

Angular

Angular

- Angular é uma plataforma e framework para construção da interface de aplicações usando HTML, CSS e, principalmente, JavaScript, criada pelos desenvolvedores da Google.
- Possui alguns elementos básicos que tornam essa construção interessante.
- Dentre os principais, podemos destacar os componentes, templates, diretivas, roteamento, módulos, serviços, injeção de dependências e ferramentas de infraestrutura que automatizam tarefas, como a de executar os testes unitários de uma aplicação.

Angular

- Alguns outros pontos dessa plataforma que merecem destaque são o fato de que ela é open source, possui uma grande comunidade, existem várias empresas utilizando e tem muito material de estudo para quem deseja se aperfeiçoar.

Angular CLI

- Angular CLI é a ferramenta oficial para inicializar e trabalhar com projetos em Angular.
- Depois de instalar a Angular CLI, você precisa executar um comando para gerar um projeto e outro para servi-lo usando um servidor de desenvolvimento local para rodar sua aplicação.
- Como ocorre com a maioria das ferramentas de front-end nos dias de hoje, a Angular CLI foi criada com base no Node.js.

Node.js

- O Node.js é uma tecnologia de servidor que permite que você execute o JavaScript no servidor e crie aplicações da web no lado do servidor.
- No entanto, o Angular é uma tecnologia de front-end.
- Portanto, mesmo que você precise instalar o Node.js em sua máquina de desenvolvimento, é apenas para a execução da CLI.

Instalando Angular

- Primeiro, você precisa ter o Node e o npm instalados na sua máquina de desenvolvimento. Existem muitas maneiras de se fazer isso, tais como:
 - usar o NVM (Node Version Manager) para a instalação e trabalho com muitas versões do Node em seu sistema
 - usar o gerenciador de pacotes oficial do seu sistema operacional
 - instalar a partir do site oficial da web.
 - <https://nodejs.org/en/download/>

Instalando Angular

- Certifique-se de que o Node esteja instalado em seu sistema executando o comando abaixo em um prompt de comando, que deverá exibir a versão instalada do Node:
 - `$ node -v`
- Depois, execute o seguinte comando para instalar a Angular CLI:
 - `$ npm install @angular/cli`

AngularCLI

- Depois de instalar a Angular CLI, você pode executar diversos comandos. Começamos conferindo a versão instalada da CLI:
 - `$ ng version`
- Um segundo comando que talvez você queira executar é o comando `help`, que ajudará a obter uma lista de comandos a serem usados:
 - `$ ng help`

A CLI oferece os comandos a seguir:

- `add`: adiciona suporte a uma biblioteca externa em seu projeto.
- `build (b)`: compila uma aplicação do Angular em seu diretório de saída, chamado `dist/` no caminho de saída fornecido. Este comando deve ser executado dentro de um diretório de espaço de trabalho.
- `config`: obtém ou configura os valores de configuração do Angular.
- `doc (d)`: abre a documentação oficial do Angular (angular.io) em um navegador, buscando uma palavra-chave determinada.
- `e2e (e)`: faz o `build` e serve uma aplicação do Angular, depois executa testes de ponta a ponta usando o Protractor.

A CLI oferece os comandos a seguir:

- `generate (g)`: gera e/ou modifica arquivos com base em um esquema.
- `help`: lista os comandos disponíveis e suas descrições breves.
- `lint (l)`: executa as ferramentas de linting no código da aplicação em Angular em uma determinada pasta de projeto.
- `new (n)`: cria um espaço de trabalho e uma aplicação inicial em Angular.
- `run`: executa um destino personalizado definido em seu projeto.

A CLI oferece os comandos a seguir:

- `serve (s)`: faz o build e serve sua aplicação, fazendo um novo build a cada alteração de arquivos.
- `test (t)`: executa os testes unitários em um projeto.
- `update`: atualiza sua aplicação e suas dependências. Consulte <https://update.angular.io/> (em inglês)
- `version (v)`: mostra a versão da Angular CLI.
- `xi18n`: extrai as mensagens i18n do código-fonte.

Criando um projeto novo

- Você pode usar a Angular CLI para gerar rapidamente seu projeto em Angular executando o seguinte comando em sua interface de linha de comando:
 - `$ ng new OiMundo`
- A CLI perguntará Would you like to add Angular routing? ("Quer adicionar o roteamento do Angular?"). Você pode responder com y (Sim) ou n (Não). Não, neste caso, é a opção padrão.
- Também será perguntado sobre o formato da folha de estilos (stylesheet) que você quer usar (por exemplo, o CSS). Escolha suas opções e pressione Enter para continuar.

Criando um projeto novo

- Depois disso, seu projeto estará criado com uma estrutura de diretórios e alguns arquivos com configurações e código.
- A maioria estará nos formatos TypeScript e JSON.

Criando um novo projeto

- Vejamos a função de cada diretório/arquivo:
 - `/e2e/`: contém os testes end-to-end (simulação do comportamento do usuário) do site
 - `/node_modules/`: todas as bibliotecas de terceiros são instaladas nesta pasta usando `npm install`
 - `/src/`: contém o código-fonte da aplicação. A maior parte do trabalho será feita aqui
 - `/app/`: contém módulos e componentes
 - `/assets/`: contém os ativos estáticos, como imagens, ícones e estilos

Criando um novo projeto

- Vejamos a função de cada diretório/arquivo:
 - `/environments/`: contém arquivos de configuração específicos do ambiente (produção e desenvolvimento)
 - `browserslist`: necessário para o autoprefixador para suporte ao CSS
 - `favicon.ico`: o favicon
 - `index.html`: o arquivo HTML principal
 - `karma.conf.js`: o arquivo de configuração para o Karma (uma ferramenta de testes)
 - `main.ts`: o arquivo inicial principal, a partir de onde o AppModule é iniciado

Criando um novo projeto

- Vejamos a função de cada diretório/arquivo:
 - polyfills.ts: polyfills necessários ao Angular
 - styles.css: o arquivo de folha de estilos (stylesheet) global do projeto
 - test.ts: um arquivo de configuração para o Karma
 - tsconfig.*.json: os arquivos de configuração para o TypeScript
 - angular.json: contém as configurações para a CLI

Criando um novo projeto

- Vejamos a função de cada diretório/arquivo:
 - package.json: contém as informações básicas do projeto (nome, descrição e dependências)
 - README.md: um arquivo em markdown que contém a descrição do projeto
 - tsconfig.json: o arquivo de configuração para o TypeScript
 - tslint.json: o arquivo de configuração para o TSlint (uma ferramenta de análise estática)

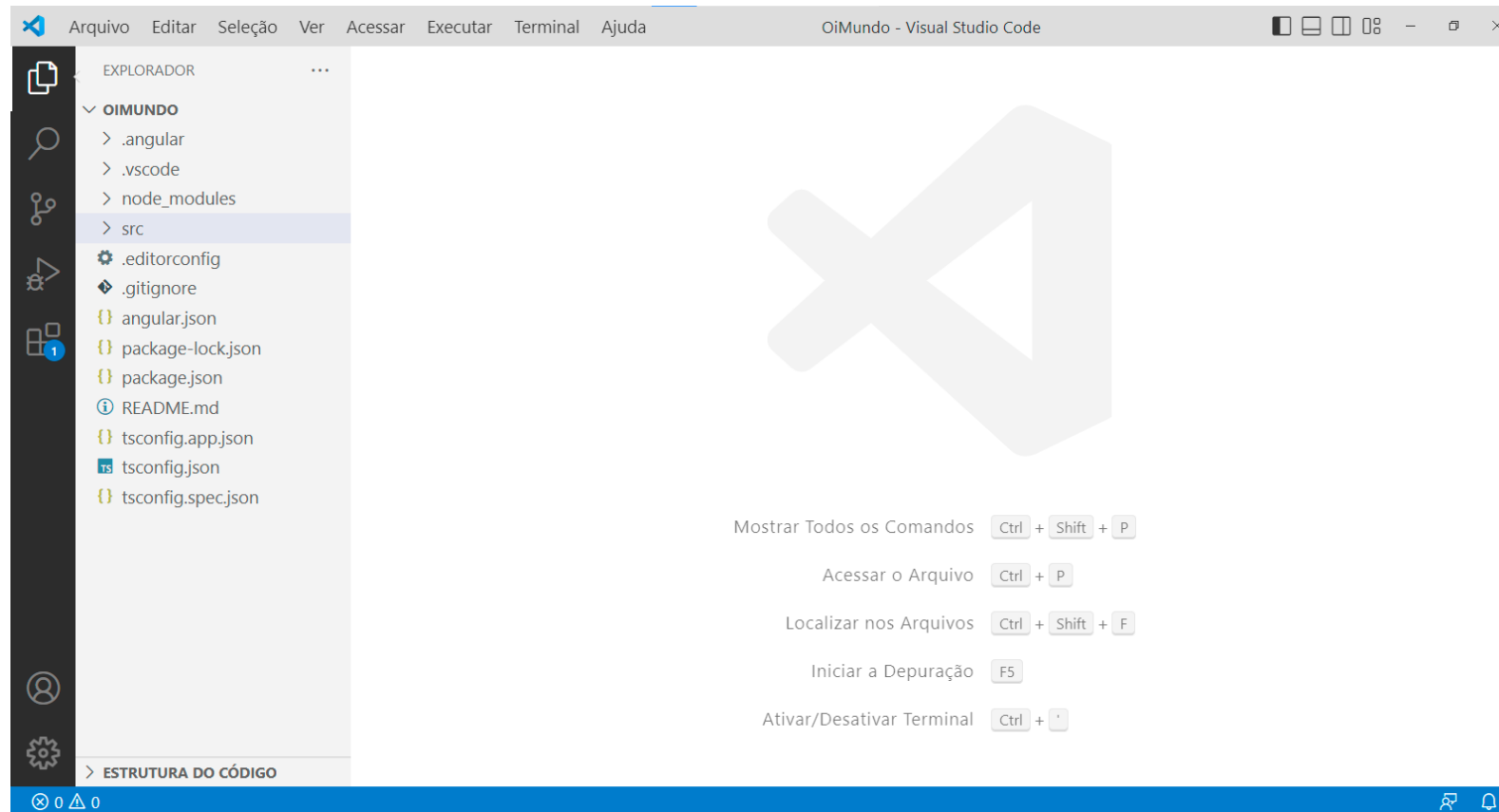
Como servir o projeto?

- A Angular CLI fornece um conjunto de ferramentas completo para desenvolver aplicações de front-end em sua máquina local.
- Assim, não é preciso instalar um servidor para servir o projeto—você pode, simplesmente, usar o comando `ng serve` a partir do terminal para servir seu projeto localmente.
 - `$ cd OiMundo`
 - `$ ng serve`

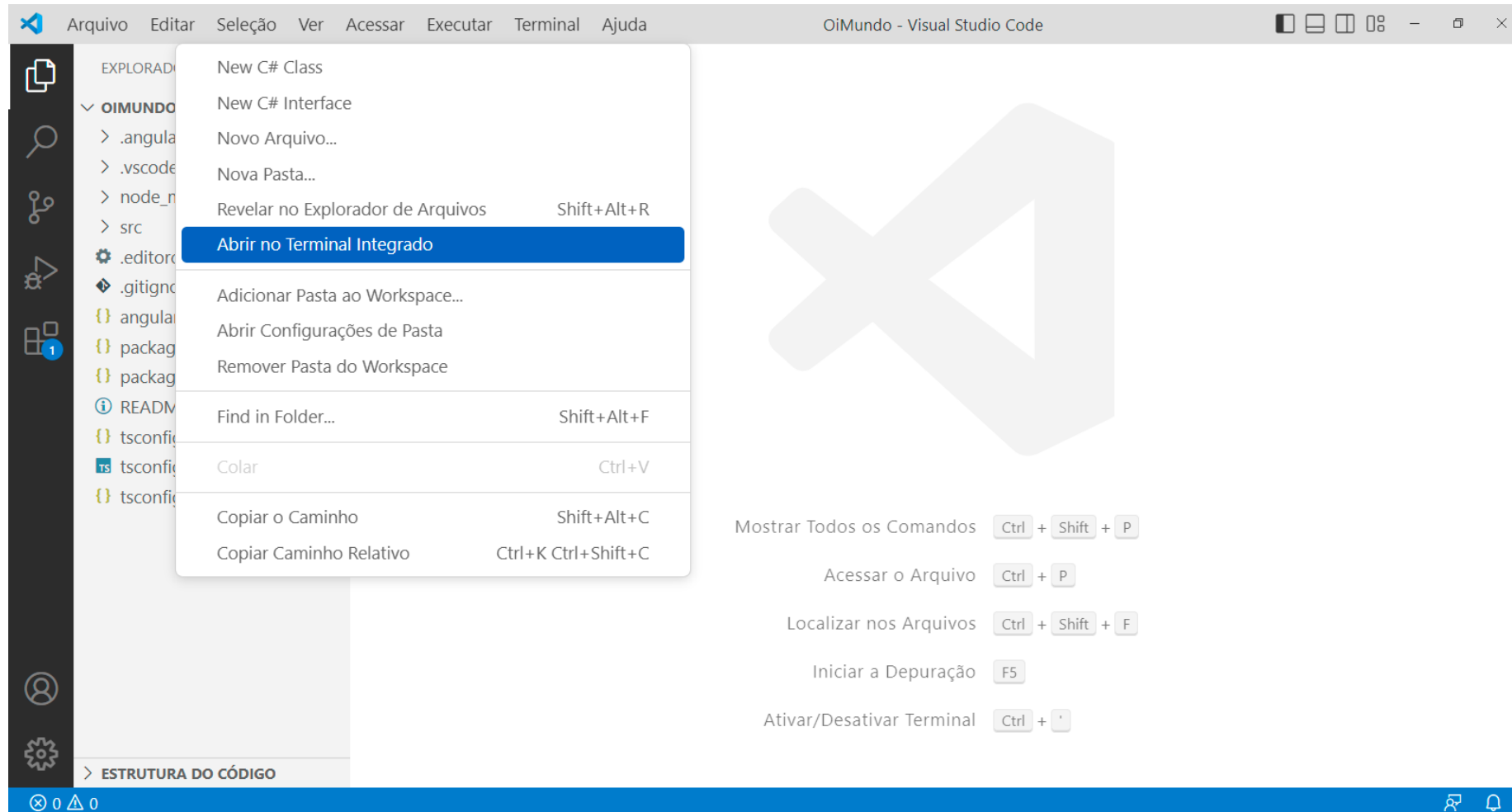
Como servir o projeto?

- Agora, você pode navegar até o endereço `http://localhost:4200/` para começar a mexer com sua aplicação de front-end.
- A página recarregará automaticamente caso você mude qualquer arquivo do código-fonte.
- Para abrir o projeto no Visual code digite:
 - `$ code .`

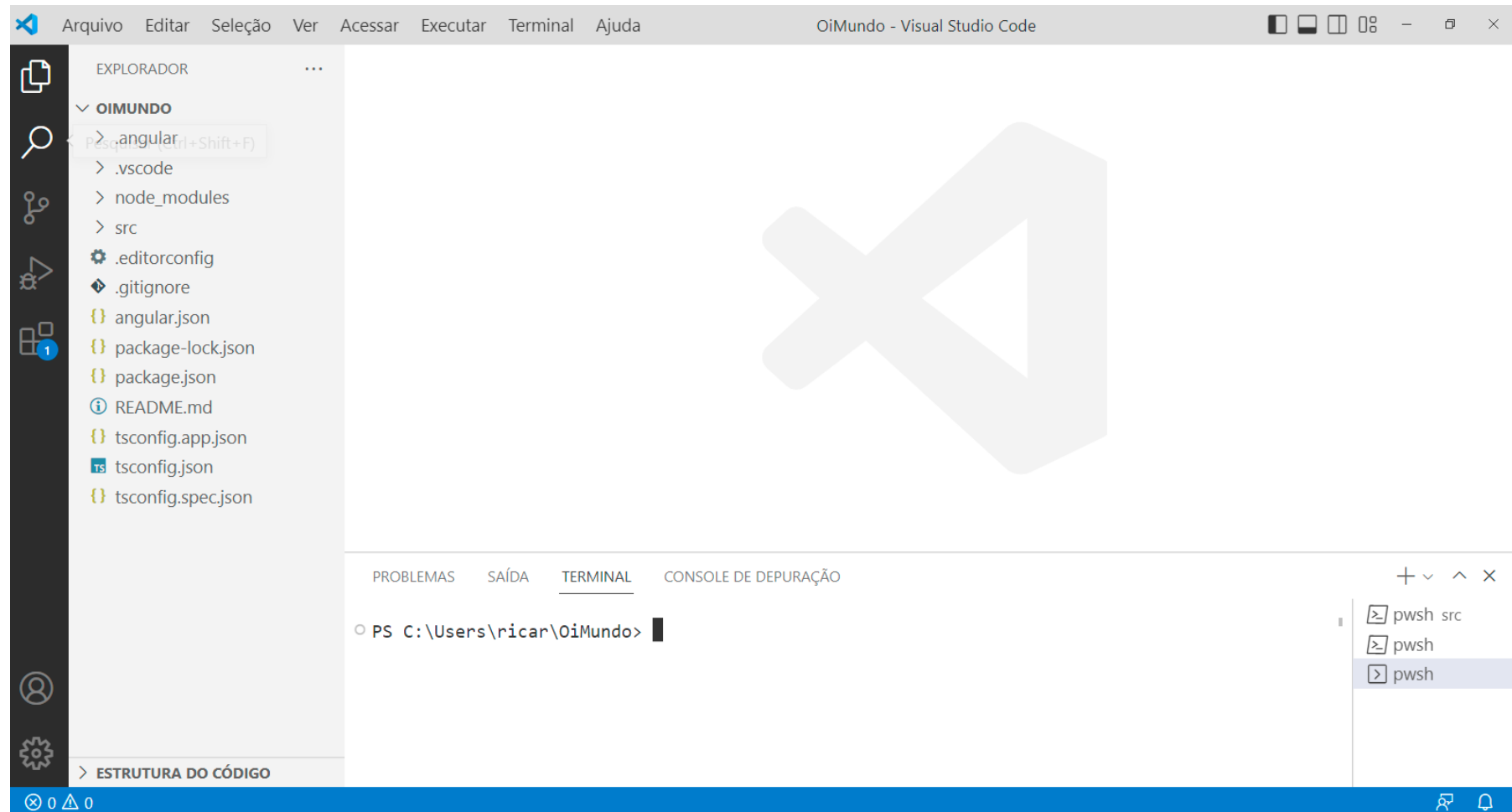
Visual code



Visual code



Visual code



Geração de artefatos do Angular

- A Angular CLI fornece um comando `ng generate` que ajuda os desenvolvedores a gerar artefatos básicos do Angular, como módulos, componentes, diretivas, pipes e serviços:
 - `$ ng generate component meu-componente`
- `meu-componente` é o nome do componente. A Angular CLI automaticamente adicionará uma referência a `components`, `directives` e `pipes` no arquivo `src/app.module.ts`.

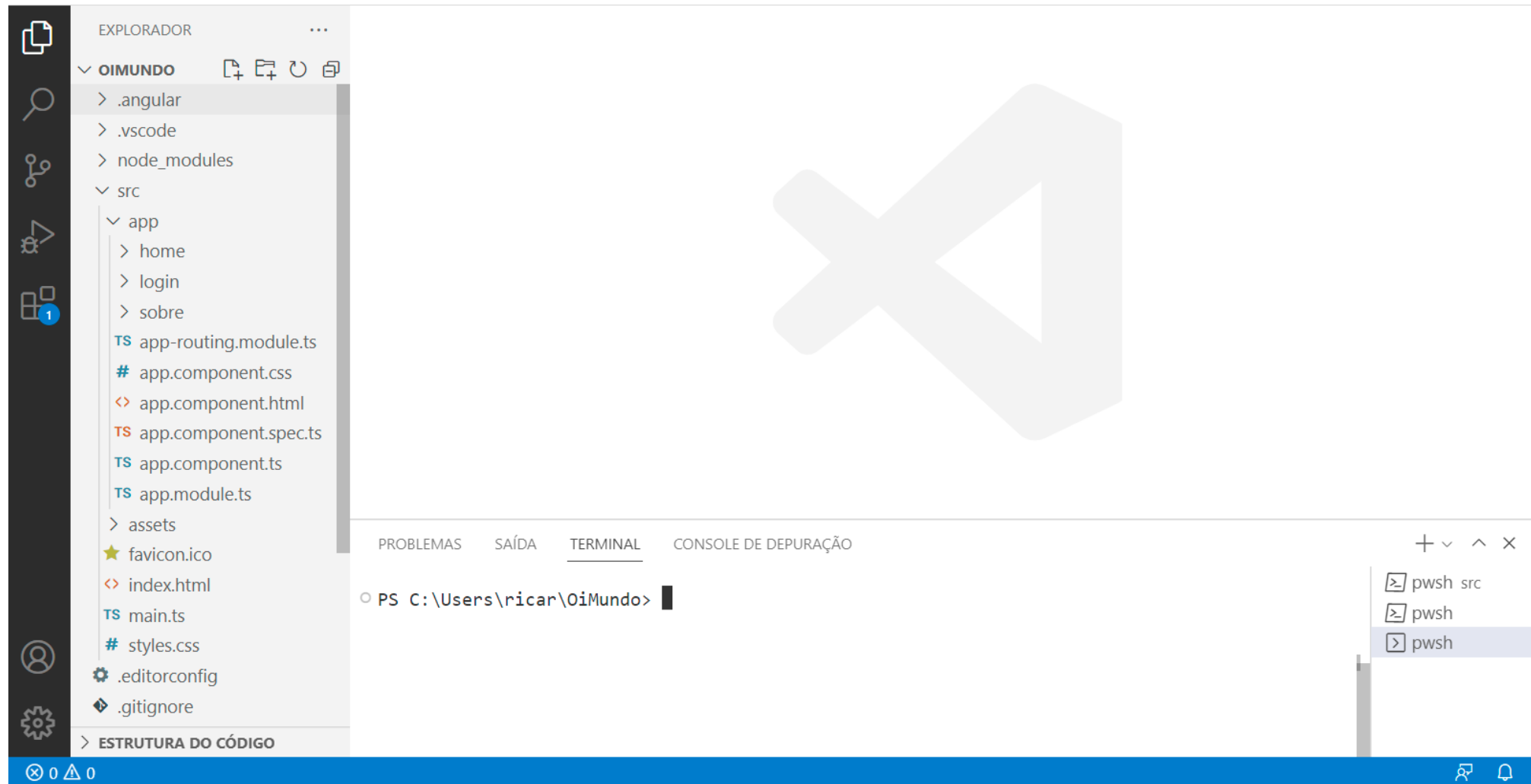
Geração de artefatos do Angular

- Se quiser adicionar seu componente, diretiva ou pipe para outro módulo (que não seja o módulo principal da aplicação, `app.module.ts`), você pode simplesmente prefixar o nome do componente com o nome do módulo e uma barra assim:
 - `$ ng g component meu-modulo/meu-componente`
- `meu-modulo` é o nome de um módulo existente.

Angular

- Digite:
 - \$ ng g c home
 - \$ ng g c login
 - \$ ng g c sobre

Angular



Angular

TS app-routing.module.ts ×



src > app > TS app-routing.module.ts > ...

```
1 import { NgModule } from '@angular/core';
2 import { RouterModule, Routes } from '@angular/router';
3 import { LoginComponent } from '../login/login.component';
4 import { SobreComponent } from '../sobre/sobre.component';
5 import { HomeComponent } from '../home/home.component';
6
7 const routes: Routes = [
8   {path: 'login', component: LoginComponent},
9   {path: 'sobre', component: SobreComponent},
10  {path: 'home', component: HomeComponent}
11 ];
12
13 @NgModule({
14   imports: [RouterModule.forRoot(routes)],
15   exports: [RouterModule]
16 })
```

Angular

<> index.html ×



src > <> index.html > html > body > search

```
1  <!doctype html>
2  <html lang="en">
3  <head>
4    <meta charset="utf-8">
5    <title>Olá mundo</title>
6    <base href="/">
7    <meta name="viewport" content="width=device-width, initial-scale=1">
8    <link rel="icon" type="image/x-icon" href="favicon.ico">
9  </head>
10 <body>
11   <menu></menu>
12   <search></search>
13   <app-root></app-root>
14 </body>
15 </html>
16
```

Angular

- Coloque texto no html de cada componente criado

<> sobre.component.html × TS sobre.component.ts

src > app > sobre > <> sobre.component.html > ...

Go to component

1 <p>Sobre o angular!</p>

2 <h1>Tudo isso fica bem legal pois as coisas vão se conectando</h1>

Angular

- O app.component é onde é iniciado todo o projeto.


```
<> app.component.html •
src > app > <> app.component.html > router-outlet
Go to component
1 <h1>{{title}}</h1>
2 <nav>
3   <a routerLink="/">Index</a> <br>
4   <a routerLink="/home">Home </a><br>
5   <a routerLink="/login">Login </a><br>
6   <a routerLink="/sobre">Sobre</a><br>
7 </nav>
8 <router-outlet></router-outlet>
```

Consumindo uma API com Angular

- Para usar o HttpClient, precisamos adicionar o modulo HttpClientModule no arquivo app.module.ts.
- Para fazer isso, vamos abrir o arquivo app.module.ts
- Dentro de imports do decorator @NgModule, adicione o modulo HttpClientModule
 - `import { HttpClientModule } from '@angular/common/http';`

Consumindo uma API com Angular

- Vamos aproveitar e adicionar o modulo FormsModule, esse modulo nos ajudará em nosso formulário para nossa tela de exemplo, porém não é necessário para o uso do HttpClient.

src > app > TS app.module.ts >  AppModule

```
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6  import { LoginComponent } from './login/login.component';
7  import { SobreComponent } from './sobre/sobre.component';
8  import { HomeComponent } from './home/home.component';
9  import { HttpClientModule } from '@angular/common/http';
10 import { FormsModule } from '@angular/forms';
```


Consumindo uma API com Angular

```
13  ∨ @NgModule({  
14  ∨    declarations: [  
15      AppComponent,  
16      LoginComponent,  
17      SobreComponent,  
18      HomeComponent  
19    ],  
20  ∨    imports: [  
21      BrowserModule,  
22      AppRoutingModule,  
23      HttpClientModule  
24    ],  
25    providers: [HttpClientModule, FormsModule],  
26    bootstrap: [AppComponent]  
27  })
```

Model

- Vamos criar uma interface de modelo para os dados.
- Na raiz do projeto vamos executando o seguinte comando:
 - `ng g interface models/pessoa`

Model

- Agora vamos adicionar o seguinte conteúdo dentro de nosso model pessoa.ts

```
src > app > models > TS pessoa.ts > ...  
1  export interface Pessoa {  
2      id?: number;  
3      nome: string;  
4  }
```

Service

- Vamos criar os métodos responsáveis pelas requisições http que faremos usando o HttpClient, mas antes devemos criar um arquivo service, no angular é recomendado criar services para os métodos que faz chamadas http.
- Para criar o serviço, digite o seguinte comando:
 - `ng g service services/pessoa`
- Será criado um arquivo com o nome `pessoa.services.ts` dentro da pasta `services`.
- Após criar nosso service, já podemos começar a utilizar o HttpClient

HttpClient

- Dentro do nosso service pessoa.services.ts, vamos criar alguns métodos http utilizando o HttpClient.
- Abra o arquivo pessoa.services.ts dentro da pasta services e inclua o seguinte código:

```
TS pessoa.service.ts •
src > app > services > TS pessoa.service.ts > PessoaService
1 import { Injectable } from '@angular/core';
2 import { HttpClient, HttpResponse, HttpHeaders } from '@angular/common/http';
3 import { Observable, throwError } from 'rxjs';
4 import { retry, catchError } from 'rxjs/operators';
5 import { Pessoa } from '../models/pessoa';
6
7 @Injectable({
8   providedIn: 'root'
9 })
```

HttpClient

- Continuando:

```
10 export class PessoaService {
11   url = 'http://localhost:5258/api/';
12
13   // injetando o HttpClient
14   constructor(private httpClient: HttpClient) { }
15   // Headers
16   httpOptions = {
17     headers: new HttpHeaders({ 'Content-Type': 'application/json' })
18   }
19   // Obtem todas pessoas
20   getPessoas(): Observable<Pessoa[]> {
21     return this.httpClient.get<Pessoa[]>(this.url)
22       .pipe(
23         retry(2),
24         catchError(this.handleError))
25   }
26 }
```

HttpClient

- Continuando:

```
27     // Obtem uma pessoa pelo id
28     getPessoaById(id: number): Observable<Pessoa> {
29         return this.httpClient.get<Pessoa>(this.url + '/' + id)
30             .pipe(
31                 retry(2),
32                 catchError(this.handleError))
33     }
34
35     // salva uma pessoa
36     savePessoa(pessoa: Pessoa): Observable<Pessoa> {
37         return this.httpClient.post<Pessoa>(this.url, JSON.stringify(pessoa), this.httpOptions)
38             .pipe(
39                 retry(2),
40                 catchError(this.handleError)
41             )
42     }
```

HttpClient

- Continuando:

```
44 // atualiza uma pessoa
45 updatePessoa(pessoa: Pessoa): Observable<Pessoa> {
46     return this.httpClient.put<Pessoa>(this.url + '/' + pessoa.id, JSON.stringify(pessoa),
47         this.httpOptions)
48     .pipe(
49         retry(1),
50         catchError(this.handleError)
51     )
52 }
53
54 // deleta uma pessoa
55 deletePessoa(pessoa: Pessoa) {
56     return this.httpClient.delete<Pessoa>(this.url + '/' + pessoa.id, this.httpOptions)
57     .pipe(
58         retry(1),
59         catchError(this.handleError)
60     )
61 }
--
```


HttpClient

- Continuando:

```
// Manipulação de erros
handleError(error: HttpResponse) {
  let errorMessage = '';
  if (error.error instanceof ErrorEvent) {
    // Erro ocorreu no lado do client
    errorMessage = error.error.message;
  } else {
    // Erro ocorreu no lado do servidor
    errorMessage = `Código do erro: ${error.status}, ` +
      `mensagem: ${error.message}`;
  }
  console.log(errorMessage);
  return throwError(errorMessage);
};
```

No app.componente.ts

```
1  import { Component } from '@angular/core';
2  import { PessoaService } from '../services/pessoa.service';
3  import { Pessoa } from '../models/pessoa';
4  import { NgForm } from '@angular/forms';
5
6  @Component({
7    selector: 'app-root',
8    templateUrl: './app.component.html',
9    styleUrls: ['./app.component.css']
10 })
11 export class AppComponent {
12   title = 'OiMundo';
13
14   pessoa = {} as Pessoa;
15   pessoas: Pessoa[] = [];
```

No app.componente.ts

```
19 |   ngOnInit() {  
20 |     |   this.getPessoas();  
21 |   }  
--
```

```
46 |  ∨  getPessoas() {  
47 |  ∨  |   this.pessoaService.getPessoas().subscribe((pessoas: Pessoa[]) => {  
48 |  |   |   this.pessoas = pessoas;  
49 |  |   |   });  
50 |  |   }
```

Entendendo nosso serviço

- Injetamos o HttpClient e atribuímos a uma variável chamada httpClient, observe como é simples usar os métodos get, post, put e delete do nosso httpClient:
 - `this.httpClient.get<Pessoa[]>(this.url)`
 - `this.httpClient.post< Pessoa >(this.url, JSON.stringify(pesspa), this.httpOptions)`
 - `this.httpClient.put< Pessoa >(this.url + '/' + pessoa.id, JSON.stringify(pessoa), this.httpOptions)`
 - `this.httpClient.delete< Pessoa >(this.url + '/' + pessoa.id, this.httpOptions)`

Entendendo nosso serviço

- Note que passamos uma variável chamada “url” essa conterá o endereço <http://localhost:5258/api/> disponibilizada pela API REST que temos desenvolvido no Visual Studio ASP.NET Web API .
- Adicionamos em nosso HttpClient um objeto do tipo HttpOptions, contendo nosso header através da classe HttpHeaders.

Usando nosso serviço

- Agora vamos chamar nosso serviço através do component `app.component.ts`. Para isso vamos editar o component `app.component.ts` e adicionar o seguinte conteúdo:

```
16 <div class="container">
17   <div class="card list-pessoa">
18     <h5 class="card-header">Lista de pessoa</h5>
19     <div class="card-body">
20       <table class="table">
21         <thead>
22           <tr>
23             <th scope="col">Id</th>
24             <th scope="col">Nome</th>
25           </tr>
26         </thead>
27         <tbody>
28           <tr *ngFor="let pessoa of pessoas">
29             <td>{{pessoa.id}}</td>
30             <td>{{pessoa.nome}}</td>
31           </tr>
32         </tbody>
33       </table>
34     </div>
35   </div>
36
```

Caso ocorra erro ao acessar a API

- Adicione no Program.cs da API

```
8
9
10 var builder = WebApplication.CreateBuilder(args);
11 var MyAllowSpecificOrigins = "_myAllowSpecificOrigins";
12 // Add services to the container.
13
14 builder.Services.AddCors(options =>
15 {
16     options.AddPolicy(name: MyAllowSpecificOrigins,
17                       policy =>
18                       {
19                           policy.AllowAnyOrigin();
20                           //policy.WithOrigins("http://example.com" "http://www.contoso.com");
21                       });
22 });
```

```
37 app.UseHttpsRedirection();
38
39 app.UseAuthorization();
40 app.UseCors(MyAllowSpecificOrigins);
41
42 app.MapControllers();
43
44 app.Run();
```


Referências

- <https://www.freecodecamp.org/portuguese/news/como-instalar-o-angular-no-windows-um-guia-de-angular-cli-node-js-e-ferramentas-de-criacao/>
- <https://medium.com/nave-recife/entendendo-a-estrutura-de-um-projeto-angular-aa22833a7491>
- <https://vidafullstack.com.br/angular/o-que-e-um-e-como-criar-um-componente-com-angular/>
- <https://balta.io/blog/angular-rotas-guardas-navegacao>
- <https://medium.com/angularbr/angular-5-trabalhando-com-rotas-8335617fcdbc>
- <https://www.devmedia.com.br/angular-http-como-realizar-requisicoes-em-suas-aplicacoes/40642>
- <https://www.youtube.com/watch?v=6QXBTevf0VY>
- https://medium.com/@fernandoevangelista_28291/consumindo-api-rest-com-httpclient-no-angular-8-62c5d733ffb6
- <https://learn.microsoft.com/pt-br/aspnet/core/security/cors?view=aspnetcore-7.0>
- https://www.macoratti.net/18/05/aspcore_cors1.htm