# Technical Assessment: Intelligent Library System ("LuminaLib")

**Role:** Senior Backend & GenAI Engineer

**Stack:** Python (FastAPI), PostgreSQL, Generative AI (Llama 3), Docker

## 1. Project Context

You are architecting **"LuminaLib,"** a next-generation library system. This is not a simple metadata CRUD application. It is a content-aware system that manages actual book files, uses a local LLM to synthesize reader sentiment, and employs machine learning to drive discovery.

**Evaluation Goal:** We are looking for production-grade engineering. We value **Clean Architecture**, **SOLID principles**, **Configurability**, and **Async mastery** over quick-and-dirty scripts.

## 2. Functional Requirements

### A. Authentication & User Management

- **Security:** Implement a robust authentication system. The API must not be open; it requires **JWT-based stateless authentication**.
- **User Lifecycle:** Provide endpoints for **Signup**, **Login**, **Profile Management**, and **Signout**.
- **Design Challenge:** You must design the users table and associated security flows (password hashing, token expiration) from scratch.

### B. Book Ingestion & Management (The "Content" Layer)

- **True Ingestion:** Creating a book is not just inserting a title and author into a DB. You must implement a mechanism to **upload and store the actual book content** (text or PDF).
    - *Storage:* You may simulate S3 using a local container (like MinIO) or manage local file storage, but the architecture must treat storage as an abstraction.
- **Full CRUD:** Implement Create, Read, Update, and Delete operations for book metadata and content.
- **Library Mechanics:** Implement **Borrow** and **Return** functionality.
    - *Constraint:* Users cannot review a book they haven't borrowed.

### C. The Intelligence Layer (GenAI & ML)

- **Book Summarization:** Upon ingestion, the system must asynchronously read the book content and generate a summary using **LLM** of your choice.
- **Review Analysis:** When a user submits a review (POST /books/{id}/reviews), it must trigger a background task. This task uses the LLM to update a "rolling consensus" or summary of what users feel about the book.
- **Recommendation Engine:**
    - **User Preferences:** The requirement is vague on purpose. You must design a DB schema to store "User Preferences." How you model this (explicit tags, implicit borrow history, etc.) is part of the test.
    - **ML Model:** Implement a recommendation algorithm (e.g., Content-Based or Collaborative Filtering) that uses these preferences to suggest books.

## 3. Architecture & Code Quality

- **Clean Architecture:** Use **Dependency Injection (DI)** and **Interface-Driven Development**.
    - *Extensibility:* We will evaluate how easily we can swap components. For example, can we switch the file storage from Local Disk to AWS S3 by changing a single config line? Can we swap Llama 3 for OpenAI without rewriting business logic?
- **Code Standards:**
    - **Linting & Formatting**
    - **Proper import sequencing**
    - **Type Safety**

## 4. Deployment & Infrastructure

## Shutterstock Explore

- **Dockerization:** The application must be fully containerized.
- **One-Command Start:** We expect a docker-compose.yml file. We must be able to run a single command (e.g., docker-compose up --build) to spin up:
    1. The API Service
    2. The PostgreSQL Database
    3. The Local LLM Service (e.g., via Ollama or a mock container if hardware is limited)

- **Configuration:** All environment variables (DB URLs, API Keys, Model paths) must be managed via .env files and injected into the containers.

## 5. API Specification (Expanded)

Your RESTful API must support at least the following:

| Domain | Method | Endpoint | Description |
|---|---|---|---|
| **Auth** | POST | /auth/signup | Register a new user. |
| | POST | /auth/login | Return JWT access token. |
| **Books** | POST | /books | Upload book file & metadata. Triggers async summary. |
| | GET | /books | List books (pagination required). |
| | PUT | /books/{id} | Update book details. |
| | DELETE | /books/{id} | Remove book and associated file. |
| | POST | /books/{id}/borrow | User borrows a book. |
| | POST | /books/{id}/return | User returns a book. |
| **Reviews** | POST | /books/{id}/reviews | Submit review. Triggers async sentiment analysis. |
| **Intel** | GET | /books/{id}/analysis | Get GenAI-aggregated summary of all reviews. |
| | GET | /recommendations | Get ML-based suggestions for the current user. |

## 6. Deliverables

1. **Source Code:** A clean, structured repository.
2. **docker-compose.yml:** The orchestration file.
3. **ARCHITECTURE.md:** A document explaining:
    - Why you chose your specific DB Schema for User Preferences.
    - How you handled the "Async" aspect of LLM generation (Queues? Background Tasks?).
    - Your strategy for the ML recommendation model.
4. **README.md:** Instructions to run the application using the one-command start.

## Evaluation Rubric

1. **Modularity:** Can we swap the Storage/LLM providers easily?
2. **Docker Proficiency:** Does the compose file work seamlessly?
3. **Code Hygiene:** Are imports sorted? Is the code linted?

4. **GenAI Implementation:** Is the prompt engineering handled in a structured, reusable way?