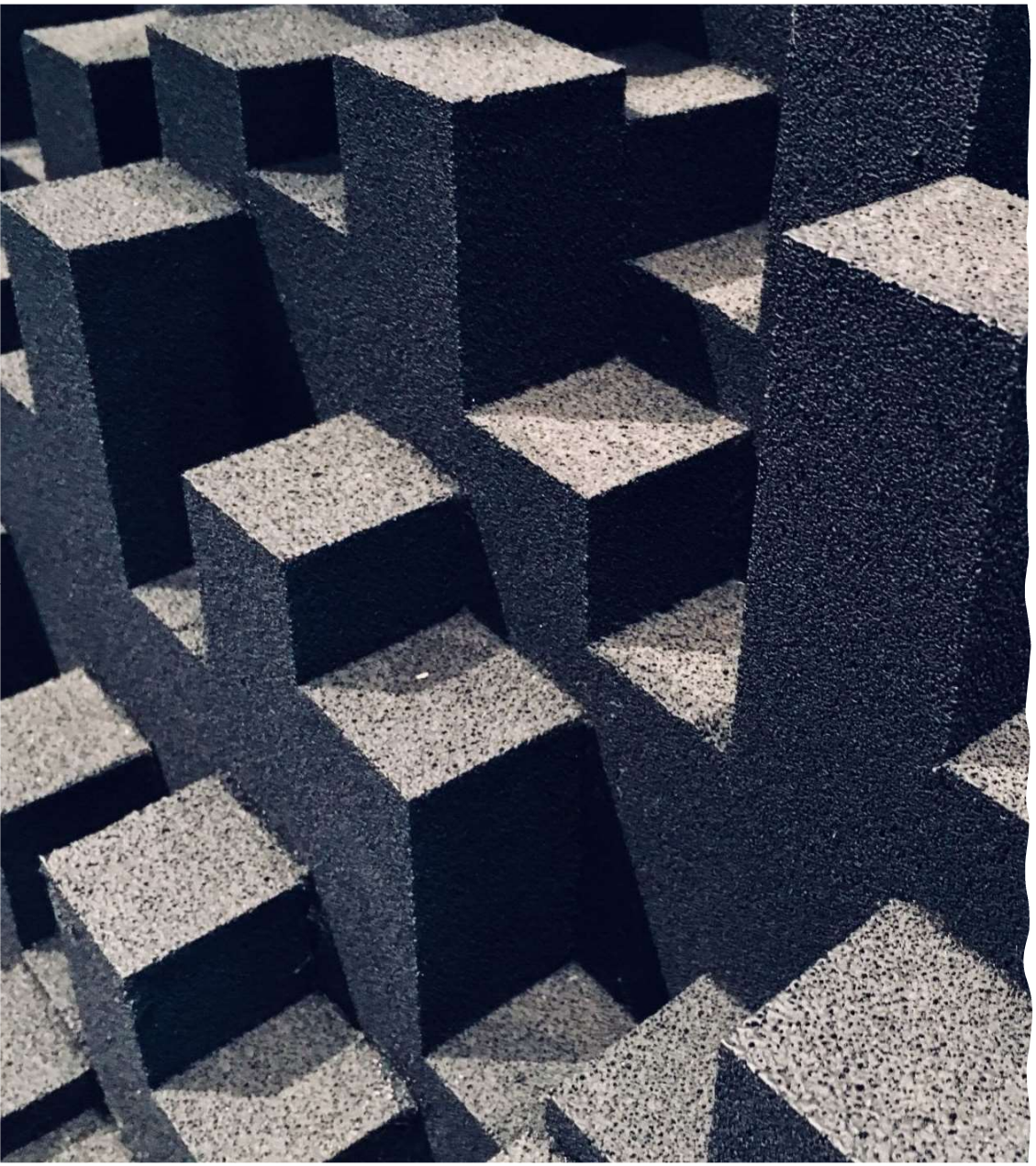
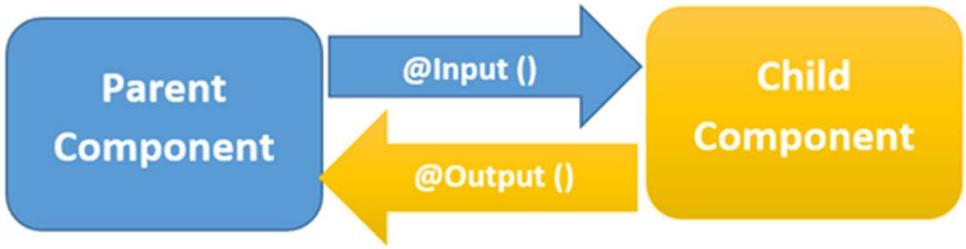


Angular

Comunicação entre diferentes componentes, criação de diretivas personalizadas

Comunicação entre componentes

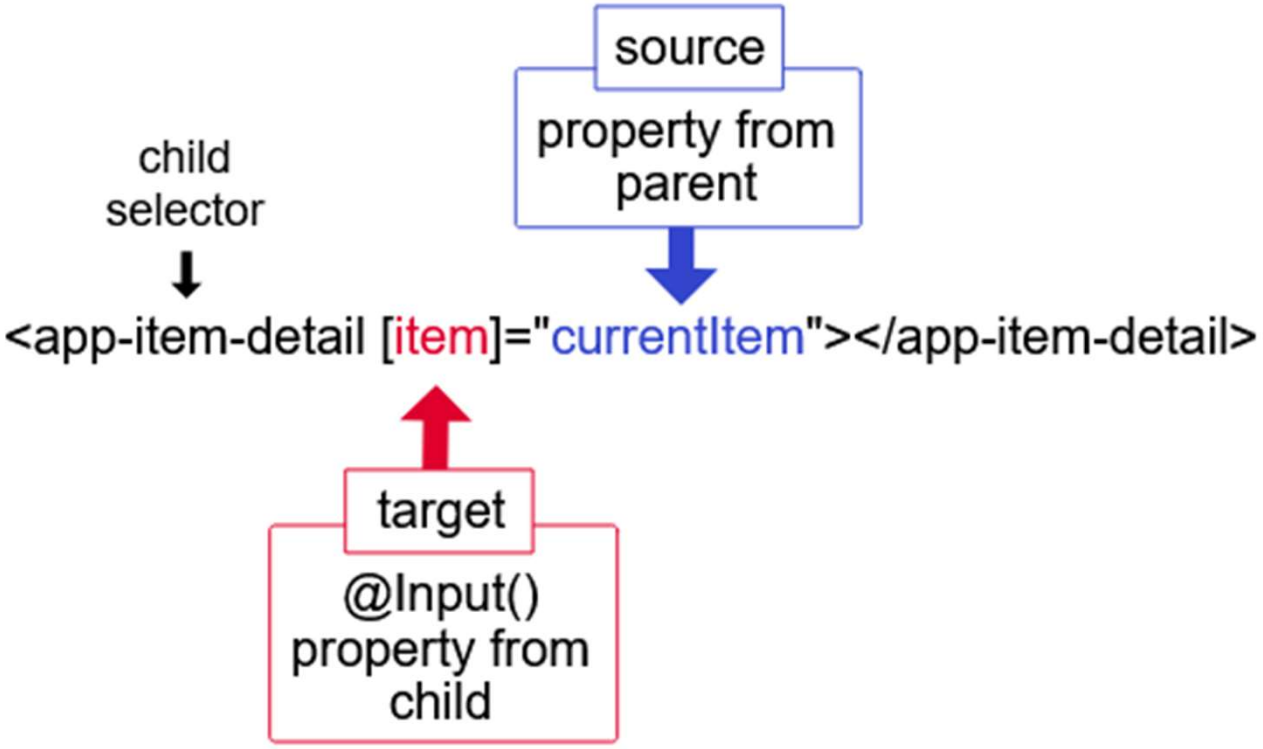
- Componentes são os blocos de construção da interface do usuário de um aplicativo;
- Normalmente, eles precisam se comunicar entre si, passando dados de um componente para outro;
- Existem diversas maneiras de se comunicar entre componentes;
  - @Input Decorator
  - @Output Decorator



@Input Decorator

- Permite que a passagem de dados de um componente pai para um componente filho;
- Estabelece um mecanismo de compartilhamento de informações entre componentes, possibilitando a criação de componentes de UI reutilizáveis e modulares;
- Ao aplicar o decorador @Input a uma propriedade em um componente filho, é sinalizado que essa propriedade pode receber dados do componente pai;
- O componente pai pode então vincular-se (bind) a essa propriedade e transmitir dados ao componente filho.

@Input Decorator



```
tarefas_appComponents = ['Comprar pão', 'ir ao dentista', 'ir ao supermercado'];
```

app.componente.ts

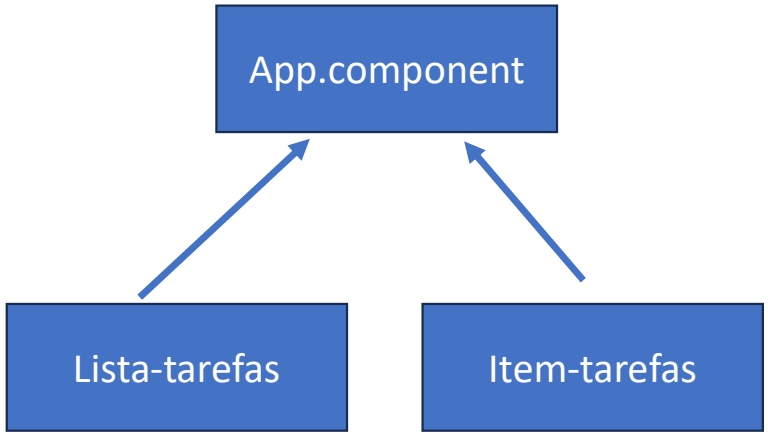
```
<app-lista-tarefas [listaTarefas_listaTarefasComponent]="tarefas_appComponents"
></app-lista-tarefas>
```

app.componente.html

```
@Input() listaTarefas_listaTarefasComponent: string[] = [];
```

lista-tarefas.componente.html

- @Input Decorator



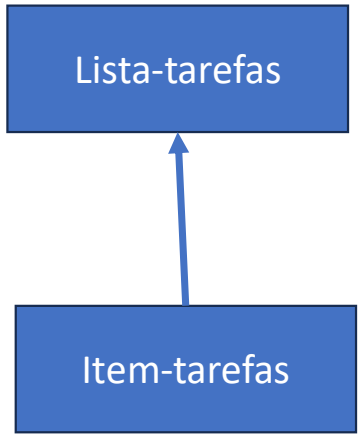
@Input Decorator

```ts
export class ItemTarefasComponent {
  @Input() itemTarefa: string = '';
}
```

item-tarefas.component.ts

lista-tarefas.component.html

```html
<div *ngFor="let tarefa of listaTarefas_listaTarefasComponent">
  <app-item-tarefas [itemTarefa]="tarefa"></app-item-tarefas>
</div>
```



```
@Output() tarefaAdicionada = new EventEmitter<string>();
```

Classe EventEmitter

- Usado em componentes com a diretiva @Output para emitir eventos personalizados de forma síncrona ou assíncrona;
- Registrar manipuladores (handlers) para esses eventos inscrevendo-se em uma instância.

@Output Decorator

- Usado para emitir eventos personalizados de um componente;
- Esses eventos podem ser inscritos e acionados por outros componentes;

```ts
@Output() tarefaAdicionada = new EventEmitter<string>();

addTarefa(todo: string) {
  todo = "J->" + todo;
  this.tarefaAdicionada.emit(todo);
}
```

lista-tarefas.component.ts

```html
<input #novaTarefa type="text">
<button (click)="addTarefa(novaTarefa.value)">Adiciona Tarefa</button>
```

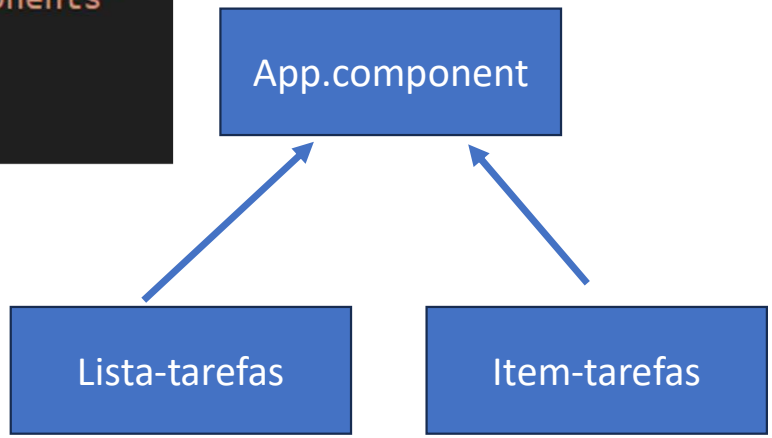
lista-tarefas.component.html

app.component.html

```html
<app-lista-tarefas [listaTarefas_listaTarefasComponent]="tarefas_appComponents"
(tarefaAdicionada)="onTarefaAdicionada($event)"
></app-lista-tarefas>
```

app.component.ts

```ts
onTarefaAdicionada(todo: string) {
  this.tarefas_appComponents.push(todo);
}
```



@ViewChild Decorator

- Utilizado para acessar um componente filho, uma diretiva ou um elemento DOM de uma classe de componente pai;
- Pode ser muito útil quando é preciso chamar métodos ou acessar propriedades de um componente filho.

```
@ViewChild('teste') titulo!:  ElementRef<any>;
```

app.component.ts

```
//mudar o do h1
this.titulo.nativeElement.innerText = "Numero de Tarefas (" + this.numeroDeTarefas + ")";
```

app.component.html

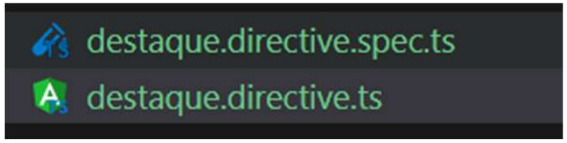
```
<h3 #teste></h3>
```

Criando Directives

- Existem muitas diretivas integradas disponíveis no Angular que foram vistas na Aula 06;
  - ngStyle
  - ngClass
  - *ngIf
  - *ngFor
  - Etc...
- Para criar uma diretiva personalizada é como criar um componente Angular;
- É preciso substituir @Component decorator por @Directive decorator;

Criando Directives

- Alterar a aparência ou comportamento de elementos DOM e componentes Angular com diretivas de atributo personalizada.
  - ng generate directive **destaque**




```typescript
import { Directive } from '@angular/core';

@Directive({
  selector: '[appDestaque]'
})
export class DestaqueDirective {
  constructor() { }
}
```

1.1_Directives\src\app\directives\destaque.directive.ts

Criando Directives

```typescript
import { Directive, ElementRef, OnInit } from '@angular/core';

@Directive({
  selector: '[appDestaque]'
})
export class DestaqueDirective implements OnInit{
  constructor(private elemento: ElementRef) {

  }

  ngOnInit(): void {
    this.elemento.nativeElement.style.backgroundColor = 'red';
    this.elemento.nativeElement.style.color = 'white';
  }
}
```

destaque.directive.ts

lista-tarefas.component.html

```html
<p appDestaque>Directiva destaque</p>
```

1.0_binding

Criando Directives com Renderer2

- Classe base para implementar a renderização personalizada;
- Por padrão, o Angular renderiza um modelo no DOM;
- Você pode usar a renderização personalizada para interceptar chamadas de renderização ou para renderizar em algo diferente do DOM.

Criando Directives com Renderer2

```typescript
import { Directive, ElementRef, OnInit, Renderer2 } from '@angular/core';

@Directive({
  selector: '[appDestaque]'
})
export class DestaqueDirective implements OnInit{
  constructor(private elemento:ElementRef, private renderizador: Renderer2) { }
  ngOnInit(): void {
    this.renderizador.setStyle(this.elemento.nativeElement, 'background-color', 'red');
    this.renderizador.setStyle(this.elemento.nativeElement, 'color', 'white');
  }
}
```

```html
<p appDestaque>Directiva personalizada utilizando a classe Renderer2</p>
```

https://angular.io/api/core/Renderer2

1.1_Directives\src\app\directives\destaque.directive.ts

@HostListener Decorator

```
export class AppComponent {
  title = 'Diretivas';

  @HostListener('window:click')
  onClick() {
    console.log('click');
  }

  @HostListener('window:resize')
  onResize() {
    console.log('resize');
  }
}
```

- Declara um evento DOM para escutar e fornece um método manipulador (handler) para ser executado quando esse evento ocorrer.

1.1_Directives\src\app\app.component.ts

Criando Directives com Renderer2 e Eventos

```
import { Directive, ElementRef, HostListener, OnInit, Renderer2 } from '@angular/core';

@Directive({
  selector: '[appDestComEvento]'
})
export class DestComEventoDirective implements OnInit {

  constructor(private elemento: ElementRef, private renderizador: Renderer2) {

   }

  ngOnInit(): void {
    throw new Error('Method not implemented.');
  }

  @HostListener('mouseenter') onMouseEnter(evento: Event) {
    this.renderizador.setStyle(this.elemento.nativeElement, 'background-color', 'red');
    this.renderizador.setStyle(this.elemento.nativeElement, 'color', 'white');
  }

  @HostListener('mouseleave') onMouseLeave(evento: Event) {
    this.renderizador.setStyle(this.elemento.nativeElement, 'background-color', 'transparent');
    this.renderizador.setStyle(this.elemento.nativeElement, 'color', 'black');
  }

}
```

Criando Directives com Renderer2 e Eventos

```
@HostListener('mouseenter') onMouseEnter(evento: Event) {
  this.renderizador.setStyle(this.elemento.nativeElement, 'background-color', 'red');
  this.renderizador.setStyle(this.elemento.nativeElement, 'color', 'white');
}

@HostListener('mouseleave') onMouseLeave(evento: Event) {
  this.renderizador.setStyle(this.elemento.nativeElement, 'background-color', 'transparent');
  this.renderizador.setStyle(this.elemento.nativeElement, 'color', 'black');
}
```

```
<p appDestComEvento>Directiva personalizada utilizando a classe Renderer2
    e tratando eventos sob o elemento</p>
```

Directiva personalizada utilizando a classe Renderer2 e tratando eventos sob o elemento

@HostBinding Decorator

- Possibilita a manipulação de propriedades, atributos e classes de um elemento DOM;
  - Propriedades de estilo, como altura, largura, cor, margem, borda, etc;
  - Quaisquer outras propriedades internas do elemento host na classe diretiva.

```
@HostBinding('style.backgroundColor') backgroundColor: string = '';
@HostBinding('style.color') color: string = '';
```

@HostBinding Decorator

```ts
import { Directive, HostBinding, HostListener } from '@angular/core';

@Directive({
  selector: '[appBackBlack]'
})
export class BackBlackDirective {

  @HostBinding('style.backgroundColor') backgroundColor: string = '';
  @HostBinding('style.color') color: string = '';

  constructor() { }

  @HostListener('mouseenter') onMouseEnter() {
    this.backgroundColor = 'black';
    this.color = 'white';
  }

  @HostListener('mouseleave') onMouseLeave() {
    this.backgroundColor = 'transparent';
    this.color = 'black';
  }

}
```

1.1_Directives\src\app\directives\back-black.directive.ts

@HostBinding Decorator

```
//passando parametros para a diretiva
@Input() corPadrao: string = 'transparent';
@Input() corDestaque: string = 'black';

constructor() { }

@HostListener('mouseenter') onMouseEnter() {
  this.backgroundColor = this.corDestaque;
  this.color = 'white';
}

@HostListener('mouseleave') onMouseLeave() {
  this.backgroundColor = this.corPadrao;
  this.color = 'black';
}
```

```
<h1 appBackBlack corDestaque="brown"
corPadrao="blue">
Outra forma de escrever</h1>
```

```
<h1 appBackBlack [corDestaque]="'pink'" [corPadrao]="'blue'">
    Passagem de parâmetros</h1>
```

Diretivas Estruturais

- Diretivas estruturais são diretivas que alteram o layout do DOM adicionando e removendo elementos DOM;

Diretiva personalizada `appEstrutSome`

- Diretiva que se parece com a diretiva estrutural do Angular *ngIF;
- Controla se o elemento será renderizado pelo Angular;

```
<h1 *appEstrutSome="true">teste</h1>
```

Diretiva personalizada appEstrutSome

```ts
import { Directive, Input, TemplateRef, ViewContainerRef } from '@angular/core';

@Directive({
  selector: '[appEstrutSome]'
})
export class EstrutSomeDirective {

  constructor(
    private templateRef: TemplateRef<any>,
    private viewContainer: ViewContainerRef
  ) {}

  @Input() set appEstrutSome(condicao: boolean) {
    if (condicao) {
      this.viewContainer.createEmbeddedView(this.templateRef);
    } else {
      this.viewContainer.clear();
    }
  }

}
```

```html
<h1 *appEstrutSome="true">teste</h1>
```

```html
<p appDestaque  *appEstrutSome="false">Diretiva estrutural</p>
```

1.1_Directives\src\app\directives\estrut-some.directive.ts

<ng-template>

- Definir o conteúdo do modelo que só será renderizado pelo Angular quando for instruí-lo especificamente para fazê-lo;
- Permite o controle total sobre como e quando o conteúdo é exibido;
- Agrupar o conteúdo dentro de um <ng-template> sem instruir o Angular para renderizá-lo, faz com que esse conteúdo não apareça em uma página.

<ng-template>

```
<ng-template>
  <mat-form-field class="example-full-width">
    <mat-label>Deixe um comentário</mat-label>
    <textarea matInput placeholder="Ex. Isso me faz sentir...."></textarea>
  </mat-form-field>
</ng-template>
```

- Esse campo não aparecerá na renderização da página a não ser que seja instruído para fazer;

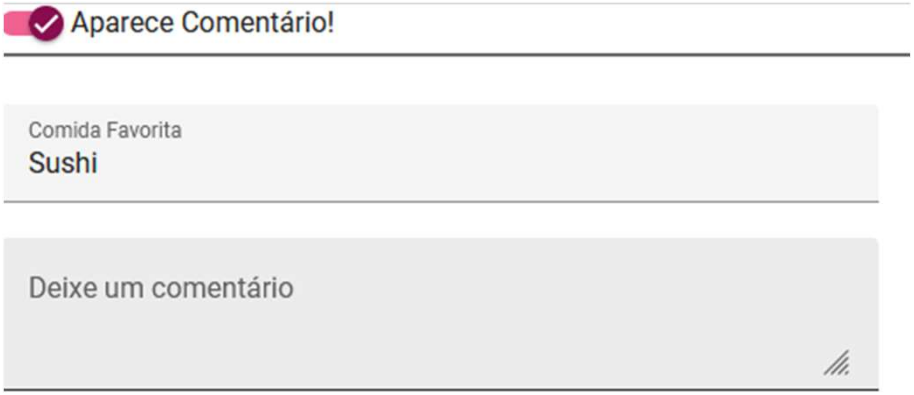
ngModelo\src\app\app.component.html

<ng-template>

```html
<ng-template [ngIf]="apareceCampoComentario">
  <mat-form-field class="example-full-width">
    <mat-label>Deixe um comentário</mat-label>
    <textarea matInput placeholder="Ex. Isso me faz sentir...."></textarea>
  </mat-form-field>
</ng-template>
```

```typescript
export class AppComponent {
  title = 'ngModelo';
  apareceCampoComentario: boolean = true;

  toggleComentario() {
    this.apareceCampoComentario = !this.apareceCampoComentario;
  }
```



ngModelo\src\app\app.component.html

<ng-template>

```
  <mat-form-field class="example-full-width" *ngIf="apareceCampoComentario">
    <mat-label>Deixe um comentário</mat-label>
    <textarea matInput placeholder="Ex. Isso me faz sentir...."></textarea>
  </mat-form-field>


<!-- <ng-template [ngIf]="apareceCampoComentario">
  <mat-form-field class="example-full-width">
    <mat-label>Deixe um comentário</mat-label>
    <textarea matInput placeholder="Ex. Isso me faz sentir...."></textarea>
  </mat-form-field>
</ng-template> -->
```

ngModelo\src\app\app.component.html

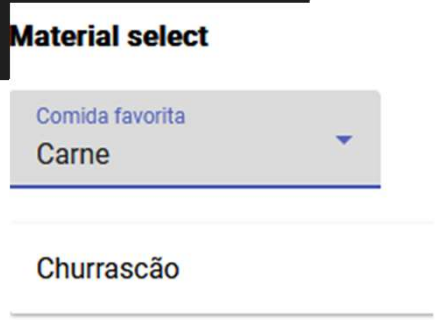
Diretiva *ngSwitch

- A diretiva [ngSwitch] em um container especifica uma expressão para correspondência;
- As expressões correspondentes são fornecidas pelas diretivas ngSwitchCase nas visualizações dentro do container;
- Cada visualização correspondente é renderizada;
- Se não houver correspondências, uma visualização com a diretiva ngSwitchDefault será renderizada;
- Os elementos dentro da instrução [NgSwitch], mas fora de qualquer diretiva NgSwitchCase ou ngSwitchDefault, são preservados no local.

Diretiva *ngSwitch

```
<h4>Material select</h4>
<mat-form-field>
  <mat-label>Comida favorita</mat-label>
  <mat-select [(value)]="selecionado">
      <mat-option *ngFor="let food of foods" [value]="food.value"> {{food.viewValue}}</mat-option>
  </mat-select>
</mat-form-field>

<mat-card [ngSwitch]="selecionado">
  <mat-card-content *ngSwitchCase="'steak-0'">Churrascão</mat-card-content>
  <mat-card-content *ngSwitchCase="'pizza-1'">Pizza da Mama Adubada!!!</mat-card-content>
  <mat-card-content *ngSwitchCase="'tacos-2'">Muita Pimenta!</mat-card-content>
</mat-card>
```



ngModelo\src\app\app.component.html

Exercicio 01

- Crie uma diretiva estrutural personalizada que seja uma variação do *ngFor. A ideia é que ao usar a sua diretiva, você passe o número de vezes que um elemento deve ser repetido e a sua diretiva irá fazer o trabalho.

  - Ex: <div *repitaXvezes="“3”>
    - teste
    - teste
    - teste
  -

1.1_Directives\src\app\directives\repete.directive.ts

Exercício 02

- Crie uma diretiva personalizada que aplique uma animação CSS a um elemento ele quando for clicado;
- Utilize @HostListener decorator para “ouvir” os eventos de clique e a classe Renderer2 para manipular o elemento;

Referências

- https://angular.io/guide/attribute-directives
- https://angular.io/guide/structural-directives
- https://angular.io/api/common/NgSwitch
- https://angular.io/api/core/Renderer2
- https://angular.io/api/core/HostListener
- https://medium.com/@matsal.dev/angular-how-to-use-hostlistener-9ea8500128a6