Instituto Federal de Educação, Ciência e Tecnologia do RN
Campus Parnamirim

Prof. Dr. Bruno Emerson Gurgel Gomes (bruno.gurgel@ifrn.edu.br)

2 - Comunicação *template* componente

última atualização em 15/04/2018

Objetivos

- Apresentar os elementos principais de interação entre *template* e componente;
- Criar uma pequena aplicação para explorar os conceitos aprendidos.

Pré-requisitos

- Saber criar uma aplicação e um componente usando o **@angular/cli**

1 Componentes, Templates e Serviços

No capítulo anterior vimos que uma aplicação Angular é composta basicamente por módulos e componentes dentro de módulos. Ao criar uma nova aplicação com o **`@angular/cli`** criamos ao menos um módulo, denominado **`AppModule`** (src/app/**app.module.ts**) e um componente associado a ele, denominado de **`AppComponent`**. Esse último é por padrão o componente que é iniciado ao se rodar a aplicação sendo composto basicamente por três arquivos, um *TypeScript* (src/app/**app.component.ts**), um HTML (src/app/**app.component.html**) e um CSS (src/app/**app.component.css**).

Em um componente, o *template* é composto pelos arquivos HTML e CSS e representa as visões, ou seja, as telas ou parte das telas da aplicação. O *template* é responsável por obter informações do usuário por meio de formulários e repassar ao componente, bem como por realizar ações a partir de informações obtidas de atributos ou métodos definidos no componente. Embora o componente seja o conjunto dos html, css e *TypeScript*, neste curso vamos usar a palavra "componente" para se referir ao arquivo *TypeScript* do componente e "template" para a parte HTML e CSS.

É importante observar que para uma melhor organização e possibilidade de reúso de código o componente deve fornecer a apenas a interface básica das informações a ser acessadas ou modificadas pelo *template*. Caso seja necessário a produção de determinado conjunto de informações, por exemplo, uma consulta a uma base de dados ou a um serviço *Web*, essas ações devem ser inseridas em outras classes e essas por sua vez importadas e utilizadas no componente. Isto por ser feito tanto em uma ou mais classes *TypeScript* padrão, como em uma ou mais classes de "serviço". Os serviços serão explorados mais adiante neste curso.

Por enquanto, vamos nos concentrar nos elementos de comunicação e interação entre *templates* e componentes e demonstrar o uso da interpolação, eventos e ligação de propriedades (*property binding*).

2 Interpolação

Interpolação é uma forma de "imprimir" o valor de uma expressão *TypeScript* no corpo de uma *tag* ou no valor de um atributo de *tag* do *template*. O valor da expressão é avaliado e o resultado é devolvido como uma *string*. A expressão que se quer calcular e imprimir deve ser inserida entre a marcação **{{ }}** (*abre duas chaves – fecha duas chaves*). No exemplo do capítulo

anterior, utilizamos interpolação para imprimir uma mensagem de boas vindas ao usuário a partir da propriedade (atributo) "mensagem" construída no componente **ExibirMensagemComponent.** No caso, a mensagem é inicialmente vazia e o método **alterarMensagem** cria o texto de mensagem e atualiza a propriedade correspondente depois que o usuário digita o seu nome no formulário.

```
<form>
<label for="nome">Digite seu nome</label>
<input id="nome" name="nome" type="text" #nome (change)="alterarMensagem(nome.value)">
</form>

<div class="msg">
  <p>{{mensagem}}</p>
</div>
```

Código 1: Template do componente ExibirMensagemComponent. Fonte: autoria própria.

Neste mesmo projeto crie um novo componente chamado de **cursos** para que possamos testar mais alguns exemplos. Na linha de comando, entre no diretório do projeto **primeiro-app** e digite o comando:

```
ng g c cursos
```

Agora abra no Visual Studio Code (ou no editor que preferir) o componente **CursosComponent** (src/app**/cursos/cursos.component.ts**) e veja qual o seletor para o componente. Por padrão, o *Angular* coloca a palavra *app* antes do nome do componente. Assim, o seletor deve ser algo como **app-cursos**.

Pois bem, agora insira o seletor **app-cursos** no *template* do componente principal (src/app/**app.component.html)**. Se preferir comente o seletor do componente exibir mensagem para que ele não fique aparecendo.

```
<!--<app-exibir-mensagem></app-exibir-mensagem>-->
<app-cursos></app-cursos>
```

Código 2: Modificação no template do componente principal app.component.html. Fonte: autoria própria.

Na linha de comando do *VS Code*, abra uma outra aba de terminal e vá para o diretório da aplicação (**primeiro-app**), caso ainda não esteja nele, e rode a aplicação com **ng serve**. Abra o navegador na URL http://localhost:4200 e veja se está tudo funcionando corretamente. Deve aparecer o texto "App-cursos works!" na tela do navegador. Esse texto é colocado por padrão no *template* ao criar um novo componente. Para testar alguns casos de interpolação, abra o componente de cursos (src/app**/cursos/cursos.component.ts**) e altere-o conforme o Código 3 (Parte do código foi omitida, apenas a classe aparece).

```
export class CursosComponent implements OnInit {
  cursos: string[];
  quantCursos: number;
  faviconURL: string;
  constructor() {
    this.cursos = ['Integrado em Informática', 'Integrado em Mecatrônica',
                   'Redes Subsequente', 'Mecatrônica Subsequente',
                   'Superior em Sistemas para Internet'];
    this.quantCursos = this.cursos.length;
    this.faviconURL = '../../favicon.ico';
  }
}
```

Código 3: Parte do componente CursosComponent. Fonte: autoria própria.

O componente define uma lista de **cursos** e a inicializa com 5 (cinco) nomes de cursos do IFRN Parnamirim, uma propriedade **quantCursos** que armazena o tamanho dessa lista (a quantidade de cursos) e um atributo **faviconURL** contendo o caminho para o *favicon* da aplicação que encontra-se no diretório *src*.

O Código 4 por sua vez exibe o *template* **cursos.componente.html** com alguns exemplos de uso de interpolação sobre os atributos definidos no componente. No primeiro parágrafo do Código 4 é criada apenas uma expressão simples de soma. Já no segundo, há uma referência à propriedade **quantCursos**. Alguns cálculos são realizados e a mensagem completa é impressa na tela. No caso do terceiro parágrafo é impressa a própria lista de cursos. Mais adiante veremos como organizar melhor essa última exibição usando tabelas ou listas. Esse caso trata-se de um uso comum de interpolação quando o que se quer é apenas ter uma ideia do valor armazenado em um atributo ou retornado por um método.

```
<p>
  A soma de 1 + 1 é {{1 + 1}}
</p>
<p>
  O IFRN Parnamirim possui {{quantCursos-1}} técnicos. Sendo
  {{(quantCursos-1) / 2}} de nível médio integrado.
</p>
<p>{{cursos}}</p>
<p>
  <img src="{{faviconURL}}" alt="favicon">
</p>
```

Código 4: Exemplos de interpolação.

A impressão do resultado da interpolação também pode ser feita em um atributo de *tag*, como pode ser visto no último parágrafo do exemplo do Código 5. Nesse caso a interpolação obtém o valor do atributo **faviconURL** que contém o caminho para o *favicon* da aplicação (**../../favicon.ico**) Os **../../** indicam que o arquivo está duas pastas acima, ou seja, na pasta **src**. O resultado na tela é exibido na Figura 1. Este último exemplo no fim das contas é traduzido em outra forma de comunicação, denominada de *property binding* (em português livre, ligação de propriedade). Desse modo, para esses casos de associação de um valor a uma propriedade HTML é uma melhor prática usar o *property binding* como faremos na Seção 3 a seguir.

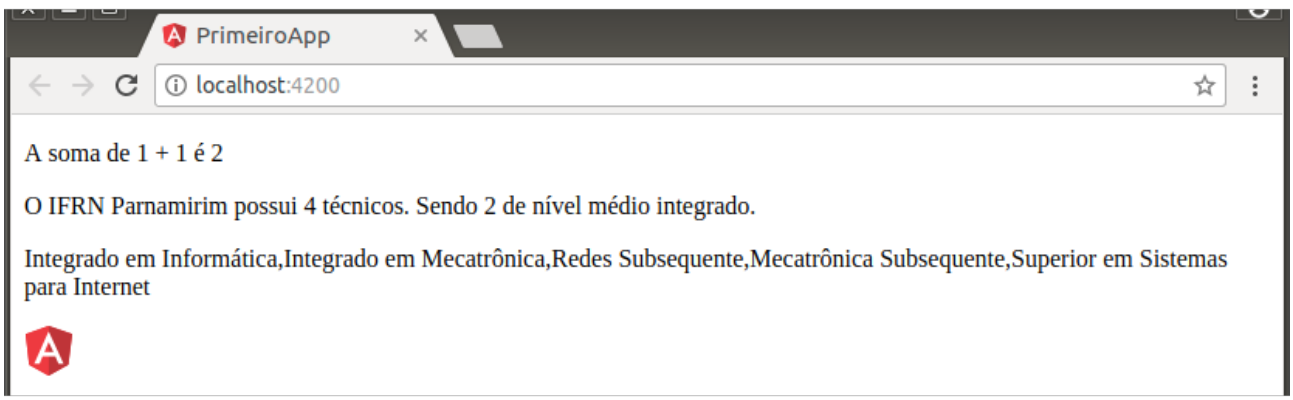

Figura 1: Resultado das expressões de interpolação no template cursos.component.html. Fonte: autoria própria.

3 Property binding

O uso de *property binding* permite modificar o valor de uma propriedade em uma *tag* do *template* a partir do resultado de uma expressão aplicada a ela. Ao contrário da *interpolação*, essa ligação não se dá apenas com um resultado do tipo *string*, podendo ter como resultado qualquer tipo conforme esperado pela propriedade no *template* (*string*, *number*, *boolean*, objeto, etc.).

Um uso bastante comum é relacionar uma propriedade do *template* a uma propriedade (atributo) do componente. O exemplo que vimos anteriormente da inserção da imagem (*favicon*) poderia ser feito com *property binding* na propriedade "src" como mostra o Código 5.

```
<p>
  <img [src]="faviconURL" alt="favicon">
</p>
```

Código 5: *property binding* em src para o valor da propriedade *faviconURL* de *CursosComponent*.

O *property binding* é feito com a notação **[]** (abre e fecha colchetes) ao redor da propriedade, no caso **src**. Assim, o valor do atributo **faviconURL** é usando para inicializar o valor de **src**. Esse tipo de ligação *template-componente* vai apenas na direção do componente para o *template*. Uma propriedade no *template* pode então receber um valor para que alguma ação seja tomada, como exibir uma imagem, habilitar ou desabilitar um botão, esconder ou exibir uma determinada *tag*, aplicar um *CSS* de acordo com uma condição, inserir um valor em uma caixa de texto de formulário, dentre outas possibilidades. Porém, não pode ser usado para exportar valores para fora do *template*. Para esse caso, há outras formas interação que veremos mais adiante, como os eventos (*event binding*) e o *two way data binding*.

3.1 Binding de atributo, estilo e classe

Casos especiais de ligação (*binding*) são os de atributo, estilo e classe. No primeiro caso, usa-se a notação **attr.nomedoatributo**. Por exemplo, na tabela do Código 6 a seguir, o atributo **colspan** da célula de título da tabela (**th**) é modificado para que essa célula ocupe o espaço das células adjacentes de acordo com a quantidade de elementos a ser mostrado (o valor de **quantCursos**). No caso, **colspan** não é uma propriedade de **th** e sim um atributo, por isso a necessidade da notação **attr.colspan**. Há outras situações nos quais os atributos podem ser explorados, por exemplo, os atributos *checked* (em elementos *radio* e *checkbox*) e *selected* (em uma *option* de um *select*).

```
<table>
<tr>
   <th [attr.colspan]="quantCursos">Cursos Técnicos e Superiores do IFRN Campus
Parnamirim</th>
</tr>
<tr>
  <td>{{cursos[0]}}<td>
  <td>{{cursos[1]}}<td>
  <td>{{cursos[2]}}<td>
  <td>{{cursos[3]}}<td>
  <td>{{cursos[4]}}<td>
</tr>
</table>
```

Código 6: ligação de atributo no atributo colspan (attr.colspan). Fonte: Adaptado de (ANGULAR, 2018).

A ligação de classes (*class binding*) é útil para aplicar uma classe de estilo usando o atributo *class* de um elemento *HTML*. A vantagem aqui é ser possível controlar a aplicação do estilo de acordo com uma condição. Por exemplo, defina a classe CSS **`.cursos`** do Código 7 no arquivo *.CSS* do componente **`CursosComponent`** (`src/app/`**`cursos.component.css`**).

```
.cursos {
  font-size: x-large;
  border: 1px solid #000;
  border-collapse: collapse;
}

.cursos tr, td {
  border: 1px solid #000;
}
```

Código 7: classes *CSS* inserida no arquivo cursos.component.css. Fonte: autoria própria.

Vamos aplicar essa classe à tabela caso a lista de cursos não seja vazia (**`quantCursos > 0`**). Para tanto, modifique a *tag table* de acordo com o Código 8.

```
<table [class.cursos]="quantCursos > 0">
```

Código 8: class binding aplicada à tabela caso se tenham cursos. Fonte: autoria própria.

No caso do Código 8 a notação **`class.nome_da_classe`** permite que essa classe seja aplicada somente se o resultado da sua condição seja verdadeiro. A expressão **`quantCursos > 0`** pode devolver verdadeiro ou falso. Nesse caso, como há cursos definidos na lista, o resultado é verdadeiro e a classe cursos (**`class.cursos`**) é aplicada. Para casos em que mais de uma classe necessite ser aplicada é possível fazer um "class.nome_da_classe" para cada condição ou usar uma diretiva denominada de **`ngClass`**. Demonstraremos o uso de **`ngClass`** no próximo capítulo no qual abordamos as diretivas.

O *style binding* serve para ligar estilos diretamente a uma *tag* (de modo *inline*). Esse tipo de emprego de estilo é adequado apenas para pequenas mudanças, por exemplo, quando se quer sobrescrever um estilo herdado de uma folha de estilos externa. No caso geral, é preferível usar *class binding* ou a diretiva **`ngClass`** para inserir classes de estilos. É possível também usar uma expressão condicional para aplicar ou não o estilo assim como foi feito para o *class binding*.

Para testar o *style binding* defina uma propriedade **`tipoCurso`** no componente e a inicialize no construtor com o valor 'integrado' conforme exemplificado no código 9.

```
export class CursosComponent implements OnInit {
  //...
  tipoCurso: string;
  constructor() {
    //...
    this.tipoCurso = 'integrado';
  }
  //...
}
```

Código 9. Parte do código do componente CursosComponent mostrando a definição e inicialização do atributo tipoCurso.

O Código 10 exibe o cabeçalho da tabela alterado para inserir uma cor de fundo (*backgroundColor*) na tabela caso o valor de **`tipoCurso`** seja 'integrado' e uma outra cor caso não

seja. Nesse caso é pois inserimos manualmente!. Daqui a pouco veremos como automatizar a escolha do tipo do curso usando um formulário com botões de rádio. Observe que no Código 10 utilizamos o operador de condicional **`exp ? exp_true : exp_false`**. Nesse operador o valor da expressão do lado esquerdo *exp* é avaliado. Caso seja verdadeiro (true) a expressão após a **`?`** (interrogação) é executada. Caso contrário, a expressão após o **`:`** (dois pontos) é feita.

```
<table [class.cursos]="quantCursos > 0"
       [style.backgroundColor]="tipoCurso == 'integrado' ? '#0fb' : '#f00'">
```

Código 10: *style binding* aplicando uma cor de fundo na tabela de acordo com o tipo de curso. Fonte: autoria própria.

Assim como há a diretiva **`ngClass`** como uma forma mais prática de se inserir mais de uma classe também há a diretiva **`ngStyle`** que é abordada posteriormente no capítulo sobre diretivas.

4 Event Binding

Revisando o que foi visto até agora, a interpolação (**`{{expressão}}`**) é uma forma de realizar um cálculo de expressão e imprimir o resultado desse cálculo em uma *tag* ou propriedade de *tag html*. Por sua vez, as ligações (*binding*) de propriedade (*property*, **`[propriedade]`**) associam um valor de um atributo definido na classe do componente com uma propriedade no HTML. Assim, é possível obter esse valor para modificar a propriedade *HTML* de algum modo. De forma similar, há as ligações de atributo (*attribute*), estilo (*style*) e classe (*class*) que também podem usar valores definidos no componente.

Em todas as formas de ligação que vimos até agora a informação é passada da classe do componente para o *template HTML*. Dá para perceber que isso não é suficiente para uma página *Web* na qual o usuário tem diversas possibilidades de interação ao clicar em um botão, ao digitar em um campo *input* de formulário e enviar esse valor ao componente, ao passar o mouse por cima de um elemento, etc. Assim, o *event binding* (ligação por meio de eventos) é um meio utilizado para enviar uma informação em sentido inverso, ou seja, do *template* para o componente a partir de uma ação do usuário (evento) e tem a forma **`()`** abre e fecha parênteses circulando um nome de evento.

No exemplo do Código 11, há um evento de clique (*click*) associado a uma *tag button* de um formulário que ao ser disparado (o botão é clicado) chama o método **`onSave()`** no componente correspondente.

```
<button (click)="onSave()">Save</button>
```

Código 11: evento de clique associado a um botão.

Vamos alterar um pouco o *template* do componente **`Cursos`**. Se preferir, comente os códigos dos parágrafos relacionados com a interpolação e deixe apenas o formulário que iremos inserir e a tabela, da forma como mostrado no Código 12.

```
<form>
    <button type="submit" (click)="mostrarCursos()">Mostrar cursos</button>
</form>

<table [class.cursos]="quantCursos > 0"
       [style.backgroundColor]="tipoCurso == 'integrado' ? '#0fb' : '#f00'"
       [hidden]="esconderCursos">
<tr>
   <th [attr.colspan]="quantCursos" [style.color]="'#00f'">
      Cursos Técnicos e Superiores do IFRN Campus Parnamirim
   </th>
</tr>
<tr>
  <td>{{cursos[0]}}<td>
  <td>{{cursos[1]}}<td>
  <td>{{cursos[2]}}<td>
  <td>{{cursos[3]}}<td>
  <td>{{cursos[4]}}<td>
</tr>
</table>
```

Código 12: Inserção de formulário com botão para mostrar ou esconder cursos. Fonte: autoria própria.

Como estamos trabalhando com formulário, verifique se o módulo **FormsModule** foi adicionado ao módulo principal **AppModule** (src/app/**app.module.ts**). Esse módulo é necessário ao trabalhar com formulários, principalmente ao se associar os controles do formulário às propriedades dos componente, o que faremos mais adiante. A não inserção dele pode acarretar erros na aplicação como o *template* em que o formulário está inserido ficar em *loop*, chamando a si mesmo de forma indefinida. Desse modo, caso não tenha sido adicionado, adicione em **AppModule** o **import** para o **FormsModule** e nome do módulo **FormsModule** na declaração *imports* do decorator **@NgModule**. O módulo principal deve estar da forma como aparece no Código 13.

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

import { AppComponent } from './app.component';
import { ExibirMensagemComponent } from './exibir-mensagem/exibir-mensagem.component';
import { CursosComponent } from './cursos/cursos.component';

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

Código 13: Módulo AppModule com FormsModule e o componente IntepolaçãoComponent inserido. Fonte: autoria própria.

O objetivo das alterações no *template* mostradas no Código 13 é exibir ou esconder a tabela de cursos a partir do clique no botão feito pelo usuário. Como pode ser visto no formulário do Código 13, isso é feito associando o evento de *click* no botão com uma chamada a um método **mostrarCursos()** que deve ser definido no componente ((click)="mostrarCursos()"). O método **mostrarCursos()** irá alterar o valor de um atributo *boolean* **esconderCursos** que por sua vez é usado na tabela em um *property binding* na propriedade *hidden*

([hidden]="esconderCursos") para esconder (**hidden="true"**) ou mostrar (**hidden="false"**) a tabela de cursos. A implementação completa do componente ficará como exibido no código 14.

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'app-cursos', templateUrl: './cursos.component.html',
  styleUrls: ['./cursos.component.css']
})
export class CursosComponent implements OnInit {
  cursos: string[];
  quantCursos: number;
  faviconURL: string;
  tipoCurso: string;
  esconderCursos: boolean;

  constructor() {
    this.cursos = ['Integrado em Informática', 'Integrado em Mecatrônica',
'Redes Subsequente', 'Mecatrônica Subsequente', 'Superior em Sistemas para Internet'];
    this.quantCursos = this.cursos.length;
    this.faviconURL = '../../favicon.ico';
    this.tipoCurso = 'integrado';
    this.esconderCursos = true;
  }

  ngOnInit() { }

  mostrarCursos() {
    this.esconderCursos = !this.esconderCursos;
  }
}
```

Código 14: Implementação completa de CursosComponent. Fonte: autoria própria.

Nessa implementação, **esconderCursos** é inicializado com **true** para que ao iniciar o componente a tabela não seja exibida. Sempre que o usuário clicar no botão o método **mostrarCursos** é chamado alterando o valor verdade do atributo **esconderCursos**. Assim, o método o valor associado a propriedade **hidden** na tabela também muda. Sempre que **esconderCursos** estiver em verdadeiro (*true*) **hidden** receberá esse valor verdade e a tabela será ocultada. Ao se mudar para falso, a tabela será exibida.

Essa implementação também poderia ser feita com *style binding* ou através de uma diretiva ***ngIf** que veremos no próximo capítulo. No caso de *style binding* o valor de **esconderCursos** poderia ser associado à propriedade **display** de **css**. Para esconder, *display* recebe o valor *none* e para exibir, o valor *block* (Código 15).

```
[style.display]="esconderCursos ? 'none' : 'block'"
```

Código 15: Alternativa a [hidden] usando *style binding* na propriedade *display*. Fonte: autoria própria.

5 Two Way Data Binding

O *two way data binding* (ligação de dados de duas vias) é uma ligação que combina ao mesmo tempo evento com *property binding*. Ou seja, caso o valor no *template* seja modificado pelo usuário essa modificação é enviada à propriedade no componente (evento) e, em sentido inverso, se o valor da propriedade no componente mudar a modificação é refletida no *template*.

A notação para se usar aplicar esta forma de ligação é conhecida como *banana in a box* (banana na caixa) pois consiste em parênteses dentro de colchetes **[()]**. No caso de um controle de formulário, para associar o controle a uma propriedade deve-se inserir a palavra **ngModel** dentro da

caixa e ligá-la ao nome da propriedade correspondente. Por exemplo, no componente **ExibirMensagem** do capítulo anterior há uma propriedade chamada **nome**. Se quisermos ligar essa propriedade com *two way data binding* então devemos fazer:

```
<input id="nome" name="nome" type="text" [(ngModel)]="nome">
```

Código 15: *two way data binding* na propriedade nome. Fonte: autoria própria.

Assim, a sempre que o usuário digitar seu nome no campo *input*, a atualização é feita automaticamente na propriedade **nome**, letra por letra, até que ele termine a digitação e saia do *input* (retire o foco). Da mesma forma, se em algum método do componente o valor de nome for modificado posteriormente essa modificação irá aparecer no formulário. Para testar, faça essa modificação no *template* do componente **ExibirMensagem** (**exibir-mensagem.component.html**) e usando interpolação imprima em um parágrafo no *template* o valor da propriedade (** {{nome}} **).

Continuando no componente **Cursos** vamos adicionar ao *template* (**cursos.component.html**) três controle de *input* do tipo botão de rádio e associá-los usando **[(ngModel)]** ao atributo **tipoCurso** já definido no componente (Código 14). Para tanto, adicione a **div** do Código 16 ao código já existe no *template* (Código 12). Como pode ser visto no Código 16, por se tratar de um botão de rádio, a ligação deve ser repetida para cada botão. E, lembrando, eles devem ter o mesmo **name** no HTML. Observe também que no componente (Código 14) o valor de **tipoCurso** foi inicializado como "**integrado**". Isso foi feito para que o botão do curso técnico integrado fique selecionado por padrão assim que o usuário abre o *template*.

```
<div>
  <input type="radio" id="int" name="tipo-curso"
         [(ngModel)]="tipoCurso" value="integrado">
  <label for="int">Integrado</label>
  <input type="radio" id="sub" name="tipo-curso"
         [(ngModel)]="tipoCurso" value="subsequente">
  <label for="sub">Subsequente</label>
  <input type="radio" id="sup" name="tipo-curso"
         [(ngModel)]="tipoCurso" value="superior">
  <label for="sup">Superior</label>
</div>
```

Código 16: Botões de rádio ligados à propriedade tipoCurso. Fonte: autoria própria.

Apenas para verificar se a ligação está sendo realizada da forma correta imprima no *template* a propriedade **tipoCurso** usando interpolação.

8 Considerações Finais

A comunicação entre o template e o componente é uma parte bastante importante Angular. Sabemos que o *template* é o componente de visão que irá interagir diretamente com o usuário. Assim, o template é responsável por fornecer ou receber informações do componente. Para que essa troca de informações possa ocorrer deve-se usar um dos três mecanismos vistos nesta seção, a saber, evento, *property binding* ou *two way data binding*. O primeiro caso depende de uma ação do usuário no *template*, como um clique de botão, o pressionar de uma tecla, sair ou entrar em um controle de formulário, dentre outras, e executa um método no componente, podendo esse método receber ou não algum valor em seus argumentos. O property binding é a ligação de uma propriedade do HTML com uma propriedade (atributo) no componente. O HTML recebe o valor da

propriedade. Por fim, o *two way data binding* é uma ligação que ocorre nos dois sentidos, sendo ao mesmo tempo um evento e um *property binding* sobre uma propriedade.

Cumpre destacar que o componente deve ser o mais simples possível, de preferência contendo atributos para receber valores do *template*, se necessário, atributos para realizar algum controle no *template*, como exibir ou não uma parte da página, bem como outros atributos simples, para, por exemplo, armazenar referências para imagens ou pastas de arquivos que podem ser usados no *template*. Qualquer outro código com maior lógica idealmente não deve ficar no componente, mais sim em outras classes comuns ou em classes de serviço. Os serviços serão abordados mais adiante e são um tipo especial de classes cujos objetos podem ser instanciados e gerenciados pelo ambiente de execução Angular e são resposáveis por tarefas mais pesadas como se definir um CRUD para uma tabela em um banco de dados ou acessar objetos por meio de um serviço *Web*.

Referências

ANGULAR. **Template Syntax**. Disponível em: <https://angular.io/guide/template-syntax#template-expressions> Acesso em: 06/04/2018.

GRONER, Loiane. Angular. **Curso de Angular**. Disponível em: <https://loiane.training/course/angular/>. Acesso em: 15/04/2018.