**Estrategia para Manejar Sesiones de Usuarios Autenticados con Tokens JWT**

**Descripción**

En nuestra aplicación, utilizamos tokens JWT (JSON Web Tokens) para autenticar a los usuarios y proteger ciertos endpoints en nuestro backend API REST desarrollado en Node.js y MySQL. Los usuarios autenticados obtienen un token JWT válido durante el proceso de inicio de sesión, y este token se almacena en el localStorage del navegador. Cuando los usuarios intentan acceder a los endpoints protegidos, el token se adjunta automáticamente a las solicitudes mediante un interceptor en Angular.

**Pasos**

**1. Configuración Inicial**

Asegúrate de tener los siguientes elementos configurados en tu proyecto:

Un backend API REST desarrollado en Node.js y MySQL que implemente la autenticación de usuarios y genere tokens JWT válidos.

Un servicio de autenticación en Angular que maneje el inicio de sesión de los usuarios y almacene el token JWT en el localStorage.

**2. Creación del Interceptor**

Crea un interceptor en Angular que se encargue de adjuntar el token JWT a las solicitudes salientes a los endpoints protegidos. Esto se hace para garantizar que los usuarios autenticados puedan acceder a los recursos protegidos.

// auth.interceptor.ts

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

import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent } from '@angular/common/http';

import { Observable } from 'rxjs';

@Injectable()

export class AuthInterceptor implements HttpInterceptor {

intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

// Obtén el token JWT del localStorage

const token = localStorage.getItem('token');

// Si se encuentra un token, adjúntalo a la solicitud

if (token) {

request = request.clone({

setHeaders: {

Authorization: `Bearer ${token}`

}

});

}

return next.handle(request);

}

}

**3. Configuración del Módulo**

Registra el interceptor en el módulo de tu aplicación para que se aplique a todas las solicitudes HTTP salientes.

// app.module.ts

import { BrowserModule } from '@angular/platform-browser';

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

import { HttpClientModule, HTTP\_INTERCEPTORS } from '@angular/common/http';

import { AuthInterceptor } from './auth.interceptor';

@NgModule({

declarations: [...],

imports: [

BrowserModule,

HttpClientModule,

// Otros módulos y configuraciones

],

providers: [

{

provide: HTTP\_INTERCEPTORS,

useClass: AuthInterceptor,

multi: true

}

],

bootstrap: [...]

})

export class AppModule { }

**4. Protección de Endpoints**

Asegúrate de que los endpoints en tu backend estén protegidos y verifiquen la validez del token JWT enviado con las solicitudes. Si el token es válido, permite el acceso; de lo contrario, responde con un error de autorización.

**5. Cierre de Sesión**

Implementa una función de cierre de sesión en Angular que elimine el token JWT del localStorage cuando el usuario cierra sesión.

// auth.service.ts

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

@Injectable({

providedIn: 'root'

})

export class AuthService {

// ... Otras funciones y lógica de autenticación

logout() {

// Elimina el token JWT del localStorage al cerrar sesión

localStorage.removeItem('token');

}

}

**Conclusiones**

Con esta estrategia, hemos documentado cómo manejamos las sesiones de usuarios autenticados utilizando tokens JWT en nuestra aplicación Dory. Los usuarios autenticados pueden acceder a los endpoints protegidos, y el token JWT se adjunta automáticamente a las solicitudes gracias al interceptor. El cierre de sesión es seguro y elimina el token del localStorage, lo que garantiza la seguridad de las sesiones de los usuarios.