

# Ngx-permissions

## 0. Introducción

En este documento se explica algunos usos de la librería “Ngx-permissions” de Angular, haciendo las pruebas sobre el repositorio “kgomez95/Angular\_Perfiles\_Permisos” ([https://github.com/kgomez95/Angular\\_Perfiles\\_Permisos](https://github.com/kgomez95/Angular_Perfiles_Permisos)).

En este documento se explica un uso básico y sencillo para el acceso a nuestra aplicación, es por eso que se debe tomar como una guía de iniciación y no algo para obtener resultados inmediatos.

## 1. Instalación

Para instalar “Ngx-permissions” tenemos que abrir un terminal (o línea de comandos) y situarnos dentro del directorio donde tenemos nuestra aplicación. Acto seguido, debemos ejecutar el siguiente comando:

```
npm i ngx-permissions --save
```

**NOTA 1:** Desde la versión 5.0.0 de npm, la instrucción “--save” ya se ejecuta de forma automática, por lo que podrías omitirla si trabajas con dicha versión de npm o versiones superiores.

**NOTA 2:** Al ejecutar la instrucción “--save” cuando instalamos un paquete, éste también se nos añadirá de forma automática a nuestro listado de dependencias situado en el fichero “package.json” de nuestro proyecto. El añadir los paquetes que instalamos a nuestro listado de dependencias nos facilitará las instalaciones de las mismas, ya que con un simple comando npm instalará todas las dependencias de una sentada (más información en el siguiente enlace: <https://docs.npmjs.com/cli/v9/commands/npm-install>).

## 2. Configuración

Una vez instalada la librería “ngx-permissions”, debemos importar el módulo “NgxPermissionsModule” en todos nuestros módulos de la aplicación donde vayamos a utilizarlo. Es posible importarlo de forma manual en cada uno de tus módulos, pero no es recomendable hacerlo de esta forma, ya que tu aplicación irá creciendo cada vez más y al final, hacerlo de forma manual, implica que vas a tener código duplicado en muchos sitios.

Así pues, vamos a proceder a importar el módulo “NgxPermissionsModule” únicamente en nuestro fichero “shared.module.ts”. Tal y como su nombre indica, este fichero va a ser nuestro módulo compartido. Esto quiere decir que todos los módulos que exportemos en nuestro “SharedModule” vamos a poder utilizarlos en cualquier otra parte de la aplicación (siempre y cuando importemos el “SharedModule” en el módulo en cuestión). Para verlo de forma más clara, vamos a ver este proceso separado en pasos:

1. Nos dirigimos al fichero “shared.module.ts”, importamos la referencia al módulo “NgxPermissionsModule”, y éste lo añadimos en la sección “imports” (para utilizarlo en otros componentes que estén dentro de “shared”) y “exports” (para que, quien importe “SharedModule” pueda también utilizarlo).

```
Angular_Perfiles_Permisos / app / src / app / shared / shared.module.ts
kgomez95 - Creación del proyecto.

Code Blame 21 lines (18 loc) · 517 Bytes

1  import { NgModule } from '@angular/core';
2  import { CommonModule } from '@angular/common';
3  import { RouterModule } from '@angular/router';
4  import { ReactiveFormsModule, FormsModule } from '@angular/forms';
5
6  import { NgxPermissionsModule } from 'ngx-permissions';
7
8  const MODULES: any[] = [
9    CommonModule,
10   RouterModule,
11   ReactiveFormsModule,
12   FormsModule,
13   NgxPermissionsModule,
14 ];
15
16 @NgModule({
17   imports: [...MODULES],
18   exports: [...MODULES],
19   declarations: [],
20 })
21 export class SharedModule { }
```

2. Nos dirigimos al módulo principal de la aplicación (en mi caso “AppModule”), importamos la referencia al módulo “NgxPermissionsModule” y esta vez solamente lo añadiremos en la sección “imports”, a la misma vez que llamamos a su función “forRoot” (para más información de la función “forRoot” mirar el siguiente enlace: <https://angular.io/api/router/RouterModule>).

Code

main

Go to file

- app
  - .vscode
  - src
    - app
      - core
      - routes
      - shared
        - shared.module.ts
      - theme
        - app-routing.module.ts
        - app.component.html
        - app.component.scss
        - app.component.spec.ts
        - app.component.ts
        - app.module.ts** (1)
      - assets
        - favicon.ico
        - index.html
        - main.ts
        - styles.scss
        - .editorconfig

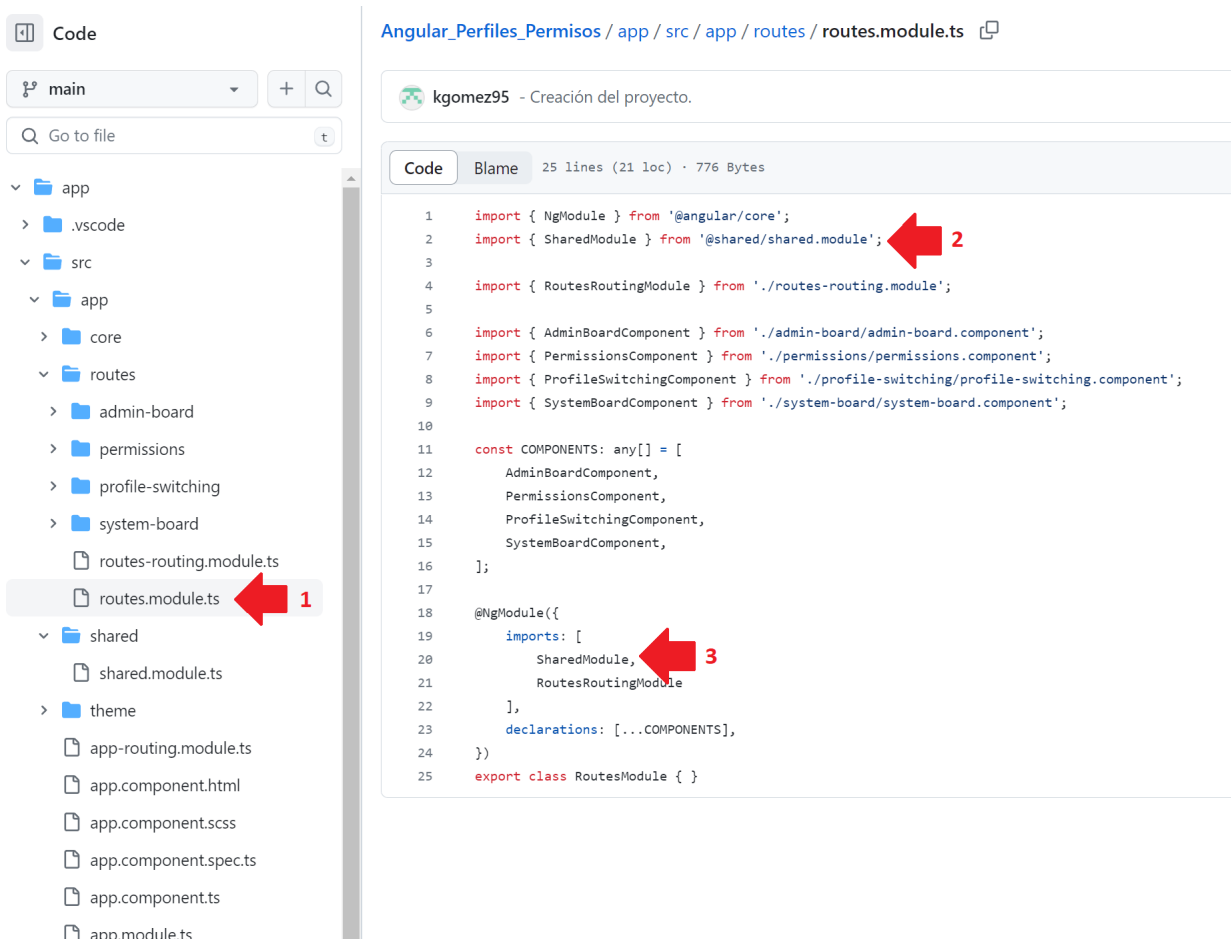
Angular\_Perfiles\_Permisos / app / src / app / app.module.ts

kgomez95 - Creación del proyecto.

Code Blame 31 lines (27 loc) · 785 Bytes

```
1 import { NgModule } from '@angular/core';
2 import { BrowserModule } from '@angular/platform-browser';
3 import { NgxPermissionsModule } from 'ngx-permissions'; (2)
4
5 //import { AppRoutingModule } from './app-routing.module';
6 import { AppComponent } from './app.component';
7
8 import { ThemeModule } from '@theme/theme.module';
9 import { SharedModule } from '@shared/shared.module';
10 import { RoutesModule } from './routes/routes.module';
11
12 import { appInitializerProviders } from '@core';
13
14 @NgModule({
15   declarations: [
16     AppComponent
17   ],
18   imports: [
19     BrowserModule,
20     //AppRoutingModule,
21     ThemeModule,
22     RoutesModule,
23     SharedModule,
24     NgxPermissionsModule.forRoot(), (3)
25   ],
26   providers: [
27     appInitializerProviders
28   ],
29   bootstrap: [AppComponent]
30 })
31 export class AppModule { }
```

- Ahora ya podemos hacer uso de la librería “ngx-permissions” allá donde importemos nuestro módulo “SharedModule”. En mi caso, las pruebas que voy a hacer se encuentran dentro de la carpeta “routes”, por lo que voy a tener que importar el “SharedModule” dentro de mi “RoutesModule”.



Una vez hayamos realizado los pasos 1 y 2, podemos replicar el paso 3 en cualquier otro módulo de nuestra aplicación. De esta manera, no solo tendremos a nuestra disposición la librería “ngx-permissions”, sino todos los módulos que hayamos exportado en el “SharedModule”.

### 3. Conceptos

Antes de comenzar a utilizar la librería, es necesario aclarar una serie de puntos para empezar a conocer la librería y evitar posibles comportamientos no deseados:

- La librería “ngx-permissions” nos ofrece dos servicios: “NgxPermissionsService” y “NgxRolesService”.
- El “NgxPermissionsService” nos permite administrar los permisos de la aplicación.
- El “NgxRolesService” nos permite administrar los roles de la aplicación (aunque también es posible añadir permisos a la vez que se añade un rol).
- No es estrictamente necesario el uso de los roles, siempre y cuando tengamos permisos. Es decir, yo puedo tener uno o varios permisos asignados sin la necesidad de tener un rol.
- Un rol puede tener asignados uno o varios permisos, pero es importante que dichos permisos existan en el “NgxPermissionsService”, ya que si algún rol tiene asignado algún permiso que no está registrado, este rol dejaría de ser “válido” (para más información, revisar las anotaciones de la función “addRemovePermission” del fichero “profile-switching.component.ts” que hay en el repositorio).

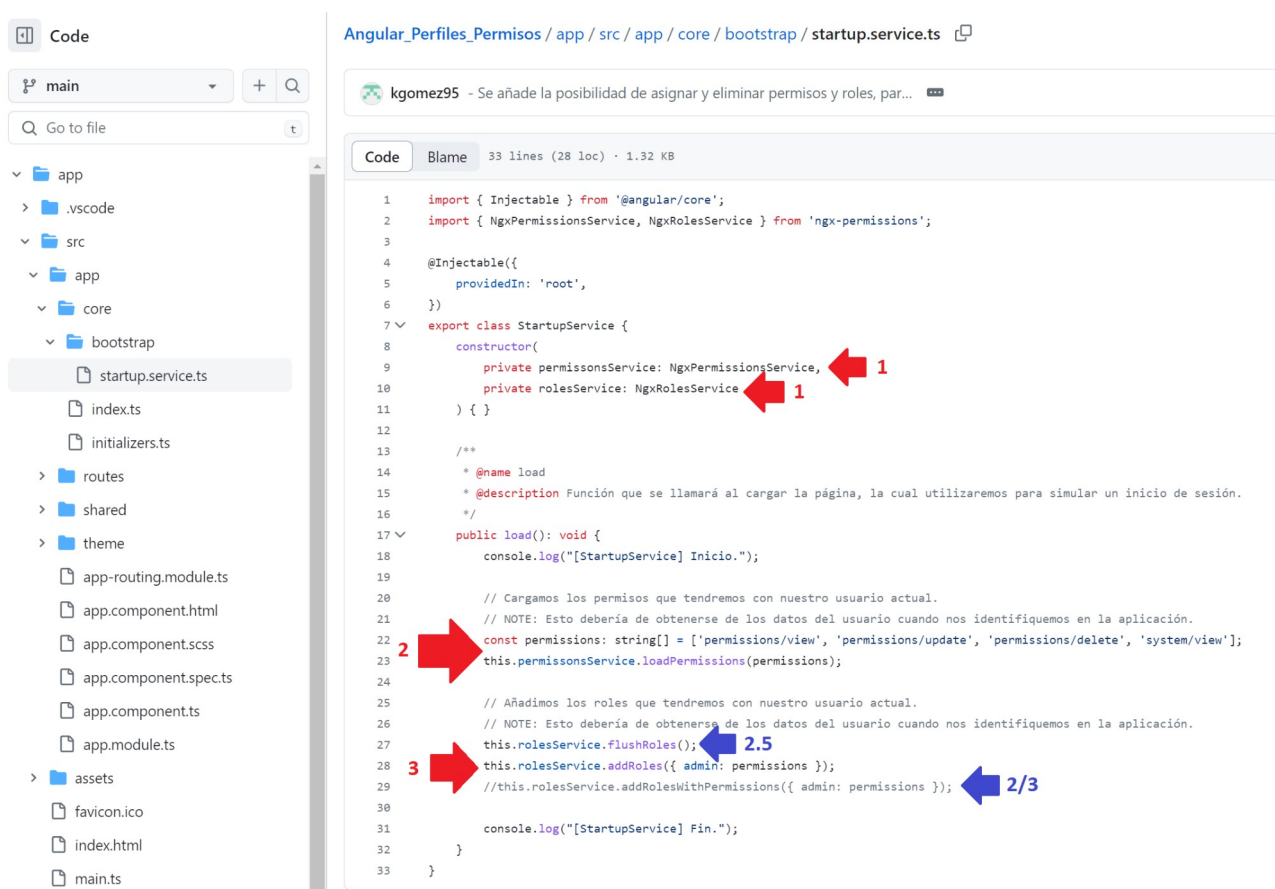
## 4. Uso

### 4.1. Creación de roles y permisos

En el repositorio “Angular\_Perfiles\_Permisos” podemos encontrar este apartado en los ficheros “startup.service.ts” (servicio que se llama al cargar la aplicación) y “profile-switching.component.ts” (página de prueba donde cambiar los roles y permisos en tiempo real).

En este ejemplo veremos que los roles y permisos los creamos directamente en el código. Esto es solamente para comprobar su uso. En un caso real, los roles y los permisos se deberían de recuperar de un servicio web, por ejemplo, cuando el usuario se identifique en la aplicación (al hacer *login*).

Para crear permisos utilizaremos el servicio “NgxPermissionsService” y para crear los roles utilizaremos el servicio “NgxRolesService” (tal y como podemos ver en el punto 1 de la siguiente imagen).



```
1 import { Injectable } from '@angular/core';
2 import { NgxPermissionsService, NgxRolesService } from 'ngx-permissions';
3
4 @Injectable({
5   providedIn: 'root',
6 })
7 export class StartupService {
8   constructor(
9     private permissionsService: NgxPermissionsService,
10    private rolesService: NgxRolesService
11  ) {}
12
13  /**
14   * @name load
15   * @description Función que se llamará al cargar la página, la cual utilizaremos para simular un inicio de sesión.
16   */
17  public load(): void {
18    console.log("[StartupService] Inicio.");
19
20    // Cargamos los permisos que tendremos con nuestro usuario actual.
21    // NOTE: Esto debería de obtenerse de los datos del usuario cuando nos identifiquemos en la aplicación.
22    const permissions: string[] = ['permissions/view', 'permissions/update', 'permissions/delete', 'system/view'];
23    this.permissionsService.loadPermissions(permissions);
24
25    // Añadimos los roles que tendremos con nuestro usuario actual.
26    // NOTE: Esto debería de obtenerse de los datos del usuario cuando nos identifiquemos en la aplicación.
27    this.rolesService.flushRoles();
28    this.rolesService.addRoles({ admin: permissions });
29    //this.rolesService.addRolesWithPermissions({ admin: permissions });
30
31    console.log("[StartupService] Fin.");
32  }
33 }
```

En el punto 2 podemos ver la creación de múltiples permisos con la función “loadPermissions” del NgxPermissionsService. También es posible crear los permisos de uno en uno con la función “addPermission” (en el fichero “profile-switching.component.ts” hay un ejemplo con esta función).

En el punto 3 podemos ver la creación del rol “admin” con la función “addRoles” del NgxRolesService, en donde le estamos asignando (en este caso) todos los permisos que hemos añadido en el punto 2. Al igual que con los permisos, también es posible añadir roles de uno en uno con la función “addRole”. Además, Estas dos funciones no solamente aceptar un *array* de *strings* para indicarle los permisos, sino que es posible indicarle una función de validación, la cual se evaluará cada vez que sea necesario para autorizar o denegar el acceso (en el fichero “profile-

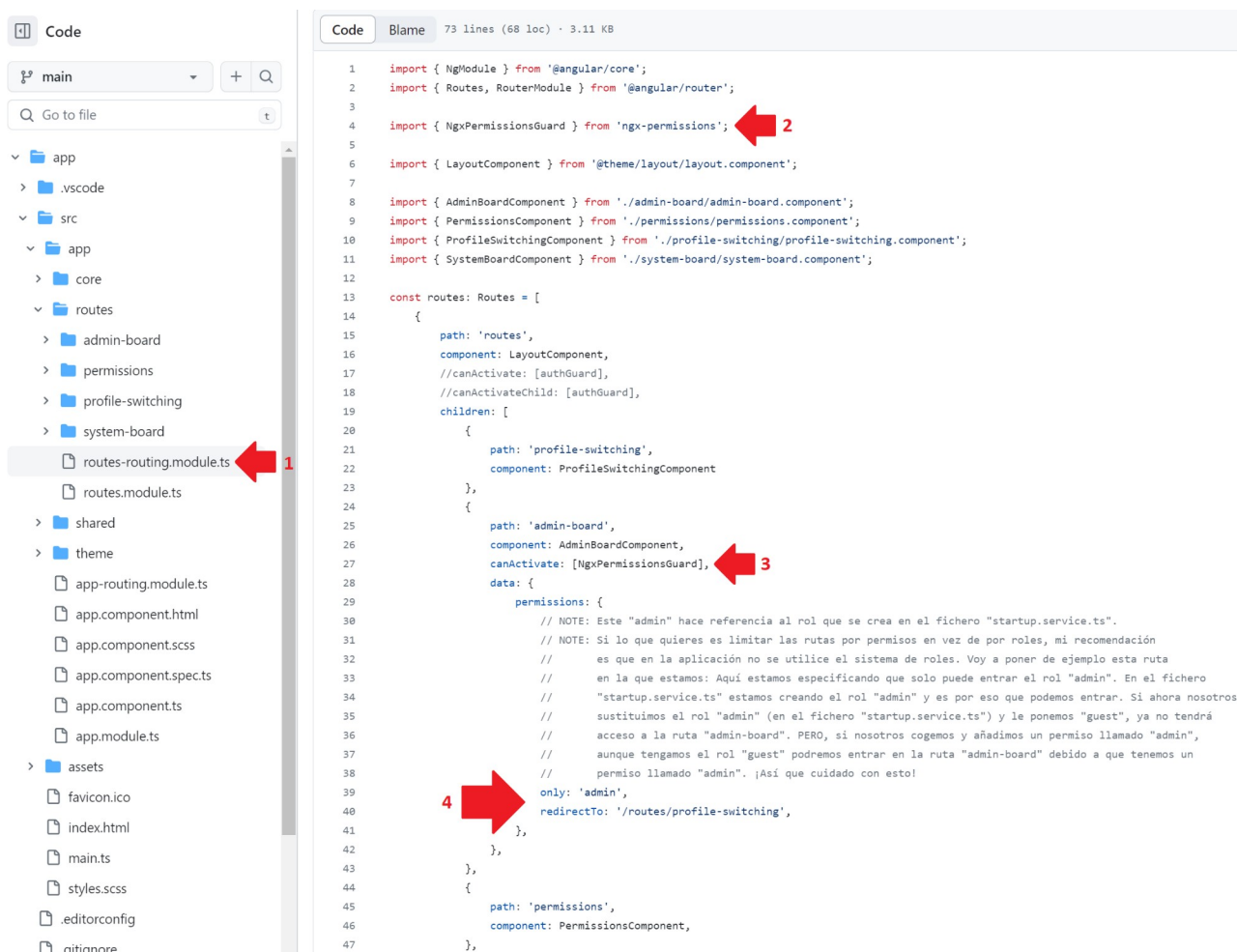
switching.component.ts”, concretamente en la función “addRemoveRole”, hay un ejemplo comentado con esta función de validación).

Otros puntos interesantes son los puntos 2.5 y 2/3 que he señalado en color azul en la imagen anterior. El punto 2.5 llama a la función “flushRoles”, la cual se encarga de eliminar todos los roles que había hasta ahora. La función “addRoleWithPermissions” del punto 2/3 hace lo mismo que el punto 2 y el punto 3, pero en una sola instrucción, es decir, que al mismo tiempo que crea el rol también crea los permisos y se los asigna al rol.

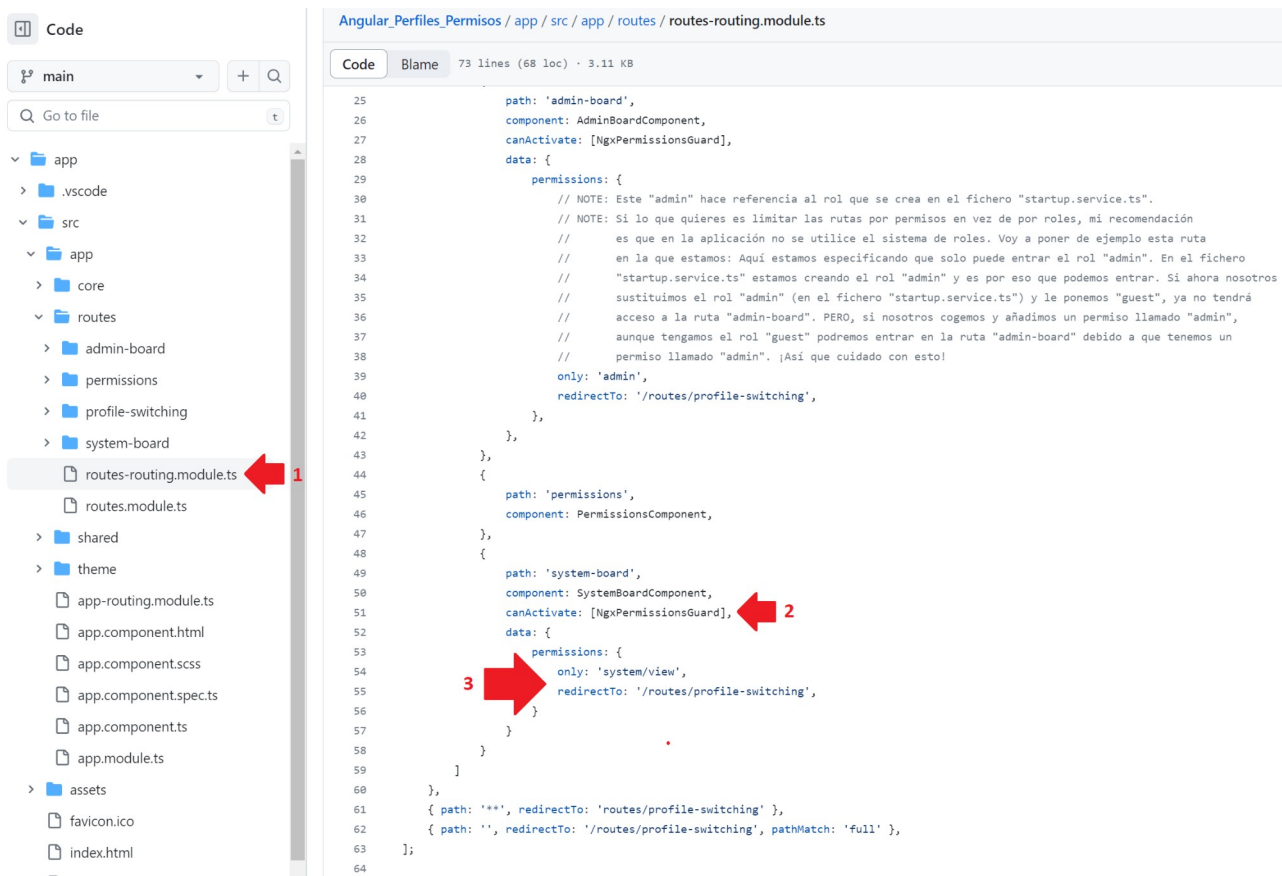
## 4.2. Roles y permisos sobre las rutas

Una vez creados los roles y los permisos llega la hora de sacarles su uso, y un uso común es el de restringir el acceso a las diferentes páginas de la aplicación. Para este ejemplo, vamos a partir del fichero “routes-routing.module.ts” que hay en el repositorio. En este fichero de rutas tenemos un ejemplo de una ruta restringida por rol y otra ruta restringida por un permiso.

En el punto 2 y 3 de la siguiente imagen, se importa y se hace uso del componente `NgxPermissionsGuard`, el encargado de comprobar que realmente tenemos acceso a dicha ruta (en este caso, lo tenemos en la ruta “admin-board”). Para indicarle al `NgxPermissionsGuard` los parámetros necesarios, debemos crear el objeto “data”, y dentro de este objeto crear otro objeto llamado “permissions”. Este objeto “permissions” puede coger varios campos, pero en este ejemplo de hace uso de los campos “only” y “redirectTo” (en el punto 4 de la imagen). El campo “only” sirve para especificarle que solamente tienen autorización los roles o permisos especificados en este campo (en la imagen se le asigna un *string*, pero también es posible asignarle un *array* de *strings*). Y el campo “redirectTo” sirve para redirigir a la ruta indicada en el caso de que no esté autorizado para entrar en la ruta (en este caso, si no tenemos acceso a la página “admin-board” nos redirige a la página “profile-switching”).



Como ya hemos comentado, también es posible autorizar el acceso a una ruta comprobando los permisos. Y la forma de hacerlo es exactamente a como se hace con los roles, pero en vez de especificar el rol se especifica el permiso (tal y como se aprecia en el punto 3 de la siguiente imagen).

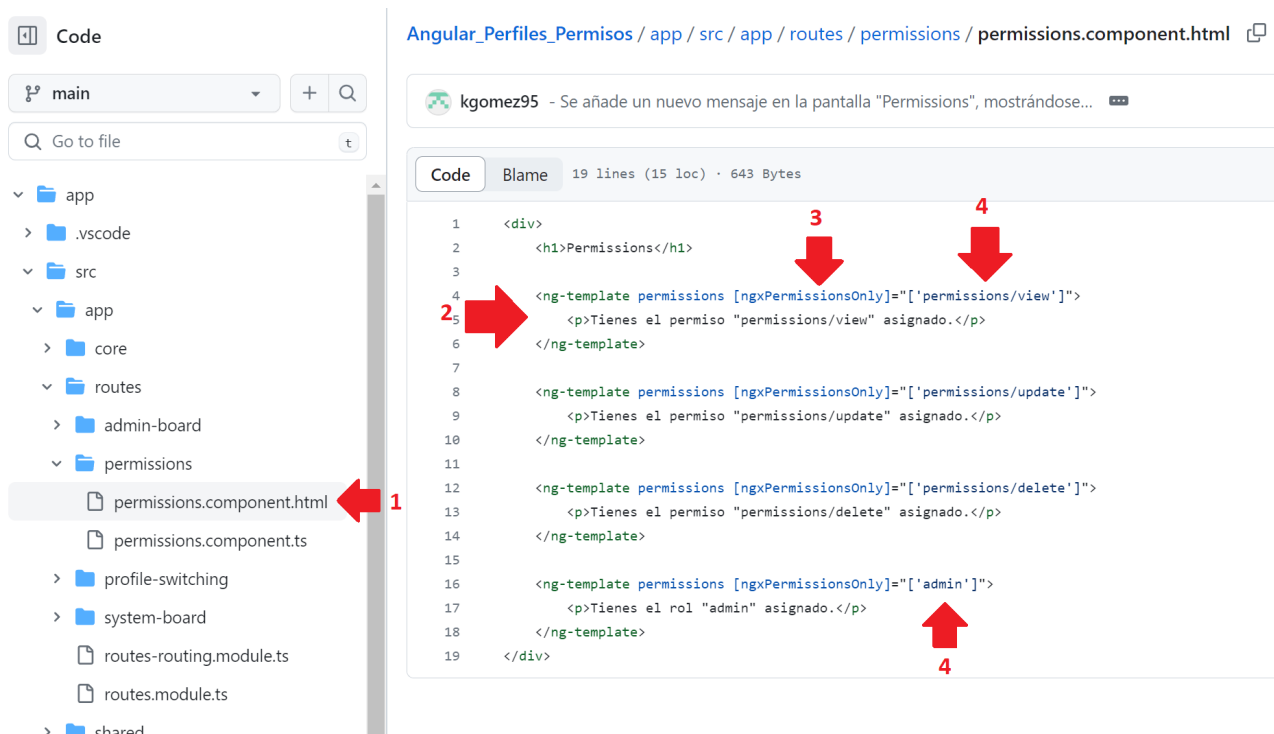


**NOTA:** En estos dos ejemplos que hemos visto, hemos utilizado el componente `NgxPermissionsGuard` que ya viene por defecto en la librería “`ngx-permissions`”, pero también es posible crear nosotros mismos un componente que se encargue de comprobar la autorización a las diferentes rutas de nuestra aplicación (más información en el siguiente enlace: <https://angular.io/api/router/CanActivate>).



### 4.3. Roles y permisos sobre los componentes

El uso de la librería “ngx-permissions” no solo abarca a la rutas, también podemos restringir el acceso a diferentes partes de un componente. En la siguiente imagen, veremos un ejemplo limitando el acceso por roles y perfiles en el fichero “permissions.component.html” que se encuentra en este repositorio.



Lo primero a destacar es la etiqueta “[ngxPermissionsOnly]” (punto 3), ya que con esta etiqueta vamos a poder limitar el acceso al elemento en cuestión para unos determinados roles y/o permisos. Así pues, esta etiqueta puede aceptar dos tipos de valores (punto 4): un *string* o un *array* de *strings*. Y por último, y no menos importante, esta etiqueta debe estar aplicada a un elemento “ng-template” (punto 2), ya que sino no funcionará y se producirá un error por consola al cargar el componente.

En este repositorio hemos probado únicamente la etiqueta “[ngxPermissionsOnly]”, pero hay más etiquetas disponibles para limitar el acceso a elementos del componente, así como la etiqueta “[ngxPermissionsExcept]” (que funciona de manera similar a la que hemos visto, pero esta autoriza el acceso si no se tiene el rol o el permiso asignado) y las etiquetas “(permissionsAuthorized)” y “(permissionsUnauthorized)” (las cuales se les debe indicar una función en vez de un *string* o *array* de *strings*, ya que la comprobación de acceso la tenemos que programar nosotros mismos).