Test Strategy: Drop — Fintech Payment App

Test Strategy: 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 strategy — aligned to Drop tech stack (Vitest + Playwright)

---

1. Testing Philosophy & Principles

Core Principles:

1. **Tests are first-class code** — reviewed, maintained, refactored like production code
2. **Test the behavior, not the implementation** — tests enable safe refactoring
3. **Fast feedback** — unit/integration tests run in seconds; blocking tests run in < 3 minutes
4. **Tests document intent** — a failing test explains what broke and why it matters
5. **Shift left** — find bugs as early as possible, before they reach users
6. **Security-by-testing** — all security invariants (no balance column, no CVV, bcrypt-only) are encoded as automated tests

Testing philosophy: Drop follows the test pyramid with a fintech-specific emphasis: critical financial invariants (no stored balances, PCI-DSS compliance, pass-through model) are verified in

the unit/integration layer on every commit. E2E tests cover the 5 critical user journeys (registration, login, remittance, QR payment, merchant registration). We do not aim for 100% E2E coverage — instead we validate the most valuable and highest-risk paths.

---

2. Test Pyramid

```
graph TB
    subgraph E2E["E2E Tests — Playwright (3 projects)"]
        E2E_DESC["5 critical user journeys<br/>user-flows + full-flows + input-chaos<br/>Slow, expensive, high-confidence"]
    end
    subgraph INT["Integration Tests — Vitest (api-endpoints, db, middleware)"]
        INT_DESC["API request/response contracts<br/>DB schema integrity + FK constraints<br/>Rate limiter + auth middleware"]
    end
    subgraph UNIT["Unit Tests — Vitest (auth, validation, transactions)"]
        UNIT_DESC["bcrypt hashing, JWT verification<br/>Fee calculations, input validation<br/>Fast, cheap, developer-owned"]
    end
    subgraph PERF["Performance Tests — Vitest benchmarks"]
        PERF_DESC["api-benchmarks.test.ts<br/>bcrypt timing, query latency<br/>50 concurrent rate limit calls"]
    end
```

2.1 Unit Tests

Attribute	Value
Scope	Individual functions: auth helpers, fee calculators, input validators
External dependencies	Mocked (no real DB, no real BaaS, no real Sumsub)
Framework	Vitest (via `npm run test`)
Coverage target	≥ 80% overall; 100% for auth + transaction paths
Execution time	Full suite < 30 seconds
Runs on	Every commit, pre-commit hook, CI
Written by	Builder agent (Claude Sonnet) + reviewed by Validator

What to unit test:

- `verifyPassword` — bcrypt correctness; SHA-256 hash rejection
- `signJWT` / `verifyJWT` — JWT signing, tampered token rejection
- Fee calculation — 0.5% remittance fee, 1% QR fee
- Input validation — age ≥ 18, Norwegian phone (+47), password length, XSS/injection rejection

What NOT to unit test:

- Framework boilerplate (Next.js API routes scaffolding)
- Third-party library internals (jose, bcrypt)
- Simple data fetching without logic

2.2 Integration Tests

Attribute	Value
Scope	API endpoints, DB schema, middleware (rate limiter, auth middleware)
External dependencies	Real SQLite test DB; mock BaaS (NEXT_PUBLIC_SERVICE_MODE=mock)
Framework	Vitest
Coverage target	All 26 API routes tested; all DB invariants asserted
Execution time	Full suite < 2 minutes
Runs on	Every PR, blocking merge
Written by	Builder agent + Validator

What to integration test:

- `api-endpoints.test.ts` — all 26 routes: status codes, response shapes, auth gating
- `db.test.ts` — schema invariants: no balance column, no card_number/cvv, FK constraints, transaction type enum
- `middleware.test.ts` — rate limiting (10 req/min auth; 429 on 11th); JWT auth; CSRF protection
- `auth.test.ts` — bcrypt rounds 12; SHA-256 rejection; JWT tampered token rejection

2.3 E2E Tests

Attribute	Value
Scope	Critical user journeys through the real staging environment
External dependencies	Real staging (https://drop-staging.fly.dev/) with mock BaaS
Framework	Playwright (3 projects: user-flows, full-flows, input-chaos)

Attribute	Value
Coverage target	5 critical journeys; input-chaos covers 20+ validation edge cases
Execution time	Full suite < 10 minutes
Runs on	Post-staging deploy, pre-production gate
Written by	QA + Builder agent collaboration

Critical journeys covered by E2E:

1. User registration (3 steps: email+DOB → OTP → PIN)
2. User login + dashboard access
3. Remittance (NOK → RSD/BAM/PKR/TRY/PLN/EUR)
4. QR payment (scan → confirm → receipt)
5. Merchant registration + QR code generation

---

3. Testing Tools

Type	Tool	Version	Purpose	Config File
Unit + Integration	Vitest	2.x	Unit/integration tests	`vitest.config.ts`
E2E testing	Playwright	1.x	Browser E2E (3 projects)	`playwright.config.ts`
Coverage	Vitest (`@vitest/coverage-v8`)	Built-in	Coverage reports	`vitest.config.ts`
Performance	Vitest (bench)	Built-in	Benchmark tests	`api-benchmarks.test.ts`
Mocking	Vitest (`vi.mock`)	Built-in	Mock BaaS, Sumsub, external deps	Built-in
CI/CD	GitHub Actions	—	Automated CI pipeline	`.github/workflows/`

---

4. Test Environments & Data Management

Test Environments

Test Type	Environment	Database	External Services
Unit	Local (no env)	None / in-memory	Mocked via `vi.mock`
Integration	Local + CI	SQLite test DB (`:memory:` or temp file)	Mock BaaS (`NEXT_PUBLIC_SERVICE_MODE=mock`)
E2E	Staging (Fly.io, Stockholm)	Staging SQLite / PostgreSQL	Mock BaaS + Mock Sumsub
Performance	Local (api-benchmarks.test.ts)	SQLite	Mock

Test Data Management

Approach	Used For	Tool	Notes
Fixtures in test files	Unit + integration tests	Vitest `beforeEach`/`afterEach`	Reset per test
Database seeding	E2E tests	`npm run db:seed`	Reset per test run
Synthetic seed data only	All tests	Hardcoded test data	NEVER use real user data (NFR-D04)
Shared test accounts	E2E (consumer + merchant roles)	Vaultwarden "Drop UAT" entries	Never modified by tests

Data cleanup policy: All test data cleaned up after test run via Vitest `afterEach` teardown hooks. SQLite test DB wiped between runs.

---

5. Test Automation Strategy

Test Type	Automation	Trigger	Tooling
Unit tests	100% automated	Pre-commit + CI	Vitest
Integration tests	100% automated	CI on PR	Vitest
E2E (critical paths)	100% automated	Post-staging deploy	Playwright
E2E (input-chaos)	100% automated	Post-staging deploy	Playwright input-chaos project
Performance tests	Automated baseline	Every CI run	Vitest benchmarks (api-benchmarks.test.ts)
Security (SAST)	100% automated	Every PR	GitHub Actions + `npm audit`
DB compliance	100% automated	Every commit	`db.test.ts` (Vitest)
Manual exploratory	Manual	Pre-release UAT	CEO (Alem)

6. Manual Testing Scope

Manual testing is required for:

- CEO (Alem) UAT walkthrough before Phase 1 production launch
- BankID SCA flow (Phase 2 — not yet integrated)
- Real BaaS payment flow (Phase 2 — mock only in MVP)
- Mobile device testing on physical iOS/Android hardware (Playwright covers 375px viewport)
- Accessibility usability on actual assistive technology

Manual testing is NOT required for:

- Regression of all automated test paths
- Fee calculations (fully automated in unit tests)
- Input validation (covered by input-chaos Playwright project)
- DB compliance checks (fully automated in db.test.ts)

---

7. Code Coverage Requirements

Layer	Lines	Branches	Functions	Notes
Auth module	100%	100%	100%	Strictly enforced — financial security
Transaction processing	100%	100%	100%	Strictly enforced — financial correctness
API handlers	≥ 80%	≥ 70%	≥ 80%	
Input validation	≥ 90%	≥ 85%	100%	Security-critical
DB layer	≥ 90%	—	≥ 90%	Compliance assertions required
Overall minimum	**≥ 80%**	—	—	CI gate — build fails below

Coverage enforcement: CI pipeline fails if coverage drops below 80% overall **Coverage report:** Published as CI artifact on every PR

---

8. Quality Gates

PR Merge Gate (must pass before merge)

- ☐ All unit tests pass (`npm run test`)
- ☐ All integration tests pass
- ☐ Coverage ≥ 80% (≥ 100% for auth + transaction paths)
- ☐ No HIGH/CRITICAL security findings (`npm audit`)
- ☐ No secrets detected (pre-commit hook)
- ☐ TypeScript compiles (`npm run type-check`)
- ☐ Linting passes (`npm run lint`)

Staging Deploy Gate

- ☐ All PR gates passed
- ☐ Build artifact created and pushed to Fly.io

Production Deploy Gate (must pass before production deploy)

- ☐ All Playwright E2E tests pass on staging (user-flows, full-flows, input-chaos)
- ☐ Performance baseline not degraded > 10% (api-benchmarks.test.ts)
- ☐ Manual UAT sign-off from Alem Bašić (CEO)
- ☐ Security audit score ≥ 80/100

---

9. Responsibility Matrix

Test Type	Writes Tests	Reviews Tests	Maintains Tests	Signs Off
Unit tests	Builder agent	Validator agent	Builder agent	John (AI Director)
Integration tests	Builder agent	Validator agent	Builder agent	John (AI Director)
E2E tests	Builder agent	Validator agent	Builder agent	John (AI Director)
Performance tests	Builder agent	Validator agent	Builder agent	John (AI Director)
Security tests (automated)	Builder agent	Validator agent	Builder agent	John (AI Director)
DB compliance tests (db.test.ts)	Builder agent	Validator agent	Builder agent	John (AI Director)

Test Type	Writes Tests	Reviews Tests	Maintains Tests	Signs Off
UAT (manual)	N/A	John	N/A	Alem Bašić (CEO)

---

10. Test Reporting & Metrics

Metric	Target	Reporting
Test pass rate	≥ 100% (unit/integration), ≥ 95% (E2E)	CI dashboard
Flaky test rate	< 2%	Per-run analysis
Test execution time (unit+integration)	< 3 minutes total	CI dashboard
Coverage trend	Stable or improving	PR comments
DB compliance tests	100% pass always	CI dashboard

---

11. Continuous Testing in CI/CD

Stage	Tests Run	Blocking
Pre-commit (local)	Unit tests, linting, type-check	Yes
PR open / update	Unit + integration + SAST (`npm audit`)	Yes — blocks merge
Staging deploy	Playwright E2E (all 3 projects)	Yes — blocks production
Production deploy	Smoke tests (health + critical journeys)	Yes — auto-rollback on failure
Scheduled (nightly)	Full E2E suite	No — alerts only

Current test inventory (14 test files):

- Unit: `auth.test.ts`, `validation.test.ts`, `transactions.test.ts`, `rates.test.ts`
- Integration: `api-endpoints.test.ts`, `db.test.ts`, `middleware.test.ts`
- Performance: `api-benchmarks.test.ts`
- Regression: `regression-suite.test.ts`, `feature-flags.test.ts`, `sumsub-integration.test.ts`, `cards-integration.test.ts`
- E2E: `user-flows.spec.ts`, `full-flows.spec.ts`, `input-chaos.spec.ts`

Related Documents

- Test Plan
- E2E Test Plan
- Performance Test Plan
- Definition of Done
- Testing Guide
- Test Inventory

Approval

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

Revision #5
Created 2026-02-23 12:05:59 UTC by John
Updated 2026-05-31 20:03:25 UTC by John