drop-supporting-systems-plan

Plan: Drop Supporting Systems — Monitoring, Logging, Alerts, Backups

Research Summary

What Exists

- Health check endpoint (`GET /api/health`) — DB ping, latency, uptime, version
- Container health checks (Docker Compose 30s, Fly.io 30s)
- Auto-restart on failure (`restart: unless-stopped`)
- CI/CD pipeline (5 GitHub Actions jobs: lint, test, build, e2e, docker)
- Security basics (JWT httpOnly, bcrypt, CSRF, rate limiting, parameterized SQL)
- Manual SQLite backup/restore documented in DEPLOYMENT.md

What's Missing (from docs/infrastructure/MONITORING.md)

- External uptime monitoring
- Error tracking (Sentry)
- Structured logging (JSON + request IDs)
- Log aggregation
- Alerting (Slack/email)
- Audit logging (compliance requirement — PSD2, AML, GDPR)
- Database performance monitoring
- Automated backups
- Security scanning in CI

Tech Stack Context

- Next.js 16 + React 19, API Routes
- SQLite (better-sqlite3) for demo, PostgreSQL for prod
- Docker multi-stage builds
- Fly.io staging config ready (not deployed)
- Vitest + Playwright tests

Objective

Implement the missing supporting systems that make Drop operationally ready: structured logging, error tracking, audit logging, automated backups, alerting, and CI security scanning.

Team Orchestration

Team Members

ID	Name	Role	Agent Type
B1	logging-builder	Build structured logging + audit log system	builder
V1	logging-validator	Validate logging implementation	validator
B2	monitoring-builder	Build error tracking (Sentry) + uptime + alerting	builder
V2	monitoring-validator	Validate monitoring setup	validator
B3	backup-builder	Build automated backup system + CI security scanning	builder
V3	backup-validator	Validate backup + CI security	validator

Step-by-Step Tasks

Phase 1: Structured Logging + Audit Log (Foundation)

Task 1: Implement structured logging library

- Owner: B1
- BlockedBy: none
- Files: `src/drop-app/src/lib/logger.ts` (new)

- Acceptance:
  - [ ] JSON-formatted log output with timestamp, level, requestId, message, metadata
  - [ ] Request ID generation middleware (UUID per request, passed through all handlers)
  - [ ] Log levels: debug, info, warn, error
  - [ ] Writes to stdout (Docker-friendly, no file writes)
  - [ ] All existing API routes use logger instead of console.log
  - [ ] No new dependencies if possible (use built-in, or pino if needed for perf)

Task 2: Implement audit log table + middleware

- Owner: B1
- BlockedBy: 1
- Files: `src/drop-app/src/lib/db.ts` (schema), `src/drop-app/src/lib/audit.ts` (new)
- Acceptance:
  - [ ] `audit_log` table: id, timestamp, user_id, action, resource, details (JSON), ip_address, user_agent, request_id
  - [ ] Actions logged: login_success, login_failure, logout, register, password_change, transfer_initiated, transfer_completed, qr_payment, kyc_submitted, session_created, session_revoked
  - [ ] Audit entries created in same transaction as domain action where possible
  - [ ] Retention: no auto-delete (5-year compliance requirement noted in comments)
  - [ ] GET /api/admin/audit endpoint (admin-only, paginated) for future use

Task 3: Validate logging + audit log

- Owner: V1
- BlockedBy: 2
- Acceptance:
  - [ ] All API routes produce structured JSON logs on request
  - [ ] Request IDs are consistent across a single request's log entries
  - [ ] Login success/failure both produce audit log entries
  - [ ] Transfer/payment actions produce audit log entries
  - [ ] Audit table schema matches spec
  - [ ] Build passes, all existing tests still pass
  - [ ] No console.log left in API routes (replaced with logger)

Phase 2: Error Tracking + Monitoring + Alerting

Task 4: Integrate Sentry error tracking

- Owner: B2
- BlockedBy: 1 (needs logger for context)
- Files: `src/drop-app/src/lib/sentry.ts` (new), `src/drop-app/src/app/layout.tsx` (init), `src/drop-app/next.config.ts`
- Acceptance:
  - ☐ @sentry/nextjs installed and initialized (client + server)
  - ☐ DSN configurable via `SENTRY_DSN` env var
  - ☐ Sentry disabled when env var not set (no crash in dev)
  - ☐ Unhandled errors + rejected promises captured
  - ☐ API route errors captured with request context (user_id, requestId)
  - ☐ Source maps uploaded in Docker build (or skipped if no SENTRY_AUTH_TOKEN)
  - ☐ Environment tag: production/staging/development
  - ☐ Performance monitoring (traces sample rate configurable via env)

Task 5: Add health check monitoring + Slack alerting hook

- Owner: B2
- BlockedBy: 4
- Files: `src/drop-app/src/lib/alerts.ts` (new), env vars
- Acceptance:
  - ☐ Slack webhook integration via `SLACK_WEBHOOK_URL` env var
  - ☐ Alert on: application startup, graceful shutdown, unhandled error spike (>5 in 1 min)
  - ☐ Alert format: emoji + severity + message + timestamp + link to Sentry
  - ☐ Cooldown: max 1 alert per type per 10 minutes (no spam)
  - ☐ No-op when SLACK_WEBHOOK_URL not configured
  - ☐ UptimeRobot setup documented in MONITORING.md (external, free tier, checks /api/health)

Task 6: Validate monitoring + alerting

- Owner: V2
- BlockedBy: 5
- Acceptance:
  - ☐ Sentry captures thrown errors in API routes (test with intentional throw)
  - ☐ Sentry DSN not hardcoded (env var only)
  - ☐ Sentry disabled gracefully in dev (no SENTRY_DSN = no crash)
  - ☐ Slack alert function works with mock webhook (unit test)
  - ☐ No sensitive data in Sentry events (no passwords, tokens, card numbers)
  - ☐ MONITORING.md updated with full stack docs

- [ ] Build passes, all existing tests still pass

Phase 3: Automated Backups + CI Security

Task 7: Create automated backup script + CI security scanning

- Owner: B3
- BlockedBy: none (independent)
- Files: `src/drop-app/scripts/backup.sh` (new), `.github/workflows/ci.yml` (edit)
- Acceptance:
  - [ ] backup.sh: SQLite `.backup` command (safe, atomic), timestamped output
  - [ ] backup.sh: Configurable retention (default 30 days, delete older)
  - [ ] backup.sh: Exit code for cron/monitoring integration
  - [ ] backup.sh: Works inside Docker container (volume mount)
  - [ ] Docker Compose: backup service or documented cron setup
  - [ ] CI: `npm audit --audit-level=high` step added to GitHub Actions
  - [ ] CI: Fails build on HIGH/CRITICAL vulnerabilities
  - [ ] .github/dependabot.yml created (weekly npm updates)
  - [ ] DEPLOYMENT.md updated with backup schedule + restore procedure

Task 8: Validate backups + CI security

- Owner: V3
- BlockedBy: 7
- Acceptance:
  - [ ] backup.sh creates valid SQLite backup (can be opened, tables exist)
  - [ ] backup.sh handles missing DB gracefully (exit 1, clear message)
  - [ ] Old backups cleaned up after retention period
  - [ ] npm audit step present in CI workflow
  - [ ] dependabot.yml valid YAML, correct config
  - [ ] DEPLOYMENT.md backup section accurate
  - [ ] Build passes, all existing tests still pass

Validation Commands

```
# Phase 1: Logging + Audit
cd ~/ALAI/products/Drop/src/drop-app
npm run build                          # Build passes
npm test                               # All tests pass
# Start app, hit /api/auth/login → check stdout for JSON log
# Check /api/health → verify request ID in logs
# SELECT * FROM audit_log → verify entries after login

# Phase 2: Monitoring
# Set SENTRY_DSN → start app → trigger error → check Sentry dashboard
# Set SLACK_WEBHOOK_URL → trigger alert → check Slack
# npm run build with SENTRY_AUTH_TOKEN → verify sourcemaps

# Phase 3: Backups + CI
bash scripts/backup.sh                 # Creates timestamped backup
sqlite3 backups/drop-*.db ".tables"    # Verify backup integrity
# Push to GitHub → CI runs → npm audit step visible
# Check dependabot.yml in .github/
```

Summary

Phase	What	Effort
1	Structured logging + audit log	~1 day
2	Sentry + Slack alerts + uptime docs	~1 day
3	Automated backups + CI security scanning	~0.5 day

Total: ~2.5 days with 3 builder/validator pairs running in parallel.

All 3 phases can run in parallel (Phase 1 and 3 are independent, Phase 2 depends on Phase 1 Task 1 for logger context).

Revision #3
Created 2026-02-18 08:44:47 UTC by John
Updated 2026-05-24 20:00:49 UTC by John