Login & Authentication (Backend)

Login & Authentication — Backend Architecture

> Backend-specific authentication details for the Drop fintech platform. Covers JWT token structure, token refresh mechanism, session revocation, rate limiting on auth endpoints, audit logging, demo mode implementation, and cookie security settings.

JWT Token Structure

Token Generation

Source: `src/drop-api/src/lib/auth.ts:42-48` **Library:** `jose` (HS256 default, RS256 opt-in)

```
new jose.SignJWT({ userId, email, role })
  .setProtectedHeader({ alg: "HS256" })
  .setIssuedAt()
  .setIssuer("drop-api")
  .setAudience("drop")
  .setExpirationTime("7d")
  .sign(key);
```

JWT Claims

Claim	Type	Value	Source

`userId`	`string`	`usr_<16 hex chars>` (e.g., `usr_a1b2c3d4e5f6g7h8`)	`auth.ts:44` — from user record
`email`	`string`	`usr_xxx@bankid.drop.local` (BankID placeholder) or real email	`auth.ts:44` — from user record
`role`	`string`	`user` or `merchant`	`auth.ts:44` — from `users.role` column
`iat`	`number`	Unix timestamp of issuance	`auth.ts:45` — `setIssuedAt()`
`exp`	`number`	`iat` + 7 days (604800 seconds)	`auth.ts:45` — `setExpirationTime("7d")`
`iss`	`string`	`drop-api`	`auth.ts:45` — `setIssuer()`
`aud`	`string`	`drop`	`auth.ts:45` — `setAudience()`

Algorithm Selection

Algorithm	Condition	Key Source	Use Case
HS256 (default)	`JWT_RS256_PRIVATE_KEY` not set	`JWT_SECRET` env var → `TextEncoder.encode()`	Standard deployment
RS256 (opt-in)	Both `JWT_RS256_PRIVATE_KEY` and `JWT_RS256_PUBLIC_KEY` set	PEM-encoded RSA key pair	Multi-service verification (API gateway, microservices)

Source: `auth.ts:12-34` — `getAlgorithm()` auto-detects based on available keys.

Token Verification

Source: `auth.ts:50-66`

1. Determine algorithm (HS256 or RS256)
2. Call `jose.jwtVerify(token, key, { issuer: "drop-api", audience: "drop" })`
3. Extract `userId`, `email`, `role` from payload
4. Type-check: both `userId` and `email` must be strings
5. Default `role` to `"user"` if not present

JWT Refresh Flow

```
sequenceDiagram
    participant Client as Client (Web/Mobile)
    participant API as drop-api
    participant DB as Database

    Client->>API: POST /v1/auth/refresh<br/>Authorization: Bearer <current-jwt>

    API->>API: Extract token from Bearer header or cookie
    API->>API: Verify JWT signature (jose)
    API->>DB: SELECT session WHERE token_hash = SHA256(token)<br/>AND revoked = 0 AND
expires_at > NOW()
    DB-->>API: Session valid

    API->>DB: SELECT user WHERE id = userId AND deleted_at IS NULL
    DB-->>API: User record

    Note over API: Revoke ALL existing sessions
    API->>DB: UPDATE sessions SET revoked = 1<br/>WHERE user_id = ?

    Note over API: Create new session
    API->>API: Sign new JWT (7d expiry)
    API->>DB: INSERT INTO sessions<br/>(id, user_id, token_hash, expires_at)

    Note over API: Set cookie for web clients
    API->>API: Set-Cookie: drop_token=<new-jwt>;<br/>HttpOnly; Path=/; Max-Age=604800;
SameSite=Lax

    API-->>Client: { data: { id, email, firstName, ... }, token: "<new-jwt>" }
```

Refresh Behavior

Source: `routes/auth.ts:201-210`

1. Auth middleware validates current token
2. **All existing sessions revoked** (`revokeAllSessions(user.id)`)
3. New JWT signed with fresh `iat` and `exp` (7 days from now)
4. New session record created in `sessions` table
5. Cookie set for web clients (`Set-Cookie` header)
6. Token returned in JSON body for mobile clients

Key design decision: Token refresh performs a full session rotation — old sessions are invalidated immediately. This limits the window for token theft: a stolen token becomes invalid as soon as the legitimate user refreshes.

Session Revocation

Session Revocation Flow

```
sequenceDiagram
    participant Client as Client
    participant API as drop-api
    participant DB as Database

    alt Logout (user-initiated)
        Client->>API: POST /v1/auth/logout<br/>Authorization: Bearer <jwt>
        API->>API: Verify token (authMiddleware)
        API->>DB: UPDATE sessions SET revoked = 1<br/>WHERE user_id = ?
        Note over DB: ALL sessions for this user revoked
        API->>DB: INSERT INTO audit_log<br/>(action: 'logout', resource_type: 'session')
        API->>API: Set-Cookie: drop_token=; Max-Age=0
        API-->>Client: { data: { message: "Logged out" } }

    else Security incident (admin-initiated)
        Note over API: Admin detects compromised account
        API->>DB: UPDATE sessions SET revoked = 1<br/>WHERE user_id = ?
        API->>DB: INSERT INTO audit_log<br/>(action: 'security_revocation')
        Note over Client: Next request fails auth check
        Client->>API: Any authenticated request
        API->>DB: SELECT session WHERE token_hash = ?<br/>AND revoked = 0
        DB-->>API: No valid session found
        API-->>Client: 401 Unauthorized

    else Token refresh (rotation)
        Client->>API: POST /v1/auth/refresh
        API->>DB: UPDATE sessions SET revoked = 1<br/>WHERE user_id = ?
        Note over DB: Old sessions invalidated
        API->>DB: INSERT INTO sessions (new session)
```

```
        API-->>Client: { token: "<new-jwt>" }
    end
```

Session Verification on Every Request

Source: `auth.ts:108-117`

Every authenticated request performs these checks:

1. **Token signature verification** — JWT must be valid and not expired
2. **Session lookup** — `SELECT id FROM sessions WHERE token_hash = SHA256(token) AND revoked = 0 AND expires_at > NOW()`
3. **Session count check** — If user has any sessions in DB but none match the current token, reject (prevents use of tokens from before session tracking was enabled)
4. **User existence check** — `SELECT * FROM users WHERE id = ? AND deleted_at IS NULL` (soft-deleted users are blocked)

Session Table Schema

Column	Type	Description
`id`	TEXT PK	Format: `ses_<16 hex chars>`
`user_id`	TEXT FK	References `users.id`
`token_hash`	TEXT	SHA-256 hash of the JWT string
`created_at`	TEXT	ISO timestamp of session creation
`expires_at`	TEXT	ISO timestamp, 7 days from creation
`revoked`	INTEGER	`0` = active, `1` = revoked

Indexes: `idx_sessions_user` (user_id), `idx_sessions_token` (token_hash)

Rate Limiting on Auth Endpoints

Rate Limit Configuration

Endpoint	Limit	Window	Source
`GET /v1/auth/bankid/initiate`	10 requests	60 seconds	`routes/auth.ts:19`

Endpoint	Limit	Window	Source
`POST /v1/auth/bankid/callback`	10 requests	60 seconds	`routes/auth.ts:43`
`POST /v1/auth/demo-login`	Inherits from service mode check	N/A	Only available in demo mode
`GET /v1/auth/me`	No additional rate limit	N/A	Auth required (implicit protection)
`POST /v1/auth/logout`	No additional rate limit	N/A	Auth required
`POST /v1/auth/refresh`	No additional rate limit	N/A	Auth required

Rate Limiting Implementation

Source: `middleware/rate-limit.ts:7-23`

```
Storage: rate_limits table (persistent across restarts)
Key: Client IP address
Algorithm: Fixed window counter
Cleanup: Every 100 requests, expired entries are deleted
Atomic: Uses runUpsert for race-condition-safe counter updates
```

The rate limiter uses the `rate_limits` database table:

Column	Type	Description
`key`	TEXT PK	Client IP address
`count`	INTEGER	Request count in current window
`reset_at`	INTEGER	Unix timestamp when window resets

Client IP Extraction

Source: `middleware/rate-limit.ts:25-27`

Priority order:

1. `x-real-ip` header (nginx/Cloudflare)
2. First IP in `x-forwarded-for` chain (proxy chain)
3. Fallback: `127.0.0.1`

Audit Logging for Auth Events

Audit Actions

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

Action	Trigger	Data Recorded
`REGISTER`	New user created via BankID	`userId`, `method: bankid`, `isNewUser: true`, IP, user agent
`LOGIN`	Existing user authenticated via BankID	`userId`, `method: bankid`, `isNewUser: false`, IP, user agent
`LOGOUT`	User calls `/v1/auth/logout`	`userId`, `resourceType: session`
`REFRESH`	Token refresh	`userId`, `resourceType: session`

Audit Log Schema

Table: `audit_log`

Column	Type	Auth-Specific Usage
`id`	TEXT PK	Format: `aud_<16 hex chars>`
`timestamp`	TEXT	ISO timestamp of event
`user_id`	TEXT FK	Authenticated user ID
`action`	TEXT	One of `REGISTER`, `LOGIN`, `LOGOUT`, `REFRESH`
`resource_type`	TEXT	`auth` or `session`
`resource_id`	TEXT	Session ID (for session events)
`details`	TEXT	JSON: `{ method, isNewUser, platform }`
`ip_address`	TEXT	Client IP from middleware
`user_agent`	TEXT	`User-Agent` header value
`request_id`	TEXT	Correlation ID from `x-request-id` header

Audit Log Example

```
{
  "id": "aud_a1b2c3d4e5f6g7h8",
```

```
  "timestamp": "2026-02-21T12:00:00.000Z",
  "user_id": "usr_f1e2d3c4b5a69788",
  "action": "LOGIN",
  "resource_type": "auth",
  "details": "{\"method\":\"bankid\",\"isNewUser\":false}",
  "ip_address": "203.0.113.42",
  "user_agent": "Mozilla/5.0 (iPhone; CPU iPhone OS 18_0 like Mac OS X)",
  "request_id": "550e8400-e29b-41d4-a716-446655440000"
}
```

Demo Mode Implementation

Overview

Source: `routes/auth.ts:131-159`

Demo mode provides authentication without BankID for development and testing. API-side demo mode is controlled by `DROP_MODE` env var (checked via `isDemoMode()` in `services/mode.ts`). `NEXT_PUBLIC_SERVICE_MODE` is the client-side equivalent (set to `demo` in docker-compose.yml).

Demo Login Endpoint

Endpoint: `POST /v1/auth/demo-login`

Aspect	Behavior
Availability	Only when `isDemoMode()` returns `true`
Authentication	None required
User	Fixed demo user: `usr_demo1` (seeded in `db.ts`)
Response	JWT token + user data (same format as BankID callback)
Feature flag	Returns 404 when demo mode is disabled

Demo User Profile

Field	Value	Source
ID	`usr_demo1`	`db.ts` seed data
Email	`demo@example.test`	`db.ts` seed data

Field	Value	Source
Name	Demo User	`db.ts` seed data
Phone	`+4700000000`	`db.ts` seed data
Role	`merchant`	Upgraded in `initDb()`
KYC Status	`approved`	Set on seed
Bank accounts	DNB (45,000 NOK), Nordea (12,350 NOK)	`db.ts` seed data

BankID Mock Mode

Source: `bankid.ts:126-128`

When `BANKID_MOCK=true`, the BankID OIDC flow is mocked:

- `exchangeAndVerify()` skips token exchange and JWKS verification
- Returns a mock user based on the auth code value:
  - Code starting with `underage`: returns user born 2010 (fails age check)
  - Default: returns `Test Bankersen`, born 1990 (passes age check)

Deprecated Endpoints

Source: `routes/auth.ts:109-128`

Endpoint	Status Code	Message
`POST /v1/auth/login`	410 Gone	"Email/password login is no longer supported. Please use BankID."
`POST /v1/auth/register`	410 Gone	"Email/password registration is no longer supported. Please use BankID."
`POST /v1/auth/verify-otp`	410 Gone	"OTP verification is no longer supported. Authentication is handled via BankID."

Cookie Security Settings

Cookie Configuration

Source: `routes/auth.ts:195, 206`

Property	Value	Purpose	Source
`HttpOnly`	`true`	Prevents JavaScript access — mitigates XSS token theft	`auth.ts`, cookie string
`Secure`	`true` (production)	Cookie only sent over HTTPS	Implied by deployment (Cloudflare enforces HTTPS)
`SameSite`	`Lax`	Prevents CSRF — cookie not sent on cross-origin POST requests	`routes/auth.ts:206`
`Path`	`/`	Cookie available to all routes	`routes/auth.ts:206`
`Max-Age`	`604800` (7 days)	Session lifetime matching JWT expiry	`routes/auth.ts:206`
`Domain`	Not set (defaults to current domain)	Scoped to `getdrop.no` in production	Default browser behavior

Cookie Lifecycle

Event	Cookie Action	Source
BankID callback (web)	Set `drop_token=<jwt>` with full security attributes	BFF redirect handler
Token refresh	Set new `drop_token=<new-jwt>`, same attributes	`routes/auth.ts:206`
Logout	Clear cookie: `drop_token=; Max-Age=0`	`routes/auth.ts:195`

SameSite=Lax Behavior

Request Type	Cookie Sent?	Reason
Same-origin GET	Yes	Normal navigation
Same-origin POST	Yes	Form submissions
Cross-origin GET (top-level navigation)	Yes	Allows BankID redirect back
Cross-origin POST	No	CSRF protection
Cross-origin AJAX/fetch	No	CSRF protection
Subdomain requests	Depends on Domain setting	No Domain set = strict origin match

Web vs Mobile Token Strategy

Aspect	Web (Next.js)	Mobile (Expo)

Token delivery	httpOnly cookie (`drop_token`)	JSON body (`{ token }`)
Token storage	Browser cookie jar (managed by browser)	`AsyncStorage` (React Native encrypted storage)
Token transmission	Automatic via `Cookie` header	Manual via `Authorization: Bearer` header
CSRF protection	`SameSite=Lax` + CORS origin validation	Not needed (no cookies, Bearer token)
Token extraction	`auth.ts:96-99` — parse from `cookie` header	`auth.ts:93-94` — extract from `Authorization` header

Cross-References

- **BankID OIDC flow:** AUTHENTICATION.md — Full BankID authentication sequence
- **Auth source:** `src/drop-api/src/lib/auth.ts` — JWT signing, verification, session management
- **Auth routes:** `src/drop-api/src/routes/auth.ts` — Endpoint handlers
- **Auth middleware:** `src/drop-api/src/middleware/auth.ts` — Request authentication
- **Rate limiter:** `src/drop-api/src/middleware/rate-limit.ts` — IP-based rate limiting
- **BankID library:** `src/drop-api/src/lib/bankid.ts` — OIDC flow, pid parsing, user creation
- **Security architecture:** SECURITY-ARCHITECTURE.md — Cookie settings, JWT configuration
- **API reference:** API-REFERENCE.md — Full endpoint documentation
- **Database schema:** DATABASE-SCHEMA.md — `sessions`, `users`, `audit_log` tables

Revision #5
Created 2026-02-21 05:58:50 UTC by John
Updated 2026-05-23 10:51:47 UTC by John