Project: ATTN

Phase 3: Configuration, Tools & Modules - Completion Report

Date: June 22, 2025

1. Overview

Phase 3 of the ATTN project, focused on building the application's core utilities and external communication modules, is now complete. This phase successfully established a robust configuration system and implemented the critical logic for interacting with the external Aksis system, as well as internal verification tools.

All development was guided by a Test-Driven Development (TDD) approach, ensuring that each module is reliable, well-tested, and ready for use by the upcoming service layer.

2. Key Accomplishments & Artifacts

2.1. Centralized Configuration

A secure and robust configuration system was established to manage all application settings.

- **.env File:** All sensitive and environment-specific variables (database URLs, external service URLs, credentials) were centralized in a .env file.
- **config.py:** A dedicated configuration module (app/backend/config/config.py) was created to load these variables into a single, easily accessible settings object, ensuring clean and consistent access to configuration throughout the application.

2.2. Module Implementation (with TDD)

The core modules for handling external data and internal business rules have been completed.

- **lesson_finder.py:**
  - A function to parse the complex JSON data from the Aksis schedule was implemented.
  - The logic was successfully adapted to find all lessons for a given day, providing more flexibility than the original "active lesson" plan.
  - The module was validated with unit tests to ensure correct parsing and filtering.
- **aksis.py:**
  - A comprehensive AksisClient was built to handle the entire lifecycle of interacting with the university's web portal.

    - Functionality includes logging in, handling authentication tokens and redirects, scraping user profile data, and fetching the daily lesson schedule.
    - The client was validated with a full suite of **integration tests** against the live Aksis server, confirming its real-world functionality.

2.3. Tools Implementation (with TDD)

The reusable verification tools for the application have been built and tested. The verifier.py module was split into two more focused modules for clarity:

- **wifi_verifier.py:**
    - A simple but effective function to compare a user's IP address with the session's required IP was created.
    - This was validated with an integration test using a live FastAPI test server to simulate a real-world request and confirm the IP check logic.
- **face_verifier.py:**
    - A two-step client for interacting with the face verification microservice was implemented (verify_face_submit_job and verify_face_get_result).
    - This module was validated with a robust integration test that submits multiple jobs to the live microservice and polls for their results, ensuring the asynchronous workflow is handled correctly.

3. Current Status

Phase 3 is **100% complete**. All foundational tools and external communication modules are implemented, tested, and ready.

4. Next Steps

We are now ready to begin **Phase 4: Background Tasks**. We will create the tasks directory and implement the Celery responsible for resource and connection handling effificently and FastAPI Scheduler cron jobs (sweep_attendances_task, sweep_users_task) responsible for periodically cleaning Redis and moving data to the PostgreSQL database.