Sentry Observability

Sentry Observability Integration

> Error tracking, performance monitoring, alerting, and SLO/SLI definitions for the Drop fintech platform. Covers Sentry SDK integration, data scrubbing, Slack alerting, structured logging, and release tracking.

Observability Data Flow

```
flowchart LR
    subgraph DropAPI["drop-api (Hono v4)"]
        EH[Error Handler<br/>middleware/error-handler.ts]
        Logger[Structured Logger<br/>lib/logger.ts]
        SentryLib[Sentry SDK<br/>lib/sentry.ts]
        AlertLib[Alert System<br/>lib/alerts.ts]
    end

    subgraph SentryCloud["Sentry Cloud"]
        Issues[Issue Tracking]
        Perf[Performance Monitoring<br/>Transaction traces]
        Releases[Release Tracking<br/>Source maps + commits]
    end

    subgraph Alerting["Alert Channels"]
        Slack[Slack Webhook<br/>SLACK_WEBHOOK_URL]
        Console[stdout<br/>JSON structured logs]
    end

    subgraph Monitoring["Infrastructure"]
        CloudWatch[AWS CloudWatch<br/>Container logs + metrics]
    end
```

```
    EH -->|Unhandled errors| SentryLib
    EH -->|5xx errors| AlertLib
    EH -->|All errors| Logger
    SentryLib -->|captureException| Issues
    SentryLib -->|Transaction traces| Perf
    SentryLib -->|Release metadata| Releases
    AlertLib -->|Error spikes| Slack
    AlertLib -->|Startup/shutdown| Slack
    Logger -->|JSON log entries| Console
    Console -->|Container stdout| CloudWatch

    SentryLib -.->|beforeSend| Scrubber[Data Scrubber<br/>Removes PII, auth
tokens,<br/>fødselsnummer, passwords]
```

Sentry Configuration

SDK Setup

Source: `src/drop-api/src/lib/sentry.ts` **Package:** `@sentry/node` **Initialization:** Lazy — initialized on first `captureError()` or `captureMessage()` call.

```
Sentry.init({
  dsn: process.env.SENTRY_DSN,
  environment: process.env.NODE_ENV || "development",
  release: `drop-api@${version}`,  // from package.json
  tracesSampleRate: parseFloat(process.env.SENTRY_TRACES_SAMPLE_RATE || "0.1"),
  beforeSend(event, hint) {
    // Scrub sensitive data before sending to Sentry
    // See "Data Scrubbing" section below
  },
});
```

Environment Variables

Variable	Required	Default	Description

`SENTRY_DSN`	No	Not set (no-op mode)	Sentry project DSN. When absent, all Sentry calls are no-ops and errors are only logged to console.
`SENTRY_TRACES_SAMPLE_RATE`	No	`0.1` (10%)	Percentage of transactions to trace for performance monitoring.
`NODE_ENV`	No	`development`	Maps to Sentry `environment` tag.
`SLACK_WEBHOOK_URL`	No	Not set (skip alerts)	Slack incoming webhook for alert notifications.

Sentry Project Structure

Project	Environment	Purpose
`drop-api` (staging)	`staging`	Pre-release error tracking, higher trace sample rate (0.5)
`drop-api` (production)	`production`	Production error tracking, 10% trace sample rate
`drop-web` (planned)	`production`	Client-side error tracking via `@sentry/nextjs`
`drop-mobile` (planned)	`production`	React Native error tracking via `@sentry/react-native`

Data Scrubbing

Source: `sentry.ts:108-139`

All events pass through a `beforeSend` hook that removes sensitive data before transmission to Sentry.

String Scrubbing Patterns

Pattern	Replacement	Example
`password=...`	`password=[REDACTED]`	Request bodies, exception messages
`pin=...`	`pin=[REDACTED]`	Card PIN values
`cardNumber=...`	`cardNumber=[REDACTED]`	Card numbers (PCI-DSS)
`cvv=...`	`cvv=[REDACTED]`	Card verification values

Pattern	Replacement	Example
`fødselsnummer=...`	`fødselsnummer=[REDACTED]`	Norwegian national ID (PII)
`authorization:...`	`authorization:[REDACTED]`	JWT tokens
`cookie:...`	`cookie:[REDACTED]`	Session cookies

Object Scrubbing

Sensitive keys are recursively scrubbed from event `extra` data, breadcrumbs, and request bodies:

- Keys containing: `password`, `pin`, `cardNumber`, `cvv`, `fødselsnummer`, `authorization`, `cookie`
- Values replaced with `[REDACTED]`

Request Header Removal

The following headers are deleted from Sentry events entirely:

- `authorization` (Bearer tokens)
- `cookie` (session cookies)
- `set-cookie` (response cookies)

Sentry SDK Functions

Source: `sentry.ts:141-244`

Function	Purpose	When Used
`captureError(error, context?)`	Capture exception with optional user ID, request ID, tags, and extra data. Returns Sentry event ID.	`error-handler.ts` on 5xx errors
`captureMessage(message, level?, context?)`	Capture informational message with severity level.	Manual diagnostic logging
`setUser(userId, metadata?)`	Associate current scope with user ID for error grouping.	After successful authentication
`clearUser()`	Remove user association from current scope.	On logout
`addBreadcrumb(message, category, level?, data?)`	Add navigation/action breadcrumb for error context. Data is scrubbed.	Key user actions
`setTag(key, value)`	Set tag on current scope for filtering.	Environment, feature flags

Function	Purpose	When Used
`flush(timeout?)`	Flush pending events on graceful shutdown. Default 2000ms.	Process exit handler
`isSentryEnabled()`	Check if `SENTRY_DSN` is set.	Conditional logic

Dual Output Pattern

All Sentry functions also log to `console.error` / `console.log`. This ensures errors are captured in CloudWatch logs even if Sentry is unavailable, and provides a local development experience without requiring a Sentry DSN.

Alert System

Source: `src/drop-api/src/lib/alerts.ts`

Slack Alerting

Alerts are sent to a Slack channel via incoming webhook (`SLACK_WEBHOOK_URL`).

Alert Type	Severity	Trigger	Cooldown
Error spike	`critical`	5+ errors within 60 seconds	10 minutes per unique title
API startup	`info`	Application startup	None
API shutdown	`info`	Graceful shutdown	None

Error Spike Detection

Source: `alerts.ts:65-83`

```
Window: 60 seconds (sliding)
Threshold: 5 errors
Cooldown: 10 minutes per alert title
```

The `trackError()` function is called from the global error handler (`error-handler.ts:10`) for every 5xx error. It maintains a sliding window of error timestamps and triggers a Slack alert when the spike threshold is breached.

Alert Message Format

```
CRITICAL *Error spike detected*
7 errors in last minute
_2026-02-21T12:00:00.000Z_
View in Sentry (link if available)
```

Structured Logging

Source: `src/drop-api/src/lib/logger.ts`

Log Entry Format

All logs are written to stdout as newline-delimited JSON (NDJSON):

```
{
  "timestamp": "2026-02-21T12:00:00.000Z",
  "level": "info",
  "message": "BankID login successful",
  "requestId": "550e8400-e29b-41d4-a716-446655440000",
  "metadata": {
    "userId": "usr_a1b2c3d4e5f6g7h8",
    "method": "bankid"
  }
}
```

Log Levels

Level	Usage	Example
`debug`	Detailed diagnostic info	SQL query parameters
`info`	Normal operations	Login success, transaction created
`warn`	Recoverable issues	Token verification failed, rate limit approached
`error`	Errors requiring attention	BankID callback error, database connection failure

Request Context

Each request gets a unique `requestId` (from `x-request-id` header or `crypto.randomUUID()`). This ID is:

1. Set as a Hono context variable (`app.ts:34`)
2. Returned in the `x-request-id` response header (`app.ts:36`)
3. Included in all log entries for the request
4. Attached as a Sentry tag for cross-referencing

Error Handling Flow

```
sequenceDiagram
    participant Client
    participant Route as Route Handler
    participant ErrHandler as Global Error Handler
    participant Logger as Structured Logger
    participant Sentry as Sentry SDK
    participant Alerts as Alert System
    participant Slack as Slack Webhook

    Route->>Route: Business logic throws error

    alt HTTPException
        Route-->>ErrHandler: HTTPException (known status)
        ErrHandler->>Logger: Log error with request context
        ErrHandler-->>Client: { error: "http_error", message: "..." }, status
    else Unhandled Error
        Route-->>ErrHandler: Error thrown
        ErrHandler->>Logger: logger.error(message, stack, path, method)
        ErrHandler->>Sentry: captureError(error)
        Sentry->>Sentry: beforeSend → scrub PII
        Sentry->>Sentry: Send to Sentry Cloud
        ErrHandler->>Alerts: trackError(error)
        Alerts->>Alerts: Add to sliding window
        alt Error spike (5+ in 60s)
            Alerts->>Slack: POST webhook (CRITICAL alert)
        end
        ErrHandler-->>Client: { error: "internal_error", message: "An unexpected error
occurred" }, 500
    end
```

SLO/SLI Definitions

Service Level Indicators (SLIs)

SLI	Measurement	Source
Availability	Percentage of successful health check responses (200)	App Runner health checks, Cloudflare origin monitoring
Latency	p50, p95, p99 response time for API endpoints	Sentry performance monitoring, CloudWatch
Error Rate	Percentage of 5xx responses out of total requests	Sentry issue counts, structured logs
Database Latency	`dbLatencyMs` from `/v1/health` endpoint	Health check response, CloudWatch custom metric
Auth Success Rate	Percentage of successful BankID authentications	Sentry custom metrics, audit_log table

Service Level Objectives (SLOs)

SLO	Target	Measurement Window	Error Budget
API Availability	99.9%	30-day rolling	43 minutes/month downtime
API Latency (p95)	< 300ms	30-day rolling	5% of requests may exceed
API Latency (p99)	< 1000ms	30-day rolling	1% of requests may exceed
Error Rate (5xx)	< 0.1%	30-day rolling	1 in 1000 requests
Database Latency	< 50ms	30-day rolling	5% of queries may exceed
Auth Success Rate	> 99%	30-day rolling	1% of auth attempts may fail
Transaction Success Rate	> 99.5%	30-day rolling	0.5% of transactions may fail

Performance Targets (from architecture document)

Metric	Target	Source
First Contentful Paint (FCP)	< 1.5s	`architecture-document.md` section 8

Metric	Target	Source
Largest Contentful Paint (LCP)	< 2.5s	`architecture-document.md` section 8
Time to First Byte (TTFB)	< 200ms	`architecture-document.md` section 8
API p95 response time	< 300ms	`architecture-document.md` section 8
Lighthouse score	> 90	`architecture-document.md` section 8
Build time	< 60s	`architecture-document.md` section 8

Alert Rule Configuration

Rule	Condition	Severity	Channel	Cooldown
Error spike	5+ errors in 60s window	`critical`	Slack webhook	10 min
API startup	Application start event	`info`	Slack webhook	None
API shutdown	Graceful shutdown event	`info`	Slack webhook	None
Health check failure	`/v1/health` returns 503	`critical`	App Runner auto-restart + CloudWatch alarm	N/A
High error rate	5xx rate > 5% for 2 minutes	`critical`	CloudWatch alarm → SNS → Slack	5 min
High latency	p95 > 500ms for 3 minutes	`warning`	CloudWatch alarm → SNS → Slack	5 min
Database slow query	Query > 1000ms	`warning`	Sentry performance alert	10 min
Auth failure spike	10+ auth failures in 5 minutes	`warning`	Sentry alert rule	15 min

---

Release Tracking

Source Map Upload

```
# In CI/CD pipeline after build
sentry-cli releases new "drop-api@${VERSION}"
```

```
sentry-cli releases set-commits "drop-api@${VERSION}" --auto
sentry-cli sourcemaps upload --release "drop-api@${VERSION}" ./dist/
sentry-cli releases finalize "drop-api@${VERSION}"
```

Release Metadata

Each release in Sentry includes:

- **Version:** `drop-api@{package.json version}` (e.g., `drop-api@0.1.0`)
- **Environment:** `development`, `staging`, or `production`
- **Commits:** Automatically linked via `--auto` flag
- **Source maps:** Uploaded for stack trace deobfuscation

Cross-References

- **Error handler:** `src/drop-api/src/middleware/error-handler.ts` — Global error handler that triggers Sentry capture
- **Logger:** `src/drop-api/src/lib/logger.ts` — Structured JSON logging
- **Sentry SDK:** `src/drop-api/src/lib/sentry.ts` — Full Sentry integration with data scrubbing
- **Alert system:** `src/drop-api/src/lib/alerts.ts` — Slack webhook alerting with spike detection
- **Security architecture:** SECURITY-ARCHITECTURE.md — Security monitoring context
- **Deployment:** deployment-architecture.md — CloudWatch and infrastructure monitoring

---

Revision #4
Created 2026-02-21 05:59:04 UTC by John
Updated 2026-05-23 10:57:06 UTC by John