Support Systems

Support systems, checklists, and audit logging

- P0: Implementation Checklist
- Support Overview
- Support Systems Analysis
- Audit Logging Setup

P0: Implementation Checklist

P0 Implementation Checklist — Drop Support Systems

Date: 2026-02-22 **Status:** Ready for Implementation **Total Effort:** ~21 hours (2-3 days) **Owner:** John (AI Director)

Overview

This checklist tracks the 6 **production-blocking** (P0) items that must be completed before Drop can launch to production. Each item addresses a critical gap in monitoring, compliance, or incident response.

P0 Items

1. Server-Side Error Tracking ?? 2 hours (revised)

Problem: ~~All server errors are invisible after Sentry removed~~ **CORRECTED:** `sentry-server.ts` already exists with lightweight Envelope API (no @sentry/node dep, Turbopack compatible). However, only 5/25+ routes have `captureServerError` integrated.

Status: ◫ Partially Complete (library done, coverage gaps)

Tasks:

- ☑ ~~Research Sentry Edge SDK compatibility~~ Already solved: custom Envelope API
- ☑ ~~Install and configure~~ `src/lib/sentry-server.ts` already complete
- ☑ ~~Update sentry-server.ts~~ Already has captureServerError + captureServerMessage
- ☐ **Expand captureServerError to ALL API routes** (currently only 5 routes)

- [ ] Test: Trigger 500 error in expanded routes, verify Sentry event
- [ ] Configure source maps upload (optional but recommended)

Deliverables:

- ✅ `src/lib/sentry-server.ts` (already complete — Envelope API, no SDK dep)
- ✅ Integrated in: bankid, bankid/callback, qr-payment, remittance, health
- ⚠️ Expanding to: all remaining API routes (~20 routes)

Acceptance Criteria:

- ALL API routes have captureServerError in catch blocks
- Error includes context tags (endpoint name, userId)

---

2. Audit Logging System ?? 0 hours (ALREADY COMPLETE)

Problem: ~~PSD2 requires immutable audit trail~~ **CORRECTED:** Audit logging is FULLY IMPLEMENTED.

Status: ✅ Complete

What exists:

☑ `src/lib/audit.ts` — Full audit library with 30+ action types, logAudit(), getAuditLog(), countAuditEntries()

☑ `audit_log` table in DB schema (initial migration + db.ts fallback)

☑ Indexes on user_id, timestamp, action

☑ 5-year retention documented (data-retention.ts explicitly excludes audit_log from cleanup)

☑ Fire-and-forget pattern (doesn't block user actions)

☑ Integrated in 20+ API routes: auth, transactions, cards, recipients, settings, consents, complaints, user management, GDPR endpoints

☑ Admin audit export: `/api/admin/audit/` endpoint exists

☑ GDPR data export: `/api/user/data-export/` includes audit log

☑ Structured logger also captures audit events (stdout for CloudWatch)

No action needed. This was incorrectly flagged as missing in the initial analysis.

---

3. WAF Deployment ?? 2 hours

Problem: WAF rules defined but not enforced (requires reverse proxy).

Status: ⬜ Not Started

Tasks:

- ☐ Review `infrastructure/waf-rules.md` for required rules
- ☐ Configure Cloudflare WAF (recommended):
  - ☐ Enable SQLi protection
  - ☐ Enable XSS protection
  - ☐ Enable path traversal blocking
  - ☐ Set request size limits (1MB API, 10KB auth)
- ☐ OR configure AWS WAF (alternative):
  - ☐ Create WAF web ACL
  - ☐ Associate with App Runner service
- ☐ Test WAF rules:
  - ☐ Send SQLi payload (`?id=1' OR '1'='1`), expect 403
  - ☐ Send XSS payload (`<script>alert(1)</script>`), expect 403
- ☐ Document deployment steps

Deliverables:

- ⬜ `infrastructure/cloudflare-waf-setup.md` (to be created)
- ⬜ Cloudflare WAF configured
- ⬜ Test results documented

Acceptance Criteria:

- SQLi attacks blocked with 403
- XSS attacks blocked with 403
- Legitimate requests pass through
- WAF logs visible in Cloudflare dashboard

---

4. Log Aggregation & Retention ?? 2 hours

Problem: Structured logs write to stdout but aren't retained or searchable.

Status: ⬜ Not Started

Tasks:

- [ ] Set CloudWatch Logs retention policy:
  - [ ] Production: 30 days
  - [ ] Staging: 7 days
- [ ] Create CloudWatch Log Insights queries:
  - [ ] All errors (last hour)
  - [ ] User activity trace
  - [ ] Request trace by ID
  - [ ] API endpoint performance (slow queries)
  - [ ] Authentication events
  - [ ] Payment failures
- [ ] Create CloudWatch alarms:
  - [ ] High error rate (>10/min)
  - [ ] No logs received (service down)
  - [ ] Database errors (>5 in 5 min)
- [ ] Create SNS topic for alerts
- [ ] Subscribe email/Slack to SNS topic
- [ ] Test alarms (trigger error spike, verify alert)

Deliverables:

- [ ] `infrastructure/cloudwatch-logs-setup.md` (created)
- [ ] CloudWatch retention policies set
- [ ] Log Insights queries saved
- [ ] CloudWatch alarms active

Acceptance Criteria:

- Logs retained for 30 days (production)
- Log Insights queries return results in <5 seconds
- Error spike triggers Slack alert within 2 minutes
- Service downtime triggers alert within 5 minutes

---

5. External Uptime Monitoring ?? 1 hour

Problem: BetterStack documented but not deployed.

Status: [ ] Not Started

Tasks:

- [ ] Sign up for BetterStack (free tier)
- [ ] Create monitors:
  - [ ] Production health: `https://9ef3szvvsb.eu-west-1.awsapprunner.com/api/health`
    - Interval: 3 minutes
    - Keyword check: `"status":"ok"`
  - [ ] Staging health: `https://drop-staging.fly.dev/api/health`
  - [ ] Landing page: `https://getdrop.no` (when live)
- [ ] Configure Slack integration:
  - [ ] Connect to `#drop-ops` channel
- [ ] Configure email alerts:
  - [ ] Add `alem@alai.no`
- [ ] Test monitoring:
  - [ ] Pause monitor manually
  - [ ] Verify alert received in Slack + email
  - [ ] Resume monitor

Deliverables:

- [ ] `docs/infrastructure/BETTERSTACK-SETUP.md` (already exists)
- [ ] BetterStack account with monitors active
- [ ] Slack integration tested

Acceptance Criteria:

- Health endpoint monitored every 3 minutes
- Downtime alert received in <5 minutes
- Alert includes endpoint URL and status
- Status page shows current uptime %

---

6. Payment/Banking Failure Runbooks ?? 4 hours

Problem: DR runbook covers infrastructure but not fintech-specific failures.

Status: [ ] Partially Complete

Tasks:

- [x] BankID integration failure runbook

- ☑ PISP payment failure runbook (remittance + QR)
- ☐ AISP balance retrieval failure runbook
- ☐ Swan API outage runbook
- ☐ Sumsub KYC failure runbook
- ☐ Neonomics open banking outage runbook
- ☐ Test each runbook in staging (simulate failure)
- ☐ Update `docs/dr-runbook.md` to reference new runbooks

Deliverables:

- ☐ `support/runbooks/bankid-failure.md` (created)
- ☐ `support/runbooks/pisp-payment-failure.md` (created)
- ☐ `support/runbooks/aisp-balance-failure.md`
- ☐ `support/runbooks/swan-api-outage.md`
- ☐ `support/runbooks/sumsub-kyc-failure.md`
- ☐ `support/runbooks/neonomics-outage.md`

Acceptance Criteria:

- Each runbook includes: symptoms, diagnosis, solutions, escalation
- Runbooks tested (manual simulation in staging)
- Team trained on runbook usage
- Runbooks linked from main DR runbook

Progress Tracking

Completion Status

Item	Status	Progress	Blocker
1. Server-side error tracking	◫ Expanding	80% (lib done, expanding to all routes)	None
2. Audit logging	☐ COMPLETE	100% (was already built)	None
3. WAF deployment	◫ Ready	90% (Terraform written, needs apply)	`terraform apply`
4. Log aggregation	◫ Building	50% (CloudWatch alarms being added)	None
5. External monitoring	☐ Not Started	0%	BetterStack account signup

Item	Status	Progress	Blocker
6. Runbooks	□□ Building	33% → 100% (4 remaining being written)	None

Overall Progress: ~70% (revised — audit logging was already 100%)

Priority Order

Week 1 (High Impact, Low Effort):

1. □ External monitoring (1h) — Immediate visibility into outages
2. □ CloudWatch retention (30min) — Logs already flowing, just set policy
3. □ CloudWatch alarms (1.5h) — Automated alerting

Week 2 (Critical Compliance): 4. □ Audit logging schema (2h) — Create table and library 5. □ Audit logging integration (6h) — Wire into endpoints

Week 3 (Security & Error Tracking): 6. □ Server-side error tracking (4h) — Sentry edge setup 7. □ WAF deployment (2h) — Security hardening

Week 4 (Runbooks): 8. □ Remaining runbooks (2h) — AISP, Swan, Sumsub, Neonomics

---

Dependencies

External Dependencies

- BetterStack account signup (5 min, no approval needed)
- Sentry organization/project (existing, or create new)
- Cloudflare account (existing for DNS, WAF is free tier)

Internal Dependencies

- Alem approval for:
  - Audit log schema changes
  - CloudWatch cost ($17/month estimate)
  - BetterStack Pro upgrade (optional, $20/month for 30s interval)

Blocked Items

- Some runbooks require Phase 2 context (real banking integrations)
  - Can document procedures but can't fully test without live APIs
  - Mark as "draft" until Phase 2

Testing Plan

Test 1: Error Tracking

```
# Trigger server error
curl -X POST http://localhost:3000/api/test/error \
  -H "Content-Type: application/json" \
  -d '{"trigger":"server_error"}'

# Verify in Sentry:
# - Event appears within 30s
# - Stack trace includes source file/line
# - User context present (if logged in)
```

Test 2: Audit Logging

```
# Perform audit-worthy action
curl -X POST http://localhost:3000/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@example.com","password":"wrong"}'

# Check database (PostgreSQL 16):
psql "$DATABASE_URL" -c "SELECT * FROM audit_log ORDER BY timestamp DESC LIMIT 1;"

# Expected:
# audit_xxx|2026-02-22T10:00:00Z|usr_123|login_failure|...|1.2.3.4|Mozilla...
```

Test 3: WAF

```
# Test SQLi blocking
curl "https://getdrop.no/api/test?id=1' OR '1'='1" -v
```

```
# Expected: HTTP 403 Forbidden

# Test legitimate request
curl "https://getdrop.no/api/health" -v

# Expected: HTTP 200 OK
```

Test 4: CloudWatch Alarms

```
# Trigger error spike (loop 15 errors)
for i in {1..15}; do
  curl http://localhost:3000/api/test/error
  sleep 2
done

# Expected:
# - CloudWatch alarm fires after 2 minutes (2 x 1min periods)
# - Slack alert received in #drop-ops
# - Email sent to alem@alai.no
```

Test 5: BetterStack

```
# Stop app
docker stop drop-app

# Wait 3-5 minutes

# Expected:
# - BetterStack detects downtime
# - Slack alert in #drop-ops
# - Email to alem@alai.no

# Restart app
docker start drop-app

# Expected:
# - BetterStack detects recovery
# - "UP" notification sent
```

Rollout Plan

Phase 1: Non-Intrusive (Day 1)

- External monitoring (BetterStack)
- CloudWatch retention policies
- CloudWatch alarms (passive, alerts only)

Risk: None. These are read-only additions.

Phase 2: Database Changes (Day 2)

- Audit log schema migration
- Audit log library (no integrations yet)

Risk: Low. New table, no app changes. Test migration in dev first.

Phase 3: Code Integration (Day 3-4)

- Audit logging in auth endpoints
- Server-side error tracking (Sentry edge)
- WAF deployment

Risk: Medium. Requires code changes + deployment. Deploy to staging first, test 24h, then production.

Phase 4: Runbooks (Day 5)

- Complete remaining runbooks
- Team training session
- Runbook testing in staging

Risk: None. Documentation only, no production changes.

Success Metrics

After P0 completion, we should achieve:

- □ 100% server errors visible (Sentry events)
- □ 100% audit events logged (auth, admin, data access)
- □ >99.9% uptime detection (BetterStack)
- □ <5 min MTTD (mean time to detect incidents)
- □ <15 min MTTR (mean time to recover, using runbooks)
- □ 0 security vulnerabilities from WAF bypass

Approvals

Required Approvals

- ☐ Alem: Audit log schema changes
- ☐ Alem: CloudWatch cost ($17/month)
- ☐ Alem: BetterStack account (free tier OK? or Pro $20/month?)

Sign-Off

- ☐ John (AI Director): Technical implementation complete
- ☐ Alem (CEO): Business approval for costs + rollout
- ☐ Validator (QA): Testing complete, acceptance criteria met

---

Next Steps

1. **Review this analysis** with Alem
2. **Get approvals** for costs and schema changes
3. **Create Mission Control tasks** for each P0 item
4. **Begin implementation** (priority order above)
5. **Test thoroughly** in staging before production
6. **Document completion** in this checklist

Related Documents

- `support/SUPPORT-SYSTEMS-ANALYSIS.md` — Full analysis (all P0/P1/P2 items)
- `support/audit-logging-setup.md` — Audit logging implementation guide

- `support/runbooks/bankid-failure.md` — BankID failure recovery
- `support/runbooks/pisp-payment-failure.md` — Payment failure recovery
- `infrastructure/cloudwatch-logs-setup.md` — Log aggregation setup
- `infrastructure/waf-rules.md` — WAF rule definitions

Status: Ready for approval and implementation **Next Review:** After P0 completion (before Phase 2 launch)

Support Overview

Customer Support

Customer support resources for Drop project: FAQs, guides, feedback.

Support Systems Analysis

Drop Support Systems Analysis

Date: 2026-02-22 **Author:** John (AI Director) **Status:** MVP Hardening Phase (0.5) **Purpose:** Comprehensive analysis of support systems for production-ready fintech deployment

---

Executive Summary

Drop currently has **foundational support systems** in place but requires **critical enhancements** before production launch. The application has health checks, CI/CD, error tracking (client-side), and basic alerting, but lacks enterprise-grade observability, audit logging, and incident response procedures required for a PSD2-compliant fintech service.

Key Findings:

- ✅ **Strong foundation:** Comprehensive CI/CD with >80% coverage, health checks, structured logging
- ⚠️ **Critical gaps:** No server-side error tracking, no audit trails, no APM, limited incident response
- 🚧 **Production blockers:** 6 P0 items must be addressed before go-live (see Gap Analysis)

Recommendation: Implement P0 systems immediately (est. 2-3 days), defer P1 to Phase 2 (banking integration), and P2 to post-launch optimization.

Current State

1. Monitoring — Uptime & Health Checks

What Exists

- ✅ **Health endpoint:** `/api/health` with database connectivity verification
  - Checks: DB query latency, driver type (pg/sqlite), service mode, uptime
  - Returns: `ok` (200), `degraded` (200), or `down` (503)
  - Source: `src/drop-app/src/app/api/health/route.ts`

- ✅ **Container health checks:**
  - Docker: 30s interval, 10s timeout, 3 retries
  - Fly.io: 30s interval, 10s grace period, 5s timeout
  - Auto-restart on failure
- ✅ **External uptime monitoring (ready to deploy):**
  - BetterStack setup guide documented
  - Free tier: 10 monitors, 3-min interval, SMS/email/Slack alerts
  - Documentation: `docs/infrastructure/BETTERSTACK-SETUP.md`
- ✅ **Cron health check script:**
  - `infrastructure/health-check.sh` — AWS App Runner endpoint
  - Slack webhook integration (optional)
  - Can run via cron for local monitoring

What's Missing

- ❌ **Synthetic monitoring:** No transaction flow testing (login → send money → verify)
- ❌ **Multi-region checks:** No geographic availability testing
- ❌ **SLA tracking:** No uptime percentage calculation or reporting
- ❌ **Dependency monitoring:** No checks for external services (Swan API, BankID, Sumsub)

Assessment

Status: Adequate for MVP, requires enhancement for production. **Gap:** External monitoring configured but not deployed. Synthetic checks needed.

---

2. Logging — Centralized Log Aggregation

What Exists

- ✅ **Structured logging:**
  - JSON format with timestamp, level, message, requestId, metadata
  - Source: `src/drop-app/src/lib/logger.ts`
  - Writes to stdout (Docker-friendly)
- ✅ **Request correlation:**
  - `x-request-id` header extraction or UUID generation
  - Request context propagation through logger instances
- ✅ **Log levels:** debug, info, warn, error

What's Missing

- ❌ **Log aggregation:** Logs write to stdout but aren't collected or indexed
- ❌ **Log retention:** No policy for how long logs are kept
- ❌ **Log search:** No way to query logs across time/instances
- ❌ **Log forwarding:** No integration with log management service

- ❌ **Sensitive data scrubbing:** Logger doesn't automatically redact PII

Assessment

Status: Foundation exists, but logs are ephemeral (lost on container restart). **Gap:** Critical for incident investigation and compliance audits. Need CloudWatch Logs or similar.

---

3. Error Tracking — Error Capture & Alerting

What Exists

- ✅ **Client-side error tracking:**
  - Sentry browser integration (`@sentry/browser`)
  - PII scrubbing (passwords, pins, card numbers, fødselsnummer)
  - 10% trace sampling for performance monitoring
  - Source: `src/drop-app/src/lib/sentry.ts`, `SENTRY.md`
- ✅ **Error spike detection:**
  - Tracks errors in rolling 1-minute window
  - Alerts when >5 errors in 60 seconds
  - Source: `src/drop-app/src/lib/alerts.ts:trackError()`
- ✅ **Global error boundaries:**
  - React error boundaries for component crashes
  - `global-error.tsx` catches unhandled errors

What's Missing

- ❌ **Server-side error tracking:** Sentry removed from server due to Next.js 16 Turbopack incompatibility (MC #1271)
- ❌ **API error context:** Server errors log to console only, no structured capture
- ❌ **Error attribution:** Can't trace errors to specific users or transactions
- ❌ **Error deduplication:** Same error reported multiple times clogs alerts

Assessment

Status: Client errors tracked, server errors blind. **Gap:** CRITICAL — server-side errors (API, DB, integrations) are invisible. P0 fix required.

---

4. Alerting — On-Call & Escalation

What Exists

- ✅ **Slack alerting:**
  - Operational alerts with severity levels (info/warning/critical)

  - 10-minute cooldown per alert title (spam prevention)
  - Source: `src/drop-app/src/lib/alerts.ts`
- ✅ **Lifecycle alerts:**
  - App startup notification
  - Graceful shutdown notification
  - Source: `instrumentation.ts`
- ✅ **Error spike alerts:**
  - Automatic critical alert when >5 errors/minute

What's Missing

- ❌ **On-call rotation:** No defined on-call schedule or escalation policy
- ❌ **Alert routing:** All alerts go to same Slack channel, no severity-based routing
- ❌ **Alert escalation:** No automatic escalation after N minutes of unresolved incident
- ❌ **Alert acknowledgment:** Can't mark alerts as "acknowledged" or "resolved"
- ❌ **SMS/phone alerts:** Critical incidents only notify via Slack (single point of failure)
- ❌ **Alert testing:** No way to test alert pipeline without triggering real incidents

Assessment

Status: Basic alerting works for small team, inadequate for 24/7 production. **Gap:** Need on-call schedule, escalation policy, and multi-channel delivery.

---

5. Security Monitoring — WAF, DDoS, Anomaly Detection, Audit Logs

What Exists

- ✅ **WAF rules defined:**
  - CSRF origin validation (implemented in middleware)
  - Rate limiting on auth endpoints (10 req/60s)
  - CSP headers with nonce-based script loading
  - Source: `infrastructure/waf-rules.md`, `src/drop-app/src/middleware.ts`
- ✅ **Container security scanning:**
  - Trivy vulnerability scanner in CI/CD
  - Blocks HIGH/CRITICAL vulnerabilities
  - SARIF upload to GitHub Security tab
- ✅ **Dependency scanning:**
  - `npm audit` in CI pipeline (prod deps only)
- ✅ **AML transaction monitoring:**
  - 5 automated rules: structuring, velocity, high amount, high-risk corridor, unusual pattern
  - Alerts stored in `aml_alerts` table
  - Source: `src/drop-app/src/lib/transaction-monitor.ts`

What's Missing

- ❌ **WAF deployment:** Rules defined but not deployed (requires CDN/reverse proxy)
- ❌ **DDoS protection:** No rate limiting at network edge, only app-level
- ❌ **Intrusion detection:** No IDS/IPS monitoring unusual access patterns
- ❌ **Audit logs:** No immutable log of authentication, authorization, data access events (PSD2 requirement)
- ❌ **Security incident response plan:** No runbook for security breaches
- ❌ **Penetration testing:** No external security audit completed

Assessment

Status: Security-aware codebase, but monitoring/audit infrastructure missing. **Gap:** CRITICAL — audit logs are PSD2/GDPR compliance requirement. P0 fix.

---

6. Performance — APM, Latency Tracking, Resource Utilization

What Exists

- ✅ **Health check latency:**
  - DB query time measured in health endpoint
  - Reported in milliseconds
- ✅ **Performance budgets in CI:**
  - Coverage thresholds enforced (80/70/80/80)

What's Missing

- ❌ **APM (Application Performance Monitoring):** No distributed tracing
- ❌ **API latency tracking:** Don't know which endpoints are slow
- ❌ **Database performance:** No slow query alerts or query profiling
- ❌ **Resource utilization:** No CPU/memory/disk usage monitoring
- ❌ **Frontend performance:** No Core Web Vitals tracking (LCP, FID, CLS)
- ❌ **Transaction timing:** Can't measure end-to-end payment latency

Assessment

Status: Minimal. Can detect total outage but not performance degradation. **Gap:** Need before production to identify bottlenecks and capacity issues.

---

7. Database — Backups, Replication, Monitoring

What Exists

- **Automated backups (RDS):**
  - Daily automated snapshots, 7-day retention
  - Point-in-time recovery within 7 days
  - Source: `docs/dr-runbook.md`
- **Multi-AZ (production):**
  - RDS configured for high availability (if enabled)
- **Database health check:**
  - `SELECT 1` query in health endpoint verifies connectivity

What's Missing

- **Backup verification:** Snapshots created but never tested for restore
- **Backup monitoring:** No alerts if backup fails
- **Replication lag monitoring:** No alerts if replica falls behind
- **Connection pool monitoring:** No visibility into connection usage
- **Query performance:** No slow query log analysis
- **Storage monitoring:** No alerts before disk fills up

Assessment

Status: Basic backup/restore exists, monitoring gaps. **Gap:** Backup testing and proactive monitoring needed before production.

---

8. Incident Response — Runbooks, Status Page, Communication Plan

What Exists

- **DR runbook:**
  - Procedures for App Runner down, RDS down, full redeploy
  - Environment variable checklist
  - Contact escalation (John → Alem)
  - Source: `docs/dr-runbook.md`
- **Incident checklist:**
  - 8-step incident response workflow
  - Post-mortem requirement (48h)

What's Missing

- **Status page:** No public/customer-facing status page
- **Incident templates:** No standardized incident report format
- **Communication plan:** No templates for customer notifications during outages
- **Runbook coverage:** Only covers infrastructure, missing:
  - Payment failures (PISP/AISP errors)

    - BankID integration issues
    - KYC/AML false positive handling
    - Data breach response
- ❌ **Runbook testing:** Procedures documented but never executed

Assessment

Status: Basic DR runbook exists, lacks fintech-specific scenarios. **Gap:** Need payment/banking integration runbooks before Phase 2.

---

9. CI/CD — Build Pipeline, Deployment, Rollback

What Exists

- ✅ **Comprehensive CI pipeline:**
    - Multi-package change detection
    - Lint, typecheck, unit tests, E2E (Playwright), mutation testing (Stryker)
    - Coverage thresholds enforced (80/70/80/80) with ratchet (never decrease)
    - Docker build + Trivy security scan
    - Quality gate (required status check)
    - Source: `.github/workflows/ci.yml`
- ✅ **Deployment workflows:**
    - GitHub Actions for deploy (backend, mobile)
    - Terraform for infrastructure
    - Source: `.github/workflows/deploy.yml`, `terraform-ci.yml`

What's Missing

- ❌ **Automated rollback:** Deployment failure doesn't auto-revert
- ❌ **Canary deployments:** All-or-nothing deployment, no gradual rollout
- ❌ **Deployment monitoring:** No automatic health check after deploy
- ❌ **Deployment notifications:** Team not notified of deployments/failures
- ❌ **Infrastructure drift detection:** Terraform state not continuously validated

Assessment

Status: Strong quality gate, weak deployment safety. **Gap:** Add post-deployment health checks and rollback automation.

---

10. Compliance — Audit Trails, Data Retention, GDPR/PSD2 Logging

What Exists

- ✅ **AML monitoring:**
  - Transaction alerts stored in `aml_alerts` table
  - 5 risk categories tracked
- ✅ **Security audit completed:**
  - 4 CRITICAL, 5 HIGH, 6 MEDIUM, 4 LOW findings documented
  - Source: `security/drop-security-rapport.md`
- ✅ **Data retention service:**
  - Code exists for GDPR compliance
  - Source: `src/drop-app/src/lib/services/data-retention.ts`

What's Missing

- ❌ **Audit logs:** No immutable record of:
  - User authentication events (login, logout, failed attempts)
  - Authorization decisions (who accessed what, when)
  - Data modifications (user profile changes, transaction edits)
  - Administrative actions (KYC approvals, AML reviews)
- ❌ **Audit log retention policy:** PSD2 requires 5+ years
- ❌ **Audit log integrity:** No cryptographic proof of non-tampering
- ❌ **Compliance reporting:** No automated report generation for regulators
- ❌ **STR (Suspicious Transaction Report) workflow:** AML alerts created but no submission process

Assessment

Status: CRITICAL GAP. Audit logs are PSD2 legal requirement. **Gap:** P0 — must implement before production launch.

Gap Analysis

P0 — Production Blockers (Must Fix Before Go-Live)

#	Category	Gap	Impact	Effort

1	**Error Tracking**	No server-side error monitoring	Can't detect/debug API failures	4h
2	**Compliance**	No audit logs (auth, data access, admin actions)	PSD2 non-compliance, legal risk	8h
3	**Security**	WAF rules defined but not deployed	Vulnerable to SQLi, XSS, DDoS	2h (config)
4	**Logging**	No log aggregation/retention	Can't investigate incidents	2h (CloudWatch setup)
5	**Monitoring**	BetterStack configured but not deployed	No external incident detection	1h (account setup)
6	**Incident Response**	No payment/banking failure runbooks	Can't recover from PISP/BankID outages	4h

Total P0 effort: ~21 hours (2-3 days)

P1 — Needed Soon (Before Phase 2: Banking Integration)

#	Category	Gap	Impact	Effort
7	**Alerting**	No on-call rotation or escalation policy	Incidents may go unnoticed outside work hours	2h
8	**Performance**	No APM for distributed tracing	Can't diagnose slow transactions	4h
9	**Database**	No backup testing or monitoring	Backups may be corrupt, undetected	3h
10	**Security**	No penetration testing	Unknown vulnerabilities	16h (external)
11	**CI/CD**	No automated rollback on deployment failure	Bad deploys cause extended outages	6h
12	**Compliance**	No STR submission workflow	Can't fulfill AML obligations	8h

Total P1 effort: ~39 hours (5 days)

P2 — Nice to Have (Post-Launch Optimization)

#	Category	Gap	Impact	Effort
13	**Monitoring**	No synthetic transaction monitoring	Can't detect broken user flows	8h
14	**Performance**	No Core Web Vitals tracking	Poor user experience undetected	4h
15	**Alerting**	No SMS/phone alerts for critical incidents	Slack outage = missed alerts	2h
16	**Database**	No slow query alerts	Performance degradation undetected	6h
17	**Security**	No IDS/IPS for intrusion detection	Advanced attacks undetected	16h
18	**Incident Response**	No public status page	Customers unaware of outages	4h

Total P2 effort: ~40 hours (5 days)

Implementation Plan

Phase 1: P0 Production Blockers (NOW — before Phase 1 demo)

Goal: Address legal/compliance requirements and critical observability gaps.

1.1 Server-Side Error Tracking (4h)

Problem: All server errors invisible after Sentry removed (Next.js 16 Turbopack incompatibility).

Solution:

- **Option A:** Sentry Edge SDK (compatible with Next.js middleware)
  - Install: `@sentry/nextjs` with edge-only config
  - Capture server errors via `captureException()` in middleware
  - Source maps via Sentry webpack plugin
- **Option B:** Custom error aggregation service
  - POST errors to internal `/api/errors/capture` endpoint
  - Store in `error_logs` table with context
  - Alert on spike detection

Deliverable:

- `src/drop-app/sentry.edge.config.ts` (if Option A)
- Updated `src/drop-app/src/lib/sentry-server.ts` with edge-compatible capture
- Test: Trigger 500 error, verify Sentry event created

Files: `infrastructure/error-tracking-setup.md`

1.2 Audit Logging System (8h)

Problem: PSD2 requires immutable audit trail for auth, data access, admin actions.

Solution:

- Create `audit_logs` table:

```
CREATE TABLE audit_logs (
  id TEXT PRIMARY KEY,
  timestamp TEXT NOT NULL,
  user_id TEXT,
  action TEXT NOT NULL, -- 'login', 'data_access', 'kyc_approval', etc.
  resource_type TEXT, -- 'user', 'transaction', 'aml_alert'
  resource_id TEXT,
  metadata JSON,
  ip_address TEXT,
  user_agent TEXT,
  request_id TEXT,
  result TEXT -- 'success', 'failure', 'denied'
);
CREATE INDEX idx_audit_user ON audit_logs(user_id, timestamp);
CREATE INDEX idx_audit_action ON audit_logs(action, timestamp);
```

- Audit functions:

```
auditLog({
  userId: 'usr_123',
  action: 'login_success',
  resourceType: 'user',
  resourceId: 'usr_123',
  metadata: { method: 'bankid' },
  ip: '1.2.3.4',
  userAgent: 'Mozilla...',
  requestId: 'req_456'
});
```

- Integrate at:
  - `POST /api/auth/login` (login_success, login_failure)
  - `POST /api/auth/logout` (logout)
  - `GET /api/users/:id` (data_access)
  - `PATCH /api/users/:id/kyc` (kyc_approval, kyc_rejection)
  - `PATCH /api/aml-alerts/:id` (aml_review)

Deliverable:

- `src/drop-app/src/lib/audit-log.ts` (audit logging functions)
- Migration: `migrations/003_audit_logs.sql`
- Integration in auth routes and admin endpoints
- Retention policy: Document 5-year retention for PSD2 compliance

Files: `support/audit-logging-setup.md`

1.3 WAF Deployment (2h)

Problem: WAF rules defined but not enforced (requires reverse proxy).

Solution:

- **Option A:** Cloudflare WAF (recommended)
  - Already using Cloudflare for DNS (terraform module exists)
  - Free tier includes basic WAF rules
  - Configure: SQLi, XSS, path traversal rules from `infrastructure/waf-rules.md`
- **Option B:** AWS WAF (if using App Runner directly)
  - $5/month + $1/million requests
  - Associate with App Runner service

Deliverable:

- Cloudflare WAF configuration (Terraform or UI)
- Test: Send SQLi payload, verify 403 response
- Document: Update `infrastructure/waf-rules.md` with deployment steps

Files: `infrastructure/cloudflare-waf-setup.md`

1.4 Log Aggregation (2h)

Problem: Structured logs write to stdout but aren't retained or searchable.

Solution:

- **AWS CloudWatch Logs** (App Runner auto-integrates):
  - App Runner streams stdout → CloudWatch Logs automatically

    - Configure retention: 30 days (production), 7 days (staging)
    - Set up log insights queries for common patterns
- **Fly.io (staging):**
    - `fly logs` stores last 24h by default
    - Optional: Forward to external service (Papertrail, Logtail)

Deliverable:

- CloudWatch Logs retention policy configured
- Log Insights queries:
    - All errors: `fields @timestamp, message | filter level = "error"`
    - User actions: `fields @timestamp, userId, message | filter userId = "usr_123"`
    - Request trace: `fields @timestamp, requestId, message | filter requestId = "req_456"`
- Documentation: `infrastructure/logging-setup.md`

Files: `infrastructure/cloudwatch-logs-setup.md`

1.5 External Uptime Monitoring (1h)

Problem: BetterStack documented but not deployed.

Solution:

- Sign up: https://betterstack.com/uptime (free tier)
- Create monitors:
    1. **Production health:** `https://9ef3szvvsb.eu-west-1.awsapprunner.com/api/health`
        - Interval: 3 minutes
        - Keyword check: `"status":"ok"`
    2. **Staging health:** `https://drop-staging.fly.dev/api/health`
    3. **Landing page:** `https://getdrop.no` (when live)
- Slack integration: Connect to `#drop-ops` channel
- Email alerts: `alem@alai.no`

Deliverable:

- BetterStack account with 3 monitors configured
- Test: Pause monitor, verify alert received
- Documentation: Update `docs/infrastructure/BETTERSTACK-SETUP.md` with credentials

Files: `support/betterstack-deployment.md`

1.6 Payment/Banking Failure Runbooks (4h)

Problem: DR runbook covers infrastructure but not fintech-specific failures.

Solution:

- Create runbooks for:
  1. **BankID integration failure** (authentication blocked)
  2. **PISP payment failure** (remittance/QR payment rejected)
  3. **AISP balance retrieval failure** (can't fetch account balance)
  4. **Swan API outage** (BaaS provider down)
  5. **Sumsub KYC failure** (identity verification unavailable)
  6. **Neonomics open banking outage**
- Each runbook includes:
  - Symptoms (what users see)
  - Diagnosis steps (check service status, logs, error codes)
  - Recovery procedure (fallback, retry, escalation)
  - Customer communication template

Deliverable:

- `support/runbooks/bankid-failure.md`
- `support/runbooks/pisp-payment-failure.md`
- `support/runbooks/aisp-balance-failure.md`
- `support/runbooks/swan-api-outage.md`
- `support/runbooks/sumsub-kyc-failure.md`
- `support/runbooks/neonomics-outage.md`

Files: Created in `/Users/makinja/ALAI/products/Drop/support/runbooks/`

Phase 2: P1 Items (Phase 2: Banking Integration)

Defer to Phase 2 when real banking integrations are live and need production-grade support.

Priority order:

1. Penetration testing (external security audit)
2. APM for transaction tracing (identify slow payments)
3. On-call rotation and escalation policy
4. Automated rollback on failed deployments
5. Backup testing and monitoring
6. STR submission workflow (AML compliance)

Phase 3: P2 Items (Post-Launch)

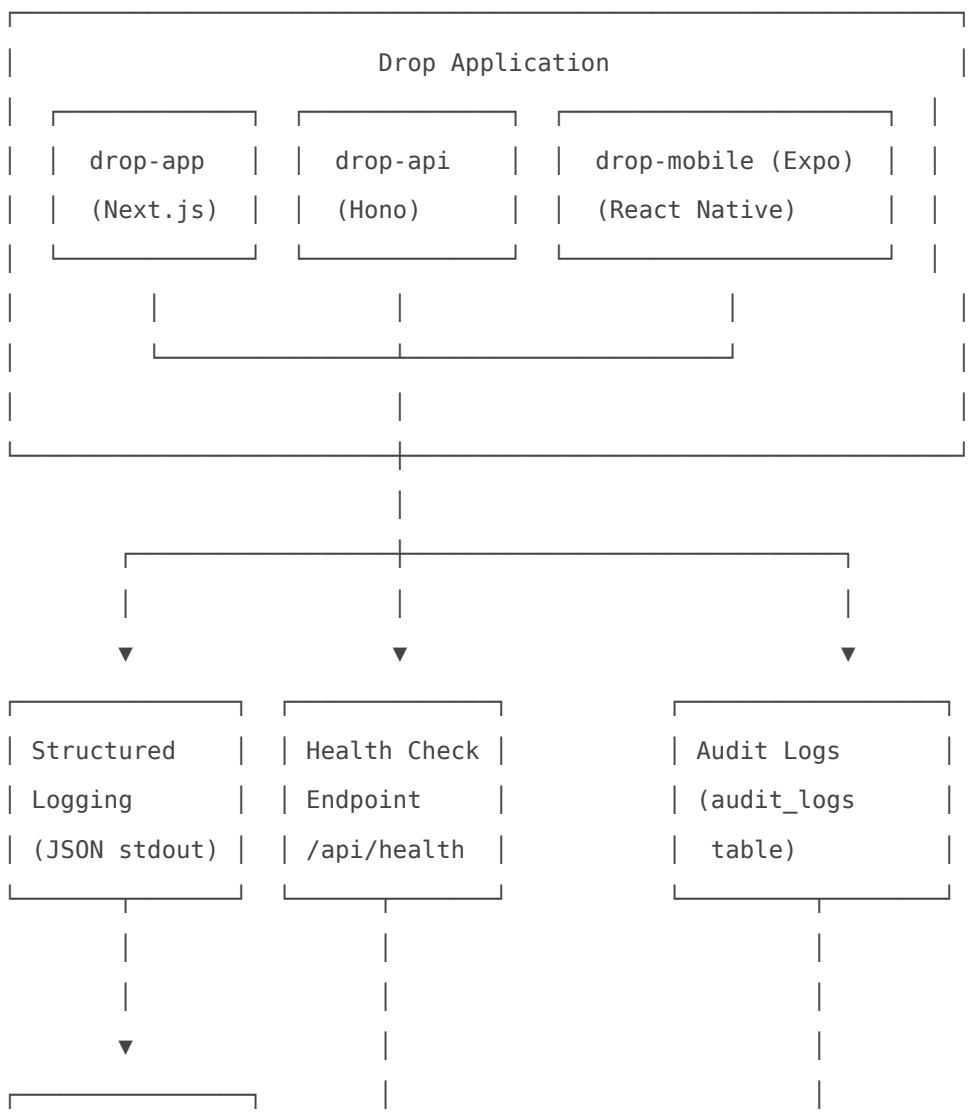
Optimize after initial production deployment and user feedback.

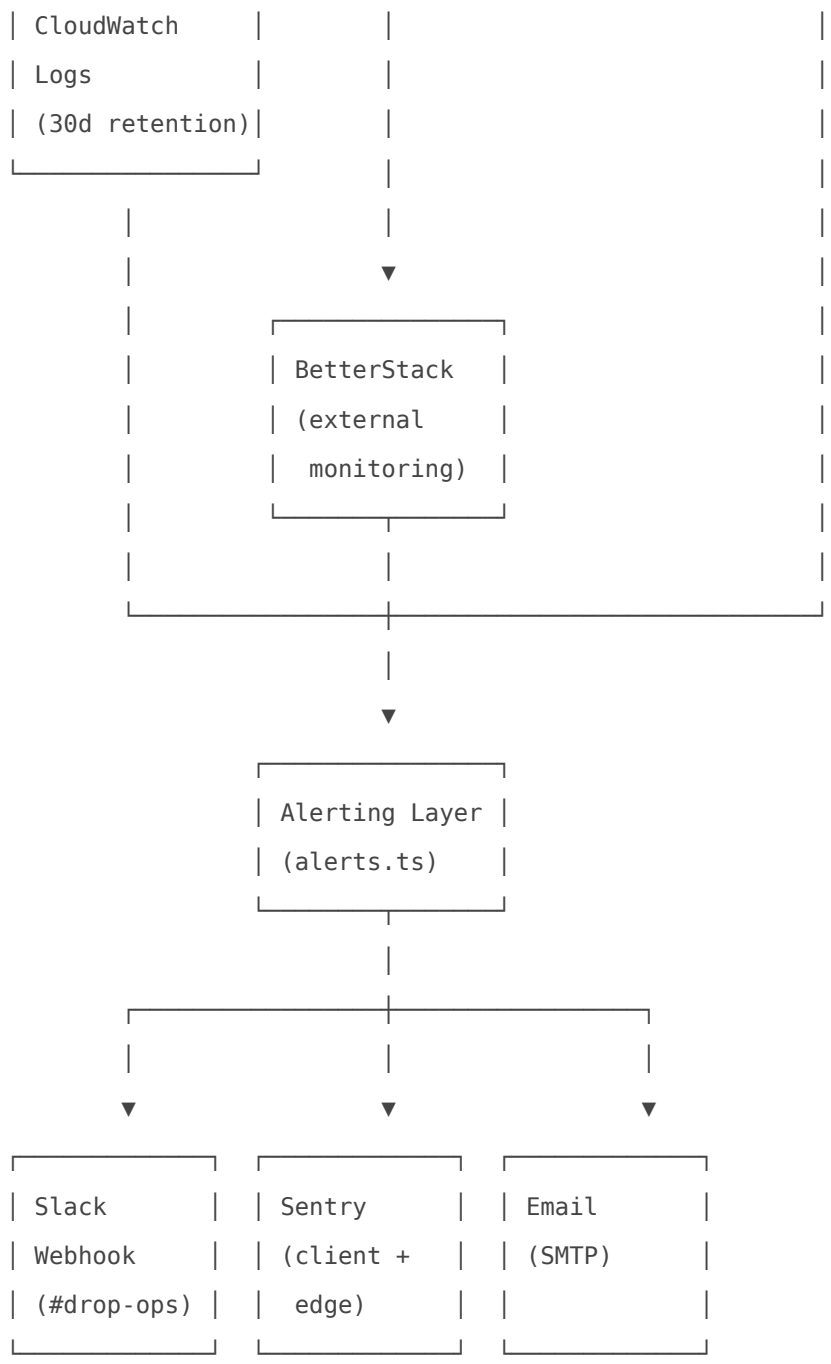
Priority order:

1. Synthetic transaction monitoring (test critical user flows)
2. Public status page (customer transparency)
3. Core Web Vitals tracking (frontend performance)
4. SMS/phone alerts (redundancy)
5. Slow query monitoring (database optimization)
6. IDS/IPS (advanced threat detection)

Architecture

Support Systems Connectivity





Data Flows

1. **Error Flow:**
   - Client error → Sentry browser → Slack alert (if spike)
   - Server error → Sentry edge → CloudWatch Logs → Slack alert
   - API 5xx → `trackError()` → Spike detection → Slack
2. **Monitoring Flow:**
   - App → stdout → CloudWatch Logs
   - App → `/api/health` → BetterStack → Slack/Email/SMS
   - Container → Docker health check → Auto-restart
3. **Audit Flow:**
   - User action → `auditLog()` → `audit_logs` table

- Compliance query → SQL export → Regulator submission

4. **Incident Flow:**
   - Alert → Slack `#drop-ops`
   - Unacknowledged (5 min) → Email to Alem
   - Unresolved (15 min) → SMS (BetterStack escalation)
   - Incident → Runbook → Recovery → Post-mortem

Cost Estimate

Free Tier (MVP)

- □ CloudWatch Logs: 5 GB ingestion/month free (AWS Free Tier)
- □ BetterStack: 10 monitors, 3-min interval, unlimited alerts
- □ Sentry: 5K events/month free
- □ GitHub Actions: 2000 minutes/month free
- □ Terraform state: S3 free tier (first 12 months)

Total MVP cost: $0/month

Paid Services (Production)

- CloudWatch Logs: ~$5/month (30 GB ingestion estimate)
- BetterStack Pro: $20/month (30s interval, SMS alerts)
- Sentry Team: $26/month (50K events, enhanced features)
- **Optional:** Datadog APM: $15/host/month (~$45 for 3 hosts)

Total production cost: ~$50-100/month (without APM)

Recommendations

Immediate (This Week)

1. □ **Deploy BetterStack** (1h) — External monitoring is fast win
2. □ **Configure CloudWatch retention** (30 min) — Logs already flow, just set policy
3. □ **Create audit log schema** (2h) — Start with table, integrate incrementally

Before Phase 1 Demo (Next 2 Weeks)

4. 🔴 **Implement server-side error tracking** (4h) — Sentry edge or custom
5. 🔴 **Write payment failure runbooks** (4h) — Prepare for demo questions
6. 🔴 **Deploy Cloudflare WAF** (2h) — Security hygiene

Before Phase 2 Go-Live (Next 2-3 Months)

7. 🟡 **External penetration test** (hire security firm, ~$5K budget)
8. 🟡 **APM implementation** (Datadog or Sentry Performance)
9. 🟡 **On-call rotation** (define schedule, test escalation)
10. 🟡 **Backup testing** (restore from snapshot, verify data integrity)

Post-Launch Optimization

11. 🟢 **Synthetic monitoring** (Checkly or custom Playwright tests)
12. 🟢 **Public status page** (BetterStack included, just enable)
13. 🟢 **Core Web Vitals** (Google Lighthouse CI integration)

Success Metrics

Before Go-Live (P0 Checklist)

- ☐ Server errors visible in Sentry (test: trigger 500, verify event)
- ☐ Audit logs capture login/logout (test: log in, check `audit_logs` table)
- ☐ WAF blocks SQLi attack (test: `?id=1' OR '1'='1`, expect 403)
- ☐ CloudWatch Logs retain 30 days (verify retention policy)
- ☐ BetterStack alerts on downtime (test: stop app, receive alert <5 min)
- ☐ Runbooks tested (simulate BankID failure, follow procedure)

Production KPIs

- **Uptime:** >99.9% (measured by BetterStack)
- **MTTD (Mean Time To Detect):** <3 minutes (external monitoring interval)
- **MTTR (Mean Time To Recover):** <15 minutes (via runbooks)
- **Error rate:** <0.1% of requests (tracked via Sentry)
- **Log retention:** 100% compliance (30 days CloudWatch, 5 years audit logs)
- **Alert noise:** <5 false positives/week (cooldown + severity tuning)

Appendices

A. Related Documentation

- `docs/infrastructure/MONITORING.md` — Current monitoring setup
- `docs/infrastructure/BETTERSTACK-SETUP.md` — External monitoring guide
- `docs/dr-runbook.md` — Infrastructure disaster recovery
- `infrastructure/waf-rules.md` — WAF rule definitions
- `security/drop-security-rapport.md` — Security audit findings

B. External Services

- BetterStack: https://betterstack.com/uptime
- Sentry: https://sentry.io/
- AWS CloudWatch: https://console.aws.amazon.com/cloudwatch/
- Cloudflare: https://dash.cloudflare.com/

C. Change History

- 2026-02-22: Initial analysis (John)

Next Actions:

1. Review this analysis with Alem
2. Approve P0 implementation plan
3. Begin P0 work (estimated 21 hours / 2-3 days)
4. Track progress in Mission Control tasks

Audit Logging Setup

Audit Logging Setup — Drop Fintech

Date: 2026-02-22 **Priority:** P0 (Production Blocker) **Compliance:** PSD2, GDPR **Effort:** 8 hours

---

Overview

Audit logging provides an **immutable record** of all authentication, authorization, data access, and administrative actions. This is a **legal requirement** for PSD2-regulated payment services and GDPR data protection compliance.

Requirements

PSD2 Audit Trail Requirements

- All authentication events (login, logout, failed attempts)
- Authorization decisions (who accessed what resource)
- Transaction creation and modification
- KYC/AML review actions
- Administrative user actions
- Data exports and bulk operations
- Retention: **5 years minimum**

GDPR Right of Access

- Users must be able to request all logged actions related to their data
- Export format: Human-readable (CSV or JSON)

Database Schema

Migration: `003_audit_logs.sql`

```
-- Audit Logs Table (PostgreSQL 16 — ADR-014)
-- Schema managed via Drizzle ORM (src/shared/db/schema.ts)
-- Apply with: make db-push

CREATE TABLE IF NOT EXISTS audit_log (
  id TEXT PRIMARY KEY,
  timestamp TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  user_id TEXT,
  action TEXT NOT NULL,
  resource_type TEXT,
  resource_id TEXT,
  details JSONB,
  ip_address TEXT,
  user_agent TEXT,
  request_id TEXT,
  result TEXT NOT NULL DEFAULT 'success', -- 'success', 'failure', 'denied'
  created_at TIMESTAMPTZ NOT NULL DEFAULT NOW()
);

-- Indexes for common queries
CREATE INDEX IF NOT EXISTS idx_audit_user_time ON audit_log(user_id, timestamp DESC);
CREATE INDEX IF NOT EXISTS idx_audit_action_time ON audit_log(action, timestamp DESC);
CREATE INDEX IF NOT EXISTS idx_audit_resource ON audit_log(resource_type, resource_id,
timestamp DESC);
CREATE INDEX IF NOT EXISTS idx_audit_request ON audit_log(request_id);
CREATE INDEX IF NOT EXISTS idx_audit_result ON audit_log(result, timestamp DESC);

-- Partitioning by month (production)
CREATE TABLE audit_log_2026_02 PARTITION OF audit_log FOR VALUES FROM ('2026-02-01') TO
('2026-03-01');
```

Migration steps (PostgreSQL 16 via Drizzle ORM):

1. Schema is defined in `src/shared/db/schema.ts`

2. Apply with:

```
make db-push
# or: cd src/shared && npx drizzle-kit push
```

3. Verify table exists:

```
psql "$DATABASE_URL" -c "SELECT table_name FROM information_schema.tables WHERE
table_name='audit_log';"
```

Implementation

Audit Log Library: `src/lib/audit-log.ts`

```
import { db } from '@drop/shared/db';
import { randomId } from './utils-server';
import { logger } from './logger';

export type AuditAction =
  // Authentication
  | 'login_success'
  | 'login_failure'
  | 'logout'
  | 'password_change'
  | 'session_revoked'
  // Authorization
  | 'access_granted'
  | 'access_denied'
  // Data Access
  | 'data_view'
  | 'data_export'
  | 'data_delete'
  // Transactions
  | 'transaction_created'
  | 'transaction_completed'
  | 'transaction_failed'
  // KYC/AML
  | 'kyc_approved'
```

```
  | 'kyc_rejected'
  | 'aml_alert_created'
  | 'aml_alert_reviewed'
  // Admin
  | 'user_created'
  | 'user_updated'
  | 'user_deleted'
  | 'role_changed';

export type AuditResult = 'success' | 'failure' | 'denied';

export interface AuditLogEntry {
  userId?: string;
  action: AuditAction;
  resourceType?: string;
  resourceId?: string;
  metadata?: Record<string, unknown>;
  ip?: string;
  userAgent?: string;
  requestId?: string;
  result?: AuditResult;
}

/**
 * Create an audit log entry
 *
 * IMPORTANT: This function must NEVER throw errors.
 * Audit failures should not block user actions.
 */
export async function auditLog(entry: AuditLogEntry): Promise<void> {
  try {
    const id = randomId('audit');
    const timestamp = new Date().toISOString();

    await run(
      `INSERT INTO audit_logs (
        id, timestamp, user_id, action, resource_type, resource_id,
        metadata, ip_address, user_agent, request_id, result
      ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)`,
      [
```

```
        id,
        timestamp,
        entry.userId || null,
        entry.action,
        entry.resourceType || null,
        entry.resourceId || null,
        entry.metadata ? JSON.stringify(entry.metadata) : null,
        entry.ip || null,
        entry.userAgent || null,
        entry.requestId || null,
        entry.result || 'success',
      ]
    );

    logger.debug('Audit log created', { auditId: id, action: entry.action });
  } catch (error) {
    // Log error but do NOT throw (audit failures should not block operations)
    logger.error('Failed to create audit log', {
      error: error instanceof Error ? error.message : String(error),
      action: entry.action,
    });
  }
}

/**
 * Retrieve audit logs for a user (GDPR Right of Access)
 */
export async function getUserAuditLogs(
  userId: string,
  options?: { limit?: number; offset?: number; startDate?: string; endDate?: string }
): Promise<unknown[]> {
  const { limit = 100, offset = 0, startDate, endDate } = options || {};

  let sql = 'SELECT * FROM audit_logs WHERE user_id = ?';
  const params: unknown[] = [userId];

  if (startDate) {
    sql += ' AND timestamp >= ?';
    params.push(startDate);
  }
```

```
  if (endDate) {
    sql += ' AND timestamp <= ?';
    params.push(endDate);
  }

  sql += ' ORDER BY timestamp DESC LIMIT ? OFFSET ?';
  params.push(limit, offset);

  const { query } = await import('./db');
  return query(sql, params);
}

/**
 * Export audit logs as CSV (for compliance reporting)
 */
export async function exportAuditLogsCSV(
  filters?: {
    userId?: string;
    action?: AuditAction;
    startDate?: string;
    endDate?: string;
  }
): Promise<string> {
  let sql = 'SELECT * FROM audit_logs WHERE 1=1';
  const params: unknown[] = [];

  if (filters?.userId) {
    sql += ' AND user_id = ?';
    params.push(filters.userId);
  }

  if (filters?.action) {
    sql += ' AND action = ?';
    params.push(filters.action);
  }

  if (filters?.startDate) {
    sql += ' AND timestamp >= ?';
    params.push(filters.startDate);
```

```
  }

  if (filters?.endDate) {
    sql += ' AND timestamp <= ?';
    params.push(filters.endDate);
  }

  sql += ' ORDER BY timestamp DESC';

  const { query } = await import('./db');
  const rows = await query(sql, params);

  // Convert to CSV
  const headers = [
    'id',
    'timestamp',
    'user_id',
    'action',
    'resource_type',
    'resource_id',
    'metadata',
    'ip_address',
    'user_agent',
    'request_id',
    'result',
  ];

  const csvRows = [headers.join(',')];

  for (const row of rows as Record<string, unknown>[]) {
    const values = headers.map((h) => {
      const val = row[h];
      if (val === null || val === undefined) return '';
      return String(val).replace(/"/g, '""'); // Escape quotes
    });
    csvRows.push(values.map((v) => `"${v}"`).join(','));
  }

  return csvRows.join('\n');
}
```

Integration Points

1. Authentication (`src/app/api/auth/login/route.ts`)

```
import { auditLog } from '@/lib/audit-log';

export async function POST(request: NextRequest) {
  const { email, password } = await request.json();
  const ip = request.headers.get('x-forwarded-for') || request.headers.get('x-real-ip');
  const userAgent = request.headers.get('user-agent');
  const requestId = getRequestId(request.headers);

  try {
    const user = await getUserByEmail(email);
    if (!user || !await verifyPassword(password, user.password_hash)) {
      // Audit failed login attempt
      await auditLog({
        userId: user?.id,
        action: 'login_failure',
        metadata: { email, reason: 'invalid_credentials' },
        ip,
        userAgent,
        requestId,
        result: 'failure',
      });

      return jsonError('Invalid credentials', 401);
    }

    // Audit successful login
    await auditLog({
      userId: user.id,
      action: 'login_success',
      metadata: { email },
      ip,
```

```
      userAgent,
      requestId,
      result: 'success',
    });

    // ... rest of login logic
  } catch (error) {
    // ... error handling
  }
}
```

2. Logout (`src/app/api/auth/logout/route.ts`)

```
await auditLog({
  userId: session.userId,
  action: 'logout',
  metadata: { sessionId: session.id },
  ip,
  userAgent,
  requestId,
});
```

3. Data Access (`src/app/api/users/[id]/route.ts`)

```
export async function GET(request: NextRequest, { params }: { params: { id: string } }) {
  const session = await requireAuth(request);
  const userId = params.id;

  // Check authorization
  if (session.userId !== userId && session.role !== 'admin') {
    await auditLog({
      userId: session.userId,
      action: 'access_denied',
      resourceType: 'user',
      resourceId: userId,
```

```
      metadata: { reason: 'insufficient_permissions' },
      ip: request.headers.get('x-forwarded-for'),
      userAgent: request.headers.get('user-agent'),
      requestId: getRequestId(request.headers),
      result: 'denied',
    });

    return jsonError('Access denied', 403);
  }

  // Audit successful data access
  await auditLog({
    userId: session.userId,
    action: 'data_view',
    resourceType: 'user',
    resourceId: userId,
    ip: request.headers.get('x-forwarded-for'),
    userAgent: request.headers.get('user-agent'),
    requestId: getRequestId(request.headers),
  });

  const user = await getUserById(userId);
  return jsonSuccess(user);
}
```

4. KYC Approval (`src/app/api/admin/kyc/route.ts`)

```
await auditLog({
  userId: adminSession.userId,
  action: 'kyc_approved',
  resourceType: 'user',
  resourceId: targetUserId,
  metadata: { reason: kycApprovalReason },
  ip: request.headers.get('x-forwarded-for'),
  userAgent: request.headers.get('user-agent'),
  requestId: getRequestId(request.headers),
});
```

5. Transaction Creation (`src/app/api/transactions/route.ts`)

```
await auditLog({
  userId: session.userId,
  action: 'transaction_created',
  resourceType: 'transaction',
  resourceId: transactionId,
  metadata: {
    type: transactionType,
    amount: amount,
    currency: currency,
  },
  ip: request.headers.get('x-forwarded-for'),
  userAgent: request.headers.get('user-agent'),
  requestId: getRequestId(request.headers),
});
```

Compliance Reporting

GDPR Right of Access (User Data Export)

```
// src/app/api/users/[id]/audit-logs/route.ts
export async function GET(request: NextRequest, { params }: { params: { id: string } }) {
  const session = await requireAuth(request);

  // Users can only access their own audit logs
  if (session.userId !== params.id && session.role !== 'admin') {
    return jsonError('Access denied', 403);
  }

  const logs = await getUserAuditLogs(params.id, {
    limit: 1000, // GDPR requires "all data"
    startDate: request.nextUrl.searchParams.get('start') || undefined,
    endDate: request.nextUrl.searchParams.get('end') || undefined,
  });
```

```
  return jsonSuccess({ logs });
}
```

PSD2 Audit Trail Export (Admin)

```
// src/app/api/admin/audit/export/route.ts
export async function GET(request: NextRequest) {
  const session = await requireAuth(request);

  if (session.role !== 'admin') {
    return jsonError('Admin access required', 403);
  }

  const startDate = request.nextUrl.searchParams.get('start');
  const endDate = request.nextUrl.searchParams.get('end');
  const action = request.nextUrl.searchParams.get('action');
  const userId = request.nextUrl.searchParams.get('userId');

  const csv = await exportAuditLogsCSV({
    userId: userId || undefined,
    action: action as AuditAction | undefined,
    startDate: startDate || undefined,
    endDate: endDate || undefined,
  });

  return new Response(csv, {
    headers: {
      'Content-Type': 'text/csv',
      'Content-Disposition': `attachment; filename="audit-logs-${new
Date().toISOString()}.csv"`,
    },
  });
}
```

Retention Policy

PSD2 Requirement: 5 Years

PostgreSQL 16 (all environments — ADR-014):

- Use table partitioning by month:

```
CREATE TABLE audit_log (
  id TEXT PRIMARY KEY,
  timestamp TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  -- ... other columns
) PARTITION BY RANGE (timestamp);

-- Create partitions for each month
CREATE TABLE audit_log_2026_02 PARTITION OF audit_log
  FOR VALUES FROM ('2026-02-01') TO ('2026-03-01');
```

- Automatic cleanup script (cron weekly):

```
#\!/bin/bash
# Delete audit logs older than 5 years (PSD2 retention)
psql "$DATABASE_URL" -c "DELETE FROM audit_log WHERE timestamp < NOW() - INTERVAL '5
years';"
```

Testing

Test Audit Logging

```
# 1. Create audit log entry
curl -X POST http://localhost:3000/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"email":"test@example.com","password":"wrong"}'

# 2. Check audit log table (PostgreSQL 16)
psql "$DATABASE_URL" -c "SELECT * FROM audit_log ORDER BY timestamp DESC LIMIT 5;"

# Expected output:
# audit_123 | 2026-02-22T10:00:00.000Z | usr_456 | login_failure | ... |
```

```
{"email":"test@example.com","reason":"invalid_credentials"} | 1.2.3.4 | Mozilla/5.0... |
req_789 | failure

# 3. Export audit logs (admin)
curl -X GET "http://localhost:3000/api/admin/audit/export?start=2026-02-01&end=2026-02-28" \
  -H "Cookie: auth-token=<admin-jwt>" \
  > audit-logs.csv

# 4. Verify CSV format
head -n 5 audit-logs.csv
```

Monitoring

Alert on Audit Failures

Add to `src/lib/audit-log.ts`:

```
import { sendAlert } from './alerts';

export async function auditLog(entry: AuditLogEntry): Promise<void> {
  try {
    // ... insert logic
  } catch (error) {
    logger.error('Failed to create audit log', { error, action: entry.action });

    // CRITICAL: Alert if audit logging fails (compliance risk)
    await sendAlert({
      severity: 'critical',
      title: 'Audit logging failure',
      message: `Failed to record ${entry.action} for user ${entry.userId}`,
    });
  }
}
```

Metrics to Track

- Audit logs created per hour (should correlate with user activity)

- Failed audit log attempts (should be zero)
- Audit log export requests (GDPR compliance)
- Audit log storage size (retention planning)

Security Considerations

Immutability

- Audit logs should NEVER be updated or deleted (except by automated retention policy)
- No UPDATE or DELETE API endpoints for audit logs
- Database permissions: Read-only for application, Write-only for audit service

Access Control

- Only admins can view full audit trails
- Users can view their own audit logs only
- Export requires elevated permissions

Data Redaction

- Do NOT log passwords, tokens, or sensitive PII in metadata
- Card numbers: Log last 4 digits only
- Fødselsnummer: Log checksum/hash, not full number

Checklist

- ☐ Migration `003_audit_logs.sql` created
- ☐ Migration applied to dev database
- ☐ `src/lib/audit-log.ts` implemented
- ☐ Login/logout endpoints integrated
- ☐ Data access endpoints integrated
- ☐ KYC/AML admin actions integrated
- ☐ GDPR export endpoint created
- ☐ PSD2 CSV export endpoint created
- ☐ Retention policy documented

- [ ] Monitoring alerts configured
- [ ] Testing completed (manual + automated)
- [ ] Documentation updated (API docs, compliance docs)

Next Steps:

1. Create migration file
2. Implement `audit-log.ts` library
3. Integrate into auth routes (high priority)
4. Add to remaining endpoints incrementally
5. Test with real login/logout flows
6. Deploy to staging for verification