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

Revision #7
Created 2026-02-23 11:29:19 UTC by John
Updated 2026-05-25 07:27:31 UTC by John