IMT - Instituto Mauá de Tecnologia

Engenharia de Computação

ECM252 – Linguagens de Programação II

Professor Rodrigo Bossini

Aula 11 - Angular - Diretivas customizadas - Pipes

1 Introdução

Como vimos, há diferentes tipos de diretivas no Angular: de atributos, estruturais, componentes. Além de utilizar as diretivas pré-definidas pelo framework, também podemos definir **nossas próprias diretivas**, como veremos neste material.

Além disso, o Angular possui um mecanismo conhecido como **pipe**. Em geral, um pipe é alimentado com um dado qualquer e produz algo que pode ser apresentado na tela para o usuário.

2 Desenvolvimento

2.1 (Novo projeto) Comece abrindo um novo terminal e navegando até o seu workspace, ou seja, a pasta que você está usando para abrigar os seus projetos Angular. Tome o cuidado de não acessar a pasta de nenhum projeto já existente. Use os comandos a seguir para criar um novo projeto Angular e navegar até o seu diretório.

ng new nome-do-projeto
cd nome-do-projeto

- Quando perguntado, escolha não adicionar um módulo para roteamento e use o modelo CSS simples.

- Instale o Bootstrap com

npm install bootstrap@latest

- Abra uma instância do VS Code vinculada ao diretório atual com o seguinte comandos

code .

- A seguir, configure o Bootstrap no arquivo **angular.json** adicionando o arquivo **node_modules/bootstrap/dist/css/bootstrap.css** ao vetor cuja chave é **projects/nome-do-projeto/architect/build/options/styles**.

- Coloque o servidor de testes em execução com

ng serve --open

2.2 (Estrutura inicial) Nossa aplicação terá um campo textual e um botão que permite ao usuário cadastrar lembretes. Os lembretes são exibidos logo abaixo, em uma lista simples. Apague o conteúdo do arquivo **app.component.html** e adicione a ele o conteúdo da Lilstagem 2.2.1.

Listagem 2.2.1

```
<div class="container">
  <div class="row">
    <div class="form-group col-12">
      <label for="lembreteInputText">Lembrete</label>
      <input type="text" id="lembreteInputText" placeholder="Digite seu lembrete" class="form-
control">
    </div>
  </div>
  <div class="row">
    <div class="col-12">
      <button type="button" class="btn btn-primary btn-block">Salvar</button>
    </div>
  </div>
</div>
```

- Quando o usuário clicar no botão, desejamos adicionar o lembrete que ele tiver digitado no campo. Para isso, vamos criar uma lista no componente. Além disso, vamos criar uma função que é chamada quando o botão é clicado. Vamos usar two-way data binding para vincular o que o usuário digitou a uma variável do componente. Veja os ajustes no arquivo HTML na Listagem 2.2.2.

Listagem 2.2.2

```
<div class="container">
  <div class="row">
    <div class="form-group col-12">
      <label for="lembreteInputText">Lembrete</label>
      <input type="text" id="lembreteInputText" [(ngModel)]="lembrete" placeholder="Digite
seu lembrete"
        class="form-control">
    </div>
  </div>
  <div class="row">
    <div class="col-12">
      <button type="button" class="btn btn-primary btn-block"
(click)="salvar()">Salvar</button>
    </div>
  </div>
  <div class="row">
    <table class="table table-striped table-hover mt-3">
      <thead>
        <tr>
          <th class="text-center">Lembretes</th>
        </tr>
      </thead>
      <tbody>
        <tr *ngFor="let lembrete of lembretes">
          <td class="text-center">{{lembrete}}</td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
```

- A Listagem 2.2.3 mostra as variáveis e função que devem ser declaradas no arquivo **app.component.ts.**

Listagem 2.2.3

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  lembretes: Array<string> = [];
  lembrete: string;


  salvar() {
    this.lembretes = [this.lembrete, ...this.lembretes];
    this.lembrete = '';
  }
}
```

- Não se esqueça de, no arquivo **app.module.ts**, importar o módulo **FormsModule**, pois a diretiva **ngModel** pertence a ele. Veja a Listagem 2.2.4.

Listagem 2.2.4

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms'

import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

2.3 (Diretiva personalizada para colocar sombra na tabela) Vamos criar uma nova diretiva que simplesmente coloca uma sombra na tabela. O objetivo é ilustrar o funcionamento básico de uma diretiva e a forma como ela pode ser criada.

- Para criar uma diretiva com o Angular CLI, use o seguinte comandos

ng g d sombra-na-tabela --skipTests=true

- Avalie o código gerado. Note que é semelhante a um componente. Há, inclusive, um seletor que é o nome por meio do qual a diretiva pode ser aplicada a algum componente.

- Para adicionar borda à tabela, precisamos especificar que o construtor da diretiva (que é chamado quando ela é usada recebe:

- Um ElementRef, que dá acesso ao elemento sobre o qual ela será aplicada
- Um Renderer2, que é uma classe do Angular que permite a manipulação de propriedades referentes à renderização de componentes.

Veja a Listagem 2.3.1.

Listagem 2.3.1

```
import { Directive, ElementRef, Renderer2 } from '@angular/core';

@Directive({
  selector: '[appSombraNaTabela]'
})
export class SombraNaTabelaDirective {

  constructor(
    private elementRef: ElementRef,
    private renderer: Renderer2
  ) { }

}
```

- A seguir, vamos usar o objeto renderer para aplicar um estilo desejado ao elemento que pode ser obtido pelo ElementRef. Veja a Listagem 2.3.2.

Listagem 2.3.2

```
import { Directive, ElementRef, Renderer2 } from '@angular/core';

@Directive({
  selector: '[appSombraNaTabela]'
})
export class SombraNaTabelaDirective {

  constructor(
    private elementRef: ElementRef,
    private renderer: Renderer2
  ) {
    this.renderer.setStyle(
      this.elementRef.nativeElement,
      'box-shadow', '10px 10px'
    )
  }
}
```

- Para aplicar a diretiva a um elemento qualquer, basta utilizar o seu seletor, definido nos metadados aplicados à classe TypeScript. Nesse caso, ele se chama **appSombraNaTabela**. Veja a sua aplicação sobre a tabela na Listagem 2.3.3.

Listagem 2.3.3

```
<table appSombraNaTabela class="table table-striped table-hover mt-3">
```

- Note que o seletor está entre **colchetes**. Neste caso, temos um seletor de atributo. Ou seja, o nome do seletor é aplicado como se fosse um atributo ao elemento de interesse, exatamente como fizemos. Há outros tipos de seletores. É possível, por exemplo, selecionar os elementos pelo seu tipo. Veja o exemplo da Listagem 2.3.4. Para testar, remova a aplicação da diretiva do elemento table, como destacado.

Listagem 2.3.4

```
@Directive({
  selector: 'table'
})
<table class="table table-striped table-hover mt-3">
```

2.4 (Lidando com eventos) Digamos que desejamos aplicar a sombra à tabela somente quando o mouse estiver sobre ela. Para isso, vamos começar criando um método que contém o código que escrevemos no construtor no exemplo anterior. Veja o ajuste da Listagem 2.4.1.

Listagem 2.4.1

```
quandoOMousePassarPorCima() {
   this.renderer.setStyle(
     this.elementRef.nativeElement,
     'box-shadow', '10px 10px'
   )
  }
```

- Dizemos que o elemento sobre o qual uma diretiva é aplicada é seu **Host**. Assim, precisamos dizer que esse método é seu "observador". Isso pode ser feito por meio da anotação **HostListener**. Entre parênteses, especificamos o evento de interesse. Veja a Listagem 2.4.2. Certifique-se de que a diretiva foi aplicada como um atributo, como destacado.

Listagem 2.4.2

```
 @HostListener('mouseover') quandoOMousePassarPorCima() {
   this.renderer.setStyle(
     this.elementRef.nativeElement,
     'box-shadow', '10px 10px'
   )
  }
<table appSombraNaTabela class="table table-striped table-hover mt-3">
```

- Note que o efeito é aplicado quando o mouse passa sobre o elemento. Contudo, o efeito permanece depois que o mouse sai. Podemos registrar outro observador que irá remover o estilo mediante este evento. Veja a Listagem 2.4.3.

Listagem 2.4.3

```
@HostListener('mouseleave') quandoOMouseSair() {
   this.renderer.removeStyle(
     this.elementRef.nativeElement,
     'box-shadow');
  }
```

- Como um teste, aplique a diretiva ao elemento input como na Listagem 2.4.4 e veja o resultado.

Listagem 2.4.4

```
<input appSombraNaTabela type="text" id="lembreteInputText" [(ngModel)]="lembrete"
      placeholder="Digite seu lembrete" class="form-control">
```

- Uma outra forma para se fazer a aplicação de estilos é por meio do vínculo de propriedades do hospedeiro (o elemento sobre o qual a diretiva foi aplicada). Para começar, declare uma variável e comente o código de configuração como na Listagem 2.4.5.

Listagem 2.4.5

```
import { Directive, ElementRef, Renderer2, HostListener } from '@angular/core';

@Directive({
  selector: '[appSombraNaTabela]'
})
export class SombraNaTabelaDirective {

  sombra: string;

  /*constructor(
    private elementRef: ElementRef,
    private renderer: Renderer2
  ) {

  }*/

  @HostListener('mouseover') quandoOMousePassarPorCima() {
    /*this.renderer.setStyle(
      this.elementRef.nativeElement,
      'box-shadow', '10px 10px'
    )*/
    this.sombra = '10px 10px';
  }

  @HostListener('mouseleave') quandoOMouseSair() {
    /*this.renderer.removeStyle(
      this.elementRef.nativeElement,
      'box-shadow');*/
    this.sombra = "";
  }
}
```

- O que desejamos fazer a seguir é vincular a variável **sombra** a uma propriedade do elemento hospedeiro. Para isso, vamos usar a anotação **HostBinding**. Entre parênteses, dizemos qual a propriedade cujo valor deve ser atribuído de acordo com o valor da propriedade sombra. Veja a Listagem 2.4.6.

Listagem 2.4.6

```
@HostBinding('style.boxShadow') sombra: string;
```

- Salve todos os arquivos e teste novamente.

- Quando aplicamos uma diretiva a um elemento, podemos entregar a ela valores a serem utilizados internamente. Por exemplo, o valor de sombra por enquanto está fixo. Nada que impede que especifiquemos um valor diferente para cada elemento a que a diretiva seja aplicada. Para começar, declare a variável da Listagem 2.4.7 e faça as atribuições destacadas nos métodos.

Listagem 2.4.7

```
import { Directive, ElementRef, Renderer2, HostListener, HostBinding } from '@angular/core';

@Directive({
  selector: '[appSombraNaTabela]'
})
export class SombraNaTabelaDirective {
  @HostBinding('style.boxShadow') sombra: string;

  /*constructor(
    private elementRef: ElementRef,
    private renderer: Renderer2
  ) {

  }*/
  sombraEntrada: string;

  @HostListener('mouseover') quandoOMousePassarPorCima() {
    /*this.renderer.setStyle(
      this.elementRef.nativeElement,
      'box-shadow', '10px 10px'
    )*/
    this.sombra = this.sombraEntrada;
  }

  @HostListener('mouseleave') quandoOMouseSair() {
    /*this.renderer.removeStyle(
      this.elementRef.nativeElement,
      'box-shadow');*/
    this.sombra = "";
  }
}
```

- Para que uma diretiva possa receber um valor de um elemento, aplicamos a anotação @**Input** à variável que receberá o valor. Veja a Listagem 2.4.8.

Listagem 2.4.8

```
@Input() sombraEntrada: string;
```

- Para passar um valor a ser atribuído a essa variável, faça o ajuste destacado na Listagem 2.4.9.

Listagem 2.4.9

```
<input appSombraNaTabela sombraEntrada="10px 10px #CCC" type="text"
id="lembreteInputText" [(ngModel)]="lembrete"
       placeholder="Digite seu lembrete" class="form-control">


<table appSombraNaTabela sombraEntrada="10px 10px" class="table table-striped table-hover
mt-3">
```

- Quando a expressão a ser atribuída é uma expressão Typescript (como uma variável declarada na classe, por exemplo, usamos property binding (colchetes). Para ver isso, declare uma variável no componente e utilize seu valor a seguir. Veja a Listagem 2.4.10.

Listagem 2.4.10

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  lembretes: Array<string> = [];
  lembrete: string;

  variavelSombra: string = '10px 10px yellow';


  salvar() {
    this.lembretes = [this.lembrete, ...this.lembretes];
    this.lembrete = '';
  }
}
<input appSombraNaTabela [sombraEntrada]="variavelSombra" type="text" id="lembreteInputText"
      [(ngModel)]="lembrete" placeholder="Digite seu lembrete" class="form-control">

<table appSombraNaTabela [sombraEntrada]="variavelSombra" class="table table-striped table-hover mt-3">
```

- Lembra-se que é possível atribuir valores a nomes de diretivas (por exemplo [(ngModel)]=”nome”) ? Para viabilizar isso, basta alterar o nome da variável de entrada na diretiva. Isso pode ser feito nos parênteses da anotação @Input. Veja a Listagem 2.4.11. Faça a aplicação conforme destacado.

Listagem 2.4.11

```
@Input('appSombraNaTabela') sombraEntrada: string;

<input [appSombraNaTabela]="variavelSombra" type="text" id="lembreteInputText"
[(ngModel)]="lembrete"
      placeholder="Digite seu lembrete" class="form-control">

<table [appSombraNaTabela]="variavelSombra" class="table table-striped table-hover mt-3">
```

- Dizemos que os métodos que uma diretiva define constituem a sua API. O Angular possui um mecanismo que permite que diretivas **exportem a sua API**, ou seja, tornem seus métodos acessíveis a outros componentes que poderão chamá-los conforme desejarem. O primeiro passo é especificar um nome que ficará associado à propriedade **exportAs** pertencente aos metadados da diretiva. Veja a Listagem 2.4.12.

Listagem 2.4.12

```
@Directive({
  selector: '[appSombraNaTabela]',
  exportAs: 'sombraNoComponente'
})
```

- A seguir, declaramos uma variável de template para armazenar a API por meio de seu nome exportado. Veja a Listagem 2.4.13.

Listagem 2.4.13

```
<input [appSombraNaTabela]="variavelSombra" type="text" id="lembreteInputText"
[(ngModel)]="lembrete"
      placeholder="Digite seu lembrete" class="form-control"
#apiSombra="sombraNoComponente">
```

- A seguir, vamos criar dois botões para que possamos aplicar e remover as sombras como desejarmos, chamando os métodos da API da diretiva. Veja a Listagem 2.4.14.

Listagem 2.4.14

```
<div class="container">
  <div class="row">
    <div class="form-group col-12">
      <label for="lembreteInputText">Lembrete</label>
      <input [appSombraNaTabela]="variavelSombra" type="text" id="lembreteInputText" [(ngModel)]="lembrete"
        placeholder="Digite seu lembrete" class="form-control" #apiSombra="sombraNoComponente">
    </div>
  </div>
  <div class="row">
    <div class="col-12">
      <button type="button" class="btn btn-primary btn-block" (click)="salvar()">Salvar</button>
    </div>
  </div>
  <div class="row">
    <div class="col-12">
      <button type="button" class="btn btn-primary btn-block mt-3"
        (click)="apiSombra.quandoOMousePassarPorCima()">Aplicar sombra</button>
    </div>
  </div>

  <div class="row">
    <div class="col-12">
      <button type="button" class="btn btn-primary btn-block mt-3"
(click)="apiSombra.quandoOMouseSair()">Remover
        sombra</button>
    </div>
  </div>

  <div class="row">
    <table [appSombraNaTabela]="variavelSombra" class="table table-striped table-hover mt-3">
      <thead>
        <tr>
          <th class=" text-center">Lembretes</th>
        </tr>
      </thead>
      <tbody>
        <tr *ngFor="let lembrete of lembretes">
          <td class="text-center">{{lembrete}}</td>
        </tr>
      </tbody>
    </table>
  </div>
</div>
```

2.5 (Pipes) Um pipe é um tipo de componente que pode ser alimentado com algum tipo de dado e que produz algo que pode ser exibido na tela. Em geral, são usados para a formatação de valores numéricos, monetários, datas etc.

- Para testar alguns pipes conhecidos do Angular, crie um novo componente com o seguinte comando.

ng g c teste-pipes --skipTests=true

- A seguir, **comente o conteúdo de app.component.html** (você pode fazer isso selecionando tudo e apertando CTRL + /) e adicione a ele o conteúdo da Listagem 2.5.1, para que passemos a ver o conteúdo especificado pelo novo componente criado.

Listagem 2.5.1

```
<app-teste-pipes></app-teste-pipes>
```

- No arquivo **teste-pipes.component.ts**, declare as variáveis da Listagem 2.5.2.

Listagem 2.5.2

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

@Component({
  selector: 'app-teste-pipes',
  templateUrl: './teste-pipes.component.html',
  styleUrls: ['./teste-pipes.component.css']
})
export class TestePipesComponent {

  endereco: object = {
    rua: "Rua K",
    numero: 121,
    bairro: 'Vila J',
    cidade: 'Itu'
  }
  valorDoDolar: number = 5.05;
  data = new Date();
  inflacao: number = 0.03;
  valorDecimal: number = 0.55431;

  nome: string = "josé da silva";
}
```

- A seguir, usando o operador de interpolação, exiba o valor de cada variável. Estamos no arquivo **teste-pipes.component.html** agora. Veja a Listagem 2.5.3.

Listagem 2.5.3

```
<div class="container">
   <p>{{endereco}}</p>
   <p>{{valorDoDolar}}</p>
   <p>{{data}}</p>
   <p>{{inflacao}}</p>
   <p>{{valorDecimal}}</p>
   <p>{{nome}}</p>
</div>
```

- Note que a exibição deles pode não ser a desejada. Para especificar uma forma de exibição desejada, podemos usar um pipe. A forma é {{ expressao | pipe }}. Note a semelhança com o operador pipe dos terminais *nix. Veja os exemplos da Listagem 2.5.4.

NOTA: O Angular possui diversos pipes prontos para uso. Sua documentação pode ser encontrada no Link 2.5.1.

Link 2.5.1
https://angular.io/api?type=pipe

Listagem 2.5.4

```
<div class="container">
   <p>{{endereco | json}}</p>
   <p>{{valorDoDolar | currency}}</p>
   <p>{{data | date}}</p>
   <p>{{inflacao | percent}}</p>
   <p>{{valorDecimal | number}}</p>
   <p>{{nome | titlecase}}</p>
</div>
```

- Podemos também personalizar o funcionamento dos pipes. Veja os exemplos da Listagem 2.5.5.

Listagem 2.5.5

```
<div class="container">

   <p>{{data| date:'short'}}</p>
   <p>{{data| date:'full'}}</p>
   <p>{{data| date:'shortDate'}}</p>
   <p>{{data| date:'d'}}</p>
   <p>{{data| date:'dd'}}</p>
   <p>{{data| date:'dd/MM'}}</p>
   <p>{{data| date:'y'}}</p>
   <p>{{data| date:'yy'}}</p>
   <!--numero minimo de digitos para a parte inteira, numero minimo e maximo para a parte decimal-->
   <p>{{valorDecimal | number: '2.1-5'}}</p>
</div>
```

Referências

Angular. 2020. Disponível em <https://angular.io>. Acesso em maio de 2020.