API Audit Information – OnTime Ethiopia

API Audit Information – OnTime Ethiopia

Organization: AI TECHNOLOGIES PLC
Platform: OnTime Ethiopia (Web Application + Backend APIs)
Prepared for: Information Network Security Administration (INSA)
Date: November 2025

---

1. Name of the Assets to be Audit (APIs)

This section lists the primary API assets that are in scope for the security assessment. All endpoints are served under the same base domain, with different paths.

1.1 Base URLs

- **Production (example):** `https://ontime.aitechnologiesplc.com/`
- **API Base Path:** `/api/`

1.2 Assets / API Groups in Scope

Asset Name	URL / Path Example	Description
Authentication & Session APIs	`/api/token/`, `/api/token/refresh/`, `/api/logout/`	Login, refresh, logout, user session management
User Profile APIs	`/api/me/`	Current user info and profile accounts
Dynamic Forms APIs	`/api/forms/*`	JSON-driven forms (login, register, profile, etc.)
Version & Feature Management APIs	`/api/channels/version/*`, `/api/channels/features/`	App version checks and feature flags

Asset Name	URL / Path Example	Description
Channel Management APIs	`/api/channels/*`	Channels list, detail, logos
Shorts APIs	`/api/channels/shorts/*`	Shorts playlists and feed
Admin & User Management APIs	`/api/admin-only/`, `/api/users/`	Admin-only resources and user listing
Session Management APIs	`/api/sessions/*`	List and revoke active user sessions

---

2. API Types (Mandatory)

- **Type:** RESTful JSON APIs over HTTPS.
- **Style:** Resource-based endpoints with standard HTTP verbs (GET, POST, PUT, PATCH, DELETE).
- **Access Pattern:**
  - **Public / semi-public:** Selected endpoints may be callable by anonymous users (e.g. health/static info if enabled).
  - **Authenticated:** Majority of endpoints require JWT-based authentication.
  - **Admin-only:** Sensitive operations are restricted to admin/editor roles.
- **Synchronous / Asynchronous:**
  - HTTP requests are processed synchronously.
  - Longer-running operations (e.g. shorts ingestion) are managed via Celery workers triggered by API calls.

---

3. Updated API Documentation (Overview)

All API calls must include tenant and, where required, authorization headers:

- `X-Tenant-Id: ontime` (or other configured tenant)
- `Authorization: Bearer <ACCESS_TOKEN>` (for protected endpoints)
- `Content-Type: application/json` for JSON payloads

3.1 Authentication & User APIs

- **POST** `/api/token/` – Obtain JWT access and refresh tokens using **username + password**.
- **POST** `/api/token/refresh/` – Refresh access token using refresh cookie.

- **POST** `/api/logout/` – Logout and invalidate tokens.
- **POST** `/api/register/` – Register a new account (rate-limited).
- **GET / PUT / PATCH** `/api/me/` – Get or update current user profile.

3.2 OTP Authentication APIs

- **POST** `/api/auth/otp/request/` – Request OTP for email/phone.
- **POST** `/api/auth/otp/verify/` – Verify OTP and issue tokens.
- **POST** `/api/auth/otp/resend/` – Resend OTP.

3.3 Social Authentication APIs

- **POST** `/api/auth/social/login/` – Social login using provider token.
- **POST** `/api/auth/social/link/` – Link social account to existing user.
- **POST** `/api/auth/social/unlink/` – Unlink social account.

3.4 Dynamic Forms APIs

- **GET** `/api/forms/schema/` – Retrieve form schema using `action` query param (e.g. `action=login|register|verify_otp|reset_password`).
- **POST** `/api/forms/validate/` – Validate a single field (`field`, `value`, etc.).
- **POST** `/api/forms/submit/` – Submit a form with `action` (login/register/…) and `data` payload.
- **GET** `/api/forms/config/` – Retrieve form configuration/feature flags.

3.5 Version & Feature Management APIs

- **POST** `/api/channels/version/check/` – Check if app version is supported and whether an update is required.
- **GET** `/api/channels/version/latest/` – Get latest supported version for a platform.
- **GET** `/api/channels/version/supported/` – List all supported versions.
- **GET** `/api/channels/features/` – Retrieve feature flags for a given platform and version.

3.6 Channel Management APIs

- **GET** `/api/channels/` – List channels (paginated).
- **GET** `/api/channels/{id}/` – Get specific channel.
- **POST** `/api/channels/` – Create channel (admin only).
- **PUT/PATCH** `/api/channels/{id}/` – Update channel (admin only).
- **DELETE** `/api/channels/{id}/` – Delete channel (admin only).
- **GET** `/api/channels/{slug}/logo/` – Retrieve channel logo asset for use in the web frontend.

3.7 Shorts APIs

- **GET** `/api/channels/shorts/playlists/` – Recent shorts playlists per channel, tenant-aware.
- **GET** `/api/channels/shorts/feed/` – Deterministically shuffled shorts feed for end users.

3.8 Admin & Session Management APIs

- **GET** `/api/admin-only/` – Admin-only endpoint for tests / admin checks.
- **GET** `/api/users/` – User listing for admins (permission-based).
- **GET** `/api/sessions/` – List active sessions for current user.
- **DELETE** `/api/sessions/{session_id}/` – Revoke a specific session.
- **POST** `/api/sessions/revoke-all/` – Revoke all sessions for current user.

For full request/response examples, see `API_ENDPOINTS.md` and/or Swagger.

---

4. API Endpoints and Functionality (Mandatory)

This section summarizes each main endpoint category with its functionality. Detailed fields and payloads are available in `API_ENDPOINTS.md`.

4.1 Authentication & Session

- `POST /api/token/` – Username/password login. Issues JWT access token and refresh token (also set as httpOnly cookie). Rate-limited (5 attempts/minute).
- `POST /api/token/refresh/` – Use existing refresh cookie to obtain new access (and possibly refresh) tokens.
- `POST /api/logout/` – Invalidates the current refresh token (blacklisting/cleanup) and clears auth cookies.
- `POST /api/register/` – Create new user account; enforces password and email policies.
- `GET/PUT/PATCH /api/me/` – Read or update the authenticated user profile.

4.2 OTP & Social Auth

- `POST /api/auth/otp/request/` – Generate OTP code and send via configured channel (email/SMS).
- `POST /api/auth/otp/verify/` – Verify OTP; on success, issues tokens similar to password login.
- `POST /api/auth/otp/resend/` – Reissue OTP, obeying rate limits.
- `POST /api/auth/social/login/` – Exchange an OAuth/OpenID Connect token (e.g. Google ID token) for OnTime JWT tokens.

- `POST /api/auth/social/link/` – Attach external provider to an existing user account.
- `POST /api/auth/social/unlink/` – Remove link to external provider when allowed.

4.3 Dynamic Forms

- `GET /api/forms/schema/{form_type}/` – Returns JSON schema used by the web frontend to render forms dynamically (e.g. login, register).
- `POST /api/forms/validate/` – Validates form data against server-side rules, returning structured errors where relevant.
- `POST /api/forms/submit/` – Executes the underlying operation (e.g. login, register) based on `form_type` and payload.

4.4 Version & Feature Management

- `POST /api/channels/version/check/` – Client submits platform and version; server replies whether an update is required/available, and provides metadata and flags.
- `GET /api/channels/version/latest/` – Reads latest version info for a platform; used by clients or admin tools.
- `GET /api/channels/version/supported/` – Returns all supported app versions; useful for audit and troubleshooting.
- `GET /api/channels/features/` – Returns feature flags for the calling client, enabling server-driven feature toggling.

4.5 Channel & Media APIs

- `GET /api/channels/` – List channels visible for the current tenant and permissions.
- `GET /api/channels/{id}/` – Detailed channel information.
- `POST /api/channels/` – Admin/editor creates a new channel entry.
- `PUT/PATCH /api/channels/{id}/` – Admin/editor updates channel properties.
- `DELETE /api/channels/{id}/` – Admin/editor removes a channel.
- `GET /api/channels/{slug}/logo/` – Serves channel logo file; used directly in web frontend image tags.

4.6 Shorts APIs

- `GET /api/channels/shorts/playlists/` – Returns recent shorts playlists; supports filters like `days`, `limit`, `per_channel_limit`, `channel`.
- `GET /api/channels/shorts/feed/` – Returns a randomized but deterministic shorts feed, supporting `seed`, `X-Device-Id`, and recency bias parameters.

4.7 Admin & Session APIs

- `GET /api/admin-only/` – Admin-only test endpoint for verifying privileged access.
- `GET /api/users/` – Admin/authorized listing of users, typically paginated.
- `GET /api/sessions/` – Returns active sessions/devices for current user.
- `DELETE /api/sessions/{session_id}/` – Revokes a single session/device.
- `POST /api/sessions/revoke-all/` – Revokes all sessions for the user, forcing logout elsewhere.

---

5. Authentication Mechanism (Mandatory)

5.1 Mechanisms Used

- **JWT-based Authentication:**
  - Access tokens are provided to clients in responses and are sent via `Authorization: Bearer <ACCESS_TOKEN>`.
  - Refresh tokens are stored in **httpOnly cookies** to reduce XSS risk.
- **Username/Password Login (Web Admin Frontend):**
  - `/api/token/` accepts **username and password** (no email login flow is used for the web admin frontend) and, if valid, issues tokens.
- **OTP-based Login (API capability):**
  - OTP request/verify endpoints allow passwordless login using one-time codes, primarily for mobile or future client flows.
- **Social Login (API capability):**
  - External identity providers (e.g. Google, Apple) can be used to authenticate and receive OnTime tokens where configured. In the current web admin frontend, the active login flow is username/password.

5.2 Flow Overview

1. **Login**: Client calls `POST /api/token/` or `/api/auth/social/login/`.
2. **Token Use**: Client includes `Authorization: Bearer <ACCESS_TOKEN>` in subsequent API calls.
3. **Refresh**: When access token is close to expiry, client calls `POST /api/token/refresh/` with refresh cookie.
4. **Logout**: Client calls `POST /api/logout/`; server invalidates/blacklists tokens and clears cookies.

All authentication-required endpoints validate the JWT token and tenant header before processing.

---

6. Third-Party Integrations (Mandatory)

6.1 External Video & Playlist Sources

- **YouTube / external playlists** used in the shorts ingestion pipeline. Backend workers and admin actions may fetch and process playlist metadata and videos using the configured `YOUTUBE_API_KEY` and related tooling.

6.2 CDN / Object Storage

- Video streams (HLS), thumbnails, and logos are hosted on external storage/CDN infrastructure (e.g., object storage + CDN).

- APIs typically return URLs or metadata that reference these assets rather than serving large media directly.

6.3 Push Notifications (Firebase Cloud Messaging)

- Firebase Admin SDK is used for push notifications to registered devices via **Firebase Cloud Messaging (FCM)**.

- Device tokens are stored via the `user_sessions` app and notifications are dispatched using FCM.

6.4 Identity Providers (Optional / Environment-specific)

- **Google** and **Apple** OAuth2/OpenID Connect providers are supported by the backend social auth service.

- Only tokens from explicitly configured client IDs (via environment variables) are accepted for social login endpoints.

- Whether social login is exposed to end users depends on the specific client (mobile/web) configuration.

  Exact providers, SMTP servers, SMS gateways, and CDN/storage backends are configured via environment variables and may differ between development and production.

---

7. Compliance and Regulatory Requirements (Mandatory)

The APIs and platform are designed with the following considerations based on the current implementation and settings:

- **Data Protection & Privacy:**

  - Alignment with applicable Ethiopian data protection and privacy regulations.

  - Personal data minimization: the system primarily stores authentication data and content metadata needed for platform operation.

  - Passwords are stored using Django's secure hashing mechanisms; there is no plaintext password storage.
- **Secure Transmission:**
  - All external API traffic in production is expected to use HTTPS (TLS), enforced via `SECURE_SSL_REDIRECT`, HSTS (`SECURE_HSTS_SECONDS`, preload, subdomains), and proxy SSL settings.
- **Logging & Audit:**
  - Logging is configured to stream to standard output (`LOGGING` config) and includes security-relevant events such as authentication failures (via django-axes), tenant resolution debugging, and selected admin operations.

  - Log retention and rotation are handled at the infrastructure/hosting level (e.g., container logs, server logs); there is no hard-coded application-level log deletion policy in the codebase.
- **Rate Limiting & Brute Force Protection:**
  - DRF throttling is configured for anonymous and authenticated users (e.g., `20/hour` for anonymous, `1000/hour` for authenticated, login and registration rate limits).

  - `django-axes` provides additional brute-force protection, including lockout after repeated failed login attempts.
- **Security Best Practices:**
  - OWASP Top 10 and OWASP API Security Top 10 risks are considered in the design (e.g., use of Django ORM, CSRF protection where applicable, secure cookie flags, and strict security headers).

---

8. Authorization and Access Control (Mandatory)

8.1 Roles and Permissions

- **Anonymous Users:**
  - Very limited or no access to protected APIs (depending on configuration).
- **Authenticated Users:**
  - Access to standard end-user APIs such as `/api/me/`, shorts feed, and channel listing.
- **Admin / Editor Users:**

  - Additional permissions for managing channels, users, and configuration.
  - Access to admin-specific endpoints like `/api/admin-only/` and `/api/users/`.
- **System / Service Accounts:**
  - Used internally for integrations or workers if required.

8.2 Enforcement

- Implemented at the **Django REST Framework** layer using:
  - Authentication classes (JWT and session where applicable).
  - Permission classes that check user roles and tenant.
- Access control checks ensure that:
  - Non-admins cannot access admin endpoints.
  - Tenants cannot access data belonging to other tenants.

8.3 Tenant Awareness

- All tenant-aware APIs require `X-Tenant-Id` header to be present.
- Backend enforces data isolation between tenants based on this header and server-side configuration.

---

9. Test Accounts for API Testing (Mandatory)

The following accounts and headers are provided for security testing purposes. **Credentials here are examples**; actual test credentials should be confirmed and kept in a secure channel.

9.1 End-User Test Account

- **Username:** `adminweb`
- **Password:** `Root@1324`
- **Role:** Standard authenticated user
- **Usage:**
  - Test authentication, profile, channels list, and shorts feed APIs.

9.3 Required Headers for All Authenticated API Calls

- `X-Tenant-Id: ontime` (or appropriate tenant)
- `Authorization: Bearer <ACCESS_TOKEN>`

- `Content-Type: application/json` for JSON bodies

If additional service or system accounts exist (for automation or monitoring), they can be documented here with their intended scope and limitations.

---

10. References

- `API_ENDPOINTS.md` – Detailed per-endpoint documentation with request/response examples, rate limits, error formats, and pagination structure.
- `WEB_APPLICATION_TESTING_WEB.md` – Overall architecture and security/testing information for the web application.