DWC

(Desarrollo Web en entorno cliente)

Índice

1.- Los servicios

Los componentes de nuestra aplicación tienen un trabajo, presentar datos y permitir que el usuario interactúe con esa información.

La documentación de Angular indica, **un Componente no debe tener la responsabilidad de consultar datos o almacenarlos, esa responsabilidad es para los Servicios**.

El trabajo de un servicio es el de controlar la información, desde obtenerla, almacenarla, actualizarla y compartirla con los componentes.

No hay nada especial acerca de un Servicio en Angular, excepto que estos **deben de integrarse con los componentes vía el inyector de Dependencias de Angular**.

Un servicio **es una clase, comúnmente decorada con el decorador Injector de Angular**, mismo que indica que este Servicio puede inyectar otras dependencias de la aplicación, ya sean otros servicios como el de Http para hacer consultas al servidor.

Un aspecto importante de los servicios es que **son objetos Singleton**, lo que esto quiere decir es que nunca se crean dos objetos de una misma clase de Servicio, por lo que puedes estar seguro de que, si dos o más componentes requieren del Servicio, todos estarán usando el mismo objeto.

¿Por qué esto es importante? Porque **todos los componentes comparten la misma información, si un componente actualiza los datos de un servicio, este cambio se verá reflejado en todos los componentes que estén usando esa misma información**.

Esto hace de los servicios el punto a través del cual varios componentes pueden compartir información entre sí.

Internamente, un servicio tiene métodos y propiedades, como cualquier otro objeto, estos se utilizarán para consultar información, enviarla a los componentes, guardarla en algún almacén, etc.

Para resumir, un servicio debe usarse cuando:

- Queremos consultar datos para nuestra aplicación, por ejemplo, usando peticiones Http, en Angular será mediante Observables.
- Queremos almacenar información, ya sea en un servicio web o en algún almacén local.
- Queremos compartir datos entre componentes, porque en una App de angular nunca se crean dos objetos de un Servicio, siempre es uno y ese se envía a todos los componentes.
- Los servicios son clases que se envían a los componentes vía el inyector de dependencias

Al igual que con el resto de elementos de Angular, crearemos una carpeta para almacenar los servicios.

Para crear un servicio desde el CLI de Angular deberemos ejecutar:

```
ng g s /Services/articulos
```

En este caso hemos creado un servicio llamado artículos en la carpeta Services.

Inicialmente Angular habrá creado el esqueleto del servicio, al igual que hacía con los componentes. Un servicio después de crearlo quedaría así:

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

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

  constructor() { }
}
```

Como podemos ver ha creado un fichero prácticamente vacío en el cual podemos observar que aparece el decorador **@Injectable** que será el que nos permitirá inyectar el servicio en los componentes o incluso en otro servicio.

Como hemos dicho que un servicio es el que debería de acceder y manipular los datos que posteriormente manejaran los componentes, en este servicio articulos haremos eso.

Vamos a crear un servicio que sea el que se encargue de manipular los datos de nuestra aplicación, en nuestro caso se encargará de obtener los datos (será el mock que tenemos definido) e implementar los métodos adecuado para poder gestionar esos datos.

Para ello definiremos una variable que sea la que tenga los datos de los artículos y definiremos los métodos que manipularan esos datos:

- La variable será artículos y se incializara con los datos de la variable ARTICULOS que tenemos definida en la carpeta Modelos.
- Los métodos que definiremos serán los habituales de una api REST: getArticulos, getArticulo, postArticulo, putArticulo y deleteArticulo. Podriamos definir todos los métodos que nos hicieran falta para poder manipular los datos de artículos.

```
import { Injectable } from '@angular/core';
import { ARTICULOS, Articulo } from '../Modelos/articulo';


@Injectable({
  providedIn: 'root'
})

export class ArticulosService {

  constructor() { }

  articulos:Articulo[]=ARTICULOS

  getArticulos(){
    return this.articulos
  }

  getArticulo(id:string){
    let pos=this.articulos.findIndex(a=>a.id==id)
    return this.articulos[pos]
  }

  postArticulo(articulo:Articulo){
    let pos=this.articulos.findIndex(a=>a.id==articulo.id)
    if(pos==-1)
     this.articulos.push(articulo)
    else
      alert("Ya existe un artículo con ese id")
  }

  putArticulo(articulo:Articulo){
    let pos=this.articulos.findIndex(a=>a.id==articulo.id)
    this.articulos[pos]=articulo
  }

  deleteArticulo(id:string){
    let pos=this.articulos.findIndex(a=>a.id==id)
    this.articulos.splice(pos, 1)
  }
}
```

Ya tenemos un servicio que podrán usar todos los componentes que necesiten trabajar con los datos de los artículos.

2.- Inyección de dependencia (ID)

Una vez creado nuestro servicio tenemos que ver como lo podrán utilizar nuestros componentes. Para ello utilizaremos **la inyección de dependencias**.

Inyección de dependencia es una pieza clave del framework de Angular y se usa en todas partes para proporcionar nuevos componentes con los servicios u otras cosas que necesitan. Los componentes consumen servicios; es decir, podemos inyectar un servicio en un componente, dándole acceso al componente a ese servicio.

Para definir una clase como un servicio en Angular, usa el decorador **@Injectable ()** para proporcionar los metadatos que le permitan a Angular inyectarlo en un componente como una dependencia. De manera similar, usa el decorador @Injectable() para indicar que un componente u otra clase (como otro servicio, un pipeline o un NgModule) tiene una dependencia.

El inyector es el mecanismo principal. Angular crea un inyector para toda la aplicación durante el proceso de arranque e inyectores adicionales según sea necesario. No es necesario crear inyectores. Un inyector crea dependencias y mantiene un contenedor de instancias de dependencia que reutiliza si es posible.

Un proveedor es un objeto que le dice a un inyector cómo obtener o crear una dependencia.

Para cualquier dependencia que necesitemos en nuestra aplicación, debemos registrar un proveedor con el inyector de la aplicación, con el fin de que el inyector pueda utilizar el proveedor para crear nuevas instancias. Para un servicio, el proveedor suele ser la propia clase de servicio. **Cuando Angular crea una nueva instancia de una clase de componente, determina qué servicios u otras dependencias necesita ese componente al observar los tipos de parámetros del constructor.**

Vamos a utilizar nuestro servicio articulos en un componente llamado artículos.

```
ng g c /Components/articulos
```

para hacer esto deberemos decirle al constructor del componente que va a recibir un parámetro del tipo Servicio, previamente deberemos de haber importado el servicio

```
import { ArticulosService } from '../../Services/articulos.service';
```

```
constructor(private miServicio:ArticulosService){}
```

Cuando Angular descubre que un componente depende de un servicio, primero verifica si el inyector tiene instancias existentes de ese servicio. Si una instancia de servicio solicitada aún no existe, el inyector crea una utilizando el proveedor registrado y la agrega al inyector antes de devolver el servicio a Angular. Cuando todos los servicios solicitados se han resuelto y devuelto, Angular puede llamar al constructor del componente con esos servicios como argumentos.

Realmente lo que hemos hecho al inyectar el servicio en el constructor del componente es decirle al componente que cree un variable del tipo servicio articulos para que pueda usarse en los métodos del componente. Con este mecanismo no debemos crear ninguna variable nosotros ya lo hace directamente el constructor. Al seguir un **patrón singleton** sólo se creará una instancia del servicio lo cual nos permitirá tener los datos actualizados para cualquier componente que utilice dicho servicio.

Desde el momento que definimos el parámetro en el constructor ya podríamos usar los datos y los métodos del servicio.

Es habitual que nuestro componente inicialmente muestre los datos del servicio y posteriormente el usuario vaya solicitando los métodos que necesite del servicio. Para ello el componente al iniciarse debería de tener los datos disponibles para poder interpolarlos en la vista, esto es una tarea que se deja al **método ngOnInit** que será el encargado de asignar los datos iniciales del servicio a la variable del componente que interpolaremos en la vista.

El proceso de inyección quedaría asi:

```
import { Component } from '@angular/core';
import { ArticulosService } from '../../Services/articulos.service';
import { Articulo } from '../../Modelos/articulo';

@Component({
  selector: 'app-articulos',
  standalone: true,
  imports: [],
  templateUrl: './articulos.component.html',
  styleUrl: './articulos.component.css'
})
export class ArticulosComponent {

  articulos!:Articulo[]

  constructor(private miServicio:ArticulosService){}

  ngOnInit(){
    this.articulos=this.miServicio.getArticulos()
  }
}
```

La variable pasada en el constructor deberá de tener un ámbito **public o private** de lo contrario nos dará un error. La diferencia entre ambas es que:

- Si usamos **private,** en la vista del componente **no podremos utilizar una interpolación de métodos** del servicio, es decir el uso del servicio quedar restringido al controlador.
- Si usamos **public**, podremos utilizar los métodos del servicio para poder interpolar en la vista, es decir **el uso del servicio se podrá realizar tanto en el controlador como en la plantilla**.

En html deberíamos recorrer el array y mostrar cada uno de los artículos.

```
<h3>Listado de articulos</h3>

<table class="table ">
    <thead>
      <tr>
        <th scope="col">Id</th>
        <th scope="col">Nombre</th>
        <th scope="col">Descripcion</th>
        <th scope="col">Precio</th>
        <th scope="col">Unidades</th>
      </tr>
    </thead>
    <tbody>
    @for(articulo of articulos; track articulo.id){
        <tr>
            <td>{{articulo.id}}</td>
            <td>{{articulo.nombre}}</td>
            <td>{{articulo.descripcion}}</td>
            <td>{{articulo.precio}}</td>
            <td>{{articulo.unidades}}</td>
        </tr>
    }@empty{
        <div class="alert alert-danger">No hay articulos</div>
    }
    </tbody>
</table>
```

El resultado

Listado de articulos

Id	Nombre	Descripcion	Precio	Unidades
m1	Galaxy A32	4GB + 128GB libre	229	10
m2	Oppo A94	8GB + 128GB libre	269	10
m3	Galaxy S22	5G Amoled libre	859	10
m4	Apple iPhone	14 Pro móvil libre	339	0
m5	Galaxy Z Flip4	5G móvil libre	1990	10
m6	Note 11	6GB + 128GB	300	10
m7	Realme 9	8GB + 128GB	300	10
p1	Asus Zen	i5 16Gb SSD	900	10
p2	HP Pavillion	SSD Windows 11	750	10
p3	MacBook	MacOs 13,3"	1115	10
t1	Xiaomi P1E	Android 9 HDR10	300	10
t2	Toshiba 55UAG	Android UHD 4K	450	10
t3	Samsung UE305	Full HD, HDR	350	10

3.- Proporcionar servicios

Deberíamos registrar al menos un proveedor de cualquier servicio que vayamos a utilizar, esto quiere decir que según donde lo registremos tendrá un ámbito de uso u otro. El proveedor puede formar parte de los propios metadatos del servicio, haciendo que ese servicio esté disponible en todas partes, o puede registrar proveedores con módulos o componentes específicos.

Para cada caso debemos registrar proveedores en:

- Los metadatos del servicio, en el decorador @Injectable()
- Los metadatos @NgModule()
- Los metadatos @Component()

Registro en el decorador @Injectable()

Por defecto, el comando CLI de Angular ng generate service registra un proveedor con el inyector raíz para nuestro servicio al incluir metadatos del proveedor en el decorador **@Injectable()**. El tutorial utiliza este método para registrar el proveedor de la definición de clase HeroService.

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

Cuando proporcionas el servicio en el nivel raíz, Angular crea una instancia única compartida del servicio y lo inyecta en cualquier clase que lo solicite. El registro del proveedor en los metadatos @Injectable() también permite a Angular optimizar una aplicación eliminando el servicio de la aplicación compilada si no se utiliza.

Si eliminanos el inyector en el servicio:

```
import { Injectable } from '@angular/core';
import { ARTICULOS, Articulo } from '../Modelos/articulo';

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

No habrá ningún inyector y cuando el componente intente acceder al servicio verá que no tiene acceso y dará un error.

```
▶ERROR NullInjectorError: R3InjectorError(Environment      main.ts:5
Injector)[ArticulosService -> ArticulosService]:
  NullInjectorError: No provider for ArticulosService!
    at NullInjector.get (core.mjs:1654:27)
    at R3Injector.get (core.mjs:3093:33)
    at R3Injector.get (core.mjs:3093:33)
    at ChainedInjector.get (core.mjs:15678:36)
    at lookupTokenUsingModuleInjector (core.mjs:5730:39)
    at getOrCreateInjectable (core.mjs:5778:12)
    at Module.ɵɵdirectiveInject (core.mjs:11005:19)
    at NodeInjectorFactory.ArticulosComponent_Factory [as factory] (a
rticulos.component.ts:13:32)
    at getNodeInjectable (core.mjs:5984:44)
    at instantiateAllDirectives (core.mjs:11862:27)
```

Lo más habitual es siempre definir el inyector en el servicio con el decorador. También existe la posibilidad de definir el inyector en el componente o en un módulo (versiones anteriores a la 17). En ambos casos hay que utilizar el array de **providers**

Registro en el decorador @Component()

Cuando registramos un proveedor a nivel del componente, obtenemos una nueva instancia del servicio con cada nueva instancia de ese componente. A nivel del componente, registraremos un proveedor de servicios en la propiedad providers de los metadatos @Component().

```
@Component({
  selector: 'app-articulos',
  standalone: true,
  imports: [],
  providers:[ArticulosService],
  templateUrl: './articulos.component.html',
  styleUrl: './articulos.component.css'
})
```

Registro en el decorador @Module()

Cuando registramos un proveedor con un **@NgModule** específico, la misma instancia de un servicio está disponible para todos los componentes en ese NgModule. Para registrar en este nivel, usaremos la propiedad providers del decorador @NgModule() dentro del fichero ts del módulo.

```
@NgModule({
  declarations: [
    AppComponent,
    ArticulosComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule
  ],
  providers: [ArticulosService],
  bootstrap: [AppComponent]
})
```