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

**1 INTRODUÇÃO**

A SER IMPLEMENTADA.

**2 DEFINIÇÃO**

A SER IMPLEMENTADA

**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, utilizando todos os conceitos vistos no curso.

## 3.2 O que é necessário saber para realizar o curso?

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

## 3.3 O que é o Angular 2?

## Angular 2[[1]](#footnote-1) é um *framework* nascido da parceira da Google com a Microsoft; escrito em *TypeScript* e possui código *Open Source* disponível no *GitHub*2[[2]](#footnote-2). O Angular 2 não é continuação do Angular 1, pois foi reescrito para fazer melhor uso da tecnologia *HTML*.

## Esse *framework* é totalmente orientado a Componente, ou seja, tudo 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 nos Blocos Principais, que são: Componentes; Diretivas; Roteamento; Serviços; *Template*; *Metadata*; Data *Binding* e Injeção de dependência.

## 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 seguite 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:

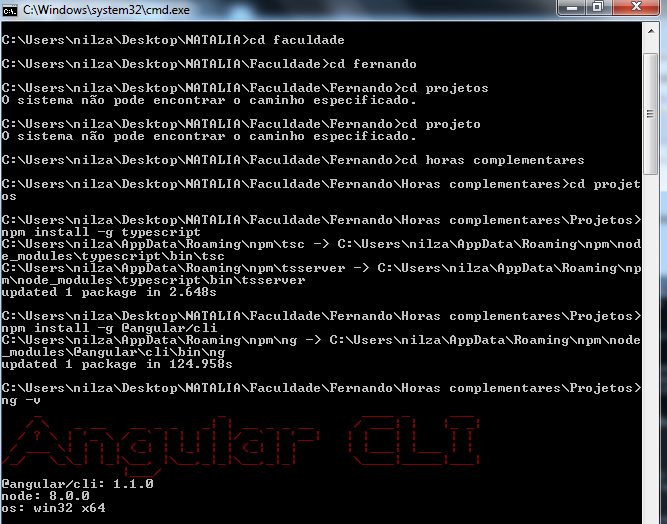
- npm install -g @angular/cli (se seu SO for Windows) ou

- 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:



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 *Sublime* *Text*, 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 especifica para sua instalação.

- ATOM: Ferramenta gratuita. Para utilizar esse editor, é necessário baixar *plugin* do *TypeScript*. Esse *plugin* está disponível para *dowloand* 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 especifica para sua instalação.

- Sublime *Text*: Ferramenta gratuita, porém caso desejar, você pode adquiri 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, será utilizado o editor Visual *Studio Code*.

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

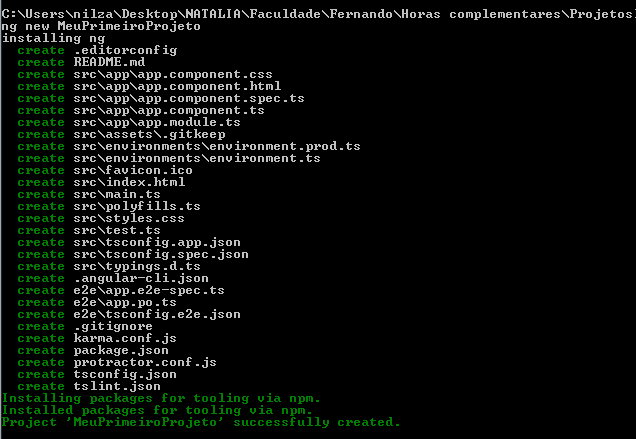
**OBS:** Projeto disponível em: github.com/nasouza2/Angular2B/tree/master/MeuPrimeiroProjeto.

**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

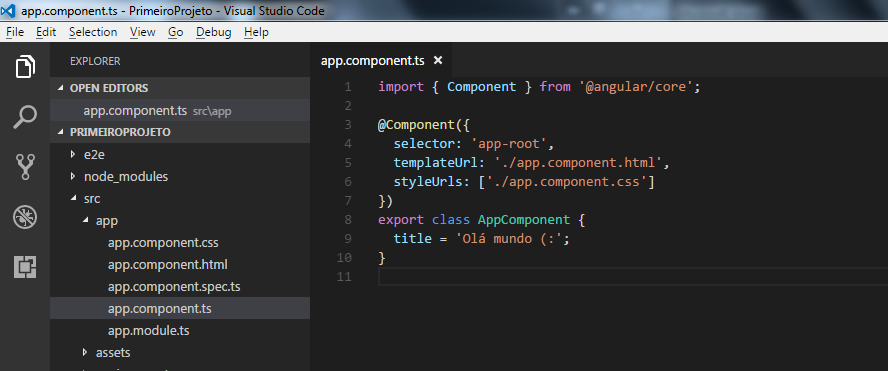
No final, o angular/cli vai criar toda a estrutura padrão do projeto e também vai instalar as dependências do NPM:



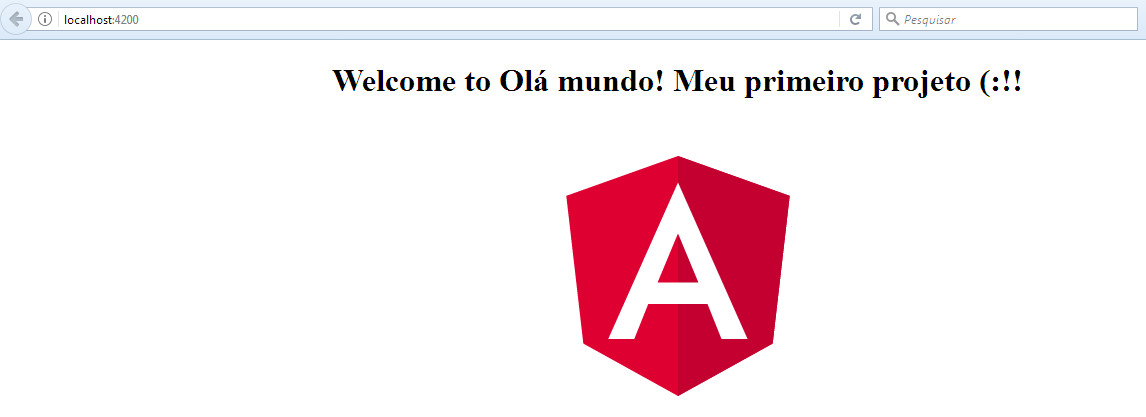
- Ainda com o Prompt de Comando aberto, acesse seu projeto e digite o comando: ng serve. Esse comando indica que nosso projeto vai ser servido ao *browser*.

Abra o projeto com o editor que texto. Dentro do diretório scr – app é onde os componentes serão criados.

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



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

****

**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 do 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 a tag ‘meu-primeiro-component’ e a cole no arquivo app.component.html.

OBS: Para deixar o código mais limpo, foram removidos os links que haviam nesse arquivo, como pode ser visto na figura x:

<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 componente, serviço, diretiva criada, deve ser informada em um módulo. Abra o arquivo app.modules.ts e declare seu componente, dentro das declarações do modulo e o importe (no import, não é necessário informar a extensão .js):

import { BrowserModule } from '@angular/platform-browser';

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

import { AppComponent } from './app.component';

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

@NgModule({

declarations: [

AppComponent,

MeuPrimeiroComponent

],

imports: [

BrowserModule

],

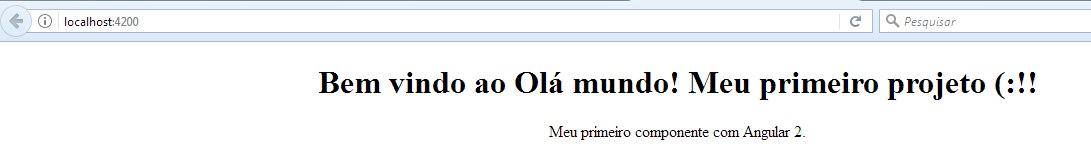
providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

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



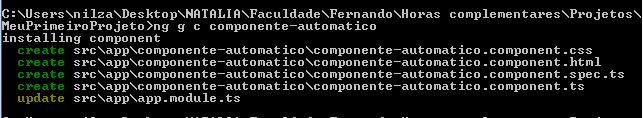
**Dificuldade encontrada:** Atente-se para usar crase e não aspas simples no Template. Se usar aspas simples, ocorrerá erro de compilação. “Atente-se também para extensão do seu componente, que deve ser “.ts” de *TypeScript*. Devido a um erro de digitação, a aplicação não conseguia encontrar o componente; 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**

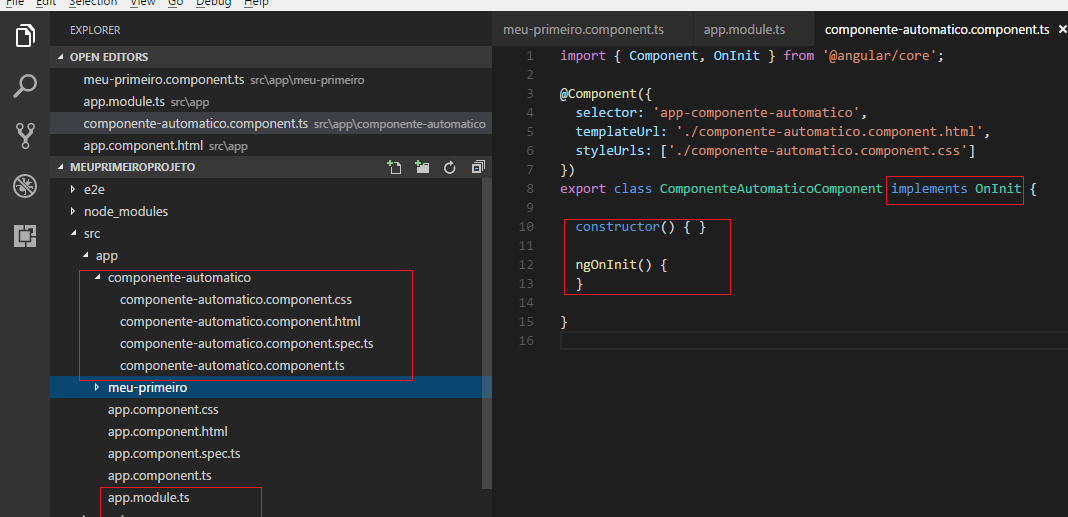
O componente foi criado componente, apenas para entendermos o passo a passo de sua criação e o padrão de nomenclatura. Agora vamos ver com criar os componentes de forma automática.

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

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



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



No seu 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 import do component:

import { BrowserModule } from '@angular/platform-browser';

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

import { AppComponent } from './app.component';

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

import { ComponenteAutomaticoComponent } from './componente-automatico/componente-automatico.component';

@NgModule({

declarations: [

AppComponent,

MeuPrimeiroComponent,

ComponenteAutomaticoComponent

],

imports: [

BrowserModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

Abra o arquivo app.component.html e informe a tag do seu component:

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

<h1>

Bem vindo {{title}}!!

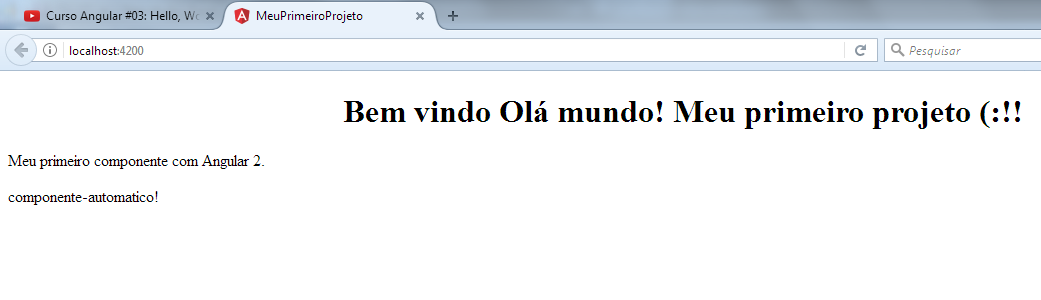
</h1>

</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 foi executado corretamente:



**Link útil:** Instalar e configurar o git no Windows: http://gabsferreira.com/instalando-o-git-e-configurando-github.

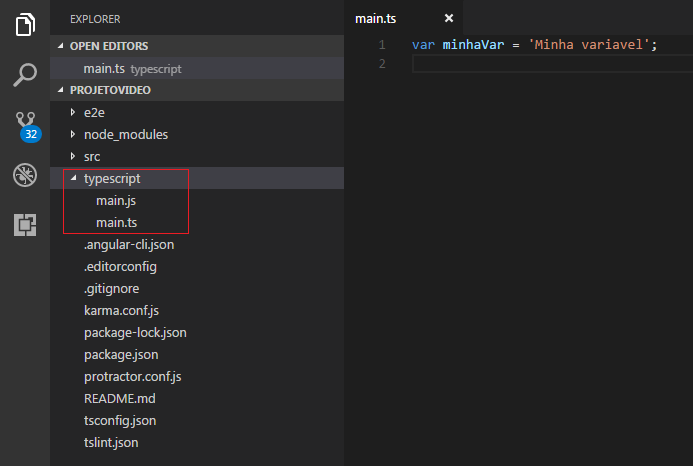
**6 VÍDEO #04: INTRODUÇÃO AO TYPE SCRIPT PARA ANGULAR**

**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:

var minhaVar = '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 Java Script puro.

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

var minhaVar = 'Minha variavel';

function minhaFuncao(x, y) {

return x + y;

}

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

var minhaVar = 'Minha variavel';

function minhaFuncao(x, y){

return x + y;

}

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

var minhaVar = 'Minha variavel';

function minhaFuncao(x, y){

return x + y;

}

let num = 2;

Nem todos os *browser*, estão adaptados para utilizar o *ECMAScprit* 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:

var minhaVar = 'Minha variavel';

function minhaFuncao(x, y) {

return x + y;

}

var num = 2;

Mais um exemplo com ES6:

var minhaVar = 'Minha variavel';

function minhaFuncao(x, y){

return x + y;

}

//ES6

let num = 2;

const PI = 3.14;

O arquivo main.js ficou da seguinte forma:

var minhaVar = 'Minha variavel';

function minhaFuncao(x, y) {

return x + y;

}

var num = 2;

var PI = 3.14;

**Dificuldades encontradas:** 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: 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:**

-Todas as alterações do ECMA2015, podem ser encontradas em: es6-features.org/#Constants.

*- Transpiler* Babel: babeljs.io/repl/#?babili=false&evaluate=true&lineWrap=false&presets=es2015%2Creact%2Cstage-2&targets=&browsers=&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 BrowserModule prepara a aplicação para ser usada na web

import { BrowserModule } from '@angular/platform-browser';

// Indicando da onde o NgModule esta

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: [

BrowserModule

],

// 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]

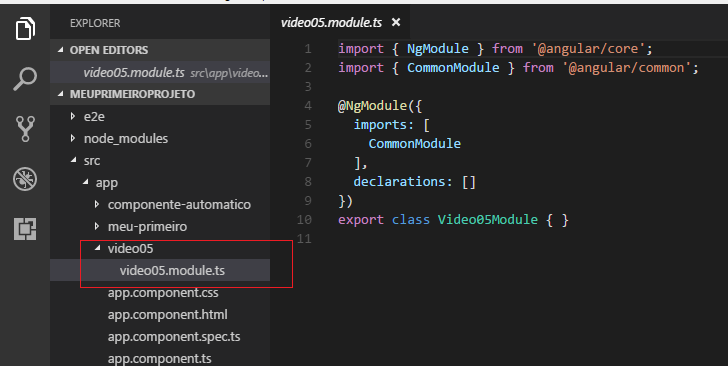
})

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 nomedomodulo (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 dentro de duas chaves, informe a variável:

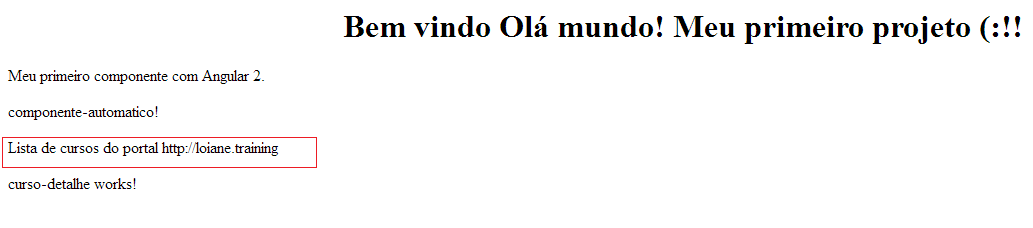
<p>

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

</p>

<app-curso-detalhe> </app-curso-detalhe>

No browser, o projeto ficou assim:



Para fazer listagem manualmente, utilizamos a tag <ul>:

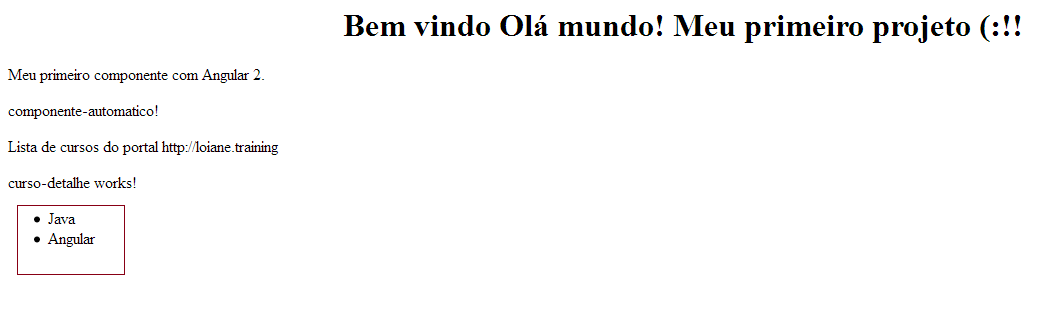
<!-- Para fazer listagem manualmente, geralmente utilizamos a tage <ul>-->

<ul>

<li> Java </li>

<li> Angular </li>

</ul>



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 <li>, 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 váriavel curso que vai receber a posição do array cursos -->

<li \*ngFor="let curso of cursos">

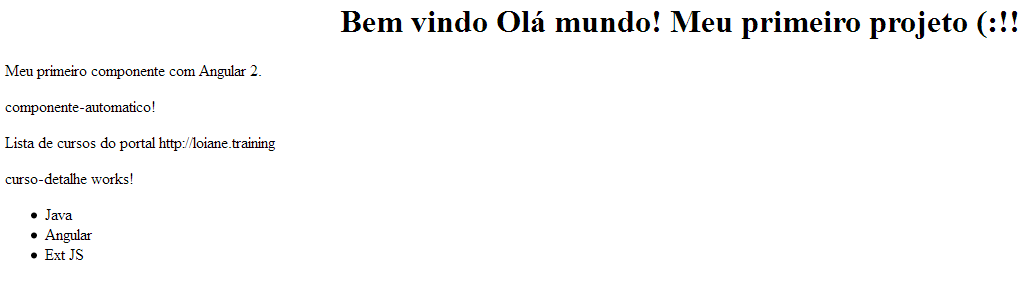
<!-- Depois é feita a interpolação para a saída da variável curso -->

{{ curso }}

</li>

</ul>

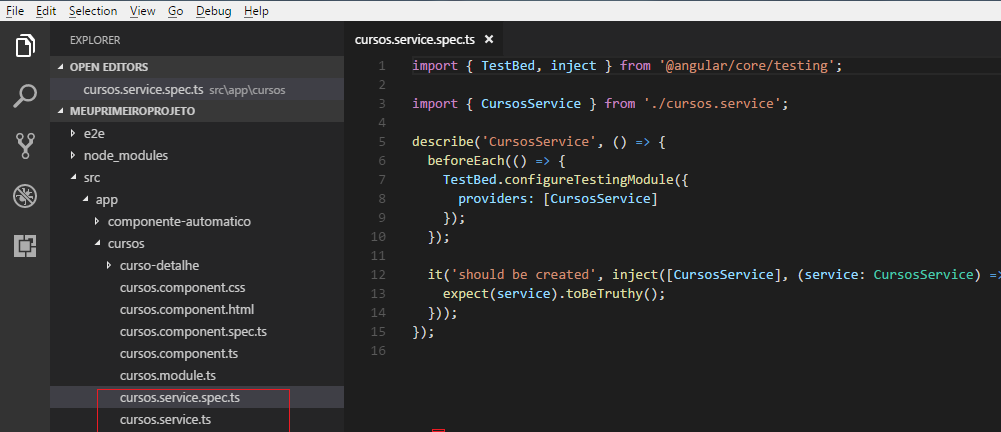
No *browser*, o projeto ficou assim:



**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 componente apenas passando o resultado do nosso método. Precisamos importar, usar a classe cursos.*service* e criar variável que instancia 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 {

// Podemos instanciar a variável por aqui, informando o que ela vai receber ou

nomePortal: string;

cursos: string[]; // Deixamos o componente passando apenas o resultado

// podemos iniciar ela através do construtor, usando a palavra "this"

constructor() {

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

// Criando a variável

var servico = new CursosService();

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 {

// Podemos instanciar a variável por aqui, informando o que ela vai receber ou

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) {

// podemos iniciar ela através do construtor, usando a palavra "this"

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 P*roviders* 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 CommomModule

import { CommonModule } from '@angular/common';

import { CursosComponent } from './cursos.component';

import { CursoDetalheComponent } from './curso-detalhe/curso-detalhe.component';

import { CursosService } from './cursos.service';

@NgModule({

imports: [

CommonModule

],

declarations: [

CursosComponent,

CursoDetalheComponent

],

exports: [

CursosComponent

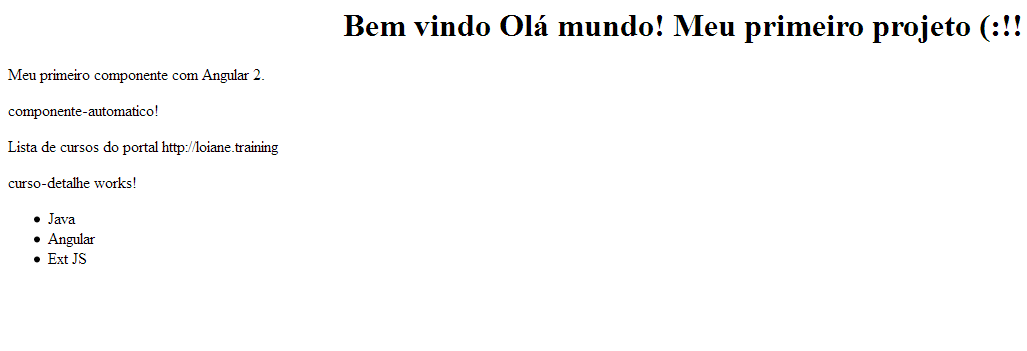
],

providers: [ CursosService ]

})

export class CursosModule { }

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



**10 VÍDEO #08: DICAS PLUGINS ANGULAR PARA ATOM E VC 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 *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* *upcoming* *latest* *TypeScrit* 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

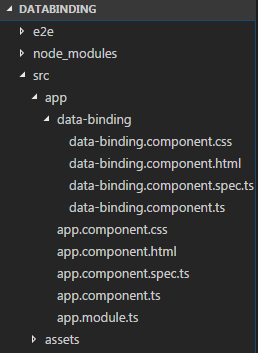
- *Ionic* 2 *Commands* *with* *Snippets* (para quem for trabalhar com *Ionic*).

- *vscode*-*icons*: Plugin que cria os ícones dos arquivos, facilitando a visualização dos mesmos.

**11 VÍDEO #09: *PROPERTY* *BINDING* E INTERPOLAÇÃO**

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

Data *Binging* 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.compenent.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 *Property* *Binding*:

<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 metódos e: -->

<p> Resultado de 1 + 1 não é: {{ 1 + 1 + getValor() }} </p>

<!-- Resolver expressões boobleanas: -->

<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: -->

<img src="{{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

// Metódo

getValor(){

return 1;

}

// Metódo utilizado com a expressão booleana

getCurtirCurso(){

return true;

}

constructor() { }

ngOnInit() {

}

}

**21 VÍDEO #10: CLASS E STYLE BINDING**

O Class e o Style Binding também são classificadas como Property Binding; a diferença entre eles, é que ao invés de utilizarmos váriaveis 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.

**Dificuldades encontradas:** No vídeo 10, o comando para instalar o bootstrap era ”ng2-bootstrap –save” porem a página do boostrap foi alterada e agora o comando é “npm install ngx-bootstrap bootstrap --save”

**Links úteis:** Bibliotecas do bootstrap para o Angular 2 e Angular 4: http://valor-software.com/ngx-bootstrap/#/

1. Site oficial do framework: https://angular.io/ [↑](#footnote-ref-1)
2. Diretório do Angular 2 no GitHub: https://github.com/angular [↑](#footnote-ref-2)