Transcrição

Agora que criamos a aplicação base e removemos o *placeholder*, como o *Angular* chama, e o código excessivo que o `ng` disponibilizou, com apenas o `<h1>` e o `<body>` com um belo gradiente, podemos dar o próximo passo.

Lembra que mencionamos que Angular é sobre componentes? É exatamente isso que vamos fazer agora: iremos **criar um componente Angular**.

Criando o primeiro componente

Nós poderíamos criar tudo manualmente, mas já mencionamos que o *Angular CLI* vai nos ajudar em vários momentos da aplicação. Criar um componente é tão comum que o próprio `ng` disponibilizou um comando que podemos utilizar para criar novos componentes.

No terminal principal, está rodando o comando `ng serve` e disponibilizando a aplicação na porta 4200. Vamos minimizá-lo e abrir o

terminal dentro do *VS Code*, com o atalho "Command + J" no *Mac* e "Ctrl + J" no *Windows*. Será aberto o terminal já na pasta do projeto "indexa". Os comandos que vamos executar precisam estar dentro do projeto "indexa".

Agora, vamos criar nosso primeiro componente. Vamos chamar o comando `ng`, mas em vez de chamar o `new` e o `serve`, comandos que já aprendemos, vamos chamar o `generate`.

O Angular gera várias coisas diferentes para adiantar nosso processo. Uma das coisas que ele sabe gerar é componente, mas "componente" no comando é escrito em inglês (`component`), porque é um argumento que passamos. `ng` é o nome do comando, `generate` é o segundo argumento, `component` é o terceiro argumento, e por último vem o nome do componente.

Vamos criar uma pasta chamada "componentes", onde tudo que gerarmos de componentes ficará, apenas para manter organizado. Para dizer ao `ng generate` que queremos gerar um componente dentro de uma pasta, vamos digitar `componentes`, em português, pois estamos passando o nome da pasta; depois colocamos uma barra para o `ng` entender que isso é uma pasta, então ele vai criar essa pasta pra nós.

Antes de criar de fato o primeiro componente, vamos analisar o *Figma*. A aplicação Indexa fica em um *container* branco sólido, que contrasta com o

gradiente. Vamos criar um componente que vai representar esse container, que é o componente mais externo da aplicação.

Para criar o container, vamos completar o comando `ng generate component componentes/` com o nome do componente `container`. Assim, pedimos ao Angular que gere um componente, indicando que queremos que ele fique na pasta "componentes" e que o nome dele componente seja `container`.

```
ng generate component componentes/container
```

COPIAR CÓDIGO

Conhecendo a pasta "*container*"

No VS Code, ao acessar a estrutura de pastas, temos "app > componentes > **container**". Se olharmos o arquivo HTML (`container.component.html`), encontramos o parágrafo `<p>container works!</p>`, algo que o Angular colocou apenas para indicar que o componente funciona.

O arquivo `container.component.ts` é uma **classe**, bem familiar do *TypeScript*.

`container.component.ts`:

```ts
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';

@Component({
  selector: 'app-container',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './container.component.html',
  styleUrl: './container.component.css'
})
export class ContainerComponent {

}
```

COPIAR CÓDIGO

Ele tem um *decorator* chamado `@Component()`, que recebe um objeto de configuração. Esse objeto de configuração diz que o seletor do componente é `app-container`, que ele é `standalone`, importa o `CommonModule`, o template dele é `./container.component.html`, e os estilos estão em `./container.component.css`.

> No momento, falamos de **TypeScript**, e não de Angular.

Alterando o arquivo `app.component.html`

A marcação HTML do componente se chama *template*. A propriedade `styleUrl` indica o caminho para os estilos, que estão no arquivo `container.component.css`, e a primeira propriedade é o seletor (`selector`), definida como `app-container`. Deixamos essa propriedade para o final por uma razão.

Queremos usar esse componente, testar e verificar se esse código funciona. Então, no arquivo `app.component.html`, em vez de colocar a tag `<h1>` diretamente, vamos chamar esse novo componente pelo seletor `<app-container>`. O VS Code irá sublinhar em vermelho, indicando que há algo errado, mas vamos ignorar por enquanto e colocar a tag `<h1>` dentro dela.

`app.component.html`:

```
<app-container>
  <h1>Indexa</h1>
</app-container>
```

COPIAR CÓDIGO

No navegador, temos uma mensagem de erro em vermelho dizendo que `app-container` não é um elemento conhecido. Ele tenta dar algumas dicas

de como resolver isso. Quando codificamos algo errado, o Angular tenta nos ajudar com uma mensagem de erro indicando mais ou menos o que aconteceu.

Como primeiro tópico, o Angular diz que, se `app-container` for um componente do Angular, deve ser verificado se ele está incluso nos `import` do componente. Assim, começamos a falar em **módulos**.

Alterando o arquivo `app.component.ts`

Quando temos um componente, precisamos indicar que ele usa o outro componente. Vamos analisar o arquivo `app.component.ts`. Ele é uma classe TypeScript com o decorator `@Component`. Quem implementou esse decorator foi o Angular, então o objeto que o `@Component` recebe são coisas que o Angular entende.

`app.component.ts`:

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterOutlet } from '@angular/router';

@Component({
```

```
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule, RouterOutlet],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'indexa';
}
```

COPIAR CÓDIGO

Uma das coisas existentes nesse arquivo, no nosso caso, na linha 8, é o `imports`. Ele importa o `CommonModule` e outra coisa chamada `RouterOutlet`. Queremos importar o novo componente que acabamos de criar. Se olharmos em `container.component.ts`, o nome da classe é `ContainerComponent`.

No `imports` do arquivo `app.component.ts`, vamos chamar `ContainerComponent`. O VS Code sugere se nos referimos ao `ContainerComponent` que está dentro da pasta "componentes".

```
import { Component } from '@angular/core';
import { CommonModule } from '@angular/common';
```

```
import { RouterOutlet } from '@angular/router';
import { ContainerComponent } from './componentes/container/

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule, RouterOutlet, ContainerComponent],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'indexa';
}
```

COPIAR CÓDIGO

Ao fazer isso, no arquivo `app.component.html`, onde o VS Code tentava dar uma dica de que algo estava errado, não há mais o sublinhado vermelho.

Conclusão

Se olharmos agora para o navegador, o console estará limpo e temos o texto "container works!" na página, ou seja, nosso componente funcionou.

Porém, queríamos exibir a tag `<h1>Indexa</h1>`, não a mensagem "container works!". Vamos começar a aprender essa linguagem do Angular, para que ele entenda o que tem ali dentro e renderize para nós no lugar correto!