Great choice! 🚀 Let’s build a **"Bookstore REST API"** project in Python step by step.

We’ll use **FastAPI + SQLite (SQLAlchemy ORM)** because:

* FastAPI auto-generates **Swagger docs** (/docs)
* SQLAlchemy manages the DB schema cleanly
* Postman can be used for testing

**Project: Bookstore REST API**

**Objective**

Create a REST API to manage inventory of books with CRUD operations, search/filter, and validations.

**Tools**

* **FastAPI** → REST API framework
* **SQLite** → database
* **SQLAlchemy** → ORM
* **Postman** → testing
* **Uvicorn** → server

**Step-by-Step Mini Guide**

**1. Define Database Schema (Books)**

Schema → id, title, author, price, quantity, published\_date.

# models.py

from sqlalchemy import Column, Integer, String, Float, Date

from database import Base

class Book(Base):

\_\_tablename\_\_ = "books"

id = Column(Integer, primary\_key=True, index=True)

title = Column(String, index=True, nullable=False)

author = Column(String, index=True, nullable=False)

price = Column(Float, nullable=False)

quantity = Column(Integer, default=0)

published\_date = Column(Date, nullable=True)

**2. Database Setup**

# database.py

from sqlalchemy import create\_engine

from sqlalchemy.ext.declarative import declarative\_base

from sqlalchemy.orm import sessionmaker

SQLALCHEMY\_DATABASE\_URL = "sqlite:///./books.db"

engine = create\_engine(SQLALCHEMY\_DATABASE\_URL, connect\_args={"check\_same\_thread": False})

SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

Base = declarative\_base()

**3. Create API Routes (CRUD)**

# main.py

from fastapi import FastAPI, Depends, HTTPException

from sqlalchemy.orm import Session

from database import SessionLocal, engine, Base

import models

from pydantic import BaseModel

from typing import List, Optional

from datetime import date

# Create tables

Base.metadata.create\_all(bind=engine)

app = FastAPI(title="📚 Bookstore REST API")

# Dependency

def get\_db():

db = SessionLocal()

try:

yield db

finally:

db.close()

# Pydantic schemas

class BookCreate(BaseModel):

title: str

author: str

price: float

quantity: int

published\_date: Optional[date] = None

class BookOut(BookCreate):

id: int

class Config:

orm\_mode = True

# --- CRUD Routes ---

@app.post("/books/", response\_model=BookOut)

def create\_book(book: BookCreate, db: Session = Depends(get\_db)):

db\_book = models.Book(\*\*book.dict())

db.add(db\_book)

db.commit()

db.refresh(db\_book)

return db\_book

@app.get("/books/", response\_model=List[BookOut])

def read\_books(skip: int = 0, limit: int = 10, db: Session = Depends(get\_db)):

return db.query(models.Book).offset(skip).limit(limit).all()

@app.get("/books/{book\_id}", response\_model=BookOut)

def read\_book(book\_id: int, db: Session = Depends(get\_db)):

book = db.query(models.Book).filter(models.Book.id == book\_id).first()

if not book:

raise HTTPException(status\_code=404, detail="Book not found")

return book

@app.put("/books/{book\_id}", response\_model=BookOut)

def update\_book(book\_id: int, book: BookCreate, db: Session = Depends(get\_db)):

db\_book = db.query(models.Book).filter(models.Book.id == book\_id).first()

if not db\_book:

raise HTTPException(status\_code=404, detail="Book not found")

for key, value in book.dict().items():

setattr(db\_book, key, value)

db.commit()

db.refresh(db\_book)

return db\_book

@app.delete("/books/{book\_id}")

def delete\_book(book\_id: int, db: Session = Depends(get\_db)):

db\_book = db.query(models.Book).filter(models.Book.id == book\_id).first()

if not db\_book:

raise HTTPException(status\_code=404, detail="Book not found")

db.delete(db\_book)

db.commit()

return {"message": "Book deleted"}

**4. Add Search/Filtering Options**

@app.get("/books/search/", response\_model=List[BookOut])

def search\_books(

title: Optional[str] = None,

author: Optional[str] = None,

min\_price: Optional[float] = None,

max\_price: Optional[float] = None,

db: Session = Depends(get\_db),

):

query = db.query(models.Book)

if title:

query = query.filter(models.Book.title.ilike(f"%{title}%"))

if author:

query = query.filter(models.Book.author.ilike(f"%{author}%"))

if min\_price:

query = query.filter(models.Book.price >= min\_price)

if max\_price:

query = query.filter(models.Book.price <= max\_price)

return query.all()

**5. Error Handling & Validation**

Already handled using HTTPException and **Pydantic validation**.

**6. API Docs with Swagger/Postman**

* Swagger: auto at **http://127.0.0.1:8000/docs**
* ReDoc: **/redoc**
* Export **Postman collection** from /docs → “Export OpenAPI JSON”

**Deliverables**

1. **API script** → main.py, models.py, database.py
2. **SQLite DB file** → books.db (auto-created)
3. **Postman Collection** → export from Swagger/OpenAPI

**Run Project**

uvicorn main:app --reload

Open in browser:  
 Swagger UI → http://127.0.0.1:8000/docs