Project ATTN - Status Report

Date: June 26, 2025
Subject: Completion of Phase 7: Production Hardening & Finalization
Status: COMPLETE

1. Overview

This report certifies the successful completion of all objectives outlined in Phase 7. The backend application has been significantly enhanced with production-grade features, making it secure, stable, and observable. The system is now technically complete and ready to proceed to the final deployment phase.

All development goals for this phase were met through a methodical approach focused on planning, modular implementation, and rigorous testing, which ensured that architectural integrity was maintained throughout the process.

2. Completed Objectives & Implementation Details

Objective 1: Integrate Periodic Tasks

- **Task:** Automate critical background maintenance jobs, such as cleaning up expired Redis sessions and processing asynchronous tasks.
- **Implementation Plan:**
  - The initial fastapi-scheduler was successfully replaced with the more robust, industry-standard **apscheduler** library to handle all cron jobs.
  - The AsyncIOScheduler was integrated directly into the application's lifespan context manager, ensuring tasks start and stop gracefully with the application.
  - Jobs for sweeping user sessions, archiving attendances, and checking face verification results are now fully automated and running on schedule.

Objective 2: Implement a Robust Rate Limiting System

- **Task:** Introduce strategic request limits on API endpoints to **prevent brute-force and Denial-of-Service (DoS) attacks.**
- **Implementation Plan:**
  - A RATE_LIMITER_REDIS_URL was added to the .env file, pointing to a separate Redis database (/1) to isolate rate-limiting data.
  - The **slowapi** library was integrated. The limiter logic was refactored into a dedicated module (app/backend/api/utilities/limiter.py) to resolve circular dependencies and improve code organization.
  - A dynamic key function was implemented, applying limits based on the user's **IP address** for unauthenticated requests (e.g., login) and by the **user's school number** for authenticated requests, providing a fair and secure policy.

    - Specific, debated, and agreed-upon limits have been applied to all relevant endpoints in the auth, teacher, and student API routers.

Objective 3: Establish a Detailed Logging System

- **Task: Log critical events** (e.g., user logins, attendance session changes) and **application errors** in a **standardized format** to facilitate monitoring and debugging.
- **Implementation Plan:**
    - A central logging configuration was created in app/backend/logging/logging_config.py, which formats all logs consistently and outputs them to both the console and a rotating file (logs/app.log).
    - Meaningful logger.info(...) calls were added to the service layers (teacher_service.py, student_service.py) and external modules (aksis.py) to trace business logic and key events.
    - All critical operations (database queries, Redis commands, external API calls) have been wrapped in robust try...except blocks.
    - logger.error("...", exc_info=True) calls have been implemented within這些 except blocks to capture detailed stack traces for any unexpected errors, making the application "500-proof" and significantly easier to debug.

3. Current Status & Next Steps

With the completion of Phase 7, the backend service is now fully prepared for frontend integration and production deployment.

The project is now officially entering **Phase 8: Deployment**. The next immediate steps will involve:

- Integrating **Nginx** as a reverse proxy.
- Configuring the production environment to serve the application under its official domain name.
- Implementing SSL/TLS to secure all traffic with **HTTPS**.