Desafio de Código: Catálogo de Receitas Culinárias

Avaliação para Desenvolvedor Júnior I (ASP.NET Core API, Angular Responsivo, SQL Server & Docker)

Introdução

Este desafio tem como objetivo avaliar as suas habilidades práticas no desenvolvimento full stack para um Catálogo Interativo de Receitas Culinárias. O foco principal é na criação de uma API RESTful com ASP.NET Core persistindo dados em SQL Server, no consumo desta API por uma aplicação Angular, e na containerização de ambas as aplicações utilizando Docker.

As habilidades avaliadas são aquelas esperadas para um Desenvolvedor Júnior I, incluindo a aplicação de conceitos de DDD, SOLID e Clean Code no backend.

Concentre-se em escrever um código limpo, bem estruturado, funcional e que atenda aos requisitos propostos. A organização do projeto, a clareza do código e a aplicação dos princípios de design serão consideradas na avaliação.

Escopo do Projeto

Você desenvolverá uma solução composta por:

- **Backend**: Uma API RESTful em ASP.NET Core para gerir receitas, com dados persistidos em SQL Server.
- **Frontend**: Uma aplicação Angular para exibir uma lista de receitas consumindo a API.
- **Containerização**: Ambas as aplicações (backend e frontend) e o banco de dados deverão ser configurados para rodar em containers Docker.

Tarefas

Parte 1: Desenvolvimento do Backend (ASP.NET Core API com SQL Server)

Tarefa 1.1: API de Receitas com Persistência Backend

Crie uma API RESTful utilizando ASP.NET Core para gerir receitas, com persistência de dados em SQL Server.

Requisitos da API:

- Deve expor os seguintes endpoints:
  - GET /api/recipes: Retorna uma lista de todas as receitas.
  - GET /api/recipes/{id}: Retorna uma receita específica pelo seu ID.
  - POST /api/recipes: Cria uma nova receita.
  - DELETE /api/recipes: Deleta uma receita.
- Os dados das receitas devem ser persistidos num banco de dados SQL Server.
- Utilize Entity Framework Core para a camada de acesso a dados.
- Inclua migrations do Entity Framework Core para criar e atualizar o schema do banco de dados.

- A estrutura de uma receita (entidade e DTOs, se usar) deve ser:

```
// Entidade Recipe.cs
public class Recipe
{
    public int Id { get; set; }
    public string Name { get; set; } // Nome da Receita
    public string Description { get; set; } // Breve descrição
    public string Ingredients { get; set; } // Lista de ingredientes (pode ser um JSON string ou texto simples)
    public string Instructions { get; set; } // Modo de preparo
    public int PrepTimeMinutes { get; set; } // Tempo de preparo em minutos
    public string Category { get; set; } // Ex: Sobremesa, Prato Principal, Salada
    public string ImageUrl { get; set; } // URL para uma imagem da receita
}
```

- Configure o CORS na API para permitir requisições da sua aplicação Angular (ex: de http://localhost:4200 ou do container Docker do frontend).

Princípios de Design e Arquitetura (Backend):

- **Clean Code**: Escreva código legível, com nomes significativos. Mantenha métodos curtos e focados.
- **SOLID**: Aplique os princípios, especialmente SRP.
- **DDD**:
  - ❖ Entidade Recipe no seu domínio.
  - ❖ Interface IRecipeRepository no Domínio, Implementação na Infraestrutura utilizando EF Core.
  - ❖ Serviço de Aplicação (ex: 'RecipeAppService') para orquestrar a lógica.

Estrutura do Projeto (Sugestão):

- ❖ SeuProjetoReceitas.Api (Controllers, Program.cs/Startup.cs).
- ❖ SeuProjetoReceitas.Application (Serviços de Aplicação, DTOs).
- ❖ SeuProjetoReceitas.Domain (Entidades, Interfaces de Repositório).
- ❖ SeuProjetoReceitas.Infrastructure (DbContext do EF Core, Migrations, Implementações de Repositório com SQL Server).

Docker:

- ❖ Crie um Dockerfile para containerizar a aplicação ASP.NET Core API.
- ❖ A string de conexão com o SQL Server deve ser configurável (ex: via variáveis de ambiente) para funcionar dentro do ambiente Docker.

Os dados iniciais para popular o banco, podem ser inseridos via seeding no EF Core ou manualmente após a criação das tabelas pelas migrations. Use os dados abaixo como base (adapte os campos conforme a sua entidade Recipe):

```
[

  new Recipe { Name = "Bolo de Chocolate Fofinho", Description = "Um bolo de chocolate clássico, perfeito para qualquer ocasião.", Ingredients = "Farinha, Açúcar, Chocolate em Pó, Ovos, Leite, Fermento", Instructions = "Misture os secos, adicione os molhados, asse por 40 min.", PrepTimeMinutes = 60, Category = "Sobremesa", ImageUrl = "https://placehold.co/300x200/d97706/white?text=Bolo+Chocolate" },

  new Recipe { Name = "Salada Caesar Simples", Description = "Uma salada refrescante e saborosa.", Ingredients = "Alface Americana, Frango Grelhado, Croutons, Molho Caesar", Instructions = "Monte a salada e sirva com o molho.", PrepTimeMinutes = 20, Category = "Salada", ImageUrl = "https://placehold.co/300x200/10b981/white?text=Salada+Caesar" },

  new Recipe { Name = "Lasanha à Bolonhesa", Description = "Uma lasanha rica e reconfortante.", Ingredients = "Massa de Lasanha, Molho Bolonhesa, Molho Branco, Queijo Mussarela", Instructions = "Monte as camadas e asse até dourar.", PrepTimeMinutes = 90, Category = "Prato Principal", ImageUrl = "https://placehold.co/300x200/ef4444/white?text=Lasanha" }

]
```

Parte 2: Desenvolvimento do Frontend (Angular Responsivo)

Tarefa 2.1: Serviço de Receitas Angular (`RecipeService`) Frontend - Serviço

Adapte o serviço Angular `RecipeService`.

- Deve consumir a API ASP.NET Core de receitas. O endereço da API deve ser configurável para facilitar a execução em diferentes ambientes (local vs. Docker).
- Implemente métodos para:
  - ❖ getRecipes(): Chama GET /api/recipes.
  - ❖ getRecipeById(id: number): Chama GET /api/recipes/{id}.
  - ❖ addRecipe(recipe: any): Chama POST /api/recipes.
  - ❖ delRecipe(id: number): Chama Delete /api/recipes/{id}.
- Utilize `HttpClientModule` e `HttpClient`.

Tarefa 2.2: Componentes Angular Responsivos (‘RecipeCardComponent’, ‘RecipeListComponent’) Frontend

Adapte os componentes ‘RecipeCardComponent’ e ‘RecipeListComponent’ com foco total na responsividade e experiência mobile.

‘RecipeCardComponent’:

- Deve exibir os dados da receita (nome, categoria, tempo de preparo, imagem).
- Um botão "Ver Receita" (ou similar) e outros elementos interativos devem ter tamanho e espaçamento adequados para fácil.
- A imagem da receita deve ser responsiva, adaptando-se ao tamanho do card sem distorção.
- Considere a legibilidade do texto em telas menores (tamanho de fonte, contraste).
- Ao clicar em "Ver Receita", deve emitir um evento (ex: ‘viewRecipeDetails’) com o `id` da receita.

‘RecipeListComponent’:

- Deve usar o `RecipeService` para buscar e exibir as receitas.
- A transição entre os layouts deve ser fluida.
- Garanta que a performance de renderização da lista seja boa, especialmente com muitas receitas.
- Adicione um formulário simples para adicionar novas receitas, garantindo que o formulário também seja responsivo e fácil de usar.
- Utilize CSS (Flexbox, Grid, Media Queries) de forma eficaz para alcançar os layouts

Tarefa 2.3: Dockerização do Frontend Frontend - Docker

- Crie um Dockerfile para containerizar a aplicação Angular.
- O Dockerfile deve realizar a build de produção da aplicação Angular (ng build --configuration production).

Parte 3: Orquestração com Docker Compose

Tarefa 3.1: Docker Compose Infraestrutura

- Crie um arquivo docker-compose.yml para orquestrar os seguintes serviços:
- O container da API ASP.NET Core.
- O container da aplicação Angular.
- Um container para o banco de dados SQL Server (utilize a imagem oficial da Microsoft: mcr.microsoft.com/mssql/server).
- Configure as dependências entre os serviços (ex: a API depende do SQL Server).
- Configure as variáveis de ambiente necessárias (ex: string de conexão do SQL Server para a API, URL da API para o frontend se necessário).
- Mapeie as portas para acesso externo (ex: API na porta 8080, Frontend na 4200, SQL Server na 1433).
- Configure volumes para persistência de dados do SQL Server.

O que Entregar

Você deve entregar:

- O código-fonte completo do projeto da API ASP.NET Core (incluindo `Dockerfile`).
- Os arquivos de código-fonte Angular (incluindo `Dockerfile`).
- O arquivo docker-compose.yml.
- Scripts SQL para criação de tabelas e/ou seeding de dados, caso não utilize migrations ou seeding do EF Core para tudo. (Preferencialmente, use migrations/seeding do EF Core).
- Um breve arquivo `README.md` explicando como executar a solução completa com Docker Compose.

Organize os projetos (backend, frontend) em pastas separadas. Se possível, disponibilize tudo num repositório Git.

Critérios de Avaliação

- Funcionalidade Completa e Experiência do Utilizador: A solução integrada (API, Frontend, DB) funciona conforme o esperado via Docker Compose. A experiência do utilizador no frontend, fluida, intuitiva e sem falhas visuais ou de usabilidade.
- Qualidade do Código Backend (ASP.NET Core):
- Aplicação correta de C#, ASP.NET Core, Entity Framework Core.
- Configuração e uso de SQL Server, incluindo migrations.
- Estrutura do projeto, separação de responsabilidades, aplicação de SOLID, Clean Code e DDD (simplificado).
- Dockerfile da API funcional e otimizado.
- Qualidade do Código Frontend (Angular):
- Clareza, legibilidade e organização do código.
- Dockerfile do Frontend funcional, servindo a build de produção.
- Docker e Docker Compose:
- Configuração correta e funcional do `docker-compose.yml`.
- Comunicação entre containers (API <-> DB, Frontend <-> API).
- Persistência de dados do SQL Server.
- Boas Práticas Gerais: Convenções de nomenclatura, README claro e informativo.