**DESAFIO 5.3 – (7) Components**

**Item 1: Reutilização de componentes**

a) Componentes são blocos para dividirmos nossa aplicação para delegarmos tarefas específicas. Desta forma, temos uma aplicação mais organizada, com a capacidade de termos um código com menos acoplamento e, também reutilizar esses componentes.

b) os componentes são compostos por 3 arquivos, html onde fica toda a parte do esqueleto, o que será renderizado, typescript onde fica a parte lógica, e spec se desejado para realizar testes.

c) ng g componente ou n g c somente.

d) utilizando componentes reutilizáveis você diminui seu código, acelera o desenvolvimento, torna mais prático e funcional, melhora o desenvolvimento da aplicação, ao invés de ficar repetindo um código que será usado várias vezes.

e) este seletor é a tag do componente, com ela eu indico no meu html que estarei utilizando este componente.

f) **templateUrl,** quando está sendo indicado o caminho do arquivo modelo do meu template como no exemplo 1, e **template** quando está sendo indicado um hmtl embutido para o meu template como no exemplo 2, pelo o que pesquisei a melhor maneira é utilizar templateUrl pois terá o suporte de inteligência, preenchimento de código e recursos de formatação do Visual Studio.

g) **stylesUrl**: definimos um caminho para uma página de estilos, podendo utilizar vários estilos diferentes, usada para especificar uma matriz de caminhos de folha de estilo vinculados. Melhor para um layout mais organizado de declarações de estilo. **Styles:** definimos diretamente no componente o estilo, usada para especificar uma matriz de estilos declarados. Bom para definições de estilo muito simples.

**Item 2: Rotas**

a) ng new (nome da aplicação) --routing

b) ng generate module (nome do modulo) -- routing

c) navegar entre destinos

d) teste se a instancia esta ativa.

e) para definir as rotas

f) utilizando esta tag estamos informando ao componente que queremos que a visualização das rotas seja feita pelo router-outlet.

g) <button routerLink="cadastrar" (click)="createName()" color="primary">Cadastrar</button> - estou usando um exemplo que fiz no meu ultimo projeto de crud, onde routerlink esta chamando a rota e o método createName ao clique.

h) routerLink="cadastrar"

i) “carregamento-lento” o conteúdo é carregado somente quando é solicitado,isso deixa o site mais leve e tempo de carregamento menor.

{ path: 'items', loadChildren: () => import('./items/items.module').then(m => m.ItemsModule) – utilizando loadChildren.

j) “carregamento-ansioso” – iniciacializado todas as entidades assim que o código é iniciado. Pelas minhas pesquisas entendi que este modo é pouco utilizado e até não recomendado pois aumenta o tempo de carregamento e deixa a aplicação mais pesada.

using (DbTesteContainer context = new DbTesteContainer())

{

   context.ContextOptions.LazyLoadingEnabled =

        false;

    var cliente =

        context.Cliente.**Include**("Telefone").FirstOrDefault();

    try

    {

        Console.WriteLine(

            cliente.Telefone.FirstOrDefault().Numero);

    }

    catch (Exception ex)

    {

        Console.WriteLine(ex.GetBaseException());

    }

**Item 3: Pipes**

a) Pipes são funções simples que recebem um dado e transformam para outro dado.

b) [DatePipe](https://angular.io/api/common/DatePipe), [UpperCasePipe](https://angular.io/api/common/UpperCasePipe), [LowerCasePipe](https://angular.io/api/common/LowerCasePipe), [CurrencyPipe](https://angular.io/api/common/CurrencyPipe),[DecimalPipe](https://angular.io/api/common/DecimalPipe),e [PercentPipe](https://angular.io/api/common/PercentPipe).

c) [DatePipe](https://angular.io/api/common/DatePipe): Formata um valor de data de acordo com as regras locais.

[UpperCasePipe](https://angular.io/api/common/UpperCasePipe): Transforma o texto em maiúsculas.

[LowerCasePipe](https://angular.io/api/common/LowerCasePipe): Transforma o texto em minúsculas.

[CurrencyPipe](https://angular.io/api/common/CurrencyPipe): Transforma um número em uma string de moeda, formatada de acordo com as regras locais.

[DecimalPipe](https://angular.io/api/common/DecimalPipe): Transforma um número em uma string com um ponto decimal, formatado de acordo com as regras de local.

[PercentPipe](https://angular.io/api/common/PercentPipe): Transforma um número em uma string de porcentagem, formatada de acordo com as regras de localidade.

d) Custom Pipes são pipes personalizados, que não vem com o angular, criamos a partir de uma classe, inserindo o modelo de dados que queremos exibir.