

Sumário

[INTRODUÇÃO 4](#_Toc43634237)

[Informações: 4](#_Toc43634238)

[SPA – SINGLE PAGE APPLICATION 4](#_Toc43634239)

[Diferença entre os Framworks Angular 5](#_Toc43634240)

[AngularJS : 5](#_Toc43634241)

[Angular: 5](#_Toc43634242)

[Por que foi criado o Angular 2 e Quais as diferenças entre o AngularJS e o Angular 2? 5](#_Toc43634243)

[Principais melhorias das Evoluções 5](#_Toc43634244)

[Angular v4 5](#_Toc43634245)

[Angular v5 6](#_Toc43634246)

[Angular v6 6](#_Toc43634247)

[Angular v7 6](#_Toc43634248)

[Melhor performance 6](#_Toc43634249)

[Dependências 6](#_Toc43634250)

[Angular CLI 7](#_Toc43634251)

[Angular v8 7](#_Toc43634252)

[Ivy Renderer 7](#_Toc43634253)

[Resumindo a evolução 7](#_Toc43634254)

[CONFIGURANDO AMBIENTE 7](#_Toc43634255)

[NodeJS – Servidor de Aplicação 7](#_Toc43634256)

[Angular CLI 12](#_Toc43634257)

[ENTENDENDO O FRAMEWORK ANGULAR 13](#_Toc43634258)

[Estrutura de um projeto Angular com Angular CLI 13](#_Toc43634259)

[package.json 13](#_Toc43634260)

[angular.json ou angular-cli.json 13](#_Toc43634261)

[node\_modules 13](#_Toc43634262)

[src 14](#_Toc43634263)

[Index, Style e main 14](#_Toc43634264)

[environment.json 14](#_Toc43634265)

[Assets 14](#_Toc43634266)

[App 14](#_Toc43634267)

[Modules 15](#_Toc43634268)

[ Imports 15](#_Toc43634269)

[ Declarations 15](#_Toc43634270)

[ Providers 15](#_Toc43634271)

[Decorators 15](#_Toc43634272)

[Diretives 16](#_Toc43634273)

[Components 16](#_Toc43634274)

[Router 17](#_Toc43634275)

[Lidando com o two-way data binding 17](#_Toc43634276)

[Implementando two-way data binding no Angular 17](#_Toc43634277)

[Criando novo projeto 18](#_Toc43634278)

[Criar novo modulo 18](#_Toc43634279)

[Criar novos componentes do modulo 18](#_Toc43634280)

[Criar novos serviço dos componentes 18](#_Toc43634281)

[Inicializando o Projeto 19](#_Toc43634282)

[TRABALHANDO COM ROTAS NO ANGULAR 19](#_Toc43634283)

[Declarando Rotas em app-routing.module.ts 20](#_Toc43634284)

[Declarando Rotas em transferencias-routing.module.ts 20](#_Toc43634285)

[Router outlet 20](#_Toc43634286)

[Referências 22](#_Toc43634287)

**WORKSHOP ANGULAR**

# INTRODUÇÃO

Angular é uma plataforma de aplicações web de código-fonte aberto e front-end baseado em TypeScript liderado pela Equipe Angular do Google e por uma comunidade de indivíduos e corporações. Angular é uma reescrita completa do AngularJS, feito pela mesma equipe que o construiu.

## **Informações:**

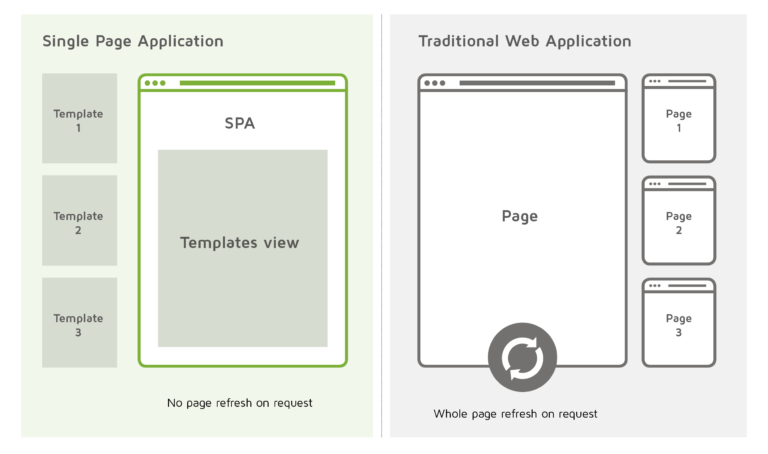
* Gravado em: TypeScript
* Plataforma: Web platform
* Lançamento: 14 setembro 2016; há 2 anos
* Versão estável: 7.1.4 (18 dezembro 2018; há 3 mês)
* Versão em teste: 7.2.0-beta.2 (11 dezembro 2018; há 4 meses)
* Desenvolvedor: Google/ Misko Hevery

# SPA – SINGLE PAGE APPLICATION

Antes de entendermos tudo sobre Angular, é muito importante entendermos o que significa uma Aplicação Single Page para o Angular.

Um dos melhores conceitos que o Angular oferece é o de “Single Page”, onde os recursos apropriados são dinamicamente carregados e adicionados à página, conforme necessário, geralmente em resposta a ações do usuário.

Para isto acontecer o framework oferece módulos que te possibilitam ter apenas uma página index, com outras páginas de conteúdo (views) sendo carregadas de acordo com uma específica rota (route).



# DIFERENÇA ENTRE OS FRAMEWORKS ANGULAR

## AngularJS

AngularJS é um framework JavaScript onde ele estende atributos HTML com diretivas e vincula dados a HTML com expressões. A sua incorporação no projeto pode ser feita através de um import a uma página HTML com uma tag <script>.

Por se tratar de um framework é válido destacar que seu core é totalmente escrito em JavaScript.

## Angular 2+

Angular foi reescrito passando a ser um framework diferente e não uma evolução. A princípio, o propósito dessa reescrita foi adequar o framework Angular para uso de padrões web atuais, a utilização de web componentes e uma melhor integração com o HTML 5, CSS 3 e Javascript.

### Por que foi criado o Angular 2 e Quais as diferenças entre o AngularJS e o Angular 2?

O Angular 1.x, por se tratar de um framework incorporado, ele precisa esperar pelo DOM para “carregar”, e se incluir na aplicação. Já o Angular 2 é totalmente diferente, ele tem o poder total no controle dos templates e consegue fazer todas as mudanças necessárias e antes mesmo delas serem alcançadas pelo DOM.

Diferente do angularJS, é utilizado Typescript em sua utilização e por isso obrigatoriamente em Angular 2 e suas versões em diante devem ser interpretados por um servidor de aplicação.

# PRINCIPAIS MELHORIAS DAS EVOLUÇÕES

## Angular v4

**Por que mudaram do angular 2 para o 4?**

Internamente no Angular, existem diversos componentes e cada componente possui a sua versão por terem passados por pequenas atualizações sem alterar a estrutura do framework.

O componente Router por sua vez já estava na versão 3.5 devido as suas atualizações e por conta deste componente, a próxima versão do framework, foi denominada como 4 subindo todos os componentes para versão 4 também para não haver confusões, principalmente por compatibilidade.

* Novo website disponibilizando uma documentação rica do framework
* Inclusão do HttpClient

## Angular v5

* Bundle, menor e mais rápido
* Suporte oficial para PWA

## Angular v6

* + - Angular Elements
    - Juntamente com o CLI, novos comando como ng update e ng add
    - Update.angular.io – guia de update de versões (Versões de para, complexibilidade, npm ou yarn)

## Angular v7

### Melhor performance

Uma grande utilidade de uma aplicação SPA é o download completo da aplicação no primeiro acesso para reduzir as taxas de downloads por navegação. Isso pode ser um problema quando lembramos que grande parte do tráfego na internet vem de dispositivos móveis que, em geral, costumam acessar a internet em baixa velocidade de conexão.

|  |
| --- |
| Para mantermos o tamanho da nossa aplicação sobre controle,**o Angular 7 conta com a função de Bundle Budgets**. Isto é, quando construímos nossa aplicação, recebemos alertas quando o seu tamanho excede um limite pré-determinado. Este recurso já existia no Angular e agora se tornou padrão. Na construção de uma aplicação que ultrapasse 2 MB receberemos um alerta e em aplicações que ultrapassem 5 MB seremos impedidos de prosseguir com a construção. É claro que esses valores são personalizáveis, e podemos modificá-los no arquivo **angular.json** do nosso projeto:   * **maximumWarning:** Tamanho da aplicação que, se excedido, disparará um alerta. * **maximumError:** Tamanho da aplicação que, se excedido, causará um erro e interromperá a construção. |

### Dependências

Algumas dependências utilizadas pelo Angular também foram atualizadas:

* **TypeScript:** A versão do TypeScript, o Superset do JavaScript utilizado pelo Angular foi atualizada para a versão 3.1, que conta apenas com algumas pequenas mudanças em relação a declaração de propriedades em funções.
* **RxJS:** O RxJS, a biblioteca que provem ao Angular, dentre outras coisas, Observables, foi atualizada para a versão 6.3. Essa nova versão implementou apenas correções.
* NodeJS: Agora é possível utilizar o Angular e o Angular CLI na versão 10 do Node.js, embora ainda seja totalmente compatível com a versão 8.

### Angular CLI

A ferramenta de linha de comando também mudou. Agora, sempre que criamos um novo projeto, o CLI pergunta se desejamos adicionar o módulo de rotas e se desejamos utilizar algum processador CSS, como SASS ou LESS.

## Angular v8

* Importes dinâmicos para carregamentos sobre demanda
* Ivy Renderer

### Ivy Renderer

O Ivy Renderer é um novo mecanismo de renderização que foi projetado para ser compatível com a renderização existente e focado para melhorar a velocidade de renderização e otimiza o tamanho do pacote final. Para o Angular, esse não será o renderizador padrão, mas você pode ativá-lo manualmente nas opções do compilador.

Com o Ivy você pode esperar melhora na velocidade de renderização e redução do tamanho de sua aplicação em até 90%.

## Resumindo a evolução

Após a entrada do angular 2 no mercado e a sua boa reputação, a Google não perdeu tempo e foi lançando atualizações do framework corrigindo bugs com a versão 4 principalmente.

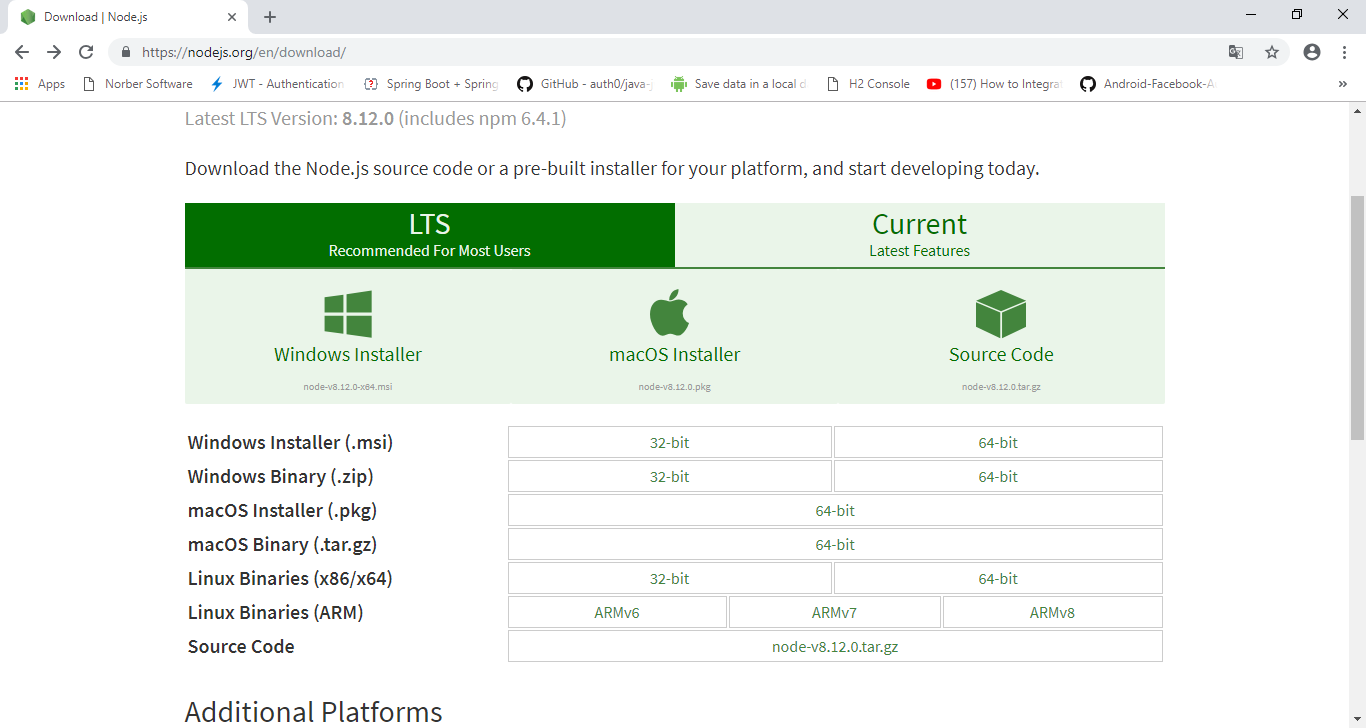
Desde o Angular 2 era utilizado uma biblioteca HTTP para que seja possível a aplicação realizar requisições externas via protocolo http. Porém nas novas versões, foi incluído o HttpClient que herda da biblioteca Http mas possui alguns atributos a mais que facilitam o desenvolvimento de um cliente, um exemplo é o suporte para tipos JSON de forma nativa em sua implementação no corpo da resposta de uma requisição.

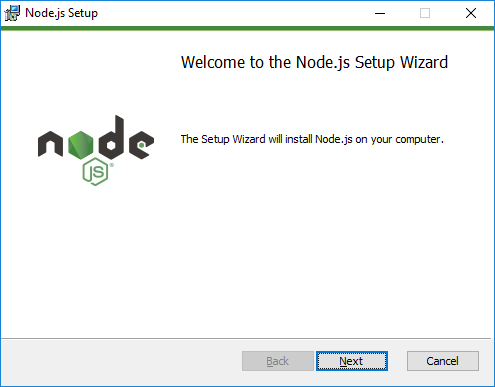
# CONFIGURANDO AMBIENTE

## NodeJS – Servidor de Aplicação

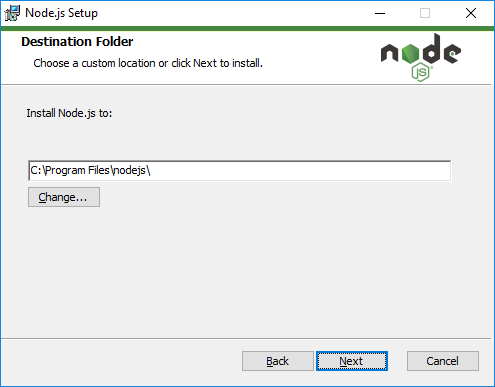
O NodeJS é um interpretador de código JavaScript com o código aberto, focado em migrar o Javascript do lado do cliente para servidores.

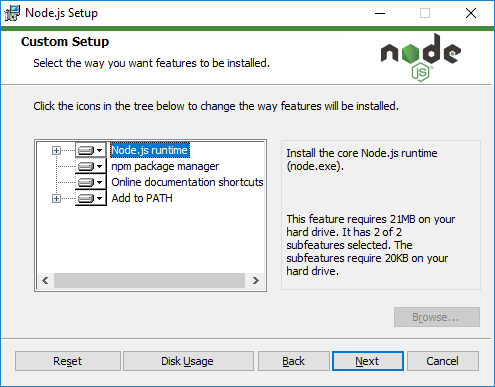
Acesse o link abaixo para realizar o download do Node: <https://nodejs.org/en/download>

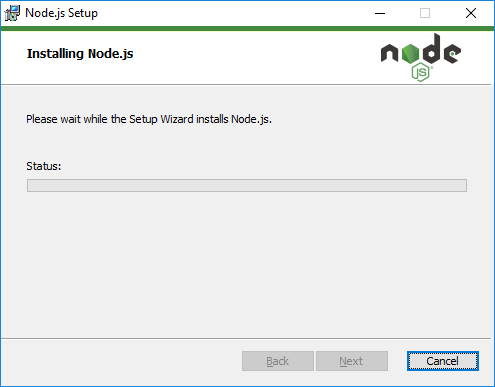


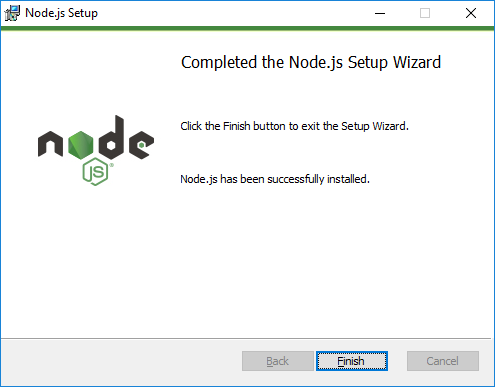






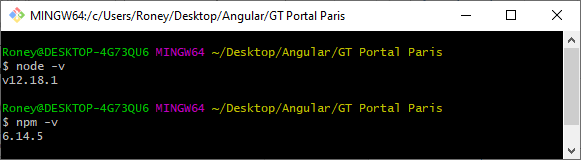






Feito a instalação, incluir nas variáveis de ambiente para sua conta o o caminho do npm na variável path.

Após realizar a instalação, abre qualquer terminal e confirma se o Node e o npm está sendo reconhecido corretamente seguindo os comandos abaixo:

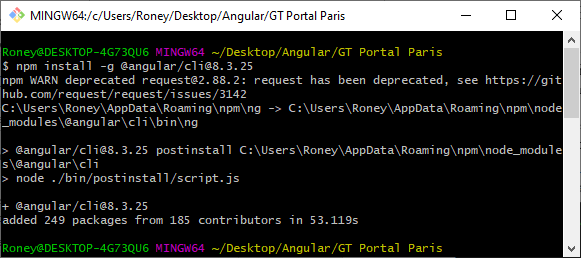


# Angular CLI

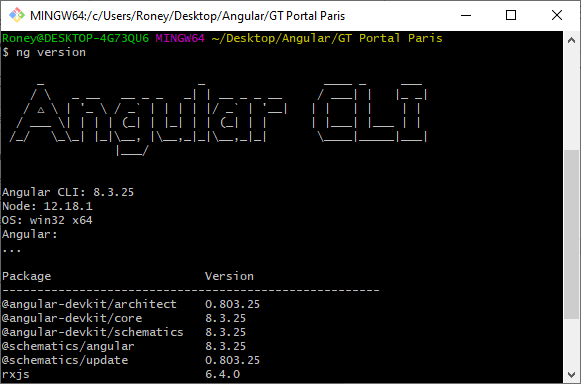
O Angular CLI é uma ferramenta para inicializar, desenvolver, sustentar e manter aplicações através de comandos básicos que podemos encontrar facilmente em sua documentação.

Com o npm reconhecido, basta executar o comando abaixo para instalar o Angular CLI globalmente:

Execute no terminal ***npm install –g @angular/cli@8.3.25***

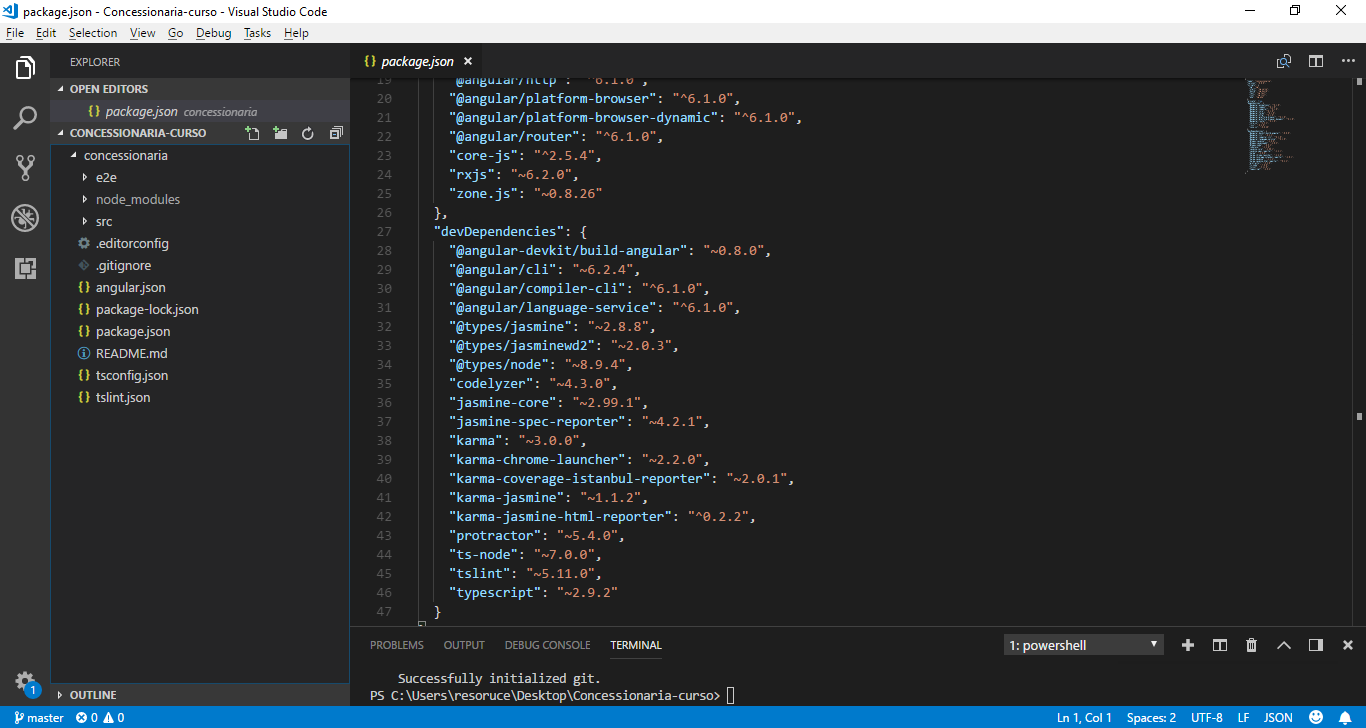


Confirma se o angular está sendo reconhecido através do comando ng version:



# ENTENDENDO O FRAMEWORK ANGULAR

## Estrutura de um projeto Angular com Angular CLI



### package.json

O arquivo package.json é um gerenciador de pacotes npm intalados localmente. Com ele, é possível listar os pacotes dos quais o projeto depende, permite que você especifique as versões de uma dependência que seu projeto pode usar e torna sua compilação reproduzível e, portanto, muito mais fácil de compartilhar com outros desenvolvedores.

### angular.json ou angular-cli.json

O mais provável é que todos nós encontremos o arquivo angular-cli.json ao gerar um projeto usando o Angular CLI.

É bem conhecido que esse arquivo é usado como o esquema de configuração para todo o projeto manipulado pela CLI, incluindo o gerenciamento de diferentes ambientes, testes, proxy, recursos de terceiros e [muitas ferramentas integradas e recursos](https://github.com/angular/angular-cli/wiki/1-x-stories) para o desenvolvimento de uma aplicação.

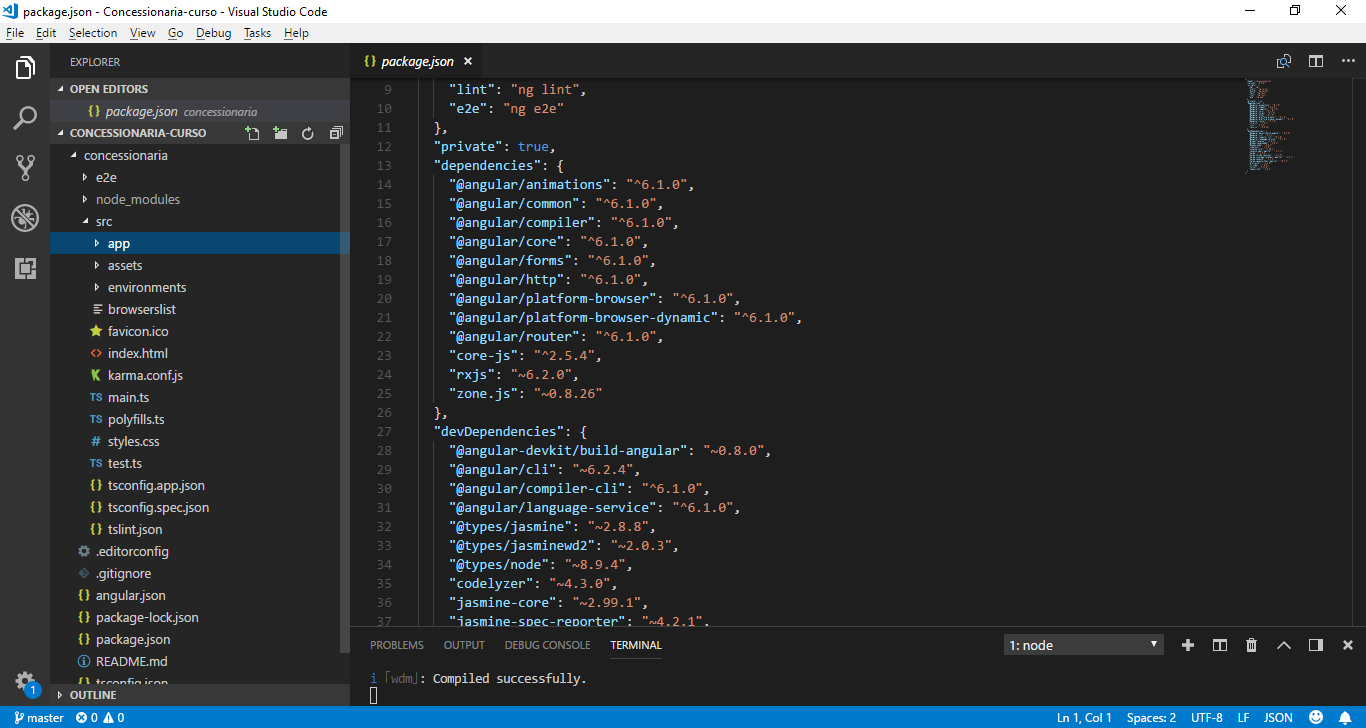
Exemplificando, nele é possível declarar todos os styles e scriptis globais a serem utilizados nas páginas seguindo a estrutura e declarando-os em assets e styles.

### node\_modules

Local onde ficam alocadas todas as dependências do projeto. A pasta node\_modules é uma cópia das dependências instaladas globalmente no ambiente fazendo com que as dependências se tornem módulos internos do projeto prontas para serem utilizadas.

Caso o projeto não possua um node\_module, basta executar o comando **npm install** na raiz do projeto.

### src



### Index, Style e main

Arquivos básicos iniciais (HTML, CSS, JS)

### environment.json

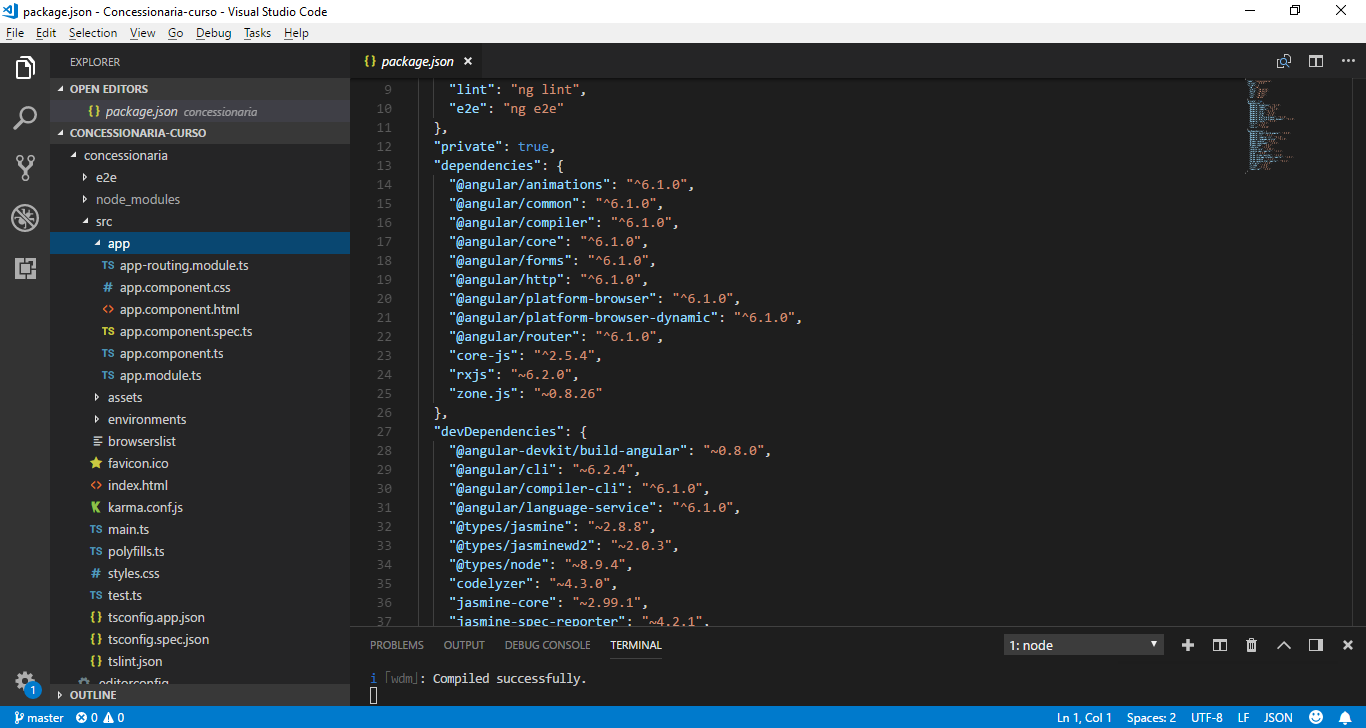
Arquivo no qual encontra-se informações de acesso a API do projeto como URL e porta.

### Assets

Local onde incluímos nossas galeria de imagens, gifs, arquivos em geral da aplicação.

### App

A pasta app é onde se encontra toda nossa estrutura da aplicação desde o modulo principal (app.modulo) até os componentes mais simples.



## Modules

Módulos tem a função de configurar o injetor e o compilador do módulo e auxiliar a organização de todos os componentes relacionados ao mesmo.

Dentro do decorator NgModule, podemos identificar alguns itens importantes na construção de um módulo.

### Imports

Imports são responsáveis por tornar as declarações exportadas de outros módulos disponíveis para o módulo atual.

### Declarations

Em declarations, incluímos todos os Componentes, diretivas, pipes que serão usados pelo módulo atual.

### Providers

Qualquer tipo de serviço, incluímos em providers para que assim possa ser injetado e utilizado pelos componentes do módulo atual através da injeção de dependência.

## Decorators

Decoradores são marcadores que são responsáveis por denominar se uma classe é um módulo, um componente ou até mesmo se um atributo é um dado de entrada ou de saída.

Os decoradores são na verdade apenas funções, são tão simples quanto isso e são chamados com o que quer que estejam decorando. Um decorador de método será chamado com o valor do método que está decorando, e um decorador de classe será chamado com a classe a ser decorada.

Abaixo, podemos identificar os Decorators disponíveis pelo framework:

|  |  |  |
| --- | --- | --- |
| * @NgModule * @Component * @Injectable * @ContentChild * @ContentChildren | * @Pipe * @Input * @Output * @ViewChild | * @Directive * @HostBinding * @HostListener * @ViewChildren |

Referência para mais detalhes: <https://medium.com/@madhavmahesh/list-of-all-decorators-available-in-angular-71bdf4ad6976>

## Diretives

Você pode definir suas próprias diretivas para anexar um comportamento personalizado aos elementos no DOM. As opções fornecem metadados de configuração que determinam como a diretiva deve ser processada, instanciada e usada no tempo de execução.

| iva | Descrição |
| --- | --- |
| ng-app | Aplicativo AngularJS de autoinicialização. |
| ng-init | Inicializa variáveis ​​AngularJS |
| ng-model | Vincula o valor do controle HTML a uma propriedade no objeto $ scope. |
| ng-controller | Anexa o controlador do MVC à visualização. |
| ng-bind | Substitui o valor do controle HTML pelo valor da expressão AngularJS especificada. |
| ng-repeat | Repete o modelo HTML uma vez por cada item na coleção especificada. |
| ng-show | Exibe o elemento HTML com base no valor da expressão especificada. |
| ng-readonly | Torna o elemento HTML somente leitura com base no valor da expressão especificada. |
| ng-desativado | Define o atributo desativar no elemento HTML se a expressão especificada for avaliada como verdadeira. |
| ng-se | Remove ou recria o elemento HTML com base em uma expressão. |
| ng-click | Especifica o comportamento personalizado quando um elemento é clicado. |

Para mais informações: <https://www.tutorialsteacher.com/angularjs/angularjs-directives>

## Components

Fornece metadados de configuração que determinam como o componente deve ser processado, instanciado e usado em tempo de execução.

## Router

Define o objeto Route que mapeia um caminho de URL para um componente e a diretiva RouterOutlet que você usa para colocar uma exibição roteada em um modelo, bem como uma API completa para configurar, consultar e controlar o estado do roteador.

## Lidando com o two-way data binding

No two-way data binding, alterações na view são refletidas na fonte de dados e atualizações na fonte refletem na view sem a necessidade de manipulação explícita do DOM. Vejamos um exemplo:

### Implementando two-way data binding no Angular

Antes de realizarmos um componente utilizando um two-way data binding, devemos importar em nosso app.module o Módulo FormsModule.

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

FormsModule é o módulo principal que manipula as diretivas do Angular.

O two-way data binding no Angular é implementado com o ngModel em um input conforme o exemplo a seguir:



|  |
| --- |
|  |

Quando o evento input é disparado, $event.target representa quem disparou o evento. Sendo um elemento do DOM, podemos acessar seu valor com $event.target.value. É este valor que é atualizado na propriedade palestrante do nosso componente.

Contudo, a equipe do Angular criou uma diretiva que funciona como uma espécie de atalho para esses dois tipos de associação.

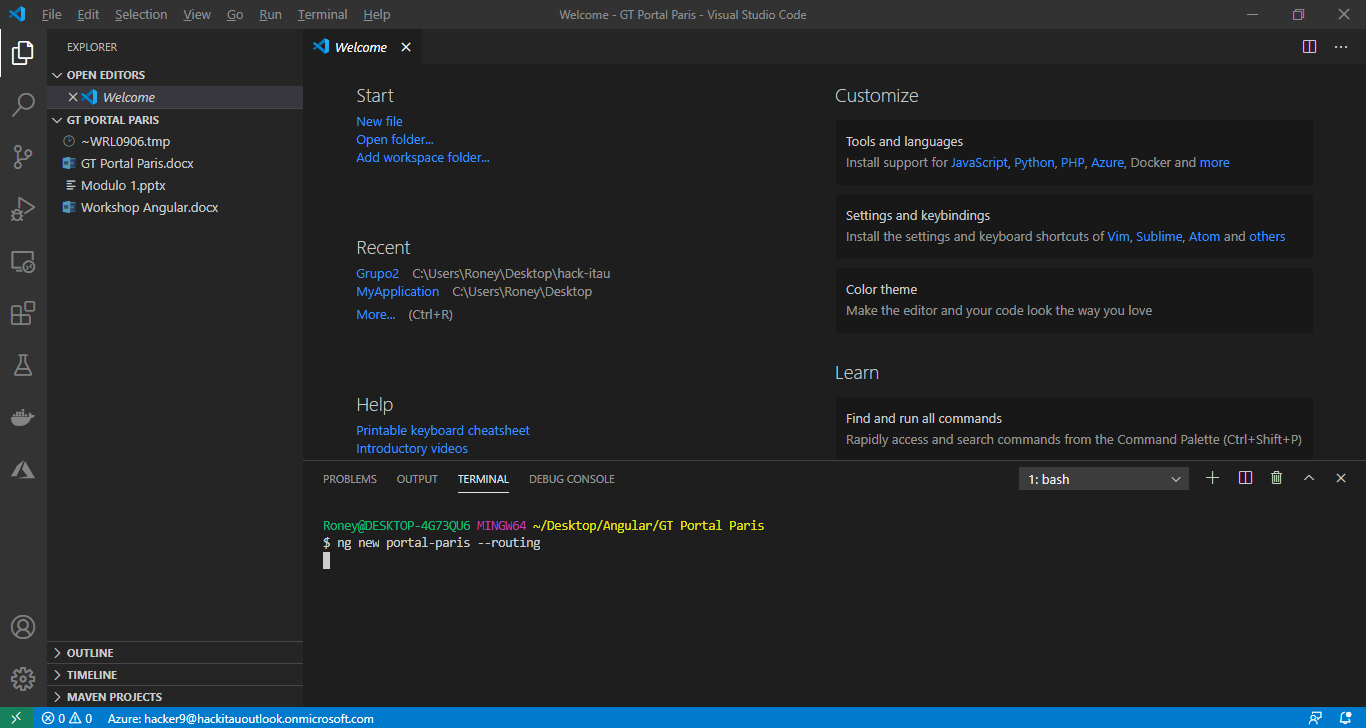
|  |
| --- |
|  |

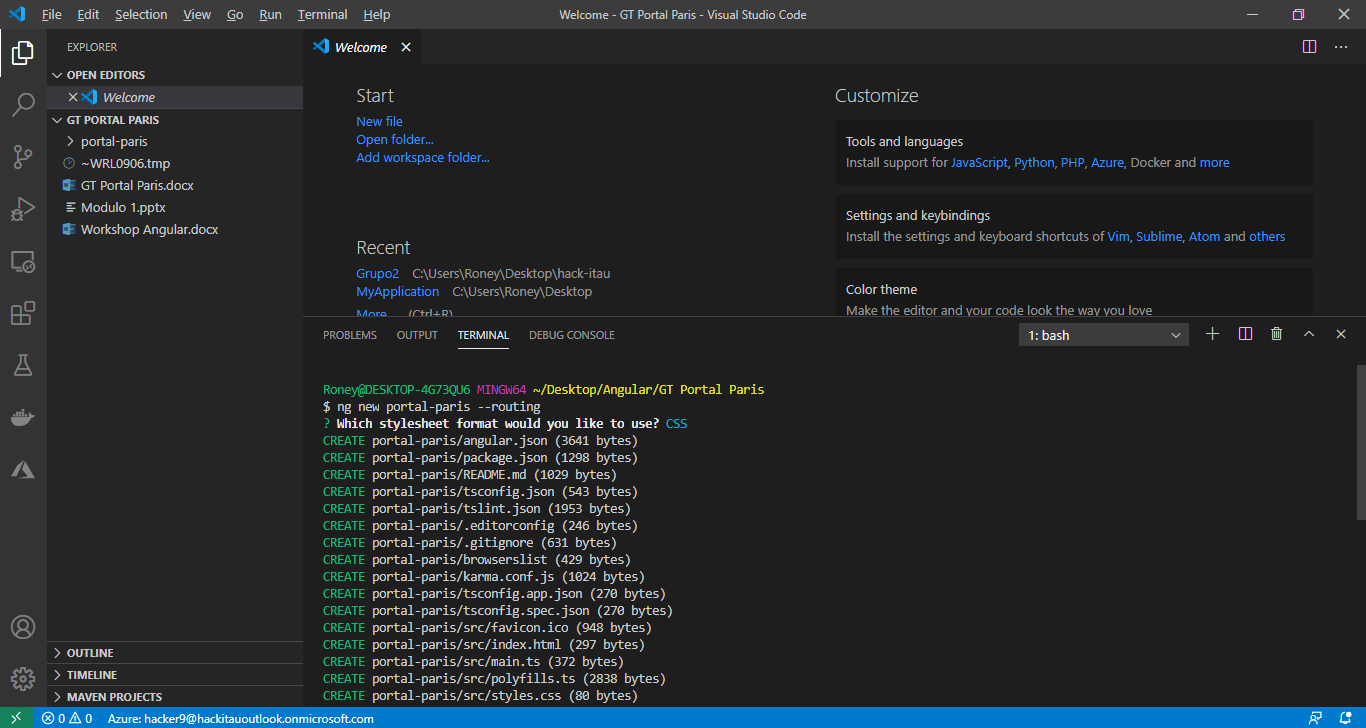
Veja que a diretiva está envolvida por [()], ou seja, temos os dois tipos de associações unidirecionais ao mesmo tempo, mas com direções opostas no fluxo de atualização.

## Criando novo projeto

Para criar um novo projeto vamos abrir nosso Visual Studio Code e digitar o seguinte comando no terminal interno:

**ng new portal-paris --routing**





## Criar novo modulo

**ng g m transferencias --routing**

## Criar novos componentes do modulo

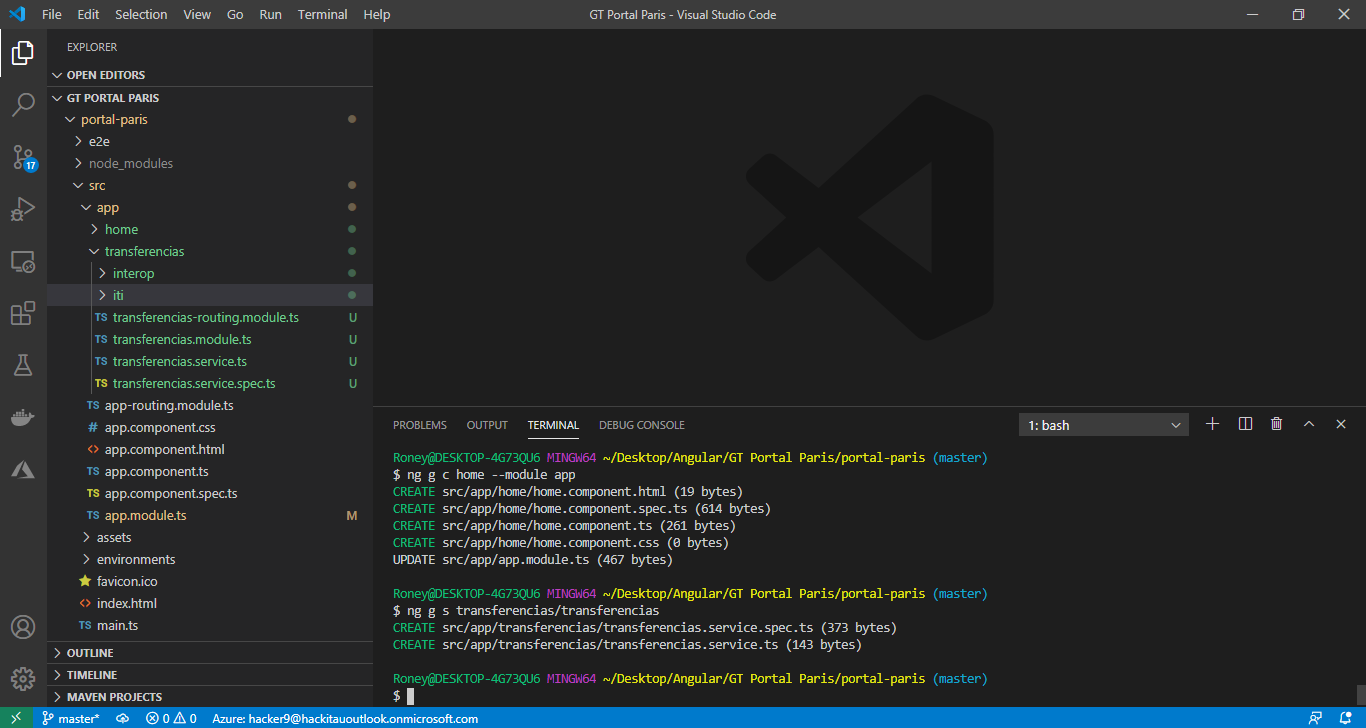
**ng g c transferencias/iti**

**ng g c transferencias/interop**

**ng g c home --module app**

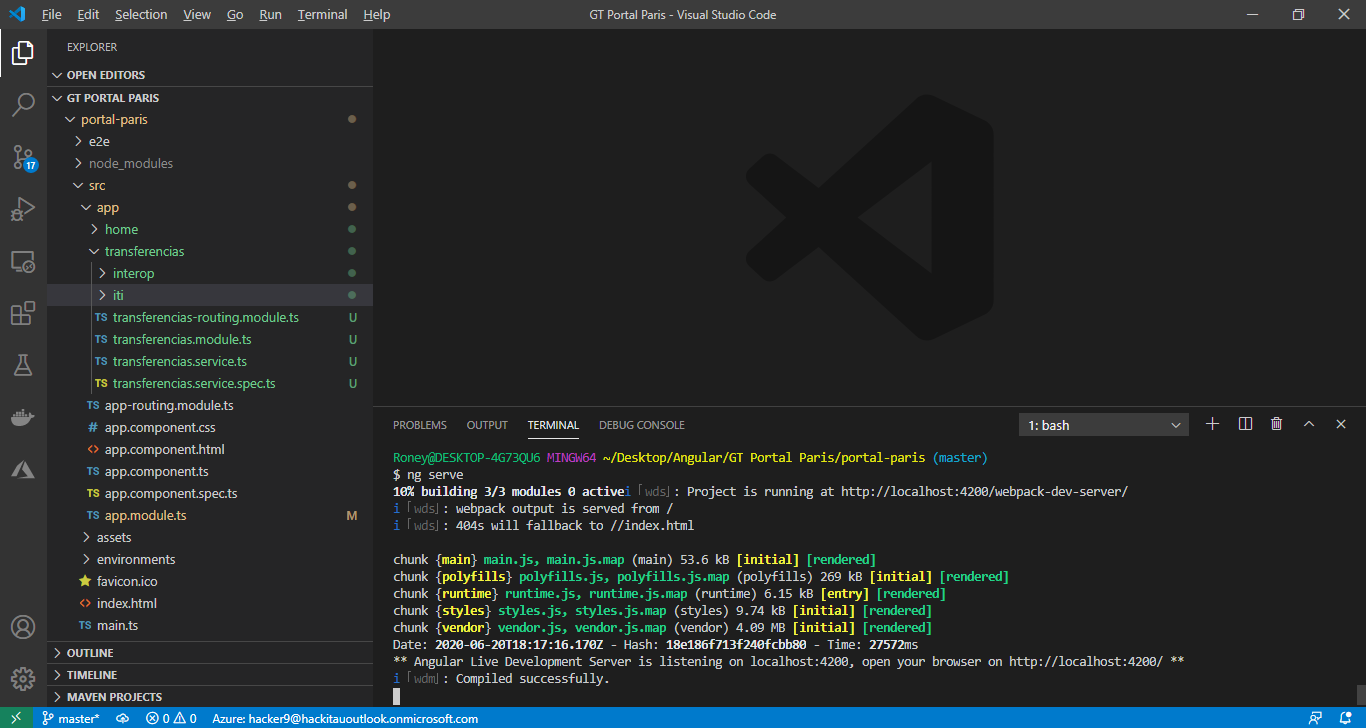
## Criar novos serviço dos componentes

**ng g s transferencias/transferências**

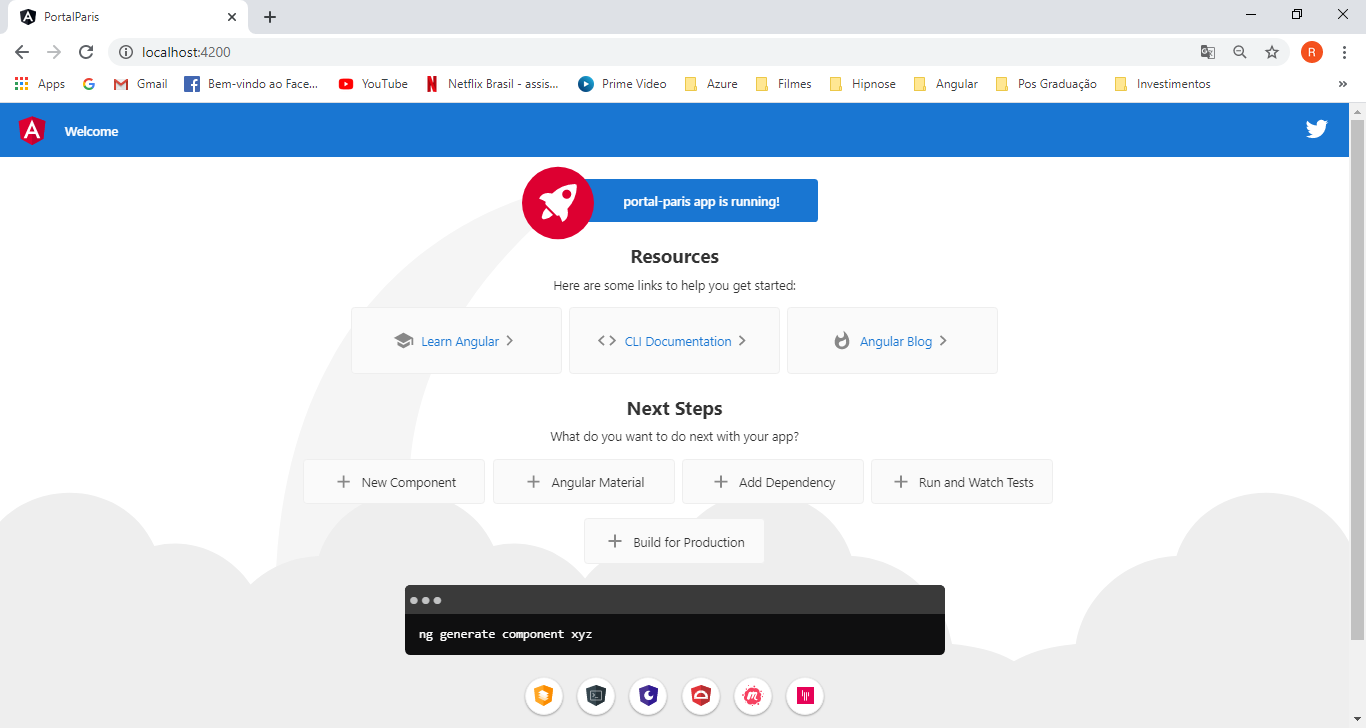


## Inicializando o Projeto

Para inicializar a aplicação, basta executar o comando **ng serve** no terminal conforme abaixo:



Por padrão, a aplicação Angular sobe na porta 4200 quando inicializado.

Para visualizar a página, acesse a rota [**http://localhost:4200/**](http://localhost:4200/) 

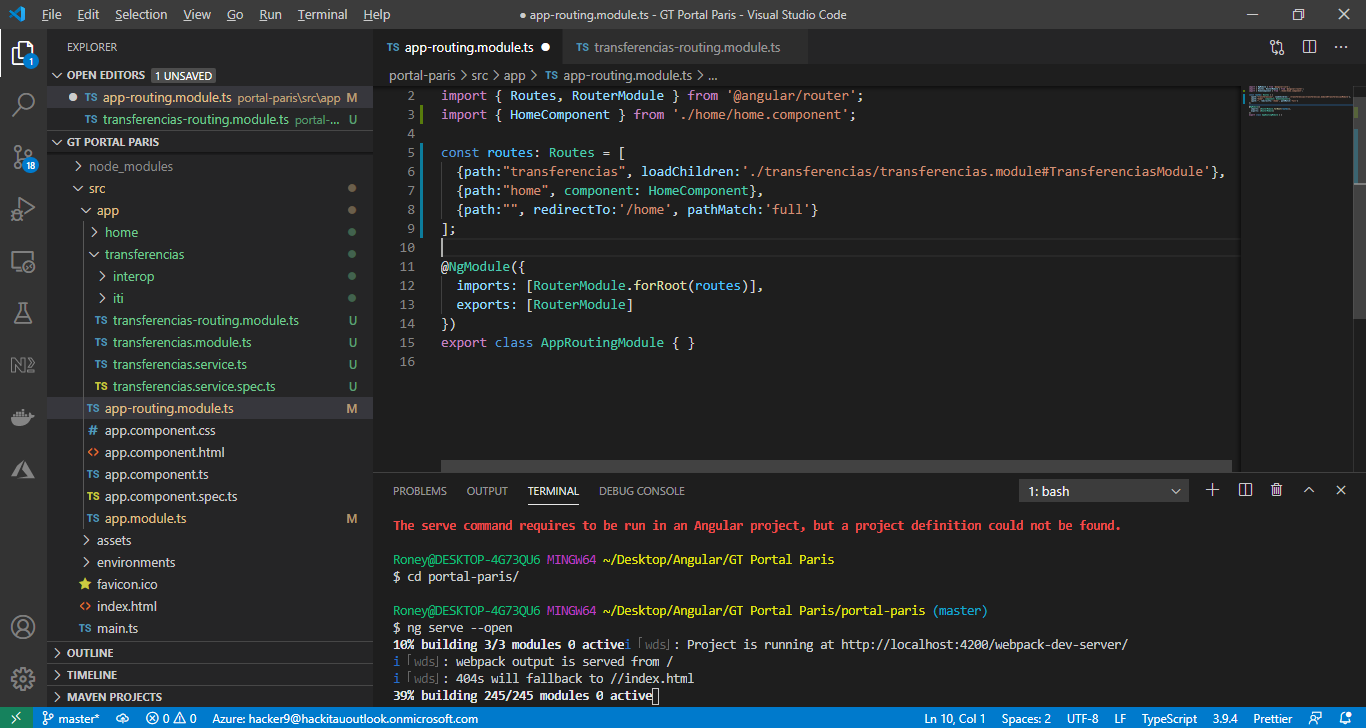
# TRABALHANDO COM ROTAS NO ANGULAR

As rotas da aplicação, por padrão são configuradas em modulos denominados como routing.module.ts

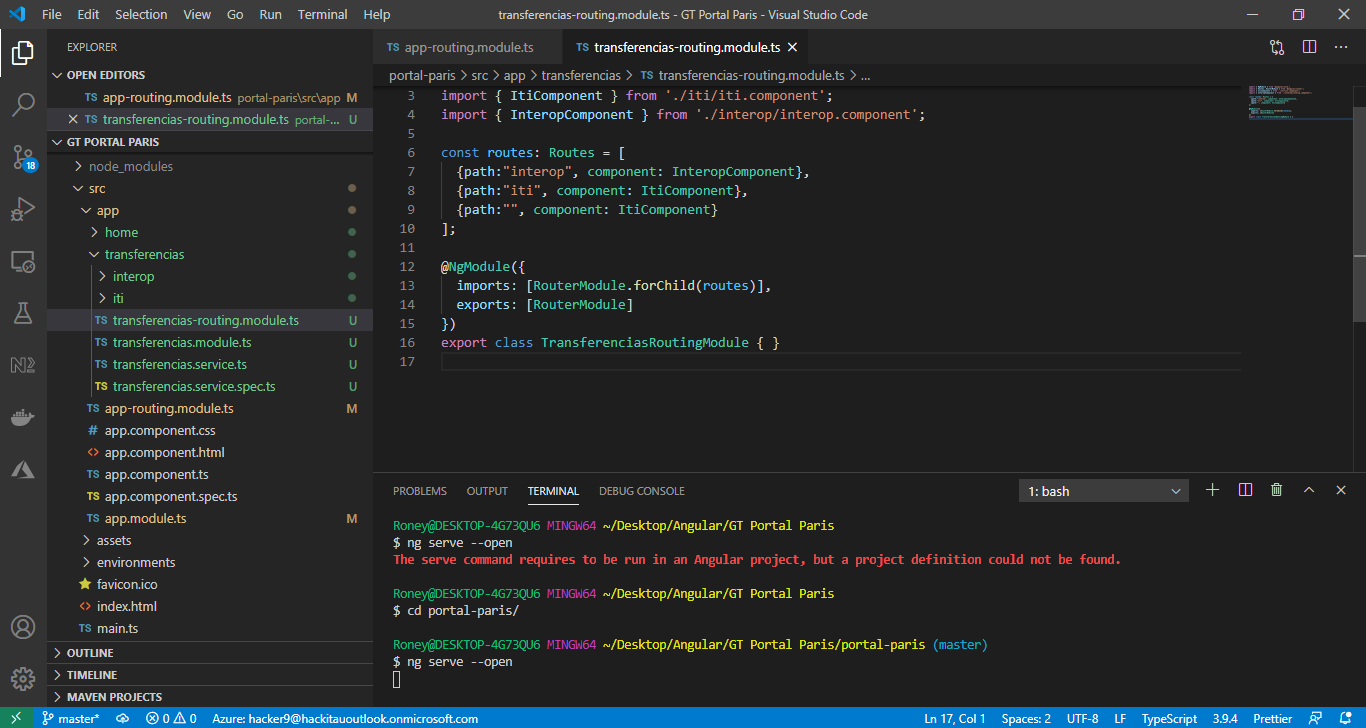
Quando criamos nosso projeto, executamos o comando de novo projeto seguido do comando –routing gerando automaticamente o arquivo app.routing.module.ts.

Podemos destacar o modulo app.routing.module.ts como o roteador principal da aplicação sendo ele o responsável por dar as rotas iniciais dos outros módulos da aplicação e os routing.modules de cada um destes módulos são responsáveis por declarar as rotas específicas dos seus componentes.

## Declarando Rotas em app-routing.module.ts

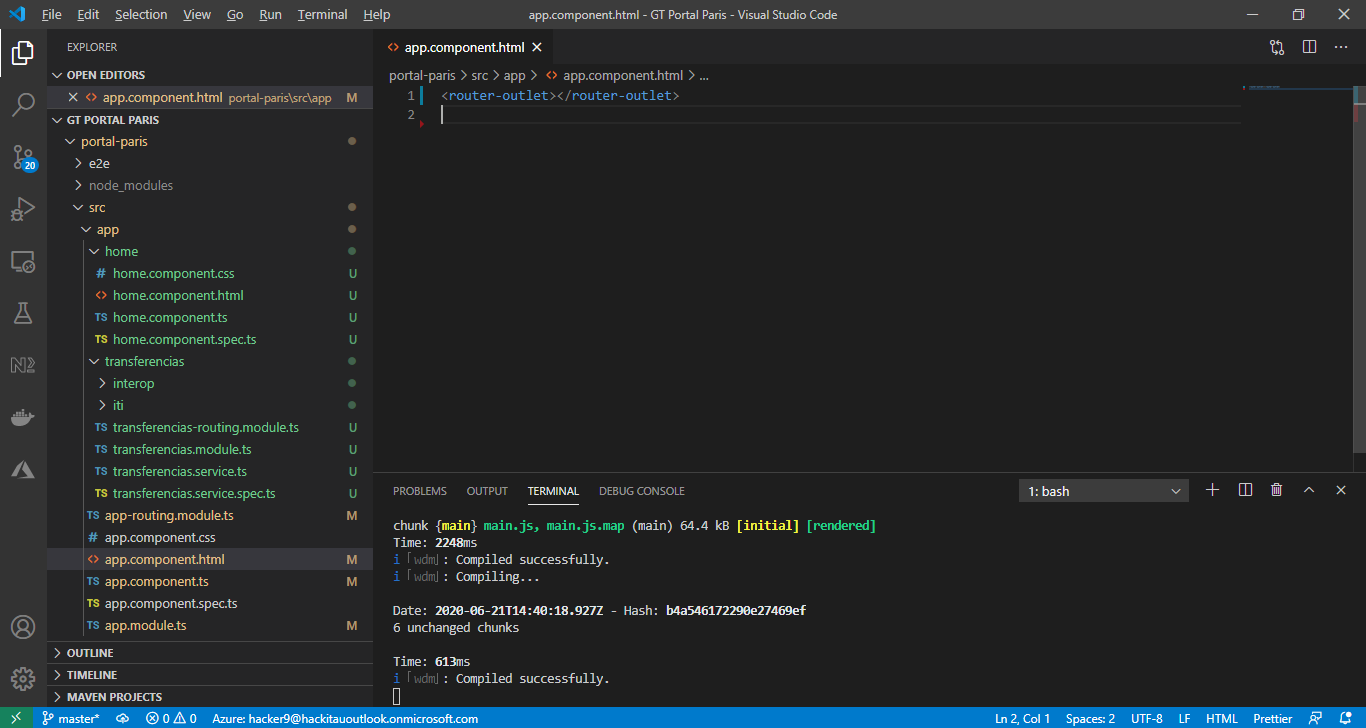


## Declarando Rotas em transferencias-routing.module.ts

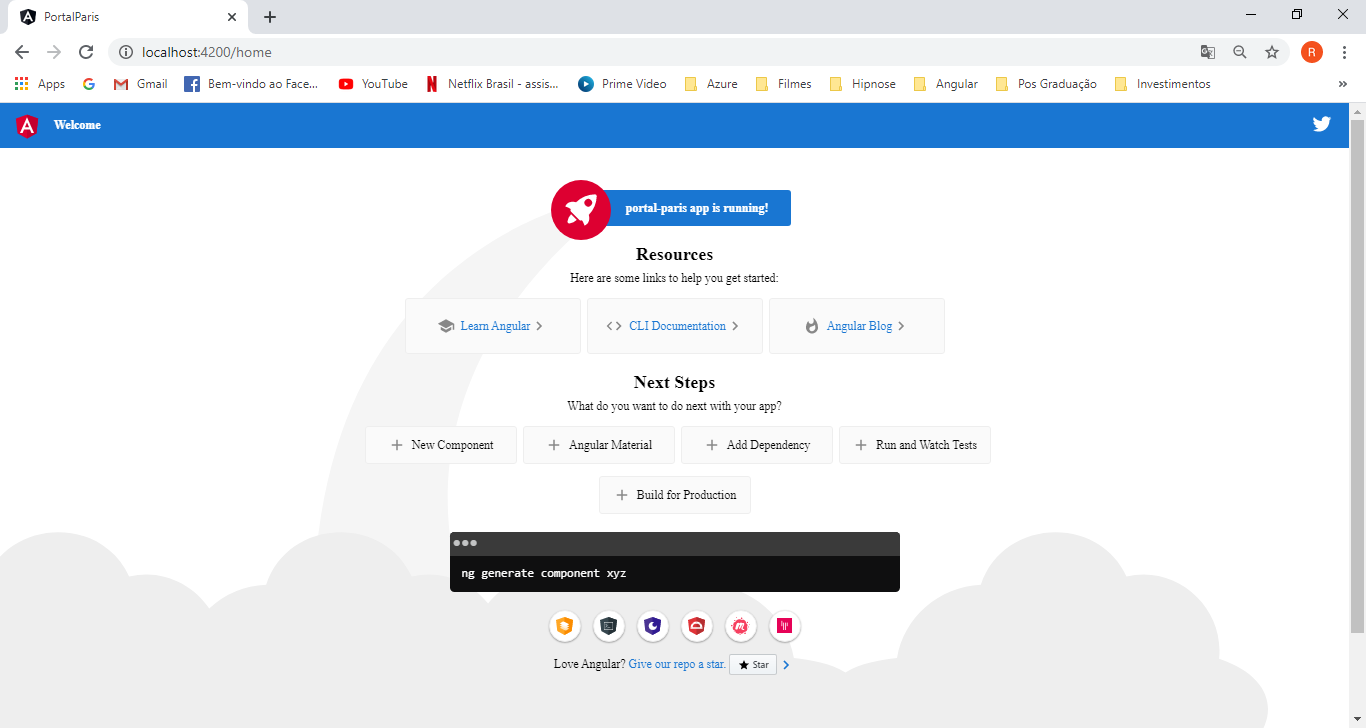


## Router outlet

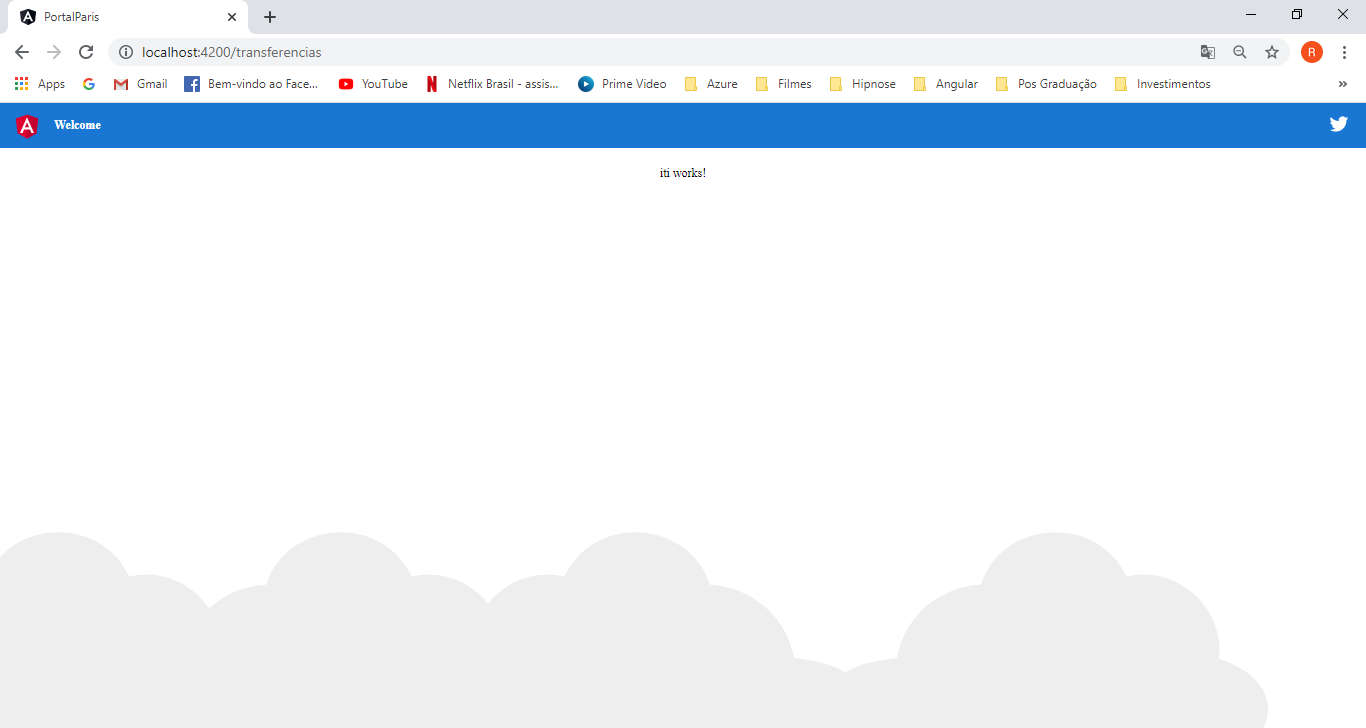
Com as rotas da aplicação definidas e estruturadas, basta incluir em seu app.component.html a seguinte diretiva que fará todo o controle da renderização dos componentes conforme o roteamento declarado.



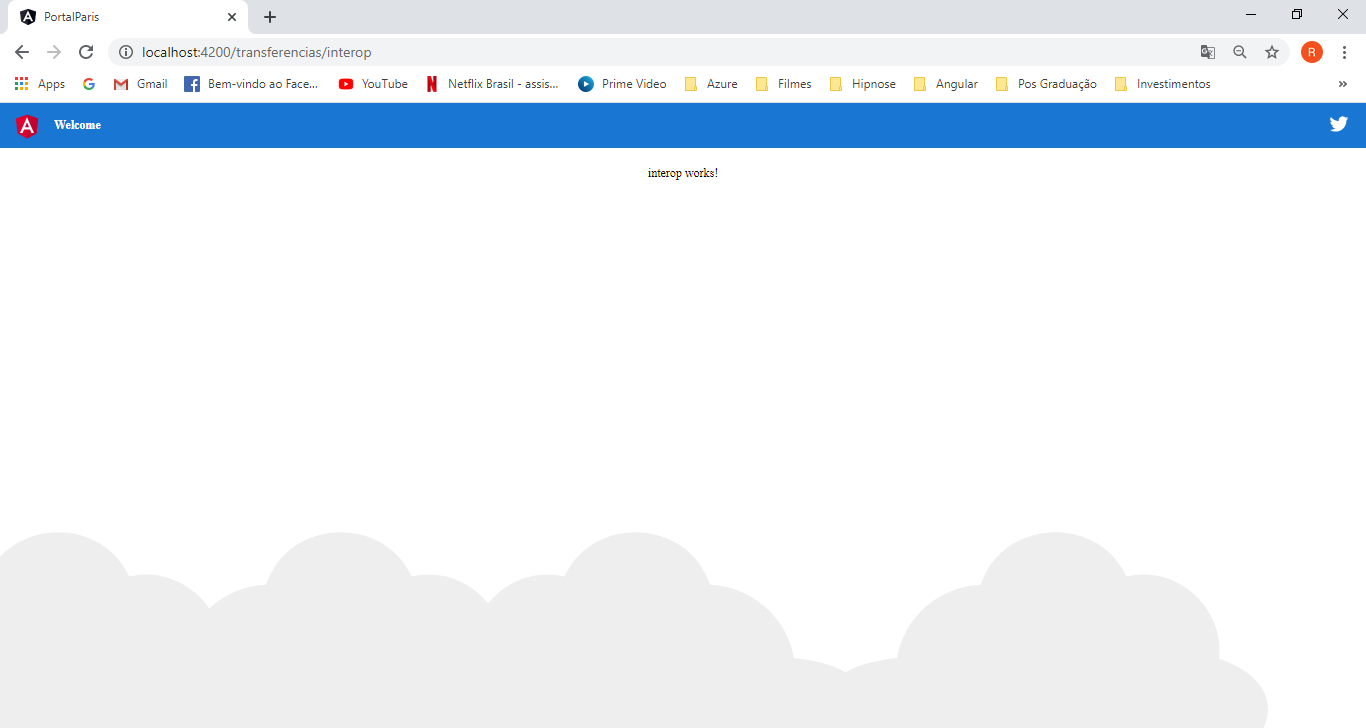
Feito isso, agora pode-se criar uma página home bem legal em seu home.component. No exemplo abaixo, transportei apenas o index padrão que o Angular já nos fornece e externalizei o css para o arquivo css global **style.css,** mas lembre-se, se for fazer isso, não esquece que o <router-outlet> deve permanecer no app.component.html.

/home

/transferencias



/transferencias/interop



# TRABALHANDO COM SHARED.MODULE

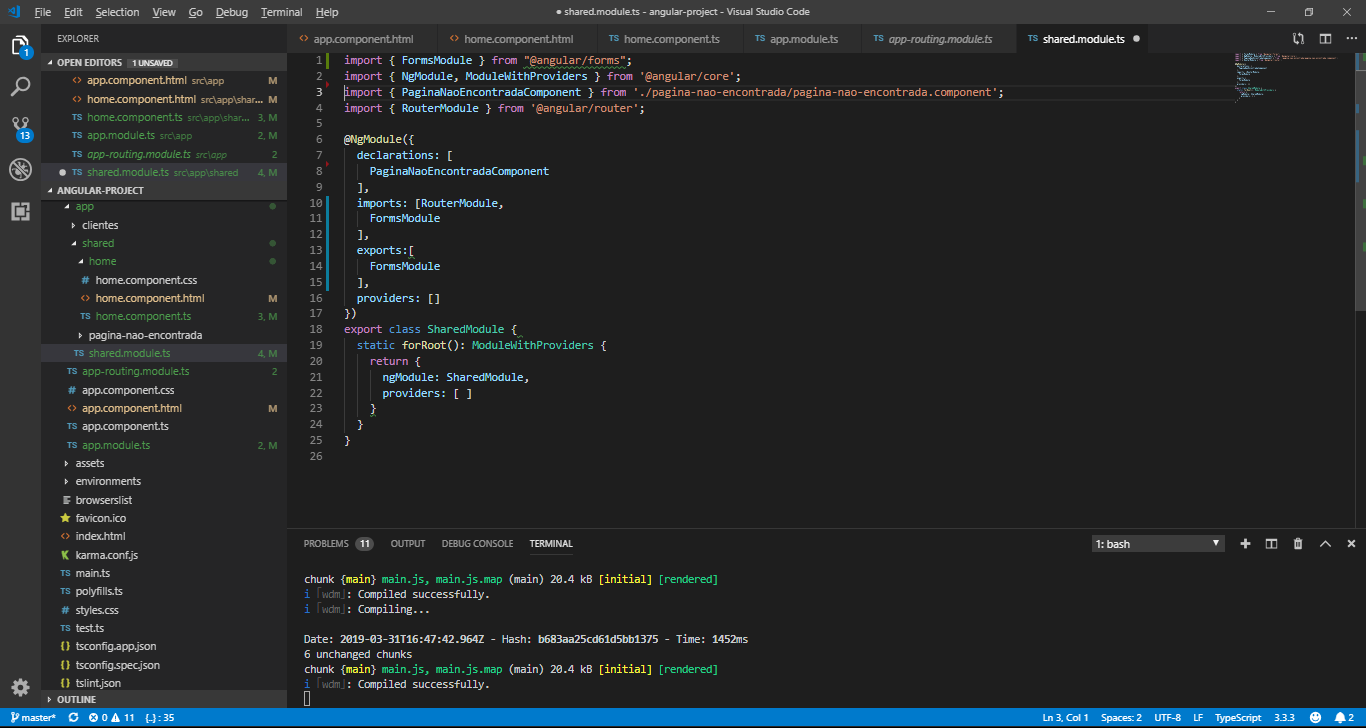
Por questões de organi­zação de código, é comum a prática de criar um Módulo apenas para declarar, importar, prover e exportar componentes, pipes, diretivas, módulos para serem compartilhados para toda a aplicação. Geralmente damos um nome para este módulo: shared.module.

## Utilizando o método estático .forRoot()

Algumas vezes necessitamos compartilhar serviços para que toda a aplicação possa utilizar, mas nem sempre há a necessidade de expor um serviço com múltiplas instancias a cada vez que é solicitado.

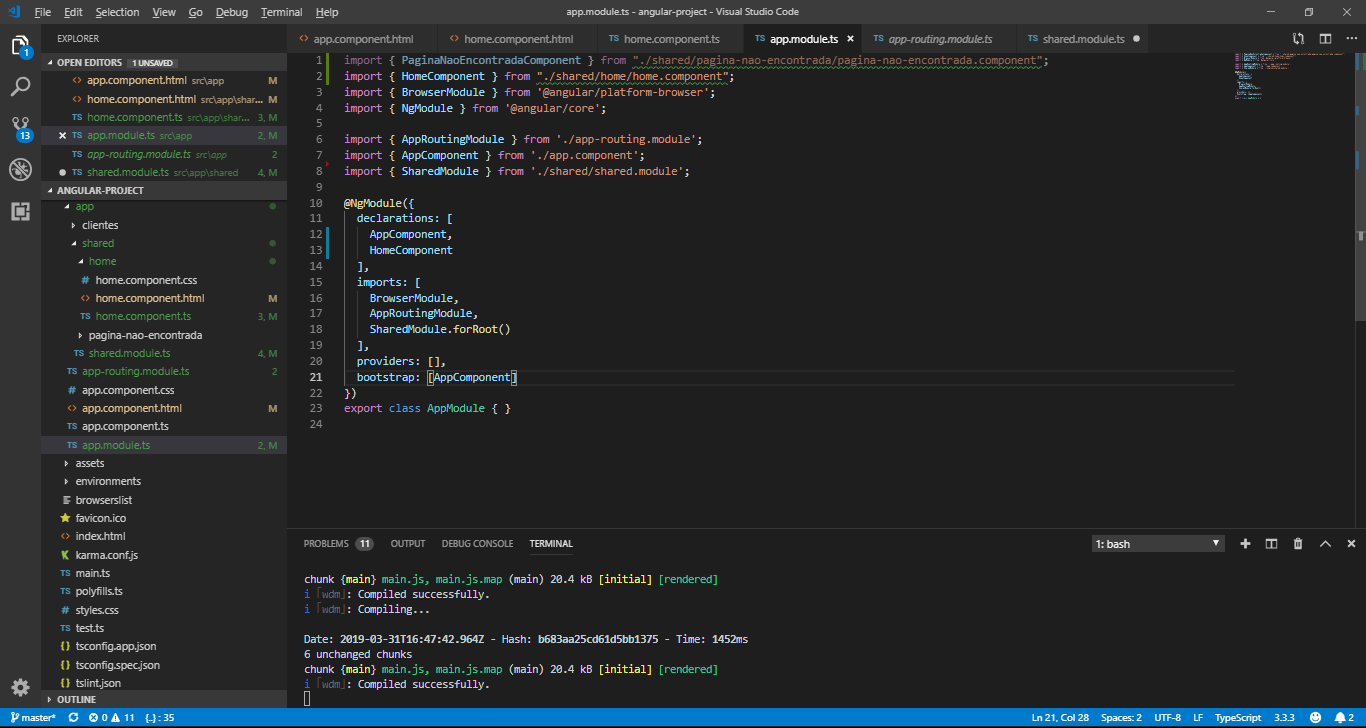
Então para resolver um caso que possa se tornar um problema para aplicação posteriormente, declararemos um método estático chamado forRoot() onde incluiremos todos nossos serviços que devem ser Singletons para aplicação.

## SharedModule



## AppModule

Com o novo modulo de componentes compartilhados, em nosso app.module podemos importar nosso shared.module chamando diretamente o método forRoot():



# CRIANDO UM BANNER GLOBAL, TELA DE LOGIN E CONTROLE DE EXIBIÇÃO DO BANNER

Dentro da pasta shared, criaremos o componente Banner e o componente login para trabalharmos com o sistema de Login mocado e o controle de exibição do Banner ao realizar o login. Para criar os componentes basta executar os comandos:

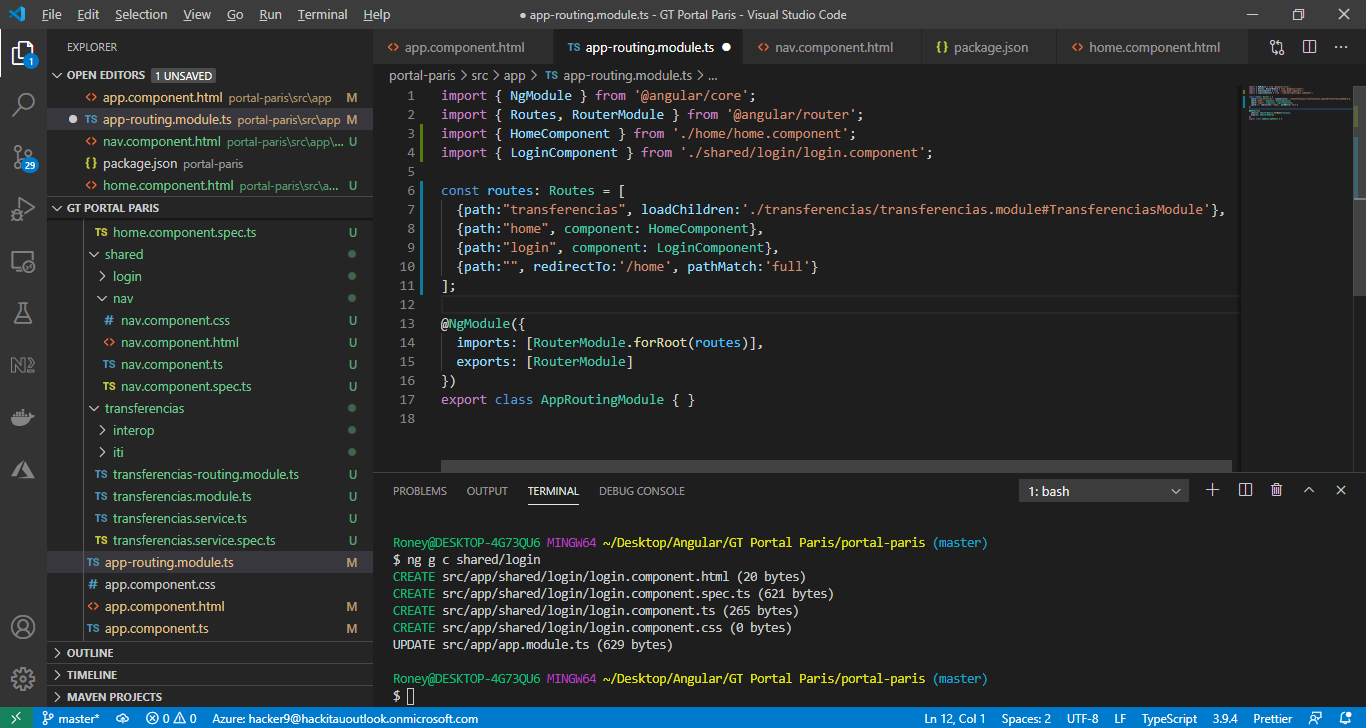
**ng g c shared/nav**

**ng g c shared/login**

Em **app.component.html**, devemos incluir a tag html do nosso componente de menu.

<app-nav></app-nav>

Em app-routing.module, devemos adicionar uma rota específica para nosso componente login



## Implementando o sistema de Login

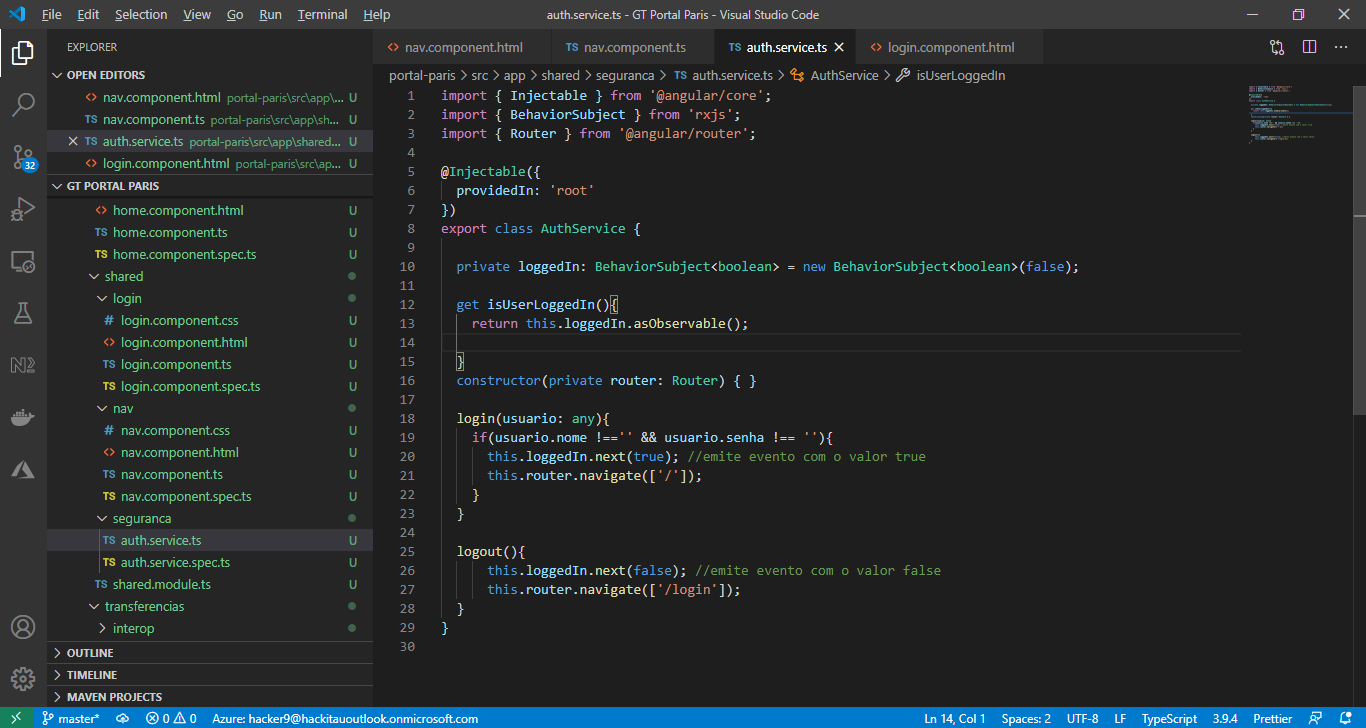
Após criar o menu e a tela de login percebemos que o menu independente da rota sempre irá aparecer. Não é muito amigável exibir um menu na tela de login.

Criar o serviço de autenticação auth.service com o seguinte comando:

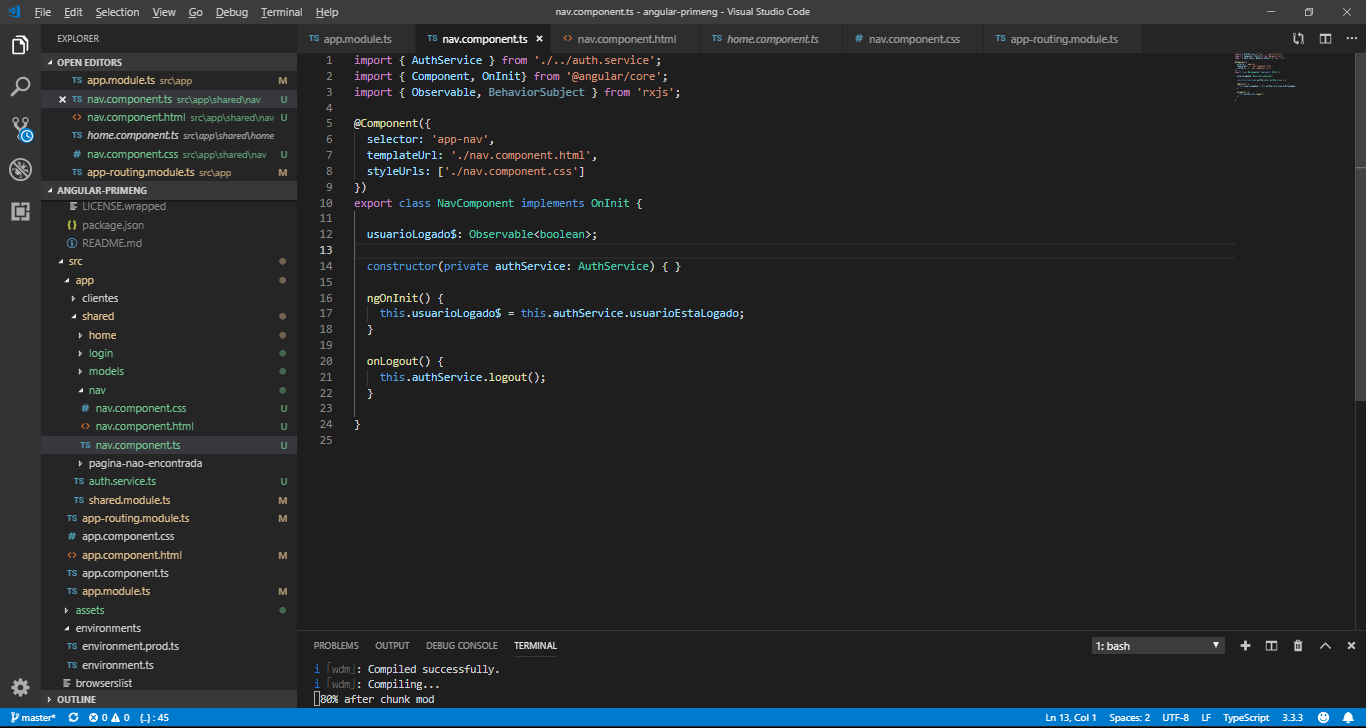
**ng g s shared/seguranca/auth**

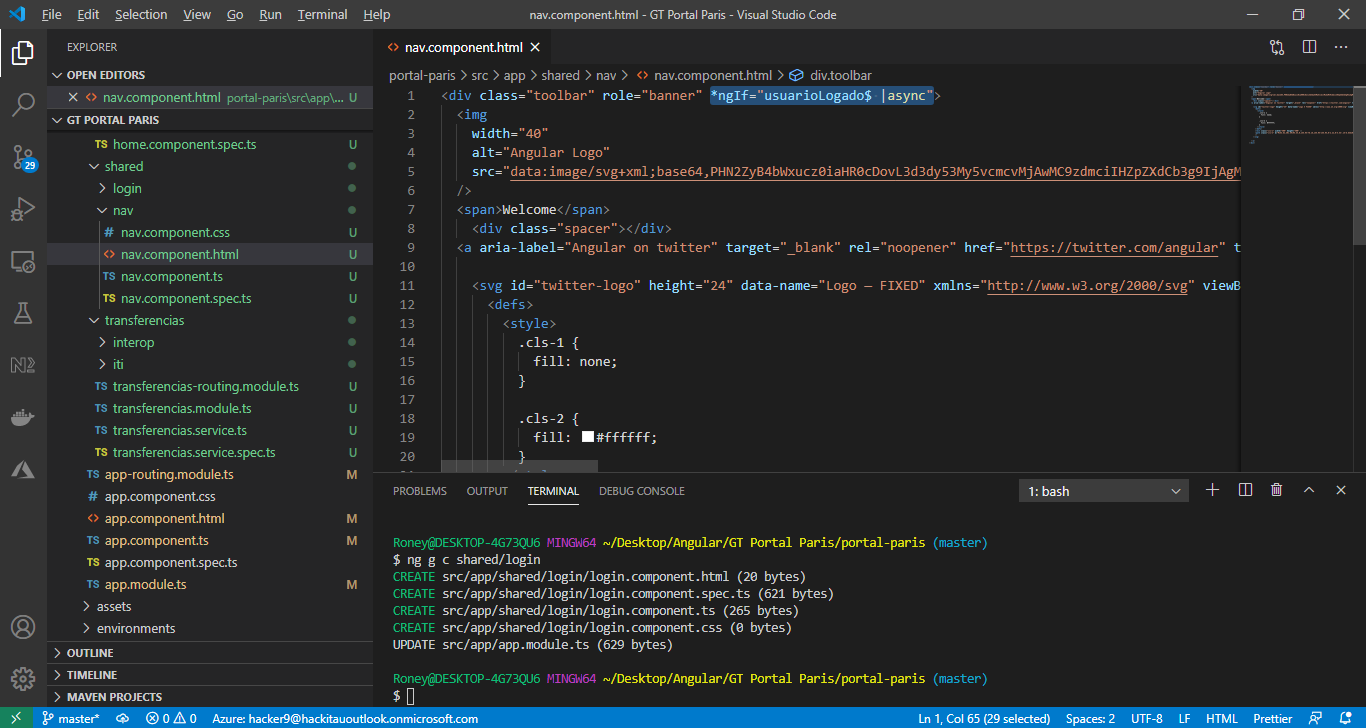
Depois do serviço já está criado, implemente os seguintes itens conforme a imagem abaixo:

* + - Método de login
    - Método de logout
    - Variável de usuário logado ­­­­



Em NavComponent, controlar a exibição do menu com a variável do usuário logado utilizando o pipe |async para garantir que a alteração dos valores da variável usuarioLogado$ reflita na renderização do componente.





Agora em nosso LoginComponent, vamos criar um método de login que solicitará a autenticação e fará a alteração do status do usuário logado.

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

import { AuthService } from '../seguranca/auth.service';

@Component({

  selector: 'app-login',

  templateUrl: './login.component.html',

  styleUrls: ['./login.component.css']

})

export class LoginComponent implements OnInit {

  usuario: any = {};

  constructor(private authService: AuthService) { }

  ngOnInit() {

  }

  login(){

    this.authService.login(this.usuario);

  }

}

Com o script da tela de login criado, finalizamos com o html que representará a tela de login.

<div class="login elevation-box-1200">

  <div class="ui-g ui-fluid">

    <div id="logo" class="ui-g-12">

        <!-- <img class="img-angular" src="../../../assets/images/angular-logo.png" alt="Angular"> -->

        <img class="img-angular" src="../../../favicon.ico" alt="Angular">

    </div>

    <form id="formLogin" autocomplete="off" #formLogin="ngForm">

        <div class="ui-g-12">

            <h1>Faça seu Login:</h1>

        </div>

        <div class="ui-g-12">

          <span class="ui-float-label">

            <input id="usuario" type="text" pInputText  name="usuario" [(ngModel)]="usuario.nome">

            <label for="usuario">Usuario</label>

          </span>

        </div>

        <div class="ui-g-12">

          <span class="ui-float-label">

            <input id="senha" pInputText type="password" name="senha" [(ngModel)]="usuario.senha">

            <label for="senha">Senha</label>

          </span>

        </div>

        <div class="btn-login ui-g-12">

          <button id="btnLogin" pButton type="submit" label="Entrar" (click)="login()">Entrar</button>

        </div>

    </form>

  </div>

</div>

# RXJS - OBSERVABLES E PROMISES

Ambos são encapsuladores e emissores de dados e quem os assinam podem manipular métodos internos de cada um.

## Algumas Diferenças

|  |  |
| --- | --- |
| **Observables** | **Promises** |
| Emite mais de um valor. | Emite um único valor. |
| Evento só é acionado quando há um assinante (Lazy). | Evento é acionado independente se existe um assinante (Eager). |
| Cada subscribe é independente de outro. | Cada then é compartilhado com o outro. |

## Maneiras de definir um atributo:

|  |  |  |
| --- | --- | --- |
|  | **SYNCHRONOUS** | **ASYNCHRONOUS** |
| **MÚLTIPLOS**  **VALORES** | Array, Lista | Observable |
| **ÚNICO**  **VALOR** | Objeto | Promise |

## Observação:

Métodos e serviços baseados em Observables e Promises são asincronos e para resolver qualquer situação de forma sincrona, podemos utilizar os assessores async e await nos métodos.

Informações adicionais, podem ser consultadas neste artigo:

<https://medium.com/tableless/entendendo-rxjs-observable-com-angular-6f607a9a6a00>

# HTTP CLIENT

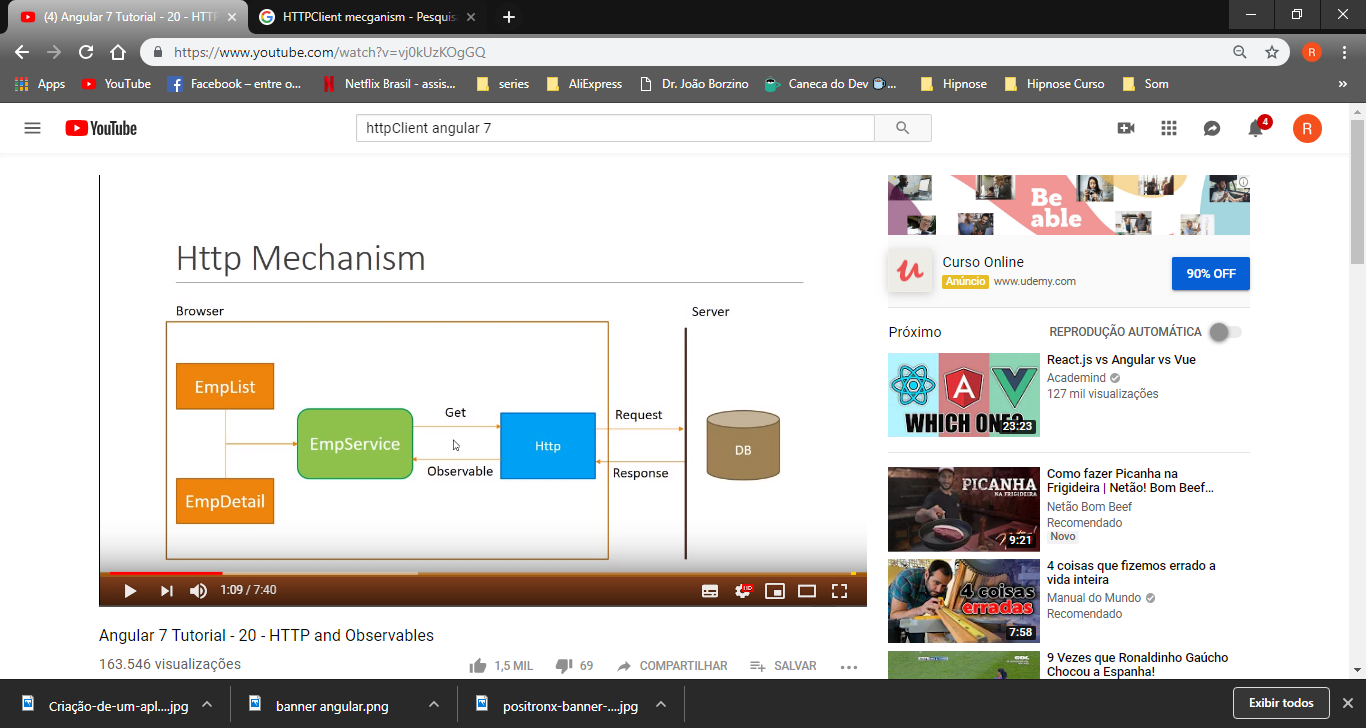
## Introdução ao módulo HTTP CLIENT

O HttpClient é usado para fazer a comunicação entre cliente e servidor usando o protocolo HTTP. Ou seja, se você está querendo consumir dados de uma API externa o HttpClient facilitará essa comunicação, através de muitos métodos disponíveis.

As várias versões do módulo HTTP angular têm uma **API baseada em Observables RxJS**. Isso significa que as múltiplas chamadas para o módulo HTTP sempre retornarão um Observable, que precisamos assinar de uma forma ou de outra.

Aqui estão alguns itens importantes para ter em mente em relação a este tipo específico de Observables retornados pelo módulo HTTP:

* se não assinarmos esses observáveis, nada acontecerá
* se assinarmos várias vezes esses observáveis, várias solicitações HTTP serão acionadas
* se a solicitação HTTP for bem-sucedida, esses observáveis ​​emitirão apenas um valor e, em seguida, serão concluídos
* esses observáveis ​​emitirão um erro se a solicitação HTTP falhar



# TRABALHANDO COM GUARDS

Guards basicamente são guardiões de acesso por rota onde seu papel é verificar se o acesso à rota que está sendo realizado deve ser permitido ou não.

Existem quatro tipos diferentes de Guard:

* **CanActivate**: válida se a rota está ativa
* **CanActivateChild**: válida se a rota filha está ativa
* **CanDeactivate**: verifica se uma rota pode ser desativada
* **CanLoad**: válida se um módulo está utilizando o lazily loading

ng g guard shared/seguranca/auth

## Implementando CanActivate

export class AuthGuard implements CanActivate {

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

canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean> {

if (!this.authService.isLoggedIn()) {

this.authService.refreshToken().subscribe(resp => {

if (this.authService.isLoggedIn()) {

return Observable.create().of(true);

} else {

this.router.navigate(['login']);

return Observable.create().of(false);

}

});

this.authService.logout();

this.router.navigate(['login']);

return Observable.create().of(false);

}

}

}

## Incluindo Guards nas rotas privadas

const routes: Routes = [

{ path: 'transferencias', loadChildren:'./transferencias/transferencias.module#TransferenciasModule', canActivate: [AuthGuard]},

{ path: 'home', component: HomeComponent, canActivate: [AuthGuard] },

{ path: 'login', component: LoginComponent},

{ path: '', redirectTo: '/home', pathMatch: 'full' },

{ path: '\*\*', component: PaginaNaoEncontradaComponent },

];

# JWT

**JWT** ou **JSON Web Token** nada mais é que um objeto **JSON** definido na [RFC 7519](https://tools.ietf.org/html/rfc7519) para realizar transferência informação de permissões de acesso entre duas pontas. Ele é codificado e assinado e possuí o seguinte formato:

header.payload.signature

## Header

No **header** (cabeçalho) ficam os dados do **token**, que informam seu tipo e o algoritmo utilizado em sua assinatura:

{

"alg": "HS256",

"typ": "JWT"

}

## Payload

No **payload** (carga) ficam os dados do usuário e alguns metadados como a expiração do **token**:

{

"sub": "1234567890",

"name": "John Do",

"iat": 1516239022

}

Para melhor esclarecendo, o Payload é um objeto JSON com as Claims (informações) da entidade tratada, normalmente o usuário autenticado.

Essas claims podem ser de 3 tipos:

### Reserved claims:

Atributos não obrigatórios (mas recomendados) que são usados na validação do token pelos protocolos de segurança das APIs.

* sub (subject) = Entidade à quem o token pertence, normalmente o ID do usuário;
* iss (issuer) = Emissor do token;
* exp (expiration) = Timestamp de quando o token irá expirar;
* iat (issued at) = Timestamp de quando o token foi criado;
* aud (audience) = Destinatário do token, representa a aplicação que irá usá-lo.

Geralmente os atributos mais utilizados são: **sub**, \*\*iss \*\*e **exp**.

### **Public claims:**

Atributos que usamos em nossas aplicações. Normalmente armazenamos as informações do usuário autenticado na aplicação.

* name
* roles
* permissions

### Private claims:

Atributos definidos especialmente para compartilhar informações entre aplicações.

## Signature

Finalmente na **signature** (assinatura) os dados de **header** e **payload** codificados em **base 64** e unidos por . (ponto) para serem assinados usando o algoritmo definido no **header**:

HMACSHA256(

base64UrlEncode(header) + "." +

base64UrlEncode(payload),

secret

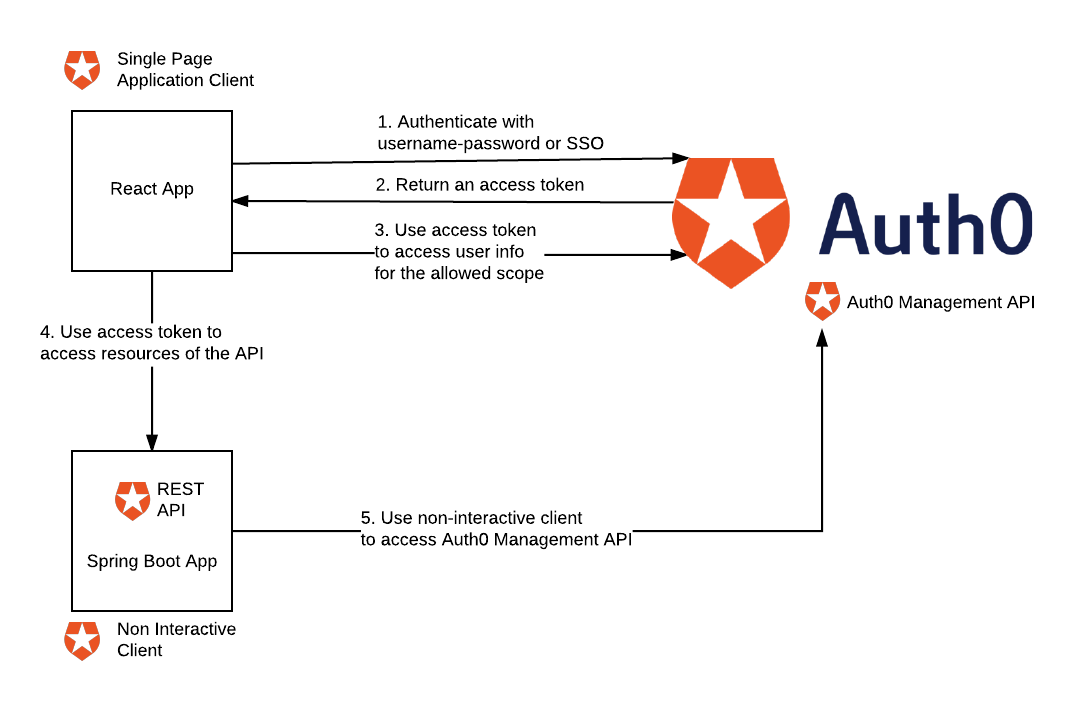
)

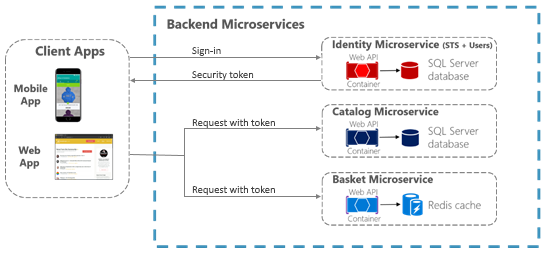
Com a assinatura é possível verificar se o **token** não foi alterado no caminho, garantindo sua integridade. Com ela também é possível confirmar a autenticidade de sua fonte.

Estes três blocos unidos por. (ponto) cada um codificado em **base 64** compõem o **JWT Token**:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.PcmVIPbcZl9j7qFzXRAeSyhtuBnHQNMuLHsaG5l804A

## Fluxo de Autenticação com JWT





## JWT no Angular

Para auxiliar na implementação do **frontend** utilizaremos algumas bibliotecas:

npm install -s moment

npm install -s jwt-decode

npm install -s @types/jwt-decode

A biblioteca [moment](https://momentjs.com/" \t "_blank) facilitará o trabalho com tempo, já que precisamos controlar a expiração do **token**e sua renovação enquanto a biblioteca [jwt-decode](https://github.com/auth0/jwt-decode" \t "_blank) cuidará do **token** em si.

## Resumindo

A classe AuthService é a responsável por no autenticar de fato no sistema. Ela implemanta as funções de login, logout e refreshToken para fazer a manutenção da sessão no sistema, auxiliada das funções setSessionque salva a sessão em **local storage**, getExpiration que realiza o cálculo de expiração para a função refreshToken decidir se é hora de atualizar o token ou não, os pares isLoggedIn e isLoggedOut que são utilizados para verificar se o usuário está logado e o **getter** token que retorna o **JWT** para ser utilizado nas requisições autenticadas.

A classe AuthInterceptor implementa os interceptadores do Angular, que neste caso intercepta todas as requisições **http** realizadas e, caso o usuário esteja logado, injeta o cabeçalho Authorization JWT <token> na requisição para realizar chamadas autenticadas na **API**.

A classe AuthGuard serve como um escudo que impede o acesso de usuário não logado nas rotas em que ela for vinculada, se um usuário não logado tenta acessar determinada rota protegida por ela, o mesmo será redirecionado para a tela de login.

## Exemplo de um Token de Acesso e seu Payload

### ACCESS\_TOKEN

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c3IiOiJyYW1vcmltIiwibmFtZSI6IlJPTkVZIEFVR1VTVE8gQU1PUklNIiwic3ViIjoiMDMzMjIwOTkzIiwiZXhwIjoiMjAyMC0xMi0wMSAwMDowMDowMCIsImF1dGhvcml0aWVzIjpbIkdfUFRQU19UUkFOU0ZFUkVOQ0lBU19FU0NSSVRBIiwiR19QVFBTX1RSQU5TRkVSRU5DSUFTX0NPTlNVTFRBIiwiR19QVFBTX1BPUlRBTF9BRE1JTiJdfQ.-cZAGeuAPHBZ9Kwl5YiECnXwaEzojNDsz6a\_mulA4fQ

### PAYLOAD

{

  "usr": "ramorim",

  "name": "RONEY AUGUSTO AMORIM",

  "sub": "033220993",

  "exp": "2020-12-01 00:00:00",

  "authorities": [

    "G\_PTPS\_TRANSFERENCIAS\_ESCRITA",

    "G\_PTPS\_TRANSFERENCIAS\_CONSULTA",

    "G\_PTPS\_PORTAL\_ADMIN"

  ]

}

# SEGREGANDO NÍVEL DE ACESSO POR ROLES

Como podemos observar na autenticação com o uso do JWT, existem algumas propriedades no response da autenticação e um destes atributos, é o authorities.

Nele é possível obtermos todas as roles que o usuário logado possui e está roles, são as chaves que darão acesso as rotas, aos componentes, aos eventos, ao o que for descrito como nível de acesso.

Com estas informações e a aplicação bem estruturada, a segregação de acesso, menu e chaveamento ficará simples e muito fácil de fazer.

## Passo 1:

Implementar métodos para verificar se o usuário logado possui roles de acesso:

hasPermission(role: string) {

return this.payload && this.payload.authorities.includes(role);

}

hasAnyPermission(roles: string[]) {

for (const role of roles) {

if (this.hasPermission(role)) {

return true;

}

}

}

## Passo 2:

Implementar verificação de acesso no AuthGuard baseando-se nas roles também

import { AuthService } from "./auth.service";

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

import { ActivatedRouteSnapshot, RouterStateSnapshot, CanActivate, Router } from '@angular/router';

import { Observable } from 'rxjs';

import { of } from 'rxjs';

@Injectable({

providedIn: 'root'

})

export class AuthGuard implements CanActivate {

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

canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<boolean> {

let loggedIn: boolean;

this.authService.isLoggedin.subscribe(x => { loggedIn = x });

if (!loggedIn) {

this.authService.refreshToken().subscribe(resp => {

if (this.authService.isLoggedIn()) {

return of(next.data.roles && this.authService.hasAnyPermission(next.data.roles));

} else {

this.router.navigate(['login']);

return of(false);

}

});

this.authService.logout();

this.router.navigate(['login']);

return of(false);

}

return of(next.data.roles && this.authService.hasAnyPermission(next.data.roles));

}

}

Nos routings modules, os menus protegidos pelo AuthGuard, basta passar no atributo data as roles do módulo, contexto, funções conforme o exemplo abaixo:

const routes: Routes = [

{ path: 'transferencias', loadChildren:'./transferencias/transferencias.module#TransferenciasModule', canActivate: [AuthGuard],

data:{ roles: ['G\_PTPS\_TRANSFERENCIAS\_ESCRITA, 'G\_PTPS\_TRANSFERENCIAS\_CONSULTA'] }},

{ path: 'home', component: HomeComponent, canActivate: [AuthGuard]},

{ path: 'login', component: LoginComponent},

{ path: '', redirectTo: '/home', pathMatch: 'full' },

{ path: '\*\*', component: PaginaNaoEncontradaComponent },

];

Para manipular o acesso de menus com a roles basta utilizar os métodos do auth service de verificação de permissão passando as roles desejadas como parâmetros para cada item da lista de menus:

this.items = [

{ label: 'Home', icon: 'fa fa-home', routerLink:['/'] },

{ label: 'Transferências', icon: 'fa fa-users', routerLink:['/transferencias],

visible: this.authService.hasAnyPermission([' G\_PTPS\_TRANSFERENCIAS\_ESCRITA', 'G\_PTPS\_TRANSFERENCIAS\_CONSULTA']),

}

];

# FONTES

* <https://angular.io/>
* <https://cli.angular.io/>
* <https://update.angular.io/>
* <https://www.tutorialsteacher.com/angularjs/angularjs-directives>
* <https://medium.com/@madhavmahesh/list-of-all-decorators-available-in-angular-71bdf4ad6976>
* <https://medium.com/tableless/entendendo-rxjs-observable-com-angular-6f607a9a6a00>
* <https://jwt.io/>
* <https://humberto.io/pt-br/blog/autenticacao-jwt-com-angular-e-django/>
* <https://jasonwatmore.com/post/2018/05/23/angular-6-jwt-authentication-example-tutorial>
* <https://stackblitz.com/edit/angular-login-hide-navbar-ngif?file=src%2Fapp%2Fauth%2Fauth.service.ts>
* <https://github.com/roney-amorim/angular-jwt-authentication>