drop-mvp-pipeline-plan

Plan: Drop MVP Pipeline A-Z

Research Summary

Actual State (13.02.2026)

Phase 4 (Implementation) — 95% done, NOT 80% as PIPELINE.md says:

- 12 frontend pages: ALL wired to real API routes via fetch()
- 24+ API routes: ALL working (auth, transactions, cards, merchants, rates, etc.)
- SQLite DB: 6 tables + seed data, parameterized queries
- Auth: JWT httpOnly cookies, rate limiting
- Validation: hardened (backend + frontend), 18+ age check
- Services: mock-swan/stripe/sumsub with config toggle (expected for MVP)
- Rebrand: ALL pages rebranded to Stitch design
- Tests: 126 unit + 91 e2e = 217 ALL GREEN

What's actually missing for MVP-ready:

1. Full remittance E2E flow test (register → login → send money → verify)
2. .env.example + environment config for production
3. Docker build verification (Dockerfile exists but untested)
4. SQLite → needs volume mount for persistence in Docker
5. 5 test iterations per testing.md standard
6. Deploy to staging (Railway/Hetzner — cost analysis says 10-170 NOK/mo)
7. Domain config (getdrop.no)
8. Landing page deploy (static HTML, separate from app)
9. PIPELINE.md outdated — needs update

Infrastructure already built:

- Dockerfile: 3-stage build, standalone output □
- docker-compose.yml: app + postgres, healthchecks □
- next.config.ts: standalone, CSP headers, security headers □
- Cloud cost analysis: done □

NOTE: docker-compose.yml has postgres service but app uses SQLite. For MVP: deploy with SQLite + volume (Railway/Fly persistent disk). PostgreSQL migration is Phase 2 (200+ users) per cost analysis.

Objective

Take Drop from "works on localhost" to "deployed MVP on staging with full test coverage." 4 phases, no detours, no cosmetics.

Team Orchestration

Team Members

ID	Name	Role	Agent Type
B1	flow-test-builder	Write full remittance E2E flow + missing flow tests	builder
V1	test-validator	Run all 5 test levels, 5 iterations, verify coverage	validator
B2	deploy-prep-builder	Create .env.example, fix docker-compose for SQLite, test Docker build	builder
V2	deploy-validator	Verify Docker image builds and runs correctly	validator
B3	staging-builder	Deploy to Railway/Hetzner, configure domain, SSL	builder
V3	staging-validator	Verify staging is live, all pages load, API responds	validator
B4	pipeline-closer	Update PIPELINE.md, close MC tasks, create post-mortem	builder

Step-by-Step Tasks

Phase 1: Complete Implementation (finish Phase 4)

Task 1: Write missing E2E flow tests + update PIPELINE.md status

- Owner: B1
- BlockedBy: none

- Files:
  - `tests/e2e/full-flows.spec.ts` (NEW)
  - `project/PIPELINE.md` (UPDATE)
  - `.env.example` (NEW)
- Instructions:
  1. Read existing test files to understand patterns
  2. Write full E2E tests for:
     - Complete registration → login → dashboard flow
     - Login → send money (remittance) → verify transaction in history
     - Login → scan QR → pay → verify in history
     - Login → view cards → order virtual card
     - Login → view accounts → check balances
     - Login → profile → logout → redirect to login
  3. Create `.env.example` with all required vars:

     ```
     JWT_SECRET=change-me-in-production
     NODE_ENV=production
     NEXT_PUBLIC_SERVICE_MODE=mock
     ```

  4. Update PIPELINE.md Phase 4 status to 100%
  5. Run tests, fix any failures
- Acceptance:
  - ☐ full-flows.spec.ts has 6+ complete user journey tests
  - ☐ All E2E tests pass (existing + new)
  - ☐ .env.example exists with documented vars
  - ☐ PIPELINE.md Phase 4 marked complete

Phase 2: Testing (5 iterations per testing.md)

Task 2: Run 5 test iterations across all levels, fix failures

- Owner: V1
- BlockedBy: 1
- Instructions:
  1. Read ~/system/rules/testing.md for requirements
  2. Run iteration 1: `npx vitest run` + `npx playwright test`
  3. Log results to `tests/logs/iteration-1.txt`
  4. Fix any failures found
  5. Repeat for iterations 2-5
  6. Check coverage: `npx vitest run --coverage` (target 80%+)
  7. Run regression tests: `npx vitest run tests/regression/`
  8. Run performance tests: `npx vitest run tests/performance/`
  9. Final summary of all 5 iterations
- Acceptance:
  - ☐ 5 iterations logged in tests/logs/

- [ ] ALL tests pass on iteration 5
- [ ] Coverage report generated
- [ ] No regressions
- [ ] Performance benchmarks baselined

Phase 3: Deploy to Staging

Task 3: Prepare Docker for SQLite deployment

- Owner: B2
- BlockedBy: 2
- Files:
  - `docker-compose.yml` (UPDATE — add SQLite volume, remove postgres for MVP)
  - `docker-compose.production.yml` (NEW — for future postgres)
  - `Dockerfile` (UPDATE if needed)
- Instructions:
  1. Read existing Dockerfile and docker-compose.yml
  2. Create `docker-compose.mvp.yml` for SQLite deployment:
     - App service with SQLite volume mount
     - JWT_SECRET from env
     - Health check on /api/health
     - No postgres (not needed for MVP)
  3. Keep existing docker-compose.yml as `docker-compose.production.yml` (postgres version)
  4. Test: `docker build -t drop-app .` — must succeed
  5. Test: `docker run -p 3000:3000 -e JWT_SECRET=test drop-app` — must serve pages
  6. Verify /api/health, /login, /onboarding all respond 200
- Acceptance:
  - [ ] Docker image builds successfully
  - [ ] Docker container starts and serves pages
  - [ ] /api/health returns 200
  - [ ] SQLite data persists across container restart (volume)
  - [ ] docker-compose.mvp.yml works with `docker-compose -f docker-compose.mvp.yml up`

Task 4: Validate Docker deployment

- Owner: V2
- BlockedBy: 3
- Acceptance:
  - [ ] Docker build < 2 minutes
  - [ ] Image size < 500MB
  - [ ] Container starts in < 10s

- [ ] All API endpoints respond correctly
- [ ] No security warnings in container logs

Task 5: Deploy to staging (Railway or Hetzner)

- Owner: B3
- BlockedBy: 4
- Instructions:
  1. Read cloud-cost-analysis.md for recommended approach
  2. Option A (Railway): `railway init` + `railway up` — easiest, persistent disk
  3. Option B (Hetzner): Docker deploy to existing VPS if available
  4. Configure:
     - JWT_SECRET (generate secure random)
     - NODE_ENV=production
     - Domain: staging.getdrop.no or drop-staging.alai.no
  5. Set up SSL (Let's Encrypt / Cloudflare)
  6. Verify all pages load on staging URL
  7. Deploy landing page (static HTML) to same domain or subdomain
- Acceptance:
  - [ ] Staging URL accessible via HTTPS
  - [ ] All 12 app pages load correctly
  - [ ] Login + registration flow works end-to-end
  - [ ] API health check passes
  - [ ] Landing page live

Task 6: Validate staging deployment

- Owner: V3
- BlockedBy: 5
- Instructions:
  1. Hit staging URL — verify HTTPS, correct domain
  2. Test all pages load (no 404, no 500)
  3. Test registration flow end-to-end
  4. Test login with demo credentials
  5. Test send money flow
  6. Test QR scan flow
  7. Check security headers (CSP, HSTS, X-Frame-Options)
  8. Check mobile responsiveness (375px viewport)
- Acceptance:
  - [ ] All pages load via HTTPS
  - [ ] Registration + login works
  - [ ] Core flows work (send, scan, cards)
  - [ ] Security headers present

- [ ] Mobile-friendly

Phase 4: Close Pipeline

Task 7: Update PIPELINE.md, close tasks, create summary

- Owner: B4
- BlockedBy: 6
- Instructions:
  1. Update PIPELINE.md:
     - Phase 4: ✅ Done
     - Phase 5: ✅ Done (5 iterations, all pass)
     - Phase 6: ✅ Done (staging live at URL)
     - Phase 7: 🔄 Monitoring
  2. Close related MC tasks: #526, #608, #791-793 (update status)
  3. Create post-mortem summary:
     - What was built
     - Test results
     - Staging URL
     - Known limitations (mock services, SQLite, no BankID)
     - Next steps for production (partner agreements, PostgreSQL, Finanstilsynet)
  4. Post summary to HiveMind
- Acceptance:
  - [ ] PIPELINE.md fully updated
  - [ ] MC tasks updated
  - [ ] Post-mortem written
  - [ ] HiveMind updated
  - [ ] Alem can access staging URL

Validation Commands

```
# Phase 1 — Tests
cd ~/ALAI/products/Drop/src/drop-app
npx vitest run                    # Unit tests
npx playwright test               # E2E tests

# Phase 2 — Coverage
npx vitest run --coverage

# Phase 3 — Docker
```

```
docker build -t drop-app .
docker run -p 3001:3000 -e JWT_SECRET=test123 drop-app
curl http://localhost:3001/api/health

# Phase 4 — Staging
curl -I https://staging.getdrop.no
curl https://staging.getdrop.no/api/health
```

Risk Register

Risk	Impact	Mitigation
Docker build fails (native deps)	Blocks deploy	better-sqlite3 needs python/g++ in Dockerfile — already there
Railway free tier limits	Slow staging	Upgrade to $5/mo Starter — within budget
SQLite concurrent writes	Data corruption under load	MVP only, < 200 users. PostgreSQL in Phase 2
No real BankID	Can't verify real users	Expected for MVP. Mock BankID with DOB field
No partner APIs (Swan/Stripe/Sumsub)	All transactions are mock	Expected. Partner agreements are business tasks, not code

Timeline

Phase	Tasks	Parallel?	Estimated
1: Complete impl	Task 1	Solo	1 builder
2: Testing	Task 2	Solo (blocked by 1)	1 validator
3: Deploy	Tasks 3-6	B2→V2→B3→V3 sequential	2 builders + 2 validators
4: Close	Task 7	Solo (blocked by 6)	1 builder

Total: 4 builders + 3 validators = 7 agent tasks

Revision #4
Created 2026-02-18 08:44:46 UTC by John
Updated 2026-05-31 20:02:20 UTC by John