ISPC
INSTITUTO SUPERIOR
POLITÉCNICO CÓRDOBA

Seguridad en aplicaciones Web

Autenticación en Angular

Al momento de realizar la autenticación, el usuario ingresa sus credenciales (mail y contraseña), los mismos son enviados al backend y una vez verificada la existencia del mismo, se anexa un token para seguridad, todo el conjunto de datos son enviados nuevamente al Frontend, y dependiendo de estos datos entregados se mostrará el componente correspondiente, como por ejemplo:

- ➔ Un tablero de control que permite realizar actividades diferentes según el rol del usuario.
- ➔ Un mensaje de bienvenida para diferentes roles.
- ➔ etc.

En nuestro ejemplo, el usuario podrá acceder al home, el cual contiene el acceso a últimos movimientos, operaciones, transacciones, etc.

Para generar la autenticación del usuario en nuestra aplicación utilizaremos una librería llamada JWT (JSON WEB TOKEN) en el backend.

Nota: JWT se deberá configurar previamente todo lo relacionado a ello (ver módulo 6,seguridad informática).

Crear el servicios para autenticación

1. Dentro de la carpeta de servicio crear una carpeta contenedora de todos los servicios necesarios para realizar la autenticación de los usuarios. Nosotros la llamaremos “Auth”

Nota: Recuerda que services es la carpeta donde le especificamos a AngularCLi que cree el archivo.

Autenticación en Angular

2. Generar el servicio para la lógica de autenticación ***aut.service.ts***
3. Editar el servicio a fin de:
   a. Importar las clases Observable de rxjs, map de rxjs y servicio usuario generado con anterioridad

```
import { Injectable } from '@angular/core';
import { BehaviorSubject, Observable } from 'rxjs';

import { map } from 'rxjs/operators';
```

   b. Referenciar la misma API utilizada en el registro de usuario.

```
url="https://reqres.in/api/login";
```

   c. Declarar el ***BehaviorSubject*** y el ***observable*** como propiedades:

```
currentUserSubject: BehaviorSubject<Usuario>;

 currentUser: Observable<Usuario>;
```

   d. Inyectar el HttpClient en el constructor e inicializamos las propiedades creadas previamente:

```
constructor(private http:HttpClient) {

    console.log("Servicio de Atuenticación está corriendo");

    this.currentUserSubject = new
BehaviorSubject<Usuario>(JSON.parse(localStorage.getItem('currentUser') || '{}'));

    this.currentUser = this.currentUserSubject.asObservable();

   }
```

   e. Agregar la lógica de inicio de sesión. Método **login**.

Método al que luego el componente (ts) podrá subscribirse. El mismo, recibe un objeto Usuario como parámetro y devuelve un objeto Usuario (modificado con el id y demás datos que puedan obtenerse del servidor).

```
login(usuario: Usuario): Observable<any> {

  return this.http.post<any>(this.url, usuario)

    .pipe(map(data => {
```

```
      localStorage.setItem('currentUser', JSON.stringify(data ));

      this.currentUserSubject.next(data);

      this.loggedIn.next(true);

      return data;

    }));
```

data. Puedes ver que contiene aquí: https://reqres.in/api/users/

Autenticación en Angular

a. Agregar la lógica, para realizar el cierre de sesión

```
logout(): void{
    localStorage.removeItem('currentUser');
    this.loggedIn.next(false);
 }
```

b. Agregar propiedades para acceder a los datos del Usuario autenticado

```
get usuarioAutenticado(): Usuario {
    return this.currentUserSubject.value;
  }
  get estaAutenticado(): Observable<boolean> {
    return this.loggedIn.asObservable();
  }
```

4. En el controlador, archivo **inicia-sesion.component.ts**, editar a fin de:

a. Importar el servicio recién creado

```
import { Router } from '@angular/router';

import {AuthService} from 'src/app/services/auth/auth.service'
```

b. Inyectar el servicio en el constructor

```
constructor(private formBuilder: FormBuilder,

    private authService: AuthService,
    private router: Router) {
    this.form= this.formBuilder.group(
      {
        password:['',[Validators.required, Validators.minLength(8)]],
        mail:['', [Validators.required, Validators.email]]
      }
```

Autenticación en Angular

c. Crear y editar el evento onSubmit (onEnviar) del formulario a fin de hacer la petición al modelo para que éste haga la petición.

```
onEnviar(event: Event, usuario:Usuario): void {
    event.preventDefault;
    this.authService.login(this.usuario)
      .subscribe(
        data => {
        console.log("DATA"+ JSON.stringify( data));
        this.router.navigate(['/home/movimientos']);
      },
        error => {
         this.error = error;
        }
      );
  }
```

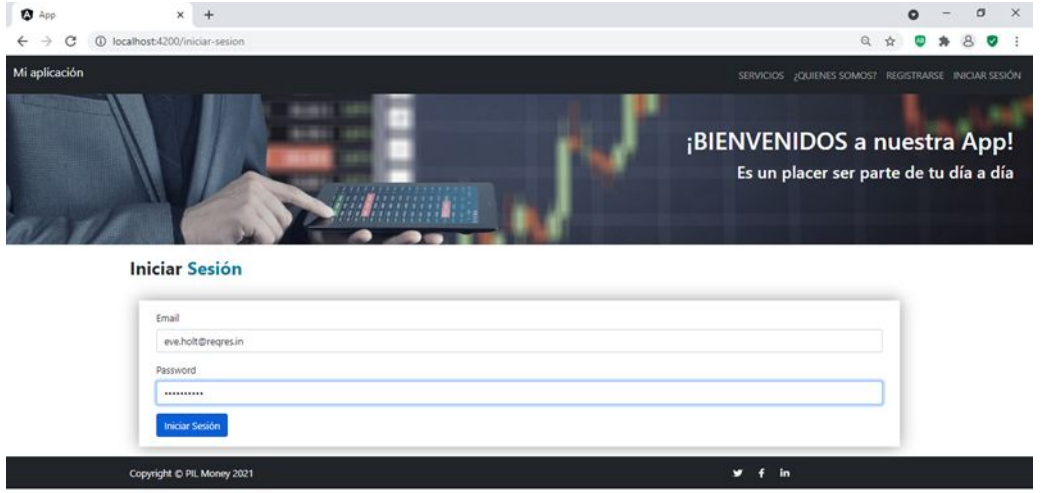
Controlador solicita al Modelo la autenticación de Usuario por medio de la ejecución del método

Redirecciona a la ruta de de movimientos al usuario. Previamente se debe importar router e inyectar en el

d. Editar la vista (inicio-sesion.component.html) a fin de agregar el evento.

```
  <form [formGroup]="form" (ngSubmit)="onEnviar($event, usuario)">
…
</form>
```

e. Finalmente, ejecutar ng-serve para evaluar el comportamiento. Observaremos que si iniciamos sesión con los datos: email: eve.holt@reqres.in y password: cityslicka, la aplicación auténtica y redirecciona al home/movimientos:

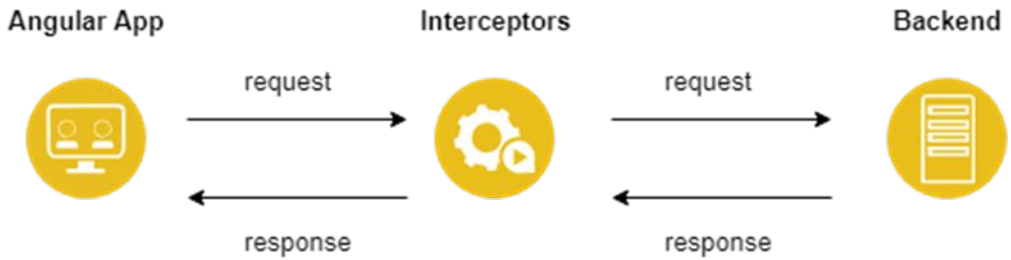


Nota: El usuario y password son los requeridos por la API de prueba que estamos usando: https://reqres.in/

Crear Interceptors

Los interceptors proveen un mecanismo para interceptar y/o mutar las solicitudes y respuestas http. Por ende, los interceptors son capaces de intervenir las solicitudes de entrada y de salida de tu app al servidor y viceversa.

Nota: No debe confundirse con los Guards. Los interceptors modifican las peticiones (enpoints, servicios, o como lo quieras llamar) y los guards controlan las rutas de navegación entre páginas de la aplicación web, permitiendo o denegando el acceso por ejemplo (https://medium.com/@insomniocode/angular-autenticaci%C3%B3n-usando-interceptors-a26c167270f4).



En nuestro caso, utilizaremos los interceptos de Angular para agregar el token en la cabecera 'Authorization' y de esta manera incrementar la seguridad de nuestra aplicación.

1. En el directorio ***auth*** generar el servicio ***interceptor.ts.*** Luego, editar a fin de:

   a. Importar el servicio recién creado, las clases e interfaces necesarias:

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

Crear Interceptors

Un interceptor permite que puedas inspeccionar y/o modificar todas las solicitudes HTTP de un HttpClient. Esto significa que antes de realizar cualquier llamado al servidor, podes editar su contenido y también la respuesta.
Fuente:https://lautarocarro.blog/usando-http-interceptors-en-blazor/

1. Generar el servicio para la lógica de ***error.interceptor.ts.***
2. Editar el servicio a fin de:
   a. Importar las clases Observable de rxjs, map de rxjs y servicio usuario generado con anterioridad.

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

import { Observable, throwError } from 'rxjs';

import { catchError } from 'rxjs/operators';
```

Es un operador de RXJS que se usa para crear un observable que emite una notificación

El catch es un operador RxJS detecta el error generado por un

b. Decorar la clase como inyectable e Implementar la interfaz HttpInterceptor:

```
@Injectable({
  providedIn: 'root'
})

export class JwtInterceptor implements HttpInterceptor {
...
```

c. En el constructor, inyectar el servicio:

```
    constructor(

        private authService: AuthService
      ) { }
```

d. Agregar la lógica que permite manipular el token que provee el API.

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

    const currentUser = this.authService.usuarioAutenticado;
    if (currentUser && currentUser.token) {
      req = req.clone({
        setHeaders: {
          Authorization: `Bearer ${currentUser.token}`
        }
      });

    }
    console.log("INTERCEPTOR: " + currentUser.token);
    return next.handle(req);
  }
```

Crear Interceptors

b. Agregar la lógica que mostrará una notificación de error:

```
@Injectable({

  providedIn: 'root'

})

export class ErrorInterceptor implements HttpInterceptor {

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

        return next.handle(req).pipe(catchError(err => {

          if (err.status === 401) {

            location.reload();

          }

          const error = err.error.message || err.statusText;

          return throwError(error);

        }));

      }
```

Devolverá una notificación de

Nota: Observa que la barra de navegación también ha cambiado en función de la autenticación. Puedes observar el fuente en el componente nav.

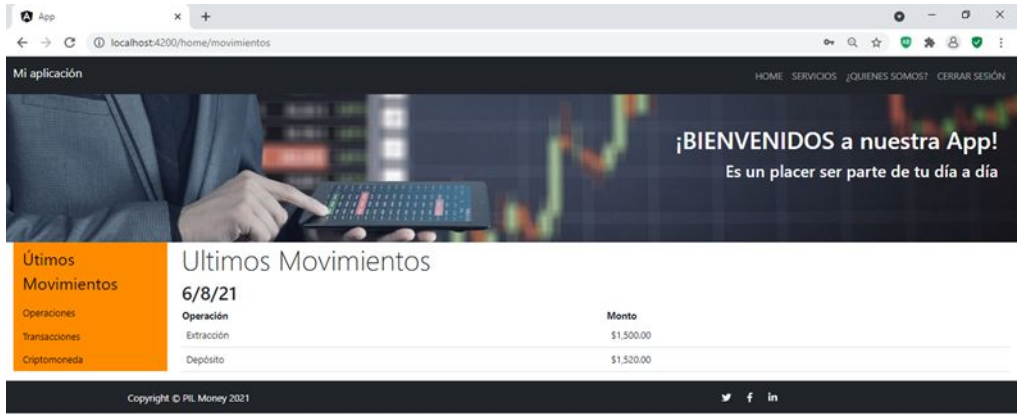
4. En el archivo app.module.ts
   a. Importar el archivo recién creado ***Interceptor*** y los servicios necesarios, ***interceptor*** y ***error.interceptor***.

```
import { HttpClientModule, HTTP_INTERCEPTORS } from '@angular/common/http';
import { JwtInterceptor } from './services/auth/interceptor';
import { ErrorInterceptor } from './services/auth/error.interceptor';
```

   b. En los providers agregar los interceptors

```
providers: [UsuarioService,
    { provide: HTTP_INTERCEPTORS, useClass: JwtInterceptor, multi: true },
    { provide: HTTP_INTERCEPTORS, useClass: ErrorInterceptor, multi: true },
  ],
```

   c. Finalmente, ejecutar ng-serve para evaluar el comportamiento.



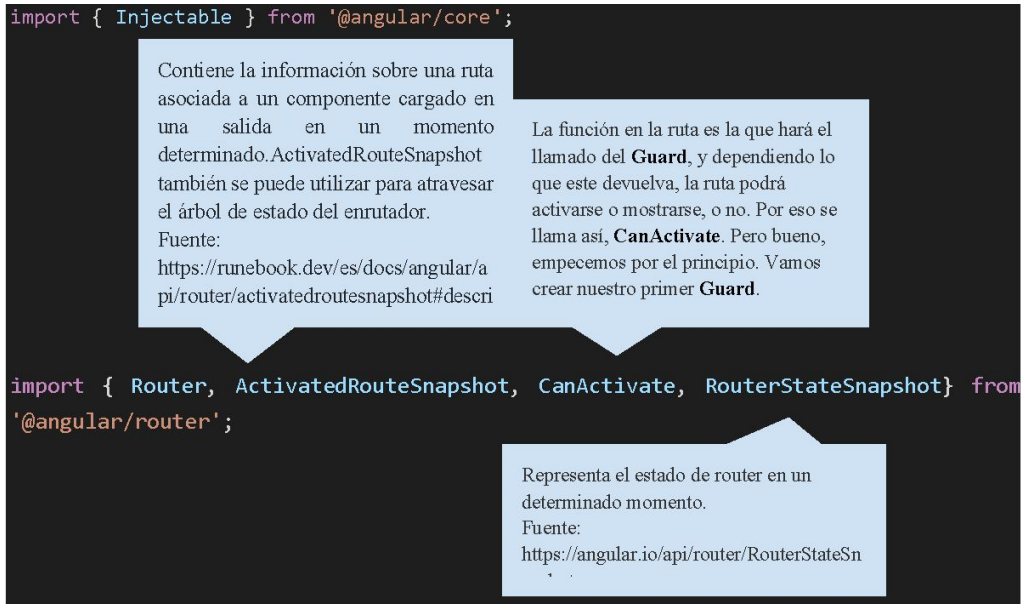
Nota: Iniciamos sesión con los datos: email: eve.holt@regres.in y password: cityslicka.

Crear Guards

Los Guards son utilizados, cuando requerimos que algunas url (o áreas) de la aplicación estén protegidas de forma que solo puedan ser vistas o accedidas cuando el usuario está autenticado o bien posea un rol específico. Caso contrario, el usuario no tendrá acceso a esta url o área de la aplicación.

1. En el directorio ***auth*** generar el servicio ***auth.guard.ts***. Luego, editar a fin de:

   a. Importar el servicio previamente creado (***auth.guard.ts***), las clases e interfaces como sigue:




```
import { AuthService } from './auth.service';
import { Observable } from 'rxjs';
import { map, take } from 'rxjs/operators';
```

   b. Decorar la clase como inyectable e Implementar la interface CanActivate

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

   c. Inyectar en el constructor el servicio

```
   constructor(

      private authService: AuthService
    ) { }
```

   d. Editar la lógica de la función de canActivate:

```
  canActivate(
        route: ActivatedRouteSnapshot,
        state: RouterStateSnapshot): Observable <boolean> {
         return this.authService.estaAutenticado.pipe(take (1),
         map((isLogged:boolean)=>isLogged));
      }
```

Crear Guards

e. En el archivo app-routing.module.ts, editar las rutas que requieren autenticación como sigue:

```
const routes: Routes = [
  {path: 'iniciar-sesion', component: IniciarSesionComponent},

  {path:'home', component: HomeComponent, canActivate: [AuthGuard],
  children:[
    {path:'operaciones', component: OperacionesComponent},
    {path:'transacciones', component: TransaccionesComponent},
    {path:'criptomoneda', component: CriptomonedaComponent},
    {path:'movimientos', component: MovimientosComponent},
  ]},
  {path:'servicios', component: ServiciosComponent},
  {path: 'quienes-somos', component: QuienesSomosComponent},
  {path: 'quienes-somos/:id', component: IntegranteComponent},
  {path: 'registro', component:RegistroComponent},
  {path: '', redirectTo: '/servicios', pathMatch: 'full'},
...
```

De esta manera, la aplicación no permitirá a los usuarios acceder al home hasta tanto no se hayan autenticado.

ISPC
INSTITUTO SUPERIOR
POLITÉCNICO CÓRDOBA

Referencias

https://wuschools.com/what-is-mvc-and-understanding-the-mvc-pattern-in-angular/

https://scotch.io/tutorials/mvc-in-an-angular-world

https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

https://codingpotions.com/angular-servicios-llamadas-http

https://codingpotions.com/angular-seguridad

https://angular.io/api/router/RouterStateSnapshot

https://lautarocarro.blog/usando-http-interceptors-en-blazor/