Plock — Demo Readiness

Canonical PLOCK baseline package, QA review, and narrow verification notes.

Demo Readiness — Index

Source: `~/ALAI/products/Plock/docs/demo-readiness/INDEX.md`

PLOCK Demo Readiness — Canonical Baseline Index

Date: 2026-03-14
Purpose: Entry point for all agents and humans working on PLOCK demo-readiness, documentation QA, and subsequent backlog generation.

Read in This Order

1. 00-canonical-source-of-truth.md
2. 01-user-stories-baseline.md
3. 02-feature-baseline.md
4. 03-gaps-and-next-steps.md
5. 04-qa-checklist-baseline.md
6. 05-initial-package-review.md
7. 06-api-spec-verification-pass.md

Working Rules

- Use **repo reality first**.
- Do **not** use `docs/ARCHITECTURE.md` as canonical architecture input.
- Do **not** trust agent narrative output unless the claimed files actually exist.
- All future QA and backlog generation for PLOCK must reference this baseline package.

Status

This package is a manually curated canonical baseline created after discovering:

- documentation drift,
- stale/conflicting architecture docs,
- and unreliable delegated documentation artifact generation.

It is the temporary source of truth until a fuller reviewed demo-readiness package replaces it.

Canonical Source of Truth

Source: `~/ALAI/products/Plock/docs/demo-readiness/00-canonical-source-of-truth.md`

PLOCK — Canonical Source of Truth

Date: 2026-03-14
Purpose: Define which files are authoritative for PLOCK and which files are stale/conflicting, so all future documentation, QA, and implementation work is based on real system state.

---

1. Why This Exists

PLOCK currently has a real codebase and substantial documentation, but not all documentation reflects the same architecture.

Without a canonical source-of-truth map:

- user stories can be extracted from stale assumptions
- feature catalogs can be built against the wrong architecture
- QA can validate one document against another document that is itself outdated
- implementation tasks can be generated against a system that does not match the actual repo

This document defines what is authoritative and what is not.

---

2. Authority Order

When files conflict, use this precedence:

1. **Actual code and repository structure**
2. **Database migrations and runtime configuration**
3. **BUILD-BLUEPRINT.md**

4. **README.md**
5. **Runbooks and operational docs**
6. **ADR documents**
7. **PRD / GTM / Regulatory docs**
8. **API-SPEC.md** (reference only unless confirmed against code)
9. **Stale/conflicting docs must not be used as source of truth**

---

3. Authoritative Sources

3.1 Codebase

The following repo areas are authoritative:

- `backend/`
- `frontend/`
- `backend/src/main/resources/db/migration/`
- `tests/`
- `e2e/`
- `infrastructure/`

3.2 Primary Architecture / Build Docs

These are currently the strongest documentation sources:

- `README.md`
- `BUILD-BLUEPRINT.md`
- `RUNBOOK.md`
- `docs/RUNBOOK.md`
- `docs/RLS-AUDIT.md`
- `docs/SECRETS-MANAGEMENT.md`

3.3 Product / Business / Compliance Docs

These are authoritative for product intent, market framing, compliance assumptions, and user-level needs — but must still be reconciled against code:

- `docs/PRD.md`
- `docs/GTM-STRATEGY.md`
- `docs/REGULATORY.md`

3.4 ADRs

Use as authoritative for declared decisions when they match repo reality:

- `adr/ADR-001-flyway-over-prisma.md`
- `adr/ADR-002-module-federation.md`
- `adr/ADR-003-rls-tenant-isolation.md`

---

4. Current Real Architecture

Based on actual code and authoritative docs, the real current system is:

Backend

- **Kotlin**
- **Ktor**
- **Exposed ORM**
- **Flyway migrations**
- **PostgreSQL**
- **Redis**

Frontend

- **React 19**
- **TypeScript**
- **Vite**
- **Module Federation**
- **Micro-frontends**
  - `shell`
  - `mfe-inventory`
  - `mfe-orders`
  - `mfe-picking`
  - `mfe-settings`
  - `mfe-ai`

Infrastructure / Deployment

- Railway-oriented deployment and infra
- Docker / docker-compose for local setup
- Terraform and deployment helper scripts under `infrastructure/`

Multi-tenancy

- Tenant boundary is `warehouse_id`
- Isolation enforced via **PostgreSQL RLS**
- Tenant context set through application request flow

Integrations

- Fortnox
- PostNord
- DHL
- Instabee

---

5. Semi-Authoritative / Needs Verification

`docs/API-SPEC.md`

This file is useful as a product/API intent document, but it must be verified against the real Ktor routing and actual backend implementation before being treated as canonical.

Use it as:

- reference
- gap analysis input
- API-intent documentation

Do **not** use it as final truth without code verification.

---

6. Stale / Conflicting Sources

`docs/ARCHITECTURE.md`

This file is currently **stale/conflicting** and must **not** be used as the main architecture source.

It describes a different architecture, including:

- Next.js
- Express
- Prisma
- Python ML microservice
- Supabase
- Upstash
- `organization_id`-based tenancy

That conflicts with the actual repo and stronger sources, which show:

- Kotlin/Ktor
- Exposed + Flyway
- React + Vite MFEs
- PostgreSQL + Redis
- `warehouse_id` tenant model

Rule: `docs/ARCHITECTURE.md` may be used only as historical/stale material until rewritten or archived.

---

7. Operational Constraints

7.1 Agent Output Is Not Source of Truth

The following are **not authoritative**:

- `/tmp/verify-*`
- `/tmp/gotcha-task-*`
- agent-generated summaries that claim files were created but no files exist
- MC task status alone (`done`, `blocked`, etc.) without artifact verification

7.2 Deliverable Existence Is Required

Any documentation task that claims output files must be verified against the actual filesystem. A claimed file path is not considered real unless:

- the file exists
- it is readable
- its contents are relevant to the requested task

---

8. What Future Docs Must Be Based On

The next canonical PLOCK documentation pass must be based on:

User Stories

- `docs/PRD.md`
- actual route/service domains in backend
- AI strategy / GTM / regulatory documents where relevant

Feature Catalog

- backend routes and services
- frontend MFEs
- migrations / schema
- integrations
- security / RLS docs

ADR / BDR

- `adr/*`
- BUILD-BLUEPRINT
- actual code decisions already visible in repo
- business decisions visible in PRD / GTM / pricing docs

Test Scenarios

- existing backend tests
- existing e2e tests
- route/service/module structure
- runbook incident scenarios
- compliance-sensitive workflows

---

9. Current Remaining Gaps

The baseline canonical package now exists under `docs/demo-readiness/`, but important follow-up artifacts and cleanups still remain.

Current remaining gaps:

- no canonical `00-program-map.md` yet
- no canonical `00a-deterministic-task-contract.md` yet
- no canonical `01-doc-inventory-gap-audit.md` yet
- `docs/API-SPEC.md` still needs explicit verification against real Ktor routes before it can be treated as canonical
- `docs/ARCHITECTURE.md` is still stale/conflicting and has not yet been rewritten or archived

Current hygiene issue:

- `infrastructure/README.md` contains secret-like example material and must be reviewed/cleaned

---

10. Working Rule for All Remaining PLOCK Work

Until a new canonical documentation package exists:

- use **repo reality first**
- use **BUILD-BLUEPRINT + README + RLS-AUDIT + runbooks** as architectural truth
- use **PRD + GTM + REGULATORY** for business, user, and market truth
- treat `docs/ARCHITECTURE.md` as stale
- treat agent narrative output as non-authoritative unless backed by real files

---

11. Decision

This document is the baseline authority map for all remaining PLOCK planning, QA, and implementation prep work.

Any future PLOCK task that produces:

- user stories
- feature lists
- ADR/BDR registers
- QA checklists
- implementation backlog

must explicitly reference this authority model before proceeding.

User Stories Baseline

Source: `~/ALAI/products/Plock/docs/demo-readiness/01-user-stories-baseline.md`

PLOCK — User Stories Baseline

Date: 2026-03-14
Purpose: Establish a first canonical user-story baseline for PLOCK using real repository evidence and authoritative product documents. This is not yet the final polished story set; it is the operational baseline for documentation, QA, and backlog slicing.

---

1. Scope and Method

This baseline is derived from:

- `docs/PRD.md`
- actual backend route/service structure
- actual frontend MFE structure
- integration modules present in the repository
- operational and regulatory documentation where relevant

This document intentionally avoids inventing features not supported by either:

- current product docs, or
- current repository structure

---

2. Story Status Model

Each story is tagged as one of:

- **exists** — strongly supported by real code/routes/docs
- **partial** — visible in docs and/or code structure, but maturity unclear
- **planned** — described in docs, but implementation evidence is limited or phase-dependent

---

3. Receiving / Inbound

US-INB-001 — Receive goods via barcode

Status: exists
Evidence: `PRD.md`, `ReceivingRoutes.kt`, `ReceivingService.kt`, barcode-related backend structure

As a warehouse worker
I want to scan a barcode and identify the purchase order it belongs to
So that I can receive goods correctly without manual lookup

US-INB-002 — View open purchase orders

Status: exists
Evidence: `PRD.md`, `ReceivingRoutes.kt`, supplier/receiving route structure

As a warehouse manager
I want to see all open purchase orders and expected arrival dates
So that I can plan staffing and dock capacity

US-INB-003 — Guided putaway after receiving

Status: partial
Evidence: `PRD.md`, `LocationRoutes.kt`, `PutAwayService.kt`

As a warehouse worker
I want to be guided to the correct bin location after receiving
So that I do not need to memorize where products belong

US-INB-004 — Detect receiving discrepancies

Status: exists
Evidence: `PRD.md`, `ReceivingDiscrepancyTest.kt`, discrepancy migration/table evidence

As a warehouse manager
I want to be notified immediately when received quantities do not match the purchase order
So that I can resolve supplier issues before inventory is corrupted

---

4. Inventory Management

US-INV-001 — Real-time inventory visibility

Status: exists
Evidence: `PRD.md`, `InventoryRoutes.kt`, `InventoryService.kt`

As a warehouse manager
I want to see the exact quantity of a SKU by location in real time
So that I can answer operational questions without walking the warehouse

US-INV-002 — Reorder point alerting

Status: partial
Evidence: `PRD.md`, inventory domain present; implementation maturity unclear

As a warehouse manager
I want to receive an alert when inventory falls below reorder point
So that I can prevent avoidable stockouts

US-INV-003 — Zone-based cycle counts

Status: exists
Evidence: `PRD.md`, `CycleCountRoutes.kt`, `CycleCountService.kt`, tests

As a warehouse manager
I want to run a cycle count for a specific zone without stopping all operations
So that I can maintain inventory accuracy continuously

US-INV-004 — Mobile-assisted physical inventory count

Status: partial
Evidence: `PRD.md`, V7 mobile support migrations, runbook/mobile references

As a warehouse worker
I want to perform physical inventory counts on a mobile device
So that counting is faster and less error-prone than paper workflows

---

5. Orders / Outbound

US-OUT-001 — Release urgent orders quickly

Status: exists
Evidence: `PRD.md`, `OrderRoutes.kt`, order/picking domain structure

As a warehouse manager
I want to release urgent orders with a single action
So that I can respond quickly to escalations and deadlines

US-OUT-002 — Work from structured picking flow

Status: exists
Evidence: `PickingRoutes.kt`, `PickingService.kt`, `mfe-picking`

As a picker
I want to receive a structured picking workflow
So that I can pick efficiently and consistently

US-OUT-003 — Verify packing with product photo

Status: partial
Evidence: `PRD.md`, frontend/product/packing intent visible in docs

As a packing worker
I want to see a product photo during packing
So that I can confirm I am sealing the correct item

US-OUT-004 — Unified shipment tracking view

Status: partial
Evidence: `PRD.md`, `ShipmentRoutes.kt`, carrier integration structure

As a warehouse manager
I want to see outbound shipment status in one place
So that I can proactively handle delays and customer issues

---

6. Shipping / Carrier Operations

US-SHP-001 — Print carrier labels directly from Plock

Status: exists
Evidence: `PRD.md`, PostNord/DHL/Instabee integration code, shipment/carrier routes

As a warehouse worker
I want to print shipping labels directly from Plock
So that I do not need to log into separate carrier portals

US-SHP-002 — Use multiple carriers in one operational flow

Status: exists
Evidence: `CarrierRoutes.kt`, `CarrierIntegrationRoutes.kt`, multiple carrier adapters/services

As a warehouse manager
I want to work with PostNord, DHL, and Instabee in one system
So that outbound shipping does not depend on multiple manual carrier workflows

US-SHP-003 — Track carrier sync and integration health

Status: partial
Evidence: `integration_sync_log` schema, runbook sections for Fortnox/carrier failures

As a system operator
I want to inspect carrier/integration sync behavior
So that I can detect and troubleshoot failures quickly

---

7. AI / Smart Warehouse Operations

US-AI-001 — Ask warehouse questions in Swedish

Status: partial-to-exists
Evidence: `PRD.md`, `AI-STRATEGY.md`, `mfe-ai/`, product positioning

As a warehouse manager
I want to ask warehouse questions in Swedish and get immediate answers
So that I can make decisions without SQL, exports, or manual searching

US-AI-002 — Execute warehouse commands via AI with confirmation

Status: planned / partial
Evidence: `PRD.md`, `AI-STRATEGY.md`

As a warehouse manager
I want to trigger operational actions through natural language with explicit confirmation
So that I can manage exceptions faster than through deep UI navigation

US-AI-003 — Optimize pick routes

Status: partial-to-exists
Evidence: `PRD.md`, `AI-STRATEGY.md`, picking domain, runbook references to Smart Picking

As a warehouse manager
I want to use smart pick route optimization
So that workers spend less time walking and more time picking

---

8. Dashboard / Monitoring

US-DASH-001 — Operational dashboard

Status: exists
Evidence: `DashboardRoutes.kt`, `PRD.md`, dashboard domain

As a warehouse manager
I want to see a real-time dashboard with today’s operational KPIs
So that I can spot bottlenecks before they become incidents

US-DASH-002 — Monitor shipment and fulfillment progress

Status: partial
Evidence: dashboard/shipment routes, PRD

As a operations lead
I want to monitor fulfillment progress and pending shipments
So that I can intervene before SLAs are missed

---

9. Users / Roles / Multi-Tenancy

US-SEC-001 — Manage users and roles

Status: exists
Evidence: `UserRoutes.kt`, `RoleRoutes.kt`, `RbacService.kt`

As an admin
I want to manage users and roles
So that only the right people can perform sensitive warehouse actions

US-SEC-002 — Restrict data by warehouse

Status: exists
Evidence: `TenantPlugin.kt`, `RLS-AUDIT.md`, RLS migrations

As a warehouse operator
I want to only see data for my warehouse
So that tenant/customer separation is enforced safely

US-SEC-003 — Keep tenant isolation enforced at the DB layer

Status: exists
Evidence: `RLS-AUDIT.md`, migration `V6__rls_tenant_isolation.sql`

As the platform
I want to enforce tenant isolation in PostgreSQL using RLS
So that application bugs do not automatically become cross-tenant data leaks

---

10. Integrations / Accounting

US-INT-001 — Sync warehouse/accounting data with Fortnox

Status: exists
Evidence: Fortnox integration modules, docs, secrets, runbook

As a Swedish SMB operator
I want to integrate Plock with Fortnox
So that warehouse operations and accounting stay aligned

US-INT-002 — Re-authorize or recover from Fortnox sync failure

Status: partial-to-exists
Evidence: Fortnox OAuth/sync routes, runbook incident section

As a tenant admin
I want to reconnect or recover the Fortnox integration when it fails
So that business operations are not blocked by stale tokens or sync errors

---

11. 3PL / Client Isolation (Phase-Dependent)

US-3PL-001 — Separate client inventory views

Status: planned
Evidence: `PRD.md`, `GTM-STRATEGY.md`, regulatory references

As a 3PL operations lead
I want to see inventory separated by client
So that I can manage a multi-client warehouse safely

US-3PL-002 — Support 3PL billing workflows

Status: planned
Evidence: `REGULATORY.md`, GTM and pricing logic

As a 3PL operator
I want to support warehousing-service billing workflows
So that customer charging is structured and scalable

---

12. Business / Sales / Market Layer

US-BIZ-001 — Position as Fortnox-next-step WMS

Status: planned/business
Evidence: `GTM-STRATEGY.md`

As the business
I want to position Plock as the natural next step after Fortnox Lager
So that market entry is focused and credible

US-BIZ-002 — Support Swedish SMB self-serve onboarding motion

Status: planned/business
Evidence: GTM and product docs

As the product business
I want to enable a low-friction onboarding and trial motion
So that acquisition cost stays low and distribution scales

---

13. Summary

Strongly supported by current code/docs

- receiving
- inventory
- orders
- picking
- shipments
- carriers

- RBAC
- tenant isolation
- Fortnox integration
- dashboard
- barcode
- audit/webhook/SSE infrastructure

Present but maturity unclear

- AI chat execution depth
- smart picking maturity
- mobile/PWA operational depth
- packing/photo verification
- full shipment monitoring UX
- reorder alerting maturity

More Phase 2 / strategic than current MVP

- 3PL client isolation workflows
- 3PL billing
- full commercial/demo/support packaging
- polished sales enablement materials

---

14. Working Rule

This user-story baseline is the first canonical operational extraction.

Any future:

- feature catalog
- QA gate
- implementation backlog
- demo script
- support playbook

must map back to these stories and mark each item as:

- exists
- partial
- planned

Feature Baseline

Source: `~/ALAI/products/Plock/docs/demo-readiness/02-feature-baseline.md`

PLOCK — Feature Baseline

Date: 2026-03-14
Purpose: Establish a first canonical feature baseline for PLOCK using real repository evidence and authoritative documentation. This is the operational feature map that future QA, backlog slicing, implementation planning, and demo prep should build on.

---

1. Method

This baseline is based on:

- actual repository structure
- backend route/service modules
- frontend micro-frontend structure
- database migrations
- runbooks, security, and integration docs
- PRD / GTM / regulatory context where relevant

Every feature is tagged with a maturity label:

- **exists** — strongly supported by code + docs
- **partial** — real evidence exists, but depth/completeness is unclear
- **planned** — documented as intended direction, but implementation evidence is weak or phase-dependent

---

2. Feature Groups

1. Platform / architecture
2. Backend domain features
3. Frontend / UX features
4. Database / tenant isolation

5. Integrations
6. AI features
7. Testing / QA
8. Ops / deployment / observability
9. Security
10. Business / GTM / support
11. Known contradictions and gaps

---

3. Platform / Architecture

F-PLAT-001 — Kotlin/Ktor backend platform

Status: exists
Evidence: `backend/build.gradle.kts`, `Application.kt`, Ktor plugins and route modules

PLOCK uses a Kotlin/Ktor backend, not an Express/Node backend.

F-PLAT-002 — React micro-frontend frontend platform

Status: exists
Evidence: `frontend/`, `frontend/shell/package.json`, MFE folders, Vite setup

PLOCK frontend is a React 19 + TypeScript + Vite Module Federation system with multiple MFEs.

F-PLAT-003 — Monorepo with mixed backend/frontend tooling

Status: exists
Evidence: root `package.json`, `backend/`, `frontend/`, workspaces, Turborepo scripts

PLOCK uses:

- Gradle for backend
- npm/Turborepo workspace structure for frontend packages and MFEs

F-PLAT-004 — Railway-oriented deployment model

Status: partial-to-exists
Evidence: `infrastructure/README.md`, Terraform files, Railway env references

The deployment model is strongly Railway-oriented, although production readiness should still be independently verified.

---

4. Backend Domain Features

F-BE-001 — Authentication

Status: exists
Evidence: `AuthRoutes.kt`, `Authentication.kt`, JWT config in `application.yaml`

F-BE-002 — Warehouse management

Status: exists
Evidence: `WarehouseRoutes.kt`, warehouse references in API/docs, migrations

F-BE-003 — Location/bin management

Status: exists
Evidence: `LocationRoutes.kt`, `LocationService.kt`, locations schema

F-BE-004 — Product management

Status: exists
Evidence: `ProductRoutes.kt`, `products` table, DTOs/services

F-BE-005 — Inventory management

Status: exists
Evidence: `InventoryRoutes.kt`, `InventoryService.kt`, inventory schema, tests

F-BE-006 — Order management

Status: exists
Evidence: `OrderRoutes.kt`, `OrderService.kt`, order schema

F-BE-007 — Picking workflow

Status: exists
Evidence: `PickingRoutes.kt`, `PickingService.kt`, `mfe-picking`, tests

F-BE-008 — Receiving workflow

Status: exists
Evidence: `ReceivingRoutes.kt`, `ReceivingService.kt`, receiving tables/migrations, discrepancy test

F-BE-009 — Shipment management

Status: exists
Evidence: `ShipmentRoutes.kt`, `ShipmentService.kt`, shipment schema, tests

F-BE-010 — Carrier management

Status: exists
Evidence: `CarrierRoutes.kt`, `CarrierService.kt`, `carriers` schema

F-BE-011 — Carrier integration endpoints

Status: exists
Evidence: `CarrierIntegrationRoutes.kt`, integration clients and services

F-BE-012 — Cycle counts

Status: exists
Evidence: `CycleCountRoutes.kt`, `CycleCountService.kt`, tests, schema

F-BE-013 — Stock movements / audit trail for inventory changes

Status: exists
Evidence: `StockMovementRoutes.kt`, `stock_movements` table, schema

F-BE-014 — Returns workflow

Status: exists
Evidence: `ReturnRoutes.kt`, `ReturnService.kt`, return schema

F-BE-015 — Supplier management

Status: exists
Evidence: `SupplierRoutes.kt`, `SupplierService.kt`, supplier schema

F-BE-016 — Zone management

Status: exists
Evidence: `ZoneRoutes.kt`, `ZoneService.kt`, zones schema

F-BE-017 — User and role management

Status: exists
Evidence: `UserRoutes.kt`, `RoleRoutes.kt`, `RbacService.kt`

F-BE-018 — Dashboard endpoints

Status: exists
Evidence: `DashboardRoutes.kt`

F-BE-019 — Barcode support

Status: exists
Evidence: `BarcodeRoutes.kt`, `BarcodeService.kt`, ZXing dependencies

F-BE-020 — Audit endpoints

Status: exists
Evidence: `AuditRoutes.kt`, `AuditService.kt`, audit schema

F-BE-021 — SSE / real-time update infrastructure

Status: exists
Evidence: `SseRoutes.kt`, `SseService.kt`, Ktor SSE dependency

F-BE-022 — Webhooks

Status: exists
Evidence: `WebhookRoutes.kt`, `WebhookService.kt`, webhook tables

F-BE-023 — Offline/mobile sync support

Status: partial
Evidence: `OfflineSyncRoutes.kt`, `OfflineSyncService.kt`, `V7__mobile_pwa_support.sql`

F-BE-024 — Health endpoints

Status: exists
Evidence: `HealthRoutes.kt`, runbooks

---

5. Frontend / UX Features

F-FE-001 — Shell application

Status: exists
Evidence: `frontend/shell`

F-FE-002 — Inventory MFE

Status: exists
Evidence: `frontend/mfe-inventory`

F-FE-003 — Orders MFE

Status: exists
Evidence: `frontend/mfe-orders`

F-FE-004 — Picking MFE

Status: exists
Evidence: `frontend/mfe-picking`

F-FE-005 — Settings MFE

Status: exists
Evidence: `frontend/mfe-settings`

F-FE-006 — AI MFE

Status: partial-to-exists
Evidence: `frontend/mfe-ai`, AI strategy docs

F-FE-007 — Shared shell + MFE architecture

Status: exists
Evidence: frontend repo structure, module federation dependency, blueprint/docs

---

6. Database / Tenant Isolation Features

F-DB-001 — PostgreSQL primary database

Status: exists
Evidence: build config, env docs, migrations, runbooks

F-DB-002 — Flyway-managed schema

Status: exists
Evidence: Flyway dependency, migrations, ADR references

F-DB-003 — RLS tenant isolation by `warehouse_id`

Status: exists
Evidence: `RLS-AUDIT.md`, V6 migration, TenantPlugin docs

F-DB-004 — Warehouse-scoped data model

Status: exists
Evidence: RLS docs, warehouse-based route and domain model

F-DB-005 — Encrypted Fortnox token storage

Status: exists
Evidence: `FortnoxTokens.kt`, `CryptoService.kt`, docs

F-DB-006 — Mobile/offline support tables

Status: partial
Evidence: `V7__mobile_pwa_support.sql`

---

7. Integration Features

F-INT-001 — Fortnox OAuth

Status: exists
Evidence: `FortnoxOAuthRoutes.kt`, secrets docs

F-INT-002 — Fortnox sync flow

Status: exists
Evidence: `FortnoxSyncRoutes.kt`, `FortnoxSyncService.kt`, runbook incident section

F-INT-003 — PostNord integration

Status: exists
Evidence: PostNord client/service files, docs, secrets config

F-INT-004 — DHL integration

Status: exists
Evidence: DHL client/service files, docs, secrets config

F-INT-005 — Instabee integration

Status: exists
Evidence: Instabee client/service files, docs, secrets config

F-INT-006 — Integration sync logging

Status: exists
Evidence: `integration_sync_log` schema, runbook references

---

8. AI Features

F-AI-001 — AI Chat positioning and product surface

Status: partial
Evidence: `AI-STRATEGY.md`, PRD, `mfe-ai`

F-AI-002 — Smart Picking / route optimization

Status: partial
Evidence: PRD, AI strategy, runbook mentions

F-AI-003 — Natural language warehouse querying

Status: planned-to-partial
Evidence: PRD + AI strategy

---

9. Testing / QA Features

F-QA-001 — Backend automated tests

Status: exists
Evidence: `backend/src/test/kotlin/*`

F-QA-002 — E2E tests

Status: exists
Evidence: `e2e/app.spec.ts`, `e2e/inventory.spec.ts`

F-QA-003 — Performance testing structure

Status: partial
Evidence: `tests/performance/`, `infrastructure/k6/`

F-QA-004 — Regression test structure

Status: partial
Evidence: `tests/regression/`

---

10. Ops / Deployment / Observability Features

F-OPS-001 — Local docker-based dev environment

Status: exists
Evidence: `docker-compose.yml`, runbooks

F-OPS-002 — Deployment scripts / Terraform infra

Status: exists
Evidence: `infrastructure/README.md`, `main.tf`, scripts

F-OPS-003 — Runbooks for incident handling

Status: exists
Evidence: `RUNBOOK.md`, `docs/RUNBOOK.md`

F-OPS-004 — Sentry-based monitoring

Status: partial-to-exists
Evidence: backend and frontend Sentry references, runbooks, secrets docs

---

11. Security Features

F-SEC-001 — JWT auth

Status: exists
Evidence: auth plugin, `application.yaml`, docs

F-SEC-002 — BCrypt password hashing

Status: exists
Evidence: backend dependencies, security docs

F-SEC-003 — AES-256-GCM token encryption

Status: exists
Evidence: `CryptoService.kt`, Fortnox token docs

F-SEC-004 — PostgreSQL RLS enforcement

Status: exists
Evidence: RLS migration and audit docs

F-SEC-005 — Role-based access control

Status: exists
Evidence: role/user routes and services

F-SEC-006 — Secret management model

Status: exists but needs cleanup
Evidence: `docs/SECRETS-MANAGEMENT.md`, infra docs

---

12. Business / GTM / Commercial Features

F-BIZ-001 — Pricing model

Status: exists in docs
Evidence: `README.md`, `PRD.md`, `GTM-STRATEGY.md`

F-BIZ-002 — Fortnox ecosystem GTM strategy

Status: exists in docs
Evidence: `GTM-STRATEGY.md`

F-BIZ-003 — Swedish SMB WMS positioning

Status: exists in docs
Evidence: README, PRD, GTM

F-BIZ-004 — 3PL commercial direction

Status: planned
Evidence: PRD, GTM, regulatory references

F-BIZ-005 — Regulatory/compliance framing

Status: exists in docs
Evidence: `REGULATORY.md`

---

13. Known Contradictions / Risks

RISK-001 — Stale architecture document

`docs/ARCHITECTURE.md` conflicts with actual repo reality and must not be used as canonical architecture input.

RISK-002 — API spec likely drifted

`docs/API-SPEC.md` appears to reflect an org-centric model and may not be fully synchronized with current Ktor routing and warehouse-based tenancy.

RISK-003 — Runbook drift

There are conflicting operational instructions around:

- npm vs pnpm
- startup flow
- JDK/runtime expectations

RISK-004 — Secrets hygiene problem

`infrastructure/README.md` contains secret-like Terraform environment variable examples that should be reviewed immediately.

RISK-005 — Missing canonical demo-readiness package

The intended documentation package under `docs/demo-readiness/` does not exist yet.

---

14. Summary by Maturity

Exists

- Kotlin/Ktor backend
- React MFE frontend
- Flyway + PostgreSQL
- RLS tenant isolation
- receiving / inventory / orders / picking / shipments
- carrier integrations
- RBAC
- dashboard/backend operational domains
- barcode
- audit/webhooks/SSE
- backend tests
- e2e presence

- pricing/GTM/business framing

Partial

- AI execution depth
- smart picking maturity verification
- mobile/offline maturity
- shipment monitoring UX depth
- reorder alerting maturity
- ops/monitoring completeness
- performance/regression readiness
- API spec accuracy

Planned

- 3PL billing and mature 3PL workflows
- polished demo-readiness package
- full support/onboarding package
- fully canonical feature and QA documentation set

---

15. Working Rule

Future planning and QA work must use this feature baseline together with:

- canonical source-of-truth map
- user stories baseline

Nothing should be marked “implemented” solely because it appears in a stale document or in an agent-generated summary without artifact proof.

Gaps and Next Steps

Source: `~/ALAI/products/Plock/docs/demo-readiness/03-gaps-and-next-steps.md`

PLOCK — Gaps and Next Steps

Date: 2026-03-14
Purpose: Convert the current PLOCK reality into an actionable plan: what is missing, what is conflicting, what is risky, and in which order the remaining work should happen.

---

1. Goal

The original goal was to get PLOCK into a state where we can:

1. understand the real product and architecture,
2. produce deterministic documentation,
3. QA that documentation,
4. then generate implementation tasks by feature area,
5. and move toward demo-readiness / production-readiness in a controlled way.

This document defines what blocks that goal today and what sequence should be followed next.

---

2. Current State Summary

PLOCK already has:

- substantial real code,
- real integrations,
- real tests,
- real infrastructure scaffolding,
- real product and GTM documentation.

However, PLOCK is currently blocked by:

- conflicting architecture documentation,

- missing canonical documentation package,
- weak documentation-to-artifact execution flow,
- and lack of a clean, approved “source-of-truth → QA → backlog” chain.

---

3. Gap Categories

3.1 Canonical Documentation Gaps

GAP-DOC-001 — Canonical `docs/demo-readiness/` package exists, but is still incomplete

Severity: P0

The canonical demo-readiness package now exists, but it is still only a baseline layer.

Already present:

- source-of-truth baseline
- user stories baseline
- feature baseline
- gaps/next-steps baseline
- QA checklist baseline

Still missing or incomplete for the fuller package:

- program map
- deterministic task contract
- doc inventory gap audit
- package-level review expansion
- later demo/sales/support readiness docs
- program map
- deterministic task contract
- doc inventory gap audit
- user stories baseline
- feature catalog
- QA gate checklist

Impact:
Without this package, there is no stable documentation handoff point for QA or implementation backlog generation.

GAP-DOC-002 — No canonical feature catalog

Severity: P0

Features are spread across:

- code structure,
- PRD,
- GTM docs,
- API spec,
- runbooks,
- integration docs.

But there is no single authoritative feature catalog that says:

- what exists,
- what is partial,
- what is planned.

Impact:
Implementation task generation will drift or duplicate work.

GAP-DOC-003 — User stories are present, but not operationalized

Severity: P1

User stories exist mainly in `docs/PRD.md`, but not yet in a clean, implementation-ready canonical user-story document.

Impact:
Traceability between product vision and implementation work is weaker than it should be.

GAP-DOC-004 — Canonical QA checklist exists, but QA execution is still incomplete

Severity: P0

A trusted QA checklist now exists, but the review process is still immature and must continue with explicit review records and package verdicts.

What now exists:

- deterministic QA checklist baseline
- package review workflow

What still needs strengthening:

- continued document-by-document review records as the package grows
- repeatable reviewer discipline for future generated docs
- explicit re-review after any canonical doc changes

Impact:
QA can now start approving or rejecting the documentation layer, but the process still depends on careful manual review.

3.2 Architecture / Source-of-Truth Gaps

GAP-ARCH-001 — Conflicting architecture documents

Severity: P0

`docs/ARCHITECTURE.md` conflicts with the actual repo and stronger docs.

Examples of stale/conflicting claims:

- Next.js instead of MFE shell
- Express instead of Ktor
- Prisma instead of Exposed + Flyway
- `organization_id` instead of `warehouse_id`

Impact:
Any agent or human using the wrong document can generate incorrect output.

GAP-ARCH-002 — API spec drift

Severity: P1

`docs/API-SPEC.md` appears useful, but likely drifted from the actual backend implementation and tenant model.

Impact:
API-facing work may be built or reviewed against incorrect assumptions.

GAP-ARCH-003 — Runbook drift

Severity: P1

Different docs disagree on:

- npm vs pnpm
- startup flow
- JDK/runtime expectations

Impact:
Ops/docs/support/demo execution can become inconsistent and brittle.

3.3 Product / Feature Gaps

GAP-FEAT-001 — AI feature maturity unclear

Severity: P1

AI Chat and Smart Picking are core product promises, but actual implementation depth is not yet cleanly verified against code and runtime behavior.

Impact:
Demo promises may exceed current product maturity.

GAP-FEAT-002 — 3PL capability is more strategy than current product

Severity: P1

3PL/client portal/billing appears strongly in docs and GTM, but current implementation evidence suggests this is more Phase 2 than current MVP.

Impact:
Sales/demo scope can overpromise if not constrained.

GAP-FEAT-003 — Sales/marketing/support artifacts not packaged

Severity: P2

The content exists in fragments:

- GTM strategy
- pricing
- regulatory notes
- runbooks

But not yet as:

- demo script,
- objection handling pack,
- support playbook,
- onboarding checklist,
- sales-facing feature matrix.

Impact:
Cross-functional readiness is incomplete.

3.4 Security / Hygiene Gaps

GAP-SEC-001 — Secret-like material in infra docs

Severity: P0/P1

`infrastructure/README.md` contains environment variable examples that resemble real secrets or secret material.

Impact:
Security hygiene issue. Must be reviewed and cleaned immediately.

GAP-SEC-002 — Secrets/process docs need explicit verification pass

Severity: P1

Secrets strategy exists, but should be validated against:

- actual deployment practice,
- Vaultwarden usage,
- Railway variable setup,
- non-committed local env discipline.

Impact:
Production readiness risk if docs and reality differ.

3.5 Process / Orchestration Gaps

GAP-PROC-001 — Delegated doc output quality remains unreliable

Severity: P0

Even after improving task gating, agents still often:

- narrate output,
- claim files were created,
- fail to produce actual artifacts.

Impact:
Important documentation work cannot yet be trusted blindly.

GAP-PROC-002 — Contradictory orchestration logging still exists

Severity: P1

Observed behavior:

- task blocked by QA gate,
- then orchestrator still logs it as completed.

Impact:
Operational trust and status reporting are still imperfect.

GAP-PROC-003 — Forge worker pool saturation

Severity: P1

Some tasks are delayed or requeued because:

- forge pool is full,
- H-priority tasks compete with unrelated workload.

Impact:
Even correctly defined tasks may not progress quickly.

---

4. What Must Happen Before Feature-by-Feature Implementation Tasks

Before generating backend/frontend/db/integration/security/sales/marketing task waves, we need:

Required Preconditions

- canonical source-of-truth note
- canonical user stories baseline
- canonical feature baseline
- canonical gaps/next-steps note
- canonical QA checklist
- clear exclusion of stale architecture doc from planning
- security hygiene review of infra secret examples

If these are skipped, implementation tasks will be generated against mixed truths.

---

5. Recommended Execution Order

Phase 1 — Documentation Truth Lock

Priority: P0

Deliverables

1. Canonical source-of-truth note
2. User stories baseline
3. Feature baseline
4. Gaps and next steps
5. QA checklist baseline

Outcome:
One trusted planning layer.

Phase 2 — Documentation QA

Priority: P0

QA should validate the documentation set against:

- actual repo structure
- actual code domains
- actual migrations
- actual integrations
- actual security/tenant model

QA must explicitly check:

- no stale architecture assumptions
- no references to nonexistent files
- no contradictory stack descriptions
- no unsupported feature claims
- every feature mapped to status: exists / partial / planned

Outcome:
Trusted doc package.

Phase 3 — Controlled Backlog Generation

Priority: P1

Once the doc package passes QA:

- generate backend task list
- frontend task list
- DB/data task list
- integration task list
- security task list
- sales/marketing/support task list

Rules for backlog generation

Every generated task must include:

- exact output contract
- exact deliverable path or target
- acceptance check
- source references
- explicit feature/user-story linkage

Outcome:
Deterministic implementation backlog.

Phase 4 — Demo Readiness Pass

Priority: P1

Once the controlled backlog exists:

- create demo script
- create UI/demo journey map
- create sales narrative
- create support/onboarding flow
- create go/no-go checklist

Outcome:
Real demo package, not vague “prod-ready” claims.

Phase 5 — Production Readiness Pass

Priority: P2

Only after docs + QA + controlled backlog + demo prep:

- ops readiness verification

- secrets/process cleanup
- observability checks
- deployment validation
- test coverage/reliability review
- support and incident readiness

Outcome:
Production-readiness based on evidence, not optimism.

---

6. Immediate Next Actions

NOW-001 — Freeze stale architecture usage

- Treat `docs/ARCHITECTURE.md` as stale
- Do not use it for planning or QA

NOW-002 — Clean security hygiene issue

- Review and sanitize `infrastructure/README.md`
- Assume secret-like values may need rotation review

NOW-003 — Finalize canonical documentation baseline

- source-of-truth
- user stories
- feature baseline
- gaps/next steps
- QA checklist baseline

NOW-004 — Do not launch broad implementation waves yet

- no broad backend/frontend/db waves
- no mass task fan-out based on stale or mixed docs

NOW-005 — Generate only narrow, deterministic tasks

Until process reliability improves, every task should be:

- narrow,
- artifact-bound,
- source-referenced,
- QA-verifiable.

---

7. Success Condition

We are ready to resume the original plan when all of the following are true:

- authoritative vs stale docs are clearly defined
- canonical user stories exist
- canonical feature baseline exists
- canonical QA checklist exists
- documentation package passes QA
- implementation tasks can be generated from trusted sources only

---

8. Decision

PLOCK should proceed through a **documentation-truth-first** path, not a broad parallel execution wave.

This is not extra process for its own sake.
It is the minimum needed to ensure the next backlog is based on the real product rather than conflicting documentation or fabricated agent output.

QA Checklist Baseline

Source: `~/ALAI/products/Plock/docs/demo-readiness/04-qa-checklist-baseline.md`

PLOCK — QA Checklist Baseline

Date: 2026-03-14
Purpose: Define the deterministic review checklist that must be applied to every PLOCK demo-readiness document before it is treated as usable input for planning, QA, demo prep, or backlog generation.

---

1. Why This Exists

PLOCK documentation work has already shown two failure modes:

1. documents can be generated from conflicting or stale sources, and
2. tasks can claim deliverables were created when the files do not actually exist.

This checklist prevents both.

A document is not considered approved because:

- an agent says it wrote it,
- a task says `done`, or
- the text sounds plausible.

A document is approved only if:

- the file exists,
- the path is correct,
- the content matches the requested deliverable,
- the claims are grounded in authoritative sources,
- contradictions have been checked,
- and the document passes the go/no-go review below.

---

2. Review Scope

Apply this checklist to every document under:

- `~/ALAI/products/Plock/docs/demo-readiness/`

At minimum, this includes:

- `00-canonical-source-of-truth.md`
- `01-user-stories-baseline.md`
- `02-feature-baseline.md`
- `03-gaps-and-next-steps.md`
- `04-qa-checklist-baseline.md`
- any later files added to the same package

---

3. Authority Order

When a demo-readiness document makes a claim, validate it using this precedence:

1. **Actual code and repo structure**
2. **Runtime configuration and DB migrations**
3. `BUILD-BLUEPRINT.md`
4. `README.md`
5. **Runbooks / security / ops docs**
6. **ADRs**
7. **PRD / GTM / Regulatory docs**
8. `docs/API-SPEC.md` **only if verified against code**
9. **Never use stale/conflicting docs as canonical truth**

Explicit stale-doc rule

`docs/ARCHITECTURE.md` is currently treated as **stale/conflicting** and must not be used as a canonical architecture source until rewritten or formally re-approved.

---

4. Required QA Outcome Labels

Each reviewed document must end in one of these states:

- **PASS** — safe to use as canonical planning input

- **PASS WITH NOTES** — usable, but with explicit caveats recorded
- **FAIL** — not safe to use; must be corrected before downstream work
- **BLOCKED** — cannot be reviewed properly because source evidence is missing or inaccessible

No silent approval is allowed.

---

5. Pre-Review Checks

Before reading content, verify:

5.1 File existence

- The file exists on disk.
- The file is readable.
- The filename matches the intended deliverable exactly.

5.2 Path correctness

- The file is located under the expected canonical path.
- If the task specified an exact path, the file is at that exact path.
- A similarly named file elsewhere does **not** count as pass.

5.3 Index discoverability

- `docs/demo-readiness/INDEX.md` links the file if it belongs in the canonical package.
- `docs/INDEX.md` points to the canonical package when appropriate.

5.4 Basic format sanity

- Markdown renders as a coherent document.
- Headings are structured.
- The document is not obviously truncated.
- Placeholder/TBD/filler text is either resolved or clearly marked as unresolved risk.

---

6. Artifact Verification Checklist

A document automatically fails if any of the following are true:

- claimed file does not exist
- wrong filename
- wrong directory
- empty or near-empty content
- content unrelated to requested deliverable
- generic narrative that does not match the task contract
- references to files, modules, or paths that do not exist without explicit “unverified” marking

Required artifact review questions

- Does the file exist exactly where promised?
- Is this the exact deliverable requested?
- Does the document content materially satisfy the task?
- Is it specific to PLOCK, not generic boilerplate?

---

7. Content Evidence Checklist

Every substantive claim should be supported by at least one of:

- repo path
- code module
- migration
- config file
- runbook/security/ADR reference
- product/business doc reference

Minimum evidence standard

For each major section, the reviewer should be able to answer:

- **What source supports this claim?**
- **Is that source authoritative?**
- **Does the claim overstate what the source proves?**

Preferred evidence style

Use explicit references such as:

- `backend/src/main/kotlin/...`
- `backend/src/main/resources/db/migration/...`

- `frontend/mfe-*`
- `docs/PRD.md`
- `docs/RLS-AUDIT.md`
- `README.md`
- `BUILD-BLUEPRINT.md`

---

8. Contradiction Check

The reviewer must actively search for contradictions between the reviewed document and stronger sources.

Mandatory contradiction checks

Verify the document does **not** incorrectly describe PLOCK as:

- Next.js instead of React/Vite MFEs
- Express instead of Ktor
- Prisma instead of Exposed + Flyway
- Supabase/Upstash as primary current stack without proof
- `organization_id` as the current tenant model where repo reality shows `warehouse_id`

Contradiction rule

If a claim conflicts with code or stronger docs, the document fails unless the conflict is explicitly called out as historical/stale context.

---

9. User Story and Feature Traceability Check

Every story or feature-oriented document must support traceability.

For user-story documents

Check that stories are:

- grounded in PRD and/or real code domains
- labeled clearly (`exists`, `partial`, `planned` or equivalent)

- not invented purely from marketing language
- not duplicated under different names without explanation

For feature documents

Check that each major feature maps back to at least one of:

- a route/service/module
- a migration/schema element
- a frontend MFE/module
- a product doc with explicit “planned” labeling

Required traceability rule

A feature that has no code evidence may still appear only if it is clearly labeled **planned** and anchored to an authoritative product/business doc.

---

10. Source-of-Truth Compliance Check

The reviewed document must align with `00-canonical-source-of-truth.md`.

Reviewer must verify:

- repo reality is treated as primary
- `docs/ARCHITECTURE.md` is not treated as canonical truth
- `docs/API-SPEC.md` is used carefully and only with verification caveat if needed
- agent-generated narrative is not cited as authoritative evidence

Automatic fail if the document:

- treats agent output as primary evidence
- uses stale architecture claims as current truth
- ignores stronger repo evidence

---

11. Security Hygiene Check

Every demo-readiness document that touches environments, infra, auth, secrets, integrations, or deployment must be reviewed for security hygiene.

Required checks

- No real secrets or secret-like material are copied into the doc.
- No environment variable values are presented in a way that looks reusable as production material.
- Any reference to `infrastructure/README.md` must respect the known hygiene issue around secret-like examples.
- If secret examples are mentioned, they must be described as hygiene risks, not copied forward as acceptable practice.
- Auth, token, encryption, and tenant-isolation claims must match real security docs/code where possible.

Automatic fail examples

- document repeats secret-looking example values as if acceptable
- document instructs users to commit secrets
- document misstates RLS/tenant isolation model
- document weakens existing security posture in prose without evidence

---

12. Canonical Path and Naming Check

The canonical package should remain predictable.

Reviewer should verify:

- numbering is consistent
- filenames are descriptive and stable
- index references are correct
- no duplicate “canonical” documents exist with competing names

Naming rule

If a document supersedes another document, the relationship must be explicit. Confusing parallel variants should fail review or be flagged for consolidation.

---

13. Acceptance Criteria by Document Type

13.1 Canonical source-of-truth docs

Must:

- identify authoritative vs stale sources
- define precedence clearly
- match repo reality
- explicitly quarantine stale architecture assumptions

13.2 User-story docs

Must:

- identify user roles clearly
- phrase stories in a usable operational way
- mark maturity/status
- avoid unsupported invention
- allow later mapping to features and backlog

13.3 Feature baseline docs

Must:

- group features coherently
- distinguish exists vs partial vs planned
- reference code/docs evidence
- capture key risks/contradictions

13.4 Gap / next-step docs

Must:

- separate current fact from recommendation
- prioritize gaps clearly
- define execution order
- avoid pretending that future work already exists

13.5 QA docs

Must:

- be deterministic
- define pass/fail rules
- include evidence expectations
- include final review workflow and go/no-go outcome

---

14. Reviewer Workflow

Use this exact workflow:

Step 1 — Confirm artifact

- file exists
- path is exact
- file is readable

Step 2 — Confirm task-fit

- compare document against requested deliverable
- reject generic filler or off-target content

Step 3 — Check evidence

- scan each major section for source grounding
- verify stronger-source precedence is respected

Step 4 — Check contradictions

- compare against repo reality and known stale-doc risks
- record any mismatch explicitly

Step 5 — Check traceability

- ensure user stories/features/gaps can be traced to evidence

Step 6 — Check security hygiene

- especially for env, infra, auth, tokens, integrations, deployment

Step 7 — Record verdict

Document one of:

- PASS
- PASS WITH NOTES
- FAIL
- BLOCKED

Step 8 — Record remediation

If not PASS, specify:

- exact issue
- exact file/section
- exact correction needed
- whether downstream work must pause

---

15. Review Record Template

Use this template for every reviewed document:

```
## QA Review Record
- Document:
- Path:
- Reviewer:
- Date:
- Verdict: PASS | PASS WITH NOTES | FAIL | BLOCKED

### Artifact Check
- Exists:
- Correct path:
- Correct filename:
- Indexed:

### Evidence Check
- Primary sources used:
- Any unsupported claims:
```

```
### Contradiction Check
- Conflicts found:
- Stale-doc contamination found:

### Traceability Check
- User-story traceability:
- Feature traceability:

### Security Hygiene Check
- Secret-like material copied: yes/no
- Security posture accurately described: yes/no

### Notes / Remediation
-
```

---

16. Go / No-Go Checklist

A demo-readiness document is **GO** only if all answers below are yes:

- Does the file exist at the exact canonical path?
- Does it match the requested deliverable?
- Is it grounded in authoritative sources?
- Does it avoid stale architecture assumptions?
- Are major claims evidence-backed?
- Are features/stories/gaps traceable?
- Are security-sensitive claims hygienic and accurate?
- Is the document specific enough to be used downstream?

If any answer is **no**, the outcome is **NO-GO**.

---

17. Package-Level Go / No-Go Rule

The full `docs/demo-readiness/` package is not ready for downstream backlog generation until:

- core baseline documents exist,
- each has passed review,
- no unresolved P0 contradiction remains,
- and no document relies on stale architecture as current truth.

Package-level blockers

Any of these are automatic package-level **NO-GO**:

- missing canonical file
- unresolved architecture contradiction
- unverified API assumptions treated as fact
- security hygiene issue copied into canonical docs
- major feature/story claims without evidence

---

18. Decision

This checklist is the minimum deterministic QA gate for PLOCK demo-readiness documentation.

No document in this package should be treated as approved planning input until it passes this checklist with an explicit recorded verdict.

Initial Package QA Review

Source: `~/ALAI/products/Plock/docs/demo-readiness/05-initial-package-review.md`

PLOCK — Initial Package QA Review

Date: 2026-03-14
Reviewer: John
Scope: Manual QA review of the current canonical PLOCK demo-readiness baseline package under `~/ALAI/products/Plock/docs/demo-readiness/`.

---

1. Package Verdict

Document Quality Verdict: PASS WITH NOTES
Downstream Usage Verdict: GO for narrow documentation-truth work; NO-GO for broad implementation fan-out until remaining P0/P1 issues are addressed.

Why

The package now exists on disk, is indexed, and provides a coherent baseline for source-of-truth, user stories, features, gaps, and QA review.

However, it still carries important caveats:

- `docs/ARCHITECTURE.md` remains stale/conflicting in the repo
- `docs/API-SPEC.md` still requires explicit verification against code
- `infrastructure/README.md` still needs security-hygiene cleanup
- the fuller demo-readiness package is not complete yet

---

2. Artifact Check

Reviewed files

- `INDEX.md`
- `00-canonical-source-of-truth.md`
- `01-user-stories-baseline.md`
- `02-feature-baseline.md`
- `03-gaps-and-next-steps.md`
- `04-qa-checklist-baseline.md`

Result

- All reviewed files exist on disk
- All are readable
- All are located under the expected canonical path
- `docs/demo-readiness/INDEX.md` links the core package correctly
- `docs/INDEX.md` points to the canonical package

Artifact verdict: PASS

---

3. Per-Document Review Records

QA Review Record

- Document: Canonical baseline index
- Path: `~/ALAI/products/Plock/docs/demo-readiness/INDEX.md`
- Reviewer: John
- Date: 2026-03-14
- Verdict: PASS

Artifact Check

- Exists: yes
- Correct path: yes
- Correct filename: yes
- Indexed: n/a (entry point document)

Evidence Check

- Primary sources used: package structure itself
- Any unsupported claims: none material

Contradiction Check

- Conflicts found: none
- Stale-doc contamination found: no

Traceability Check

- User-story traceability: n/a
- Feature traceability: n/a

Security Hygiene Check

- Secret-like material copied: no
- Security posture accurately described: yes

Notes / Remediation

- None

QA Review Record

- Document: Canonical source of truth
- Path: `~/ALAI/products/Plock/docs/demo-readiness/00-canonical-source-of-truth.md`
- Reviewer: John
- Date: 2026-03-14
- Verdict: PASS WITH NOTES

Artifact Check

- Exists: yes
- Correct path: yes
- Correct filename: yes
- Indexed: yes

Evidence Check

- Primary sources used: repo structure, build docs, runbooks, RLS/security docs, PRD/GTM/REGULATORY docs
- Any unsupported claims: no major unsupported claims after path correction and gap refresh

Contradiction Check

- Conflicts found: corrected ADR path references from `docs/adr/*` to `adr/*`; refreshed outdated “missing package” statements
- Stale-doc contamination found: no; stale architecture doc is explicitly quarantined

Traceability Check

- User-story traceability: indirect but adequate
- Feature traceability: indirect but adequate

Security Hygiene Check

- Secret-like material copied: no
- Security posture accurately described: yes

Notes / Remediation

- Continue using this as the authority map, but refresh remaining-gap section when new canonical docs are added.

QA Review Record

- Document: User stories baseline
- Path: `~/ALAI/products/Plock/docs/demo-readiness/01-user-stories-baseline.md`
- Reviewer: John
- Date: 2026-03-14
- Verdict: PASS WITH NOTES

Artifact Check

- Exists: yes
- Correct path: yes
- Correct filename: yes

- Indexed: yes

Evidence Check

- Primary sources used: `docs/PRD.md`, backend route/service structure, integrations, runbook/regulatory context
- Any unsupported claims: some maturity labels remain approximate and should be revalidated during runtime/demo verification

Contradiction Check

- Conflicts found: no major architecture contradictions found
- Stale-doc contamination found: no

Traceability Check

- User-story traceability: yes
- Feature traceability: mostly yes; evidence is sufficient for a baseline but not yet exhaustive at section-by-section code depth

Security Hygiene Check

- Secret-like material copied: no
- Security posture accurately described: yes

Notes / Remediation

- Good canonical baseline. Later passes should tighten evidence granularity and confirm AI/mobile maturity against runtime behavior, not just repo structure.

QA Review Record

- Document: Feature baseline
- Path: `~/ALAI/products/Plock/docs/demo-readiness/02-feature-baseline.md`
- Reviewer: John
- Date: 2026-03-14
- Verdict: PASS WITH NOTES

Artifact Check

- Exists: yes
- Correct path: yes
- Correct filename: yes
- Indexed: yes

Evidence Check

- Primary sources used: routes, services, migrations, frontend MFE structure, test folders, ops/security docs
- Any unsupported claims: a few maturity judgments remain intentionally approximate (`partial`, `planned-to-partial`) and should be revisited during deeper verification

Contradiction Check

- Conflicts found: none material
- Stale-doc contamination found: no

Traceability Check

- User-story traceability: strong enough for baseline planning
- Feature traceability: yes, with direct module/file references for most sections

Security Hygiene Check

- Secret-like material copied: no
- Security posture accurately described: yes

Notes / Remediation

- Strong baseline. Future revisions should verify ops/monitoring and AI claims against live behavior before demo commitments.

QA Review Record

- Document: Gaps and next steps
- Path: `~/ALAI/products/Plock/docs/demo-readiness/03-gaps-and-next-steps.md`
- Reviewer: John
- Date: 2026-03-14
- Verdict: PASS WITH NOTES

Artifact Check

- Exists: yes
- Correct path: yes
- Correct filename: yes
- Indexed: yes

Evidence Check

- Primary sources used: package state, repo findings, orchestration incident findings, doc drift findings
- Any unsupported claims: none material after current-state refresh

Contradiction Check

- Conflicts found: refreshed outdated statements that previously said the package and QA checklist did not exist
- Stale-doc contamination found: no

Traceability Check

- User-story traceability: indirect but acceptable
- Feature traceability: indirect but acceptable

Security Hygiene Check

- Secret-like material copied: no
- Security posture accurately described: yes

Notes / Remediation

- Keep this document current as additional canonical docs are created; it is time-sensitive by nature.

QA Review Record

- Document: QA checklist baseline
- Path: `~/ALAI/products/Plock/docs/demo-readiness/04-qa-checklist-baseline.md`
- Reviewer: John

- Date: 2026-03-14
- Verdict: PASS

Artifact Check

- Exists: yes
- Correct path: yes
- Correct filename: yes
- Indexed: yes

Evidence Check

- Primary sources used: actual failure patterns observed in orchestration and documentation work
- Any unsupported claims: none material

Contradiction Check

- Conflicts found: none
- Stale-doc contamination found: no

Traceability Check

- User-story traceability: n/a
- Feature traceability: n/a

Security Hygiene Check

- Secret-like material copied: no
- Security posture accurately described: yes

Notes / Remediation

- This should be applied to all future additions to the package.

---

4. Package-Level Notes

What is now good enough

- canonical source-of-truth layer exists
- user stories and feature baseline exist
- gaps/next-steps are documented
- deterministic QA criteria now exist
- package is discoverable via both filesystem and HiveMind

What still blocks a broad implementation wave

- stale `docs/ARCHITECTURE.md` still exists in the repo and can mislead future agents
- `docs/API-SPEC.md` still needs verification against code
- infra security-hygiene issue still needs cleanup
- fuller demo-readiness docs remain incomplete

---

5. Final Go / No-Go

GO

- narrow documentation-truth work
- narrow source-referenced QA work
- controlled backlog preparation based on these canonical docs
- explicit review of stale/conflicting docs

NO-GO

- broad parallel implementation wave based on legacy docs
- claims of demo-ready or prod-ready status without further verification
- trusting `docs/ARCHITECTURE.md` as current truth
- trusting `docs/API-SPEC.md` as canonical without code verification

---

6. Decision

The current PLOCK canonical baseline package passes manual review as a usable planning foundation, but only with explicit caveats. It is good enough to guide the next narrow, deterministic documentation and QA steps. It is not yet a license for broad implementation fan-out or readiness claims.

API Spec Verification Pass

Source: `~/ALAI/products/Plock/docs/demo-readiness/06-api-spec-verification-pass.md`

PLOCK — API Spec Verification Pass

Date: 2026-03-14
Reviewer: John
Scope: Narrow manual verification of `docs/API-SPEC.md` against the real Ktor route surface under `backend/src/main/kotlin/no/alai/plock/`.

---

1. Verdict

Canonical verdict: FAIL as a canonical API source
Reference verdict: PASS WITH NOTES as API-intent/reference material

Why

`docs/API-SPEC.md` contains useful product intent, but it does not currently match the real backend route surface closely enough to be treated as canonical.

The biggest problems are:

- org-centric model appears where repo reality is warehouse-centric
- several documented endpoint groups do not exist in the current Ktor app
- several real Ktor route groups are missing or materially different in the spec
- multiple path shapes differ even when the domain concept is roughly aligned

---

2. Verification Sources

Primary sources used for this pass:

- `backend/src/main/kotlin/no/alai/plock/plugins/Routing.kt`
- `backend/src/main/kotlin/no/alai/plock/routes/AuthRoutes.kt`
- `backend/src/main/kotlin/no/alai/plock/routes/WarehouseRoutes.kt`
- `backend/src/main/kotlin/no/alai/plock/routes/LocationRoutes.kt`
- `backend/src/main/kotlin/no/alai/plock/routes/ProductRoutes.kt`
- `backend/src/main/kotlin/no/alai/plock/routes/InventoryRoutes.kt`
- `backend/src/main/kotlin/no/alai/plock/routes/OrderRoutes.kt`
- `backend/src/main/kotlin/no/alai/plock/routes/PickingRoutes.kt`
- `backend/src/main/kotlin/no/alai/plock/routes/ReceivingRoutes.kt`
- `backend/src/main/kotlin/no/alai/plock/routes/CarrierRoutes.kt`
- `backend/src/main/kotlin/no/alai/plock/routes/CarrierIntegrationRoutes.kt`
- `backend/src/main/kotlin/no/alai/plock/routes/ShipmentRoutes.kt`
- `backend/src/main/kotlin/no/alai/plock/routes/RoleRoutes.kt`
- `backend/src/main/kotlin/no/alai/plock/routes/UserRoutes.kt`
- `backend/src/main/kotlin/no/alai/plock/routes/DashboardRoutes.kt`
- `backend/src/main/kotlin/no/alai/plock/routes/SseRoutes.kt`
- `backend/src/main/kotlin/no/alai/plock/integrations/fortnox/FortnoxOAuthRoutes.kt`
- `backend/src/main/kotlin/no/alai/plock/integrations/fortnox/FortnoxSyncRoutes.kt`

---

3. Real Route Surface Snapshot

`Routing.kt` mounts these main groups under `/api/v1` (plus unauthenticated Fortnox OAuth + health/auth):

- `/auth` and `/auth/me`
- `/products`
- `/users`
- `/dashboard`
- `/warehouses`
- `/locations`
- `/inventory`
- `/orders`
- `/picking`
- `/zones`
- `/suppliers`
- `/receiving`
- `/carriers`
- `/shipments`
- `/roles`
- `/audit`
- `/stock-movements`
- `/cycle-counts`
- `/returns`

- `/webhooks`
- `/events*`
- `/barcodes`
- `/integrations/fortnox/*`

This is already enough to show that the live API surface is broader in some places and materially different in others than `docs/API-SPEC.md` suggests.

---

4. Sections That Are Roughly Aligned

These areas are directionally aligned, even when path or payload details differ:

4.1 Auth

`docs/API-SPEC.md` documents auth/login concepts, and real routes do include:

- `POST /auth/register`
- `POST /auth/login`
- `GET /auth/me`

4.2 Warehouses / products / users / dashboard

These domains exist in the real backend and in the API spec at a high level.

4.3 Carrier + shipment integrations

The spec correctly signals that carrier and Fortnox integration domains exist, although the actual route design is different.

4.4 Real-time events

The spec includes SSE/event streaming, and the real backend does expose:

- `GET /events/stream`
- `GET /events` (legacy alias)
- `GET /events/connections`
- `POST /events/test`

---

5. Major Drift / Mismatch Areas

5.1 Tenancy model drift

Spec problem: org-centric model
Repo reality: warehouse-centric model

Examples in `docs/API-SPEC.md`:

- `Organizations`
- `organization_name`
- `slug`
- organization object inside auth responses
- `/organizations/me`

Repo reality:

- real tenant model is `warehouse_id`
- auth/user DTOs expose `warehouseId`
- no `organization` route group exists in the Ktor app

Impact: High. This is not a cosmetic difference; it changes the domain model.

5.2 Auth flow drift

Spec says

- `POST /auth/register` creates organization + admin user
- `POST /auth/refresh` exists
- `POST /auth/logout` exists

Actual routes

- `POST /auth/register`
- `POST /auth/login`
- `GET /auth/me`

Mismatch

Real auth routes do **not** currently expose documented refresh/logout endpoints, and register/login response shapes differ from the org-centric API spec.

Verdict: partial alignment only

5.3 Warehouses and locations path drift

Spec says

- `/warehouses/:warehouseId/zones`
- `/warehouses/:warehouseId/bins`
- nested warehouse-scoped bin endpoints

Actual routes

- `/warehouses`
- `/warehouses/{id}`
- `/locations`
- `/locations/{id}`
- `/zones`
- `/zones/{id}`
- `/put-away/suggest`

Mismatch

The real backend models zones and locations as first-class route groups, not nested warehouse bin endpoints as described in the spec.

5.4 Inventory route drift

Spec says

- `GET /inventory`
- `POST /inventory/adjustments`
- `GET /inventory/transactions`
- `POST /inventory/cycle-count`
- `GET /inventory/anomalies`

Actual routes

- `GET /inventory`
- `GET /inventory/balance`
- `GET /inventory/summary`
- `GET /inventory/search`
- `POST /inventory`
- `POST /inventory/recalculate`
- `GET /inventory/{id}`
- `POST /inventory/{id}/adjust`
- `DELETE /inventory/{id}`

Mismatch

The real system exposes different operational endpoints and splits stock adjustment and audit trail differently. Inventory transaction history is represented more directly via `/stock-movements`, not the spec's `/inventory/transactions` route.

5.5 Receiving / inbound drift

Spec says

- `/inbound/purchase-orders`
- `/inbound/purchase-orders/:poId/receive`

Actual routes

- `/receiving`
- `/receiving/{id}`
- `/receiving/{id}/receive-item`
- `/receiving/{id}/complete`
- `/receiving/{id}/cancel`
- `/receiving/{id}/process`
- `/receiving/{id}/discrepancies`
- `/receiving/discrepancies/{discrepancyId}/resolve`

Mismatch

The real backend has a receiving-order workflow, not the purchase-order route structure described in the spec.

5.6 Orders / outbound drift

Spec says

- `/outbound/orders`
- `/outbound/orders/release`
- `/outbound/orders/:orderId/status`

Actual routes

- `/orders`
- `/orders/{id}`
- `/orders/{id}/allocate`
- `/orders/{id}/deallocate`
- `/orders/{id}/ship`

Mismatch

The domain overlaps, but path structure and supported actions differ materially.

5.7 Picking drift

Spec says

- `/picking/routes`
- `/picking/routes/:routeId/confirm-pick`
- `/picking/routes/:routeId/assign`

Actual routes

- `/picking/waves`
- `/picking/waves/{id}`
- `/picking/waves/{id}/start`
- `/picking/waves/{id}/pick-item`
- `/picking/{pickListId}/items/{itemId}/confirm`

Mismatch

The live system is pick-wave/pick-item oriented, not route-assignment oriented in the way the spec describes.

5.8 Carrier / shipment drift

Spec says

- `/carriers/labels/generate`
- `/carriers/labels/:trackingNumber/status`
- `/carriers/services`
- `/carriers/labels/:trackingNumber/void`

Actual routes include

- carrier CRUD under `/carriers`
- shipment CRUD and lifecycle under `/shipments`
- provider-specific endpoints under `/carriers/postnord/*`, `/carriers/dhl/*`, `/carriers/instabee/*`
- generic integration shipment routes under `/shipments` in `CarrierIntegrationRoutes.kt`

Mismatch

The spec compresses real shipping/carrier behavior into a simplified label-oriented interface that does not match the current route surface.

5.9 Dashboard / reports drift

Spec says

- `/dashboard/stats`
- `/reports/inventory-health`
- `/reports/performance`
- `/reports/export`

Actual routes

- `/dashboard/stats`
- `/inventory/low-stock`

Mismatch

Only `dashboard/stats` is clearly aligned. The `/reports/*` route group is not present in the current Ktor app.

5.10 AI route drift

Spec says

- `/ai/chat`
- `/ai/actions/confirm`
- `/ai/chat/history`
- `/ai/pick-route/optimize`
- `/ai/pick-route/job/:jobId`

Actual routes

- no `/ai/*` route group was found in the current backend route surface

Mismatch

This is major product-intent content, not current API reality.

5.11 Integrations drift

Spec says

- `GET /integrations`
- `GET /integrations/fortnox/connect`
- `GET /integrations/fortnox/callback`
- `POST /integrations/fortnox/sync`

- `DELETE /integrations/:provider`
- webhooks under integrations

Actual routes include

- `GET /integrations/fortnox/auth`
- `GET /integrations/fortnox/callback`
- `GET /integrations/fortnox/status`
- `POST /integrations/fortnox/sync/articles`
- `POST /integrations/fortnox/sync/orders`
- `POST /integrations/fortnox/sync/stock`
- `POST /integrations/fortnox/sync/products`
- `POST /integrations/fortnox/confirm-shipment/{orderId}`

Mismatch

The real Fortnox integration surface is more concrete and warehouse-scoped than the generic integration control plane described in the spec.

5.12 Users / RBAC drift

Spec says

- `POST /users/invite`
- `PATCH /users/:userId`
- `GET /users/me`
- mostly flat user-role assumptions

Actual routes include

- `/users`
- `/users/{id}` with `GET`, `PUT`, `DELETE`
- `/roles`
- `/roles/assign`
- `/roles/user/{userId}`
- `/roles/user/{userId}/permissions`
- `GET /auth/me`

Mismatch

RBAC is present, but it is organized differently and more explicitly in the real backend than in the spec.

---

6. Important Real Route Groups Missing from the Spec

These live route groups exist in the backend but are not represented well, or at all, in `docs/API-SPEC.md`:

- `/audit`
- `/stock-movements`
- `/cycle-counts`
- `/returns`
- `/webhooks`
- `/barcodes`
- `/suppliers`
- `/zones`
- `/put-away/suggest`
- `/events/connections`
- `/events/test`
- `/health` and `/health/ready`

---

7. Practical Conclusion

Use `docs/API-SPEC.md` only for:

- product/API intent
- domain discovery hints
- identifying candidate workflows to verify against code

Do **not** use it for:

- canonical route inventory
- API QA signoff
- implementation planning without code check
- tenant-model assumptions

---

8. Recommended Next Narrow Step

Do **not** rewrite the full API spec yet.

Instead, use this sequence:

1. keep the current warning on `docs/API-SPEC.md`
2. use this verification note as the current interpretation layer
3. later perform a targeted API-spec correction pass section-by-section, starting with:
   - tenancy/auth
   - inventory
   - receiving/orders/picking
   - carriers/integrations

---

9. Decision

`docs/API-SPEC.md` is now a **verified non-canonical reference**.

Until corrected, the canonical API truth for PLOCK remains:

- the actual Ktor route files, and
- the canonical demo-readiness baseline package.