Build a Simple Product API using .NET Core

Objective:

The goal of this challenge is to create a RESTful API using .NET Core that performs basic CRUD operations on a "Product" resource. The API should allow users to create, retrieve, update, and delete products in a database.

Requirements:

1. **API Endpoints**:
    - **GET** `/api/products` – Get a list of all products.
    - **GET** `/api/products/{id}` – Get a single product by its ID.
    - **POST** `/api/products` – Create a new product.
    - **PUT** `/api/products/{id}` – Update an existing product by its ID.
    - **DELETE** `/api/products/{id}` – Delete a product by its ID.
2. **Product Model**:
    - `Id` (Guid): Auto-generated primary key.
    - `Name` (string): The name of the product.
    - `Price` (decimal): The price of the product.
    - `Description` (string): A brief description of the product.
    - `Stock` (int): Quantity of the product in stock.
3. **Tech Stack**:
    - Use **.NET Core** (version 8) to build the API
    - Docker
    - Localstack (https://hub.docker.com/r/localstack/localstack)
        - Download the image and create a container with this image
        - Use DynamoDB to save the Product Data
            1. Use single-table design https://aws.amazon.com/blogs/compute/creating-a-single-table-design-with-amazon-dynamodb/ for it
    - Use **Dependency Injection** for managing services.
    - Use AWS SDKs to **manage database operations** https://www.nuget.org/packages/AWSSDK.Core/3.7.400.20 and https://www.nuget.org/packages/AWSSDK.DynamoDBv2/3.7.400.20
    - Use Fluent https://www.nuget.org/packages/FluentValidation.AspNetCore for **Model Validation** to ensure valid data is passed to the API.
    - Include basic **error handling** (e.g., return 404 for not found resources).

Task Instructions:

Setup the Project:

- Create a new .NET Core Web API project.
- Set up the folder structure with `Controllers`, `Models`, and `Data` folders.
- Install the necessary NuGet packages, including FluentValidation.AspNetCore for model validation and AWS SDK to handle the DynamoDB libraries.

Create the Product Model: Define a `Product` class that represents the product entity with properties for `Id`, `Name`, `Price`, `Description`, and `Stock`.

```
public class Product
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
    public string Description { get; set; }
    public int Stock { get; set; }
}
```

Create the Data Access: Set up a `Repository` class to manage data operations.

```
public class Repository
{
    public Repository(IAmazonDynamoDB dynamoDbClient, DynamoDbConfiguration
configuration)
}
```

Product CRUD Operations: Implement the following methods in the `ProductController`:

- `GetProduct` - Return a single product by ID.
- `CreateProduct` - Add a new product to the database.
- `UpdateProduct` - Update an existing product by ID.
- `DeleteProduct` - Remove a product by ID.

Here's an example of a basic controller:

```
[ApiController]
[Route("api/[controller]")]
public class ProductsController : ControllerBase
{
```

```
// GET: api/products/{id}
[HttpGet("{id}")]
public ActionResult<Product> GetProduct(int id)
{
    var product = _service.GetProduct(id);

    if (product == null)
    {
        return NotFound();
    }

    return product;
}

// POST: api/products
[HttpPost]
public ActionResult<Product> CreateProduct(Product product)
{
    _service.CreateProduct(product);

    return Created();
}

// PUT: api/products/{id}
[HttpPut("{id}")]
public IActionResult UpdateProduct(Guid id, Product updatedProduct)
{
    var product = _service.GetProduct(id);

    if (product == null)
    {
        return NotFound();
    }

    product.Name = updatedProduct.Name;
    product.Price = updatedProduct.Price;
    product.Description = updatedProduct.Description;
    product.Stock = updatedProduct.Stock;

    _service.UpdateProduct(product);

    return NoContent();
```

```
    }

    // DELETE: api/products/{id}
    [HttpDelete("{id}")]
    public IActionResult DeleteProduct(int id)
    {
        var product = _service.GetProduct(id);

        if (product == null)
        {
            return NotFound();
        }

        _service.DeleteProduct(product);

        return NoContent();
    }
}
```

Testing the API: Ensure that the API is functional and that each CRUD operation works as expected. Use tools like **Postman** or **curl** to test the endpoints.

Bonus (Optional):

1. Add **Swagger/OpenAPI** to auto-generate API documentation. You can install the `Swashbuckle.AspNetCore` package for this.
2. Implement **Unit Tests** for the controller using `Microsoft.NET.Test.Sdk` or `MSTest`.

Submission Guidelines:

- Push your solution to a public GitHub repository.
- Provide instructions in the `README.md` on how to run the project locally.