Services y dependency injection | Práctica

Índice

Introducción

Para asimilar un concepto, nada mejor que **practicarlo**. Por eso, si todavía no tienes soltura con los servicios (en inglés, ***services)*** o con la inyección de dependencias (en inglés, *dependency injection),* te traigo un ejercicio para que cojas práctica. ¡Vamos allá! 💪

Requisitos previos

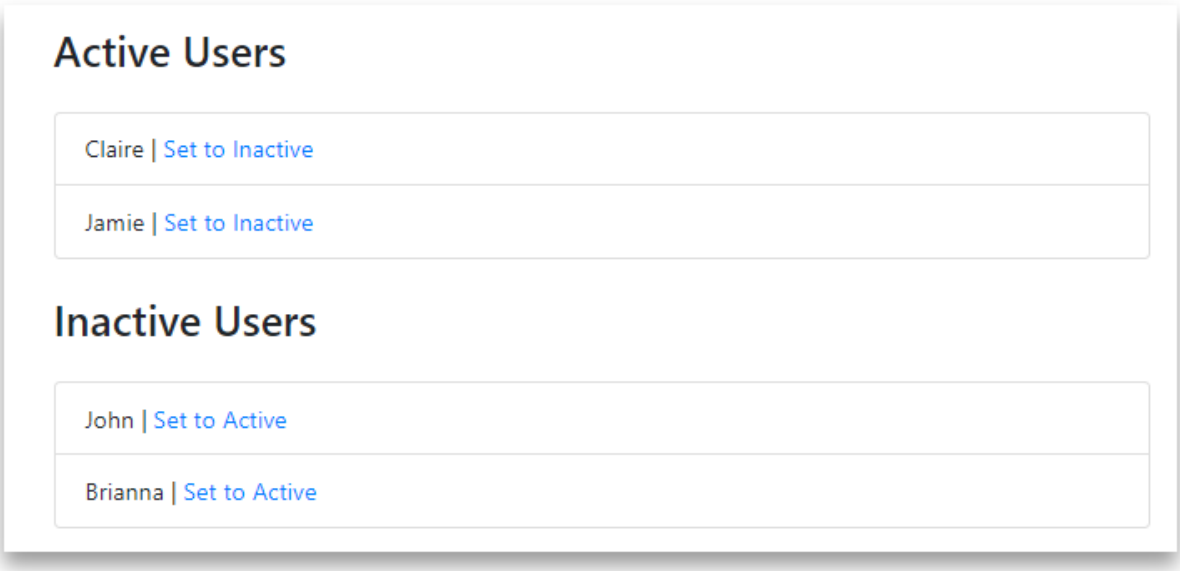
Antes de ponernos manos a la obra, debes conocer los pilares **básicos** sobre los *services* y la *dependency injection.* Aquí tienes una guía completa sobre el tema.

Puedes **descargar** el ejercicio en su estado inicial desde aquí (mi cuenta de GitHub). Sigue los siguientes pasos:

1. Haz clic en el botón Clone or download y luego en *Download ZIP*.

2. Descomprime el proyecto en la ubicación de tu pc que tú elijas y ábrelo con tu editor de código.

3. En tu terminal (yo uso la integrada en VSCode), ejecuta ***npm install.*** Esto instalará los *node_modules.* Una vez instalados, ejecuta *ng serve -o,* lo cual te debería lanzar un servidor **local** con la app, que tiene esta pinta:



Estado inicial de nuestra app

Verás que **funciona** perfectamente, pudiendo hacer clic en los botones azules para cambiar usuarios de una lista a otra (de activos a inactivos o viceversa). Pero si echamos un vistazo al código, comprobarás que es **innecesariamente** complejo para las sencillas funciones que tiene la app.

Objetivos del ejercicio

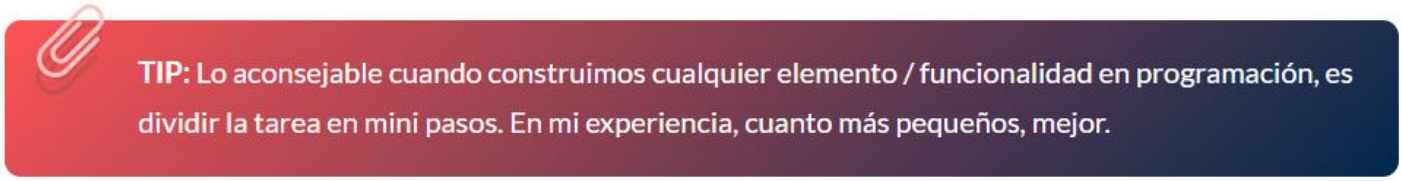
Funcionar, funciona, pero podemos mejorarlo y **simplificarlo** usando servicios. Por tanto, nuestros objetivos son:

- crear un servicio (*UsersService,* por ejemplo) que **gestione** los *arrays* de usuarios activos e inactivos de manera **global**, con métodos para cambiar de una lista a otra.

- crea un servicio (*CounterService,* por ejemplo) que cuente y muestre el número de veces que un usuario (cualquiera, sin especificar) **pasa** de una lista a otra (o sea, de

activo a inactivo o de inactivo a activo). Sin complicarnos la vida, el objetivo es construir un **contador** que sume 1 a una lista de "clics de activos a inactivos" y otro que haga lo contrario.

Para lograrlo, vamos a seguir una serie de pasos.



PASO 1: generar los dos servicios con la CLI de Angular.

PASO 2: crear a los usuarios en el *UsersService* para controlarlos desde ahí.

PASO 3: añadir métodos para pasar a un usuario de activo a inactivo y viceversa.

PASO 4: limpiar el *AppComponent.*

PASO 5: usar el *UsersService* en el *InactiveUsersComponent* y limpiar el *inactive-users.component.ts.* Hacer lo mismo con el *ActiveUsersComponent.*

PASO 6: crear propiedades y métodos en el *CounterService* para imprimir por consola un cambio en el estado del usuario y la **suma** a un lista. Es decir, que registre cuando un usuario hace clic en alguno de los botones azules (Set to Inactive / Set to Active).

PASO 7: inyectar el *CounterService* en el *UsersService* y usarlo ahí.
Vamos a verlos todos en detalle. 😉

Resolviendo el ejercicio

1. Generamos los dos **servicios** usando el comando *ng generate service users-service counter-service,* a la altura del *AppComponent.* Al generarlos con la CLI, ya vienen configurados para poder usar **una** sola instancia de cada servicio a largo de toda nuestra

app, sin que tengamos que hacer ninguna configuración extra, tal y como nos explica Angular en su documentación oficial. (PASO 1)

2. Cortamos los dos arrays del *app.component.ts* y los **pegamos** en el *UsersService.* (PASO 2)

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

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

  activeUsers = ['Claire', 'Jamie'];
  inactiveUsers = ['John', 'Brianna'];
```

3. Creamos un método llamado ***setToActive,*** que espera un *id* como parámetro, que hace referencia a la **posición** en los arrays. En el método, añadimos el elemento del array que el usuario haya seleccionado de la lista de "Inactive Users" y lo **añadimos** al array de *activeUsers.* (PASO 3)

4. Lo eliminamos del array de *inactiveUsers.* (PASO 3)

5. Repetimos el paso 3 pero invertido, bajo otro método llamado *setToInactive.* (PASO 3)

```
 setToActive(id: number) {
    this.activeUsers.push(this.inactiveUsers[id]); // add to active list
    this.inactiveUsers.splice(id, 1); // remove from inactive list
  }

  setToInactive(id: number) {
    this.inactiveUsers.push(this.activeUsers[id]); // add to inactive list
    this.activeUsers.splice(id, 1); // remove from active list
  }

  constructor() { }
}
```

Y con esto, hemos **terminado** de configurar el servicio. ¡Genial! 👍

6. Nos deshacemos de los **métodos** del *app.component.ts* porque ya no los vamos a necesitar. Así que se nos queda un archivo completamente limpio, como en su estado inicial cuando se creó. (PASO 4)

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


@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent { }
```

7. En el *app.component.html,* eliminamos las propiedades de los ***custom selectors,*** porque ahora esos selectores pueden comunicarse con nuestros Servicios ellos solitos, **sin** nuestra ayuda (PASO 4).

```
<app-active-users></app-active-users>
<app-inactive-users></app-inactive-users>
```

8. Utilizamos el ***UsersService*** en el *inactive-users.component.ts.* Para eso, lo inyectamos en el constructor y lo importamos. (PASO 5)

9. Eliminamos el *@Input* porque ya no vamos a recibir el valor de *users* del **exterior**, y nuestro *custom event.* Podemos también **limpiar** los *imports* que ya no necesitemos. (PASO 5)

10. **Inicializamos** el array *users* en el método *ngOnInit,* porque es una buena práctica hacerlo ahí. Para eso, debemos **implementar** la *interface "OnInit".* Le damos el valor del array ***inactiveUsers,*** al que accedemos a través del servicio. (PASO 5)
En el método *onSetToActive,* accedemos al método ***setToActive*** del servicio y le pasamos el *id* por parámetro. (PASO 5)

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

@Component({
  selector: 'app-inactive-users',
  templateUrl: './inactive-users.component.html',
  styleUrls: ['./inactive-users.component.css']
})
export class InactiveUsersComponent implements OnInit {
  users: string[];

  constructor(private usersService: UsersService) { }

  ngOnInit() {
    this.users = this.usersService.inactiveUsers;
  }

  onSetToActive(id: number) {
    this.usersService.setToActive(id);
  }
}
```

11. Hacemos lo **mismo** en el *active-users.component.ts,* pero accediendo al array de *activeUsers* y al método *setToInactive*. (PASO 5)

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

@Component({
  selector: 'app-active-users',
  templateUrl: './active-users.component.html',
  styleUrls: ['./active-users.component.css']
})
export class ActiveUsersComponent implements OnInit {
  users: string[];

  constructor(private usersService: UsersService) { }

  ngOnInit() {
    this.users = this.usersService.activeUsers;
  }

  onSetToInactive(id: number) {
    this.usersService.setToInactive(id);
  }
}
```

¡Casi hemos terminado! 😤 Si guardas y vas a tu navegador, verás que todo funciona igual que antes de empezar esta **refactorización** del código. Es decir, podemos pasar usuarios de una lista a otra. ¡Chachi! Vamos a por el toque final. 🦁

12. En el *CounterService,* creamos dos propiedades de tipo *number* que llevarán el **recuento** (empezando desde 0) de veces que alguien pasa de una lista a otra. (PASO 6)

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

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

  activeToInactiveCounter = 0;
  inactiveToActiveCounter = 0;

  constructor() { }
}
```

13. Creamos un método llamado *incrementActiveToInactive,* donde **incrementaremos** en +1 la propiedad *activeToInactiveCounter* y mostraremos el resultado en la consola. Replicamos esta funcionalidad pero en otro método llamado *incrementInactiveToActive,* con la propiedad *inactiveToActiveCounter.* (PASO 6)

```
incrementActiveToInactive() {
  this.activeToInactiveCounter++;
  console.log('Number of times an inactive user has changed to active: ' +
this.activeToInactiveCounter);
}

incrementInactiveToActive() {
  this.inactiveToActiveCounter++;
  console.log('Number of times an active user has changed to inactive: ' +
this.inactiveToActiveCounter);
}
```

14. Inyectamos y **usamos** el *CounterService* en los métodos del *UsersService.* (PASO 7)

```
setToActive(id: number) {
  this.activeUsers.push(this.inactiveUsers[id]); // add to active list
  this.inactiveUsers.splice(id, 1); // remove from inactive list
  this.counterService.incrementActiveToInactive();
}

setToInactive(id: number) {
  this.inactiveUsers.push(this.activeUsers[id]); // add to inactive list
  this.activeUsers.splice(id, 1); // remove from active list
  this.counterService.incrementInactiveToActive();
}

constructor(private counterService: CounterService) { }
```

¡Y ya lo tenemos! 😸

👩‍🎓 Recuerda que hay miles de maneras de conseguir el **mismo** resultado. Esta es simplemente una de tantas.