Architectural Refactoring and Evaluation Report: Project ATTN

Date: June 27, 2025
Status: Analysis of Proposed Architectural Changes

1. Executive Summary

This report provides a comprehensive evaluation of the current architecture of the **Project ATTN** application and analyzes a set of proposed architectural refactoring changes. The initial design of the application is strong, employing a modern technology stack and adhering to sound software engineering principles. However, a critical design flaw related to data synchronization between Redis and PostgreSQL has been identified, which compromises data integrity.

The proposed changes directly and effectively address this core issue by redesigning the data persistence workflow. By unifying cron jobs, enriching the data held in Redis, and leveraging native Redis features for session management, the new architecture will eliminate the critical race condition, significantly simplify the system's logic, and enhance its overall robustness and efficiency.

Our evaluation concludes that the proposed refactoring is an exceptional improvement. The initial architecture is rated **75/100**, reflecting a strong but flawed foundation. The architecture, post-implementation of the proposed changes, is rated **95/100**, representing a stable, scalable, and production-ready system.

2. Current State Analysis (Pre-Refactoring)

The current architecture of Project ATTN is built on a solid foundation, leveraging FastAPI, PostgreSQL, and Redis. It features a clean, multi-layered design and a commendable commitment to Test-Driven Development (TDD).

Strengths:

- **Modern Technology Stack:** The choice of FastAPI for the API, PostgreSQL for persistent storage, and Redis for caching and session management is a robust, high-performance, and industry-standard stack.
- **Clean Architecture:** The separation of concerns into distinct API, Service, and Data Access layers facilitates maintainability and testing.
- **Development & Deployment:** The use of Docker, docker-compose, and a plan for Gunicorn deployment demonstrates a mature approach to creating a consistent and stable production environment.

Identified Weakness & Critical Flaw:

- **Data Integrity Race Condition:** The primary weakness lies in the data persistence strategy. The system uses two separate, asynchronous cron jobs:
    1. sweep_users_task: Periodically saves user data from Redis to PostgreSQL.
    2. sweep_attendances_task: Periodically saves attendance records from Redis to PostgreSQL.

This decoupled design creates a critical race condition. The sweep_attendances_task can, and does, attempt to insert an AttendanceRecord into PostgreSQL for a student whose User record has not yet been saved by the sweep_users_task. This results in a **Foreign Key constraint violation**, causing data loss and undermining the core reliability of the application.

Initial Score: 75/100

3. Proposed Architectural Refactoring

A series of well-reasoned changes have been proposed to resolve the identified issues and improve the system.

Key Changes:

1. **Enriched Redis Data Models:** A new AttendenceRecordRedis model will be introduced. Critically, this model will store the student's username and other necessary user details directly within the attendance record in Redis. This denormalization ensures each attendance record is a self-contained unit of information.
2. **Unified and Atomic Cron Job:** The sweep_users_task will be eliminated. The logic will be consolidated into the sweep_attendences task. This refactored job will now have a single responsibility:
    - For each finished attendance session, it will first process the student records.
    - Using the enriched data from AttendenceRecordRedis, it will perform an "upsert" operation for each student (e.g., INSERT ... ON CONFLICT DO NOTHING), guaranteeing the student record exists in PostgreSQL.
    - Only after ensuring the user records are present will it proceed to insert the Attendance and AttendanceRecord data. This fundamentally resolves the race condition.
3. **Efficient Session Management via Redis TTL:** The need for a user-sweeping cron job is removed entirely by leveraging Redis's native Time-To-Live (TTL) feature. User session data will be set with an automatic expiration, offloading the cleanup task to Redis itself, which is a more efficient and standard practice.
4. **New Feature Integration:**

    - **Manual Failure:** Functionality will be added for teachers to manually mark a student as failed during a live attendance session.
    - **Admin Service:** An admin_service and corresponding router will be created, laying the groundwork for future administrative capabilities.

4. Impact Analysis and Evaluation (Post-Refactoring)

The proposed changes are a significant architectural improvement that will have a profoundly positive impact on the application.

Key Benefits:

- **Data Integrity Guaranteed:** By unifying the data persistence logic into a single, atomic operation, the foreign key violation error is completely eliminated. The system's data will be consistent and reliable.
- **Architectural Simplification:** Removing a redundant cron job and complex inter-job dependencies makes the system easier to understand, maintain, and debug. The data flow becomes linear and predictable.
- **Increased Efficiency:** Using Redis TTL for session management is far more performant than running a periodic Python script to scan and delete keys. It reduces computational overhead on the application server.
- **Enhanced Scalability:** A simpler, more robust architecture is inherently more scalable. The removal of race conditions means the system can handle a higher load without succumbing to data synchronization errors.

Post-Changes Score: 95/100

5. Recommendations for Further Enhancement

The proposed refactoring is excellent. To further harden the system for production, the following suggestions should be considered:

- **Idempotent Job Design:** Formally ensure the sweep_task is idempotent by using database-level conflict resolution (e.g., INSERT ... ON CONFLICT DO NOTHING). This prevents duplicate data if the job is ever re-run over the same time window.
- **Batch Error Handling:** Within the unified cron job, wrap the processing for each individual attendance session in a try...except block. This ensures that a single corrupted record does not cause the entire batch processing task to fail.
- **Configuration-Driven Settings:** Externalize settings like Redis TTL values into environment variables (.env file) rather than hardcoding them. This allows for flexible configuration across different environments (development, staging, production).
- **Monitoring and Alerting:** Implement monitoring for the sweep_task. Log a

summary upon completion (e.g., "Processed X sessions, saved Y records"). Configure alerts to notify administrators if the job fails, as it is a critical component for data persistence.
- **Update Documentation:** Given the significance of this architectural change, it is crucial to update all design documents, architecture diagrams, and sequence diagrams to reflect the new, simplified data flow.

6. Conclusion

The proposed architectural refactoring addresses the most critical flaw in the Project ATTN application with an elegant and robust solution. The shift from a complex, multi-process synchronization model to a simplified, atomic data persistence workflow represents a significant maturation of the system design.

By implementing these changes, Project ATTN will evolve from a well-designed but flawed application into a reliable, efficient, and scalable platform ready for production deployment. The development team is commended for identifying the root cause of the issue and proposing a high-quality, architectural solution.