**Book & Task Manager API**

A modern, secure, and full-stack web application built with FastAPI, PostgreSQL, and a lightweight Vanilla JS + HTML/CSS frontend. The system enables user authentication, book management, and task tracking with user-specific data isolation.

**Features**

**User Authentication (JWT-based)**

* Register and Login using FastAPI's OAuth2PasswordBearer with password hashing via passlib.
* Secure token generation and validation using JWT (python-jose).
* Isolated data access: each user's books and tasks are accessible only after login.

**Book Management**

* Create, view, and delete books linked to the current user.
* Filter books by name, author, or publisher.
* Sort results by ID or any field in ascending or descending order.

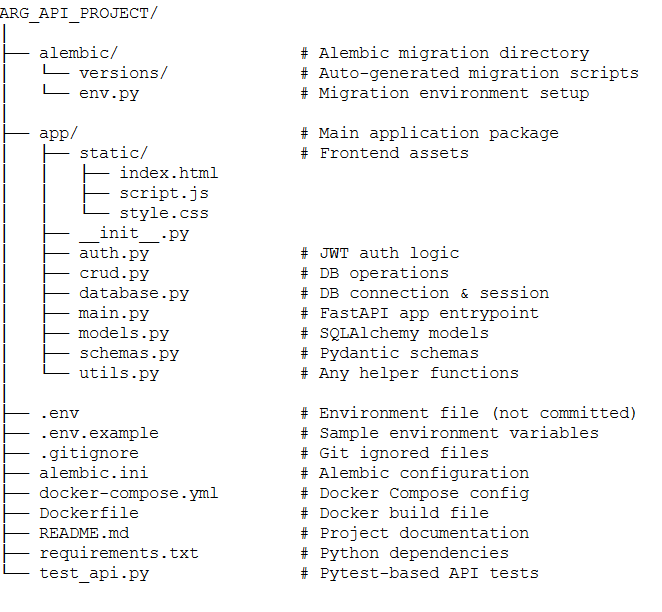
**Task Management**

* Add, update, complete, and delete tasks.
* Filter by completion status or title keyword.
* Pagination (skip & limit) and sorting support.

**Frontend (Static + Vanilla JS)**

* Clean responsive UI powered by HTML, CSS, and JavaScript.
* Login/Register, Books, and Tasks management fully integrated with backend.
* Secure logout and local storage-based token handling.

**Project Structure**



**Tech Stack**

* Backend: FastAPI, SQLAlchemy, Pydantic v2
* Database: PostgreSQL
* Authentication: JWT (HS256)
* Frontend: HTML, CSS, JavaScript (no frameworks)
* Testing: Pytest

**Running Locally**

**Prerequisites**

* Python 3.10+
* PostgreSQL (running locally on port 5432 with DB arg\_books)
* Node.js (optional for extended tooling)

**Installation**

* git clone https://github.com/your-username/arg-api-project.git
* cd arg-api-project
* python -m venv venv
* source venv/bin/activate # On Windows: venv\Scripts\activate
* pip install -r requirements.txt

**Environment**

Ensure PostgreSQL is running and accessible using the credentials defined in database.py:

DATABASE\_URL=postgresql://postgres:postgres@localhost:5432/arg\_books

**Run the app**

* uvicorn app.main:app –reload

Visit [*http://127.0.0.1:8000/*](http://127.0.0.1:8000/) to use the frontend.

**API Endpoints Summary**

| **Method** | **Endpoint** | **Description** | **Auth Required** | **Request Body / Parameters** | **Response Format** |
| --- | --- | --- | --- | --- | --- |
| GET | /health | Health check to verify if server is running | No | None | JSON message |
| POST | /register | Register a new user | No | JSON: { "username": str, "password": str } | JSON with user ID and username |
| POST | /login | User login to receive JWT token | No | Form: username, password | JSON with access token |
| GET | /books/ | Retrieve all books with optional filters | Yes | Query params: name, author, publisher, sort\_by, sort\_order | JSON list of books |
| POST | /books/ | Create a new book | Yes | JSON: book\_name, description, pages, author, publisher | JSON with created book |
| DELETE | /books/{book\_id} | Delete a specific book by ID | Yes | Path param: book\_id | JSON message |
| GET | /tasks/ | Retrieve user tasks with filters & pagination | Yes | Query params: skip, limit, completed, title, sort\_by, sort\_order | JSON list of tasks |
| POST | /tasks/ | Create a new task | Yes | JSON: title, description, completed (optional) | JSON with created task |
| POST | /tasks/{task\_id}/complete | Mark a task as completed | Yes | Path param: task\_id | JSON message |
| PUT | /tasks/{task\_id} | Update a specific task | Yes | JSON: Fields to update (title, description, completed) | JSON with updated task |
| DELETE | /tasks/{task\_id} | Delete a task by ID | Yes | Path param: task\_id | JSON message |

**Testing**

**About**

Testing is handled via Pytest, and tests are defined in test\_api.py. These tests ensure API correctness and include edge cases.

What It Covers

***User Auth:***

* Successful registration and login
* Handling duplicate usernames
* Incorrect credentials handling

***Book Management:***

* Unauthorized creation of books
* Book creation, retrieval, and deletion for authenticated users
* Filtering books by author

***Task Management:*** (To be extended similarly)

* Create tasks for logged-in users
* Update and mark tasks as complete
* Filter tasks by status and title
* Enforce task ownership (users can access only their tasks)
* Delete tasks and validate persistence

**Run Tests**

* pytest test\_api.py

This will execute all test functions and give pass/fail results for each route scenario.

**Docker Support (Config Included – Final Testing in Progress)**

Docker setup is included to enable easy deployment and environment replication.

*Included Files:*

* Dockerfile – Containerizes the FastAPI backend
* docker-compose.yml – Runs FastAPI + PostgreSQL together
* .env.example – Manages environment variables

*Status:*

Docker is implemented and currently under testing. Final deployment will be a one-command setup:

* docker-compose up –build

*Components:*

Web (FastAPI): Serves API + frontend, connects to DB

DB (PostgreSQL 15): Stores app data with persistent volume support

*Planned Access:*

* App: <http://localhost:8000>
* Docs: <http://localhost:8000/docs>

**Alembic: Database Migrations**

Alembic enables seamless version control for your database schema.

* **Setup**

🡪 pip install alembic

🡪 alembic init alembic

* **Configuration**

Edit alembic.ini and set your DB URL:

🡪 sqlalchemy.url = postgresql://postgres:postgres@localhost:5432/arg\_books

Modify alembic/env.py to import Base and DATABASE\_URL from the app.

* **Generate and Apply Migration**

**🡪** alembic revision --autogenerate -m "Initial migration"

**🡪** alembic upgrade head

**License**

MIT License

**Author**

Pranav Asalekar