Project Structure:

Create a Solution:

Create a new .NET solution named "WebApiSolution."

Class Libraries:

Create the following class libraries within the solution:

Domain: Define two model classes (Product and Category).

Data: Implement a DbContext class (AppDbContext) and manage database migrations.

Services: Create a service class (ProductService) responsible for business logic related to products.

Web API Project:

Create a Web API project named "WebApiApp."

Set up appsettings.json with a connection string for the database.

Controller:

Implement a controller (ProductsController) in the Web API project.

Use dependency injection to inject the ProductService into the controller.

Create API endpoints for:

Retrieving all products.

Retrieving a single product by ID.

Adding a new product.

Updating an existing product.

Deleting a product.

Retrieving all products in a specific category.

Retrieving the total price of products in a specific category.

Retrieving the total price of products per category.

Database Integration:

Configure Entity Framework Core in the Web API project to use the AppDbContext.

Models:

Implement the Product and Category models in the Domain class library.

Product Domain Class:

Id:

Type: Integer

Description: Unique identifier for the product.

Name:

Type: String

Description: Name of the product.

Price:

Type: Decimal

Description: Price of the product.

CategoryId:

Type: Integer

Description: Identifier indicating the category to which the product belongs.

Additional Properties (Optional):

Description, CreatedAt, or any other properties that might be relevant to your application.

Category Domain Class:

Id:

Type: Integer

Description: Unique identifier for the category.

Name:

Type: String

Description: Name of the category.

Additional Properties (Optional):

Description, CreatedAt, or any other properties that might be relevant to your application.

DbContext:

Create the AppDbContext class in the Data class library, inheriting from DbContext.

Include DbSet properties for both Product and Category.

Services:

Implement the ProductService class in the Services class library.

Include methods for CRUD operations on products.

Controller Actions:

Implement actions in the ProductsController to interact with the ProductService.

Use appropriate HTTP methods (GET, POST, PUT, DELETE) for each action.

Dependency Injection:

Inject the ProductService into the ProductsController using constructor injection.

Testing:

Use a tool like Postman or Swagger to test the API endpoints.

Test each endpoint for various scenarios, especially those involving the enhanced service logic.

Note:

For the endpoints that accept parameters (e.g., category ID), ensure proper validation and error handling.

Implement error responses with meaningful messages and appropriate HTTP status codes.