

IMT - Instituto Mauá de Tecnologia

Engenharia de Computação

ECM252 – Linguagens de Programação II

Professor Rodrigo Bossini

Aula 09 - Angular - Two Way Data Binding - Diretivas - Diretivas estruturais

1 Introdução

Na aula anterior, lidamos com alguns tipos de Data Binding. Este é um mecanismo cujo funcionamento está descrito em detalhes na documentação oficial do Angular, que pode ser encontrada no Link 1.1.

Link 1.1

<https://angular.io/guide/glossary#data-binding>

Neste material, iremos ver a forma de Data Binding chamada **Two Way Data Binding**. Além disso, iremos abordar outros conceitos fundamentais do Angular, como o uso de diretivas personalizadas e diretivas estruturais, as quais são capazes de alterar a estrutura da árvore DOM.

2 Desenvolvimento

2.1 (Abrindo o projeto da aula passada) Abra um terminal e navegue até o diretório do projeto da aula passada. Use o comando a seguir para abrir uma instância do Visual Studio Code vinculada ao diretório do projeto.

code .

2.2 (Colocando o projeto em execução) Ainda no terminal, digite o seguinte comando para colocar uma instância do servidor de testes em execução e abrir uma aba do navegador padrão.

ng serve --open

2.3 (Two-way Data Binding) A ideia do Two-way data binding é muito simples. Um componente visual fica vinculado a uma variável. Quando a variável tem seu valor alterado, o componente visual exibe imediatamente o novo valor. E o inverso também é verdade. Quando o componente visual tem seu conteúdo alterado, a variável que está vinculada a ela tem seu valor alterado.

- Abra o arquivo app.component.js e declare uma nova variável chamada “produto, como na Listagem 2.3.1.

Listagem 2.3.1

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  nome;
  exibirCaixa = false;
  numero;
  produto;

  gerarNumero() {
    this.numero = Math.floor(Math.random() * 6) + 1;
  }

  alterarNome(nome) {
    console.log(nome.target.value);
    this.nome = nome.target.value;
  }
}
```

```
adicionar(nomeInput) {  
  this.nome = nomeInput.value;  
  this.exibirCaixa = true;  
}  
}
```

- Abra o arquivo **app.component.html** e adicione um novo campo para entrada de dados textual e um novo botão. Note o uso da diretiva **ngModel**. Ela caracteriza o uso do Two-way data binding. Veja a Listagem 2.3.2. Note que o conteúdo está **fora do form**. Estudaremos a razão para isso em breve.

Listagem 2.3.2

```
<div class="container">  
  <form>  
    <div class="form-group">  
      <label for="nomeInput">Nome</label>  
      <input type="nome" class="form-control" id="nomeInput" placeholder="Digite um nome"  
#nomeInput>  
    </div>  
    <button type="button" class="btn btn-primary btn-block"  
(click)="adicionar(nomeInput)">Adicionar</button>  
    <div class="alert alert-primary mt-3 p-3" [hidden]="!exibirCaixa">  
      {{ nome }}  
    </div>  
  </form>  
  <label for="produtoInput">Produto</label>  
  <input type="text" class="form-control" id="produtoInput" [(ngModel)]="produto"  
#produtoInput>  
  <button type="button" class="btn btn-primary btn-block"  
(click)="alterarProduto(produtoInput)">  
    Alterar produto  
  </button>  
</div>
```

- Implemente a função **alterarProduto** no arquivo **app.component.ts**. Ela se encarrega de alterar o valor da variável **produto**. Veja a Listagem 2.3.3.

Listagem 2.3.3

```
alterarProduto(produtolnput) {  
  this.produto = 'Novo produto: ' + produtolnput.value;  
}
```

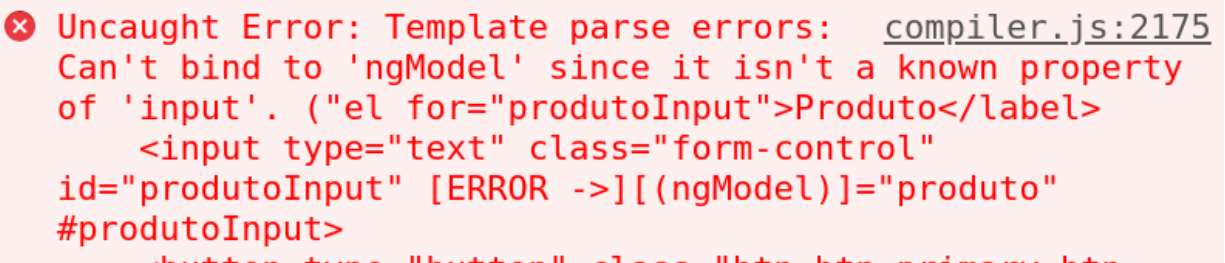
- A fim de verificar o valor da variável conforme ela é alterada, seja pelo botão, seja pela entrada de dados no campo textual, mostre seu conteúdo com o operador de interpolação, no arquivo **app.component.html**. Veja a Listagem 2.3.4. Coloque essa expressão abaixo do último botão, por exemplo.

Listagem 2.3.4

```
{{produto}}
```

- Neste momento, a aplicação deve falhar com o erro exibido pela Figura 2.3.1. Para ver o erro, basta abrir o Chrome Dev Tools (CTRL + SHIFT + I, por exemplo). A mensagem indica que a `ngModel` não é uma propriedade conhecida do componente `input`. Ocorre que a diretiva `ngModel` não pertence a nenhum módulo declarado ou definido pela aplicação. Ela faz parte de um módulo chamado **FormsModule** e, para utilizá-la, basta adicioná-lo ao arquivo **app.module.ts**, como mostra a Listagem 2.3.5.

Figura 2.3.1



```
✖ Uncaught Error: Template parse errors: compiler.js:2175  
Can't bind to 'ngModel' since it isn't a known property  
of 'input'. ("el for="produtoInput">Produto</label>  
  <input type="text" class="form-control"  
id="produtoInput" [(ngModel)]="produto"  
#produtoInput>  
  <button type="button" class="btn btn-primary btn-
```

Listagem 2.3.5

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

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

import { FormsModule } from '@angular/forms'
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

2.4 (Diretivas) Passamos ao estudo do mecanismo do Angular conhecido como **Diretiva**. Segundo a documentação oficial, que pode ser encontrada no Link 2.4.1, há três tipos de diretivas no Angular.

Link 2.4.1

<https://angular.io/guide/attribute-directives>

Os tipos de diretivas são:

- **Componentes:** diretivas com um template associado. São os componentes cuja criação já testamos.
- **Diretivas estruturais:** São diretivas que, quando aplicadas, podem alterar a estrutura da árvore DOM.
- **Diretivas de atributo:** são capazes de alterar a aparência ou comportamento de um elemento, componente ou de outra diretiva.

2.4.1 (Criando um novo projeto) A fim de testar as diretivas, crie um novo projeto com o comando a seguir. Certifique-se de executá-lo no seu workspace porém fora do diretório do projeto anterior.

ng new nome-do-projeto

Quando perguntado, não será necessário adicionar um módulo de roteamento. Além disso, escolha o formato CSS para as folhas de estilos.

2.4.2 (Abrindo o projeto) Navegue até o diretório do projeto e abra uma instância do VS Code vinculada a ele.

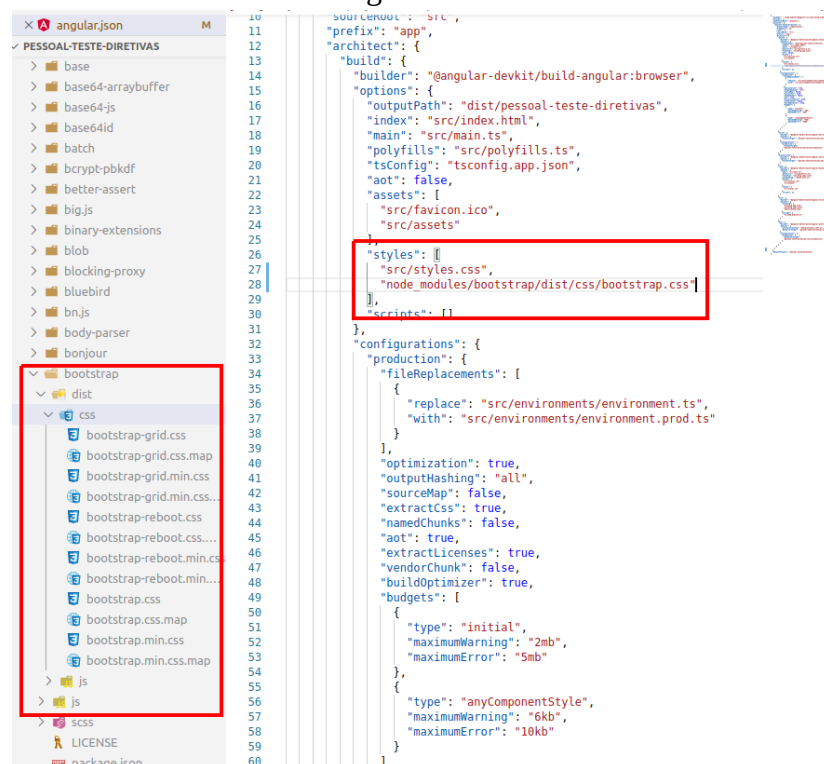
cd nome-do-projeto
code .

2.4.3 (Bootstrap) Vamos manter o uso do Bootstrap. Por isso, faça a sua instalação componente

npm install bootstrap@latest

- A seguir, ajuste o arquivo Angular.json para que seu atributo **projects.nome-do-projeto.architect.build.options.styles**, que está associado a um vetor, inclua também **node_modules/bootstrap/dist/css/bootstrap.css**. Veja a Figura 2.4.3.1.

Figura 2.4.3.1



2.4.4 (A diretiva estrutural ngIf) A diretiva estrutural ngIf, quando aplicada a algum elemento ou componente, é associada a um valor booleano. Ela garante que o elemento ou componente será exibido na tela somente nos casos em que a expressão associada for avaliada como verdadeira.

- Para testá-la, comece apagando todo o conteúdo do arquivo **app.component.html**. A seguir, inclua o conteúdo dado na Listagem 2.4.4.1. Temos uma tabela cuja exibição depende do clique no botão logo abaixo dela.

Listagem 2.4.4.1

```
<div class="container">
  <div class="row">
    <table *ngIf="!esconder" class="table table-striped table-hover">
      <thead>
        <tr>
          <td>Nome</td>
          <td>Idade</td>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>José</td>
          <td>18</td>
        </tr>
        <tr>
          <td>Maria</td>
          <td>22</td>
        </tr>
      </tbody>
    </table>
  </div>
  <div class="row">
    <button class="btn btn-primary btn-block mt-3" (click)=alterarExibicao()>{{textoBotao}}</button>
  </div>
</div>
```

- Note que o template usa variáveis do componente para decidir se a tabela será exibida ou não (a variável **esconder**) e para decidir qual texto exibir no botão (a variável **textoBotao**). Ele também usa um método que é chamado quando o botão é clicado. Veja as suas definições na Listagem 2.4.4.2. Elas são feitas no arquivo **app.component.ts**.

Listagem 2.4.4.2

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  textoBotao = "Esconder";
  esconder = false;

  alterarExibicao() {
    this.textoBotao = this.esconder ? "Exibir" : "Esconder";
    this.esconder = !this.esconder;
  }
}
```

- Clique no botão para testar o funcionamento da diretiva.

2.4.5 (A diretiva hidden) Também é possível deixar de exibir um elemento na página usando a diretiva de atributo **hidden**. Para usar uma diretiva de atributo, usamos o operador colchetes. Faça a alteração exibida pela Listagem 2.4.4.3 no arquivo **app.component.html**.

Listagem 2.4.4.3

```
<table [hidden]="esconder" class="table table-striped table-hover">
```

Nota: Embora visualmente possam ter o mesmo efeito, as diretiva **hidden** e **ngIf** tem uma diferença fundamental:

- A diretiva **hidden** somente decide sobre ocultar ou exibir um determinado elemento. Independente do valor lógico associado a ela, o elemento e seus descendentes permanecem na árvore DOM.
- A diretiva **ngIf**, por sua vez, quando associada a uma expressão booleana que resulta em **true**, **remove o elemento e todos os seus descendentes da árvore DOM**, o que pode significar economia de memória.
- Cabe ao desenvolvedor decidir se é necessário ou não manter a estrutura na árvore.

- Como teste, abra o Chrome Dev Tools e verifique o que ocorre com a tabela nos dois casos, usando a diretiva `ngIf` e a diretiva `hidden`.

Nota: Perceba que as diretivas estruturas, quando utilizadas, devem ser precedidas pelo símbolo `*`.

2.4.6 (A diretiva estrutural `ngFor`) A diretiva estrutural `ngFor` nos permite percorrer uma coleção de itens dando origem a um elemento DOM para cada um deles. Por exemplo, quando aplicada a um elemento **`div`**, ela irá gerar e adicionar um elemento `div` à árvore DOM para cada item que existir na coleção especificada em sua expressão.

- Para testar a diretiva estrutural `ngFor`, começamos declarando uma lista no arquivo **`app.component.ts`**, incluindo dois objetos que representam pessoas. Veja a Listagem 2.4.6.1.

Listagem 2.4.6.1

```
export class AppComponent {  
  textoBotao = "Esconder";  
  esconder = false;  
  
  pessoas = [  
    { nome: "José", idade: 18 },  
    { nome: "Maria", idade: 22 }  
  ];  
  
  alterarExibicao() {  
    this.textoBotao = this.esconder ? "Exibir" : "Esconder";  
    this.esconder = !this.esconder;  
  }  
}
```

- Desejamos que a tabela possua uma linha para cada objeto JSON existente na lista. Aplicaremos a diretiva `ngFor` para isso. Veja a Listagem 2.4.6.2.

Listagem 2.4.6.2

```
<table *ngIf="!esconder" class="table table-striped table-hover">
  <thead>
    <tr>
      <td>Nome</td>
      <td>Idade</td>
    </tr>
  </thead>
  <tbody>
    <tr *ngFor="let pessoa of pessoas">
      <td>{{pessoa.nome}}</td>
      <td>{{pessoa.idade}}</td>
    </tr>
  </tbody>
</table>
```

- A seguir, adicionamos uma nova linha ao container do Bootstrap. Ele terá campos para que o usuário possa inserir novas pessoas. Abaixo dele, em uma nova linha, um botão que, quando clicado, faz a inserção na lista. Veja a Listagem 2.4.6.3.

Listagem 2.4.6.3

```

<div class="container">
  <div class="row">
    <table *ngIf="!esconder" class="table table-striped table-hover">
      <thead>
        <tr>
          <td>Nome</td>
          <td>Idade</td>
        </tr>
      </thead>
      <tbody>
        <tr *ngFor="let pessoa of pessoas">
          <td>{{pessoa.nome}}</td>
          <td>{{pessoa.idade}}</td>
        </tr>
      </tbody>
    </table>
  </div>
  <div class="row">
    <div class="col-md-6 form-group">
      <label for="nomeInput">Nome</label>
      <input type="text" class="form-control" id="nomeInput" #nomeInput>
    </div>
    <div class="col-md-6 form-group mb-2">
      <label for="Idade">Idade</label>
      <input type="text" class="form-control" id="nomeInput" #idadeInput>
    </div>
  </div>
  <div class="row">
    <button class="btn btn-primary btn-block" (click)="adicionar(nomeInput.value,
idadeInput.value)"
type="button">Cadastrar</button>
  </div>
  <div class="row">
    <button class="btn btn-primary btn-block mt-3" (click)="alterarExibicao()>{{textoBotao}}</
button>
  </div>
</div>

```

- No arquivo **app.component.ts** implementamos o método adicionar. Ele recebe o nome e a idade, constrói um objeto JSON com essas propriedades e adiciona na lista. Veja a Listagem 2.4.6.4. Construímos uma nova lista que contém o novo objeto como primeiro elemento e, a seguir, usamos o operador **spread** (...) para extrair os elementos da lista antiga e adicioná-los à nova.

Listagem 2.4.6.4

```
adicionar(nome, idade) {  
  this.pessoas = [{ nome: nome, idade: idade }, ...this.pessoas];  
}
```

2.5 (Vínculo de propriedades personalizadas) O Angular nos permite criar componentes que recebem informações quaisquer por meio de propriedades. Por exemplo, poderíamos ter um componente cuja finalidade é exibir os dados de uma determinada pessoa. Um outro componente que o utiliza dentro de um ngFor pode lhe entregar uma pessoa via property binding.

- Para testar essa possibilidade, passaremos a exibir as pessoas como cartões do Bootstrap. Para começar, ajuste o conteúdo do arquivo **app.component.html** como mostra a Listagem 2.5.1. Note que não temos mais a tabela nem o botão para escondê-la.

Listagem 2.5.1

```
<div class="container">  
  <div class="row">  
    <div class="col-md-6 form-group">  
      <label for="nomeInput">Nome</label>  
      <input type="text" class="form-control" id="nomeInput" #nomeInput>  
    </div>  
    <div class="col-md-6 form-group mb-2">  
      <label for="Idade">Idade</label>  
      <input type="text" class="form-control" id="nomeInput" #idadeInput>  
    </div>  
  </div>  
  <div class="row">  
    <button class="btn btn-primary btn-block" (click)="adicionar(nomeInput.value,  
idadeInput.value)"  
      type="button">Cadastrar</button>  
  </div>  
</div>
```

- A seguir, adicione uma linha do Bootstrap logo abaixo do botão que adiciona os elementos. Ela terá um cartão do Bootstrap. Veja a Listagem 2.5.2.

Listagem 2.5.2

```
<div class="container">
  <div class="row">
    <div class="col-md-6 form-group">
      <label for="nomeInput">Nome</label>
      <input type="text" class="form-control" id="nomeInput" #nomeInput>
    </div>
    <div class="col-md-6 form-group mb-2">
      <label for="Idade">Idade</label>
      <input type="text" class="form-control" id="nomeInput" #idadeInput>
    </div>
  </div>
  <div class="row">
    <button class="btn btn-primary btn-block" (click)="adicionar(nomeInput.value,
idadeInput.value)"
      type="button">Cadastrar</button>
  </div>
  <div class="row">
    <div class="card col-4 p-0 m-2">
      <div class="card-header">
        Pessoa
      </div>
      <div class="card-body">
        <p class="card-text">
          <p>Nome: Nome da Pessoa</p>
          <p>Idade: Idade da pessoa</p>
        </div>
      </div>
    </div>
  </div>
```

- Note que desejamos gerar um cartão para cada pessoa existente na lista do componente. Para isso, basta adicionar uma diretiva ngFor e usar o operador de interpolação para pegar as propriedades, como feito anteriormente. Veja a Listagem 2.5.3.

Listagem 2.5.3

```
<div class="row">
  <div class="card col-4 p-0 m-2" *ngFor="let pessoa of pessoas">
    <div class="card-header">
      Pessoa
    </div>
    <div class="card-body">
      <p class="card-text">
        <p>Nome: {{pessoa.nome}}</p>
        <p>Idade: {{pessoa.idade}}</p>
      </div>
    </div>
  </div>
```

- A exibição de pessoas dessa forma pode ser de interesse em diversas partes da aplicação. Ao invés de copiar e colar a definição do cartão sempre que necessário, podemos definir esse trecho como um componente à parte e utilizá-lo quando preciso. Assim, use o comando a seguir para gerar um componente cuja finalidade será exibir uma única pessoa em um cartão. Note que **g** e **c** são atalhos para **generate** e **component**, respectivamente. A opção **skip-tests** instrui o Angular CLI a não criar um arquivo com código apropriado para teste, já que não faremos testes no momento.

```
ng g c pessoa-cartao --skip-tests
```

- A seguir, recorte a definição do cartão do arquivo **app.component.html** e cole ela no arquivo **pessoa-cartao.component.html**. Veja a Listagem 2.5.4. Note que **removemos a diretiva ngFor**, já que o novo componente não tem acesso à coleção de pessoas. Remova também a classe **col-4**.

Listagem 2.5.4

```
<div class="card p-0 m-2">
  <div class="card-header">
    Pessoa
  </div>
  <div class="card-body">
    <p class="card-text">
      <p>Nome: {{pessoa.nome}}</p>
      <p>Idade: {{pessoa.idade}}</p>
    </div>
  </div>
```

- Para utilizar o componente que acabou de ser criado, basta escrever o seu seletor como se fosse uma tag HTML. Assim, abra o arquivo **app.component.html** e faça o ajuste da Listagem 2.5.5. Colocamos o componente dentro de uma div sobre a qual aplicaremos a diretiva ngFor a seguir.

Listagem 2.5.5

```
<div class="container">
  <div class="row">
    <div class="col-md-6 form-group">
      <label for="nomeInput">Nome</label>
      <input type="text" class="form-control" id="nomeInput" #nomeInput>
    </div>
    <div class="col-md-6 form-group mb-2">
      <label for="Idade">Idade</label>
      <input type="text" class="form-control" id="nomeInput" #idadeInput>
    </div>
  </div>
  <div class="row">
    <button class="btn btn-primary btn-block" (click)="adicionar(nomeInput.value,
idadeInput.value)"
      type="button">Cadastrar</button>
  </div>
  <div class="row">
    <div class="col-4">
      <app-pessoa-cartao></app-pessoa-cartao>
    </div>
  </div>
</div>
```

- Note que há um problema com o componente pessoa-cartao: ele tenta aplicar o operador de interpolação sobre um objeto pessoa que não existe. Esse é um objeto que faz parte da coleção do componente principal da aplicação. Neste momento, precisamos utilizar um mecanismo que nos permita fazer com que o componente principal entregue ao componente pessoa-cartao o objeto JSON que representa a pessoa que ele deseja exibir.

- O primeiro passo é adicionar uma propriedade ao componente pessoa-cartao e anotá-la como @Input(), indicando que seu valor é uma “entrada” do componente, algo que ele recebe quando é utilizado. Veja a Listagem 2.5.6. Estamos no arquivo **pessoa-cartao.component.ts**.

Listagem 2.5.6

```
import { Component, OnInit, Input } from '@angular/core';  
@Input() pessoa;
```

- A seguir, no arquivo **app.component.html**, aplique a diretiva `ngFor` à `div` que engloba o componente `pessoa-cartao`. Use `property binding` para entregar o objeto `pessoa` a ele, a cada iteração. Veja a Listagem 2.5.7.

Listagem 2.5.7

```
<div class="row">  
  <div class="col-4" *ngFor="let pessoa of pessoas">  
    <app-pessoa-cartao [pessoa]="pessoa"></app-pessoa-cartao>  
  </div>  
</div>
```

- Neste instante já possível adicionar novas pessoas. A cada pessoa adicionada, um novo cartão deve aparecer na tela contendo os seus dados.
- Um componente filho pode emitir eventos de interesse para o componente que o utiliza. Por exemplo, um componente que tem um botão pode emitir um evento do tipo “click” e este evento pode ser de interesse para o componente que o utiliza. Assim como componentes podem receber dados com `@Input ()`, eles também podem entregar eventos (com dados associados, possivelmente) com `@Output ()`.
- Para ilustrar esse recurso, vamos criar um novo componente que terá como template os campos que usuário utiliza para inserir dados e o botão. O comando para criá-lo é o seguinte

ng g c pessoa-cadastro --skip-tests

- A seguir, recorte o conteúdo da primeira row do arquivo **app.component.html** e cole ele no arquivo **pessoa-cadastro.component.html**. Veja a Listagem 2.5.8.

Listagem 2.5.8

```
<div class="row">
  <div class="col-md-6 form-group">
    <label for="nomeInput">Nome</label>
    <input type="text" class="form-control" id="nomeInput" #nomeInput>
  </div>
  <div class="col-md-6 form-group mb-2">
    <label for="Idade">Idade</label>
    <input type="text" class="form-control" id="nomeInput" #idadeInput>
  </div>
</div>
<div class="row">
  <button class="btn btn-primary btn-block" (click)="adicionar(nomeInput.value,
idadeInput.value)"
    type="button">Cadastrar
</button>
</div>
```

- A seguir, podemos usar o novo componente no arquivo **app.component.html** por meio de seu seletor. Veja a Listagem 2.5.9.

Listagem 2.5.9

```
<div class="container">
  <app-pessoa-cadastro></app-pessoa-cadastro>
  <div class="row">
    <div class="col-4" *ngFor="let pessoa of pessoas">
      <app-pessoa-cartao [pessoa]="pessoa"></app-pessoa-cartao>
    </div>
  </div>
</div>
```

- O componente **pessoa-cadastro** tenta fazer uso do método adicionar que, até então, está definido no componente principal da aplicação. Por isso, recorte o método do componente principal e cole ele no componente pessoa-cadastro. Leve junto também a coleção de pessoas, já que o método adicionar a utiliza. Veja a Listagem 2.5.10.

Listagem 2.5.10

```
@Component({
  selector: 'app-pessoa-cadastro',
  templateUrl: './pessoa-cadastro.component.html',
  styleUrls: ['./pessoa-cadastro.component.css']
})
export class PessoaCadastroComponent {

  pessoas = [
    { nome: "José", idade: 18 },
    { nome: "Maria", idade: 22 }
  ];
  adicionar(nome, idade) {
    this.pessoas = [{ nome: nome, idade: idade }, ...this.pessoas];
  }
}
```

- Teste a aplicação agora. Note que a aplicação não exibe os cartões. Isso ocorre pois a diretiva ngFor está tentando acessar uma lista que pertenceria ao componente principal, porém ela agora faz parte do componente pessoa-cadastro.
- Transporte a lista de volta para o componente principal. Veja, porém, que isto também não funciona já que o método adicionar do componente pessoa-cadastro não terá onde adicionar as pessoas conforme o botão cadastrar for clicado.
- Quando um evento de clique no botão ocorre, desejamos que o componente pessoa-cadastro informe ao componente principal sobre isso, para que ele possa se atualizar. Isso pode ser feito por meio de uma **emissão de eventos** que, no Angular, é realizada por meio de um **EventEmitter**.
- Assim, no componente pessoa-cadastro, declare uma variável chamada pessoaAdicionada e faça com que ela referencie uma instância de EventEmitter. Importe EventEmitter do pacote core do Angular. Veja a Listagem 2.5.11.

Listagem 2.5.11

```
import { Component, OnInit, EventEmitter } from '@angular/core';  
pessoaAdicionada = new EventEmitter();
```

- Depois disso, o nome usado para a variável (pessoaAdicionada, neste caso) se torna o nome de um evento que outros componentes podem “escutar”. Sendo assim, no componente principal, no arquivo **app.component.html**, aplique um eventBinding usando o novo evento. Veja a Listagem 2.5.12. Note o uso do método onAdicionarPessoa. Ele não existe e será criado em breve.

Listagem 2.5.12

```
<div class="container">  
  <app-pessoa-cadastro (pessoaAdicionada)="onAdicionarPessoa()"></app-pessoa-cadastro>  
  <div class="row">  
    <div class="col-4" *ngFor="let pessoa of pessoas">  
      <app-pessoa-cartao [pessoa]="pessoa"></app-pessoa-cartao>  
    </div>  
  </div>  
</div>
```

- Para que o vínculo de evento possa ocorrer, aplique a anotação @Output à variável pessoaAdicionada, do componente **pessoa-cadastro**. Veja a Listagem 2.5.13.

Listagem 2.5.13

```
import { Component, OnInit, EventEmitter, Output } from '@angular/core';  
@Output() pessoaAdicionada = new EventEmitter();
```

- Ainda no componente pessoa-cadastro, vamos ajustar seu método **adicionar** para que um evento seja emitido quando o botão for clicado. Veja a Listagem 2.5.13. Note que quando a emissão de evento acontece, o dado associado ao evento (a pessoa adicionada, neste caso) é enviado junto para quem tiver interesse no evento.

Listagem 2.5.13

```
@Component({
  selector: 'app-pessoa-cadastro',
  templateUrl: './pessoa-cadastro.component.html',
  styleUrls: ['./pessoa-cadastro.component.css']
})
export class PessoaCadastroComponent {

  @Output() pessoaAdicionada = new EventEmitter();

  adicionar(nome, idade) {
    const pessoa = {
      nome: nome,
      idade: idade
    };
    this.pessoaAdicionada.emit(pessoa);
  }
}
```

- De volta ao arquivo **app.component.html**, precisamos de alguma forma fazer com que o método **onAdicionarPessoa** tenha acesso à pessoa enviada junto com o evento. Para isso, vamos usar o nome implícito **\$event**. Veja a Listagem 2.5.14. O **\$event** é justamente o que foi passado pelo método **emit**. Neste caso, o objeto **pessoa**.

Listagem 2.5.14

```
<div class="container">
  <app-pessoa-cadastro (pessoaAdicionada)="onAdicionarPessoa($event)"></app-pessoa-cadastro>
  <div class="row">
    <div class="col-4" *ngFor="let pessoa of pessoas">
      <app-pessoa-cartao [pessoa]="pessoa"></app-pessoa-cartao>
    </div>
  </div>
</div>
```

- Resta implementar o método **onAdicionarPessoa**, o que deve ser feito no arquivo **app.component.ts**. Ele recebe uma pessoa e a adiciona na lista, como fazia antes. Veja a Listagem 2.5.15.

Listagem 2.5.15

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {

  pessoas = [
    { nome: "José", idade: 18 },
    { nome: "Maria", idade: 22 }
  ];

  onAdicionarPessoa(pessoa) {
    this.pessoas = [pessoa, ...this.pessoas];
  }

}
```

Referências

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