

Material de estudio OBLIGATORIO EJE 2 Angular - Autenticación

Sitio: [Instituto Superior Politécnico Córdoba](#)
Curso: Programador Web - TSDWAD - 2022
Libro: Material de estudio OBLIGATORIO EJE 2 Angular -
Autenticación

Imprimido por: Ezequiel Maximiliano GIAMPAOLI
Día: domingo, 11 junio 2023, 11:18 PM

Tabla de contenidos

1. Registro de Usuarios

1.1. Servicio Usuarios (Modelo)

1.2. Registro de Usuario (Vista y Controlador)

2. Autenticación en Angular

2.1. Servicios de Autenticación

2.2. Guards

2.3. Interceptors

3. Referencias

1. Registro de Usuarios

Continuando con la aplicación que venimos desarrollando, y pensando en el patrón MVC desde la perspectiva de Angular, veamos como hacer un registro de usuario.

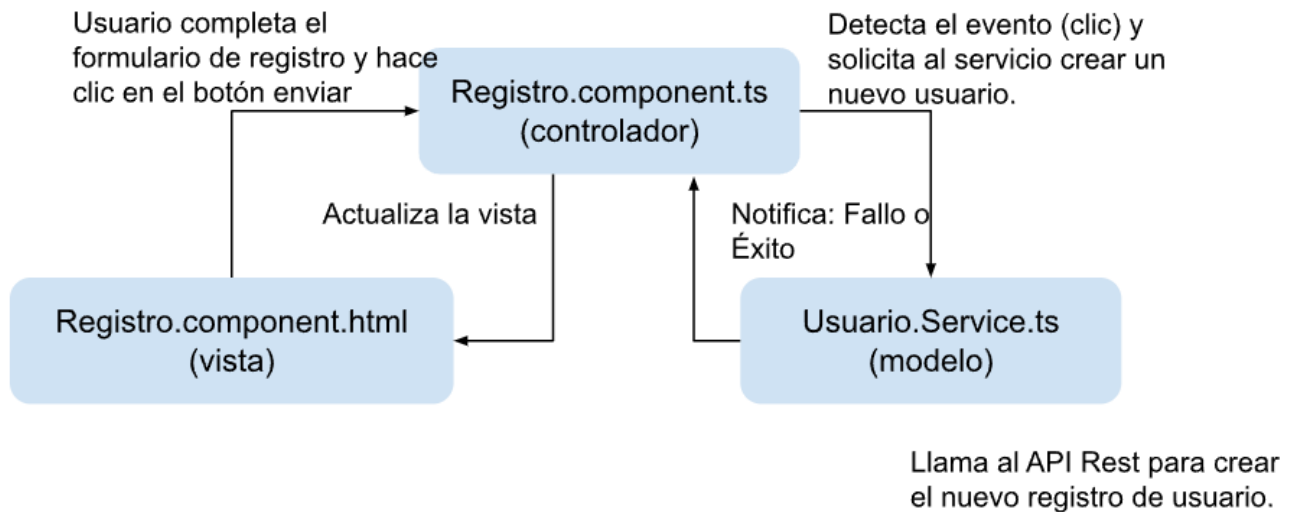


Figura: Funcionalidad "Crear Registro de Usuario" desde la perspectiva MVC de Angular.

Como pudimos observar en los gráficos de MVC, el modelo contiene los datos y los mecanismos necesarios para manipularlos. Es entonces el modelo, el punto final del frontend dado que, es a partir de ahí debe conectarse con un servicio web o API Rest para acceder a los datos, es decir con el backend. Los modelos de datos pueden ser simples como por ejemplo para trabajar un CRUD (Create, Read, Update y Delete) o mucho más complejos.

En Angular, como mencionamos previamente, el modelo está codificado en servicios.

Nota: A fin de poder hacer las pruebas desde el frontend se utilizará el API Rest disponible en <https://reqres.in/>

1.1. Servicio Usuarios (Modelo)

Crear el Servicio Usuarios (Modelo)

En nuestro ejemplo, el servicio UsuarioService es quién nos permitirá acceder y manipular los datos. Para ello,

1. Ir a la consola DOS o "Símbolo del Sistema" del sistema operativo o Terminal de VSCode.
2. Ejecutar el comando: **ng generate service services/usuario**

o su abreviado: **ng g s services/usuario**

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

Una vez ejecutado el comando, AngularCLI crea un archivo generará el archivo usuario.service.ts

3. Editar el servicio a fin de:
 1. Importar las clases HttpClient de @angular/common/http y Observable de rxjs.
 2. Editar el archivo usuario.service.ts a fin de hacer la petición POST.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

export class Usuario
{
  nombre:string="";
  apellido:string="";
  dni:string="";|
  fechaNacimiento:string="";
  password:string="";
  email:string="";
  id:number=0;
  //A modo de ejemplo se deja así pero lo ideal es crear propiedades para acceder a
los atributos.
}
```

```
@Injectable({
  providedIn: 'root'
})
export class UsuarioService {

  url="https://reqres.in/api/users/1";

  constructor(private http:HttpClient) {
    console.log("Servicio Usuarios está corriendo");
  }

  onCrearUsuario(usuario:Usuario):Observable<Usuario>{
    return this.http.post<Usuario>(this.url, usuario);
  }
}
```

API Rest de
ejemplo:
<https://reqres.in/>

Método al que luego el componente (ts) podrá suscribirse. 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).

Observa que, además de importar **HttpClient**, también hemos importado la clase **Observable**. Esto es porque después necesitamos acceder a los datos desde el componente.

Además, y tal como se definió previamente, se puede observar que el servicio (modelo) contiene datos (objeto usuario) y el mecanismo para acceder y manipular dichos datos (método onCrearUsuario).

Nota: No olvidar importar el módulo HttpClientModule, en la sección imports de el app.module.ts

1.2. Registro de Usuario (Vista y Controlador)

Crear el componente Registro de Usuario (Vista y Controlador)

El siguiente paso es crear un componente que hará de Vista (html) y Controlador (ts).

1. Ir a la consola DOS o "Símbolo del Sistema" del sistema operativo o Terminal de VSCode.
2. Ejecutar el comando: `ng generate c pages/registro`
3. o su abreviado: `ng g s pages/registro`

Nota: Recuerda que estos archivos ya los hemos creado previamente (en el módulo 5: Angular)

4. En la vista, archivo `registro.component.html`, editar a fin de crear un formulario de registro:

```
<div class="container">
  <div class="row">
    <div class="col-md-12 registro-form-header">
      <p class="registro-form-font-header">Registro<span> Usuario</span></p>
    </div>
  </div>
  <div class="row m-3">
    <div class="col-md-12 registro-form">
      <form [formGroup]="form" class="form" (ngSubmit)="onEnviar($event, usuario)" >
        <div class="m-3">
          <label for="nombre" class="form-label">Nombre</label>
          <input type="text" id="nombre" class="form-control"
formControlName="nombre" [(ngModel)]="usuario.nombre" [class.border-danger]="NombreValid" >
          <div *ngIf="Nombre?.errors && Nombre?.touched">
            <p *ngIf="Nombre?.hasError('required')" class="text-danger">
              El nombre es requerido.
            </p>
          </div>
        </div>
        <div class="m-3">
          <label for="apellido" class="form-label">Apellido</label>
        </div>
      </form>
    </div>
  </div>
</div>
```

```

        <input type="text" id="apellido" class="form-control"
formControlName="apellido" [(ngModel)]="usuario.apellido"
[class.border-danger]="ApellidoValid">
        <div *ngIf="Apellido?.errors && Apellido?.touched">
            <p *ngIf="Apellido?.hasError('required')" class="text-danger">
                El apellido es requerido.
            </p>
        </div>
    </div>
    <div class="m-3">
        <label for="dni" class="form-label">Dni</label>
        <input type="number" id="dni" class="form-control" formControlName="dni"
[(ngModel)]="usuario.dni" [class.border-danger]="DniValid">
        <div *ngIf="Dni?.errors && Dni?.touched">
            <p *ngIf="Dni?.hasError('required')" class="text-danger">
                El dni es requerido.
            </p>
        </div>
    </div>
    <div class="m-3">
        <label for="fecha_nacimiento" class="form-label">Fecha de
Nacimiento</label>
        <input type="date" id="fecha_nacimiento" class="form-control"
formControlName="fechaNacimiento" [(ngModel)]="usuario.fechaNacimiento"
[class.border-danger]="FechaNacimientoValid">
        <div *ngIf="FechaNacimiento?.errors && FechaNacimiento?.touched">
            <p *ngIf="FechaNacimiento?.hasError('required')" class="text-danger">
                La fecha de nacimiento es requerida.
            </p>
        </div>
    </div>
    <div class="m-3">
        <label for="email" class="form-label">Correo Electrónico</label>
        <input type="email" id="email" class="form-control"
formControlName="email" [(ngModel)]="usuario.email" [class.border-danger]="MailValid">
        <div *ngIf="Mail?.errors && Mail?.touched">
            <p *ngIf="Mail?.hasError('required')" class="text-danger">
                La fecha de nacimiento es requerida.
            </p>
        </div>
    </div>
    <div class="m-3">
        <label for="email" class="form-label">Correo Electrónico</label>
        <input type="email" id="email" class="form-control"
formControlName="email" [(ngModel)]="usuario.email" [class.border-danger]="MailValid">
        <div *ngIf="Mail?.errors && Mail?.touched">
            <p *ngIf="Mail?.hasError('required')" class="text-danger">
                | El email es requerido.
            </p>
        </div>
    </div>
    <div class="m-3">
        <label for="password1" class="form-label">Contraseña</label>
        <input type="password" id="password1" class="form-control"
formControlName="password1" [(ngModel)]="usuario.password"
[class.border-danger]="Password1Valid">
        <div *ngIf="Password1?.errors && Password1?.touched">
            <p *ngIf="Password1?.hasError('required')" class="text-danger">
                El password es requerido.
            </p>
        </div>
    </div>

```

```

    </div>
    <div class="m-3">
      <label for="password2" class="form-label">Reingrese la Contraseña</label>
      <input type="password" id="password2" class="form-control"
formControlName="password2" [class.border-danger]="Password2Valid">
      <div *ngIf="Password2?.errors && Password2?.touched">
        <p *ngIf="Password2?.hasError('required')" class="text-danger">
          El password es requerido.
        </p>
      </div>
    </div>
    <div class="m-3">
      <button class="btn btn-primary mt-3" >Enviar</button>
    </div>
  </form>
</div>
</div>
</div>

```

Nota: Observa que el mismo incluye formulario, validaciones y two way binding por lo que, se deberá además configurar previamente todo lo relacionado a ello (ver módulo 5, formularios reactivos).

5. En el controlador, archivo registro.component.ts, editar a fin de:

a. Importar el UsuarioService y Usuario:

```

import { UsuarioService , Usuario} from
'src/app/services/usuario.service';

```

b. Inyectar el UsuarioService en el constructor

```

constructor(private usuarioService: UsuarioService)

```

c. Editar el evento onSubmit (onEnviar) del formulario a fin de hacer la petición al modelo para que éste finalmente haga la petición http.

```

onEnviar(event: Event, usuario:Usuario): void {
  event.preventDefault;

  if (this.form.valid)
  {
    console.log("Enviando al servidor...");
    console.log(usuario);

    this.usuarioService.onCrearUsuario(usuario).subscribe(
      data => {

```

Controlador solicita al Modelo Crear un Nuevo Usuario por medio de la ejecución del método onCrearUsuario(usuario). Pasa por parámetros el objeto Usuario.

Subscribe, funciona como las promesas de Javascript permitiendo esperar hasta que la petición termine. Observa que dentro del método se ejecuta una arrow function, la misma permite devolver el objeto data que contiene la respuesta del UsuarioService.

data. Puedes ver que contiene aquí:
<https://regres.in/api/users/1>

```

    if (data.id>0)
    {
        alert("El registro ha sido creado satisfactoriamente. A continuación, por favor Inicie Sesión.");

        this.router.navigate(['/iniciar-sesion'])
    }
  })
}
else
{
  this.form.markAllAsTouched();
}
};

```

Redirecciona a la ruta de inicio de sesión a fin de obligar al usuario iniciar sesión. Previamente se debe importar router e inyectar en el constructor.

El fuente completo del controlador, archivo usuario.component.ts:

```

import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
import { UsuarioService, Usuario } from 'src/app/services/usuario.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-registro',
  templateUrl: './registro.component.html',
  styleUrls: ['./registro.component.css']
})
export class RegistroComponent implements OnInit {
  form: FormGroup;
  usuario: Usuario = new Usuario();

  constructor(private formBuilder: FormBuilder, private usuarioService:
  UsuarioService, private router: Router) {
    this.form= this.formBuilder.group(
      {
        nombre:['', [Validators.required]] ,
        apellido:['', [Validators.required]],
        dni:['', [Validators.required]],
        fechaNacimiento:['', [Validators.required]],
        password1:['',[Validators.required]],
        password2:['',[Validators.required]],
        email:['', [Validators.required, Validators.email]]
      }
    )
  }
}

```

```
    )
  }

  ngOnInit(): void {
  }

  onEnviar(event: Event, usuario: Usuario): void {
    event.preventDefault();

    if (this.form.valid)
    {
      console.log("Enviando al servidor...");
      console.log(usuario);
      this.usuarioService.onCreatearUsuario(usuario).subscribe(
        data => {
          console.log(data.id);
          if (data.id > 0)
          {
            alert("El registro ha sido creado satisfactoriamente. A continuación, por favor Inicie Sesión.");
            this.router.navigate(['/iniciar-sesion'])
          }
        })
    }
    else
    {
      this.form.markAllAsTouched();
    }
  };

  get Password1()
  {
    return this.form.get("password1");
  }
}
```

```
get Password2()
{
  return this.form.get("password2");
}

get Mail()
{
  return this.form.get("email");
}

get Nombre()
{
  return this.form.get("nombre");
}

get Apellido()
{
}
```

```
    return this.form.get("apellido");
  }

  get FechaNacimiento()
  {
    return this.form.get("fechaNacimiento");
  }

  get Dni()
  {
    return this.form.get("dni");
  }

  get MailValid()
  {
    return this.Mail?.touched && !this.Mail?.valid;
  }

  get NombreValid()
  {
    return this.Nombre?.touched && !this.Nombre?.valid;
  }

  get ApellidoValid()
  {
    return this.Apellido?.touched && !this.Apellido?.valid;
  }

  get Password1Valid()
  {
    return this.Password1?.touched && !this.Password1?.valid;
  }
```

```
  get Password2Valid()
  {
    return this.Password2?.touched && !this.Password2?.valid;
  }

  get FechaNacimientoValid()
  {
    return this.FechaNacimiento?.touched && !this.FechaNacimiento?.valid;
  }

  get DniValid()
  {
    return this.Dni?.touched && !this.Dni?.valid;
  }
}
```

Finalmente, si ejecutamos `ng serve -o`, en el caso que la aplicación no esté corriendo, podremos contemplar la funcionalidad completa de MVC en Angular funcionando.

The screenshot shows a web browser window with the address bar displaying `http://localhost:4200/registro`. The page has a dark header with a logo on the left and navigation links on the right. A white modal box is centered on the screen, displaying a success message: "localhost:4200 dice: El registro ha sido creado satisfactoriamente. A continuación, por favor inicie Sesión." with a green "Aceptar" button. Below the modal, the "Registro Usuario" form is visible, containing fields for Name, Last Name, Address, City, ID Number, Date of Birth, Gender, Email, Password, Confirm Password, and a checkbox for "Acepto la Condición". A blue "Enviar" button is at the bottom of the form.

Figura: Formulario de Registro - MVC

2. Autenticación en Angular

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).

2.1. Servicios de Autenticación

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.

2. Generar el servicio para la lógica de autenticación auth.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';  
  
import { Usuario } from '../usuario.service';
```

- 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 Autenticación 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://regres.in/api/users/1>

Nota: Utilizaremos el API Rest disponible en <https://regres.in/> como lo hicimos en el Registro de usuario.

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

```
logout(): void{
  localStorage.removeItem('currentUser');

  this.loggedIn.next(false);
}
```

g. 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]]
    }
  )
}

```

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 login(usuario). Pasa por parámetros el objeto Usuario.

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

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:

App

localhost:4200/iniciar-sesion

Mi aplicación

SERVICIOS ¿QUÉNES SOMOS? REGISTRARSE INICIAR SESIÓN

¡BIENVENIDOS a nuestra App!
Es un placer ser parte de tu día a día

Iniciar Sesión

Email
eve.holt@reqres.in

Password
xxxxxxxxxx

Iniciar Sesión

Copyright © PL Money 2021

Twitter Facebook LinkedIn

Figura 7: Formulario de inicio de sesión

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

2.2. 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:



Contiene la información sobre una ruta asociada a un componente cargado en una salida en un momento determinado. ActivatedRouteSnapshot también se puede utilizar para atravesar el árbol de estado del enrutador.
Fuente: <https://runebook.dev/es/docs/angular/api/router/activatedroutesnapshot#descripcion>

La función en la ruta es la que hará el llamado del **Guard**, y dependiendo lo que este devuelva, la ruta podrá activarse o mostrarse, o no. Por eso se llama así, **CanActivate**. Pero bueno, empecemos por el principio. Vamos crear nuestro primer **Guard**.
Fuente: <https://www.codigocorrecto.com/programacion/angular-2020-como-usar-guard-canactivate-ejemplo-de-uso/>

Representa el estado de router en un determinado momento.
Fuente: <https://angular.io/api/router/RouterStateSnapshot>

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

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

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

- b. Decorar la clase como injectable 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((isLoggedIn:boolean)=>isLoggedIn));
}
```

- 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.

2.3. 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 (endpoints, 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>).

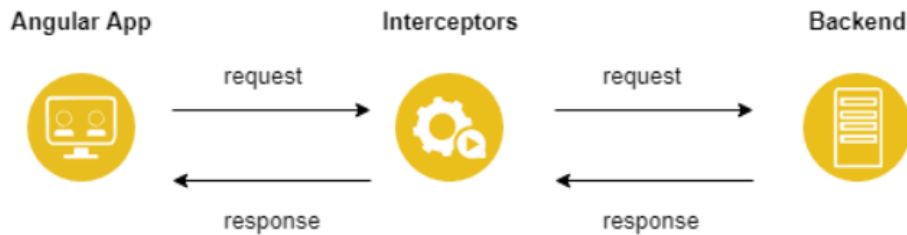


Figura: Interceptors

Fuente: https://res.cloudinary.com/practicaldev/image/fetch/s--skBUyM52--/c_limit%2Cf_auto%2Cf1_progressive%2Cq_auto%2Cw_880/https://dev-to-uploads.s3.amazonaws.com/uploads/articles/51sp8fpfkexj3u234nwo.png_

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';
```

- b. Decorar la clase como injectable 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);
}

```

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, puedes editar su contenido y también la respuesta. Fuente: <https://lutarocarro.blog/usando-http-interceptors-en-blazor/>

2. Generar el servicio para la lógica de error.interceptor.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 { 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 de error. Fuente: <https://www.concretepage.com/angular/angular-throwerror>

El catch es un operador RxJS detecta el error generado por un observable y manipula devolviendo un observable al usuario. Fuente: <https://www.concretepage.com/angular/angular-throwerror>

- 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();
      }
    }));
  }
}

```

Devolverá una notificación de error de por falta de credenciales válidas (401).

```

    }

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

    return throwError(error);

  });
}

```

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@reqres.in y password: cityslicka.

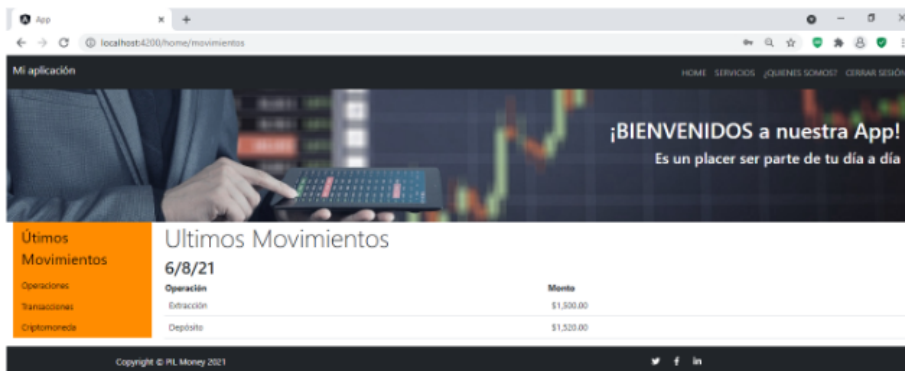


Figura: Redireccionamiento luego de una autenticación exitosa.

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

3. 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/>