

Workshop Angular 7

Na aula 2, realizamos nosso primeiro, iniciamos o entendimento do PrimeNG com um exercício prático e implementamos uma camada de autorização para uma tela de login com o controle de exibição de menu.

Neste módulo trabalharemos com camadas de serviços e autenticação com padrão JWT.

HttpClient

Primeiramente, vamos entender o que é e como funciona Gerenciador de Requisição HttpClient.

### Introdução ao novo módulo HTTP

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 algumas coisas 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 (veja este [post](http://blog.angular-university.io/angular-2-rxjs-common-pitfalls/) para mais detalhes)
* Esse tipo específico de Observables são fluxos de valor único: 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, mais sobre isso mais tarde.

Exercício HttpClient – Criando uma chamada Http

Ng g s shared/seguranca/auth

**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 g shared/seguranca/auth

Implementar CanActivate

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

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

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

import { Observable } from 'rxjs';

import { map, take } from 'rxjs/operators';

@Injectable({

providedIn: 'root'

})

export class AuthGuard implements CanActivate {

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

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

return this.authService.isLoggedIn.pipe(

take(1),

map((isLoggedIn: boolean) => {

if (!isLoggedIn) {

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

return false;

}

return true;

})

);

}

}

Incluir Guards nas rotas privadas

const routes: Routes = [

{ path:'clientes', loadChildren:'./clientes/clientes.module#ClientesModule', 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

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

{

"alg": "HS256",

"typ": "JWT"

}

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

{

"sub": "1234567890",

"name": "John Do",

"iat": 1516239022

}

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

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

npm install -s moment

npm install -s jwt-decode

npm install -s @types/jwt-decode

A biblioteca [moment](https://momentjs.com/) 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) 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**.

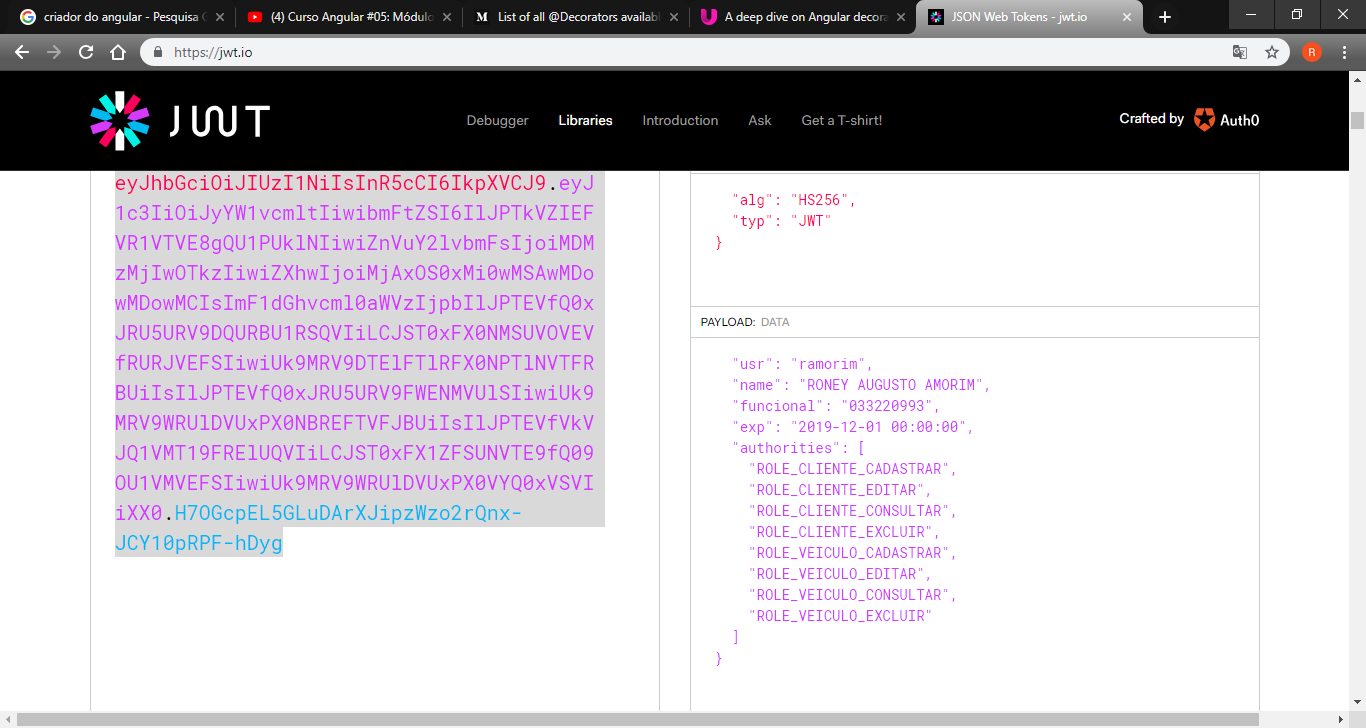
A classe AuthGuard serve como um escudo que impede o acesso de usuário não logado nas rotas em que ela for vinculada, se um usuário não logado tenta acessar determinada rota protegida por ela, o mesmo será redirecionado para a tela de login.

Interceptors de segurança

Segregando nível de acesso por Roles

Token de acesso

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c3IiOiJyYW1vcmltIiwibmFtZSI6IlJPTkVZIEFVR1VTVE8gQU1PUklNIiwiZnVuY2lvbmFsIjoiMDMzMjIwOTkzIiwiZXhwIjoiMjAxOS0xMi0wMSAwMDowMDowMCIsImF1dGhvcml0aWVzIjpbIlJPTEVfQ0xJRU5URV9DQURBU1RSQVIiLCJST0xFX0NMSUVOVEVfRURJVEFSIiwiUk9MRV9DTElFTlRFX0NPTlNVTFRBUiIsIlJPTEVfQ0xJRU5URV9FWENMVUlSIiwiUk9MRV9WRUlDVUxPX0NBREFTVFJBUiIsIlJPTEVfVkVJQ1VMT19FRElUQVIiLCJST0xFX1ZFSUNVTE9fQ09OU1VMVEFSIiwiUk9MRV9WRUlDVUxPX0VYQ0xVSVIiXX0.H7OGcpEL5GLuDArXJipzWzo2rQnx-JCY10pRPF-hDyg



https://www.youtube.com/watch?v=z4JUm0Bq9AM

<https://stackoverflow.com/questions/37364973/promise-vs-observable>

Referencias

<https://www.devmedia.com.br/novidades-do-angular-7/40194>

<https://angular.io/cli>

<https://blog.angular.io/version-6-of-angular-now-available-cc56b0efa7a4>

<https://ultimatecourses.com/blog/angular-decorators>

<https://blog.caelum.com.br/angular-2-o-fim-do-two-way-data-binding/>

<https://imasters.com.br/front-end/o-que-ha-de-novo-no-angular-6>

Pré Requisitos

Git Bash

Visual Studio Code

Possíveis Problemas

Proxy e comunicação por IP

1