Project: ATTN

Phase 6: Core Backend Development - Completion Report

Date: June 25, 2025

This document summarizes the successful completion of Phase 6, which involved building out the API layer and the corresponding end-to-end tests. It also outlines the roadmap for the final stage, **Phase 7: Production Hardening & Finalization**.

✅ Phase 6: Core Backend Development - Completed!

In this phase, all core business logic, the corresponding API endpoints, and the end-to-end tests required to validate the application's main functionality have been successfully completed.

- **Key Accomplishments:**
  - **Teacher & Student APIs (teacher.py, student.py):** All necessary endpoints for both user roles are fully functional.
  - **Reliable Authentication (auth.py):** A secure and robust authentication system is in place.
  - **Complete Service Layer (teacher_service.py, student_service.py):** The business logic has been fully implemented and separated from the API layer.
  - **Comprehensive E2E Testing (test_teacher.py, test_student.py):** The system has been rigorously tested across all scenarios, including all security levels, IP verification, and the face recognition workflow.
- **Outcome:** The project's backend is now **functionally complete.**

🚀 Next Steps: Phase 7 - Production Hardening & Finalization

The objective of this final phase is to make the completed backend robust, secure, and efficient for a live production environment.

Primary Objectives:

1. **Integrate Periodic Tasks (cron.py):**
   - **Task:** Implement the defined maintenance (e.g., archiving old Redis records to the database) and functional (e.g., processing face verification results) cron jobs within the main application (main.py).
   - **Tooling:** A library such as FastAPI-Scheduler or APScheduler can be utilized for scheduling.
2. **Implement Rate Limiting:**
   - **Task:** Introduce request limits on sensitive endpoints (especially login) to

prevent brute-force and Denial-of-Service (DoS) attacks.
    - **Implementation Plan:**
        - Add a RATE_LIMITER_REDIS_URL to the .env file, pointing to a separate Redis database (e.g., /1).
        - Read this variable via config.py.
        - Apply Depends(RateLimiter(...)) to the relevant endpoints using the fastapi-limiter library.
3. **Establish a Detailed Logging System:**
    - **Task:** Log critical events (e.g., user logins, attendance session changes) and application errors (e.g., 500 Internal Server Errors) in a standardized format to facilitate monitoring and debugging.
    - **Implementation Plan:**
        - Configure Python's standard logging module.
        - Ensure logs are output to both the console and a file (app.log) in the production environment.
        - Add logger.info(...) and logger.error(...) calls to key service functions and except blocks.

Upon completion of this phase, the backend service will be fully prepared for frontend integration and production deployment.