BASF Backend Engineering Case Study

Book Recommendation System

Challenge Overview

Build a simplified book recommendation system backend API that supports user login, listing books, and managing ratings & reviews. The API will be consumed by a frontend (you may assume one exists; no frontend implementation is required).

The goal is to evaluate your skills in:

- Python and FastAPI fundamentals (asynchronous programming, dependency injection, Pydantic validation).
- Designing and implementing clean, maintainable, and scalable backend architectures using layered or service-oriented principles.
- REST API design and database integration.

Technical Specification

- **Backend Framework:** FastAPI (Python)
- **Data Layer:** in-memory DB (e.g., SQLite) or SQLAlchemy (optionally Postgres container, optionally Alembic migrations)
- **Data Validation:** Pydantic models
- **Architecture:** Layered (API layer, service/business layer, repository/data access layer)
- **Optional:** Asynchronous endpoints, Celery for background tasks

Challenge Components

1. User Authentication

- Implement a **login endpoint** that accepts username & password.
- Use in-memory user data (no registration needed).
- Return a JWT token on successful login.
- Protect all book-related endpoints so they require authentication.

2. Book List Endpoint

- Create an endpoint `/books` returning a list of books with:
  - `title`
  - `author`
  - `genre`
  - `average_rating`
- Data can be seeded from a static JSON file or an external API like **Google Books**.
- Support **search** by title or author via query parameters.
- Optional: Implement pagination (`limit` & `offset` parameters).

Internal

3. Review and Rating System

- Endpoint to **add or update a rating & review** for a book.
- Ratings range from **1 to 5**; reviews are text.
- Endpoint to **retrieve all reviews** for a given book.
- Average rating should be calculated dynamically.

Architecture Requirements

- Follow a **layered architecture**:
    - **API Layer:** FastAPI routes and request/response models.
    - **Service Layer:** Business logic (e.g., average rating calculation).
    - **Repository Layer:** Database queries via SQLAlchemy.
- Use **dependency injections** to make services and repositories easy to replace/mocks in tests.

Testability Requirements

- Write at least **2–3 unit tests** for the service layer.
- Use FastAPI's dependency override mechanism to mock the repository layer in tests.
- Mock any external APIs (e.g., Google Books) so tests run offline.

Technical Submission Requirements

To ensure the project runs on any machine without special setup:

1. **Reproducible Environment**
    - Provide either:
        - A `docker-compose.yml` file that starts all required services (e.g., FastAPI app and PostgreSQL container).
        - **OR** an SQLite-based setup that runs locally without additional services.
2. **Dependency Management**
    - Include a `requirements.txt` (or `pyproject.toml` + lock file) with **all** dependencies.
    - Pin critical dependencies to specific versions.
    - State the Python version used in the README.
3. **Configuration**
    - Use environment variables for configuration (no secrets in code).
    - Include a `.env.example` file with all required variables.
4. **Project Structure**
    - Organize code into clear modules for API, service, and repository layers.
    - Avoid circular dependencies and global state.
5. **Documentation**
    - Include a `README.md` with:
        - i. Project overview
        - ii. Setup and run instructions (both Docker and non-Docker, if applicable)
        - iii. Architecture explanation
        - iv. How to run tests
6. **Cross-Platform Compatibility**
    - Solution must run on macOS, Windows, and Linux or run inside Docker.

Internal

Optional Bonus Features

- Asynchronous endpoints (`async def`).
- PostgreSQL database instead of SQLite.
- Background task (e.g., Celery) to periodically refresh book data from Google Books API.
- API versioning (e.g., `/api/v1/...`).

Submission Guidelines

Submit a GitHub or GitLab repository containing:

- All source code
- Database migration files (if applicable)
- Tests
- README
- `.env.example`
- Docker Compose setup if not using SQLite

Internal