Approach to the Problem

I started by setting up the base infrastructure of the application. To speed up the initial development, I used AI assistance to generate the base Express.js project structure, allowing me to focus more on system design and implementation rather than boilerplate setup.

After establishing the core structure, I designed the database schema with scalability and efficiency in mind. I followed a mild normalization approach, relying on primary and foreign key relationships to reference data across tables rather than duplicating values. This helps reduce redundancy while keeping queries flexible and maintainable.

For database interaction, I chose Sequelize ORM. I defined the expected models based on an enhanced interpretation of the requirements and used Sequelize's migration system to manage schema changes in a structured and version-controlled manner.

Reasoning

The primary goal of my approach was to balance development speed with long-term maintainability. Using a pre-generated Express structure reduced setup time without affecting code quality, allowing more focus on architectural decisions.

The decision to mildly normalize the database schema was made to ensure data integrity and scalability. By using relational references instead of duplicated fields, the system can grow without introducing inconsistencies or complex update logic.

Sequelize was selected as it provides a clear abstraction over SQL while still allowing fine control over queries and migrations. Its migration system also makes schema evolution safer and more predictable, especially in collaborative or production environments.

Overall, the chosen approach prioritizes clarity, scalability, and ease of future extension while keeping the solution aligned with real-world backend development practices.

Trade-offs

- **ORM vs Raw SQL:**
  Using Sequelize improves development speed and readability but introduces some overhead compared to raw SQL. For highly complex or performance-critical queries, raw SQL could offer more control.

- **Mild Normalization:**
  While normalization improves data integrity, it can require additional joins, which may impact performance at scale. This trade-off was accepted in favor of cleaner data relationships, with the option to optimize later if needed.
- **AI-assisted Project Setup:**
  AI was used only for boilerplate generation. While this accelerates development, it requires careful review to ensure generated code follows best practices and project standards.
- **Early Architectural Decisions:**
  Some design choices were made early based on assumptions about system usage. These choices may need revisiting as requirements evolve, but they provide a solid foundation for an initial implementation.