DATA-236 - Distributed Systems for Data Engineering
Homework 5

Overview

Create a RESTful API for a Library Management System using FastAPI and MySQL with full CRUD operations for books and authors.

Requirements:

Database Design (3 points)

Design two related tables.

The Authors table should include id (primary key), first_name, last_name, email (unique), and timestamps.

The Books table should contain id (primary key), title, isbn (unique), publication_year, available_copies (default 1), author_id (foreign key), and timestamps. Use appropriate data types and constraints.

API Implementation (5 points)

Implement CRUD endpoints for both entities.

For authors, create endpoints to add new authors, retrieve all authors with pagination, get individual authors, update author information, and delete authors.

For books, provide similar functionality including creating books, listing all books, retrieving specific books, updating book details, and removing books.

Include one additional endpoint to get all books written by a specific author.

Data Validation and Error Handling (2 points)

Use Pydantic schemas for request and response validation.

Implement proper HTTP status codes and error messages for scenarios like resource not found, validation errors, and constraint violations.

Ensure email format validation and prevent deletion of authors who have associated books.

Submit screenshots of each action tested using API endpoints. Submit a screenshot of your DB

Frontend Integration with Redux (5 points)

In this part, you will build a React + Redux frontend that integrates with the FastAPI + MySQL backend you created in Part I. The frontend should use Axios for API calls and Redux Toolkit to manage application state.

I. Redux Setup (1 point)

- Create a Redux store using Redux Toolkit.
- Define a booksSlice to manage the state of books.
- Include reducers/actions for fetching, adding, updating, and deleting books.

II. Async Thunks with Axios (1 point)

- Implement async thunks that call your FastAPI endpoints from Part I:
    - fetchBooks → GET /books
    - createBook → POST /books
    - updateBook → PUT /books/{id}
    - deleteBook → DELETE /books/{id}
- Handle success and error cases properly.

III. Home Screen (1 point)

- Display all books by reading from Redux state.
- The list should update automatically when books are created, updated, or deleted.

IV. Create / Update Screens (1 point)

- Build a form to create a new book.
- Build a form to update an existing book (select a book by ID).
- On submission, dispatch the correct Redux thunk and show the updated results on the Home Screen.

V. Delete Functionality (1 point)

- Add a delete button next to each book in the list.
- On click, dispatch the delete thunk and update the Redux state to remove the book.

Submission Guidelines for Redux Part

- Submit one screenshot per feature (Home, Create, Update, Delete) showing the code snippet (Redux thunk or slice) and the corresponding UI output together.
- Make sure the screenshot includes both code and browser output in the same frame (side-by-side or one above the other).
- Do not submit full files. Only the relevant code + working result.

LLM Chat (Ollama + FastAPI + React)

Note: you can make this part of your homework along with the above 2 parts, in the same application, but if it's easier for you, make it separately.

1. **Backend — Models (SQLAlchemy / MySQL)**
    - Conversation: id, user_id, title, timestamps, relation to messages.
    - Message: id, conversation_id (Foreign Key), Enum role (user/assistant/system), content, created_at.
2. **Backend — Schemas (Pydantic)**
    - ChatIn: user_id, message, optional conversation_id & title.
    - ChatOut: conversation_id, reply message.
    - ConversationOut, MessagesOut, ChatMessageOut.
3. **Backend — Router (/ai endpoints)**
    - POST /ai/chat: create/continue conversation, save user msg, call Ollama, save assistant reply, return reply.
    - GET /ai/conversations: list user's chats.
    - GET /ai/messages/{conversation_id}: get all messages in a conversation.
    - call_ollama(): async httpx request to Ollama API.
4. **Backend Setup**
    - Add router in app/main.py.

- Run ollama serve and pull model (llama3.1).
- Alembic migration or Base.metadata.create_all(engine).

5. **Frontend — Redux Slice**

    - fetchConversations, fetchMessages, sendMessage thunks using Axios.
    - Store: conversations, messages, currentConversationId, error handling.

6. **Frontend — Chat UI (React)**

    - Main: show messages, input box, send message (dispatch thunk).
    - Basic classes for styling.

7. **Testing**

    - cURL examples: start conversation, continue, list conversations, fetch messages.

8. **Submission Screenshots**

    - API responses in Swagger/Postman.
    - MySQL tables with rows screenshots
    - code + working chat UI screenshots