Developer Onboarding Guide: Drop — Fintech Payment App

Developer Onboarding Guide: Drop — Fintech Payment App

> **Project:** Drop — Remittance + QR Payments **Version:** 1.0 **Date:** 2026-02-23
> **Author:** John (AI Director) **Status:** Approved **Reviewers:** Alem Bašić (CEO)

Document History

Version	Date	Author	Changes
0.1	2026-02-23	John	Initial onboarding guide — AI-native team (Builder + Validator agents)

Welcome

Welcome to **Drop** — remittance + QR payments for everyone in Scandinavia.

Drop is built by an AI-native team at ALAI Holding AS. Our team structure is unusual: John (AI Director, Claude Opus) coordinates Builder agents (Claude Sonnet) and Validator agents (Claude Sonnet, read-only). Alem Bašić (CEO) provides direction and final approvals.

If you are a new Builder or Validator agent joining a Drop task, this guide will get you oriented quickly. If anything in this guide is unclear or out of date, please update it as you go.

Your first session at a glance:

- Step 1: Read this guide + `CLAUDE.md` in the project root

- Step 2: Set up local environment (Section 2) + verify tests pass
- Step 3: Read the architecture overview (Section 4)
- Step 4: Pick up your assigned Mission Control task

---

1. Prerequisites

Hardware Requirements

Component	Minimum	Recommended
CPU	4 cores	8+ cores (bcrypt is CPU-intensive in tests)
RAM	8GB	16GB
Disk	10GB free	SSD with 20GB free
OS	macOS (primary), Linux, Windows (WSL2)	macOS

Accounts You Need

Account	Why	How to Get Access
GitHub (alai-org)	Code repository	Ask John (AI Director)
Fly.io	Staging deployment (drop-app)	Ask John (AI Director)
Vaultwarden (`vault.basicconsulting.no`)	Secrets / credentials	Ask John (AI Director)
alai-talk.slack.com	Team communication	Ask Alem Bašić (CEO)
Mission Control (`node ~/system/tools/mc.js`)	Task management	Local system tool

Expected setup time for all accounts: 1 session (AI agents are provisioned per-task)

---

2. Development Environment Setup

2.1 macOS Setup

```
# Install Homebrew (if not already installed)
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"

# Install Node.js via nvm
brew install nvm
echo 'export NVM_DIR="$HOME/.nvm"' >> ~/.zshrc
echo '[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"' >> ~/.zshrc
source ~/.zshrc

# Install Node 20 (Drop uses Next.js 16 which requires Node 20+)
nvm install 20
nvm use 20
nvm alias default 20
```

2.2 Linux Setup

```
# Install nvm
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash
source ~/.bashrc

# Install Node 20
nvm install 20
nvm use 20
```

2.3 Required Software & Versions

Software	Required Version	Install
Node.js	`20.x`	`nvm install 20`
npm	`10.x` (included with Node 20)	Included with Node
flyctl	Latest	`brew install flyctl` (for staging access)
Playwright browsers	Latest	`npx playwright install`

Check your versions:

```
node --version    # Should be v20.x.x
npm --version     # Should be 10.x.x
```

2.4 IDE Setup

Recommended IDE: VS Code

Required extensions:

Extension	Purpose
`dbaeumer.vscode-eslint`	ESLint inline linting
`esbenp.prettier-vscode`	Format on save
`bradlc.vscode-tailwindcss`	Tailwind CSS IntelliSense
`ms-playwright.playwright`	Playwright test runner
`vitest.explorer`	Vitest test explorer

2.5 Environment Variables

```
# Copy the example env file
cp src/drop-app/.env.example src/drop-app/.env.local

# Fill in your local values
# Retrieve from Vaultwarden: vault.basicconsulting.no
# CLI: bw get item "Drop Dev .env" --session $(cat /tmp/bw-session)
```

Key variables:

Variable	Value for Local Dev	Notes
`DATABASE_URL`	`./local.db` (SQLite)	Auto-created on first migration
`JWT_SECRET`	Any random 32+ char string	NEVER use in production
`NEXT_PUBLIC_SERVICE_MODE`	`mock`	Always `mock` for local dev
`BCRYPT_ROUNDS`	`10`	Use 10 locally (faster); production uses 12
`SUMSUB_API_KEY`	`mock`	Mocked in dev
`BAAS_API_KEY`	`mock`	Mocked in dev

Never commit `.env.local`: It's in `.gitignore`.

---

3. Repository Setup

```
# Step 1: Navigate to Drop app directory
cd ~/ALAI/products/Drop/src/drop-app

# Step 2: Install dependencies
npm install

# Step 3: Set up the database
npm run db:migrate

# Step 4: Seed development data
npm run db:seed

# Step 5: Configure environment (see 2.5)
cp .env.example .env.local
# Edit .env.local with your values

# Step 6: Start the development server
npm run dev

# Step 7: Open the application
open http://localhost:3000
# Expected: Drop landing page or login screen
```

---

4. Architecture Overview

High-level architecture: Drop is a PSD2 pass-through fintech app — it NEVER holds customer money.

CRITICAL — Read Before Writing Any Code:

> Drop uses an AISP/PISP pass-through model (ADR-003). The `users` table MUST NOT have a `balance` column. The `cards` table MUST NOT have `card_number` or `cvv` columns. These are tested in `db.test.ts` on every commit. Violating this is a P0 incident.

In brief:

- **Frontend:** Next.js 16 (App Router), React 19, TypeScript, Tailwind CSS — in `src/drop-app/`
- **Backend:** Next.js API Routes (26 routes) — in `src/drop-app/src/app/api/`
- **Database:** SQLite (dev) → PostgreSQL (Phase 1 production) — schema in `src/drop-app/db/`
- **Key external services:** Mock BaaS (AISP + PISP), Mock Sumsub KYC, exchange rates (static seed)
- **Deployment:** Fly.io (Stockholm region) for backend; Vercel for landing page
- **Auth:** JWT in httpOnly cookie, SameSite=Strict, 7-day expiry; bcrypt 12 rounds

Codebase tour (key directories):

Directory	Purpose
`src/drop-app/src/app/api/`	26 API routes (auth, transactions, rates, merchants, cards, health)
`src/drop-app/src/app/(app)/`	App pages (dashboard, send money, QR scan, history, profile)
`src/drop-app/src/components/`	Shared UI components (drop-logo.tsx, etc.)
`src/drop-app/db/`	SQLite schema, migrations, seed scripts
`src/drop-app/__tests__/`	14 test files (unit, integration, performance, E2E)
`mockups/figma-make-export/`	UI source of truth (10 screens — ALWAYS check here before UI changes)

---

5. Key Documentation Links

Document	Link	Purpose
Project overview	`CLAUDE.md`	Pass-through model rules, branding, features
API Reference	`docs/backend/API-REFERENCE.md`	All 26 API endpoints
Database Schema	`docs/backend/DATABASE-SCHEMA.md`	Table definitions, compliance rules
Architecture	`project/architecture/drop-architecture.md`	System design, ADRs
Testing Guide	`docs/testing/TESTING-GUIDE.md`	How to run Vitest + Playwright
Test Inventory	`docs/testing/TEST-INVENTORY.md`	All 14 test files documented
Coding Standards	coding-standards.md	How we write code
Definition of Done	`docs/templates-testing/definition-of-done.md`	What "done" means
Security Audit	`project/docs/security-qa-audit.md`	Known issues + status

Document	Link	Purpose
Roadmap	`ROADMAP.md`	Phases 0/0.5/1/2/3 + blockers

---

6. Coding Standards Reference

Standards document: coding-standards.md

Quick reference:

- **Language:** TypeScript — strict mode, no `any`
- **Commit messages:** Conventional Commits format (`feat:`, `fix:`, `test:`, etc.)
- **Branch naming:** `feature/MC-{task-id}-short-description`
- **PR size:** Aim for < 300 lines changed per PR
- **Testing:** Every PR must include tests for new code
- **SQL:** Parameterized queries ONLY — never string concatenation
- **Passwords:** bcrypt ONLY — SHA-256 is explicitly rejected in `auth.test.ts`

---

7. Git Workflow & Branching Strategy

```
# Create a feature branch (aligned to Mission Control task)
git checkout -b feature/MC-{task-id}-description

# Make changes, then commit using Conventional Commits
git add src/drop-app/...
git commit -m "feat(auth): add bcrypt rounds configuration"

# Push your branch
git push origin feature/MC-{task-id}-description

# Open PR to main branch
```

PR requirements:

1. Title follows Conventional Commits: `type(scope): description`
2. All Vitest tests passing (`npm run test`)
3. TypeScript compiles (`npm run type-check`)
4. ESLint passes (`npm run lint`)
5. Validator agent approved (read-only review)

8. First Task Checklist

- ☐ Development environment running locally (`npm run dev` at `http://localhost:3000`)
- ☐ Test suite passing: `npm run test` (40+ Vitest tests all green)
- ☐ E2E tests passing: `npx playwright test` (3 projects)
- ☐ Read `CLAUDE.md` in `~/ALAI/products/Drop/` — understand pass-through model
- ☐ Read `docs/testing/TESTING-GUIDE.md` — understand test structure
- ☐ Mission Control task assigned and marked `in_progress`
- ☐ First change includes appropriate tests
- ☐ Definition of Done checklist completed before marking task done

---

9. Team Rituals & Communication

Drop uses an AI-native async workflow:

Communication Type	Channel	Purpose
Task coordination	Mission Control (`node ~/system/tools/mc.js`)	All task assignments, status
Technical decisions	`comms/decisions/` in the repo	Architecture decisions, ADRs
CEO updates	MCP email (John → alem@alai.no)	Business milestones, blockers
Engineering	alai-talk.slack.com #drop	Engineering discussions
Security issues	alai-talk.slack.com #drop-security	Immediate escalation

No daily standups — async by default. John (AI Director) coordinates via Mission Control.

---

10. Key Contacts

Role	Name	Contact	Help With
AI Director / Tech Lead	John (Claude Opus)	Mission Control DM	Architecture, task assignment, code review
QA Lead	Validator Agent	Task coordination	Test review, compliance checks

Role	Name	Contact	Help With
CEO / Product Owner	Alem Bašić	alem@alai.no	Business decisions, UAT sign-off

Related Documents

- Local Development Setup
- Coding Standards
- Developer Offboarding Guide
- Definition of Done

Approval

Role	Name	Date	Signature
Author	John (AI Director)	2026-02-23	Approved (AI)
Tech Lead	John	2026-02-23	Approved
CEO (Alem)	Alem Bašić	TBD	

Revision #5
Created 2026-02-23 12:06:29 UTC by John
Updated 2026-05-31 20:03:33 UTC by John