

# TypeScript e Angular

*Instrutor: Klaus Cavalcante*



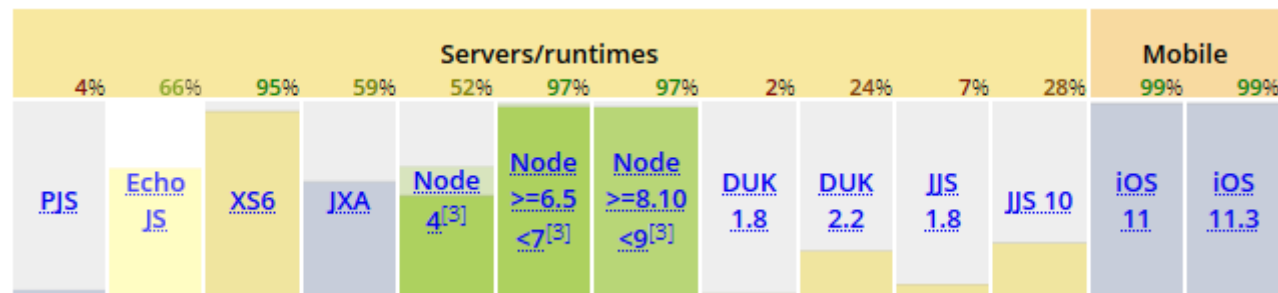
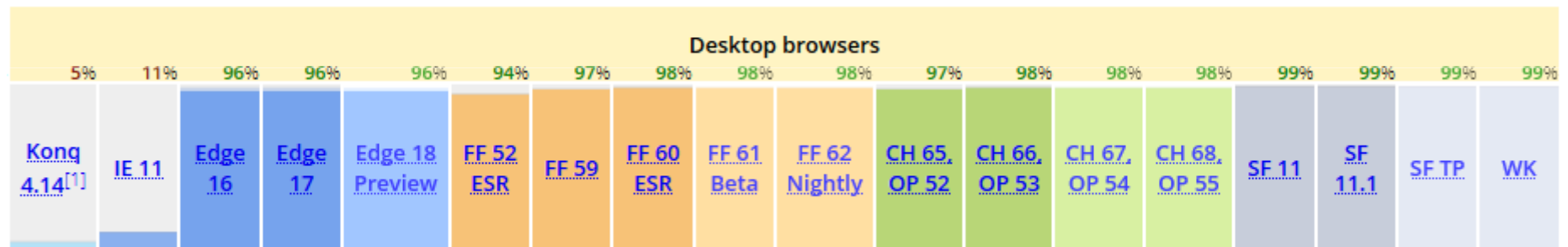
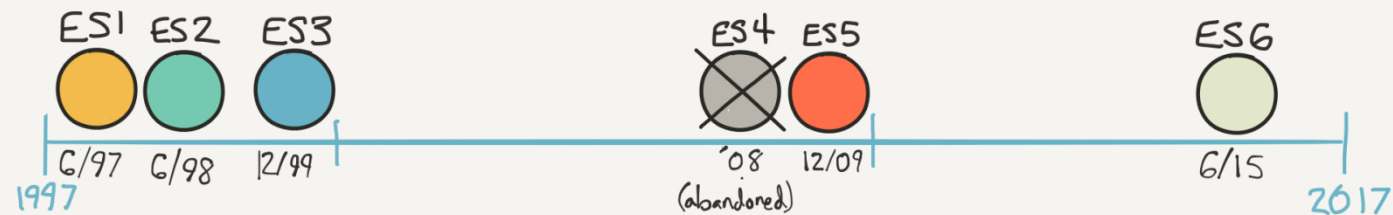
# MOTIVAÇÃO

- ▶ Este curso tem como pré-requisitos HTML e CSS (básico), e JavaScript (básico/intermediário)
- ▶ **NÃO** é necessário conhecer AngularJS 1.x
- ▶ Mais importante que assistir às aulas, é fazer os exercícios propostos. Pratique o máximo possível (dentro e fora do curso)!
- ▶ Normalmente o sucesso em qualquer projeto depende de 1% de inspiração e 99% de transpiração, ou seja, um pouquinho de talento e muito trabalho duro!



# ECMA

## ECMAScript Releases



Fonte: <https://kangax.github.io/compat-table/es6> (20/05/2018)



# PRINCIPAIS RECURSOS DO ES6

- ▶ Variáveis *"let"* e *"const"* (*block-scoped*)
- ▶ Classes e Herança
- ▶ *Getters* e *Setters*
- ▶ Valores *"default"* para parâmetros em funções
- ▶ *Template Strings*
- ▶ Coleções (*Sets* e *Maps*)
- ▶ Iteradores e operadores *for-in* e *for-of*
- ▶ *Arrow Functions*

Fonte: <http://es6-features.org>



# ES6 - VARIÁVEIS LET E CONST

```
let a = 10;
const b = 20;

if (true) {
  let c = 30;
}
console.log(a); // 10
console.log(b); // 20
console.log(c); // ReferenceError: c is not defined.

b = 40; // TypeError: Assignment to constant variable.
```



# ES6 - CLASSES

```
class Pessoa {  
  constructor(nome) {  
    this.nome = nome;  
  }  
  
  getNome() {  
    return this.nome;  
  }  
}  
  
let pessoa = new Pessoa('Zezinho');  
console.log(pessoa.getNome());
```



# ES6 - GETTERS E SETTERS

```
class Pessoa {  
  constructor(nome) {  
    this._nome = nome;  
  }  
  
  get nome() {  
    return this._nome;  
  }  
  
  set nome(valor) {  
    this._nome = valor;  
  }  
}  
  
let pessoa = new Pessoa('Zezinho');  
pessoa.nome = 'Huguinho';  
console.log(pessoa.nome);
```



# ES6 - HERANÇA

```
class Funcionario extends Pessoa {  
  constructor(nome, departamento) {  
    super(nome);  
    this.departamento = departamento;  
  }
```

```
  getDepartamento() {  
    return this.departamento;  
  }  
}
```

```
let func = new Funcionario('Zezinho', 'SUPDE');  
console.log(func.getNome() + ' - ' + func.getDepartamento());
```





# ES6 - VALORES DEFAULT PARA FUNÇÕES

```
function soma(a = 0, b = 0) {  
  return a + b;  
}  
console.log(soma(1, 2)); // 3  
console.log(soma(1)); // 1
```



# ES6 - TEMPLATE STRINGS

```
const multilinhas = `texto string linha 1  
texto string linha 2  
texto string linha 3`;
```

```
console.log(multilinhas);
```

```
const nome = 'Rex';  
const idade = 2;  
const frase = `Meu cachorro ${nome} tem ${idade*7} anos.`;
```

```
console.log(frase);
```



# ES6 - COLEÇÕES

```
let carros = new Set();  
carros.add('Ferrari');  
carros.add('BMW');
```

```
console.log(carros.has('Ferrari')); // true  
console.log(carros.has('BMW'));    // true  
console.log(carros.has('Honda'));  // false
```

```
carros.delete('BMW');
```

```
console.log(carros.has('BMW')); // false
```

```
let map = new Map();  
map.set('server', 'http://git.serpro/');  
map.set('project', 'sief-angular');  
console.log(map.get('server') + map.get('project'));
```



# ES6 - ITERADORES

```
let cores = ['azul', 'amarelo', 'verde'];
```

```
// iterator forEach  
cores.forEach(function (cor) {  
    console.log(cor);  
});
```

```
// iterator for-of  
for (let cor of cores) {  
    console.log(cor);  
}
```

```
// iterator for  
for (let i = 0; i < cores.length; i++) {  
    console.log(cores[i]);  
}
```



# ES6 - ARROW FUNCTIONS

```
let a = ['Hidrogênio', 'Hélio', 'Lítio', 'Berílio'];
```

```
console.log(a.map(function (s) {  
  return s.length;  
})); // [ 10, 5, 5, 7 ]
```

```
console.log(a.map( s => s.length )); // [ 10, 5, 5, 7 ]
```

```
let notas = [8, 4, 10, 5, 7];
```

```
console.log(notas.filter(function (nota) {  
  return nota >= 7;  
})); // [ 8, 10, 7 ]
```

```
console.log(notas.filter(nota => nota >= 7)); // [ 8, 10, 7 ]
```

# TypeScript





# TYPESCRIPT

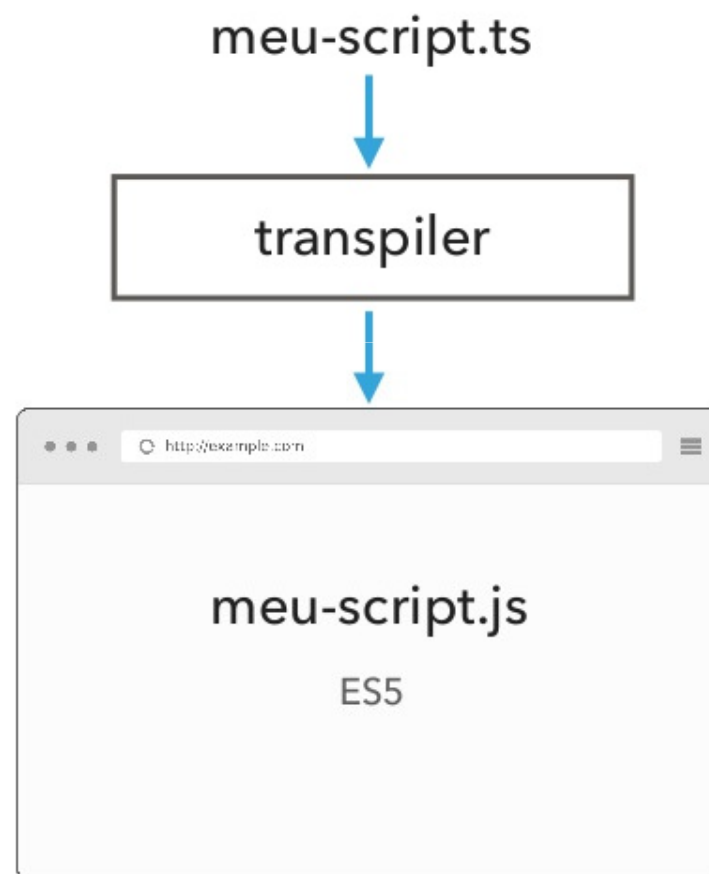
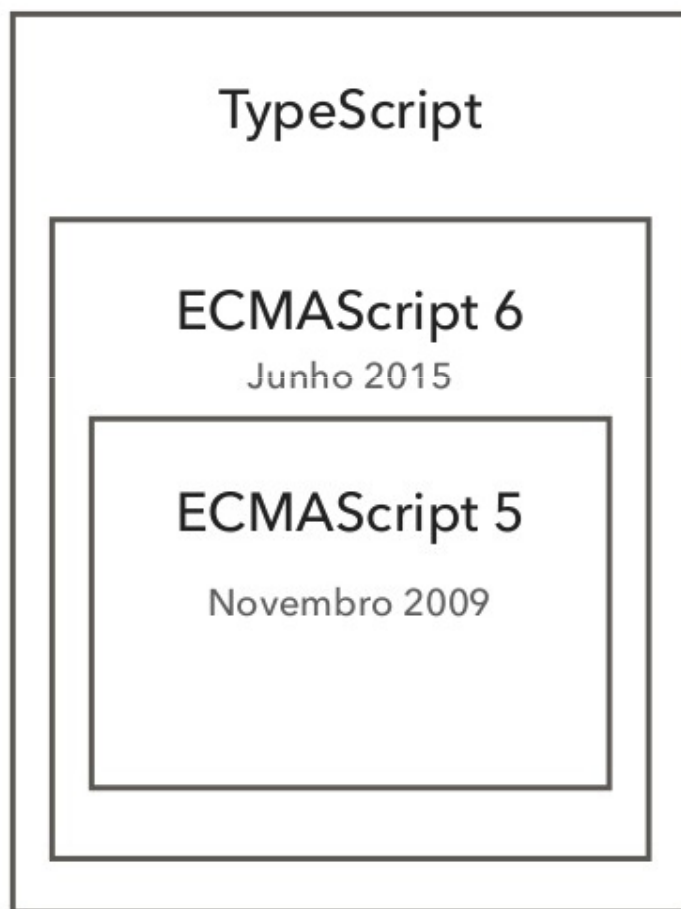
- ▶ TypeScript é um "superset" do ECMAScript 6 (ES6) com um sistema de tipos, *generics* ( $\langle T \rangle$ ) e anotações (*annotations*)

```
1 let nome: string = "Aluno";  
2  
3 Type 'number' is not assignable to type 'string'.  
4 let nome: string  
5 nome = 2;  
6
```

- ▶ Criada por Anders Hejlsberg (criador do Turbo Pascal, Delphi e C#)
- ▶ Projeto *opensource*, desenvolvido e mantido pela Microsoft



# TYPESCRIPT x ECMA







# POR QUE UTILIZAR TYPESCRIPT?

- ▶ Esquema de tipos permite checagem em tempo de compilação (o que torna o código menos propenso a erros)
- ▶ Pode ser compilado para versões anteriores do JavaScript (ou seja, pode ser executado em *browsers* mais antigos)
- ▶ Permite ser "misturado" com bibliotecas nativas em JavaScript
- ▶ O Angular é desenvolvido em TypeScript (normalmente a documentação oficial é disponibilizada primeiro em TypeScript)
- ▶ Veremos os elementos do TypeScript nos exemplos mostrados ao longo do curso.

# Angular





# AGENDA

- ▶ Introdução
- ▶ Ambiente de Desenvolvimento
- ▶ Angular CLI: Instalação e Criação de Projetos
- ▶ Angular CLI: Criação de Arquivos
- ▶ Componentes e *Templates*
- ▶ *Data Binding*
  - *Interpolation*
  - *Property Binding*
  - *Class e Style Binding*
  - *Event Binding*
  - *Two-Way Data Binding*





# AGENDA

- ▶ *Input/Output Properties*
- ▶ Ciclo de Vida dos Componentes
- ▶ Diretivas
- ▶ Formulários: *Template-Driven Forms*
- ▶ Formulários: *Reactive/Model-Driven Forms*
- ▶ Serviços
- ▶ Injeção de Dependências
- ▶ Roteamento
- ▶ Integração com Servidor
- ▶ *Pipes*
- ▶ Angular CLI: Gerando a Build de Produção



# Angular

## Introdução





# ANGULAR

- ▶ <https://angular.io> (site oficial)
- ▶ Parceria Google + Microsoft
- ▶ *Open Source* (código no GitHub)
- ▶ Não é a continuação da versão 1 (AngularJS)
- ▶ Foi totalmente reescrito
- ▶ Utiliza *Web Standards* (recomendações do W3C para o uso de boas práticas de desenvolvimento) e *Web Components* (conjunto de diferentes API's que permitem criar *tags* HTML customizadas)



# BLOCOS PRINCIPAIS

COMPONENTES

DIRETIVAS

ROTEAMENTO

SERVIÇOS

TEMPLATE

METADATA

DATA BINDING

INJEÇÃO  
DEPENDÊNCIA



# BLOCOS PRINCIPAIS

## Componente



*Encapsula:*

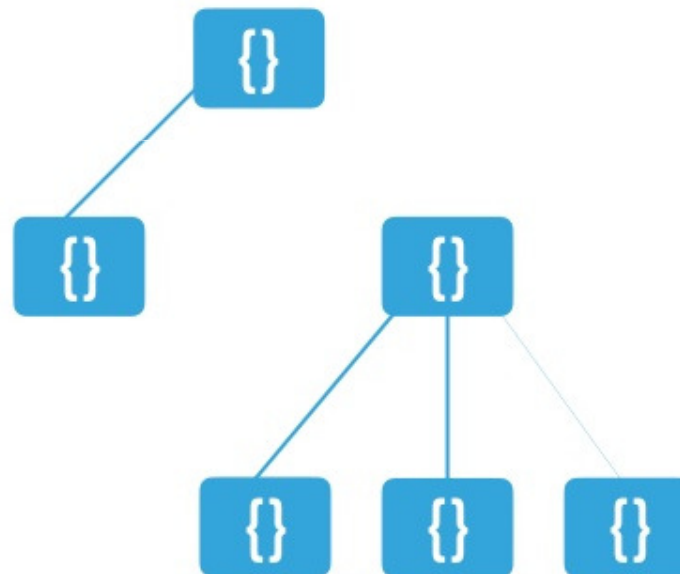
- Template
- Metadata: processamento das classes
- Dado a ser mostrado na tela (Data Binding)
- Comportamento da VIEW





# COMPONENTE

Componente Raiz (Root)





# COMPONENTE



Cabeçalho (Barra de navegação)



Barra  
Lateral



Posts



# COMPONENTE



Cabeçalho (Barra de navegação)



Barra  
Lateral



Post



Post



Post



Post



# COMPONENTE



Cabeçalho (Barra de navegação)



Barra  
Lateral



Star Widget



Post



Post

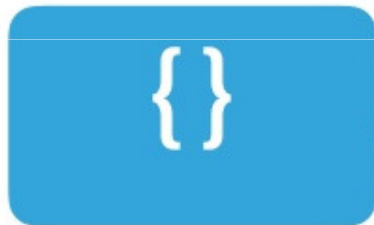


Post

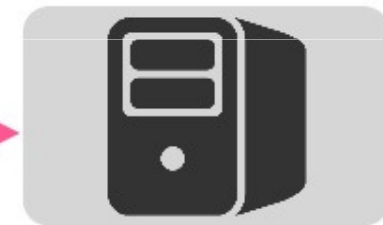


# COMPONENTE

Componente



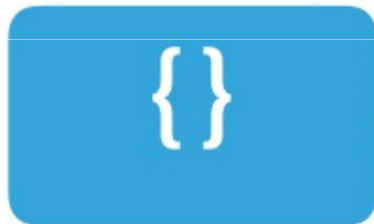
Backend



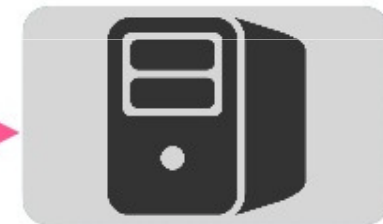


# COMPONENTE

Componente



Backend



Node.JS

Java

.NET

Ruby

Python



# SERVIÇO

Componente



Serviço (Service)



Backend



Node.JS

Java

.NET

Ruby

Python



# SERVIÇO

Componente



Serviço (Service)



Pode ser injetado  
em outras classes

Backend



Node.JS

Java

.NET

Ruby

Python





# ROTEAMENTO

Router



Responsável pela navegação



# DIRETIVA

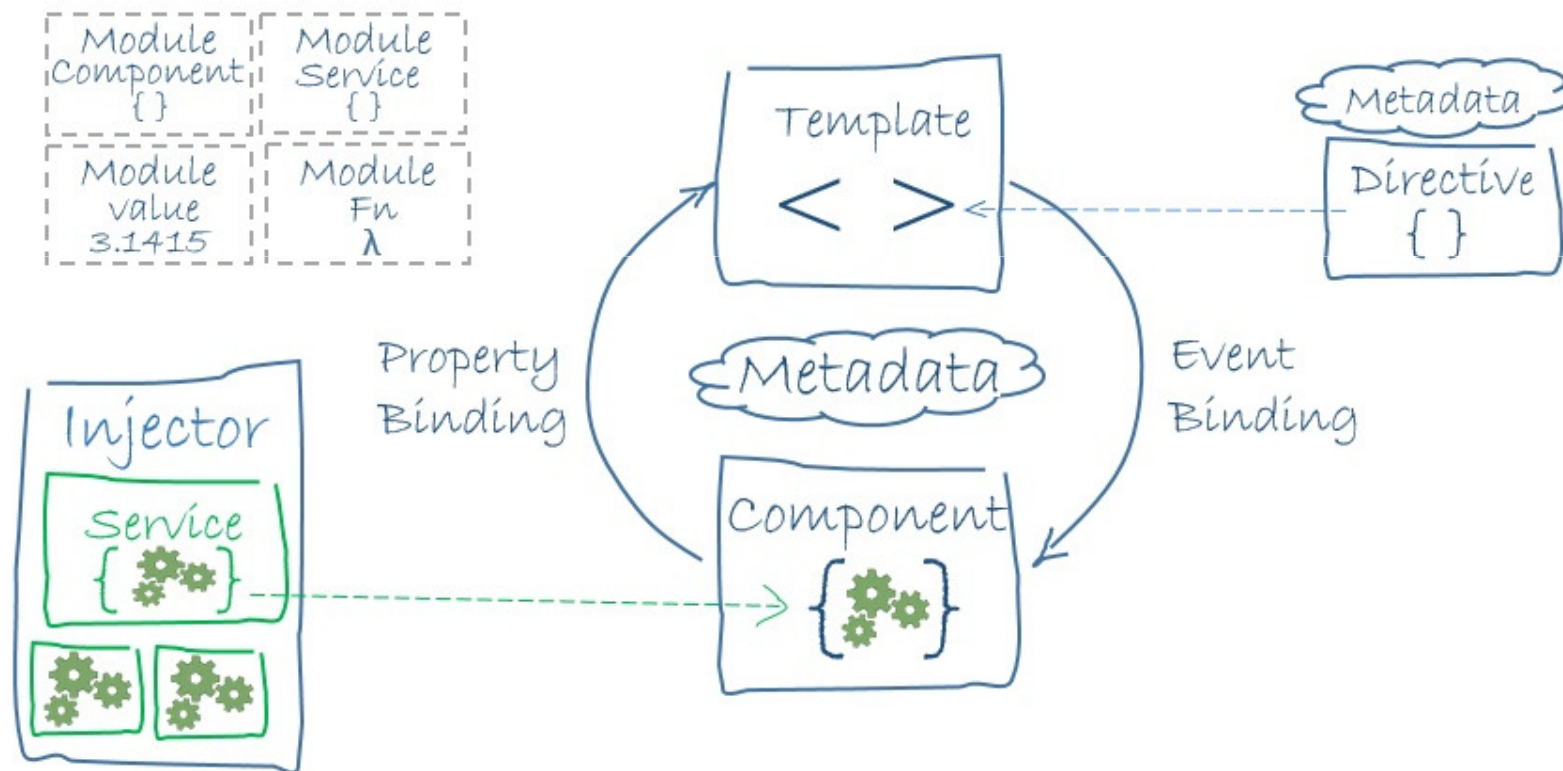
## Diretiva



Responsável por modificar elementos DOM e/ou seu comportamento



# BLOCOS PRINCIPAIS





**CONSULTE A DOCUMENTAÇÃO!**

**A documentação é a sua melhor amiga!**

**Desenvolva o hábito de consultar os docs!**

**<https://angular.io/docs>**

**Adicionalmente, consulte outras fontes na web!**

# Angular

Ambiente de  
Desenvolvimento

A stylized, light blue illustration of a city skyline is positioned at the bottom of the slide. It features various building shapes, including a prominent tower with a spire on the left and a cluster of buildings on the right. The illustration is rendered in a flat, geometric style.



# O QUE PRECISAMOS INSTALAR?

- ▶ NodeJS + NPM (<https://nodejs.org>)
- ▶ Angular CLI (<https://github.com/angular/angular-cli>)
- ▶ Editor de texto de sua preferência
- ▶ Bootstrap (opcional para o curso!)



# NODEJS

- ▶ Plataforma para desenvolvimento de aplicações “*server-side*” usando JavaScript, ideal para prototipações rápidas de pequenas aplicações, e “*backend*” para aplicações REST
- ▶ Quando instalado, permite a criação/manutenção de aplicações através do gerenciador de pacotes NPM (*Node Package Manager*)
- ▶ Baseado na Engine V8 do Google Chrome



<https://nodejs.org>



# INSTALANDO O NODEJS MANUALMENTE

```
$ sudo apt-get update
```

```
$ sudo apt-get install nodejs
```

// atualizar o node para a versão mais recente

```
$ sudo npm cache clean -f
```

```
$ sudo npm install -g n
```

```
$ sudo n stable
```

**DEPRECATED**

Fonte: <https://davidwalsh.name/upgrade-nodejs>





# INSTALANDO O NODEJS USANDO O NVM

```
$ curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.8/install.sh | bash (após o comando, fechar e abrir um novo terminal!)
```

```
$ nvm install node
```

```
$ node -v (para verificar a instalação)
```

Fonte: <https://github.com/creationix/nvm>



# EDITOR DE TEXTO



JÁ SUPORTA TS

Visual Studio Code

[HTTPS://ATOM.IO/PACKAGES/ATOM-TYPESCRIPT](https://atom.io/packages/atom-typescript)



ATOM



JÁ SUPORTA TS

WebStorm

[HTTPS://GITHUB.COM/MICROSOFT/TYPESCRIPT-SUBLIME-PLUGIN](https://github.com/microsoft/typescript-sublime-plugin)



Sublime Text



# VISUAL STUDIO CODE

- ▶ Editor de texto gratuito e *open source* da Microsoft
- ▶ Possui suporte completo para aplicações em TypeScript/Angular
- ▶ <https://code.visualstudio.com/download>
- ▶ Shortcuts para Visual Studio Code
  - <https://code.visualstudio.com/shortcuts/keyboard-shortcuts-windows.pdf>
- ▶ Plugins interessantes para uso com aplicações em Angular
  - Auto Import
  - Angular Essentials (John Papa)
  - vscode-icons



# Angular

## Angular CLI: Instalação e Criação de Projetos





# PARA QUE SERVE O ANGULAR CLI?

- ▶ Cria a estrutura do projeto
- ▶ Gera página HTML inicial, arquivos TypeScript iniciais, arquivos CSS e arquivos de testes unitários
- ▶ Cria arquivo de conf. com todas as dependências do Angular
- ▶ Instala todas as dependências do NodeJS (`npm install`)
- ▶ Configura o Karma para executar os testes unitários com Jasmine
- ▶ Configura o Protractor para executar os testes *end-to-end* (E2E)
- ▶ Inicializa um repositório Git no projeto e faz o *commit* inicial
- ▶ Cria *builds* de desenvolvimento e produção



# ANGULAR CLI

// opcional (caso exista alguma versão anterior)

```
$ npm uninstall -g angular-cli @angular/cli
```

```
$ npm cache clean
```

```
$ npm install -g @angular/cli
```

Pré-requisitos: NodeJS 8.9+ e NPM 5.5.1+

Fonte: <https://github.com/angular/angular-cli#installation>



# ANGULAR CLI: CRIANDO O PROJETO

```
$ ng new NomeProjeto
```

```
$ cd NomeProjeto
```

```
$ ng serve
```

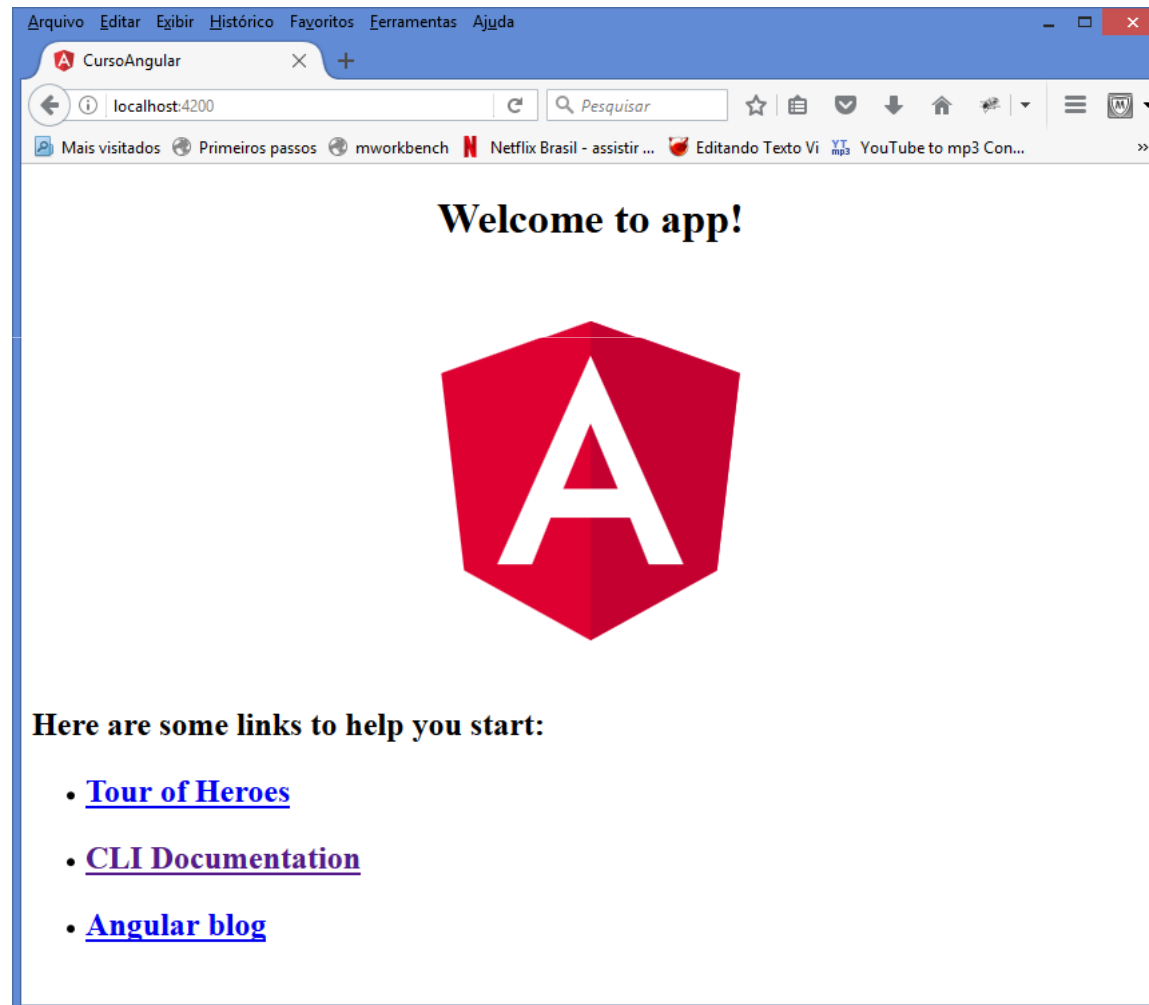
Cria um projeto Angular  
e executa npm install

Sobe um servidor local para  
visualização do projeto no *browser*  
Similar ao npm start



# VISUALIZAÇÃO DA APLICAÇÃO

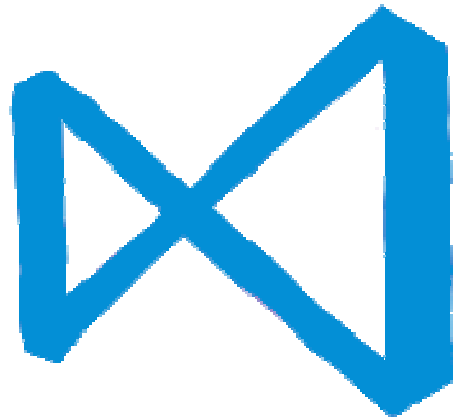
**http://localhost:4200**



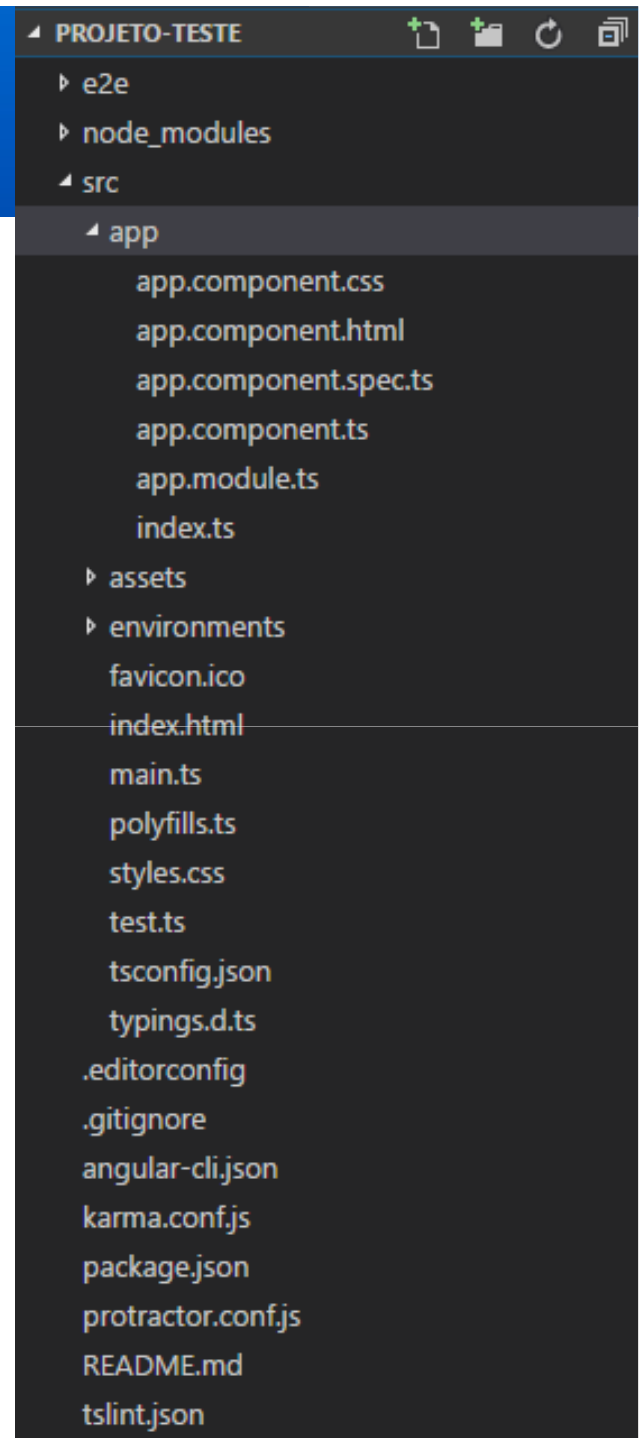




# ESTRUTURA DO PROJETO



Visual Studio Code





## src/index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>CursoAngular</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-
    scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```



## src/app/app.component.ts

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

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



# src/app/app.component.html

```
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1>
    Welcome to {{title}}!
  </h1>
  ...
</div>
...
```



## src/app/app.module.ts

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

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

@NgModule({
  declarations: [
    AppComponent
  ],
  ...
  bootstrap: [AppComponent]
})
export class AppModule { }
```



# ADICIONANDO BOOTSTRAP (OPCIONAL)

► Editar arquivo src/index.html:

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.3/css/bootstrap.min.css" integrity="s

    <title>Hello, world!</title>
  </head>
  <body>
    <h1>Hello, world!</h1>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIkvYIK3UENzmM7KCKRr/rE9/Qpg6aA
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js" integrity="sha384-ApNbgh9B+Y1QK
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.3/js/bootstrap.min.js" integrity="sha384-a5N7Y/aK3qNe
  </body>
</html>
```

Copy



Fonte: <http://getbootstrap.com/docs/4.0/getting-started/introduction/>

# Angular

## Angular CLI: Criação de Arquivos





# ANGULAR CLI: CRIANDO ARQUIVOS

Tipo Arquivo	Comando
Component	<code>ng g c component meu-componente</code>
Service	<code>ng g s service meu-servico</code>
Directive	<code>ng g d directive minha-diretiva</code>
Pipe	<code>ng g p pipe meu-pipe</code>
Class	<code>ng g class minha-classe</code>
Interface	<code>ng g interface minha-interface</code>
Enum	<code>ng g enum meu-enum</code>





# ANGULAR CLI: CRIANDO COMPONENTES

```
$ cd NomeProjeto
```

```
$ ng generate component meu-primeiro
```

ou

```
$ ng g c meu-primeiro
```



# ANGULAR CLI: CRIANDO COMPONENTES

```
$ cd NomeProjeto
```

```
$ ng generate component meu-primeiro
```

ou

```
$ ng g c meu-primeiro
```



# ANGULAR CLI: CRIANDO COMPONENTES

```
$ ng g c meu-primeiro
```

installing component

```
create src\app\meu-primeiro\meu-primeiro.component.css
```

```
create src\app\meu-primeiro\meu-primeiro.component.html
```

```
create src\app\meu-primeiro\meu-primeiro.component.spec.ts
```

```
create src\app\meu-primeiro\meu-primeiro.component.ts
```

```
update src\app\app.module.ts
```



# ANGULAR STYLE GUIDE

```
$ ng g c meu-primeiro
```

installing component

```
create src\app\meu-primeiro\meu-primeiro.component.css
```

```
create src\app\meu-primeiro\meu-primeiro.component.html
```

```
create src\app\meu-primeiro\meu-primeiro.component.spec.ts
```

```
create src\app\meu-primeiro\meu-primeiro.component.ts
```

```
update src\app\app.module.ts
```



# ANGULAR STYLE GUIDE

```
$ ng g c meu-primeiro
```

installing component

```
create src\app\meu-primeiro\meu-primeiro.component.css
```

```
create src\app\meu-primeiro\meu-primeiro.component.html
```

```
create src\app\meu-primeiro\meu-primeiro.component.spec.ts
```

```
create src\app\meu-primeiro\meu-primeiro.component.ts
```

```
update src\app\app.module.ts
```

Arquivos são gerados  
usando o padrão de  
nomenclatura e boas  
práticas segundo o  
*style guide*

# Angular

## Componentes e Templates





# meu-primeiro.component.ts

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

@Component({
  selector: 'app-meu-primeiro',
  templateUrl: './meu-primeiro.component.html',
  styleUrls: ['./meu-primeiro.component.css']
})
export class MeuPrimeiroComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```



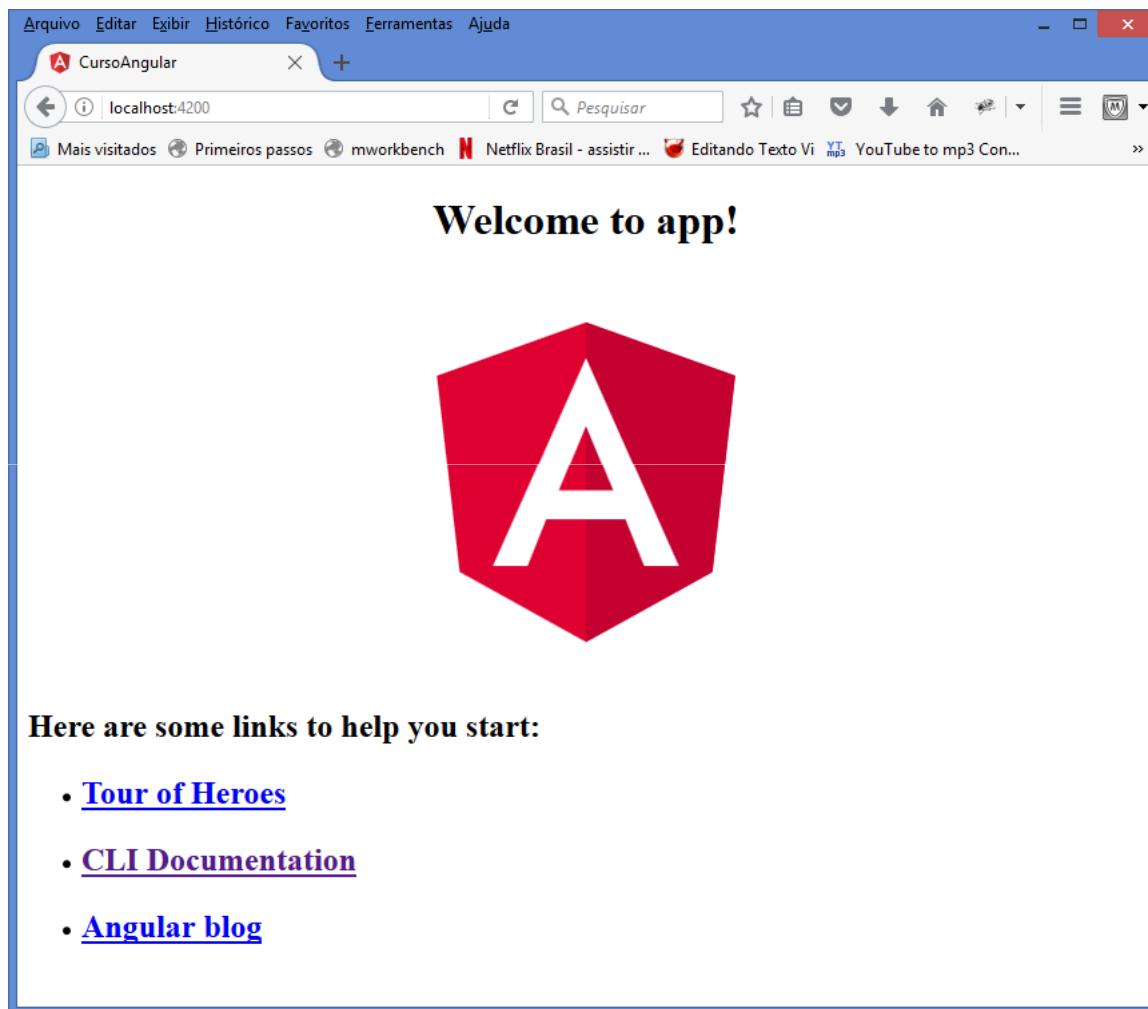
# meu-primeiro.component.html

```
<p>  
  meu-primeiro works!  
</p>
```





# Visualizando o novo componente



**NADA MUDOU!**



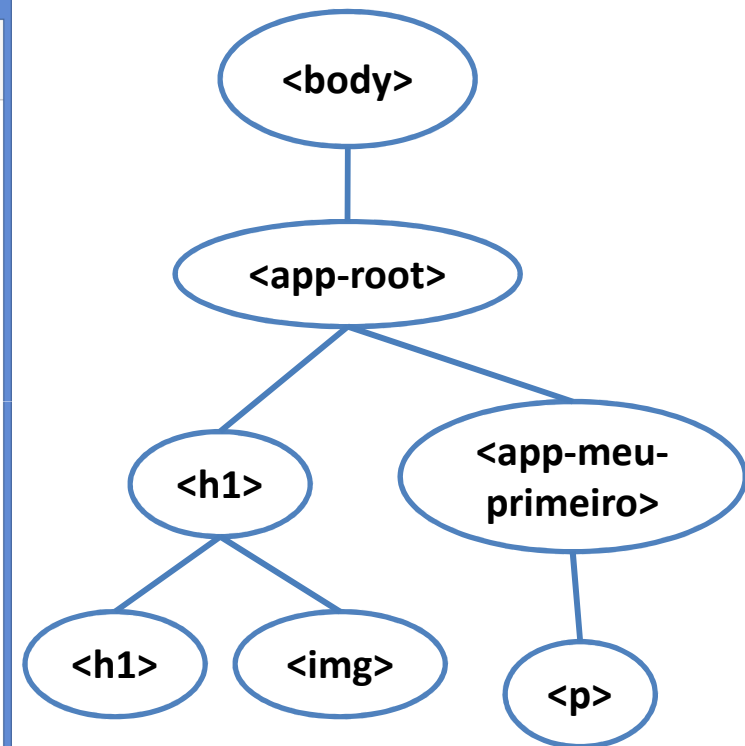
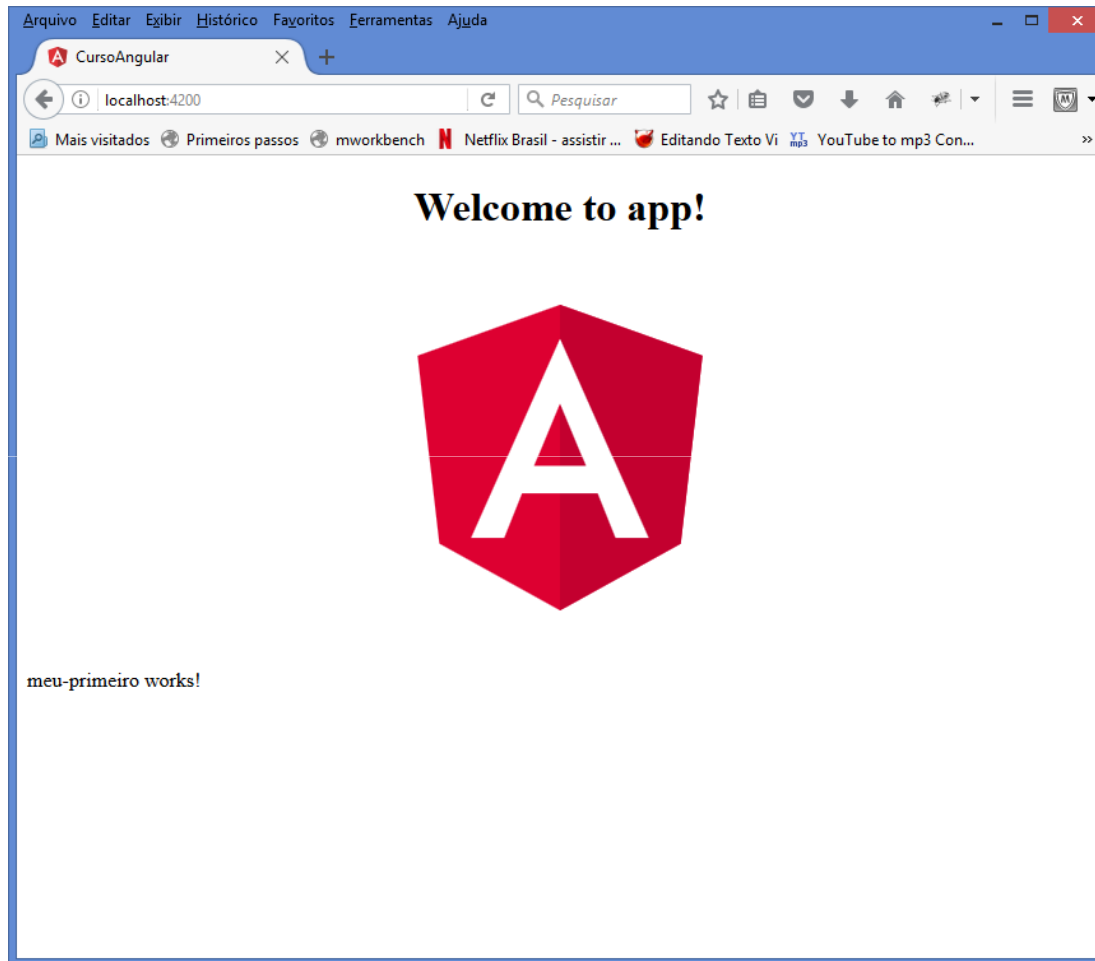
# src/app/app.component.html

```
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1>
    Welcome to {{title}}!
  </h1>
  ...
</div>
<app-meu-primeiro></app-meu-primeiro>
```

Inclusão do novo  
componente



# Visualizando o novo componente



# Angular

## Data Binding





# DATA BINDING

<TEMPLATE>

{COMPONENT}



# DATA BINDING

<TEMPLATE>



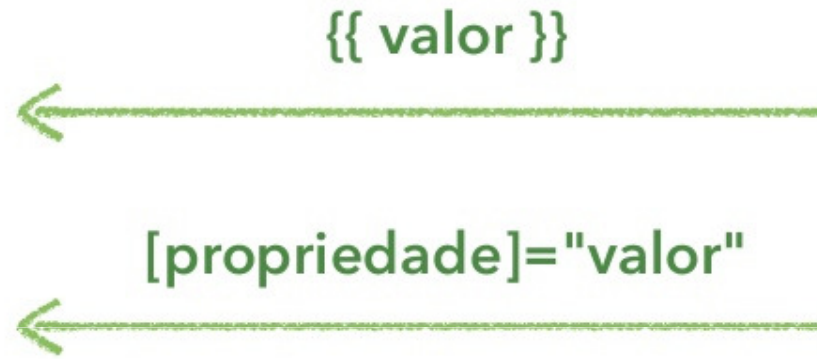
{{ valor }}

{COMPONENT}



# DATA BINDING

<TEMPLATE>

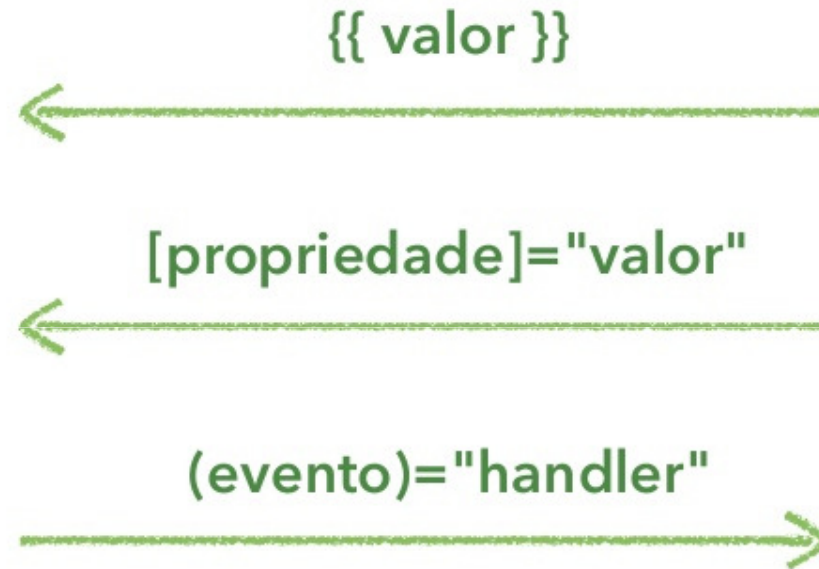


{COMPONENT}



# DATA BINDING

<TEMPLATE>



{COMPONENT}

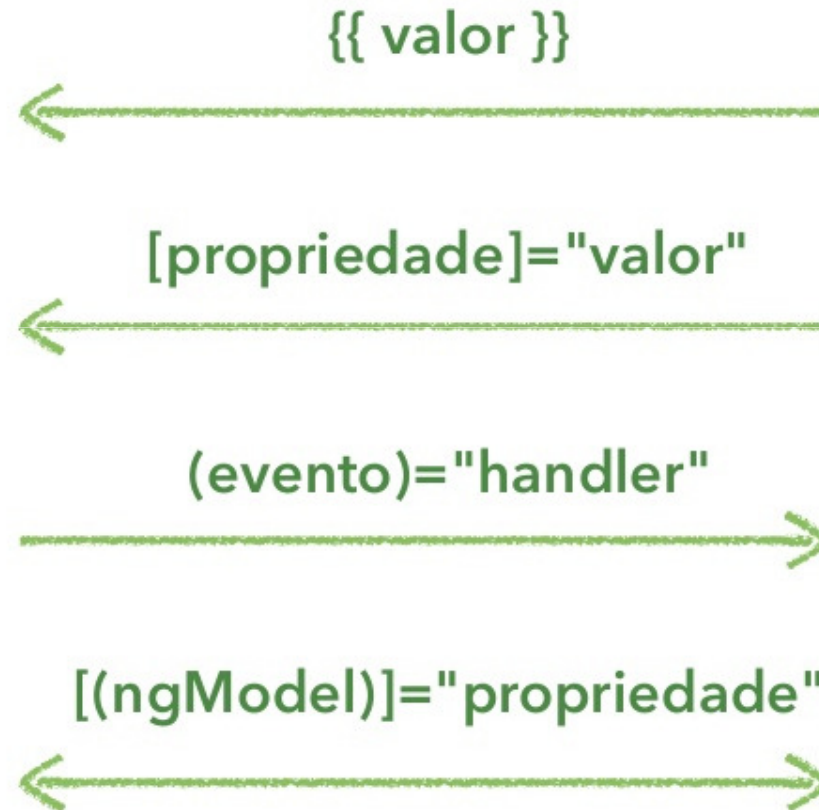




# DATA BINDING

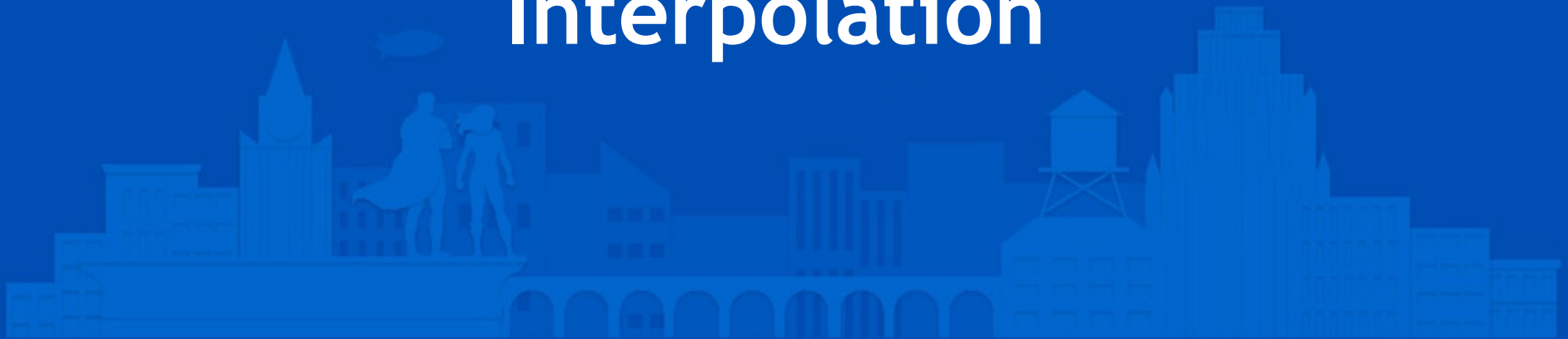
<TEMPLATE>

{COMPONENT}



# Angular

## Interpolation





# INTERPOLATION

- ▶ Valor do *Component* para o *Template*
- ▶ Usado normalmente para saída de dados!
- ▶ Usa-se chaves duplas diretamente no template HTML
  - `{{ variável | expressão | chamada de método }}`
- ▶ Exemplos:
  - `{{ minhaVar }}`
  - `{{ 1 + 1 }}`
  - `{{ (minhaVar && minhaVar2) }}`
  - `{{ getValor() }}`



# INTERPOLATION

```
export class DataBindingComponent implements OnInit {  
  
    private nome = 'SERPRO';  
    ...  
  
    public getUrl() {  
        return 'http://serpro.gov.br';  
    }  
}
```

<h3>Interpolation</h3>

<p>Nome: {{ nome }}</p>

<p>Url: {{ getUrl() }}</p>



# INTERPOLATION

```
export class DataBindingComponent implements OnInit {  
  
    private nome = 'SERPRO';  
    ...  
  
    public getUrl() {  
        return 'http://serpro.gov.br';  
    }  
}
```

<h3>Interpolation</h3>

<p>Nome: {{ nome }}</p>

<p>Url: {{ getUrl() }}</p>

# Angular

## Property Binding





# PROPERTY BINDING

- ▶ Valor do *Component* para o *Template*
- ▶ Pode ser realizado com:
  - *Propriedades HTML, e*
  - *Classes/Propriedades CSS*
- ▶ Usa-se colchetes nas propriedades dos elementos HTML
  - `<tag [propriedade]="variável | expressão | cham. método" />`
- ▶ Exemplos:
  - `<div [hidden]="minhaVar"> ... </div>`
  - `<input type="text" [value]="minhaVar2" />`



# PROPERTY BINDING

```
export class DataBindingComponent implements OnInit {  
  
    nome = 'SERPRO';  
    ocultarUrl = true;  
  
    desativarBotao() {  
        return true; //  
    }  
    ...  
}
```

```
<h3>Property Binding:</h3>  
<p>Nome: {{ nome }}</p>  
<p [hidden]="ocultarUrl">Url: {{ getUrl() }}</p>  
<button type="button" [disabled]="desativarBotao()">Submit</button>
```





# PROPERTY BINDING

```
export class DataBindingComponent implements OnInit {  
  
    nome = 'SERPRO';  
    ocultarUrl = true;  
  
    desativarBotao() {  
        return true; //  
    }  
    ...  
}
```

```
<h3>Property Binding:</h3>  
<p>Nome: {{ nome }}</p>  
<p [hidden]="ocultarUrl">Url: {{ getUrl() }}</p>  
<button type="button" [disabled]="desativarBotao()">Submit</button>
```

# Angular

## Diretiva ngModel e Variáveis de Template





# NGMODEL E VARIÁVEIS DE TEMPLATE

- ▶ A diretiva `ngModel` permite ao Angular rastrear o valor, estado e a validade dos campos de entrada (`<input>`, `<select>`, ...)
- ▶ Pode ser usado em combinação com variáveis de *template* (`#var`) para ter acesso às informações mencionadas acima
- ▶ Variáveis de *template* (`#var`) são variáveis locais que referenciam elementos DOM dentro de um *template*



# NGMODEL E VARIÁVEIS DE TEMPLATE

- ▶ Para utilizar a diretiva `ngModel`, é necessário importar o módulo *FormsModule* do Angular no arquivo `app.module.ts`

```
import { FormsModule } from '@angular/forms';
```

```
@NgModule({  
  imports: [  
    BrowserModule,  
    FormsModule  
  ],  
  ...  
})
```



# NGMODEL E VARIÁVEIS DE TEMPLATE

- ▶ Para utilizar a diretiva `ngModel`, é necessário importar o módulo *FormsModule* do Angular no arquivo `app.module.ts`

```
import { FormsModule } from '@angular/forms';
```

```
@NgModule({  
  imports: [  
    BrowserModule,  
    FormsModule  
  ],  
  ...  
})
```



# NGMODEL E VARIÁVEIS DE TEMPLATE

```
<input type="text" ngModel #entrada1 >
{{ entrada1.value }}
```

```
<input type="text" ngModel #entrada2="ngModel" required >
<p>Valor: {{ entrada2.value }}</p>
<p>Visitado? {{ entrada2.touched }}</p>
<p>Modificado? {{ entrada2.dirty }}</p>
<p>Valido? {{ entrada2.valid }}</p>
```

```
<select ngModel #selecao>
  <option value="1">Opção 1</option>
  <option value="2">Opção 2</option>
</select>
{{ selecao.value }}
```



# NGMODEL E VARIÁVEIS DE TEMPLATE

```
<input type="text" ngModel #entrada1 >
{{ entrada1.value }}
```

```
<input type="text" ngModel #entrada2="ngModel" required >
<p>Valor: {{ entrada2.value }}</p>
<p>Visitado? {{ entrada2.touched }}</p>
<p>Modificado? {{ entrada2.dirty }}</p>
<p>Valido? {{ entrada2.valid }}</p>
```

Propriedades do  
ngModel

```
<select ngModel #selecao>
  <option value="1">Opção 1</option>
  <option value="2">Opção 2</option>
</select>
{{ selecao.value }}
```



# NGMODEL E VARIÁVEIS DE TEMPLATE

► Variável de *template* sem a diretiva ngModel, funciona?

```
<input type="text" #entrada1 >  
<p>Valor: {{ entrada1.value }}</p>
```

Qual é o  
resultado?





# NGMODEL E VARIÁVEIS DE TEMPLATE

- Variável de *template* sem a diretiva ngModel, funciona?

```
<input type="text" #entrada1 >  
<p>Valor: {{ entrada1.value }}</p>
```

```
<input type="text" #entrada1 (input)="onInput(entrada1)">  
<p>Valor: {{ entrada1.value }}</p>
```

Com *event binding*, Angular  
passa a “observar” o  
elemento DOM

# Angular

## Class e Style Binding





# CLASS BINDING – EXEMPLO 1

```
<label>Selecione uma opção:</label>
<select #selecao ngModel>
  <option value="">Selecione uma opção</option>
  <option value="success">Sucesso</option>
  <option value="info">Info</option>
</select>

<div class="alert" role="alert"
  [hidden]="selecao.value !== 'success'"
  [class.alert-success]="selecao.value === 'success'">
  Mensagem de sucesso
</div>
<div class="alert" role="alert"
  [hidden]="selecao.value !== 'info'"
  [class.alert-info]="selecao.value === 'info'">
  Mensagem informativa
</div>
```



# CLASS BINDING EXEMPLO 1

Variável local  
chamada selecao

```
<label>Selecione uma opção:</label>  
<select #selecao ngModel>  
  <option value="">Selecione uma opção</option>  
  <option value="success">Sucesso</option>  
  <option value="info">Info</option>  
</select>
```

Class Binding

```
<div class="alert" role="alert"  
  [hidden]="selecao.value !== 'success'"  
  [class.alert-success]="selecao.value === 'success'">  
  Mensagem de sucesso  
</div>  
<div class="alert" role="alert"  
  [hidden]="selecao.value !== 'info'"  
  [class.alert-info]="selecao.value === 'info'">  
  Mensagem informativa  
</div>
```



# STYLE BINDING

```
<div [style.display]="selecao.value === 'alert-danger' ? 'block' :  
  'none'" class="alert alert-danger" role="alert">  
  Esse texto só aparece em caso de erro!  
</div>
```



# STYLE BINDING

```
<div [style.display]="selecao.value === 'info' ? 'block' : 'none'"  
  class="alert alert-info" role="alert">  
  Esse texto só aparece com o valor 'info'  
</div>
```

Expressão com o  
valor da variável

```
<p [style.border]="selecao.value === 'info' ? '1px solid red' : ''">  
  Caixa de texto com borda vermelha  
</p>
```

```
<div [style.background-color]="selecao.value === 'info' ?  
  'yellow' : ''">  
  Caixa de texto com fundo amarelo  
</div>
```

# Angular

## Event Binding





# EVENT BINDING

- ▶ Valor do *Template* para o *Component*
- ▶ Usa-se parênteses no evento
  - `<tag (evento)="handler()">`
- ▶ Método “handler” declarado no Componente
- ▶ Exemplo:
  - ▶ `<button (click)="onClick()" />`





# EVENT BINDING - EXEMPLO 1

```
<button class="btn" (click)="onClick()">Clicar</button>
```



# EVENT BINDING - EJEMPLO 1

```
<button class="btn" (click)="onClick()">Clickar</button>
```

Evento a ser  
escutado

Método a ser  
executado



# EVENT BINDING - EXEMPLO 1

```
<button class="btn" (click)="onClick()">Clicar</button>
```

Evento a ser  
escutado

Método a ser  
executado

```
export class DataBindingComponent implements OnInit {
```

```
...
```

```
onClick() {  
    alert('botão clicado!');  
}
```

```
}
```



# EVENT BINDING - EXEMPLO 1

```
<button class="btn" (click)="onClick()">Clicar</button>
```

Evento a ser  
escutado

Método a ser  
executado

```
export class DataBindingComponent implements OnInit {
```

```
...
```

```
onClick() {  
    alert('botão clicado!');  
}
```

```
}
```



## EVENT BINDING - EXEMPLO 2

```
<input type="text" ngModel #fone />
<p>Telefone: {{ fone.value }}</p>
<button (click)="onClick(fone.value)">Chamar</button>
<p>{{ mensagem }}</p>
```

```
export class DataBindingComponent implements OnInit {
    mensagem = '';

    onClick(telefone: string) {
        if(telefone) {
            this.mensagem = 'Chamando número ' + telefone + '...';
        } else {
            this.mensagem = 'Nenhum número para chamar!';
        }
    }
}
```



## EVENT BINDING - EXEMPLO 2

```
<input type="text" ngModel #fone />
<p>Telefone: {{ fone.value }}</p>
<button (click)="onClick(fone.value)">Chamar</button>
<p>{{ mensagem }}</p>
```

```
export class DataBindingComponent implements OnInit {
    mensagem = '';
```

```
    onClick(telefone: string) {
        if(telefone) {
            this.mensagem = 'Chamando número ' + telefone + '...';
        } else {
            this.mensagem = 'Nenhum número para chamar!';
        }
    }
}
```

```
}
```



## EVENT BINDING - EXEMPLO 3

```
<input type="text"  
      (keyup)="onKeyUp($event)"  
      (keyup.enter)="onSave(entrada.value)"  
      (blur)="onSave(entrada.value)"  
      #entrada >
```

```
<div>Conteúdo Atual: {{conteudoAtual}}</div>  
<div>Conteúdo Salvo: {{conteudoSalvo}}</div>
```



## EVENT BINDING - EXEMPLO 3

\$event é do tipo  
evento DOM

```
<input type="text"  
      (keyup)="onKeyUp($event)"  
      (keyup.enter)="onSave(input.value)"  
      (blur)="onSave(input.value)"  
#input >
```

```
<div>Conteúdo Atual: {{conteudoAtual}}</div>  
<div>Conteúdo Salvo: {{conteudoSalvo}}</div>
```





## EVENT BINDING - EXEMPLO 3

\$event é do tipo  
evento DOM

```
<input type="text"  
  (keyup)="onKeyUp($event)"  
  (keyup.enter)="onSave(input.value)"  
  (blur)="onSave(input.value)"  
  #input >
```

Acesso à variável  
local declarada

```
<div>Conteúdo Atual: {{conteudoAtual}}</div>  
<div>Conteúdo Salvo: {{conteudoSalvo}}</div>
```



## EVENT BINDING - EXEMPLO 3

```
export class DataBindingComponent implements OnInit {  
  
    conteudoAtual = '';  
    conteudoSalvo = '';  
  
    ...  
  
    onKeyup(event: KeyboardEvent) {  
        console.log(event);  
        this.conteudoAtual = (<HTMLInputElement> event.target).value;  
    }  
  
    onSave(texto: string) {  
        this.conteudoSalvo = texto;  
    }  
}
```



## EVENT BINDING - EXEMPLO 3

```
export class DataBindingComponent implements OnInit {  
  
  conteudoAtual = '';  
  conteudoSalvo = '';  
  
  ...  
  
  onKeyUp(event: KeyboardEvent) {  
    console.log(event);  
    this.conteudoAtual = (<HTMLInputElement> event.target).value;  
  }  
  
  onSave(value: string) {  
    this.conteudoSalvo = value;  
  }  
}
```

\$event tem propriedades  
como event.target e  
event.target.value



## EVENT BINDING - EXEMPLO 4

```
<span
  (mouseover)="onToggleHover()"
  (mouseout)="onToggleHover()"
  [class.highlight]="isMouseOver">
  Passe o mouse sobre o texto!
</span>
```

```
export class DataBindingComponent implements OnInit {
  ...

  isMouseOver = false;

  onToggleHover() {
    this.isMouseOver = !this.isMouseOver;
  }
}
```



## EVENT BINDING - EXEMPLO 4

```
.highlight {  
  font-weight: bold;  
  background-color: yellow;  
  color: black;  
}
```



# DICA

► Quais eventos posso usar?

- <http://developer.mozilla.org/en-US/docs/Web/Events>



# EXERCÍCIO 1

1. Criar o componente CelsiusComponent que leia do usuário um valor em graus Fahrenheit, através de um `<input>`. Calcular o seu equivalente em graus Celsius e exibir o resultado. Fórmula  $\Rightarrow C = (F - 32) * 5/9$
2. Ao validar o campo, exibir a mensagem "Campo inválido!", se este não for preenchido ou não corresponder a um valor em ponto flutuante válido.
3. Usar Event Binding para enviar dados do template para o componente
4. Usar as classes do Bootstrap para layout em formulário (<https://getbootstrap.com/docs/4.0/components/forms/>)
5. Usar o **Alert** do Bootstrap para exibir o resultado ('alert-success') e a mensagem de validação ('alert-warning') (<https://getbootstrap.com/docs/4.0/components/alerts/>)



## EXERCÍCIO 2

1. Criar o componente EleicaoComponent que leia do usuário os votos brancos, nulos e válidos de uma eleição, através de três <input>'s.
2. Calcular e exibir o percentual que cada um representa em relação ao total
3. Ao validar os campos, exibir a mensagem "Campo inválido!" para cada um dos campos que não foram preenchidos ou não correspondam a valores inteiros válidos.
4. Usar Event Binding para enviar dados do template para o componente
5. Usar as classes do Bootstrap para layout em formulário (<https://getbootstrap.com/docs/4.0/components/forms/>)
6. Usar o **Alert** do Bootstrap para exibir o resultado ('alert-success') e a mensagem de validação ('alert-warning') (<https://getbootstrap.com/docs/4.0/components/alerts/>)





## EXERCÍCIO 3 (extraclasse)

1. Criar o componente BancoComponent que leia do usuário o saldo de uma conta e o valor da operação, através de dois <input>, e o tipo de operação (depósito-1, saque-2), através de um <select>.
2. Calcular a operação e exibir o novo saldo.
3. Exibir "Conta com saldo negativo!", caso o saldo da conta fique negativo.
4. Usar Event Binding para enviar dados do template para o componente
5. Usar as classes do Bootstrap para layout em formulário (<https://getbootstrap.com/docs/4.0/components/forms/>)
6. Usar o **Alert** do Bootstrap para exibir o resultado ('alert-success') e a mensagem de validação ('alert-warning') (<https://getbootstrap.com/docs/4.0/components/alerts/>)

# Angular

## Two-way Data Binding





# TWO-WAY DATA BINDING

Como manter o *template* e o componente atualizados?



# TWO-WAY DATA BINDING

Como manter o *template* e o componente atualizados?

```
<input type="text"  
  [value]="nome"  
  (input)="nome = $event.target.value" />
```

*Property binding +  
event binding*



# TWO-WAY DATA BINDING

- ▶ Valor do *Template* para o *Component* e vice-versa
- ▶ Usa-se o *binding* de eventos + o *binding* de propriedades
- ▶ A forma mais simples é fazer ambos os *bindings* com a diretiva `ngModel`:
  - `<input type="text" [(ngModel)]="nome"/>`



# TWO-WAY DATA BINDING

*Two-way data binding  
com ngModel*

```
<input type="text" [(ngModel)]="nome" />  
<br>  
<p>Você digitou {{nome}}</p>
```

```
export class DataBindingComponent implements OnInit {  
  
    nome = '';  
    ...  
}
```



# TWO-WAY DATA BINDING

- ▶ Obs: *Two-way Data Binding* é um recurso que deve ser evitado ao máximo por razão de seu alto custo de processamento.



## EXERCÍCIO 4

1. Modificar o Exercício 1 (CelsiusComponent) para usar o mecanismo Two-Way Data Binding para o campo de entrada.
2. Obs1: O *handler* do evento do botão não deve mais passar como parâmetro de entrada, o valores do <input> utilizado na solução. Desta forma, usaremos este evento apenas para acionar a lógica a ser executada e atualização das variáveis do componente.





## EXERCÍCIO 5

1. Modificar o Exercício 2 (EleicaoComponent) para utilizar o mecanismo de Two-Way Data Binding para os campos de entrada.
2. Obs1: O *handler* do evento do botão não deve mais passar como parâmetros de entrada, os valores dos `<input>` utilizados na solução. Desta forma, usaremos este evento apenas para acionar a lógica a ser executada e atualização das variáveis do componente.



## EXERCÍCIO 6 (extraclasse)

1. Modificar o Exercício 3 (BancoComponent) para utilizar o mecanismo de Two-Way Data Binding para os campos de entrada.
2. Obs1: O *handler* do evento do botão não deve mais passar como parâmetros de entrada, os valores dos `<input>` e `<select>` utilizados na solução. Desta forma, usaremos este evento apenas para acionar a lógica a ser executada e atualização das variáveis do componente.

# Angular

## Input Properties





# CRIANDO COMPONENTES REUTILIZÁVEIS

```
<app-curso nome="Angular"></app-curso>
```

Objetivo: criar um componente onde podemos incluir propriedades customizadas para que outros componentes possam usá-lo



# CRIANDO COMPONENTES REUTILIZÁVEIS

```
@Component({  
  selector: 'app-curso',  
  templateUrl: './curso.component.html',  
  styleUrls: ['./curso.component.css']  
})  
export class CursoComponent implements OnInit {  
  nome = null;  
  ...  
}
```

<p>{{nome}}</p>

**1 – Criamos o componente normalmente**



# INPUT PROPERTIES

```
@Component({  
  selector: 'app-curso',  
  templateUrl: './curso.component.html',  
  styleUrls: ['./curso.component.css']  
})  
export class CursoComponent implements OnInit {  
  @Input() nome = null;  
  ...  
}
```

**2 – Usamos a anotação @Input nos atributos que queremos expor**



# INPUT PROPERTIES

```
@Component({
  selector: 'app-curso',
  templateUrl: './curso.component.html',
  styleUrls: ['./curso.component.css']
})
export class CursoComponent implements OnInit {
  @Input('nomeCurso') nome = null;
  ...
}
```

```
<app-curso nomeCurso="Angular"></app-curso>
```

**3 – Caso queira expor o atributo com um nome diferente, podemos passar como parâmetro para o @Input**



## EXERCÍCIO 7

1. Criar o componente `TesteCursoComponent`, que possua um `<input>` e um componente `<app-curso>` do exemplo anterior .
2. O componente `<app-curso>` deverá ser atualizado simultaneamente pelo valor digitado no `<input>`



# Angular

## Output Properties





# CRIANDO COMPONENTES REUTILIZÁVEIS

A custom component UI consisting of three elements: a blue square button with a white minus sign (-) on the left, a white rectangular input field in the center containing the text '10', and a blue square button with a white plus sign (+) on the right.

Componente customizado

Objetivo: disparar um evento “mudou” toda a vez que o usuário clicar nos botões de + ou –

Evento recebe novo valor do input



# CRIANDO COMPONENTES REUTILIZÁVEIS

```
<app-contador valor="10" (mudou)="onMudou($event)">  
</app-contador>
```

Objetivo: poder escutar eventos customizados, ou seja, criados por nós

```
export class TesteContadorComponent implements OnInit {  
  ...  
  
  onMudou(dados: any) {  
    alert(dados.novoValor);  
  }  
}
```



# CRIANDO COMPONENTES REUTILIZÁVEIS

-  +

```
<div>  
  <button class="btn btn-primary" (click)="decrementa()">-</button>  
  <input type="text" [value]="valor" readonly>  
  <button class="btn btn-primary" (click)="incrementa()">+</button>  
</div>
```

**1 – Criamos o componente normalmente**



# CRIANDO COMPONENTES REUTILIZÁVEIS

-

10

+

```
export class ContadorComponent implements OnInit {  
  
  @Input() valor = 0;  
  
  decrementa() {  
    this.valor--;  
  }  
  
  incrementa() {  
    this.valor++;  
  }  
}
```

**1 – Criamos o componente normalmente**



# CRIANDO COMPONENTES REUTILIZÁVEIS

```
export class ContadorComponent implements OnInit {  
  
    @Output() mudou = new EventEmitter();  
  
    ...  
}
```

**2 – Criamos um atributo do tipo EventEmitter (que irá emitir o evento “mudou”)**

**Anotamos com o @Output para mostrar que o atributo será exposto como valor de saída do Component**



# CRIANDO COMPONENTES REUTILIZÁVEIS

## 3 – Emitimos o evento com o valor que desejamos expor

```
decrementa() {  
  this.valor--;  
  this.mudou.emit({novoValor: this.valor});  
}
```

```
incrementa() {  
  this.valor++;  
  this.mudou.emit({novoValor: this.valor});  
}
```



# CRIANDO COMPONENTES REUTILIZÁVEIS

```
export class ContadorComponent implements OnInit {  
  
    @Output('atualizou') mudou = new EventEmitter();  
  
    ...  
}
```

```
<app-contador valor="10" (atualizou)="onMudou($event)">  
</app-contador>
```

**4 – Também podemos expor o evento com um nome diferente do que utilizado internamente**





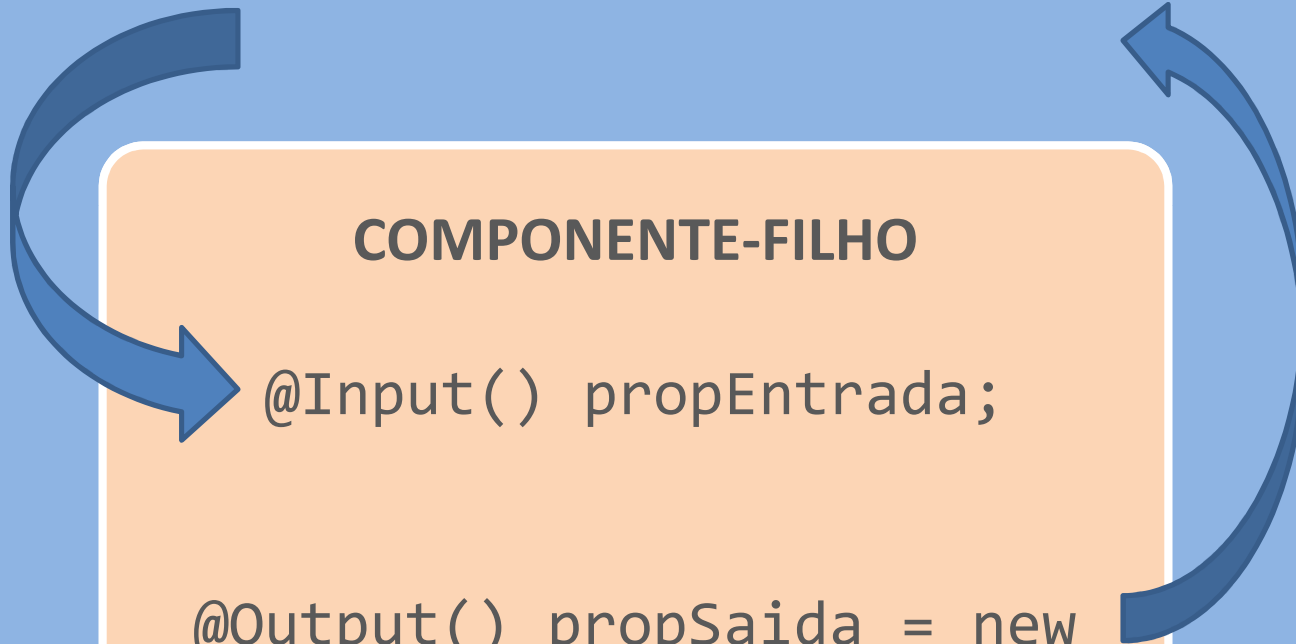
# RESUMO INPUT/OUTPUT PROPERTIES

## COMPONENTE-PAI

### COMPONENTE-FILHO

`@Input() propEntrada;`

`@Output() propSaida = new  
EventEmitter();`





## RESUMO

- ▶ A comunicação entre componentes pode ser realizada através de atributos `@Input()` e `@Output()` como visto anteriormente



## EXERCÍCIO 8

1. Criar o componente CalculadoraComponent, que implemente uma calculadora para op. aritméticas. Incluir dois `<input>` para os operandos e quatro botões para as operações.
2. Exibir o resultado em um `<input>` somente leitura.
3. Ao validar os campos, exibir a mensagem "Campo inválido!" para cada um dos campos que não foram preenchidos ou não correspondam a valores numéricos válidos.
4. Exibir a mensagem "Operação inválida! Divisão por zero!", caso seja verificado tal cenário durante a operação.
5. Incluir um botão de memória (M+) para armazenar o resultado atual calculado e outro botão (M-) para atribuir o valor da memória ao segundo operando.



## EXERCÍCIO 8 (cont.)

6. Definir como @Input um atributo se a calculadora deve ser exibida como padrão (ops. aritméticas) ou científica (potência e raiz - no segundo input).
7. Definir como @Output um evento "calculado" que ao ser emitido, possuirá o valor da operação realizada.
8. Encapsular o componente em um painel do Bootstrap (<https://getbootstrap.com/docs/4.0/components/card/#header-and-footer>) com título apropriado
9. Criar o componente FerramentasComponent que possua o componente <calculadora>. Definir a forma de exibição da calculadora ('padrao' ou 'cientifica') através de um <select>, e exibir em um <input> somente leitura o resultado das operações da calculadora provenientes do evento "calculado".



## EXERCÍCIO 9

1. Criar o componente LoginComponent que contenha dois `<input>` e um botão. Ao clicar no botão, autenticar os dados de um usuário.
2. Definir como `@Input` os nomes dos *labels* dos campos 'login' e 'senha'.
3. Definir como `@Output` um evento "checado" que retorna um booleano correspondente ao resultado do processo de autenticação do usuário, e uma mensagem (para o caso de falha na autenticação).
4. Encapsular o componente em um painel do Bootstrap (<https://getbootstrap.com/docs/4.0/components/card/#header-and-footer>) com título apropriado.



## EXERCÍCIO 9 (cont.)

5. Criar o componente MainComponent que possua o componente <login>.
6. Definir os nomes dos *labels* do componente <login> através de dois <input>.
7. Implementar *handler* para o evento 'chechado', a fim de exibir e ocultar um menu do Bootstrap (<https://getbootstrap.com/docs/4.0/components/navbar/>) (exibir o menu apenas se o usuário foi autenticado com sucesso, e ocultar, caso contrário).
8. Caso email e senha não correspondam a 'admin@serpro.gov.br' e '123456', exibir a mensagem “Login e/ou senha inválidos!”, recebida do componente de login no MainComponent (em um alert do Bootstrap) (<https://getbootstrap.com/docs/4.0/components/alerts/>)



## EXERCÍCIO 9 (cont.)

9. Limpar os campos 'login' e 'senha' do componente <login> caso a autenticação tenha sido realizada com sucesso
10. Incluir no menu um *link* (<a>) para a funcionalidade de 'deslogar' (*Log off*).

# Angular

## Ciclo de Vida dos Componentes

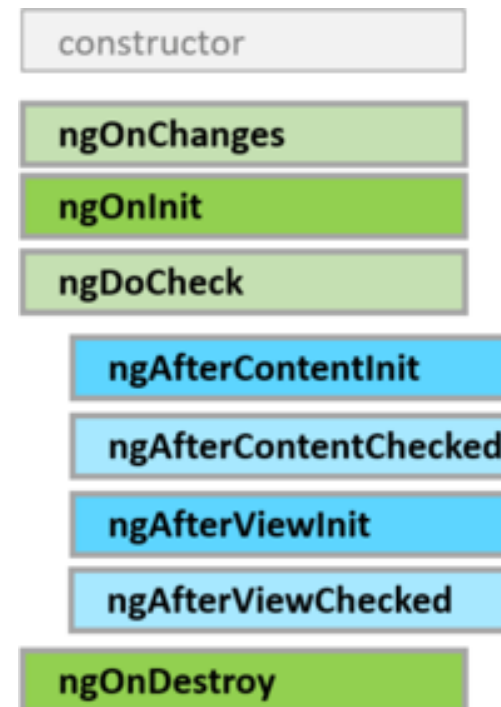






# CICLO DE VIDA DOS COMPONENTES

- ▶ Um componente tem um ciclo de vida gerenciado pelo Angular
- ▶ O Angular cria e renderiza um componente, cria e renderiza seus filhos, checa quando seus atributos (*data-bounded*) mudam e o destrói removendo-o da DOM





# EVENTOS DO CICLO DE VIDA

```
export class LifecycleComponent implements OnInit, OnDestroy,
  AfterViewInit {

  ngOnInit() {
    console.log('onInit');
  }

  ngOnDestroy() {
    console.log('onDestroy');
  }

  ngAfterViewInit() {
    console.log('afterViewInit');
  }
}
```

Ocorre após a  
montagem dos  
componentes na DOM

# Angular

## Diretivas





# DIRETIVAS

- ▶ Diretivas são instruções do Angular que mudam a aparência ou comportamento de um elemento DOM. Existem três tipos de diretivas:

**COMPONENTES**

**DIRETIVAS  
ESTRUTURAIS**

**DIRETIVAS DE  
ATRIBUTOS**



# TIPOS DE DIRETIVAS

## DIRETIVAS ESTRUTURAIS

Interagem com a view e modificam a estrutura do DOM e/ou código HTML

\*ngIf  
\*ngFor  
ngSwitch  
\*ngSwitchCase  
\*ngSwitchDefault

## DIRETIVAS DE ATRIBUTOS

Mudam a aparência ou comportamento de um elemento DOM ou componente

ngClass  
ngStyle  
Criadas pelo usuário



## DIRETIVAS ESTRUTURAIS - \*NGIF

```
<div *ngIf="curso !== null">  
  <label>Nome:</label>  
  <p>{{ curso.nome }}</p>  
  <label>Descrição:</label>  
  <p>{{ curso.descricao }}</p>  
</div>
```

```
<div *ngIf="!curso">  
  <h2>Carregando curso...</h2>  
</div>
```



## DIRETIVAS ESTRUTURAIS - \*NGIF - ELSE

```
<div *ngIf="curso !== null; else loading">  
  <label>Nome:</label>  
  <p>{{ curso.nome }}</p>  
  <label>Descrição:</label>  
  <p>{{ curso.descricao }}</p>  
</div>
```

```
<ng-template #loading>  
  <h2>Carregando curso...</h2>  
</ng-template>
```



# DIRETIVAS ESTRUTURAIS - NGSWITCH

```
<select #modo ngModel>
  <option value="">Selecione uma opção</option>
  <option value="mapa">Mapa</option>
  <option value="lista">Lista</option>
</select>
```

```
<div [ngSwitch]="modo.value">
  <p *ngSwitchCase="'mapa'">Modo mapa ativado</p>
  <p *ngSwitchCase="'lista'">Modo lista ativado</p>
  <p *ngSwitchDefault>Modo padrão ativado</p>
</div>
```





# DIRETIVAS ESTRUTURAIS - \*NGFOR

```
<ul>  
  <li *ngFor="let nome of nomes">{{ nome }}</li>  
</ul>
```

```
<ul>  
  <li *ngFor="let curso of cursos, let i = index">  
    {{ i + 1 }}: Nome: {{ curso.nome }}, Desc: {{ curso.descricao }}  
  </li>  
</ul>
```

```
<select>  
  <option *ngFor="let uf of listarUfs()">  
    {{ uf }}  
  </option>  
</select>
```



## EXERCÍCIO 10

1. Criar o componente PedidosComponent que contenha um `<select>` com 5 tipos de produto e um `<input>` para a quantidade a ser comprada,
2. Incluir um `<select>` contendo a forma de pagamento (boleto-10%, cartão-5%) e calcular o desconto sobre o produto de acordo com a forma escolhida.
3. Crie um arquivo contendo arrays de objetos (nome e valor) para a definição dos produtos e formas de pagamento.
4. Criar uma classe Pedido que contenha tipo do produto e quantidade a ser comprada e a forma de pagamento (boleto ou cartão). (dica: armazene apenas os índices dos arrays mencionados anteriormente).
5. O valor total do pedido pode ser implementado como um método da classe Pedido (Dica: declare como um método *getter*)



## EXERCÍCIO 10 (cont.)

6. Armazenar os pedidos em um *array* e exibí-los em uma tabela do Bootstrap (<https://getbootstrap.com/docs/4.0/content/tables/>).
7. Usar Event Binding para enviar dados do *template* para o componente



## EXERCÍCIO 11 (extraclasses)

1. Criar um componente FarmaciaComponent que contenha campos `<input>` para o código, nome do medicamento e preço de custo
2. Incluir um `<select>` para o tipo de medicamento (1-Genérico, 2-Marca)
3. Incluir um `<select>` para o laboratório (1-Bayer, 2-LAFEPE, 3-Pfizer)
4. Crie um arquivo contendo arrays de objetos (nome e valor) para a definição dos tipos de medicamento e laboratórios
5. Criar uma classe Produto que contenha código, nome, tipo de medicamento, laboratório e preço de custo.
6. Cadastrar os produtos e armazená-los em um array. Apresentar os dados em uma tabela do Bootstrap (<https://getbootstrap.com/docs/4.0/content/tables/>)



## EXERCÍCIO 11 (extraclasses) (cont.)

7. Usar Event Binding para enviar dados do template para o componente
8. Exibir para cada produto, seu preço de venda. Este valor será o preço de custo acrescido do percentual de lucro de acordo com a tabela abaixo (dica: declare como um método *getter*) :

+ - - - - - +	+ - - - - +	+ - - - - +	+ - - - - +	
	LAFEPE	Pfizer	Bayer	
Genérico	10%	15%	20%	
Marca	20%	35%	45%	
+ - - - - - +	+ - - - - +	+ - - - - +	+ - - - - +	

# Angular

## Formulários





# FORMULÁRIOS

▶ Em Angular, os formulários podem ser desenvolvidos usando duas estratégias diferentes:

- Template-Driven Forms

(<https://blog.thoughttram.io/angular/2016/03/21/template-driven-forms-in-angular-2.html>)

- Reactive/Model-Driven Forms

(<https://blog.thoughttram.io/angular/2016/06/22/model-driven-forms-in-angular-2.html>)



# FORMULÁRIOS

## ► *Template-driven Forms ou Reactive/Model-Driven Forms?*

*Neither is "better". They're two different architectural paradigms, with their own strengths and weaknesses. Choose the approach that works best for you. You may decide to use both in the same application.*

<https://angular.io/guide/reactive-forms#which-is-better-reactive-or-template-driven>





# FORMULÁRIOS

## ► Porém ...

*Reactive forms are synchronous. Template-driven forms are asynchronous. It's a difference that matters.*

*In reactive forms, you create the entire form control tree in code. You can immediately update a value or drill down through the descendents of the parent form because all controls are always available. Template-driven forms delegate creation of their form controls to directives. (...)*

*The asynchrony of template-driven forms also complicates unit testing. You must wrap your test block (...) to avoid looking for values in the form that aren't there yet. With reactive forms, everything is available when you expect it to be.*

<https://angular.io/guide/reactive-forms#async-vs-sync>



# FORMULÁRIOS

► Porém...

*(...) One advantage of working with form control objects directly is that value and validity updates are always synchronous and under your control. You won't encounter the timing issues that sometimes plague a template-driven form and reactive forms can be easier to unit test.*

<https://angular.io/guide/reactive-forms#reactive-forms-1>



# TEMPLATE-DRIVEN FORMS

- ▶ Para utilizar a API de Forms, é necessário importar o módulo `FormsModule` no arquivo `app.module.ts` do projeto:

```
import { FormsModule } from '@angular/forms';
```

```
@NgModule({  
  imports: [  
    BrowserModule,  
    FormsModule,  
    HttpClientModule  
  ],  
  ...  
})  
export class AppModule { }
```



# TEMPLATE-DRIVEN FORMS

- ▶ Quando trabalhamos com formulários em HTML, utilizamos a tag `<form>` delimitando os campos de entrada. Nada especial até aqui.

```
<form>  
  <label>Código:</label>  
  <input type="text">  
  
  <label>Título:</label>  
  <input type="text">  
  
  <label>Descrição:</label>  
  <input type="text">  
  
  <button type="submit">Submit</button>  
</form>
```



# TEMPLATE-DRIVEN FORMS

- ▶ Para o Angular gerenciar o formulário, utilizamos a diretiva ngForm associada à tag <form>. Esta diretiva fornece informações sobre o estado do formulário, incluindo sua validade e uma representação dos dados em JSON.

```
<form #form="ngForm">  
  <label>Código:</label>  
  <input type="text">  
  
  ...  
  
  <button type="submit">Submit</button>  
</form>
```



# TEMPLATE-DRIVEN FORMS

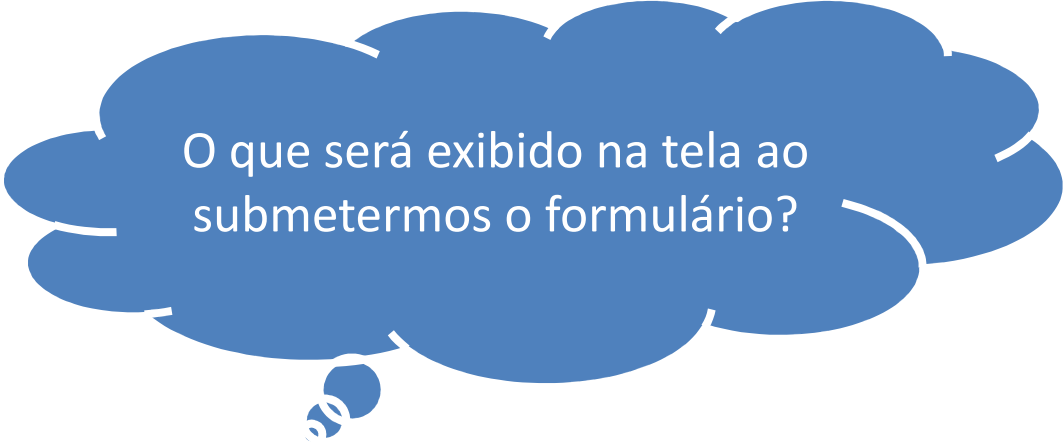
- ▶ Para submeter o formulário para processamento, utilizamos a diretiva `ngSubmit`, associando a um “*callback*” no componente.

```
<form #form="ngForm" (ngSubmit)="onSubmit(form.value)">  
  <label>Código:</label>  
  <input type="text">  
  
  <label>Título:</label>  
  <input type="text">  
  
  <label>Descrição:</label>  
  <input type="text">  
  
  <button type="submit">Submit</button>  
</form>
```



# TEMPLATE-DRIVEN FORMS

```
@Component({  
  selector: 'app-tdform-component',  
  ...  
})  
class TDFormComponent {  
  
  onSubmit(value: any) {  
    console.log(value);  
  }  
}
```



O que será exibido na tela ao submetermos o formulário?



# TEMPLATE-DRIVEN FORMS

- ▶ Para o Angular controlar os campos do formulário utilizamos a diretiva `ngModel`, em combinação com o atributo *“name”*. Ao registrarmos cada campo com `ngModel`, automaticamente os valores do formulário passarão a ser exibidos:

```
<form #form="ngForm" (ngSubmit)="onSubmit(form.value)">
  <label>Código:</label>
  <input type="text" name="codigo" ngModel>

  <label>Título:</label>
  <input type="text" name="titulo" ngModel>
  ...
```





# TEMPLATE-DRIVEN FORMS

...

```
<label>Descrição:</label>
```

```
<input type="text" name="descricao" ngModel>
```

```
<button type="submit">Submit</button>
```

```
</form>
```



# TEMPLATE-DRIVEN FORMS

- ▶ Quando é necessário agrupar os campos de entrada, utilizamos a diretiva `ngModelGroup`:

```
<label>Codigo:</label>
<input type="text" name="codigo" ngModel>

<fieldset ngModelGroup="dados">
  <label>Título:</label>
  <input type="text" name="titulo" ngModel>

  <label>Descrição:</label>
  <input type="text" name="descricao" ngModel>
</fieldset>
```



# TEMPLATE-DRIVEN FORMS

- ▶ Com `ngModel`, podemos realizar o *binding* de propriedades. Isto é bastante útil, por exemplo, em um cenário onde é necessário inicializar os campos de entrada com valores *default*.

```
<label>Codigo:</label>
<input type="text" name="codigo" [ngModel]="codigo">

<fieldset ngModelGroup="dados">
  <label>Título:</label>
  <input type="text" name="titulo" [ngModel]="titulo">

  <label>Descrição:</label>
  <input type="text" name="descricao" [ngModel]="descricao">
</fieldset>
```



# TEMPLATE-DRIVEN FORMS

```
@Component({  
  selector: 'app-tdform-component',  
  ...  
})  
class TDFormComponent {
```

```
  codigo = '100';  
  titulo = 'TypeScript e Angular 2';  
  descricao = 'Curso de Angular 2 - UNISERPRO';
```

```
  onSubmit(value: any) {  
    console.log(value);  
  }  
}
```



# TEMPLATE-DRIVEN FORMS

- ▶ Uma otimização comum para formulários consiste em não associar individualmente os campos de entrada com variáveis espalhadas no componente.
- ▶ Podemos criar um objeto e associar seus atributos aos campos de entrada do formulário:



# TEMPLATE-DRIVEN FORMS

```
export class Dados {  
    constructor(public titulo: string,  
                public descricao: string) { }  
}
```

```
export class Curso {  
    constructor(public codigo: string,  
                public dados: Dados) { }  
}
```



# TEMPLATE-DRIVEN FORMS

```
@Component({  
  selector: 'app-tdform-component',  
  ...  
})  
class TDFormComponent {
```

```
  curso = new Curso('100', new Dados('Angular', 'Curso de Angular  
    - UNISERPRO'));
```

```
  ...  
}
```



# TEMPLATE-DRIVEN FORMS

```
<label>Codigo:</label>
```

```
<input type="text" name="codigo" [ngModel]="curso.codigo">
```

```
<fieldset ngModelGroup="dados">
```

```
  <label>Título:</label>
```

```
  <input type="text" name="titulo" [ngModel]="curso.dados.titulo">
```

```
  <label>Descrição:</label>
```

```
  <input ... name="descricao" [ngModel]="curso.dados.descricao">
```

```
</fieldset>
```





## EXERCÍCIO 12

1. Criar o TDPedidosComponent, para implementar o Exercício 10 (PedidosComponent) através de um formulário *Template-Driven*.



# MODEL-DRIVEN FORMS

- ▶ Enquanto a estratégia *Template-Driven* permite a criação de formulários com pouco código JavaScript (e mais HTML), na estratégia *Reactive/Model-Driven*, os formulários são gerados de forma imperativa, onde o objetivo é criar um modelo do formulário no componente para representar a estrutura do DOM.
- ▶ Desta forma, é possível testá-los através de *frameworks* de testes unitários como o Jasmine (<https://github.com/jasmine/jasmine>) e o Karma (<https://karma-runner.github.io>)



# MODEL-DRIVEN FORMS

- ▶ Para utilizar a API de ReactiveForms, é necessário importar o módulo `ReactiveFormsModule` no arquivo `app.module.ts`:

```
import { ReactiveFormsModule } from '@angular/forms';
```

```
@NgModule({  
  imports: [  
    BrowserModule,  
    ReactiveFormsModule,  
    HttpClientModule  
  ],  
  ...  
})  
export class AppModule { }
```



# MODEL-DRIVEN FORMS

- ▶ Usaremos o mesmo formulário como exemplo:

```
<form>
  <label>Codigo:</label>
  <input type="text">

  <fieldset>
    <label>Título:</label>
    <input type="text">

    <label>Descrição:</label>
    <input type="text">
  </fieldset>

  <button type="submit">Submit</button>
</form>
```



# MODEL-DRIVEN FORMS

- ▶ Para o Angular gerenciar um formulário usando a estratégia *Reactive/Model-Driven*, utilizamos os elementos FormGroup e FormControl da API de Forms

```
import { FormGroup, FormControl } from '@angular/forms';
```

```
export class MDFormComponent {  
  form = new FormGroup({  
    codigo: new FormControl(),  
    dados: new FormGroup({  
      titulo: new FormControl(),  
      descricao: new FormControl()  
    })  
  });  
}
```



# MODEL-DRIVEN FORMS

```
<form [formGroup]="form">
  <label>Código:</label>
  <input type="text" formControlName="codigo">

  <fieldset formGroupName="dados">
    <label>Título:</label>
    <input type="text" formControlName="titulo">

    <label>Descrição:</label>
    <input type="text" formControlName="descricao">
  </fieldset>

  <button type="submit">Submit</button>
</form>
```



# MODEL-DRIVEN FORMS

- Podemos associar um evento à submissão do formulário:

```
<form [formGroup]="form" (ngSubmit)="onSubmit(form.value)">  
  ...  
</form>
```

```
<form [formGroup]="form">  
  ...  
  <button type="submit" (click)="onSubmit(form.value)"></button>  
</form>
```

```
<form [formGroup]="form">  
  ...  
  <button type="button" (click)="onSubmit()"></button>  
</form>
```

O conteúdo do formulário já se encontra no componente! 😊



# MODEL-DRIVEN FORMS

- Podemos desativar o botão de submissão caso o formulário esteja inválido. Para isto usamos a propriedade “*valid*” do FormGroup:

```
<form [formGroup]="form">  
  ...  
  <button type="button" [disabled]="!form.valid"  
    (click)="onSubmit()">Submit</button>  
</form>
```

```
this.form = ...  
dados: new FormGroup({  
  titulo: new FormControl(null, Validators.required),  
  descricao: new FormControl(null, Validators.required)  
})
```





# MODEL-DRIVEN FORMS

- ▶ Adicionaremos também um botão de *reset* de formulário:

```
<form [formGroup]="form">  
  ...  
  <button type="button" (click)="onReset()">Reset</button>  
</form>
```

```
ngOnInit() {  
  this.buildForm();  
}  
buildForm() {  
  this.form = ...  
}  
onReset() {  
  this.buildForm();  
}
```



# MODEL-DRIVEN FORMS

- ▶ Uma forma menos “verbosa” para criar formulários Model/Driven é através do uso de um FormBuilder:

```
export class MDFormComponent {  
  form: FormGroup;  
  constructor(private FormBuilder: FormBuilder) { }  
  ngOnInit() {  
    this.form = this.formBuilder.group({  
      codigo: null,  
      dados: this.formBuilder.group({  
        titulo: null,  
        descricao: null  
      })  
    });  
  }  
}
```



# MODEL-DRIVEN FORMS

- ▶ Acrescentando validadores ao nosso modelo:

```
ngOnInit() {  
  this.form = this.formBuilder.group({  
    codigo: null,  
    dados: this.formBuilder.group({  
      titulo: [null, Validators.required],  
      descricao: [null,  
        [Validators.required, Validators.minLength(5)]]  
    })  
  });  
}
```



# MODEL-DRIVEN FORMS

```
<form [formGroup]="form">
  ...
  <fieldset formGroupName="dados">
    <label>Título:</label>
    <input type="text" formControlName="titulo">
    <p *ngIf="form.controls.dados.controls.titulo.touched &&
      form.controls.dados.controls.titulo.errors">
      Campo obrigatório!
    </p>
    ...
  </fieldset>
</form>
```



# MODEL-DRIVEN FORMS

- ▶ Podemos criar validadores customizados para os formulários, bastando criar funções com a assinatura a seguir:

```
interface ValidatorFn {  
    (c: AbstractControl): ValidationErrors | null  
}
```

- ▶ Exemplo:

```
function validarCodigo(c: AbstractControl): ValidationErrors | null {  
    return (+c.value < 100) ? null : { vlInvalido : true };  
}
```



# MODEL-DRIVEN FORMS

```
ngOnInit() {  
  this.form = this.formBuilder.group({  
    codigo: [null, validarCodigo],  
    dados: this.formBuilder.group({  
      titulo: [null, Validators.required],  
      descricao: [null,  
        [Validators.required, Validators.minLength(5)]]  
    })  
  });  
}
```



# MODEL-DRIVEN FORMS

- ▶ Para passar parâmetros aos validadores, precisamos criar 'factories', uma vez que não podemos mudar a assinatura da função de validação:

```
function validarCodigoFactory(max: number) {  
  return (c: AbstractControl): ValidationErrors|null => {  
    return (+c.value < max) ? null : { vlInvalido : true };  
  }  
}
```



# MODEL-DRIVEN FORMS

```
ngOnInit() {  
  this.form = this.formBuilder.group({  
    codigo: [null, validarCodigoFactory(100)],  
    dados: this.formBuilder.group({  
      titulo: [null, Validators.required],  
      descricao: [null,  
        [Validators.required, Validators.minLength(5)]]  
    })  
  });  
}
```





## EXERCÍCIO 13

1. Criar o MDPedidosComponent, para implementar o Exercício 10 (PedidosComponent) através de um formulário *Reactive/Model-Driven*, usando um FormBuilder.
2. Crie uma função 'factory' que retorne um validador para validar se o campo quantidade é informado entre 0 e 50.



## EXERCÍCIO 14 (extraclasse)

1. Alterar o Exercício 11 (FarmaciaComponent) para incluir um formulário utilizando a estratégia *Reactive/Model-Driven*.

# Angular

## Serviços





# SERVIÇOS

## COMPONENTE 1

manipularDados()

fazerAlgumaCoisa()

## COMPONENTE 2

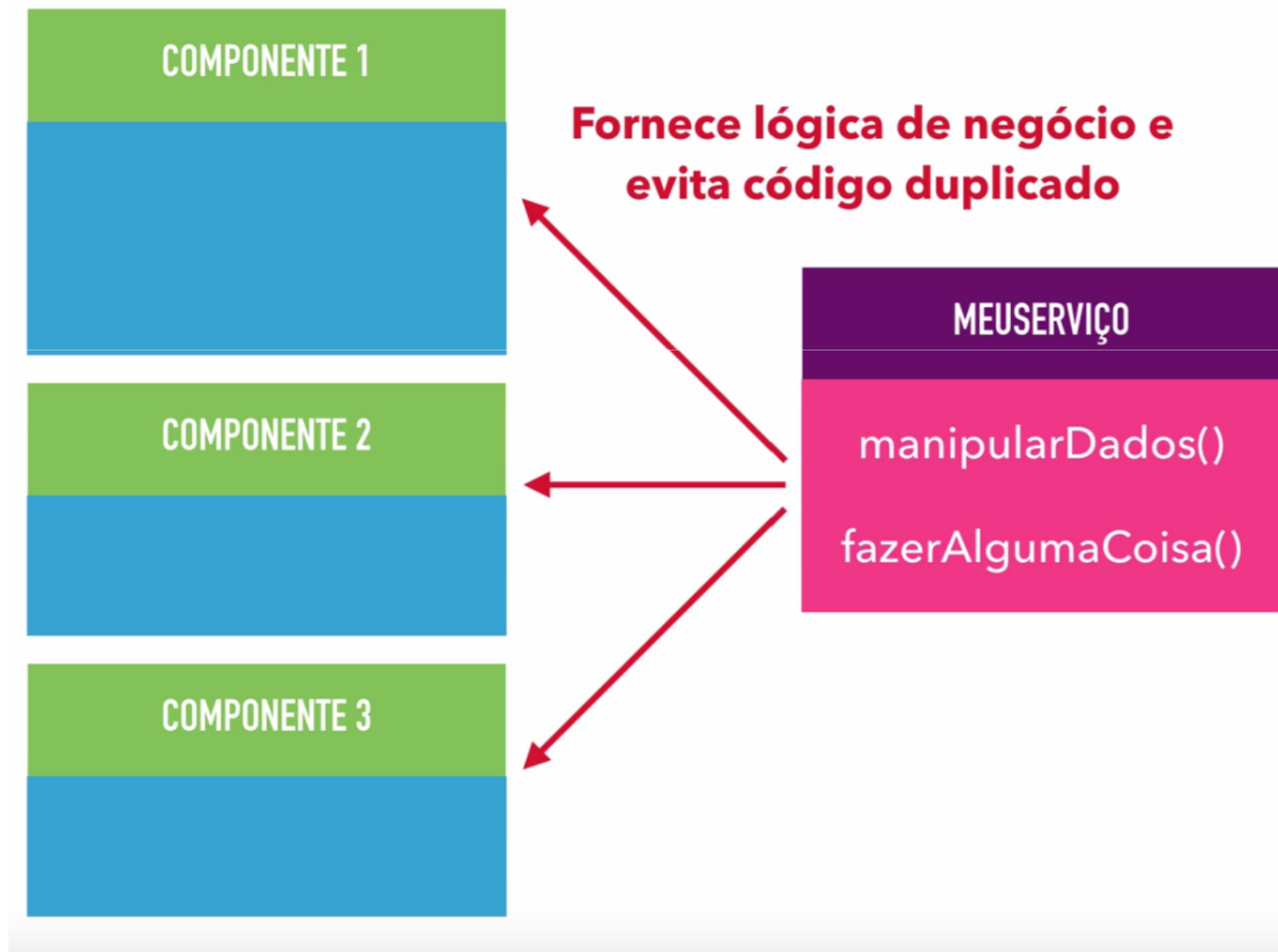
fazerAlgumaCoisa()

## COMPONENTE 3

fazerAlgumaCoisa()

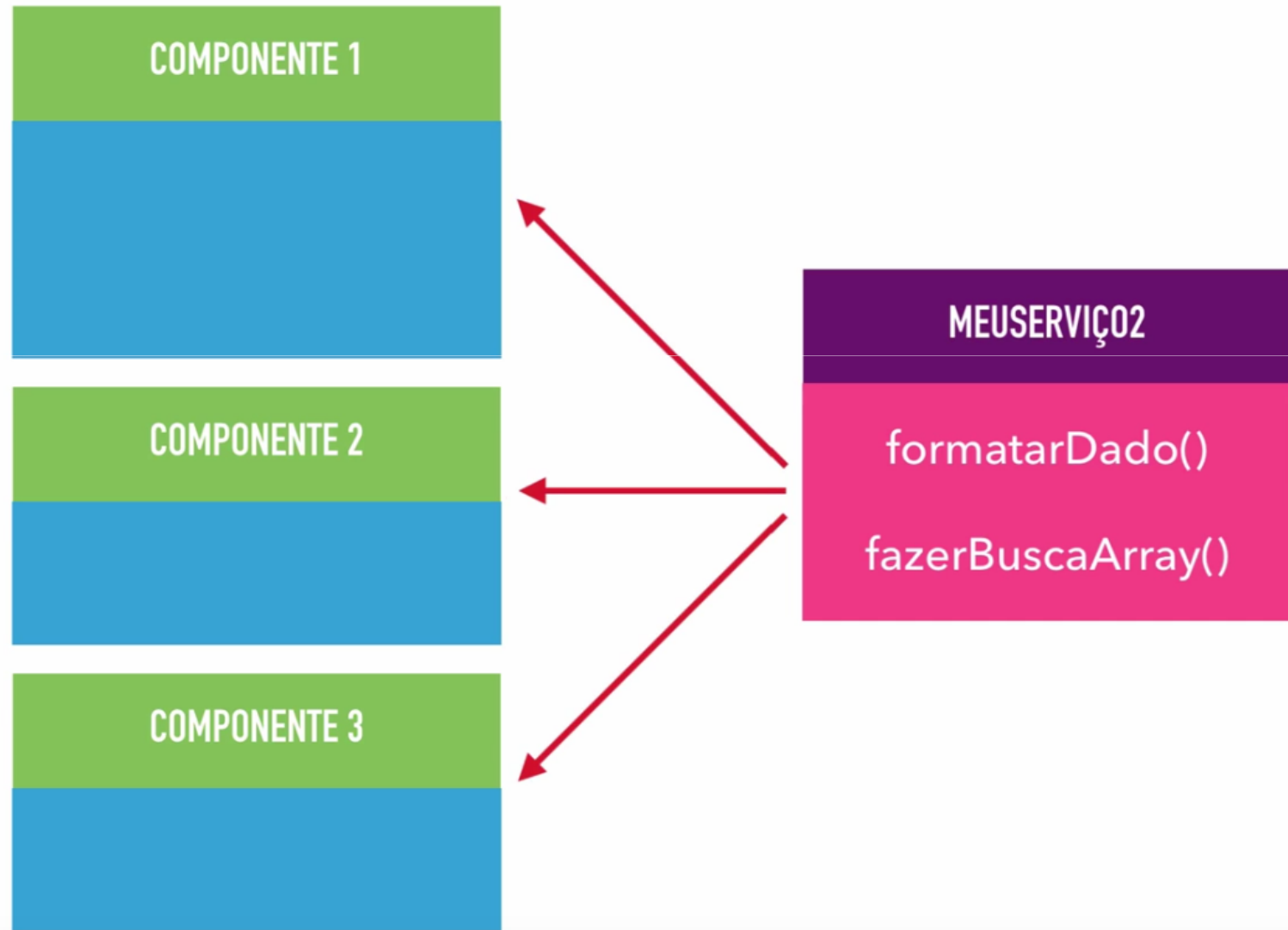


# SERVIÇOS: LÓGICA DE NEGÓCIO





# SERVIÇOS: CLASSES UTILITÁRIAS





# SERVIÇOS

```
export class CursosComponent implements OnInit {  
  
  cursos: string[] = ['Angular', 'Java', 'JavaScript'];  
  
  constructor() { }  
  
  ngOnInit() {  
  }  
}
```

```
<h5>Lista de Cursos</h5>  
<ul>  
  <li *ngFor="let curso of cursos">  
    {{ curso }}  
  </li>  
</ul>
```



# SERVIÇOS

- ▶ Vamos criar um serviço para recuperar a lista de cursos utilizada no componente

```
export class CursosService {  
  
  getCursos() {  
    return ['Angular', 'Java', 'Javascript'];  
  }  
}
```





# SERVIÇOS

```
export class CursosComponent implements OnInit {  
  
  cursos: string[] = [];  
  cursosService: CursosService;  
  
  constructor() {  
    this.cursosService = new CursosService();  
  }  
  
  ngOnInit() {  
    this.cursos = this.cursosService.getCursos();  
  }  
}
```



# SERVIÇOS

```
export class CursosComponent implements OnInit {  
  
  cursos: string[] = [];  
  cursosService: CursosService;  
  
  constructor() {  
    this.cursosService = new CursosService();  
  }  
  
  ngOnInit() {  
    this.cursos = this.cursosService.getCursos();  
  }  
}
```



## SERVIÇOS

- ▶ A classe CursosService pode depender de outras classes (ex.: HttpClient), e assim por diante
- ▶ Controlar a criação dos objetos não é uma boa ideia
- ▶ Em Angular, existe um mecanismo de injeção de dependências para resolver esta questão

# Angular

## Injeção de Dependências





# O QUE É DEPENDÊNCIA?

**Classe1 precisa da OutraClasse para funcionar**





# INJETANDO DEPENDÊNCIAS NA CLASSE

**Significa que não deveríamos instanciar  
a OutraClasse manualmente**





# INJEÇÃO DE DEPENDÊNCIAS

- ▶ Utiliza-se a anotação `@Injectable()`
- ▶ O Angular CLI já cria um serviço com esta anotação:  

```
$ ng g s cursos
```
- ▶ Serviços devem ser “providos” em algum momento. Isto pode ocorrer a partir do início da aplicação (módulo principal – ‘root’), a partir da criação de um sub-módulo, ou a partir da criação de um determinado componente
- ▶ Para tal, utiliza-se a seção “providers” das anotações `@NgModule` e `@Component`, ou a propriedade “providedIn” da anotação `@Injectable`



# INJEÇÃO DE DEPENDÊNCIAS

- Configurando na anotação @NgModule ou @Component

```
@NgModule({  
  ...  
  providers: [ CursosService ]  
})  
export class AppModule { }
```

```
@Component({  
  ...  
  providers: [ CursosService ]  
})  
export class CursosComponent implements OnInit { ... }
```





# INJEÇÃO DE DEPENDÊNCIAS

- ▶ Configurando na anotação `@Injectable()` manualmente ou de forma automática com a opção `--module=<nome-modulo>`

```
$ ng g s cursos --module=app
```



# INJEÇÃO DE DEPENDÊNCIAS

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

```
@Injectable({  
  providedIn: 'root'  
})
```

```
export class CursosService {
```

```
  constructor() { }
```

```
}
```



# INJEÇÃO DE DEPENDÊNCIAS

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

```
@Injectable({  
  providedIn: 'root'  
})
```

```
export class CursosService {
```

```
  constructor() { }
```

```
  getCursos(): string[] {  
    return ['Angular', 'Java', 'Javascript'];  
  }
```

```
}
```



# INJEÇÃO DE DEPENDÊNCIAS

```
export class CursosComponent implements OnInit {  
  
  cursos: string[] = [];  
  cursosService: CursosService;  
  
  constructor(_cursosService: CursosService) {  
    //this.cursosService = new CursosService();  
    this.cursosService = _cursosService;  
  }  
  
  ngOnInit() {  
    this.cursos = this.cursosService.getCursos();  
  }  
}
```



# INJEÇÃO DE DEPENDÊNCIAS

```
export class CursosComponent implements OnInit {  
  
  cursos: string[] = [];  
  //cursosService: CursosService;  
  
  constructor(private cursosService: CursosService) {  
    //this.cursosService = new CursosService();  
    //this.cursosService = _cursosService;  
  }  
  
  ngOnInit() {  
    this.cursos = this.cursosService.getCursos();  
  }  
}
```



# INJEÇÃO DE DEPENDÊNCIAS

```
export class CursosComponent implements OnInit {  
  
  cursos: string[] = [];  
  
  constructor(private cursosService: CursosService) { }  
  
  ngOnInit() {  
    this.cursos = this.cursosService.getCursos();  
  }  
}
```



## EXERCÍCIO 15

1. Criar um serviço que forneça uma funcionalidade para recuperar a lista (*array*) de pedidos. Este serviço deve ainda prover funcionalidades para inclusão e exclusão de registros.
2. Modificar o Exercício 12 (PedidosComponent) para incluir o serviço criado na solução.
3. Incluir a funcionalidade de exclusão de registros na tabela de pedidos.



## EXERCÍCIO 16 (extraclasse)

1. Criar um serviço que forneça uma funcionalidade para recuperar a lista (*array*) de produtos. Este serviço deve ainda prover funcionalidades para inclusão e exclusão de registros.
2. Modificar o Exercício 14 (FarmaciaComponent) para incluir o serviço criado na solução.
3. Incluir a funcionalidade de exclusão de registros da tabela de produtos.



# Angular

## Roteamento





SPA

# Single Page Applications



# COMO FUNCIONA O ROTEAMENTO

<http://meuprojeto.com.br/usuarios>



# COMO FUNCIONA O ROTEAMENTO

<http://meuprojeto.com.br/usuarios>



**ListaUsuariosComponent**



# COMO FUNCIONA O ROTEAMENTO

<http://meuprojeto.com.br/usuarios/2/edit>



# COMO FUNCIONA O ROTEAMENTO

<http://meuprojeto.com.br/usuarios/2/edit>



**Angular 2 lê a rota**

**Angular 2 identifica a  
rota**



# COMO FUNCIONA O ROTEAMENTO

**`/usuarios/2/edit`**



# COMO FUNCIONA O ROTEAMENTO

**/usuarios/2/edit**

**/usuarios**

**ListaUsuariosComponent**





# COMO FUNCIONA O ROTEAMENTO

**/usuarios/2/edit**

**/usuarios**

**ListaUsuariosComponent**

**/usuarios/:id**

**UsuarioDetalhesComponent**



# COMO FUNCIONA O ROTEAMENTO

**/usuarios/2/edit**

**/usuarios**

**ListaUsuariosComponent**

**/usuarios/:id**

**UsuarioDetalhesComponent**

**/usuarios/:id/edit**

**UsuarioFormComponent**



# app.routing.ts

- ▶ Para utilizar roteamento no projeto, precisamos criar um arquivo para conter as definições:

```
import { ModuleWithProviders } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { HomeComponent } from '../home/home.component';
import { CursosComponent } from '../cursos/cursos.component';
import { LoginComponent } from '../login/login.component';

const APP_ROUTES: Routes = [
  { path: 'cursos', component: CursosComponent },
  { path: 'home', component: HomeComponent },
  { path: '', redirectTo: '/home', pathMatch: 'full' }
];

export const AppRouting: ModuleWithProviders =
  RouterModule.forRoot(APP_ROUTES);
```



# app.module.ts

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

```
import { AppComponent } from './app.component';  
import { AppRoutingModule } from './app.routing';
```

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



# app.component.html

```
<ul>
  <li><a routerLink="/home">Home</a></li>
  <li><a routerLink="/cursos">Cursos</a></li>
</ul>

<div class="container">
  <router-outlet></router-outlet>
</div>
```



# PARÂMETROS DE ROTEAMENTO

- ▶ Podemos utilizar parâmetros na definição das rotas do sistema:

```
const APP_ROUTES: Routes = [  
  { path: 'cursos', component: CursosComponent },  
  { path: 'cursos/:id', component: CursoDetalheComponent },  
  { path: 'home', component: HomeComponent },  
  { path: '', redirectTo: '/home', pathMatch: 'full' }  
];
```



# PARÂMETROS DE ROTEAMENTO

```
<a [routerLink]="['/cursos', idCurso]">Curso #{{idCurso}}</a>
```

```
export class CursosComponent implements OnInit {  
  
    idCurso: string = '1';  
}
```



## curso-detache.component.ts

```
import { Subscription } from 'rxjs';

export class CursoDetalheComponent implements OnInit, OnDestroy {

  id: string;
  inscricao: Subscription;

  constructor(private route: ActivatedRoute) { }

  ngOnInit() {
    this.inscricao = this.route.params.subscribe(
      params => {
        this.id = params['id'];
      });
  }

  ngOnDestroy() {
    this.inscricao.unsubscribe();
  }
}
```





# REDIRECIONAMENTO VIA CÓDIGO

```
constructor(private route: ActivatedRoute,  
             private router: Router,  
             private cursosService: CursosService) {  
  
}  
  
ngOnInit() {  
  this.inscricao = this.route.params.subscribe(  
    params => {  
      this.id = params['id'];  
  
      this.curso = this.cursosService.getCurso(this.id);  
  
      if(this.curso === null) {  
        this.router.navigate(['/naoEncontrado']);  
      }  
    });  
}
```



# REDIRECIONAMENTO VIA CÓDIGO

```
const APP_ROUTES: Routes = [  
  { path: 'cursos', component: 'CursosComponent' },  
  { path: 'cursos/:id', component: 'CursoDetalheComponent' },  
  { path: 'home', component: 'HomeComponent' },  
  { path: 'naoEncontrado', component: 'NaoEncontradoComponent' },  
  { path: '', redirectTo: '/home', pathMatch: 'full' },  
];
```



## EXERCÍCIO 17

1. Criar um arquivo de configuração de roteamento (caso não exista), com o objetivo de incluir as rotas dos componentes PedidosComponent, MainComponent e FerramentasComponent
2. Incluir a tag `<router-outlet></router-outlet>` no template do AppComponent a fim de exibir os componentes
3. Incluir um menu (<https://getbootstrap.com/docs/4.0/components/navbar/>) para conter os links (usando a diretiva RouterLink) para os componentes incluídos anteriormente.



## EXERCÍCIO 18

1. Incluir um método no PedidosService para recuperar um dado pedido
2. Criar um componente PedidosDetalheComponent que deve exibir os detalhes de um dado pedido. O componente deve receber o código do pedido através do parâmetro obtido da rota.
3. Injetar o PedidosService no PedidosDetalheComponent.
4. Incluir uma rota para o PedidosDetalheComponent no arquivo de configuração (app.routing.ts)
5. Incluir na listagem de pedidos do PedidosComponent, a opção de detalhar cada um dos registros.

# Angular

## Integração com Servidor





# INTEGRAÇÃO COM SERVIDOR

- ▶ A integração da aplicação Angular com um *backend* REST é realizada de forma assíncrona
- ▶ Utilizamos a classe **HttpClient** para realizar as requisições HTTP (GET, POST, PUT, DELETE, ...)
- ▶ A classe Observable (da biblioteca RxJS) é usada para “escutar” quando a requisição foi retornada com os dados de resposta



# INTEGRAÇÃO COM SERVIDOR

- ▶ Para utilizar a API de HttpClient, é necessário importar o módulo HttpClientModule no arquivo app.module.ts:

```
import { HttpClientModule } from '@angular/common/http';
```

```
@NgModule({  
  imports: [  
    BrowserModule,  
    HttpClientModule  
  ],  
  ...  
})  
export class AppModule { }
```



# cursos.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class CursosService {

  constructor(private http: HttpClient): { }

  getCursos(): Observable<Curso[]> {
    return this.http
      .get<Curso[]>('http://<ip:porta>/api/listar');
  }
}
```





# cursos.component.ts

```
export class CursosComponent implements OnInit {  
  
  cursos: Curso[] = [];  
  
  constructor(private service: CursosService) { }  
  
  ngOnInit() {  
    this.service.getCursos().subscribe(value => {  
      this.cursos = value;  
    },  
    error => {  
      alert('Erro do servidor durante a consulta de cursos!');  
    });  
  }  
}
```



## cursos.component.ts

```
<ul>
  <li *ngFor="let curso of cursos">
    {{curso.id}} - {{curso.descricao}}
  </li>
</ul>
```



# INTEGRAÇÃO COM SERVIDOR

- ▶ É possível configurar o tipo de retorno além do JSON (padrão) :

```
this.http.get<Curso[]>(this.url, { responseType: 'text' });
```

- ▶ Para acessar o 'response', configuramos a propriedade 'observe':

```
this.http.get<Curso[]>(this.url, { observe: 'response' });
```

...

```
this.service.getCursos().subscribe(response => {  
  this.type = response.headers.get('Content-Type');  
  this.cursos = <Curso[]>response.body;  
});
```



## EXERCÍCIO 19

1. Atualizar o CursosService para incluir os métodos listar e consultar que devem apontar para os serviços nas respectivas urls:

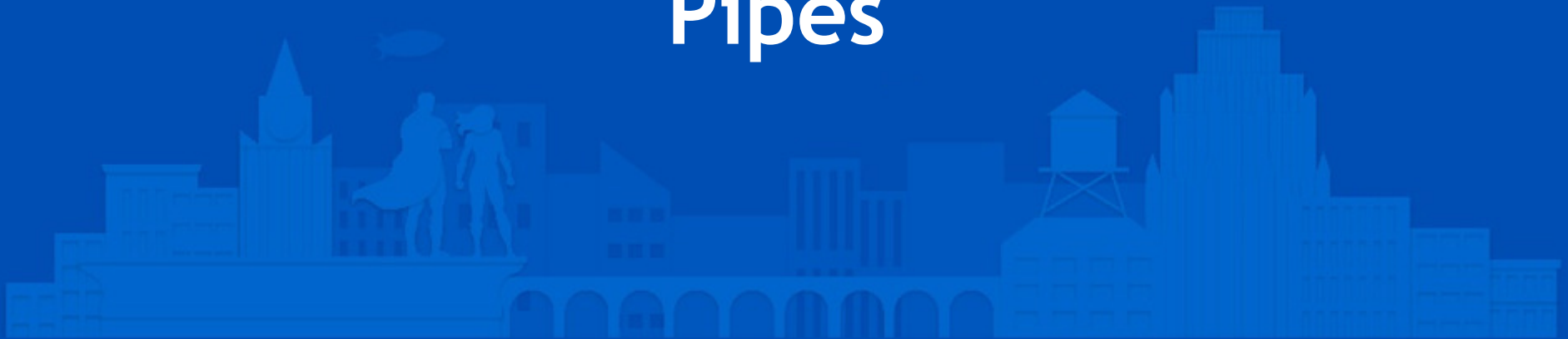
`http://<ip:porta>/api/cursos`

<http://<ip:porta>/api/cursos/id>

2. Atualizar o componente CursosComponent para que a listagem de cursos seja recuperada do serviço listar() do CursosService atualizado.
3. Criar o componente CursoDetalheComponent para apresentar o detalhamento do curso recuperado do serviço consultar() do CursosService, e acessado através da listagem de cursos.

# Angular

## Pipes





# PIPES

- ▶ Um Pipe permite transformar/formatar dados antes de exibí-los para o usuário
- ▶ Angular já conta com diversos 'built-in' pipes:
  - DatePipe, JsonPipe
  - UpperCasePipe, LowerCasePipe, TitleCasePipe
  - DecimalPipe, CurrencyPipe, PercentPipe
- ▶ Além destes, podemos criar nossos próprios pipes



# USANDO BUILT-IN PIPES

```
export class ExemplosPipesComponent implements OnInit {  
  
  livro: any = {  
    titulo: 'O Hobbit',  
    estrelas: 4.53524,  
    preco: 57.70,  
    lancamento: new Date(1937, 8, 21)  
  }  
  
  constructor(): { }  
  
  ngOnInit() {  
  }  
}
```



# USANDO BUILT-IN PIPES

<h5>Exemplos de Pipes</h5>

<p>Título: {{ livro.titulo | uppercase }}</p>

<p>Estrelas: {{ livro.rating | number:'1.1-1' }}</p>

<p>Preço: {{ livro.preco | currency:'BRL':'symbol' }}</p>

<p>Lançamento: {{ livro.lancamento | date:'dd/MMM/yyyy' }}</p>

<br>

<p>Livro: {{ livro | json }}</p>





# CRIANDO UM PIPE CUSTOMIZADO

► Criamos um Pipe através do Angular CLI

```
$ ng g p Capitalize
```



# CRIANDO UM PIPE CUSTOMIZADO

```
@Pipe({ name: 'capitalize'})
export class CapitalizePipe implements PipeTransform {

  transform(value: any, args?: any): any {
    let result = '';
    for (const v of value.split(' ')) {
      result += this.capitalize(v) + ' ';
    }
    return result;
  }

  private capitalize(value: string) {
    return value.substr(0, 1).toUpperCase() +
      value.substr(1).toLowerCase();
  }
}
```



# USANDO BUILT-IN PIPES

<h5>Exemplos de Pipes</h5>

<p>Título: {{ livro.titulo | capitalize }}</p>

<p>Estrelas: {{ livro.estrelas | number:'1.1-1' }}</p>

<p>Preço: {{ livro.preco | currency:'BRL':'symbol' }}</p>

<p>Lançamento: {{ livro.lancamento | date:'dd/MMM/yyyy' }}</p>

<br>

<p>Livro: {{ livro | json }}</p>



## EXERCÍCIO 20

1. Criar *pipes* apropriados para a exibição do produto, valor total (currency) e forma de pagamento nos componentes PedidosComponent e PedidoDetalheComponent.

# Angular

Angular CLI: Gerando a  
Build de Produção





# GERANDO A BUILD DE PRODUÇÃO

```
$ cd NomeProjeto
```

```
$ ng build --target=production --environment=prod
```

```
$ ng build --prod -env=prod
```

```
$ ng build --prod
```

Os três comandos são  
equivalentes!

# Angular

Atualizações (What's new)





# ANGULAR - ATUALIZAÇÕES

## ▶ Angular 4

- ▶ AOT – Builds menores e mais rápidas
- ▶ Compatibilidade com Typescript 2.1+
- ▶ Novo pipe titlecase, ngIf com else, mudanças na API do HTTP, ...
- ▶ <http://blog.ninja-squad.com/2017/03/24/what-is-new-angular-4/>

## ▶ Angular 5

- ▶ AOT – Compilador mais rápido que na versão 5
- ▶ Forms: atualização dos campos com `updateOn` (*change, blur, submit*)
- ▶ Nova API `HttpClient` (API `Http` tornou-se deprecated!)
- ▶ <https://blog.ninja-squad.com/2017/11/02/what-is-new-angular-5/>





# ANGULAR - ATUALIZAÇÕES

## ▶ Angular 6

- ▶ Novo renderer: Ivy (diminuição do tempo de comp. e tamanho da build)
- ▶ @Injectable – Novo atributo 'providedIn'
- ▶ RxJs 6
- ▶ <https://blog.ninja-squad.com/2018/05/04/what-is-new-angular-6/>

# Angular

## Projeto Final





# PROJETO FINAL

- ▶ A Secretaria de Educação quer um sistema para fazer um relatório sobre as escolas da cidade de Recife.
- ▶ Para cada escola são informados os seguintes dados: código da escola, nome, total de alunos matriculados, total de professores, e se possui laboratório de informática.
- ▶ A secretaria quer saber quanto deve ser liberado de verba para cada escola. O cálculo será feito da seguinte forma:
  - ▶ Para cada aluno da escola são liberados: R\$ 50



## PROJETO FINAL (cont.)

- ▶ Para cada professor são liberados: R\$ 400
- ▶ Se a escola não tem laboratório de Informática: R\$ 10.000
- ▶ Ex: uma escola com 200 alunos, com 15 professores e tem laboratório de informática:  $(50 \times 200) + (400 \times 15) + 10.000 = 10.000 + 6.000 + 10.000 = \text{R\$ } 26.000$



## PROJETO FINAL (cont.)

1. Desenvolver um sistema denominado EscolaAngular, para realizar o cadastro das escolas da cidade de Recife
2. O projeto deve considerar todos os conceitos aprendidos em sala de aula. Seguir os passos abaixo:
  - a) Criar a classe Escola para representar a entidade principal na subpasta *'app/shared/model'*. (`$ ng g class shared/model/escola`)
  - b) Desenvolver o serviço EscolaService para realizar chamadas REST no servidor (*backend*) com o objetivo de listar, consultar, incluir, alterar e excluir registros de escolas no banco de dados. Criar o serviço na subpasta *'app/shared/services'*. (`$ ng g service shared/services/escola`)



## PROJETO FINAL (cont.)

- c) Desenvolver o componente `ListarEscolaComponent` para a listagem das escolas. Apresentar o valor da verba liberada de acordo com cálculo mencionado anteriormente. Apresentar os dados em uma tabela <https://getbootstrap.com/docs/4.0/content/tables/> (`$ ng g c listar-escola`)
- d) Desenvolver o componente `EditarEscolaComponent` para a edição de uma escola, usando *Model-Driven Forms*. (`$ ng g c editar-escola`)
- e) Desenvolver o componente `DetalharEscolaComponent` para detalhar uma escola em um painel <https://getbootstrap.com/docs/4.0/components/card/>, usando sistema de layout em grid <https://getbootstrap.com/docs/4.0/layout/grid/> (`$ ng g c detalhar-escola`)



## PROJETO FINAL (cont.)

- f) Criar o arquivo `app.routing.ts` com as rotas (e regras) necessárias para os componentes criados.
- g) Desenvolver o componente `LoginComponent` para realizar o *login* do usuário (usuário: 'admin', password: 'serpro'). Criar regra de roteamento para autorizar o acesso aos componentes apenas se o usuário estiver autenticado, caso contrário, redirecionar para a tela de *login*. A tela inicial será a listagem de escolas (\*`Guards`)  
(`$ ng g c login`)
- h) Criar um *pipe* para apresentar os valores do campo 'possui lab. de inf.', como 'Sim' e 'Não' (\*`Pipes`) (`$ ng g p shared/pipes/ simNao`)
- i) Exibir as mensagens do sistema em um *alert* <https://getbootstrap.com/docs/4.0/components/alerts/>



## PROJETO FINAL (cont.)

- j) Criar um menu <https://getbootstrap.com/docs/4.0/components/navbar/> para acessar as funcionalidades do sistema: listar escolas e criação de uma nova escola.
  - k) Incluir um paginador para o componente de listagem de escolas <https://getbootstrap.com/docs/4.0/components/pagination/>. Definir o valor 5 para o tamanho de cada página.
3. Os nomes dos campos da entidade Escola esperados no servidor (*backend*) serão: id, nome, matriculados, professores, e labinformatica.
  4. O campo 'id' não deve ser editado na tela, pois trata-se da chave primária da tabela (auto-incrementada pelo MySQL)





## PROJETO FINAL (cont.)

### 5. URLs dos serviços REST do backend:

- a) <http://<ip:porta>/api/listar> - Listar Escolas (GET)
- b) <http://<ip:porta>/api/consultar/id> - Consultar Escola (GET)
- c) <http://<ip:porta>/api/incluir> - Incluir Escola (POST - os dados são passados como *payload* da mensagem)
- d) <http://<ip:porta>/api/alterar> - Alterar Escola (PUT - os dados são passados como *payload* da mensagem)
- e) <http://<ip:porta>/api/excluir/id> - Excluir Escola (DELETE)

# Angular

## Referências





# REFERÊNCIAS

1. <https://angular.io/docs/ts/latest/>
2. <https://github.com/angular/angular-cli>
3. <http://loiane.training/>
4. <https://blog.thoughttram.io/>
5. <https://toddmotto.com/>
6. <https://scotch.io/>
7. <https://alligator.io/>



**Dúvidas?**

*Agradecemos pela atenção.*

**Klaus Cavalcante**

*klaus.cavalcante@serpro.gov.br*