Project ATTN: Architectural Refactoring Report - V2

Date: June 29, 2025
Status: Completed
Author: Project ATTN Development Team

1. Executive Summary

This report documents a significant architectural refactoring of the Project ATTN backend system. The primary motivation for this initiative was to address complexities in the caching layer and data persistence logic that posed risks to data integrity and maintainability.

The core of the refactoring involves consolidating the Redis data models to create a single, authoritative source of truth for live attendance sessions. This simplification cascades through all layers of the application—from the data access layer to the API endpoints—resulting in a more robust, resilient, and maintainable system.

The refactoring was executed in four distinct phases:

1. **Data Model & Data Access Layer (DAL) Simplification:** Overhauling the core data structures in Redis and enforcing stricter architectural boundaries.
2. **Service Layer Adaptation:** Aligning the application's business logic with the new, simplified data models.
3. **Cron Job Robustness Enhancement:** Re-engineering the data persistence task to guarantee data integrity and atomicity.
4. **API Endpoint & Schema Alignment:** Updating the public-facing API to reflect backend changes and improve client-side efficiency.

This document details the specific changes made in each phase and outlines the resulting improvements.

2. Phase 1: Data Model and Data Access Layer (DAL) Simplification

Objective: To eliminate data redundancy in the caching layer and enforce a clean separation of concerns between the database client and the caching client.

2.1. Redis Data Model (models/redis_models.py)

The foundational change of this refactoring was the consolidation of our Redis data models.

- **DELETED:** The LessonRedis model was completely removed. Its purpose was to

make lessons discoverable, but it created data duplication and required complex logic to keep it synchronized with the full attendance session data.
- **ADDED:** A new, comprehensive AttendanceRedis model was introduced. This model serves as the **single source of truth** for an active attendance session in Redis. It is a superset of the db_models.Attendance and includes the teacher_full_name, which was previously stored in LessonRedis.

This change ensures that all information required to manage, discover, and persist a live session is stored in a single, unified object.

2.2. Redis Client (db/redis_client.py)

The RedisClient was significantly refactored to support the new data model.

- **REMOVED:** All methods related to LessonManagement (save_lesson, get_lesson, find_lessons_by_name, find_lessons_by_teacher, delete_lesson) were deleted.
- **MODIFIED:** The Full Attendance Session Management methods now operate exclusively on the new AttendanceRedis model.
- **ADDED:** Two new methods were implemented to replace the functionality of the deleted lesson methods:
  - get_attendance_sessions_by_name(lesson_name, teacher_name): Provides an efficient way for students to find active sessions.
  - get_attendance_session_of_teacher(teacher_school_number): A crucial utility to enforce the business rule that a teacher can only have one active session at a time.

2.3. Database Client (db/db_client.py)

A critical architectural improvement was made to enforce stricter boundaries.

- **Enforced Strict Data Typing:** The AsyncPostgresClient was modified to **only accept and return db_models**. It is no longer aware of any Redis-specific models (like AttendanceRecordRedis).
- **Logic Relocation:** The logic for creating a User record from an AttendanceRecordRedis object within the add_attendance_records method was removed. This responsibility has been shifted to the cron job, which is the appropriate layer to handle the synchronization of data from the cache to the persistent database. This change decouples the database client from the caching strategy, making it more reusable and easier to test.

3. Phase 2: Service Layer Adaptation

Objective: To align the application's business logic with the newly refactored data

models and DAL methods.

3.1. Teacher Service (services/teacher_service.py)

The TeacherService was updated to leverage the simplified data structures.

- start_attendance: Now uses get_attendance_session_of_teacher for its pre-flight check and creates a single AttendanceRedis object, simplifying the session startup process.
- finish_attendance: No longer needs to perform separate cleanup of a LessonRedis object. It now operates directly on the AttendanceRedis session.
- get_live_attendances_by_teacher & get_and_verify_attendance_owner: These methods were updated to use the new, more direct lookup methods in the RedisClient.

3.2. Student Service (services/student_service.py)

The StudentService was adapted to the new lookup mechanisms.

- find_active_lessons_by_name: Was updated to call the new redis_client.get_attendance_sessions_by_name method.
- attend_to_attendance: Now retrieves all necessary session details (like security_option) from the consolidated AttendanceRedis object, making the logic more direct.

4. Phase 3: Cron Job Robustness Enhancement

Objective: To re-engineer the data persistence task to be more resilient, reliable, and to guarantee data integrity.

4.1. Unified Persistence Task (tasks/cron.py)

The unified_persistence_task was entirely rewritten with a more robust workflow, leveraging the consolidated AttendanceRedis model. The new, safer process is as follows:

1. **Scan for Expired Sessions:** The task scans Redis for attendance_session:* keys whose end_time has passed.
2. **Single Source of Truth:** For each expired session, the corresponding AttendanceRedis object is fetched. This object contains all data needed for persistence.
3. **Guarantee Teacher Existence:** The teacher's information (user_school_number, user_full_name) is extracted from the AttendanceRedis object, and a User record is upserted into the database. This critical step ensures the foreign key constraint will be met when the Attendances record is saved.

4. **Fetch Student Records:** All associated AttendanceRecordRedis objects are fetched.
5. **Persist Session:** The main Attendance session is saved to the PostgreSQL database.
6. **Persist Student Records:** The list of AttendanceRecords is passed to the db_client, which first upserts any new User records for students before saving the attendance records themselves.
7. **Atomic Cleanup:** All Redis keys related to the processed session (AttendanceRedis key and all AttendanceRecordRedis keys) are deleted in a single batch operation.

This new workflow eliminates the previous risks of race conditions and foreign key violations, making the data synchronization process significantly more reliable.

5. Phase 4: API Endpoint and Schema Alignment

Objective: To update the API layer to reflect the backend refactoring and improve the experience for API consumers.

5.1. API Endpoints (api/teacher.py, api/student.py)

The API controller logic was cleaned up to better adhere to the separation of concerns.

- The get_live_attendances endpoint in teacher.py was simplified. It no longer contains logic for iterating through indices and making direct calls to the redis_client. Instead, it makes a single call to the appropriate service-layer method, which now encapsulates this logic.
- Student-facing endpoints were updated to use the new service methods for finding and interacting with attendance sessions.

5.2. API Schemas (api/schemas/)

The response models were updated to be more efficient for clients.

- **AttendanceResponse:** The teacher_full_name field was added to this schema. Since this data is now readily available in the AttendanceRedis object, it can be returned directly to the client. This is a significant efficiency gain, as it prevents the client from needing to make a separate API call to resolve the teacher's name.

6. Conclusion and Summary of Improvements

This architectural refactoring initiative has successfully achieved its goals. The system is now defined by a simpler, more coherent data flow. The key benefits realized are:

- **Simplified Architecture:** The elimination of the LessonRedis model and the consolidation of live session data into a single object (AttendanceRedis) have significantly reduced the system's complexity.
- **Enhanced Data Integrity:** The re-engineered cron job and stricter DAL boundaries guarantee the correct order of operations, preventing race conditions and database foreign key violations.
- **Improved Maintainability:** With a clearer separation of concerns and less complex logic, the codebase is easier to understand, debug, and extend.
- **Increased Robustness:** The application is now more resilient to errors, particularly in the critical process of persisting temporary session data to the long-term database.

This refactoring represents a critical step in the maturation of the Project ATTN architecture, resulting in a more stable, scalable, and high-quality product.