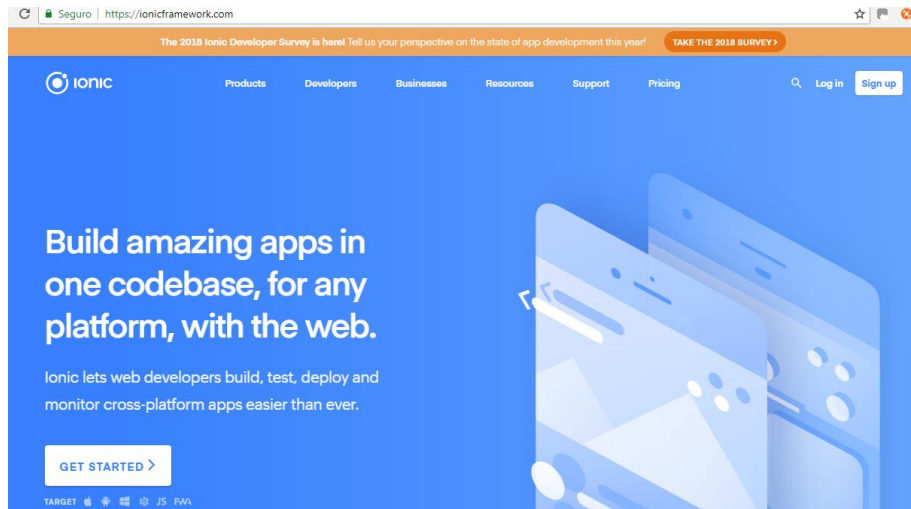


IONIC

CODIFICANDO

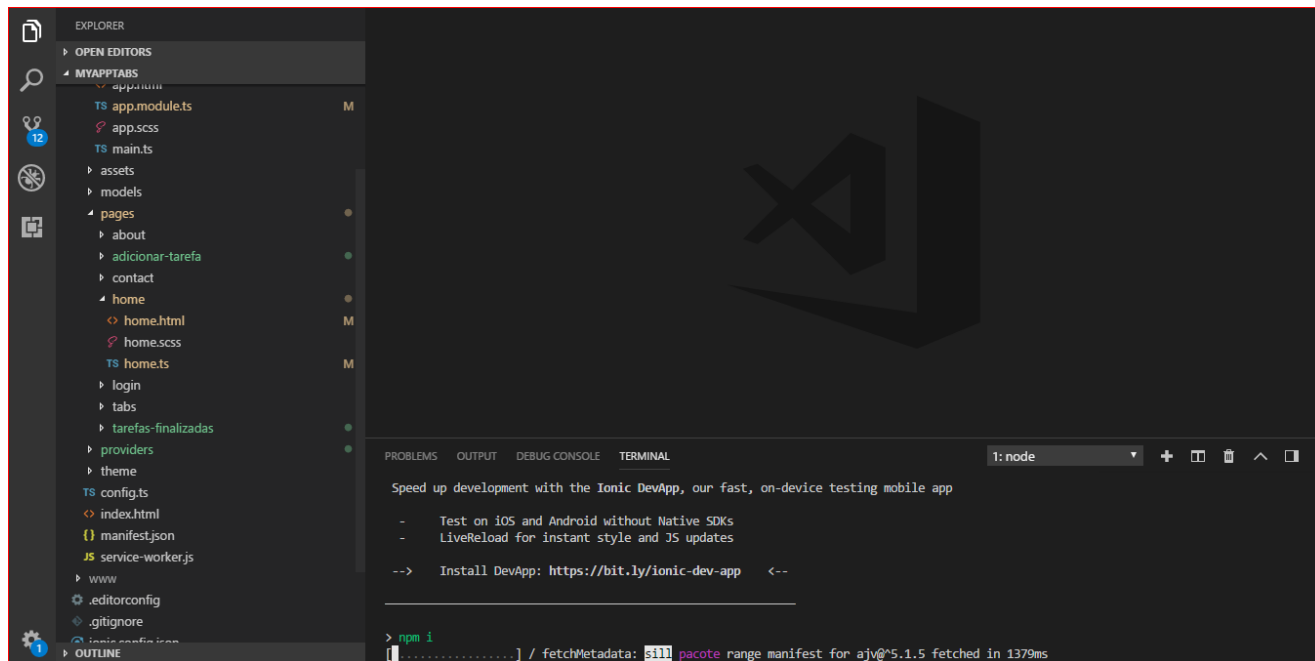


PROJETO INICIAL

CODIFICANDO

PASSOS

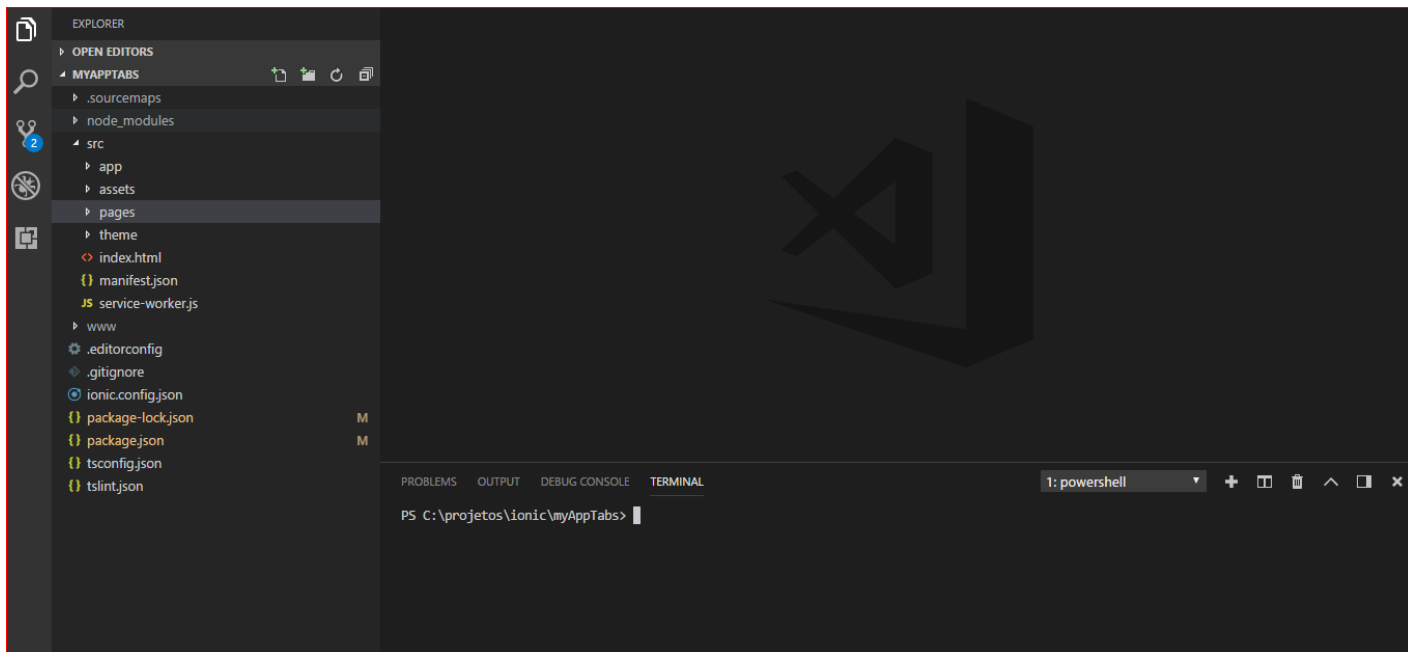
- ▶ Criar um projeto ionic com TABS



CODIFICANDO

PASSOS

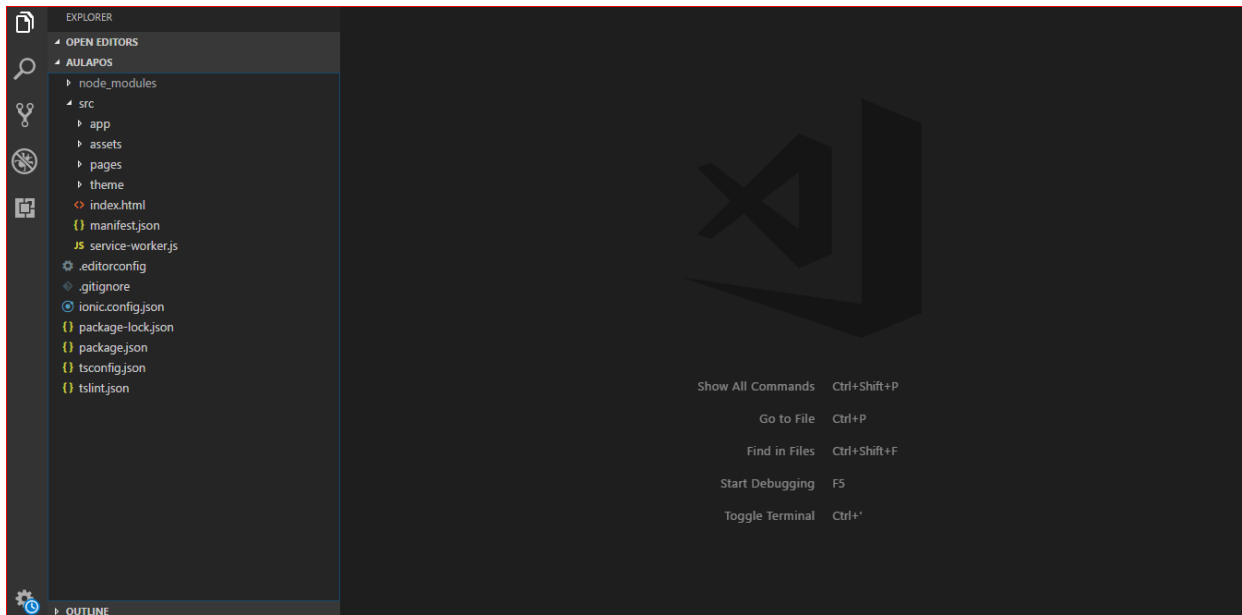
- ▶ Abrir o projeto com tabs no visual studio code



CODIFICANDO

PASSOS

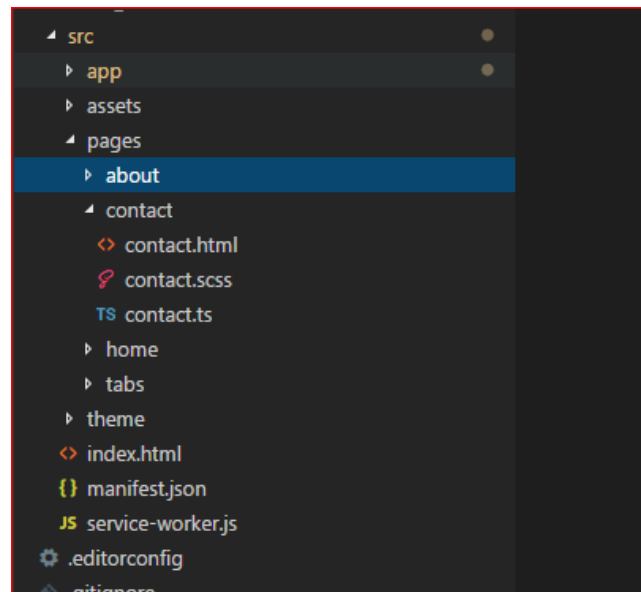
- ▶ Entre as diversas pastas criadas, a pasta SRC é a principal, onde iremos atualizar sempre que for possível.



CODIFICANDO

PASSOS

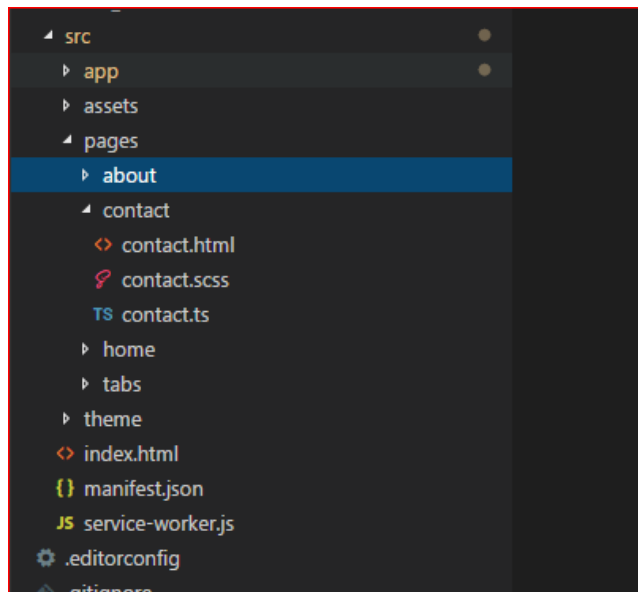
▶ Pelo fato do ionic ser baseado no **Angular**, tudo que for criado é **componente**, devemos trabalhar sempre com os componentes. (pagina, serviço, etc).



CODIFICANDO

PASSOS

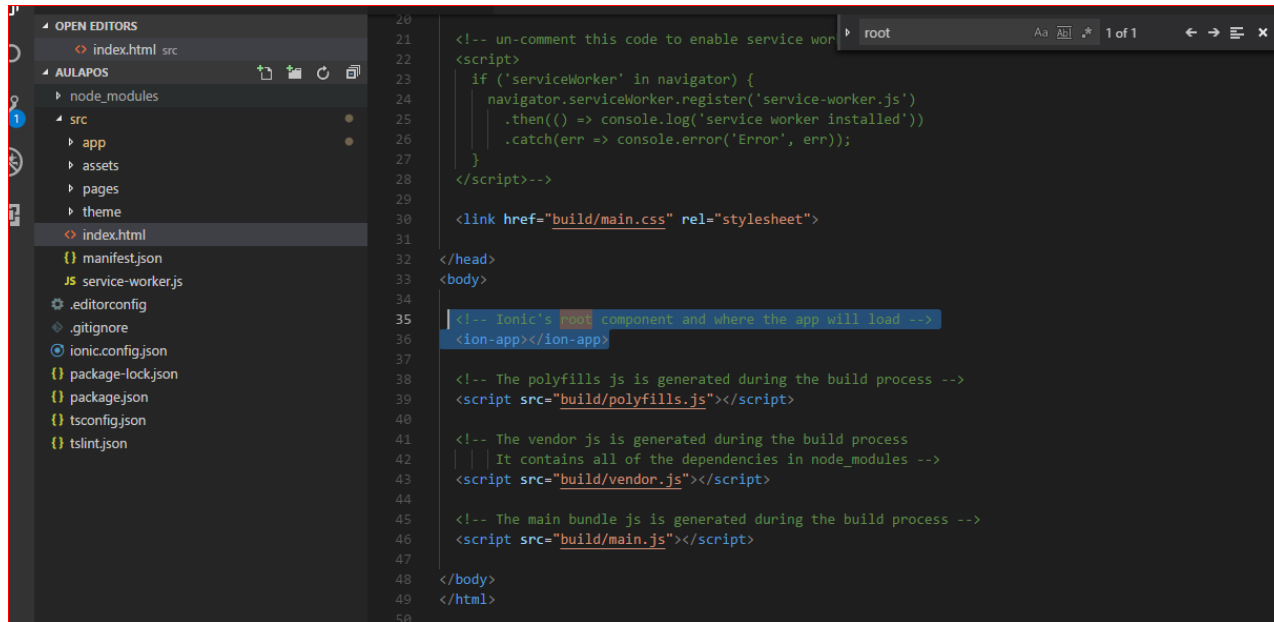
- ▶ Um componente é formado por 3 arquivos:
- ▶ .ts (configura o componente e suas dependências)
- ▶ .html (front end do componente, pode ou não ter)
- ▶ .scss (no formato SASS, para o estilo)



CODIFICANDO

PASSOS

- ▶ Na pasta `src` temos o arquivo “`index.html`” que faz uma chamada ao componente “`app`”



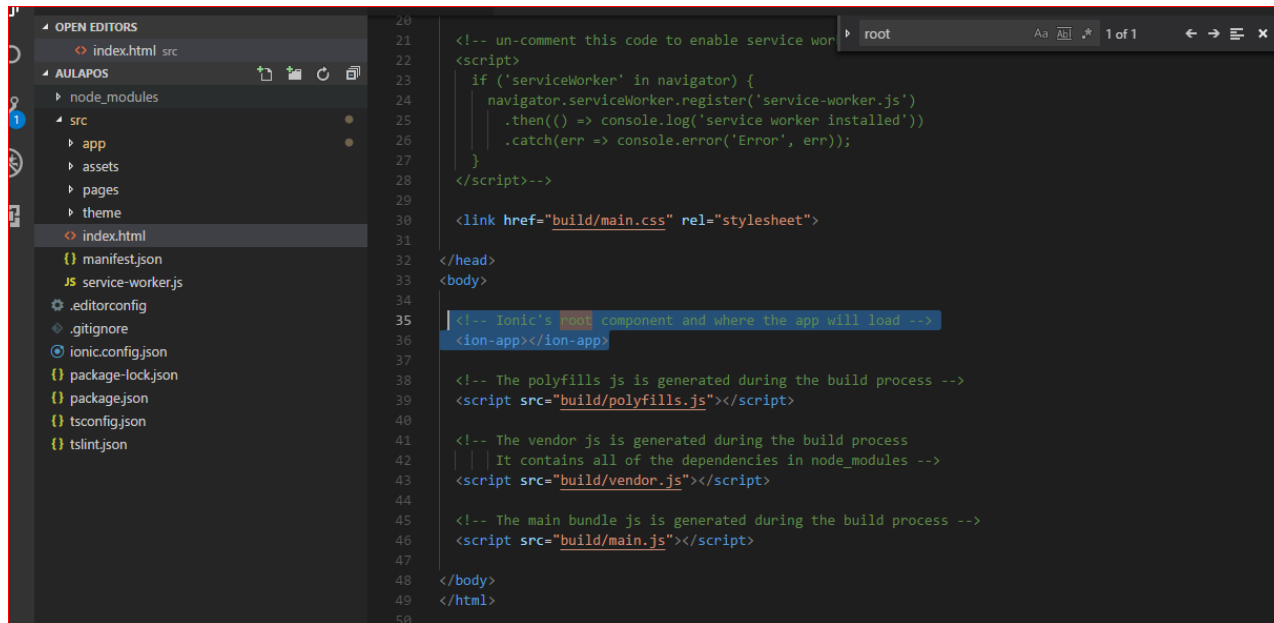
The screenshot shows a code editor with a dark theme. On the left, the 'OPEN EDITORS' sidebar shows a file tree for a project named 'AULAPOS'. The tree includes a 'src' directory with files like 'index.html', 'manifest.json', 'service-worker.js', and various configuration files. The 'index.html' file is selected and open in the main editor. The code in the editor is HTML, showing a service worker registration script, a link to 'build/main.css', and an `<ion-app>` component. The `<ion-app>` tag is highlighted with a blue selection bar. The code also includes comments about the build process and the location of the app component.

```
20
21
22 <!-- un-comment this code to enable service wor
23 <script>
24   if ('serviceWorker' in navigator) {
25     navigator.serviceWorker.register('service-worker.js')
26     .then(() => console.log('service worker installed'))
27     .catch(err => console.error('Error', err));
28   }
29 </script-->
30
31 <link href="build/main.css" rel="stylesheet">
32
33 </head>
34 <body>
35   <!-- Ionic's root component and where the app will load -->
36   <ion-app></ion-app>
37
38   <!-- The polyfills js is generated during the build process -->
39   <script src="build/polyfills.js"></script>
40
41   <!-- The vendor js is generated during the build process
42   | | It contains all of the dependencies in node_modules -->
43   <script src="build/vendor.js"></script>
44
45   <!-- The main bundle js is generated during the build process -->
46   <script src="build/main.js"></script>
47
48 </body>
49 </html>
50
```


CODIFICANDO

PASSOS

- ▶ O comando `<ion-app>` vai chamar o componente app



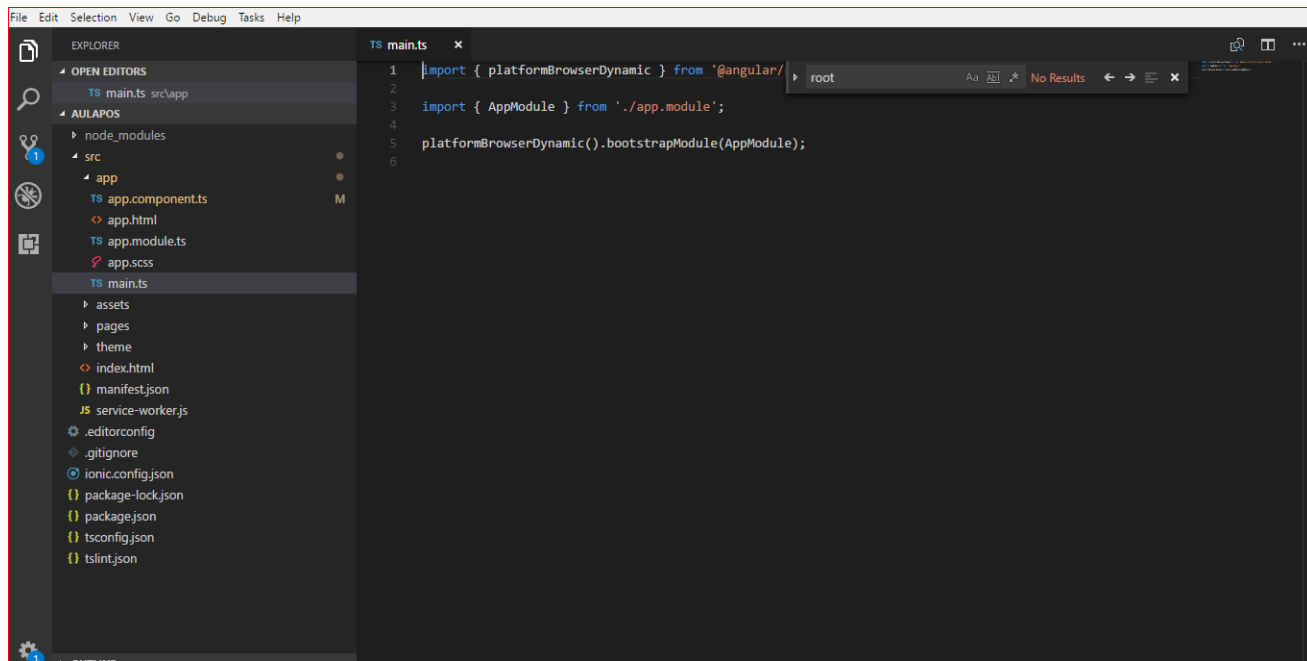
The screenshot shows a code editor with a dark theme. On the left, the 'OPEN EDITORS' sidebar shows a file tree for a project named 'AULAPOS'. The tree includes a 'src' directory with files like 'index.html', 'manifest.json', 'service-worker.js', and various configuration files. The 'index.html' file is selected and open in the main editor. The code in the editor is HTML, showing the head and body sections. A comment at line 35 indicates the location for the Ionic app component. The `<ion-app>` tag is highlighted in blue at line 36. Other scripts and links are also visible in the code.

```
20
21
22 <!-- un-comment this code to enable service wor
23 <script>
24   if ('serviceWorker' in navigator) {
25     navigator.serviceWorker.register('service-worker.js')
26     .then(() => console.log('service worker installed'))
27     .catch(err => console.error('Error', err));
28   }
29 </script-->
30
31 <link href="build/main.css" rel="stylesheet">
32
33 </head>
34 <body>
35   <!-- Ionic's root component and where the app will load -->
36   <ion-app></ion-app>
37
38   <!-- The polyfills js is generated during the build process -->
39   <script src="build/polyfills.js"></script>
40
41   <!-- The vendor js is generated during the build process
42   | | It contains all of the dependencies in node_modules -->
43   <script src="build/vendor.js"></script>
44
45   <!-- The main bundle js is generated during the build process -->
46   <script src="build/main.js"></script>
47
48 </body>
49 </html>
50
```

CODIFICANDO

PASSOS

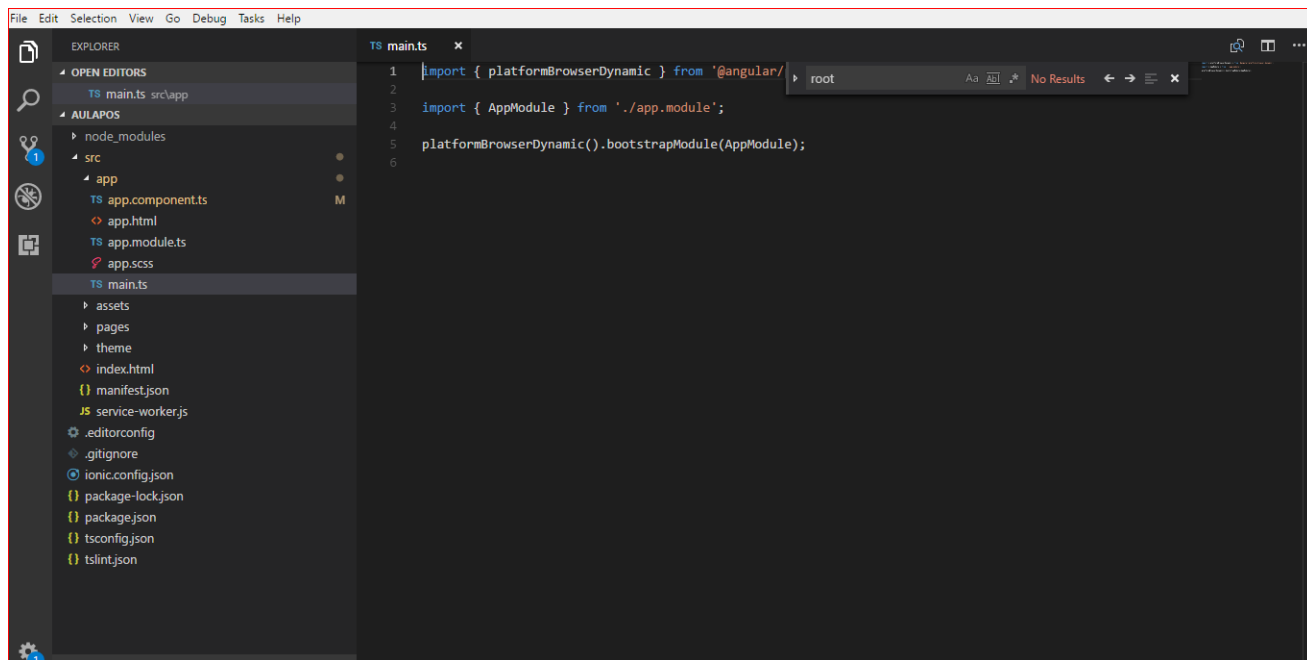
- ▶ Dentro do componente App (pasta app), o arquivo principal é o main.ts



CODIFICANDO

PASSOS

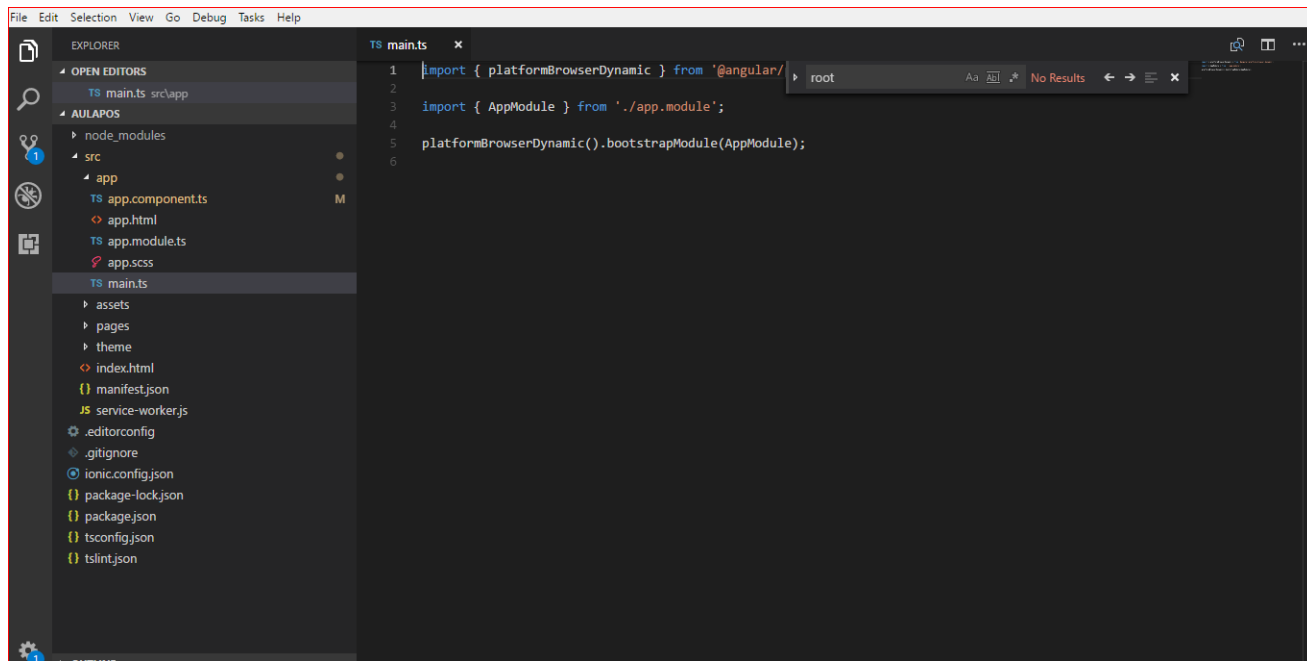
- ▶ O main.ts importa o arquivo AppModule, que está em “app.module.ts”



CODIFICANDO

PASSOS

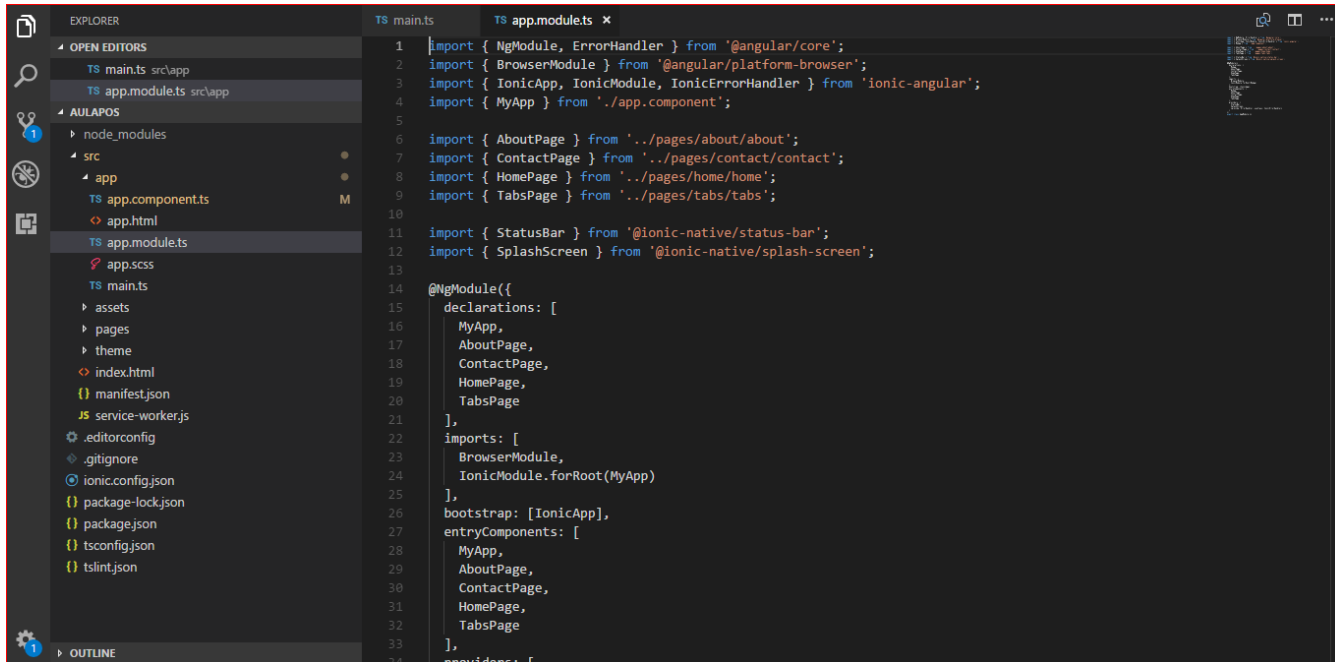
- ▶ O main.ts também coloca o bootstrap no “AppComponent”



CODIFICANDO

PASSOS

- ▶ Dentro do arquivo “app.module.ts” ele diz que o root vai ser a classe “MyApp”

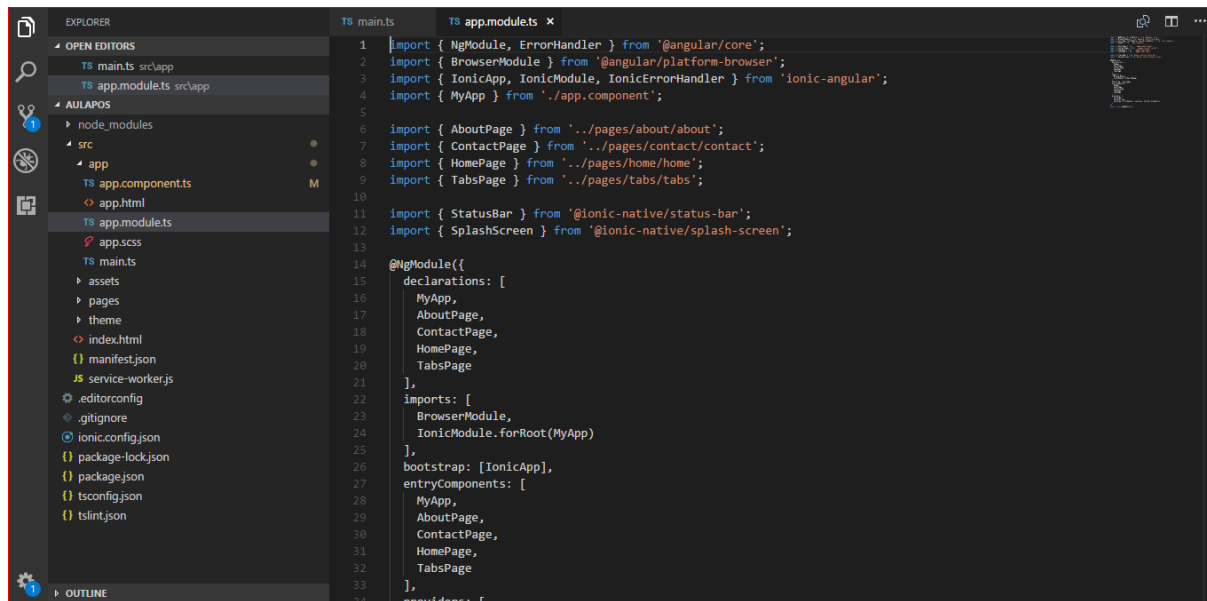


```
1 import { NgModule, ErrorHandler } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3 import { IonicApp, IonicModule, IonicErrorHandler } from 'ionic-angular';
4 import { MyApp } from './app.component';
5
6 import { AboutPage } from '../pages/about/about';
7 import { ContactPage } from '../pages/contact/contact';
8 import { HomePage } from '../pages/home/home';
9 import { TabsPage } from '../pages/tabs/tabs';
10
11 import { StatusBar } from '@ionic-native/status-bar';
12 import { SplashScreen } from '@ionic-native/splash-screen';
13
14 @NgModule({
15   declarations: [
16     MyApp,
17     AboutPage,
18     ContactPage,
19     HomePage,
20     TabsPage
21   ],
22   imports: [
23     BrowserModule,
24     IonicModule.forRoot(MyApp)
25   ],
26   bootstrap: [IonicApp],
27   entryComponents: [
28     MyApp,
29     AboutPage,
30     ContactPage,
31     HomePage,
32     TabsPage
33   ],
34   providers: [
```

CODIFICANDO

PASSOS

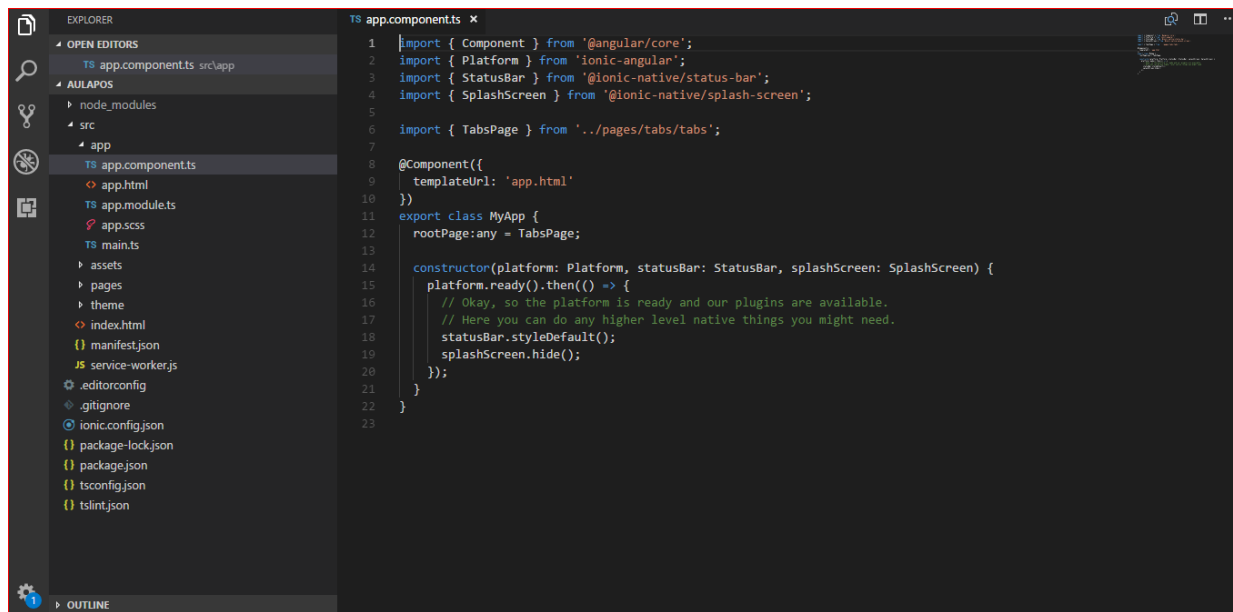
- ▶ A classe MyApp que este componente utiliza está dentro do arquivo “app.component.ts” (CTRL+clique para abrir)



CODIFICANDO

PASSOS

- ▶ O “app.component.ts” (typescript) é o arquivo de código que faz a configuração do componente.



```
1 import { Component } from '@angular/core';
2 import { Platform } from 'ionic-angular';
3 import { StatusBar } from '@ionic-native/status-bar';
4 import { SplashScreen } from '@ionic-native/splash-screen';
5
6 import { TabsPage } from '../pages/tabs/tabs';
7
8 @Component({
9   templateUrl: 'app.html'
10 })
11 export class MyApp {
12   rootPage:any = TabsPage;
13
14   constructor(platform: Platform, statusBar: StatusBar, splashScreen: SplashScreen) {
15     platform.ready().then(() => {
16       // Okay, so the platform is ready and our plugins are available.
17       // Here you can do any higher level native things you might need.
18       statusBar.styleDefault();
19       splashScreen.hide();
20     });
21   }
22 }
23
```

CODIFICANDO

PASSOS

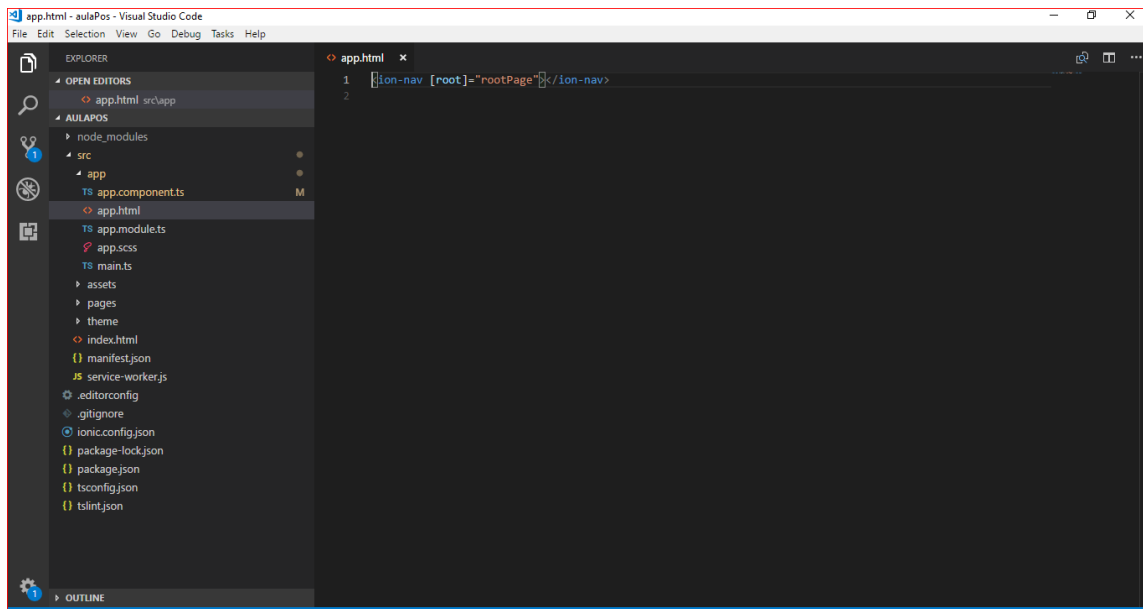
- ▶ Ela usa uma anotação de Componente que diz que um componente “templateUrl” utiliza a página que está na url “app.html”

```
@Component({  
  templateUrl: 'app.html'  
})  
|
```


CODIFICANDO

PASSOS

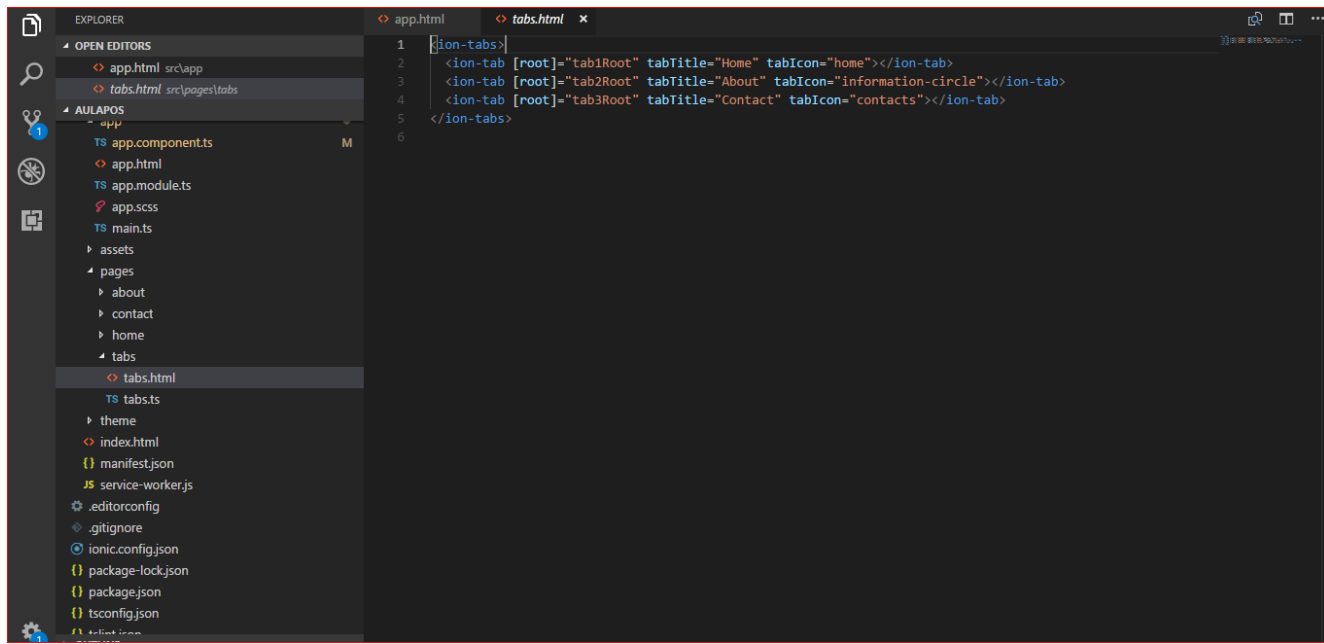
- ▶ No nosso projeto, dentro do arquivo HTML ele usa o `<ion-nav>`, pois o projeto está dentro de abas



CODIFICANDO

PASSOS

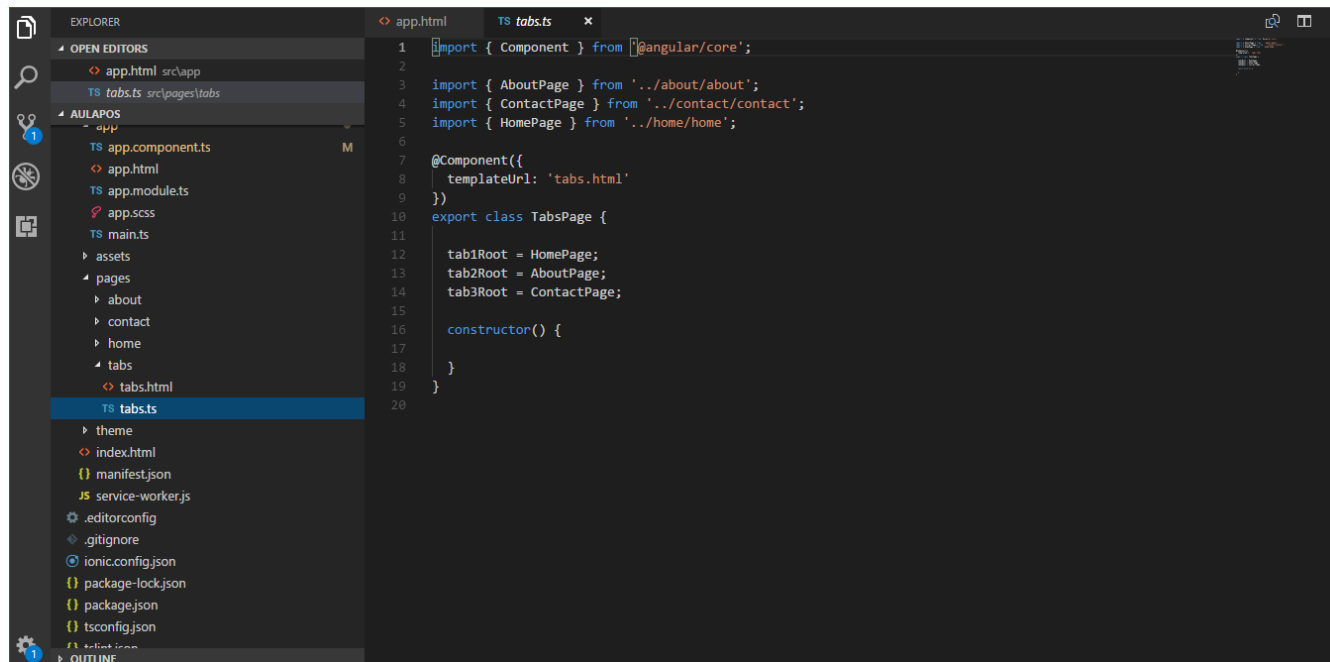
- ▶ Dentro do arquivo `pages/tabs.ts` é que podemos ver as tabs que foram criadas.



CODIFICANDO

PASSOS

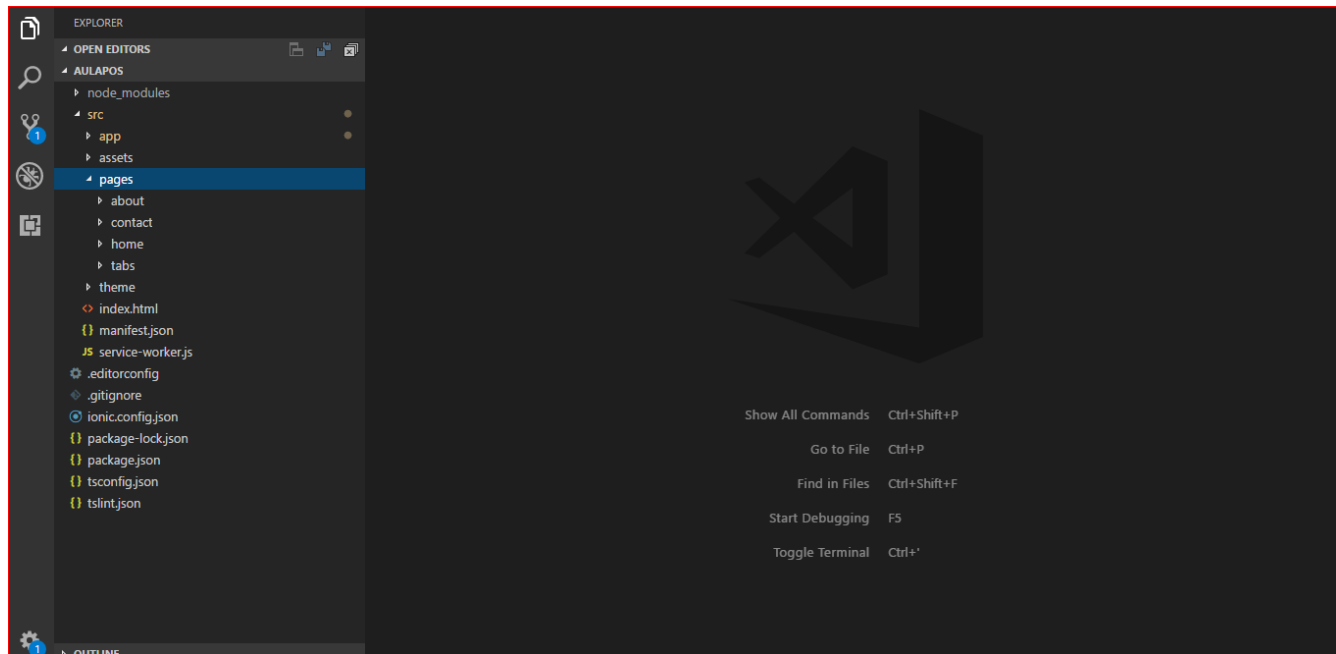
- ▶ Em tabs.ts está o controller que define o que vai ser carregado em cada tab (quais classes)



CODIFICANDO

PASSOS

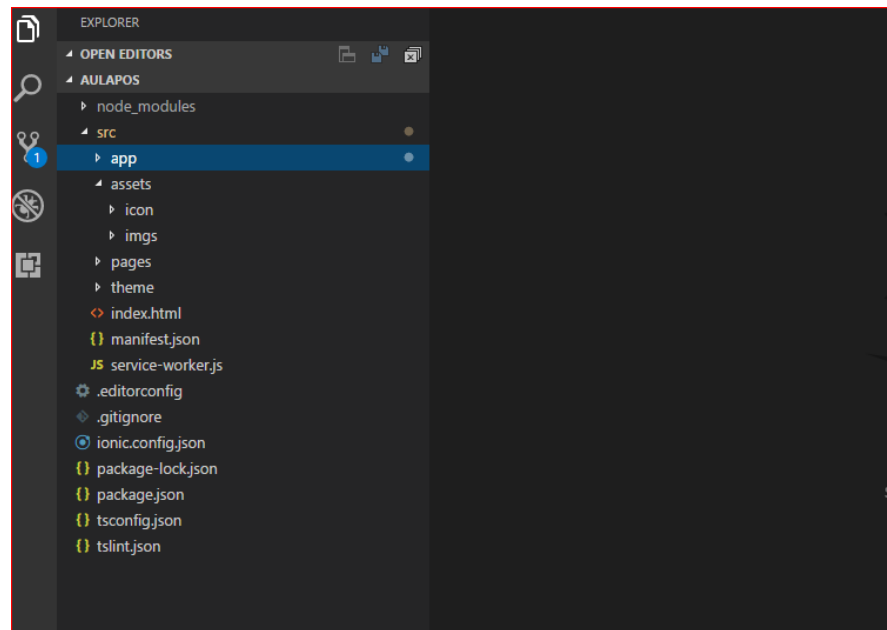
- ▶ As classes das páginas que nosso app utiliza estão dentro da pasta “pages”



CODIFICANDO

PASSOS

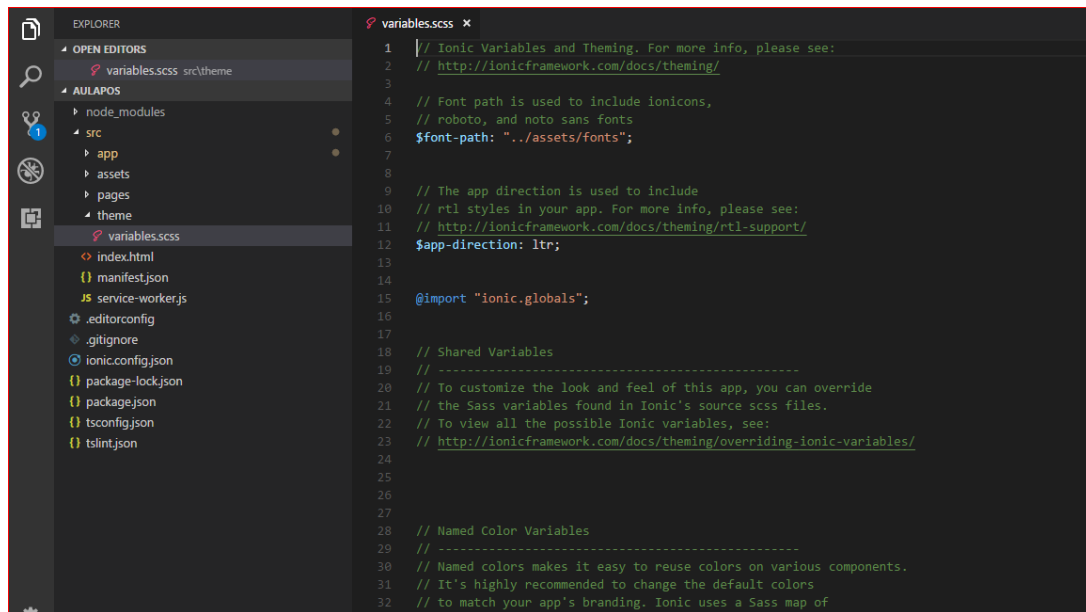
- ▶ A pasta assets é onde ficam os arquivos de assets globais para a aplicação (css, imagens...)



CODIFICANDO

PASSOS

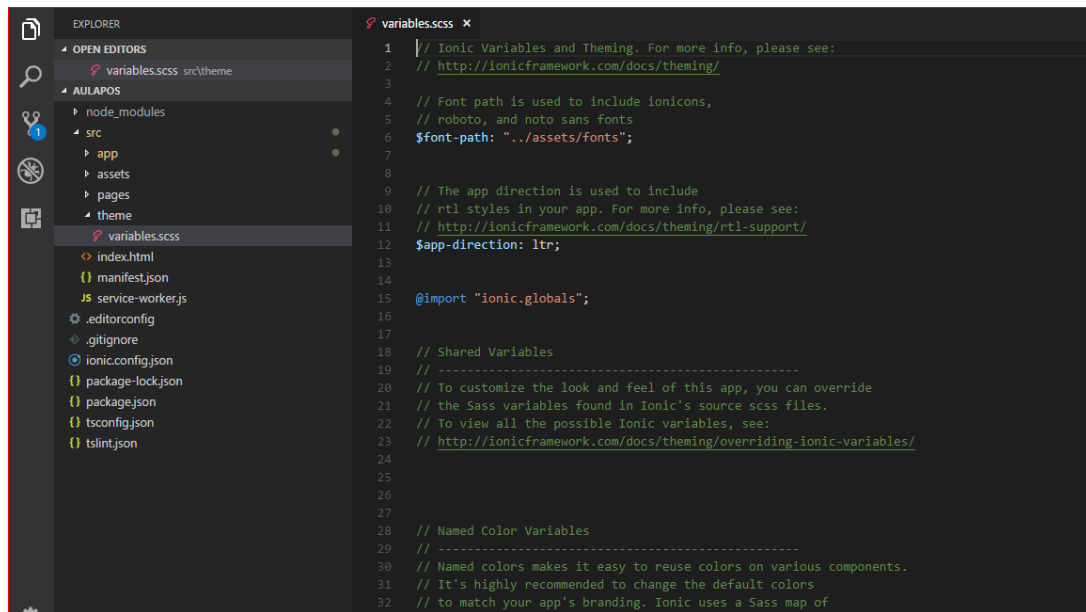
- ▶ A pasta theme é onde podemos alterar todo tema do nosso projeto.



CODIFICANDO

PASSOS

- ▶ Com o projeto rodando, alterar o tema primário para verde, por exemplo.



CODIFICANDO

PASSOS

- ▶ Outra pasta importante é a WWW.
- ▶ Nela as pastas assets e build é para onde os arquivos veem depois de dar um build.

