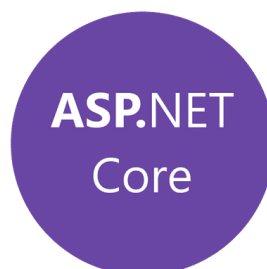




+



## Sobre Este Curso

### **Público Alvo**

Programadores que têm como objetivo aprimorar seus conhecimentos na Linguagem C# utilizando o Framework .Net Core.

### **Pré-Requisitos**

Conhecimentos de Lógica de Programação, Banco de Dados e a base da Linguagem C#.



## Índice

<b>SOBRE ESTE CURSO .....</b>	<b>I</b>
PÚBLICO ALVO .....	I
PRÉ-REQUISITOS .....	I
<b>ÍNDICE.....</b>	<b>I</b>
<b>COPYRIGHT .....</b>	<b>II</b>
EQUIPE.....	II
HISTÓRICO DAS EDIÇÕES.....	II
<b>CAPÍTULO 01 – O QUE É ANGULAR.....</b>	<b>3</b>
<b>CAPÍTULO 02 - ANGULARJS E ANGULAR .....</b>	<b>4</b>
<b>CAPÍTULO 03 - PADRÃO SPA .....</b>	<b>6</b>
<b>CAPÍTULO 04 - PREPARANDO O AMBIENTE.....</b>	<b>7</b>
<b>CAPÍTULO 05 - INSTALANDO O BOOTSTRAP.....</b>	<b>11</b>
<b>CAPÍTULO 06 - NG-IF .....</b>	<b>14</b>
<b>CAPÍTULO 07 - NG-SWITCH .....</b>	<b>15</b>
<b>CAPÍTULO 08 - NG-FOR .....</b>	<b>17</b>
<b>CAPÍTULO 09 - NG-CLASS .....</b>	<b>18</b>
<b>CAPÍTULO 10 - MÓDULO HTTP .....</b>	<b>20</b>
<b>CAPÍTULO 11 - FUNCIONAMENTO DO ANGULAR NO .NET CORE .....</b>	<b>21</b>
<b>CAPÍTULO 12 - COMUNICANDO A APLICAÇÃO ANGULAR COM .NET COE.....</b>	<b>24</b>
<b>CAPÍTULO 13 - FAZENDO REQUISIÇÕES VIA REST PARA O CONTROLLER .....</b>	<b>25</b>
<b>CAPÍTULO 14 - CRIANDO PROJETO COM ANGULAR COM .NET CORE .....</b>	<b>29</b>

	Anotações



## Copyright

As informações contidas neste material se referem ao curso de **Angular JS e Asp.NET Core** e estão sujeitas a alterações sem comunicação prévia, não representando um compromisso por parte do autor em atualização automática de futuras versões.

A **Apex** não será responsável por quaisquer erros ou por danos acidentais ou consequenciais relacionados com o fornecimento, desempenho, ou uso desta apostila ou os exemplos contidos aqui.

Os exemplos de empresas, organizações, produtos, nomes de domínio, endereços de e-mail, logotipos, pessoas, lugares e eventos aqui representados são fictícios. Nenhuma associação a empresas, organizações, produtos, nomes de domínio, endereços de e-mail, logotipos, pessoas, lugares ou eventos reais é intencional ou deve ser inferida.

A reprodução, adaptação, ou tradução deste manual mesmo que parcial, para qualquer finalidade é proibida sem autorização prévia por escrito da **Apex**, exceto as permitidas sob as leis de direito autoral.

## Equipe

### Conteúdos

- Ralf S. Lima
- Gustavo Rosauro

### Diagramação

- Brissany C. Beltrão

### Revisão

- Fernanda Pereira

## Histórico das Edições

Edição	Idioma	Edição
1ª	Português	Agosto de 2019

© Copyright 2019 **Apex**. Desenvolvido por Ralf S. Lima e Gustavo Rosauro e licenciado para Apex Ensino

Anotações	

## Capítulo 01 – O que é Angular

Angular é um framework mantido pela Google, que inicialmente foi desenvolvido em JavaScript. Suas principais funcionalidades se concentram em reutilizar componentes, trabalhar com o padrão SPA e interação rápida e fácil com eventos, fornecendo ao usuário uma experiência de uso muito boa.

Atualmente o Angular está na versão 8 utilizando a tecnologia TypeScript desenvolvida pela Microsoft, onde há o conceito de orientação a objetos, facilitando a reutilização de métodos e estruturação de projetos considerados complexos.

O framework conta com uma comunidade bem ativa e possui diversos complementos para atuar com o front-end e o back-end, sendo assim é muito normal encontrar vagas em aberto para profissionais com conhecimento em Angular.

Vale lembrar que com o Angular é possível integrar com qualquer back-end, sendo assim você pode desenvolver aplicações utilizando Java, C#, PHP, NodeJS, entre outras linguagens com acesso ao banco de dados de sua preferência.



	Anotações



## Capítulo 02 - AngularJS e Angular

Talvez você já se deparou com livros, tutoriais, cursos ou vagas de emprego com o nome Angular ou AngularJS, mas qual é a diferença entre eles? Bem para entendermos melhor vamos voltar para o ano de 2009 e citar seus criadores Misko Hevery e Adam Abrons, a ideia era criar um framework em JavaScript com uma organização mais eficaz para sistemas de grande porte.

O JavaScript possui uma gama muito grande de funcionalidades que podemos implementar em nossos projetos, porém a falta de padronização deixava a linguagem muitas vezes mal vista. Esse projeto chamado AngularJS teve como principal característica remover essa má fama da linguagem, e propor aos desenvolvedores uma ferramenta simples, rápida e de fácil compreensão.

Na época o Google tinha algo parecido que era o GWT ou Google Web Toolkit, além de outras tecnologias para auxiliar em projetos internos. Em um determinado projeto Misko estava com dificuldades pelos códigos repetitivos, o projeto estava com 17.000 linhas e estava sendo desenvolvido há seis meses.

Misko pensou em fazer uma aposta com seus superiores dizendo que conseguiria fazer o mesmo projeto de seis meses em apenas duas semanas utilizando outras tecnologias, no caso dando ênfase no AngularJS.

A aposta foi firmada, Misko infelizmente perdeu, conseguindo desenvolver o sistema em três semanas ao invés de duas, mesmo perdendo a aposta o resultado foi visto como positivo, outro ponto que chamou a atenção foi o número de linhas, o projeto de seis meses criado pela equipe estava com 17.000 linhas aproximadamente, já o projeto remodelado de Misko conseguiu o mesmo feito com apenas 1.500 linhas.

Anotações	



O Google adotou o framework como um de seus principais para desenvolvimento front-end, atualmente o Google possui mais de duzentos projetos utilizando a tecnologia criada por Misko e Adam, porém ainda podia ser melhorada e a Google viu o que poderia ser adicionado.

Em 2012 a Microsoft lança uma evolução do JavaScript, ele se chamava TypeScript, uma linguagem baseada em JavaScript com o conceito de orientação a objetos embutida.

A Google viu o potencial no uso do TypeScript e abraçou a linguagem, a partir da versão 2.0 o nome AngularJS é abandonado e a partir desse momento seria chamado apenas de Angular.

	Anotações

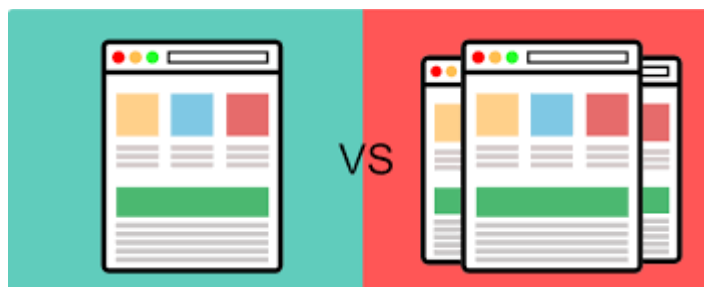


## Capítulo 03 - Padrão SPA

O padrão SPA ou Single Page Application, é um padrão de projetos web, onde a aplicação não atualiza a página. Os dados são carregados unicamente com o Ajax, já a comunicação entre as páginas é feita através de rotas.

Esse conceito permite o uso de páginas web mais dinâmicas, além de conseguir exportar sua aplicação de maneira híbrida, sendo assim o desenvolvedor pode criar aplicativos móveis sem a utilização de linguagens como o Java ou o Kotlin para o desenvolvimento Android ou o Swift para o desenvolvimento iOS.

O uso do SPA é algo bem visado pelas empresas e podemos nos deparar com muitos exemplos criados por empresas como Microsoft no Outlook ou Google para o Gmail, abaixo vamos compreender a diferença entre um projeto SPA e ou projeto onde temos três páginas individuais:



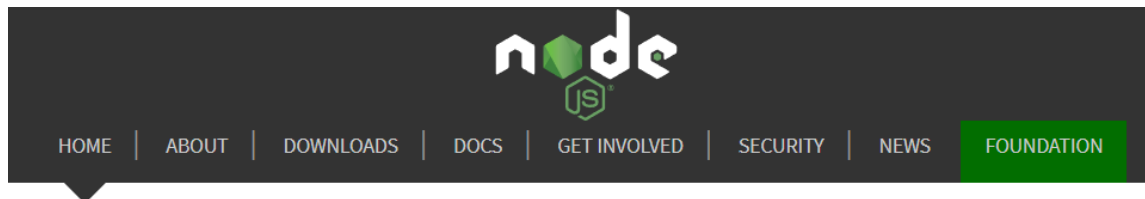
Na imagem anterior é possível notar que a estrutura da esquerda está englobada em uma espécie de caixa, isso significa que as páginas serão exibidas em um único arquivo, já o exemplo da direita temos cada uma de maneira individual, sendo assim haverá uma atualização sempre que clicar em algum link.

Anotações	



## Capítulo 04 - Preparando o ambiente

Para desenvolvermos nossas aplicações utilizando o Angular precisamos ter instalado o NodeJS, para isso acesse o link <http://www.nodejs.org> e baixe a versão recomendada.



Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

### Download for Windows (x64)

10.16.3 LTS

Recommended For Most Users

12.8.1 Current

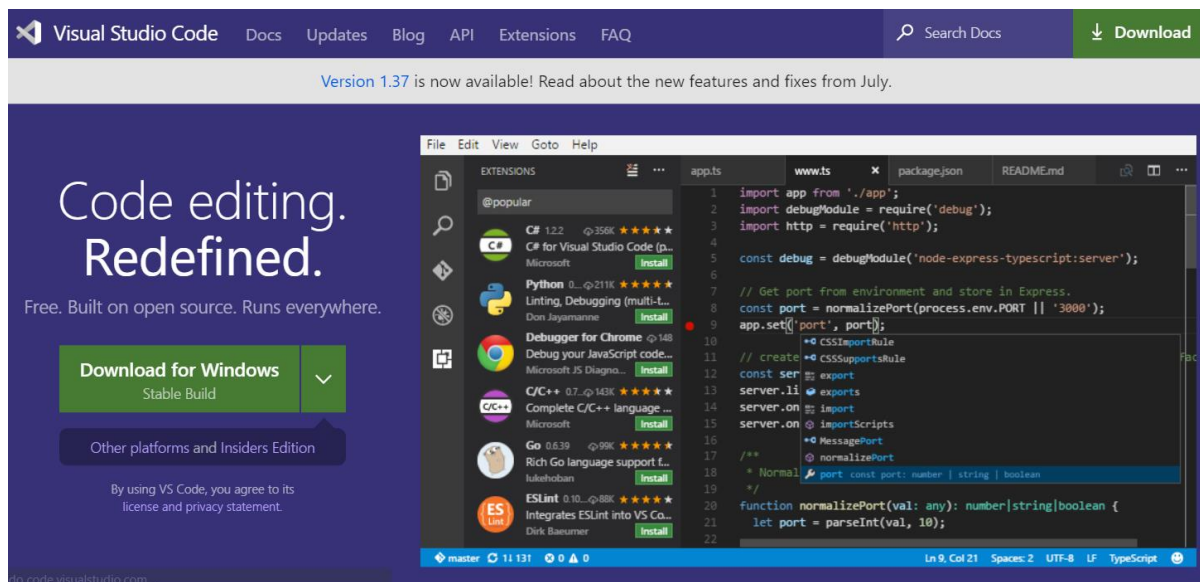
Latest Features

O NodeJS é um interpretador de código aberto baseado em JavaScript, com ele é possível criar aplicações que não precisam de um navegador para ser executado, o NodeJS quando baixado também irá servir para emular um servidor para testarmos nossas aplicações em Angular. Além disso podemos baixar o Angular e algumas dependências extras via NPM, que é atualmente o maior repositório de arquivos do mundo, sendo assim não há mais necessidade em utilizar arquivos externos ou baixar, descompactar e fazer a referência.

	Anotações



Para desenvolvermos nossas aplicações em Angular será utilizado o Visual Studio Code da Microsoft, um editor muito leve, simples e gratuito, porém se preferir utilizar outro editor ou IDE fique à vontade.



Antes de iniciarmos nossa aplicação em Angular, vamos verificar se está tudo instalado corretamente, começando pelo NodeJS, abra o prompt de comando e digite **node -v** e verifique se a versão é exibida.

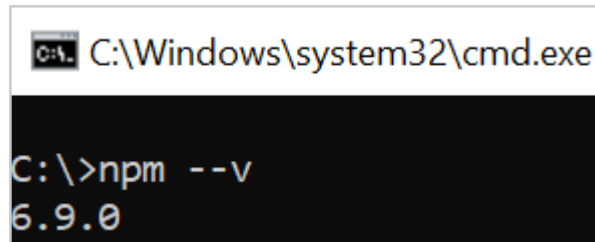
```
C:\Windows\system32\cmd.exe

C:\>node -v
v10.16.0
```

Se porventura o comando não aparecer, tente reiniciar o computador ou instale novamente o NodeJS.

Anotações	

Agora vamos verificar se temos instalado o NPM, que é o gerenciador de pacotes do NodeJS, com ele será possível realizar instalações de softwares como o Angular por exemplo. Por padrão o NPM vem instalado com o NodeJS, mas vamos garantir que está tudo ok utilizando o comando **npm --v**:



```
C:\Windows\system32\cmd.exe
C:\>npm --v
6.9.0
```

Quando verificado que o NPM está instalado, você precisa saber que os comandos que iremos utilizar a partir de agora relacionados ao Angular pertencem ao pacote **Angular Cli**, através dele será possível criar projetos, criar componentes, módulos, realizar testes, exportar projeto, entre outras funcionalidades.

Para mais informações você pode acessar o site <https://cli.angular.io>, na página inicial você já se depara com os comandos mais utilizados, se eventualmente algum comando do Angular não funcionar, basta baixá-lo.

	Anotações



Podemos estar garantido que está instalado com o comando **ng --version**:

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

C:\>ng --version

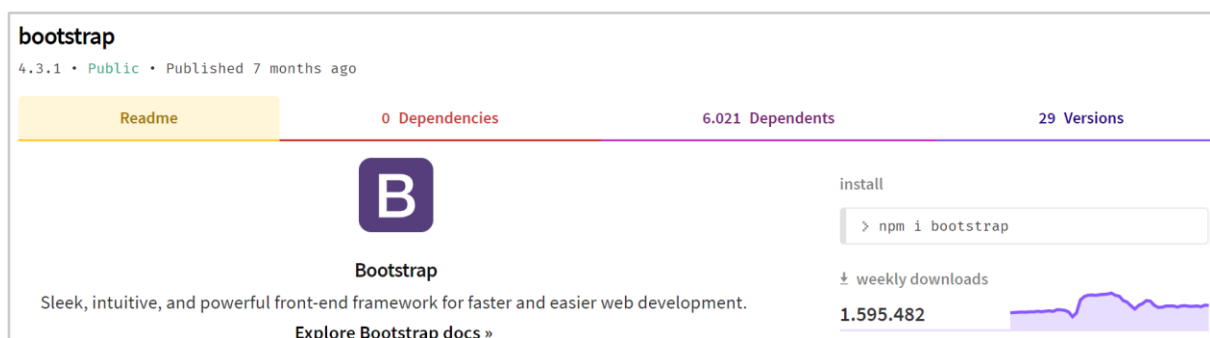
Angular CLI
Angular CLI: 8.2.1
Node: 10.16.0
OS: win32 x64
Angular:
...

Package                                Version
-----
@angular-devkit/architect              0.802.1
@angular-devkit/core                   8.2.1
@angular-devkit/schematics             8.2.1
@schematics/angular                   8.2.1
@schematics/update                     0.802.1
rxjs                                   6.4.0
```

## Capítulo 05 - Instalando o Bootstrap

Para deixar nossas aplicações com Angular mais bonitas, podemos utilizar diversos frameworks como o Bootstrap, porém caso em projetos futuros queira utilizar outras como o Materialize.css, esteja à vontade.

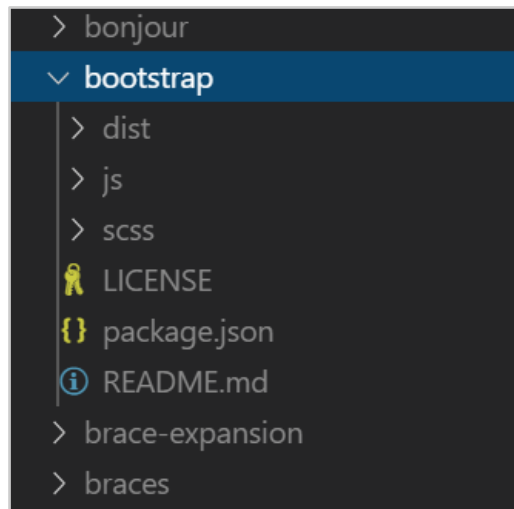
Inicialmente podemos acessar o site oficial do NPM e copiar o link para realizar o download utilizando nosso terminal:



Abra o seu terminal no Visual Studio Code e digite o comando **npm install bootstrap@4.1.3 jquery@3.3.1 popper.js@1.14.3 --save**, em seguida os arquivos do Bootstrap serão baixados. Agora é necessário configurar o arquivo **angular.json**, nele haverá duas dependências, uma de CSS e outra de JavaScript, nelas iremos informar que o Bootstrap está disponível para utilizarmos.

Para garantirmos que o Bootstrap está realmente instalado, favor clicar na pasta **node\_modules** e procurar pela pasta do Bootstrap:

	Anotações



Garantindo que temos o Bootstrap à disposição, podemos configurar nosso arquivo **angular.json** da seguinte maneira:

```
26 "styles": [  
27   "node_modules/bootstrap/dist/css/bootstrap.css",  
28   "src/styles.css"  
29 ],  
30 "scripts": [  
31   "node_modules/jquery/dist/jquery.js",  
32   "node_modules/popper.js/dist/umd/popper.js",  
33   "node_modules/bootstrap/dist/js/bootstrap.js"  
34 ]
```

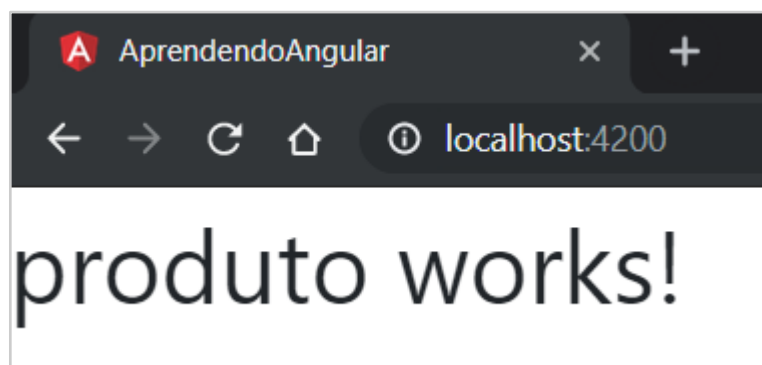
As linhas 27, 31, 32 e 33 foram adicionadas manualmente, referenciando o Bootstrap instalado anteriormente. Agora nossa aplicação terá acesso a estilização e aos componentes interativos como menus, banners, modais, entre outras funcionalidades.

Vamos fazer um teste, para isso podemos adicionar nosso componente de produto como o principal, para isso abra o arquivo **app.component.html** e implemente a seguinte estrutura:

Anotações	

```
<> app.component.html ×
1  <app-produto></app-produto>
2
```

Veja o resultado, nosso texto estará com outra fonte e nosso elemento body sem as margens:



	Anotações



## Capítulo 06 - Ng-if

A diretiva **\*ngIf** é muito utilizada para exibir elementos, como caixa de mensagens ou partes específicas da sua página web, para elaborarmos esse exemplo, podemos criar uma variável chamada **gostandoDeAngular** como boolean valendo verdadeiro:

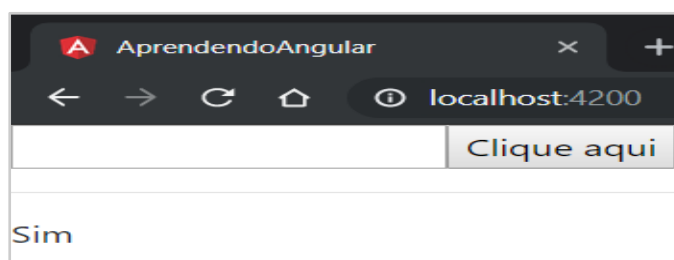
```
8  export class ProdutoComponent implements OnInit {
9
10     //Variáveis
11     texto:string = "Aprendendo Angular";
12     url:string = "http://apexensino.com.br";
13     imagem:string = "http://lorempixel.com/400/200";
14     campo:string;
15     gostandoDeAngular:boolean = true;
16
17     //Construtor
18     constructor() { }
19
20     //Inicialização
21     ngOnInit() { }
22
23     //Método
24     mensagem():void{
25         alert(this.campo);
26     }
```

Na linha 18 e 19 as diretivas verdadeiras e falsas são criadas com a seguinte estrutura no arquivo

HTML:

```
13  <input type="text" [(ngModel)]="campo">
14  <button (click)="mensagem()">Clique aqui</button>
15
16  <hr>
17
18  <p *ngIf="gostandoDeAngular">Sim</p>
19  <p *ngIf="!gostandoDeAngular">Não</p>
```

Criaremos dois **\*ngIf**, um caso seja true e outro caso seja false, o resultado será esse:



Anotações	



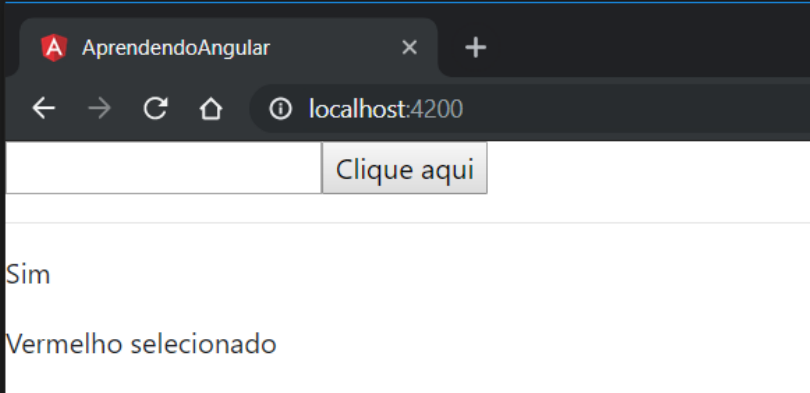
## Capítulo 07 - Ng-switch

Há uma opção parecida com o **\*ngIf**, que é o **[ngSwitch]**, que nos fornece uma estrutura com várias opções de resposta, diferente de um **\*ngIf**, que trabalha apenas com verdadeiro ou false, para exemplificarmos, crie no seu arquivo **produto.component.ts** a seguinte variável na linha 16:

```
10 //Variáveis
11 texto:string = "Aprendendo Angular";
12 url:string = "http://apexensino.com.br";
13 imagem:string = "http://lorempixel.com/400/200";
14 campo:string;
15 gostandoDeAngular:boolean = true;
16 cor:string = "vermelho";
```

No nosso arquivo HTML, vamos adicionar a seguinte estrutura a partir da linha 22:

```
21
22 <div [ngSwitch]="cor">
23   <p *ngSwitchCase="'azul'">Azul selecionado</p>
24   <p *ngSwitchCase="'verde'">Verde selecionado</p>
25   <p *ngSwitchCase="'amarelo'">Amarelo selecionado</p>
26   <p *ngSwitchCase="'vermelho'">Vermelho selecionado</p>
27 </div>
28
29
30
31
32
33
```



	Anotações



Na linha 22 iniciamos nossa validação selecionando uma variável, em cada elemento de parágrafo é utilizando um **\*ngSwitchCase**, assim será verificado se a cor que está entre os apóstrofes são iguais. Note que utilizamos apóstrofes dentro das aspas, pois são informações textuais, caso não estivesse entre apóstrofes, o Angular iria entender que há variáveis chamadas azul, verde, amarelo e vermelho.

Anotações	

## Capítulo 08 - Ng-for

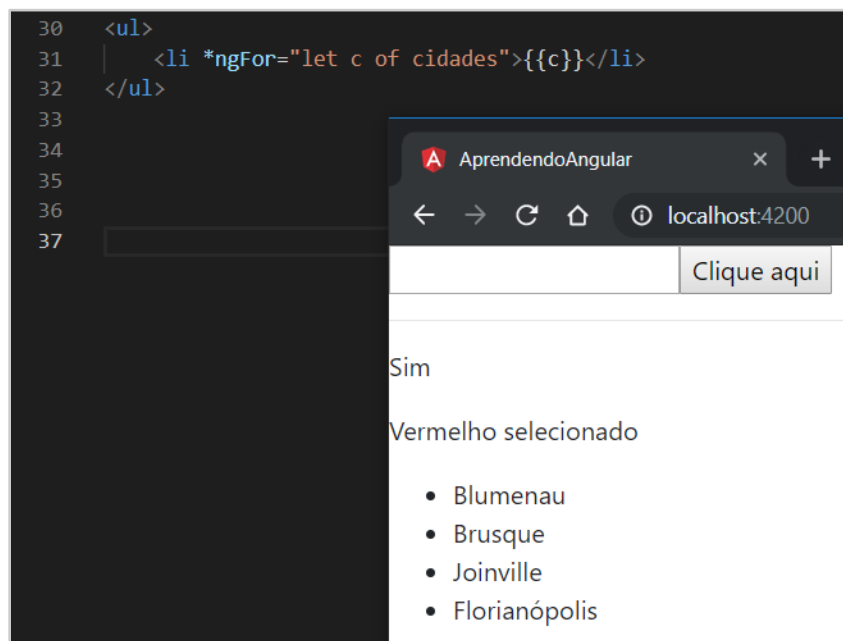
O comando **\*ngFor**, tem a função de criar um laço de repetição, vamos supor que temos um vetor em nosso arquivo TypeScript:

```

8   export class ProdutoComponent implements OnInit {
9
10    //Variáveis
11    texto:string = "Aprendendo Angular";
12    url:string = "http://apexensino.com.br";
13    imagem:string = "http://lorempixel.com/400/200";
14    campo:string;
15    gostandoDeAngular:boolean = true;
16    cor:string = "vermelho";
17    cidades:string[] = ["Blumenau", "Brusque", "Joinville", "Florianópolis"];

```

Na linha 17 é criado um vetor contendo quatro cidades, já no arquivo HTML, basta adicionarmos o comando em qualquer tag, neste exemplo será utilizada a tag `<ul>`:



As quatro cidades são exibidas de maneira bem simples, isso utilizando um simples array, mas você pode utilizar também um JSON por exemplo.

	Anotações



## Capítulo 09 - Ng-class

Imagine que você possua duas classes para utilizar em uma determinada estrutura, ao invés de fazer dois `*ngIf` ou um `[ngSwitch]`, você pode implementar o uso de classes, para exemplificarmos crie uma variável chamada `fonte`, iniciando com `true`:

```
8   export class ProdutoComponent implements OnInit {
9
10    //Variáveis
11    texto:string = "Aprendendo Angular";
12    url:string = "http://apexensino.com.br";
13    imagem:string = "http://lorempixel.com/400/200";
14    campo:string;
15    gostandoDeAngular:boolean = true;
16    cor:string = "vermelho";
17    cidades:string[] = ["Blumenau", "Brusque", "Joinville", "Florianópolis"];
18    fonte:boolean = true;
19
20    //Construtor
21    constructor() { }
```

E no HTML podemos implementar uma condicional, dependendo do valor da nossa variável `fonte`:

```
34  <style>
35    .fonteVermelha{
36      color: ■ red;
37    }
38
39    .fonteAzul{
40      color: ■ blue;
41    }
42  </style>
43
44  <h1 [className]="!fonte ? 'fonteVermelha' : 'fonteAzul'">Classes no Angular</h1>
```

Entre as linhas 34 e 42, foram criadas duas classes do CSS para alterar a cor de fonte, já na linha 44 criamos uma diretiva de classe, onde realizamos uma condicional, se verdadeiro retorna a fonte vermelha, caso contrário a fonte azul.

	Anotações



## Capítulo 10 - Módulo HTTP

O módulo HTTP têm como finalidade desenvolver requisições ao servidor, podendo inserir, selecionar, alterar ou excluir dados que estejam em algum array, json ou banco de dados.

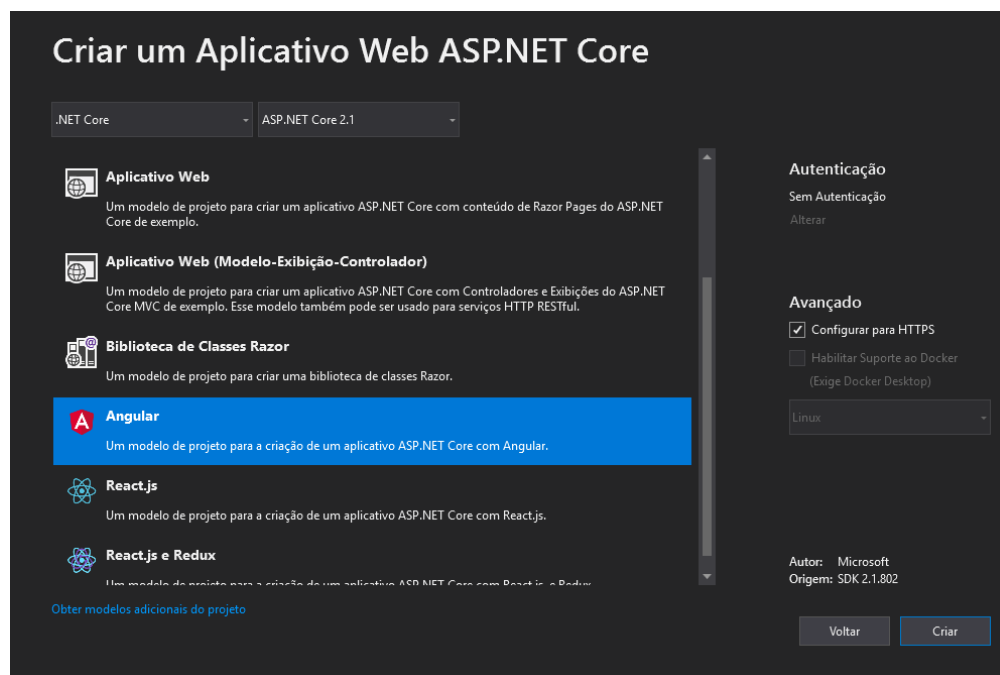
A utilização do módulo HTTP também facilita a ter acesso a sistemas desenvolvidos por terceiros sem o acesso ao servidor ou ao banco de dados, apenas tendo o endereço do site, já é possível realizar ações desse tipo.

Anotações	

## Capítulo 11 - Funcionamento do Angular no .Net Core

Quando abrimos nosso Visual Studio e escolhemos a opção web existem alguns templates que podemos escolher. Iremos marcar opção Angular, conforme ilustra a figura 1.

Figura 1

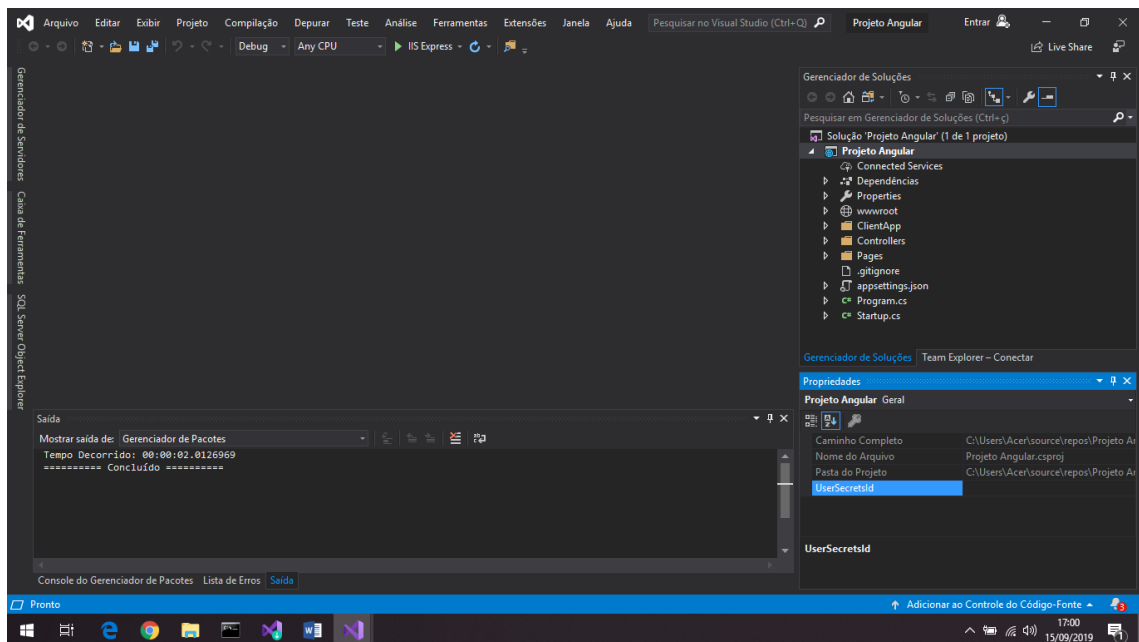


Após escolhido o template que vamos utilizar, ele nos cria o projeto conforme mostrado na figura 2, abaixo:

	Anotações

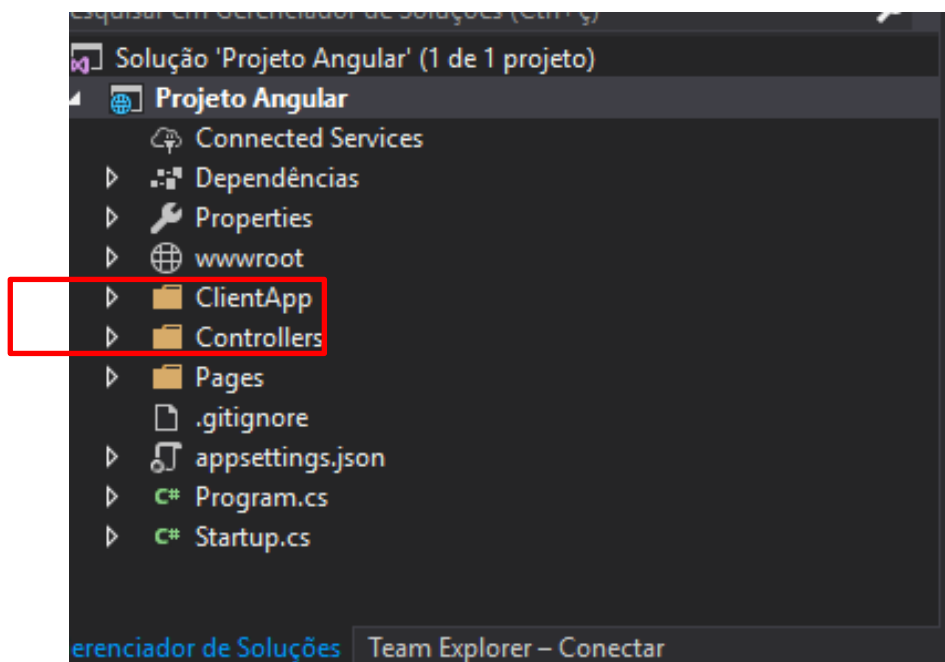


Figura 2



No nosso projeto podemos ver que temos a pasta Controller e a pasta ClientApp.

Figura 3



Anotações	



Na pasta ClientApp temos o nosso template em angular e na pasta Controller está o nossa regra de negócios do servidor que é onde a nossa aplicação angular irá se comunicar.

Para que possa ocorrer essa comunicação o nosso Controller irá funcionar como uma API. Por meio do Route gerenciamos o roteamento do Controller que estamos chamando.

E para os métodos utilizamos:

- HttpGet quando queremos buscar uma informação
- HttpPost quando queremos enviar uma informação
- HttpPut para alterar uma informação
- HttpDelete para deletar um registro

Conforme imagem abaixo:

Figura 4

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5  using Microsoft.AspNetCore.Mvc;
6
7  namespace Projeto_Angular.Controllers
8  {
9      [Route("api/[controller]")]
10     public class SampleDataController : Controller
11     {
12         private static string[] Summaries = new[]
13         {
14             "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"
15         };
16
17         [HttpGet("action")]
18         public IEnumerable<WeatherForecast> WeatherForecasts()
19         {
20             var rng = new Random();
21             return Enumerable.Range(1, 5).Select(index => new WeatherForecast
22                 {
23                     Date = DateTime.Now.AddDays(index),
24                     TemperatureC = rng.Next(-20, 50),
25                     Summary = Summaries[rng.Next(Summaries.Length)]
26                 });
27         }
28     }
29 }

```

	Anotações



## Capítulo 12 - Comunicando a aplicação Angular com .Net Coe

Para que as duas aplicações se comuniquem existe uma configuração que é feita no arquivo startup no método configure que é o app.useSpa. Dentro dele, ele está mandando executar um escript no nível da nossa aplicação angular por meio do soucepath. Conforme imagem abaixo:

Figura 5

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseStaticFiles();
    app.UseSpaStaticFiles();

    app.UseMvc(routes =>
    {
        routes.MapRoute(
            name: "default",
            template: "{controller}/{action=Index}/{id?}");
    });

    app.UseSpa(spa =>
    {
        // To learn more about options for serving an Angular SPA from ASP.NET Core,
        // see https://go.microsoft.com/fwlink/?linkid=864501

        spa.Options.SourcePath = "ClientApp";

        if (env.IsDevelopment())
        {
            spa.UseAngularCliServer(npmScript: "start");
        }
    });
}
```

Se entrarmos na nossa aplicação dentro de ClientApp, veremos que dentro dela temos um arquivo chamado de package.json. Dentro desse arquivo podemos ver que ele possui um objeto scripts e dentro dele uma parametro start.

Quando executamos a aplicação ele executa esse comando junto, então executando o ngserve que é o comando do angular para iniciar o servidor.

## Capítulo 13 - Fazendo requisições via Rest para o Controller

Para que possamos fazer as requisições via rest para o nosso Controller precisamos importar o modulo HttpClient, conforme mostrado na figura abaixo:

Figura 6

```

1 import { Component, Inject } from '@angular/core';
2 import { HttpClient } from '@angular/common/http';
3
4 @Component({
5   selector: 'app-fetch-data',
6   templateUrl: './fetch-data.component.html'
7 })
8 export class FetchDataComponent {
9   public forecasts: WeatherForecast[];
10
11   constructor(http: HttpClient, @Inject('BASE_URL') baseUrl: string) {
12     http.get<WeatherForecast[]>(baseUrl + 'api/SampleData/WeatherForecasts').subscribe(result => {
13       this.forecasts = result;
14     }, error => console.error(error));
15   }
16 }
17
18 interface WeatherForecast {
19   dateFormatted: string;
20   temperatureC: number;
21   temperatureF: number;
22   summary: string;
23 }
24

```

Após importarmos ela em nossa página, temos que configurar ela em nosso arquivo app-module.ts. Como ele é um modulo temos que inseri-lo dentro de imports.

Obs: Caso esse passo não seja feito a aplicação não funcionará!

	Anotações



Figura 7

```

C:\Users\Acer\source\repos\Projeto Angular\Projeto Angular\src\app\app.module.ts
1  import { NgModule } from '@angular/core';
2  import { FormsModule } from '@angular/forms';
3  import { HttpClientModule } from '@angular/common/http';
4  import { RouterModule } from '@angular/router';
5
6  import { AppComponent } from './app.component';
7  import { NavMenuComponent } from './nav-menu/nav-menu.component';
8  import { HomeComponent } from './home/home.component';
9  import { CounterComponent } from './counter/counter.component';
10 import { FetchDataComponent } from './fetch-data/fetch-data.component';
11
12 @NgModule({
13   declarations: [
14     AppComponent,
15     NavMenuComponent,
16     HomeComponent,
17     CounterComponent,
18     FetchDataComponent
19   ],
20   imports: [
21     BrowserModule.withServerTransition({ appId: 'ng-cli-universal' }),
22     HttpClientModule,
23     FormsModule,
24     RouterModule.forRoot([
25       { path: '', component: HomeComponent, pathMatch: 'full' },
26       { path: 'counter', component: CounterComponent },
27       { path: 'fetch-data', component: FetchDataComponent },
28     ])
29   ],
30   providers: [],
31   bootstrap: [AppComponent]
32 })
33 export class AppModule {}
34
35
```

Agora para que possamos interagir com o nosso controller, basta passarmos nossa requisição para ele de acordo com o Route do Controller e o atributo daquele método, conforme mostrado nas imagens abaixo:

Figura 8

```

C:\Users\Acer\source\repos\Projeto Angular\Projeto Angular\src\app\fetch-data.component.ts
1  import { Component, Inject } from '@angular/core';
2  import { HttpClient } from '@angular/common/http';
3
4  @Component({
5    selector: 'app-fetch-data',
6    templateUrl: './fetch-data.component.html'
7  })
8  export class FetchDataComponent {
9    public forecasts: WeatherForecast[];
10
11    constructor(http: HttpClient, @Inject('BASE_URL') baseUrl: string) {
12      http.get<WeatherForecast[]>(baseUrl + 'api/SampleData/WeatherForecasts').subscribe(result => {
13        this.forecasts = result;
14      }, error => console.error(error));
15    }
16  }
17
18  interface WeatherForecast {
19    dateFormatted: string;
20    temperatureC: number;
21    temperatureF: number;
22    summary: string;
23  }
24
```

Anotações	

Figura 9

```

[Route("api/[controller]")]
public class SampleDataController : Controller
{
    private static string[] Summaries = new[]
    {
        "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering"
    };

    [HttpGet("action")]
    public IEnumerable<WeatherForecast> WeatherForecasts()
    {
        var rng = new Random();
        return Enumerable.Range(1, 5).Select(index => new WeatherForecast
        {
            DateFormatted = DateTime.Now.AddDays(index).ToString("d"),
            TemperatureC = rng.Next(-20, 55),
            Summary = Summaries[rng.Next(Summaries.Length)]
        });
    }
}

```

Como podemos ver na imagem 8 estamos utilizando o api/SampleData, que corresponde ao nosso Route api/[controller] e logo em seguida colocamos o nosso método que seria o [action] ficando api/SampleData/Weatherforecasts, por meio de uma requisição HttpGet

	Anotações

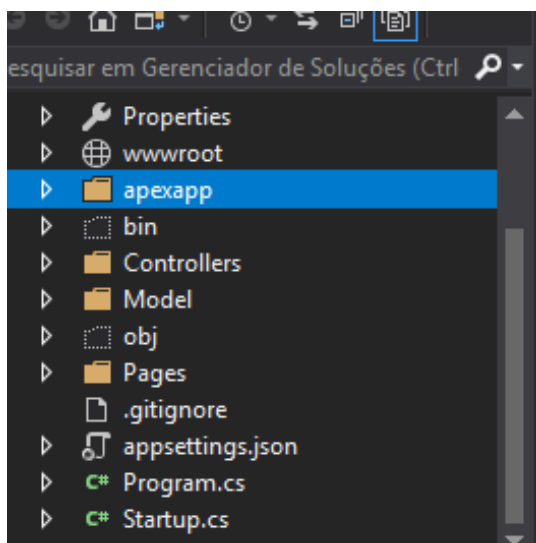


Anotações	

## Capítulo 14 - Criando projeto com angular com .net core

Nesse primeiro momento do projeto iremos excluir a nossa pasta ClientApp do projeto e iremos iniciar um novo projeto chamado apexapp nosso diretório ficará assim.

Figura 10



Nesse primeiro momento criamos o nosso UsuariosController que aonde a nossa aplicação fará a comunicação.

	Anotações



Figura 11

```
namespace ProjetoAngular.Controllers
{
    [Route("api/[controller]")]
    public class UsuariosController: Controller
    {
        [HttpGet("[action]")]
        public List<Usuarios> RetornaUsuarios()
        {
            return new UsuariosDAO().RetornaUsuarios();
        }
        [HttpPost("[action]")]
        public void Inserir([FromBody]Usuarios usuarios)
        {
            new UsuariosDAO().InseriUsuario(usuarios.Nome,usuarios.Idade);
        }
        [HttpDelete("[action]/{id}")]
        public void Delete(int id)
        {
            new UsuariosDAO().Delete(id);
        }
        [HttpGet("[action]/{id}")]
    }
}
```

Feito essa parte implementamos o nosso UsuariosDAO que fará a nossa comunicação com o banco de dados.

Figura 12

```
{
    string sql = @"SELECT * FROM USUARIOS";
    using (var conn = new SqlConnection(conexao))
    {
        conn.Open();
        SqlCommand cmd = new SqlCommand(sql, conn);
        SqlDataReader reader = cmd.ExecuteReader();
        var lista = new List<Usuarios>();
        while (reader.Read())
        {
            Usuarios u = new Usuarios(
                Convert.ToInt32(reader["ID"]),
                reader["NOME"].ToString(),
                Convert.ToInt32(reader["IDADE"])
            );
            lista.Add(u);
        }
        return lista;
    }
}

public void InseriUsuario(string nome, int idade)
{
    string sql = string.Format(@"INSERT INTO USUARIOS (NOME,IDADE) VALUES ('{0}','{1}");
    using (var conn = new SqlConnection(conexao))
    {
        conn.Open();
        SqlCommand cmd = new SqlCommand(sql,conn);
        cmd.ExecuteNonQuery();
    }
}
```

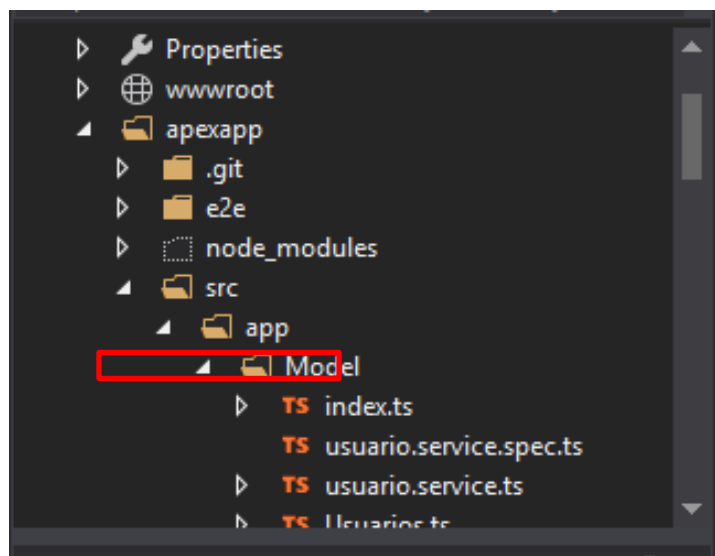
Com a nossa regra de negócios já definida no C# vamos para o angular configurar nossas telas.

Anotações	



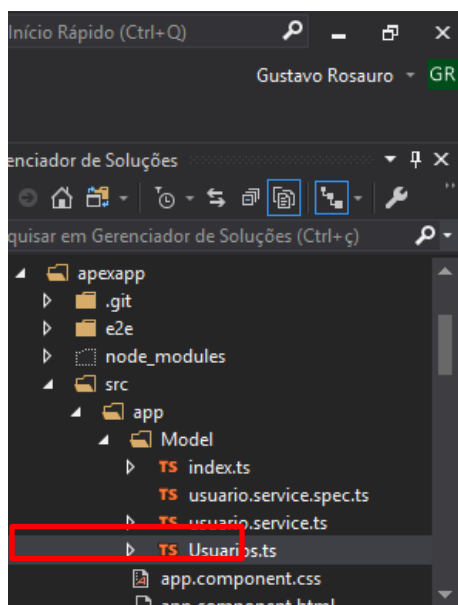
Clicando dentro de apexapp/src/app criamos uma pasta chamada model, conforme imagem abaixo:

Figura 13



Após criada essa pasta criamos agora um typescript com o nome usuario.ts e dentro dele criamos uma classe com os seguintes atributos:

Figura 14



	Anotações

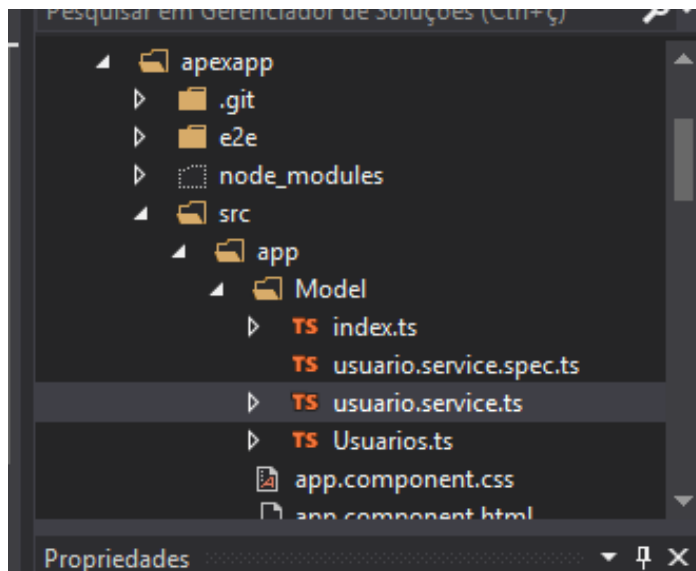


Figura 15

```
export class Usuarios {  
  id: number;  
  nome: string;  
  idade: number;  
}
```

Feito isso vamos configurar agora nossas requisições dentro de um arquivo chamado `servisse.usuario.ts`, conforme imagem abaixo:

Figura 16



Anotações	

Figura 17

```

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

@Injectable({
  providedIn: 'root'
})
export class UsuarioService {
  constructor(private usuarios: Usuarios, private http: HttpClient) { }

  RetornaUsuarios() {
    return this.http.get<Usuarios[]>("/api/Usuarios/RetornaUsuarios");
  }

  InserirUsuario(usuario: Usuarios) {
    return this.http.post("/api/Usuarios/Inserir", usuario);
  }

  DeleteUsuario(id: number) {
    return this.http.delete("/api/Usuarios/Delete/" + id);
  }

  GetUserById(id: number) {
    return this.http.get<Usuarios>("/api/Usuarios/ReturnUser/" + id);
  }

  UpdateUser(usuario: Usuarios) {
    return this.http.put("/api/Usuarios/Atualiza", usuario);
  }
}

```

Configurando as nossas requisições rest para o nosso C# agora é só configurar como será feito a chamada dos métodos em tela.

Chamando o os usuários cadastrados em tela para que possamos chamar esses usuários temos de configurar um método para que possamos retorna-los em tela. Para isso vamos em nosso app-component, ts e criamos um método chamado retornaUsuarios.

	Anotações



Figura 18

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  title = 'apexapp';
  constructor(private service: UsuarioService) { }
  usuarios: any = [];
  usuario: any;
  button: any;
  ngOnInit() {
    this.button = "Salvar";
    this.usuario = new Usuarios;
    this.retornaUsuarios();
  }
  retornaUsuarios(): void {
    this.service.RetornaUsuarios().subscribe(result => {
      this.usuarios = result;
    }, error => console.log(error))
  }
  exibiid(id: number) {
    this.service.DeleteUsuario(id).subscribe(result => {
      this.retornaUsuarios();
      this.usuario = new Usuarios;
    })
  }
}
```

Para que possamos fazer a chamada do nosso método configuramos o nosso service criado anteriormente dentro do nosso constructor, conforme mostrado na figura anterior.

Se observamos na figura anterior temos uma lista criada abaixo do nosso construtor, com o nome usuarios. Essa lista é preenchida no retorno do método retornaUsuarios, que irá retornar para uma lista de usuários.

Após essa lista ser preenchida vemos que seu método é chamado dentro da função ngOnInit, que é uma função que é executada antes de carregar a nossa tela.

Então quando a nossa tela é carregada temos essa lista em memória e podemos utilizá-la front por meio de um ngFor conforme imagem abaixo:

Anotações	

Figura 19

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
</div>
<label>Nome</label>
<input class="form-control" [(ngModel)]="usuario.nome"/>
<label>Idade</label>
<input class="form-control" [(ngModel)]="usuario.idade"/>
<button class="btn btn-success" (click)="Inseri(usuario)">{{button}}</button>
<table class="table">
  <thead>
    <tr>
      <th>Nome</th>
      <th>Idade</th>
    </tr>
  </thead>
  <tbody *ngFor="let usuario of usuarios">
    <tr>
      <td>{{usuario.nome}}</td>
      <td>{{usuario.idade}}</td>
      <td><button class="btn btn-danger" (click)="exibiid(usuario.id)">Deletar</button></td>
      <td><button class="btn btn-warning" (click)="SearchUserById(usuario.id)">Alterar</button></td>
    </tr>
  </tbody>
</table>
```

Nesse momento estamos preenchendo a nossa table com os dados carregados lá nosso typescript.

No nosso ngFor iremos criar uma variável como no exemplo. O nosso let usuario que em tempo de execução irá receber um item da lista usuários, e a cada repetição e iremos exibir dentro das chaves conforme o exemplo {{usuario.nome}}.

E caso queiramos salvar um usuário, criamos uma variável no nosso typescript que iremos dar o valor da nossa classe para que possamos trabalhar com o objeto no front, conforme figura abaixo.

	Anotações



Figura 20

```
1 import { Component, OnInit } from '@angular/core';
2 import { UsuarioService, Usuarios } from './Model';
3
4 @Component({
5   selector: 'app-root',
6   templateUrl: './app.component.html',
7   styleUrls: ['./app.component.css']
8 })
9 export class AppComponent implements OnInit {
10   title = 'apexapp';
11   constructor(private service: UsuarioService) { }
12   usuarios: any = [];
13   usuario: any;
14   button: any;
15   ngOnInit() {
16     this.button = "Salvar";
17     this.usuario = new Usuarios();
18     this.retornaUsuarios();
19   }
20   retornaUsuarios(): void {
21     this.service.RetornaUsuarios().subscribe(result => {
22       this.usuarios = result;
23     }, error => console.log(error))
24   }
25   exibiId(id: number) {
```

Agora o nosso projeto está iniciando com a variável usuário carregando o objeto em tela.

Esse objeto será usado com o ngModel no nosso html, para que cada tag receba o nosso parametro do objeto, conforme a ilustra a imagem abaixo:

Figura 21

```
<!--The content below is only a placeholder and can be replaced.-->
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
</div>
<label>Nome</label>
<input class="form-control" [(ngModel)]="usuario.nome"/>
<label>Idade</label>
<input class="form-control" [(ngModel)]="usuario.idade"/>
<button class="btn btn-success" (click)="Inseri(usuario)">{{button}}</button>
<table class="table">
```

Feito essa parte com o nosso objeto em tela, quando clicarmos no salvar iremos chamar o método inserir, que terá dentro dele. O nosso usuário já carregado e fará a inserção.

Anotações	

Figura 22

```

    }, error => console.log(error))
  }
  exibiid(id: number) {
    this.service.DeleteUsuario(id).subscribe(result => {
      this.retornaUsuarios();
      this.usuario = new Usuarios;
      this.button = "Cadastrar";
    }, error => console.log(error));
  }
  Inserir(usuario: Usuarios) {
    if (usuario.id != null) {
      this.Atualiza(usuario);
    } else {
      this.service.InserirUsuario(usuario).subscribe(result => {
        this.retornaUsuarios();
      }, error => console.log(error));
    }
  }
  SearchUserById(id: number) {
    this.service.GetUserById(id).subscribe(result => {
      this.usuario = result;
    }, error => console.log(error));
    this.button = "Alterar";
  }
}

```

Nesse momento estamos mandando a requisição para o c# com o nosso objeto preenchido em tela.

Para deletar o nosso usuário temos clicar no botão deletar que está junto na nossa tabela. Perceba que dentro do evento clique temos uma função deletarUser já com o id do usuário que queremos deletar, conforme ilustra a figura abaixo:

Figura 23

```

</tr>
</thead>
<tbody *ngFor="let usuario of usuarios">
  <tr>
    <td>{{usuario.nome}}</td>
    <td>{{usuario.idade}}</td>
    <td><button class="btn btn-danger" (click)="deletaUser(usuario.id)">Deletar</button></td>
    <td><button class="btn btn-warning" (click)="SearchUserById(usuario.id)">Alterar</button></td>
  </tr>
</tbody>
</table>
<router-outlet></router-outlet>

```

Esse método no nosso evento clique quando apertarmos irá chamar a função deletaUser passando o id do nosso usuário como parâmetro.

	Anotações



Abaixo método deleteUser, conforme a figura:

Figura 24

```
deletaUser(id: number) {  
  this.service.DeleteUsuario(id).subscribe(result => {  
    this.retornaUsuarios();  
    this.usuario = new Usuarios;  
    this.button = "Cadastrar";  
  }, error => console.log(error));  
}
```

Feito isso temos o método searchUserById que irá encontrar o usuário pelo valor do seu id e logo em seguida preenche a nossa tela.

Figura 25

```
SearchUserById(id: number) {  
  this.service.GetUserById(id).subscribe(result => {  
    this.usuario = result;  
  }, error => console.log(error));  
  this.button = "Alterar";  
}
```

Com o nosso usuário em tela podemos alterar o registro informado.

Então o próximo evento é responsável por fazer a alteração do nosso usuário:

Figura 26

```
Atualiza(usuario: Usuarios) {  
  this.service.UpdateUser(usuario).subscribe(result => {  
    this.retornaUsuarios();  
    this.usuario = new Usuarios;  
    this.button = "Cadastrar";  
  }, error => console.log(error))  
}
```

Anotações	



	Anotações