

# Projeto de Interfaces WEB

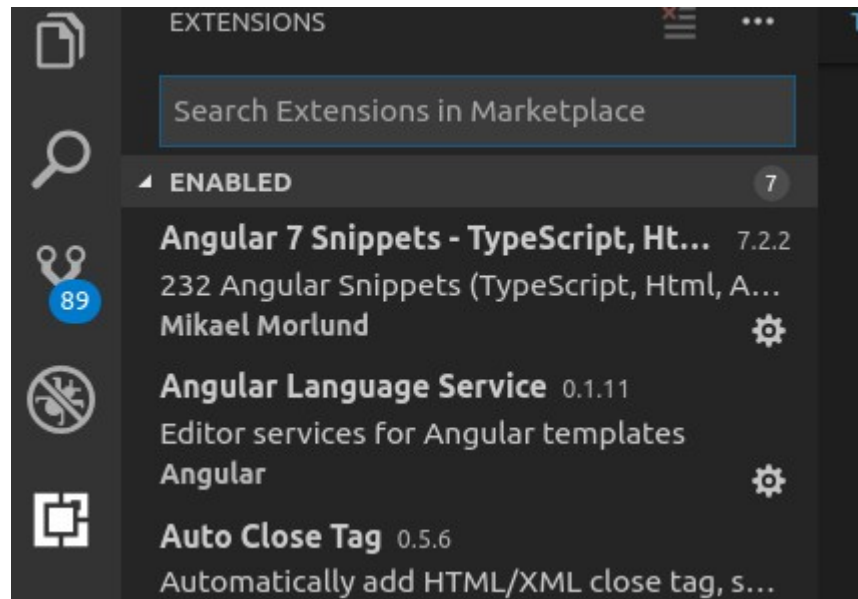
## Introdução ao Angular Aula 01

# Introdução ao Angular

- Agenda
  - Preparando e instalando o ambiente
  - Primeiro projeto
  - Instalando e usando o BootStrap
  - Data Binding
    - Event Biding
    - Property Biding
    - Interpolation
    - 2-Way Data Biding
    - Variável de referência

# Preparando o Ambiente

- Instalar o **VSCode** (<https://code.visualstudio.com/download>)
- Instalar as extensões
  - HTML CSS Support
  - Auto Close Tag
  - Auto Rename Tag
  - Auto import
  - Path Intellisense
  - Angular Language Service
  - Angular v5 Snippets



# Instalar o Angular - Ubuntu

- **Instalar Node**

- `sudo apt-get update`
- `curl -sL https://deb.nodesource.com/setup_9.x | sudo -E bash -`
- `sudo apt-get install -y nodejs`
- `sudo apt-get install -y build-essential`

# Instalar o Angular - Ubuntu

- **Instalar TypeScript**

- `sudo npm install -g typescript`
- `sudo npm install -g typings`

# Instalar o Angular - Ubuntu

- **Instalar ferramentas Angular**
  - `sudo npm install --unsafe-perm -g @angular/cli`
  - `sudo npm install -g nodemon`

# Angular CLI

- What Is **Angular CLI**?
  - Angular CLI is a command-line interface (CLI) to automate your development workflow. It allows you to:
    - create a new Angular application;
    - run a development server with LiveReload support to preview your application during development;
    - add features to your existing Angular application;
    - run your application's unit tests;
    - run your application's end-to-end (E2E) tests;
    - build your application for deployment to production;

# Novo projeto

- Criar novo projeto (Angular Workspace)
  - `ng new <nomedoprojeto>`
- Rodar servidor
  - `ng serve` (dentro da pasta do projeto)
    - Carrega o arquivo de configuração “angular.json”

Servidor roda por padrão na porta 4200

Ou seja, aplicação rodando: <http://localhost:4200/>



# Componentes

- A página inteira é construída por componentes;
- Um componente pode usar outra para se compor;
- Componente raiz padrão: **app component**;
- Um componente precisa ter, pelo menos, um **template (o html+css)** e uma **classe componente**
- Componente app é composto basicamente por:
  - app.component.html (template)
  - app.component.css (template)
  - app.component.ts (classe)

# app.component.ts

```
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})  
  
export class AppComponent {  
  title = "Hello World!";  
}
```

# app.component.ts

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

- @Component é um decorator
  - Assinala que classe é um componente
  - Provê informações adicionais sobre o componente (metadados)
- selector - nome do elemento HTML referente a esse componente
- templateUrl - indica onde está o template HTML do elemento
- styleUrls - indica lista de arquivos CSS do componente

# app.component.html

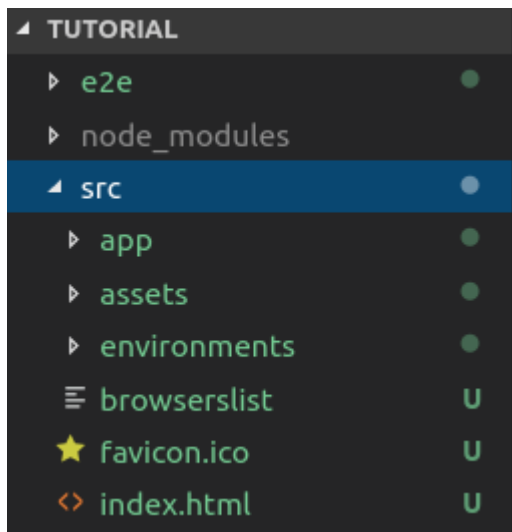
```
<h1>
```

Título da aplicação

```
</h1>
```

# Bootstrapping e AppModule

- Como a aplicação é iniciada?
  - A nossa página inicial (servida) é chamada pelo arquivo **index.html** (definido no angular.json)



```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Tutorial</title>
  <base href="/">

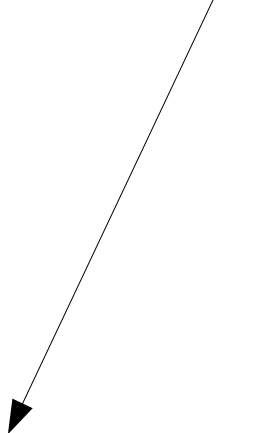
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

# Bootstrapping e AppModule

- Arquivos javascript na página principal são “injetados” pelo Angular (ver código fonte).

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Tutorial</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>
  <script type="text/javascript" src="runtime.js"></script><script type="text/javascript" src="es2015-polyfills.js"
nomodule></script><script type="text/javascript" src="polyfills.js"></script><script type="text/javascript"
src="styles.js"></script><script type="text/javascript" src="vendor.js"></script><script type="text/javascript"
src="main.js"></script></body>
</html>
```



# Bootstrapping e AppModule

- A tag **<app-root>** usa o componente principal da nossa aplicação, o AppComponent.

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

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

# Bootstrapping e AppModule

- Template HTML (app.component.html)
  - Modificado para melhor visualização.

```
<div style="text-align:center">  
  <h1>  
    Welcome to {{ title }}!  
  </h1>  
</div>
```



# Bootstrapping e AppModule

- Quem chama o AppComponent?
  - O AppModule (app.module.ts)

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

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

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

# Bootstrapping e AppModule

- Mas...e como o Angular sabe que é o AppModule?
  - main.ts (carregado pelo **angular.json**)

```
import { enableProdMode } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

# Componentes

- Uma aplicação Angular é uma combinação de Componentes.

app-root

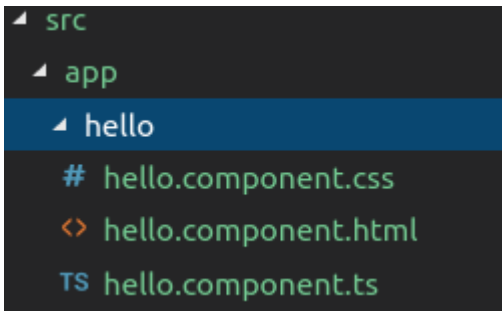
app-head

app-body

app-footer

# Componentes

- Criando um componente
  - Dentro da pasta do projeto:
    - `ng generate component <novo-componente> --spec=false`
    - `ng g c <novo-componente> --spec=false`



# Componentes

- Verifique que:
  - O novo componente já vem com o nome Component, concatenado (apenas uma convenção).
  - O novo componente é adicionado em “declarations” de AppModule
  - Automaticamente é criado o seu template html e seus metadados

# Componentes

- Se você quiser apagar um componente?
  - Apague as referências do componente criado no arquivo `app.module.ts`;
  - Apague a pasta do novo componente criado;

# Componentes

- Template literals (uso da **crase** ` dentro do meta-dado)

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

```
@Component({  
  selector: 'app-hello',  
  template: `  
    <h2>  
      Olá template literals!  
    </h2>  
  `,  
  styleUrls: ['./hello.component.css']  
})  
export class HelloComponent{  
  
}
```

# Componentes

- Dentro de app.component.html:
  - Não mexa no **index.html!!!!**

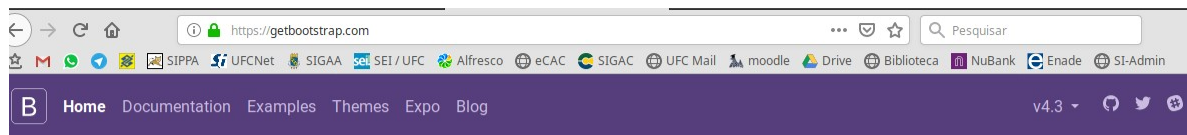
```
<div style="text-align:center">  
  <h1>  
    Welcome to {{ title }}!  
  </h1>  
  <app-hello></app-hello>  
</div>
```





# Instalando o Bootstrap

- **Bootstrap** é um framework css, independente do Angular, deixando as páginas HTML muito mais bonitas.



## Bootstrap

Build responsive, mobile-first projects on the web with the world's most popular front-end component library.

Bootstrap is an open source toolkit for developing with HTML, CSS, and JS. Quickly prototype your ideas or build your entire app with our Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful plugins built on jQuery.

[Get started](#)

[Download](#)

Currently v4.3.1

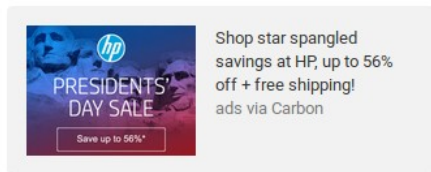


# Instalando o Bootstrap

- Instalando no seu projeto.
  - Procure no site do Bootstrap o componente “Jumbotron”
  - Guarde essa página por enquanto...

## Jumbotron

Lightweight, flexible component for showcasing hero unit style content.



A lightweight, flexible component that can optionally extend the entire viewport to showcase key marketing messages on your site.

# Instalando o Bootstrap

- Abra o terminal no seu projeto:
  - **npm install bootstrap --save**
- Veja a pasta node\_modules
  - Pasta “bootstrap”
  - Pasta “dist”
  - Pasta “css”
  - Arquivo “bootstrap.css”
- Agora, adicione a chamada a esse arquivo dentro do arquivo “**angular.json**” do seu projeto:

```
],  
  "styles": [  
    "src/styles.css",  
    "./node_modules/bootstrap/dist/css/bootstrap.css"  
  ],  
  "scripts": [],  
  "es5BrowserSupport": true
```

# Testando o Bootstrap

- Com a página do Jumbotron aberta, copie o código exemplo:

```
<div class="jumbotron">
  <h1 class="display-4">Hello, world!</h1>
  <p class="lead">This is a simple hero unit, a simple jumbotron-style component for calling extra attention
to featured content or information.</p>
  <hr class="my-4">
  <p>It uses utility classes for typography and spacing to space content out within the larger container.</p>
  <a class="btn btn-primary btn-lg" href="#" role="button">Learn more</a>
</div>
```

# Testando o Bootstrap

- ...e cole dentro de **app.component.html** (ou de outro component.html)

```
<div class="container">
  <div class="jumbotron">
    <h1 class="display-4">Hello, world!</h1>
    <p class="lead">This is a simple hero unit, a simple jumbotron-style component for calling extra attention to
    featured content or information.</p>
    <hr class="my-4">
    <p>It uses utility classes for typography and spacing to space content out within the larger container.</p>
    <a class="btn btn-primary btn-lg" href="#" role="button">Learn more</a>
  </div>
</div>
```

# Testando o Bootstrap

- Rode novamente o **ng serve**
  - Não esqueça de **salvar** os arquivos antes!

Hello, world!

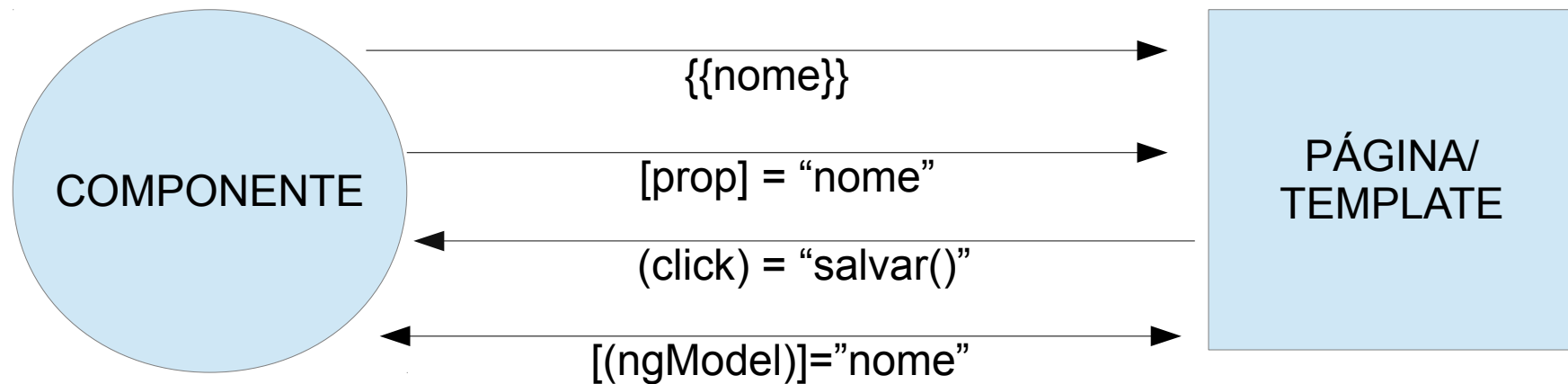
This is a simple hero unit, a simple jumbotron-style component for calling extra attention to featured content or information.

It uses utility classes for typography and spacing to space content out within the larger container.

[Learn more](#)

# Data Biding

- Forma de troca de informações entre o componente (ts) e a página/template (html). Vem em 4 sabores.



# Data Biding

- `{{nome}}` → **Interpolação**
- `[prop] = "nome"` → **Property bidding**
- `(click) = "salvar()"` → **Event Biding**
- `[(ngModel)]="nome"` → **2-way Data Biding** (banana box)



# Interpolação {{}}

- No arquivo app.component.ts

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

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

```
export class AppComponent {  
  nome = 'Jefferson de Carvalho';  
}
```

# Interpolação {{}}

- No arquivo app.component.html

```
<h1>Meu nome é {{nome}}</h1>
```

- Exercício: mostre agora a idade.
- É possível também que sejam feitos cálculos dentro da interpolação.
- Crie um método para retornar a idade e use o método na interpolação.

# Event Biding () - salvar

- Reagindo as iterações do usuário no template.
- No arquivo app.component.html:
  - Procure o component do Bootstrap Forms → **Form groups** (ou apenas copie o código abaixo, já modificado)
  - Copie e cole o código em app.component.html:

```
<div class="container">  
  <div class="form-group">  
    <label>Nome</label>  
    <input type="text" class="form-control">  
  </div>  
</div>
```

# Event Biding () - salvar

- Ainda no site do Bootstrap, procure o componente **Alerts**. Escolha um deles.
- Copie também no app.component.html

```
<div class="container">  
  <div class="alert alert-primary" role="alert">  
    Seu nome é {{nome}}  
  </div>  
  <div class="form-group">  
    <label>Nome</label>  
    <input type="text" class="form-control">  
  </div>  
</div>
```

# Event Biding () - salvar


- Procure também, um componente **Buttons**, do Bootstrap, no site.
- Cole no final de app.component.html

```
<div class="container">
  <div class="alert alert-primary" role="alert">
    Seu nome é {{nome}}
  </div>
  <div class="form-group">
    <label>Nome</label>
    <input type="text" class="form-control">
  </div>
  <button type="button" class="btn btn-primary">Salvar</button>
</div>
```

# Event Biding () - salvar

- Em app.component.ts


```
@Component({  
  selector: 'app-root',  
  templateUrl: './app.component.html',  
  styleUrls: ['./app.component.css']  
})  
export class AppComponent {  
  nome = 'Jefferson de Carvalho';  
  
  salvar(){  
    console.log(`Salvando ${this.nome}`);  
  }  
}
```



# Event Biding (click) - salvar

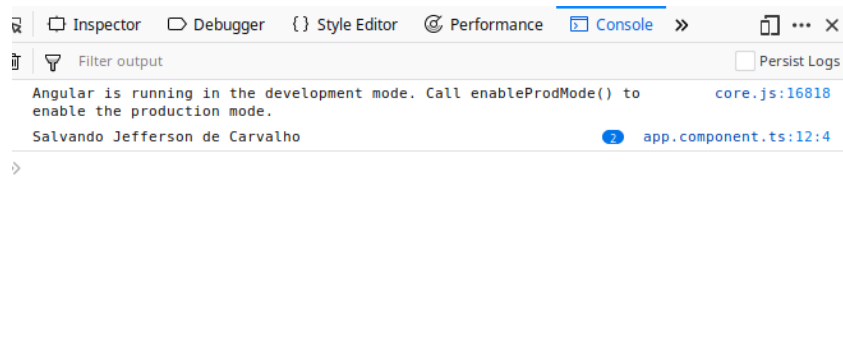
- Em app.component.html

```
<div class="container">
  <div class="alert alert-primary" role="alert">
    Seu nome é {{nome}}
  </div>
  <div class="form-group">
    <label>Nome</label>
    <input type="text" class="form-control">
  </div>
  <button type="button"
    class="btn btn-primary"
    (click)="salvar()">Salvar</button>
</div>
```



# Event Biding (click) - salvar

- Veja no console do navegador...



Seu nome é Jefferson de Carvalho

Nome

Salvar

Lista de possíveis eventos:

[https://www.w3schools.com/jsref/dom\\_obj\\_event.asp](https://www.w3schools.com/jsref/dom_obj_event.asp)



# Event Biding (click) - salvar

- Agora, mude o nome em app.component.ts, dentro do método salvar(). Crie também um contador.

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  nome = 'Jefferson de Carvalho';
  contador = 0;

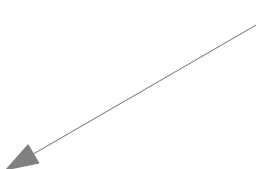
  salvar(){
    console.log(`Salvando ${this.nome}`);
    this.nome = "Jefferson " + this.contador;
    this.contador++;
  }
}
```



# Event Biding (input) - dinâmico

- Modificando o input (mudar o nome enquanto digita...)

```
<div class="container">  
  <div class="alert alert-primary" role="alert">  
    Seu nome é {{nome}}  
  </div>  
  <div class="form-group">  
    <label>Nome</label>  
    <input type="text" class="form-control" (input)="mudar($event)">  
  </div>  
  <button type="button"  
    class="btn btn-primary"  
    (click)="salvar()">  
    >Salvar</button>  
</div>
```



# Event Biding (input) - dinâmico

- No componente ts...

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

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

```
export class AppComponent {  
  nome = 'Jefferson de Carvalho';  
  contador = 0;
```

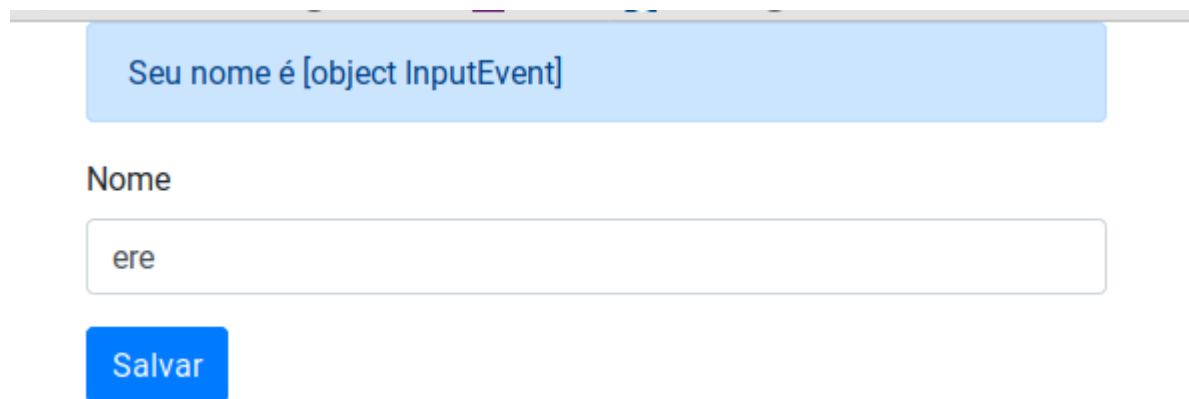
```
  salvar(){  
    console.log(`Salvando ${this.nome}`);  
    this.nome = "Jefferson " + this.contador;  
    this.contador++;  
  }
```

```
  mudar(evento:any){  
    this.nome = evento;  
  }  
}
```



# Event Biding (input) - dinâmico

- Veja no navegador:



Seu nome é [object InputEvent]

Nome

Salvar

The image shows a browser window with a light blue header bar. Below the header, there is a light blue rectangular box containing the text 'Seu nome é [object InputEvent]'. Below this box, the label 'Nome' is followed by a text input field containing the text 'ere'. At the bottom of the form is a blue button with the text 'Salvar'.

# Event Biding (input) - dinâmico

- Faça uma pesquisa no Google sobre o [object InputEvent]
  - Ou apenas use o console do seu navegador.
- Você verá que ele (InputEvent) tem uma **propriedade** chamada **target** e dentro dela, **value**.

```
mudar(evento:any){  
  this.nome = evento.target.value;  
}
```

- Agora, enquanto você digita o nome no input, o valor do seu nome também é alterado dinamicamente, refletindo na página HTML

## Event Biding (input) - finalizando...

- Agora, clique em salvar. Veja que o nome no alerta mudou mas não no input... Como resolver?
- Objetivo: ao clicar em salvar, quero que o valor do input também mude, para o mesmo do alerta, ou seja, que o input seja preenchido com **this.nome** alterado pelo salvar...

# Event Biding (input) - finalizando...

- Solução: use interpolação

```
<div class="form-group">  
  <label>Nome</label>  
  <input type="text" class="form-control" (input)="mudar($event)"  
    value={{this.nome}}>  
</div>
```

- Clique em salvar agora e veja que o input também foi modificado.
- Essa é uma forma manual do **2-way biding**.

# Variável de Referência (click)

- O objetivo agora é modificar o valor do alerta **apenas** quando clicar em salvar, e não enquanto está digitando.
- Comente o código em app.component.ts (vamos focar em salvar):

```
export class AppComponent {  
  nome = 'Jefferson de Carvalho';  
  contador = 0;  
  
  salvar(){  
    //console.log(`Salvando ${this.nome}`);  
    /*this.nome = "Jefferson " + this.contador;  
    this.contador++;*/  
  }  
  
  /*mudar(event:any){  
    this.nome = event.target.value;  
  }*/  
}
```



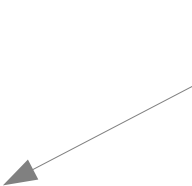
Comente estes trechos.



# Variável de Referência (click)

- Em app.component.html nós criamos a variável de referência:

```
<div class="container">
  <div class="alert alert-primary" role="alert">
    Seu nome é {{nome}}
  </div>
  <div class="form-group">
    <label>Nome</label>
    <input type="text" class="form-control" #nomeInput>
  </div>
  <button type="button"
    class="btn btn-primary"
    (click)="salvar()"
    >Salvar</button>
</div>
```



# Variável de Referência (click)

- #nomeInput pode ser usada em **qualquer** lugar do template, mas não pode ser vista pelo componente.
- O jeito é, então, passar #nomeInput via parâmetro para algum método:

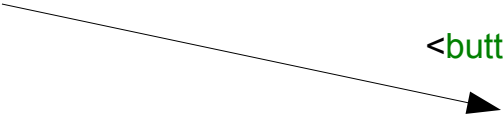
```
<button type="button"  
        class="btn btn-primary"  
        (click)="salvar(nomeInput)">  
    Salvar  
</button>
```

# Variável de Referência (click)

- Agora vamos receber o nomeInput no app.component.ts

```
salvar(nomeInput:any){  
  console.log(nomeInput);  
  //console.log(`Salvando ${this.nome}`);  
  /*this.nome = "Jefferson " + this.contador;  
  this.contador++;*/  
}
```

- Veja no console do navegador.
- Recebemos diretamente o valor “**target**”
- Sendo assim, passe diretamente o **value** para o método, diretamente no HTML.



```
<button type="button"  
  class="btn btn-primary"  
  (click)="salvar(nomeInput.value)"  
>Salvar</button>
```

# Variável de Referência (click)

- Agora no app.component.ts

```
salvar(nome:string){  
  this.nome = nome;  
  //console.log(` Salvando ${this.nome}`);  
  /*this.nome = "Jefferson " + this.contador;  
  this.contador++;*/  
}
```

- Use diretamente o tipo string, já que você está passando **nomeInput.value**.
- Veja que agora ao clicar em salvar, o nome do alerta muda de acordo com o que está no input.

# Property Biding [ ]

- Usado quando você quer vincular uma **propriedade do html no template**, com alguma **expressão TS**.
- Por exemplo: imagine que você queira desabilitar o botão salvar, caso o input esteja vazio.
- Nessa caso, você quer modificar a propriedade **disabled** do input dinamicamente.

# Property Biding [ ]

- Vamos voltar ao código HTML anterior (Event Biding - input)

```
<div class="container">
  <div class="alert alert-primary" role="alert">
    Seu nome é {{nome}}
  </div>
  <div class="form-group">
    <label>Nome</label>
    <input type="text" class="form-control"
      (input)="mudar($event)"
      value="{{nome}}">
  </div>
  <button type="button"
    class="btn btn-primary"
    (click)="salvar()"
    >Salvar</button>
</div>
```


# Property Biding [ ]

- Em app.component.ts

```
export class AppComponent {  
  nome = 'Jefferson de Carvalho';  
  contador = 0;  
  
  salvar(){  
    this.nome = "Jefferson " + this.contador;  
    this.contador++;  
  }  
  
  mudar(event:any){  
    this.nome = event.target.value;  
  }  
}
```

# Property Biding [ ]

- Criando a propriedade no html:

```
<button type="button"  
  class="btn btn-primary"  
  (click)="salvar()"   
  [disabled]="nome.length==0">Salvar</button>m
```

- Você também pode fazer o mesmo com o input (ao invés de usar interpolação):

```
<input type="text" class="form-control"  
  (input)="mudar($event)"  
  [value]="nome">
```



## 2-Way Biding [()]

- Event Biding + Property Biding (ou vice-versa)
- Nesses casos, para um mesmo elemento, é interessante usar o [()]
- No nosso caso, tudo que acontecer com o **input** no template, vai mudar o **nome** no componente. E tudo que acontecer com o **nome** no componente, vai também mudar o **input**.

## 2-Way Biding [()]

- Simplificando o código do HTML, podemos chamar diretamente o código do método mudar, dentro das aspas:

```
<input type="text" class="form-control"  
      (input)="nome = $event.target.value"  
      [value]="nome">
```

- Perceba que um evento (input) modifica o valor nome quando o input é modificado.
- E uma propriedade [value] do input é modificada quando o nome no componente app.component.ts é alterado (pelo click do botão, por exemplo...).

## 2-Way Biding [()]

- Nesse caso, podemos usar a diretiva ngModel:

```
<div class="form-group">  
  <label>Nome</label>  
  <input type="text" class="form-control"  
    [(ngModel)]="nome">
```

- Não esqueça de importa no app.modules:

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

```
@NgModule({  
  declarations: [  
    AppComponent,  
    HelloComponent  
  ],  
  imports: [  
    BrowserModule,  
    AppRoutingModule,  
    FormsModule  
  ],  
  ...
```

E...

- Por hoje é só.