Estrutura do Projeto - Asp.Net Core MVC

Estrutura do Projeto

Asp.Net Core MVC com camadas, respeitando os princípios SOLID.

O projeto é estruturado de forma modular, com cada camada responsável por uma função específica,

facilitando a manutenção, escalabilidade e a aplicação de boas práticas de desenvolvimento.

Descrição das Camadas e Responsabilidades

-- ProductManagerWeb: Asp.Net Core MVC

-- Controllers: Endpoints responsáveis pelo controle das ações de CRUD e pelas Views.

-- Data: Configuração do acesso ao banco de dados (DbContext, Migrations).

-- Models: Entidades de domínio (Regras de negócio, entidades).

-- Repositories: Camada de acesso a dados, responsável pela interação com o banco de dados.

-- Services: Camada de serviços, que atua como intermediária entre as requisições da controller e a camada de acesso a dados.

-- Views: Camada de apresentação ao usuário (Interface).

-- ProductManagerWeb.Teste: Projeto de testes unitários para o sistema.

Explicação sobre a Escolha de Tecnologias e Padrões de Projeto

O projeto foi desenvolvido utilizando Razor Pages devido à integração simplificada com o ASP.Net. Entity Framework foi escolhido pelo seu eficiente e fácil acesso a dados, permitindo mapeamento objeto-relacional e manipulação de dados de maneira intuitiva. FluentMigrator foi adotado para facilitar

as operações de criação de tabelas, alterações de schema e outras operações de gerenciamento de banco de dados.

FluentValidation foi integrado para permitir validações automáticas das models. Bootstrap foi utilizado

para a estilização das páginas de forma simplificada, garantindo uma interface responsiva e de fácil manutenção.

Desafios Encontrados e Como Foram Solucionados

Durante o desenvolvimento, alguns desafios surgiram, como a integração com o FluentMigrator.

Essa foi a primeira vez em que trabalhei com essa tecnologia, o que gerou algumas dificuldades de integração

com o Entity Framework e com o Docker. Após estudar a documentação e assistir a tutoriais no YouTube, consegui

entender melhor a ferramenta e sua aplicação no projeto.

Outro desafio foi o uso do Docker Hub. Embora já tivesse trabalhado com Docker, nunca havia utilizado o Docker Hub

para trazer uma imagem e executar a aplicação diretamente com o container, sem a necessidade de clonar o projeto.

Após alguns testes, consegui configurar o Docker Hub para facilitar o processo de execução da aplicação em containers.

Plano de Testes

Testes Unitários - ProdutoService

GetProdutosAsync_ShouldReturnListOfProdutos: Verifica se a service retorna uma lista de produtos.

Objetivo: Testar o retorno correto de produtos.

GetProdutoByIdAsync_ShouldReturnProduto_WhenProdutoExists: Verifica se a service retorna o

produto quando ele existe.

Objetivo: Testar o retorno do produto existente.

GetProdutoByIdAsync_ShouldReturnNull_WhenProdutoDoesNotExist: Verifica se a service retorna `null` quando o produto não existe.

Objetivo: Testar o comportamento quando o produto não é encontrado.

InsertProdutoAsync_ShouldCallRepositoryInsert: Verifica se o método de inserção da service chama o repositório.

Objetivo: Testar a inserção no repositório.

UpdateProdutoAsync_ShouldCallRepositoryUpdate: Verifica se o método de atualização da service chama o repositório.

Objetivo: Testar a atualização no repositório.

DeleteProdutoAsync_ShouldCallRepositoryDelete: Verifica se o método de deleção da service chama o repositório.

Objetivo: Testar a remoção do produto no repositório.

ValidarProdutosAsync_ShouldReturnTrue_WhenPredicateMatches: Verifica se a service retorna `true` quando o produto corresponde ao predicado.

Objetivo: Testar a validação de produtos com predicado.

ValidarProdutosAsync_ShouldReturnFalse_WhenPredicateDoesNotMatch: Verifica se a service retorna `false` quando o predicado não encontra correspondência.

Objetivo: Testar a validação de produtos com predicado falho.

Testes Unitários - ProdutoRepository

GetProdutosAsync_ShouldReturnAllProdutos: Verifica se o método retorna todos os produtos armazenados no banco.

Objetivo: Testar o retorno correto de todos os produtos cadastrados.

GetProdutoByIdAsync_ShouldReturnCorrectProduto: Verifica se o método retorna o produto correto quando consultado pelo ID.

Objetivo: Testar a busca por produto utilizando o ID corretamente.

InsertProdutoAsync_ShouldAddProdutoToDatabase: Verifica se o método insere corretamente um produto no banco de dados.

Objetivo: Testar a inserção de um novo produto na tabela `Produtos`.

UpdateProdutoAsync_ShouldUpdateProdutoInDatabase: Verifica se o método atualiza corretamente um produto existente no banco.

Objetivo: Testar a atualização de um produto no banco de dados.

DeleteProdutoAsync_ShouldRemoveProdutoFromDatabase: Verifica se o método remove corretamente um produto do banco de dados.

Objetivo: Testar a exclusão de um produto da tabela `Produtos`.

ValidarProdutosAsync_ShouldReturnTrue_WhenProdutoMatchesCondition: Verifica se o método retorna `true` quando um produto corresponde a uma condição específica.

Objetivo: Testar a validação de um produto com base em um predicado.

ValidarProdutosAsync_ShouldReturnFalse_WhenProdutoDoesNotMatchCondition: Verifica se o método retorna `false` quando nenhum produto corresponde a uma condição específica.

Objetivo: Testar a validação de um produto com base em um predicado falho.