

## Laboratório **Angular**

Professor:  
**Vitor Figueiredo**

Disciplina  
**Sistemas Distribuídos**

Versão:  
**1.0**



Conceitos básicos



Integração com REST



JavaScript e TypeScript



Component e Service

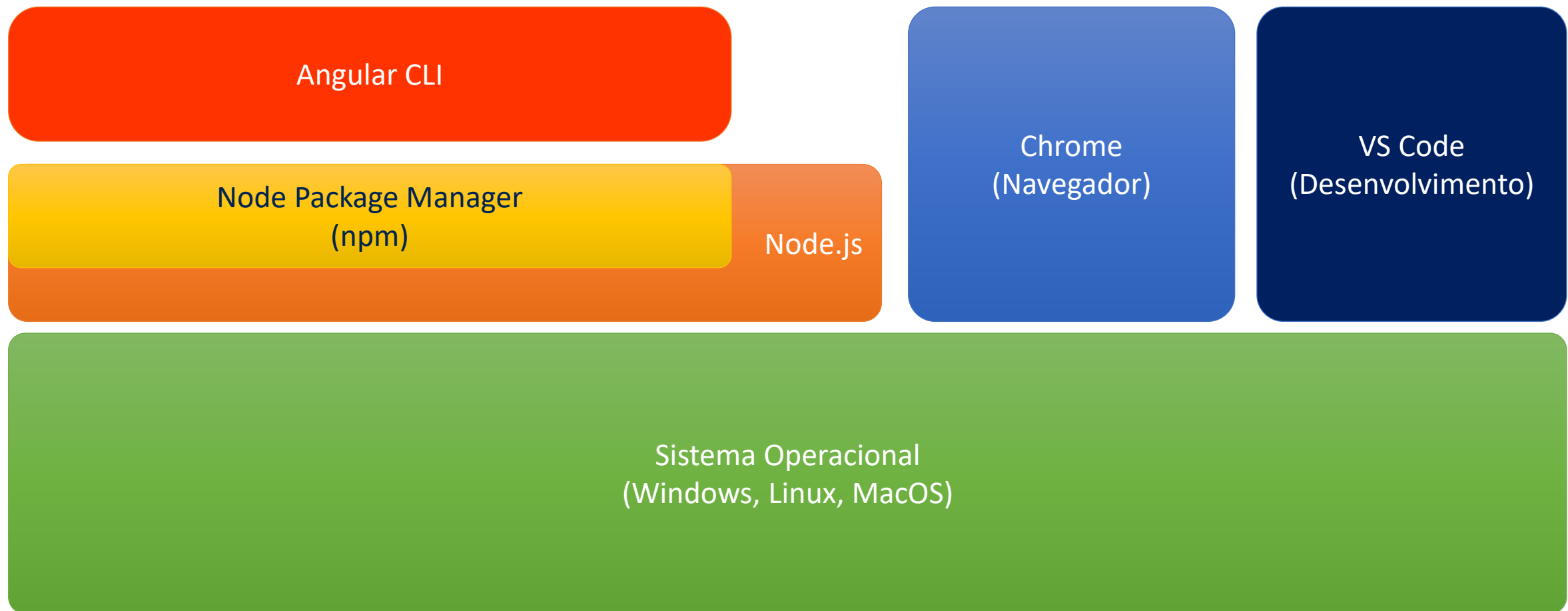
# *Conceitos básicos*

# *Integração com APIs REST*

- Plataforma de desenvolvimento construída sobre o TypeScript
- Framework baseado em **componentes** para criar aplicações SPA
- Coleção de bibliotecas para diversas funções:
  - Gerenciamento de formulários
  - Roteamento
  - Comunicação cliente-servidor
  - ...
- Ferramentas de desenvolvimento, build e testes

- Significa **Single Page Application**
- A partir de um componente inicial, existe uma “*lacuna*” onde se preenche seu conteúdo dinamicamente
- Em sistemas web tradicionais, dizemos navegação para página
- Em sistemas SPA, dizemos **ativação de uma rota**
- A ativação de uma rota é justo o preenchimento da “*lacuna*” com conteúdo

- Angular Client Line Interface – CLI
- Componentes, Módulos e Services
- Injeção de dependência
- Templates
- Bibliotecas de primeira classe:
  - Angular HttpClient
  - Angular Router
  - Angular Forms
  - Angular Animations
  - Angular PWA
  - Angular Schematics





- > CLI significa **Command Line Interface**
- > Um projeto Angular usa muitos módulos e bibliotecas e configurá-los “na mão” é muito complicado e trabalhoso
- > O **Angular CLI** cria e configura nossos projetos automaticamente.
- > Usando o **npm**, instalamos o **Angular CLI** assim:

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

↑  
Global. Precisa ter acesso de administrador

> Há 2 estilos de instalação:

\* Versão atual:

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

\* Versão antiga:

```
npm install -g @angular/cli@9.7.2
```

## > Desinstalando Angular CLI

```
npm uninstall -g @angular/cli  
npm cache clean --force
```

## Configurações

>Abrir terminal de comando e consultar a versão do Node:

```
node --version
```

Nota: A versão ideal é acima da 12




>Se o NodeJS não estiver instalado, baixar a versão LTS do site oficial e instalar:

<https://nodejs.org/en/download/>

Downloads

Latest LTS Version: **16.15.1** (includes npm 8.11.0)

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-v16.15.1-x64.msi</small>	 macOS Installer <small>node-v16.15.1.pkg</small>	 Source Code <small>node-v16.15.1.tar.gz</small>

Windows Installer (.msi)      32-bit      64-bit

>Abrir um terminal e digitar a versão mais atual do Angular:

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

Nota: Se houver outra versão do Angular já instalada, desinstalar e instalar a mais atual

>Verificar a versão instalada:

```
ng version
```

```
c:\DEV\git_clones>ng version
```

Angular CLI

```
Angular CLI: 14.0.1
Node: 16.13.2
Package Manager: npm 8.1.2
OS: win32 x64
```

```
Angular:
...
```

Package	Version
@angular-devkit/architect	0.1400.1 (cli-only)
@angular-devkit/core	14.0.1 (cli-only)
@angular-devkit/schematics	14.0.1 (cli-only)
@schematics/angular	14.0.1 (cli-only)

- > Abrir o terminal de comando
- > Ir para **C:\LAB\ws\_sd\_labs**
- > Entrar na pasta **curso-project** (criar caso não exista)
- > Criar o projeto **curso-frontend**, sem os testes unitários e sem arquivos do git:

```
ng new curso-frontend --skip-tests --skip-git
```

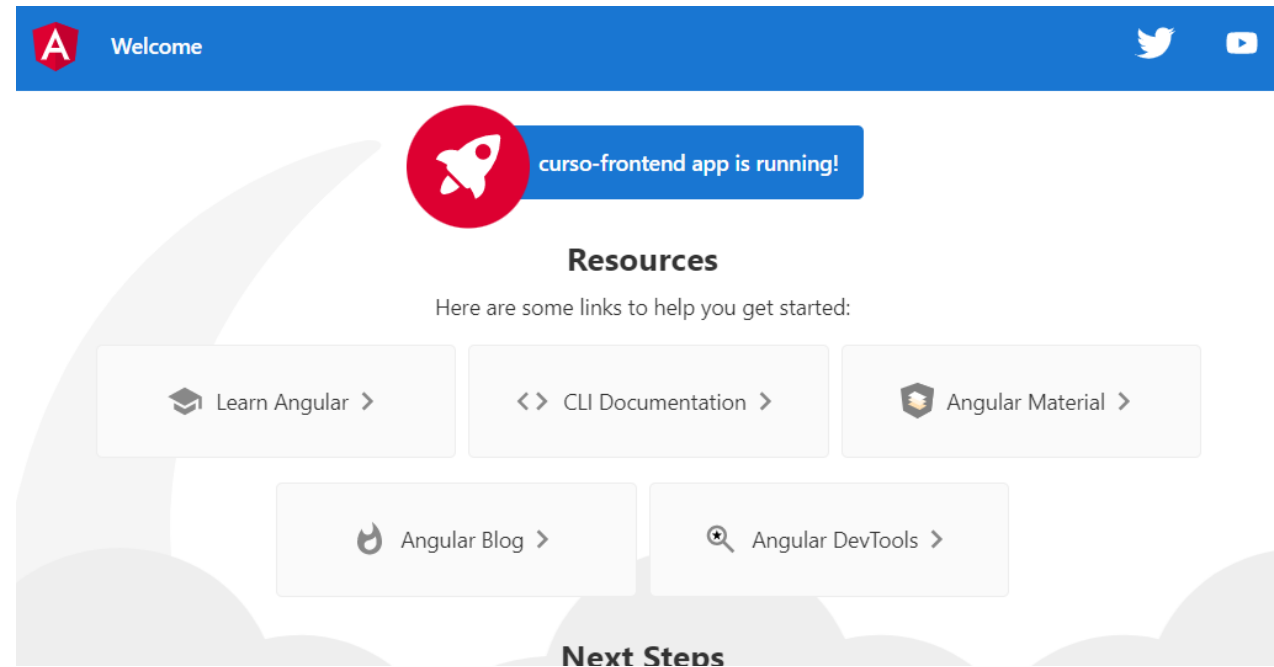
>Entrar na pasta criada e subir o servidor Angular

```
cd curso-frontend
```

```
ng serve --open
```

Atenção: É **serve** e não **server**

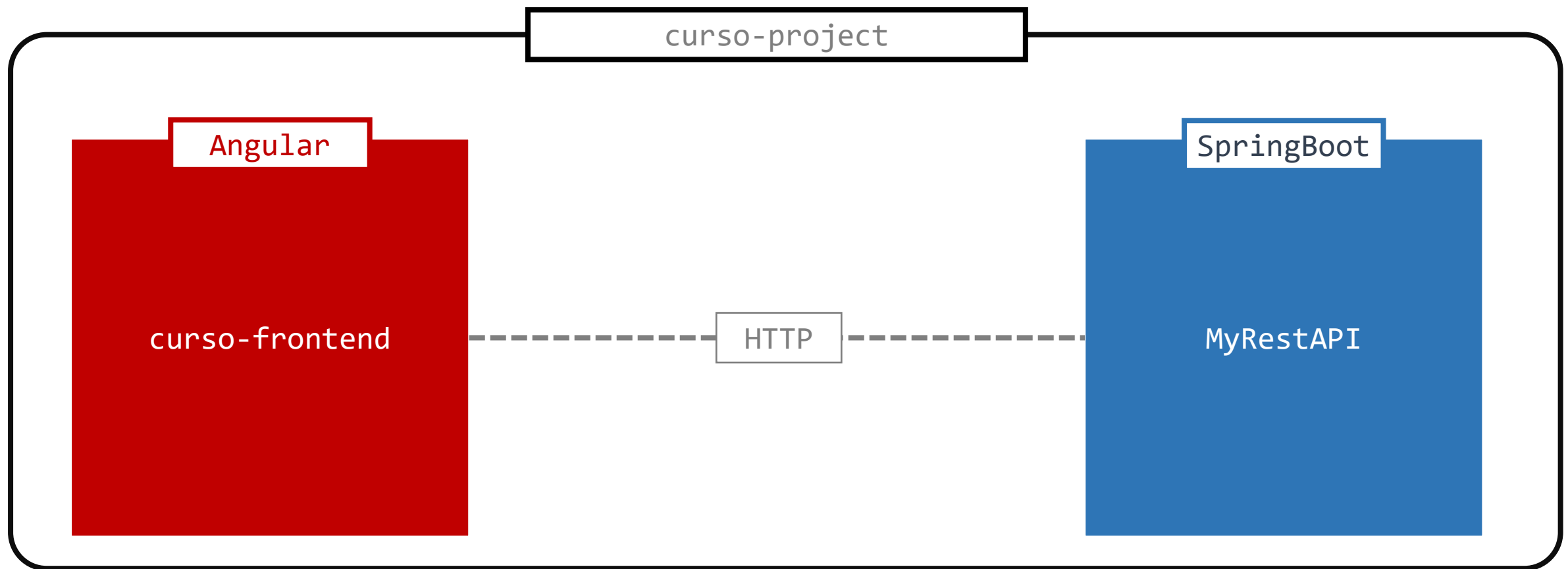
>Quando o navegador, vai exigir:





> Para encerrar o servidor, teclar **Ctrl + C**

# *Integração com APIs REST*



## Subindo serviço *MyRestAPI*

*Recuperando todos os cursos*

<http://localhost:8080/curso>

*Recuperando um curso específico*

<http://localhost:8080/curso/1>

## Plugins recomendados:

- **Dracula Official:** É um tema de cores escuro projetado para dar conforto visual ao desenvolvedor  
Assim que instalar, entre as opções de Color Theme, “Dracula” e “Dracula Soft”, selecione a primeira.
- **Color Highlight:** Esta extensão vai colocar uma borda ou cor de fundo toda vez que escrevemos um código RGB (exemplo `#7159c1`). É bem útil para editar propriedades nos arquivos CSS.
- **Material Icon Theme:** Usa ícones elegantes e coloridos conforme o tipo de arquivo.

# *JavaScript e TypeScript*

## *A grande mudança na linguagem JavaScript*

ECMAScript 6 = ECMAScript 2015 = ES2015 = ES6

## Exemplos de novas features:

### Arrow Function

#### Programação funcional:

- `map()`
- `reduce()`

### Variáveis **let** e **const**

#### Recursos de **Orientação a Objeto**:

- Classes
- Construtores
- `get` e `set`

### Template String

#### Promisses



- **Arrow Function:** maneira simplificada e elegante de escrever funções

```
form.addEventListener('submit', function(event) {  
    // ...  
})
```

```
form.addEventListener('submit', (event) => {  
    // ...  
})
```

```
campos.forEach(function(campo) {
```

```
campos.forEach( (campo) => {
```

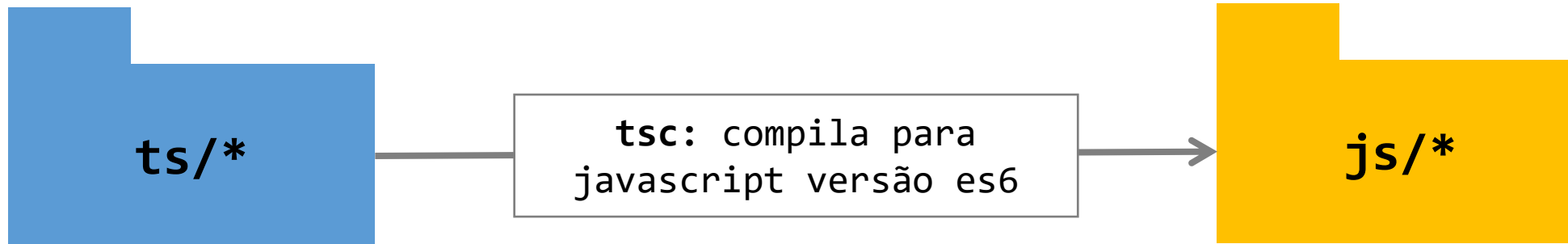
- **let e const**
  - Restringuem ao escopo local
  - Menor probabilidade de bug
  - **let**: usado para declarar variáveis
  - **const**: usado para declarar constantes. Usado também para declarar functions.
  
- **var**
  - sempre é de escopo global
  - Mais bugs

- Criado pelo Microsoft
- É um superset do ES6:
  - Tipagem estática: string, number, etc
  - Encapsulamento real
  - Generics (coleções com tipo)
  - Decorators (meta-programação)
- Usa a plataforma **Node.js**
- Vantagens de usar tipagem estáticas:
  - **Compilation-errors** são melhores que **Runtime-errors**
  - Auto-completamento pela IDE

*superset*  
**TypeScript**

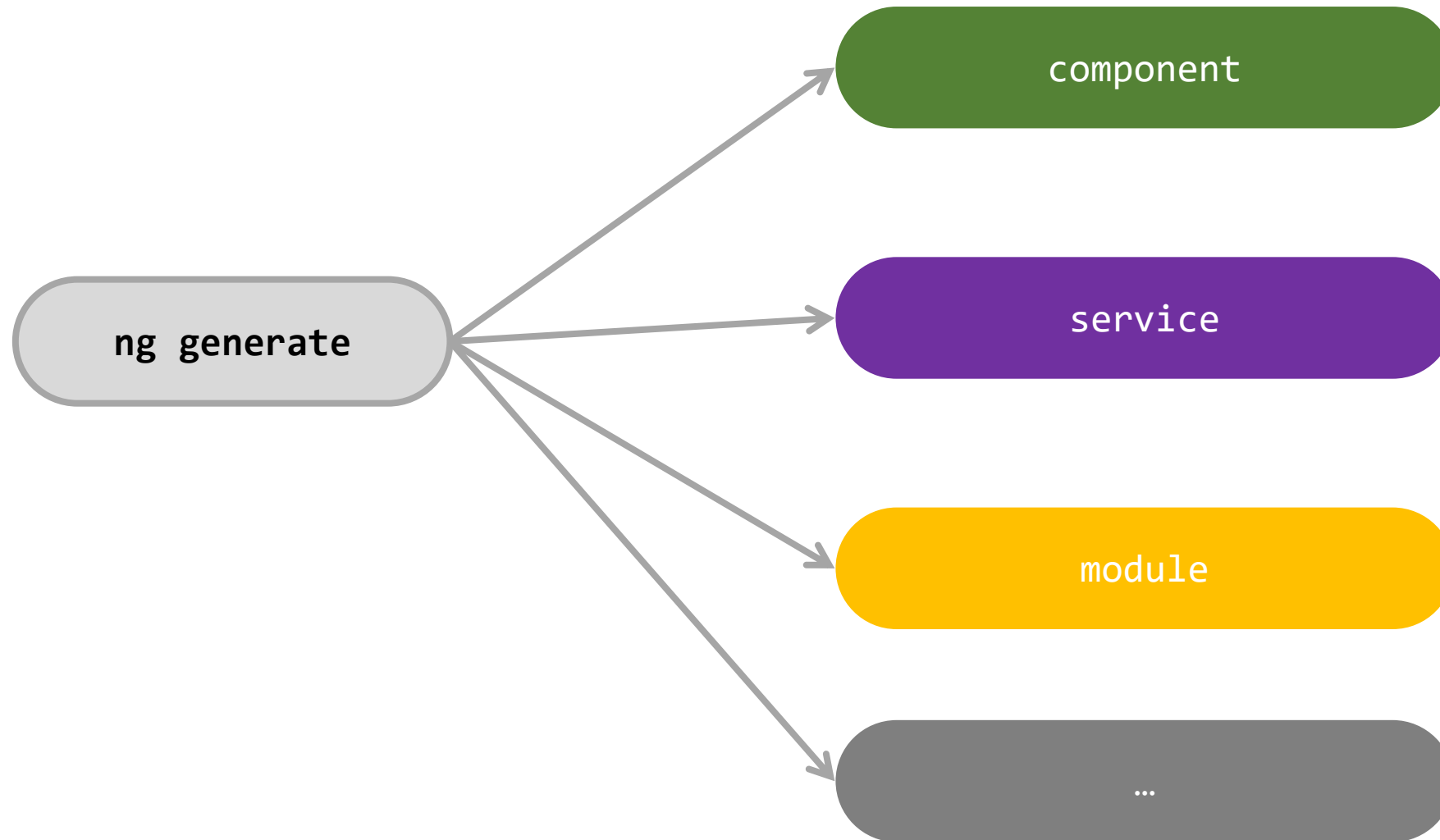
**JavaScript**  
2015

- **tsc** é compilador do TypeScript
- Navegadores só entendem JS
- Todo código TS é compilado para JS com as devidas verificações
- Tudo de forma transparente

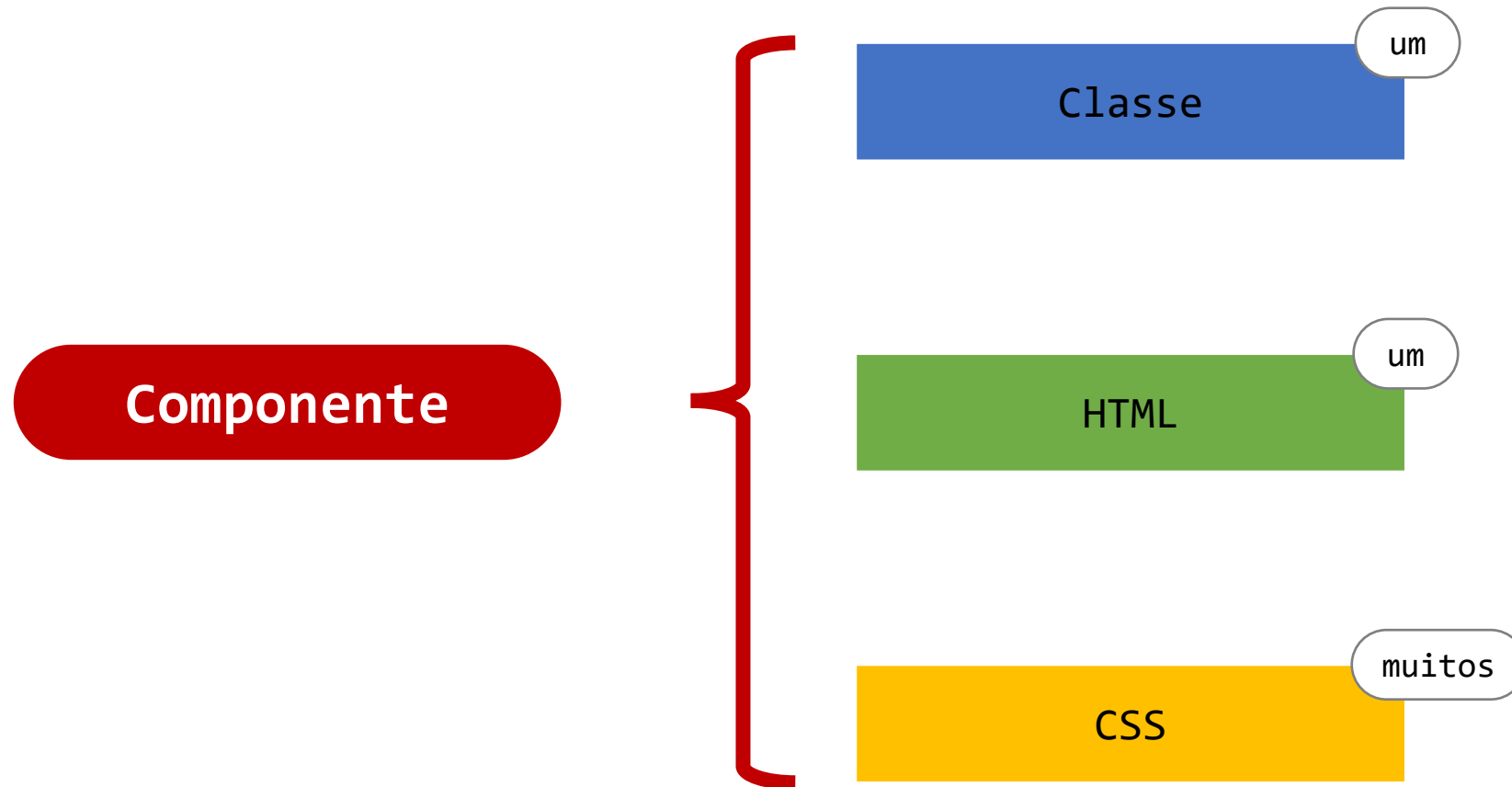


***Vamos aprender TypeScript conforme aprendemos Angular***

# *Component e Service*







app.component.ts X

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9
10     //atributos ...
11
12     //métodos ...
13
```

HTML

CSS

Classe

> Em **index.html**, temos uma tag “estranha”:

```
index.html X
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>ProjetoEstoque</title>
6   <base href="/">
7   <meta name="viewport" content="width=device-width, initial-scale=1">
8   <link rel="icon" type="image/x-icon" href="favicon.ico">
9 </head>
10 <body>
11   <app-root></app-root>
12 </body>
13 </html>
14
```

← Não é tag HTML

> Esta tag é o link com o respectivo **Component**

```
index.html X
1 <!doctype html>
2 <html lang="en">
3 <head>
4   <meta charset="utf-8">
5   <title>ProjetoEstoque</title>
6   <base href="/">
7   <meta name="viewport" content="width=device-width, initial-scale=1">
8   <link rel="icon" type="image/png" href="assets/icon.png">
9 </head>
10 <body>
11   <app-root></app-root>
12 </body>
13 </html>
14
```

```
app.component.ts X
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'projeto-estoque';
10 }
11
```

*“Associação de dados entre classes e páginas”*

*>Tem 2 maneiras de fazer Data Binding:*

- 1) Angular Expression**
- 2) OneWay Binding**

## 1) Angular Expression:

- O simbolo `{{ }}` é uma **AngularExpression**
- Usado em: **no conteúdo de tags**
- Exemplos:
  - `<span>{{ title }}</span>`
  - `<h1>{{ capítulo }}</h1>`
  - `<div>{{ outraVariavel }}</div>`

> O conteúdo HTML se encontra em **app.component.html**

```
app.component.html X
344
345   <span>{{ title }} app is running!</span>
346
347 |
348   <svg id="rocket-smoke" alt="Rocket Ship Sm
```

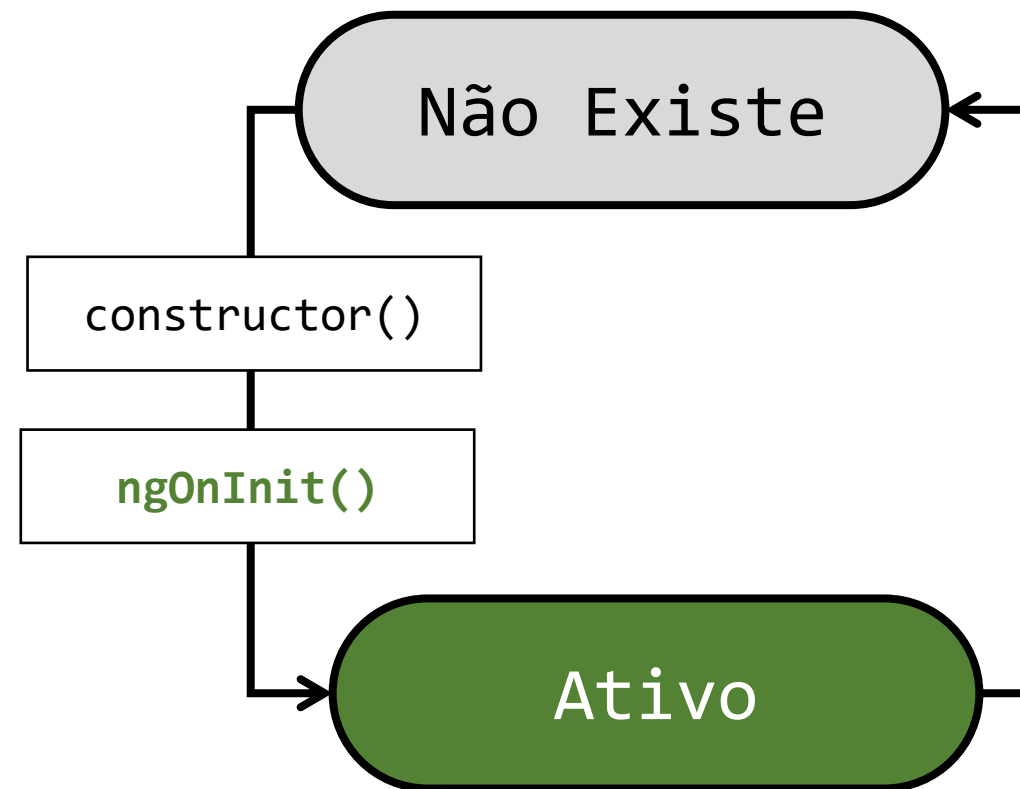
```
app.component.ts X
src > app > app.component.ts > ...
1   import { Component } from '@angular/core';
2
3   @Component({
4     selector: 'app-root',
5     templateUrl: './app.component.html',
6     styleUrls: ['./app.component.css']
7   })
8   export class AppComponent {
9     title = 'curso-frontend';
10  }
11
```

*Atributo é público.  
NÃO precisa de método get*

\*A injeção de dependência acontece no **construtor**

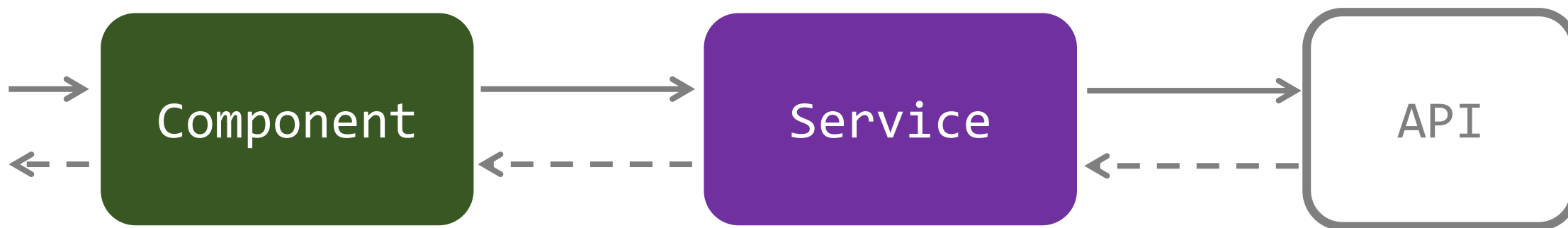
\*Inicializações que dependem de objetos injetados devem acontecer no método de *callback* de **ngOnInit**

\*ngOnInit é da interface OnInit

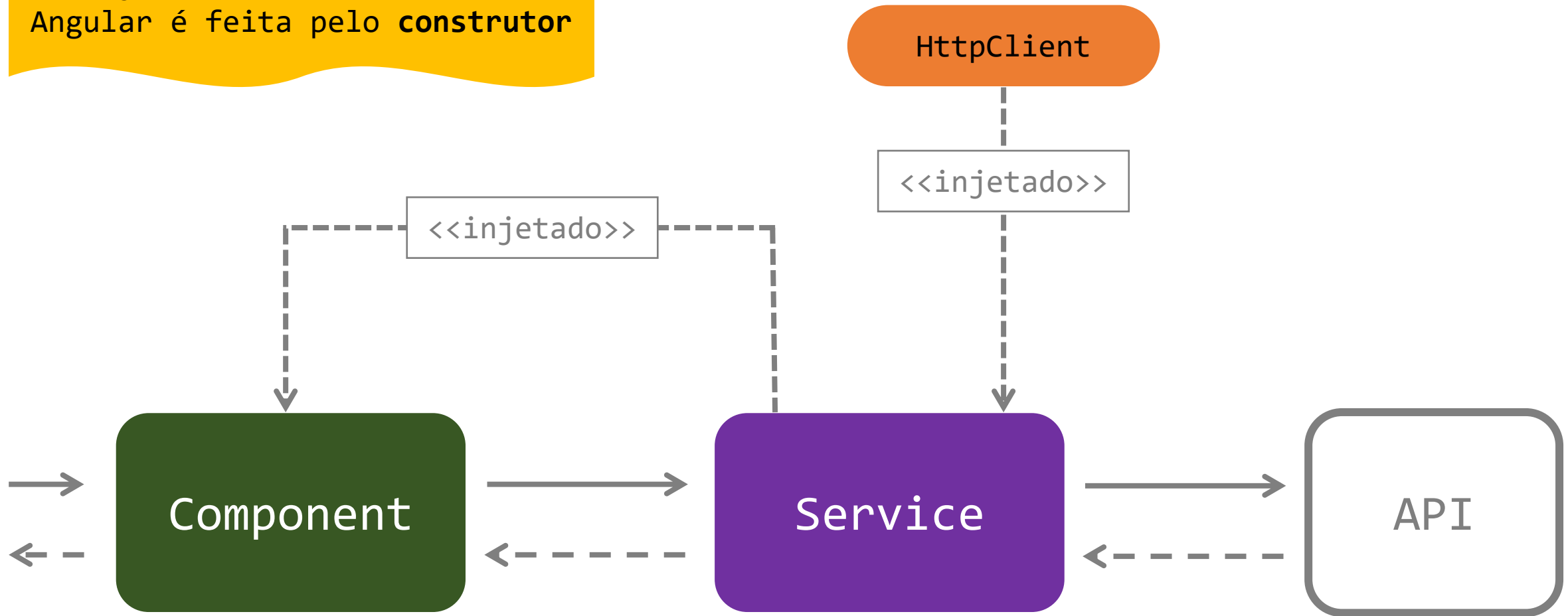




>A idéia da classes **Service** é ser especializada em acessar API's.



Injeção de dependência no Angular é feita pelo **construtor**



- >O Angular disponibiliza o **HttpClient** como biblioteca para acesso a APIs
- >Faz parte do módulo **HttpClientModule** e portanto precisa ser importado em **app.module.ts**:

```
@NgModule({
  declarations: [
    AppComponent,
    FotoComponent,
    ProdutoGridComponent,
    ProdutoFormComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

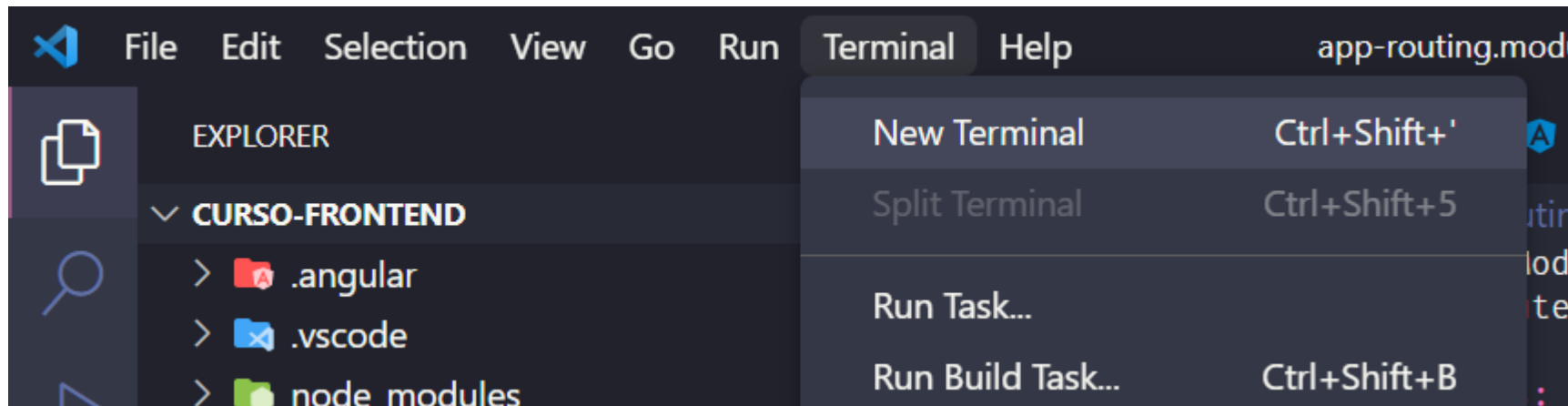
***Primeiro componente***

>Na pasta do projeto criado, digitar o comando para abrir o VSCode:

```
code .
```

>Analisar as pastas e os arquivos criados

>Abrir um terminal no próprio VSCode:

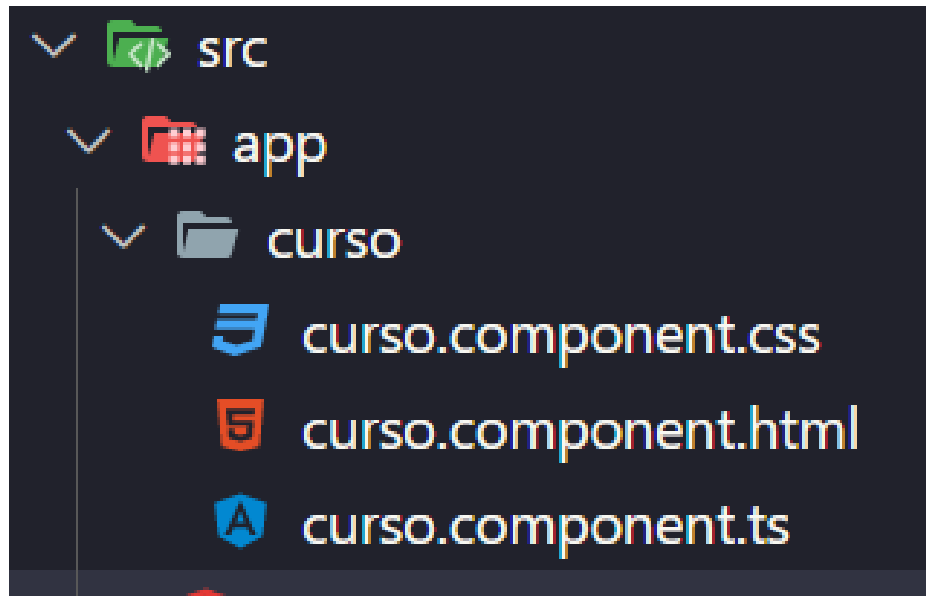


>Gerar o componente **curso**:

```
ng g c curso
```

Importante: O servidor Angular deve estar parado sempre que se genera um componente

>A pasta curso será criado com os respectivos artefatos:



>Analisar os artefatos

# *Rotas*



- > Em aplicações SPA, não fazemos de “navegação entre páginas”. Fazemos “**ativação de rotas**”.
- > Rotas é uma de-para entre a URL (path) e o Componente:
- > Elas são configuradas no módulo em **app.routing.module.ts**:

## 1. Configuramos o **path** e o **component**:

```

A app-routing.module.ts X
src > app > A app-routing.module.ts > ...
1  import { CursoComponent } from './curso/curso.component';
2  import { NgModule } from '@angular/core';
3  import { RouterModule, Routes } from '@angular/router';
4
5  const routes: Routes = [
6    {
7      path: '/curso',
8      component: CursoComponent
9    }
10 ];
11
12 @NgModule({
13   imports: [RouterModule.forRoot(routes)],
14   exports: [RouterModule]
15 })
16 export class AppRoutingModule { }
17

```

2. Para que funcione, o HTML do componente App deve ter somente a tag:

```
app.component.html X
1
2 <router-outlet></router-outlet>
3
```

Esta tag é a “lacuna”!  
Será substituída pelo HTML do  
componente selecionado pela  
rota

## ***Configurando Rota para o componente Curso***

1. Em app-routing.module.ts, configurar uma entrada na constante **routes**:

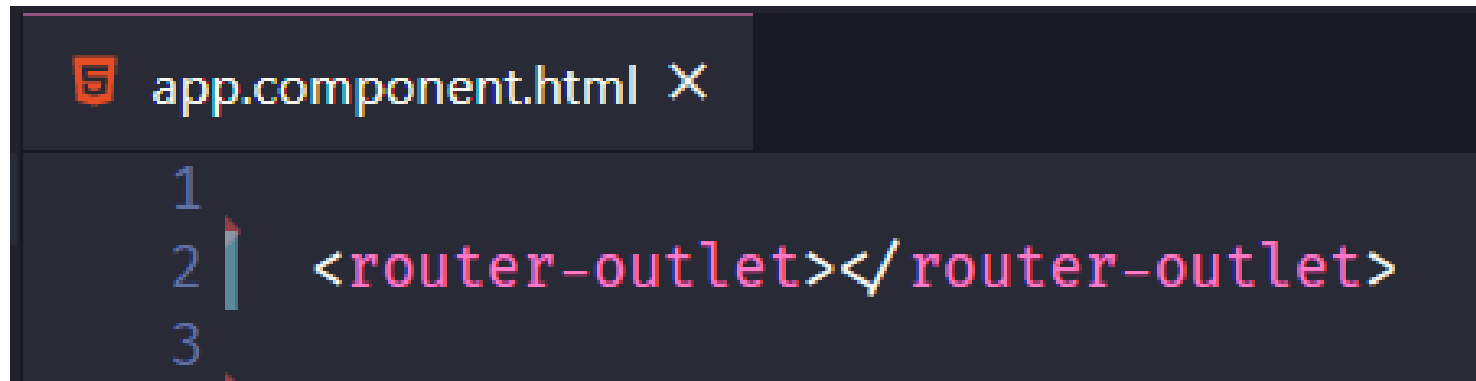
```

A app-routing.module.ts X
src > app > A app-routing.module.ts > ...
1  import { CursoComponent } from './curso/curso.component';
2  import { NgModule } from '@angular/core';
3  import { RouterModule, Routes } from '@angular/router';
4
5  const routes: Routes = [
6    {
7      path: '/curso',
8      component: CursoComponent
9    }
10 ];
11
12 @NgModule({
13   imports: [RouterModule.forRoot(routes)],
14   exports: [RouterModule]
15 })
16 export class AppRoutingModule { }
17

```

\*Configurando uma rota para o componente **curso**:

2. Usar a tag **<router-outlet>**



```
app.component.html X
1
2 <router-outlet></router-outlet>
3
```

3. Acessar <http://localhost:4200/curso> e ver o resultado

***Continua na próxima aula***

