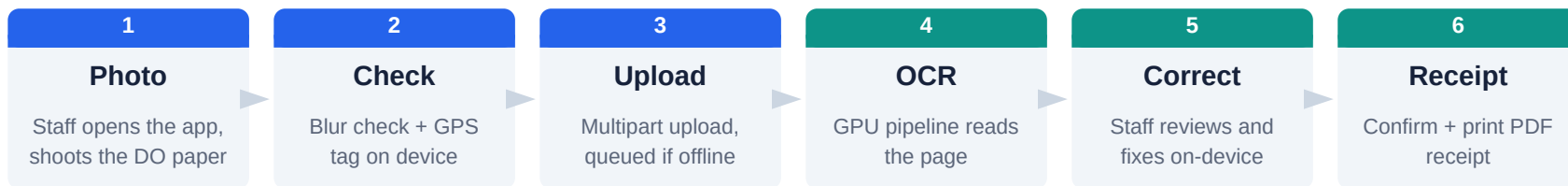
Every diagram is a native Impress/Draw shape — click and edit it, nothing is a flattened image.

This repo is documentation-heavy on purpose. If this deck and a doc in the repo ever disagree, the repo wins — it is the one that gets updated by whoever ships code.

Docs that already exist: README.md (305 lines) · docs/ (8 files, ~2,300 lines) · backend/docs/ (4 files) · plans/next-enhancements.md (539 + 522 lines).

What the system actually does

One sentence: a store staff member photographs the paper Delivery Order when a delivery arrives, and the system reads it into the database so nobody re-types it.



Who the user is

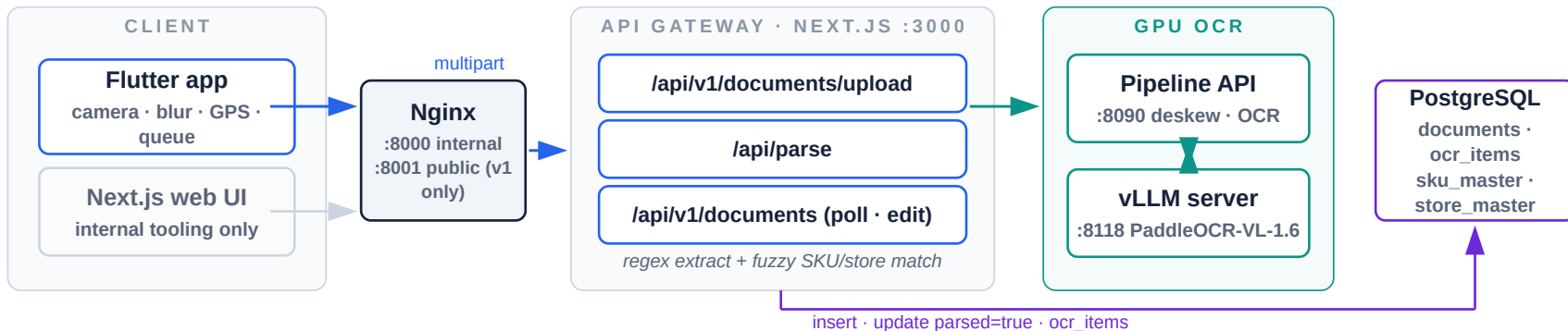
Store staff (petugas toko) on duty at a Prima Fresh Mart store — NOT the delivery driver.
Each login account is bound to exactly one store (accounts.role = 'store'), so whoever is on shift there uses the same login.

Two clients, one backend

Flutter app (lib/) — the real, store-facing product.
Next.js web pages (backend/pfm-web-app/src/app/*.tsx) — internal tooling only: manual labelling, accuracy comparison, OCR 'arena'. Not shipped to stores, no login screen, and never will have one.

No cloud OCR API is used anywhere. Layout detection, text recognition and LLM structuring all run on your own GPU — that is a deliberate client requirement, not an implementation detail.

System architecture — the boxes and the ports



SECONDARY FLOW — PRODUCT SCAN:

single-product photo → `classify_ocr_server.py` on :8120 → DINOv2 embedding similarity search (YOLO classifier as fallback) matches the SKU, PaddleOCR reads the expiry date. Same Nginx entry point, separate service.

Single GPU pipeline, no horizontal scaling: there is one pipeline-api and one vllm-server container. Concurrent uploads queue behind the GPU — this is a real throughput ceiling, load-test it before any multi-store rollout.

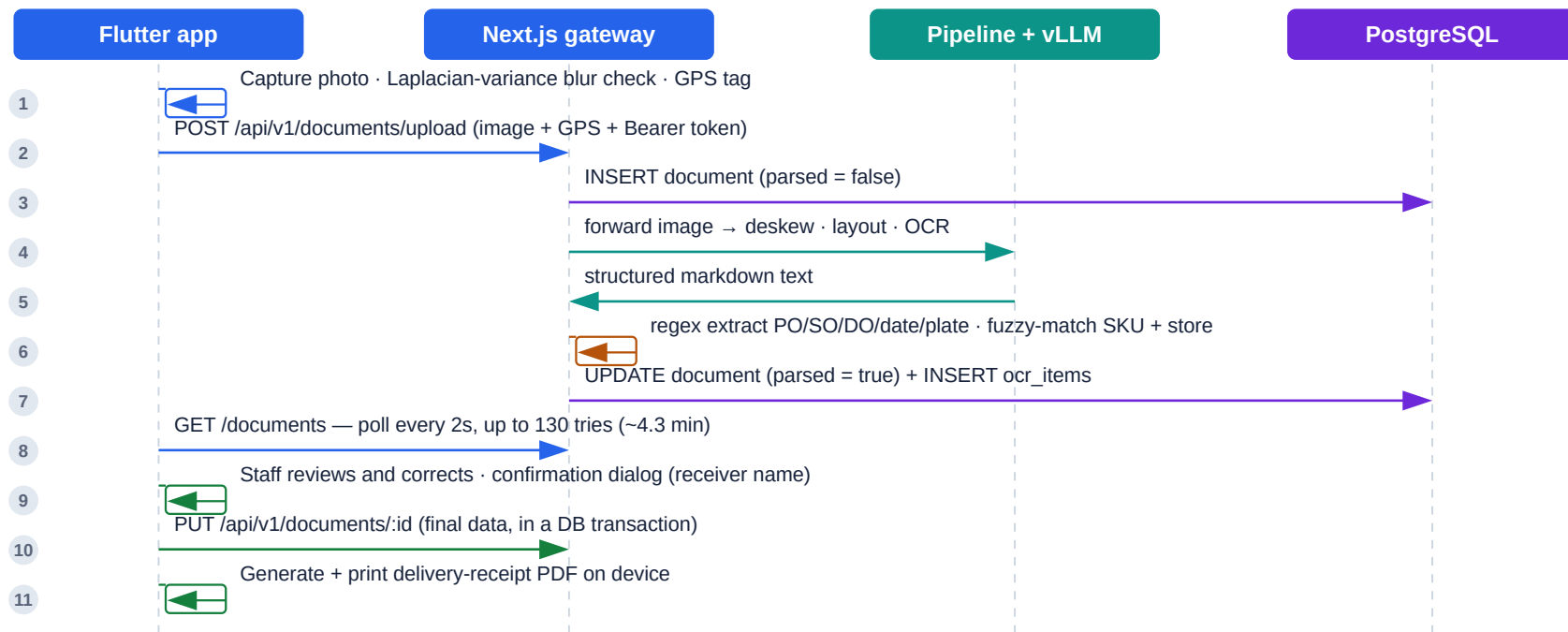
Source: README.md § Architecture. Nginx has two server blocks: :8000 exposes the full internal surface, :8001 proxies only /api/v1/ and 404s everything else — that is the one the ngrok tunnel points at. Everything except the Flutter app runs in one Compose stack.

Who is responsible for what

If a bug report lands, this is the table that tells you which container or package to open first.

TECHNOLOGY	WHERE IT LIVES	ROLE
Flutter (Riverpod, Dio, Hive, go_router)	lib/	Camera capture, blur detection, offline-tolerant upload queue, correction editor, native PDF/print
Next.js App Router (TypeScript)	backend/pfm-web-app/	Auth, upload handling, orchestrates the pipeline, regex post-processing, fuzzy SKU/store matching, internal web tooling
Nginx	backend/nginx.conf	Single entry point on :8000, routes to gateway / pipeline / vLLM
PaddleOCR v6 + PP-DocLayoutV3 (FastAPI, GPU)	backend/config/	Auto-deskew and unwarp, layout segmentation, text detection and recognition
vLLM serving PaddleOCR-VL-1.6-0.9B	backend/config/vllm_config.yaml	Reassembles OCR text fragments into coherent structured text
PostgreSQL 15	backend/db/ + src/db/init.ts	Documents, line items, SKU / vendor / customer / store master data
DINOv2 similarity search + PaddleOCR	config/classify_ocr_server.py	Product Scan: photo → SKU match via nearest-embedding lookup + expiry extraction
Docker Compose (multi-stage GPU Dockerfile)	docker-compose.yml (repo root)	Runs the whole backend as one stack

The DO scan, end to end



Source: README.md § How It Works + lib/features/documents/pending_documents_provider.dart. The poll loop is 2s x 130 attempts before it surfaces a timeout error.

The mobile app — structure

lib/ is the root Dart package. One folder per feature; models and core infrastructure are shared. Tests live flat in test/, not mirrored to lib/.

lib/config/	app_config.dart — theme + dynamic API base URL resolution
lib/core/	Dio client, Hive storage, geolocation, go_router
lib/features/auth/	real login against /api/v1/auth/login, token storage
lib/features/camera/	capture, blur check, image preview, gallery pick
lib/features/documents/	pending-upload queue, documents list
lib/features/editor/	correction form, confirmation dialog, PDF receipt
lib/features/splash/	cold-start auth check and routing
lib/models/	DocumentModel and friends
lib/data/master_sku.dart	bundled static SKU map (see gotchas)

Commands you will use daily

```
flutter pub get
flutter run
flutter test (all) · flutter test test/x.dart
flutter analyze lib
flutter build apk --release
```

State & navigation

Riverpod providers per feature: authProvider, pendingDocumentsProvider, scanModeProvider. go_router for routing, Dio for HTTP, Hive for on-device persistence of the pending queue and the cached document list.

The five parts that are not obvious

Read these before changing anything in the app — each one has already caused a field problem or is documented as a known gap.

1. Dynamic API base URL

`AppConfig.initializeApiBaseUrl()` resolves the backend at startup — it tries a fixed reserved ngrok domain first and falls back to a hardcoded LAN IP. It runs ONCE, at startup, with no mid-session failover. If a device can't log in and the backend is healthy, check this first.

2. Blur check is cosmetic today

`ImagePreviewScreen` computes Laplacian variance and shows a 'Foto Terdeteksi Blur!' badge, but the Upload button is gated only on `!_analyzing` — a photo flagged blurry can still be uploaded. Backlog item 2.4.

3. The pending queue is persisted

As of 2026-07-08 the upload queue lives in a Hive box (`lib/core/storage/local_storage.dart`). An OS-level app kill mid-upload no longer loses the document — on next launch the queue rehydrates and resumes anything non-terminal. Upload dedupes server-side by file hash, so a re-run is safe.

4. Polling, not push

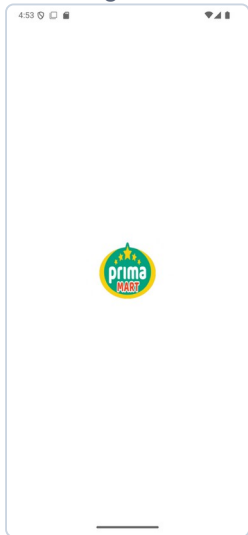
After upload the app polls `GET /documents` every 2s for up to 130 attempts. Any exception inside a poll iteration is swallowed and retried, so a dead Wi-Fi connection eventually reports as 'Gagal mengekstrak data (Timeout)' — a real diagnosis trap. Backlog item 3.5.

5. SKU validation is offline

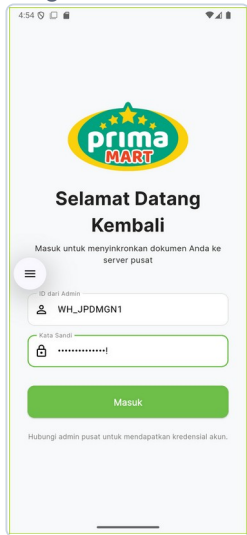
The DO editor validates item SKUs against `lib/data/master_sku.dart` — a Map compiled into the app — not the live `/api/v1/master/skus` endpoint the Product Scan flow already uses. Central SKU changes need an app rebuild. Backlog item 4.5, and a blocker for the Stock feature.

The shell — login, permissions, drawer, logout

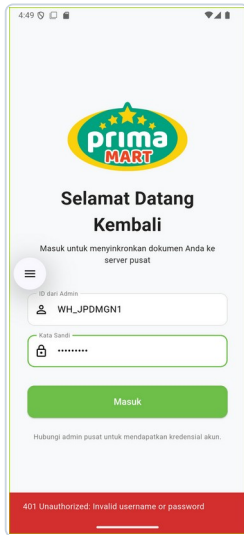
Captured on a real device against the live backend. These are the screens around the two scan flows: getting in, granting location, switching mode, and getting out.



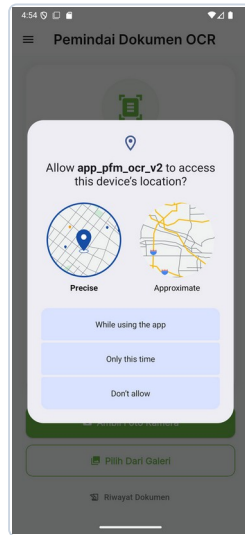
- 1 Splash. Runs ~3s of fixed delay BEFORE the auth check — backlog 1.5.



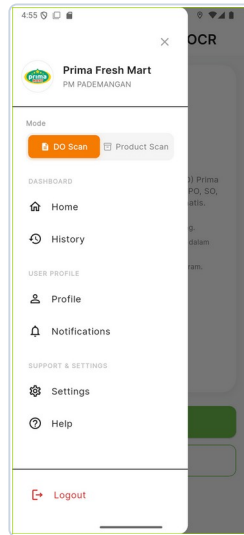
- 2 Login. Real bcrypt check against the accounts table.



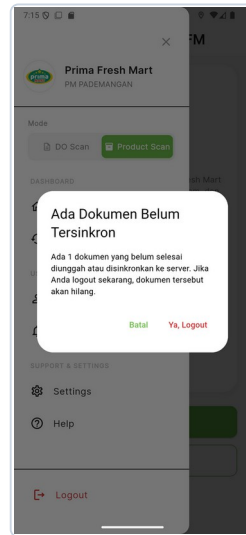
- 3 Wrong password — a real 401 from /api/v1/auth/login.



- 4 Location permission. Denial is only logged today — backlog 2.1.



- 5 Drawer. Mode toggle, help, logout.



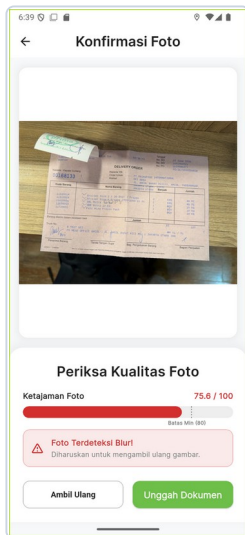
- 6 Logout warns about an unsynced queue.

DO scan — capture, upload, OCR result

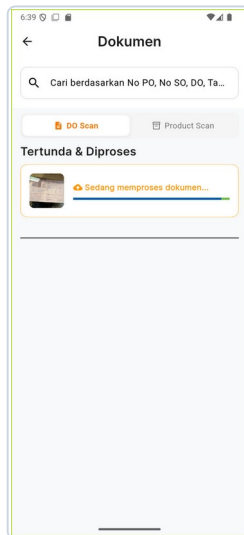
The core product loop, part 1. Steps 1–5 of the sequence in slide 6, as the store staff member actually sees them.



1 Camera home, DO mode. Blue = DO.



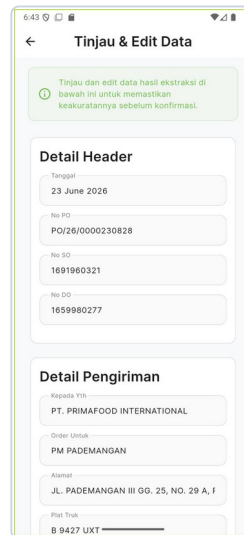
2 Real Laplacian blur check. The badge does NOT block upload — backlog 2.4.



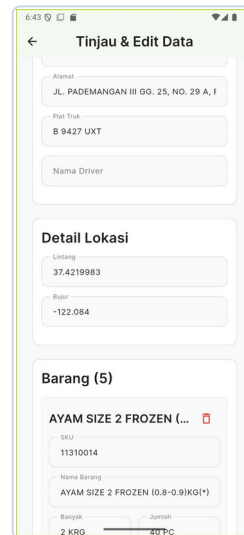
3 Queued. Persisted to Hive, survives an app kill.



4 parsed=true. The 2s poll found it.



5 Editor, header fields — real OCR output.



6 Line items after SKU/unit correction.

DO scan — validation, confirm, receipt

Part 2: what happens between a parsed document and a printed receipt — including the silent-validation failure recorded as a known limitation.

- 7 Simpan tapped, nothing happens — validation failed silently.

- 8 The same form once the error is surfaced.

- 9 Confirmation. Receiver name is free text — backlog 4.6.

- 10 Saved. PUT runs in a DB transaction.

- 11 History, DO tab.

- 12 The delivery receipt PDF, generated on-device.

The Next.js gateway — routes and the auth split

This is the single most important backend distinction to understand, and it is deliberate: there are two API surfaces with different security postures.

PRODUCTION SURFACE · /API/V1/*

- **Real JWT auth.** /api/v1/auth/login checks a bcrypt-hashed password against the accounts table and signs a token.
- **Enforced.** /api/v1/documents/* (list, upload, PUT-by-id) return a real 401 on a missing or invalid token.
- **One store per account.** Every account is bound to exactly one kode_toko.
- **This is what the Flutter app talks to**, always with Authorization: Bearer.

DEV-ONLY SURFACE · CLASSIC ROUTES

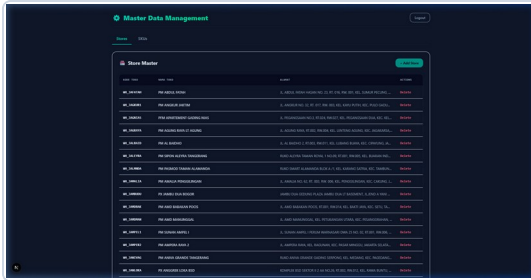
- **No auth, by design.** /api/upload, /api/parse, /api/history, /api/scan-pfm and the root / scan-pfm / manual-label web pages have no login flow and never will.
- **Internal tooling only** — labelling, accuracy comparison, engine 'arena'.
- **Not part of the store-facing product.** Don't build product features on these routes.

Every API route sets Access-Control-Allow-Origin: *, and nginx :8000 also serves the unauthenticated classic routes — so :8000 must never face the internet. The hardened :8001 block (only /api/v1/, 404 for everything else) exists precisely so a tunnel has a safe target; start-dev-tunnel.ps1 points ngrok at 8001, not 8000.

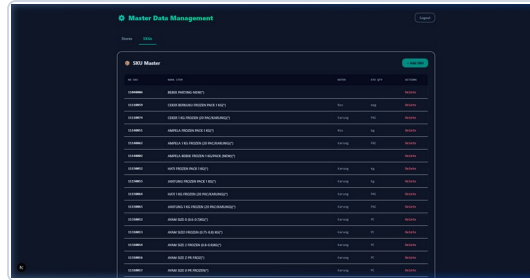
Source: backend/CLAUDE.md § How this project differs + README.md § What to Consider. Post-processing lives in backend/pfm-web-app/src/utils/parser.ts.

The internal web tooling (not the product)

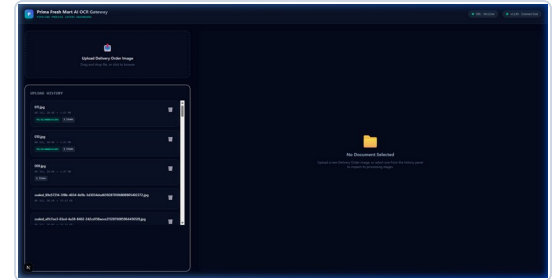
The Next.js pages from the previous slide's dev-only surface. The successor inherits these too — they are how master data is inspected and how ground truth gets labelled. No login screen, by design.



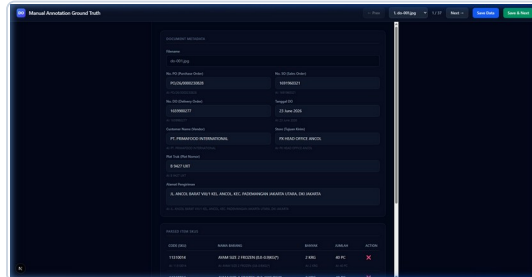
Admin — stores



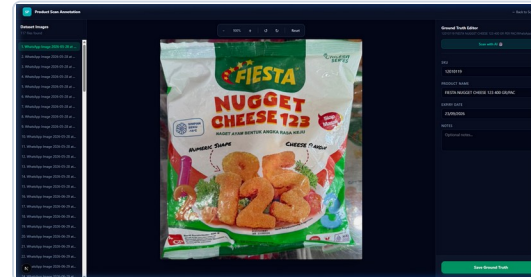
Admin — SKU master



Dashboard



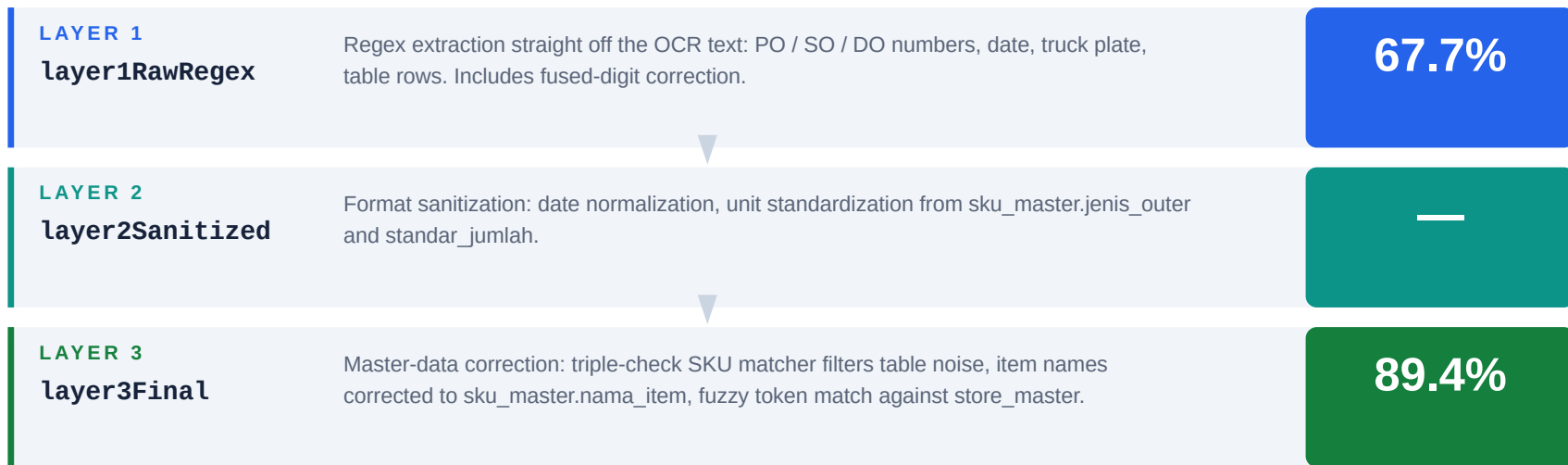
Manual labelling — DO ground truth



Manual labelling — product ground truth

Why accuracy is a three-layer story

The single most valuable thing to understand about this system: raw OCR + regex only gets you to 67.7%. The gain to 89.4% comes from correcting against master data.



Stage names are real: trace `layer1RawRegex` / `layer2Sanitized` / `layer3Final` into `backend/pfm-web-app/src/utlis/parser.ts` to see where each is produced. Layer 2's own percentage is not reported separately in `accuracy_history.jsonl`.

Current accuracy, field by field

Measured against 37 hand-labeled real DO photos. Overall 89.4% exact-field-match, up from an 82.2% baseline, against a 95% target.

FIELD	RAW REGEX	FINAL
Customer (Kepada Yth)	100%	100%
Kode Barang (SKU)	98.4%	98.4%
Item count	13.5%	97.3%
Nama Barang	0%	94.4%
Banyak (qty)	77.6%	95.2%
Jumlah (total)	76.8%	94.4%
No. PO	91.9%	91.9%
No. DO	91.9%	91.9%
Tanggal	89.2%	89.2%
No. SO	86.5%	86.5%
Store	0%	64.9%
Plat Truk	62.2%	62.2%
Alamat	0%	37.8%

Known gaps toward the 95% target

Alamat 37.8% — CAPPED. The ground-truth labels themselves use two different phrasings for the same address. Needs the ground truth unified, not more parsing logic.

Plat 62.2% — mostly genuine OCR misses (faint or absent truck line).

Store 64.9% — bounded by how much store_master data is loaded.

The lever nobody expects: photo SOP

Much of the test set was shot with the DO paper lying on top of other papers. That breaks auto-deskew: page tilt is estimated from the average angle of detected text blocks, and the sheet underneath corrupts it.

Tightening the field SOP (DO alone, flat contrasting surface, lit, squared) should raise these numbers with zero code changes. Today's figure is a floor, not a ceiling.

How you verify you haven't broken it

There is a real regression harness. Run it after ANY change to `parser.ts` — this is the closest thing this project has to a safety net.

The accuracy harness

```
cd backend
node pfm-web-app/scripts/accuracy-check.mts
# reuse cached OCR results (fast)

node pfm-web-app/scripts/accuracy-check.mts --refresh-ocr
# force a real pipeline re-run
```

Other test entry points

```
npx tsx backend/pfm-web-app/src/utils/parser.test.ts (no
Docker needed)
backend/pfm-web-app/run_batch_test.js
flutter test · flutter analyze lib (Flutter side)
```

What it does

Hits the live `/api/parse` endpoint for every image in `backend/sources/test-images/`, diffs the result against the hand-labeled ground truth in `sources/manual_labels.json` at all three post-processing stages, and appends the run to `sources/accuracy_history.jsonl` — so run-over-run movement is recorded, not just the latest number.

Human-readable reports

`compare_accuracy.py`, `compare_sources_accuracy.py` and `generate_excel.py` build `sources/comparison_report.xlsx` and `comparison_side_by_side.html`.
Analysis tooling — not part of the running app.

There is no single unified test runner across both halves. Flutter and backend are tested independently — remember to run both.

Product Scan — SKU match + expiry

A separate flow from the DO scan: photograph one product, get its SKU and its expiry date. Feature-complete on the backend as of 2026-07-08.



What exists today

Backend service, DB schema, both routes (scan-pfm, produk-pfm).
Reference photos: 81 SKU folders (from 16; target ~230).
models/dinov2_index.pkl — 2,493/2,493 photos indexed (2026-07-14).
YOLO fallback classifier — 85.8% top-1, 94.4% top-5 across 81 classes, retrained 2026-07-14 in 54m21s on an RTX 2060.
Desktop scan-pfm/page.tsx at feature parity, plus the Flutter flow.

Deliberate non-goals + open items

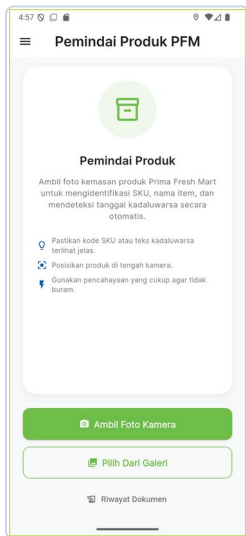
NO mobile web page is planned. m-scan-pfm/page.tsx and its nginx route are intentionally left dead — real mobile scanning goes through the Flutter app. scan-pfm/page.tsx is desktop-only, for testing.

Still not done: an actual browser pass uploading a photo through /scan-pfm end to end. Verified via container logs and model loading so far, not a UI test.

Full architecture, endpoints, classification internals and the retraining procedure: [backend/docs/scan-product.md](#) (214 lines).

Product Scan — the flow on screen

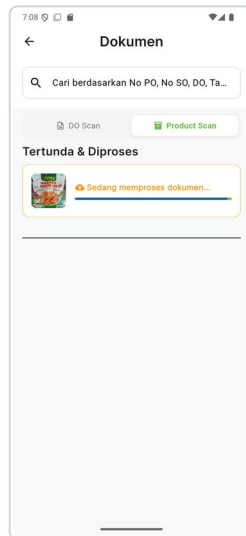
The secondary flow from the previous slide, as the staff member sees it. Teal chrome marks Product mode, the way blue marks DO mode.



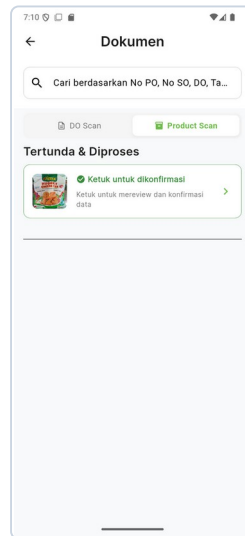
1 Camera home, Product mode.



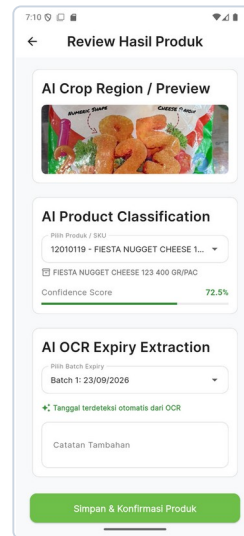
2 Single-product photo, same blur check.



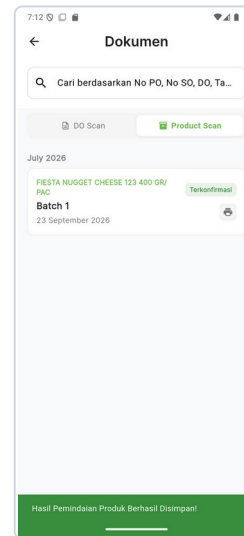
3 Queued for classification.



4 Classified and ready to review.



5 Real DINOv2 match + OCR expiry date.



6 Saved.
noSo/platTruk are still hardcoded placeholders — backlog 7.4.

The database and where schema really comes from

There is no manual migration step, and that surprises people. Read this slide before you go looking for a migration runner.

src/db/init.ts — THE SOURCE OF TRUTH

Runs idempotent CREATE TABLE IF NOT EXISTS + seed statements on EVERY app start. Schema and reference data (vendor, customer, starter SKU catalog) are created automatically on first request. It also hashes the seed password and migrates any pre-existing plaintext account rows on startup.

db/migrations/*.sql — FIRST-BOOT ONLY

Runs once via Postgres's own docker-entrypoint-initdb.d, but only on a brand-new volume. Useful as a record of intent; do NOT treat it as the live schema. If the two ever disagree, init.ts is what the running system has.

MAIN TABLES

documents

one row per scanned DO / product scan

sku_master

product catalog + packaging columns (standar_jumlah, jenis_outlet, ...)

accounts

login, bcrypt password, bound to one kode_toko

ocr_items

line items; rewritten inside a transaction on PUT

store_master

stores — SHIPS EMPTY, see gotchas

vendor / customer

master identities, seeded from init.ts

Running the stack — and the rule you must not break

Two modes exist and they are not interchangeable. This is written as a POLICY in CLAUDE.md, not a suggestion.

LOCAL DEVELOPMENT

`docker compose up --build`

Runs the gateway via npm run dev with source bind-mounted in — hot-reloads pfm-web-app, which is what you want while iterating on parser.ts. Single dev-server process with a known throughput ceiling.

DEMOS / FIELD TESTING — MANDATORY

`docker compose -f docker-compose.yml \ -f docker-compose.demo.yml up -d --build`

Drops the dev bind-mount, runs npm start against the image's own build output. REBUILD (--build) BEFORE EVERY DEMO — this mode does not hot-reload. Never demo on the dev server; it bottlenecks the moment two devices upload at once.

Host prerequisites

NVIDIA GPU, CUDA 12.6+ driver, ~8GB+ VRAM. Works on native Linux and on Windows + Docker Desktop with WSL2 GPU passthrough. 60GB+ free disk — pipeline-api and vllm-server images are ~30GB each.
First build is long. Budget real time for it.

First-time setup

`cp backend/.env.example backend/.env`
Set CUDA_VISIBLE_DEVICES (GPU index), APP_PORT if 8000 is taken, and JWT_SECRET — the example file ships with 'change-me'.
Verify with `curl :8000/health`, `curl :8000/v1/models`, and a POST to `/api/v1/auth/login`.

Repo map and how work is governed

The repo carries its own operating rules. Two separate governance zones — the Flutter root and backend/ — with separate backlogs that must not be mixed.

ZONE 1 — FLUTTER APP (REPO ROOT)

lib/, test/	the app
CLAUDE.md + AGENTS.md	the operating rules
SKILLS.md	review roles
plans/next-enhancements.md	the live Flutter backlog (539 lines)
docs/feature-list.md	shipped-feature log
docs/iteration-log.md	code-review + audit trail (908 lines)
docs/api-contract-map.md	Flutter/backend contract, gaps G1–G10
docs/workflow_detail*.md	field-by-field extraction rules
docs/stock-feature-plan.md	planned Stocks feature
docs/expiry-tracking-plan.md	planned per-sale expiry tracking
screenshots/v2/WORKFLOW.md	screen-by-screen app walkthrough

ZONE 2 — BACKEND (BACKEND/)

pfm-web-app/	Next.js gateway + internal tooling
src/app/api/	all routes
src/utils/parser.ts	regex extraction + sanitization
src/db/init.ts	live schema + seed
scripts/	accuracy tooling
config/	pipeline + vLLM YAML, classifier server
db/migrations/	first-boot SQL only
sources/	test images, labels, real master data
uploads/	real scanned DO photos + OCR JSON
CLAUDE.md + AGENTS.md	backend's own rules — read first
plans/next-enhancements.md	the backend backlog (522 lines)

backend/ has its own CLAUDE.md, AGENTS.md, SKILLS.md, feature-list, iteration-log and backlog — deliberately independent of the root ones. Never merge the two backlogs, and defer to backend/CLAUDE.md for anything inside backend/.

What will bite you — infrastructure

All of these are documented in the repo and several have already caused real incidents during this project's own testing.

Two docker-compose.yml files exist

`./docker-compose.yml` is canonical. `backend/docker-compose.yml` is a near-duplicate standalone copy with a different Compose project name (`ai-ocr-pfm-2026`), left from when the backend was its own repo. Running compose from inside `backend/` causes container-name conflicts with anything started from root — not hypothetical, it has happened here. ALWAYS run from the repo root.

The dev server is not demo-safe

docker compose up runs `npm run dev`. It has a known throughput ceiling and will bottleneck when multiple devices upload at once. Use the `.demo.yml` override for any client demonstration or field test, and rebuild first.

Mobile connectivity is two manually-synced paths

The ngrok domain in `app_config.dart` is fixed and reserved, but the ngrok PROCESS is not started automatically — you or `start-dev-tunnel.ps1` must launch it. The LAN IP fallback is hardcoded and goes stale silently the moment the host machine's IP changes. If a device can't log in and the backend is healthy, check these two first.

Single GPU, no horizontal scaling

One pipeline-api, one vllm-server. Concurrent uploads queue behind the GPU. Load-test before any multi-store or multi-device rollout — a single-user smoke test proves nothing here.

nginx.conf has dead routes

`/do-pfm` and `/m-do-pfm` location blocks are left over from the old per-route structure and have no matching Next.js route — they would 404. `m-scan-pfm` is intentionally dead too. Don't assume a route exists because nginx mentions it.

What will bite you — app and data

store_master ships EMPTY

The schema is created automatically but no store data is seeded. Fuzzy store matching (`resolveStoreFromText`) resolves nothing until you load real store data. `backend/sources/toko_aktif.json` looks like the right source but is not wired into an automatic import. This is why Store accuracy is capped at 64.9%.

The Android release build is debug-signed

`android/app/build.gradle.kts` still has `'// TODO: Add your own signing config'` and signs release builds with the debug key. Fine for internal installs, blocks Play Store distribution.

Request/response bodies are logged unconditionally

`LogInterceptor(requestBody: true, responseBody: true)` in `api_client.dart` is not gated behind `kDebugMode`, so bearer tokens, receiver names, GPS coordinates and DO contents are printed in release builds too — readable via `adb logcat`. Backlog item 5.4.

The LAN leg is plain http://

`_lanBaseUrl` is unencrypted, so the token and all document contents travel in the clear over the store's Wi-Fi whenever the LAN endpoint is active. The ngrok leg is HTTPS; the LAN leg has no equivalent. Backlog item 5.6.

Silent validation failure in the DO review form

If a required field (e.g. driver name) is empty, `Form.validate()` returns false and the confirm button's `onPressed` just returns — no toast, no scroll-to-error. The operator sees a button that does nothing. Listed as a known limitation in `README.md`.

Confidentiality: backend/ holds real client data

`sources/` and `uploads/` contain the client's real SKU / store / vendor / customer master data, real scanned DO photos and hand-labeled ground truth, committed to source. Read `backend/CLAUDE.md` § Confidentiality before exporting, logging or sharing anything.

Access and credentials checklist

The repo cannot tell you any of this — only the outgoing engineer can. Fill every row in before the last day, and verify the successor can actually log in to each one WHILE you are still reachable.

Infrastructure

- ☐ GPU host machine — address, OS account, SSH key or physical access
- ☐ Docker / registry access if images are pushed anywhere
- ☐ Who owns the machine, and who to call when it is down

Mobile distribution

- ☐ Android signing key (or confirm none exists yet — see gotchas)
- ☐ How the APK reaches store devices today
- ☐ The list of devices/stores currently running it

Secrets

- ☐ backend/.env — real values for JWT_SECRET, CUDA_VISIBLE_DEVICES, APP_PORT
- ☐ Postgres credentials and where the data volume lives
- ☐ Backup location for the DB — or confirm none exists

Accounts & people

- ☐ Admin login for the backend, and the per-store account list
- ☐ Client-side contacts at Prima Fresh Mart + escalation path
- ☐ GitHub/repo ownership transfer, and any CI credentials

Connectivity

- ☐ ngrok account + the reserved domain hardcoded in app_config.dart
- ☐ The LAN IP / network the stores actually connect over
- ☐ Any VPN or firewall rules involved

Recommendation: don't hand these over as a document. Sit with your successor and have THEM run each one — start the stack, log in from a real device, run the accuracy harness, build an APK. A credential you never watched someone use is a credential you have not actually handed over.

The Flutter backlog

From plans/next-enhancements.md — 10 sections, all written down, most still [TODO]. These are the section headings and a representative open item from each.

1. Auth & Splash

Structured 401 handling instead of string-matching; global 401/403 interceptor; splash runs its 3s of fixed delay BEFORE the auth call

2. Camera & Geotagging

Surface location failures to the user; fix-quality indicator; ENFORCE the blur check; flag gallery-sourced captures

3. Pending Queue

Pending-count badge on the camera screen; per-request timeouts; orphaned image files are never deleted; throttle queue processing on reconnect

4. Editor & PDF Receipt

Unsaved-changes guard; block save on unregistered SKU or zero line items; put GPS on the printed receipt; stop using the bundled SKU list

5. Connectivity & Endpoints

Mid-session endpoint failover; endpoint status + manual re-probe; make both URLs configurable without a code edit; stop logging PII

6. API Contract & Sync

toJson() silently drops confirmed/parseStatus/product-scan fields on every local persist; offline/stale indicator; search item contents

7. Product Scan Review

Stop submitting hardcoded placeholder noSo/platTruk; fix the hardcoded store-name fallback; guard submit against double-tap

8. Scan Mode UX

Persist active scan mode across restarts; confirm before deleting a pending document; show how long a document has awaited confirmation

9. Stocks Menu

Stock data layer, stock-entry screen, Stocks menu, Product Scan rewrite onto real batches — see the next slide

10. Cashier Fast Path

One-tap expiry confirm at the cashier, 'inferred' badge + end-of-day review, burst capture — see the next slide

§3.1 (persist the pending queue) and all of §8.1 are already [DONE] — the plan file records what shipped, with dates, alongside what is open. Sections 9 and 10 are full design plans, not one-line items.

The backend backlog

From `backend/plans/next-enhancements.md` — a separate, independently governed backlog. Several sections are already fully shipped.

ALREADY SHIPPED

§1 API Gateway

File-hash dedup, AbortSignal timeout

§3 Postgres

ocr_items updates wrapped in DB transactions

§7 Auth

Store accounts + profile-sourced metadata

§9 Flutter client contract

v1 surface completion — fully DONE

Two decisions were deliberately CANCELLED, not forgotten: auth on the classic routes (§1.3) and a mobile web scan page (§2.2). Don't re-open them without reading why.

STILL OPEN

§2 OCR Pipeline & Accuracy

Ongoing accuracy work; includes a human review of do-008.jpg's ground truth

§6 Product Scan ground truth

Annotation + accuracy measurement for the classifier

§8 Master Data Management

The store_master gap lives here

§10 Confirmation gate & hygiene

Document confirmation + data hygiene

§11 Single-pass classification

Collapse the classify + OCR round trips

§12 Stock Management

Schema + CRUD, Product Scan decrement hook, admin view

§13 Per-sale expiry resolution

resolveExpiryFromEvidence() matcher, wire into both scan routes, multi-frame union, on-prem dot-matrix recognizer

The two big rocks — Stocks and per-sale expiry

These are the only two items with full written design plans rather than backlog one-liners. Both are full-stack and sequenced. Read the plans before touching either.

[docs/stock-feature-plan.md](#)

Full-stack Stocks feature: new `src/db/init-stock.ts` schema, `/api/v1/stock/*` routes, a decrement hook on document confirmation, in-stock candidate filtering for Product Scan, a read-only admin view, and on the Flutter side a stock data layer, stock-entry screen, Stocks menu, and a Product Scan rewrite onto real batches. Sequenced in 7 chunks: backend → Flutter → admin.

[docs/expiry-tracking-plan.md](#)

Per-sale expiry tracking, built ON TOP of the stock plan: a batch registry plus a `resolveExpiryFromEvidence()` candidate matcher. The reframe is that you don't need to OCR the expiry perfectly — you need to pick the right batch from a small set of known candidates. Phased, with an expected accuracy argument for why this reaches ~90%+ where free-form OCR cannot.

EXPIRY — THE THREE PHASES

Phase 1

Batch registry + candidate matching wired into both scan routes

Phase 2

Multi-frame evidence union — burst capture from a mounted camera

Phase 3

On-prem dot-matrix recognizer trained on synthetic data

Hard dependency to respect: the DO editor still validates SKUs against the bundled `lib/data/master_sku.dart` map. A SKU that exists in the backend but not in that bundled copy would fail client-side and never reach stock intake. Fix backlog item 4.5 BEFORE or alongside the stock data layer (9.1).

What is explicitly NOT in scope

Just as important as the roadmap: these are decided non-goals, each with a recorded reason.

No cloud OCR, ever	Everything runs on your own GPU. This is a client requirement, not a cost optimization.
No mobile web scan page	m-scan-pfm/page.tsx and its nginx route are intentionally unbuilt. Real mobile scanning goes through the Flutter app. scan-pfm/page.tsx stays desktop-only, for testing the pipeline. (Backend plan §2.2, cancelled.)
No auth on the classic routes	/api/upload, /api/parse, /api/history and the internal web pages will not get a login flow. They are dev tooling, not product. (Backend plan §1.3, cancelled — §1.4 shipped instead.)
Not internet-facing	CORS is wide open and the LAN leg is plain http. This stack is designed for a controlled LAN or demo deployment. Exposing it publicly is a project, not a config change.
Not Play Store distributable	Release builds are debug-signed. Internal installs only until a real signing config exists.
The two governance zones stay separate	The root e/enhance + n/next workflow, its 256-line file-split rule, and its Demo/Live and Cloud/Local switches govern the FLUTTER APP ONLY. backend/ is separately governed and has its own copy. Never run one zone's workflow against the other's code.
Demo/Live and Cloud/Local do not exist yet	AGENTS.md §5 and §6 describe them, but nothing implements them. AppConfig's URL resolution is NOT a Demo/Live switch — it always talks to the live backend with real data. Also: docker-compose.demo.yml uses 'demo' for an unrelated concept (production mode). Don't conflate them.

A suggested first 30 days

You have a mixed-skill team and no single owner. This is the ordering that de-risks the handover fastest — prove you can operate it before you try to change it.

WEEK 1

Get it running

Everyone: stand the stack up from the repo root, on the real GPU host.
Log in from a real device end-to-end. Scan one DO, correct it, print the PDF.
Run flutter test + flutter analyze, and the accuracy harness. Record the number.
Outcome: nobody is blocked on the outgoing engineer for basic operation.

WEEK 2

Assign owners

Split into three lanes: Flutter client · gateway + parser · GPU pipeline + models.
Each owner reads their zone's CLAUDE.md/AGENTS.md and their backlog end to end.
Each owner ships ONE small [TODO] item in their lane and logs it in feature-list.md.
Outcome: three people have made a real change and know the review workflow.

WEEK 3

Close the risk gaps

Recommended first real work, in this order: 5.4 (stop logging tokens and PII),
2.4 (enforce the blur check), 4.5 (live SKU endpoint instead of the bundled map),
and load store_master from toko_aktif.json.
Outcome: the two security items and the two accuracy ceilings are gone.

WEEK 4

Decide the roadmap

With the codebase understood, decide as a team whether Stocks + per-sale expiry
is the next investment, and re-confirm the plans' open implementation-time
decisions before writing code.
Outcome: the team owns the roadmap instead of inheriting it.

Strongest single recommendation: before your last day, sit with the team while THEY run the full loop on their own machines. Watching them succeed is the only real proof the handover worked.

Where everything lives

Read first

[README.md](#)

architecture, accuracy, install, gotchas

[CLAUDE.md](#)

root operating rules + repo overview

[AGENTS.md](#)

the e/enhance + n/next workflow, scope rules

[SKILLS.md](#)

the review roles

[backend/CLAUDE.md](#)

MUST read before touching backend/

[backend/AGENTS.md](#)

uv conventions, backend workflow

Deep dives

[docs/workflow_detail.md](#)

the DO workflow

[docs/workflow_detail_aplikasi.md](#)

field-by-field extraction rules

[docs/regex_rules_example.md](#)

the regex rule set

[docs/simulasi_ekstraksi_lengkap.md](#)

worked extraction example

[docs/api-contract-map.md](#)

Flutter/backend contract, gaps G1–G10

[backend/docs/scan-product.md](#)

Product Scan internals + retraining

[backend/docs/vllm-service.md](#)

the vLLM service

[screenshots/v2/WORKFLOW.md](#)

screen-by-screen app walkthrough

Status and history

[plans/next-enhancements.md](#)

Flutter backlog (live)

[backend/plans/next-enhancements.md](#)

backend backlog (live)

[docs/feature-list.md](#)

what shipped, Flutter

[backend/docs/feature-list.md](#)

what shipped, backend

[docs/iteration-log.md](#)

review + audit trail, 908 lines

[backend/docs/iteration-log.md](#)

same, backend

[docs/stock-feature-plan.md](#)

planned Stocks feature

[docs/expiry-tracking-plan.md](#)

planned expiry tracking

Structural questions ("what calls X", "what breaks if Y is renamed") — an AST knowledge graph lives in graphify-out/, rebuilt by the git post-commit hook. Query it before grepping.