Angular S4: Servicios

Después de esta lección podrás:

1. Entender y crear servicios en Angular.
2. Acceder a datos compartidos en tu aplicación.
3. Comunicar componentes sin relación padre e hijo.
4. Usar el modulo HttpClient para lanzar peticiones contra una API .

Ahora que ya sabemos pasar información de unos componentes a otros, vamos a aprender como compartir información de otra manera. ¿Qué ocurre si la información que hay que compartir tiene que estar disponible para muchos componentes? ¿Nos lo vamos pasando de padres a hijos, en cadena?

Pues para esta función existen los **Servicios de Angular**, son artefactos diseñados para compartir información entre varios componentes. La información pueden ser: variables, funciones, etc.

Angular Service

Un servicio de angular, se instancia **una única vez** (*singleton*). Esto es lo que permite que se pueda compartir información entre varios componentes, ya que la **instancia** que consultan y comparten, siempre es la misma.

Un servicio no es un componente, y funcionan de manera transversal en nuestra aplicación, por eso es importante sobre qué modulo declararlos a la hora de proveer datos, ya que en caso de declararlos en varios módulos estaríamos cometiendo el error de crear varias varias el servicio (varias instancias), esto haría imposible compartir información entre los componentes.

A continuación plantearemos un pequeño proyecto, en el que crearemos un componente para listar mensajes, y otro componente para crearlos. Como ya sabemos comunicarlos mediante **input/output**, ahora vamos a ver como se comunicarían a través de un servicio.

Servicios para compartir datos

Vamos a crearnos un proyecto base sobre el que trabajar:

```
ng new message-list-app
cd message-list-app
cd src/app
```

Y a continuación, crearemos los dos componentes: `message-list` y `new-message` además del servicio `messages`:

```
ng generate component message-list
ng generate component new-message
ng generate service messages
```

Cómo en otras ocasiones, limpiamos el **app.component.html**:

```
<app-message-list></app-message-list>
<app-new-message></app-message>
```

Y preparamos nuestro **app.module.ts** para poder declarar dichos componentes, así como generar el **provider** de nuestro servicio `messages`:

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule, ReactiveFormsModule } from '@angular/forms';

import { AppComponent } from './app.component';
import { MessageListComponent } from './message-list/message-list.component';
import { NewMessageComponent } from './new-message/new-message.component';

import { MessagesService } from './messages.service';

@NgModule({
  declarations: [
    AppComponent,
    MessageListComponent,
    NewMessageComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    ReactiveFormsModule
  ],
  providers: [MessagesService], // Aquí proveemos nuestro servicio al módulo
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Dentro de nuestro servicio, debemos declarar las **variables** y **funciones compartidas**. Si os fijáis, por defecto los servicios se generan de manera que el proveedor es el '*root*'. Para mejorar la escalabilidad de nuestra aplicación, seremos nosotros los que decidimos en qué módulo proveerlo. Para ello modificamos `@Injectable`, como se muestra a continuación:

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

// @Injectable({
//   providedIn: 'root'
// })
@Injectable()
export class MessagesService {
  messageList: string[];

  constructor() {
    this.messageList = ['Hola!', 'Soy Pedro', 'Cómo estás?'];
  }

  getMessageList() {
    return this.messageList;
  }

  pushNewMessage(message: string) {
    this.messageList.push(message);
  }
}
```

Ya tenemos un servicio para compartir una variable con los mensajes, y dos funciones para listarlos y crear uno nuevo.

Por parte del componente listado, expondremos en su vista el listado compartido, en el **message-list.component.html**:

```
<h3>Lista de mensajes</h3>
<div *ngIf="list.length">
  <p *ngFor="let message of list">{{ message }}</p>
</div>
```

Y a nivel de lógica en **message-list.component.ts**, tendremos que inyectar el servicio en el constructor, para poder tener acceso a sus variables compartidas en el componente:

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

```
import { MessagesService } from '../messages.service';

@Component({
  selector: 'app-message-list',
  templateUrl: './message-list.component.html',
  styleUrls: ['./message-list.component.scss']
})
export class MessageListComponent implements OnInit {
  list: string[]; // Listado local del componente

  // Inyección de dependencia del servicio
  constructor(messagesService: MessagesService) {
    this.list = messagesService.getMessageList();
  }

  ngOnInit() {

  }

}
```

Repasemos hasta este punto, hemos declarado un listado en un servicio, y lo hemos visualizado en un componente. Ahora vamos a meter nuevos mensajes en el listado, desde otro componente. ¡Y todo esto ahorrándonos los input/output!

Para el componente de creación de mensajes, vamos a crear un pequeño formulario en **new-message.component.html**:

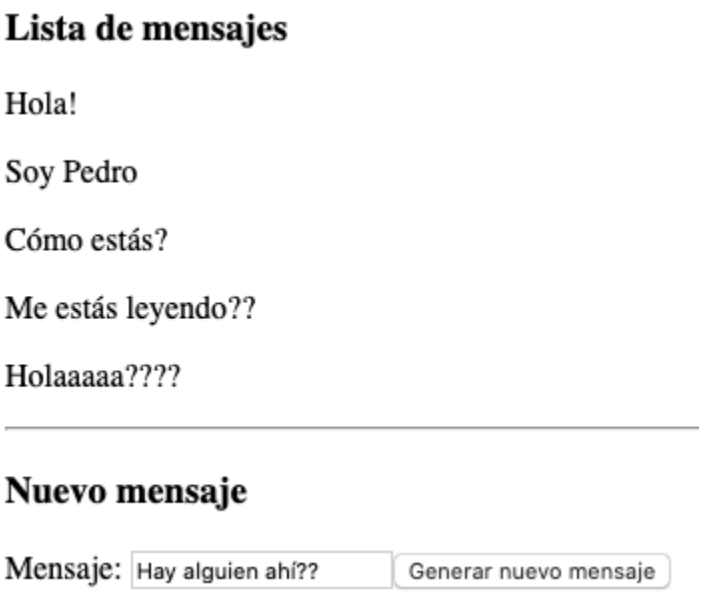
```
<h3>Nuevo mensaje</h3>
<form novalidate (ngSubmit)="onSubmit()" [formGroup]="messageForm">
  <label>Mensaje: <input type="text" formControlName="message" /></label>
  <button class="button" type="submit">Crear</button>
</form>
```

Y en su fichero de lógica, vamos a crear mensajes en el servicio, lo que provocará que el componente del listado se entere de los nuevos mensajes.

```
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormControl, Validators } from '@angular/forms';
```

```
import { MessagesService } from '../messages.service';


@Component({
  selector: 'app-new-message',
  templateUrl: './new-message.component.html',
  styleUrls: ['./new-message.component.scss']
})
export class NewMessageComponent implements OnInit {
  messageForm: FormGroup;

  constructor(private messagesService: MessagesService) {
    this.messageForm = new FormGroup({
      message: new FormControl('', Validators.minLength(2)),
    });
  }

  ngOnInit() {}

  onSubmit(): void {
    this.messagesService.pushNewMessage(this.messageForm.value.message);
    this.messageForm.reset();
  }
}
```

Finalmente, nos quedará una aplicación tal que así:



Y ya hemos aprendido otra manera de **comunicarnos** entre componentes. Por lo general, usaremos siempre el patrón **input/output** entre componentes dentro de un mismo módulo.

Los servicios son útiles para información muy general que necesita ser compartida por varios componentes.