Reto: Tienda Online- Listado de Productos

En esta ocasión les dejaremos modificar la aplicación de tienda online para aplicar los conceptos estudiados en esta lección en el listado de productos, y el formulario para agregar un nuevo producto.

Ahora si ya vamos a utilizar cada uno de los componentes que hemos creado anteriormente, y aplicar el concepto de modularización y comunicación entre componentes para mejorar nuestra aplicación de tienda online.

Aunque visualmente no realizaremos grandes cambios, lo que realmente modificaremos será el código para hacer más modular nuestra aplicación. El resultado final es el siguiente:

Los cambios a realizar son los siguientes:

1) El componente de listado-productos será nuestro componente padre. Y a partir de este componente agregaremos dos componentes hijos.
2) Componente hijo de formulario-producto. Se debe agregar este componente hijo, y este componente tendrá ahora la funcionalidad para agregar un nuevo producto a la lista de productos. Usará el concepto de @Output para emitir un evento al momento de agregar un nuevo producto. Por otro lado, los campos del formulario se pueden procesar con @ViewChild en lugar de ngModel para que pongan en práctica este concepto.
3) El componente de producto ahora se utilizará para mostrar el detalle de cada producto. A partir del componente padre listado-productos, cada vez que se itere un producto, usaremos el componente de producto para recibir el detalle de cada producto. Para ello usaremos el decorador @Input y así poder modificar la información del componente hijo.
4) Estos son los puntos generales a modificar. Mucho éxito y a continuación veamos la solución de la aplicación tienda online – listado de productos.

Solución Tienda Online – Listado de Productos

Vamos a ver a continuación los cambios en cada componente, así como lo necesario para que todo funcione correctamente.

En primer lugar vamos a crear el componente de formulario para poder separar todo el código del formulario para agregar un nuevo producto dentro de este componente:

ng g c formulario --skip-tests

Código formulario.componente.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"
    (submit)="agregarProducto($event)">
    <input type="text" id="descripcion" name="descripcion"
        placeholder="Descripción Producto" class="form-control me-2 w-75"
        autocomplete="off"
        #descripcionInput/>

    <input type="number" id="precio" name="precio"
        placeholder="Precio" class="form-control me-2 w-25"
        #precioInput />

    <button type="submit" class="btn btn-primary">Agregar</button>
</form>
```

Código formulario.componente.ts:

```
import { Component, ElementRef, EventEmitter, Output, ViewChild } from '@angular/core';
import { Producto } from '../producto/producto.model';

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

  @ViewChild('descripcionInput') descripcionInput!: ElementRef;
  @ViewChild('precioInput') precioInput!: ElementRef;
  @Output() nuevoProducto = new EventEmitter<Producto>();

  agregarProducto(evento: Event){
    evento.preventDefault();

    //Validar que sean valores correcto
    if(this.descripcionInput.nativeElement.value.trim() === ''
      || this.precioInput == null || this.precioInput.nativeElement.value <=0){
      console.log('Debe ingresar una descripción y un precio válidos');
      return;
    }

    const producto = new Producto(this.descripcionInput.nativeElement.value,
      this.precioInput.nativeElement.value);

    //Emitir el evento de nuevo producto
    this.nuevoProducto.emit(producto);

    // Limpiamos los campos del formulario
    this.descripcionInput.nativeElement.value = '';
    this.precioInput.nativeElement.value = null;
  }
}
```

Explicación del Código:

- **(submit)="agregarProducto($event)"**: Aquí capturas el evento de envío del formulario, y al recibir el evento, puedes prevenir el comportamiento por defecto en caso de que la página se recargue en automático.

- **event.preventDefault()**: Evita que el formulario recargue la página cuando hacemos click el botón de Agregar.

Código listado-productos.componente.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 (nuevoProducto)="agregarProducto($event)" />

</div>
```

Código listado-productos.componente.ts:

```
import { Component } from '@angular/core';
import { ProductoComponent } from '../producto/producto.component';
import { Producto } from '../producto/producto.model';
import { FormularioComponent } from "../formulario/formulario.component";

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

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

Código producto.componente.html:

```
<li class="list-group-item text-center">
    {{ producto.descripcion }},
    ${{ producto.precio }}
</li>
```

Código producto.componente.ts:

```
import { Component, Input } from '@angular/core';
import { Producto } from './producto.model';

@Component({
  selector: 'app-producto',
  standalone: true,
  imports: [],
  templateUrl: './producto.component.html',
  styleUrl: './producto.component.css'
})
export class ProductoComponent {
  @Input() producto!: Producto;

}
```

Resultado final:

Saludos!

Ing. Ubaldo Acosta

Fundador de GlobalMentoring.com.mx