

Prima Fresh Mart Scanner

app-pfm-ocr-v2

An on-premise, GPU-accelerated OCR system that turns a store staff member's phone photo of a Delivery Order into structured, database-backed data. A Flutter mobile client on one end, a Dockerized Next.js / PaddleOCR / vLLM / PostgreSQL stack on the other. No cloud OCR API is involved anywhere.

Prepared by	[TO FILL IN] (outgoing engineer)
Handover date	[TO FILL IN]
Successors	[TO FILL IN]
Repository	app-pfm-ocr-v2 (git, branch main)
Companion deck	docs/handover/PFM-Scanner-Knowledge-Transfer.odp (25 slides)

READ THIS FIRST Everything in this document is drawn from what is actually committed in the repository. Nothing is invented. Where only the outgoing engineer can know something — accounts, secrets, client contacts, deployment history — the text says [TO FILL IN] rather than guessing. Fill those in before handing it over.

Contents

- 1 About this document** — How to read it, and which repo docs are authoritative
- 2 The system at a glance** — The product, the user, the one constraint
- 3 Architecture** — Components, the two nginx surfaces, the throughput ceiling
- 4 The DO scan, end to end** — The nine stages of a scan
- 5 The Flutter client** — Structure, and the five parts that are not obvious
- 6 The Next.js gateway** — The two API surfaces, and where accuracy is won
- 7 The OCR pipeline and vLLM** — The GPU services and the deskew lever
- 8 Product Scan** — DINOv2 matching, model artifacts, non-goals
- 9 The database** — Where schema really comes from, and the main tables
- 10 Accuracy** — The three layers, the numbers, and how to prove no regression
- 11 Prerequisites and first-time setup** — Host requirements and first run
- 12 Operational runbook** — Start, demo, tunnel, APK, retrain, backup
- 13 Troubleshooting** — Symptom, cause, fix
- 14 Gotchas and tribal knowledge** — What is true, consequential and non-obvious
- 15 Access and credentials handover** — The checklist only you can complete
- 16 What's next** — Both backlogs and the two big rocks
- 17 What is explicitly not in scope** — Decided non-goals
- 18 A suggested first 30 days** — How a mixed-skill team takes this over
- A Glossary** — Domain and pipeline vocabulary
- B Command reference** — Every command in one place
- C Where everything lives** — The full document index
- D The app, screen by screen** — Every screen of both flows, captured live

1. About this document

This is the long-form companion to the 25-slide handover deck. The deck is for a walkthrough session; this document is for the person who has to work on the system alone, three weeks later, at 11pm, with something broken. It covers the same ground in more depth and adds three things the deck does not have: an operational runbook (section 12), a troubleshooting guide (section 13), and a glossary (Appendix A).

How to read it

- **Sections 2–9** describe what exists: the product, the architecture, and each component in turn. Read these once, in order.
- **Section 10** is about accuracy — where the numbers come from, what the ceilings are, and how to prove you have not regressed. This is the part most often misunderstood.
- **Sections 11–13** are operational: setup, runbook, troubleshooting. Keep these to hand.
- **Sections 14–18** are the handover proper: what will bite you, what must be transferred, what is planned, what is deliberately out of scope, and a suggested first month.

The one rule

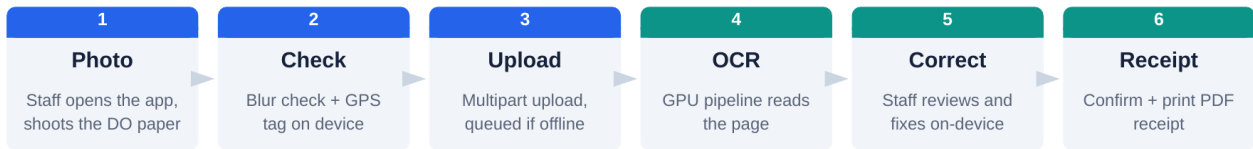
If this document and the repository ever disagree, the repository wins. It is the one that gets updated by whoever ships code. This document was written on the state of the repo at handover time and will drift.

Where the authoritative docs live

DOCUMENT	WHAT IT IS AUTHORITATIVE FOR
README.md	Architecture, accuracy figures, install, known limitations
CLAUDE.md	Repo overview and how work is done at the root (Flutter)
AGENTS.md	The e/enhance and n/next workflow, and the scope boundary
backend/CLAUDE.md	Backend history, gotchas, confidentiality — read before touching backend/
backend/AGENTS.md	uv conventions, backend workflow, backend's own debt list
docs/workflow_detail_aplikasi.md	Field-by-field extraction rules
docs/api-contract-map.md	The Flutter/backend contract and its known gaps
plans/next-enhancements.md	The live Flutter backlog
backend/plans/next-enhancements.md	The live backend backlog

2. The system at a glance

A store staff member photographs the paper Delivery Order when a delivery arrives, and the system reads it into the database so nobody has to re-type it. That is the whole product in one sentence. Everything else is in service of making that one loop reliable enough to trust.



The six steps a store staff member actually experiences.

Who the user is

The user is a store staff member — petugas toko — on duty at a Prima Fresh Mart store. It is not the delivery driver. This matters more than it sounds: the app is designed around someone who is standing at their own store, with their own login, receiving goods on the store's behalf. Each account is bound to exactly one store (`accounts.role = 'store'`, one `kode_toko` per account), so whoever is on shift there uses the same login, and the store on an upload comes from the account rather than from OCR guessing at the address text.

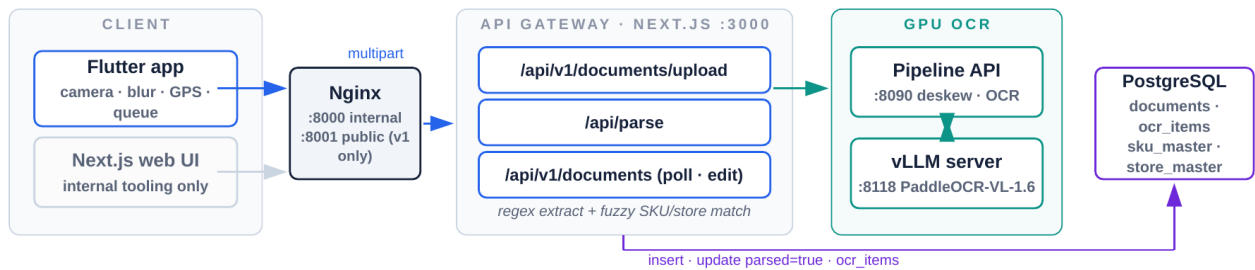
Two clients, one backend

- **Flutter mobile app (lib/)** — the real, store-facing product. This is what ships to stores.
- **Next.js web pages (backend/pfm-web-app/src/app/*.tsx)** — internal tooling only: manual ground-truth labelling, accuracy comparison, and an OCR 'arena' for comparing engines. These have no login screen, are not shipped to stores, and are not part of the product.

The non-negotiable constraint

CONSTRAINT Everything runs on your own GPU. Layout detection, text recognition and LLM structuring are all on-premise. This is a client requirement, not a cost optimisation, and it shapes every other decision in the system — including the single-GPU throughput ceiling described in section 3.

3. Architecture



Source: README.md § Architecture. Ports as declared in docker-compose.yml and backend/nginx.conf.

Component responsibilities

TECHNOLOGY	WHERE IT LIVES	ROLE
Flutter (Riverpod, Dio, Hive, go_router)	lib/	Camera capture, blur detection, offline-tolerant upload queue, correction editor, native PDF and print
Next.js App Router (TypeScript)	backend/pfm-web-app/	Auth, upload handling, pipeline orchestration, regex post-processing, fuzzy SKU and store matching, internal tooling
Nginx	backend/nginx.conf	Entry point. Two server blocks — see below
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/ and src/db/init.ts	Documents, line items, SKU / vendor / customer / store master data
DINOv2 similarity search + PaddleOCR	backend/config/classify_ocr_server.py	Product Scan: photo to SKU match by nearest-embedding lookup, plus expiry extraction
Docker Compose (multi-stage GPU Dockerfile)	docker-compose.yml (repo root)	Runs the whole backend as one stack

Nginx has two server blocks, and the difference matters

This is under-described in the deck and worth being precise about, because it is the system's actual security boundary.

PORT	WHAT IT EXPOSES
:8000	The full internal surface — the Next.js app at /, the dev-only classic routes under /api, /history, /arena, /gpu, plus /layout-parsing and /health proxied to the pipeline API and /v1 proxied to vLLM. This is the development and internal-tooling entry point.

PORT	WHAT IT EXPOSES
:8001	A hardened public endpoint. It proxies /api/v1/ to the Next.js app and returns 404 for literally everything else. This is the surface the Flutter app is meant to reach over a tunnel.

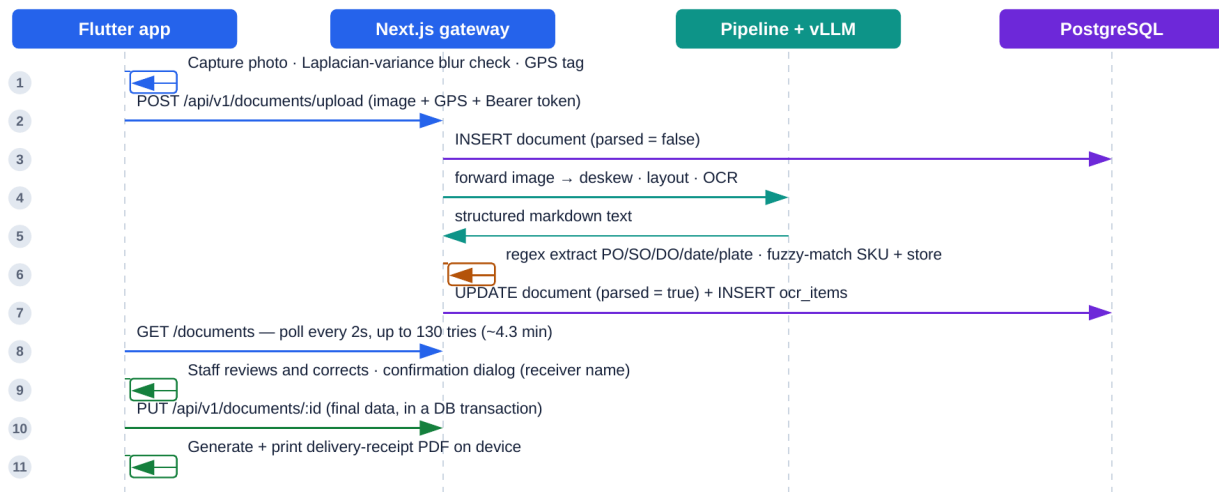
start-dev-tunnel.ps1 points ngrok at 8001, not 8000 — so the public tunnel exposes only the authenticated v1 API, with the dev tooling unreachable. If you ever change which port the tunnel targets, you change the security posture of the whole deployment. Do not do it casually.

ALSO There is also an opt-in cloudflared service in docker-compose.yml (profile: tunnel, started with docker compose --profile tunnel up). Note that it tunnels paddleocr-nginx:80 — the full :8000 surface, not the hardened :8001 one. It is a convenience for ad-hoc device testing on any network; treat anything it exposes as internal-only.

The throughput ceiling

There is one pipeline-api container and one vllm-server container, both bound to a single GPU. Concurrent uploads queue behind that GPU. This is a real, structural limit — not a tuning problem — and it is the single most important thing to load-test before any multi-store or multi-device rollout. A single-user smoke test proves nothing about it.

4. The DO scan, end to end



Source: [README.md § How It Works](#) and [lib/features/documents/pending_documents_provider.dart](#).

What happens at each stage

- 1. Capture and pre-flight.** The app takes a photo, computes a Laplacian variance blur score, and attaches a GPS position. See section 5 for the important caveat about the blur check.
- 2. Upload.** A multipart POST to /api/v1/documents/upload with the image, the GPS coordinates, and an Authorization: Bearer token. The upload is deduped server-side by file hash, which is what makes a resumed or retried upload safe.
- 3. Insert.** The gateway writes a document row with parsed = false immediately, so the work is durable before any GPU time is spent.
- 4. OCR.** The image goes to the pipeline API: deskew and unwarp, layout segmentation, then text detection and recognition, with vLLM reassembling fragments into coherent structured text.
- 5. Post-process.** The gateway runs regex extraction for the PO/SO/DO numbers, date and truck plate, then corrects the result against master data — fuzzy SKU matching, unit standardisation, fuzzy store resolution. This is where most of the accuracy comes from; see section 10.
- 6. Persist.** The document is updated to parsed = true and the line items are written to ocr_items.
- 7. Poll.** The app polls GET /documents every 2 seconds, up to 130 attempts — about 4.3 minutes — before surfacing a timeout.
- 8. Review and confirm.** The staff member corrects any wrong fields on-device, then confirms through a dialog that captures the receiver's name and a consent checkbox.
- 9. Save and print.** PUT /api/v1/documents/:id writes the final data inside a database transaction, and the app generates and prints the delivery-receipt PDF locally.

Every one of these stages has a screenshot in Appendix D, captured on a real device against the live backend.

DESIGN NOTE The system is poll-based, not push-based. There are no websockets and no push notifications. Any change to how long OCR takes directly consumes the app's 130-attempt budget, so a pipeline slowdown shows up to the user as 'Gagal mengekstrak data (Timeout)'.

5. The Flutter client

lib/ is the root Dart package — the Flutter app is not in a subfolder. One folder per feature, with shared infrastructure in core/ and models in models/. Tests live flat in test/, not mirrored to the lib/ tree.

Structure

PATH	CONTENTS
lib/config/	app_config.dart — theme and dynamic API base URL resolution
lib/core/	Dio client, Hive storage, geolocation, go_router setup
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 the caveat below

State is Riverpod, one provider group per feature (authProvider, pendingDocumentsProvider, scanModeProvider). Routing is go_router. HTTP is Dio. On-device persistence is Hive.

Screenshots of every screen described below are in Appendix D.

The five parts that are not obvious

Each of these has either caused a field problem already or is recorded as a known gap in the backlog. Read them before changing anything in the app.

5.1 The API base URL is resolved once, at startup

AppConfig.initializeApiBaseUrl() picks the backend at launch: it tries a fixed, reserved ngrok domain first and falls back to a hardcoded LAN IP. It runs exactly once and there is no mid-session failover — if the endpoint it chose dies, every subsequent call fails until the app is restarted. When a device cannot log in and the backend is confirmed healthy, this is the first thing to check. Backlog items 5.1 through 5.3 cover failover, status visibility, and making both URLs configurable without a code edit.

5.2 The blur check is currently cosmetic

ImagePreviewScreen computes the Laplacian variance and displays a 'Foto Terdeteksi Blur! Diharuskan untuk mengambil ulang gambar' badge — but the Unggah Dokumen button is gated only on !_analyzing, not on !_blurResult!.isBlur. A photo flagged as blurry can still be uploaded as-is.

Backlog item 2.4. This is a small change with a direct effect on accuracy, which is why it is recommended as early work in section 18.

5.3 The pending queue is persisted, and resumes correctly

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 not yet terminal. A persisted 'uploading' item re-runs the whole upload — safe because the endpoint dedupes by file hash — while a persisted 'processing' item resumes polling rather than re-uploading. Storage failures are caught and swallowed everywhere, so a Hive error degrades to the old in-memory behaviour instead of crashing the queue.

5.4 Polling swallows every error, which hides real causes

The per-iteration catch in `_pollUntilParsed()` silently absorbs all exceptions, including plain connectivity errors, and simply retries. If all 130 retries are consumed that way, the user is told 'Gagal mengekstrak data (Timeout)' — so a dead Wi-Fi connection is reported as an OCR problem. This is a genuine diagnosis trap in the field. Backlog item 3.5.

5.5 SKU validation is offline and can drift

The DO editor validates item SKUs against `lib/data/master_sku.dart` — a Map compiled into the app — rather than the live `/api/v1/master/skus` endpoint that the Product Scan flow already uses. Any SKU added or changed centrally will not validate correctly until the app is rebuilt and redeployed to every device, and the bundled copy can silently drift from the real master data. Backlog item 4.5. It is also a hard blocker for the planned Stocks feature — see section 16.

6. The Next.js gateway

backend/pfm-web-app/ is an App Router TypeScript project that plays three roles at once: the authenticated API for the mobile app, the orchestrator that drives the OCR pipeline, and the host for the internal web tooling. Understanding which of those three a given route belongs to is the key to working in it safely.

Two API surfaces with different security postures

The production surface — /api/v1/*

- **Real JWT auth.** /api/v1/auth/login checks a bcrypt-hashed password against the accounts table and signs a token. bcryptjs is used rather than native bcrypt because the pfm-web-app Docker stage is node:20-slim with no build toolchain for native addons.
- **Enforced, not decorative.** /api/v1/documents/* — list, upload, and 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 an Authorization: Bearer header.

The dev-only surface — the classic routes

/api/upload, /api/parse, /api/history, /api/scan-pfm and the root, scan-pfm and manual-label web pages have no login flow and never will. Appendix D.4 shows what they look like. They are internal tooling — labelling, accuracy comparison, engine comparison — and are not part of the store-facing product. Do not build product features on them. This was an explicit decision (backend plan task 1.3, cancelled; 1.4 shipped instead), not an oversight.

SECURITY Every API route sets Access-Control-Allow-Origin: *. Combined with the :8000 server block exposing the unauthenticated classic routes, that means :8000 must never be exposed to the open internet. The hardened :8001 block exists precisely so that a tunnel can be pointed somewhere safe. Exposing this stack publicly is a project, not a config change.

Post-processing

src/utils/parser.ts holds the regex extraction and sanitisation rules — parseDOMetadata and sanitizeParsedMetadata. This is the highest-leverage and highest-risk file in the backend: it is where accuracy is won, and the only file with a dedicated regression harness pointed at it. Never change it without running that harness (section 10.3).

Two other routes are worth knowing about: api/vllm-proxy/[[...path]]/route.ts is a passthrough proxy to the vLLM server that logs every call for observability, and is not part of the OCR path itself; docker-compose.override.yml exposes the database on 5432 and the pipeline API on 8090 directly to the host for local development.

7. The OCR pipeline and vLLM

Two GPU services, both started by the same Compose stack, both talking to each other over the internal Docker network.

SERVICE	WHAT IT DOES
pipeline-api (:8090)	PaddleOCR v6 with PP-DocLayoutV3. Auto-deskew and unwarp, layout segmentation, text detection and recognition. Exposes /layout-parsing and /health.
vllm-server (:8118)	vLLM serving PaddleOCR-VL-1.6-0.9B. Reassembles recognised text fragments into coherent structured text. Exposes the OpenAI-style /v1 surface.
classify_ocr_server (:8120)	The Product Scan classifier and OCR service — see section 8. Started alongside the pipeline API.

Deskew is the quiet accuracy lever

The pipeline estimates page tilt from the average angle of the detected text blocks. That estimate is only as good as the photo: when the DO paper is lying on top of other papers, the edges and text of the sheet underneath corrupt the angle estimate, which is the failure mode behind the unwarp-retry logic. This is not a code problem and cannot be fixed in the parser — it is a photo-capture SOP problem, and it is quantified in section 10.2.

Running the Python services outside Docker

The services are uv-managed and can be run bare-metal on Linux for development. Note that this does not work on a Windows host: the paddlepaddle-gpu wheels are Linux-only, which is why the retraining procedure in section 12.6 goes through a container even on a Windows machine.

```
./scripts/install.sh # bootstrap .venv for the vLLM server
./scripts/install-pipeline.sh # bootstrap .venv-api
./scripts/serve.sh # vLLM genai server on :8118
./scripts/serve-pipeline.sh # pipeline API :8090 + classify_ocr_server :8120
```

8. Product Scan (secondary feature)

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



How the classification works

Stage 1 is a DINOv2 nearest-neighbour lookup. The query image is embedded (resized to 224², ImageNet normalisation) and dot-producted against precomputed L2-normalised 384-dimension embeddings of every reference photo; per-class score is the maximum similarity of any reference photo in that class. A fine-tuned yolo26n-cls checkpoint is the fallback when the index or model is unavailable or throws.

TRAP The DINOv2 'confidences' in `all_probabilities` are cosine similarities, not probabilities. They do not sum to 1 and are typically all high (0.4–0.9). Do not threshold them as if they were softmax outputs. The YOLO fallback's probabilities are real softmax values — the two are not comparable.

Stage 2 is PaddleOCR over the same image, extracting the product name, the SKU and the expiry date from the recognised text lines.

Model artifacts

FILE (UNDER PFM-WEB-APP/PUBLIC/PRODUK-PFM/)	WHAT IT IS
<code>foto-kemasan-v2/</code>	Reference photo dataset — 81 classes, 2,493 photos as of 2026-07-14 (target ~230 SKU)
<code>models/dinov2_index.pkl</code>	DINOv2 embeddings and metadata; rebuild after adding photos
<code>models/produk-pfm-classifier-26n-100e-2026-07-14.pt</code>	Fine-tuned YOLO classifier — 85.8% top-1, 94.4% top-5 validation across all 81 classes; retrained 2026-07-14 in 54m21s on an RTX 2060
<code>index_dinov2.py</code>	Rebuilds the pickle index from <code>foto-kemasan-v2/</code>
<code>train_classifier.py</code>	Splits 80/20 into <code>yolo_dataset/</code> , fine-tunes <code>yolo26n-cls.pt</code> , writes a dated checkpoint

Classifier weights are auto-discovered: `CLASSIFIER_MODEL_PATH` wins if set, otherwise the newest `produk-pfm-classifier-26n-*e-*.pt` in `models/` by date-in-filename then mtime. Retraining therefore just means dropping in a new dated file — no config change. The retraining procedure itself is in section 12.6.

Deliberate non-goals

No mobile web page is planned. `m-scan-pfm/page.tsx` and its `nginx.conf` route are intentionally left unbuilt and dead — real mobile product scanning goes through the Flutter app, and `scan-pfm/page.tsx` stays desktop-only as a way to test the pipeline. Still outstanding: an actual browser pass uploading a photo through `/scan-pfm` end to end. Verification so far has been via container logs and model loading, not a UI test.

9. The database

PostgreSQL 15, one container, one named volume (paddleocr_pgdata). There is no manual migration step, and that surprises people — so be precise about where schema actually comes from.

src/db/init.ts is the source of truth

It runs idempotent CREATE TABLE IF NOT EXISTS and seed statements on every application start. Schema and reference data — vendor, customer, a starter SKU catalogue — are created automatically on first request. It also hashes the seed password and idempotently migrates any pre-existing plaintext account rows on each startup.

db/migrations/*.sql is first-boot only

Those files run once, through Postgres's own docker-entrypoint-initdb.d mechanism, and only on a brand-new volume. They are useful as a record of intent but must not be treated as the live schema. If the two ever disagree, init.ts is what the running system actually has.

Main tables

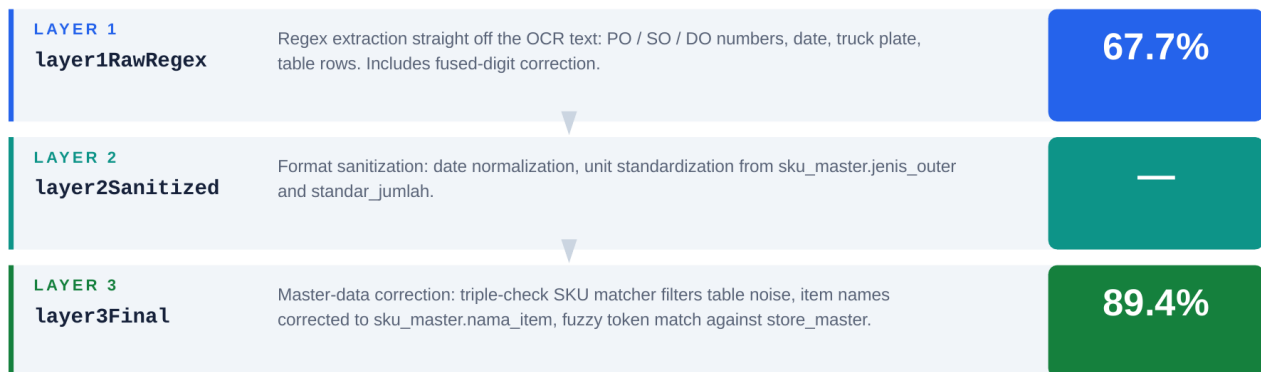
TABLE	CONTENTS
documents	One row per scanned DO or product scan
ocr_items	Line items; rewritten inside a transaction on PUT
sku_master	Product catalogue plus packaging columns — standar_jumlah, berat_kemasan, isi_outer_kg, isi_outer_pac, jenis_outer
store_master	Stores. Ships EMPTY — see section 14
accounts	Login, bcrypt password hash, bound to one kode_toko
vendor / customer	Master identities, seeded from init.ts

CONFIDENTIALITY backend/sources/ and backend/uploads/ contain the client's real SKU, store, vendor and customer master data, real scanned DO photos, and hand-labelled ground truth — committed directly to source. Read backend/CLAUDE.md § Confidentiality before exporting, logging or sharing anything from those paths, and do not export the accounts table.

10. Accuracy

This is the section most worth reading carefully. Almost everyone's first assumption about this system — that accuracy is an OCR-quality problem — is wrong.

10.1 Accuracy is a three-layer story



Raw regex extraction straight off the OCR text gets 67.7%. The final figure is 89.4%. The entire gain comes from a correction pass against master data: fuzzy SKU and store matching, unit standardisation, and format sanitisation. The stage names are real — trace layer1RawRegex, layer2Sanitized and layer3Final into backend/pfm-web-app/src/utils/parser.ts to see where each is produced.

The practical consequence: when accuracy drops, the master data is at least as likely to be the cause as the OCR. Check whether sku_master and store_master are populated before you go looking at the pipeline.

10.2 Where the numbers stand

Measured against 37 hand-labelled real DO photos in backend/sources/test-images with ground truth in manual_labels.json. Overall 89.4% exact-field-match, up from an 82.2% baseline when tracking began, against a 95% target.

FIELD	RAW REGEX	FINAL	WHAT CLOSES THE GAP
Customer (Kepada Yth)	100%	100%	—
Kode Barang (SKU)	98.4%	98.4%	Already reliable at the OCR layer
Item count	13.5%	97.3%	Table-noise rows filtered by the triple-check
Nama Barang	0%	94.4%	Corrected to sku_master.nama_item on SKU match
Banyak (qty)	77.6%	95.2%	Unit standardised from jenis_outter
Jumlah (total)	76.8%	94.4%	Unit standardised from standar_jumlah
No. PO	91.9%	91.9%	Fused-digit correction at the regex layer

FIELD	RAW REGEX	FINAL	WHAT CLOSES THE GAP
No. DO	91.9%	91.9%	—
Tanggal	89.2%	89.2%	—
No. SO	86.5%	86.5%	—
Store	0%	64.9%	Fuzzy token match against store_master
Plat Truk	62.2%	62.2%	Mostly genuine OCR misses on the truck line
Alamat	0%	37.8%	Canonicalised against customers where phrasing matches

The three gaps, and which are actually fixable

- **Alamat, 37.8% — capped.** The ground-truth labels themselves use two different phrasings for the same physical address across photo batches. Closing this needs the ground truth unified, not more parsing logic. Do not spend engineering time on the parser here.
- **Plat, 62.2% — mostly not a bug.** These are largely genuine OCR misses: the truck line is often faint or absent in the photo.
- **Store, 64.9% — bounded by data.** It can only improve as far as store_master is populated, and store_master currently ships empty. This is a data-loading task, not a parsing task.

The lever nobody expects: photo capture SOP

A meaningful share of the current test set was photographed with the DO paper placed on top of other papers rather than a plain, flat surface, which breaks the deskew angle estimate as described in section 7. Tightening the field SOP — DO paper alone, flat contrasting surface, reasonably lit, squared to the camera — should raise these numbers with no code changes at all. Today's figures are a floor, not a ceiling.

10.3 How to prove you have not regressed

There is a real regression harness, and it is the closest thing this project has to a safety net. Run it after any change to parser.ts.

```
cd backend
node pfm-web-app/scripts/accuracy-check.mts # cached OCR, fast
node pfm-web-app/scripts/accuracy-check.mts --refresh-ocr # real pipeline run
node pfm-web-app/scripts/accuracy-check.mts --detail <filename> # one image, all stages
```

It hits the live /api/parse endpoint for every image in backend/sources/test-images/, diffs against the hand-labelled ground truth 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. Commit that file with your change; it is the project's accuracy history.

The Product Scan flow has its own equivalent, backend/scripts/accuracy-check-scan.mts, which grades against a frozen validation snapshot in sources/product-test-images-fixed/ (79 images, built by

scripts/freeze-validation-set.mjs) so a rerun always grades the same set. It also reports a training-set score over the gallery photos — that number measures memorisation, not generalisation, so do not quote it as accuracy.

10.4 The rest of the test tooling

WHAT	HOW
Parser regex and sanitisation rules	<code>npx tsx backend/pfm-web-app/src/utils/parser.test.ts</code> — no Docker needed
Batch OCR regression	<code>backend/pfm-web-app/run_batch_test.js</code>
Flutter widget and unit tests	<code>flutter test</code> — covers blur detection, camera navigation, editor validation, geotagging, the pending queue
Static analysis	<code>flutter analyze lib</code>
Human-readable comparison reports	<code>compare_accuracy.py</code> , <code>compare_sources_accuracy.py</code> , <code>generate_excel.py</code> — build <code>sources/comparison_report.xlsx</code> and <code>comparison_side_by_side.html</code>

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

11. Prerequisites and first-time setup

Host requirements

COMPONENT	REQUIREMENT
Backend host	NVIDIA GPU, CUDA 12.6+ driver, roughly 8GB+ VRAM. Tested on native Linux and on Windows with Docker Desktop and WSL2 GPU passthrough.
Docker	Docker Engine or Desktop with the NVIDIA Container Toolkit. docker info should list nvidia under Runtimes.
Disk	60GB+ free. The pipeline-api and vllm-server images are around 30GB each once built, plus model weight caches.
Flutter	SDK >=3.2.0 <4.0.0, with Android Studio or Xcode for device tooling.
Node.js	v20+, optional — only for running the parser tests or accuracy tooling outside Docker.

Backend, first run

```
cp backend/.env.example backend/.env
# edit backend/.env:
# APP_PORT=8000 host port for the :8000 nginx block
# CUDA_VISIBLE_DEVICES=1 GPU index to allocate
# JWT_SECRET=change-me MUST be changed – see below

docker compose up --build # from the REPO ROOT, not backend/
```

CHECK THIS backend/.env.example ships JWT_SECRET=change-me. If that value ever reached a real deployment, every token the system has ever signed is forgeable by anyone who has read the repo. Verify what the running deployment actually uses as part of the credentials handover (section 15).

The first build pulls and builds roughly 60GB of GPU images and downloads model weights. Budget real time for it; subsequent starts are fast. Database schema and reference data are created automatically on the first request — there is no migration command to run.

Verifying the stack is up

```
curl http://localhost:8000/health # pipeline API health
curl http://localhost:8000/v1/models # vLLM model list
curl http://localhost:8001/api/v1/health # hardened public surface
curl -X POST http://localhost:8000/api/v1/auth/login \
-H 'Content-Type: application/json' \
-d '{"username":"admin","password":"password"}'
```

Flutter, first run

```
flutter pub get
./start-dev-tunnel.ps1 # detects LAN IP, patches app_config.dart, starts ngrok
flutter run
flutter build apk --release # -> build/app/outputs/flutter-apk/app-release.apk
```

12. Operational runbook

Step-by-step procedures for the things that actually get done. Every command assumes you are at the repository root unless stated otherwise.

12.1 Start the stack for development

```
docker compose up --build
```

Runs the gateway via `npm run dev` with the source bind-mounted, so `pfm-web-app` hot-reloads. This is what you want while iterating on `parser.ts`. It is a single dev-server process with a known throughput ceiling.

12.2 Start the stack for a demo or field test

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

MANDATORY This is a POLICY in `CLAUDE.md`, not a suggestion. The dev server will bottleneck the moment two devices upload at once. Always use this override for client demonstrations and field testing, and always pass `--build` — this mode does not hot-reload, so without a rebuild you are demoing stale code.

12.3 Stop, restart, and inspect

```
docker compose ps # what is running
docker compose logs -f pfm-web-app # gateway logs
docker compose logs -f pipeline-api # OCR pipeline logs
docker compose logs paddleocr-tunnel # cloudflared URL, if that profile is up
docker compose restart pipeline-api # reload models without a full rebuild
docker compose down # stop everything (volumes survive)
docker compose down -v # ALSO deletes the database volume
```

DESTRUCTIVE `docker compose down -v` destroys `paddleocr_pgdata` — every scanned document, every line item, and the accounts table. It is not a reset button. Take a dump first (section 12.7).

12.4 Refresh mobile connectivity

Two independent paths reach the backend from a phone, and both are synced by hand.

1. Run `./start-dev-tunnel.ps1` on the host. It detects the current LAN IP, rewrites `_lanBaseUrl` in `lib/config/app_config.dart`, launches ngrok against the fixed reserved domain pointed at port 8001, and then polls both the LAN and tunnel `/api/v1/health` endpoints until they answer or 60 seconds

pass.

2. If the script reports that the health check failed, the backend is not up yet — start it first and re-run.
3. Rebuild and redeploy the app if `_lanBaseUrl` changed, since it is compiled into the binary.

Re-run this any time the host machine's IP changes: a new network, a DHCP renewal, a reboot. A stale IP here is the most common cause of 'the app cannot log in' once the backend is confirmed healthy.

12.5 Ship a new APK to stores

```
flutter analyze lib # must be clean
flutter test # must be green
flutter build apk --release
```

The artifact is `build/app/outputs/flutter-apk/app-release.apk`. Note that release builds are currently signed with the debug key — fine for internal installs, and a blocker for Play Store distribution until a real signing config exists. How the APK physically reaches store devices today: [TO FILL IN].

12.6 Retrain the Product Scan classifier

Required whenever reference photos are added to `foto-kemasan-v2/`. The procedure goes through a container even on a Windows host, because `paddlepaddle-gpu` wheels are Linux-only.

1. Add the new photos to `pfm-web-app/public/produk-pfm/foto-kemasan-v2/`, one folder per SKU or class.
2. `docker compose build pipeline-api` — from the repository root.
3. Run a one-off `docker run --gpus all` from that image with `models/` mounted `WRITABLE`. The live service mounts it read-only, which is why a one-off container is needed.
4. Inside it, run `index_dinov2.py` to rebuild the embedding index, then `train_classifier.py train --imgsz 224` to fine-tune and write a new dated checkpoint.
5. `docker compose restart pipeline-api`.
6. Verify in docker logs: 'DINOv2 index loaded with N reference images' and 'Using classifier weights: <new dated file>'.

WINDOWS From Git Bash on Windows, prefix commands with `MSYS_NO_PATHCONV=1` or the `/app/...` arguments get mangled into Windows paths. A full worked example is in `backend/plans/next-enhancements.md` task 2.1.

12.7 Back up and restore the database

No automated backup exists in the repo. [TO FILL IN] — confirm whether one runs outside it. Manual dump and restore:

```
docker compose exec db pg_dump -U <user> <dbname> > backup-$(date +%F).sql  
cat backup.sql | docker compose exec -T db psql -U <user> <dbname>
```

12.8 Load store master data

store_master ships empty, which caps store-matching accuracy at 64.9%. backend/sources/toko_aktif.json appears to be the right source but is not wired into any automatic import — writing that importer is an open task (backend backlog §8).

13. Troubleshooting

Symptoms are listed roughly in the order you are likely to meet them. Every cause here is a documented behaviour of this system, not speculation.

SYMPTOM	MOST LIKELY CAUSE	WHAT TO DO
A device cannot log in, but the backend is healthy	Stale LAN IP in <code>app_config.dart</code> , or ngrok is simply not running — the reserved domain is fixed but the process is not auto-started	Re-run <code>./start-dev-tunnel.ps1</code> and rebuild the app if <code>_lanBaseUrl</code> changed (section 12.4)
Container name conflicts on docker compose up	Compose was run from inside <code>backend/</code> , whose <code>duplicate</code> file declares the project name <code>ai-ocr-pfm-2026</code>	<code>docker compose</code> down from wherever it was started, then always run from the repository root
'Gagal mengekstrak data (Timeout)' in the app	Could be a genuine OCR timeout OR a plain connectivity failure — the poll loop swallows both and reports the same message	Check the gateway and pipeline logs for whether the document was ever received. Backlog item 3.5 exists to stop conflating these
Uploads are slow or queue up during a demo	The dev server is running, or several devices are queued behind the single GPU	Use the demo override (section 12.2). If it persists, it is the structural single-GPU ceiling
OCR returns nothing useful for a document	Deskew failed — often the DO photographed on top of other papers	Re-shoot on a plain flat surface before touching any code (section 10.2)
Store field is never resolved	<code>store_master</code> is empty — it ships that way	Load real store data (section 12.8)
Item names or units look wrong after a parse	SKU did not match <code>sku_master</code> , so the layer-3 correction never applied	Verify <code>sku_master</code> is populated and the SKU was recognised; run <code>accuracy-check.mts --detail</code> on the image
docker info does not list nvidia under Runtimes	NVIDIA Container Toolkit missing or GPU not passed through to WSL2	Fix the host GPU setup — nothing in this repo can work around it
Build fails or stalls on first run	Roughly 60GB of images and model weights are being fetched	Confirm free disk, then wait. This is expected on a first build
Product Scan returns the wrong SKU with high confidence	DINOv2 scores are cosine similarities, typically all high — high confidence is not evidence of a good match	Compare relative ranking, not the absolute number. Consider adding reference photos and retraining (section 12.6)
Classifier does not pick up a newly trained model	Weights are auto-discovered by date-in-filename then mtime, and <code>models/</code> is mounted read-only in the live service	Confirm the new file is named <code>produk-pfm-classifier-26n-*e-*.pt</code> , then restart <code>pipeline-api</code> and check the log line
The confirm button in the DO review form does nothing	A required field is empty. <code>Form.validate()</code> returns false and <code>onPressed</code> simply returns — with no toast and no scroll-to-error	Check every required field. This is a known limitation recorded in <code>README.md</code>

SYMPTOM	MOST LIKELY CAUSE	WHAT TO DO
A schema change does not appear	You edited db/migrations/*.sql, which only runs on a brand-new volume	Make the change in src/db/init.ts — that is the live schema

14. Gotchas and tribal knowledge

These are the things that are true, consequential, and not discoverable by reading the code in the obvious order.

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 over from when the backend was its own repository. Compose names containers and volumes after whichever project file you invoke, so running it from inside `backend/` produces container-name conflicts with anything already started from the root. This is not hypothetical — it happened during this project's own testing. Always run from the repository root.

store_master ships empty

The schema is created automatically but no store data is seeded, so `resolveStoreFromText` resolves nothing until real data is loaded. This is the direct cause of the 64.9% store accuracy figure.

The Android release build is debug-signed

`android/app/build.gradle.kts` still carries a '// TODO: Add your own signing config' and signs release builds with the debug key.

Request and 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 through `adb logcat` on any store device. Backlog item 5.4, and the single item most worth fixing first.

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 protection. Backlog item 5.6.

nginx.conf contains dead routes

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

The two governance zones are deliberately separate

The repository root and `backend/` each have their own `CLAUDE.md`, `AGENTS.md`, `SKILLS.md`,

feature list, iteration log and backlog. The root e/enhance and n/next workflow, its 256-line file-split rule, and its Demo/Live and Cloud/Local switches govern the Flutter app only. Never merge the two backlogs, and defer to backend/CLAUDE.md for anything inside backend/.

'Demo' means two unrelated things

docker-compose.demo.yml is a PRODUCTION-mode override (npm start instead of npm run dev). AGENTS.md §5's Demo/Live switch is a mock-data mode that does not exist yet. They are unrelated. Do not conflate them.

15. Access and credentials handover

The repository cannot tell you any of this. Fill in every line before the last day, and — more importantly — verify the successor can actually use each one while you are still reachable.

Infrastructure

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

Secrets

- ☐ backend/.env — the real JWT_SECRET, CUDA_VISIBLE_DEVICES, APP_PORT
- ☐ PostgreSQL credentials and where the data volume lives
- ☐ Backup location for the database, or explicit confirmation that none exists

Connectivity

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

Mobile distribution

- ☐ Android signing key, or confirmation that none exists yet
- ☐ How the APK reaches store devices today
- ☐ The list of devices and stores currently running it

Accounts and people

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

RECOMMENDATION Do not hand this 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.

Deployment state — only you know this

Nothing in the repository records which stores are live, on what hardware, since when, or with what agreed support expectations. Write it here before you go:

[TO FILL IN]

16. What's next

Both backlogs are live, written down, and maintained in the repository. What follows is a map, not a replacement — read the plan files themselves before acting.

16.1 The Flutter backlog

plans/next-enhancements.md, ten sections. A representative open item from each:

SECTION	REPRESENTATIVE OPEN WORK
1. Auth & Splash	Structured 401 handling instead of string-matching; a global 401/403 interceptor; the splash screen's 3s of fixed delay runs before the auth call rather than alongside it
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 an 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 and manual re-probe; configurable URLs; stop logging PII
6. API Contract & Sync	toJson() silently drops confirmed, parseStatus and product-scan fields on every local persist; offline/stale indicator; search item contents
7. Product Scan Review	Stop submitting hardcoded placeholder noSo and platTruk; fix the hardcoded store-name fallback; guard submit against double-tap
8. Scan Mode UX	Persist the 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 16.3
10. Cashier Fast Path	One-tap expiry confirm at the cashier, inferred badge and end-of-day review, burst capture — see 16.3

16.2 The backend backlog

backend/plans/next-enhancements.md — separately governed. Already shipped: §1 API gateway (file-hash dedup, AbortSignal timeout), §3 Postgres (ocr_items updates wrapped in transactions), §7 auth (store accounts and profile-sourced metadata), and §9 Flutter client contract, which is fully done. Still open: §2 OCR pipeline and accuracy, §6 Product Scan ground truth, §8 master data management (the store_master gap), §10 confirmation gate and data hygiene, §11 single-pass classification, §12 stock management, and §13 per-sale expiry resolution.

Two decisions were deliberately cancelled, not forgotten: auth on the classic routes (§1.3) and a mobile web scan page (§2.2). Do not re-open either without reading why they were closed.

16.3 The two big rocks

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

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

A full-stack Stocks feature: a 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 seven chunks, backend then Flutter then admin. The plan also records open implementation-time decisions that were flagged during planning and not re-confirmed — read those before starting.

[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 at its heart is that you do not need to OCR the expiry date perfectly — you need to pick the right batch from a small set of known candidates, which is a far easier problem and is why the plan argues it can reach roughly 90%+ where free-form OCR cannot.



SEQUENCING 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 validation and never reach stock intake. Fix backlog item 4.5 before or alongside the stock data layer (9.1).

17. What is explicitly not in scope

Just as important as the roadmap. Each of these is a decided non-goal with a recorded reason — re-opening one is a decision, not a bug fix.

- **No cloud OCR, ever.** Everything runs on your own GPU. A client requirement, not a cost optimisation.
- **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. 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. Backend plan §1.3, cancelled; §1.4 shipped instead.
- **Not internet-facing as a whole.** The `:8001` block is the only surface designed for tunnel exposure. `:8000` is internal, CORS is wide open, and the LAN leg is plain http.
- **Not Play Store distributable.** Release builds are debug-signed. Internal installs only, until a real signing config exists.
- **The two governance zones stay separate.** 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; nothing implements them. AppConfig's URL resolution is not a Demo/Live switch — it always talks to the live backend with real data.

18. A suggested first 30 days

A mixed-skill team with no single owner de-risks a handover fastest by proving it can operate the system before trying to change it.

Week 1 — get it running

- Everyone stands the stack up from the repository 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 and parser, GPU pipeline and models.
- Each owner reads their zone's CLAUDE.md and 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 highest-risk gaps

- Backlog 5.4 — stop logging tokens and PII in release builds.
- Backlog 2.4 — enforce the blur check instead of only displaying it.
- Backlog 4.5 — validate SKUs against the live endpoint instead of the bundled map.
- Load store_master from toko_aktif.json.
- **Outcome:** two security holes closed and two accuracy ceilings lifted.

Week 4 — take ownership of the roadmap

- Decide as a team whether Stocks and per-sale expiry is the next investment.
- Re-confirm the plans' open implementation-time decisions before writing code.
- **Outcome:** the team owns the roadmap instead of inheriting it.

THE SINGLE BEST THING YOU CAN DO 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.

Appendix A — Glossary

Domain and pipeline vocabulary a newcomer hits immediately.

TERM	MEANING
DO	Delivery Order — the paper document that arrives with a delivery. The thing this whole system reads.
PO	Purchase Order number, extracted from the DO header.
SO	Sales Order number, extracted from the DO header.
SKU	Stock Keeping Unit — the product code. Called Kode Barang on the DO.
kode_toko	Store code. Each account is bound to exactly one.
petugas toko	Store staff on duty — the actual user of the app. Not the driver.
Kepada Yth	'To' / addressee field on the DO — the customer.
Nama Barang	Product name line on the DO.
Banyak	Quantity on a DO line.
Jumlah	Total on a DO line.
Plat Truk	Truck licence plate, extracted from the DO.
Outer	Packaging unit. jenis_outer, isi_outer_kg and isi_outer_pac in sku_master drive unit standardisation.
standar_jumlah	The standard quantity unit for a SKU, used to standardise the Jumlah field.
Deskew	Correcting page rotation before OCR, estimated from the average angle of detected text blocks.
Unwarp	Correcting page curvature/perspective distortion before OCR.
Layout parsing	Segmenting a page into regions (tables, headers, text blocks) before recognition.
layer1RawRegex	Stage 1 — regex extraction straight off the OCR text (67.7%).
layer2Sanitized	Stage 2 — format sanitisation and unit standardisation.
layer3Final	Stage 3 — master-data correction and the triple-check SKU matcher (89.4%).
Triple-check	The SKU matcher that filters table-noise rows by cross-checking SKU and unit against master data.
Ground truth	The hand-labelled correct answers in sources/manual_labels.json, used to score accuracy.
DINOv2	The embedding model behind Product Scan's nearest-neighbour SKU match.
vLLM	The inference server hosting PaddleOCR-VL-1.6-0.9B.
Arena	An internal web page for comparing OCR engines. Dev tooling only.
e / enhance, n / next	The repository's own planning workflow triggers, defined in AGENTS.md.

Appendix B — Command reference

Flutter (repository root)

```
flutter pub get install dependencies
flutter run run on a connected device
flutter test all tests
flutter test test/blur_detector_test.dart a single test file
flutter analyze lib static analysis
flutter build apk --release release APK
./start-dev-tunnel.ps1 sync LAN IP + start ngrok
```

Backend stack (repository root)

```
docker compose up --build development
docker compose -f docker-compose.yml \
-f docker-compose.demo.yml up -d --build demo / field test
docker compose --profile tunnel up add cloudflared
docker compose logs -f <service> follow logs
docker compose restart pipeline-api reload models
docker compose down stop (volumes kept)
```

Web app (backend/pfm-web-app)

```
npm run dev next dev -H 0.0.0.0 (binds all interfaces for LAN testing)
npm run build
npm run start
npm run lint
```

Accuracy and tests (backend)

```
node pfm-web-app/scripts/accuracy-check.mts DO flow, cached OCR
node pfm-web-app/scripts/accuracy-check.mts --refresh-ocr force a pipeline run
node pfm-web-app/scripts/accuracy-check.mts --detail <f> one image, all stages
node scripts/accuracy-check-scan.mts Product Scan flow
npx tsx pfm-web-app/src/utils/parser.test.ts parser unit tests
```

Python services (uv-managed, Linux only)

```
./scripts/install.sh bootstrap .venv (vLLM server)
./scripts/install-pipeline.sh bootstrap .venv-api
./scripts/serve.sh vLLM genai server :8118
./scripts/serve-pipeline.sh pipeline :8090 + classify_ocr_server :8120
```

Appendix C — Where everything lives

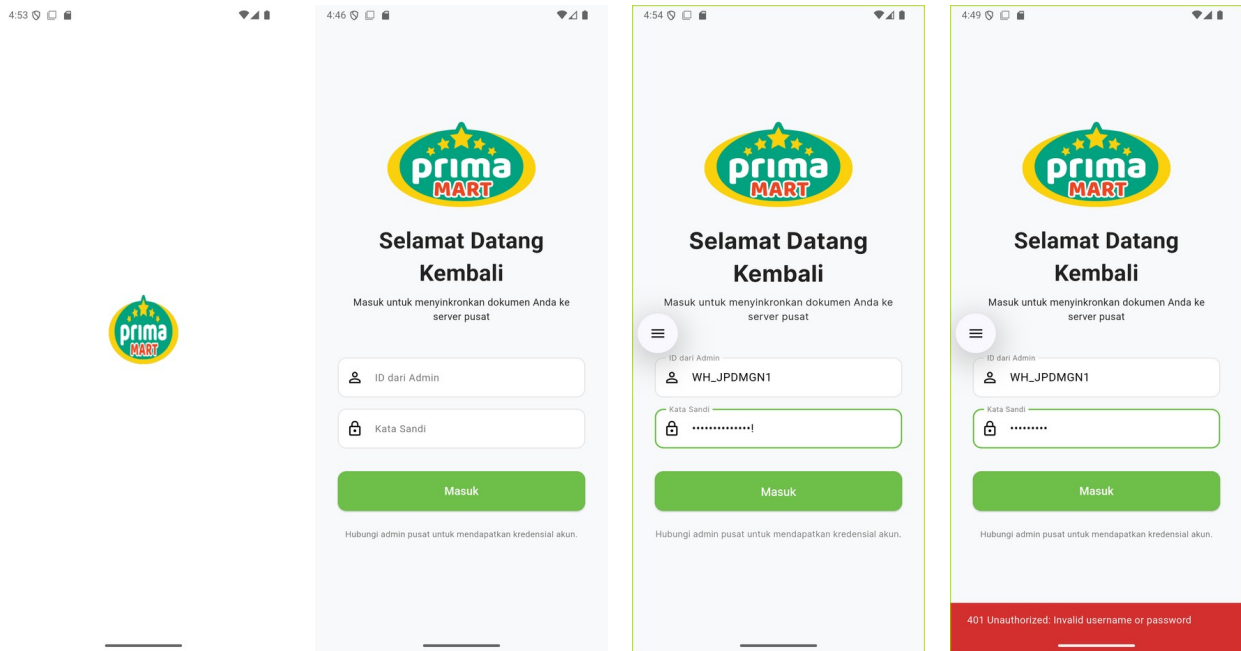
PATH	WHAT IT IS
README.md	Architecture, accuracy, install, known limitations
CLAUDE.md / AGENTS.md / SKILLS.md	Root operating rules, workflow, review roles
backend/CLAUDE.md / AGENTS.md	Backend rules — read before touching backend/
docs/workflow_detail.md	The DO workflow
docs/workflow_detail_aplikasi.md	Field-by-field extraction rules
docs/regex_rules_example.md	The regex rule set
docs/simulasi_ekstraksi_lengkap.md	A worked extraction example
docs/api-contract-map.md	Flutter/backend contract and gaps G1–G10
docs/stock-feature-plan.md	Planned Stocks feature
docs/expiry-tracking-plan.md	Planned per-sale expiry tracking
docs/feature-list.md	What shipped, Flutter
docs/iteration-log.md	Review and audit trail, Flutter
plans/next-enhancements.md	Flutter backlog (live)
backend/docs/scan-product.md	Product Scan internals and retraining
backend/docs/vllm-service.md	The vLLM service
backend/docs/feature-list.md	What shipped, backend
backend/docs/iteration-log.md	Review and audit trail, backend
backend/plans/next-enhancements.md	Backend backlog (live)
screenshots/v2/WORKFLOW.md	Screen-by-screen app walkthrough (narrated)
screenshots/v2/*.png	The 30 phone screenshots used in Appendix D
screenshots/*.png	Admin and internal web tooling screenshots
docs/handover/	This document, the deck, and their generators

For structural questions — what calls X, what breaks if Y is renamed — an AST-derived knowledge graph lives in `graphify-out/`, rebuilt automatically by the `git post-commit` hook. Query it before grepping.

Appendix D — The app, screen by screen

Captured on a real device against the live backend — real OCR output, real DINOv2 classification, real error states. Source images live in screenshots/v2/ (phone) and screenshots/ (web), with the original narrated walkthrough in screenshots/v2/WORKFLOW.md.

D.1 The shell — login, permissions, drawer, logout

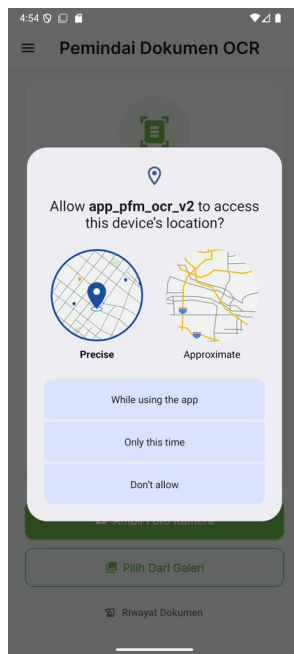


1 Splash. Runs about 3s of fixed delay before the auth check even starts — backlog 1.5.

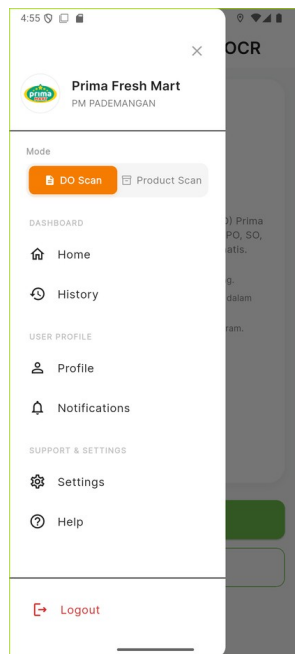
2 Login, empty.

3 Login, filled. Real bcrypt check against the accounts table.

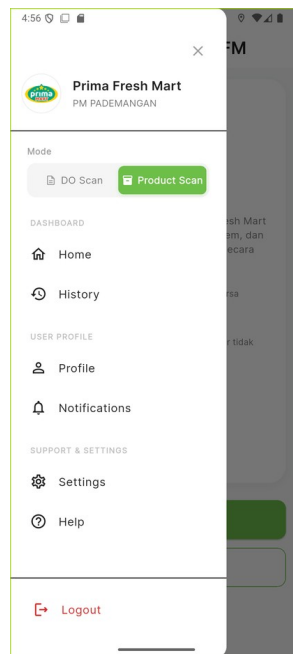
4 Wrong password — a real 401 from /api/v1/auth/login.



5 Location permission. Every denial path is only logged today — backlog 2.1.



6 Navigation drawer, DO to mode.



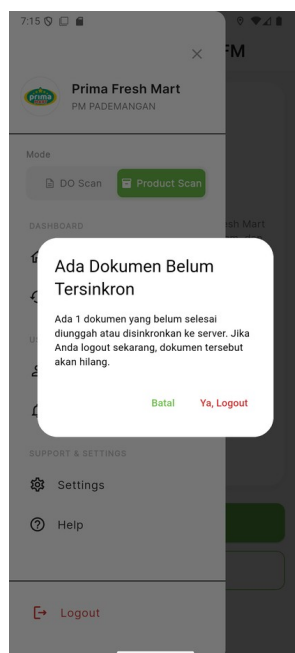
7 Mode toggle — DO to Product.



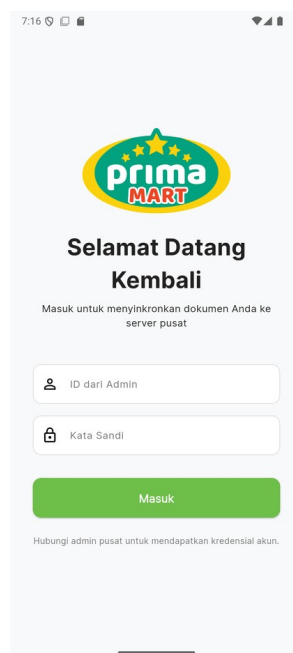
8 Help dialog.



9 An unimplemented drawer entry.



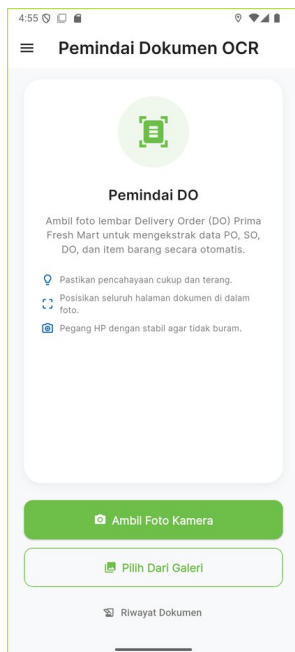
10 Logout warns about an unsynced queue.



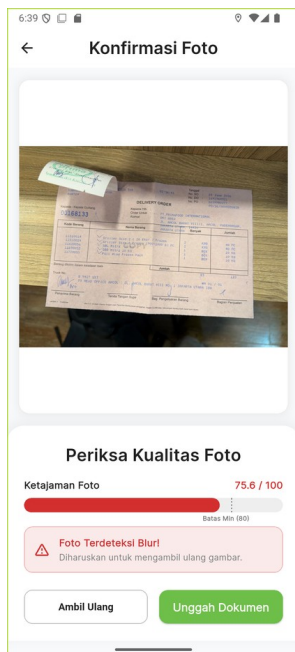
11 Logged out, back to login.

D.2 DO scan — the main flow

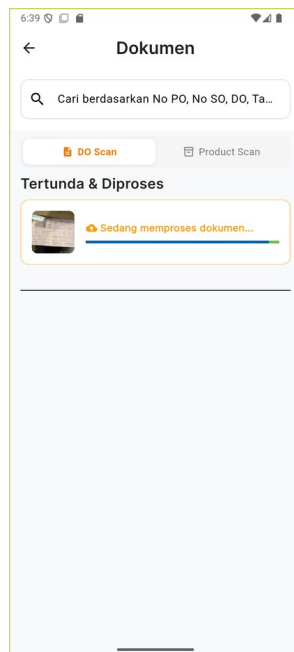
The nine stages from section 4, as the store staff member actually sees them.



1 Camera home, DO mode. Blue chrome marks DO.



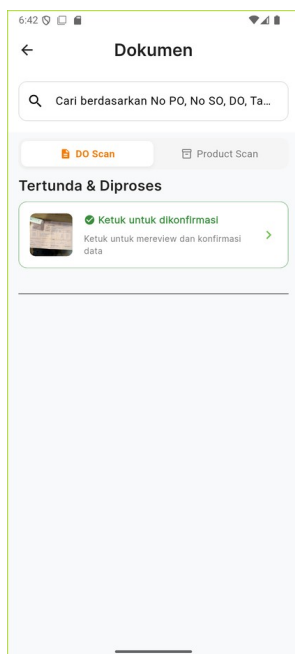
2 Real Laplacian blur check. The badge is shown but does NOT block upload — backlog 2.4.



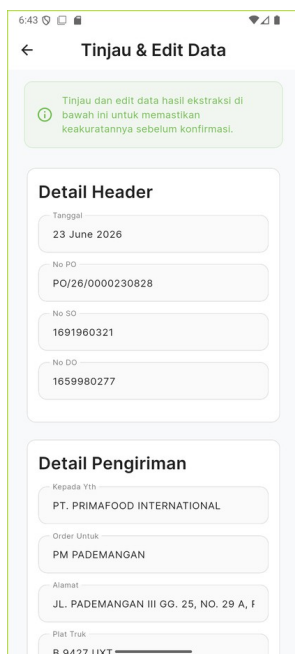
3 Queued. Persisted to Hive, so it survives an app kill.



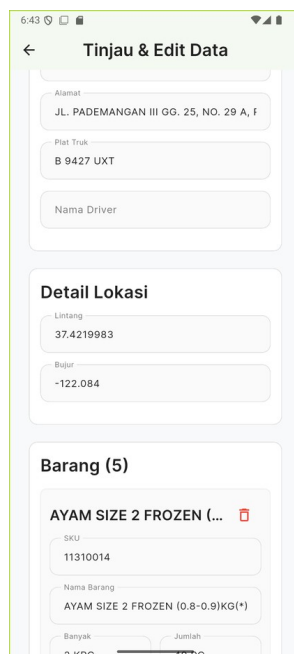
4 Timeout. Connectivity failures report the same way — backlog 3.5.



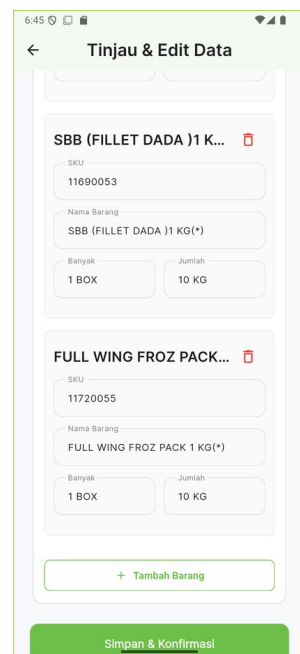
5 parsed = true. The 2s poll found it.



6 Editor, header fields — real OCR output.



7 Line items, after SKU and unit correction against master data.



8 Simpan tapped and nothing happens — the silent validation failure.

9 The same form once the error is surfaced.

10 Confirmation dialog.

11 Filled. The receiver name is free text — backlog 4.6.

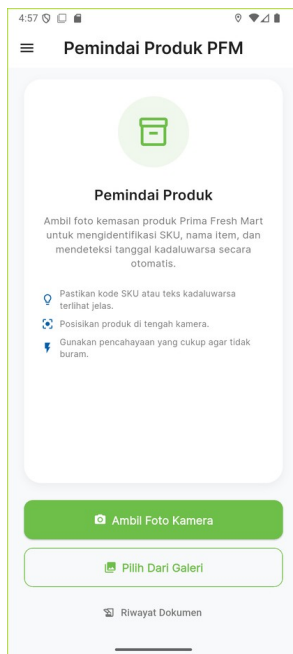
12 Saved. The PUT runs inside a DB transaction.

13 History, DO tab.

14 The delivery receipt PDF, generated on-device.

D.3 Product Scan

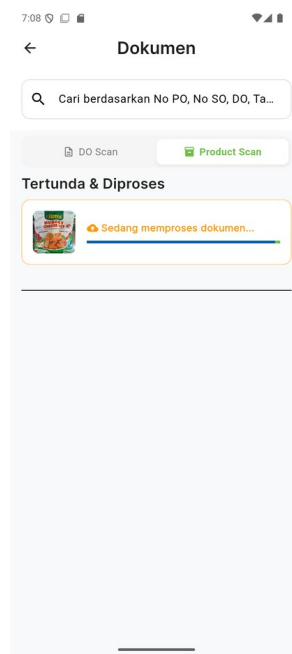
The secondary flow from section 8. Teal chrome marks Product mode the way blue marks DO mode.



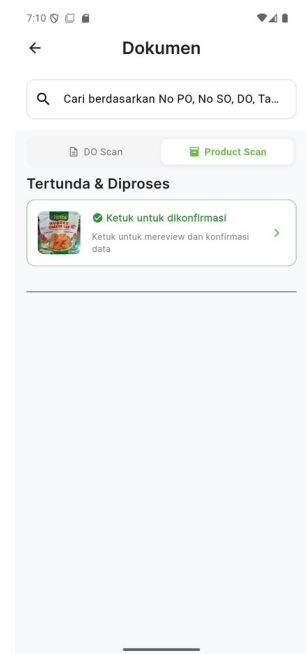
1 Camera home, Product mode.



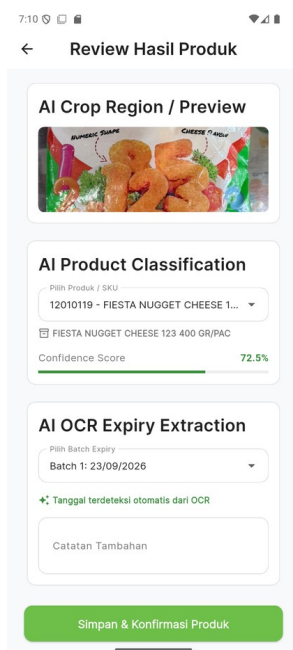
2 Single-product photo, same blur check.



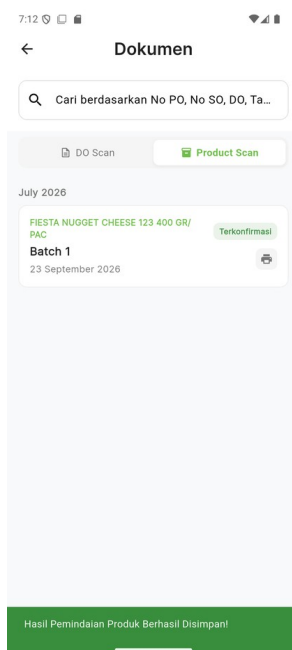
3 Queued for classification.



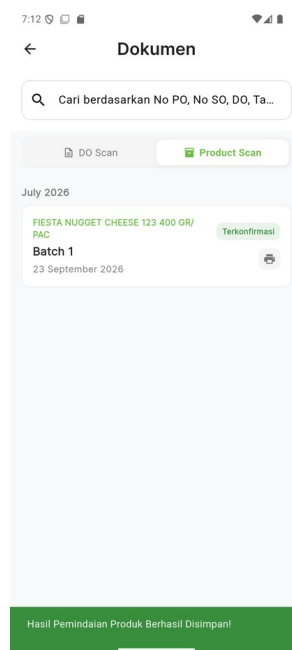
4 Classified and ready to review.



5 Real DINOv2 match plus the OCR expiry date.



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



7 History, Product tab.

D.4 The internal web tooling

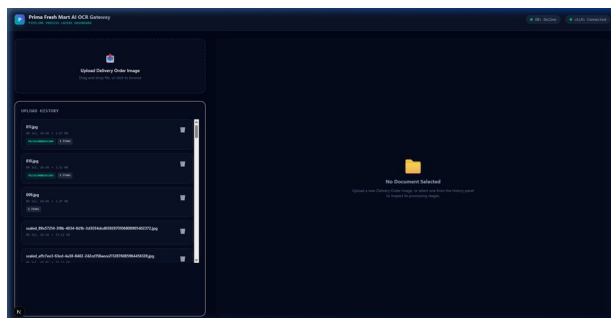
The dev-only Next.js surface from section 6. The successor inherits these too — they are how master data is inspected and how ground truth gets labelled for the accuracy harness. No login screen, by design.

ID	Name	Type	Address	Action
1000001	PT. JAYA BAKAR	1	Jl. Raya Bontomatene No. 12, RT. 02, RW. 02, Kec. Bontomatene, Kota Makassar	Detail
1000002	PT. JAYA BAKAR 2	1	Jl. Raya Bontomatene No. 12, RT. 02, RW. 02, Kec. Bontomatene, Kota Makassar	Detail
1000003	PT. JAYA BAKAR 3	1	Jl. Raya Bontomatene No. 12, RT. 02, RW. 02, Kec. Bontomatene, Kota Makassar	Detail
1000004	PT. JAYA BAKAR 4	1	Jl. Raya Bontomatene No. 12, RT. 02, RW. 02, Kec. Bontomatene, Kota Makassar	Detail
1000005	PT. JAYA BAKAR 5	1	Jl. Raya Bontomatene No. 12, RT. 02, RW. 02, Kec. Bontomatene, Kota Makassar	Detail
1000006	PT. JAYA BAKAR 6	1	Jl. Raya Bontomatene No. 12, RT. 02, RW. 02, Kec. Bontomatene, Kota Makassar	Detail
1000007	PT. JAYA BAKAR 7	1	Jl. Raya Bontomatene No. 12, RT. 02, RW. 02, Kec. Bontomatene, Kota Makassar	Detail
1000008	PT. JAYA BAKAR 8	1	Jl. Raya Bontomatene No. 12, RT. 02, RW. 02, Kec. Bontomatene, Kota Makassar	Detail
1000009	PT. JAYA BAKAR 9	1	Jl. Raya Bontomatene No. 12, RT. 02, RW. 02, Kec. Bontomatene, Kota Makassar	Detail
1000010	PT. JAYA BAKAR 10	1	Jl. Raya Bontomatene No. 12, RT. 02, RW. 02, Kec. Bontomatene, Kota Makassar	Detail

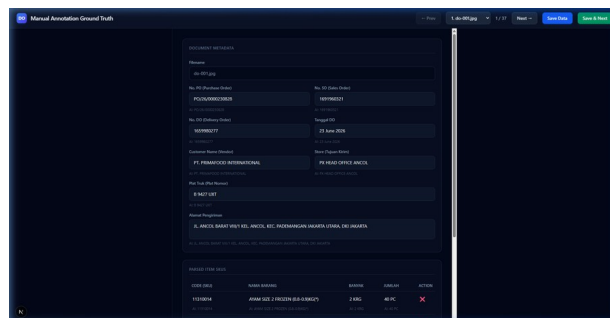
1 Admin — stores.

ID	Name	Type	Price	Action
1000001	SKU 001	1	10000	Detail
1000002	SKU 002	1	10000	Detail
1000003	SKU 003	1	10000	Detail
1000004	SKU 004	1	10000	Detail
1000005	SKU 005	1	10000	Detail
1000006	SKU 006	1	10000	Detail
1000007	SKU 007	1	10000	Detail
1000008	SKU 008	1	10000	Detail
1000009	SKU 009	1	10000	Detail
1000010	SKU 010	1	10000	Detail

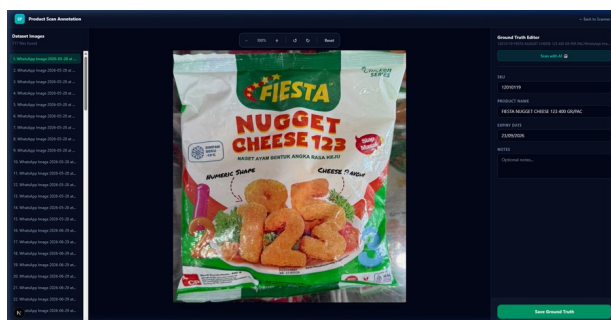
2 Admin — SKU master.



3 Dashboard.



4 Manual labelling — DO ground truth. This is what produces sources/manual_labels.json.



5 Manual labelling — product ground truth.