Teste Técnico para Desenvolvedor(a) Pleno(a) C# + Angular

Este teste técnico foi elaborado para nos ajudar a entender melhor suas habilidades e experiência com desenvolvimento backend em C# e frontend com Angular. Buscamos avaliar sua capacidade de resolver problemas, aplicar boas práticas de código, e sua atenção a aspectos como performance, segurança e testes.

Não há um tempo limite estrito para a conclusão. Queremos que você tenha a oportunidade de pensar e desenvolver a solução com calma. O objetivo é avaliar seu raciocínio lógico e a qualidade do seu código, não a velocidade de entrega sob pressão.

Cenário do Desafio: Sistema de Gerenciamento de Configurações de Funcionalidades (Feature Toggles)

Você deverá desenvolver um pequeno sistema que permita gerenciar o estado (ativo/inativo) de funcionalidades de uma aplicação hipotética. Este sistema deve permitir que configurações de funcionalidades possam variar por diferentes ambientes (ex: Desenvolvimento, Homologação, Produção).

Foco do Teste: Aproximadamente 70% da avaliação será no backend (C#) e 30% no frontend (Angular).

Requisitos do Backend (C#)

1. **Modelagem de Dados e Persistência:**
    - Modele e implemente o schema do banco de dados necessário. Sugerimos o uso do Entity Framework Core. Você pode utilizar um banco de dados como SQLite (para simplicidade) ou SQL Server.
    - Entidades sugeridas:
        - FeatureToggle: Deve conter pelo menos Id, NomeUnico (string, para identificação programática), Descricao (string) e AtivoGlobalmente (booleano, indicando se a feature está ativa por padrão).
        - Ambiente: Deve conter pelo menos Id e NomeUnico (string, ex: "DEV", "HML", "PROD").
        - ConfiguracaoAmbienteFeature: Deve relacionar uma FeatureToggle a um Ambiente e especificar se a feature está ativa nesse ambiente específico (AtivoNesteAmbiente - booleano). Esta configuração sobrescreve o estado AtivoGlobalmente da FeatureToggle.
2. **Regra de Negócio Principal:**
    - Ao consultar o estado de uma FeatureToggle para um Ambiente específico:
        1. Verifique se existe uma ConfiguracaoAmbienteFeature para a FeatureToggle e o Ambiente informados.
        2. Se existir, o valor de AtivoNesteAmbiente dessa configuração determina o estado da feature.

3. Caso contrário, o valor de AtivoGlobalmente da FeatureToggle determina o estado.

3. **API Endpoints (ASP.NET Core Web API):**
    - POST /api/featuretoggles: Cria uma nova FeatureToggle.
        - Corpo esperado: { "nomeUnico": "...", "descricao": "...", "ativoGlobalmente": true/false }
    - GET /api/featuretoggles: Lista todas as FeatureToggles cadastradas.
    - PUT /api/featuretoggles/{id}: Atualiza os dados de uma FeatureToggle existente (ex: Descricao, AtivoGlobalmente).
    - POST /api/ambientes: Cria um novo Ambiente.
        - Corpo esperado: { "nomeUnico": "..." }
    - GET /api/ambientes: Lista todos os Ambientes cadastrados.
    - POST /api/featuretoggles/{featureToggleId}/ambientes/{ambienteId}/config: Define ou atualiza a configuração específica de uma FeatureToggle para um Ambiente.
        - Corpo esperado: { "ativoNesteAmbiente": true/false }
    - GET /api/featuretoggles/status?featureName={nomeUnicoFeature}&environmentName={nomeUnicoAmbiente}: Retorna o estado (ativo/inativo) de uma FeatureToggle para um Ambiente específico, aplicando a regra de negócio principal.
4. **Segurança:**
    - Implemente autenticação baseada em token JWT para proteger os endpoints de escrita (POST, PUT).
    - Você não precisa criar um sistema completo de gerenciamento de usuários e senhas. Pode-se:
        - Criar um endpoint simples que gere um token de teste com claims fixas para ser usado nas requisições, OU
        - Mockar a validação do token, focando em demonstrar a configuração da proteção nas rotas.
5. **Testes Unitários:**
    - Crie pelo menos **dois testes unitários** significativos utilizando um framework de sua escolha (xUnit, NUnit, MSTest).
    - Sugestões: Testar a lógica de resolução do estado da feature (regra de negócio principal) e/ou uma action de um controller (validações, comportamento esperado).

Requisitos do Frontend (Angular)

1. **Interface de Usuário:**

- Crie uma interface simples para interagir com o backend. O foco não é a estética, mas a funcionalidade, organização do código e boas práticas do Angular.
- **Páginas/Componentes Sugeridos:**
  - Uma tela para listar as FeatureToggles e seus estados globais. Nesta tela, permita ao usuário alterar o estado AtivoGlobalmente.
  - (Opcional, mas valorizado) Uma forma de visualizar e definir as configurações específicas (ConfiguracaoAmbienteFeature) para uma FeatureToggle selecionada em relação aos Ambientes cadastrados.
  - Uma tela simples para cadastrar novos Ambientes e listá-los.

2. **Funcionalidades:**
   - Consumir os endpoints da API C# desenvolvida.
   - Implementar a comunicação com os endpoints protegidos, enviando o token JWT (você pode ter um campo na interface para o usuário colar um token de teste).
   - Utilize serviços para as chamadas HTTP e organize o código em componentes de forma lógica.
   - Aplique estilos básicos para garantir a usabilidade da interface.

Entregáveis

1. **Código Fonte:** O código completo do backend (C#) e do frontend (Angular) em um repositório Git público ou privado (neste caso, nos forneça acesso).
2. **README.md:** Um arquivo README.md na raiz do repositório contendo:
   - Instruções claras sobre como configurar o ambiente de desenvolvimento.
   - Passos para executar o backend.
   - Passos para executar o frontend.
   - Breve descrição das decisões de arquitetura ou design que você julgar importantes.
   - Como testar a API (ex: exemplos de requisições com cURL, Postman, ou se criou alguma interface para isso).
   - Como obter/utilizar o token JWT para os endpoints protegidos.

Critérios de Avaliação

Analisaremos seu teste considerando:

- **Lógica de Programação e Resolução de Problemas:** Clareza e eficiência na implementação das regras de negócio.
- **Qualidade do Código:** Organização, legibilidade, manutenibilidade e boas práticas de desenvolvimento em C# e Angular.
- **Modelagem de Dados:** Estrutura do banco de dados e uso do ORM.

- **Implementação da API:** Design dos endpoints e conformidade com os requisitos.
- **Segurança:** Implementação da proteção por token.
- **Testes Automatizados:** Qualidade e relevância dos testes unitários.
- **Frontend:** Estrutura dos componentes Angular, comunicação com o backend e usabilidade básica.
- **Documentação:** Clareza e completude do README.md.