Project: ATTN

Phase 5: Service Layer Implementation - Completion Report

Date: June 23, 2025

1. Overview

Phase 5 of the ATTN project, focused on implementing the crucial Service Layer, is now fully complete. This phase successfully built out the business logic for student attendance, including advanced asynchronous verification workflows, ensuring a robust and flexible system. All development in this phase continued to adhere to a Test-Driven Development (TDD) approach, guaranteeing reliability and correctness.

The primary objective was to define how the application's core functions, particularly student attendance, operate by orchestrating interactions between the data access layer (Redis and PostgreSQL clients), external modules (like Wi-Fi and Face Verifiers), and background tasks.

2. Key Accomplishments & Artifacts

2.1. Enhanced StudentService (app/backend/services/student_service.py)

The StudentService has been significantly enhanced to handle student attendance requests with dynamic verification requirements:

- **attend_to_attendance Logic Refinement**: The core logic for students attending a session was implemented to dynamically adapt based on security_option values (1: Manual Only, 2: Manual + Wi-Fi Check, 3: Wi-Fi Check + Face Recognition).
- **Wi-Fi Verification Integration**: Integrated synchronous Wi-Fi verification (verify_wifi) to immediately mark attendance as failed if Wi-Fi conditions are not met, based on security_option 2 and 3.
- **Asynchronous Face Verification Integration**: For security_option 3, the service now:
  - Initiates an asynchronous face verification job by calling verify_face_submit_job.
  - Sets the AttendanceRecord's initial status to is_attended=False and fail_reason="FACE_RECOGNITION_PENDING".
  - Adds the student's school number, attendance ID, and the returned job_id to a dedicated Redis queue (user_waiting_for_face_verification) for later processing by a cron job.
- **Detailed Failure Reasons**: Records specific fail_reason values (e.g., "WIFI_FAILED", "FACE_RECOGNITION_PENDING",

"FACE_VERIFICATION_SUBMISSION_FAILED", "WIFI_REQUIRED_BUT_IP_MISSING", "FACE_VERIFICATION_REQUIRED_BUT_IMAGES_MISSING") for improved auditability and teacher insight.

2.2. Redis Client Extensions for Asynchronous Flow (app/backend/db/redis_client.py)

To support the asynchronous face verification, the RedisClient was extended with new methods:

- **add_user_to_face_verification_queue**: Adds an entry to a Redis Hash to track pending face verification jobs. The key stores user_school_number and attendance_id, and the value stores the job_id received from the external microservice.
- **remove_user_from_face_verification_queue**: Deletes a completed or problematic job entry from the Redis queue.
- **get_users_in_face_verification_queue**: Retrieves all pending jobs from the queue, allowing the cron task to process them.
- **get_attendance_by_id and check_lesson_status adjustments**: Ensuring consistent and correct fetching of attendance sessions and their index data.

2.3. New Background Task for Face Verification (app/backend/tasks/cron.py)

A critical new cron job was implemented to manage the asynchronous verification lifecycle:

- **check_face_verification_results_task**: This task periodically:
  - Queries the user_waiting_for_face_verification queue in Redis.
  - Polls the external face verification API Gateway using verify_face_get_result with the stored job_id.
  - Parses the microservice's detailed response (SUCCESS/FAILURE/PENDING/IN_PROGRESS) to determine the final is_verified status.
  - Updates the corresponding AttendanceRecord in Redis to reflect the actual verification outcome (setting is_attended to True or False and updating fail_reason).
  - Removes the processed job from the Redis queue.
  - Handles various error scenarios (e.g., microservice communication errors, malformed responses) by logging and keeping jobs in the queue for retry if appropriate.

2.4. Comprehensive Test Coverage (tests/services/test_student_service.py, tests/db/test_redis_client.py, tests/tasks/test_cron.py)

Extensive unit and integration tests were developed or updated to cover all new and modified functionalities:

- **test_student_service.py**: New tests validate attend_to_attendance across all security_option scenarios, including Wi-Fi success/failure, face verification submission (pending state), and submission failures. Mocking was used to isolate the StudentService from external dependencies (verify_wifi, verify_face_submit_job).
- **test_redis_client.py**: Added dedicated tests for add_user_to_face_verification_queue, remove_user_from_face_verification_queue, and get_users_in_face_verification_queue to ensure correct Redis interactions.
- **test_cron.py**: New tests for check_face_verification_results_task cover successful verification, failed verification, pending states, gateway errors, tool errors, and scenarios with missing records. Existing sweep_users_task and sweep_attendances_task tests were also reviewed and adjusted to ensure compatibility with updated Redis client signatures.

3. Current Status

Phase 5 is **100% complete**. All core business logic for student attendance, including the intricate asynchronous face verification workflow and its supporting components, has been successfully implemented and rigorously tested. The system is now capable of managing attendance records with detailed status tracking and automated verification updates.

4. Next Steps

With the robust backend services in place, we are fully prepared to begin **Phase 6: The API Layer**. This will involve creating the FastAPİ endpoints to expose the functionality developed in the service layer to the client applications (frontend, mobile, etc.), completing the full application stack.