Petter Graff Architecture Review

Drop Fintech Platform — Architecture Review

Reviewer: Petter Graff (Software Architect, CTO Pratexo) **Review Date:** 2026-02-22 **Subject:** Drop — Norwegian fintech payment application (remittance + QR payments) **Model:** PSD2 pass-through (AISP/PISP) — Drop never holds customer funds

> **NOTE (2026-03-03):** This review was conducted against the pre-ADR-014 codebase. SQLite/dual-driver findings are historical — resolved by ADR-014 (PostgreSQL-only + Drizzle ORM).

---

1. Overall Assessment

Grade: C+ (6.5/10)

Drop has a **solid conceptual foundation** for a PSD2-regulated fintech application. The architectural documentation is comprehensive, the compliance framework is thoughtfully designed, and the core technical decisions (monolith-first, dual-database abstraction, BankID-only auth) are appropriate for an MVP. However, **critical production-readiness gaps** exist across security, data integrity, and operational resilience that would prevent regulatory approval and create significant business risk.

What I see here: An engineering team that understands fintech compliance requirements at a *documentation level* but has not yet built the operational muscle memory to implement them correctly. The Vault Squad analysis is accurate — you have 17 CRITICAL findings that must be fixed before processing a single real transaction.

This is **not a failing grade** — it's a realistic early-stage fintech build. But you're not production-ready, and pretending otherwise would be dangerous.

---

2. What's Done Well

2.1 Regulatory Awareness

- **19-table schema** includes 7 dedicated compliance tables (`audit_log`, `aml_alerts`, `str_reports`, `screening_results`, `consents`, `data_access_requests`, `complaints`) — most fintechs bolt these on later
- **Pass-through PSD2 model** correctly avoids e-money license complexity — Drop positions as PISP/AISP only
- **BankID-only authentication** is the right call for Norwegian fintech (SCA by design, no password management)
- **Dual-database abstraction** (`db.ts`) with SQL compatibility translation is clean engineering — SQLite for dev speed, PostgreSQL for production reliability

2.2 Security Fundamentals

- **Parameterized SQL throughout** — zero SQL injection vulnerabilities
- **bcrypt with 12 rounds** for password hashing
- **Idempotency keys** on transactions with unique index — double-charge protection exists
- **Structured audit logging** with IP, user-agent, request-id correlation

2.3 AML/Compliance Framework

The transaction monitoring module (`transaction-monitor.ts`) has 5 rule types:

- Structuring detection (multiple small txns avoiding thresholds)
- Velocity checks (>5 txns/hour)
- High-amount flagging (>25K NOK)
- High-risk corridor detection (FATF grey-list countries)
- Unusual pattern analysis (3x user average)

Problem: It's never called from any API route (Vault Squad finding BE-C2). Dead code doesn't count.

2.4 Documentation Quality

C4 diagrams, ADRs, and HLD documents are better than 80% of fintech startups. Architecture is understandable, decisions documented with rationale, trade-offs acknowledged.

---

3. Critical Improvements (Production Blockers)

3.1 Authentication Catastrophe

Finding	Impact	Source
JWT secret defaults to hardcoded `"dev-secret-change-in-production"`	Anyone can forge tokens if env var missing	`auth.ts:8`
No OIDC `state` validation	CSRF on authentication flow	`bankid.ts:80-108`
No OIDC `nonce` validation	ID token replay attacks	`bankid.ts:93-101`
Cookie missing `Secure` flag	Tokens sent over HTTP in plaintext	`auth.ts:206`

Risk: Complete account takeover possible.

Fix:

```
// Startup validation
if (process.env.NODE_ENV === 'production' &&
    process.env.JWT_SECRET === 'dev-secret-change-in-production') {
  throw new Error('FATAL: Production requires real JWT_SECRET');
}

// BankID callback — validate state + nonce
const storedState = cookies.get('bankid_state');
const storedNonce = cookies.get('bankid_nonce');
if (state !== storedState) throw new Error('Invalid state');
if (idToken.nonce !== storedNonce) throw new Error('Invalid nonce');

// Cookie flags
cookies.set('drop_token', token, {
  httpOnly: true,
  secure: true,
  sameSite: 'strict',
  maxAge: 1800  // 30min, not 7 days
});
```

3.2 PISP Call Inside Database Transaction

External HTTP call (30s timeout) while holding database write lock. Under load → deadlocks, data loss, double-charging.

Fix: Two-phase commit:

```
// Phase 1: Create pending transaction (fast)
const txId = await createPendingTransaction(...);

// Phase 2: Initiate payment OUTSIDE transaction
try {
  const result = await initiateRemittance(...);
  await updateTransactionStatus(txId, result.status);
} catch (error) {
  await updateTransactionStatus(txId, 'failed');
}
```

3.3 TEXT Timestamps Everywhere

All 30+ timestamp columns use TEXT storing ISO 8601 strings. No timezone awareness, no native date math, ISO 20022 non-compliant.

Fix: Migrate to TIMESTAMPTZ in PostgreSQL migration.

3.4 No Database Connection Management

PostgreSQL pool has no timeout configuration. One slow query exhausts entire pool → cascading failure.

Fix:

```
const pool = new Pool({
  connectionString: process.env.DATABASE_URL,
  max: 20,
  connectionTimeoutMillis: 5000,
  idleTimeoutMillis: 30000,
  statement_timeout: 30000,
  query_timeout: 30000,
});
```

3.5 Zero Transaction Monitoring

AML monitoring code exists but is never called from any API route. Hvitvaskingsloven §7 requires real-time monitoring.

Fix: Wire `checkTransaction()` before PISP initiation. Block critical alerts.

3.6 Missing Open Banking Consent Lifecycle

No `ob_consents` table. Can't enforce 90-day consent renewal, access frequency limits, or consent revocation.

Fix: Create `ob_consents` table with consent tracking, expiry, access counting.

---

4. Strategic Improvements (Scale)

4.1 Network Topology & Failure Modes

No documented failure scenarios or retry strategies. Need failure domain mapping for BankID, PISP, PostgreSQL.

4.2 No Double-Entry Bookkeeping

Single-entry transaction model. Finanstilsynet will ask for general ledger proving credits = debits.

Fix: Add `transaction_legs` table with debit/credit entries per transaction.

4.3 Missing Composite Indexes

High-frequency queries will full-table-scan at scale.

Fix: Add composite indexes: `(user_id, created_at DESC)` on transactions, audit_log, notifications.

4.4 No Table Partitioning Strategy

`audit_log` will reach 90M rows in 5 years. Plan partitioning by month.

---

5. Architecture Smell Flags

5.1 Rate Limiting Writes to PostgreSQL

Every request writes to PostgreSQL for rate limiting. Should use in-memory (Map/Redis).

5.2 Fire-and-Forget Audit Logging

Critical actions (login, payment, consent) use async audit — should be synchronous for regulatory compliance.

5.3 No Circuit Breaker for External APIs

If PISP API goes down, every request waits 30s then fails. Need circuit breaker (fail fast after 5 failures, retry after 60s).

---

6. Questions Before Sign-Off

Security & Auth

1. JWT key rotation process?
2. BankID downtime fallback?
3. Session hijacking detection (2 IPs simultaneously)?
4. Direct database access logging?

Data Integrity

5. DST transition handling with TEXT timestamps?
6. Exchange rate locking between quote and execution?
7. Transaction status if PISP call times out?
8. Client-generated idempotency key security?

Compliance

9. How to enforce 90-day AISP consent expiry?
10. Who reviews AML alerts and how fast?
11. STR filing integration with Økokrim?
12. GDPR data export time for user request?

Operations

13. Database backup RPO/RTO?
14. Zero-downtime schema migration strategy?
15. PagerDuty alert rules?
16. AWS eu-north-1 failover plan?

---

7. Priority Roadmap

Period	Focus	Priority
Month 1-2	Security hardening (auth, CSRF, PISP fix, DB timeouts, AML wiring)	CRITICAL
Month 3-4	Data integrity (timestamps, ob_consents, double-entry, indexes)	HIGH
Month 5-6	Operational resilience (circuit breakers, Multi-AZ, DR runbook)	HIGH
Month 7+	Scale preparation (Redis rate limit, read replicas, monitoring)	MEDIUM

Final Verdict

> "Fintech is not about having fancy architecture diagrams — it's about operational discipline. Every timeout needs a value. Every transaction needs an audit log. Every external call needs a circuit breaker. These aren't optional extras — they're the difference between a production system and a demo. You're 60% there. The remaining 40% is where most fintechs fail."

— Petter Graff, CTO Pratexo

Revision #6
Created 2026-02-23 11:29:29 UTC by John
Updated 2026-05-23 10:51:09 UTC by John