🔬 Lab Assignment: Online Bookstore Web API

🎯 Objective:

Build a RESTful Web API for managing an **online bookstore**, applying real-world development principles such as layered architecture, best practices, and complete CRUD operations.

📘 Domain Context:

An online bookstore wants to manage its inventory, customers, and orders using a Web API. The backend system should allow authorized users to perform CRUD operations on books, customers, and orders.

🔧 Technical Requirements:

✅ Technologies:

- ASP.NET Core Web API (.NET 6+)
- Entity Framework Core (Code-First)
- SQL Server (or SQLite for local testing)
- Swagger/OpenAPI for documentation
- Postman for testing (optional)

🗂️ Requirements

1. Entities to Create

A. Book

- `BookId` (int, PK)
- `Title` (string)
- `Author` (string)
- `Category` (string)
- `Price` (decimal)
- `InStock` (int)

B. Customer

- `CustomerId` (int, PK)
- `Name` (string)
- `Email` (string)
- `PhoneNumber` (string)

C. Order

- `OrderId` (int, PK)
- `CustomerId` (FK)
- `OrderDate` (DateTime)
- `TotalAmount` (decimal)

D. OrderItem

- `OrderItemId` (int, PK)
- `OrderId` (FK)
- `BookId` (FK)
- `Quantity` (int)
- `UnitPrice` (decimal)

💡 Use **navigational properties** to define relationships properly using EF Core.

🛠️ Features to Implement

CRUD Endpoints

A. BooksController

- `GET /api/books` — Get all books
- `GET /api/books/{id}` — Get book by ID
- `POST /api/books` — Add a new book
- `PUT /api/books/{id}` — Update book details
- `DELETE /api/books/{id}` — Delete a book

B. CustomersController

- Similar CRUD endpoints for managing customers

C. OrdersController

- `POST /api/orders` — Create a new order with order items
- `GET /api/orders` — Get all orders with related data
- `GET /api/orders/{id}` — Get order details by ID
- `DELETE /api/orders/{id}` — Cancel/delete an order

📋 Best Practices to Follow

🧱 Project Structure

- Use **Layered Architecture**:
  - `Controllers` (API layer)
  - `Services` (Business logic layer)
  - `Repositories` (Data access layer)
  - `DTOs` (Data Transfer Objects)
  - `Models` (EF Entities)

🔐 Validation & Error Handling

- Use `ModelState.IsValid` for input validation
- Return appropriate HTTP status codes:
  - `200 OK`, `201 Created`, `204 No Content`
  - `400 Bad Request`, `404 Not Found`, `500 Internal Server Error`
- Use `try-catch` blocks for exception handling

📄 Swagger/OpenAPI

- Configure Swagger to display your endpoints
- Document response types and possible error codes using `[ProducesResponseType]`

🌐 CORS

- Enable CORS to allow frontend apps to interact with API

🧪 Sample Use Cases

1. **Create a book** → Add a new book with title, author, category, price, and stock.
2. **Place an order** → Select a customer and book(s), and create an order.
3. **List all orders** → View a customer's order history with order total and date.
4. **Update stock** → Update stock count after each successful order.
5. **Delete customer** → If a customer has no orders, allow deletion.

🚀 Bonus Challenges

- Add search and filter options to `GET /api/books?author=xyz&category=xyz`
- Implement pagination for large book listings
- Add JWT-based Authentication (for advanced students)
- Use AutoMapper for DTO conversion

- Add unit tests for services and controllers

👨‍💻 Deliverables

- Complete ASP.NET Core Web API project in GitHub or ZIP
- SQL database (or migrations)
- Postman Collection (optional)
- README.md with setup instructions and API endpoint documentation