

UNIÃO EDUCACIONAL DE CASCAVEL - UNIVEL
FACULDADE DE CIÊNCIAS SOCIAIS APLICADAS DE CASCAVEL
CURSO TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

NATALIA ALVES DE SOUZA

RELATÓRIO DE HORAS COMPLEMENTARES DA DISCIPLINA DE SISTEMAS
DISTRIBUÍDOS

CURSO DE ANGULAR 2

Cascavel

2017

Sumário

1 INTRODUÇÃO	6
2 DEFINIÇÃO.....	6
3 VÍDEO #01: INTRODUÇÃO AO ANGULAR 2	7
3.1 O que será estudado no curso?	7
3.2 O que é necessário saber para realizar o curso?.....	7
3.3 O que é o Angular 2?	7
Links úteis.....	7
3.4: Visão geral sobre os blocos	7
4 VÍDEO #02: AMBIENTE DE DESENVOLVIMENTO (NODE.JS, TYPESCRIPT, ANGULAR CLI)	8
5 VÍDEO #03: HELLO, WORLD! CRIANDO PRIMEIRO PROJETO E O PRIMEIRO COMPONENTE.....	10
5.1: Hello World! Criando meu primeiro projeto	10
5.2 Criando componente manualmente.....	12
Dificuldade encontrada	14
5.3 Criando componente automaticamente.....	14
6 VÍDEO #04: INTRODUÇÃO AO TYPE SCRIPT PARA ANGULAR.....	16
Dificuldade encontrada	19
Links úteis.....	19
7 VÍDEO #05: MÓDULOS (ngMODULE).....	19
7.1 Criando módulo.....	20
8 VÍDEO #06: INTRODUÇÃO AOS TEMPLATES	21
9 VÍDEO #07: INTRODUÇÃO AOS SERVIÇOS E INJEÇÃO DE DEPENDÊNCIA (DI).....	23
10 VÍDEO #08: DICAS PLUGINS ANGULAR PARA ATOM E VS CODE.....	27
10.1 Dicas de plugins para o Atom	27
10.2 Dicas de plugins para o Visual Studio Code	27
11 VÍDEO #09: <i>PROPERTYBINDING</i> E INTERPOLAÇÃO.....	27
12 VÍDEO #10: CLASS E STYLE BINDING	29
Dificuldade encontrada	36
Link útil	36
13 VÍDEO #11: EVENT BINDING	36
Link útil:	38
14 VÍDEO #12: TWO-WAY DATA BINDING.....	38
15 VÍDEO #13: REUSANDO COMPONENTES <i>INPUT PROPERTIES</i>	39
16 VÍDEO #14: EMITINDO EVENTOS COM <i>OUTPUT PROPERTIES</i>	41

17 VÍDEO #15: CICLO DE VIDA DE UM COMPONENTE	43
18 VÍDEO #16: ACESSO AO <i>DOM</i> E AO <i>TEMPLATE</i> COM <i>VIEWCHILD</i>	46
19 VÍDEO #17: ANGULAR CLI: INSTALAÇÃO E CRIAÇÃO DE PROJETOS: <i>NG NEW</i> E <i>NG SERVE</i>	47
20 VÍDEO #18: CRIANDO <i>COMPONENTS</i> , <i>SERVICES</i> : <i>NG GENERATE</i>	49
21 VÍDEO #19: ANGULAR CLI: USANDO PRÉ-PROCESSADORES (<i>SASS</i> , <i>LESS</i> , <i>STYLUS</i>).....	49
22 VÍDEO #20: ANGULAR CLI: <i>NG LINT</i> , <i>NG TEST</i> , <i>NG E2E</i>	50
Link útil	53
23 VÍDEO #21: ANGULAR CLI: ESTRUTURA DO PROJETO.....	53
24 VÍDEO #22: ANGULAR CLI: GERANDO BUILD DE PRODUÇÃO	57
25 VÍDEO #23: ANGULAR CLI: INSTALANDO BIBLIOTECAS (<i>BOOTSTRAP</i> , <i>JQUERY</i> , <i>MATERIALIZE</i> , <i>LODASH</i>)	59
Link útil	59
26 VÍDEO #24: ANGULAR CLI: INTRODUÇÃO E TIPO DE DIRETIVAS NO ANGULAR 2	59
27 VÍDEO #25: ANGULAR CLI: DIRETIVAS: <i>NGIF</i>	60
28 VÍDEO #26: ANGULAR CLI: DIRETIVAS: <i>NGSWITCH</i> , <i>NGSWITCHCASE</i> E <i>NGSWITCHDEFAULT</i>	61
29 VÍDEO #27: ANGULAR CLI: DIRETIVAS: <i>NGFOR</i>	62
30 VÍDEO #28: ANGULAR CLI: DIRETIVAS: SOBRE O <i>*</i> E <i>TEMPLATE</i>	64
31 VÍDEO #29: ANGULAR CLI: DIRETIVAS: <i>NGCLASS</i>	64
Link útil	65
32 VÍDEO #30: ANGULAR CLI: DIRETIVAS: <i>NGSTYLE</i>	65
33 VÍDEO #31: OPERADOR <i>ELVIS</i> (" <i>?</i> ")	66
34 VÍDEO #32: <i>NG-CONTENT</i>	67
35 VÍDEO #33: CRIANDO UMA DIRETIVA DE ATRIBUTO	68
36 VÍDEO #34: DIRETIVAS <i>HOSTLISTENER</i> E <i>HOSTBINDING</i>	69
37 VÍDEO #35: DIRETIVAS: <i>INPUT</i> E <i>PROPERTY BINDING</i>	70
38 VÍDEO #36: CRIANDO UMA DIRETIVA DE ESTRUTURA (<i>NGELSE</i>).....	71
39 VÍDEO #37: INTRODUÇÃO A SERVIÇOS.....	71
40 VÍDEO #38: CRIANDO UM SERVIÇO	71
41 VÍDEO #39: INJEÇÃO DE DEPENDÊNCIA (<i>DI</i>) + COMO USAR UM SERVIÇO EM UM COMPONENTE.....	72
42 VÍDEO #40: ESCOPO DE INSTÂNCIAS DE SERVIÇOS + MÓDULOS (<i>SINGLETON</i> E VÁRIAS INSTÂNCIAS).....	72
43 VÍDEO #41: COMUNICAÇÃO ENTRE COMPONENTES USANDO SERVIÇOS (<i>BROADCAST</i> DE EVENTOS).....	72
Dificuldade encontrada	73
44 VÍDEO #42: INJETANDO UM SERVIÇO EM OUTRO SERVIÇO.....	73

45 VÍDEO #43: PIPES (USANDO PIPES, PARÂMETROS E PIPES ANINHADOS)	73
Link útil	75
46 VÍDEO #44: CRIANDO UM PIPE	75
47 VÍDEO #45: APLICANDO LOCALE (INTERNACIONALIZAÇÃO) NOS PIPES.....	77
48 VÍDEO #46: PIPES: CRIANDO UM PIPE "PURO"	78
49 VÍDEO #47: PIPES: CRIANDO UM PIPE "IMPURO"	78
50 VÍDEO #48: PIPES: ASYNC (ASSÍNCRONO).....	79
51 VÍDEO #49: ROTAS: INTRODUÇÃO	80
Link útil	80
52 VÍDEO #50: CONFIGURANDO ROTAS SIMPLES	80
53 VÍDEO #51: ROTAS ROUTERLINK: DEFININDO ROTAS NO TEMPLATE	82
54 VÍDEO #52: ROTAS: APLICANDO CSS EM ROTAS ATIVAS	83
55 VÍDEO #53: ROTAS: DEFININDO E EXTRAINDO PARÂMETROS DE ROTEAMENTO	84
56 VÍDEO #54: ROTAS: ESCUTANDO MUDANÇAS NOS PARÂMETROS DE ROTEAMENTO	85
57 VÍDEO #55: ROTAS IMPERATIVAS: REDIRECIONAMENTO VIA CÓDIGO	86
58 VÍDEO #56: ROTAS: DEFININDO E EXTRAINDO PARÂMETRO DE URL (QUERY)	89
59 VÍDEO #57: ROTAS: CRIANDO UM MÓDULO DE ROTAS	90
60 VÍDEO #58: CRIANDO UM MÓDULO DE FUNCIONALIDADE	91
61 VÍDEO #59: ROTAS: CRIANDO UM MÓDULO DE FUNCIONALIDADE	92
62 VÍDEO #60: ROTAS FILHAS	93
63 VÍDEO #61: ROTAS FILHAS: DESENVOLVENDO AS TELAS	94
64 VÍDEO #62: ROTAS: DICA DE PERFORMANCE: CARREGAMENTO SOB DEMANA (<i>LAZY LOADING</i>)	94
65 VÍDEO #63: ROTAS: TELA DE <i>LOGIN</i> E COMO NÃO MOSTAR O MENU (<i>NAVBAR</i>).....	96
67 VÍDEO #64: USANDO GUARDA DE ROTAS <i>CANACTIVATE</i>	98
68 VÍDEO #65: USANDO GUARDA DE ROTAS <i>CANACTIVATECHILD</i>	99
69 VÍDEO #66: USANDO GUARDA DE ROTAS <i>CANDEACTIVATECHILD</i>	99
70 VÍDEO #67: USANDO GUARDA DE ROTAS: <i>CANDEACTIVATE</i> COM INTERFACE GENÉRICA.....	101
71 VÍDEO #68: USANDO GUARDA DE ROTAS: <i>RESOLVE</i> : CARREGANDO DADOS ANTES DA ROTA SER ATIVADA	102
72 VÍDEO #69: USANDO GUARDA DE ROTAS: <i>CANLOAD</i> : COMO NÃO CARREGAR MÓDULO SEM PERMISSÃO	103
73 VÍDEO #70: DEFININDO ROTA PADRÃO E WILDCARD (ROTA NÃO ENCONTRADA).....	104
Links úteis.....	105
74 VÍDEO #71: ROTAS: ESTILO DE URL: HTML 5 OU #.....	105

75 VÍDEO #72: FORMULÁRIOS (TEMPLATES VS DATA/REATIVO) INTRODUÇÃO.....	106
76 VÍDEO #73: FORMULÁRIOS - CRIANDO O PROJETO INICIAL COM BOOTSTRAP 3	106
77 VÍDEO #74: FORMS (TEMPLATE DRIVEN) CONTROLES NGFORM, NGSUBMIT E NGMODEL	108
80 VÍDEO #77: FORMS (TEMPLATE DRIVEN) APLICANDO VALIDAÇÃO NOS CAMPOS	111
Links úteis.....	111
81 VÍDEO #78: FORMS (TEMPLATE DRIVEN) APLICANDO CSS NA VALIDAÇÃO DOS CAMPOS	111
82 VÍDEO #79: FORMS (TEMPLATE DRIVEN) MOSTRANDO MENSAGENS DE ERROS DE VALIDAÇÃO	112
83 VÍDEO #80: FORMS (TEMPLATE DRIVEN) DESABILITANDO O BOTÃO DE SUBMIT PARA FORMULÁRIO	113
84 VÍDEO #81: FORMS (DICA): VERIFICANDO DADOS DO FORM NO TEMPLATE COM JSON	113
85 VÍDEO #82: FORMS (TEMPLATE DRIVEN) ADICIONANDO CAMPOS DE ENDEREÇO (FORM LAYOUT).....	114
86 VÍDEO #83: FORMS (TEMPLATE DRIVEN) REFATORANDO (SIMPLICANDO) CSS E MENSAGENS DE ERRO.....	115
87 VÍDEO #84: FORMS (TEMPLATE DRIVEN) FORM GRUPS (AGRUPANDO DADOS).....	116
88 VÍDEO #85: FORMS (TEMPLATE DRIVEN) PESQUISANDO ENDEREÇO AUTOMATICAMENTE COM CEP	117
Links úteis.....	118
89 VÍDEO #86: FORMS (TEMPLATE DRIVEN) POPULANDO CAMPOS COM SETVALUE E PATCHVALUE (CEP)	118
90 VÍDEO #87: FORMS (TEMPLATE DRIVEN) SUBMETENDO VALORES COM HTTP POST	119
Link útil	119
91 CONCLUSÃO	120

1 INTRODUÇÃO

Esse projeto é referente ao complemento de horas da disciplina de Sistemas Distribuídos, ministrada pelo professor Fernando D'Agostini. Ele tem como base, o curso de Angular 2 disponibilizado em vídeos por Loiane Groner em seu canal no Youtube.

Aqui serão apresentados os principais pontos abordados durante o curso, com alguns comentários explicativos no código e *links* externos que podem facilitar o entendimento de algumas questões. Atualmente o canal do *Youtube* possui 98 vídeos, mas o estudo foi feito apenas sobre os vídeos 01 ao 87; os demais vídeos não foram abordados nesse projeto.

O código de todos os projetos estão disponibilizados no *GitHub* e possuem um *link* para o vídeo estudado.

Repositório no *GitHub*: <https://github.com/nasouza2/Angular2B>.

Playlist do curso:

<https://www.youtube.com/playlist?list=PLGxZ4Rq3BOBoSRcKWEdQACbUCNWLczg2G>

2 DEFINIÇÃO

Esse projeto foi realizado utilizando o *Node.js* (versão 8.0.0) e o *Visual Studio Code* (versão 1.13.1), com os seguintes *plugins*: *Angular 2, 4 and up coming latest TypeScript HTML Snippets*; *Angular v4 TypeScript Snippets*; *Aton One Dark*; *Auto Import*; *HTML Snippets* e *vscode-icons*.

3 VÍDEO #01: INTRODUÇÃO AO ANGULAR 2

3.1 O que será estudado no curso?

Ao longo do curso, estudaremos os seguintes assuntos: componentes e *Templates*; *Data binding*; Diretivas; Serviços; Formulários; Roteamento; Interação com servidor e *CRUD* Mestre Detalhe.

3.2 O que é necessário saber para realizar o curso?

É necessário ter conhecimento das linguagens *HTML*, *CSS* e *JavaScript*. Não é necessário ter conhecimento de Angular JS 1.x.

3.3 O que é o Angular 2?

Angular 2 é um *framework* nascido da parceria da *Google* com a *Microsoft*; escrito em *TypeScript* e possui código *Open Source* disponível no *GitHub*. O Angular 2 não é continuação do Angular 1, pois foi reescrito para fazer melhor uso *HTML*.

Esse *framework* é totalmente orientado a Componente, ou seja, toda aplicação é um componente. Nós criaremos um componente raiz (também chamado de *Root*), que será o Pai ou a Mãe da nossa aplicação. Esse componente pode ser uma lista de contatos, de clientes, um cabeçalho; fica a cargo de o desenvolvedor declarar o que será o componente raiz. Um componente também pode ter outros componentes, assim podemos dividir a aplicação em partes menores, o que facilita os testes unitários a serem realizados pelo desenvolvedor.

O *framework* é dividido em Blocos Principais, que são: Componentes; Diretivas; Roteamento; Serviços; *Template*; *Metadata*; *Data Binding* e Injeção de Dependência.

Links úteis:

- 1) Site oficial do *framework*: <https://angular.io/>.
- 2) Repositório do Angular 2 no *GitHub*: <https://github.com/angular>.
- 3) Instalar e configurar o *GIT* no Windows: <http://gabsferreira.com/instalando-o-git-e-configurando-github>.

3.4: Visão geral sobre os blocos:

a) Componentes: O objetivo do componente é mostrar dados, então ele pode realizar integração com o *BackEnd*. Ele é responsável por todo o comportamento da *VIEW* (junção dos

Componentes, *Controller* e Escopo da aplicação). Nessa junção, ele é também responsável pelo o que o usuário vai ver, pois encapsula o *Template*, o *Metadata* e o *Data Binding*.

b) *Template*: *Layout* da tela como botões e formulários.

c) *Metadata*: Processamento dos metadados; esses metadados permitem que o *framework* ler as classes e fazer seu processamentos.

d) *Data Binding*: Associação dos componentes do nosso projeto + os componentes do *template*.

e) Serviço: Como boa prática, regras de negócios não são escritas no Componente; para isso nós utilizamos um Serviço que se comunicará com o *backend*, além de poder ser injetado em outras classes; esse processo recebe o nome de Injeção de Dependência.

f) Roteamento: Responsável pela navegação da aplicação, não só em relação a páginas, mas também a telas (como ir de uma tela a outra).

g) Diretiva: Responsável por modificar elementos *DOM* e/ou seu comportamento.

4 VÍDEO #02: AMBIENTE DE DESENVOLVIMENTO (NODE.JS, TYPESCRIPT, ANGULAR CLI)

Para iniciarmos o desenvolvimento da nossa aplicação, é necessário instalarmos a última versão o *Node.JS*, que está disponível no seguinte link: <https://nodejs.org/en/>. Para a instalação, não é necessária nenhuma configuração específica; basta ir clicando *Next>Next>Install*, até a instalação ser concluída. O *Node.js* já vem com um repositório exclusivo com as bibliotecas do *angular/cli*. Chamado de *NPM*, esse repositório possui todas as bibliotecas necessárias para o desenvolvimento da aplicação (todas as dependências do nosso projeto, serão baixadas desse repositório) e pode ser acessado pelo link: <https://www.npmjs.com/package/@angular/cli>.

Depois de instalar o *Node.js*, é necessário instalar o *TypeScript* através do *Prompt* de Comando. Siga os seguintes passos para realizar a instalação:

- Abra o *Prompt* de Comando (ou CMD) e digite o seguinte comando:
- `npm install -g typescript` (se seu SO for Windows) ou
- `sudo npm install -g typescript` (se seu SO for Linux ou MAC)

Através dessa instalação, o *TypeScript* será baixado do diretório NPM.

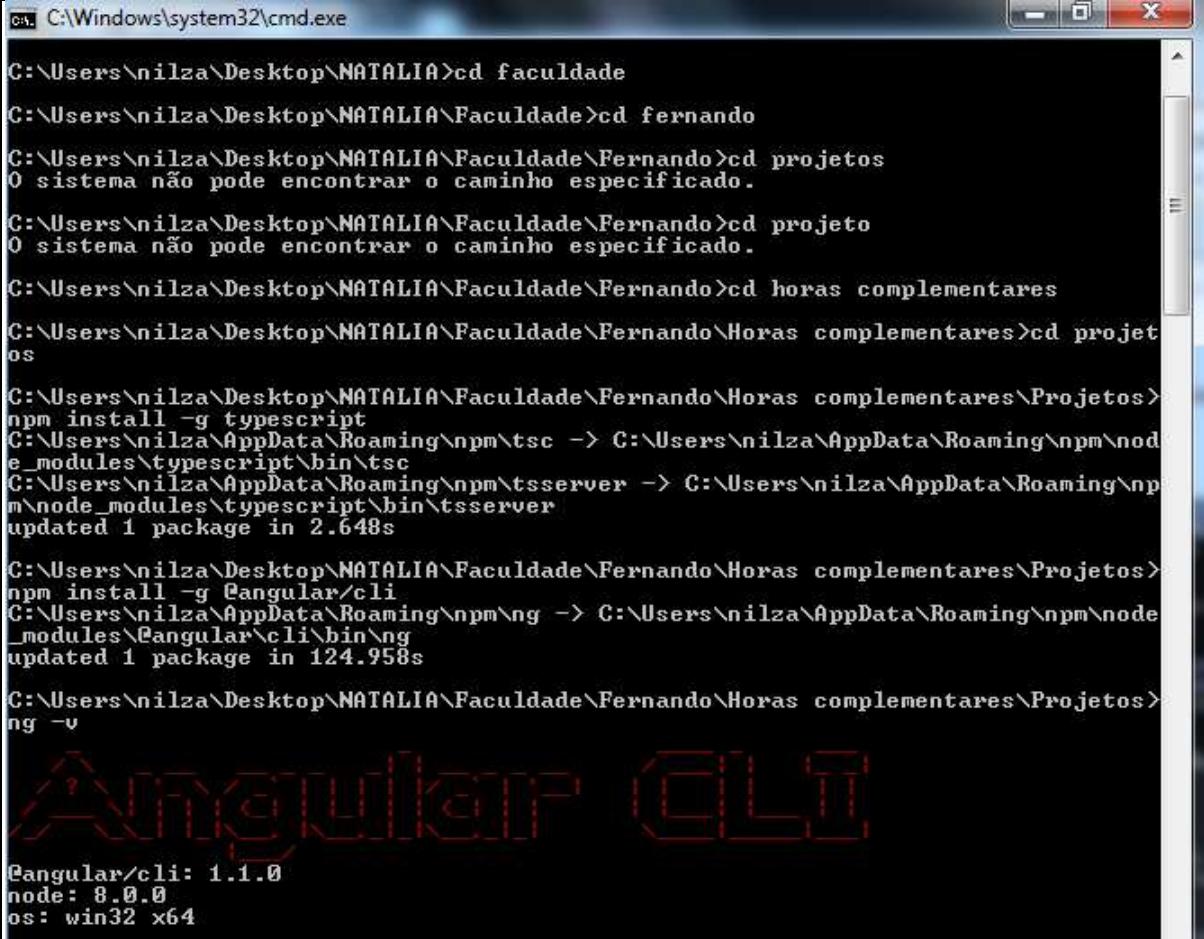
Ainda com o *Prompt* de Comando aberto, vamos instalar o angular/cli. Execute o seguinte comando:

- npm install -g @angular/cli (se seu SO for Windows) ou
- sudo npm install -g @angular/cli (se seu SO for Linux ou MAC)

Para verificarmos se nosso ambiente está tudo OK, depois que o angular/cli for instalado, execute o seguinte comando:

- ng -v

Se o ambiente estiver OK, será apresentada a tela abaixo com a versão do angular/cli e do Node.JS:



```
C:\Windows\system32\cmd.exe

C:\Users\nilza\Desktop\NATALIA>cd faculdade
C:\Users\nilza\Desktop\NATALIA\Faculdade>cd fernando
C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando>cd projetos
O sistema não pode encontrar o caminho especificado.
C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando>cd projeto
O sistema não pode encontrar o caminho especificado.
C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando>cd horas complementares
C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando\Horas complementares>cd projetos
C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando\Horas complementares\Projetos>
npm install -g typescript
C:\Users\nilza\AppData\Roaming\npm\tsc -> C:\Users\nilza\AppData\Roaming\npm\node_modules\typescript\bin\tsc
C:\Users\nilza\AppData\Roaming\npm\tsserver -> C:\Users\nilza\AppData\Roaming\npm\node_modules\typescript\bin\tsserver
updated 1 package in 2.648s
C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando\Horas complementares\Projetos>
npm install -g @angular/cli
C:\Users\nilza\AppData\Roaming\npm\ng -> C:\Users\nilza\AppData\Roaming\npm\node_modules\@angular\cli\bin\ng
updated 1 package in 124.958s
C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando\Horas complementares\Projetos>
ng -v

Angular CLI
@angular/cli: 1.1.0
node: 8.0.0
os: win32 x64
```

Para o desenvolvimento, precisamos também de um editor de texto. Abaixo serão listados quatro editores, que possuem um suporte melhor para o Angular 2. Você pode utilizar o de sua preferência. Todos eles estão disponíveis para Windows, Linux e Mac.

Como o *TypeScript* é mantido pela Microsoft, assim como o *Visual Studio Code*, há uma ótima interação entre eles; para o *ATOM*, *WebStorm* e *SublimeText*, há *plugins* disponíveis para o uso do *TypeScript* (apesar do *WebStorm* também suportar o *TypeScript*, há *plugins* disponíveis para ele):

- *Visual Studio Code*: Ferramenta gratuita que suporta o *TypeScript*. Pode ser baixado do seguinte link: <https://code.visualstudio.com/download>. Não é necessária nenhuma configuração específica para sua instalação.

- *ATOM*: Ferramenta gratuita. Para utilizar esse editor é necessário baixar *plugin* do *TypeScript*. Esse *plugin* está disponível para *download* no link: <https://atom.io/packages/atom-typescript>.

- *WebStorm*: Ferramenta paga. O editor também suporta o *TypeScript*, porém há *plugin* disponível para ele e que pode ser baixado do seguinte link: <https://www.jetbrains.com/webstorm/download/#section=windows>. Não é necessária nenhuma configuração específica para sua instalação.

- *Sublime Text*: Ferramenta gratuita, porém caso desejar, você pode adquirir a versão paga. Para utilizar o *TypeScript* nesse editor, é necessário baixar *plugin* que está disponível no link: <https://github.com/Microsoft/TypeScript-Sublime-Plugin>.

Para o desenvolvimento desse trabalho, foi utilizado o editor *Visual Studio Code*.

5 VÍDEO #03: HELLO, WORLD! CRIANDO PRIMEIRO PROJETO E O PRIMEIRO COMPONENTE

OBS: Projeto disponível em: github.com/nasouza2/Angular2B/tree/master/MeuPrimeiroProjeto.

5.1: Hello World! Criando meu primeiro projeto

Abra o *Prompt* de Comando e navegue até o diretório onde seu projeto será criado. Em seguida, digite o seguinte comando:

- `ng new NomeDoProjeto`

No final, o *angular/cli* vai criar toda a estrutura padrão do projeto e também vai instalar as dependências do *NPM*:

```

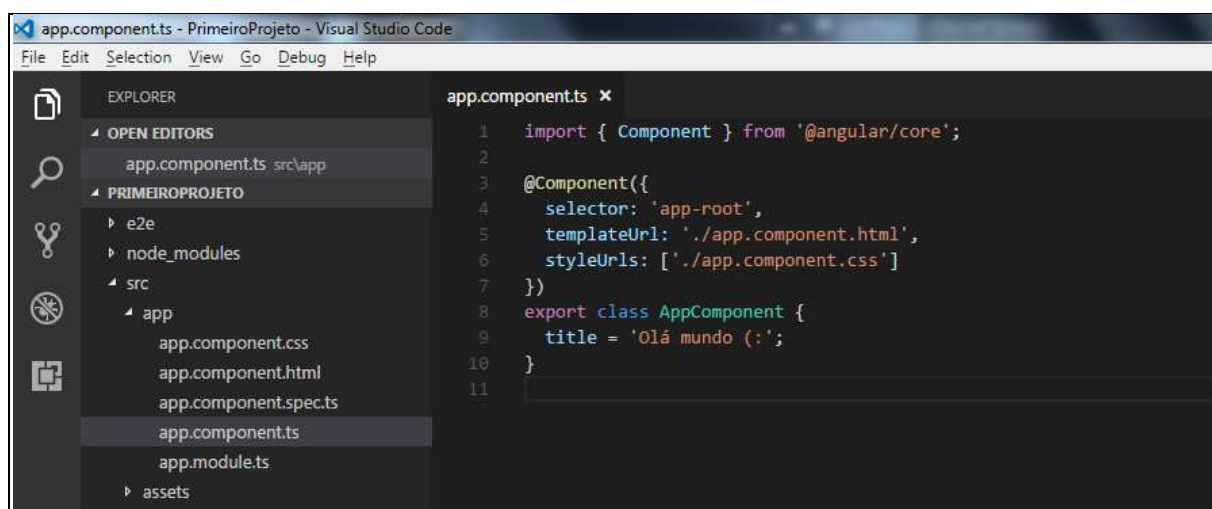
C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando\Horas complementares\Projetos
ng new MeuPrimeiroProjeto
installing ng
  create .editorconfig
  create README.md
  create src\app\app.component.css
  create src\app\app.component.html
  create src\app\app.component.spec.ts
  create src\app\app.component.ts
  create src\app\app.module.ts
  create src\assets\gitkeep
  create src\environments\environment.prod.ts
  create src\environments\environment.ts
  create src\favicon.ico
  create src\index.html
  create src\main.ts
  create src\polyfills.ts
  create src\styles.css
  create src\test.ts
  create src\ttsconfig.app.json
  create src\ttsconfig.spec.json
  create src\ttypings.d.ts
  create .angular-cli.json
  create e2e\app.e2e-spec.ts
  create e2e\app.po.ts
  create e2e\ttsconfig.e2e.json
  create .gitignore
  create karma.conf.js
  create package.json
  create protractor.conf.js
  create tsconfig.json
  create tslint.json
Installing packages for tooling via npm.
Installed packages for tooling via npm.
Project 'MeuPrimeiroProjeto' successfully created.

```

- Ainda com o Prompt de Comando aberto, acesse seu projeto e digite o comando: `ng serve`. Esse comando indica que nosso projeto vai ser servido ao navegador.

Abra o projeto com o editor de texto. Dentro do diretório `src - app` é onde os componentes serão criados.

Abra o arquivo `app.component.ts`, altere o título do seu projeto e salve. O próprio angular/cli faz o *build* novamente e atualiza a URL:



```

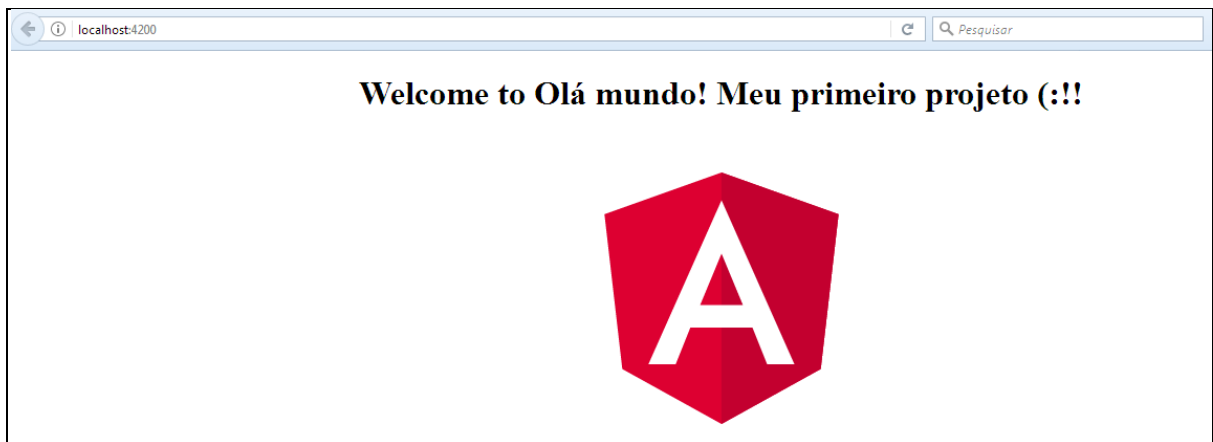
app.component.ts - PrimeiroProjeto - Visual Studio Code
File Edit Selection View Go Debug Help

EXPLORER
├─ OPEN EDITORS
│   └─ app.component.ts src/app
├─ PRIMEIROPROJETO
│   ├── e2e
│   ├── node_modules
│   └─ src
│       ├── app
│       │   ├── app.component.css
│       │   ├── app.component.html
│       │   ├── app.component.spec.ts
│       │   └─ app.component.ts
│       ├── app.module.ts
│       └─ assets
└─ 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 = 'Olá mundo (:';
10 }
11

```

Com o navegador de sua preferência, acesse <http://localhost:4200>. Note que seu projeto está funcionando corretamente:



5.2 Criando componente manualmente

Com seu projeto aberto, clique com botão direito no pacote app e clique em *New Folder*. Informe o nome do seu diretório, que por convenção, caso exista mais de uma letra, ele deve ser escrito com hífen (exemplo: meu-primeiro). Dentro desse diretório, clique com o botão direito, selecione a opção *New File* e informe o nome do seu componente com a extensão *.ts* (exemplo: meu-primeiro.component.ts).

No projeto, crie uma classe com um nome qualquer; toda classe deve ter a primeira letra de cada palavra em maiúscula. Depois de a classe ser criada, precisamos informar ao Angular que essa classe é um componente; para isso, usamos a anotação `@Component`. Nesse ponto, precisamos informar ao Angular, onde está o pacote com essa anotação, para que o *import* seja realizado. Isso deve ser feito através da linha: `import {Component} from '@angular/core'`.

Depois de o *import* ser realizado, deve ser informado os metadados do componente; isso é feito através de um *selector* dentro da anotação `@Component`. Depois é necessário criar um *template*. A imagem abaixo mostra como ficou a classe:

```
@Component({
  selector: 'meu-primeiro-component',
  template: `
    <p> Meu primeiro componente com Angular 2.</p>
  `
})
class MeuPrimeiroComponent{}
```

Após o componente ser criado, é necessário utilizá-lo em alguma classe. Copie o *selector* 'meu-primeiro-component' e a cole no arquivo *app.component.html*.

OBS: Para deixar o código mais limpo, foram removidos os links que são criados automaticamente pelo Angular ao criar o projeto.

```
<div style="text-align:center">
<h1>
  Bem vindo ao {{title}}!!
</h1>
</div>

<meu-primeiro-component></meu-primeiro-component>
```

Da forma que criamos a classe *MeuPrimeiroComponent*, ela é vista apenas internamente; para que ela possa ser vista por outras classes e assim evitar erro de compilação, é necessário informar "export" antes da classe:

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

@Component({
  selector: 'meu-primeiro-component',
  template: `
    <p> Meu primeiro componente com Angular 2.</p>
  `
})
export class MeuPrimeiroComponent {}
```

Só isso não basta que nosso projeto rode corretamente; todo *component*, serviço, diretiva criada, deve ser informada em um módulo. Abra o arquivo *app.modules.ts* e declare seu *component*, dentro das declarações do módulo e o importe (no *import*, não é necessário informar a extensão .js):

```
import { NavegadorModule } from '@angular/platform-navegador';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { MeuPrimeiroComponent } from './meu-primeiro/meu-primeiro.component';

@NgModule({
  declarations: [
    AppComponent,
    MeuPrimeiroComponent
```

```

    ],
    imports: [
      NavegadorModule
    ],
    providers: [],
    bootstrap: [AppComponent]
  })
  export class AppModule { }

```

Salve sua aplicação e abra novamente o navegador; note que seu projeto foi atualizado corretamente:



Dificuldade encontrada:

Atente-se para usar crase e não aspas simples no Template. Se usar aspas simples, ocorrerá erro de compilação. Atente-se também para extensão do seu componente, que deve ser “.ts” de *TypeScript*. Devido a um erro de digitação, a aplicação não conseguia encontrar o componente; todo o projeto teve que ser revisto, desde a instalação até a declaração do componente no módulo, para então perceber a grafia errada (“js” ao invés de “ts”).

5.3 Criando componente automaticamente

Anteriormente criamos um componente manualmente, apenas para entendermos o passo a passo de sua criação e o padrão de nomenclatura. Agora vamos ver como criar os componentes de forma automática.

No *Prompt* de Comando, execute o seguinte comando: `ng g c nome-do-componente` (g = gerar; c = componente. Você também pode escrever “component”).

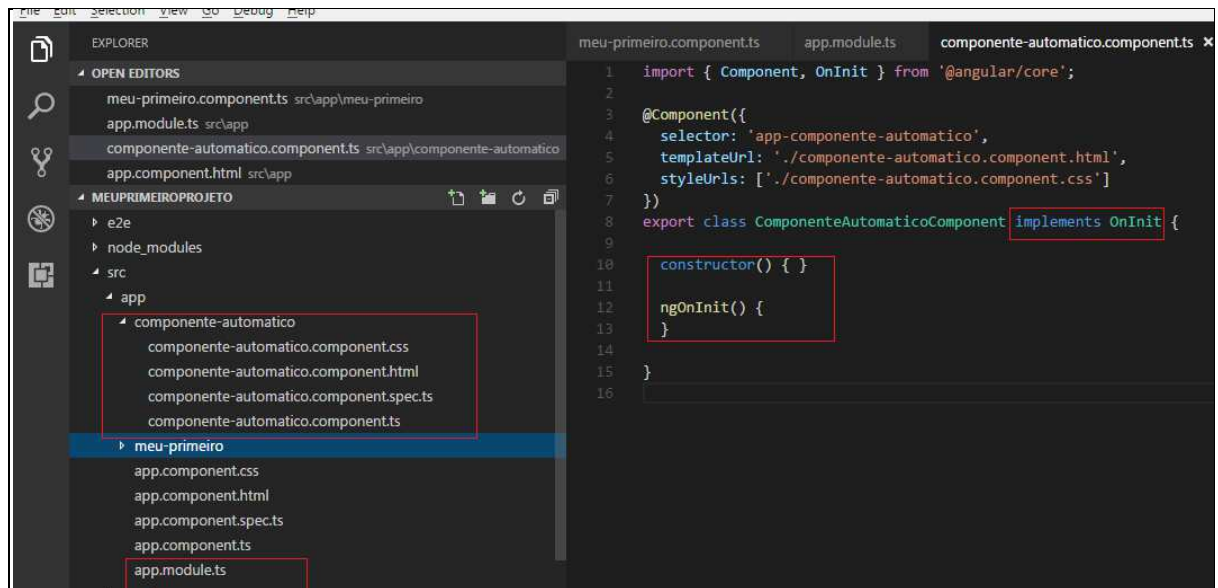
OBS: Se o *ng serve* estiver sendo executado, pare sua execução (através de Ctrl + C).

```

C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando\Horas complementares\Projetos\
MeuPrimeiroProjeto>ng g c componente-automatgico
installing component
  create src\app\componente-automatgico\componente-automatgico.component.css
  create src\app\componente-automatgico\componente-automatgico.component.html
  create src\app\componente-automatgico\componente-automatgico.component.spec.ts
  create src\app\componente-automatgico\componente-automatgico.component.ts
  update src\app\app.module.ts

```

Depois de ser executado, o angular/cli cria os arquivos automaticamente:



Em *componente-automático.component.ts*, remova os códigos destacados em vermelho, pois não utilizaremos ele nesse momento. Acesse o arquivo *app.module.ts*; note que foi criado o *component* foi importado corretamente:

```
import { NavegadorModule } from '@angular/platform-navegador';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { MeuPrimeiroComponent } from './meu-primeiro/meu-primeiro.component';
import { ComponenteAutomaticoComponent } from './componente-automático/componente-automático.component';

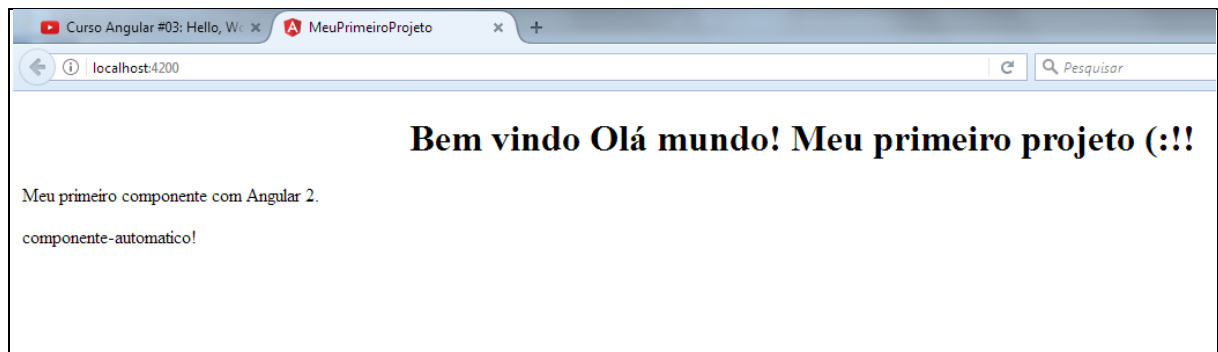
@NgModule({
  declarations: [
    AppComponent,
    MeuPrimeiroComponent,
    ComponenteAutomaticoComponent
  ],
  imports: [
    NavegadorModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Abra o arquivo *app.component.html* e informe o *selector* de seu *component*:

```
<div style="text-align:center">
<h1>
```

```
Bem vindo {{title}}!!  
</h1>  
</div>  
<meu-primeiro-component></meu-primeiro-component>  
<app-componente-automatico></app-componente-automatico>
```

No *Prompt* de Comando, execute novamente o *ng serve* e atualize sua *URL*; note que o projeto está sendo executado corretamente:



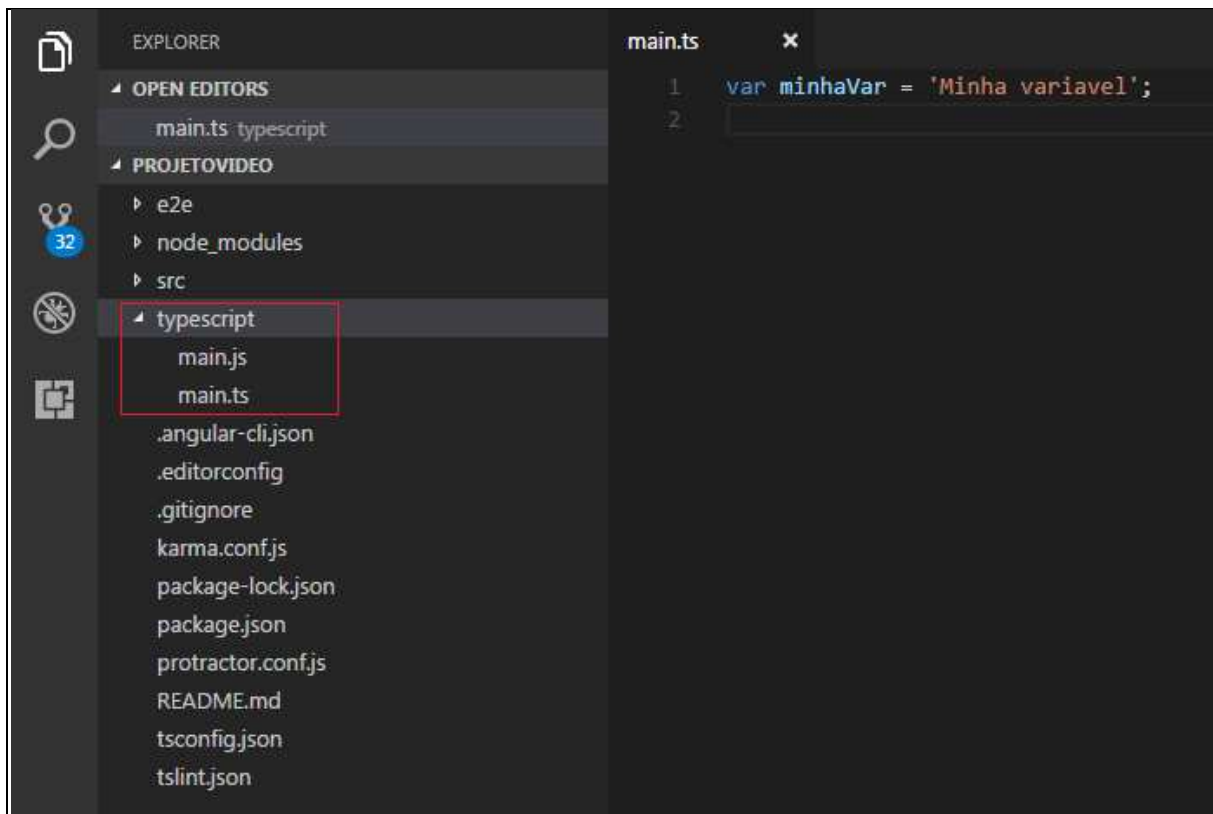
6 VÍDEO #04: INTRODUÇÃO AO TYPE SCRIPT PARA ANGULAR

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/ProjetoVideo04>.

Abra o projeto criado anteriormente e crie um novo diretório com qualquer nome e crie um arquivo com extensão *.ts* (exemplo *main.ts*). Nesse arquivo, declare uma variável qualquer, como exemplo abaixo:

```
varminhaVar = 'Minha variavel';
```

Abra o *Prompt* de Comando e execute o comando: *tsc main.ts*. Através desse comando, estamos pedindo para o *TypeScript* compilar nosso arquivo; depois dessa compilação, será criado um arquivo chamado *main.js*, conforme imagem abaixo:



Esse arquivo *main.js* é exatamente a mesma coisa que o arquivo *main.ts*. A diferença é que no arquivo *main.ts*, utilizamos *JavaScript* puro.

Na compilação, o *TypeScript* remove os espaços em branco; nosso arquivo *main.js* ficou a mesma coisa do *main.ts*:

```
varminhaVar = 'Minha variavel';  
functionminhaFuncao(x, y) {  
  returnx + y;  
}
```

No arquivo *main.ts*, crie uma função qualquer, conforme exemplo abaixo e compile novamente o projeto:

```
varminhaVar = 'Minha variavel';  
  
functionminhaFuncao(x, y){  
  returnx + y;  
}
```

Uma das alterações que teve na *ECMAScript* 2015 (que é a nova versão do *JavaScript*), é utilizar *“let”* ao invés de *“var”*:

```
varminhaVar = 'Minha variavel';

functionminhaFuncao(x, y){
returnx + y;
}

letnum = 2;
```

Nem todos os navegadores estão adaptados para utilizar o *ECMAScript* 2015 (também chamado de ES6), dessa forma nós precisamos de um *transpiler*, que vai compilar esse código em *JavaScript* puro. Depois que for compilado o código acima, no arquivo *main.js* ficará da seguinte forma:

```
varminhaVar = 'Minha variavel';
functionminhaFuncao(x, y) {
returnx + y;
}
varnum = 2;
```

Mais um exemplo com ES6:

```
varminhaVar = 'Minha variavel';

functionminhaFuncao(x, y){
returnx + y;
}

//ES6
letnum = 2;
constPI = 3.14;
```

O arquivo *main.js* ficou da seguinte forma:

```
varminhaVar = 'Minha variavel';
functionminhaFuncao(x, y) {
returnx + y;
}
varnum = 2;
varPI = 3.14;
```

Dificuldade encontrada:

Ao executar o comando `tsc main.ts`, ocorria o seguinte erro: "TS6053: arquivo 'main.ts' não encontrado". Para resolvê-lo, foi utilizado o comando “`npm install tsd tsd reinstall`” na pasta do projeto; depois foi acessada a pasta `typescript` e então o comando `tsc main.ts` foi novamente executado; dessa vez não apresentando mais erro de compilação (ajuda retirada do link abaixo:

[https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&prev=search&rurl=translate.google.com.br&sl=en&sp=nmt4&u=https://github.com/Microsoft/ngconf2015demo/issues/31&usg=ALkJrhjUGfonih-_ZyKmpv7hwDpki5gMIA)

[BR&prev=search&rurl=translate.google.com.br&sl=en&sp=nmt4&u=https://github.com/Microsoft/ngconf2015demo/issues/31&usg=ALkJrhjUGfonih-_ZyKmpv7hwDpki5gMIA](https://translate.googleusercontent.com/translate_c?depth=1&hl=pt-BR&prev=search&rurl=translate.google.com.br&sl=en&sp=nmt4&u=https://github.com/Microsoft/ngconf2015demo/issues/31&usg=ALkJrhjUGfonih-_ZyKmpv7hwDpki5gMIA)).

Links úteis:

- 1) Todas as alterações do ECMA2015 podem ser encontradas em: es6-features.org/#Constants.
- 2) *Transpiler* Babel:
babeljs.io/repl/#?babili=false&evaluate=true&lineWrap=false&presets=es2015%2Creact%2Cstage-2&targets=&navegadores=&builtIns=false&debug=false&code=

7 VÍDEO #05: MÓDULOS (ngMODULE)

Módulo é o arquivo que vai ajudar a organizar nossa aplicação, pois centraliza todos os arquivos num mesmo diretório. Abra o primeiro projeto criado e abra o arquivo `app.modules.ts`; abaixo segue a explicação:

```
// O NavegadorModule prepara a aplicação para ser usada na web
import { NavegadorModule } from '@angular/platform-navegador';
// Indicando da onde o NgModule está
import { NgModule } from '@angular/core';

// Import's da classe de nosso projeto
import { AppComponent } from './app.component';
import { MeuPrimeiroComponent } from './meu-primeiro/meu-primeiro.component';
import { ComponenteAutomaticoComponent } from './componente-automatico/componente-automatico.component';

// Declarando a classe
@NgModule({
  // Metadados
  // Dentro do declarations, nós listamos todos os componentes, diretivas e pipes que vamos utilizar no módulo
  declarations: [
    AppComponent,
    MeuPrimeiroComponent,
```

```

ComponenteAutomaticoComponent
  ],
  // Em imports, colocamos outros módulos que queremos utilizar nesse módulo
  imports: [
    NavegadorModule
  ],
  // Em providers, informados quais são os serviços que ficarão disponíveis para
  todos os componentes declarados no módulo,
  // como por exemplo, autenticação de login, rotas.
  providers: [],
  // O bootstrap é encontrado apenas no módulo raiz e indica qual componente
  deve ser iniciado ao executar a aplicação (componente
  // que vai servir de container da app ou seja, qual é o componente principal)
  bootstrap: [AppComponent]
})
export class AppModule { } //Declaração da classe

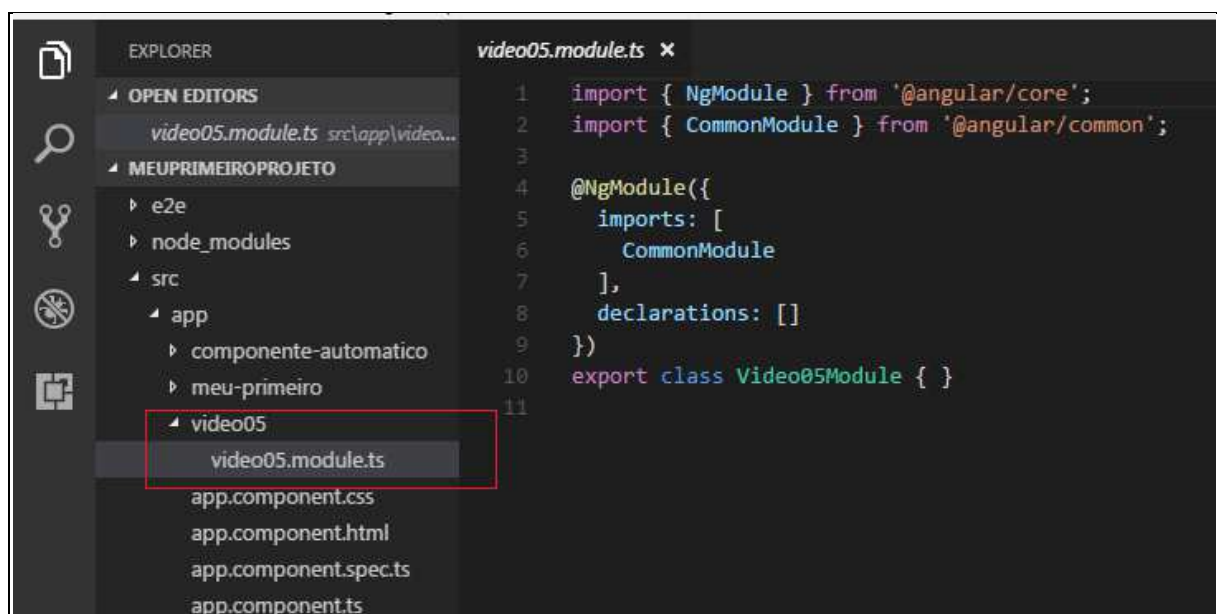
```

7.1 Criando módulo

OBS: Projeto disponível em:

github.com/nasouza2/Angular2B/tree/master/MeuPrimeiroProjeto/src/app/cursos.

Abra o *Prompt* de Comando até seu projeto e digite o comando: `ng g m nomedomódulo` (você pode utilizar apenas o “m” ou informar “module”). Depois de executado o comando, note que o módulo foi criado na sua árvore de arquivos:



8 VÍDEO #06: INTRODUÇÃO AOS TEMPLATES

Os componentes que nós criamos, sempre têm um código HTML, porque dentro dessa *classe* é onde vamos colocar nossa lógica de programação. Dentro do código *HTML*, é onde vamos construir o template que o usuário vai ver.

Abra o arquivo *cursos.componente.ts*, crie uma variável do tipo *String* e instancie essa variável.

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

@Component({
  selector: 'app-cursos',
  templateUrl: './cursos.component.html',
  styleUrls: ['./cursos.component.css']
})
export class CursosComponent implements OnInit {

  // Podemos instanciar a variável por aqui, informando o que ela vai receber ou
  nomePortal: string;
  // podemos iniciar ela através do construtor, usando a palavra "this"
  constructor() {
    this.nomePortal = 'http://loiane.training';
  }

  ngOnInit() {
  }
}
```

Para utilizar a variável, abra o arquivo *cursos.component.html* e entre duas chaves, informe a variável:

```
<p>
  Lista de cursos do portal {{ nomePortal }} <!-- Isso se chama interpolação-
->
</p>

<app-curso-detalle></app-curso-detalle>
```

No navegador, o projeto ficou assim:

Bem vindo Olá mundo! Meu primeiro projeto (:!!

Meu primeiro componente com Angular 2.

componente-automatico!

Lista de cursos do portal <http://loiane.training>

curso-detalle works!

Para fazer listagem manualmente, utilizamos a tag ``:

```
<!-- Para fazer listagem manualmente, geralmente utilizamos a tag <ul>-->
<ul>
<li> Java </li>
<li> Angular </li>
</ul>
```

Bem vindo Olá mundo! Meu primeiro projeto (:!!

Meu primeiro componente com Angular 2.

componente-automatico!

Lista de cursos do portal <http://loiane.training>

curso-detalle works!

- Java
- Angular

Porém os *templates* do Angular nos permitem fazer as listagens dinamicamente. Em *cursos.component.ts*, crie uma variável do tipo *array*:

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

@Component({
  selector: 'app-cursos',
  templateUrl: './cursos.component.html',
  styleUrls: ['./cursos.component.css']
})
export class CursosComponent implements OnInit {

  // Podemos instanciar a variável por aqui, informando o que ela vai receber ou
  nomePortal: string;
  cursos: string[] = ['Java', 'Angular', 'Ext JS']
  // podemos iniciar ela através do construtor, usando a palavra "this"
  constructor() {
    this.nomePortal = 'http://loiane.training';
  }
}
```

```
ngOnInit() {  
}
```

Para que cada posição ocupe corretamente uma ``, será utilizada a diretiva *NgFor*, onde é declarada uma variável local utilizando o *let* e depois o nosso *array*:

```
<!-- Lista criada pelo template-->  
<ul>  
<!-- Declaramos uma variável curso que vai receber a posição do array cursos -  
-->  
<li *ngFor="let curso of cursos">  
<!-- Depois é feita a interpolação para a saída da variável curso -->  
  {{ curso }}  
</li>  
</ul>
```

No *navegador*, o projeto ficou assim:

Bem vindo Olá mundo! Meu primeiro projeto (:!!

Meu primeiro componente com Angular 2.

componente-automático!

Lista de cursos do portal <http://loiane.training>

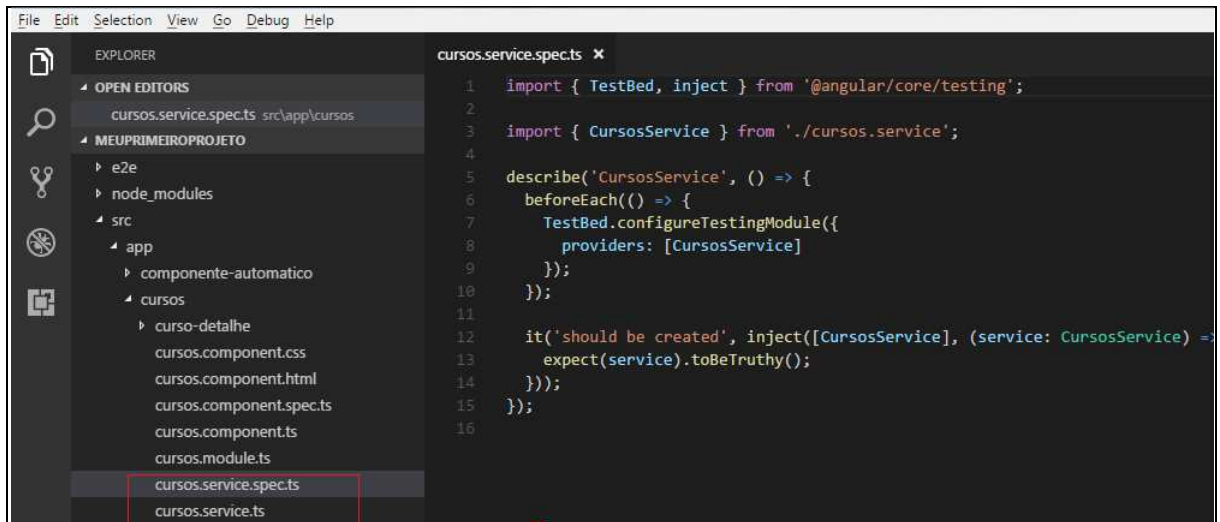
curso-detalle works!

- Java
- Angular
- Ext JS

9 VÍDEO #07: INTRODUÇÃO AOS SERVIÇOS E INJEÇÃO DE DEPENDÊNCIA (DI)

Como uma boa prática, colocamos na classe *Component*, apenas o código responsável por mostrar as informações para o usuário ou interagir com ele e numa classe *Service*, colocamos toda a lógica de comunicação.

Para criar um serviço, use o comando: `ng g s nomeservico` (se preferir pode usar “*service*” ao invés de apenas “*s*”). Após executar o comando, o Angular criará dois arquivos no nosso projeto:



No arquivo de *cursos.service.ts*, criamos o método que vai retornar o nosso *array*:

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

@Injectable()
export class CursosService {

  constructor() { }

  getCursos() {
    return ['Java ', 'Angular ', 'Ext JS '];
  }
}
```

No arquivo *cursos.component.ts*, deixamos o component apenas passando o resultado do nosso método. Precisamos importar, usar a classe *cursos.service* e criar variável que instância essa classe:

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

// Importação da classe
import { CursosService } from './cursos.service';

@Component({
  selector: 'app-cursos',
  templateUrl: './cursos.component.html',
  styleUrls: ['./cursos.component.css']
})
export class CursosComponent implements OnInit {

  nomePortal: string;
  cursos: string[];

  constructor() {
```



```
this.nomePortal = 'http://loiane.training';

// Criando a variável
varservico = newCursosService();
```

É nesse ponto que entra o conceito de Injeção de Dependência, que é fazer com que o Angular nos forneça uma instância da classe de serviço, assim não precisamos nos preocupar em ter que instanciá-la manualmente. O que indica que um serviço pode ser injetado é a anotação `@Injectable()` na classe de serviço:

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

@Injectable() // Anotação indicando que o serviço pode ser injetado
export class CursosService {

  constructor() { }

  getCursos(){
    return ['Java ', 'Angular ', 'Ext JS '];
  }
}
```

A injeção de dependência é dentro do construtor da classe de seu componente:

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

// Importação da classe
import { CursosService } from './cursos.service';

@Component({
  selector: 'app-cursos',
  templateUrl: './cursos.component.html',
  styleUrls: ['./cursos.component.css']
})
export class CursosComponent implements OnInit {

  nomePortal: string;
  cursos: string[]; // Deixamos o componente passando apenas o resultado
  // Declarando o serviço e adicionando o modificador de acesso (que pode ser
  // público ou privado)
  constructor(private cursosService: CursosService) {
    this.nomePortal = 'http://loiane.training';
    this.cursos = this.cursosService.getCursos();
  }

  // Criando a variável
```

```
//var servico = new CursosService(); como temos a classe serviço, a essa
variável por ser apagada.
}
```

Depois de instanciarmos o ser serviço, no arquivo *cursos.module.js* é necessário criar o *Providers* e importar o serviço:

```
import { NgModule } from '@angular/core';
// Módulo de funcionalidade não possui bootstrap
// A diferença de um módulo de funcionalidade e um de raiz, é o CommonModule
import { CommonModule } from '@angular/common';

import { CursosComponent } from './cursos.component';
import { CursoDetalheComponent } from './curso-detalle/curso-
detalle.component';
import { CursosService } from './cursos.service';

@NgModule({
  imports: [
    CommonModule
  ],
  declarations: [
    CursosComponent,
    CursoDetalheComponent
  ],
  exports: [
    CursosComponent
  ],
  providers: [ CursosService ]
})

export class CursosModule { }
```

Depois disso é só executar seu projeto e o resultado no navegador será esse:

Bem vindo Olá mundo! Meu primeiro projeto (:!!

Meu primeiro componente com Angular 2.

componente-automatico!

Lista de cursos do portal <http://loiane.training>

curso-detalle works!

- Java
- Angular
- Ext JS

10 VÍDEO #08: DICAS PLUGINS ANGULAR PARA ATOM E VS CODE

10.1 Dicas de plugins para o Atom

Para instalar os plugins, clique no menu Atom > *Preferences* (se você estiver usando Windows ou Linux, vá em *File > Settings*). Clique em *Install* e pesquise pelo pacote. Plugins sugeridos:

- *angular-2-typeScript-snippets*: Plugin que cria o esqueleto dos métodos.
- *atom-typescript*: plugin que é um compilador do *TypeScript*.
- *linter*: plugin responsável por sublinhar com a cor vermelha, as linhas onde ocorreram erros de compilação, facilitando assim a identificação das falhas.
- *file-icons*: Plugin que cria os ícones dos arquivos, facilitando a visualização dos mesmos.
- *open-recent*: Plugin que abre os projetos recentes e
- *pigments*: Plugin para ser usado com *.css*; ele preenche a cor de fundo do texto, com a cor passada no *.css*.

10.2 Dicas de plugins para o Visual Studio Code

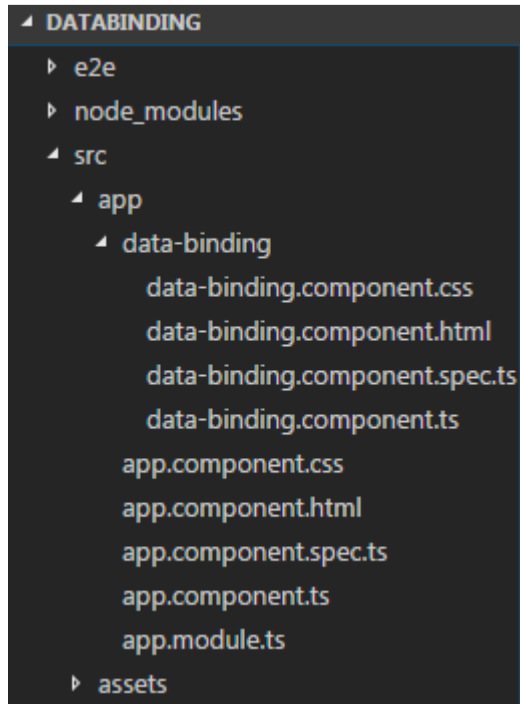
Para instalar os plugins, clique em *Extensions* no menu esquerdo, pesquise por "Angular2" e serão listados todos os plugins disponíveis para o Angular. Plugins sugeridos:

- *Angular 2, 4 and up coming latest TypeScript HTML Snippets*: Plugin que cria o esqueleto dos métodos
- *Aton One Dark*: Plugin para deixar o tema colorido.
- *Auto Import*: Plugin que realiza automaticamente a importação da classe.
- *HTML Snippets*: Plugin que cria o esqueleto dos métodos
- *Ionic 2 CommandswithSnippets* (para quem for trabalhar com *Ionic*).
- *vscode-icons*: Plugin que cria os ícones dos arquivos, facilitando a visualização dos mesmos.

11 VÍDEO #09: PROPERTYBINDING E INTERPOLAÇÃO

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/databinding>.

Data *Binding* significa associar informações do componente para o template ou vice-versa. Para esse capítulo, crie um novo projeto chamado *databinding* e crie um componente chamado *databinding*.



Abra o arquivo *data-binding.component.ts*, copie o *selector* e o cole no *app.component.html* (remova todo código criado automaticamente nesse arquivo):

```
<app-data-binding></app-data-binding>
```

No arquivo *databinding.component.html* é onde incluímos a interpolação ou a *PropertyBinding*:

```
<section class="property-binding">
<article>
<h3> Interpolation / Interpolação </h3>
<!-- O Angular consegue fazer interpolação de variável:-->
<p> String renderizada com Interpolação: {{ url }} </p>
<!-- Ele também consegue resolver expressões matemáticas: -->
<p> Resultado de 1 + 1: {{ 1 + 1 }} </p>
<!-- Chamar métodos e: -->
<p> Resultado de 1 + 1 não é: {{ 1 + 1 + getValor() }} </p>
<!-- Resolver expressões booleanas: -->
<p> Curso Angular e gostou do Curso {{ cursoAngular && getCurtirCurso () }}
</p>
</article>
<article>
```

```

<h3> Interpolação com imagem e Property Binding </h3>
<!--Exemplo de interpolação com imagem: -->
<imgsrc="{{urlImagem}}">
<!--Exemplo de interpolação com Property Binding: -->
<!-- Nessa situação, usa-se [] em volta do nome da propriedade; o Angular
consegue identificar uma Property Binding, não sendo necessário informar as
{}. Fica a critério do programador utilizar a interpolação ou a Property
Binding-->
<img [src]="urlImagem">
<p></p>
</article>
</section>

```

Lógica do arquivo *databinding.component.ts*:

```

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

@Component({
  selector: 'app-databinding',
  templateUrl: './databinding.component.html',
  styleUrls: ['./databinding.component.css']
})
export class DatabindingComponent implements OnInit {
  url: string = 'http://loiane.com'; // Variável
  cursoAngular: boolean = true; // Expressão booleana
  urlImagem = 'http://lorempixel.com/400/200/nature/'; // Imagem

  // Método
  getValor(){
    return 1;
  }
  // Método utilizado com a expressão booleana
  getCurtirCurso(){
    return true;
  }
  constructor() { }

  ngOnInit() {
  }
}

```

12 VÍDEO #10: CLASS E STYLE BINDING

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video10>.

O *Class* e o *Style Binding* também são classificados como *Property Binding*; a diferença entre eles, é que ao invés de utilizarmos variáveis e expressões no *Component* (como visto no último capítulo), é que utilizamos *.css*.

Para esse capítulo, vamos utilizar os componentes de alerta do *Bootstrap*, que é adicionado ao nosso projeto, através do comando: `npm install ngx-bootstrap bootstrap --save`. Após a instalação, no *package.json* estarão as dependências do projeto:

```
"private": true,
"dependencies": {
  "@angular/animations": "^4.0.0",
  "@angular/common": "^4.0.0",
  "@angular/compiler": "^4.0.0",
  "@angular/core": "^4.0.0",
  "@angular/forms": "^4.0.0",
  "@angular/http": "^4.0.0",
  "@angular/platform-browser": "^4.0.0",
  "@angular/platform-browser-dynamic": "^4.0.0",
  "@angular/router": "^4.0.0",
  "bootstrap": "^3.3.7",
  "core-js": "^2.4.1",
  "ngx-bootstrap": "^1.7.1",
  "rxjs": "^5.1.0",
  "zone.js": "^0.8.4"
```

Agora vamos configurar o *style*: Abra o *.angular-cli.json* e na parte de *styles*, adicione `"../node_modules/bootstrap/dist/css/bootstrap.min.css"`.

```
"index": "index.html",
"main": "main.ts",
"polyfills": "polyfills.ts",
"test": "test.ts",
"tsconfig": "tsconfig.app.json",
"testTsconfig": "tsconfig.spec.json",
"prefix": "app",
"styles": [
  "../node_modules/bootstrap/dist/css/bootstrap.min.css",
  "styles.css"
],
```

Vamos criar um *combobox* com as opções do bootstrap: Abra o *databinding.component.html* e inclua uma *<div>* da seguinte forma:

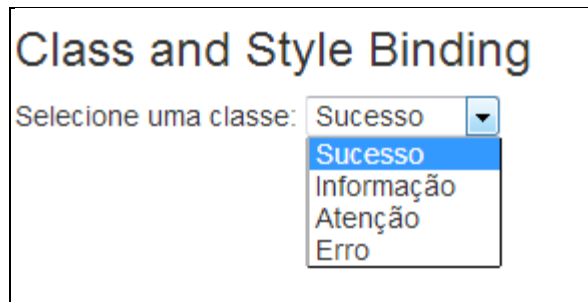
```
<section class="property-binding">
  <article>
    <h3> Class and Style Binding </h3>
    <div>
      Selecione uma classe:
      <select>
        <option value="alert-sucess">Sucesso</option>
```

```

        <option value="alert-info">Informação</option>
        <option value="alert-warning">Atenção</option>
        <option value="alert-danger">Erro</option>
    </select>
</div>
</article>
</section>

```

No navegador, o projeto ficou da seguinte forma:



Com o código dessa forma, não acontece nenhuma ação ao trocar de classe. Vamos então adicionar essa funcionalidade:

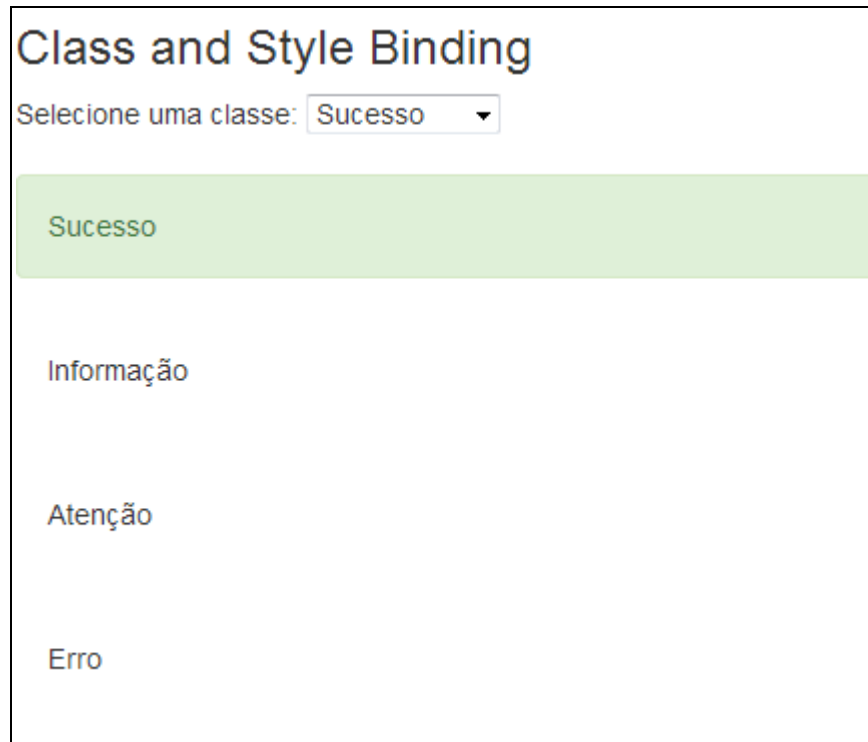
```

<section class="property-binding">
  <article>
    <h3> Class and Style Binding </h3>
    <div>
      Selecione uma classe:
      <!-- Primeiro é necessário criar uma variável local. Para isso use o
           #nomevariavel;
           Para ouvir a variável e saber quando houve uma mudança, use o "change";
           O "0" indica que nosso componente não executará nenhum método ao trocar
           de opção-->
      <select #classe (change) = "0">
        <option value="alert-success">Sucesso</option>
        <option value="alert-info">Informação</option>
        <option value="alert-warning">Atenção</option>
        <option value="alert-danger">Erro</option>
      </select>
      <br><br>

      <!--Adicionando as <div> para cada opção que pode ser selecionada;
           Para que seja apresentada apenas a div/cor correspondente a opção
           selecionada pelo usuário, será utilizado o class binding, que é definido
           entre []; Nesse class binding, estamos comparando se a opção selecionada no
           combobox, é o alert-success-->
      <div class="alert" role="alert"
        [class.alert-success]="classe.value == 'alert-success'">Sucesso</div>
      <div class="alert" role="alert">Informação</div>
      <div class="alert" role="alert">Atenção</div>
      <div class="alert" role="alert">Erro </div>
    </div>
  </article>
</section>

```

Se a opção selecionada for “Sucesso”, o fundo ficará na cor verde; as outras opções não possuem ação. Resultado no navegador:



Aplicando o *class binding* para as outras opções:

```
<div class="alert" role="alert"
  [class.alert-success]="classe.value == 'alert-success'">Sucesso</div>
<div class="alert" role="alert"
  [class.alert-info]="classe.value == 'alert-info'">Informação</div>
<div class="alert" role="alert"
  [class.alert-warning]="classe.value == 'alert-warning'">Atenção</div>
<div class="alert" role="alert"
  [class.alert-danger]="classe.value == 'alert-danger'">Erro </div>
</div>
```

No navegador, o resultado foi:

- Cor verde para Sucesso;
- Cor azul para Informação;
- Cor amarela para Atenção e
- Cor vermelha para Erro.

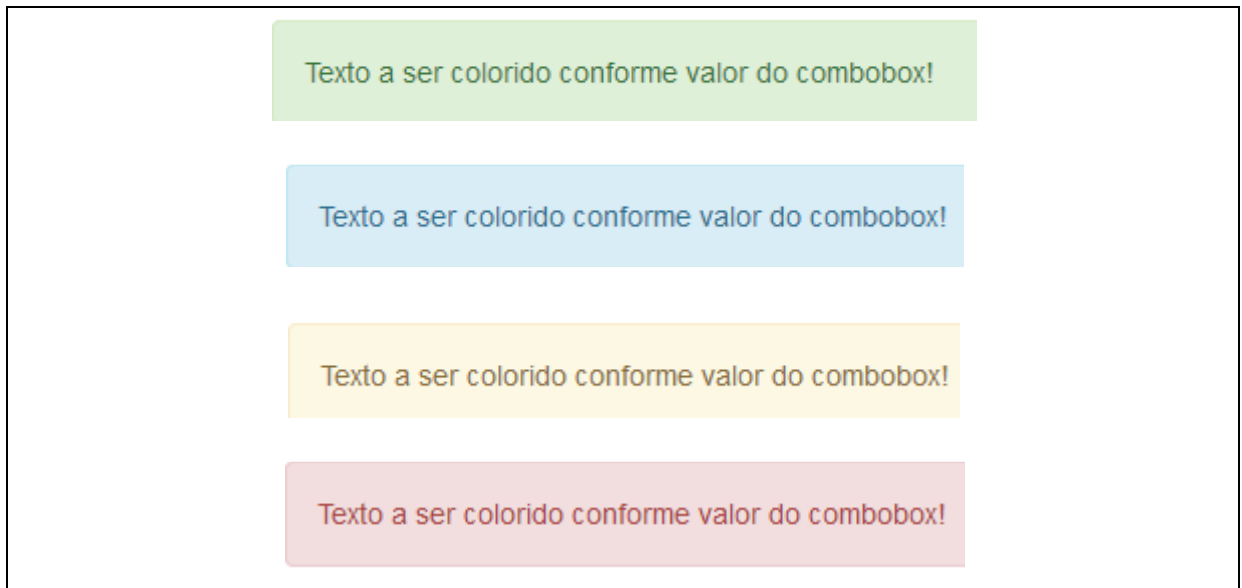


Também é possível fazer interpolação com o *Class Binding*: crie uma nova `<div>` com o `class="alert"`. Como o nome da classe do `.css` já é o valor do *combobox*, podemos pegar a `classe.value`:

```
</select>
<br><br>
<!-- Interpolação com Class Binding -->
<div class="alert" {{classe.value }}" role="alert"> Texto a ser colorido conforme
    valor do combobox!</div>
<div class="alert" role="alert"
    [class.alert-success]="classe.value == 'alert-success'">Sucesso</div>
<div class="alert" role="alert"
    [class.alert-info]="classe.value == 'alert-info'">Informação</div>
<div class="alert" role="alert"
    [class.alert-warning]="classe.value == 'alert-warning'">Atenção</div>
<div class="alert" role="alert"
    [class.alert-danger]="classe.value == 'alert-danger'">Erro </div>
```

No navegador, o resultado é esse:

- Cor verde para Sucesso;
- Cor azul para Informação;
- Cor amarela para Atenção e
- Cor vermelha para Erro.



Exemplo de *Style Binding*:

```
<div class="alert" role="alert"
[class.alert-danger]="classe.value == 'alert-danger'">Erro </div>

<!-- Exemplo de Style Binding
Aqui estamos comparando se a opção selecionada é do tipo "alert-danger";
se for, então executa o estilo "block"; senão executa o style "none"
block = mostrar nome = esconder-->
<div class="alert alert-danger" role="alert"
[style.display]="classe.value == 'alert-danger' ? 'block' : 'none'">
Esse texto só vai aparecer em caso de erro! </div>
</div>
```

No navegador, o resultado é esse:

- Qualquer opção diferente de Erro, não apresenta o texto:

Class and Style Binding

Selecione uma classe: Sucesso ▾

Texto a ser colorido conforme valor do combobox!

Sucesso

Informação

Atenção

Erro

- Se a opção for igual a Erro, então apresenta:

Class and Style Binding

Selecione uma classe:

Texto a ser colorido conforme valor do combobox!

Sucesso

Informação

Atenção

Erro

Esse texto só vai aparecer em caso de erro!

Dificuldade encontrada:

No vídeo 10, o comando para instalar o bootstrap era "ng2-bootstrap --save" porém a página do *bootstrap* foi alterada e agora o comando é "npm install ngx-bootstrap bootstrap --save"

Link útil:

- 1) Bibliotecas do *bootstrap* para o Angular 2 e Angular 4: <http://valor-software.com/ngx-bootstrap/#/>

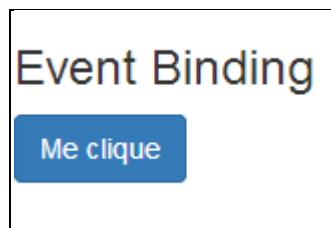
13 VÍDEO #11: EVENT BINDING

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video11>.

O *Event Binding* é a forma que escutamos o nosso component e executamos uma ação com ele. No *databinding.component.html*, crie o seguinte código:

```
<section class="event-binding">
  <article>
    <h3> Event Binding </h3>
    <div>
      <button class="btn btn-primary"> Me clique</button>
    </div>
  </article>
</section>
```

No navegador, o resultado será esse:



Se você clicar no botão, não haverá nenhuma ação, porque não implementamos nenhum método para ele.

Implementando um event com método:

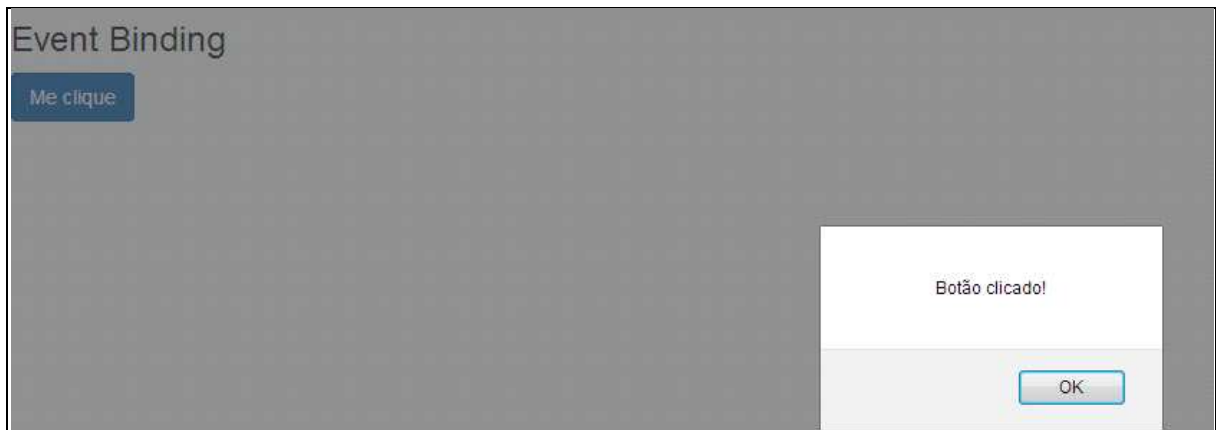
```
<section class="event-binding">

<article>
  <h3> Event Binding </h3>
  <div>
    <!-- O evento deve ser passado entre ();
        Podemos também chamar um método, como é o caso de "BotaoClicado"
    -->
    <button (click)="BotaoClicado()"
      class="btn btn-primary"> Me clique</button>
  </div>
</article>
</section>
```

No *databinding.component.ts*, criamos o método *BotaoClicado*, que exibe uma alerta ao ser clicado:

```
BotaoClicado(){
  alert('Botão clicado!');
}
```

No navegador, o resultado é:



Link útil:

- 1) Todos os eventos que são possíveis de serem feitos: <https://developer.mozilla.org/en-US/docs/Web/Events>.

14 VÍDEO #12: TWO-WAY DATA BINDING

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video12>.

Two-Way data binding deve ser usado quando queremos atualizar o template e o componente ao mesmo tempo.

Como lembrar da sintaxe correta? A sintaxe correta é chamada de Sintaxe de Banana, ou seja, primeiro as chaves simbolizando a caixa e depois os parênteses, simbolizando a banana: `[(())]`.

```
<section>
  <article class="two-way data binding">
    <h3> Two way data binding </h3>
    <div>
      <!--Primeiro exemplo:
      [value] associação de propriedade
      (input) associação de evento-->
      <input type="text"
      [value]="nome"
      (input)="nome = $event.target.value"
      />

      <!--Segundo exemplo:
      Quando utilizamos o ngModelChange, não é necessário usar o target.value-->
      <input type="text"
      [ngModel]="nome"
      (ngModelChange)="nome = $event" />
```

```

<!--Fazemos o two-way informando no evento, o binding de propriedade [] + o binding de
evento ()-->
    <input type="text" [(ngModel)]="nome">
    <p>Você digitou : {{nome}} </p>
</div>
</article>
</section>

```

Exemplo de *two-way* com objeto:

Em *databinding.component.ts*, foi criado o objeto *peessoa*:

```

peessoa: any = {
  nome: 'def',
  idade: 20
}

```

Em *databinding.component.html*, utilizamos o objeto + atributo e para apresentar os dados, usamos a interpolação:

```

<div>
  <p>Meu nome é {{peessoa.nome}} e tenho {{peessoa.idade}} anos de idade!</p>
  <input type="text" [(ngModel)]="peessoa.nome" > <!--Pegamos o objeto e o atributo -->
  <input type="text" [(ngModel)]="peessoa.idade">
</div>

```

15 VÍDEO #13: REUSANDO COMPONENTES *INPUT PROPERTIES*

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video13>.

O objetivo de usarmos *input properties*, é de deixar o código mais limpo e mais organizado.

Exemplo: No nosso projeto, foi criado um componente chamado *input-component*:

```

└─ input-property
   ├── input-property.component.css
   ├── input-property.component.html
   ├── input-property.component.spe...
   └── input-property.component.ts

```

Em *databinding.component.ts*, criamos uma variável do tipo *string*:

```

nomeDoCurso: string = 'Curso de Angular';

```

Em *databinding.component.html*, usamos a variável criada:

```
<section class="input-output-properties">
  <article>
    <h3> Input/Output Properties</h3>
    <app-curso [nome]="nomeDoCurso"> </app-curso>
  </article>
</section>
```

Em *input-property.component.ts*, usamos o *Input Properties* da seguinte forma:

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

@Component({
  selector: 'app-curso',
  templateUrl: './input-property.component.html',
  styleUrls: ['./input-property.component.css']
})
export class InputPropertyComponent implements OnInit {
  // Primeira forma de usar o input properties:
  // Através do @Input nós conseguimos expor uma propriedade chamada "nome" para o seletor
  // app-curso
  // @Input() nome: string = '';

  // Também podemos passar o properties como parâmetro, usando ela internamente como uma
  // variável normal (nomeCurso) e externamente com a variável de exposição
  @Input('nome') nomeCurso: string = '';

  constructor() { }

  ngOnInit() {
  }
}
```

Em *input-property.component.html*, usamos a interpolação com a variável criada:

```
<p>
  {{nomeCurso }}
</p>
```

No navegador, o resultado é esse:

Input/Output Properties

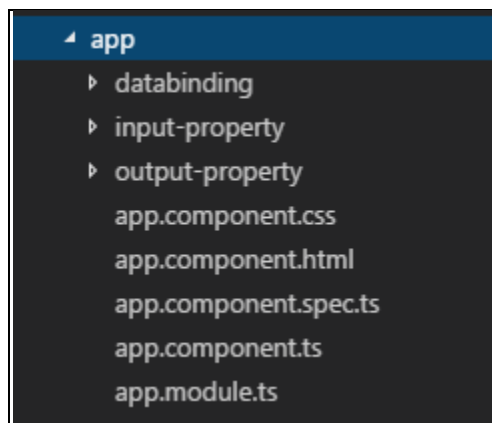
Curso de Angular

16 VÍDEO #14: EMITINDO EVENTOS COM *OUTPUT PROPERTIES*

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video14>.

O objetivo desse capítulo é disparar um evento chamado “mudou”, toda vez que o usuário clicar nos botões “+” ou “-”.

Utilizando o projeto do capítulo anterior, criamos um novo component chamado de *output-property*:



Em *output-property.components.ts*, renomeie o *selector* para “contador” e em *databinding.component.html*, passe esse *selector*:

- *output-property.components.ts*:

```
import { Component, OnInit } from '@angular/core';
@Component({
  selector: 'contador',
  templateUrl: './output-property.component.html',
  styleUrls: ['./output-property.component.css']
})
export class OutputPropertyComponent implements OnInit {
  constructor() { }
  ngOnInit() {
  }
}
```

- *databinding.component.html*

```
<section class="input-output-properties">
  <article>
    <h3> Input/Output Properties</h3>
```

```

    <app-curso [nome]="nomeDoCurso"> </app-curso>
    <contador> </contador>
  </article>
</section>

```

Vamos trabalhar em cima do *output-property.component.html*:

- Criamos dois botões e um *input*, com a variável “valor” iniciando em “0”:

```

<div>
  <button class="btn-primary">+ </button>
  <input type="text" [value]="valor" readonly> <!--Usa-se o readonly para não deixar o
                                                usuário alterar o campo-->
  <button class="btn-primary">- </button>
</div>

```

No navegador, o resultado é esse:

- Ao clicar nos botões não acontecerá nada, porque não existe ação para eles ainda.



Em *output-property.components.html*, vamos ouvir o clique do mouse e chamar os métodos incrementa e decrementa:

```

<div>
  <button class="btn-primary" (click)="incrementa ()">+ </button>
  <input type="text" [value]="valor" readonly>
  <button class="btn-primary" (click)="decrementa ()">- </button>
</div>

```

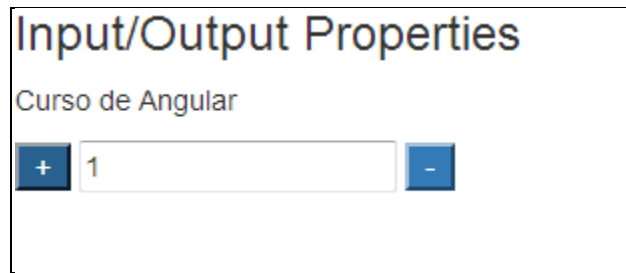
Metódos criados em *output-property.components.ts*:

```

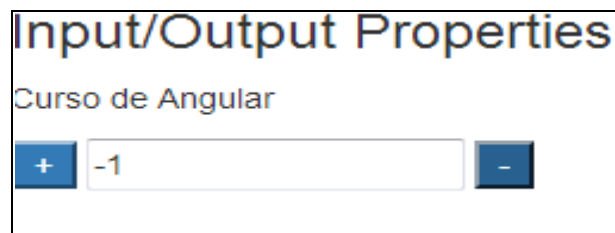
incrementa(){
  this.valor++;
}
decrementa(){
  this.valor--;
}

```

No navegador, o resultado é esse para o botão “+”:



E esse para o botão “-”:



17 VÍDEO #15: CICLO DE VIDA DE UM COMPONENTE

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/video15>.

Os principais eventos do ciclo de vida de um *component* são:

- a) *ngOnChanges*: Indica quando o *component* é atualizado;
- b) *ngOnInit*: Indica quando o *component* é inicializado;
- c) *ngDoCheck*: Verifica as mudanças do *component* a cada ciclo;
- d) *ngAfterContentInit*: Usado para inserir conteúdo externo na *view*;
- e) *ngAfterContentChecked*: Verifica os conteúdos inseridos;
- f) *ngAfterViewChecked*: Verifica os conteúdos e os conteúdos filhos e
- g) *ngOnDestroy*: Quando o *component* é destruído

Foi criado um *component* chamado “ciclo” e nele foram implementados todos esses eventos:

```
import { Component, OnInit, OnChanges, DoCheck, AfterContentInit, AfterContentChecked,
OnDestroy, SimpleChanges, Input } from '@angular/core';

@Component({
  selector: 'app-ciclo',
  templateUrl: './ciclo.component.html',
  styleUrls: ['./ciclo.component.css']
})
//É uma boa prática de programação, implementar todas as interfaces dos eventos do ciclo
de vida
```

```

export class CicloComponent implements OnInit, DoCheck, AfterContentInit,
AfterContentChecked, OnDestroy, OnChanges {

  @Input() valorInicial: number = 10;

  constructor() {
    this.log('constructor');
  }

  ngOnChanges(){
    this.log('ngOnChanges');
  }

  ngOnInit() {
    this.log('ngOnInit');
  }

  ngDoCheck(){
    this.log('ngDoCheak');
  }

  ngAfterContentInit(){
    this.log('ngAfterContentInit');
  }

  ngAfterContentChecked(){
    this.log('ngAfterContentChecked');
  }

  ngAfterViewInit(){
    this.log('ngAfterViewInit');
  }

  ngAfterViewChecked(){
    this.log('ngAfterViewChecked');
  }

  ngOnDestroy(){
    this.log('ngOnDestroy');
  }

  private log(hook: string){
    console.log(hook)
  }
}

```

- Em *app.component.html* criamos um botão para mudar o valor da variável:

```

<app-ciclo [valorInicial]="valor"> </app-ciclo>
<button (click)="mudarValor()">Mudar valor</button>

```

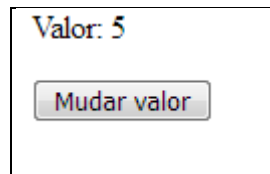
Método para mudar o valor da variável (*app.component.ts*):

```

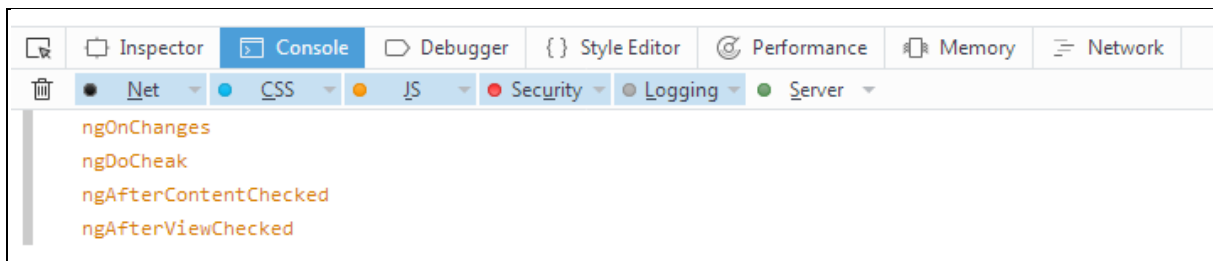
mudarValor(){
  this.valor++;
}

```

No navegador, o resultado é esse:



Clique no botão Mudar valor; note que no *console* foram os seguintes eventos foram disparados:



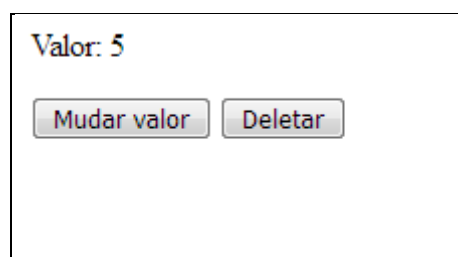
- Evento *ngOnDestroy*:

```
// Criada uma variável booleana com tipo False:
deletarCiclo: boolean = false;
// Método para destruir o ciclo; passa a variável como True:
destruirCiclo(){
  this.deletarCiclo = true;
}
}
```

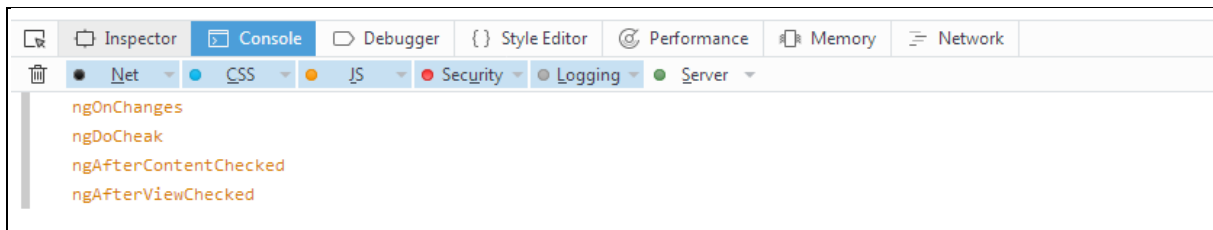
- Em *app.component.html*:

```
<!--Se a opção escolhida for diferente de deletarCiclo, então mostra os
eventos; senão, mostra o ngOnDestroy-->
<app-ciclo [valorInicial]="valor" *ngIf="!deletarCiclo"> </app-ciclo>
<button (click)="mudarValor()">Mudar valor</button>
<button (click)="destruirCiclo()">Deletar</button>
```

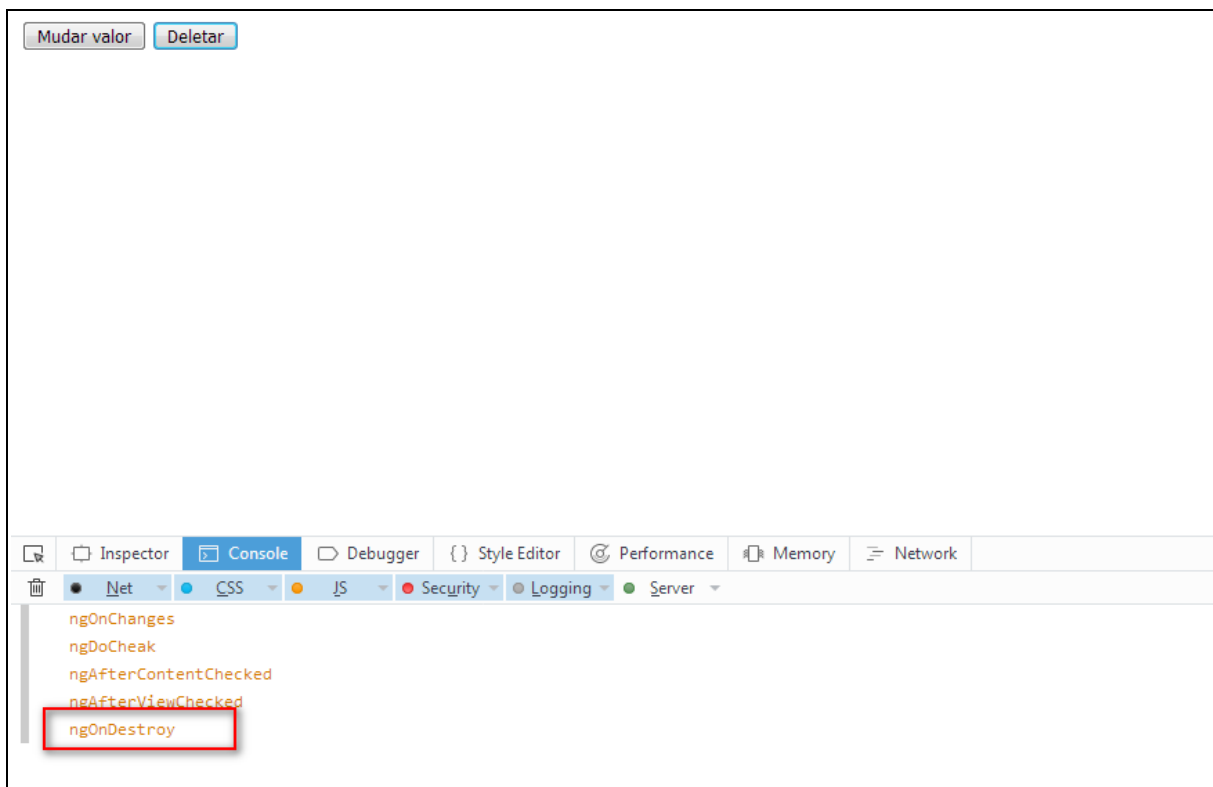
No navegador, o resultado é esse:



Ao clicar em Mudar valor, os seguintes eventos disparados:



Ao clicar em Deletar, note que o *input* é destruído e o evento *ngOnDestroy* é disparado:



18 VÍDEO #16: ACESSO AO DOM E AO TEMPLATE COM VIEWCHILD

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video16>.

Nesse capítulo vamos aprender a como acessar uma variável através do *input*, sem a necessidade de termos *ngModel* ou o *Value*.

- Em *output-property.component.html* criamos uma variável local chamada *#campoInput*:

```
<div>
  <button class="btn-primary" (click)="incrementa ()">+ </button>
  <input type="text" [value]="valor" readonly #campoInput> <!--Usa-se o readonly para não
  deixar o usuário alterar o campo-->
```

```
<button class="btn-primary" (click)="decrementa ()">- </button>
</div>
```

- E em `output-property.component.ts` utilizamos o `@ViewChild` para referenciar essa variável:

```
import { Component, OnInit, Input, Output, ViewChild, ElementRef } from '@angular/core';
import { EventEmitter } from "events";

@Component({
  selector: 'contador',
  templateUrl: './output-property.component.html',
  styleUrls: ['./output-property.component.css']
})
export class OutputPropertyComponent implements OnInit {

  @Input() valor: number = 0;
  @Output() mudouValor = new EventEmitter();

  // No ViewChild passamos qual é a variável no template que está associada ao component
  @ViewChild('campoInput') campoValueInput: ElementRef;

  incrementa(){
    // Ao invés de utilizarmos o this.valor++, utilizamos o value++ que referencia a
    // variável direto pelo component.
    console.log(this.campoValueInput.nativeElement.value++);
    this.mudouValor.emit({novoValor: this.valor});
  }
  decrementa(){
    console.log(this.campoValueInput.nativeElement.value--);
    this.mudouValor.emit({novoValor: this.valor});
  }
  constructor() { }
  ngOnInit() {
  }
}
```

19 VÍDEO #17: ANGULAR CLI: INSTALAÇÃO E CRIAÇÃO DE PROJETOS: NG NEW E NG SERVE

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/video17>.

Para instalar o Angular/Cli, é necessário que a versão no *Node* seja igual ou superior a 4. Com o *Prompt* de Comando aberto, instale o angular/cli, através do seguinte comando:

- `npm install -g @angular/cli` (se seu SO for Windows) ou
- `sudo npm install -g @angular/cli` (se seu SO for Linux ou MAC)

Ainda com o *Prompt* de Comando, navegue até o diretório onde seu projeto será criado. Em seguida, digite o seguinte comando:

- `ng new NomeDoProjeto`

No final, o angular/cli vai criar toda a estrutura padrão do projeto e também vai instalar as dependências do *NPM*:

```
C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando\Horascomplementares\ProjetosB>ng new video17
installing ng
create .editorconfig
create README.md
create src\app\app.component.css
create src\app\app.component.html
create src\app\app.component.spec.ts
create src\app\app.component.ts
create src\app\app.module.ts
create src\assets\gitkeep
create src\environments\environment.prod.ts
create src\environments\environment.ts
create src\favicon.ico
create src\index.html
create src\main.ts
create src\polyfills.ts
create src\styles.css
create src\test.ts
create src\tscconfig.app.json
create src\tscconfig.spec.json
create src\typings.d.ts
create .angular-cli.json
create e2e\app.e2e-spec.ts
create e2e\app.po.ts
create e2e\tscconfig.e2e.json
create .gitignore
create karma.conf.js
create package.json
create protractor.conf.js
create tsconfig.json
create tslint.json
Directory is already under version control. Skipping initialization of git.
Installing packages for tooling via npm.
Installed packages for tooling via npm.
Project 'video17' successfully created.
```

- Ainda com o *Prompt* de Comando aberto, acesse o diretório de seu projeto e digite o comando: `ng serve`. Esse comando indica que nosso projeto vai ser servido ao *navegador*.

- Nessa execução o `ng serve` apresenta qual será a porta utilizada pelo nosso projeto:

```
C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando\Horascomplementares\ProjetosB\video17>ng serve
** NG Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200 **
Hash: 2d8e8aa25882a7ab138d
Time: 26795ms
chunk {0} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 160 kB [initial] [rendered]
chunk {1} main.bundle.js, main.bundle.js.map (main) 5.36 kB [initial] [rendered]
chunk {2} styles.bundle.js, styles.bundle.js.map (styles) 10.5 kB [initial] [rendered]
chunk {3} vendor.bundle.js, vendor.bundle.js.map (vendor) 2.18 MB [initial] [rendered]
chunk {4} inline.bundle.js, inline.bundle.js.map (inline) 0 bytes [entry] [rendered]
webpack: Compiled successfully.
```

Com o navegador de sua preferência, acesse `http://localhost:4200`; note que seu projeto está funcionando:

Welcome to app!!

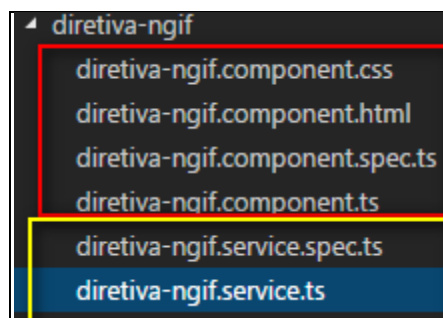


20 VÍDEO #18: CRIANDO COMPONENTS, SERVICES: NG GENERATE

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video18>.

O comando para criar um *component* é: `ng g c nomeDoComponente`. Ao invés de usar apenas “g”, você pode usar o “generate”: `ng generate c nomedoComponente`. Para criarmos um *service*, é utilizado o comando `ng g service nomeDoServiço`. Geralmente o nome do serviço é o mesmo nome do *component*.

Exemplo de *component* e *service*:



21 VÍDEO #19: ANGULAR CLI: USANDO PRÉ-PROCESSADORES (SASS, LESS, STYLUS)

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video19>.

Existem duas formas de serem definidos os processadores de `.css` no nosso projeto: A primeira é definindo o processador durante a criação do projeto, utilizando o comando “`—style=nomedoProcessador`” e a segunda é alterando a extensão de `.css` padrão do projeto, através do comando: `ng set defaults.styleExt nomedoProcessador`.

Primeira forma:

```
ng new nomeDoProjeto --style=sass
```

```
ng new nomeDoProjeto --style=less
```

```
ng new nomeDoProjeto --style=stylus
```

Segunda forma:

```
ng set defaults.styleExt scss
```

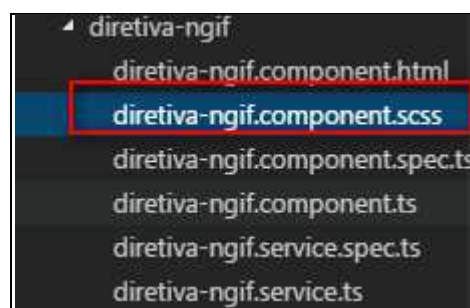
```
ng set defaults.styleExt less
```

```
ng set defaults.styleExt syly
```

A alteração do estilo vale apenas para novos *components*; o estilo dos *componentes* já existentes não são alterados:



Para alterar o estilo de um component que já existe, é necessário alterá-lo manualmente; clique com o botão direito no arquivo – opção *Rename*; renomeie o estilo do *component*:



Três ferramentas que o Angular CLI nos oferece:

- *ngLint* que scanea o código e verifica pontos que podem causar erros de compilação e verifica se o código está de acordo com o *Style Guide*;
- *ngTest* ferramenta que executa testes com Jasmine e
- *ngE2E* que também executa testes, mas com *Protractor*.

Exemplo de uso do *ng Lint*: Foi criada uma variável com um espaço entre seu nome e sua definição (o que não é uma boa prática). Ao executar o *ng Lint*, temos o resultado:

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

@Component({
  selector: 'app-diretiva-ngif',
  templateUrl: './diretiva-ngif.component.html',
  styleUrls: ['./diretiva-ngif.component.css']
})
export class DiretivaNgifComponent implements OnInit {

  minhaVariavel : string; // variável com espaço

  constructor() { }

  ngOnInit() {
  }

}
```

```
C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando\Horascomplementares\ProjetosB\Video19 - Cópia>ng lint
Warning: The 'no-use-before-declare' rule requires type checking
ERROR: C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando\Horascomplementares\ProjetosB\Video19 - Cópia/src/app/diretiva-ngif/diretiva-ngif.component.ts[10, 16]: expected nospace before colon in property-declaration
Lint errors found in the listed files.
```

Quando o projeto passar pelo *Lint*, é apresentada a mensagem:

```
C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando\Horascomplementares\ProjetosB\Video19 - Cópia>ng lint
Warning: The 'no-use-before-declare' rule requires type checking
All files pass linting.
```

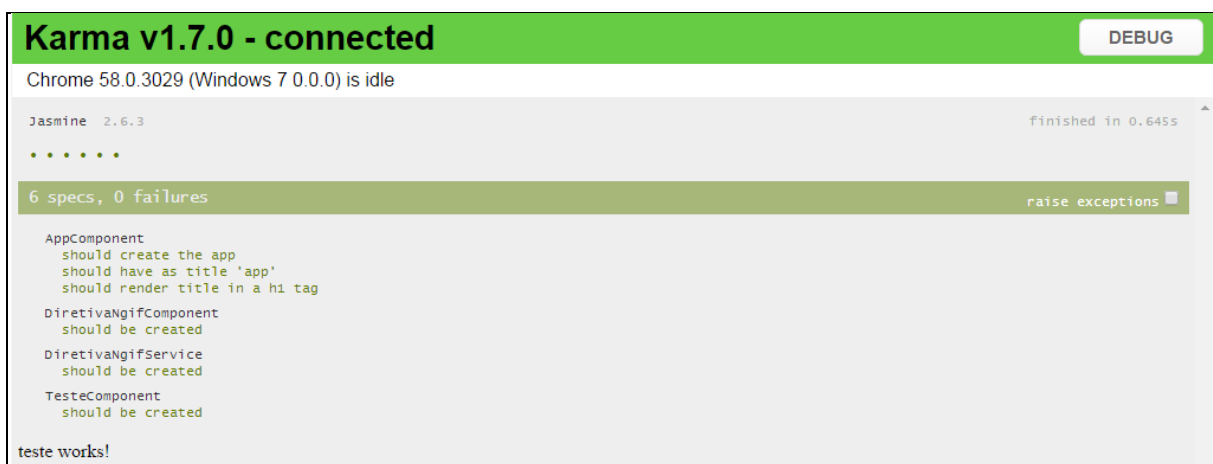
Exemplo do *ng Test*: O *ng Test* verifica todos os arquivos com extensão *.spec.ts* e realiza testes unitários em cada *providers* de teste, verificando se há algum erro. Quando o processo for

finalizado, seu navegador padrão será aberto com o *Karma*¹, repassando detalhes dos testes efetuados.

No terminal, caso não existam falhas, será apresentada a mensagem:

```
C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando\Horascomplementares\ProjetosB\Video19 - Cópia>ng test
10% building modules 2/4 modules 2 active ...osB\Video19 - Cópia\src /\.spec\.ts$16 06 2017 18:45:43.145:WARN [karma]: No captured
browser, open http://localhost:9876/
16 06 2017 18:45:43.169:INFO [karma]: Karma v1.7.0 server started at http://0.0.0.0:9876/
16 06 2017 18:45:43.171:INFO [launcher]: Launching browser Chrome with unlimited concurrency
16 06 2017 18:45:43.192:INFO [launcher]: Starting browser Chrome
16 06 2017 18:46:08.211:WARN [karma]: No captured browser, open http://localhost:9876/
16 06 2017 18:46:16.818:INFO [Chrome 58.0.3029 (Windows 7 0.0.0)]: Connected on socket 5E8DT5s6y48LcOhrAAAA with id 36398578
Chrome 58.0.3029 (Windows 7 0.0.0): Executed 6 of 6 SUCCESS (0.681 secs / 0.622 secs)
```

No navegador, o resultado é:

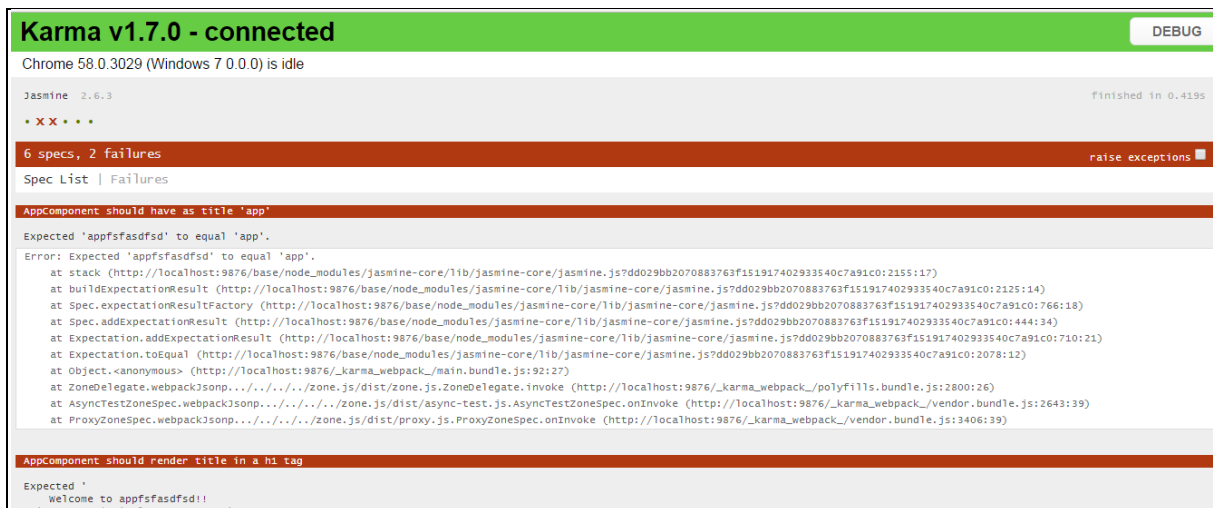


Exemplo de erro:

```
Chrome 58.0.3029 (Windows 7 0.0.0) AppComponent should have as title 'app' FAILED
Expected 'appfsfasdfsd' to equal 'app'.
    at Object.<anonymous> (http://localhost:9876/_karma_webpack_/main.bundle.js:92:27)
    at ZoneDelegate.webpackJsonp.../.../zone.js/dist/zone.js.ZoneDelegate.invoke (http://localhost:9876/_karma_webpack_/polyfills.bundle.js:2800:26)
    at AsyncTestZoneSpec.webpackJsonp.../.../zone.js/dist/async-test.js.AsyncTestZoneSpec.onInvoke (http://localhost:9876/_karma_webpack_/vendor.bundle.js:2643:39)
    at ProxyZoneSpec.webpackJsonp.../.../zone.js/dist/proxy.js.ProxyZoneSpec.onInvoke (http://localhost:9876/_karma_webpack_/vendor.bundle.js:3406:39)
Chrome 58.0.3029 (Windows 7 0.0.0): Executed 2 of 6 (1 FAILED) (0 secs / 0.266 secs)
Chrome 58.0.3029 (Windows 7 0.0.0) AppComponent should have as title 'app' FAILED
Expected 'appfsfasdfsd' to equal 'app'.
    at Object.<anonymous> (http://localhost:9876/_karma_webpack_/main.bundle.js:92:27)
    at ZoneDelegate.webpackJsonp.../.../zone.js/dist/zone.js.ZoneDelegate.invoke (http://localhost:9876/_karma_webpack_/polyfills.bundle.js:2800:26)
    at AsyncTestZoneSpec.webpackJsonp.../.../zone.js/dist/async-test.js.AsyncTestZoneSpec.onInvoke (http://localhost:9876/_karma_webpack_/vendor.bundle.js:2643:39)
    at ProxyZoneSpec.webpackJsonp.../.../zone.js/dist/proxy.js.ProxyZoneSpec.onInvoke (http://localhost:9876/_karma_webpack_/vendor.bundle.js:3406:39)
Chrome 58.0.3029 (Windows 7 0.0.0) AppComponent should render title in a h1 tag FAILED
Expected '
Welcome to appfsfasdfsd!!
' to contain 'Welcome to app!!'.
    at Object.<anonymous> (http://localhost:9876/_karma_webpack_/main.bundle.js:98:58)
```

No navegador, o resultado é:

¹ Karma é uma ferramenta de testes que permite executar cada linha do código em diferentes navegadores (assim é possível verificar a compatibilidade da aplicação com o navegador, processo chamado de cross-navegador)



Link útil:

- 1) Documentação da Jasmine: <https://jasmine.github.io/>

23 VÍDEO #21: ANGULAR CLI: ESTRUTURA DO PROJETO

Ao criar uma aplicação no Angular CLI:

- São criadas todas as estrutura do projeto;
- Inclusive a página *HTML* inicial, os arquivos *Typescript* iniciais, os arquivos *.CSS* e os arquivos para testes unitários (*spec.ts*);
- É criado o arquivo *package.json* com todas as dependências do Angular 2;
- Todas as dependências do *Node.js* são instaladas (ao rodar o comando `npm install`);
- O *Karma* é configurado para executar os testes unitários com Jasmine;
- O *Protractor* também é configurado para executar os testes *end-to-end* (E2E);
- É inicializado um repositório git no projeto e é feito o *commit* inicial e
- Todos os arquivos necessários para ser feito o *build* da aplicação para produção são criados.

Por baixo de sua estrutura, o Angular usa a ferramenta *EmberCLI* para deixar tudo organizado.

Essa é a estrutura do projeto:

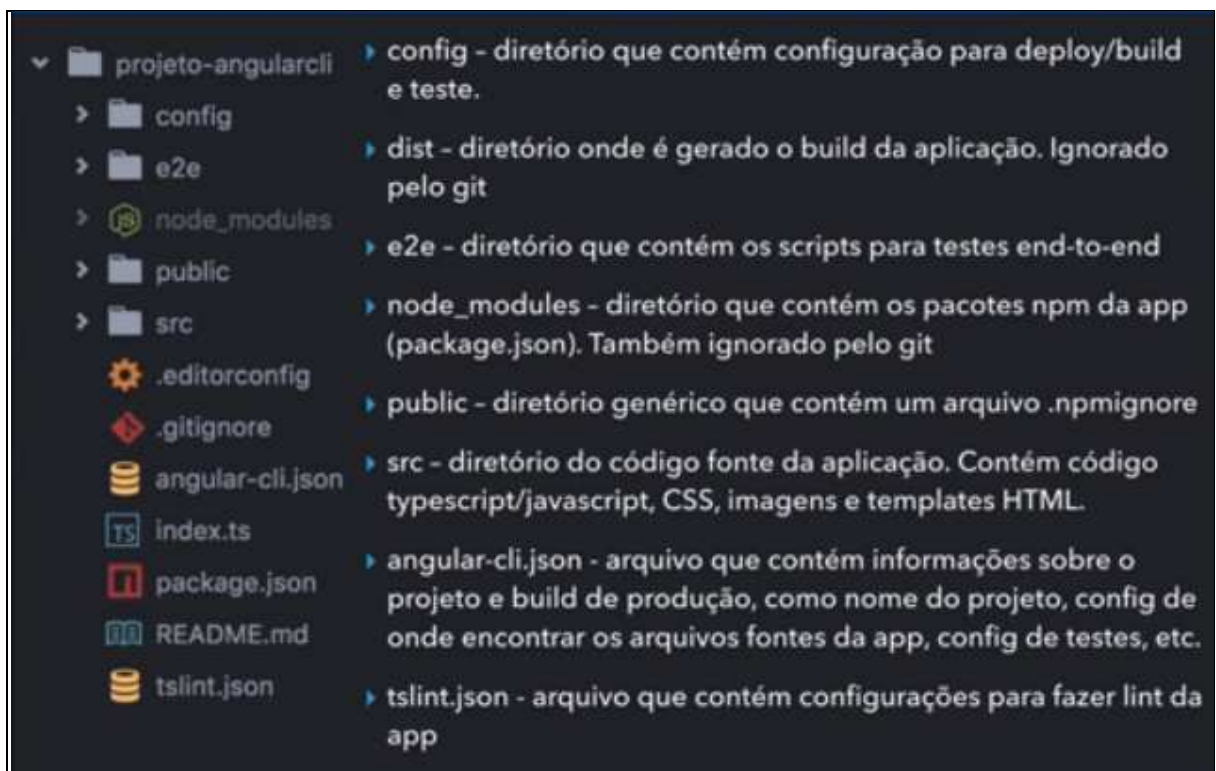


Imagem retirada do vídeo

Essa é a estrutura do código:

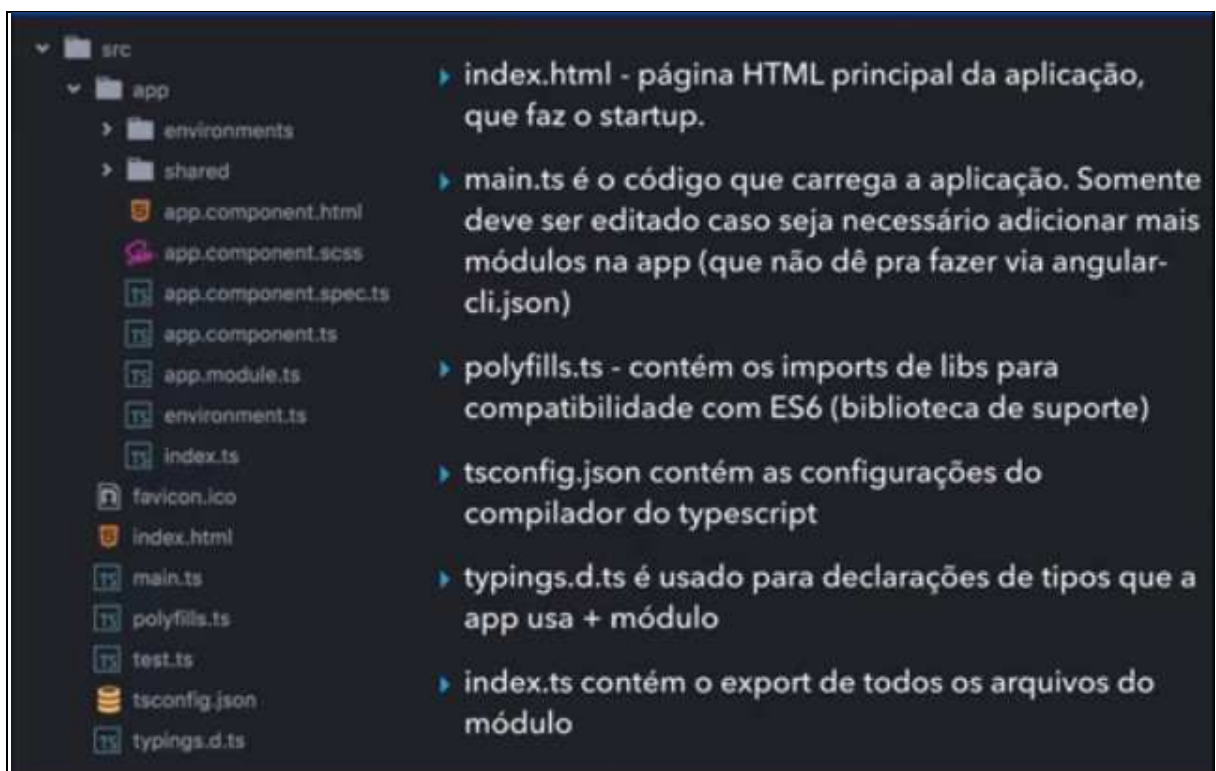


Imagem retirada do vídeo

Estrutura do *Package.json*: O *package.json* possui todas as configurações e todas as bibliotecas que serão utilizadas no projeto

Dependencies x *DevDependencies*:

- *Dependencies*: Dependências necessárias para executar a aplicação
- *DevDependencies*: Dependências necessárias para o desenvolvimento da aplicação, que não serão utilizadas no build de produção.

Dependencies:

- ▶ @angular/core - pacote principal do framework Angular 2. Contém decorators e metadados, Component, Directive, injeção de dependência e os hooks de ciclo de vida do Component.
- ▶ @angular/common - Serviços, pipes e diretivas comuns fornecidas pelo time de dev do Angular.
- ▶ @angular/compiler - Template de compilação do angular. Entende o código dos templates e converte em código que faz a app ser executada e renderizada. Desenvolvedores não interagem com esse pacote diretamente (apenas usamos seu código).
- ▶ @angular/forms - contém todo o código para construção de formulários no angular 2.
- ▶ @angular/platform-browser - contém todo o código relacionado ao DOM e ao browser, especialmente as parte que ajudam a renderizar o DOM. Esse pacote também contém o método para fazer o bootstrap da aplicação para builds de produção que pré-compila os templates.
- ▶ @angular/platform-browser-dynamic - Contém os Providers e o método para iniciar as aplicações que compilam templates no lado cliente. Não usa compilação offline. Usada para fazer bootstrap durante desenvolvimento e exemplos plunker.
- ▶ @angular/http - fornece o cliente HTTP.
- ▶ @angular/router - classes de roteamento.

Imagem retirada do vídeo

Dependencies Polyfills (bibliotecas auxiliares):

- ▶ core-js - biblioteca que permite compatibilidade de engines JS antigas com a especificação do ES 2015, ou seja, emula as funcionalidades do ES 2015 (ES6) e ES 7 em browsers que suportam somente ES5.
- ▶ reflect-metadata - dependência compartilhada entre o Angular e o compilador TypeScript. Permite o uso de decorators no código (annotations). Isso permite ao desenvolvedores fazer upgrade no TypeScript sem precisar de fazer upgrade no Angular. Esse é o motivo desta ser uma dependência da aplicação e não do angular.
- ▶ rxjs - extensão para a especificação dos Observables (programação assíncrona). Reactive extensions for JavaScript.
- ▶ ts-helpers - biblioteca auxiliar que permite otimização de código typescript quando o mesmo é compilado para ES 5.
- ▶ zone.js - extensão (plugins) útil para tarefas assíncronas (chamadas de Zones).

Imagem retirada do vídeo

Dependencies Polyfills (bibliotecas auxiliares):

- ▶ @types/jasmine: definição jasmine para typescript (antigo typings)
- ▶ @types/protractor: definição protractor para typescript (antigo typings)
- ▶ angular-cli: ferramenta de linha de comando para gerenciar projetos angular 2.
- ▶ codelyzer: lint (análise de código) para angular 2
- ▶ jasmine-core: arquivos principais jasmine para node.js
- ▶ jasmine-spec-reporter: relatório em tempo real para BDD com Jasmine
- ▶ karma: ferramenta de testes que cria um web server e executa código de teste em cada um dos browsers conectados
- ▶ karma-chrome-launcher: launcher do karma para o chrome
- ▶ karma-jasmine: adaptador para o jasmine
- ▶ karma-remap-istanbul: adaptador para code coverage (relatório)
- ▶ protractor: framework de teste end to end (integração) para angular
- ▶ ts-node: módulo typescript para node.js
- ▶ tslint: lint (análise de código) para typescript
- ▶ typescript: compilador typescript

Imagem retirada do vídeo

24 VÍDEO #22: ANGULAR CLI: GERANDO BUILD DE PRODUÇÃO

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video22>.

Para gerar o *build* de desenvolvimento, podem ser utilizados os comandos abaixo (todos eles possuem a mesma funcionalidade, mas são escritos de formas diferentes):

- `ng build --target=development --environment=dev`
- `ng build --dev --e=dev`
- `ng build --dev`
- `ng build`

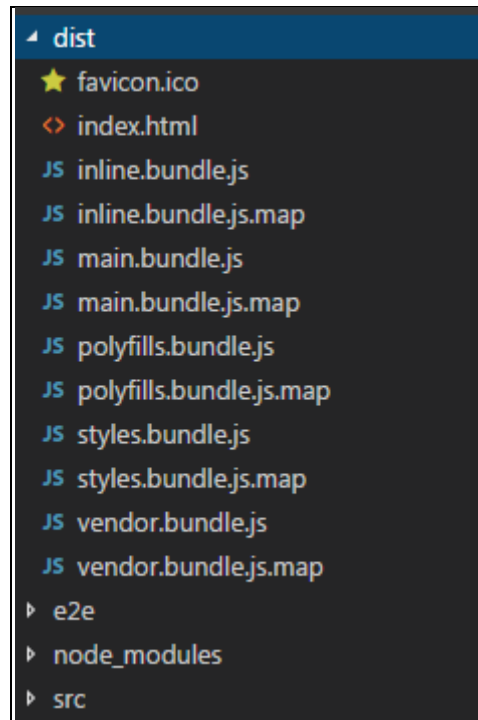
Para gerar o build de produção, podem ser utilizados os comandos abaixo (todos eles possuem a mesma funcionalidade, mas são escritos de formas diferentes):

- `ng build --target=production --environment=prod`
- `ng build --dev --e=prod`
- `ng build --prod`

Gerando um *build* de desenvolvimento:

```
C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando\Horascomplementares\ProjetosB\Video22>ng build
Hash: 935aaba62fb939e1dd9c
Time: 22534ms
chunk {0} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 160 kB {4} [initial] [rendered]
chunk {1} main.bundle.js, main.bundle.js.map (main) 8.84 kB {3} [initial] [rendered]
chunk {2} styles.bundle.js, styles.bundle.js.map (styles) 10.5 kB {4} [initial] [rendered]
chunk {3} vendor.bundle.js, vendor.bundle.js.map (vendor) 1.88 MB [initial] [rendered]
chunk {4} inline.bundle.js, inline.bundle.js.map (inline) 0 bytes [entry] [rendered]
```

Ao concluir o *build*, será criado o pacote *dist* com todos os arquivos minificados do seu projeto:

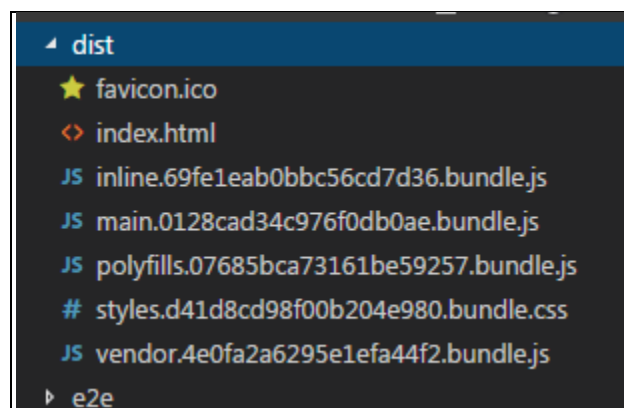


Gerando um *build* de produção:

Ao concluir o *build*, será criado o pacote *dist* com todos os arquivos minificados do seu projeto. Se for criado um *build* de produção, o pacote *dist* será substituído:

```
C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando\Horascomplementares\ProjetosB\Video22>ng build --prod
Hash: 68163342a6b86d100aca
Time: 18440ms
chunk {0} polyfills.07685bca73161be59257.bundle.js (polyfills) 160 kB {4} [initial] [rendered]
chunk {1} main.0128cad34c976f0db0ae.bundle.js (main) 14.2 kB {3} [initial] [rendered]
chunk {2} styles.d41d8cd98f00b204e980.bundle.css (styles) 69 bytes {4} [initial] [rendered]
chunk {3} vendor.4e0fa2a6295e1efa44f2.bundle.js (vendor) 850 kB [initial] [rendered]
chunk {4} inline.69fe1eab0bbc56cd7d36.bundle.js (inline) 0 bytes [entry] [rendered]
```

OBS: Os arquivos do *build* de produção possuem números, para evitar que sua aplicação não seja atualizada, devido cachê do projeto.



25 VÍDEO #23: ANGULAR CLI: INSTALANDO BIBLIOTECAS (BOOTSTRAP, JQUERY, MATERIALIZE, LODASH)

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video23/angular-cli-lib-externas>.

Para instalar o *Bootstrap*, execute o comando: `npm install bootstrap@next` (o `@next` indica que será baixada a última versão). Como dependência do *bootstrap*, também será baixado o *jquery* e o *tether*

Para instalar o *Materialize*, execute os comandos: `npm install materialize-css --save` e depois, `npm install angular2-materialize --save`

Para instalar o *Lodash*, use os comandos: `npm install --save lodash` e `npm install --save @types/lodash`

Depois de executados esses comandos, instale também o *jquery*: `npm install jquery@^2.2.4 --save`

Ao final das instalações, o *package.json* estará com as seguintes dependências:

```
"@types/lodash": "^4.14.66",
"angular2-materialize": "^15.0.4",
"bootstrap": "^4.0.0-alpha.6",
"core-js": "^2.4.1",
"jquery": "^2.2.4",
"lodash": "^4.17.4",
"materialize-css": "^0.98.2",
"rxjs": "^5.1.0",
"zone.js": "^0.8.4"
```

Link útil:

- 1) Documentação do Materialize: <http://materializecss.com/>

26 VÍDEO #24: ANGULAR CLI: INTRODUÇÃO E TIPO DE DIRETIVAS NO ANGULAR 2

Diretivas formas de passarmos instruções para nosso *template* (código *HTML*). Existem dois tipos de diretivas:

- Diretivas Estruturais que interagem com a *view* e modificam a estrutura do *DOW* e/ou código *HTML*, como o `*ngFor` e `*ngIf` e

- Diretivas de atributos que interagem com os elementos em que foram aplicadas, como é o caso da *ng-class* e *ng-style*.

27 VÍDEO #25: ANGULAR CLI: DIRETIVAS: NGIF

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video34>.

A diretiva *ngIf* é como um *IF* em qualquer linguagem de programação, a diferença é que ela não possui o *else*, então para tratar das exceções, é necessário criar outra *ngIf*.

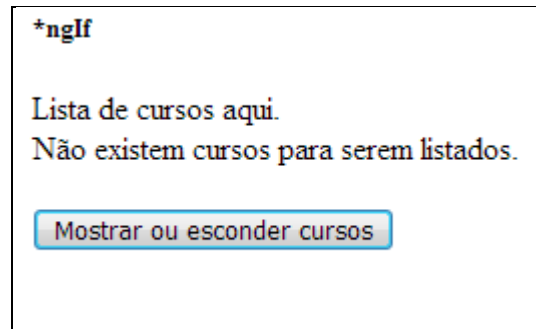
Foi criado um component chamado *diretiva-ngif*, em *diretiva-ngif.component.ts*, foram criadas duas variáveis e o método *onMostrarCursos*:

```
curso: string[] = ["Angular 2"];  
mostrarCursos: boolean = false;  
  
onMostrarCursos(){  
  this.mostrarCursos = !this.mostrarCursos;  
}
```

- Em *diretiva-ngif.component.html*, criamos o seguinte código:

```
<h5>*ngIf</h5>  
  
<!--Para utilizar a diretiva ngIf, use: *ngIf="Expressão"-->  
<div *ngIf="curso.length > 0"> Lista de cursos aqui.  
</div>  
  
<!-- A diretiva ngIf não possui else, então caso precise fazer um else  
é necessário criar outra diretiva ngIf-->  
<div *ngIf="curso.length == 0"> Não existem cursos para serem listados. </div>  
  
<!-- Exemplo de ngIf com variável booleana-->  
  
<div *ngIf="mostrarCursos"> Lista de cursos aqui. </div>  
  
<div *ngIf="!mostrarCursos"> Não existem cursos para serem listados. </div>  
  
<br>  
<button (click)="onMostrarCursos">Mostrar ou esconder cursos</button>
```

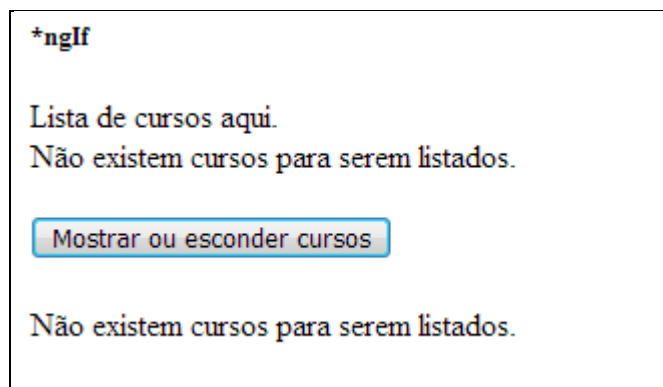
No navegador, o resultado é:



Uma desvantagem em se usar o *ngIf*, é a performance, porque ao negar um *ngIf*, o elemento é destruído. Uma alternativa é utilizar a propriedade *hidden*:

```
<!--Exemplo com hidden: -->
<div [hidden]="!mostrarCursos"> Lista de cursos aqui </div>
<div [hidden]="mostrarCursos"> Não existem cursos para serem listados.</div>
```

No navegador, o resultado é:



Quando usar *ngIf*: Recomendado para árvores de elementos grandes.

Quando usar *[hidden]*: Recomendado para árvores de elementos pequenos. A exceção para utilizar *[hidden]* em árvores grandes, é quando o custo para se criar o a estrutura com o *ngIf* seja muito grande.

28 VÍDEO #26: ANGULAR CLI: DIRETIVAS: NGSWITCH, NGSWITCHCASE E NGSWITCHDEFAULT

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/video26>

A diretiva *ngSwitchCase* pode ser usada como uma forma de *IF – ELSE*.

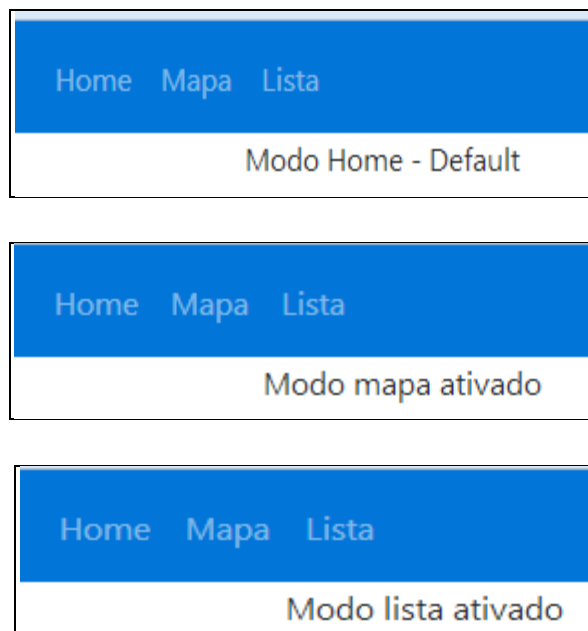
Foi criado um novo component chamado *diretiva-ngswitch*; em *diretiva-ngswitch.component.ts*, criamos a variável “aba”:

```
aba: String = 'home';
```

Em *diretiva-ngswitch.component.html*, criamos a lógica do switch case:

```
<nav class="navbar navbar-toggleable-md navbar-inverse bg-primary">
  <div class="nav navbar-nav">
    <a class="nav-link" [class.ative]="aba == 'home'" (click)="aba = 'home'" >Home</a>
    <a class="nav-link" [class.ative]="aba == 'mapa'" (click)="aba = 'mapa'" >Mapa</a>
    <a class="nav-link" [class.ative]="aba == 'lista'"(click)="aba = 'lista'">Lista</a>
  </div>
</nav>
<!--Para utilizar a diretiva ngSwitchCase, use: *ngSwitchCase="Expressão"-->
<div class="container" [ngSwitch]="aba">
  <p *ngSwitchCase="'mapa'"> Modo mapa ativado </p>
  <p *ngSwitchCase="'lista'"> Modo lista ativado </p>
  <!--Podemos também aplicar um valor padrão-->
  <p *ngSwitchDefault> Modo Home - Default </p>
</div>
```

No navegador, o resultado foi:



29 VÍDEO #27: ANGULAR CLI: DIRETIVAS: NGFOR

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video27>

Similar ao *loop for* de outras linguagens de programação.

Criado um novo *component*, chamado de *diretiva-ngfor*; em *diretiva-ngfor.component.ts*, declaramos uma variável e implementamos o método *ngOnInit*:

```
  cursos: string[] = ["Angular 2", "Java", "Phonegap"];

  constructor() { }

  ngOnInit() {
    for(let i=0; i<this.cursos.length; i++){
      let curso = this.cursos[i];
    }
  }
}
```

Em *diretiva-ngfor.component.html* implementamos a lógica do *for*:

```
<h5> Diretiva ngFor</h5>

<ul>
  <li *ngFor="let curso of cursos">
    {{ curso }}
  </li>
</ul>

<!-- Também é possível passar o valor do index -->
<ul>
  <li *ngFor="let curso of cursos, let i = index">
    {{ i + 1 }} {{ curso }}
  </li>
</ul>
```

No navegador, o resultado é:

Diretiva ngFor

- Angular 2
- Java
- Phonegap

- 1 Angular 2
- 2 Java
- 3 Phonegap

30 VÍDEO #28: ANGULAR CLI: DIRETIVAS: SOBRE O * E TEMPLATE

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video28>.

Utilizar o `*ngDIRETIVA` é apenas para facilitar o uso; podemos utilizar a diretiva usando o `<template [ngDiretiva]>` ou `<div template="ngDiretiva método">`

```
<h5>Removendo o * e usando template</h5>

<!-- Outras formas de escrever a diretiva *ngIf-->
<template [ngIf] = 'mostrarCursos'>
  <div> Lista de cursos exemplo 1 </div>
</template>

<div template="ngIf mostrarCursos">
  Lista de cursos exemplo 2
</div>
```

31 VÍDEO #29: ANGULAR CLI: DIRETIVAS: NGCLASS

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video29>.

```
<!-- Exemplo sem o ngClass-->
<h1>
  <i class="glyphicon"
    [class.glyphicon-star-empty]="!meuFavorito"
    [class.glyphicon-star]="meuFavorito"
    (click)="onClick()"
  ></i>
</h1>

<!-- Exemplo com ngClass-->
<!-- O ngClass deve ser usado quando temos mais de um class-->
<h1>
  <i class="glyphicon"
    [ngClass]="{
      'glyphicon-star-empty': !meuFavorito,
      'glyphicon-star': meuFavorito
    }"
    (click)="onClick()"
  ></i>
</h1>
```

Resultado no navegador:



Link útil:

- 1) Importar o *Bootstrap* 3 que possuem os ícones: <http://getbootstrap.com/getting-started/#download>

32 VÍDEO #30: ANGULAR CLI: DIRETIVAS: NGSTYLE

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video30>.

A diretiva *ngStyle* é estrutural, *semelhante* a diretiva *ngClass*.

Foi criado um novo component chamado *diretiva-ngStyle*; em *diretiva-ng-style.component.ts*, foram adicionadas duas variáveis e o método *mudarAtivo*:

```
ativo: boolean = false;

tamanhoFonte: number = 10;

constructor() { }

ngOnInit() {
}

mudarAtivo(){
  this.ativo = !this.ativo;
}
```

- Em *diretiva-ng-style.component.html*, temos a lógica:

```
<!--Exemplo sem o ngStyle-->
<button
```

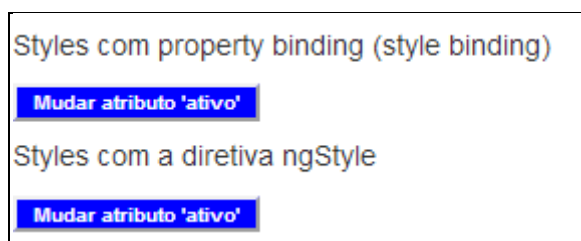
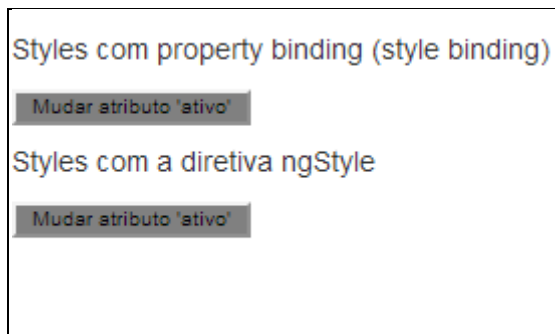
```

[style.backgroundColor]="ativo ? 'blue' : 'gray'"
[style.color]="ativo ? 'white' : 'black'"
[style.fontWeight]="ativo ? 'bold' : 'normal'"
[style.fontSize]="tamanhoFonte + 'px'"
(click)="mudarAtivo()"
> Mudar atributo 'ativo'
</button>
<br>

<h5> Styles com a diretiva ngStyle </h5>
<button
  [ngStyle]="{
    'backgroundColor': ativo ? 'blue' : 'gray',
    'color': ativo ? 'white' : 'black',
    'fontWeight': ativo ? 'bold' : 'normal',
    'fontSize': tamanhoFonte + 'px'
  }"
  (click)="mudarAtivo()"
> Mudar atributo 'ativo'
</button>

```

No navegador, o resultado é:



33 VÍDEO #31: OPERADOR ELVIS ("?")

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video31>.

Esse operador oferece uma maneira segura de navegação entre os objetos.

Foi criado um novo *component*, chamado de *operador-elvis*; em *operador-elvis.component.ts*, criamos um objeto “*tarefa*”:

```
tarefa: any = {
  desc: 'Descrição da tarefa',
  responsavel: null
}
```

- Em *operador-elvis.component.html*, implementamos a lógica:

```
<p> Descrição da tarefa: {{tarefa.desc}} </p>

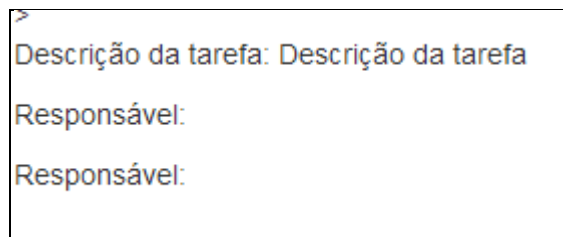
<!-- Em alguns momentos, podemos tentar ler um objeto e ele retornar como null: -->
<!--<p> Responsável: {{tarefa.responsavel.nome}} </p>-->

<!-- Para resolver isso: -->
<p> Responsável: {{tarefa.responsavel != null ? tarefa.responsavel.nome : ''}} </p>

<!-- Como o código fica muito extenso escrevendo dessa forma, usamos o operador Elvis:
Quando informamos esse operador depois de uma variável ou objeto, indicamos que ele
pode retornar como null; ele terá a mesma funcionalidade da expressão anterior, porém
escrito de forma mais prática-->

<p> Responsável: {{tarefa.responsavel?.nome}} </p>
```

No navegador, o resultado é:



34 VÍDEO #32: NG-CONTENT

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video32>.

O *ngContent* deve ser usado para passar conteúdo.

- Em *exemplo-ng-content.component.html*:

```
<!-- Exemplo com apenas um conteúdo-->
<div class="panel panel-default">
```

```

<div class="panel-heading">Título</div>
<div class="panel-body">
  <ng-content> </ng-content> <!-- Utiliza-se a tag ng-content-->
</div>
</div>

<!-- Exemplo com vários conteúdos
  Nesse caso, temos que criar seletores e passar eles no nosso app.component.html-->
<div class="panel panel-default">
  <div class="panel-heading">
    <ng-content select=".titulo"> </ng-content>
  </div>
  <div class="panel-body">
    <ng-content select=".corpo"> </ng-content>
  </div>
</div>

```

- Em *app.component.html*:

```

<!-- Exemplo com apenas um conteúdo-->
<!--<app-exemplo-ng-content>
  Conteúdo passado para o component.
</app-exemplo-ng-content>-->

<!-- Exemplo com vários conteúdos-->
<app-exemplo-ng-content>
  <div class="titulo"> Título do painel </div>
  <div class="corpo"> Conteúdo passado pelo component - segundo exemplo</div>
</app-exemplo-ng-content>

```

No navegador, o resultado é:

Título
Título do painel
Conteúdo passado pelo component - segundo exemplo

35 VÍDEO #33: CRIANDO UMA DIRETIVA DE ATRIBUTO

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video33>.

O atributo *ElementRef* faz referência ao elemento do *DOW* e o atributo *Renderer* é quem renderiza o *DOW*.

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

@Directive({
  selector: '[fundoAmarelo]'
  // Se você quer que a diretiva seja aplicada apenas a alguma parte do código como
  // botões, components, input, parágrafo informe isso na frente do seletor. Ex:
  button[fundoAmarelo]
})
export class FundoAmareloDirective {

  constructor(private elementoRef: ElementRef, private renderer: Renderer) {
    // console.log(this.elementoRef)
    // Os desenvolvedores do Angular recomendam não utilizar o nativeElement; isso porque
    // através desse
    // acessamos diretamente um elemento no árvore DOW, o que pode tornar a aplicação
    // vulneráveis a ataque.
    // this.elementoRef.nativeElement.style.backgroundColor= 'yellow';
    // A melhor prática para ser utilizada, é usar o Renderer:
    this.renderer.setStyle(this.elementoRef.nativeElement, 'background-color',
    'yellow');
  }
}
```

36 VÍDEO #34: DIRETIVAS HOSTLISTENER E HOSTBINDING

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video34>.

Os atributos *hostListener* e *hostBinding* nos permite escutar um evento, sempre que ele for alterado:

```
@Directive({
  selector: '[appHighlighMouse]'
})
export class HighlighMouseDirective {

  // Aqui nós temos dois métodos muito parecidos, que estão alterando a propriedade do
  // HTML:
  @HostListener('mouseenter') onMouseOver(){
    // this.renderer.setStyle(this.elementRef.nativeElement, 'background-color',
    // 'yellow');
    this.backgroundColor = 'yellow';
  }

  @HostListener('mouseleave') onMouseLeave(){
    // this.renderer.setStyle(this.elementRef.nativeElement, 'background-color',
    // 'white');
  }
}
```

```

    this.backgroundColor = 'white';
  }

  // O Angular possui o metadado HostBinding, que permite que nós façamos uma associação da
  // nossa diretiva
  // com o HTML:
  @HostBinding('style.backgroundColor') backgroundColor: string;

  // Construtor para o primeiro exemplo:
  // constructor(private elementRef: ElementRef, private renderer: Renderer) { }
}

```

37 VÍDEO #35: DIRETIVAS: INPUT E PROPERTY BINDING

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video35>.

Criada uma nova diretiva chamada *highlight* e utilizamos a diretiva criada no capítulo anterior; em *diretivas-customizadas.component.html*:

```

<p fundoAmarelo> Texto com fundo amarelo.</p>
<button fundoAmarelo> Botão com fundo amarelo.</button>

<p highlightMouse>
  Texto com highlight quando passo o mouse.
</p>

<p [highlight]="red" [defaultColor]= "grey">
  Texto com highlight com cores customizadas.
</p>

```

Em *highlight.directive.ts*, definimos nossos métodos:

```

export class HighlightDirective {

  @HostListener('mouseenter') onMouseOver(){
    this.backgroundColor = this.highlightColor;
  }

  @HostListener('mouseleave') onMouseLeave(){
    this.backgroundColor = this.defaultColor;
  }

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

  @Input() defaultColor: string = 'white';
  @Input('highlight') highlightColor: string = 'yellow';

  constructor() { }
}

```

```

ngOnInit(){
  this.backgroundColor = this.defaultColor;
}
}

```

38 VÍDEO #36: CRIANDO UMA DIRETIVA DE ESTRUTURA (NGELSE)

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video36>.

Uma diretiva estruturada é uma diretiva que altera a estrutura do *DOW*:

```

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

@Directive({
  selector: '[ngElse]'
})
export class NgElseDirective {

  @Input() set ngElse (condition: boolean){
    if(!condition){
      // TemplateRef faz referência ao template
      // ViewContainerRef faz referência ao conteúdo da view

      // Renderizando a view no template:
      this._viewContainerRef.createEmbeddedView(this._templateRef);
    } else{
      this._viewContainerRef.clear();
    }
  }

  constructor(private _templateRef: TemplateRef<any>, private _viewContainerRef: ViewContainerRef) { }
}

```

39 VÍDEO #37: INTRODUÇÃO A SERVIÇOS

A classe de *Service* é uma classe que busca os dados e nos retornam esses dados; também é útil para que não dupliquemos códigos na aplicação, utilizando o conceito de DRY: *Don't repeat yourself!* O ideal é que toda a lógica de negócio fique na classe de serviço. Nos serviços também podemos ter classes utilitárias, com códigos de formatação por exemplo.

40 VÍDEO #38: CRIANDO UM SERVIÇO

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video38>.

Para criar um serviço utilize o comando “*npm g s nomeDoServiço*” ou crie manualmente um arquivo com extensão *service.ts*.

41 VÍDEO #39: INJEÇÃO DE DEPENDÊNCIA (DI) + COMO USAR UM SERVIÇO EM UM COMPONENT

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video39>.

O que é Dependência? É quando uma classe precisa de outra classe para funcionar. Ao injetar dependência entre classes, a classe dependente é instanciada automaticamente. Existem três maneiras dessa injeção ser realizada: por Construtores, por Métodos *Setters* e por Atributos.

Para injetar a dependência, deve ser usado o *@Injectable()* e em *app.module.ts*, a classe deve ser informada como um *Providers* (que no caso é um Fornecedor).

42 VÍDEO #40: ESCOPO DE INSTÂNCIAS DE SERVIÇOS + MÓDULOS (SINGLETON E VÁRIAS INSTÂNCIAS)

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video40>.

Para criar um módulo de funcionalidade, utilizamos o *CommonModule* (para módulo de *root* ou raiz, utilizamos o *NavegadorModule*).

Padrão *Singleton*: ter apenas uma instância do serviço para toda a aplicação, não importa onde o serviço será declarado. Se quer um escopo para toda a aplicação, declare dentro de *app.module.ts*; caso contrário, você pode declarar o serviço no *Providers* de um módulo e todos os componentes declarados em *declarations*, terão acesso ao serviço. Se você quer que um serviço seja acessado apenas por um *providers*, no seu component (*@Component*) declare o *providers* e o serviço.

43 VÍDEO #41: COMUNICAÇÃO ENTRE COMPONENTES USANDO SERVIÇOS (BROADCAST DE EVENTOS)

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video41>.

Para os componentes se comunicarem, podemos usar o *@Input* ou o *@Output*, porém essa comunicação só acontece entre componente pai x componente filho. O *@Input* utilizamos para que

o componente pai passe informações para o componente filho e o `@Output` utilizamos para que o componente filho repasse informações para o componente pai.

Dificuldade encontrada:

Ao utilizar o `subscribe` do evento `EventEmitter`, ocorre o seguinte erro de compilação: “[ts] Property ‘subscribe’ does not exist on type ‘EventEmitter’”. Procurei na documentação do Node.js os possíveis eventos da classe `EventEmitter` e realmente não encontrei a ação “`subscribe`”; como não sei por qual ele pode ser substituído, mantive o “`subscribe`” apesar de projeto apresentar erro de compilação.

44 VÍDEO #42: INJETANDO UM SERVIÇO EM OUTRO SERVIÇO

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video42>.

Para criar um novo serviço, execute o comando: `ng g s nomeDoServiço` (você também pode utilizar “`service`” ao invés de somente “`s`”: `ng g service nomeDoServiço`). Ao criar um serviço, como não estamos utilizando ele em nenhum `providers`, será apresentada essa *warning* no console:

```
C:\Users\nilza\Desktop\NATALIA\Faculdade\Fernando\Horascomplementares\ProjetosB\Video42>ng g s shared/log
installing service
  create src/app/shared/log.service.spec.ts
  create src/app/shared/log.service.ts
WARNING Service is generated but not provided, it must be provided to be used
```

Para injetar o serviço criado dentro de outro, basta ir no construtor do serviço que vai receber a injeção e criar uma variável do tipo serviço que vai ser injetado:

```
constructor(private logService: LogService)
```

45 VÍDEO #43: PIPES (USANDO PIPES, PARÂMETROS E PIPES ANINHADOS)

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video43>.

Os *Pipes* transformam um valor e podemos mostrar esse valor transformado dentro de um *template*, como por exemplo, valores de datas e moedas. Criamos um novo projeto Angular CLI e um novo component chamado `exemplos-pipes`.

- Em `exemplos-pipes.component.ts` criamos um objeto “livro”:

```

livro: any = {
  titulo: 'Learning JavaScript Data Structures and Algorithms 2nd ed',
  rating: 4.54321,
  numeroPaginas: 314,
  preco: 44.99,
  dataLancamento: new Date(2016, 5, 23),
  url: 'http://a.co/glqjpRP'
};

```

- Em *exemplos-pipes.component.html*, apresentamos os dados:

```

<h5> Exemplo de Pipes </h5>

<p>Título:{{ livro.titulo }} </p>
<p>Estrelas: {{ livro.rating }}</p>
<p>Páginas: {{ livro.numeroPaginas }}</p>
<p>Preço: {{ livro.preco }}</p>
<p>Data Lançamento: {{ livro.dataLancamento }}</p>
<p>URL: {{ livro.url }}</p>
<br>
<p> Livro: {{ livro }}</p>

```

No navegador, o resultado é:

Exemplo de Pipes

Título: Learning JavaScript Data Structures and Algorithms 2nd ed

Estrelas: 4.54321

Páginas: 314

Preço: 44.99

Data Lançamento: Thu Jun 23 2016 00:00:00 GMT-0300 (Hora oficial do Brasil)

URL: <http://a.co/glqjpRP>

Livro: [object Object]

Para usar um pipe, basta utilizar a tecla “|” do teclado e o pipe desejado.

```

<h5> Exemplo de Pipes </h5>

<!-- | uppercase: deixa todas as letras em maiúsculo
      | lowercase: deixa tudo em letras minúsculas
      | currency: deixa o preço em moeda americana; se você quiser passar a moeda
          brasileira, basta passar o código "BRL"; caso queira mostrar o cifrão ao invés do
          código da moeda, use mais o parâmetro 'true'
      | É possível formatar a data; use a documentação para verificar todos os formatos.

```

```

    Também é possível passar parâmetros para o pipe; informe o tipo do pipe: 'parâmetro'-->
<p>Título:{{ livro.titulo | uppercase }} </p>
<p>Estrelas: {{ livro.rating | number:'1.1-2' }}</p> <!--passando o número de casas
                                                    inteiras (1) e o número mínimo de
                                                    casas decimais (1) e o número
                                                    máximo de casas decimais (2)-->
<p>Páginas: {{ livro.numeroPaginas | number }}</p>
<p>Preço: {{ livro.preco | currency: 'BRL': 'true' }}</p> <!--Se você quer passar mais de um
                                                    Parâmetro utilize novamente os
                                                    ":"-->
<p>Data Lançamento: {{ livro.dataLancamento | date:'dd-MM-yyyy' }}</p> <!--Se quiser
                                                    apresentar o mês por
                                                    extenso, use "MMM"-->
<p>URL: {{ livro.url }}</p>
<br>
<p> Livro: {{ livro |json }}</p>

```

Link útil:

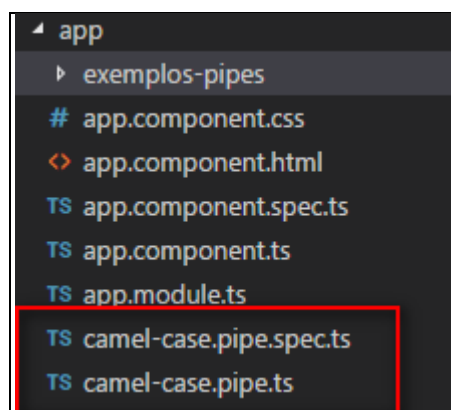
- 1) Documentação de *Pipe* disponível em: <https://angular.io/api?query=pipe>.

46 VÍDEO #44: CRIANDO UM PIPE

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video44>.

Para criar um pipe, use o comando: `ng g p nomeDoPipe` (ao invés de “p” você também pode usar “pipe”, ficando o comando assim: `ng g pipe nomeDoPipe`).

Ao criar um pipe, os seguintes arquivos são adicionados:



É importante que os pipes criados sejam declarados no módulo onde ele será utilizado:

- Em `app.module.ts`:

```

import { NavegadorModule } from '@angular/platform-navegador';
import { NgModule } from '@angular/core';

```

```

import { AppComponent } from './app.component';
import { ExemplosPipesComponent } from './exemplos-pipes/exemplos-pipes.component';
import { CamelCasePipe } from './camel-case.pipe';

@NgModule({
  declarations: [
    AppComponent,
    ExemplosPipesComponent,
    CamelCasePipe
  ],
  imports: [
    NavegadorModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

Para utilizar um pipe, basta informá-lo no *template*:

```

<h5> Exemplo de Pipes </h5>

<p>Título: {{ livro.titulo | uppercase | lowercase | camelCase }} </p>
<p>Estrelas: {{ livro.rating | number:'1.1-2' }}</p>
<p>Páginas: {{ livro.numeroPaginas | number }}</p>
<p>Preço: {{ livro.preco | currency: 'BRL': 'true' }}</p>
<p>Data Lançamento: {{ livro.dataLancamento | date:'dd-MM-yyyy' }}</p>
<p>URL: {{ livro.url }}</p>
<br>
<p> Livro: {{ livro | json }}</p>

```

- Em *camel-case.pipe.ts* temos a lógica do pipe:

```

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'camelCase'
})

// O pipe precisa implementa uma interface chamada PipeTransform
// Quando implementamos essa interface, ela sobrepõe o método transform
export class CamelCasePipe implements PipeTransform {

  // O "value" são os valores a serem transformados e os "args" são os argumentos
  // (opcional)
  // Função para deixar a primeira letra de cada palavra em maiúscula:
  transform(value: any, args?: any): any {
    let values = value.split(' ');
    let result = '';
    for (let v of values){
      result += this.capitalize(v) + ' ';
    }
    return result;
  }
}

```

```

}

capitalize(value: string){
  return value.substr(0,1).toUpperCase() + value.substr(1).toLowerCase();
}
}

```

No navegador, o resultado é:

Exemplo de Pipes

Título: Learning Javascript Data Structures And Algorithms 2nd Ed

Estrelas: 4.54

Páginas: 314

Preço: R\$44.99

Data Lançamento: 23-06-2016

URL: http://a.co/gljipRP

Livro: { "titulo": "Learning JavaScript Data Structures and Algorithms 2nd ed", "rating": 4.54321, "numeroPaginas": 314, "preco": 44.99, "dataLancamento": "2016-06-23T03:00:00.000Z", "url": "http://a.co/gljipRP" }

47 VÍDEO #45: APLICANDO LOCALE (INTERNACIONALIZAÇÃO) NOS PIPES

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video45>.

Por padrão, todas as aplicações do Angular usam o padrão americano para moedas e datas; para que possamos utilizar o padrão brasileiro, temos que fazer o seguinte:

- Em *app.module.ts*, declare o *token* LOCALE_ID como um *providers* e informe qual é o padrão que deve ser utilizado:

OBS: O LOCALE_ID deve ser importado.

```

import { SettingsService } from './settings.service';
import { NavegadorModule } from '@angular/platform-navegador';
import { NgModule, LOCALE_ID } from '@angular/core';

import { AppComponent } from './app.component';
import { ExemplosPipesComponent } from './exemplos-pipes/exemplos-pipes.component';
import { CamelCasePipe } from './camel-case.pipe';

@NgModule({
  declarations: [
    AppComponent,
    ExemplosPipesComponent,
    CamelCasePipe
  ],
  imports: [
    NavegadorModule
  ],

```

```

providers: [
  {
    provide: LOCALE_ID,
    // Existem três formas de serem feitas essa dependência:
    // Pelo useValue onde você passa apenas um valor
    // Pelo useClass onde você a classe que vai fornecer a dependência ou
    // Pelo useFactory que se usa mais para projetos de padrão Factory
    useValue: 'pt-BR'
  }
  /* Você também pode usar um serviço, que possui o seu LOCALE_ID:
  SettingsService, {
    provide: LOCALE_ID,
    deps: [SettingsService],
    useFactory: (settingsService) => settingsService.getLocale
  }*/
],
bootstrap: [AppComponent]
})
export class AppModule { }

```

No navegador, o resultado ficou assim:

Exemplo de Pipes

Título: Learning Javascript Data Structures And Algorithms 2nd Ed

Estrelas: 4,54

Páginas: 314

Preço: R\$44,99

Data Lançamento: 23-06-2016

URL: <http://a.co/gljipRP>

Livro: { "título": "Learning JavaScript Data Structures and Algorithms 2nd ed", "rating": 4.54321, "numeroPaginas": 314, "preco": 44.99, "dataLancamento": "2016-06-23T03:00:00.000Z", "url": "http://a.co/gljipRP" }

48 VÍDEO #46: PIPES: CRIANDO UM PIPE "PURO"

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video46>.

Pipe puro é um pipe que não olha as modificações do parâmetro passado no método “transform”.

49 VÍDEO #47: PIPES: CRIANDO UM PIPE "IMPURO"

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video47>.

Pipe impuro é um pipe que olha as modificações do parâmetro passado no método “*transform*”. O Angular possui um metadado que declara se o pipe puro ou não; seu padrão é “true” = puro.

```
@Pipe({
  name: 'filtroArrayImpuro',
  pure: false
})
```

50 VÍDEO #48: PIPES: ASYNC (ASSÍNCRONO)

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video48>.

Pipe assíncrono serve para fazermos saída no *template*, mesmo que o valor a ser atribuído demore um pouco para ser retornado.

- Em *exemplo-pipes.component.ts*:

```
import { Component, OnInit } from '@angular/core';
import { Observable } from "rxjs/Observable";

@Component({
  selector: 'app-exemplos-pipes',
  templateUrl: './exemplos-pipes.component.html',
  styleUrls: ['./exemplos-pipes.component.css']
})
export class ExemplosPipesComponent implements OnInit {

  livro: any = {
    titulo: 'Learning JavaScript Data Structures and Algorithms 2nd ed',
    rating: 4.54321,
    numeroPaginas: 314,
    preco: 44.99,
    dataLancamento: new Date(2016, 5, 23),
    url: 'http://a.co/glqjpRP'
  };

  livros: string[] = ['Java', 'Angular2']
  filtro: string;

  addCurso(valor){
    this.livro.push(valor);
    console.log(this.livros);
  }
  constructor() { }

  ngOnInit() {
  }
}
```

```

obterCursos(){
  if (this.livros.length === 0 || this.filtro === null || this.filtro.trim() === '') {
    return this.livros;
  }

  return this.livros.filter((v) => {
    if (v.toLowerCase().indexOf(this.filtro.toLowerCase())>= 0){
      return true;
    }
    return false;
  });
}

// Exemplo com promessa:
valorAsync = new Promise((resolve, reject) => {
  setTimeout(() => resolve('Valor assíncrono'), 2000)
});

// Exemplo com observable:
// valorAsync2 = Observable.interval(2000).map(valor => 'Valor assíncrono 2');
}

```

- Em *exemplo-pipes.component.html*

```

<!-- Depois de 02 segundos, nós objetemos não objeto promessa,
      mas sim o valor que é retornado -->
<p>{{ valorAsync | async}} <p>

<!--<p>{{ valorAsync2 | async}} <p>-->

```

51 VÍDEO #49: ROTAS: INTRODUÇÃO

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video49>.

SPA *Single Page Application*: altera as rotas do projeto, sem dar um *refresh* na aplicação. O Angular 2 lê e identifica e faz o gerenciamento correto qual component pertence aquela rota.

Link útil:

- 1) Documentação sobre rotas disponível em: <https://angular.io/guide/router>.

52 VÍDEO #50: CONFIGURANDO ROTAS SIMPLES

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video50>.

Para utilizar rotas, é necessário criar um arquivo com todas as rotas e uma variável do tipo “Routes”:

- Em *app.routing.ts*:

```
import { CursosComponent } from './cursos/cursos.component';
import { LoginComponent } from './login/login.component';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { ModuleWithProviders } from "@angular/core/src/metadata";

// As rotas são compostas de objetos e nós temos que declará-los;

const APP_ROUTES: Routes = [
  // Path: caminho para um determinado component
  // Se for digitado http://localhost:4200/, o HomeComponent será chamado
  {path: '', component: HomeComponent},
  // Se for digitado http://localhost:4200/login, o LoginComponent será chamado
  {path: 'login', component: LoginComponent},
  // Se for digitado http://localhost:4200/cursos, o CursosComponent será chamado
  {path: 'cursos', component: CursosComponent},
];

export const routing: ModuleWithProviders = RouterModule.forRoot(APP_ROUTES);
// A diferença entre uma rota de raiz e uma rota de funcionalidade, é o forRoot ou o
forChild
```

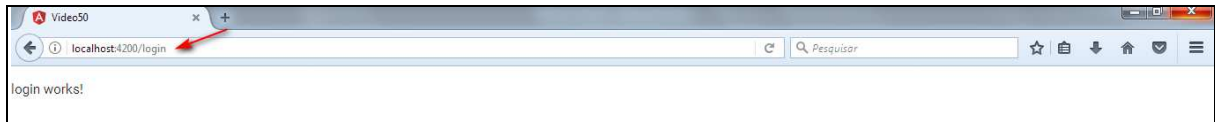
Depois de criado o arquivo de rotas, temos que importar a constante criada:

```
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    LoginComponent,
    CursosComponent
  ],
  imports: [
    NavegadorModule,
    routing //Importação
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Depois de importada, temos que configurar o *app.component.html* para ler essas rotas; isso é feito através da tag `<router-outlet>` que renderiza o component dentro dessa tag:

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

Resultados no navegador:



53 VÍDEO #51: ROTAS ROUTERLINK: DEFININDO ROTAS NO TEMPLATE

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video51>.

O ideal é que na aplicação exista um menu para se navegar entre as rotas.

- Em *app.component.html* defina seu menu e o *link* das rotas:

```
<!-- Para transformar um link em rota, use a diretiva routerLink, passando o caminho da
rota-->
<nav>
  <div class="nav-wrapper">
    <a routerLink="" class="brand-logo right">Home - Rotas NG2</a>
    <ul id="nav-mobile" class="left hide-on-med-and-down">
      <li><a routerLink="/login" > Login</a></li>
      <li><a routerLink="/cursos"> Cursos</a></li>
    </ul>
  </div>
</nav>

<div class="container">
  <router-outlet> </router-outlet>
</div>
```

Resultado no navegador:





54 VÍDEO #52: ROTAS: APLICANDO CSS EM ROTAS ATIVAS

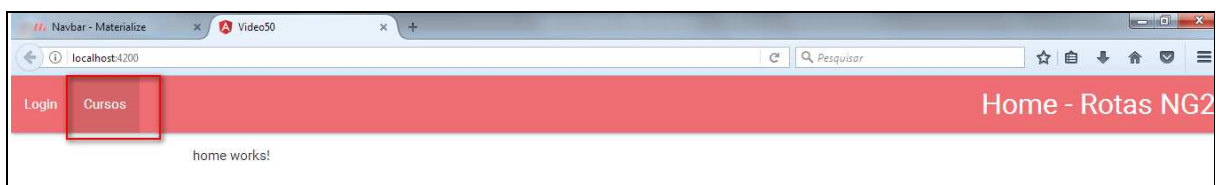
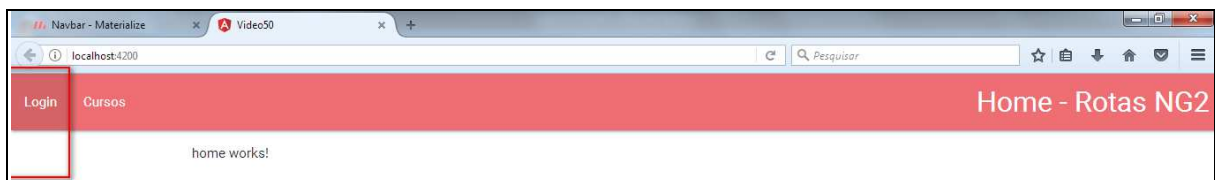
OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video52>.

Para ativar a cor numa rota ativa, use a diretiva *routerLinkActive*:

```
<nav>
  <div class="nav-wrapper">
    <a routerLink="/" class="brand-logo right">Home - Rotas NG2</a>
    <ul id="nav-mobile" class="left hide-on-med-and-down">
      <li routerLinkActive="active"><a routerLink="/login"> Login</a></li>
      <li routerLinkActive="active"><a routerLink="/cursos"> Cursos</a></li>
    </ul>
  </div>
</nav>

<div class="container">
  <router-outlet> </router-outlet>
</div>
```

Resultado no navegador (ao parar o mouse sobre as opções):



55 VÍDEO #53: ROTAS: DEFININDO E EXTRAINDO PARÂMETROS DE ROTEAMENTO

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video53>.

Quando queremos passar parâmetros para a rota, devemos usar a "/" e os ":" + nome do parâmetro. Os ":" indicam que o que foi repassado depois dele, se trata de um parâmetro da rota.

- Em *app.routing.ts* foi definido o parâmetro "id":

```
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from '../home/home.component';
import { ModuleWithProviders } from "@angular/core/src/metadata";
import { CursosComponent } from '../cursos/cursos.component';
import { LoginComponent } from '../login/login.component';
import { CursoDetalheComponent } from '../curso-detalhe/curso-detalhe.component';

const APP_ROUTES: Routes = [
  {path: '', component: HomeComponent},
  {path: 'login', component: LoginComponent},
  {path: 'cursos', component: CursosComponent},
  {path: 'curso/:id', component: CursoDetalheComponent},
];

export const routing: ModuleWithProviders = RouterModule.forRoot(APP_ROUTES);
```

- Em *app.component.html* lemos o ID:

```
<nav>
  <div class="nav-wrapper">
    <a routerLink="" class="brand-logo right">Home - Rotas NG2</a>
    <ul id="nav-mobile" class="left hide-on-med-and-down">
      <li routerLinkActive="active"><a routerLink="/login" > Login</a></li>
      <li routerLinkActive="active"><a routerLink="/cursos"> Cursos</a></li>
      <!--Ao usar o routerLink como property binding, é possível passar parâmetros -->
      <li routerLinkActive="active"><a [routerLink]="['curso', idCurso.value]"> Curso com
ID</a></li>
    </ul>
  </div>
</nav>
<div class="container">
  <p> Entre com o ID do curso: </p>
  <input #idCurso>
  <router-outlet> </router-outlet>
</div>
```

- Em *curso-detalhe.ts* criamos a variável ID:

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from "@angular/router";

@Component({
```

```

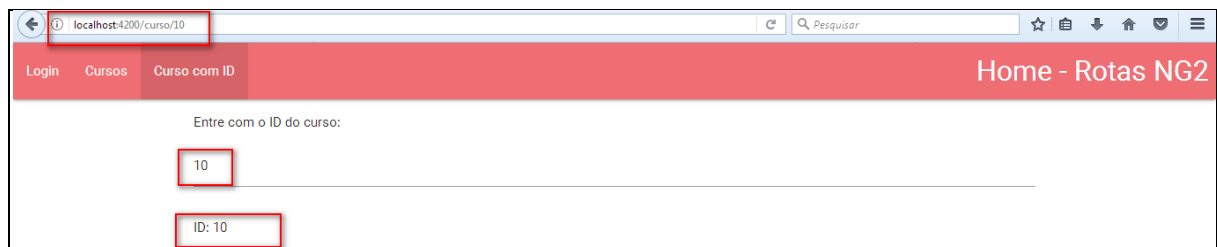
selector: 'app-curso-detalle',
templateUrl: './curso-detalle.component.html',
styleUrls: ['./curso-detalle.component.css']
}))
export class CursoDetalleComponent implements OnInit {

  id: string;

  // É possível utilizar o ActivatedRoute para obter os detalhes da rota
  constructor(private route: ActivatedRoute) {
    this.id = this.route.snapshot.params['id'];
  }
  ngOnInit() {
  }
}

```

Resultado no navegador passando o ID 10:



56 VÍDEO #54: ROTAS: ESCUTANDO MUDANÇAS NOS PARÂMETROS DE ROTEAMENTO

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video54>.

- Em *curso-detalle.component.ts*:

```

import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from "@angular/router";
import { Subscription } from "rxjs/Subscription";

@Component({
  selector: 'app-curso-detalle',
  templateUrl: './curso-detalle.component.html',
  styleUrls: ['./curso-detalle.component.css']
})
export class CursoDetalleComponent implements OnInit {

  id: string;
  inscricao: Subscription;

  constructor(private route: ActivatedRoute) {
    // Com esse construtor, temos uma foto apenas do primeiro parâmetro; se ele é alterado
    // nosso template não altera
    // this.id = this.route.snapshot.params['id'];
    // console.log(this.route);
  }
}

```

```

    }

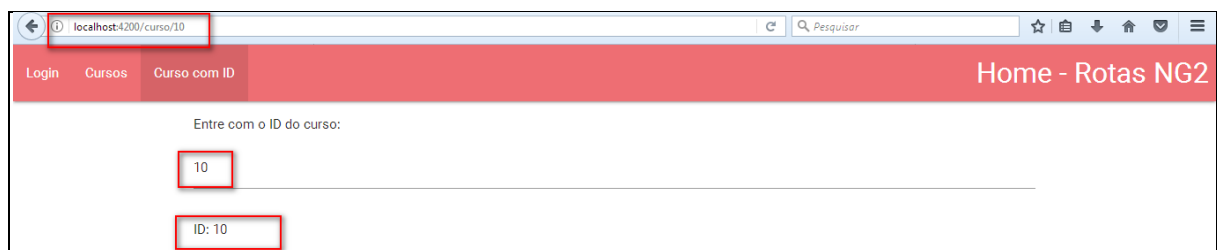
    ngOnInit() {
      // Subscribe: se inscrevendo para receber alterações do parâmetro
      //
      this.inscricao = this.route.params.subscribe((params: any) => {
        this.id = params ['id'];
      });
    }

    // Por boa prática, ao utilizar uma inscrição crie o método para se desinscrever,
    // caso o component seja excluído
    ngOnDestroy(){
      this.inscricao.unsubscribe;
    }
  }
}

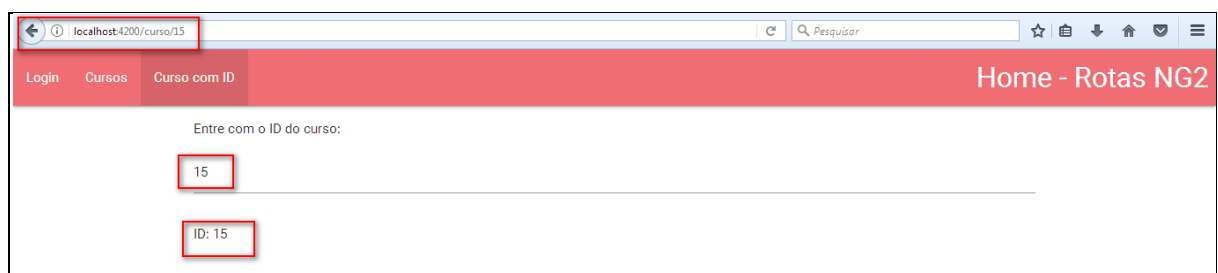
```

Resultado no navegador:

- Passando o ID 10:



- Passando o ID 15:



57 VÍDEO #55: ROTAS IMPERATIVAS: REDIRECIONAMENTO VIA CÓDIGO

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video55>.

Criado um novo serviço chamado “curso”; em *curso.service.ts*, definimos o ID do curso:

```

import { Injectable } from '@angular/core';

@Injectable()

```

```
export class CursosService {

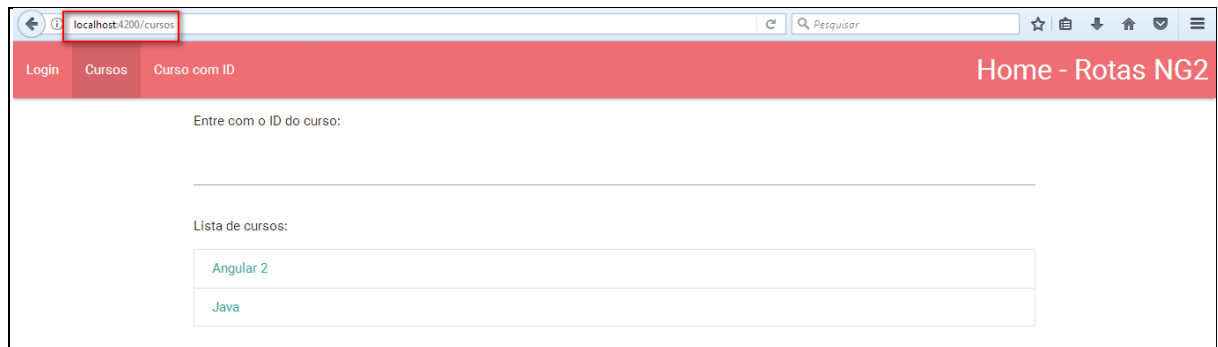
  getCursos(){
    return [
      {id: 1, nome: 'Angular 2'},
      {id: 2, nome: 'Java'}
    ];
  }
  constructor() { }
}
```

Em *cursos.component.html*:

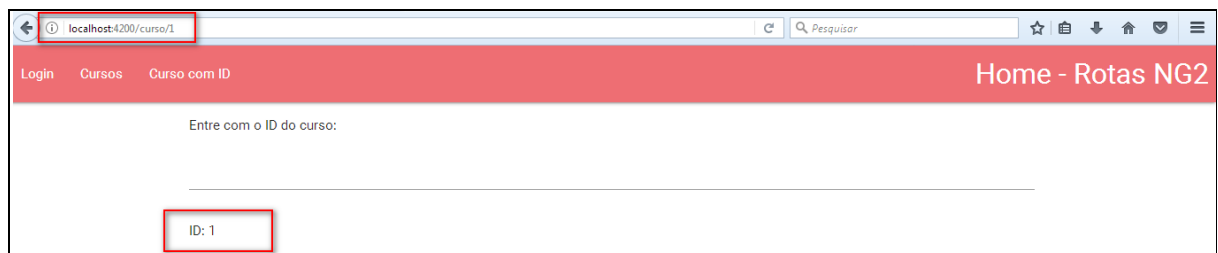
```
<p>
  Lista de cursos:
</p>

<!-- Passando o routerLink com o endereço da rota e o parâmetro: -->
<div class="collection">
  <a class="collection-item" *ngFor="let curso of cursos"
    [routerLink]="['/curso', curso.id]">
    {{ curso.nome }}</a>
</div>
```

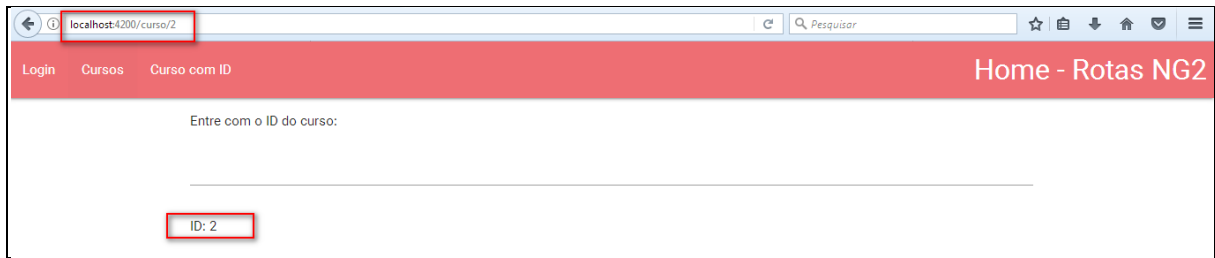
No navegador, o resultado é:



- Ao clicar em Angular 2, a app é redirecionada:



- Ao clicar em Java, a app também é redirecionada:



Para fazer o roteamento imperativo, usamos a classe *Router*:

- Em *curso-detalle.component.ts*:

```
import { CursosService } from '../cursos/cursos.service';
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from "@angular/router";
import { Subscription } from "rxjs/Subscription";

@Component({
  selector: 'app-curso-detalle',
  templateUrl: './curso-detalle.component.html',
  styleUrls: ['./curso-detalle.component.css']
})
export class CursoDetalleComponent implements OnInit {

  id: number;
  inscricao: Subscription;
  curso: any;

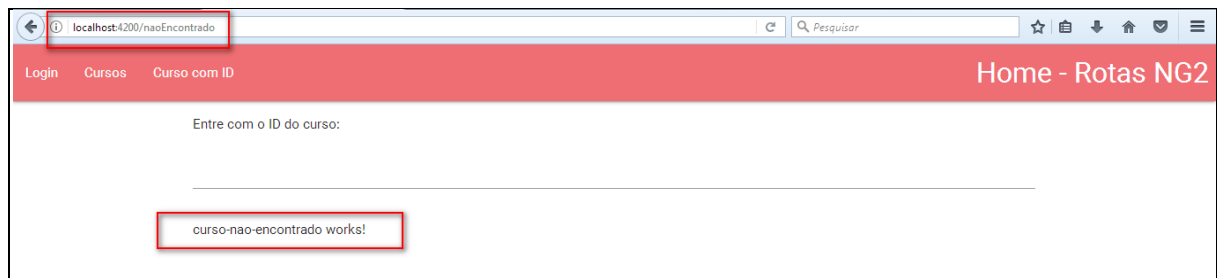
  // Existe a classe Router do Angular 2, que contém todos os métodos responsáveis pelo
  // redirecionamento:
  constructor(private route: ActivatedRoute, private cursosService: CursosService, private
router: Router) {

  }

  ngOnInit() {
    this.inscricao = this.route.params.subscribe((params: any) => {
      this.id = params['id'];
      this.curso = this.cursosService.getCurso(this.id);
      if(this.curso == null){
        // Através do "navigate" indicamos qual rota deve ser seguida:
        this.router.navigate(['/naoEncontrado']);
      }
    });
  }

  ngOnDestroy(){
    this.inscricao.unsubscribe;
  }
}
```


No navegador, ao informar um ID de curso que não existe, a rota é redirecionada corretamente:



58 VÍDEO #56: ROTAS: DEFININDO E EXTRAINDO PARÂMETRO DE URL (QUERY)

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video56>.

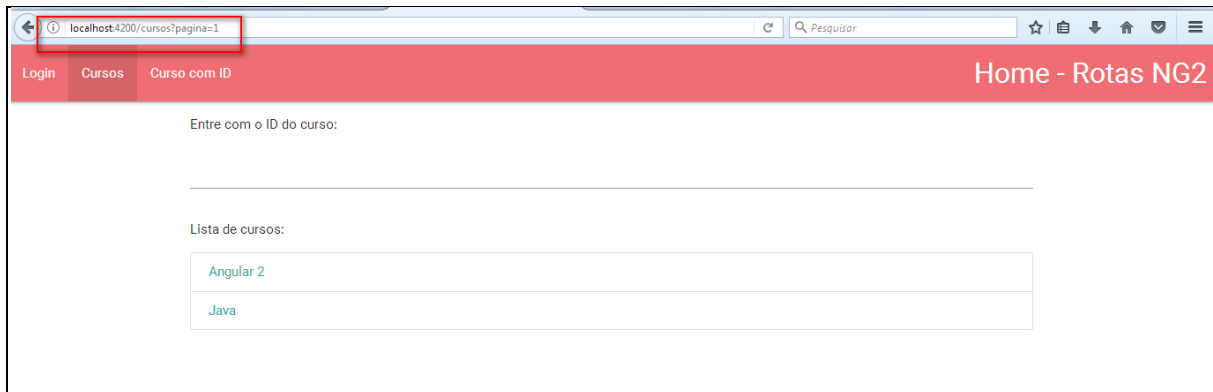
Parâmetro de *Query* é um parâmetro da *URL*.

- Em *app.component.html*:

```
<!-- No Angular 2 existe a diretiva queryParams que usamos para passar e extrair
parâmetros -->
<nav>
  <div class="nav-wrapper">
    <a routerLink="" class="brand-logo right">Home - Rotas NG2</a>
    <ul id="nav-mobile" class="left hide-on-med-and-down">
      <li routerLinkActive="active"><a routerLink="/login" > Login</a></li>
      <li routerLinkActive="active"><a routerLink="/cursos"[queryParams]="{pagina:1}">
        Cursos</a></li>
      <li routerLinkActive="active"><a [routerLink]="['curso', idCurso.value]">
        Curso com ID</a></li>
    </ul>
  </div>
</nav>

<div class="container">
  <p> Entre com o ID do curso: </p>
  <input #idCurso>
  <router-outlet> </router-outlet>
</div>
```

No navegador, o resultado ficou assim:



59 VÍDEO #57: ROTAS: CRIANDO UM MÓDULO DE ROTAS

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video57>.

O objetivo de se criar módulo de rota, é deixar o projeto mais organizado.

Na pasta de app, foi criado um novo arquivo chamado *app.routing.module.ts* e nesse arquivo, as rotas do projeto foram configuradas:

```
import { AppModule } from './app.module';
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { ModuleWithProviders } from "@angular/core/src/metadata";
import { CursosComponent } from './cursos/cursos.component';
import { LoginComponent } from './login/login.component';
import { CursoDetalheComponent } from './curso-detalhe/curso-detalhe.component';
import { CursoNaoEncontradoComponent } from './curso-nao-encontrado/curso-nao-encontrado.component';

const appRoutes: Routes = [
  {path: '', component: HomeComponent},
  {path: 'login', component: LoginComponent},
  {path: 'cursos', component: CursosComponent},
  {path: 'curso/:id', component: CursoDetalheComponent},
  {path: 'naoEncontrado', component: CursoNaoEncontradoComponent}
];

// Ao invés de termos uma constante exportando a classe, temos o módulo com o imports e exports:
// export const routing: ModuleWithProviders = RouterModule.forRoot(APP_ROUTES);
@NgModule({
  imports: [RouterModule.forRoot(appRoutes)],
  exports: [RouterModule]
})
export class AppRoutingModule{
}
```

O módulo de rotas criado, deve ser importado no *app.module.ts*:

```
import { NavegadorModule } from '@angular/platform-navegador';
import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
import { LoginComponent } from './login/login.component';
import { CursosComponent } from './cursos/cursos.component';
//import { routing } from './app.routing';
import { CursoDetalheComponent } from './curso-detalhe/curso-detalhe.component';
import { CursosService } from './cursos/cursos.service';
import { CursoNaoEncontradoComponent } from './curso-nao-encontrado/curso-nao-encontrado.component';
import { AppRoutingModuleModule } from './app.routing.module';

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    LoginComponent,
    CursosComponent,
    CursoDetalheComponent,
    CursoNaoEncontradoComponent
  ],
  imports: [
    NavegadorModule,
    AppRoutingModuleModule // Importando o módulo
    //routing
  ],
  providers: [CursosService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

60 VÍDEO #58: CRIANDO UM MÓDULO DE FUNCIONALIDADE

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video58>.

O objetivo de se criar um módulo de funcionalidade, é apenas para não deixar o *app.module.ts* muito extenso e com uma manutenção muito difícil.

Foi criado um novo arquivo chamado *cursos-module.ts*:

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

import { CursoNaoEncontradoComponent } from './curso-nao-encontrado/curso-nao-encontrado.component';
import { CursoDetalheComponent } from './curso-detalhe/curso-detalhe.component';
import { CursosComponent } from './cursos.component';
```

```

import { CommonModule } from '@angular/common/src/common';
import { RouterModule } from '@angular/router';
import { CursosService } from './cursos.service';

// Algo importante sobre o módulo de funcionalidade, é que nos imports devemos colocar os
// imports do Angular 2 que vamos utilizar
@NgModule({
  imports: [
    // NavegadorModule: O NavegadorModule não deve ser utilizado dentro de um módulo de
    // Funcionalidade. Em seu lugar deve ser usado o CommonModule
    CommonModule,
    RouterModule
  ],
  exports: [],
  declarations: [
    CursosComponent,
    CursoDetalheComponent,
    CursoNaoEncontradoComponent
  ],
  providers: [
    CursosService
  ]
})

export class CursosModule {}

```

61 VÍDEO #59: ROTAS: CRIANDO UM MÓDULO DE FUNCIONALIDADE

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video59>.

Para criar um módulo de funcionalidade de rotas, basta criar um arquivo com as rotas e ao invés de utilizar o *forRoot*, deve ser utilizado o *forChild*:

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';

import { CursosComponent } from './cursos.component';
import { CursoNaoEncontradoComponent } from './curso-nao-encontrado/curso-nao-encontrado.component';
import { CursoDetalheComponent } from './curso-detalhe/curso-detalhe.component';

const cursosRoutes: Routes = [
  { path: 'cursos', component: CursosComponent },
  { path: 'curso/:id', component: CursoDetalheComponent },
  { path: 'naoEncontrado', component: CursoNaoEncontradoComponent }
];

@NgModule({
  // Como esse módulo é de funcionalidade, não utilizamos o forRoot; utilizamos o
  // forChild
  imports: [RouterModule.forChild(cursosRoutes)],

```

```

    exports: [RouterModule]
  })
  export class CursosRoutingModule{
  }

```

62 VÍDEO #60: ROTAS FILHAS

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video60>.

Rotas filhas podem ser utilizadas para evitar a repetição de rotas comuns e para renderizar tanto o componente Pai quanto o componente Filho no mesmo momento.

- Rotas comuns: Note que “alunos” está se repetindo em todas as rotas.

```

const alunosRoutes = [
  // Para evitar colisão de rotas, informe a rotas na ordem que elas devem ser
  // executadas.
  {path: 'alunos', component: AlunosComponent},
  {path: 'alunos/novo', component: AlunoFormComponent},
  {path: 'alunos/id', component: AlunoDetalheComponent},
  {path: 'alunos/:id/editar', component: AlunoFormComponent}
]

```

- Rotas filhas:

```

import { RouterModule } from '@angular/router';
import { NgModule } from '@angular/core';

import { AlunosComponent } from './alunos.component';
import { AlunoDetalheComponent } from './aluno-detalle/aluno-detalle.component';
import { AlunoFormComponent } from './aluno-form/aluno-form.component';

const alunosRoutes = [
  // Para evitar colisão de rotas, informe a rotas na ordem que elas devem ser
  // executadas.
  // Para criar uma rota filha, deve ser usado o "children"
  {path: 'alunos', component: AlunosComponent, children: [
    {path: '/novo', component: AlunoFormComponent},
    {path: '/id', component: AlunoDetalheComponent},
    {path: '/:id/editar', component: AlunoFormComponent} ]},
]

@NgModule({
  imports: [RouterModule.forChild(alunosRoutes)],
  exports: [RouterModule]
})
export class AlunosRoutingModule {
}

```

- Em *alunos.component.ts*, deve ser passado o `<router-outlet>` para que as rotas filhas sejam executadas:

```
<p>
  alunos works!
</p>

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

63 VÍDEO #61: ROTAS FILHAS: DESENVOLVENDO AS TELAS

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video61>.

Projeto não funcionando corretamente.

64 VÍDEO #62: ROTAS: DICA DE PERFORMANCE: CARREGAMENTO SOB DEMANA (*LAZY LOADING*)

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video62>.

Ao carregar uma página, é realizado o *download* de todos os arquivos que serão utilizados na aplicação; então quanto mais arquivos existirem, maior será o tempo para que a página seja redenzada.

As mudanças principais serão no *app.routing.module.ts*. Sempre que você fizer mudanças no carregamento dinâmico, pare o *ng serve* (caso ele esteja rodando) e execute-o novamente, após as alterações.

Outro detalhe importante, é que o módulo que você implementou o *loadChildren* não pode ser importado em nenhum outro lugar da aplicação.

No arquivo de roteamento do seu módulo, deixe vazio o caminho principal.

- Em *app.routing.module.ts*:

```
import { AppModule } from './app.module';
import { NgModule } from '@angular/core';

import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { ModuleWithProviders } from '@angular/core/src/metadata';
import { LoginComponent } from './login/login.component';
import { CursosModule } from './cursos/cursos.module';

const appRoutes: Routes = [
```

```

    // Primeiro passo: Inclua um novo path com o módulo que deve ser executado
    // Use o loadChildren (que significa Carregar rotas filhas) e passe o caminho completo
    // do MÓDULO + a classe que tem o módulo de funcionalidade
    { path : 'cursos', loadChildren: 'app/cursos/cursos.module#CursosModule'},
    { path: '', component: HomeComponent },
    { path: 'login', component: LoginComponent }
  ];
}

@NgModule({
  imports: [RouterModule.forRoot(appRoutes)],
  exports: [RouterModule]
})
export class AppRoutingModule{
}

```

- Em *cursos.routing.module.ts*:

```

import { NavegadorModule } from '@angular/platform-navegador';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { HomeComponent } from './home/home.component';
import { LoginComponent } from './login/login.component';
import { AppRoutingModule } from './app.routing.module';
import { AlunosModule } from './alunos/alunos.module';

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    LoginComponent
  ],
  // Segundo passo: Remova todos os imports do módulo que utilizará o loadChildren
  imports: [
    NavegadorModule,
    AppRoutingModule,
    AlunosModule
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }

```

- Em *cursos.routing.module.ts*:

```

import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { CursosComponent } from './cursos.component';
import { CursoNaoEncontradoComponent } from './curso-nao-encontrado/curso-nao-encontrado.component';
import { CursoDetalheComponent } from './curso-detalle/curso-detalle.component';

const cursosRoutes: Routes = [
  // Terceiro passo: Deixe vazio o caminho principal do seu módulo, pois ele já foi

```

```

    informado completamente no app.routing.module.ts
    { path: '', component: CursosComponent },
    { path: 'curso/:id', component: CursoDetalheComponent },
    { path: 'naoEncontrado', component: CursoNaoEncontradoComponent }
  ];

  @NgModule({
    imports: [RouterModule.forChild(cursosRoutes)],
    exports: [RouterModule]
  })
  export class CursosRoutingModule{
  }

```

OBS: Sempre que você for utilizar *loadChildren*, mantenha um padrão para as classes.

65 VÍDEO #63: ROTAS: TELA DE *LOGIN* E COMO NÃO MOSTAR O MENU (*NAVBAR*)

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video63>.

- Em *login.component.html*, foram criados os campos Usuário e Senha e o botão Entrar:

```

<h5> Login </h5>
<div class="row">
  <div class="input-field col s12">
    <input [(ng-model)]="usuario.nome" id="usuario" type="text" class="validate">
    <label class="active" for="usuario">Usuário</label>
  </div>
</div>
<div class="row">
  <div class="input-field col s12">
    <input [(ng-model)]="usuario.senha" id="senha" type="password" class="validate">
    <label class="active" for="senha">Senha</label>
  </div>
</div>
<button class="btn waves-effect waves-light" type="submit" name="action"
  (click)="fazerLogin()">Login
  <i class="material-icons right">Entrar</i>
</button>

```

- Criado o serviço *auth.service.ts* para autenticar o usuário e para mostrar o menu apenas se o usuário está autenticado:

```

import { Router } from '@angular/router';
import { Usuario } from '../usuario';
import { Injectable } from '@angular/core';
import { EventEmitter } from "events";

@Injectable()

```



```

export class AuthService {
  private usuarioAutenticado: boolean = false;
  mostrarmenuEmitter = new EventEmitter<boolean>();

  constructor(private router: Router) { }

  // Método para autenticar o usuário:
  fazerLogin(usuario: Usuario){
    if(usuario.nome === 'usuario@email.com' && usuario.senha === '123456'){
      this.usuarioAutenticado = true;
      this.mostrarmenuEmitter.emit(true);
      this.router.navigate(['/']);
    } else{
      this.usuarioAutenticado = false;
      this.mostrarmenuEmitter.emit(false);
    }
  }
}

```

- Em *app.component.ts*, criamos o método para mostrar o menu:

```

import { AuthService } from '../login/auth.service';
import { Component } from '@angular/core';

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

  mostrarMenu: boolean = false;
  constructor(private authService: AuthService){
  }

  ngOnInit(){
    this.authService.mostrarmenuEmitter.subscribe(
      mostrar => this.mostrarMenu = mostrar);
  }
}

```

- E no *app.component.html* foi adicionada a diretiva *ngIf* para apresentar o menu:

```

<nav *ngIf="mostrarMenu">
  <div class="nav-wrapper">
    <a routerLink="" class="brand-logo right">Home - Rotas NG2</a>
    <ul id="nav-mobile" class="left hide-on-med-and-down">
      <li routerLinkActive="active"><a routerLink="/login" > Login</a></li>
      <li routerLinkActive="active"><a routerLink="/cursos"[queryParams]="{pagina:1}">
Cursos</a></li>
      <li routerLinkActive="active"><a [routerLink]="/alunos"> Alunos</a></li>

```

```

    </ul>
  </div>
</nav>

<div class="container">
  <router-outlet> </router-outlet>
</div>

```

67 VÍDEO #64: USANDO GUARDA DE ROTAS *CANACTIVATE*

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video64>.

A ideia é não deixar o usuário acessar os menus, caso ele não esteja autenticado. Guarda de rota é um tipo de serviço que implementa um determinado método, que o Angular reconhece que esse método será utilizado como guarda de rota.

O que indica se um serviço é mesmo um guarda de rota, é a implementação de *CanActivate*:

```

@Injectable()
export class AuthGuard implements CanActivate {

  constructor(private authService: AuthService, private router: Router) { }

  canActivate(
    // Recebendo a rota
    route: ActivatedRouteSnapshot,
    // Recebendo o estado da rota
    state: RouterStateSnapshot) : Observable<boolean> | boolean {
    // Se o usuário estiver autenticado, libera a rota
    if (this.authService.usuarioEstaAutenticado()){
      return true;
    }
    // Senão traz a rota do login
    this.router.navigate(['/login']);
    return false;
  }
}

```

- Aplicando a guarda de rota em *app.routing.module.ts*:

```

const appRoutes: Routes = [
  { path: 'cursos',
    loadChildren: 'app/cursos/cursos.module#CursosModule',
    canActivate: [AuthGuard]
  },
  { path: 'alunos',
    loadChildren: 'app/alunos/alunos.module#AlunosModule',

```

```

    canActivate: [AuthGuard]
  },
  { path: '', component: HomeComponent,
    canActivate: [AuthGuard]
  },
  { path: 'login', component: LoginComponent }
];

```

Como o *AuthGuard* é do tipo *@Injectable*, precisamos injetar ele no *app.module.ts*, para que ele fique disponível para toda a aplicação:

```

providers: [AuthService, AuthGuard],

```

68 VÍDEO #65: USANDO GUARDA DE ROTAS *CANACTIVATECHILD*

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video65>.

Ao criar um projeto, você pode criar um guarda de rota filha genérico ou um para cada módulo. Esse guarda de rota filha deve ser usado para controlar o acesso a determinadas partes da aplicação, por exemplo, um usuário só pode acessar o menu de consultas, mas não pode acessar um menu de inclusão.

Se você quiser que o guarda de rotas seja executado para todas as rotas, inclusive para o component Pai, então você declara o guarda de rotas dentro do *app.routing.module.ts*; se você quiser apenas as rotas filhas, não incluindo o componente pai, então você coloca o guarda dentro do módulo de rotas do seu módulo.

A forma de se criar um guarda de rotas filhas, é semelhante à criação de guarda de rotas; a diferença é que usamos o *CanActivateChild*.

69 VÍDEO #66: USANDO GUARDA DE ROTAS *CANDEACTIVATECHILD*

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video66>.

Guarda de rota para verificar se um usuário pode desativar a rota.

- Em *alunos-desactivate.guard.ts*:

```

import { Observable } from 'rxjs/Observable';

```

```

import { AlunoFormComponent } from '../alunos/aluno-form/aluno-form.component';
import { Injectable, Component } from '@angular/core';
import { CanDeactivate } from "@angular/router/src";
import { ActivatedRouteSnapshot, RouterStateSnapshot } from "@angular/router/src";

// Quando utilizado um guarda para desativar uma rota, é necessário
// especificar qual component será desativado
@Injectable()
export class AlunosDesactivateGuard implements CanDeactivate <AlunoFormComponent> {

  canDeactivate(
    component: AlunoFormComponent,
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot):
    Observable<boolean>|boolean{
      console.log('guarda de desativação');
      return true;
    }
}

```

Você deve declarar a guarda criada nos *providers* (no *app.module.ts* ou no modulo específico da sua aplicação).

- Em *alunos.module.ts*:

```

import { AlunosDesactivateGuard } from '../guards/alunos-desactivate.guard';
import { AlunosService } from './alunos.service';
import { CommonModule } from '@angular/common';
import { NgModule } from '@angular/core';
import { AlunosComponent } from './alunos.component';
import { AlunoFormComponent } from '../aluno-form/aluno-form.component';
import { AlunosRoutingModule } from './alunos.routing.module';

@NgModule({
  exports: [],
  imports: [
    CommonModule,
    AlunosRoutingModule
  ],
  declarations: [AlunosComponent, AlunoFormComponent],
  providers: [AlunosService, AlunosDesactivateGuard]
})

export class AlunosModule {
}

```

Para que o guarda de rota seja verificado, deve ser criado um novo atributo com o guarda de rota:

- Em *alunos.routing.module.ts*:

```
const alunosRoutes = [
  {path: '', component: AlunosComponent, children: [
    {path: '/novo', component: AlunoFormComponent},
    {path: '/id', component: AlunoDetalheComponent},
    {path: '/:id/editar', component: AlunoFormComponent,
      canActivate: [AlunosDesactivateGuard]} ]}, // Atribuindo o guarda de rotas
]
```

70 VÍDEO #67: USANDO GUARDA DE ROTAS: CANDEACTIVATE COM INTERFACE GENÉRICA

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video67>.

Para utilizar o guarda de rotas de desativação de forma genérica, é necessário uma interface:

- Em *iform-candeactivate.ts*:

```
// Na interface, apenas declaramos os métodos que a interface vai implementar
export interface IFormCanDeactivate{
  podeDesativar();
}
```

- Em *alunos-desactivate.guards.ts*:

```
// Ao passar a interface, todas as classes do nosso projeto que implementarem
// a interface, passarão por aquele método
@Injectable()
export class AlunosDesactivateGuard implements CanDeactivate <IFormCanDeactivate> {

  canActivate(
    component: IFormCanDeactivate,
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot):
    Observable<boolean>|boolean{
    console.log('guarda de desativação');
    //return component.podeMudarRota();
    return component.podeDesativar();
  }
}
```

- Em *aluno-form.component.ts*, informamos a interface criada e implementamos a lógica do método:

```
@Component({
  selector: 'app-aluno-form',
  templateUrl: './aluno-form.component.html',
  styleUrls: ['./aluno-form.component.css']
})
export class AlunoFormComponent implements OnInit, IFormCanDeactivate {
```

```

aluno: any = {};
inscricao: Subscription;
private formMudou: boolean = false;

constructor(private route: ActivatedRoute, private alunosService: AlunosService) { }

ngOnInit() {
  this.inscricao = this.route.params.subscribe((params: any) => {
    let id = params['id'];
    this.aluno = this.alunosService.getAluno(id);

    if (this.aluno === null){
      this.aluno = {};
    }
  });
}

ngOnDestroy(){
  this.inscricao.unsubscribe();
}

onInput(){
  this.formMudou = true;
  console.log('Mudou');
}

podeMudarRota(){
  if(this.formMudou){
    confirm('Tem certeza que deseja sair dessa página?');
  }
  return true;
}

podeDesativar(){
  this.podeMudarRota();
}
}

```

71 VÍDEO #68: USANDO GUARDA DE ROTAS: RESOLVE: CARREGANDO DADOS ANTES DA ROTA SER ATIVADA

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video68>.

A diferença entre se usar o *Resolver* é mais visível em aplicações *AJAX*, onde existe uma demora na resposta do servidor.

Criada uma classe do tipo *Resolve* e implementado qual atributo queremos que seja carregado antes da rota ser ativada (no caso, o atributo ID):

```

@Injectables()
export class AlunoDetalheResolver implements Resolve<Aluno> {
  constructor(private alunosService: AlunosService) {}
  resolve(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): Observable<any>|Promise<any>|any {
    console.log('AlunoDetalheResolver');
    let id = route.params['id'];
    return this.alunosService.getAluno(id);
  }
}

```

- Em *aluno.module.ts*, o Resolver também foi declarado como *providers*:

```

providers: [AlunosService, AlunosDesactivateGuard, AlunoDetalheResolver]

```

72 VÍDEO #69: USANDO GUARDA DE ROTAS: CANLOAD: COMO NÃO CARREGAR MÓDULO SEM PERMISSÃO

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video69>.

Mesmo o usuário sem permissão para o usuário carregar uma rota, o *navegador* ainda pode deixar na memória o arquivo; para evitar que isso ocorra, existe a diretiva *CanLoad*.

- Em *auth.guard.ts*, criamos o *CanLoad* e implementamos o método *verificarAcesso*:

```

canActivate(
  route: ActivatedRouteSnapshot,
  state: RouterStateSnapshot) : Observable<boolean> | boolean {
  console.log('AuthGuard');
  return this.verificarAcesso();
}

private verificarAcesso(){
  // Se o usuário estiver autenticado, libera a rota
  if (this.authService.usuarioEstaAutenticado()){
    return true;
  }
  this.router.navigate(['/login']);
  return false;
}

canLoad(route: Route): Observable<boolean>|Promise<boolean>|boolean {
  console.log('Verificando se o usuário pode carregar o módulo');
  return this.verificarAcesso();
}
}

```

- Em `app.routing.module.ts`, passamos o `canLoad: [AuthGuard]` também:

```
const appRoutes: Routes = [
  { path: 'cursos',
    loadChildren: 'app/cursos/cursos.module#CursosModule',
    canActivate: [AuthGuard],
    canActivateChild: [CursosGuard],
    canLoad: [AuthGuard]
  },
  { path: 'alunos',
    loadChildren: 'app/alunos/alunos.module#AlunosModule',
    canActivate: [AuthGuard],
    canLoad: [AuthGuard]
  },
],
```

73 VÍDEO #70: DEFININDO ROTA PADRÃO E WILDCARD (ROTA NÃO ENCONTRADA)

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video70>.

Caso um *component* não seja configurado para uma rota não encontrada, ocorrerá erro no *console*.

- Criamos um novo component chamado pagina-não-encontrada:

```
@Component({
  selector: 'app-pagina-nao-encontrada',
  templateUrl: './pagina-nao-encontrada.component.html',
  styleUrls: ['./pagina-nao-encontrada.component.css']
})
export class PaginaNaoEncontradaComponent implements OnInit {

  constructor() { }

  ngOnInit() {
  }

}
```

- Em `app.routing.module.ts`, configuramos o *component*:

```
const appRoutes: Routes = [
  { path: 'cursos',
    loadChildren: 'app/cursos/cursos.module#CursosModule',
    canActivate: [AuthGuard],
    canActivateChild: [CursosGuard],
    canLoad: [AuthGuard]
  },
  { path: 'alunos',
    loadChildren: 'app/alunos/alunos.module#AlunosModule',
```



```

    canActivate: [AuthGuard],
    canLoad: [AuthGuard]
  },
  { path: 'login', component: LoginComponent },
  // Sempre crie no seu arquivo de rotas, um caminho 'vazio'
  { path: '', component: HomeComponent,
    canActivate: [AuthGuard]
  },
  // E um caminho para rotas não encontrada. Aqui no caso, qualquer coisa (**) diferente
  // das rotinas acima cairá nesse component
  { path: '**', component: PaginaNaoEncontradaComponent
  },
];

```

Para definir a rota padrão, use o *redirectTo*:

```

{ path: 'home', component: HomeComponent,
  canActivate: [AuthGuard]
},
// Para direcionar sua rota, use o redirectTo
{ path: '', redirectTo: '/home', pathMatch: 'full'},

```

Links úteis:

- 1) Documentação para *pathMatch Full* ou *pathMatch Prefix*:
- a) <https://angular.io/guide/router#redirecting-routes>
- b) <http://vsavkin.tumblr.com/post/146722301646/angular-router-empty-paths-componentless-routes>.

74 VÍDEO #71: ROTAS: ESTILO DE URL: HTML 5 OU

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video71>.

Não utilizar a “#” é o padrão de roteamento do *HTML 5*, porém ao começarmos a trabalhar com algum *backend*, pode ser que a linguagem do *backend* não aceite o padrão e o contêiner/servidor não vai conseguir reconhecer o *link* e não vai saber quando você está tentando acessar um roteamento ou quando você está tentando fazer uma chamada *AJAX*.

Para configurar a “#”, basta informar no meu arquivo de rotas o `{useHash : true}`:

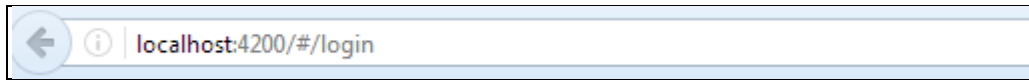
- Em *app.routing.module.ts*:

```

@NgModule({
  imports: [RouterModule.forRoot(appRoutes, {useHash: true})],

```

No navegador será inserido a “#”:



75 VÍDEO #72: FORMULÁRIOS (TEMPLATES VS DATA/REATIVO) INTRODUÇÃO

Diferenças entre Template *Driver* e Template Data *Driver* (Reativo):

Template Driven	Data Driven (Reativo)
Formulário é criado e configurado no HTML	Formulário é criado e configurado no Componente
Validações são feitas no template HTML	Validações são feitas no Component
Angular cria/deduz um FormGroup do cod HTML	Angular usa o FormGroup criado no Component
Valores do form são submetidos com ngSubmit	Form já está no Component e não precisa do ngSubmit

Imagem retirada do vídeo

76 VÍDEO #73: FORMULÁRIOS - CRIANDO O PROJETO INICIAL COM BOOTSTRAP 3

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video73>.

OBS²: Ao criar esse projeto, use o “—routing” para que o arquivo de rotas também seja criado> ng g NOMEDOPROJETO —routing.

OBS³: Verifique se o *FormsModule* está importado no módulo do seu *component*; caso não esteja, importe-o:

```
import { FormsModule } from '@angular/forms';
```

```
imports: [
  NavegadorModule,
  AppRoutingModuleModule,
  FormsModule
],
```

Para instalar o *Bootstrap*, no *Prompt* de comando, use o seguinte comando:

- *npm install ngx-bootstrap bootstrap --save*

Depois em “*styles*” do arquivo *angular-cli.json*, adicione a linha:

“*../node_modules/bootstrap/dist/css/bootstrap.min.css*”

```
"styles": [
  "../node_modules/bootstrap/dist/css/bootstrap.min.css",
  "styles.css"
],
```

Adicionamos dois novos *components*: o *template-form* e o *data-form*. No *app.component.html*, criamos uma barra de navegação para esses dois *components*:

```
<nav class="navbar navbar-default">
  <div class="container-fluid">
    <div class="navbar-header">
      <a class="navbar-brand" routerLink="">Angular Forms</a>
    </div>

    <ul class="nav navbar-nav">
      <li routerLinkActive="active"><a routerLink="/templateForm"> Form template </a></li>
      <li routerLinkActive="active"><a routerLink="/dataForm"> Data template </a></li>
    </ul>
  </div>
</nav>

<div class="container">
  <router-outlet></router-outlet>
</div>
```

- Em *app.routing.module.ts*, adicionamos as rotas para esses *components*:

```
const routes: Routes = [
  { path: 'templateForm', component: TemplateFormComponent },
  { path: 'dataForm', component: DataFormComponent },
  { path: '', pathMatch: 'full', redirectTo: 'templateForm' },
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
```

```

    exports: [RouterModule]
  })
  export class AppRoutingModule { }

```

- Em *template-form.component.html*, criamos dois novos campos:

```

<form>
<div class="form-group">
  <label for="nome">Nome</label>
  <input type="text" class="form-control" id="nome" placeholder="nome">
</div>

<div class="form-group">
  <label for="email">Email</label>
  <input type="email" class="form-control" id="email" placeholder="nome@email.com">
</div>

  <button type="submit" class="btn btn-primary">Submit</button>
</form>

```

No navegador, o resultado é:

The screenshot shows a web browser window at localhost:4200/templateForm. The browser has tabs for 'Angular Forms', 'Form template' (selected), and 'Data template'. The 'Form template' tab displays a form with two input fields: 'Nome' with the placeholder 'nome' and 'Email' with the placeholder 'nome@email.com'. Below the fields is a blue 'Submit' button.

77 VÍDEO #74: FORMS (TEMPLATE DRIVEN) CONTROLES NGFORM, NGSUBMIT E NGMODEL

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video74>.

É através da diretiva *ngForm* que informamos o Angular que aquele arquivo se trata de um formulário e através da diretiva *ngModel* associada a um *input*, informamos o Angular quais valores serão passados para o formulário.

- Em *template-form.component.html*:

```

<!-- O ngSubmit indica o objeto -->
<form #f="ngForm" (ngSubmit)="onSubmit(f)" >
<div class="form-group">
  <label for="nome">Nome</label>

```

```

    <input type="text" class="form-control" name="name" id="nome" placeholder="Nome"
      ngModel>
  </div>

  <div class="form-group">
    <label for="email">Email</label>
    <input type="email" class="form-control" name="email" id="email"
      placeholder="nome@email.com" ngModel>
  </div>

  <button type="submit" class="btn btn-primary">Submit</button>
</form>

```

- Em *template-form.component.ts* criamos o método *onSubmit*:

```

onSubmit(form){
  console.log(form);
}

```

78 VÍDEO #75: FORMS (TEMPLATE DRIVEN) INICIANDO VALORES COM NGMODEL

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video75>.

Para iniciar os valores com *ngModel*, basta criar um objeto com os valores desejados e utilizar o *property binding*.

- Em *template-form.component.ts* criamos o objeto “usuário”:

```

usuario: any = {
  nome: 'Natalia',
  email: 'natalia@email.com'
}

```

- E em *template-form.component.html* utilizamos o *property binding*:

```

<form #f="ngForm" (ngSubmit)="onSubmit(f)" >
  <div class="form-group">
    <label for="nome">Nome</label>
    <input type="text" class="form-control" name="name" id="nome"
      placeholder="Nome" [ngModel]="usuario.nome">
    <!-- Se você quiser, também pode usar o two-way data binding, ao invés de apenas o
      property binding: [(ngModel)="usuario.email"] -->
  </div>

```

```

<div class="form-group">
  <label for="email">Email</label>
  <input type="email" class="form-control" name="email" id="email"
    placeholder="nome@email.com" [(ngModel="usuario.email")]>
</div>

<button type="submit" class="btn btn-primary">Submit</button>
</form>

```

79 VÍDEO #76: FORMS (TEMPLATE DRIVEN) MÓDULOS E FORMSMODULE

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video76>.

Independente de como é o formulário, ao trabalhar com módulos, sempre temos que importar o *FormsModule*, senão haverá erro na nossa aplicação.

Foi criado um novo modulo chamado *template-form* e o *component TemplateFormComponent* foi importado para esse módulo.

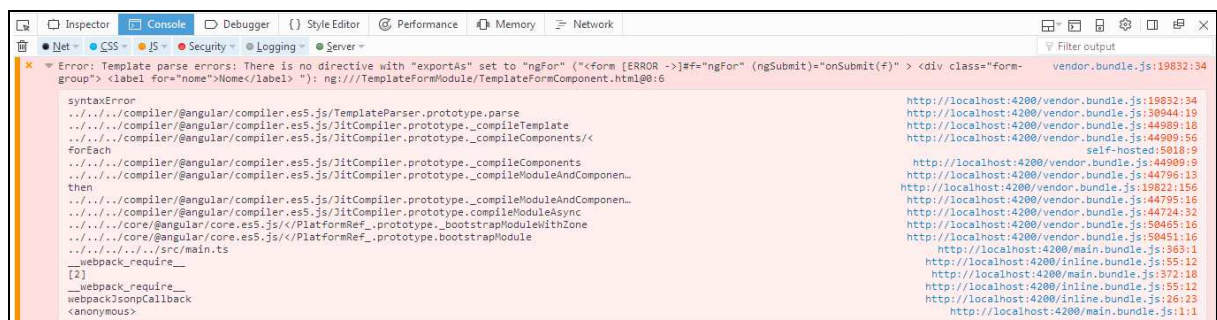
- Em *template-form.module.ts* não importamos o *FormsModule*:

```

@NgModule({
  imports: [
    CommonModule
  ],
  declarations: [
    TemplateFormComponent
  ]
})
export class TemplateFormModule { }

```

Ao rodarmos a aplicação, ocorrerá erro, pois o *ngModel*, o *ngSubmit* e o *ngForm* pertencem ao *FormsModule*:



80 VÍDEO #77: FORMS (TEMPLATE DRIVEN) APLICANDO VALIDAÇÃO NOS CAMPOS

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video77>.

O “*required*” indica que o campo é obrigatório.

- Em *template-form.component.html*:

```
<form #f="ngForm" (ngSubmit)="onSubmit(f)" >
<div class="form-group">
  <label for="nome">Nome</label>
  <input type="text" class="form-control" name="name" id="nome"
    placeholder="Nome" [ngModel]="usuario.nome" required>
</div>

<div class="form-group">
  <label for="email">Email</label>
  <input type="email" class="form-control" name="email" id="email"
    placeholder="nome@email.com" [(ngModel)="usuario.email" required email>
</div>

  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Links úteis:

- 1) Para verificar todas as validações que um formulário pode ter, consulte a documentação do Angular disponível em: <https://angular.io/api/forms/Validators>.
- 2) Validações do HTML 5: <http://www.the-art-of-web.com/html/html5-form-validation/>.

81 VÍDEO #78: FORMS (TEMPLATE DRIVEN) APLICANDO CSS NA VALIDAÇÃO DOS CAMPOS

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video78>.

Estados dos campos no formulário:

Estado	SIM	NÃO
Controle visitado	ng-touched	ng-untouched
Valor mudou	ng-dirty	ng-pristine
Controle válido	ng-valid	ng-invalid

Imagem retirada do vídeo

- Em *template-form.component.css* definimos quais as validações os campos Nome e Email terão:

```
/* Se o campo está inválid (ng-invalid) ou
   recebeu o foco (ng-touched) e continua inválido,
   ele será contornado na cor vermelha*/

.ng-invalid.ng-touched:not(form){
  border: 1px solid red;
}
```

82 VÍDEO #79: FORMS (TEMPLATE DRIVEN) MOSTRANDO MENSAGENS DE ERROS DE VALIDAÇÃO

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video79>.

Para mostrar a mensagem de validação, podem ser utilizadas a diretiva *ngIf* ou a classe *has.error* do *Bootstrap*.

- Em *template-form.component.html*:

```
<form #f="ngForm" (ngSubmit)="onSubmit(f)" >
<div class="form-group"
[class.has-error]="!nome.valid && nome.touched">
  <!-- Primeira forma de fazer a validação; nesse caso não precisa do .css:
    [class.has-erros]="!nome.valid && nome.touched">-->
  <label for="nome">Nome</label>
```



```

<input type="text" class="form-control" name="name" id="nome"
  placeholder="Nome" [ngModel]="usuario.nome" required #nome="ngModel">
  <!-- Segunda forma de fazer a validação; nesse caso tem que ser o.css -->
  <div *ng-if="!nome.valid && nome.touched" class="alert alert-danger" role="alert">
    Nome é obrigatório.
  </div>
</div>

<div class="form-group">
  <label for="email">Email</label>
  <input type="email" class="form-control" name="email" id="email"
    placeholder="nome@email.com" [(ngModel)="usuario.email" required email
#email="ngModel">
    <div *ng-if="!email.valid && email.touched" class="alert alert-danger" role="alert">
      Email é obrigatório.
    </div>
  </div>

  <button type="submit" class="btn btn-primary">Submit</button>
</form>

```

83 VÍDEO #80: FORMS (TEMPLATE DRIVEN) DESABILITANDO O BOTÃO DE SUBMIT PARA FORMULÁRIO

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video80>.

- Em *template-form.component.html*:

```

<!-- Com a propriedade disable do HTML, informamos que o formulário só será submetido
  (e o botão Submit será habilitado) quando o formulário estiver preenchido corretamente
-->
<button type="submit" class="btn btn-primary" [disabled]="!f.valid">Submit</button>

```

84 VÍDEO #81: FORMS (DICA): VERIFICANDO DADOS DO FORM NO TEMPLATE COM JSON

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video81>.

Para não precisarmos ficar clicando no botão *Submit* para validar o formulário, podemos utilizar os *Pipes* para verificar as informações na tela (isso seria uma espécie de *debug*).

- Criado um novo component chamado *form-debug*. Em *form.debug.component.ts*, criamos uma variável *"form"*:

```
@Component({
```

```

    selector: 'app-form-debug',
    templateUrl: './form-debug.component.html',
    styleUrls: ['./form-debug.component.css']
  })
  export class FormDebugComponent implements OnInit {

    @Input() form;

    constructor() { }

    ngOnInit() {
    }

  }

```

- Em *form-debug.component.html*, pegamos as informações do formulário:

```

<div style="margin-top: 20px" *ng-if="form">
  <div>
    Detalhes do formulário
  </div>
  <pre>
    Form válido: {{ form.valid }} <!-- Indica se o formulário está válido -->
  </pre>
  <pre>
    Form submetido: {{ form.submitted }} <!-- Indica se o formulário foi submetido -->
  </pre>
  <pre>
    Valores: <br> {{ form.value | json }} <!-- Apresenta os valores digitados -->
  </pre>
</div>

```

- E no *template-form.component.ts*, foi adicionado o *selector* do *component*:

```

<!-- Quando entrar em produção, basta remover essa linha: -->
<app-form-debug [form]="f"> </app-form-debug>

```

85 VÍDEO #82: FORMS (TEMPLATE DRIVEN) ADICIONANDO CAMPOS DE ENDEREÇO (FORM LAYOUT)

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video82>.

Apenas adicionados novos campos no formulário: campos CEP, Número, Complemento, Rua, Bairro, Cidade, Estado no *template-form.component.html*.

86 VÍDEO #83: FORMS (TEMPLATE DRIVEN) REFATORANDO (SIMPLICANDO) CSS E MENSAGENS DE ERRO

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video83>.

Refatoração do código: criado um novo *component* chamado *campo-control-erro* e criados métodos para fazer as validações.

- Em *campo-control-erro.component.html* foi deixada apenas uma diretiva **ngIf* e inserido todos os ``:

```
<div *ngIf="mostrarErro">
  <span class="glyphicon glyphicon-remove form-control-feedback" aria-hidden="true">
</span>
  <span class="sr-only"> (error) </span>
  <div class="alert alert-danger" role="alert"> {{ msgErro }}</div>
</div>
```

- Em *campo-control-erro.component.ts* criado os *inputs*:

```
@Input () mostrarErro: boolean;
@Input () msgErro: string;
```

- Em *template-form.component.ts* criados os métodos:

```
verificaValidTouched(campo){
  return !campo.valid && campo.touched;
}

aplicaCssErro(campo){
  return{
    'has-error': this.verificaValidTouched(campo),
    'has-feedback': this.verificaValidTouched(campo)
  }
}
```

- Em *template-form.component.html*: Com a refatoração, o código do *template* fica menor e mais fácil para se dar manutenção:

```
<form #f="ngForm" (ngSubmit)="onSubmit(f)" class="form-horizontal">
  <div class="form-group" [ngClass]="aplicaCssErro(nome)">
    <div class="col-sm-12">
      <label for="nome" class="control-label">Nome </label>
    </div>
    <div class="col-sm-12">
      <input type="texto" class="form-control" name="name" id="nome" placeholder="Nome"
```

```

    [ngModel]="usuario.nome" required #nome="ngModel">
  <app-campo-control-erro
    [mostrarErro]="verificaValidTouched(nome)"
    msgErro="Nome é obrigatório!">
  </app-campo-control-erro>
</div>
</div>

```

87 VÍDEO #84: FORMS (TEMPLATE DRIVEN) FORM GRUPS (AGRUPANDO DADOS)

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video84>.

No Angular existe a diretiva *ngModelGroup* para agrupar determinados dados.

- Em *template-form.component.html* foi criada uma nova `<div>` com o *ngModelGroup*, para agrupar os dados do endereço. Essa `<div>` deve ser fechada (`</div>`) no final dos campos que serão agrupados.

```

<div ngModelGroup="endereco">
  <div class="form-group">
    <div class="col-md-3" [ngClass]="aplicaCssErro(cep)">
      <label for="cep" class="control-label">CEP </label>
      <input type="text" class="form-control" id="cep" name="cep" ngModel required
        #cep="ngModel">
      <app-campo-control-erro
        [mostrarErro]="verificaValidTouched(cep)"
        msgErro="CEP é obrigatório!">
      </app-campo-control-erro>
    </div>
    <div class="col-md-3" [ngClass]="aplicaCssErro(numero)">
      <label for="numero" class="control-label">Número </label>
      <input type="text" class="form-control" id="numero" name="numero"
        ngModel required #numero="ngModel">
      <app-campo-control-erro
        [mostrarErro]="verificaValidTouched(numero)"
        msgErro="Número é obrigatório!">
      </app-campo-control-erro>
    </div>
    <div class="col-md-6">
      <label for="complemento" class="control-label">Complemento </label>
      <input type="text" class="form-control" id="complemento"
        name="complemento" ngModel #complemento="ngModel">
    </div>
  </div>

  <div class="form-group" [ngClass]="aplicaCssErro(rua)">
    <div class="col-sm-12">
      <label for="rua" class="control-label">Rua</label>
    </div>
    <div class="col-sm-12">
      <input type="text" class="form-control" name="name" id="rua"

```

```

        [ngModel]="usuario.rua" required #rua="ngModel" readonly>
    <app-campo-control-erro
        [mostrarErro]="verificaValidTouched(rua)"
        msgErro="Rua é obrigatória!">
    </app-campo-control-erro>
</div>
</div>
</div>

<div class="form-group" [ngClass]="aplicaCssErro(bairro)">
    <div class="col-md-5">
        <label for="bairro" class="control-label">Bairro </label>
        <input type="text" class="form-control" id="bairro" name="bairro"
            ngModel required #bairro="ngModel">
        <app-campo-control-erro
            [mostrarErro]="verificaValidTouched(bairro)"
            msgErro="Bairro é obrigatório!">
        </app-campo-control-erro>
    </div>
    <div class="col-md-4" [ngClass]="aplicaCssErro(cidade)">
        <label for="cidade" class="control-label">Cidade </label>
        <input type="text" class="form-control" id="cidade" name="cidade"
            ngModel required #cidade="ngModel">
        <app-campo-control-erro
            [mostrarErro]="verificaValidTouched(cidade)"
            msgErro="Cidade é obrigatória!">
        </app-campo-control-erro>
    </div>
    <div class="col-md-3" [ngClass]="aplicaCssErro(estado)">
        <label for="estado" class="control-label">Estado </label>
        <input type="text" class="form-control" id="estado" name="estado"
            ngModel required #estado="ngModel">
        <app-campo-control-erro
            [mostrarErro]="verificaValidTouched(estado)"
            msgErro="Estado é obrigatório!">
        </app-campo-control-erro>
    </div>
</div>
</div>
</div> <!-- Fechando a div com os campos agrupados -->

```

88 VÍDEO #85: FORMS (TEMPLATE DRIVEN) PESQUISANDO ENDEREÇO AUTOMATICAMENTE COM CEP

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video85>.

- Em *template-form.component.ts* criado um método consultaCEP:

```

consultaCEP(cep) {
    // Criando variável CEP e removendo quaisquer letras ou caracteres especiais
    cep = cep.replace(/\D/g, '');
    // Verificando se o campo CEP possui valor informado
    if (cep != "") {

```

```

// Expressão regular para verificar o CEP
var validacep = /^[0-9]{8}$/;
// Verificando o formato do CEP
if (validacep.test(cep)) {
  // Consulta ao Webservice, mapeando os dados e recebendo as informações:
  this.http.get(`//viacep.com.br/ws/${cep}/json`)
    .map(dados => dados.json())
    .subscribe(dados => console.log(dados));
}
}
}

```

- Em *template-form.component.html* criado evento (*blur*) com o método *consultaCEP* (quando o foco sair do campo CEP, a consulta ao *Webservice* será realizada):

```

<div class="col-md-3" [ngClass]="aplicaCssErro(cep)">
  <label for="cep" class="control-label">CEP </label>
  <input type="text" class="form-control" id="cep" name="cep" ngModel required
    #cep="ngModel" (blur)="consultaCEP($event.target.value)">
  <app-campo-control-erro
    [mostrarErro]="verificaValidTouched(cep)"
    msgErro="CEP é obrigatório!">
  </app-campo-control-erro>
</div>

```

Links úteis:

- 1) Webservice gratuito para consultar CEP: <http://viacep.com.br/>.
- 2) Exemplo de preenchimento de CEP com o jquery: <http://viacep.com.br/exemplo/jquery/>.

89 VÍDEO #86: FORMS (TEMPLATE DRIVEN) POPULANDO CAMPOS COM SETVALUE E PATCHVALUE (CEP)

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video86>.

- Em *template-form.component.ts* criados os métodos para preencher e limpar os campos:

```

// Método para popular os campos:
// O problema de se usar esse método, é se o formulário tiver muitos dados,
// teria que carregar todos os campos
populaDadosForm(dados, formulario) {
  /* formulario.setValue({
    nome: formulario.value.nome,
    email: formulario.value.email,
    endereco: {
      rua: dados.logradouro,
      cep: dados.cep,
      numero: '',
      complemento: dados.complemento,
      bairro: dados.bairro,

```

```

        cidade: dados.localidade,
        estado: dados.uf
    }
});*/

// A forma correta é utilizar o patchValue, que é uma propriedade do formulário;
// nesse método deve ser passado apenas os campos que realmente serão preenchidos:
formulario.form.patchValue({
    endereco: {
        rua: dados.logradouro,
        cep: dados.cep,
        complemento: dados.complemento,
        bairro: dados.bairro,
        cidade: dados.localidade,
        estado: dados.uf
    }
});
}

// Método para resetar as informações do formulário:
resetaDadosForm(formulario){
    formulario.form.patchValue({
        endereco: {
            rua: null,
            cep: null,
            complemento: null,
            bairro: null,
            cidade: null,
            estado: null
        }
    });
}
}

```

90 VÍDEO #87: FORMS (TEMPLATE DRIVEN) SUBMETENDO VALORES COM HTTP POST

OBS: Projeto disponível em: <https://github.com/nasouza2/Angular2B/tree/master/Video87>.

- Em *template-form.component.ts* no método *onSubmit*, foi implementada a consulta *HTTP*:

```

onSubmit(form) {
    console.log(form);
    /* Método para passar as informações para o servidor:
       O JSON.stringify transforma as informações do JSON em string */
    this.http.post('https://httpbin.org/get', JSON.stringify(form.value))
        .map(res => res)
        .subscribe(dados => console.log(dados));
}

```

Link útil:

- 1) Serviço gratuito para testar requisições HTTP: <https://httpbin.org/get>.

91 CONCLUSÃO

A área de TI é muito vasta e embora estudando/trabalhando nela há uns anos, nunca tinha ouvido falar sobre o Angular. Gostei de ter aprendido sobre esse *framework*, apesar do tempo para estudo ter sido curto, tanto pelo prazo de entrega quanto ao tempo que tinha disponível para realizar o projeto. Disso isso porque provavelmente o *framework* teve algumas atualizações desde que os vídeos foram disponibilizados e o tempo de pesquisa para encontrar o comando atual, estava sendo muito grande.

Alguns desses comandos consegui identificar e atualizá-los no projeto, mas outros, embora pesquisando, não encontrei o comando atual correspondente; dessa forma, todos os capítulos que não possuem o resultado no navegador, são vídeos que não apresentam erros no momento em que a aplicação é servida ao *browser* (comando `ng serve`), mas a aplicação não abre no *localhost*.

A maior dificuldade que tive durante esse estudo, não foi necessariamente ligada ao Angular, mas sim a programação. Alguns vídeos tive que assistir várias vezes para perceber o que estava sendo feito de errado, visto que no momento de servir a aplicação, ocorria erro de compilação. Grande parte desses erros consegui ajustar, porque foram detalhes que se passaram despercebidos durante o acompanhamento da aula, como grafia incorreta, *imports* declarados nos lugares errados, falta de chaves ou parênteses e principalmente, falta de espaço entre as chaves; já outros erros, como os dos vídeos 14 e 61 não consegui ajustar, então eles foram entregues não funcionando corretamente.

Agora com o projeto entregue, cabe a mim pesquisar sobre os comandos atuais do Angular e ajustar os vídeos que não deram certo, assim como terminar o curso (já que até o vídeo 87, muitos assuntos ainda não foram estudados, como os formulários reativos e mais detalhes sobre requisições HTTP e POST).