



# Full Stack development: Angular, Spring Boot, MySQL and AWS

Carlos A J Lazarin



# Introdução ao curso





## Público Alvo

- Qualquer desenvolvedor que deseja adquirir mais conhecimentos em como criar uma solução Full Stack com Angular, Spring Boot, MySQL e Amazon Web Services (AWS).

## Sobre mim

- Atualmente sou Desenvolvedor Java Back-end Sênior em uma empresa multinacional que presta consultoria para clientes nos EUA, Canadá, Alemanha, Holanda e Brasil;
- Possuo mais de 12 anos de experiência em análise e desenvolvimento de software trabalhando em empresas de diversos segmentos como financeiro/bancário, óleo & gás e varejo;
- Durante esse anos eu tive a oportunidade de exercer diferentes funções como Líder Técnico, Scrum Master e Product Owner;
- Possuo Bacharelado em Sistemas de Informação (UENP), Licenciatura em Computação (UENP) e Mestrado em Computação Aplicada (UTFPR).





## Qual o conhecimento necessário para iniciar o curso?

- Front-end (Angular):
  - JavaScript;
  - HTML;
  - CSS;
  - Type Script (pode ajudar mas não é essencial);
- Back-end (Spring Boot):
  - Java ;
  - SQL;

O que eu irei aprender?

---



## O que eu irei aprender no Front-end?

1. O que é Angular e porque eu deveria utilizá-lo;
2. Principais conceitos do Angular:
  - a. Estrutura de aplicação Angular: componentes, serviços, templates, navegação, diretivas etc.;
  - b. Comunicação entre componentes (pai -> filho; filho -> pai);
  - c. Utilizar formulários e lidar com entrada de dados do usuário;
  - d. Realizar chamadas para uma API Rest;
  - e. Rodar a aplicação local e no “modo” produção.



## O que eu irei aprender no Front-end?

3. O que é e como utilizar a biblioteca de componentes Angular Material;
4. Conceitos de CSS3 Flexbox;
5. O que é Git e como utilizar;
6. Configurar ambiente de desenvolvimento;
  - a. Instalar Visual Studio Code e suas extensões;
  - b. Instalar NodeJS;
  - c. Instalar Git;
  - d. Instalar e utilizar Angular CLI;
7. Exercício prático: criar a aplicação de Checklist.





## O que irei aprender no Back-end?

1. O que é Spring Boot e porque utilizá-lo;
2. Como criar minha primeira aplicação com Spring Boot;
  - a. Maven ou Gradle?
  - b. Dependências do Spring (H2, Web, Data, JUnit, MySQL etc.)
3. Configurar ambiente de desenvolvimento:
  - a. Instalar JDK;
  - b. Instalar e utilizar IntelliJ IDEA;



## O que irei aprender no Back-end?

5. Desenvolver uma API REST e seus principais conceitos;
  - a. Métodos HTTP;
  - b. Como definir endpoints;
  - c. Códigos de resposta e como utilizá-los;
6. Especificação da API REST: Spring Docs e Open API 3.0;
7. Configurar uma instância de banco de dados local;
8. Testes unitários;
9. Testar a camada web com `@WebMvcTest`.



## O que eu irei aprender sobre AWS?

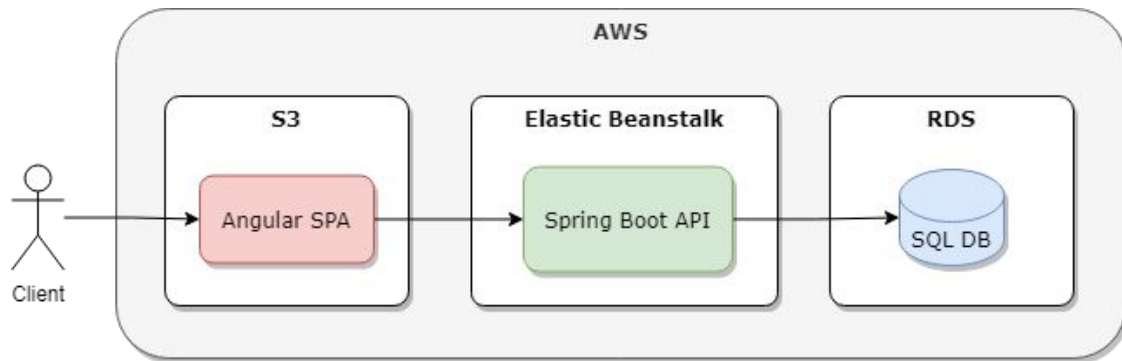
1. Criar uma conta e utilizar o período de testes (sem custo);
2. Fazer deploy do código Angular na S3 (Amazon Simple Storage Service);
3. Fazer deploy da aplicação Spring Boot no Elastic BeanStalk;
4. Configurar uma instância de banco de dados no RDS e como acessá-la de sua máquina local.



# Atividade prática - Aplicação de Checklist

---

# Visão Geral



# Aplicação de Checklist: Front-end

≡ Checklist Application

Welcome to Checklist Application



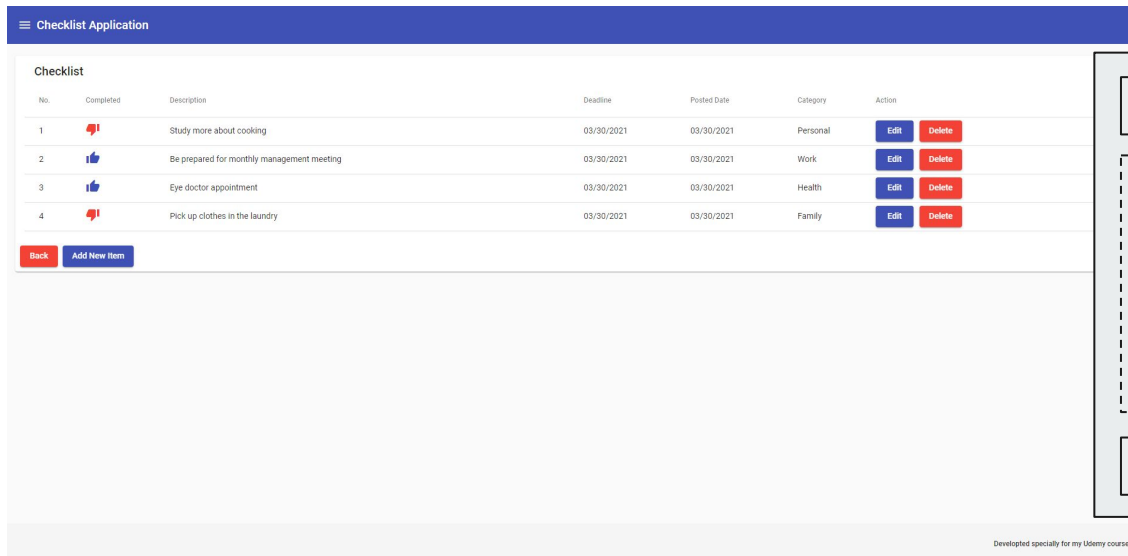
app.component.ts

header.component.ts

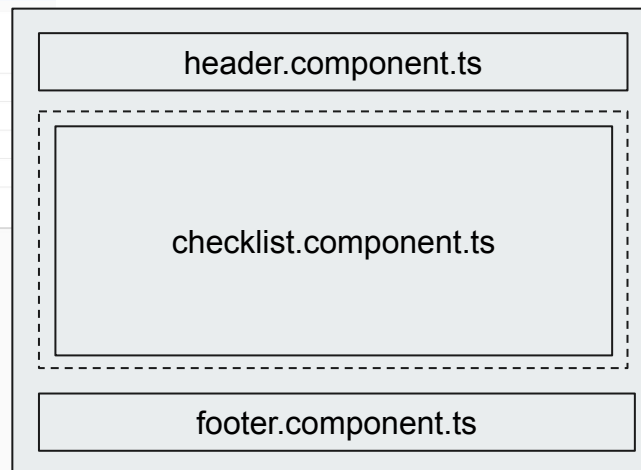
home.component.ts

footer.component.ts

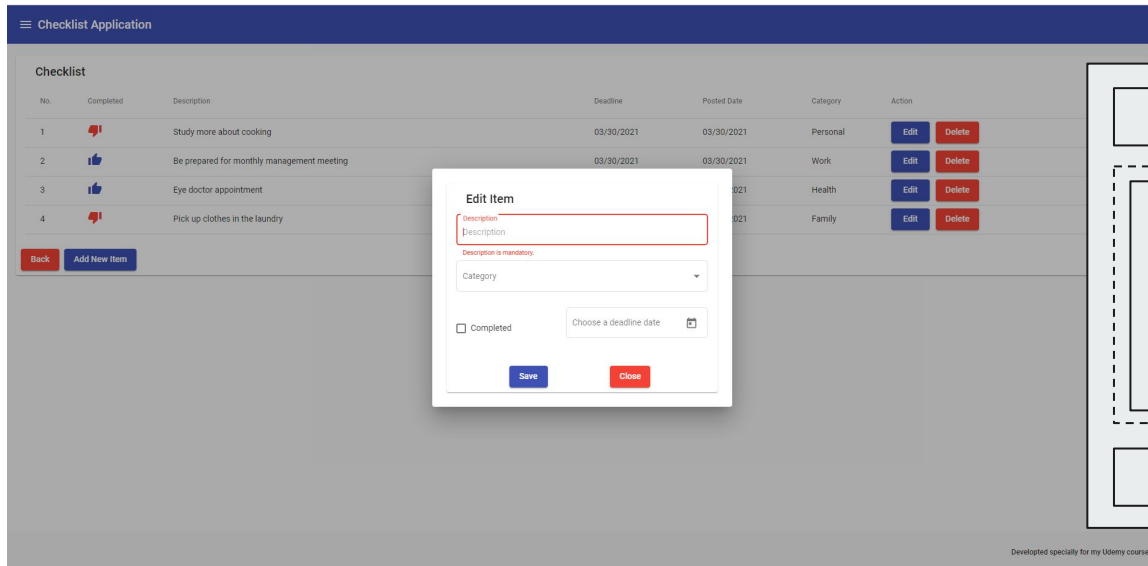
# Aplicação de Checklist: Front-end



app.component.ts



# Aplicação de Checklist: Front-end



app.component.ts

header.component.ts

checklist.component.ts

footer.component.ts





# CSS3 - Flexbox

[https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS\\_layout/Flexbox](https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox)

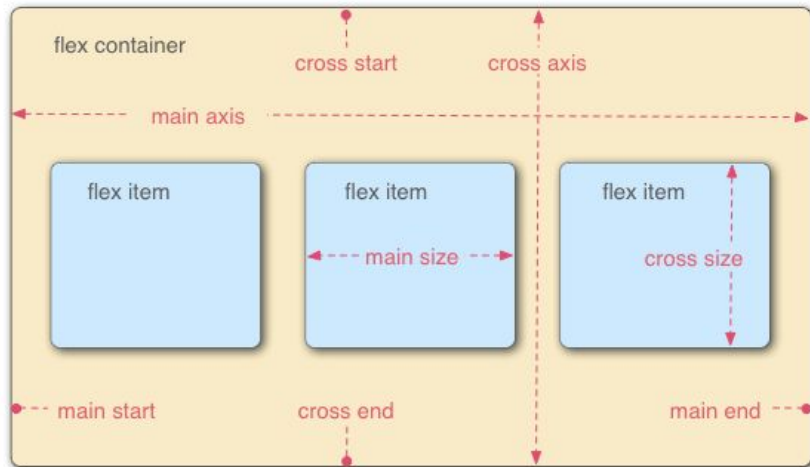




## CSS 3 - Flexbox

- Por muito tempo as propriedades “float” e “position” foram utilizadas para organizar os elementos de uma página HTML e ainda ser compatível com diferentes browsers;
- Solução: Flexbox e Grids;
- Flexbox: uma forma mais eficiente de posicionar, alinhar e distribuir o espaço entre os itens do “container” (mesmo quando o tamanho é desconhecido ou dinâmico - **flex**).

# CSS 3 - Flexbox



## Sample flexbox example

### First article

Tacos actually microdosing, pour-over semiotics banjo chicharrones retro fanny pack portland everyday carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui pickled sriracha normcore hashtag polaroid single-origin coffee cold-pressed. PBR&B tattooed trust fund twee, leggings salvia iPhone photo booth health goth gastropub hammock.

### Second article

Tacos actually microdosing, pour-over semiotics banjo chicharrones retro fanny pack portland everyday carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui pickled sriracha normcore hashtag polaroid single-origin coffee cold-pressed. PBR&B tattooed trust fund twee, leggings salvia iPhone photo booth health goth gastropub hammock.

### Third article

Tacos actually microdosing, pour-over semiotics banjo chicharrones retro fanny pack portland everyday carry vinyl typewriter. Tacos PBR&B pork belly, everyday carry ennui pickled sriracha normcore hashtag polaroid single-origin coffee cold-pressed. PBR&B tattooed trust fund twee, leggings salvia iPhone photo booth health goth gastropub hammock.

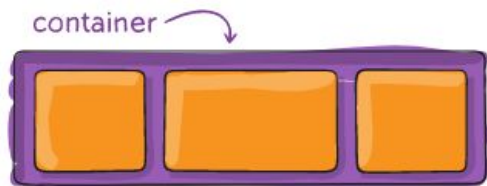
Cray food truck brunch, XOXO +1 keffiyeh pickled chambray waistcoat ennui. Organic small batch paleo 8-bit, intelligentsia umami wayfarers pickled, asymmetrical kombucha letterpress kitsch leggings cold-pressed squid chartreuse put a bird on it. Listicle pickled man bun cornhole heirloom art party.

# Propriedades do elemento pai: contêiner

---

## Propriedades do elemento pai: Flex container

- **Flex container:** ativa o contexto “flex” para todos os filhos (ou flex-items)

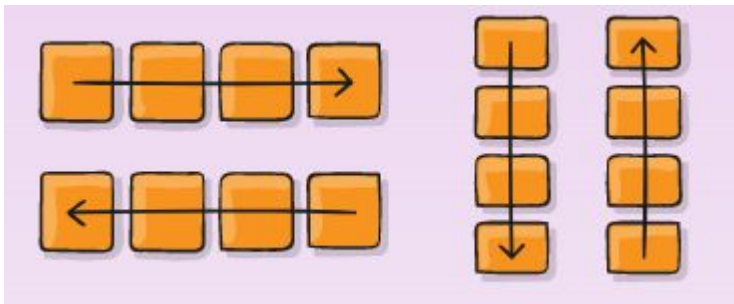


```
.container {  
  display: flex; /* or inline-flex */  
}
```

CSS

## Propriedades do elemento pai: Flex-direction

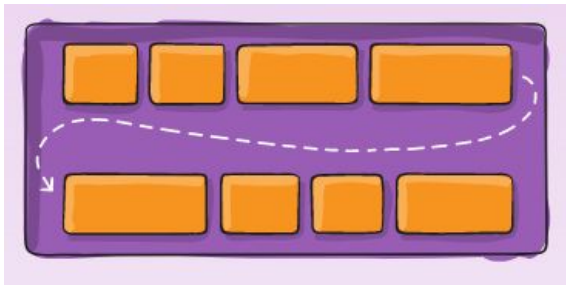
- **Flex-direction:** define o eixo principal (main-axis) e a respectiva posição dos itens(filhos) dentro do contêiner pai



```
.container {  
  flex-direction: row | row-reverse | column | column-reverse;  
}
```

## Propriedades do elemento pai: Flex-wrap

- **Flex-wrap:** por padrão, os itens flex são posicionados em um linha única (flex-row); essa propriedade controla o comportamento flex dos itens do contêiner caso ultrapassem o tamanho do contêiner pai.



```
.container {  
  flex-wrap: nowrap | wrap | wrap-reverse;  
}
```



## Propriedades do elemento pai: Flex-flow

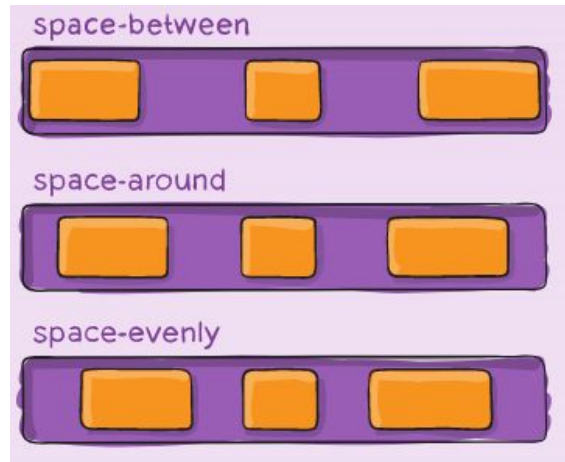
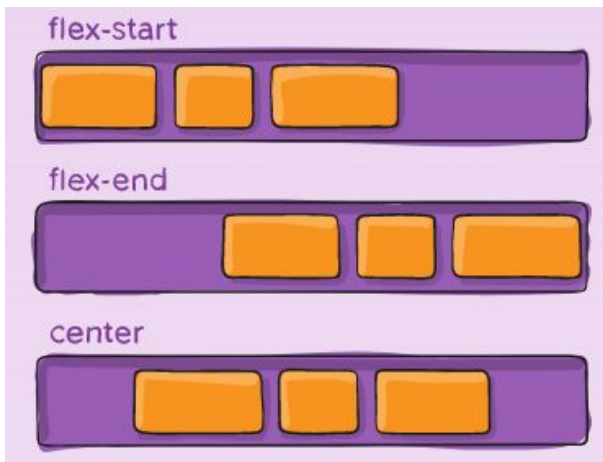
- **Flex-flow:** define flex-direction e flex-wrap em uma única propriedade

```
CSS
.container {
  flex-flow: column wrap;
}
```



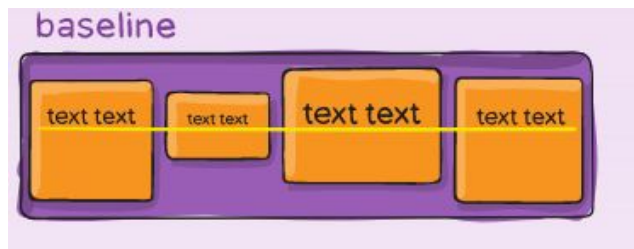
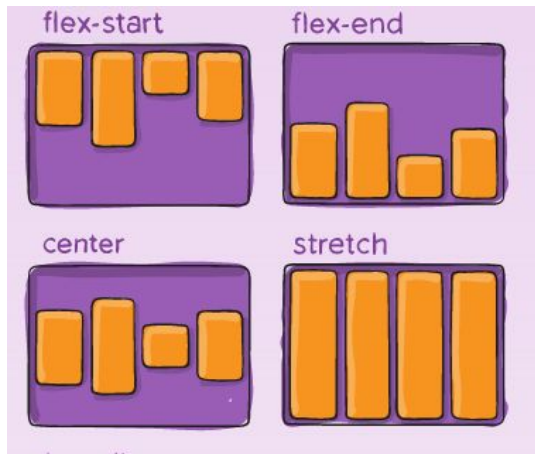
## Propriedades do elemento pai: Justify-content

- **Justify-content:** define o alinhamento dos itens do contêiner em relação ao eixo principal (main-axis)



# Propriedades do elemento pai: Align-items

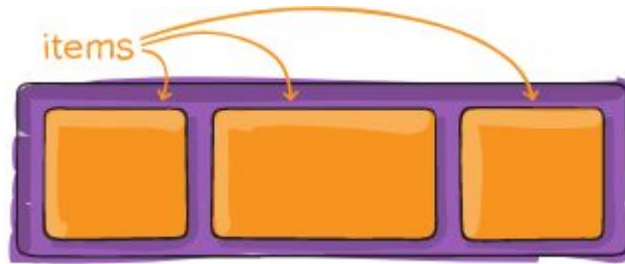
- **Align-items:** define com os elementos são posicionados em relação ao eixo vertical (cross-axis)



**Propriedades dos filhos:**  
**flex-its**

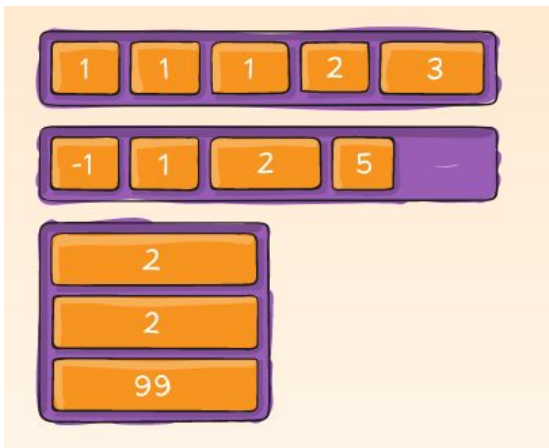
---

## Propriedades dos filhos (flex-items)



## Propriedades dos filhos: Order

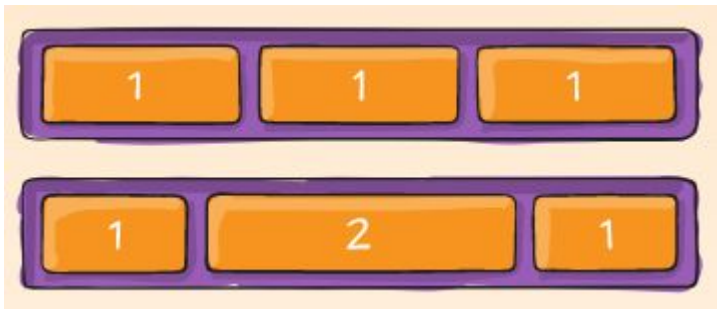
- **Order:** alterar a ordem a qual os elementos aparecem no layout (por padrão, os elementos são posicionados na ordem em que são criados)



```
.item {  
  order: 5; /* default is 0 */  
}
```

## Propriedades dos filhos: Flex-grow

- **Flex-grow:** controla a maneira a qual o elemento filho pode crescer no espaço disponível remanescente do contêiner pai



```
.item {  
  flex-grow: 4; /* default 0 */  
}
```



## Propriedades dos filhos: Flex-basis

- **Flex-basis:** define o valor inicial para um flex item

```
.item {  
  flex-basis: | auto; /* default auto */  
}
```

CSS



## Propriedades dos filhos: Flex-shrink

- **Flex-shrink:** define como os flex-itens podem crescer no espaço disponível remanescente do contêiner pai

```
.item {  
  flex-shrink: 3; /* default 1 */  
}
```

CSS





## Propriedades dos filhos: Flex

- Flex: comando de “atalho” para flex-grow, flex-shrink, e flex-basis (p.ex: default 0 1 auto)

```
.item {  
  flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]  
}
```

Referência: ótimo site para consulta durante o desenvolvimento:

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>.



# Angular Framework

<https://angular.io/>

Google



# Agenda

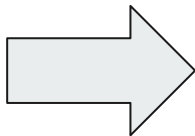
- O que é Angular?
- Configurar o ambiente de desenvolvimento:
  - Instalar o Visual Studio Code;
  - Instalar o NodeJS;
  - Visão geral & instalar o Git;
  - Visão geral & instalar Angular CLI;
- CSS3 FlexBox
- Angular Material & Material Design





# Agenda

- Angular Framework - Principais conceitos:
  - Components;
  - Directives;
  - Pipes;
  - Routes;
  - Services;
  - Forms (Template Driven e Reactive);
  - Http Client.....



**Atividade Prática: Checklist SPA**



## O que é Angular?

- Plataforma e framework utilizado para a criação de SPAs por meio de HTML e TypeScript;
- Uma aplicação Angular é composta por blocos de construção chamados de **components** que são organizados em **NgModules** (1:N);
- Uma SPA Angular também é composta por **services** que são injetados nos **components** e utilizados como dependências -> consequência: modularidade, reusabilidade e eficiência.



## O que é Angular?

- Angular templates: combinam HTML e código Angular para modificar elementos HTML antes que sejam renderizados:
  - Diretivas: permitem que haja o binding entre a classe TypeScript com a lógica da aplicação e os elementos do DOM (modelo de objetos de documentos);
- Existem 2 formas de se realizar o binding:
  - Event binding (binding de eventos);
  - Property binding (binding de propriedades)
- Antes de renderizar a view, o Angular processa as diretivas (directives) e “resolve” os bindings.



## O que é Angular?

- Angular pipes: melhoram a experiência do usuário transformando property binding antes de sua exibição (p.ex: formatação de datas, moedas etc.)
- Angular fornece pipe pré-definidos e você pode também criar seus próprios;



# Configurando o ambiente de desenvolvimento Angular





**IDE** (Integrated Development Environment)

---

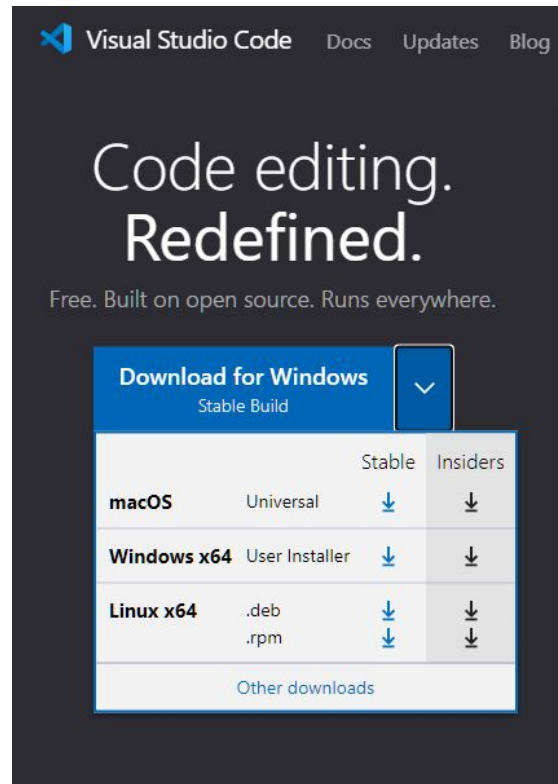


# Instalando a IDE Visual Studio Code

- Sinta-se livre para utilizar uma IDE de sua preferência
- Recomendação: <https://code.visualstudio.com/>
- Extensões que podem ajudar:
  - Auto Rename Tag - Jun Han
  - Bracket Pair Colorizer 2 - CoenraadS
  - Angular Essentials (Version 11) - John Papa



Visual Studio Code



Visual Studio Code Docs Updates Blog

## Code editing. Redefined.

Free. Built on open source. Runs everywhere.

**Download for Windows**  
Stable Build

		Stable	Insiders
<b>macOS</b>	Universal	↓	↓
<b>Windows x64</b>	User Installer	↓	↓
<b>Linux x64</b>	.deb .rpm	↓ ↓	↓ ↓

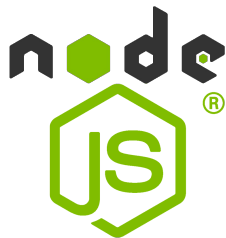
[Other downloads](#)

# Node JS

---

# Instalando Node JS

- Por que instalar o Node JS?
- Instalação: <https://nodejs.org/en/>
- Versão recomendada LTS



## Downloads

Latest LTS Version: 14.16.0 (includes npm 6.14.11)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

LTS Recommended For Most Users	Current Latest Features	
 Windows Installer <small>node-v14.16.0-x64.msi</small>	 macOS Installer <small>node-v14.16.0.pkg</small>	 Source Code <small>node-v14.16.0.tar.gz</small>

Windows Installer (.msi)

Windows Binary (.zip)

macOS Installer (.pkg)

macOS Binary (.tar.gz)

Linux Binaries (x64)

Linux Binaries (ARM)

Source Code

32-bit	64-bit
32-bit	64-bit
64-bit	
64-bit	
64-bit	
ARMv7	ARMv8
node-v14.16.0.tar.gz	

# Versionamento de Código

---



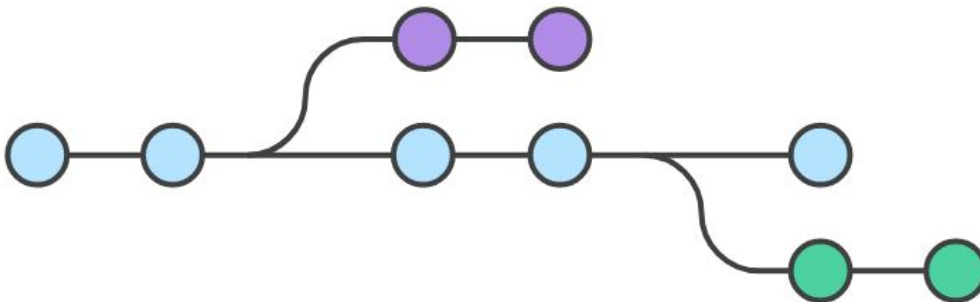
## Git: Visão Geral

- Mais utilizada ferramenta de versionamento de código;
- Gratuita, de código aberto e distribuída;
- É possível trabalhar por meio de linha de comando ou GUI (aplicação desktop);
- Os arquivos são armazenados localmente e em repositórios remotos.

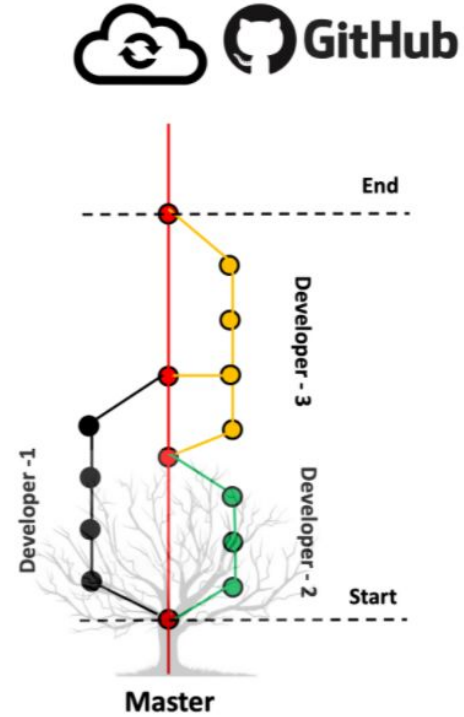
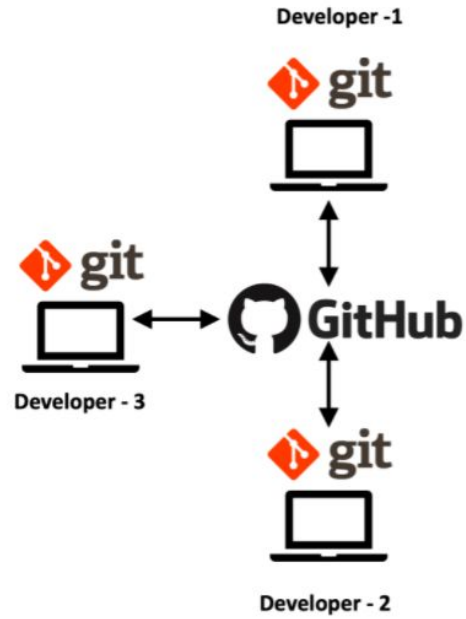


## Git - Principais conceitos:

1. Branch, checkout
2. Stage files
3. Commit files
4. Fetch, pull
5. Merge
6. Push



# Git





## Instalando o Git

- Se você ainda não tem o Git instalado, agora é hora!
- O instalador está disponível para todos SOs no próprio site da ferramenta!
  - <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>



# Devo instalar um Git Desktop GUI?

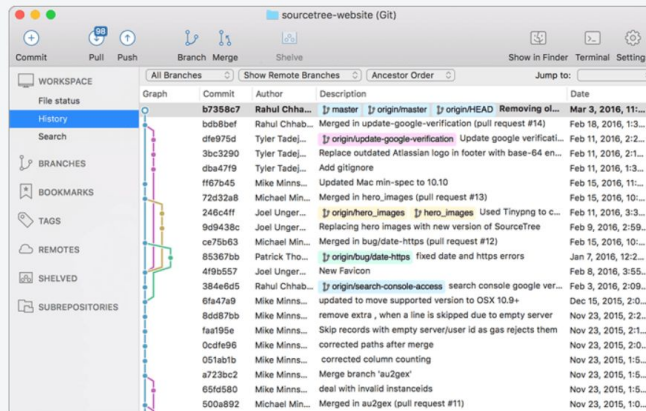
 Sourcetree

[Download free](#)

Simplicity and power in  
a beautiful Git GUI

[Download for Windows](#)

Also available for Mac OS X



A free Git client for Windows and Mac

# Angular CLI

---

# Instalando o Angular CLI

- Angular CLI - ferramenta de linha de comando para criar/desenvolver/manter aplicações Angular
- Abra uma nova janela do Prompt Command e digite:
  - `npm install -g @angular/cli`





# Criando nossa aplicação de Checklist

- Abra uma nova janela do Prompt Command
- Vá até c:\ e crie uma nova pasta para a atividade prática
- Crie uma nova aplicação Angular chamada “checklist-app”
  - c:\udemy-course>ng new checklist-app
  - Selecione “Yes” quando questionado sobre “routing”
  - Selecione “CSS” para o formato de CSS



## Testando a nova aplicação

- Ainda na linha de comando, digite:
  - `ng serve`
- Angular CLI irá compilar o código gerado e servir a aplicação em `http://localhost:4200`
- Abra o browser e navegue até a URL acima;
- **Parabéns!! Você acaba de criar sua primeira aplicação Angular!**



# Limpeza do código

---



# Limpendo app.component.html

- No arquivo src/app/app.component.html;
- Apague todo o conteúdo e mantenha as linhas abaixo:

```
<h1>{{title}}</h1>
```

```
<router-outlet></router-outlet>
```





# Angular Material

<https://material.angular.io/>

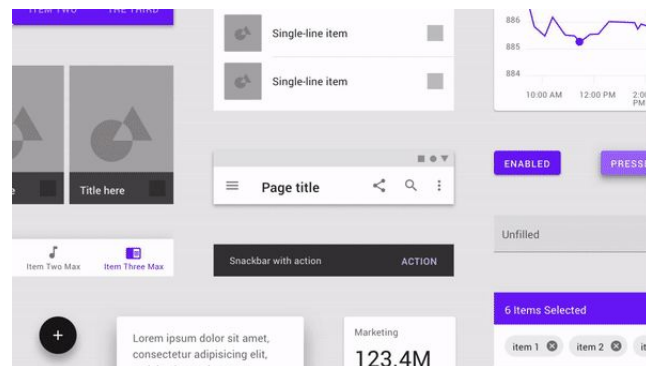
<https://material.io/>



# Material Design

- Criado pelo Google em 2014, o Material Design é um conjunto de regras, guias/recomendações e componentes focados nas melhores práticas para criação de websites e aplicativos;
- Objetivo: ter um design mais liberal com o uso de *grids-based* layouts, animações e transições responsivas, *padding* e efeitos de profundidade como luz e sombra;
- Foco total com ênfase na legibilidade e acessibilidade.

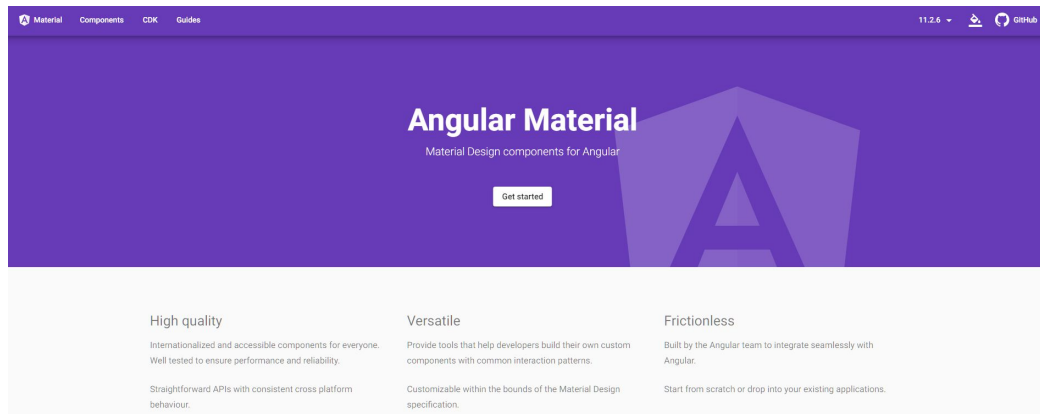
“Material Design is a unified system that combines theory, resources, and tools for crafting digital experiences.” - Google



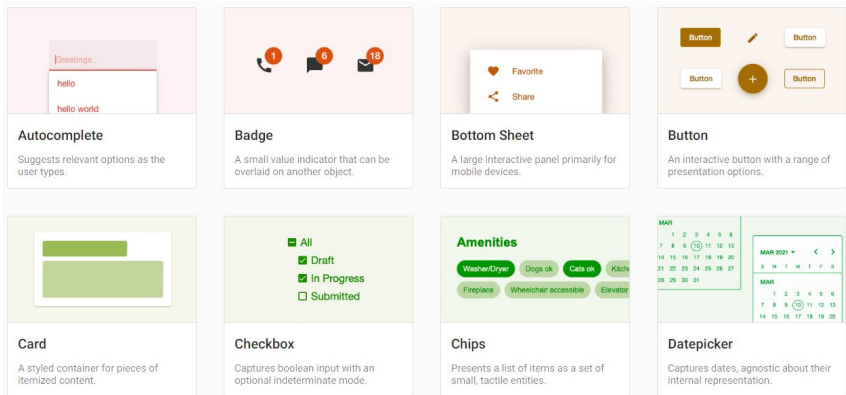


# Angular Material

- Componentes estilizados com Material Design para aplicações Angular;
- Amplamente testados e com suporte para internacionalização (i18n);
- Foco em performance e confiabilidade;
- Componentes e temas customizáveis.



# Angular Material: Componentes



[Material](#) [Components](#) [CDK](#) [Guides](#)

## Autocomplete

[OVERVIEW](#) [API](#) [EXAMPLES](#)

Checkbox

Chips

Datepicker

Dialog

Divider

Expansion Panel

Form field

Grid list

Icon

Input

List

Menu

Paginator

The autocomplete is a normal text input enhanced by a panel of suggested options.

### Simple autocomplete

Start by creating the autocomplete panel and the options displayed inside it. Each option should be defined by a `mat-option` tag. Set the `value` attribute of the `mat-option` tag to the value you'd like the value of the text input to be when that option is selected.

```
<mat-autocomplete #auto="matAutocomplete">
  <mat-option *ngFor="let option of options" [value]="option">
    {{option}}
  </mat-option>
</mat-autocomplete>
```

Next, create the input and set the `matAutocomplete` input to refer to the template reference we assigned to the autocomplete. Let's use the `ReactiveFormsModule` to track the value of the input.

Note: It is possible to use template-driven forms instead, if you prefer. We use reactive forms in this example because it makes



# Instalando o Angular Material

- Abra uma janela do Prompt Command, vá até a pasta checklist-app (criada anteriormente);
- Digite o seguinte comando:
  - `ng add @angular/material`
- Quando questionado sobre o tema, selecione “Indigo & Pink”
- Quando questionado sobre a tipografia global, selecione YES
  - Essa aplicação aplica a fonte padrão do Material Design, altura de linha etc.
- Quando questionado sobre animações do browser, selecione YES

# Components

---



# Angular Components

- Blocos de construção de uma aplicação Angular
- Formado por uma classe TypeScript, um arquivo HTML (template) e estilo (externo ou inline)
- Para criar um component usando Angular CLI:
  - `ng generate component [component-name]`
  - `ng g c [component-name]`
- Por padrão, o Angular CLI irá criar os seguintes arquivos:
  - Uma pasta com o nome do componente
  - Um classe Type Script para o component: `[component-name].component.ts`
  - Um HTML de template: `[component-name].component.html`
  - Um arquivo de CSS: `[component-name].component.css`
  - Um arquivo de especificação de teste: `[component-name].component.spec.ts`



# Angular Components

- Decorator: @Component
  - Selector: 'app-component-name'
  - Template: '<h1>Meu Componente</h1>'
  - templateUrl: './meu-component.component.html'
  - Styles: ['h1 { color: red }']
  - styleUrls: ['./meu-component.component.css']
- 
- Ciclo de vida e interação entre componentes serão abordados mais adiante com a atividade prática!



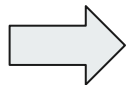
# Templates

---



# Templates

- Basicamente um arquivo de HTML com código Angular (sintaxe própria);
- Manipulação do DOM (data object model):
  - Template functions (template functions)
  - Variáveis;
  - Event listening (escuta de eventos);
  - Data binding (comunicação de dados);
- Quando criar o template, não é necessário adicionar as tags `<html>` e `<body>`;



DOM: representação estruturada de como o HTML é lido pelo browser



## Interpolação de strings (interpolation)

- Permite exibir uma variável no corpo do HTML
- Sintax: {{nome\_da\_variavel}}
- Posso utilizar expressões numéricas? sim, após o resultado, são exibidas como string!

```
<p>{{meuTxto}}</p>  
<div></div>
```



## Declarações de template (template statement)

- É possível chamar métodos ou propriedades em noção classe de dentro do template

```
<button (click)="onSave($event)">Salvar</button>
```

```
<button *ngFor="let item of Items" (click)="changeItem(item)">{{item.desc}}</button>
```

```
<form #myForm (ngSubmit)="onSubmit(myForm)"> ... </form>
```



# Binding (conexão/comunicação no template)

- Permite que o template mantenha-se atualizado em relação ao estado da aplicação
- Ex: DOM - **propriedade** disabled

```
<button [disabled]="isDisabled">OK</button>
```

- Ex: HTML - **atributo** disabled

```
<input [attr.disabled]="condition ? 'disabled' : null">
```

- Two-way data binding

```
<input [(ngModel)]="description">
```

# Angular Directives

---



# Diretivas (directives)

- Built-in directives: adicionam comportamento aos elementos da sua aplicação Angular
- Tipos:
  - Componentes - diretivas com template (mais comum)
  - Atributos - alteram o comportamento ou aparência de um elemento
    - NgClass: adicionam um classe à uma tag HTML - `[ngClass] = "condição ? 'x': 'y'"`
    - NgModel: two-way data binding - **iremos mais a fundo na seção de formulários**
    - NgStyle: adicionam um estilo à uma tag HTML - `[ngStyle] = "inlineStyle"`
  - Estrutural - modificam o DOM
    - NgIf: `*ngIf = "condição"`
    - NgFor: `*ngFor = "let item of items; let i=index"`
    - NgSwitch: `[ngSwitch] = "condição" / *ngSwitchCase="valor" / *ngSwitchDefault`

# Directivas Hands-ON!

---





## Criando nosso 1º componente: Application Header

- Abra uma janela do Prompt Command, vá até a pasta checklist-app (criada anteriormente);
- Dentro da pasta da aplicação, digite:
  - `ng g c[component] header`
- Angular CLI will create the following files:
  - `header.component.css`: used to provide styling for this component;
  - `header.component.html`: for adding HTML elements;
  - `header.component.spec.ts`: for writing test cases;
  - `header.component.ts`: to add the Typescript-based logic

## Criando um novo módulo: Material Module

- Abra uma janela do Prompt Command, vá até a pasta checklist-app (criada anteriormente);
- Crie um novo arquivo chamado material.module.ts dentro da pasta sr/app;
- A partir de agora, tudo componente do Angular Material que formos utilizar será adicionado dentro dos arrays “imports” e “exports”;
- Também adicione esse novo módulo dentro do módulo principal da aplicação utilizando o array “imports”;

```
src > app > material.module.ts > MaterialModule
1  import {NgModule} from '@angular/core';
2
3  @NgModule({
4    imports: [
5    ],
6    exports: [
7    ]
8  })
9  export class MaterialModule {}
```

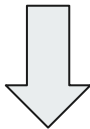


## Modificando o componente Header

- Vá até o arquivo `/app/src/material.module.ts`;
- Adicione as seguintes linhas no topo do arquivo (junto aos **imports**):
  - `import {MatToolbarModule} from '@angular/material/toolbar';`
  - `import {MatIconModule} from '@angular/material/icon';`
  - `import {MatButtonModule} from '@angular/material/button';`
- Altere os arrays “imports” e “exports” adicionando as seguintes linhas:
  - `MatToolBarModule`
  - `MatIconModule`
  - `MatButtonModule`

## Adicionando o Header na aplicação

- Vá até o arquivo src/app/app.component.ts e inclua o header:



```
src > app > app.component.html > router-outlet
1 | <app-header></app-header>
2 |
3 | <h1>{{title}}</h1>
4 | <router-outlet></router-outlet>
```

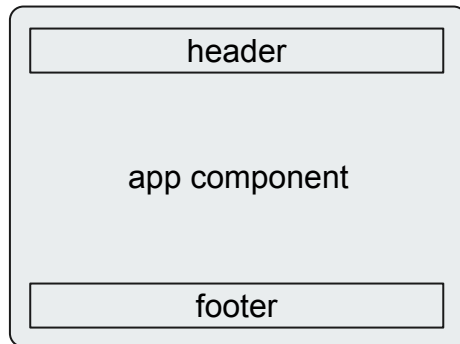
```
> app > header > header.component.ts > ...
1 import { Component, OnInit } from '@angular/core';
2
3 @Component({
4   selector: 'app-header',
5   templateUrl: './header.component.html',
6   styleUrls: ['./header.component.css']
7 })
8 export class HeaderComponent implements OnInit {
9
10   constructor() { }
11
12   ngOnInit(): void {
13   }
14
15 }
16
```

# Application Footer



## Criando o Footer da aplicação

- Abra uma janela do Prompt Command, vá até a pasta checklist-app (criada anteriormente);
- Digite o comando a seguir:
  - `ng g c footer`
- Como já é esperado, o Angular CLI irá criar todos os arquivos relacionados ao componente



# Sidenav App Component

---

# Interação entre componentes

---

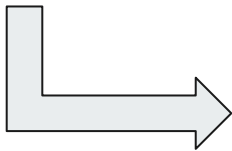




## Interação Pai -> Filho

- Possível por meio de input binding
  - @Input decorator

```
export class ChildComponent {  
  @Input('optional-alias') fromFather: fromFatherValue;  
}
```



```
@Component({  
  selector: 'app-parent',  
  template: `  
    <p>This is the father component</p>  
    <app-child [fromFather]="fromFather">  
  </app-child>  
  `,  
})  
export class ParentComponent {  
  fromFather: 'FromFather';  
}
```



## Interação Filho-> Pai

- O componente pai “escuta” eventos do filho
- O filho emite um evento utilizando um EventEmitter
- A propriedade do **event emitter** utiliza um decorator @Output para que haja um binding no componente pai

# Angular Routing

---



# Angular Routing

- SPA (Single Page Application): esconder/exibir conteúdo sem a necessidade de executar uma chamada ao servidor para recarregar o conteúdo;
- O módulo **Router** fornece um serviço que define diferentes formas de se navegar em uma aplicação:
  - Barra de URL do browser;
  - Links na página;
  - Botões de avançar e voltar do browser;
- Padrões de URL são associados a um “estado” de uma view;
- Index.html - possui uma tag `<base href="/">` que indica qual a pasta padrão da aplicação.



# Angular Routing

- AppRoutingModule é importado por padrão dentro do AppModule pelo Angular CLI quando selecionado a opção “Yes” no processo de criação (bootstrap) da SPA;
- As rotas são definidas dentro de um array e podem ser acessadas utilizando o atributo **routerLink** em um tag HTML

```
const routes: Routes = [  
  { path: 'first-component', component: FirstComponent },  
  { path: 'second-component', component: SecondComponent },  
];
```



# Angular Routing

- A ordem de definição de uma rota é importante: primeira coincidência vence wins
  - Como definir? Uma lista de rotas seguido de uma rota padrão vazia
  - Uma rota coringa pode ser adicionada ao fim do módulo caso nenhuma outra rota seja encontrada;

```
const routes: Routes = [  
  { path: '', component: HomeComponent },  
  { path: 'first-component', component: FirstComponent },  
  { path: 'second-component', component: SecondComponent },  
  { path: 'error', component: ErrorComponent },  
  { path: '**', component: PageNotFoundComponent },  
];
```

## Criando o componente da Home e adicionando rotas

- Abra uma janela do Prompt Command, vá até a pasta checklist-app (criada anteriormente);
- Digite o seguinte comando:
  - `ng g c home`
- Como já é esperado, o Angular CLI irá criar todos os arquivos relacionados ao componente





## Adicionando rotas para o componente Home

- Vá até o arquivo `src/app/app-routing.module.ts`:
- Adicione uma rota para o HomeComponent

```
const routes: Routes = [  
  { path: '', component: HomeComponent}  
];
```



# Rotas Aninhadas & Guards

---



## Rotas Aninhadas

- Útil para criação de rotas relativas à um componente específico:
- Configurável através da adição de uma nova tag `<router-outlet></router-outlet>` no componente “host”:

```
const routes: Routes = [  
  {  
    path: 'first-component',  
    component: FirstComponent, // this component has a <router-outlet> in the template  
    children: [  
      {  
        path: 'child-a', // child route path  
        component: ChildAComponent, // child route component that the router renders  
      },  
      {  
        path: 'child-b',  
        component: ChildBComponent, // another child route component that the router renders  
      },  
    ],  
  },  
];
```



# Angular Routing

- Como eu posso acessar uma rota e ler os parâmetros a partir de um componente?
  - Utilizando o componente ActivatedRoute

```
import { Router, ActivatedRoute, ParamMap } from '@angular/router';

ngOnInit() {
  this.route.queryParams.subscribe(params => {
    this.name = params['name'];
  });
}
```



## Route Guards - Protegendo rotas (controle de acesso)

- Router guards são utilizados para proteger URL restritas;
- Como criar um guardião de rota com Angular CLI?
  - Digite o comando abaixo em uma janela do Prompt Command:
    - `ng generate guard [guard-name]`

```
{  
  path: '/sample-path',  
  component: RestrictedComponent,  
  canActivate: SampleGuard,  
}
```



```
export class SampleGuard implements CanActivate {  
  canActivate(  
    next: ActivatedRouteSnapshot,  
    state: RouterStateSnapshot): boolean {  
    // your logic to protect this route here  
  }  
}
```

# Prática: Componente Categoria

---



# Criando o componente Categoria

- Adicione um novo componente Categoria:
  - `ng g[enerate] c[component] category --module app`
- Modifique o arquivo `app-routing.module.ts`:
  - Adicione uma nova rota para o componente recém criado
- Atualize o arquivo `app.component.ts`:
  - Modifique o `side navbar` e adicione links para “Home” and “Categoria”
  - Selecione 2 novos ícones para os links criados em: <https://material.io/icons/>
- Adicione um `MatDivider` entre os elementos `mat-list-items`:
  - Modifique o arquivo `material.module.ts` e adicione o novo module do Angular Material:
    - `import {MatDividerModule} from '@angular/material/divider';`



## Criando o componente Categoria

- No arquivo `material.module.ts`, adicione os módulos `MatCardModule` e `MatTableModule`:
  - `import {MatCardModule} from '@angular/material/card';`
  - `import {MatTableModule} from '@angular/material/table';`
- Adicione uma nova pasta em `src/app` para as classes de **model** da aplicação.;
- Adicione um novo arquivo chamado `category.ts` dentro da nova pasta criada:
  - Adicione um atributo público chamado `'guid'` como `string`;
  - Adicione um atributo chamado `'name'` como `string`;

# Porque um GUID?

---





## O que é um GUID?

- Identificador único global (unique global identifier)
- Um número de 128 bits
  - Únicos
  - Não dependem de uma entidade reguladora para emissão
- Para facilitar a leitura por humanos, é exibido no formato canônico em hexadecimal
  - 123e4567-e89b-12d3-a456-426655440000

# Angular Forms

---



# Angular & Forms

- Angular possui 2 diferentes formas de lidar com formulários:
  - Reactive e template-driven
- **Reactive forms (formulário reativo):**
  - Acesso direto e explícito ao objeto de modelo do formulário: mais robusto, escalável, reutilizável e testável;
  - Ideal para aplicações que utilizem N formulários como diversos campos e validações
- **Template-driven forms (formulário orientado ao template):**
  - Dependente de diretivas no template para manipulação/alteração de dados do modelo do formulário
  - Ideal para formulários únicos/simples;



## Angular Reactive Forms (formulário reativo)

- Abordagem orientada à modelo para lidar com as entradas do formulário;
- O modelo do formulário é definido de forma direta na classe do componente;
- A diretiva [formControl] é utilizada de forma explícita para criar uma instância de FormControl relacionada ao elemento do formulário;
- A view é diretamente “linkada” ao ao modelo do formulário - todas as alterações acontecem de forma síncrona

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

@Component({
  selector: 'app-reactive-favorite-color',
  template: `
    Favorite Color: <input type="text"
    [formControl]="favoriteColorControl">
  `
})
export class FavoriteColorComponent {
  favoriteColorControl = new FormControl('');
}
```



# Angular Template-Driven Forms

- O modelo do formulário está implícito;
- É necessário utilizar a diretiva NgModel para criar e controlar a instância FormControl associado ao elemento do formulário;
- Todo fluxo de dados entre a view e a classe do componente é controlado pela diretiva NgModel.

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

@Component({
  selector: 'app-template-favorite-color',
  template: `
    Favorite Color: <input type="text"
    [(ngModel)]="favoriteColor">
  `
})
export class FavoriteColorComponent {
  favoriteColor = '';
}
```



# Angular Forms - Data Validation

<https://angular.io/guide/form-validation>

# Apagar categoria

---



## Criando um Pop-up (modal) genérico de diálogo

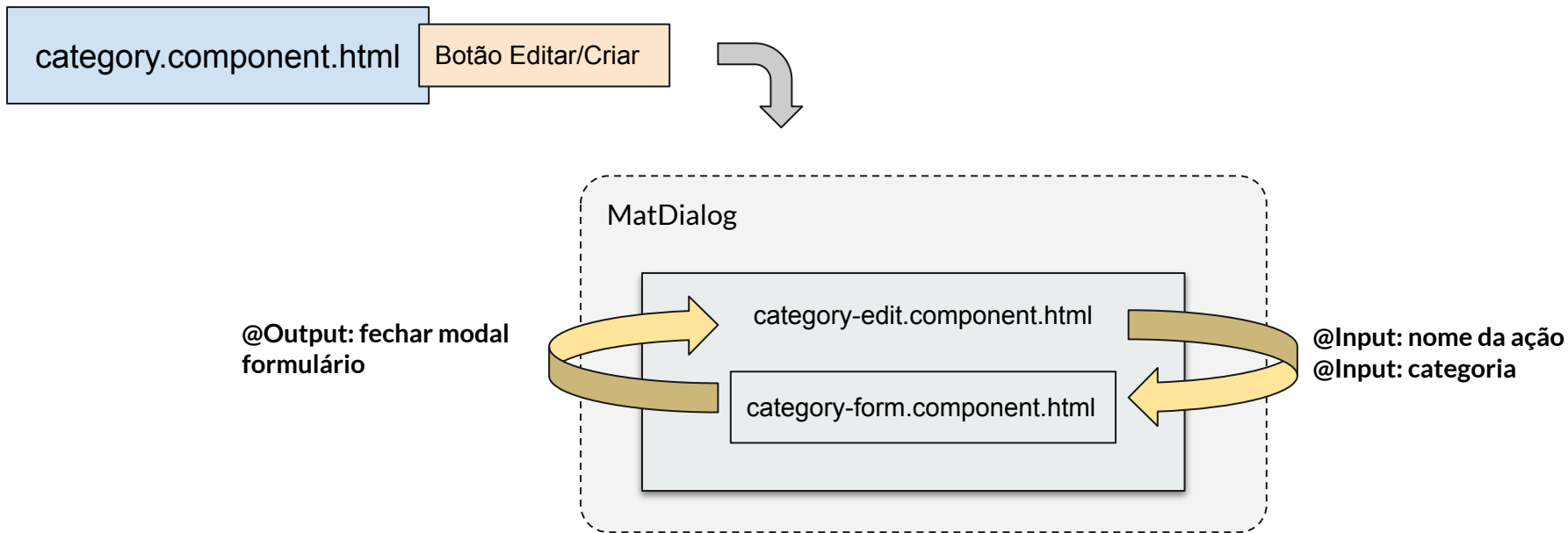
- É necessária a criação de uma modal “genérica” de confirmação;
- Adicione um novo componente chamado Dialog:
  - `ng g c dialog --module app`
- Importe o módulo MatDialogModule no arquivo material.module.ts
  - `import { MatDialogModule } from '@angular/material/dialog';`
- Atualize o arquivo category.component.ts para fazer da nova modal
- Atualize o código HTML do novo componente:
  - Botão de confirmação
  - Botão de negação
  - Mensagem



# Criando o formulário de edição da categoria

---

## Criando o componente Editar Categoria

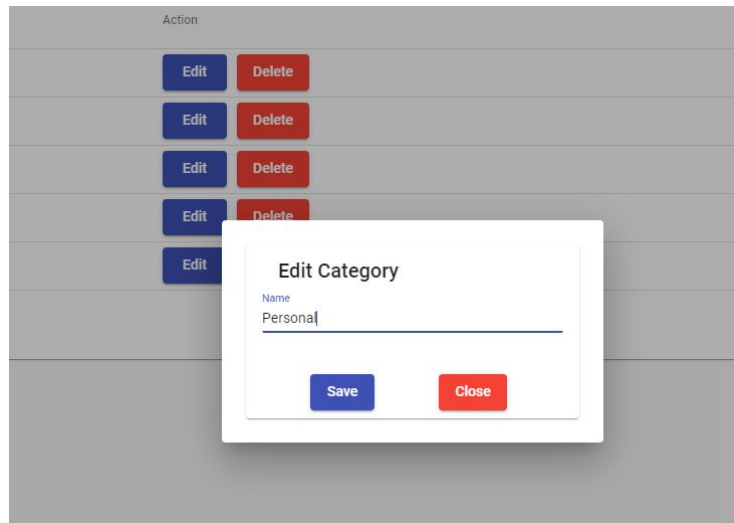


## Criando o componente Editar Categoria

- Adicionar um novo componente chamado Edit Category:
  - `ng g[enerate] c[omponent] category-edit --module app`
- Esse componente será utilizado como um contêiner para um segundo componente chamado CategoryFormComponent



- Formulário compartilhado pelo Angular Material para editar e criar uma categoria





## Criar o componente de formulário para a Categoria

- Adicione um novo componente chamado `CategoryFormComponent`:
  - `ng g[enerate] c[omponent] category-form --module app`
- Adicione o módulo **ReactiveFormsModule** no arquivo `app.module.ts`:
  - `import { ReactiveFormsModule } from '@angular/forms';`
- Adicione o módulo **MatInputModule** no arquivo `material.module.ts`:
  - `import { MatInputModule } from '@angular/material/input';`
- Crie um form reativo no HTML do componente recém criado: `category-form.component.html`;
- Importante: não se esqueça também de adicionar os novos módulos no respectivo “imports” array do módulo existente



## Configurado o MatDialog para editar uma categoria

- Importe o **MatDialogModule** no arquivo `material.module.ts`:
  - `import { MatDialogModule } from '@angular/material/dialog';`
- Adicione o **MatDialog** no construtor da classe `category.component.ts`;
- Adicione todo o código necessário para abrir o componente **category-edit.component.ts** no MatDialog



## Alterando o componente de edição de Categoria

- No arquivo de HTML `category-edit.component.html`
- Adicione a tag `<app-category-form>` e crie 3 atributos para comunicação entre os componentes:
  - **Input:** `actionName` (string) - valor 'Editar'
  - **Input:** `editableCategory` (category.ts) - irá passar a categoria selecionada para o formulário
  - **Output:** `closeFormEvent` (event emitter) - irá notificar o pai que o formulário foi fechado
- Atualize a classe do componente com todo o código para lidar com a comunicação entre pai & filho



## Criando o formulário reativo para criação/edição da Categoria

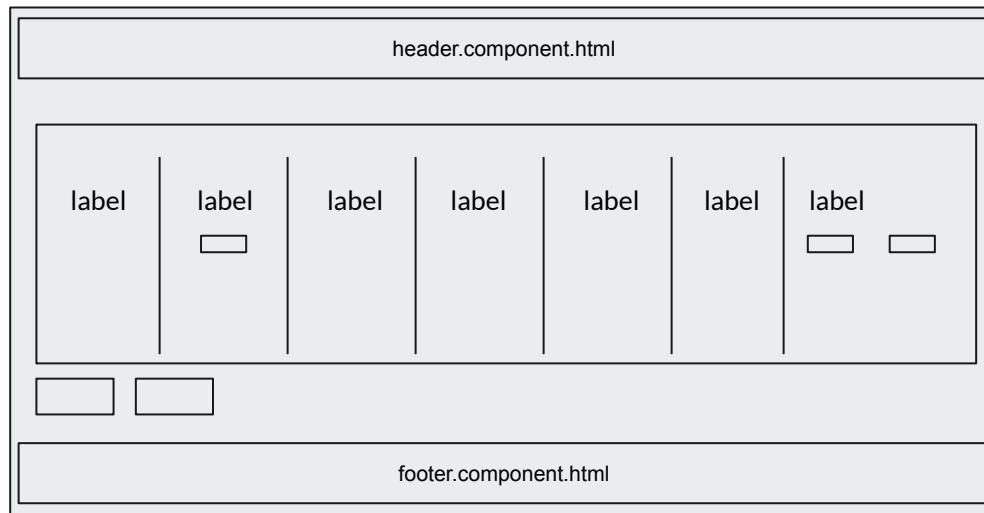
- É hora de criar o formulário para editar e criar uma categoria;
- Alterar os seguintes arquivos:
  - `category-form.component.ts`
  - `category-form.component.html`

# Componente Checklist

---



# Criando o componente Checklist





## Criando o componente Checklist

- Abra uma nova janela do Prompt Command e digite o seguinte comando:
  - `ng g c checklist --module app`
- Adicione uma nova rota para o checklist no arquivo **app-routing.module.ts**;
- Atualize o side navbar do arquivo `app.component.html` com um novo link para o componente criado
  - Sinta-se livre para escolher um ícone que mais lhe agrada
- Atualize os arquivos `checklist.component.ts` e `checklist.component.html` com código necessário para visualizar uma tabela de itens do checklist

# Angular Pipes

---



# Angular Pipes

- Expressões de template (template expressions)
- Utilizadas para transformar:
  - Strings
  - Moedas
  - Datas
  - Etc. etc.
- Suporte a i18n;



# Angular Pipes

- Mais comuns:
  - DatePipe;
  - UpperCasePipe;
  - LowerCasePipe;
  - CurrencyPipe;
  - DecimalPipe;
  - PercentPipe;

Para mais exemplos: <https://angular.io/api/common#pipes>



# Angular Pipes

- Como utilizar?
  - Parâmetros são separados por `:` e pode haver mais de um
  - Podem ser encadeados

```
<span>Meu aniversário é no dia {{ birthday | date }}</span>
```

```
<p>Saldo da conta corrente: {{total | currency: 'EUR'}}</p>
```

```
<span>Hoje é {{today | date | uppercase}}</span>
```



# Angular Pipes

- Criando seu próprio Pipe? implemente a interface **PipeTransform**

```
custom.pipe.ts:
{{ value | exponentialStrength: 2 }}

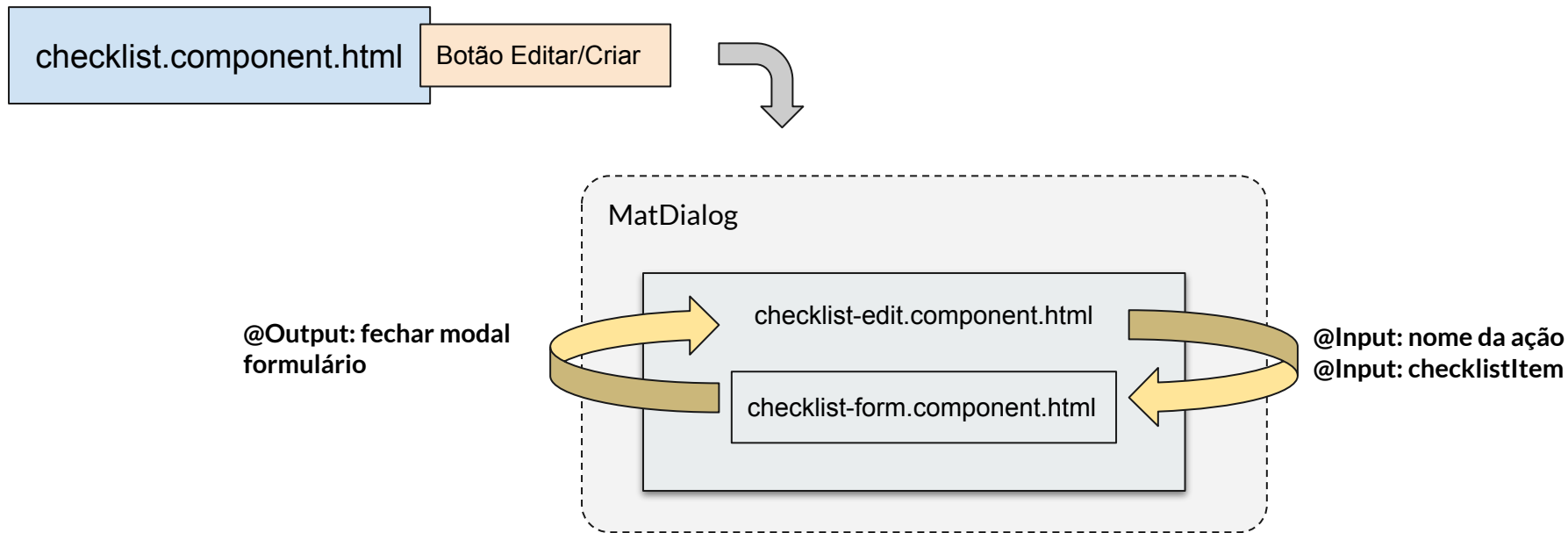
@Pipe({name: 'exponentialStrength'})
export class ExponentialStrengthPipe implements PipeTransform {
  transform(value: number, exponent?: number): number {
    return Math.pow(value, isNaN(exponent) ? 1 : exponent);
  }
}
```

# Criando o formulário de edição do checklist

---



## Criando o componente de edição do item do Checklist





## Criando o componente Editar Item do Checklist

- Abra uma nova janela do Prompt Command;
- Adicione um novo componente chamado checklist -edit:
  - `ng g[enerate] c[omponent] checklist-edit --module app`
- Esse componente será utilizado como um contêiner para um segundo componente chamado ChecklistFormComponent.



## Criando o componente Checklist Form

- Abra uma nova janela do Prompt Command e digite:
  - `ng g c[component] checklist-form --module app`
- Altere o arquivo `material.module.ts` adicionando os seguintes módulos:
  - `import {MatSelectModule} from '@angular/material/select';`
  - `import {MatDatepickerModule} from '@angular/material/datepicker';`
  - `import {MatCheckboxModule} from '@angular/material/checkbox';`
  - `import {MatNativeDateModule} from '@angular/material';`
- Crie o formulário reativo para o modelo do item do checklist.



## Configurado o MatDialog para editar um item do checklist

- O módulo **MatDialogModule** já foi importado para categoria!
- Adicione o **MatDialog** no construtor da classe `checklist.component.ts`;
- Adicione todo o código necessário para abrir o componente **checklist-edit.component.ts** no MatDialog.



## Alterando o componente de edição do item do Checklist

- No arquivo de HTML `checklist-edit.component.html`
- Adicione a tag `<app-checklist-form>` e crie 3 atributos para comunicação entre os componentes:
  - **Input:** `actionName (string)` - valor 'Editar'
  - **Input:** `editableChecklistItem (category.ts)` - irá passar a item do checklist selecionado para o formulário
  - **Output:** `closeFormEvent (event emitter)` - irá notificar o pai que o formulário foi fechado
- Atualize a classe do componente com todo o código para lidar com a comunicação entre pai & filho

# RXJS - Reactive Programming

---



# Programação Reativa

- Paradigma de programação assíncrona:
  - Data streams;
  - Propagação de eventos;
- RxJS (Reactive Extensions for JavaScript) - biblioteca da programação reativa que utiliza “observables” na criação/utilização de chamadas assíncronas (<https://rxjs.dev/guide/overview>);
- RxJS fornece uma variedade de funções para criar e manipular observables;
  - Operadores: filter(), map(), concat()
    - Recebem um observable -> executam uma determinada ação -> retornam um novo observable



## Observable X Promises

Observables	Promises
Emitem 1:N eventos	Emite apenas um evento.
Executam apenas quando subscritas.	São executadas logo após sua criação.
Canceláveis: unsubscribe()	Não canceláveis.
Fornecer operadores: filter(), forEach(), reduce(), retry(), map()	Não fornece operadores.
Os erros são retornados aos subscribers.	Os erros são cascadeados para as promises filhas.



# Ciclo de vida de um componente

---



# Ciclo de vida de um componente Angular

3 fases do ciclo de vida de um componente ou diretiva:

1. Instanciação do componente, view e componentes filhos
2. Detecção de mudanças - data bind e atualização de valores;
3. Destruição do componente e remoção do DOM;



# Lifecycle Hooks

- Interfaces disponíveis para serem implementadas quando é necessário interceptar o ciclo de vida de um componente.
- Cada interface fornece apenas um método para ser implementado - ex: `ngOnInit()`
- Os métodos implementados a partir das interfaces são chamados logo após a chamada do construtor da classe do componente;
- All lifecycle hook methods: <https://angular.io/guide/lifecycle-hooks>



## ngOnInit

- Ideal para inicializações complexas fora do construtor do componente;
  - Construtores devem ser simples e não devem chamadas ao BE para retornar dados
  - Devem ser utilizados apenas para inicialização de variáveis locais
- O bind the propriedades por meio de *inputs* não acontece até que o construtor seja finalizado;
- Invocado apenas uma vez!



## ngOnChanges & ngOnDestroy

- **ngOnChanges:**
  - Útil para quando é necessário monitorar alterações nas propriedades decoradas com *input*
  - É invocado pela primeira vez antes do `ngOnInit`;
- **ngOnDestroy:**
  - Ideal para a execução de código/lógica de limpeza;
    - Ex.: remover a subscrição de um *observable*
  - Executado antes da destruição do componente
  - Útil também para a limpeza de memória.

# Criando os serviços

---



# Injeção de dependências no Angular

- Dependências: objetos ou serviços que um classe necessita para uma determinada função;
- Injeção de dependências?
  - Ganho de flexibilidade e modularidade
  - Ocorre pelo construtor de um componente
- Decorador: @Injectable
  - providedIn: escopo em que a dependência estará disponível



## Criando o serviço Categoria

- Abra uma nova janela do Prompt Command;
- Vá até a pasta: `/src/app/services`
- Digite o seguinte comando:
  - `ng g s[ervice] category`
- Angular CLI irá gerar todos os arquivos do novo serviço
- Mova o array **CATEGORY\_DATA** do componente `category.component.ts` para o novo serviço `category.service.ts`;
- Adicione o serviço `category.service.ts` no construtor do componente `category.component.ts`;
- Altere o método `ngOnInit` para consumir os dados do serviço.





## Criando o serviço Checklist

- Abra uma nova janela do Prompt Command;
- Vá até a pasta: `/src/app/services`
- Digite o seguinte comando:
  - `ng g s[service] checklist`
- Angular CLI irá gerar todos os arquivos do novo serviço
- Mova o array **CHECKLIST\_DATA** do componente **checklist.component.ts** para o novo serviço **checklist.service.ts**;
- Adicione o serviço **checklist.service.ts** no construtor do componente **checklist.component.ts**;
- Altere o método **ngOnInit** para consumir os dados do serviço.

# Adicionando o Snackbar Service

---



## Adicionando o serviço de SnackBar

- Vamos criar um serviço para exibir o SnackBar do Angular Material
  - <https://material.angular.io/components/snack-bar/overview>
- Primeiro passo:
  - Implementar a lógica para chamar os serviços quando salvar e editar uma categoria ou item do checklist;
  - A lógica será adicionada nos componentes de formulário;
- Criar um novo serviço de SnackBar;
  - Criar um método genérico para exibir o SnackBar
- Importar o SnackBar no material.module.ts
- Ao fechar o modal do formulário, exibir uma mensagem do SnackBar;
- Implementar o código de limpeza do formulário.

# Finalizando a Home

---

## Atualizando o componente Home

- Adicione uma nova imagem para a nossa landing page;
- Personalize o título da landing page.

