

Desenvolvimento Web em Angular

por João Teixeira

Dinâmica de apresentação

- Bora conhecer-nos um pouco melhor...



O que é o Angular?

- Angular é uma framework JavaScript reativa
- Mantido por uma equipa da Google e de código aberto
- Amplamente utilizado para aplicações web modernas
- É contruído utilizando TypeScript
- E tem uma arquitetura baseada em componentes, que são pequenos grupos de funcionalidade isoladas e independente
- Os componentes são compostos geralmente por um arquivo TS (o comportamento), um arquivo HTML (a estrutura visual) e um arquivo de estilos (CSS ou um pré-processador)

Um pouco da história do Angular

- Podemos dizer que a história do Angular é um pouco caótica e até hoje podem gerar dúvidas sobre a diferença entre AngularJS e Angular
- AngularJS começou a ser concebido em 2009 com a intenção de simplificar o desenvolvimento de aplicações web
- Inicialmente o AngularJS era utilizado apenas em alguns projetos, mas percebendo na prática o quanto beneficiava no desenvolvimento de aplicações web, os criadores decidiram tornar o AngularJS open-source em 2010
- E logo no seu início o AngularJS teve uma grande aceitação, não demorando muito a haver dezenas de projetos a usar essa nova tecnologia

Um pouco da história do Angular

- Em 2015 já era utilizado por grande empresas e milhares de desenvolvedores pelo mundo inteiro
- O seu desenvolvimento foi sendo mantido oficialmente pela Google e já contava com uma equipa própria
- E por essa altura surgiu no JS o ECMAScript 6 (ES6), que trazia novos avanços tecnológicos e padrões de desenvolvimento
- Com isso a equipa do AngularJS precisou de buscar estratégias para adequar a framework a esse novo cenário
- Foi a partir disso que a equipa optou por lançar a versão 2.0
- Mas o que era para ser apenas uma atualização acabou por se tornar um dos períodos mais caóticos da história do Angular

Um pouco da história do Angular

- Esse caos deu-se pelo facto de o AngularJS ter sido completamente reescrito na sua versão 2.0, mudando totalmente conceitos e praticas utilizadas nas versões anteriores (trazendo também utilização do TS)
- É fácil imaginar a rejeição que isso gerou na grande comunidade que já tinha na época
- Com tudo isso e a impossibilidade da portabilidade das aplicações que utilizavam a versão anterior, o AngularJS praticamente tonou-se um novo framework
- Então, em 2016, o Google optou por dividir a framework em 2, nascendo assim o Angular

Um pouco da história do Angular

- A versão legada continuaria a ser desenvolvida e manteve nome original (AngularJS) e a versão 2 chamou-se só "Angular"
- As tuas versões possuem sites e documentações diferentes
- Atualmente AngularJS está na sua versão 1.8 e é utilizado geralmente em grandes projetos legados
- Já o Angular está na sua versão 16.2
- E junto com o React e Vue.js, lideram o ranking de frameworks de JavaScript mais utilizados

Preparação do ambiente de desenvolvimento

- Navegador Web (*Web browser*):
 - Google Chrome
 - Mozilla Firefox
 - Microsoft Edge
 - Safari
 - Opera
 - Brave
 - (ou muitos outros...)
- IDE (do inglês *integrated development environment*):
 - Visual Studio Code
 - Sublime Text
 - Visual Studio (2022; Comunidade)
 - NetBeans
 - IntelliJ IDEA
 - (ou muitos outros...)

Preparação do ambiente de desenvolvimento

- Eu vou usar:
 - Google Chrome - <https://www.google.com/chrome/>
 - Visual Studio Code - <https://code.visualstudio.com/download/>
 - * *São livre para usar outro Navegador ou IDE*
- Precisamos, além disso de:
 - Instalar o Node.js (versão LTS) - <https://nodejs.org/>
 - **É recomendado usar um *Node Version Manager* (nvm)**, em vez de só instalar o Node.js normal:
 - Para Windows - <https://github.com/coreybutler/nvm-windows/>
 - Para unix, macOS e windows WSL - <https://github.com/nvm-sh/nvm/>
 - Um *Node Package Manager* (npm) - geralmente instalado com o Node.js
 - Angular CLI - na linha de comando - `npm install -g @angular/cli`

"Mãos à massa!"

Exercício prático em grupo



Gerir versões no *Node Version Manager*

- Instalar uma versão:
 - > nvm install <versão>
 - > nvm install lts
 - > nvm install 20.9.0
- Ver a lista de versões instaladas:
 - > nvm list
- Usar um versão previamente instalada:
 - > nvm use <versão>
 - > nvm use 20.9.0
- Remover um versão já instalada:
 - > nvm uninstall <versão>
 - > nvm uninstall 20.9.0
- Ver a versão do NVM:
 - > nvm --version
 - > nvm version
 - > nvm v
- Mais detalhes:
 - > nvm

Gerir o Node.js e o *Node Package Manager*

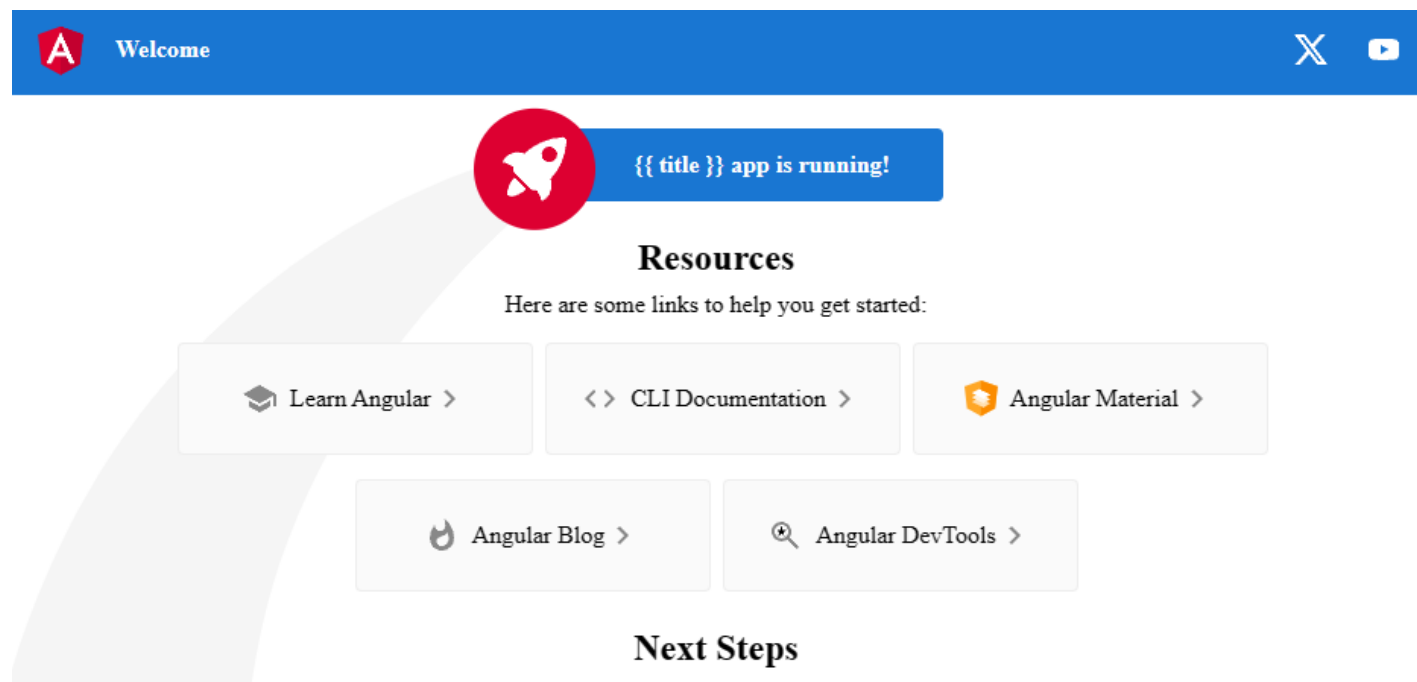
- Ver a versão do Node.js:
 - > node --version
 - > node -v
- Mais detalhes:
 - > node --help
 - > node -h
- Instalar um pacote no NPM:
 - > npm install <pacote>
 - Instalar como um pacote global:
 - > npm install -g <pacote>
- Remover um pacote no NPM:
 - > npm uninstall <pacote>
 - > npm uninstall -g <pacote>
- Listar pacotes instalados:
 - > npm ls
- Mais detalhes:
 - > npm help

Criar um novo projeto Angular

- Se ainda não temos instalado a CLI devemos o fazer:
 - > npm install -g @angular/cli
- Para criar um novo projeto usamos o comando:
 - > ng new nome-da-minha-app
 - > ng n nome-da-minha-app
 - Lembrar de entrar dentro da pasta do nosso projeto:
 - > cd nome-da-minha-app // comando em Windows
- Para começarmos a visualizar a nossa App:
 - ng serve
 - ng serve --open // para abrir logo a app no navegador predefinido
 - ng serve -o // " " "

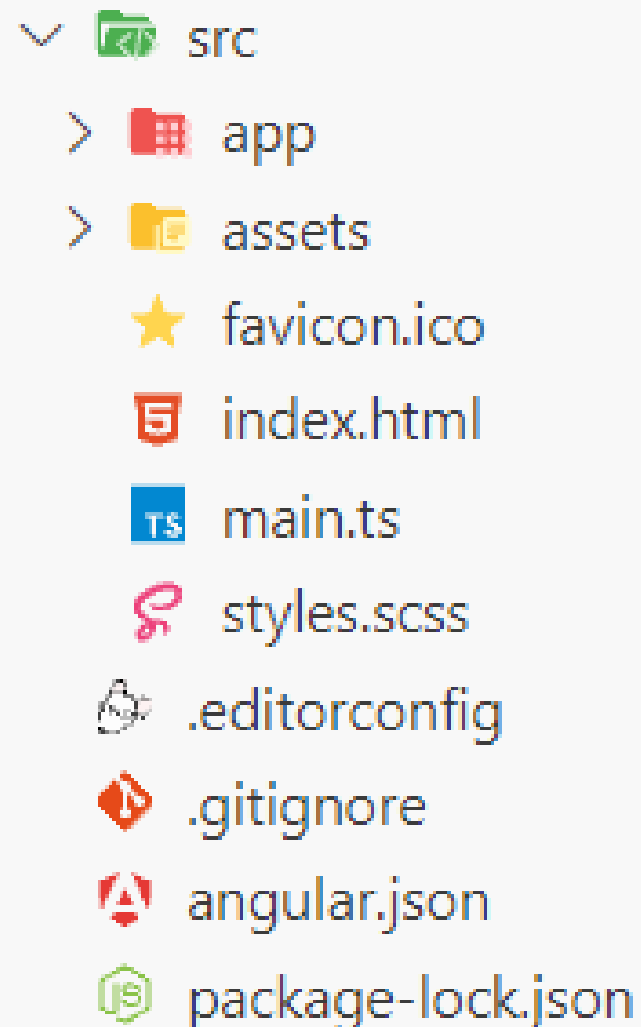
Criar um novo projeto Angular

- Normalmente a aplicação será servida em <http://localhost:4200/>
 - Caso o angular CLI não conseguir usar a porta 4200, tentara usara outra automaticamente
- Devemos ter algo parecido com:



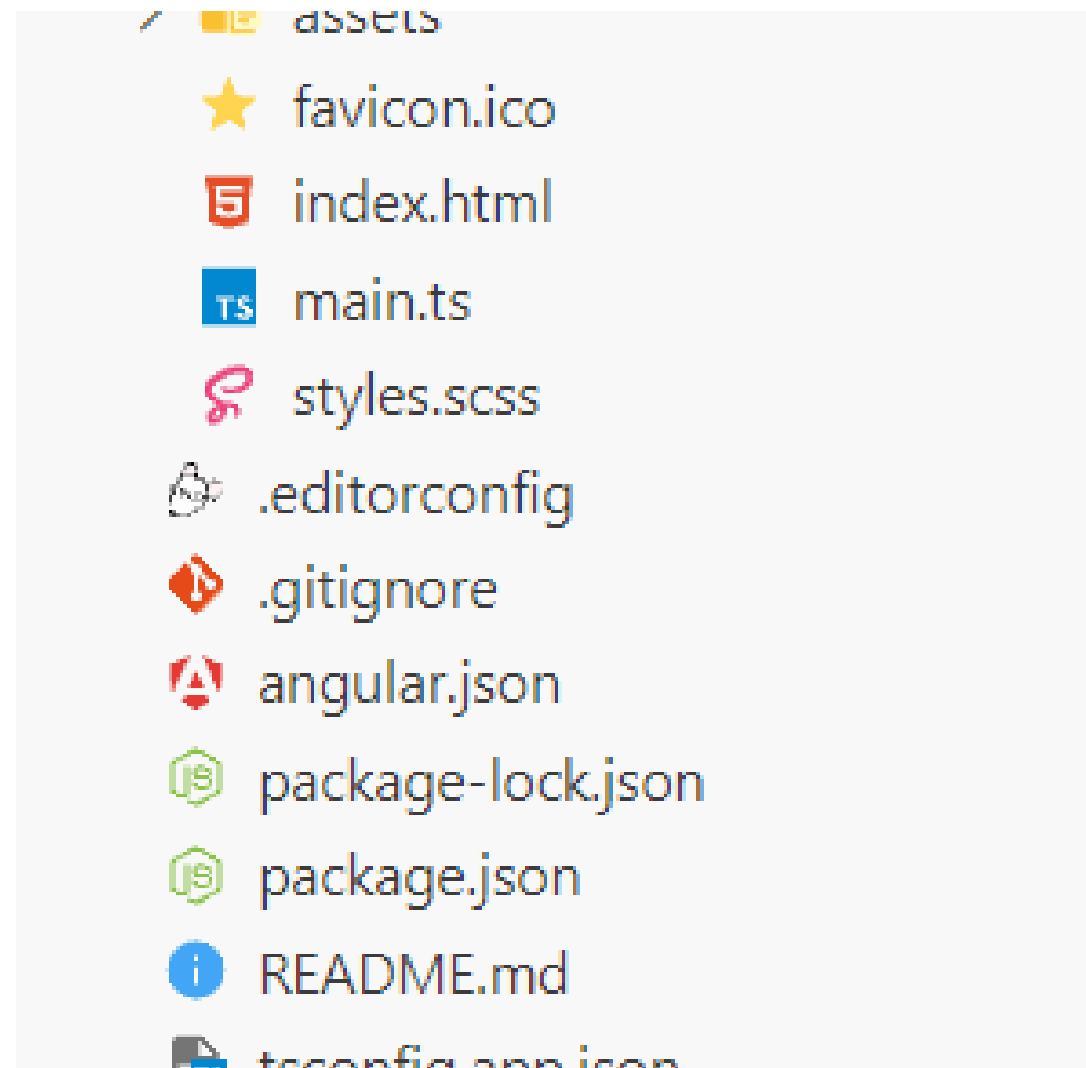
Estrutura de pastas e ficheiros

- /src/app/
 - É onde estará todo o que vamos desenvolver na nossa aplicação
- /src/assets/
 - É onde devemos por todo o conteúdo de média/outros usado pela nossa aplicação
- /src/favicon.ico
 - O ícone padrão da nossa aplicação



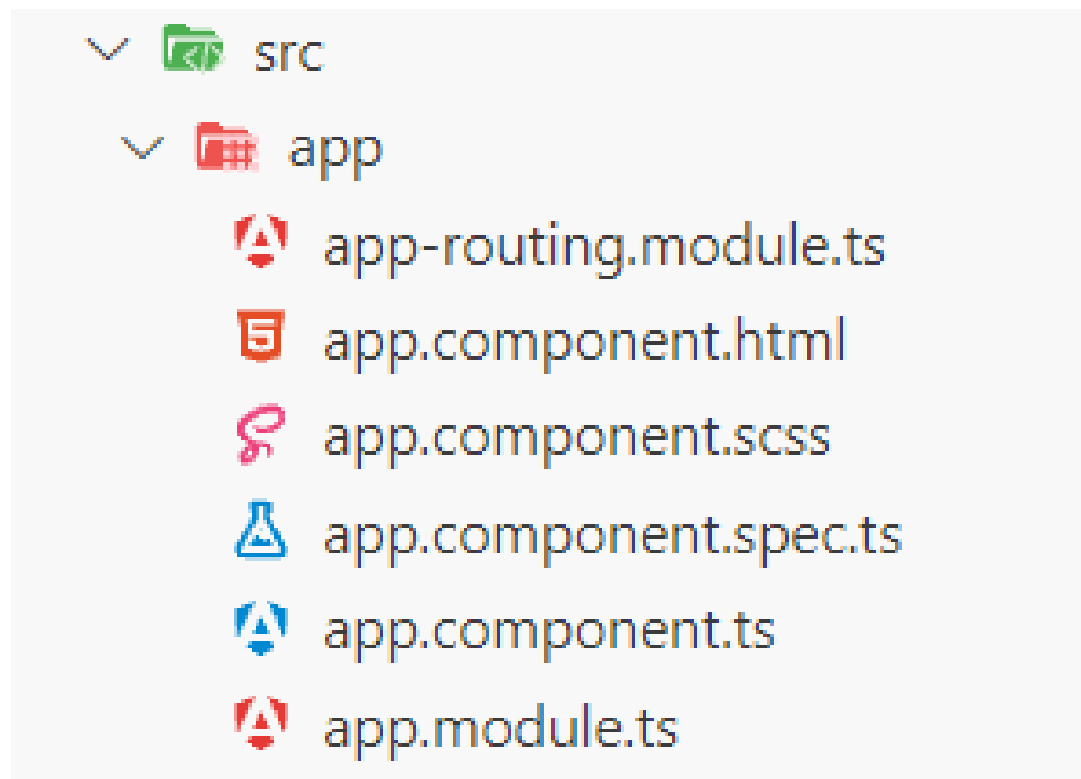
Estrutura de pastas e ficheiros

- /src/index.html
 - Onde a nossa aplicação será renderizada
 - É possível adicionar algumas definições globais
- /src/main.ts
 - Onde é possível programar algumas definições globais
- /src/style.scss
 - Onde é possível definir estilos globais para a nossa aplicação
- /angular.json
 - Onde estão algumas definições da nossa aplicações



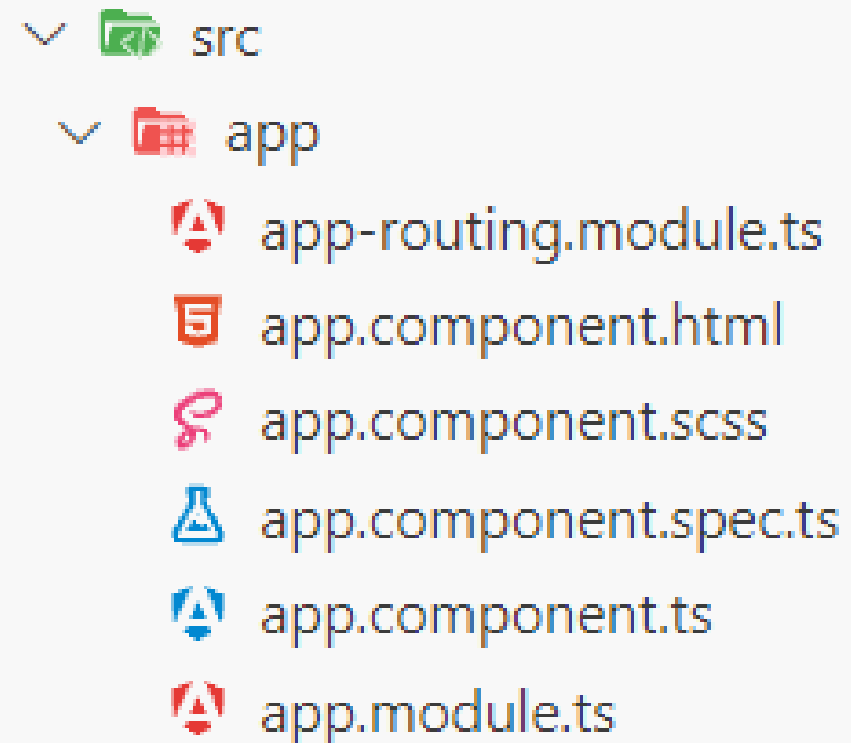
Estrutura do componente APP

- O componente base da aplicação tem na sua estrutura os ficheiros representados na imagem
- app-routing.module.ts
 - É onde fica a definição das rotas que podemos navegar
- app.component.html
 - É onde estará a estrutura HTML do componente APP



Estrutura do componente APP

- app.component.scss
 - É onde estará os estilos CSS do componente a serem usados pelo HTML
- app.component.spec.ts
 - Ficheiro de testes unitários
- app.component.ts
 - Onde estará a lógica do componente
- app.module.ts
 - Onde vamos importar os módulos a serem usados pela aplicação

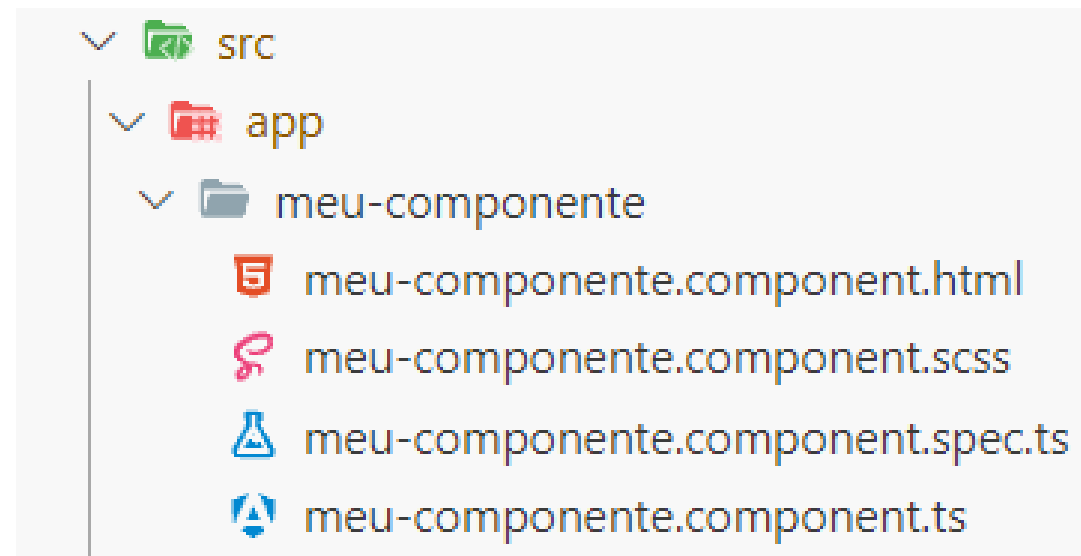


Criar um novo componente em Angular

- Usamos o comando:
 - > ng generate component meu-componente
 - > ng g c meu-componente
- É possível criar o componente numa "subpasta" (1 ou mais), ex.:
 - > ng g c components\meu-componente
 - > ng g c pages\meu-componente
 - > ng g c pasta\outra-pasta\meu-componente
- Apos o comando finalizar o componente vai ser criado em `/src/app[/pasta e subpastas]/<nome do componente>`

Estrutura padram de um componente

- *.component.html
 - Onde estará a estrutura HTML
- *.component.scss
 - Onde estará os estilos CSS para o componente em especifico
- *.component.spec.ts
 - Ficheiro de testes unitários
- *.component.ts
 - Onde estará a lógica do componente



Data binding

- Data binding é um conceito importante em Angular que permite apresentar/sincronizar os dados entre o código TypeScript e o HTML
- Existem quatro tipos de data binding em Angular:
 - Interpolation
 - Property binding
 - Event binding
 - Two-way binding

Interpolation

- É o tipo mais simples de data binding
- Permite exibir os dados do código TS no HTML usando a sintaxe
 - {{ expression }}

```
meuNome = 'João';  
mensagem = 'Eu adoro gatos! 😊';  
imagem = 'https://images.pexels.com/photos/416160/  
pexels-photo-416160.jpeg?w=500';
```

```
<h1>Olá, o meu nome é {{ meuNome }}</h1>  
<p>{{ mensagem }}</p>  

```

Property binding

- É o tipo de data binding que permite passar os dados do seu código TS para os atributos dos elementos HTML
- Usando a sintaxe
 - [property]="expression"

```
botaoDesativado = true;  
pCorTexto = '#336699';  
pCorFundo = '#99ccff'
```

```
<button [disabled]="botaoDesativado">  
  Botão  
</button>  
<p  
  [id]="pId"  
  [style.color]="pCorTexto"  
  [style.backgroundColor]="pCorFundo">  
  E também gosto de cães!  
</p>
```

Event binding

- É o tipo de data binding que permite executar uma função do código TS quando um evento de um elemento HTML é disparado
- Usando a sintaxe
 - (event)="expression"

```
mostrarMensagem(): void {  
    alert('Obrigado por clicares no botão!');  
}
```

```
<button (click)="mostrarMensagem()">  
    Clica neste botão  
</button>
```


Two-way binding

- É o tipo de data binding que permite sincronizar os dados entre o código TS e o HTML em ambas as direções
- Usando a sintaxe
 - [(ngModel)]="expression"
- Para usar esse tipo de data binding, é preciso importar o módulo FormsModule no módulo principal
- Por exemplo, se houver uma variável chamada `name` no componente, é possível usá-la para criar um campo de texto no HTML que atualiza o valor de `name` nos dois sentidos
 - [(ngModel)]="name"

Passar dados do componente pai para o componente filho

- Para esse fim é usado o decorador `@Input()` numa variável no componente filho
- Isto significa que a variável vai passar a ser uma propriedade de entrada

```
// Filho
@Input() meuTitulo: string;
@Input('meu-titulo') meuTitulo: string;
@Input({alias: 'titulo', required: true}) meuTitulo: string;
```

```
<!-- Pai -->
<app-meu-item [meuTitulo]="nome"></app-meu-item>
<app-meu-item [meu-titulo]="nome"></app-meu-item>
<app-meu-item [titulo]="nome"></app-meu-item>
```

Passar dados do componente filho para o componente pai

- Neste caso é usado o decorador @Output() numa variável do componente filho
- Isso define a variável como uma propriedade de saída
- Essa comunicação de saída é feita através de um evento
- O componente pai deve usar a sintaxe de *event binding* para se inscrever no evento do componente filho

Passar dados do componente filho para o componente pai

```
// Filho
@Output() mudancaNoContador = new EventEmitter<number>();
contador: number;

incrementar() {
  this.contador++;
  this.mudancaNoContador.emit(this.contador);
}
```

```
<!-- Filho -->
<button (click)="incrementar()">+1</button>
```

```
// Pai
public contador: number;

atualizar(valor: number) {
  this.contador = valor;
}
```

```
<!-- Pai -->
<p>Contador: {{ contador }}</p>
<app-botao-contador (mudancaNoContador)="atualizar($event)"></app-botao-contador>
```

Eventos

- Eventos são ações ou ocorrências que acontecem no sistema e que podem ser capturados e tratados pelo Angular
- Eventos podem ser gerados por interações do utilizador, como cliques, teclas, gestos, etc...
- Ou por mudanças de estado, como carregamento, atualização, erro, etc...
- Eventos permitem criar aplicações reativas e responsivas, que respondem às ações do usuário ou às alterações do sistema de forma adequada
- Eventos, normalmente, executam um função no código TS. Essa função pode receber um argumento opcional chamado `$event`, que contém o objeto do evento gerado pelo navegador

Eventos

- Usa-se a sintaxe de *event binding* para associar a função do seu componente a um evento de um elemento HTML no HTML
- A sintaxe é (event)="expression"
- Onde event é o nome do evento
 - Alguns dos eventos possíveis são: click, submit, focus, copy, keyup, keypress, keydown, keydown.shift.t, mouseenter, load, etc...
- E onde expression é a chamada da função do seu componente
 - Exemplo: myFunction(), myFunction(\$event), etc...

```
<button (click)="onClick()">Clique-me</button>  
<input type="text" (keyup)="onKey($event)">
```

Sistema de Rotas

- Rotas são um recurso do Angular que permite definir e navegar entre diferentes componentes numa aplicação
- Com rotas, é possível criar uma experiência de utilizador mais dinâmica e interativa, além de facilitar a organização e a manutenção do código
- Caso não tenha sido selecionado que se pretende usar o sistema de rotas ao criar um novo projeto
 - É preciso criar o módulo de rotas e pode-se usar o comando
 - `ng generate module app-routing`
 - Importar, caso o comando não o tenha feito, os módulos
 - *RouterModule* e *Routes*

Sistema de Rotas

- Criar uma constante para o array de rotas
 - `const routes: Routes = []`
- Passar o array de rotas para o método `RouterModule.forRoot(routes)` nos `imports[]` do `@NgModule` do módulo de rotas
- Adicionar o `RouterModule` nos `exports[]` do `@NgModule` do módulo de rotas
- Importar o módulo de rotas no módulo principal (geralmente o `AppModule`), para que as rotas sejam registradas na aplicação
- Adicionar a diretiva `<router-outlet>` no HTML do componente principal (geralmente o `AppComponent`), para que ele possa indicar onde os componentes serão apresentados de acordo com a rota atual

Sistema de Rotas

- Em const routes: Routes = [], geralmente em src/app/app-routing.module.ts, é possível adicionar as rotas que queremos
- Exemplos:
 - { path: 'home', component: HomeComponent },
 - { path: 'sobre', component: AboutComponent },
 - { path: 'contato', component: ContactComponent }
- Para navegar para uma rota, usa-se a diretiva routerLink no HTML, exemplos:
 - Home
 - Sobre
 - Contato

