Assessment Material for Holistic Mission Code: ECU–SD–04–02

Holistic Mission Title: E-Commerce Project

Mission Specifications:

You are a software specialist at Company X, a fast-growing technology firm specializing in building cutting-edge e-commerce solutions. The company has recently been awarded a new contract to develop a modern e-commerce platform for a client looking to enhance their online store. Your task is to build a functional e-commerce web API using **.NET** that will allow customers to perform basic CRUD operations for products, customers, and orders, browse products, add items to their cart, place orders, and manage their account. This platform must follow best practices in **Web API development**, utilizing **dependency injection (DI)**, the **repository pattern**, and **DTOs (Data Transfer Objects)** for efficient data handling.

1. **Set Up Environment**
   - Check for necessary packages and libraries.
2. **Create Models.**
   - Define core models such as Product, Customer, Order, Shopping Cart.

   Implement appropriate relationships.
   - Customer ↔ Order (1-to-Many)
   - Order ↔ Products (Many-to-Many).
   - Customer ↔ Shopping Cart (1-to-1)

1. **Product:**
   - Id: Integer (Auto-incremented)
   - Name: String (Required)
   - Description: String (Max-Length (100))
   - Stock Quantity: Int (Required)
   - Order: Many Order (Many-to-Many)
2. **Customer**
   - Id: Integer (Auto-incremented)
   - Name: String (Required)
   - Contact: (Validate as Phone number)
   - Email: (Validate as Email address)
   - Shopping Cart: (One-to-One)
   - Orders: Many Orders(one-to-many)
3. **Order**
   - Id: Integer (Auto-incremented)
   - Total Price: int (Required)
   - Product: Collection of Products (Many-to-Many)
   - Customer: One Customer (Many-to-One)

4. **Shopping Cart:**
   - Id: Integer (Auto-incremented)
   - Number of items: Int (Required)
   - Customer: One Customer (One-to-One)

3.Create Controllers for Product/Customer/Order/Shopping Cart:

- Each controller should support CRUD operations for its entity.
- Use DTOs to manage binding of data with models.
- Use Dependency Injection to interact with a Repository that manages data.

1. **Implement the following actions in `CustomerController` using the Repository Pattern:**

- **`POST /api/customers:`**
  - Adds a new customer with new Shopping cart (Object) and new list of Orders.
  - Required: Name, Contact (Phone Number), Email (Valid Email Address), Shopping Cart (Object), and Orders (List).
  - Returns 400 Bad Request for validation failures.
  - Returns 200 OK for successful update.
- **`GET /api/customers/id:`**
  - Retrieves a specific customer by id and related Shopping Cart and list of Orders with list of products.
  - Required: Name, Contact, Email, and Shopping Cart (Object), Orders (List) and Products (List).
  - Returns 404 Not Found if the customer does not exist.

2. **Implement the following actions in `ProductsController` using the Repository Pattern:**

- **`POST /api/products:`**
  - Adds a new product.
  - Required: Name, Stock Quantity, Description.
  - Returns 400 Bad Request for validation failures (e.g., missing required fields or invalid data).
  - Returns 200 OK for successful update.

3. **Implement the following actions in `OrderController` using the Repository Pattern:**

- **`GET /api/orders:`**
  - Retrieves a list of all orders, and related Products and Customer information.
  - Required: Includes details such as Id, Total Price, Products (List) and Customer (Object).
  - Returns 200 OK for successful retrieval, or 404 Not Found if no orders exist.
- **`POST /api/orders:`**
  - Adds a new order with list of new products and already existing Customer by its Id.
  - Required: Total Price, Products (List), CustomerId.
  - Returns 400 Bad Request for validation failures (e.g., invalid data, missing fields).
  - Returns 200 Created for successful creation.
- **`PUT /api/orders/id:`**
  - Updates an existing order by id with updating the list of its products.
  - Required: Total Price, Products (List).
  - Returns 400 Bad Request for validation errors or if the order does not exist.
  - Returns 200 OK for successful update.

- **`Delete /api/orders/id:`**
  - Delete an existing order by id.
  - Returns 400 Bad Request for validation errors or if the order does not exist.
  - Returns 200 OK for successful update.

4. Create DTOs

- Use Data Transfer Objects (DTOs) for efficient data transfer between client and server.
- Create DTOs for Product, Customer, Order, Category and Shopping Cart.

5. Implement Repository Pattern & Dependency Injection

- Implement repository interfaces for Product, Customer, and Order, Shopping-Cart
- Register services in the Program.cs file for dependency injection.

6. Testing and Validations.

- Perform thorough testing using tools like **Swagger** to ensure all the endpoints are working correctly. You can test your methods (GET, POST, PUT, and DELETE). on your controllers, as well as your validations on your properties.

Holistic Assessment Elements:

First Element: Develop an Application with Web API Technology

Evidence and proof requirements:

1- Design models and managing relationships
2- Create DTOs for data transfer
3- Implement repository pattern and DI
4- Develop controllers for BMS functions
5- Enforce validation rules

#	Student's Performance (Steps)
1	Install necessary packages and libraries.
2	Configure a connection string to connect to the database in `appsettings.json` and `Program.cs`.
3	Create `ApplicationDbContext` file.
4	Apply migration and update database.
5	Define core models/classes.
6	Apply appropriate validations on your properties.
7	Implement appropriate relationships between tables.
8	Create appropriate controllers.
9	Inject your dependencies into your controllers.
10	Develop your controllers to handle HTTP requests.
11	Create repository interfaces for entities **with all required endpoints.**
12	Create repository concrete classes **with all required endpoints**
13	Register your dependencies in `Program.cs` file.
14	Inject the appropriate dependencies into the repository class.
15	Apply required methods in the repository file.
16	Return the appropriate response from the repository file.
17	Return the appropriate status code from the controller class.
18	Apply input validations in `ProductController` for product creation and updates.
19	Apply input validations in `CustomerController` for customer details.
20	Apply input validations in `OrderController` for customer details.
21	Test **`POST/api/Products`** to add a new Product using **swagger.**
22	Test **`GET/api/customers`** to fetch customer data using **swagger.**
23	Test **`POST/api/customers`** to add a new customer using **swagger..**