# Dot Net - Milestone project assessment

## Practical Assessment: Building a Product Management System with Razor Pages and N-Tier Architecture

**Objective**:

In this practical assessment, you will build a Product Management System using **ASP.NET Core Razor Pages** with **N-Tier Architecture**. You will apply the **Repository Pattern** to separate concerns, implement **CRUD operations**, and create essential pages such as **Home** and **Details** pages. Additionally, you will ensure clean, maintainable, and scalable architecture.

---

**Scenario:**

You are tasked with building a **Product Management System** for a small e-commerce platform. This system will allow users to view, create, update, and delete products. The application will consist of the following key pages and functionalities:

- **Home Page**: A list of products available in the system.
- **Details Page**: Detailed view of a product with information such as name, description, price, and stock.
- **Category CRUD Operations**: Manage product categories (create, update, delete).
- **Product CRUD Operations**: Perform CRUD operations for products, including associating products with categories.

You will implement this system using **Razor Pages**, **N-Tier Architecturec**, and the **Repository Pattern** to ensure separation of concerns and scalability.

---

## Assessment Requirements:

**Step 1: Set Up the Razor Pages Project**

1. **Create a new ASP.NET Core Razor Pages project** in Visual Studio or Visual Studio Code.
    - Select the Razor Pages template.

    - Add the necessary dependencies in your csproj file for **Entity Framework Core**, **SQL Server** (or an in-memory database for testing purposes), and **AutoMapper** (if needed).
    - Implement database migrations if necessary.

---

**Step 2: Implement N-Tier Architecture**

Your application should follow a **3-Tier Architecture**:

- **Presentation Layer**: Razor Pages for handling user interface and HTTP requests.
- **Business Logic Layer**: Service layer to contain business logic.
- **Data Access Layer**: Repository layer to abstract and manage data access.

Structure your project into the following folders:

1. **DataAccess**: Contains Repositories and DbContext.
2. **Services**: Contains the business logic for handling product and category operations.
3. **Models**: Contains the Product and Category models.
4. **Pages**: Contains Razor Pages for CRUD operations and views.

---

**Step 3: Define Product and Category Models**

1. **Product Model**:
    - Define a Product class with the following properties:
        - Id (int): Primary key.
        - Name (string): Product name.
        - Description (string): Product description.
        - Price (decimal): Price of the product.
        - StockQuantity (int): Quantity of stock available.
        - CategoryId (int): Foreign key for the category the product belongs to.
        - Navigation Property: Category (Navigation property to Category).

2. **Category Model**:
    - Define a Category class with the following properties:
        - Id (int): Primary key.
        - Name (string): Category name.
        - Navigation Property: Products (Navigation property to Product).

---

**Step 4: Implement the Repository Pattern**

1. **Create Interfaces** for the repositories:
    - IProductRepository for managing CRUD operations for Product.
    - ICategoryRepository for managing CRUD operations for Category.
2. **Implement the Repository classes**:
    - ProductRepository class should implement IProductRepository.
    - CategoryRepository class should implement ICategoryRepository.
    - Ensure methods like Add, Update, Delete, GetById, and GetAll are included in the repositories.
3. **DbContext Configuration**:
    - Configure the ApplicationDbContext class for both Product and Category models.

---

**Step 5: Implement the Service Layer**

1. **Create a ProductService**:
    - Implement a ProductService class that uses IProductRepository to handle business logic related to products (e.g., calculating discounts, validating stock, etc.).
2. **Create a CategoryService**:
    - Implement a CategoryService class that uses ICategoryRepository to handle business logic related to categories.

---

**Step 6: Implement CRUD Operations for Products and Categories**

1. **Product CRUD Operations**:
   - Implement the following CRUD operations in Razor Pages for products:
     - **Create**: A form for creating a new product.
     - **Read**: A page for listing products (e.g., Index.cshtml).
     - **Update**: A form for updating an existing product.
     - **Delete**: A button for deleting a product.
2. **Category CRUD Operations**:
   - Implement the following CRUD operations for categories:
     - **Create**: A form for creating a new category.
     - **Read**: A page for listing categories.
     - **Update**: A form for updating an existing category.
     - **Delete**: A button for deleting a category.

---

**Step 7: Home and Details Pages**

1. **Home Page**:
   - The **Home** page should display a list of products in a table or card layout with the following information: Name, Price, StockQuantity, and Category.
   - Implement a button or link for each product that directs to the **Details Page**.
2. **Product Details Page**:
   - The **Details Page** (Details.cshtml) should display the detailed information for a single product:
     - Name, Description, Price, StockQuantity, and Category.
   - Implement an "Edit" button to navigate to the "Edit Product" page, and a "Delete" button to delete the product.

---

**Step 8: Implement Data Validation and Error Handling**

1. **Data Validation**:
    - Use data annotations in your models (e.g., [Required], [Range], [StringLength]) for input validation.
2. **Error Handling**:
    - Implement error handling for CRUD operations (e.g., show error messages if a product can't be saved due to database issues).

---

**Step 9: Testing and Postman**

1. **Test the application** using the built-in Razor Pages.
    - Add, view, update, and delete products and categories.
    - Ensure the data is saved in the database and displays correctly on the Home and Details pages.

---

**Step 10: Documentation**

1. **Project Setup Documentation**:
    - Provide detailed steps for setting up the project, including configuration and folder structure.
2. **CRUD Operations Documentation**:
    - Document how the CRUD operations work for both products and categories, and how they interact with the database using the repository pattern.
3. **Screenshots**:
    - Include screenshots of the **Home Page**, **Details Page**, and forms for adding/editing products and categories.
    - Provide screenshots showing the successful creation and deletion of products and categories.
4. **Repository Pattern Explanation**:
    - Briefly explain the use of the **Repository Pattern** and how it helps with separation of concerns and maintainability in your application.

---

## Submission Requirements:

1. **Source Code**: Submit the entire solution, including all files and configurations.
2. **Documentation**: Provide the documentation in PDF or markdown format.
3. **Screenshots**: Include relevant screenshots showing your CRUD operations in action (Product List, Product Details, and Category management).

---

## Evaluation Criteria:

1. **Correctness**:
   - The CRUD operations for both products and categories should function correctly.
   - Data should be saved and retrieved correctly from the database.
2. **Code Quality**:
   - Clean, maintainable code following N-Tier Architecture.
   - Proper use of the **Repository Pattern** and separation of concerns.
3. **Razor Pages Implementation**:
   - Correct implementation of Razor Pages for displaying and interacting with products and categories.
   - Clean and user-friendly UI.
4. **Documentation**:
   - Clear, detailed documentation explaining the project setup, CRUD operations, and screenshots of working features.
5. **Error Handling**:
   - Proper validation and error handling for form submissions and CRUD operations.

---

**Evaluation Criteria for the Product Management System Assessment**

When evaluating the tasks and the final product, the following criteria can be applied to assess both the technical aspects and the overall quality of the solution. The evaluation should cover **functional correctness**, **code quality**, **architecture**, **usability**, and **documentation**.

---

**1. Functional Correctness (40%)**

- **CRUD Operations**:
  - **Product CRUD**: Does the system allow the creation, reading, updating, and deletion of products correctly?
  - **Category CRUD**: Does the system allow the creation, reading, updating, and deletion of categories correctly?

    - Ensure that products are correctly associated with categories and updates reflect accurately in the database.
- **Data Validation**:
    - Are the forms for adding and editing products and categories properly validated (e.g., required fields, correct data types)?
    - Are error messages displayed when invalid data is entered?
- **Navigation**:
    - Is the navigation between pages (Home, Product Details, and Edit/Delete) functional and intuitive?
    - Do the "Details" and "Home" pages display correct data and allow users to perform relevant actions?
- **Error Handling**:
    - Does the system handle errors (such as failed database operations, invalid inputs) gracefully, providing helpful error messages to users or logging errors for debugging?

---

**2. Code Quality (20%)**

- **Clean and Maintainable Code**:
    - Is the code clean, readable, and properly commented?
    - Are methods and classes of manageable size, following single-responsibility principles?
- **Adherence to Best Practices**:
    - Is the project structured according to the **N-Tier Architecture**?
        - Ensure that the presentation layer (Razor Pages) is cleanly separated from the business logic (Services) and data access (Repositories).
    - Proper use of **Repository Pattern** to abstract data access logic.
    - Use of Dependency Injection for services and repositories.
- **Consistency**:
    - Is naming consistent throughout the codebase (variables, methods, classes, etc.)?

    - Are coding conventions followed (e.g., camelCase for variables, PascalCase for class names)?

- **Separation of Concerns**:
    - Are business logic, data access, and presentation concerns properly separated (using N-Tier Architecture)?
- **Testing and Debugging**:
    - Have you tested the CRUD operations to ensure they perform correctly and have no major bugs?

---

**3. Architecture and Design (15%)**

- **N-Tier Architecture Implementation**:
    - Has the N-Tier Architecture been properly implemented, with clear separation between the presentation layer (Razor Pages), business logic layer (Service classes), and data access layer (Repository classes)?
- **Repository Pattern Usage**:
    - Is the **Repository Pattern** correctly used to separate concerns between business logic and data access?
    - Are repositories only concerned with data manipulation and querying (no business logic)?
- **Scalability**:
    - Can the system easily scale if more features (such as user authentication, order management) need to be added? (This is more about ensuring a clean, extendable architecture).
- **Code Reusability**:
    - Are code components reusable and modular? For example, can the repository or service layer be reused for additional entities in the future?

---

**4. User Interface and Usability (15%)**

- **UI Design and Responsiveness**:
    - Does the **Home Page** display products in an intuitive, user-friendly layout?

    - Does the **Details Page** display comprehensive product information in a clear and attractive format?
    - Are forms for adding and editing products/categories simple to use and easy to understand?
    - Is the UI responsive (works well on different screen sizes, or is it at least properly structured for desktop viewing)?
- **Ease of Navigation**:
    - Is it easy for users to navigate between the list of products, product details, and product/category management pages?
    - Is it clear how users can add, edit, and delete products and categories?
- **Interactivity**:
    - Are actions (like adding a product, editing, and deleting) accompanied by intuitive UI feedback (e.g., confirmation messages, success/failure notifications)?

---

**5. Documentation (10%)**

- **Project Setup and Configuration**:
    - Are the instructions for setting up the project clear and easy to follow? Does the documentation cover all setup steps (dependencies, database configurations, etc.)?
- **Code Explanation**:
    - Does the documentation explain key design decisions, such as the choice of N-Tier Architecture and Repository Pattern?
    - Are the CRUD operations and their interactions with the database clearly documented?
- **API and UI Interaction**:
    - Are the **Razor Pages** (especially CRUD operations) well documented, including expected request/response formats (if applicable)?
- **Screenshots**:
    - Are there relevant screenshots included that demonstrate the application in action, such as the **Home Page**, **Product Details**, and **CRUD forms**?

---

**6. Postman Collection (Optional) (5%)**

- **API Testing (if applicable)**:
  - If API endpoints (for example, for product management) are exposed, does the Postman collection include proper testing for CRUD operations?
  - Are the requests clearly defined with headers, parameters, and body data as required by the endpoints?
- **Postman Test Assertions**:
  - Does the Postman collection include tests to verify the responses (e.g., correct status code, returned data)?