Frontend Architecture & Developer Manual

Frontend Architecture & Developer Manual

Plock WMS — AI-Native Warehouse Management System
Last updated: 2026-03-04 | Tech lead: ALAI Engineering

Table of Contents

---

1. Architecture Overview

Monorepo Structure

Plock uses a Turborepo-managed monorepo with npm workspaces. The repository is organized as follows:

```
plock/
├── frontend/
│   ├── shell/              # Host application — layout, routing, auth (port 3000)
│   ├── mfe-inventory/      # Inventory management MFE (port 3001)
│   ├── mfe-orders/         # Order management MFE (port 3002)
│   ├── mfe-picking/        # Picking & scanning MFE (port 3003)
│   ├── mfe-settings/       # Settings & admin MFE (port 3004)
│   └── mfe-ai/             # AI chat & analytics MFE (port 3005)
├── packages/
│   ├── types/              # @plock/types — shared TypeScript interfaces
│   ├── ui/                 # @plock/ui — shared React components
│   └── utils/              # @plock/utils — API client, hooks, auth
├── backend/                # Kotlin/Ktor backend API (port 8080)
├── package.json            # Root workspace config
└── turbo.json              # Turborepo pipeline config
```

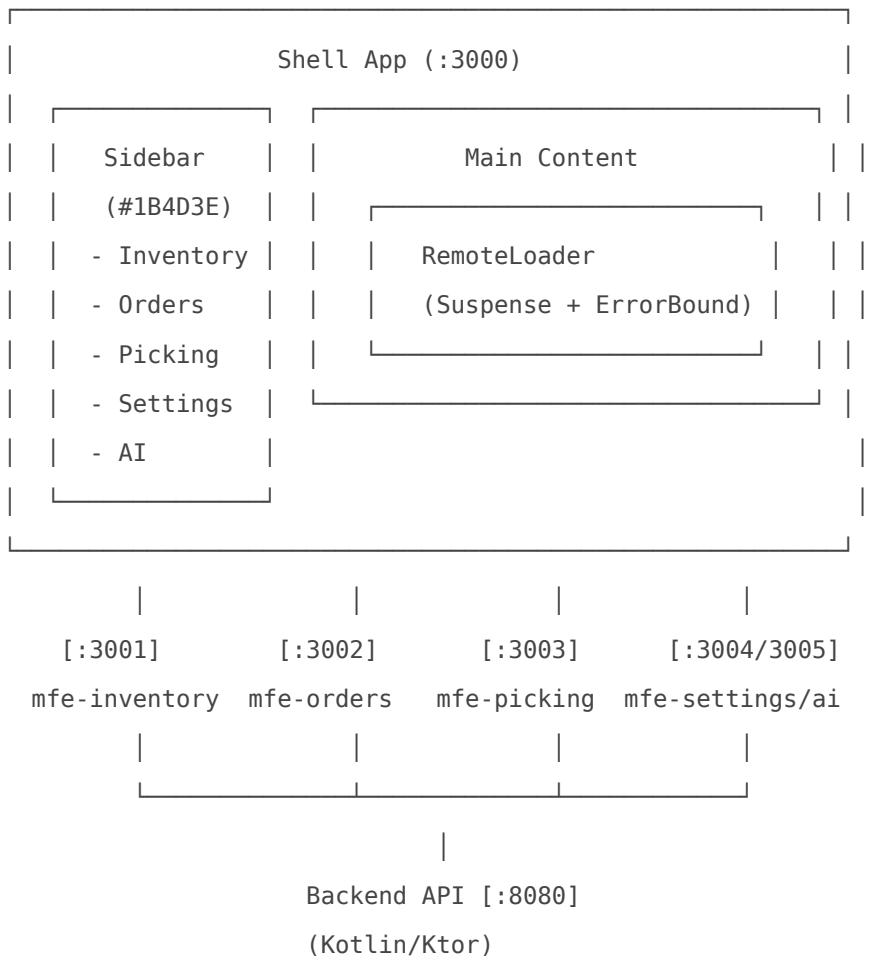
Micro-Frontend Architecture (Module Federation)

Plock implements a micro-frontend (MFE) architecture using **Vite Module Federation** (`@module-federation/vite`). Each MFE is an independently deployable application that exposes specific routes and components to the Shell host.

Application	Port	Role	Exposed
Shell	3000	Host — layout, auth, routing	Consumes all remotes
mfe-inventory	3001	Stock, receiving, locations	`./App`
mfe-orders	3002	Order lifecycle management	`./App`
mfe-picking	3003	Wave picking, barcode scanning	`./App`
mfe-settings	3004	Users, warehouses, integrations	`./App`
mfe-ai	3005	AI chat, analytics, alerts	`./App`

Application	Port	Role	Exposed
backend	8080	Kotlin/Ktor REST API	—

Architecture Diagram



---

2. Tech Stack

Layer	Technology	Version	Purpose
UI Framework	React	19.x	Component rendering
Language	TypeScript	5.x	Type safety throughout
Build Tool	Vite	6.x	Fast dev server + HMR
Styling	Tailwind CSS	4.x	Utility-first CSS
MFE Runtime	@module-federation/vite	latest	Module Federation via Vite
Monorepo	Turborepo + npm workspaces	latest	Build pipeline + caching

Layer	Technology	Version	Purpose
Icons	Lucide React	latest	Icon library
Font	Inter	—	Headings and body text
Mono Font	JetBrains Mono	—	Code blocks
Charts	CSS-only	—	No external chart libraries
Barcode	@zxing/library	latest	Picking MFE scanner
Backend	Kotlin / Ktor	—	REST API on port 8080
DB ORM	Flyway (migrations) + Prisma (introspect)	—	Flyway owns migrations; Prisma read-only

---

3. Shared Packages

3.1 @plock/types — TypeScript Type Contracts

Located at `packages/types/src/index.ts` (242 lines). Provides all shared TypeScript interfaces and enums aligned with Kotlin backend DTOs.

Enums

Enum	Values
`UserRole`	ADMIN, WAREHOUSE_MANAGER, PICKER, RECEIVER, VIEWER
`OrderStatus`	PENDING, ALLOCATED, PICKING, PACKED, SHIPPED, DELIVERED, CANCELLED
`OrderType`	OUTBOUND, INBOUND, TRANSFER, RETURN
`LocationType`	PICK, BULK, RECEIVING, SHIPPING, STAGING
`WaveStatus`	PENDING, IN_PROGRESS, COMPLETED, CANCELLED
`InventoryStatus`	AVAILABLE, RESERVED, QUARANTINE, DAMAGED

Key Interfaces

Interface	Key Fields
`User`	id, email, name, role, warehouseId, createdAt
`Warehouse`	id, name, address, timezone, isActive
`Location`	id, code, type, zoneId, warehouseId, capacity

Interface	Key Fields
`Product`	id, sku, name, description, category, weight, dimensions, barcode
`InventoryItem`	id, productId, locationId, quantity, reservedQuantity, status
`Order`	id, type, status, warehouseId, customerId, lines, createdAt
`OrderLine`	id, orderId, productId, quantity, pickedQuantity, locationId
`PickWave`	id, status, assignedTo, orderIds, items, startedAt, completedAt
`ReceivingOrder`	id, supplierId, warehouseId, lines, status, expectedDate
`DashboardStats`	totalOrders, pendingOrders, activeWaves, lowStockCount, todayShipments

API / Request Types

Type	Description
`PagedResponse<T>`	Generic paginated response: data[], total, page, size
`ApiError`	status, message, details?
`CreateWarehouseRequest`	name, address, timezone
`CreateOrderRequest`	type, warehouseId, customerId, lines
`AdjustStockRequest`	productId, locationId, delta, reason
`LoginRequest`	email, password

3.2 @plock/ui — Shared React Components (14 components)

Located at `packages/ui/src/`. All components are exported from `index.ts`.

Component	Props / Variants
Button	variant: primary | secondary | ghost | danger; size: sm | md | lg; loading: boolean
Card	Base card container with optional title, padding
StatCard	Metric card with value, label, icon, delta
Badge	color: green | amber | red | blue | gray; children
Input	label, error, hint, leftIcon, rightIcon, all native props
DataTable<T>	Generic sortable table: columns[], data[], onRowClick, loading

Component	Props / Variants
Select	Native styled select: options[], label, error
Dialog	Modal overlay with escape key close, focus trap, title, children, actions
Tabs	Horizontal tabs with keyboard navigation: tabs[], activeTab, onChange
Toast / ToastProvider / useToast	type: success | error | warning | info; auto-dismiss; context hook
Dropdown	trigger element + items[]; click-outside-to-close
Skeleton / SkeletonText / SkeletonCard	Loading placeholder components
EmptyState	icon, title, description, action (CTA button)
StatusBadge	Auto-maps status string to Swedish label + semantic color

3.3 @plock/utils — API Client and Hooks

Located at `packages/utils/src/`. Five modules:

api.ts — HTTP Client

```
import { api } from '@plock/utils';

// All methods attach JWT from localStorage automatically
await api.get('/api/v1/inventory');
await api.post('/api/v1/orders', createOrderPayload);
await api.put('/api/v1/orders/123', updatePayload);
await api.delete('/api/v1/orders/123');

// Base URL: http://localhost:8080
```

auth utilities

```
import { getToken, setToken, clearToken } from '@plock/utils';

setToken('eyJhbGciOiJIUzI1NiIs...');   // Persists to localStorage
getToken();                             // Returns string | null
clearToken();                           // Removes from localStorage
```

useAuth hook (AuthProvider)

```
// Wrap app in AuthProvider
<AuthProvider>
  <App />
</AuthProvider>

// Use inside any component
const { user, login, logout, register, isAuthenticated, isLoading } = useAuth();

// login() calls POST /auth/login, stores JWT, fetches /api/v1/auth/me
// On mount: validates stored token via GET /api/v1/auth/me
```

useFetch hook

```
// Generic data fetching with loading/error states
const { data, loading, error, refetch } = useFetch<InventoryItem[]>('/api/v1/inventory');
```

useSSE hook

```
// Server-Sent Events with auto-reconnect
const { data, connected } = useSSE<DashboardEvent>('/api/v1/events/stream');
```

---

4. Shell App (port 3000)

The Shell is the host application. It owns the layout, authentication, routing, and dynamically loads the MFEs via Module Federation.

Layout

- **Sidebar**: Collapsible left navigation, background color `#1B4D3E` (brand green). Links to all 5 MFE domains.
- **Header**: Warehouse selector dropdown, global search, notifications bell, user profile menu.

Authentication

- **ProtectedRoute**: HOC wrapping all authenticated pages. Redirects to `/login` if no valid JWT.
- Token validation: calls `GET /api/v1/auth/me` on boot via `useAuth`.
- Login page: `/login` — email/password form, calls `POST /auth/login`.

Pages

Page	Route	Description
LoginPage	/login	Email + password auth form
Dashboard	/	Live KPI stats, activity feed, order breakdown chart, low-stock alerts

MFE Loading

```
// RemoteLoader component — error boundary + Suspense
<RemoteLoader remote="inventory" fallback={<SkeletonCard />}>
  <InventoryApp />
</RemoteLoader>
```

Routes

Path	Handler
/	Dashboard (Shell)
/login	LoginPage (Shell)
/inventory/*	mfe-inventory (remote)
/orders/*	mfe-orders (remote)
/picking/*	mfe-picking (remote)
/settings/*	mfe-settings (remote)
/ai/*	mfe-ai (remote)

---

5. Inventory MFE (port 3001)

Tab Navigation

- Lagersaldo (Stock Balance)
- Inleverans (Receiving)
- Platser (Locations)
- Cykelräkning (Cycle Count)

Pages

Page	Route	Description
InventoryList	/inventory	Full stock levels with product search and status filter
StockAdjust	/inventory/adjust	Manual stock adjustment form (delta + reason)
ReceivingList	/inventory/receiving	All inbound purchase orders
ReceivingDetail	/inventory/receiving/:id	Receive goods per-line with quantity confirmation
LocationBrowser	/inventory/locations	Zone hierarchy tree view of warehouse locations
CycleCountPage	/inventory/cycle-counts	List of cycle count tasks
CycleCountCreate	/inventory/cycle-counts/new	Create new cycle count task
CycleCountDetail	/inventory/cycle-counts/:id	Execute cycle count, record variances

API Endpoints Used

- `GET/POST /api/v1/inventory`
- `POST /api/v1/inventory/adjust`
- `GET/POST /api/v1/receiving`
- `POST /api/v1/receiving/:id/receive-item`
- `GET /api/v1/locations`
- `GET/POST /api/v1/cycle-counts`
- `POST /api/v1/cycle-counts/:id/complete`

---

6. Orders MFE (port 3002)

Order Status Lifecycle

```
PENDING → ALLOCATED → PICKING → PACKED → SHIPPED → DELIVERED
                                                 ↓ (or)
                                              CANCELLED
```

Pages

Page	Route	Description
OrderList	/orders	Filterable, sortable DataTable of all orders
OrderDetail	/orders/:id	Order lines, status timeline, action buttons (allocate, ship, cancel)
OrderCreate	/orders/new	Multi-line order form: type, customer, product lines with quantities

API Endpoints Used

- `GET /api/v1/orders` — list with filters (status, type, warehouseId, page)
- `POST /api/v1/orders` — create order
- `GET /api/v1/orders/:id` — order detail
- `PUT /api/v1/orders/:id` — update order
- `POST /api/v1/orders/:id/allocate` — allocate inventory
- `POST /api/v1/orders/:id/ship` — mark as shipped

---

7. Picking MFE (port 3003)

Pages

Page	Route	Description
WaveList	/picking	All pick waves with status badges and progress indicators
WaveDetail	/picking/:waveId	Wave items, per-item progress, start wave action
PickingExecution	/picking/:waveId/execute	Mobile-optimized step-by-step picking flow with barcode scanner
ScannerPage	/picking/scan	Dedicated barcode/QR code scanning view
PickConfirmation	/picking/:waveId/confirm	End-of-wave confirmation and summary

Barcode Scanner

Uses `@zxing/library` for in-browser barcode scanning. Supported formats:

- CODE_128 (product barcodes)

- EAN_13 (retail barcodes)
- QR_CODE (location codes)

API Endpoints Used

- `GET /api/v1/picking/waves` — list waves
- `GET /api/v1/picking/waves/:id` — wave detail
- `POST /api/v1/picking/waves/:id/start` — start wave
- `POST /api/v1/picking/waves/:id/pick-item` — confirm item picked

---

8. Settings MFE (port 3004)

Tab Navigation

- Användare (Users)
- Lager (Warehouses)
- Transportörer (Carriers)
- Integrationer (Integrations)
- Profil (Profile)

Pages

Page	Route	Description
UserManagement	/settings/users	Full CRUD for users; role assignment (UserRole enum)
WarehouseSettings	/settings/warehouse	Warehouse details, timezone, active/inactive toggle
CarrierSettings	/settings/carriers	PostNord, DHL, Instabee — activate/deactivate, API keys
IntegrationSettings	/settings/integrations	Fortnox OAuth flow, sync status, manual sync trigger
ProfilePage	/settings/profile	Current user — name, email, password change

API Endpoints Used

- `GET/POST/PUT/DELETE /api/v1/users`
- `GET/PUT /api/v1/warehouses`
- `GET/POST /api/v1/carriers`

- `POST /api/v1/carriers/:id/activate`
- `POST /api/v1/carriers/:id/deactivate`
- `GET /integrations/fortnox/connect` — OAuth redirect
- `GET /integrations/fortnox/callback` — OAuth callback
- `POST /integrations/fortnox/sync` — manual sync trigger

---

9. AI MFE (port 3005)

Tab Navigation

- AI-chatt (AI Chat)
- Analys (Analytics)
- Varningar (Alerts)

Features

AI Chat

- Streaming responses via Server-Sent Events (`useSSE` hook)
- Language toggle: Swedish / English
- Example prompt chips to guide users
- Chat history within session

Analytics

- CSS-only charts (no Recharts, no Chart.js) for zero bundle overhead
- Metrics: inventory levels over time, order throughput, picking performance

Alerts

- Severity filter: INFO, WARNING, CRITICAL
- Per-alert actions: Acknowledge / Dismiss

API Endpoints (Planned)

- `POST /api/v1/ai/chat` — streaming chat endpoint (SSE)
- `GET /api/v1/analytics/inventory` — inventory trend data
- `GET /api/v1/analytics/orders` — order throughput data
- `GET /api/v1/analytics/picking` — picking performance metrics

---

10. Backend API Reference

Kotlin/Ktor backend running on **port 8080**. 24 route files, JWT authentication required on all `/api/v1/*` endpoints.

Authentication

Method	Path	Description
POST	/auth/login	Login — returns JWT token
POST	/auth/register	Register new user
GET	/api/v1/auth/me	Validate token, return current user

Warehouses

Method	Path	Description
GET	/api/v1/warehouses	List all warehouses
POST	/api/v1/warehouses	Create warehouse
GET	/api/v1/warehouses/:id	Get warehouse by ID
PUT	/api/v1/warehouses/:id	Update warehouse
DELETE	/api/v1/warehouses/:id	Delete warehouse

Products

Method	Path	Description
GET	/api/v1/products	List products (query: search, category, warehouseId, page, size)
POST	/api/v1/products	Create product
GET	/api/v1/products/:id	Get product
PUT	/api/v1/products/:id	Update product
DELETE	/api/v1/products/:id	Delete product

Inventory

Method	Path	Description

GET	/api/v1/inventory	List inventory items
POST	/api/v1/inventory	Create inventory record
GET	/api/v1/inventory/:id	Get inventory item
PUT	/api/v1/inventory/:id	Update inventory item
POST	/api/v1/inventory/adjust	Manual stock adjustment (AdjustStockRequest)
GET	/api/v1/inventory/search	Search inventory by product, location, status
GET	/api/v1/inventory/low-stock	Items below reorder threshold

Orders

Method	Path	Description
GET	/api/v1/orders	List orders (filter: status, type, warehouseId)
POST	/api/v1/orders	Create order
GET	/api/v1/orders/:id	Get order with lines
PUT	/api/v1/orders/:id	Update order
DELETE	/api/v1/orders/:id	Delete / cancel order
POST	/api/v1/orders/:id/allocate	Allocate inventory to order
POST	/api/v1/orders/:id/ship	Mark order as shipped

Picking

Method	Path	Description
GET	/api/v1/picking/waves	List pick waves
POST	/api/v1/picking/waves	Create pick wave
GET	/api/v1/picking/waves/:id	Get wave detail with items
POST	/api/v1/picking/waves/:id/start	Start wave (assigns to current user)
POST	/api/v1/picking/waves/:id/pick-item	Confirm item picked at location

Receiving

Method	Path	Description

GET/POST	/api/v1/receiving	List / create receiving orders (POs)
GET/PUT/DELETE	/api/v1/receiving/:id	Get / update / delete receiving order
POST	/api/v1/receiving/:id/receive-item	Record receipt of a specific line item
POST	/api/v1/receiving/:id/complete	Mark receiving order as complete

Locations & Zones

Method	Path	Description
GET/POST	/api/v1/locations	List / create locations
GET/PUT/DELETE	/api/v1/locations/:id	CRUD single location
GET	/api/v1/locations/put-away/suggest	AI-suggested put-away location for a product
GET/POST	/api/v1/zones	List / create zones
GET/PUT/DELETE	/api/v1/zones/:id	CRUD single zone

Suppliers

Method	Path	Description
GET/POST	/api/v1/suppliers	List / create suppliers
GET/PUT/DELETE	/api/v1/suppliers/:id	CRUD single supplier

Carriers

Method	Path	Description
GET/POST	/api/v1/carriers	List / create carriers
GET/PUT/DELETE	/api/v1/carriers/:id	CRUD single carrier
POST	/api/v1/carriers/:id/activate	Activate carrier
POST	/api/v1/carriers/:id/deactivate	Deactivate carrier
GET/POST	/api/v1/carriers/postnord/*	PostNord carrier integration endpoints
GET/POST	/api/v1/carriers/dhl/*	DHL carrier integration endpoints
GET/POST	/api/v1/carriers/instabee/*	Instabee carrier integration endpoints

Shipments

Method	Path	Description
GET/POST	/api/v1/shipments	List / create shipments
GET/PUT/DELETE	/api/v1/shipments/:id	CRUD single shipment
POST	/api/v1/shipments/:id/{transition}	State machine transition (book, dispatch, deliver, etc.)

Roles & Users

Method	Path	Description
GET/POST	/api/v1/roles	List / create roles
POST	/api/v1/roles/:id/assign	Assign role to user
POST	/api/v1/roles/:id/remove	Remove role from user
GET/POST	/api/v1/users	List / create users
GET/PUT/DELETE	/api/v1/users/:id	CRUD single user

Cycle Counts

Method	Path	Description
GET/POST	/api/v1/cycle-counts	List / create cycle count tasks
GET/PUT/DELETE	/api/v1/cycle-counts/:id	CRUD single cycle count
POST	/api/v1/cycle-counts/:id/start	Start cycle count
POST	/api/v1/cycle-counts/:id/complete	Complete and calculate variance
GET	/api/v1/cycle-counts/:id/variance	Get variance report for completed count

Returns

Method	Path	Description
GET/POST	/api/v1/returns	List / create return orders (RMAs)
POST	/api/v1/returns/:id/authorize	Authorize return
POST	/api/v1/returns/:id/receive	Receive returned items
POST	/api/v1/returns/:id/inspect	Inspect and grade returned items
POST	/api/v1/returns/:id/complete	Complete return processing

Webhooks

Method	Path	Description
GET/POST	/api/v1/webhooks	List / register webhooks
GET/PUT/DELETE	/api/v1/webhooks/:id	CRUD single webhook
POST	/api/v1/webhooks/:id/test	Send test event to webhook URL
POST	/api/v1/webhooks/retry-failed	Retry all failed webhook deliveries
GET	/api/v1/webhooks/:id/deliveries	Delivery history with status codes

SSE Events

Method	Path	Description
GET	/api/v1/events/stream	Server-Sent Events stream — real-time warehouse events

Barcodes

Method	Path	Description
POST	/api/v1/barcodes/generate	Generate barcode image for arbitrary data
GET	/api/v1/barcodes/product/:id	Generate barcode for a product
GET	/api/v1/barcodes/location/:id	Generate barcode for a location
GET	/api/v1/barcodes/lookup/:barcode	Resolve barcode to product or location

Audit & Stock Movements

Method	Path	Description
GET	/api/v1/audit	Audit log (query: warehouseId, entityType, entityId, userId, from, to)
GET	/api/v1/stock-movements	All stock movement events (IN/OUT/ADJUST/TRANSFER)

Dashboard

Method	Path	Description

GET	/api/v1/dashboard/stats	KPI stats: orders, waves, shipments, low-stock count

Fortnox Integration

Method	Path	Description
GET	/integrations/fortnox/connect	OAuth2 initiation — redirects to Fortnox
GET	/integrations/fortnox/callback	OAuth2 callback — stores tokens
POST	/integrations/fortnox/sync	Manual sync trigger (products, customers, orders)
GET	/integrations/fortnox/status	Connection status and last sync time

---

11. Brand & Design Tokens

All design tokens are defined as CSS custom properties in `frontend/shell/src/styles/globals.css` and imported by each MFE.

Color Palette

Token	Value	Usage
`--color-primary`	#1B4D3E	Sidebar, primary buttons, headings
`--color-secondary`	#2E7D5B	Hover states, secondary actions
`--color-accent`	#F5A623	Alerts, highlights, badges
`--color-surface`	#F7FAF8	Page backgrounds
`--color-text`	#1A2B23	Primary text
`--color-border`	#e4ebe7	Card borders, dividers
`--color-muted`	#8aaa97	Secondary text, placeholder

Typography

Context	Font
Headings & body	Inter (Google Fonts)
Code blocks, monospace	JetBrains Mono

Spacing & Grid

- Base unit: **8px** (Tailwind's default scale)
- Icons: **Lucide React** — consistent stroke-based icon set
- Border radius: rounded-md (6px) for cards, rounded-lg (8px) for modals

---

12. Development Commands

Setup

```
# Install all workspace dependencies (run from repo root)
npm install
```

Development Servers

```
# Run the Shell app on http://localhost:3000
npm run dev:shell

# Run the Inventory MFE on http://localhost:3001
npm run dev:inventory

# Run Orders MFE on http://localhost:3002
npm run dev --workspace=frontend/mfe-orders

# Run Picking MFE on http://localhost:3003
npm run dev --workspace=frontend/mfe-picking

# Run Settings MFE on http://localhost:3004
npm run dev --workspace=frontend/mfe-settings

# Run AI MFE on http://localhost:3005
npm run dev --workspace=frontend/mfe-ai

# Run all (Shell + Inventory concurrently)
npm run dev:all
```

Type Checking

```
# Type-check ALL workspaces simultaneously via Turborepo
npx turbo run type-check
```

Build

```
# Build all workspaces (production)
npx turbo run build

# Or via root script
npm run build
```

Backend

```
# Start Kotlin/Ktor backend on http://localhost:8080
cd backend && ./gradlew run

# Compile check only (no run)
cd backend && ./gradlew compileKotlin

# Run database migrations (Flyway)
cd backend && ./gradlew flywayMigrate
```

Linting & Formatting

```
# Lint all workspaces
npm run lint

# Format all files with Prettier
npm run format

# Check formatting without writing
npm run format:check
```

Testing

```
# Run all tests
npm run test

# Run with coverage
npm run test:coverage
```

---

13. File Counts & Project Size

Layer	Count	Notes
Frontend (TypeScript/TSX)	74 files	Shell + 5 MFEs
Backend (Kotlin)	74 files	24 route files, 21 services, models, plugins
@plock/types	1 file, 242 lines	All shared TypeScript types
@plock/ui	14 components + index	Shared React component library
@plock/utils	5 modules	api.ts, auth utils, useAuth, useFetch, useSSE

Backend Route File Summary

Domain	Route File
Auth	authRoutes.kt
Warehouses	warehouseRoutes.kt
Products	productRoutes.kt
Inventory	inventoryRoutes.kt
Orders	orderRoutes.kt
Picking	pickingRoutes.kt
Receiving	receivingRoutes.kt
Locations	locationRoutes.kt
Zones	zoneRoutes.kt
Suppliers	supplierRoutes.kt
Carriers	carrierRoutes.kt
Shipments	shipmentRoutes.kt
Roles	roleRoutes.kt

Domain	Route File
Users	userRoutes.kt
Audit	auditRoutes.kt
Stock Movements	stockMovementRoutes.kt
Cycle Counts	cycleCountRoutes.kt
Returns	returnRoutes.kt
Webhooks	webhookRoutes.kt
SSE	sseRoutes.kt
Barcodes	barcodeRoutes.kt
Dashboard	dashboardRoutes.kt
Fortnox Integration	fortnoxRoutes.kt
PostNord/DHL/Instabee	carrierIntegrationRoutes.kt

Document maintained by ALAI Engineering. Last generated: 2026-03-04.

Revision #3
Created 2026-03-04 22:51:56 UTC by John
Updated 2026-05-31 20:04:57 UTC by John