Project: ATTN

Phase 1: Foundation and Infrastructure Setup - Completion Report

Date: June 21, 2024

starts_after:"created __init__.py file for folders to make python understand them as packages,created .gitignore and docerignore files,and also created schema.sql to organize our db" commit

ends_after:"added redis objects too with redis_model.py" commit

1. Overview

Phase 1 of the ATTN project, focused on establishing a robust and scalable foundation, is now complete. The primary objective of this phase was to translate the architectural design into tangible, foundational artifacts, including the database schema, a containerized development environment, and strictly-defined data models. All goals for this phase have been successfully met, paving the way for the implementation of the data access layer in Phase 2.

2. Key Accomplishments & Artifacts

2.1. Database Schema Definition (schema.sql)

The definitive schema for the PostgreSQL database has been created and finalized. This schema directly implements the design from the technical document, with several key refinements made to enhance integrity and align with modern best practices.

- **Tables Created:** Users, Attendances, and AttendanceRecords.
- **Key Design Decisions:**
  - **Data Integrity:** The AttendanceRecords table utilizes a composite primary key (attendance_id, student_number) to enforce the critical business rule that a student can have only one record per attendance session.
  - **Strict NOT NULL Constraints:** The table definitions have been synchronized with the application-level Pydantic models, ensuring that required data fields cannot be null at the database level.
  - **Optimized Data Types:** The flexible TEXT type is used for string data, and application-generated UUIDs are used for primary keys, decoupling the database from ID generation logic.
  - **Performance:** Essential indexes have been placed on foreign key columns to

ensure efficient query performance for joins.

2.2. Containerized Development Environment (docker-compose.yml)

A streamlined, isolated development environment has been configured using Docker Compose. This provides a consistent and reproducible setup for all development and testing activities.

- **Services Defined:**
  - postgres: A PostgreSQL 15 instance that is automatically initialized using the schema.sql script on its first run.
  - redis: A Redis 7 instance for handling real-time data and session management.
- **Key Features:**
  - **Data Persistence:** A named volume (postgres_data) is configured to ensure that database information persists across container restarts.
  - **Network Isolation:** All services communicate over a dedicated Docker network (attn_network), ensuring they are isolated from other processes on the host machine.

2.3. Data Model Definition (Pydantic Models)

The "single source of truth" for data structures within the application has been established using Pydantic models, located in app/backend/models/.

- **db_models.py:** Contains the User, Attendance, and AttendanceRecord models. These models strictly enforce the data types and required fields for all data that will be persisted in the PostgreSQL database.
- **redis_models.py:** Contains the specialized UserRedis model, designed specifically for managing user session data in Redis.
- **Benefit:** By leveraging Pydantic, we gain automatic, application-level data validation. This prevents malformed data from being processed or saved, significantly reducing the potential for bugs and ensuring a high degree of reliability.

3. Current Status

Phase 1 is **100% complete**. The project now has a fully defined database schema, a running and initialized database instance, a running Redis instance, and a complete set of validated data models.

4. Next Steps

With the foundation firmly in place, we are now ready to proceed to **Phase 2: Data**

Access Layer Implementation. This phase will follow a strict Test-Driven Development (TDD) methodology to build out the AsyncPostgresClient and RedisClient classes, ensuring that every function is fully tested and reliable before moving to the business logic layer.