Web Application Testing – OnTime Ethiopia

Organization: AI TECHNOLOGIES PLC / ኤ አይ ቴክኖሎጂስ ኃላ/የተ/የግ/ማ
Platform: OnTime Ethiopia (Web Application)
Prepared for: Information Network Security Administration (INSA)
Date: November 2025

---

1. Business Architecture and Design

1.1 Background of the Organization

AI TECHNOLOGIES PLC (ኤ አይ ቴክኖሎጂስ ኃላ/የተ/የግ/ማ) is an Ethiopian technology company focused on building digital platforms that support media, communication, and innovative services aligned with the country's digital transformation agenda. The company delivers scalable software solutions and streaming platforms for global and regional audiences.

On behalf of the OnTime Ethiopia team, we would like to extend our sincere appreciation to the Information Network Security Administration for its continuous efforts in advancing Ethiopia's digital transformation and ensuring the nation's cybersecurity resilience. The Administration's leadership in creating a secure and empowering digital environment plays a crucial role in strengthening innovation, public trust, and national development.

1.2 Introduction

This document provides technical and security-related information about the OnTime Ethiopia web application. It is intended to support the Information Network Security Administration (INSA) in planning and executing a penetration test / security assessment of the platform.

1.3 Objective of this Certificate Requested

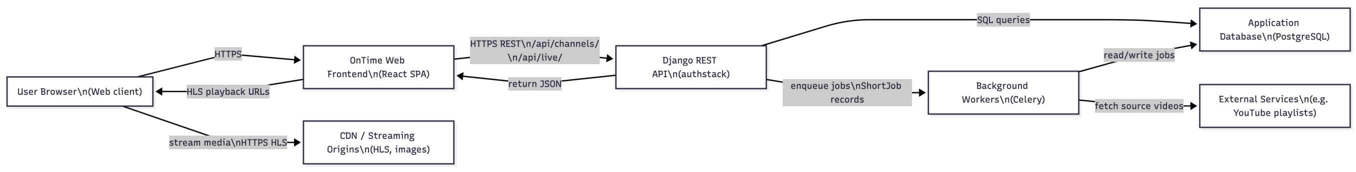
The objective is to obtain an independent security assessment and certificate for the OnTime Ethiopia web application, verifying that it has been tested against common web application security vulnerabilities and that identified issues can be remediated to strengthen the platform's security posture.

1.4 Business Architecture (Summary)

- **Product:** OnTime Ethiopia – an OTT/streaming platform providing live TV, shorts, and related content to end users.
- **Core actors:**
  - End users (web and mobile) consuming video content.
  - Admins/editors using an internal web portal for content management and configuration.
- **Core backend:** Django REST Framework application exposing REST APIs for channels, live TV, shorts ingestion, playlists, and authentication.
- **Frontend:** React-based web frontend consuming these APIs over HTTPS.

1.5 Data Flow Diagram (Text Description)

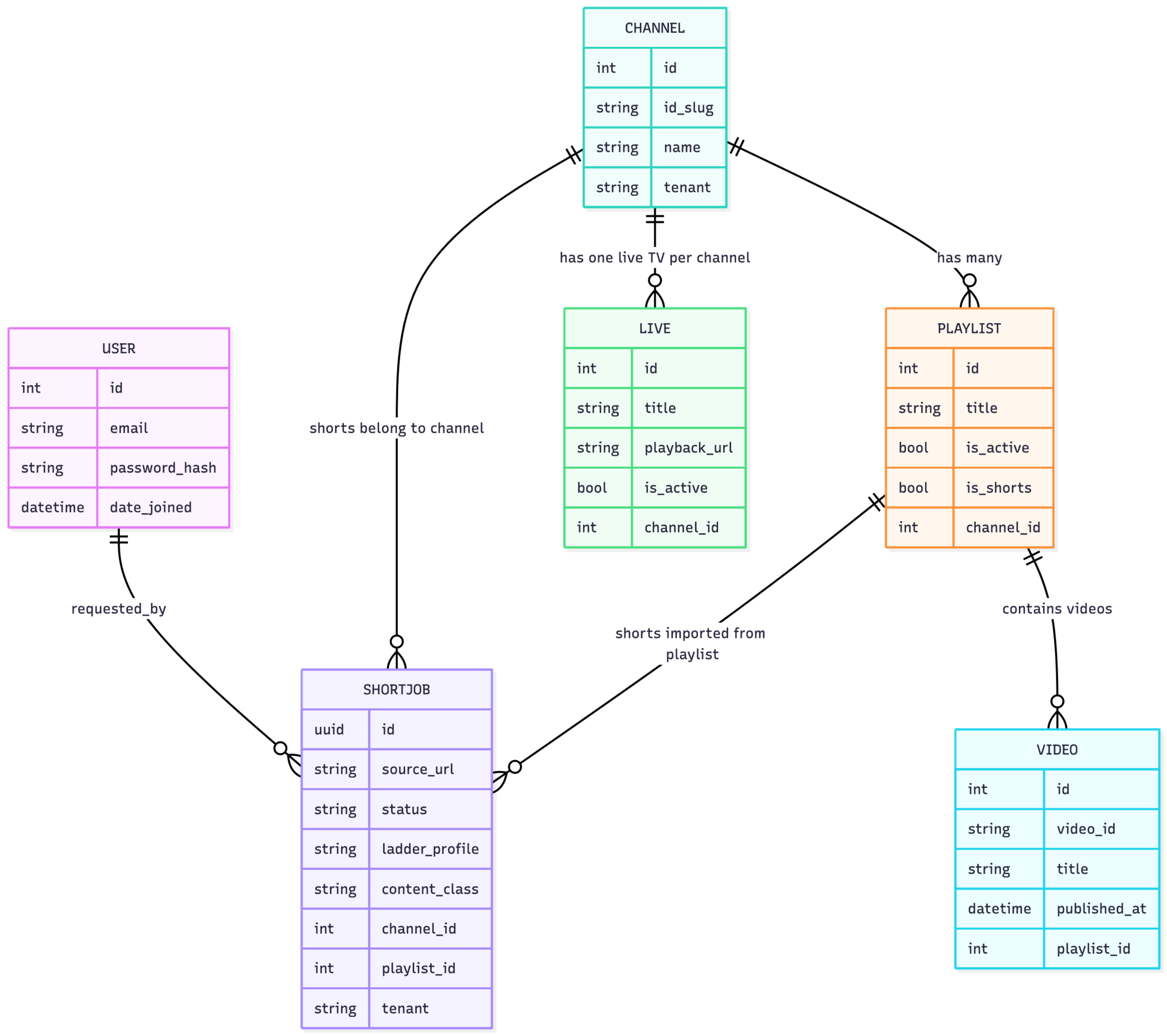
1. User’s web browser connects over HTTPS to the OnTime Ethiopia web frontend.
2. The web frontend sends HTTPS REST API requests to the Django backend (e.g. `/api/channels/`, `/api/live/`, `/api/channels/shorts/ready/feed/`).
3. The backend reads/writes data from the application database (e.g. PostgreSQL) storing channels, playlists, live metadata, users, shorts jobs, etc.
4. Celery/background workers process shorts ingestion jobs and interact with external services (e.g. YouTube playlists, storage/CDN) and update job status in the database.
5. The frontend receives JSON responses and renders UI accordingly, while media (HLS streams, images) is delivered from streaming endpoints/CDN over HTTPS.


User Browser\n(Web client)
HTTPS
HLS playback URLs
stream media\nHTTPS HLS
OnTime Web Frontend\n(React SPA)
CDN / Streaming Origins\n(HLS, images)
HTTPS REST\n/api/channels/ \n/api/live/
return JSON
Django REST API\n(authstack)
SQL queries
enqueue jobs\nShortJob records
Background Workers\n(Celery)
read/write jobs
fetch source videos
Application Database\n(PostgreSQL)
External Services\n(e.g. YouTube playlists)

1.6 System Architecture Diagram (Text Description)

- **Client Layer**
  - Web browser (desktop/mobile) running the OnTime Ethiopia web SPA.
- **Application Layer**
  - Django REST API application (gunicorn/uWSGI) behind a reverse proxy/load balancer.
  - Celery workers for background processing of shorts ingestion and media-related tasks.
- **Data Layer**
  - Relational database (e.g. PostgreSQL) for core application data.
  - Object storage / CDN for video and image assets.
- **Security/Perimeter**
  - Reverse proxy/load balancer providing TLS termination and basic protection.
  - Network firewall restricting non-essential ports.
  - (If applicable) Additional WAF or cloud-based protections.

1.7 Database Entity-Relationship Diagram



---

2. Features of the Web Application

2.1 Development Frameworks

- **Frontend:** React-based single-page application (SPA).
- **Backend:** Django + Django REST Framework (DRF).
- **Task Queue:** Celery for asynchronous processing (e.g. shorts ingestion and transcoding).

2.2 Libraries (Plugins)

- **Frontend (examples):**
  - React Router for client-side routing.
  - HTTP client (e.g. Axios/fetch wrapper) for API calls.
  - UI component libraries for layout, tables, and forms.
- **Backend (examples):**
  - Django REST Framework for API endpoints.

  - Celery + message broker (e.g. Redis/RabbitMQ) for background tasks.
  - yt-dlp and related tooling for shorts ingestion from external playlists.

2.3 Third-Party Integrations

- External video sources (e.g. YouTube playlists) for shorts ingestion.
- CDN / storage for hosting and delivering HLS playlists and media assets.
- Optional identity providers (e.g. Google sign-in) for authentication, depending on environment.

2.4 Custom Development

- Custom admin web UI to manage channels, playlists, live TV metadata, shorts ingestion jobs, and feature flags.
- Custom Django REST endpoints for:
  - Live TV channels and slug-based lookup.
  - ShortJob ingestion pipeline (import, queue, process, mark READY, randomized feed).
  - Tenant-aware content separation.

2.5 Actor / User Types

- **Anonymous users:** Can access public landing pages (depending on configuration).
- **Authenticated end users:** Can log in to the OnTime Ethiopia platform and consume authorized content.
- **Admin / Editor users:** Manage content, playlists, shorts, and configurations via the internal web admin portal.
- **System / service accounts:** Used by backend workers and integrations.

2.6 Dependencies / System Minimum Requirements (Web Frontend)

- **Client:** Modern browser with HTML5 support and JavaScript enabled.
- **Network:** Stable internet connection for media streaming over HTTPS.
- **Server:** Linux-based server with Python (Django), database (e.g. PostgreSQL), and message broker (e.g. Redis/RabbitMQ); Node.js toolchain for building the frontend.

2.7 Implemented Security Standards

- Web traffic served over HTTPS in production environments.
- Authentication and authorization enforced for admin and protected operations.
- Role-based access control (RBAC) for admin features.
- CSRF protection for state-changing requests when using cookie-based sessions.
- Input validation and server-side validation on critical API endpoints.
- Consideration of OWASP Top 10 risks in design and implementation.

2.8 Devices that May Impact Penetration Testing

- **Firewall:** Network firewall restricting inbound access to required ports (e.g. 80/443, SSH).
- **Reverse Proxy / Load Balancer:** May enforce rate limiting, TLS policies, and IP filtering.
- **Web Application Firewall (WAF) (if applicable):** May inspect and block certain payloads or exploit attempts.

Penetration testers should be aware of these controls, as they may affect test results.

---

3. Scope of Testing (Web Application)

Name of the Assets to be Audit	URL/IP	Test Account as required by the tester
Public web portal/sites	https://aitechnologiesplc.com/	Public browsing (no login) as applicable
Public application (OnTime)	https://ontime.aitechnologiesplc.com/	End-user test account (adminweb/Root@1324)
Internal web portal	https://ontime.aitechnologiesplc.com/	Admin/editor test account with appropriate admin privileges
Internal local applications	N/A or not in scope (unless otherwise specified)	To be provided if any additional internal-only apps are included

Notes: The same base URL may expose both public and authenticated functionality depending on login status and user role.

---

4. Security Functionality

Security Functionality	How the Security Functionality Has Been Implemented
User Roles and Access Control	Roles are defined at the backend, distinguishing end users from admin/editor accounts. Admin features are only accessible to authenticated users with appropriate permissions, enforced in Django viewsets and permissions.
Authorization Mechanisms	Protected API endpoints require authenticated requests (session or token-based). Authorization checks are enforced server-side to ensure that only authorized users can access or modify protected resources.
Input Validation and Sanitization	Client-side form validation is combined with server-side validation in Django serializers/models. Inputs are validated and sanitized before being processed or stored to reduce risks such as injection and malformed data.
Session Management	Authentication is managed by Django’s authentication/session framework and/or token-based mechanisms. Sessions have defined lifetimes, and users can log out to invalidate their sessions. Secure cookie attributes are used where applicable.
Error Handling and Logging	In production, API error responses avoid exposing sensitive internal details. Detailed errors and exceptions are logged server-side along with important security events (such as authentication failures and admin actions).
Secure Communications	All frontend–backend communication is conducted over HTTPS in production deployments. TLS is handled at the reverse proxy/load balancer, and mixed-content access is avoided where possible.

5. Secure Coding Standards

5.1 Coding Rules / Guidelines

- General alignment with OWASP Secure Coding Practices.
- Use of Django ORM instead of raw SQL to reduce SQL injection risks.
- Avoidance of unsafe JavaScript constructs (e.g. `eval`) in the frontend.
- Code review practices for sensitive or security-relevant changes.

5.2 Secure Coding Practices Used

- **OWASP Secure Coding Practices**
  - Consideration of OWASP Top 10 risks (authentication, access control, injection, XSS, CSRF, etc.) in design and implementation.
- **Secure Input Handling**
  - Server-side validation for all critical inputs via Django forms/serializers.
  - Client-side validation to enhance user experience, but not relied upon for security.
- **Secure File Handling**
  - Validation of uploaded files (e.g. type/size constraints).
  - Storage of media files in dedicated locations separate from application code.
- **Authentication and Session Management**
  - Use of Django's authentication mechanisms for handling user credentials and sessions.
  - Passwords stored using secure hashing algorithms (no plaintext storage).
  - Session timeout and logout capabilities provided to users.
- **Software Patching and Updates**
  - Regular updates of Python/Django, React, and third-party libraries using dependency management tools (pip, npm/yarn).
  - Monitoring of third-party libraries for security advisories and vulnerability patches.
- **Review of Third-Party Components**
  - Preference for well-maintained, community-vetted libraries.
  - Periodic review of third-party components and removal or replacement of unmaintained libraries when feasible.
- **Secure Communication**
  - Enforcement of HTTPS for production environments to protect data in transit.
  - Proper TLS configuration at the reverse proxy/load balancer level.

---

6. Previous Security Test Reports

At the time of preparing this document, there is no formal previous external security test report for this specific web application. Therefore:

- First phase security testing document: Not available / Not previously conducted.
- Re-audit security testing document: Not available / Not previously conducted.

If prior internal security reviews or tests exist, they can be shared separately as needed.

---

7. Contact Information and Communication Channel

Company: AI TECHNOLOGIES PLC / ኤ አይ ቴክኖሎጂስ ኋላ/የተ/የግ/ማ

Name	Role	Address (Email and Mobile)
Elnatan Nebiyou	Developer	Email: elnatan.nebiyu@gmail.com
		Mobile: +251911429639