Teste para Desenvolvedor Pleno

Objetivo do Teste

Neste teste, você deverá desenvolver uma API RESTful em **C# com ASP.NET Core e SQLite**, aplicando boas práticas de arquitetura e desenvolvimento para garantir eficiência, segurança e manutenibilidade.

1. Requisitos Funcionais

1. Implementar os métodos CRUD para a entidade **Produto**, com os seguintes atributos:
    - **Id** (auto gerado pelo banco de dados)
    - **Nome** (string, deve ser descritivo e único)
    - **Preço** (decimal, maior que zero)
    - **CategoriaId** (relacionamento com a entidade Categoria)
2. Implementar os métodos CRUD para a entidade **Categoria**, com os seguintes atributos:
    - **Id** (auto gerado pelo banco de dados)
    - **Nome** (string, deve ser descritivo e único)
3. Implementar **autenticação JWT**, garantindo que apenas usuários autenticados possam acessar endpoints protegidos.
4. Implementar **paginação** para a listagem de produtos.
5. Implementar **logs e monitoramento** utilizando ILogger<T>.
6. Implementar um endpoint que permita **consultar o histórico de preços** de um produto.
7. Criar um endpoint que retorne **relatórios e estatísticas**, como:
    - Total de produtos cadastrados
    - Média de preços dos produtos
    - Valor total dos produtos no estoque
8. Aplicar **validações rigorosas** na entrada de dados.
9. Criar um **aplicativo WinForms** que consuma a API, com as seguintes funcionalidades:
    - Interface gráfica com **DataGridView** para listar produtos.
    - Botões para **Criar, Atualizar e Excluir** produtos com base no Grid View.
    - Uso de **HttpClient** para realizar as requisições à API.
    - Models para manipular os dados obtidos da API.

2. Requisitos Técnicos

- Utilizar **ASP.NET Core** para desenvolver a API.
- Utilizar **Entity Framework Core** com **SQLite** para persistência de dados.
- Aplicar arquitetura em **camadas separadas** (Controllers, Services, Repositories, DTOs).
- Criar **testes unitários** para validar as funcionalidades críticas.
- Utilizar **WinForms** para criar o aplicativo cliente que consome a API.

3. Regras de Negócio Avançadas

- O **nome do produto** deve ser armazenado sempre com a primeira letra maiúscula.
- O **preço do produto** não pode ser negativo ou igual a zero.
- Implementar uma **lógica de desconto progressivo**, onde:
    - Se a quantidade comprada for maior que 5, aplicar 5% de desconto.
    - Se for maior que 10, aplicar 10% de desconto.
    - Se for maior que 20, aplicar 15% de desconto.

4. Instruções de Desenvolvimento

1. **Criando um Fork do Repositório**
    - Acesse o repositório fornecido.
    - Clique no botão "Fork" no canto superior direito para criar uma cópia em sua conta GitHub.
    - Clone o repositório do seu fork localmente com git clone <URL_DO_SEU_FORK>.
    - Crie uma nova branch com git checkout -b desenvolvimento.
    - Após implementar as mudanças, faça um commit e um push para o seu repositório.
2. Implementar todas as funcionalidades descritas acima.
3. Testar a API usando Postman ou outra ferramenta similar.
4. Criar uma documentação mínima explicando como rodar o projeto e exemplos de requisições.
5. Desenvolver o **aplicativo WinForms**, garantindo integração com a API.
6. Enviar um link para o repositório atualizado.

5. Explicações Técnicas

Paginação

A paginação permite que grandes volumes de dados sejam retornados de forma eficiente, evitando sobrecarregar o banco de dados e melhorando a experiência do usuário.

Exemplo de implementação no ASP.NET Core:

```
public async Task<IActionResult> GetProdutos(int pageNumber = 1, int pageSize = 10)
{
   var produtos = await _context.Produtos
      .OrderBy(p => p.Nome)
      .Skip((pageNumber - 1) * pageSize)
      .Take(pageSize)
      .ToListAsync();
   return Ok(produtos);
}
```

Chamando o endpoint: GET /api/produtos?pageNumber=1&pageSize=10

Monitoramento e Logs

Para registrar eventos importantes, podemos utilizar ILogger<T> no ASP.NET Core:

```
public class ProdutoService
{
   private readonly ILogger<ProdutoService> _logger;

   public ProdutoService(ILogger<ProdutoService> logger)
   {
      _logger = logger;
   }

   public void AdicionarProduto(Produto produto)
   {
      _logger.LogInformation($"Produto {produto.Nome} adicionado em
{DateTime.UtcNow}");
   }
}
```

Os logs podem ser visualizados no console ou configurados para serem salvos em arquivos.

6. Critérios de Avaliação

✅ Implementação correta dos requisitos funcionais e técnicos.
✅ Boas práticas de **código limpo e organizado**.
✅ Uso correto de **padrões de projeto** e separação de responsabilidades.
✅ Implementação de **testes unitários** e cobertura adequada.
✅ Qualidade da **documentação e instruções de uso**.
✅ Desenvolvimento do **aplicativo WinForms** com uso de DataGridView, HttpClient e Models corretamente implementados.

Boa sorte! 🚀