UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Programador Web Avanzado

Clase N° 11: Angular Guard - Interceptors

Profesor: Ing. Leandro Rodolfo Gil Carrano
Email: leangilutn@gmail.com

UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Interceptors

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Interceptors

Un interceptor inspecciona/modifica las peticiones, o sea, lo que va de tu aplicación al servidor y también lo que viene del servidor a tu aplicación. En pocas palabras, la petición y respuesta.

Podemos definir un interceptor para enviar el token de autenticación en todas las solicitudes de nuestra aplicación hacia el servidor en express

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Interceptors

Para crear nuestro interceptor, tenemos que generar un servicio usando el CLI de angular

```
C:\sites\angular\pwafront>ng generate service interceptors
```

Interceptors

una vez creado el servicio implementa la interfaz **HttpInterceptor** y otras clases extras que se encuentran en '@angular/common/http'. A continuación implementa el método que está definido en la interfaz, llamado **intercept.** El servicio quedará de la siguiente manera:

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Interceptors

```
import { Injectable } from '@angular/core';
import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class AuthInterceptorService implements HttpInterceptor {

  constructor() {}

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

  }

}
```

Interceptors

Una vez hecho todo eso implementaremos la parte en que nuestro interceptor agrega la cabecera **‘x-access-token’** con el token que vayamos a usar, como modo de ejemplo, imaginemos que tenemos el token almacenado en *localStorage* con la clave **‘auth-token’** y para obtener el token tendremos que hacer lo siguiente: localStorage.getItem(‘token’).

Interceptors

declaramos una constante que guarde el token obteniendolo del localStorage. El token puede o no existir en nuestro localStorage, y para no estar enviando la cabecera **‘x-access-token’** vacía, condicionamos si existe algún token, posteriormente si existe un token, modificamos el *request* agregándole la cabecera, utilizando el método **clone** que viene en nuestro *request* de tipo **HttpRequest** y pasándole la cabecera con el token. una vez modificado el *request lo* retornamos de nuestro método.

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Interceptors

```
if (token) {
    request = request.clone({ headers: request.headers.set('x-access-token', token) });
}

if (!request.headers.has('Content-Type')) {
    request = request.clone({ headers: request.headers.set('Content-Type', 'application/json') });
}
request = request.clone({ headers: request.headers.set('Accept', 'application/json') });
```

Interceptors

Ahora bien, hay que recordar que los **interceptors** se ejecutan en cada petición que se realiza al servidor, siempre y cuando sean registrados. Para registrar un interceptor se tiene que proveer en el array de **providers: []** en nuestro módulo raíz, por lo general *AppModule.* Importamos **HTTP_INTERCEPTORS** y **HttpClieⁿgModule,** luego proveémos nuestro *interceptor*

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Interceptors

```
import { InterceptorService } from './interceptors.service';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import {MatButtonModule, MatCheckboxModule} from '@angular/material';
@NgModule({
  declarations: [
    AppComponent,
    RegistroComponent,
    LoginComponent,
    ProductosComponent,
    ProductosDetalleComponent,
    ProductoComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    ReactiveFormsModule,
    HttpClientModule,
    BrowserAnimationsModule,
    MatButtonModule, MatCheckboxModule
  ],
  providers: [
    { provide: HTTP_INTERCEPTORS, useClass: InterceptorService, multi: true }
  ],
```

Interceptors

Nuestro provider utiliza la propiedad *multi: true,* esto permitirá agregar más interceptors si lo requerímos y no sobre escribir nuestro *interceptor.* En otras palabras, crean un array con el mismo *provider token (***HTTP_INTERCEPTORS**). En este punto, nuestro interceptor modificará cada petición que se haga al servidor siempre y cuando exista un token.

Otra cosa que podríamos realizar en nuestro interceptor sería redirigir al usuario a la página de *login* cuando el token haya expirado o no tenga un token válido por ejemplo.

Interceptors

Modificaremos nuestro interceptor un poco, importando el operador **catchError** de *rxjs,* que recibe un método que va a revisar el status de la respuesta que el servidor nos devuelve. Cuando detectemos que el *status* es 401, hacemos la redirección a la vista de *login,* de igual manera el servicio **Router** e injectarlo en el constructor para utilizarlo y hacer la redirección.

UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Guards

Guards

Hay veces que queremos que determinadas áreas de nuestra aplicación web estén protegidas y solo puedan ser accedidas si el usuario ésta logueado (un panel de control por ejemplo) o incluso que solo puedan ser accedidas por determinados tipos de usuarios. Para conseguir esto con Angular se usan los guards. Los guards pueden ser extensibles para que permitan acceder bajo las condiciones que nosotros queramos, podemos incluso hacer peticiones a un backend antes de que el usuario entre en la página.

Guards

Dentro de los guards hay 4 tipos principales:

- CanActivate: Mira si el usuario puede acceder a una página determinada.
- CanActivateChild: Mira si el usuario puede acceder a las páginas hijas de una determinada ruta.
- CanDeactivate: Mira si el usuario puede salir de una página, es decir, podemos hacer que aparezca un mensaje, por ejemplo, de comfirmación, si el usuario tiene cambios sin guardar.
- CanLoad: Sirve para evitar que la aplicación cargue los módulos perezosamente si el usuario no está autorizado a hacerlo.

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Guards

Se debe crear un servicio, por ejemplo
Ng generate guard auth

Los guards devuelven **true** o **false** para permitir el paso o no de un usuario a la ruta. También pueden devolver un Observale o una Promise si el guard no puede responder inmediatamente y tiene que esperar.

Guards

```
import { Injectable } from '@angular/core';
import { CanActivate } from '@angular/router';
import { AuthenticationService } from './authentication.service';

@Injectable({
  providedIn: 'root'
})
export class AuthGuard implements CanActivate {

  constructor(public auth: AuthenticationService) {}

  canActivate(): boolean {
    return this.auth.isAuthenticated();
  }
}
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Guards

Authentication.services.ts

```
isAuthenticated() {
  return this.authenticationState.value;
}
```

*Retorna el valor del **BehaviorSubject,** básicamente si esta o no autenticado*

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Guards

App-routing.module.ts

```
import { AuthGuard } from './auth.guard';

const routes: Routes = [{
  path:'registro',
  component: RegistroComponent
},
{
  path:'login',
  component:LoginComponent
},
{
  path:'productos',
  canActivate:[AuthGuard],
  component:ProductosComponent
},
```

EL componente (pagina) producto solo podrá ser accedida cuando el guard retorne true (es decir el usuario este autenticado)

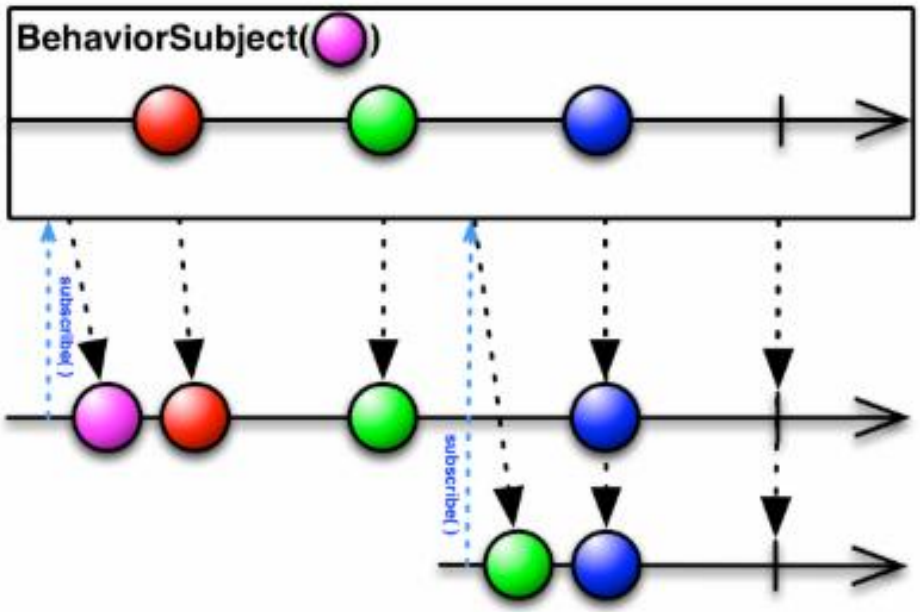
UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

BehaviorSubject

BehaviorSubject

Behaviour Subject nos permite utilizar una característica realmente útil y que es la de poder "recodar¨ el último valor emitido por el Observable a todas las nuevas subscripciones, al margen del momento temporal en que éstas se establezcan, actuando como un mecanismo de "sincronización" entre todas las subscripciones que resulta realmente útil:



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

BehaviorSubject

```
const subject = new Rx.BehaviorSubject();

// Subscripción 1
subject.subscribe(data => {
    console.log('Subscripción 1:', data);
});

subject.next(1);
subject.next(2);

// Subscripción 2
subject.subscribe(data => {
    console.log('Subscripción 2:', data);
});

subject.next(3);
```

Como podemos ver en la salida, se estable una subscripción, se emiten los valores 1 y 2, y cuando la segunda subscripción se establece, ésta recibe el último valor emitido por el Observable, es decir el 2.

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

BehaviorSubject

https://pablomagaz.com/blog/rxjs-subjects-que-son-como-funcionan