E-commerce System API with Minimal APIs

Objective

Develop a RESTful API using ASP.NET Core 7's Minimal APIs that interfaces with a database through Entity Framework Core. The API will manage an e-commerce system with **Product** and **Category** entities. CRUD operations should be handled by a separate service layer to demonstrate good practices in code organization and separation of concerns.

Tools Required

- Visual Studio 2022 or later, or Visual Studio Code with the C# extension.
- .NET 7 SDK.
- A SQL Server instance or LocalDB for development purposes.

Database Model

1. **Product Entity**:
    - **ProductId** (Primary Key, auto-increment)
    - **Name** (string)
    - **Description** (string, optional)
    - **Price** (decimal)
    - **CategoryId** (Foreign Key)
2. **Category Entity**:
    - **CategoryId** (Primary Key, auto-increment)
    - **Name** (string)

Tasks

1. **Project Setup**:
    - Create a new ASP.NET Core Web API project with .NET 7, focusing on Minimal APIs.
2. **Implement Entities and DbContext**:
    - Define **Product** and **Category** as C# record types in a Models folder.
    - Create a DbContext class within a Data folder, setting it up for your database.
3. **Code First Migration**:
    - Apply Entity Framework Core tools to generate and apply a migration for the initial database schema.
4. **Service Layer Implementation**:
    - Create a folder named Services.
    - Inside, implement **IProductService** and **ICategoryService** interfaces with necessary CRUD operations.

- Implement concrete classes for these services, handling the logic for accessing and modifying database data.

5. **Configure Services and Middleware**:
   - In **Program.cs**, register your DbContext and service classes with dependency injection.

```
services.AddDbContext<ApplicationDbContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("DefaultConnecti
on")));
```

   - Do not forget to add your connection string in **appsettings.json.**

```
"ConnectionStrings": {
    "DefaultConnection":
"Server=(localdb)\\mssqllocaldb;Database=YourDatabaseName;Trusted_Conne
ction=True;MultipleActiveResultSets=true"
}
```

6. **Implement CRUD Endpoints Using Minimal APIs**:
   - In **Program.cs**, define routes that use your service layer to perform CRUD operations. Ensure you cover:
     - Creating, reading (by ID and all), updating, and deleting for **Product**
     - Creating, reading (by ID and all), updating, and deleting for **Category**
7. **Validation and Error Handling**:
   - Implement validation within your service classes **and/or** directly in the Minimal API endpoints.
   - Ensure proper error handling, such as returning the correct status codes for not found or bad requests.
8. **Testing the API**:
   - Use Postman, Swagger, or a similar tool to test each endpoint for functionality.
9. **Additional Tasks: LINQ Queries in an E-commerce System API (ONLY For BEST STUDENTS)**
   - Add functionality to filter products by price or category. Use LINQ to query the database for products within a specified price range or belonging to a specific category.
   - Implement a search feature for products. Use LINQ to allow searching products by name or description.
   - Create an endpoint that returns the average price of all products, and another that returns the total number of products in each category. Use LINQ aggregate functions like **Average** and **Count**.
   - Implement an endpoint that accepts price range parameters (minimum and maximum price) and category IDs for filtering products. Use LINQ to dynamically build the query based on provided parameters.