



ANGULAR

Hacer un login y
recordar la sesión
sin refrescar

Coding Potions -6-

Introducción

Con lo visto hasta ahora si has ido siguiendo la serie de artículos sobre Angular realmente ya puedes realizar llamadas HTTP para loguearte en una API REST. Simplemente tienes que enviar una petición POST con el nombre de usuario y la contraseña al la dirección del endpoint de la API REST preparada para login. Aún así voy a hacer un ejemplo de cómo hacerlo y

de paso explico cómo mantener la sesión iniciada para que al cerrar o al refrescar la página sigamos logueados.

Cómo empezar a crear un login con Angular

Lo primero que vamos a necesitar es un componente de login para mostrar la página de login con el formulario junto a un servicio para realizar las llamadas HTTP al backend, suponiendo que ya tienes un proyecto en Angular, simplemente ejecutamos:

```
ng generate component login
```

Añadimos el componente al routing, en mi caso voy a crear una ruta llamada login, dentro del array de rutas del routing de nuestra app:

```
{ path: 'login', component: LoginComponent,  
  pathMatch: 'full'}
```

Ahora voy a añadir un formulario sencillo dentro del html de login:

```
<div class="container">
```

```
  <h1>Iniciar sesión</h1>
```

```
  <form class="form">
```

```
    <input type="text" placeholder="Username">
```

```
<input type="password" placeholder="Password">
```

```
<button type="submit" id="login-
```

```
button">Login</button>
```

```
</form>
```

```
</div>
```

También he añadido unos cuantos estilos de ejemplo, el resultado es el siguiente:



El siguiente paso es crear un servicio para el que el login realice peticiones http a la API del servidor, para ello creamos un archivo llamado login.service.ts dentro de la carpeta de login. A continuación creamos las llamadas HTTP, en este caso voy a usar una API de ejemplo de prueba:

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

```
@Injectable()
```

```
export class LoginService {
```

```
constructor(private http: HttpClient) {  
  
}  
  
login(username:string, password:string) {  
    return this.http.post('https://reqres.in/api/login', {  
        email: username,  
        password: password,  
    });  
}  
}
```

Como ves, he creado un método login, el cual recibe el usuario y la contraseña para realizar la llamada post insertando los datos a la API de prueba.

También tenemos que importar este servicio que acabamos de crear dentro del `app.module.ts`, concretamente en el array de providers

Ahora vamos a conectar el servicio con el componente, para ello lo importamos y lo inyectamos en el componente mediante inyección de dependencias:

```
import { Component, OnInit } from '@angular/core';  
import { LoginService } from '../login.service';
```

```
@Component({  
  selector: 'app-login',
```

```
templateUrl: './login.component.html',  
  
styleUrls: ['./login.component.css']  
  
}))  
  
export class LoginComponent implements OnInit {  
  
  constructor(private loginService: LoginService) { }  
  
  ngOnInit() {  
  
    this.loginService.login('peter@klaven',  
  
    'cityslicka').subscribe(  
  
      res => {  
  
        console.log(res);  
  
      });  
  }  
}
```

```
}
```

```
}
```

De paso he llamado al método que acabamos de crear en el `ngOnInit` para que se ejecute al inicio con unos datos de ejemplo para comprobar que el servicio realiza bien la petición `http`.

Para llamar a un método en el componente hacemos que el botón tenga un evento `(click)=" "` y le pasamos como argumento el método a ejecutar con los datos del formulario. Para no tener que poner en los

Este proceso de conexión de la vista con el controlador, se puede hacer de otra forma (con `FormControls` y `FormGroups`), con la cual puedes validar los datos y informar al usuario si ha introducido los campos mal, pero por el momento ésta forma de hacerlo nos vale. No obstante si la quieres utilizar puedes consultar otro artículo que escribí tratando este tema:

[/angular-formularios](#)

Ahora en el controlador de la vista, creamos el método para hacer login llamando al servicio:

```
login(username: string, password: string, event: Event) {
```

```
    event.preventDefault(); // Avoid default action for  
the submit button of the login form
```

```
    // Calls service to login user to the api rest  
    this.loginService.login(username,  
password).subscribe(  

```

```
    res => {  
        console.log(res);  
    }  
}
```

```
},
```

```
error => {
```

```
    console.error(error);
```

```
},
```

```
() => this.navigate()
```

```
);
```

```
}
```

```
navigate() {
```

```
    this.router.navigateByUrl('/home');
```

```
}
```

En este caso he añadido el método `preventDefault()` para que el formulario no realice la acción por defecto y haga el login bien. Además, he añadido un método que se ejecuta tras hacer el login que sirve para dirigir al usuario a otra página.

Recordando la sesión iniciada

Bien, ahora estamos logueados en el backend, pero, ¿cómo mantenemos la sesión iniciada entre las distintas páginas? ¿Cómo mantenemos la sesión cuando recargemos la página?

Para hacer esto tenemos que guardar los datos del usuario en la memoria del ordenador de forma local, también

podemos usar cookies de sesión, pero es más complicado de implementar.

Creando nuestro propio HttpClient

Utilizando lo que hemos aprendido hasta ahora sobre llamadas http podemos hacer llamadas http de todo tipo pero todavía no podemos enviar los headers de autenticación (Basic Auth) para enviar junto a la petición http para usar con APIs protegidas.

Para almacenar el user en memoria del navegador lo mejor es tener un modelo de user para que al guardar el usuario se guarde todo el objeto con toda la información.

Para ello creamos una nueva carpeta llamada user, y creamos un archivo llamado user.model.ts:

```
export class User {  
  username: string;  
}
```

Por el momento vamos a almacenar solo el nombre de usuario y no nos haría falta crear un modelo para esto, pero si en un futuro tenemos que añadir más campos al usuario nos viene bien tenerlo.

Ahora creamos un servicio para el usuario (acuérdate de importarlo en app.module.ts en la sección providers), que se encargará de guardar el user en memoria y tener un control sobre si el usuario está logueado:

```
import { Injectable } from '@angular/core';  
  
import { User } from '../user.model';
```

@Injectable()

export class UserService {

private isUserLoggedIn;

public usserLogged:User;

constructor() {

 this.isUserLoggedIn = false;

}

setUserLoggedIn(user:User) {

 this.isUserLoggedIn = true;

 this.usserLogged = user;

```
    localStorage.setItem('currentUser',  
JSON.stringify(user));  
  
}  
  
getUserLoggedIn() {  
    return  
JSON.parse(localStorage.getItem('currentUser'));  
}  
  
}
```

Como ves lo único que hacemos es crear un par de métodos, uno para setear el user cuando se loguea, añadiendolo al

localStorage del navegador, y otro para devolver el user de localStorage. Las variables que he creado no hacen falta estrictamente, pero las creo por si las quiero usar en un futuro ya que para ver si el user esta logueado, llama cada vez a localStorage cuando solo hace falta hacerlo una vez, cuando se carga toda la app Angular.

Para probar esta funcionalidad tenemos que importar éste servicio que acabamos de crear en login.component.ts, creamos el

user al hacer login y llamamos al método para que se guarde en memoria:

```
import { Component, OnInit } from '@angular/core';
```

```
import { LoginService } from './login.service';
```

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

```
import { UserService } from '../user/user.service';
```

```
import { User } from '../user/user.model';
```

```
@Component({
```

```
  selector: 'app-login',
```

```
  templateUrl: './login.component.html',
```

```
  styleUrls: ['./login.component.css']
```

```
})
```

```
export class LoginComponent implements OnInit {
```

```
constructor(private loginService: LoginService,  
private router: Router, private userService:  
UserService) { }
```

```
ngOnInit() {  
  
}
```

```
login(username: string, password: string, event:  
Event) {
```

```
    event.preventDefault(); // Avoid default action for  
the submit button of the login form
```

```
    // Calls service to login user to the api rest  
    this.loginService.login(username,  
password).subscribe(
```

```
res => {  
  let u: User = {username: username};  
  this.userService.setUserLoggedIn(u);  
  
},  
  
error => {  
  console.error(error);  
  
},  
  
() => this.navigate()  
);  
  
}
```

```
navigate() {  
  
  this.router.navigateByUrl('/home');  
  
}  
  
}
```

Por último, por ejemplo, para incluir el nombre del usuario que está logueado tenemos que recorgerlo desde el servicio con el método que creamos antes, y con un `ngIf` lo pintamos:

```
<div *ngIf="usserLogged">  
  
  
  
</div>
```

`userLogged` es una variable de tipo `User` (hace falta importar el modelo de `User`) que inicializo en el `ngOnInit()` del componente llamando al método `getUserLoggedIn()` del servicio de `user` (importado mediante inyección de dependencias).

De la misma forma, podemos crear un `guard` (lo veremos en otros artículos) para mirar si el usuario está logueado y de esta forma proteger las páginas de usuarios no registrados.

Conclusiones

Esta es la forma más simple de hacer un login con Angular, no es la mejor manera ni la más segura pero para una aplicación sencilla puede servir. Para realizar un registro de usuario lo puedes hacer con los conocimientos visto hasta ahora, simplemente creas un componente para el registro con un formulario en el html y lo conectas al componente, cuando el usuario hace click en el botón se realiza una

petición POST a backend para que se registre el usuario en la base de datos. Por último si no hay código de error desde backend, llamas a setUserLoggedIn() como hemos hecho antes para dejar el user logueado.

Si quieres seguir aprendiendo y mejorando el login y el registro de tu página y esto no te parece suficiente te dejo un par de artículos interesantes para que expandas tus conocimientos (en inglés):




- <https://blog.angular-university.io/angular-jwt-authentication/>
 - <https://medium.com/@amcdnl/authentication-in-angular-jwt-c1067495c5e0>
-

¿Te ha gustado el artículo? ♥♥

Compártelo en las redes sociales para
apoyar a Coding Potions 🍷☐

¿Quieres aprender Angular gratis?

No te pierdas este curso gratuito totalmente online en el que aprenderás uno de los mejores frameworks del momento.

-  Aprende Angular desde 0 y gratis
-  Crea tus propias aplicaciones y páginas web dinámicas
-  Aprende los fundamentos de los componentes web

-  Conecta Angular un servidor
mediante una API REST
-  Depliega tus proyectos en
producción
-    Posts
-  Collections
-  Notes
-  About