02

Criando o primeiro componente

Transcrição

Imaginemos a seguinte situação em que um novo estagiário integra seu time de desenvolvimento, ou alguém que não conheça Bootstrap. Caso seja solicitada uma imagem responsiva, a pessoa terá que estudar e se inteirar dos detalhes para isso. O que aconteceria se ela componentizasse uma imagem no Angular, com `<ap-photo>` ?

A ideia é tentarmos criar o primeiro componente que ocultará a complexidade de se criar uma imagem responsiva do Bootstrap. Vamos fazer isso?

A convenção que seguiremos é, em "ap", criarmos uma nova pasta chamada "photo", nome que diz respeito ao componente que queremos criar. Nela, criaremos o arquivo `photo.component.ts` , do TypeScript, que terá um arquivo de mesmo nome, porém com extensão `.html` . Sabemos que um componente em TypeScript também é uma classe em Angular.

Em `photo.component.ts` , usaremos `@Component` , e o Visual Code exibirá, dentre outras, a opção de *autoimport*, que selecionaremos, pressionando "Enter". Isso fará com que o programa importe automaticamente o *decorator* `Component` do módulo `@angular/core` , pois toda classe e *decorator* que for importado deverá ser informado — como em um *import* em Java, C#, ECMAScript, e por aí vai.

```
import { Component } from "@angular/core";
```

```
@Component
export class PhotoComponent {}
```

COPIAR CÓDIGO

Entretanto, o programa indica erro e, ao passarmos o mouse sobre `Component`, uma vez que o Visual Code é integrado ao TypeScript, ele nos avisa que precisamos passar uma configuração especial para o `Component`. Portanto, passaremos chaves para indicar a existência de um objeto JavaScript, cuja propriedade é `selector`.

Também informaremos que o nome do componente é `ap-photo`, e incluiremos um template, não diretamente no HTML, e sim em um arquivo separado. Portanto, utilizaremos a opção `templateUrl`.

```
import { Component } from "@angular/core";

@Component({
    selector: 'ap-photo',
    templateUrl: 'photo.component.html'
})
export class PhotoComponent {}
```

COPIAR CÓDIGO

> Prefixar todos os componentes é uma boa prática. Supondo que alguém crie um componente chamado "photo", haverá um conflito de seletores. Neste caso, utilizamos "ap" para remeter a "alurapic".

Quanto ao dado do componente, a ideia é que ele tenha a URL e a descrição. Poderemos inclusive deletar `title = 'alurapic'` de `app.component.ts`. O navegador denunciará muitos erros, mas não nos preocuparemos com isto agora. Deletaremos todo o conteúdo de `app.component.html` e voltaremos a `photo.component.ts` para incluir a URL e descrição que copiamos de `app.component.ts`.

```
export class PhotoComponent {
    description = 'Leão';

    url = 'https://upload.wikimedia.org/wikipedia/commons/thuml
}
```

COPIAR CÓDIGO

Assim, quando o template for renderizado, será carregada uma imagem cuja `class` recebe `img-thumbnail`, e realizaremos um *Data binding* da propriedade `src` com sua `url`, e `alt` com `description`. Em `photo.component.html`, teremos:

```
<img class="img-thumbnail" [src]="url" [alt]="description">
```

COPIAR CÓDIGO

Com isso, a URL do componente será lida, bem como a propriedade `description` do mesmo. Feito isso, no template de `app.component.html`, que se encontra vazio, já que queremos usar o nosso componente em um template de outro, é necessário utilizarmos seu *selector*, sua forma declarativa:

```
<app-photo></app-photo>
```

COPIAR CÓDIGO

Não esqueçamos de salvar todas as alterações feitas.

Antes de visualizarmos o resultado no navegador, pela lógica, ao carregarmos a aplicação, a `ap-photo` deverá ser carregada, e nosso dado deve ser exibido. Porém, nada disso acontece. E se abrirmos o console do navegador com "Ctrl + Shift + i", é exibido erro do Angular.

É importante entendermos tudo isso para que, quando criarmos um componente, ele seja bem executado. A razão deste erro será visto a seguir.