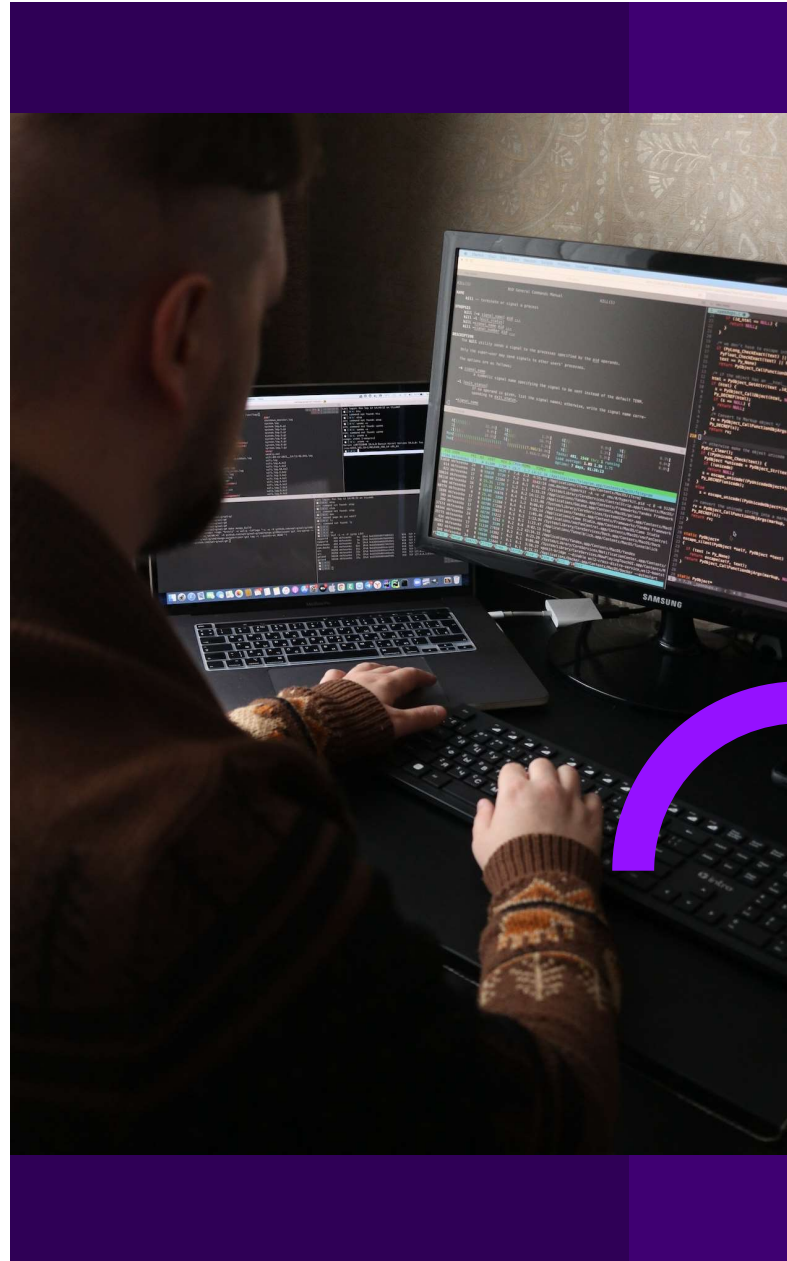
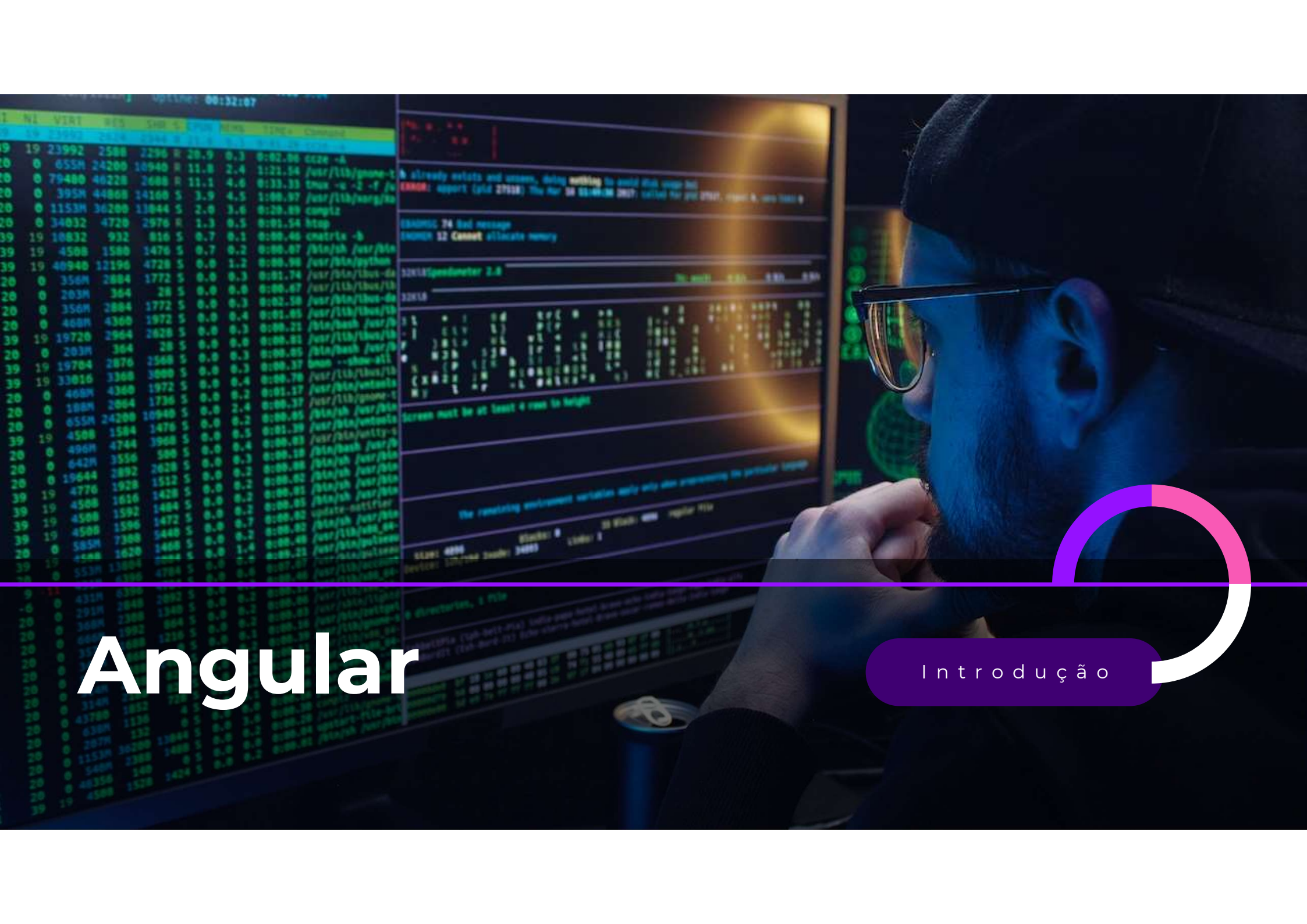


Apresentação Pessoal

Leone Costa Rocha

1. **Idade:** 35 anos
2. **Experiência:** 10 anos na área de desenvolvimento de sistemas
3. **Certificações:**
 - AZ-900: Fundamentos do Microsoft Azure
 - 70-515: Web Applications Development with Microsoft .NET Framework .
 - 498-361: Software Development Fundamentals
4. **Basic technical skills :** Back-End (C#, ASP.NET), Front-End (JavaScript, CSS, Type Script)
5. **GitHub:** <https://github.com/LeoneRocha>
6. **LinkedIn:** <https://www.linkedin.com/in/leone-costa-rocha-14049722/>
7. **E-mails :**
 - leone.rocha@cognizant.com
 - leocr_lem@yahoo.com.br





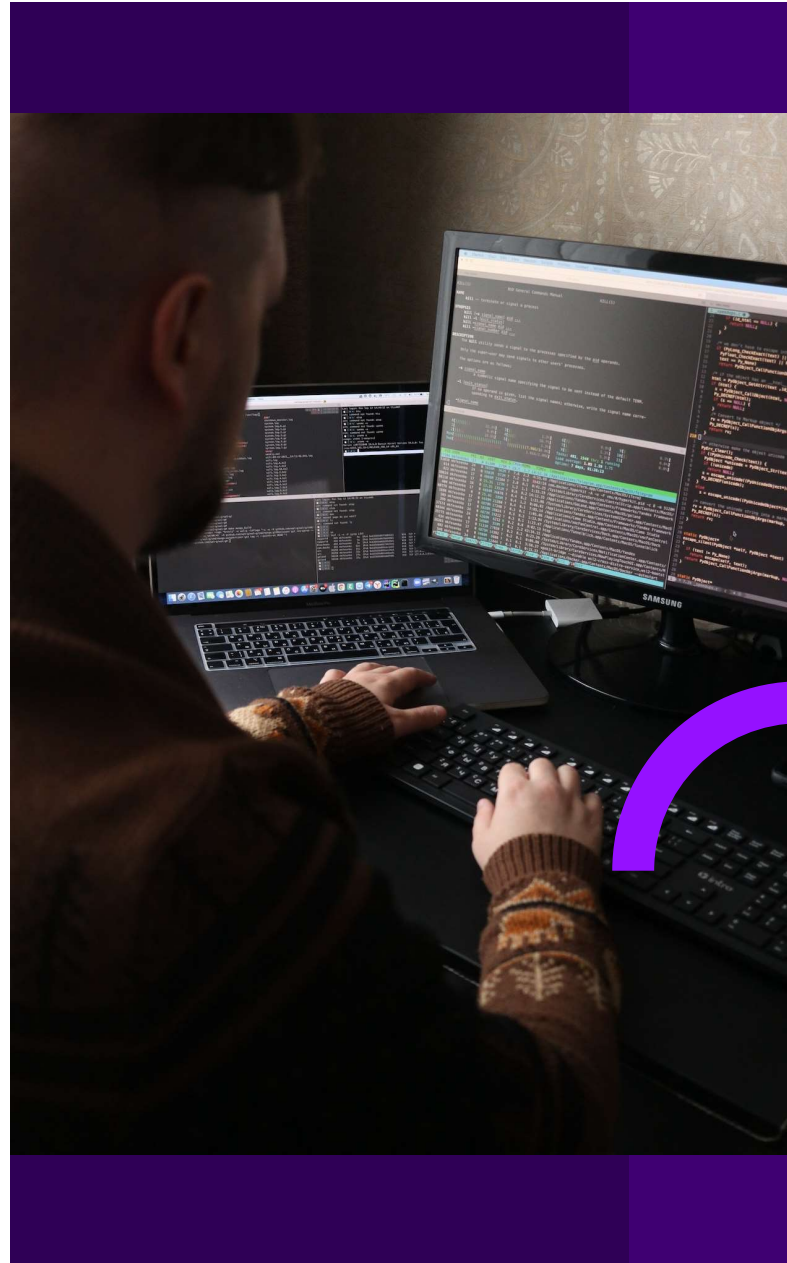
Angular

Introdução

Angular

Conteúdo

1. Introdução
2. Componentes
3. Templates
4. Tipos de bind
5. Diretivas
6. Serviços
7. Módulos



Introdução



Angular é um framework de desenvolvimento web em JavaScript criado pela Google.

TypeScript é tipo uma versão melhorada do JavaScript, com superpoderes de detecção de erros e organização. É tipo um JavaScript turbinado!. Com recursos de **tipagem** estática e a **orientação a objetos**.

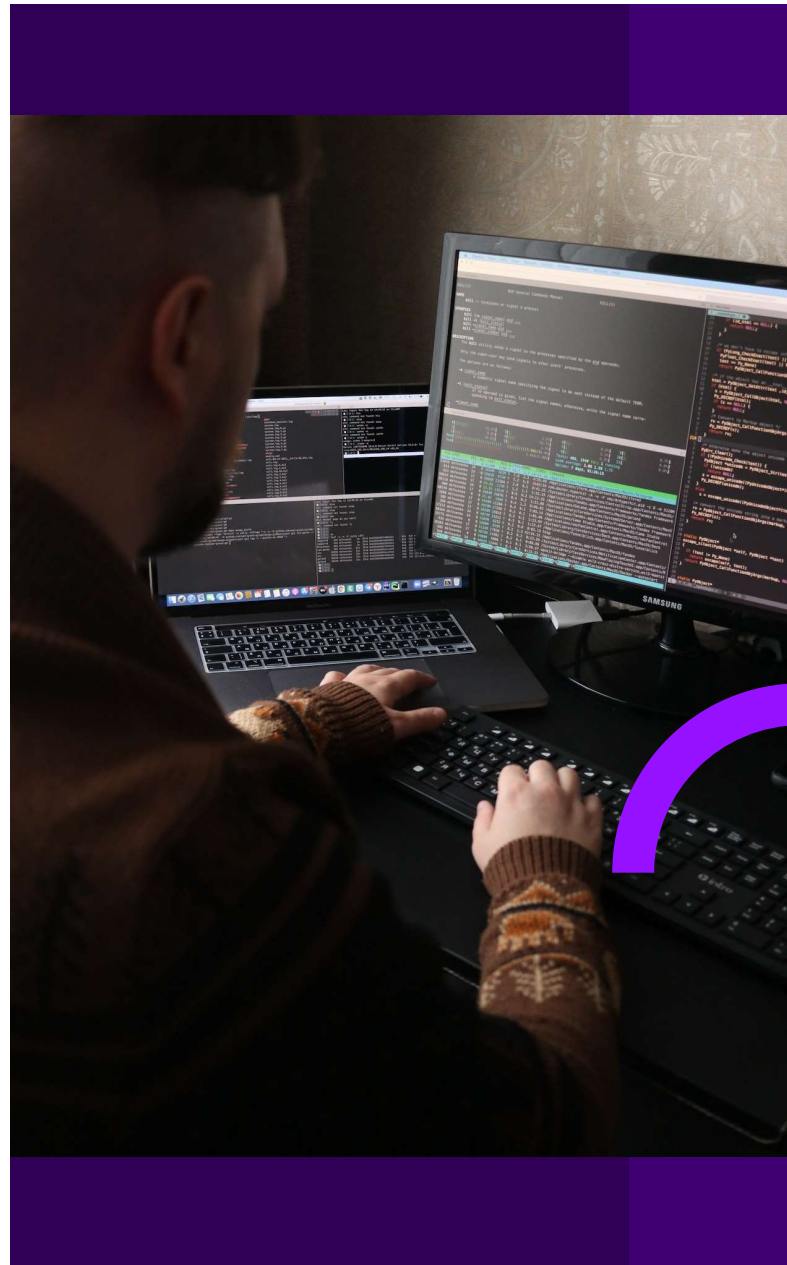
Diferenças

Angular JS

1. Surgiu para ser simples
2. Performace ruim
3. Api Cresceu incosistentemente
4. Conceitos confuses e repetidos
5. ES5 (ECMAScript 2009) * primeira grande revisão do JavaScript.

Angular (2+)

1. Mais aderente a padrões
2. Padrão para criar qualquer coisa (pipes, componentes, services
3. Olhando a era de componentes
4. ES6/ES2015 * a mais nova versão do JavaScript
5. Em typescript o codigo é transpilado para JavaScript



Como identificar AngularJS (versão 1.x) ou Angular (versão a partir do 2+)

1. **Estrutura de diretórios:** O **AngularJS** geralmente tem uma estrutura de diretórios diferente do Angular2+. No AngularJS, os arquivos **JavaScript** e **HTML** são agrupados em uma estrutura baseada em **funcionalidade**, enquanto no **Angular**, os arquivos **TypeScript** e **HTML** são organizados em **componentes** e **módulos**.
2. **Sintaxe de código:** No **AngularJS** usa **ES5** (ECMAScript 5) usa o prefixo "**ng-**" ex: **ng-if**, uso de controller para organizar os elementos html, Uso de **\$scope** para compartilhar dados entre o controlador e o template. Já no **Angular 2+ ES6** (ECMAScript 6) adotando TypeScript, Uso de **diretivas *ngIf, organização em Componentes** que substituem os semelhantes controladores do AngularJS. E uso do conceito de **injeção de dependências**.
3. **Módulos:** No Angular, o conceito de módulos é fundamental. Se você encontrar uma definição de módulo no código, é provável que esteja trabalhando com **Angular 2+**. No AngularJS, não existe um conceito de módulos tão distintos.



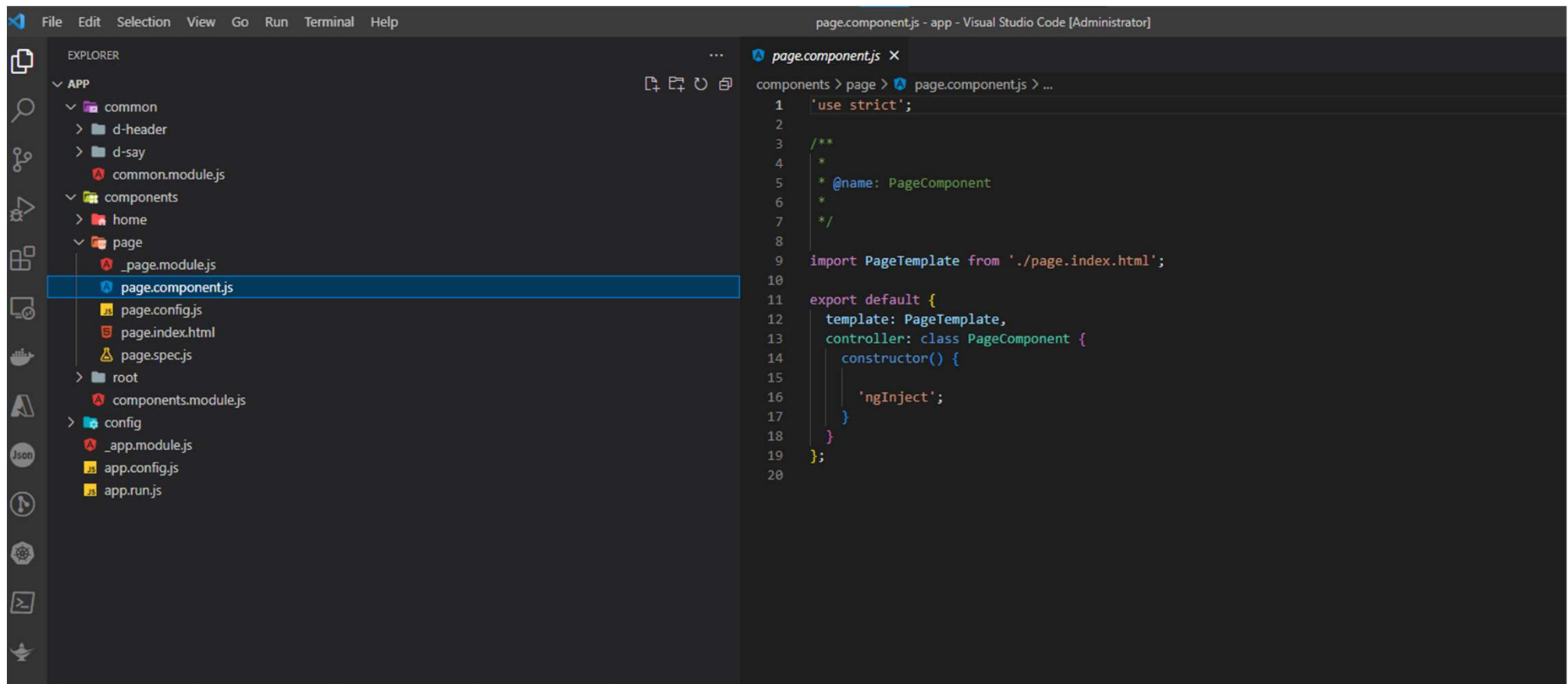
Como identificar AngularJS (versão 1.x) ou Angular (versão a partir do 2+)

- 4. **Angular CLI:** Uma ferramenta de linha de comando que facilita a criação e o gerenciamento de projetos utilizado a partir do **Angular 2+**.
- 5. **Versões do pacote:** O AngularJS tem versões como 1.x (por exemplo, 1.7.9), enquanto o **Angular 2+** possui versões a partir do 2.x (por exemplo, 12.2.5).
- 6. **Build :** Ao realiza o build de um projeto **Angular 2 +** (TypeScript) é utilizado Angular CLI para transformar seu código **TypeScript** em um código Javascript pronto para ser executado em um navegador.



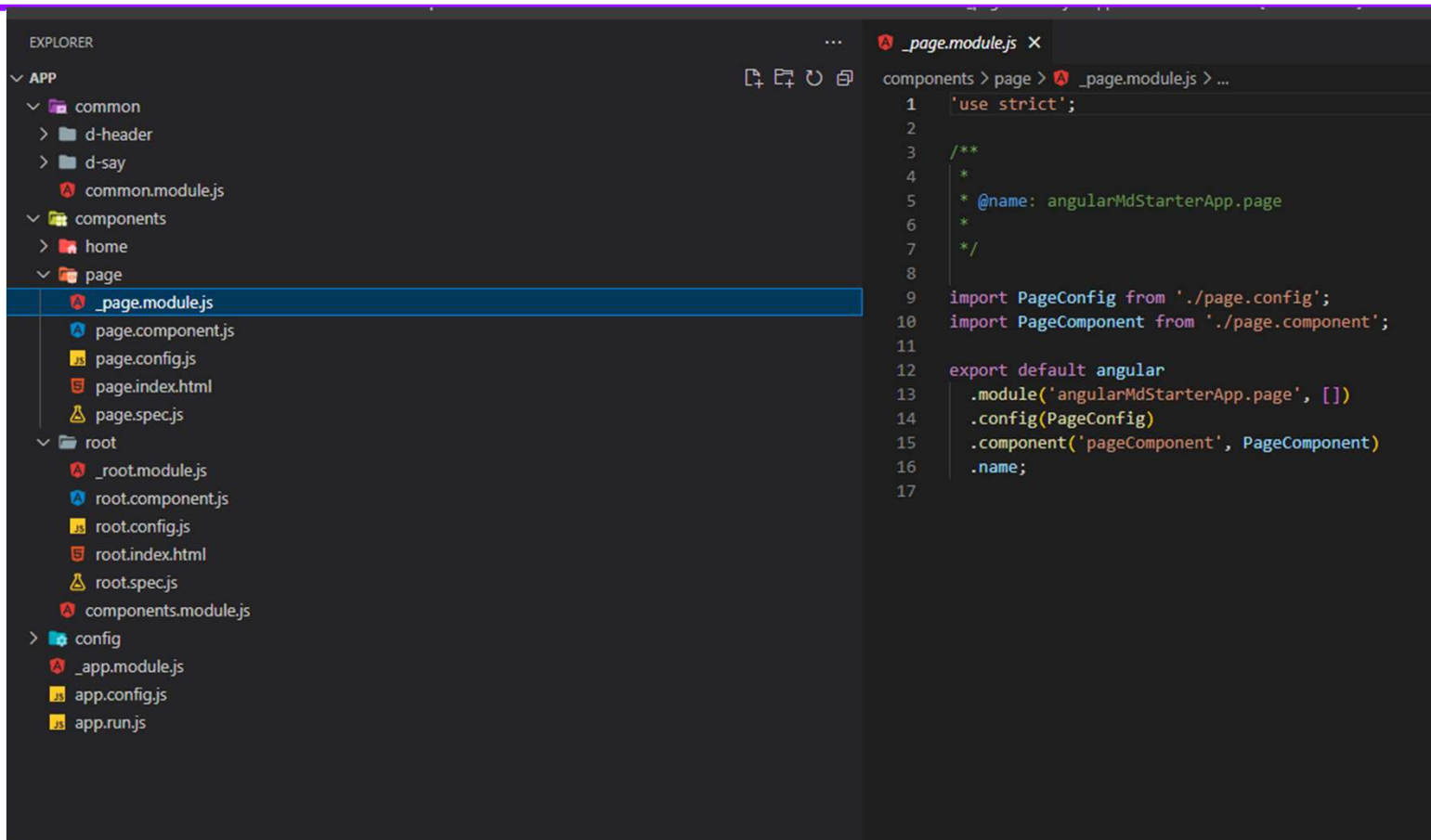


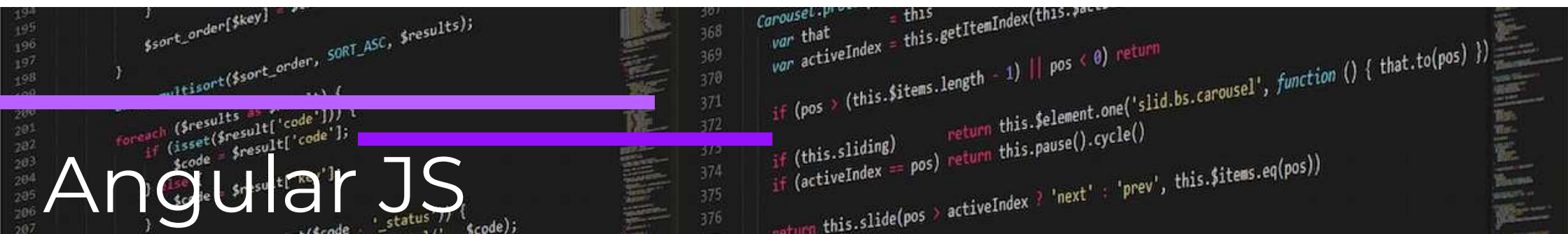
Angular JS





Angular JS





```
8
9 function RootConfig($stateProvider, $urlRouterProvider) {
10
11     'ngInject';
12
13     $stateProvider
14     .state('root', {
15         abstract: true,
16         url: '/',
17         template: '<root-component></root-component>',
18     });
19
20     $urlRouterProvider.otherwise('/');
21 }
22
23 export default RootConfig;
24
```

```
components.module.js X
components > components.module.js > ...
1 'use strict';
2
3 /**
4  *
5  * @name: angularMdStarterApp.components
6  *
7  */
8
9 import RootModule from './root/_root.module';
10 import HomeModule from './home/_home.module';
11 import PageModule from './page/_page.module';
12
13 export default angular.module('angularMdStarterApp.components', [
14     RootModule,
15     HomeModule,
16     PageModule,
17 ])
18 .name;
19
```

TypeScript

Tipos de dados

- Variáveis com tipagem de dados
- Erros de compilação
- Dados dinâmicos

```
const message: string = "Criando nossa primeira string tipada"
console.log(message)

let idade: number = 4
// idade = '1' // gera erro ao compilar

console.log("idade " + idade)
idade = idade + 1
console.log("idade " + idade)

// Não é obrigatório tipar um dado
// Funcionando como JavaScript puro
let dadoDinamico
dadoDinamico = "meu nome"
dadoDinamico = 1
```



Angular 2+ (TypeScript)

Entendendo a estrutura do app

O projeto gerado segue exatamente uma estrutura de aplicação NodeJS. Existe um **package.json** que contém todas as dependências que ele instalou.

Vamos analisar as pastas e principais arquivos que foram criadas com o comando que executamos



Entendendo a estrutura do app

- **.angular-cli.json:** arquivo que define como o Angular executará com todas as suas configurações
- **webpack:** module bundler para compilar todos os assets da nossa aplicação. Como você pode ter visto ao executar o comando `ng serve` foi exibido todos os bundles que ele gerou para rodar a aplicação
- **styles.css:** CSS globais da aplicação
- **assets:** não sofrerá nenhum impacto durante a compilação



Angular 2+ (TypeScript)

Entendendo a estrutura do app

- **polyfills.ts:** para habilitar as features para browser mais antigos que precise dar suporte
- **main.ts:** módulo de definição principal
- **index.html:** html que será renderizado



Angular 2+ (TypeScript)

Instalação e configuração do ambiente de desenvolvimento

Para começar a desenvolver com Angular e TypeScript, é necessário instalar o Node.js e o Angular CLI. Você pode fazer isso digitando

```
"npm install -g @angular/cli"
```

Na linha de comando. Em seguida, siga as instruções da documentação oficial do Angular para instalar e configurar o ambiente de desenvolvimento em seu sistema operacional.



Angular 2+ (TypeScript)

Criação de um projeto em Angular com TypeScript

Um projeto em Angular possui uma estrutura específica, com arquivos e pastas como "**src**" (código fonte), "**node_modules**" (**dependências**) e "**angular.json**" (**configurações**). Para criar um novo projeto em Angular com TypeScript, basta digitar na linha de comando.

```
ng new meu-projeto
```

```
npm run build
```

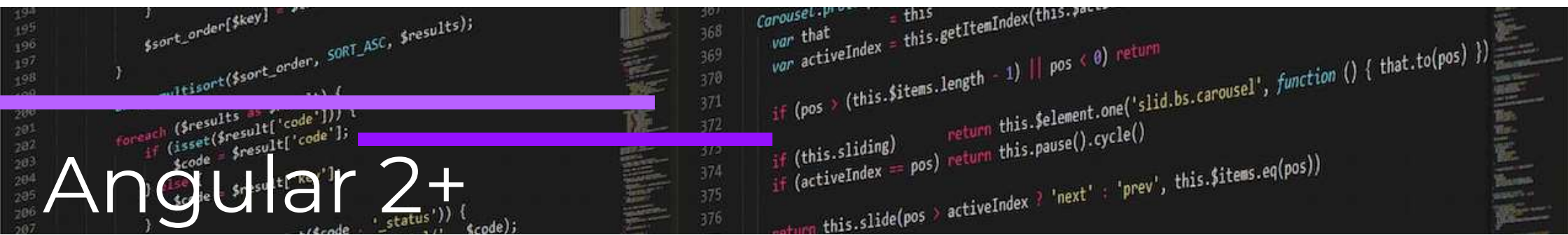
```
npm start
```


Componentes



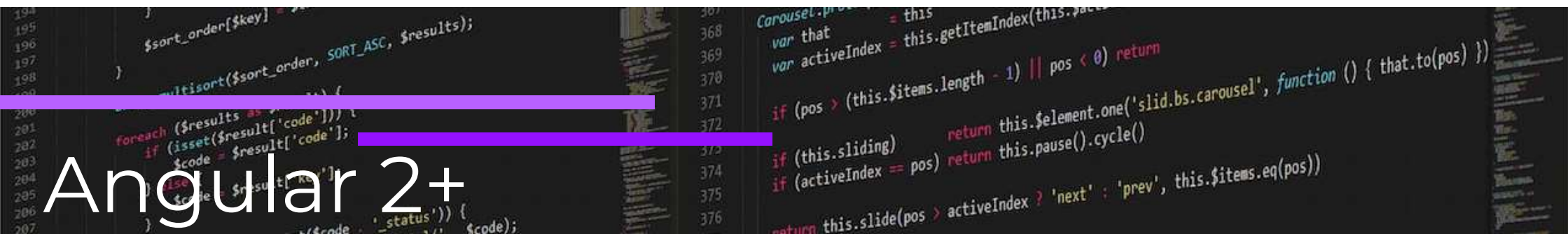
Componentes são elementos essenciais do Angular usados para criar **partes** da **interface** do usuário, como **botões**, **caixas** de texto ou **tabelas** de dados. Essas estruturas são blocos reutilizáveis que combinam **código**, **estilo** e **template** para criar partes da **interface do usuário** de forma modular e organizada.

Encapsulam funcionalidades específicas e podem ser combinados para construir aplicações complexas.



O componente

- **html:** template HTML que será renderizado
- **css:** estilos CSS para este componente
- **ts:** componente em si que foi criado



Exemplo de um componente

```
import { Component } from "@angular/core"

@Component ({ // -> decorator
  selector: 'app-first',
  templateUrl: './my-first-component.html'
  // ou
  // template: '<h1> Test </h1>'
})

export class MyFirstComponent {
  constructor (Image
}
```

Angular 2+

Utilizando um componente

Quando o componente estiver pronto é necessário informar ao módulo da sua aplicação que esse componente existe.

Assim você informa ao Angular a qual módulo um determinado componente pertence.

```
// app.module.ts

@NgModule({
  declarations: [MyFirstComponent]
})

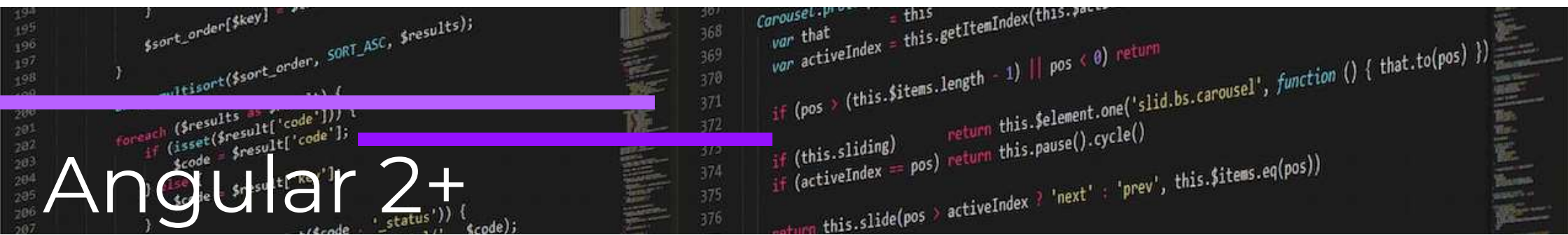
export class AppModule {}
```

Image

Templates



O template no Angular é onde você cria a **aparência visual** de um componente, usando uma mistura de **HTML**. Ele pode ser definido diretamente no arquivo do componente ou em um arquivo HTML externo. É a parte de **interface visual de componentes**.



HTML do componente

```
<!-- header.component.html -->
<header>
  <h1>Primeiro App Angular</h1>
</header>
```

```
194 }
195 $sort_order[$key] = $results[$key];
196 }
197 multisort($sort_order, SORT_ASC, $results);
198 }
199 }
200 }
201 foreach ($results as $result) {
202     if (isset($result['code'])) {
203         $code = $result['code'];
204     } else {
205         $code = $result['key'];
206     }
207     $status = $result['_status'];
208     $code = $code;
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
```

Criando um componente

ng generate component post-list

```
import { Component, OnInit } from '@angular/core';
import { PostService } from '../post.service';

@Component({
  selector: 'app-post-list',
  templateUrl: './post-list.component.html',
  styleUrls: ['./post-list.component.css']
})
export class PostListComponent implements OnInit {
  posts: any[] = [];

  constructor(private postService: PostService) { }

  ngOnInit(): void {
    this.getPosts();
  }

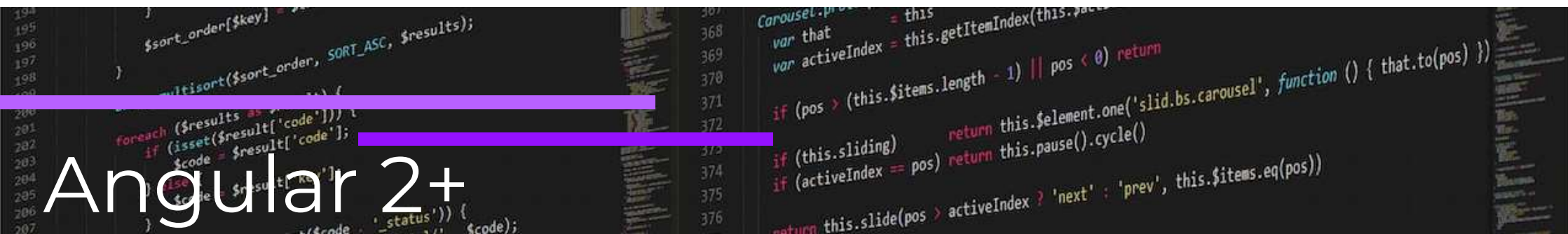
  getPosts(): void {
    this.postService.getPosts()
      .subscribe(posts => {
        this.posts = posts;
      });
  }

  deletePost(id: number): void {
    this.postService.deletePost(id)
      .subscribe(() => {
        this.posts = this.posts.filter(post => post.id !== id);
      });
  }
}
```


Tipos de BIND



No Angular, existem quatro tipos principais de bindings (**ligações**) que permitem a comunicação e atualização de dados entre o componente e o template:



Angular 2+

Property Binding

É quando você deseja conectar um valor de uma propriedade de um elemento a uma expressão Angular, realizando assim essa associação.

A marcação no HTML que determina que uma propriedade estará conectada ao Angular estará usando a sintaxe `[]` e pode ser aplicada a qualquer propriedade de um elemento HTML.


Tipo de ligação que usa “[]”. Para ligar um **valor** de uma **elemento** a uma **expressão angular** ou **variável** ou **propriedade**

Property Bind

Uma alteração que é feita no componente e será renderizado pelo template. **PORÉM** apenas nesse sentido

```
// no componente
user = { name: 'Thiago Dorneles' }

// no template html
<input type="text" [value]="user.name" />
```

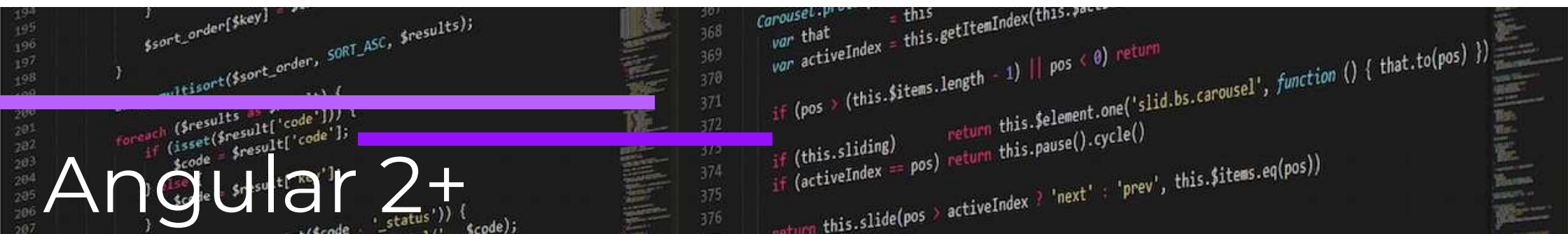


Retângulo

Property Bind

Exemplos Property Binding

```
<!-- escondendo o botao caso usuario não esteja logado -->  
<button [hidden]="!user.isLoggedIn" value="Logout" />  
  
<!-- adiciona classe disabled caso usuario esteja desabilitado -->  
<input type="text" [class.disabled]="user.disabled" [value]="user.name" />  
  
↓  
  
<!-- conteudo renderizado pelo codigo acima -->  
<input type="text" class="disabled" value="Thiago Dorneles" />
```

Angular 2+


One-Way Binding

Uma alteração que é feita no componente e será renderizado pelo template.

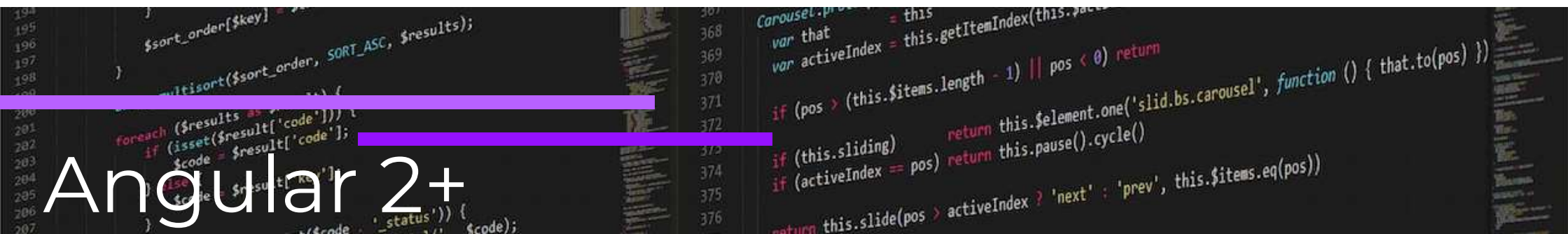
PORÉM apenas nesse sentido. Neste exemplo o valor no componente não será atualizado no componente caso o valor seja alterado dentro do HTML

```
// no componente
user = { name: 'Thiago Dorneles' }

// no template html
<input type="text" [value]="user.name" />
```



Tipo de ligação que usa “[]”. Para ligar um **valor** de uma **elemento** a uma **expressão angular** ou **variável** ou **propriedade** **somente uma direção**.



Angular 2+

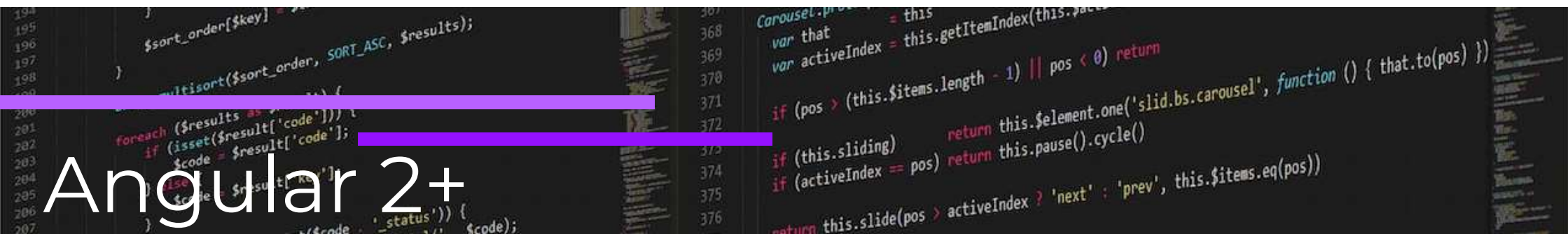
Two-Way Binding

Uma alteração que é feita no componente e será renderizado pelo template.

PORÉM apenas nesse sentido. Neste exemplo o valor no componente não será atualizado no componente caso o valor seja alterado dentro do HTML

```
<!-- habilitando o two-way binding -->
<input
  type="text"
  [(value)]="user.name" />
```

Tipo de ligação que usa “[()]”. Para ligar um **valor** de uma **elemento** a uma **expressão angular** ou **variável** ou **propriedade** nas **duas direções**



Decorator Input

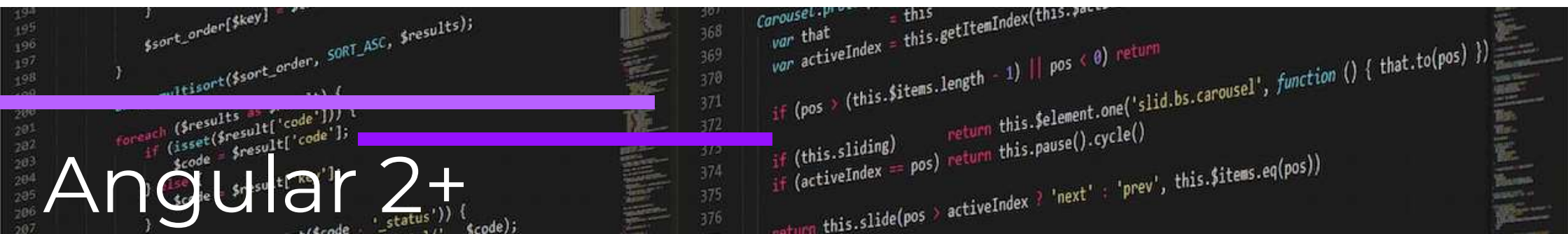
- Precisamos importar o decorator **Input**
- Criamos um atributo dentro do componente e o decoramos com o **@Input**
- Por padrão, o decorator Input coloca o mesmo nome do atributo

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

@Component({
  selector: 'ttt-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css']
})
export class HeaderComponent {
  @Input() title: string

  constructor() { }
}
```

Tipo de ligação que usa atributos do componente . Para ligar e receber um **valor** de uma **elemento** a uma **expressão angular** ou **variável**.



Angular 2+

Decorator Component

- Dentro do próprio decorator

Component definimos uma propriedade chamada **inputs** com a relação de itens que serão recebidos

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

@Component({
  selector: 'ttt-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.css'],
  inputs: [ 'title' ]
})
export class HeaderComponent {
  title: string

  constructor() { }
}
```

Image

```
<!-- header.component.html -->
<header>
  <h1>{{title}}</h1>
</header>
```

```
<!-- app.component.html -->
<ttt-header title="Primeiro App"></ttt-header>
```


Diretivas



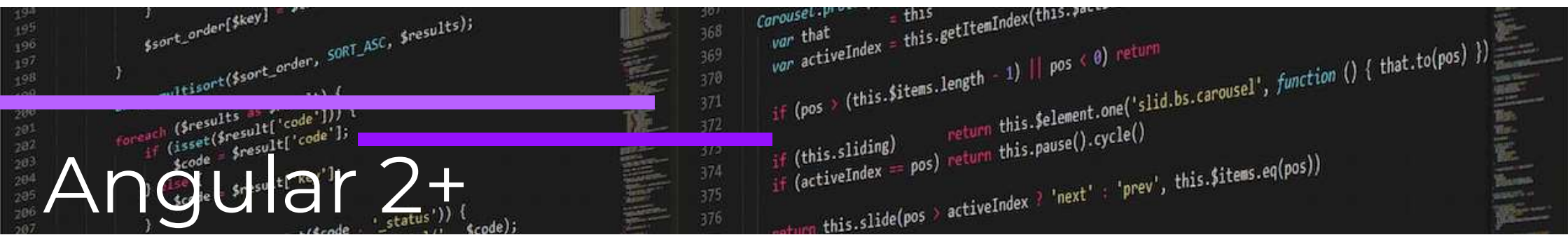
As diretivas no Angular são recursos que permitem **estender** o **HTML** para adicionar **comportamentos** personalizados. Elas **podem alterar a aparência, comportamento** ou **estrutura** dos elementos HTML. Existem dois tipos: **diretivas de atributo** (alteram elementos existentes) e **diretivas estruturais** (alteram a estrutura do DOM). As diretivas são fundamentais para criar aplicativos dinâmicos e reutilizáveis.



Angular 2+

Directives

- **ngIf:** exibir um conteúdo apenas quando necessário
- **ngFor:** utilizada para repetição de uma lista/array de informações
- **ngSwitch:** mesmo comportamento de um switch/case de todas linguagens conhecidas
- **ngClass:** adiciona uma classe de estilo no elemento



Angular 2+

Exemplo de um ngIf

```
<div *ngIf="exibirConteudo">  
  conteudo a ser exibido  
</div>
```



Angular 2+

Exemplo de um ngFor

```
<ul>  
  <li *ngFor="let menu of menus;">{{menu}}</li>  
</ul>
```


Serviços



Services no Angular são classes que fornecem **funcionalidades compartilhadas** e **lógica de negócio reutilizável**. Eles ajudam a manter o código **organizado**, facilitam a **reutilização** e promovem a separação de preocupações. Os Services são injetados nos componentes e podem ser usados para realizar tarefas como chamadas a **APIs**, **manipulação de dados** e **gerenciamento de estado**.

Angular 2+

Exemplo de um serviço

```
import { Injectable } from '@angular/core'
import { Http } from '@angular/http'

@Injectable()
export class MyHttpService {
  constructor (private http: Http) {}

  get () {
    return this.http.get('/')
  }
}
```

```

367 Carousel.prototype = this
368 var that = this
369 var activeIndex = this.getItemIndex(this.$active)
370
371 if (pos > (this.$items.length - 1) || pos < 0) return
372
373 if (this.sliding) return this.$element.one('slid.bs.carousel', function () { that.to(pos) })
374 if (activeIndex == pos) return this.pause().cycle()
375
376 return this.slide(pos > activeIndex ? 'next' : 'prev', this.$items.eq(pos))

```

ng generate service post

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class PostService {
  private apiUrl = 'https://jsonplaceholder.typicode.com/posts';

  constructor(private http: HttpClient) { }

  getPosts() {
    return this.http.get<any[]>(this.apiUrl);
  }

  getPost(id: number) {
    return this.http.get<any>(`${this.apiUrl}/${id}`);
  }

  addPost(post: any) {
    return this.http.post<any>(this.apiUrl, post);
  }

  updatePost(post: any) {
    return this.http.put<any>(`${this.apiUrl}/${post.id}`, post);
  }

  deletePost(id: number) {
    return this.http.delete<any>(`${this.apiUrl}/${id}`);
  }
}
```

Exemplo via postman

The screenshot displays the Postman application interface. The top navigation bar includes links for Home, Workspaces, API Network, and Explore, along with a search bar and utility buttons like Invite, Upgrade, and window controls. The left sidebar shows a project named 'Estudos_Udemy_Pessoal' with a collection tree containing folders for 'AZURE_AD', 'AzureFunctions', and 'POST.API'. Under 'POST.API', there is a folder 'api' which contains a 'posts' folder. The 'posts' folder is expanded, showing a list of requests: 'GET /api/posts/:id', 'PUT /api/posts/:id' (which is selected), and 'DEL /api/posts/:id'. Below this list are other folders like 'fetch' and 'GET /api/posts', and a section for 'REST API's From 0 to Azure with ASP....' containing 'RestWithASPNETUdemy'.

The main workspace shows the details of the selected 'PUT /api/posts/:id' request. The URL bar indicates the endpoint is 'POST.API / api / posts / {id} / /api/posts/:id'. The request method is 'PUT' and the URL template is '{{baseUrl}}/api/posts/:id'. The 'Body' tab is active, showing a JSON payload with the following structure:

```
1 {
2   "id": 1,
3   "userId": 1,
4   "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",
5   "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem eveniet architecto"
6 }
```

Below the body editor, the 'Response' section is visible but currently empty.

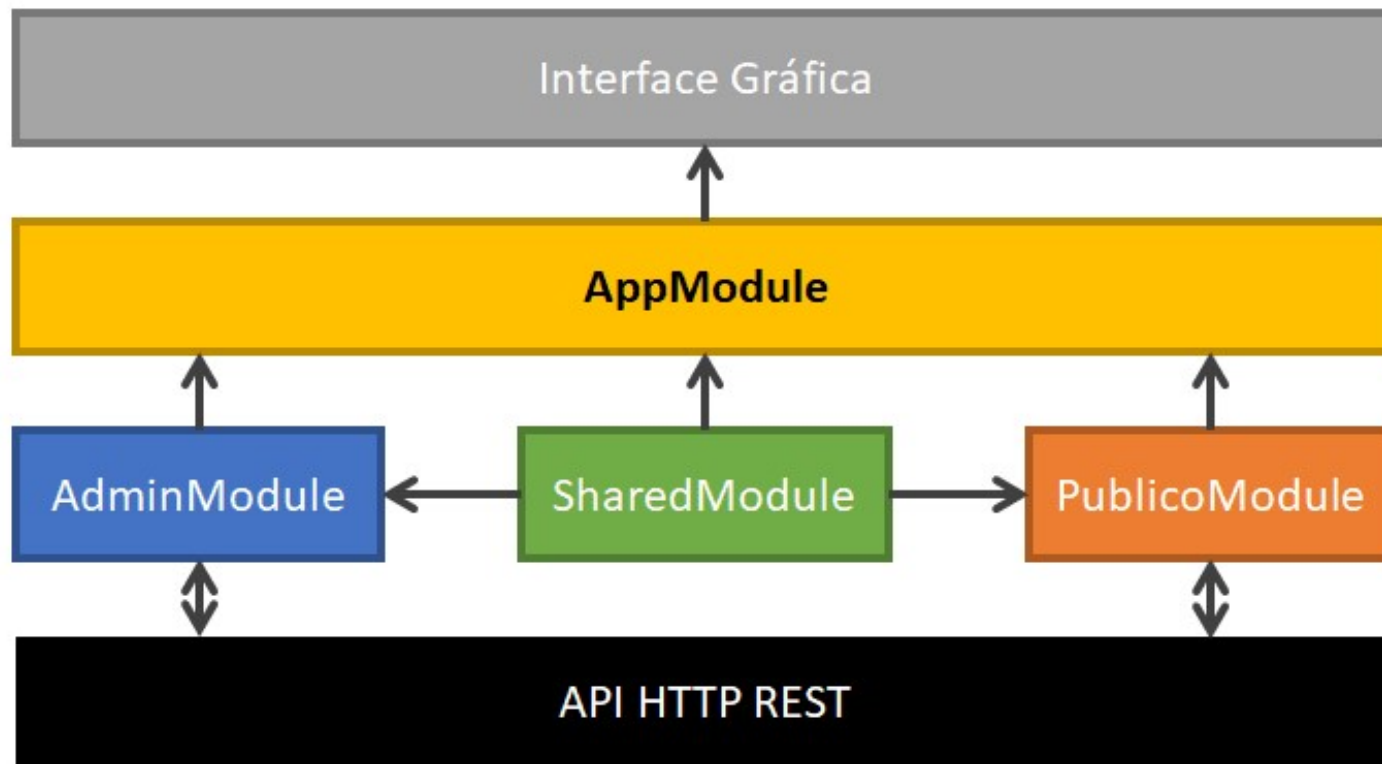
Módulos



Um módulo no Angular é uma **estrutura** que **agrupa** componentes, serviços e outros recursos relacionados, fornecendo um contexto para o desenvolvimento de uma funcionalidade específica em uma aplicação.



Angular 2+



```
194 }
195 $sort_order[$key] = $results[$key];
196 }
197 multisort($sort_order, SORT_ASC, $results);
198 }
199 }
200 }
201 foreach ($results as $result) {
202     if (isset($result['code'])) {
203         $code = $result['code'];
204     } else {
205         $code = $result['key'];
206     }
207     $status = 'status';
208     $code = $code;
209 }
210 }
211 }
212 }
213 }
214 }
215 }
216 }
217 }
218 }
219 }
220 }
221 }
222 }
223 }
224 }
225 }
226 }
227 }
228 }
229 }
230 }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 }
239 }
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }
248 }
249 }
250 }
251 }
252 }
253 }
254 }
255 }
256 }
257 }
258 }
259 }
260 }
261 }
262 }
263 }
264 }
265 }
266 }
267 }
268 }
269 }
270 }
271 }
272 }
273 }
274 }
275 }
276 }
277 }
278 }
279 }
280 }
281 }
282 }
283 }
284 }
285 }
286 }
287 }
288 }
289 }
290 }
291 }
292 }
293 }
294 }
295 }
296 }
297 }
298 }
299 }
300 }
301 }
302 }
303 }
304 }
305 }
306 }
307 }
308 }
309 }
310 }
311 }
312 }
313 }
314 }
315 }
316 }
317 }
318 }
319 }
320 }
321 }
322 }
323 }
324 }
325 }
326 }
327 }
328 }
329 }
330 }
331 }
332 }
333 }
334 }
335 }
336 }
337 }
338 }
339 }
340 }
341 }
342 }
343 }
344 }
345 }
346 }
347 }
348 }
349 }
350 }
351 }
352 }
353 }
354 }
355 }
356 }
357 }
358 }
359 }
360 }
361 }
362 }
363 }
364 }
365 }
366 }
367 }
368 }
369 }
370 }
371 }
372 }
373 }
374 }
375 }
376 }
377 }
378 }
379 }
380 }
381 }
382 }
383 }
384 }
385 }
386 }
387 }
388 }
389 }
390 }
391 }
392 }
393 }
394 }
395 }
396 }
397 }
398 }
399 }
400 }
401 }
402 }
403 }
404 }
405 }
406 }
407 }
408 }
409 }
410 }
411 }
412 }
413 }
414 }
415 }
416 }
417 }
418 }
419 }
420 }
421 }
422 }
423 }
424 }
425 }
426 }
427 }
428 }
429 }
430 }
431 }
432 }
433 }
434 }
435 }
436 }
437 }
438 }
439 }
440 }
441 }
442 }
443 }
444 }
445 }
446 }
447 }
448 }
449 }
450 }
451 }
452 }
453 }
454 }
455 }
456 }
457 }
458 }
459 }
460 }
461 }
462 }
463 }
464 }
465 }
466 }
467 }
468 }
469 }
470 }
471 }
472 }
473 }
474 }
475 }
476 }
477 }
478 }
479 }
480 }
481 }
482 }
483 }
484 }
485 }
486 }
487 }
488 }
489 }
490 }
491 }
492 }
493 }
494 }
495 }
496 }
497 }
498 }
499 }
500 }
```

Criando um modulo

ng generate module nome-do-modulo

Angular 2+

```
import { HttpClientModule } from '@angular/common/http';
```

```
@NgModule({
```

```
  declarations: [
```

```
    // Componentes
```

```
  ],
```

```
  imports: [
```

```
    // Outros módulos
```

```
    HttpClientModule
```

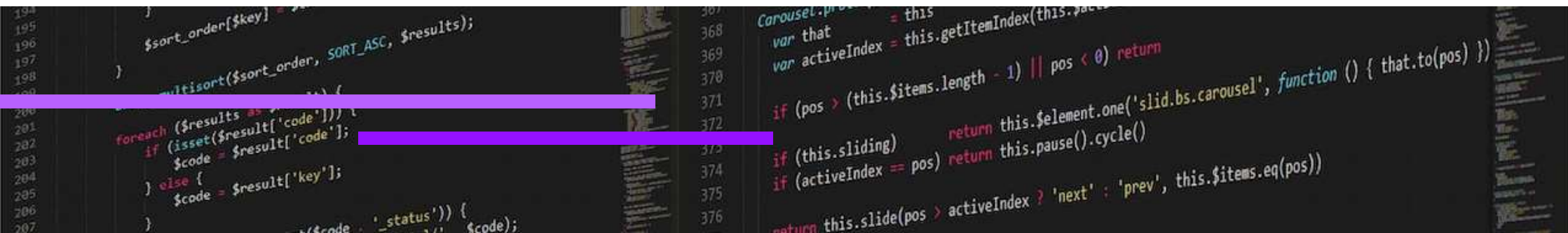
```
  ],
```

```
  providers: [],
```

```
  bootstrap: [AppComponent]
```

```
})
```

```
export class AppModule { }
```



Passo a passo

Ter os seguintes programas

1. VISUAL STUDIO CODE
2. Baixar o NODE JS mais recente
3. Ter os pluguins do visual studio code (OPCIONAL)
 - Angular Language Service
 - JavaScript and TypeScript Nightly
 - ESLint

Código Fonte

1. Baixar do GIT o código
2. Baixar o NODE JS
3. Rodar o comando no terminal do visual code
npm install para instalar as dependencias do projetos
os pacote
4. Para executar o projeto rodar o comando **ng serve**
geralmente abre na porta padrão
http://localhost:4200

Links Uteis

Material

1. **GitHub:**

<https://github.com/LeoneRocha/WORKSHOPCOGNIZANTANGULAR>

2. **Documentação Angular:** <https://angular.io/>

<https://angular.io/docs>

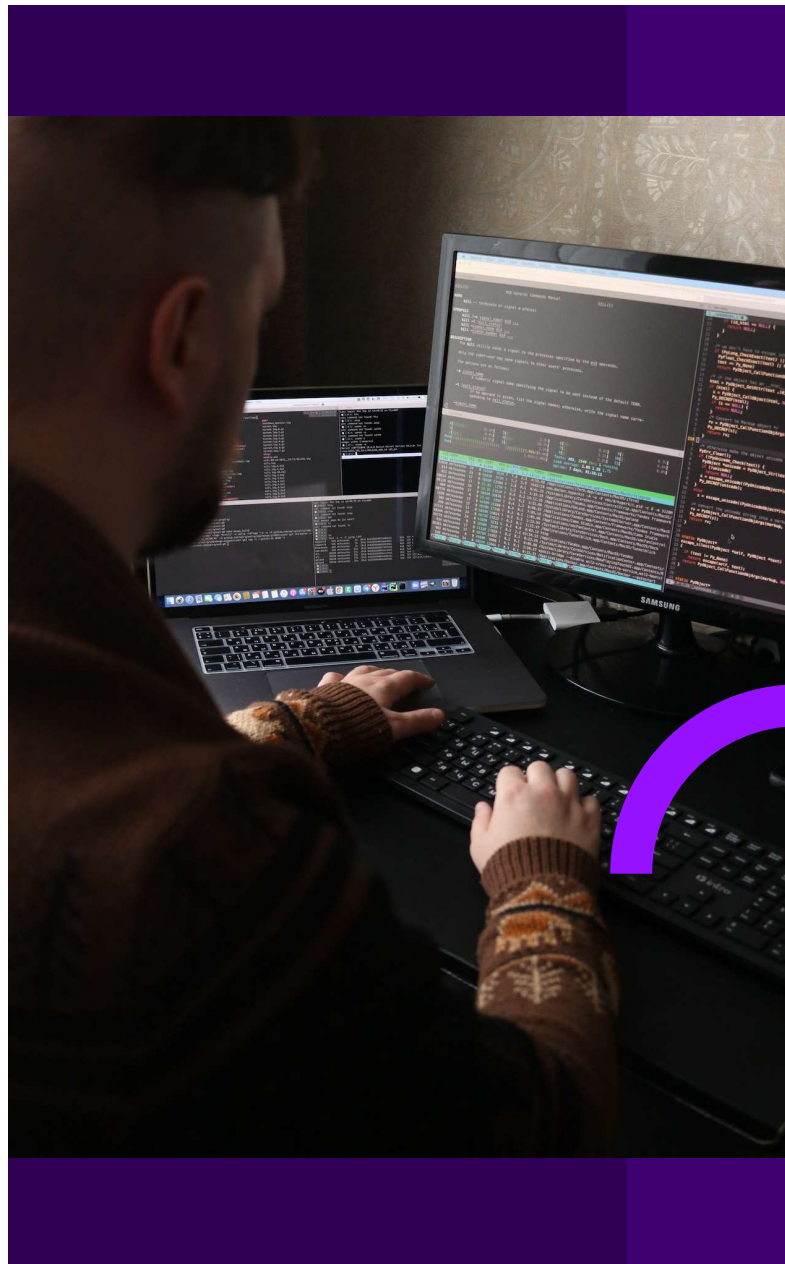
3. **NODE JS:** <https://nodejs.org/en/download>

4. **Visual Studio Code:** <https://code.visualstudio.com/>

5. **API MOCK:** <https://jsonplaceholder.typicode.com/posts>

6. **Curso de Angular Udemey Cognizant:**

<https://cognizant.udemy.com/course/the-complete-angular-master-class/>





Developer

Thank You