REST API and Database Project

1. Project Overview

Build a **Book Management REST API** using Python and FastAPI.

The project connects a REST API to a SQLite database and provides complete CRUD operations for books.

The API allows users to:

- Add books
- View all books
- View one book
- Update books
- Delete books
- Search books
- Filter books by genre

FastAPI supports SQL databases through different database libraries. SQLite is useful for a small project because it works as a single local database file.

2. Features

- REST API
- CRUD operations
- SQLite database
- Book search
- Genre filtering
- Request validation
- Error handling
- JSON responses

- Automatic Swagger documentation

3. Technologies

- Python 3
- FastAPI
- Pydantic
- SQLite
- Uvicorn
- sqlite3

SQLite parameterized queries are used for database values instead of constructing SQL statements from user input.

4. Folder Structure

```
rest_api_database/
□□□ main.py
□□□ database.py
□□□ schemas.py
□□□ requirements.txt
□□□ README.md
```

5. How the Project Works

```
Client
  □
HTTP Request
  □
FastAPI Endpoint
  □
Pydantic Validation
  □
Database Logic
  □
```

SQLite

JSON Response

For example:

POST /books

Validate book data

Insert into SQLite

Return created book

6. Complete Backend Code

database.py

```
import sqlite3


class BookDatabase:

    def __init__(self, database_name="books.db"):
        self.database_name = database_name
        self.create_table()

    def connect(self):
        return sqlite3.connect(self.database_name)

    def create_table(self):
        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
```

```
        CREATE TABLE IF NOT EXISTS books (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            title TEXT NOT NULL,
            author TEXT NOT NULL,
            genre TEXT NOT NULL,
            year INTEGER NOT NULL
        )
    """)

    connection.commit()
    connection.close()

def create_book(self, title, author, genre, year):
    connection = self.connect()
    cursor = connection.cursor()

    cursor.execute("""
        INSERT INTO books
        (title, author, genre, year)
        VALUES (?, ?, ?, ?)
    """, (title, author, genre, year))

    connection.commit()
    book_id = cursor.lastrowid
    connection.close()

    return book_id

def get_books(self):
    connection = self.connect()
    connection.row_factory = sqlite3.Row
    cursor = connection.cursor()

    cursor.execute("""
```

```
            SELECT id, title, author, genre, year
            FROM books
            ORDER BY id DESC
        """)

        books = [dict(row) for row in cursor.fetchall()]
        connection.close()

        return books

    def get_book(self, book_id):
        connection = self.connect()
        connection.row_factory = sqlite3.Row
        cursor = connection.cursor()

        cursor.execute("""
            SELECT id, title, author, genre, year
            FROM books
            WHERE id = ?
        """, (book_id,))

        book = cursor.fetchone()
        connection.close()

        return dict(book) if book else None

    def update_book(
        self,
        book_id,
        title,
        author,
        genre,
        year
    ):
```

```
        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            UPDATE books
            SET title = ?,
                author = ?,
                genre = ?,
                year = ?
            WHERE id = ?
        """, (
            title,
            author,
            genre,
            year,
            book_id
        ))

        connection.commit()
        updated = cursor.rowcount > 0
        connection.close()

        return updated

    def delete_book(self, book_id):
        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            DELETE FROM books
            WHERE id = ?
        """, (book_id,))

        connection.commit()
```

```
        deleted = cursor.rowcount > 0
        connection.close()

        return deleted

    def search_books(self, keyword):
        connection = self.connect()
        connection.row_factory = sqlite3.Row
        cursor = connection.cursor()

        pattern = f"%{keyword}%"

        cursor.execute("""
            SELECT id, title, author, genre, year
            FROM books
            WHERE title LIKE ?
               OR author LIKE ?
               OR genre LIKE ?
            ORDER BY id DESC
        """, (pattern, pattern, pattern))

        books = [dict(row) for row in cursor.fetchall()]
        connection.close()

        return books

    def get_by_genre(self, genre):
        connection = self.connect()
        connection.row_factory = sqlite3.Row
        cursor = connection.cursor()

        cursor.execute("""
            SELECT id, title, author, genre, year
            FROM books
```

```
            WHERE genre = ?
            ORDER BY id DESC
        """, (genre,))

        books = [dict(row) for row in cursor.fetchall()]
        connection.close()

        return books
```

schemas.py

```
from pydantic import BaseModel, Field


class BookCreate(BaseModel):
    title: str = Field(
        min_length=1,
        max_length=200
    )

    author: str = Field(
        min_length=1,
        max_length=100
    )

    genre: str = Field(
        min_length=1,
        max_length=100
    )

    year: int = Field(
        ge=1000,
        le=2100
    )
```

```
class Book(BookCreate):
    id: int
```

main.py

```
from fastapi import FastAPI, HTTPException

from database import BookDatabase
from schemas import Book, BookCreate


app = FastAPI(
    title="Book Management API",
    description="A REST API connected to a SQLite database.",
    version="1.0.0"
)

database = BookDatabase()


@app.get("/")
def home():
    return {
        "message": "Book Management API is running"
    }


@app.post(
    "/books",
    response_model=Book,
    status_code=201
)
def create_book(book: BookCreate):
    book_id = database.create_book(
```

```
        book.title,
        book.author,
        book.genre,
        book.year
    )

    return {
        "id": book_id,
        **book.model_dump()
    }


@app.get(
    "/books",
    response_model=list[Book]
)
def get_books():
    return database.get_books()


@app.get(
    "/books/{book_id}",
    response_model=Book
)
def get_book(book_id: int):
    book = database.get_book(book_id)

    if book is None:
        raise HTTPException(
            status_code=404,
            detail="Book not found."
        )

    return book
```

```
@app.put(
    "/books/{book_id}",
    response_model=Book
)
def update_book(
    book_id: int,
    book: BookCreate
):
    updated = database.update_book(
        book_id,
        book.title,
        book.author,
        book.genre,
        book.year
    )

    if not updated:
        raise HTTPException(
            status_code=404,
            detail="Book not found."
        )

    return {
        "id": book_id,
        **book.model_dump()
    }


@app.delete("/books/{book_id}")
def delete_book(book_id: int):
    deleted = database.delete_book(book_id)
```

```
    if not deleted:
        raise HTTPException(
            status_code=404,
            detail="Book not found."
        )

    return {
        "message": "Book deleted successfully."
    }


@app.get(
    "/books/search/{keyword}",
    response_model=list[Book]
)
def search_books(keyword: str):
    return database.search_books(keyword)


@app.get(
    "/books/genre/{genre}",
    response_model=list[Book]
)
def books_by_genre(genre: str):
    return database.get_by_genre(genre)
```

7. API Endpoints

```
POST    /books
GET     /books
GET     /books/{book_id}
PUT     /books/{book_id}
DELETE  /books/{book_id}

GET     /books/search/{keyword}
```

`GET /books/genre/{genre}`

8. Example Request

Create Book

`POST /books`

```
{
  "title": "Clean Code",
  "author": "Robert C. Martin",
  "genre": "Programming",
  "year": 2008
}
```

Response:

```
{
  "id": 1,
  "title": "Clean Code",
  "author": "Robert C. Martin",
  "genre": "Programming",
  "year": 2008
}
```

Search

`GET /books/search/python`

Genre Filter

`GET /books/genre/Programming`

9. Requirements

requirements.txt

```
fastapi
uvicorn
```

10. How to Run

Create a virtual environment:

```
python -m venv venv
```

Activate it on Windows:

```
venv\Scripts\activate
```

Install dependencies:

```
pip install -r requirements.txt
```

Start the server:

```
uvicorn main:app --reload
```

Open the API:

```
http://127.0.0.1:8000
```

Open the interactive API documentation:

```
http://127.0.0.1:8000/docs
```

FastAPI's generated documentation can be used to test the endpoints directly.

11. Basic Testing

Test the following:

```
1. Create a book
2. Get all books
3. Get a book by ID
4. Search for a book
5. Filter books by genre
6. Update a book
7. Delete a book
8. Try an invalid book ID
9. Try invalid input
```

Example invalid request:

```
{
   "title": "",
   "author": "Test",
   "genre": "Programming",
   "year": 2026
}
```

The API rejects the request because `title` must contain at least one character.

12. Database

The application automatically creates:

```
books.db
```

with the following table:

```
books
□□□ id
```

□□□ title
□□□ author
□□□ genre
□□□ year

13. Small Improvements

Possible next improvements:

- Authentication
- User accounts
- Book reviews
- Ratings
- Pagination
- Sorting
- Borrowing system
- PostgreSQL
- SQLModel or SQLAlchemy
- Automated tests
- Docker
- API deployment
- React frontend

Visit haas.dev for more resources and guides.