

Avaliação Técnica C#

1. Objetivo

O objetivo principal desta avaliação é determinar o nível de conhecimento do candidato sobre a implementação de soluções com base na demanda estipulada.

Tenha atenção aos detalhes solicitados, garantindo que todos os requisitos solicitados sejam atendidos de forma adequada.

2. Requisitos

Criar uma API REST em **ASP NET 5.0** que tem o objetivo de gerenciar e cadastrar um produto.

A) Características do produto:

- Id
- Descrição
- Categoria
- Dimensões
- Código
- Referência
- Saldo de Estoque
- Preço
- Ativo

B) Deverá ter categoria:

- Id
- Descrição
- Ativo

C) Deverá possuir operações que contemplem os seguintes métodos de HTTP

- GET
- POST
- PUT
- DELETE
- PATCH

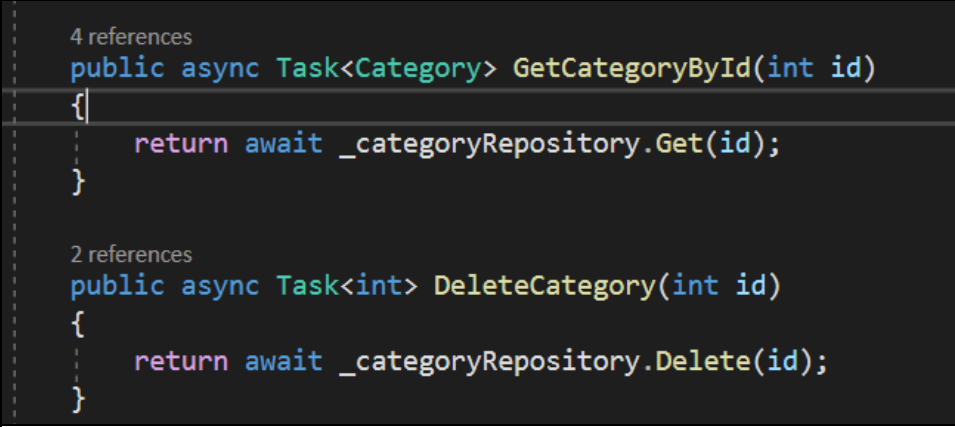
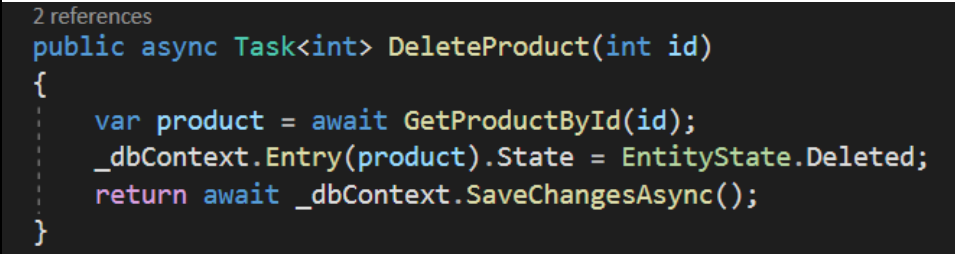
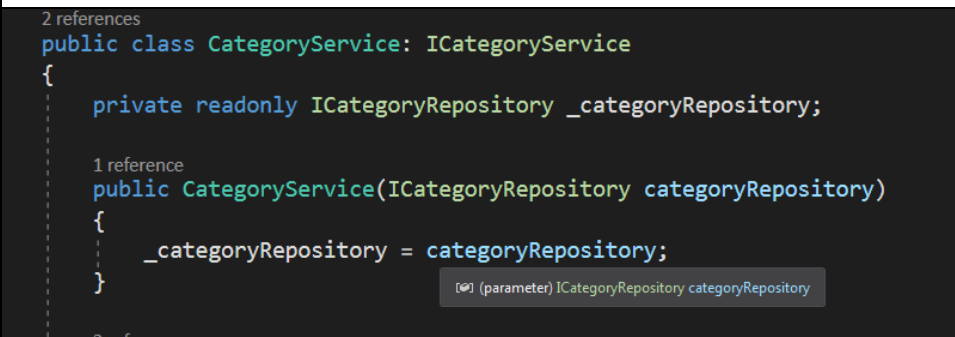
D) O serviço responsável por listar os produtos e categorias deverá ter paginação implementada

E) Utilize o Swagger para documentar a sua API

F) Deverá salvar os dados em um banco de dados SQL Server 2019

G) Utilizar Entity Framework Core com abordagem de Code First

- H) O projeto deverá implementar repository pattern
- I) Utilizar técnicas para apresentar um código limpo, portanto mostre pelo menos 3 exemplos no seu código que você considera estar escrevendo código de forma limpa e organizada e justifique ao lado por qual motivo tomou esta decisão. Para isso você deve fazer um print da parte do código como o exemplo abaixo:

| Imagem | Justificativa |
|--|---|
|  <pre> 4 references public async Task<Category> GetCategoryById(int id) { return await _categoryRepository.Get(id); } 2 references public async Task<int> DeleteCategory(int id) { return await _categoryRepository.Delete(id); } </pre> | <p>Defina bons nomes: Deve ser direto e representar bem o que ele significa, mesmo que isso pressuponha um nome extenso. Bons nomes dispensam comentários. O objeto é olhar e já entender o código.</p> |
|  <pre> 2 references public async Task<int> DeleteProduct(int id) { var product = await GetProductById(id); _dbContext.Entry(product).State = EntityState.Deleted; return await _dbContext.SaveChangesAsync(); } </pre> | <p>Notação húngara não agrega valor: atualmente, entende-se que a notação húngara (uso do tipo da variável logo após o seu nome) polui a leitura e existem novas linguagens de programação mais adequadas. (Deve-se entender o código e seu tipo de variável apenas lendo).</p> <p>DRY: o código DRY “Don’t Repeat It” – conceito trazido por Andy Hunt, um dos autores do Manifesto Ágil -, é aquele sem ambiguidade, ou seja, se você já o inseriu em algum lugar no código-fonte, ele não deve ser implementado novamente.</p> |
|  <pre> 2 references public class CategoryService: ICategoryService { private readonly ICategoryRepository _categoryRepository; 1 reference public CategoryService(ICategoryRepository categoryRepository) { _categoryRepository = categoryRepository; } } </pre> | <p>SRP - Single Responsibility Principle (SOLID):</p> <p>Aqui vemos que a classe tem somente 1 responsabilidade. O objetivo dela é fornecer um serviço para a camada de aplicação, repositório não é a responsabilidade dela. Ela nem precisa conhecer o repositório que vai ser injetado.</p> |

- J) O código deve ter pelo menos 80% de cobertura de testes unitários
- K) Faça um breve resumo dos motivos que te levaram a criar o projeto com a arquitetura escolhida, por exemplo, implementações de interfaces, separações de camadas, separações de pastas, criação das tabelas do banco de dados etc...

RESPOSTA K: Foi escolhida uma arquitetura limpa, com separação lógica de camadas e pastas, e com uma modelagem orientada a domínio, já garantido que o software pode crescer futuramente e esse tipo de modelagem torna a leitura e a manutenção mais fácil de compreender e o máximo desacoplada possível. Conforme solicitado, foi

utilizado o Repository Pattern que possibilita implementar de forma muito mais rápida novos repositórios e mudar de repositório muito facilmente também, apenas realizando a injeção de dependência e/ou inversão de controle. As interfaces desempenham um papel fundamental em tornar as camadas do projeto desacopladas e passíveis de injeção de dependências/inversão de controle.

3. Bonus 1

Caso queira entregar um plus, sugiro a implementação de Autenticação e Autorização (JWT) utilizando Identity nos endpoints criados, estabelecendo pelo menos uma role específica para cada método HTTP.

4. Bonus 2

Implementar a Autenticação e Autorização (JWT) a partir do próprio Swagger.

5. Onde disponibilizar o código fonte?

Você deverá criar um repositório em um controle de versão de sua preferência, seguindo alguns requisitos:

- Deverá ser privado
- Deverá dar acesso para lsilva@alter-solutions.com

6. Dicas

- ✓ Apesar de ser um projeto muito simples, pense em uma arquitetura de um projeto maior, desta forma podemos avaliar a sua capacidade de criar projetos escaláveis e com código reutilizável
- ✓ Lembre que em projetos um pouco maiores teremos a necessidade de mais ambientes, como Development, Production etc...
- ✓ Caso ache necessário você pode utilizar pacotes nuget de sua preferência para implementar suas funcionalidades
- ✓ **Você terá 48 horas para entregar este projeto**