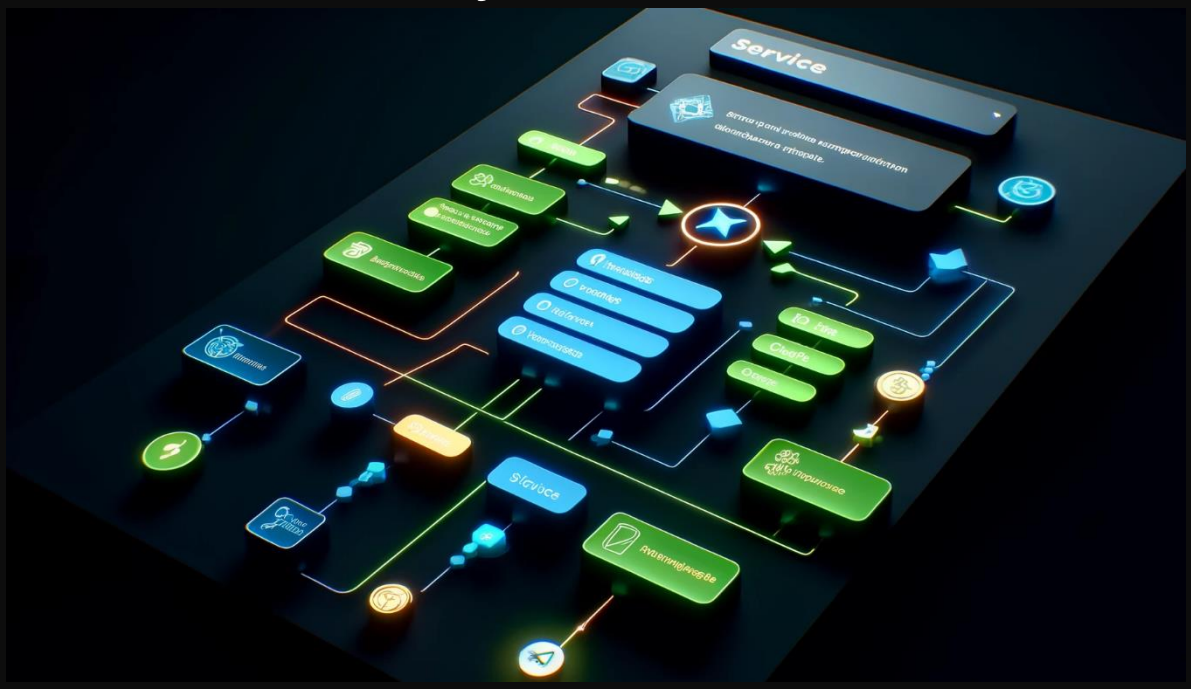
Emitir un Mensaje desde un Servicio



En Angular, se puede utilizar un **Observable** para emitir eventos desde un servicio y luego permitir que los componentes se suscriban a esos eventos para reaccionar a ellos. Esto es útil cuando queremos comunicar datos entre componentes sin necesidad de una relación directa (por ejemplo, de hijo a padre).

Vamos a crear un ejemplo simple donde el servicio de producto emite un mensaje y el componente de listado de producto se suscribe para recibir ese mensaje. En este caso, utilizaremos **RxJS** con **EventEmitter** para emitir eventos.

1. Servicio: `ProductoService`

Este servicio utiliza un **EventEmitter** para emitir y compartir mensajes entre componentes:

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

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

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

  detalleProductoEmitter = new EventEmitter<Producto>();

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

2. Componente Emisor: `ProductoComponent`

Este componente se encargará de emitir el mensaje utilizando el servicio de producto.

HTML (`producto.component.html`):

```
<li class="list-group-item text-center">
    {{ producto.descripcion }},
    ${{ producto.precio }}
    <button
    type="submit"
    class="btn btn-info ms-4"
    (click)="emitirDetalleProducto()">Ver Detalle
    </button>
</li>
```

TypeScript (`producto.component.ts`):

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

```
import { Producto } from './producto.model';
import { ProductoService } from '../producto.service';

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

  constructor(private productoService: ProductoService){}

  emitirDetalleProducto(){
    // Usamos el EventEmitter del servicio
    this.productoService.detalleProductoEmitter.emit(this.producto);
  }
}
```

3. Componente Receptor: `ListadoProductosComponent`

Este componente se suscribirá al `ProductoService` para recibir mensajes emitidos por el componente emisor (ProductoComponent en este caso).

HTML (`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>
```

TypeScript (`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) {
    this.productoService.detalleProductoEmitter.subscribe(
      (producto: Producto) => alert(`Producto: ${producto.descripcion} ,
                              $${producto.precio}`)
    );
  }

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

En este componente de listado-productos nos suscribimos al mensaje emitido por el componente producto.component.ts que a su vez fue emitido con ayuda de ProductoService. Sin embargo, nos podemos suscribir desde cualquier otro componente, no tienen que ser necesariamente el componente de listado-productos.component.ts, podría ser por ejemplo el componente de formulario.component.ts. Así que podemos observar que no hay ninguna relación de padre-hijo entre los componentes que emiten el mensaje y el que lo procesa. Lo único que tienen en común es el uso del servicio de persona.

4. Archivo principal: `app.component.html (NO tiene cambios)`

Este archivo muestra ambos componentes en la misma vista:

```
<div class="container text-center my-3">
  <h1 class="text-info">{{titulo}}</h1>
  <app-listado-productos/>
</div>
```

5. Archivo principal: `app.component.ts` (NO tiene cambios)

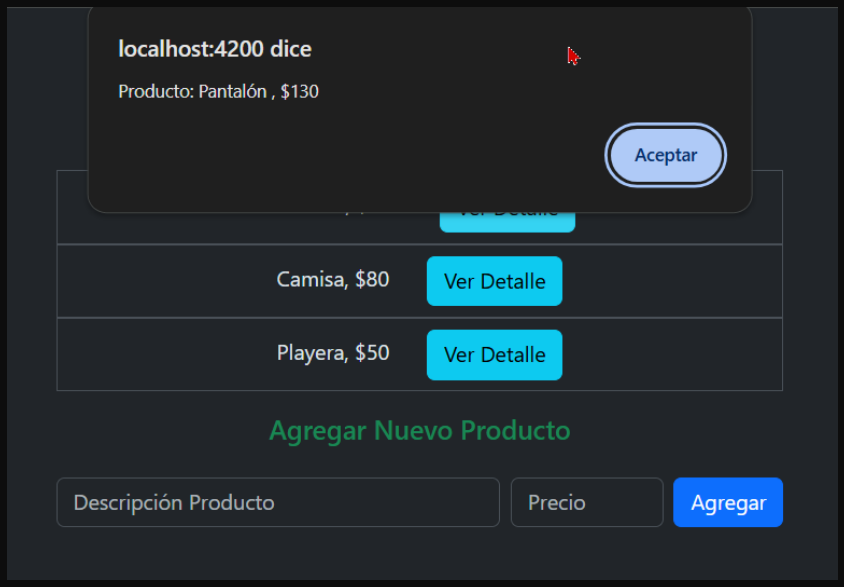
```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { ListadoProductosComponent } from "./listado-productos/listado-
productos.component";

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet, ListadoProductosComponent],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  titulo = 'Tienda Online';
}
```

¿Cómo funciona?

1. El **`ProductoComponent`** tiene un botón donde el usuario puede dar click para emitir un mensaje. Al hacer clic en el botón "Ver Detalle", el mensaje es enviado al servicio `ProductoService` usando el método `emitirDetalleProducto()`.
2. El **`ProductoService`** emite el mensaje a través del **EventEmitter**, el cual actúa como un observable.
3. El **`ListadoProductosComponent`** está suscrito al observable del servicio (`detalleProductoEmitter`). Cuando el servicio emite un nuevo mensaje, este componente lo recibe automáticamente y actualiza su vista para mostrar el mensaje. Cualquier otro componente pudo haber sido el que se suscribe para procesar el mensaje, no necesariamente este componente.

Resultado Final:

Y al presionar el botón de Ver Detalle observamos una alerta con el detalle del Producto seleccionado:



Resumen:

- **`ProductoService`** utiliza un **EventEmitter** para emitir eventos y permite que los componentes se suscriban a esos eventos.
- **`ProductoComponent`** envía el mensaje usando el servicio.
- **`ListadoProductosComponent`** se suscribe al servicio y muestra el mensaje recibido en su vista.
- El **ciclo de detección de cambios** en Angular permite que la vista se actualice automáticamente cuando el `EventEmitter` emite un nuevo valor.

Con este enfoque, puedes tener componentes comunicándose a través de servicios en Angular utilizando Observables y EventEmitter de RxJS.

Saludos!

Ing. Ubaldo Acosta

Fundador de GlobalMentoring.com.mx