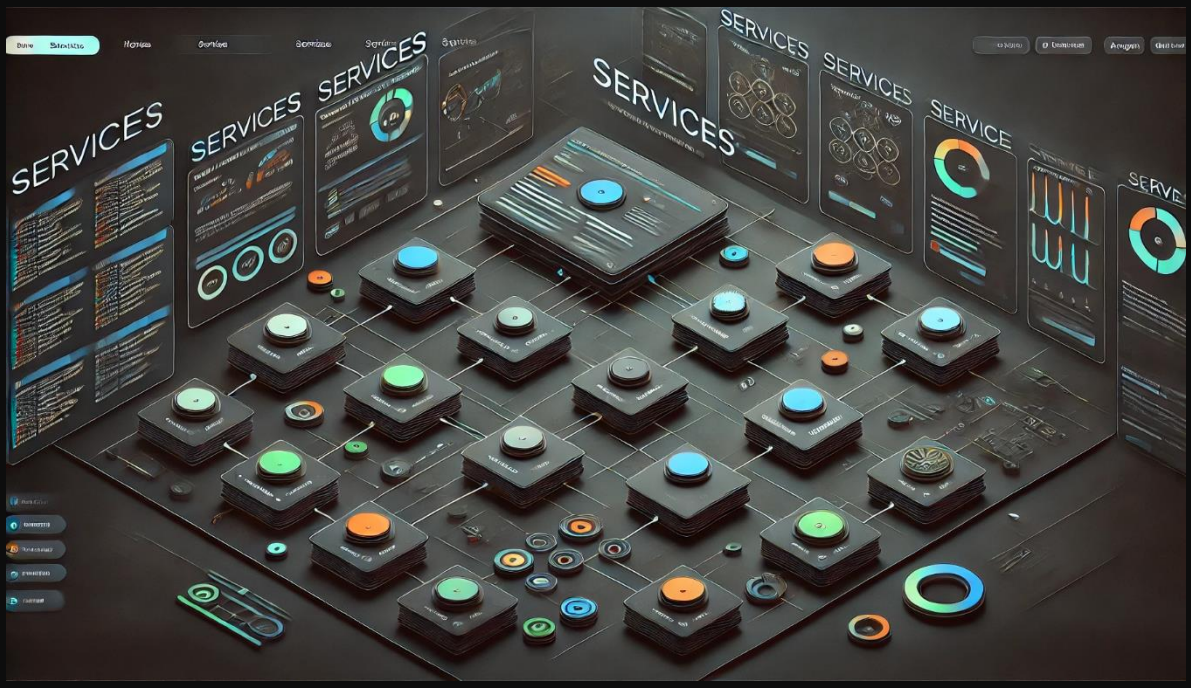
Reto: Servicio en Tienda Online



Reto: Servicios en Tienda Online – Listado de Productos

En este reto les vamos a dejar crear un Servicio de Productos. El servicio de productos será ahora el encargado de tener el arreglo de productos, y el encargado de realizar operaciones como las de agregar un nuevo producto.

Solución: Servicios en Tienda Online

Creación de un servicio

Creamos el servicio de productos. Usamos el siguiente comando de CLI de Angular:

```
ng g s producto -skip-tests
```

Código del servicio de producto.service.ts:

```
import { Injectable } from '@angular/core';
import { Producto } from './producto/producto.model';

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

```
  productos: Producto[] = [
    new Producto('Pantalón', 130.0),
    new Producto('Camisa', 80.0),
    new Producto('Playera', 50.0),
  ];

  agregarProducto(producto: Producto) {
    this.productos.push(producto);
  }
}
```

En este servicio, definimos ahora el arreglo de productos iniciales. Además agregamos el método `agregarProducto` para agregar el nuevo producto al arreglo de productos. Ya no es necesario emitir un evento desde el componente de listado-productos, sino que ahora a través del servicio realizaremos esta tarea.

El servicio está decorado con `@Injectable` y tiene un `providedIn: 'root'`, lo que indica que el servicio es **singleton** a nivel de toda la aplicación y será inyectado automáticamente cuando sea necesario.

Uso de un servicio en los componentes de listado-personas y formulario

Para utilizar el servicio en un componente, simplemente se inyecta en el constructor del componente.

Código listado-productos.component.html

```
<div class="container mt-3">
    <h3 class="text-warning">Listado de Productos</h3>

    <ul class="list-group w-50 mx-auto">
        @for (productoElemento of productos; track productoElemento) {
        <app-producto [producto]="productoElemento" />
        }
    </ul>

    <app-formulario />

</div>
```

Ya no es necesario monitorear ningún evento desde el componente hijo de <app-formulario/>

Código listado-productos.component.ts

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

```
import { ProductoComponent } from '../producto/producto.component';
import { Producto } from '../producto/producto.model';
import { FormularioComponent } from '../formulario/formulario.component';
import { ProductoService } from '../producto.service';

@Component({
  selector: 'app-listado-productos',
  standalone: true,
  imports: [ProductoComponent, FormularioComponent],
  templateUrl: './listado-productos.component.html',
  styleUrls: ['./listado-productos.component.css'],
})
export class ListadoProductosComponent {
  productos: Producto[] = [];

  constructor(private productoService: ProductoService) {}

  ngOnInit(): void {
    // Inicializamos los productos
    this.productos = this.productoService.productos;
  }
}
```

En este componente, el servicio `ProductoService` se inyecta en el constructor de manera automática por Angular y luego, en el método `ngOnInit` lo utilizamos para inicializar el arreglo de productos utilizando el `ProductoService`.

Por último, modificamos el componente de `formulario.component` como sigue:

Código formulario.component.html:

```
<h5 class="text-success mt-3">Agregar Nuevo Producto</h5>

<form class="d-flex justify-content-between align-items-center my-4 w-50 mx-auto">
    <input type="text" id="descripcion" name="descripcion"
        placeholder="Descripción Producto" class="form-control me-2 w-75"
        autocomplete="off"
        [(ngModel)]="descripcionInput" />

    <input type="number" id="precio" name="precio"
        placeholder="Precio" class="form-control me-2 w-25"
        [(ngModel)]="precioInput" />

    <button type="submit" class="btn btn-primary"
(click)="agregarProducto()">Agregar</button>
```

```
</form>
```

En este caso hemos vuelto a aplicar el concepto de two way binding, sin embargo pueden utilizar cualquier opción entre `@ViewChild` o el concepto de two way binding.

Código formulario.component.html:

```
import { Component } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { Producto } from '../producto/producto.model';
import { ProductoService } from '../producto.service';

@Component({
  selector: 'app-formulario',
  standalone: true,
  imports: [FormsModule],
  templateUrl: './formulario.component.html',
  styleUrl: './formulario.component.css'
})
export class FormularioComponent {
  descripcionInput: string = '';
  precioInput: number | null = null;

  constructor(
    private productoService: ProductoService
  ){}

  agregarProducto() {
    // Validación simple para evitar productos sin descripción o con precio cero
    if (this.descripcionInput.trim() === ''
        || this.precioInput == null || this.precioInput <= 0) {
      console.log('Debe ingresar una descripción y un precio válidos');
      return;
    }
    const producto = new Producto(this.descripcionInput, this.precioInput);
    // Agregamos el nuevo producto usando el servicio
    this.productoService.agregarProducto(producto);

    // Limpia los campos de entrada después de agregar el producto
    this.descripcionInput = '';
    this.precioInput = null;
  }
}
```

Aquí lo importante es el constructor de la clase `FormularioComponent`. En el cual podemos ver la inyección de dependencias para usar el servicio `ProductoService`.

Y posteriormente en el método `agregarProducto` ya no es necesario emitir el evento para notificar el componente padre de listado-productos que se ha agregado un nuevo producto, sino que simplemente utilizando el `ProductoService` podemos llamar al método `agregarProducto` para realizar ahora esta tarea. El componente padre de manera automática detecta un cambio en los datos de arreglo y refresca la página. Sin embargo, en un momento más vamos a revisar a detalle el proceso de actualización de datos en la siguiente lección.

Resultado de Ejecutar la aplicación de Tienda Online:

Saludos!

Ing. Ubaldo Acosta

Fundador de GlobalMentoring.com.mx