



Angular

Evelise Dib

Frontend na Loft | Mentora na Laboratória | Instrutora na Caelum

Jogadora de Don't Starve assídua,
eu vim de Santos, cultuo bons
livros e amo os animais 🎵
(tá ligado eu sou o bicho).





Dia 1



Photo: @markusspiske - Unsplash

AngularJs

- Utilizava Javascript
- Foi criado em 2009
- Conceito de S.P.A.
- Mantido pela Google

Angular 2

- Mudança de linguagem
- Semantic Version
- MVC
- Criado em 2016

O que eu preciso para começar a usar?

- Um pouco de café
- [Node](#) instalado
- [NPM](#) instalado
- [Angular CLI](#) instalado



Photo: [@nate_dumlao](#) - Unsplash

C
F
M
H
R
B
S
/

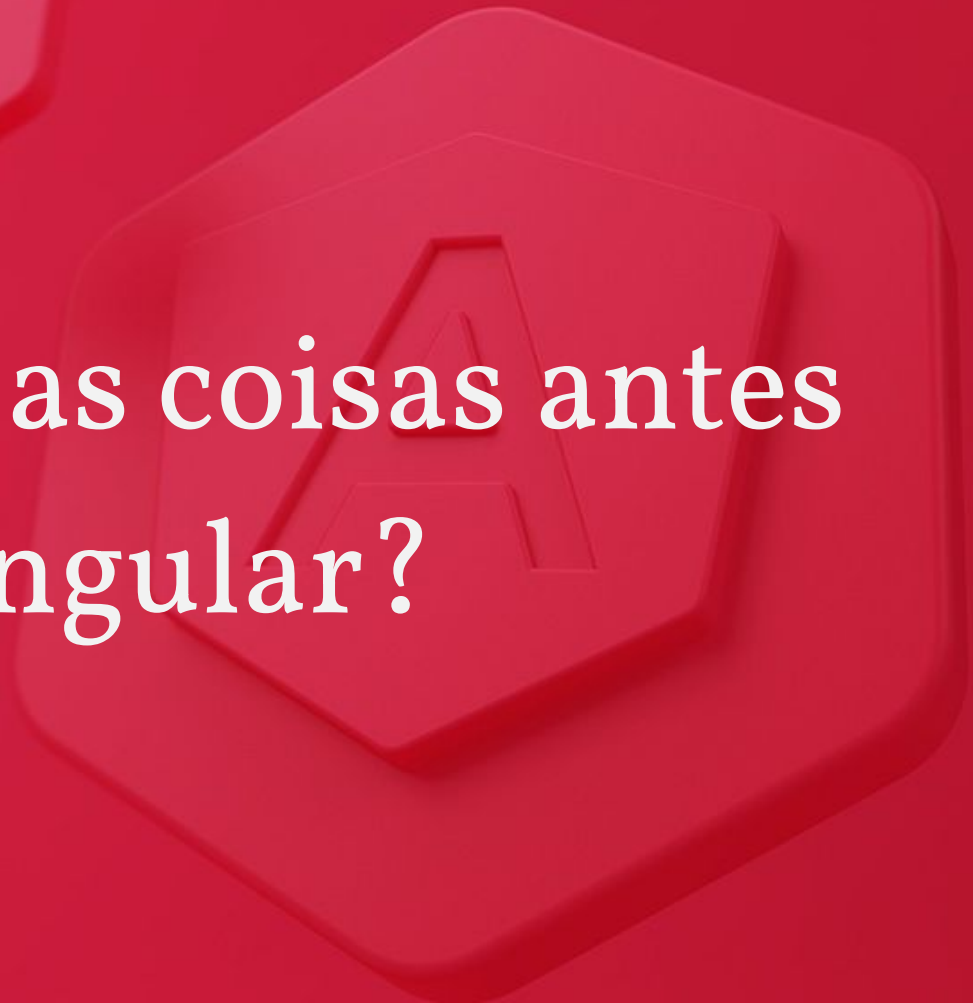


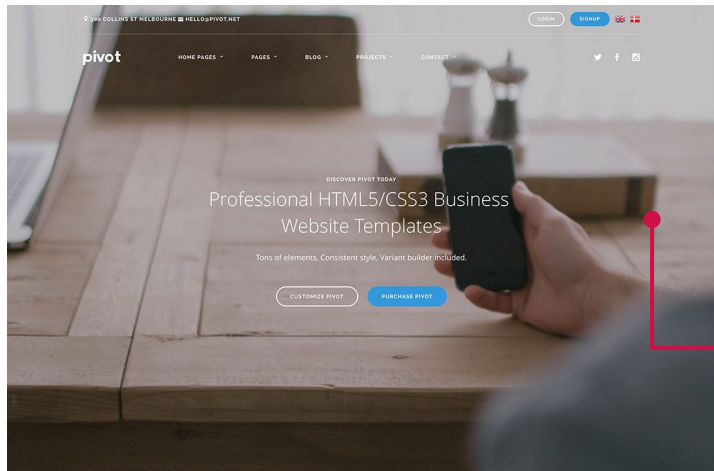
Photo: @markusspiske - Unsplash

CLI

- Command Line Interface
- Mandar comandos escritos para que o computador faça o que você está pedindo
- Abra seu terminal de preferência
- Digite **cal** e dê enter

Como eram as coisas antes
do Angular?



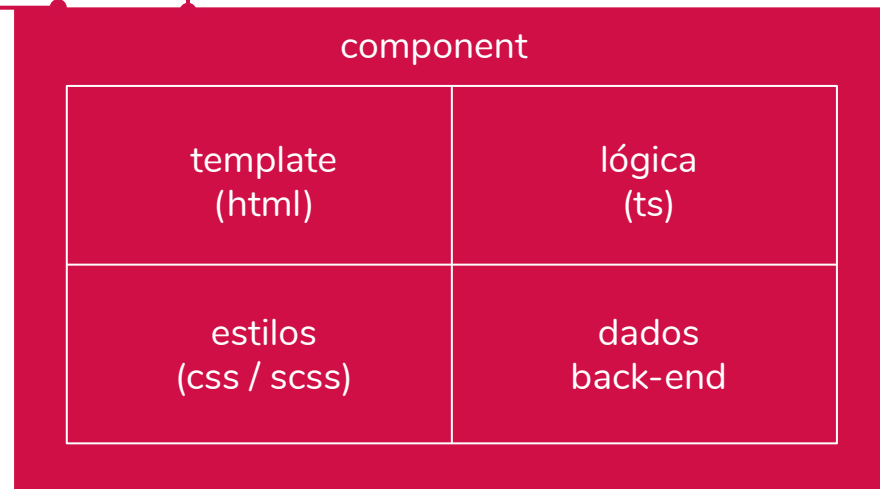
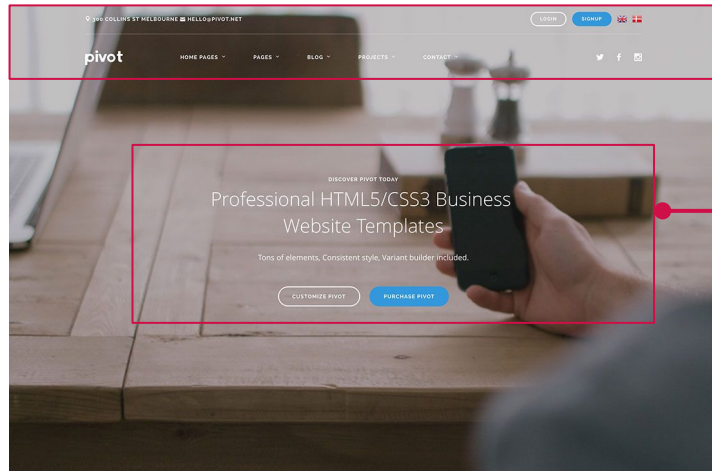


template (html)	lógica (js)
estilos (css / scss)	datos back-end

A
F
R
M
C
S
H
B
S
R
C

E como fica com Angular?





A
F
R
M
C
S
R
H
B
C

app.component.ts



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

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



```
<p> Algum teste aqui </p>  
<app-root></app-root>
```



Dia 2

Photo: @markusspiske - Unsplash

- ```
> npm install -g @angular/cli
> ng new my-dream-app
> cd my-dream-app
> ng serve
```

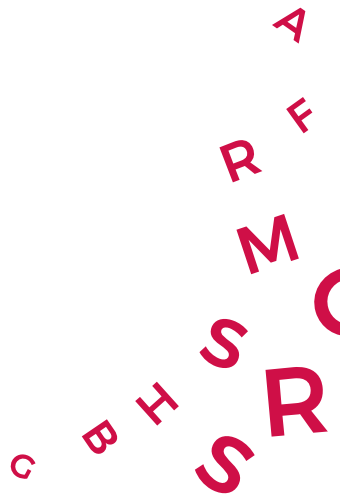
```
> npm install -g @angular/cli
> ng new my-dream-app
> cd my-dream-app
> ng serve
```

# Classes

- No final das contas, o Typescript vira Javascript, que é compilado pelo Babel e já vem configurado dentro da aplicação.
- Para que cada component exista, é preciso estar atrelado à uma classe
- A classe tem ser acessível por outras partes internas do código
- Qual a diferença de uma classe comum de um Component?

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

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





# Decorators

---

São modificadores de classes,  
definindo algum tipo ou  
categoria específica.

É usado com o @



# Estilos

## Por component

---

- É acoplado apenas ao component de referência, não afeta globalmente.

## Globalmente

---

- Afeta o escopo de todo projeto

```
<head>
<meta charset="utf-8">
<title>Página</title>
<!-- Material Design -->
<link rel="stylesheet"
href="https://code.getmdl.io/1.3.0/material.indigo-red.min.css" />
<!-- ./Material Design -->
</head>
<body>
 <app-root></app-root>
</body>
```

A  
F  
R  
M  
C  
S  
R  
H  
B  
C



**Almoço**

# Interpolação de template

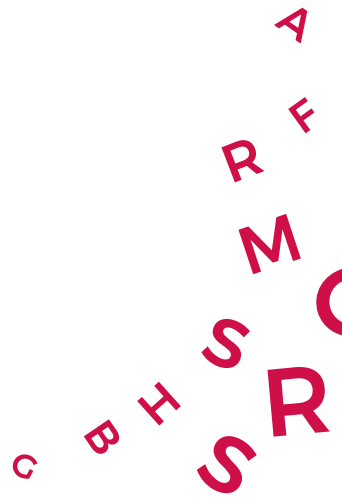


```
<label>{{textoDaLabel}}</label>
```



```
<div> {{10+13}} </div>
```

- Usado para imprimir no html algum valor fixo ou dinâmico, podendo ser de uma variável, função ou até mesmo um inteiro

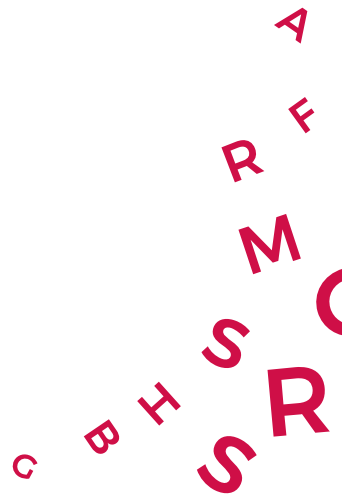


# Property binding

```
<button disabled>Clique-me</button>
<button [disabled]="false">Clique-me</button>
<button [disabled]="variavel">Clique-me</button>
<!-- ---- -->

```

- Através dos [ ] ao redor de alguma atributo é feita a associação (podendo ser um inteiro, booleano, métodos da classe...)



# One Way data-binding



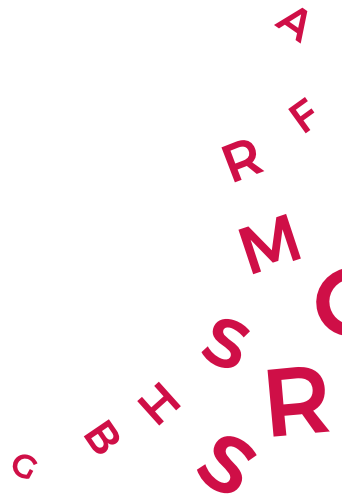
```
<button disabled>Clique-me</button>
<button [disabled]="false">Clique-me</button>
<button [disabled]="variavel">Clique-me</button>
<!-- ---- -->

```



```
<label>{{textoDaLabel}}</label>
```

- Quando passamos um valor de uma camada para outra.
- Ex.: Da classe até a view

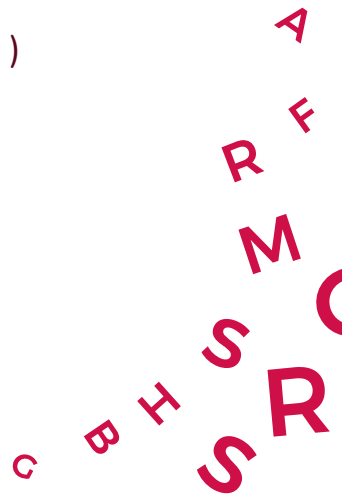


# Event Binding



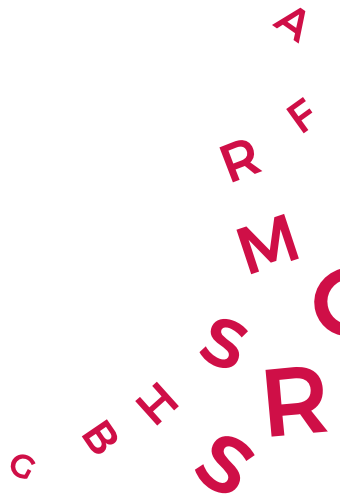
```
<button (click)="onCall()">Clique-me</button>
<input (input)="executar()" />
```

- Associação por eventos.
- Usadas para escutar os eventos padrões do browser e criar alguma lógica através dela.
- A sintaxe é usada com os ( ) em volta do evento



# O component root

- Se tudo faz parte de um component, então por onde começar?
- Quando criamos um projeto, o angular seta as configurações de um component raiz para saber de onde partir.
- Dica 1: É possível descobrir a partir do módulo!
- Dica 2: Procuro pela tag no arquivo index.html



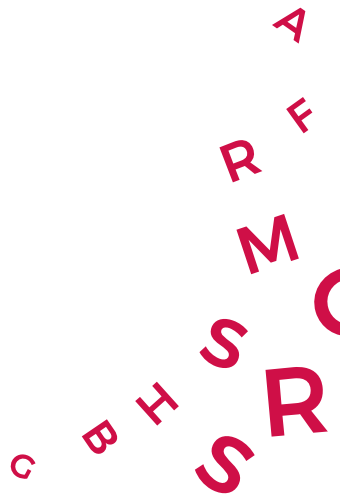
# Diretivas

- Você pode criar as suas próprias diretivas customizadas
- O ciclo de vida pode ser usado aqui

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

@Directive({
 selector: '[customField]'
})

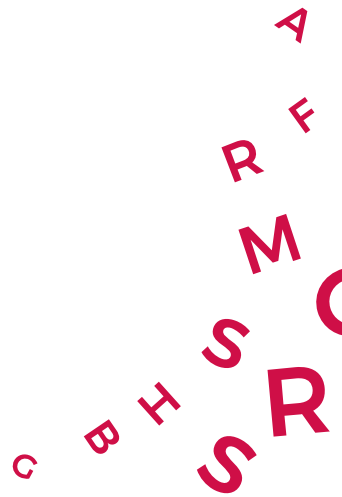
export class CustomFieldDirective {}
```





# Arquitetura LIFT

- Locate: Localizar intuitivamente: manter os arquivos relacionados próximos quando
- Identify: Identificar: nomear os arquivos de forma que representa o que ele faz
- Flat: Plano: estrutura de pastas simples -> nunca ultrapassar o limite de 9 camadas
- T-Dry: Não se repetir: Não repita sem necessidade (nomes de view em arquivos com extensão html, por exemplo)



# NgClass

- Diretiva específica para manipular classes
- Usada para manipular classes dinâmicas através de algum atributo da classe (quando queremos esconder algum item através de um click por exemplo)



```
<some-element [ngClass]='first second'>...</some-element>
```

```
<some-element [ngClass]='{\'first\': true, \'second\': true, \'third\': false}'>...</some-element>
```

```
<some-element [ngClass]='{\'class1 class2 class3\' : true}'>...</some-element>
```

A  
F  
M  
C  
S  
R  
S  
H  
B  
C

# Sintaxe



## Event (Output)

Usado para lidar com os eventos keyDown, keyUp, click



## Template

Usados para alteração do DOM (ngIf, ngFor)



## Directive (Input)

Usado para adicionar atributos à tag

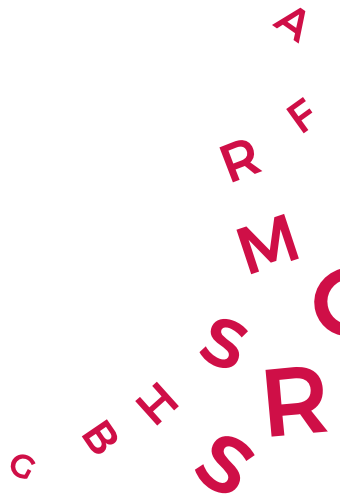




**Dia 3**

# Módulos

- Usados para separar melhor os contextos
- Também é usado para o lazy-load
- Imports: Módulos e dependências que você precisa usar
- Exports: Exportação de components ou diretivas
- Declarations: Declaração de components e diretivas
- Bootstrap: Component que será inicializado
- Providers: Serviços (são carregados antes da view)



# Ng-content

- Usado para inserir templates dinâmicos
- Funciona como um placeholder
- Insere no lugar marcado o que estiver dentro das custom tags do component

```
<app-child>
 <div header>This should be rendered in header selection of ng-
content</div>
 <div body>This should be rendered in body selection of ng-content</div>
</app-child>
```


```
<div class="header-css-class">
 <ng-content select="[header]"></ng-content>
</div>
<div class="body-css-class">
 <ng-content select="[body]"></ng-content>
</div>
```



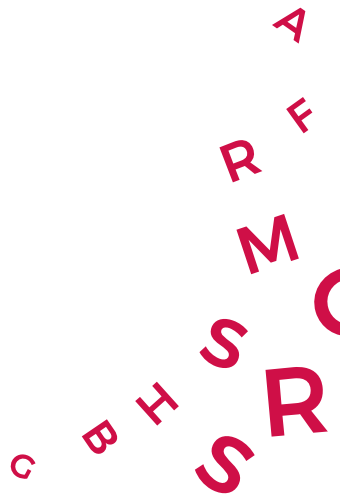
**Dia 4**

# \$event

- Usado pelo event binding
- Retorno de um 'handler' de eventos, como se fosse uma resposta do evento



```
<button (click)="cadastrar($event)">
 click-me
</button>
```





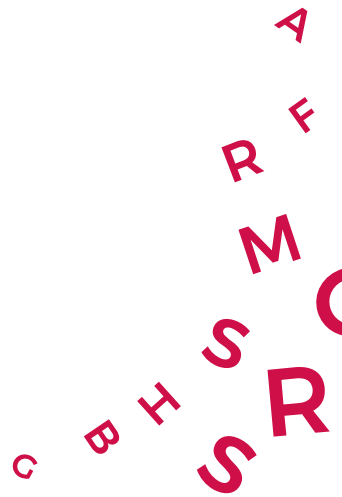
# NgFor

- Diretiva usada para iterar listas no template
- Funciona como uma estrutura de repetição padrão (for)
- \* é usado quando modificamos o DOM (adicionar ou remover algo)

```

 <li *ngFor="let usuario of listaUsuarios">
 {{usuario.nome}} - {{usuario.email}}


```



# Refêrencias

- # são usadas para criar referências rápidas de template
- Usada para consultar detalhes dos elementos
- Pode ser usada para atribuir valores



```
<input #nivel value="12" />
{{nivel.value}}
```



# NgModel

- NgModel é o que chamamos de two-way-data-binding
- Também é chamado de banana in a box [()]
- Utilizado para formulários (ao mesmo tempo que pega os eventos, salva o valor na variável)
- Precisa conter um “name” no campo
- Precisa importar o FormsModule no módulo do component



```
<input #nivel="ngModel" name="nivel" [(ngModel)]="form.nivel" />
```





**Dia 5**

# NgSubmit

- Exclusivo para a tag <form>
- Evita o reload da página (form com action faz reload, por exemplo)
- O botão de ação fica no form e não no button



```
<form #form="ngForm" (ngSubmit)="enviar($event)">
 ...
</form>
```

A  
E  
M  
C  
S  
R  
S  
H  
B  
C

# \*ngIf

- Usado para criar regras que devem ser omitidas do template de acordo com alguma variável ou condição



```
<p *ngIf="resultadoPesquisa === null">Nada foi encontrado.</p>
```

