Lessons Learned: Drop — Fintech Payment App

Lessons Learned: 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 lessons learned — Phase 0 + Phase 0.5 Security Hardening

---

> **"Lessons learned without behavior change is not learning."** — Alem Bašić (CEO), 2026-02-13 (ZAKON #0)

Every lesson in this document must produce either a system fix (hook / tool / rule / ADR) or a knowledge base contribution. If a lesson has no action item, it does not belong here — it belongs in a complaint log.

---

1. Session Metadata

Field	Value

Session Type	Phase Review (Phase 0 → Phase 0.5)
Project	Drop — Remittance + QR Payments
Phase / Sprint	Phase 0 (MVP Architecture) + Phase 0.5 (Security Hardening)
Period Covered	2026-01-01 – 2026-02-23
Facilitator	John (AI Director)
Participants	John (AI Director), Builder Agent(s), Validator Agent, Alem Bašić (CEO)
Date of Session	2026-02-23
Duration	Async — Phase retrospective

---

2. What Went Well (Keep Doing)

#	What Went Well	Why It Worked	How to Preserve	Category
KD-01	Pass-through model (ADR-003) enforced by automated tests in `db.test.ts` from Day 1	Tests as compliance guardrails mean no human discipline required to maintain the invariant	Keep `db.test.ts` assertions (no balance, no CVV) as non-negotiable CI gate; never disable	Technical
KD-02	AI-native async workflow via Mission Control eliminated coordination overhead	Builder agents pick up tasks independently; no standup time wasted; John coordinates via task system	Continue using Mission Control for all task assignments; document decisions in `comms/decisions/`	Process
KD-03	Security audit conducted before Phase 1 — found 8 critical/high issues before production	External-perspective audit at right phase (post-MVP, pre-launch) caught real vulnerabilities	Run security audit at start of each major phase (0.5 → 1 → 2 → 3), not at the end	Technical
KD-04	Mock BaaS (`NEXT_PUBLIC_SERVICE_MODE=mock`) let full app architecture be built and tested without a real BaaS partner	Decoupled business logic from partner dependency — could ship working app and test all flows	Maintain the service mode abstraction layer; expand to `SUMSUB_MODE=mock` pattern for other integrations	Technical
KD-05	SQLite for local dev and test removed Docker dependency entirely — team setup time < 5 min	Zero infrastructure overhead for development; any agent can be productive immediately	Keep SQLite for local/test; PostgreSQL migration plan documented in Phase 1 roadmap	Process

#	What Went Well	Why It Worked	How to Preserve	Category
KD-06	Comprehensive test suite (40+ tests across 14 files) written alongside MVP code	Tests document expected behavior and make refactoring in Phase 0.5 safe and fast	Maintain test-first discipline; DoD requires tests for all new code	Technical
KD-07	Figma Make export as UI source of truth (`mockups/figma-make-export/`) prevented design drift	Single source for UI decisions; agents always check Make components before modifying UI	Always reference `mockups/figma-make-export/` before any UI change; document in CLAUDE.md	Process

Highlights

KD-01 Detail: The decision to add compliance assertions in `db.test.ts` on Day 1 (Phase 0) paid dividends throughout Phase 0.5. Every security hardening change that touched the schema was automatically validated against the pass-through model invariant. No developer (human or AI) had to remember "don't add a balance column" — the test suite enforced it. This pattern should be replicated for every regulatory constraint: make it a test, not a rule someone has to remember.

KD-03 Detail: The security audit at Phase 0.5 (pre-production, post-MVP) found a security score of 57/100 — 8 critical/high issues including no CSRF, in-memory rate limiting that resets on restart, bcrypt DoS via long passwords, and missing security headers. If these had been found post-launch, they would have been P0 production incidents. The timing was correct: audit after architecture is stable, before real users are onboarded. This should be standard practice for every Drop phase.

---

3. What Didn't Go Well (Stop Doing)

#	What Didn't Work	Root Cause	Impact (1-10)	Action Item	Owner	Deadline
SD-01	In-memory rate limiting reset on server restart — not durable across deploys	Chose simplest implementation (Node process memory) without considering Fly.io restart behavior	7	Replaced with DB-backed `rate_limit_requests` table in Phase 0.5. Document in tech-debt-log.md immediately when choosing non-durable implementations.	Builder Agent / John	Completed (Phase 0.5)

#	What Didn't Work	Root Cause	Impact (1-10)	Action Item	Owner	Deadline
SD-02	No CSRF protection on POST endpoints — discovered in security audit	Security checklist not applied during API endpoint implementation	8	Added CSRF middleware in Phase 0.5. Add CSRF to DoD security checklist as mandatory gate for all POST/PATCH/DELETE.	John	Completed (Phase 0.5)
SD-03	bcrypt DoS vector via unbounded password length — no input max before hashing	Missed OWASP Input Validation guidance for bcrypt specifically	7	Added Zod `max(1000)` before bcrypt in Phase 0.5. Added regression test. Document bcrypt DoS pattern in coding-standards.md.	Builder Agent / John	Completed (Phase 0.5)
SD-04	Security score 57/100 at first audit — below target of 80/100	Security was a "Phase 0.5 concern" — deferred until post-MVP, leading to 8 issues needing remediation	7	In future phases: run OWASP Top 10 checklist during feature development, not as a batch post-MVP audit. Add security review gate to each PR in DoD.	John	Ongoing (Phase 1 onwards)
SD-05	Missing `npm audit` gate in initial CI — dependencies with CVEs could have been shipped	CI/CD pipeline focused on tests only; dependency security check not included	6	Add `npm audit --audit-level=high` to GitHub Actions CI pipeline. Fail build on HIGH/CRITICAL CVEs.	Builder Agent	Phase 1

#	What Didn't Work	Root Cause	Impact (1-10)	Action Item	Owner	Deadline
SD-06	No structured logging from Day 1 — `console.log` used for debugging in some paths	Fast MVP pace — structured logging deferred as "Phase 1 concern"	5	Add Pino structured logging in Phase 1. Document in tech-debt-log.md as TD-008.	John	Phase 1

Root Cause Analysis — Deep Dives

SD-04: Security deferred to Phase 0.5 caused remediation batch work

Problem: 8 critical/high security issues found in audit — all required in Phase 0.5 sprint.

5 Whys:

1. Why were 8 security issues found in the audit? → Because security checklist was not applied during Phase 0 feature development.
2. Why was the security checklist not applied? → Because Phase 0 goal was "working architecture" — security was explicitly a Phase 0.5 task.
3. Why was security explicitly deferred? → Because the team assumed a working architecture could be security-hardened efficiently afterward.
4. Why is that assumption problematic? → Because retrofitting security (CSRF, rate limiting, headers) into existing code is more work than building it in — and some security patterns require architectural changes.
5. Why wasn't this anticipated? → No "security baseline" was defined for Phase 0 — only for Phase 0.5.

Root Cause: Deferred security without a minimum viable security baseline for Phase 0 meant the Phase 0.5 sprint carried all security work. **Systemic Fix:** Define a Minimum Security Baseline (MSB) for every phase. Phase 0 MSB should include: parameterized SQL, bcrypt passwords, httpOnly JWT, basic rate limiting. Advanced security (CSRF, security headers, audit logging) is Phase 0.5. Document MSB in DoD. **Ljestvica Fikseva (ZAKON #1):** Rule update → DoD security section updated with MSB definition.

---

4. What to Try Next (Start Doing)

#	What to Try	Expected Benefit	Hypothesis	Success Metric	Trial Period

ST-01	OWASP Top 10 checklist review on every PR (not just security audit phases)	Catch security issues per-feature instead of in batch	If we review OWASP Top 10 per PR, then Phase 1 security audit score will be > 80/100 without a dedicated security hardening sprint	Security audit score ≥ 80/100 at Phase 1 review without a dedicated 0.5-equivalent sprint	Phase 1 (3 months)
ST-02	`npm audit` as CI gate — fail build on HIGH/CRITICAL CVEs	Prevent known vulnerable dependencies from reaching staging	If we gate on `npm audit`, then no HIGH/CRITICAL CVEs reach production	Zero HIGH/CRITICAL CVEs at Phase 1 launch	Phase 1
ST-03	Performance regression gate in CI — `api-benchmarks.test.ts` as a CI check	Prevent silent performance regressions from shipping	If we run benchmarks in CI, then no P95 regression > 15% reaches staging	0 performance regressions > 15% across Phase 1	Phase 1
ST-04	Structured logging (Pino) from Phase 1 Day 1 — not retrofitted	Better observability from launch; no retrofitting cost	If we add Pino from the start, Phase 1 debugging will be 50% faster than Phase 0	Avg time to diagnose a production issue < 30 min	Phase 1

---

5. Key Findings by Category

5.1 Technical Findings

#	Finding	Impact	Recommendation
TF-01	SQLite WAL mode handles moderate concurrency but has a hard limit (~200 concurrent writes). PostgreSQL migration is required before launch.	High	Migrate to PostgreSQL in Phase 1 before real user onboarding. Plan migration script in `db/migrations/`.
TF-02	Mock service abstraction (`NEXT_PUBLIC_SERVICE_MODE`) enables full E2E testing without BaaS partner. Pattern is sound.	High (positive)	Extend pattern to `SUMSUB_MODE`, `BANKID_MODE` for Phase 2. Document in local-development-setup.md.

#	Finding	Impact	Recommendation
TF-03	bcrypt cost factor 12 adds ~800ms to login/register. This is intentional security but users will notice on slow devices.	Medium	Profile on target device hardware. Add loading state in UI during auth operations.
TF-04	Next.js App Router with server-side API routes gives clean separation but RSC (React Server Components) hydration can cause subtle state issues.	Medium	Add E2E test coverage for client-side navigation. Monitor for hydration errors in Phase 1.
TF-05	26 API routes with consistent error format (typed responses, 402/403 for financial errors) makes client-side error handling reliable.	High (positive)	Maintain consistent error format as new routes are added in Phase 1.

Technical Patterns Observed:

- Test-as-compliance-guardrail pattern (db.test.ts) — should be replicated for every regulatory constraint going forward
- Service mode abstraction (mock/live) — excellent pattern for decoupling from external partners; extend to all integrations

5.2 Process Findings

#	Finding	Root Cause	Recommendation
PF-01	AI-native async workflow works well for implementation tasks but requires explicit handoff documentation when agents change	No persistent agent memory between sessions	Maintain CLAUDE.md, onboarding guide, and tech-debt-log.md as "agent continuity" documents. Update before every session end.
PF-02	Security as a Phase 0.5 sprint (rather than baked into Phase 0) created batch remediation work	Phase planning deferred security to avoid blocking MVP	Define Minimum Security Baseline per phase. Phase 0 = auth + parameterized SQL. Phase 0.5 = CSRF + rate limiting + headers.
PF-03	Mission Control + CLAUDE.md combination works as tribal knowledge preservation for AI agents	Explicit documentation of decisions and patterns	Maintain both. CLAUDE.md = quick reference. Mission Control = task history. `comms/decisions/` = decision log.

Process Patterns Observed:

- AI-native teams need more explicit documentation than human teams (no implicit knowledge retention between sessions) — this is a feature, not a bug, when documentation discipline is enforced

5.3 Communication Findings

#	Finding	Stakeholder Impact	Recommendation
CF-01	Alem (CEO) needs clear phase milestone signals — not technical details	CEO makes business decisions (BaaS partner, Finanstilsynet timing) based on phase readiness	Weekly email from John to alem@alai.no with phase status, blockers, and required CEO decisions.
CF-02	Security audit score (57/100) communicated effectively as "needs Phase 0.5 work before Phase 1" — Alem approved correctly	No confusion about scope or priority	Continue using numerical scores + plain language summaries for executive communication.

5.4 Tools & Infrastructure Findings

#	Finding	Impact	Recommendation
IF-01	Fly.io + SQLite (Phase 0) setup works for MVP/demo; will need PostgreSQL + Fly.io Postgres addon for Phase 1	High	Add PostgreSQL migration to Phase 1 task list in Mission Control.
IF-02	GitHub Actions CI with Vitest + Playwright covers all test types. No gaps for current scope.	High (positive)	Add `npm audit` gate to existing CI pipeline in Phase 1.
IF-03	Vaultwarden (`vault.basicconsulting.no`) as secrets manager works. bw CLI access for agents via session token.	Medium	Ensure all new Phase 1 secrets (BaaS API keys, Sumsub production keys) go into Vaultwarden immediately.

5.5 Team & Collaboration Findings

#	Finding	Impact	Recommendation
TM-01	Builder Agent + Validator Agent (read-only) separation works well for fintech — Validator catches issues Builder misses	High (positive)	Maintain read-only constraint on Validator. Never give Validator write access — reduces risk of well-intentioned but unreviewed changes.

#	Finding	Impact	Recommendation
TM-02	John (AI Director) as architecture owner with clear ADR documentation enables consistent decisions across agent sessions	High (positive)	Continue ADR pattern. All significant decisions documented in `project/architecture/`.

---

6. Action Items

#	Action Item	Category	Owner	Sprint / Deadline	Priority	Status
AI-01	Add `npm audit --audit-level=high` to GitHub Actions CI	Technical	Builder Agent	Phase 1, Sprint 1	P1	Open
AI-02	Add OWASP Top 10 checklist to PR template and DoD	Process	John	Phase 1, Sprint 1	P1	Open
AI-03	Define Minimum Security Baseline (MSB) for each phase and document in DoD	Process	John	Phase 1, Sprint 1	P1	Open
AI-04	Plan PostgreSQL migration — create `db/migrations/postgres-migration.md`	Technical	John + Builder Agent	Phase 1, Sprint 2	P1	Open
AI-05	Add Pino structured logging to Phase 1 from Day 1	Technical	Builder Agent	Phase 1, Sprint 1	P2	Open

#	Action Item	Category	Owner	Sprint / Deadline	Priority	Status
AI-06	Add performance regression gate (`api-benchmarks.test.ts`) to CI	Technical	Builder Agent	Phase 1, Sprint 2	P2	Open
AI-07	Weekly CEO status email template — phase status, blockers, required decisions	Communication	John	Ongoing from Phase 1	P2	Open
AI-08	Extend service mode abstraction: `SUMSUB_MODE=mock/live` + `BANKID_MODE=mock/live`	Technical	Builder Agent	Phase 2, Sprint 1	P3	Open

Action Item Review: At the START of each phase planning, John reviews status of all open action items. Incomplete P1 items are escalated to Alem Bašić (CEO).

---

7. Metrics Comparison — Planned vs. Actual

7.1 Timeline

Milestone	Planned	Actual	Variance	Root Cause (if significant)
Phase 0 MVP Architecture	2026-01-15	2026-02-01	+17 days	Architecture decisions (pass-through model, ADR-003) took longer to finalize
Phase 0.5 Security Hardening	2026-02-28	2026-02-23	-5 days	Security fixes were well-scoped; AI agent velocity higher than estimated
Phase 1 BaaS Integration	2026-04-30	TBD	—	Blocked on BaaS partner confirmation

7.2 Budget

Category	Planned (NOK)	Actual (NOK)	Variance	Notes
Development (AI agents)	150,000	~50,000 (est.)	Under	AI agents significantly cheaper than human developer equivalent
Infrastructure (Fly.io)	10,000/year	~2,000 (est.)	Under	MVP usage is low; scales with real users
Design	20,000	5,000 (est.)	Under	Figma Make for UI source of truth reduced design cost
Total	**~250,000**	**~57,000 (est.)**	**Under**	Budget runway preserved for Phase 2 BaaS/compliance costs

7.3 Quality

Metric	Target	Actual	Status
Test coverage	≥ 80%	~85% (est.)	Pass
Pass-through invariant tests	Always pass	Always pass	Pass
Critical bugs at Phase 0.5	0	0	Pass
Security audit score	≥ 80/100 post-0.5	57/100 → targeting 80/100	In Progress
All 40+ Vitest tests passing	Yes	Yes	Pass
Playwright E2E (3 projects)	Pass	Pass	Pass

7.4 Velocity & Estimation

Sprint	Estimated Points	Completed Points	Accuracy
Phase 0 MVP	80	85	106%
Phase 0.5 Security	40	42	105%
Average			**~105%**

Estimation Accuracy Notes: AI agents tend to complete slightly more than estimated when tasks are well-defined with clear acceptance criteria. Under-estimation typically occurs on architectural decision tasks (ADRs, design exploration) rather than implementation tasks.

---

8. Recommendations for Future Projects

#	Recommendation	Category	Confidence	Applicable When
REC-01	Define a Minimum Security Baseline (MSB) for every phase before development starts. Phase 0 MSB = parameterized SQL + bcrypt + httpOnly JWT. Phase 0.5 = CSRF + rate limiting + headers.	Technical	High	All fintech/payment projects from Day 1
REC-02	Build compliance invariants as automated tests before any feature code. (e.g., `db.test.ts` for no balance/CVV assertions.)	Technical	High	Any project with regulatory constraints
REC-03	Use service mode abstraction (mock/live) for all external integrations from the start. Enables architecture validation without partner dependency.	Technical	High	Any project with external payment/identity providers
REC-04	For AI-native teams: CLAUDE.md + Mission Control + `comms/decisions/` is the minimum viable knowledge preservation system. Update it every session.	Process	High	All AI-native team projects
REC-05	Run `npm audit` as a CI gate from Day 1. Adding it later means clearing a backlog of CVEs. Starting clean is much easier.	Technical	High	All Node.js projects

Top 3 Most Important Learnings

1. **Compliance as tests, not rules:** The most reliable way to maintain the pass-through model invariant was making it a CI test (`db.test.ts`) rather than a team convention. Tests cannot be forgotten; conventions can. For any regulatory constraint, immediately ask: "How do I make this a test?"
2. **Security baseline vs. security sprint:** Deferring all security to a dedicated Phase 0.5 sprint was better than nothing but created batch remediation work. The right model is a per-phase Minimum Security Baseline (MSB) — implement the MSB during feature development, not afterwards. Advanced hardening (CSRF, security headers, audit logging) can still be a dedicated sprint, but the baseline prevents the worst vulnerabilities from ever shipping.
3. **Mock integrations enable architecture confidence without partner dependency:** The `NEXT_PUBLIC_SERVICE_MODE=mock` pattern let Drop build, test, and validate 26 API routes and 3 Playwright E2E projects without a confirmed BaaS partner. This is the correct approach for any product that depends on external financial partners. Build to the interface; connect the real partner when the contract is signed.

---

9. Knowledge Base Contribution

Finding	Contribution Type	Target System	Status	Owner
Compliance-as-tests pattern	New rule	DoD + Coding Standards	Done	John
Per-phase Minimum Security Baseline	New rule	DoD security checklist	Open (AI-03)	John
Mock service mode abstraction pattern	New convention	CLAUDE.md + Coding Standards	Done	John
bcrypt DoS via unbounded input	New rule	Coding Standards + auth patterns	Done	John
DB-backed rate limiting required for persistence	Tech debt entry + lesson	tech-debt-log.md TD-004	Done	John

HiveMind entries to add (run after reading this document):

- `node ~/system/agents/hivemind/hivemind.js post "Drop: compliance-as-tests pattern — make regulatory invariants into CI tests, not team conventions"`
- `node ~/system/agents/hivemind/hivemind.js post "Drop: mock service mode abstraction (NEXT_PUBLIC_SERVICE_MODE) enables full E2E testing without BaaS partner — extend to SUMSUB_MODE and BANKID_MODE in Phase 2"`

CLAUDE.md updated:

- `~/ALAI/products/Drop/CLAUDE.md` — Pass-through model rules, mock mode documentation

Related Documents

- Tech Debt Log
- Security Audit Report
- Definition of Done
- Architecture Decision Records
- Risk Register

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 12:06:57 UTC by John
Updated 2026-05-31 20:03:37 UTC by John