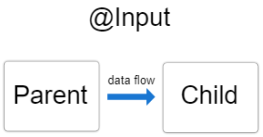
Compartición de datos entre componentes.@Input @Output.

Cuando una página o una determinada vista de una aplicación Angular está compuesto por varios componentes, es común que se tenga que compartir datos entre ellos, por ejemplo, si tenemos que crear un grid (cuadrícula) de cards, donde cada card está formada por una lógica en la que intervengan cálculos con variables, carga de datos, imágenes etc…, nos puede interesar, delegar el renderizado y la lógica de cada card a un componente hijo, en lugar de programar toda la lógica en un solo componente (padre). En este caso puede ser necesario pasar algún dato del padre (que contiene los datos) al hijo, por ejemplo el **id** de una variable contador, elementos de un array, u otros…



Vemos un ejemplo para @Input:

- Creamos un nuevo proyecto `ng new inout --no-standalone` dejamos todo por defecto (enter…)
- Cambiamos al directorio creado …
- Añadimos librería Angular material: `ng add @angular/material`
- Para hacer uso del componente **cards**, vamos a importar su correspondiente módulo en nuestro módulo raíz.

```
import {MatCardModule} from '@angular/material/card';
```

Como vamos a renderizar artículos en forma de cards, vamos a crear un **interface** con los atributos necesarios para el tipo de dato, en este caso sólo va a contener **art** (nombre del artículo) y **precio**. Podríamos haber usado el tipo <**any**> , pero con el **interface** nos aseguramos de que se está cumpliendo con el tipo en cualquier componente y es más fácil la inicialización de las variables (buenas prácticas). Para ello creamos un nuevo archivo **product.interface.ts** con el siguiente contenido.

```
export interface Product {

        art: string;

        precio: number;

}
```

Preparamos el componente 'card' y app.component

- Creamos el componente **card** (hijo) que se va a repetir , controlado por el componente principal (padre) en este caso será **AppComponent**.
- `ng g c card`
- Será necesario importar el interfaz para su uso, nos vamos a **card.component.ts** e importamos :

```
import { Product } from '../product.interface';
```

- Hacemos los mismo que en el paso anterior (importamos interface) para el componente **app.component.ts**
- Aprovechamos y en la clase **app.component.ts** vamos a crear un array de productos de tipo **Product**, el código de la clase quedaría como sigue:

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

import { Product } from './product.interface';

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  standalone: false,

  styleUrls: ['./app.component.css']

})

export class AppComponent {

  title = 'inout';

  productos:Array<Product>=[

                            {art:'Handbag', precio:120},

                            {art:'Dress', precio:60},

                            {art:'Car', precio:50000}

                          ];

}
```

- Queremos listar todos los productos del array, pero usando el componente **card**, para ello podemos empezar reemplazando el código de **app.component.html** por el siguiente :

```
<app-card *ngFor="let item of productos" ></app-card>
```

De momento podemos comprobar que sólo aparece el contenido por defecto de la plantilla del componente **card** : '*card works!*' , una vez por cada elemento del array. Aunque usemos la interpolación **{{item.art}}** en la directiva anterior, seguirá renderizando el valor por defecto, dado que estamos usando el selector/etiqueta del componente que apunta a su plantilla (html) .Sí funcionará con una etiqueta de tipo **<p>**, por ejemplo.

Vemos que los datos están en el componente padre (**app.component**), y nos hace falta pasarlo al componente hijo (**card**), para ello:

- Usaremos el decorador `@Input()` (teniéndolo que importar previamente) en la clase **card.component.ts** para declarar la variable de intercambio (**producto**). Ésta recibirá los datos de su componente padre por cada vuelta del bucle (directiva), renderizando el código de su plantilla . Completando el código de **card.component.ts** quedaría :

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

import { Product } from '../product.interface';

@Component({

  selector: 'app-card',

 standalone: false,

  templateUrl: './card.component.html',

  styleUrls: ['./card.component.css']

})

export class CardComponent implements OnInit {

  @Input() producto:Product={art:'',precio:0};

  url:string='';
```

```
  constructor() {

    }

  ngOnInit(): void {

   this.url=this.producto.art+'.png';

;

  }

}
```

- Aunque sería válido con la línea:

```
this.url='https://source.unsplash.com/random/500x500/?'+this.producto.art
```

La web a limitado su uso gratuito, por lo que es necesario copiar las imágenes a la carpeta **public**, nombrando cada una de ellas según el nombre del artículo, es decir, **Handbag.png** , **Dress.png** y **Car.png**. Así será válido la línea especificada en el código anterior:

```
this.url=this.producto.art+'.png';
```

- Se puede observar que se ha creado la entrada de la variable **producto** en el componente con **@Input**, y se ha inicializado, ya es posible usarlo en la plantilla. Se ha declarado la variable **url**, en este caso no es posible asignar su valor en el **constructor**, ya que el componente no ha sido inicializado totalmente, para ello está el método **ngOnInit**.

- A continuación, usamos la variable **producto** (enlace de propiedad) en la plantilla del componente padre, igualada a cada **item** que devuelve la directiva y que pasaremos al componente hijo , quedaría :

```
<app-card *ngFor="let item of productos" [producto]="item" ></app-card>
```

Sólo nos queda escribir el código en la plantilla del componente **card.component.html**, quedaría (teniendo en cuenta la importación del módulo correspondiente para **card** del Angular Material**)** :

```
<mat-card class="example-card">
```

```
        <mat-card-title-group>
        <mat-card-title>{{producto.art}}</mat-card-title>
        <mat-card-subtitle>Precio: {{producto.precio}} € </mat-card-subtitle>
        <img mat-card-md-image [src]='url'>
        </mat-card-title-group>
</mat-card>
```

Como podemos comprobar, se usa la librería, se interpola los datos de cada artículo o producto, y se asigna mediante enlace de propiedad el valor de la variable **url**, que a su vez se compone en la clase con los datos recibidos desde el componente padre.

Y finalmente darle un poco de estilo en **card.component.css**

```
.example-card {
    max-width: 400px;
    margin-bottom: 8px;
}
```

Quedaría algo como :

Handbag
Precio: 120 €

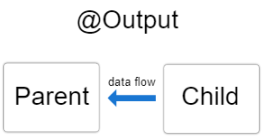
Dress
Precio: 60 €

Car
Precio: 50000 €

A continuación queremos realizar el proceso contrario, intercambiar información del componente hijo al padre.

El componente hijo usa el decorador **`@Output()`** para generar un evento y notificar al componente padre sobre el cambio. Para generar un evento **`@Output()`**, debe tener el tipo **`EventEmitter`**, es una clase de **`@angular/core`** que se utiliza para emitir eventos.

Vemos ejemplo de uso para @Output



Creamos un nuevo componente para añadir nuevas cards, estará compuesto por dos **input** y un **button**, al pulsar el botón se capturará el contenido de los inputs desde el componente hijo y luego se enviarán los datos al componente padre, el cual se encarga de almacenar los nuevos cards, todo ello a través de un evento declarado con **@Output.**

- Creamos el componente `ng g c addcard`

- En la plantilla (html) del componente copiamos el siguiente código (*de momento no se reconocen algunas de las variables*):

```
<label>Nombre:</label><input type="text"  [(ngModel)]="art" ><br>

<label>Precio:</label><input type="number"  [(ngModel)]="precio" ><br>

<input type="button" value="Add" (click)="addNew()">
```

Como se puede ver, enlazamos con las variables que declararemos en la clase y preparamos un evento (click) que llama al método **addNew ()** . Hay que tener en cuenta la importación de **FormsModule** en el módulo raíz, para poder usar **ngModel .**A continuación se crea el método en la clase del componente (**addcard.component.ts**).

El código completo para el componente se muestra a continuación :

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

import { Product } from '../product.interface';

@Component({

  selector: 'app-addcard',

   standalone: false,

  templateUrl: './addcard.component.html',

  styleUrls: ['./addcard.component.css']

})

export class AddcardComponent implements OnInit {

  art:string='';

  precio:number=0;

  @Output () addClicked:EventEmitter<any>= new EventEmitter();

  constructor() { }

  ngOnInit(): void {

  }
```

```
  addNew(): void{

    let card:Product={art:this.art,precio:this.precio}

    this.addClicked.emit(card);

  }

}
```

En el código anterior tenemos :

- Por un lado vamos a crear un evento de salida para que el padre pueda recibir los datos,es decir , este evento **addClicked** va ser reconocido en el componente padre (la sintaxis con **@Output ()** es necesaria aprenderla…), en este caso el flujo de datos es desde el hijo o componente secundario, al padre o componente principal.
- Hemos tenido en cuenta la importación de los elementos **EventEmitter** y **Output** en :

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

- Se crea el método **addNew ( )** , que compone un nuevo **Product** y envía la nueva card a través de una emisión del evento **addClicked**. Este envío lo está esperando el componente padre.
- En **app.component.html** añadimos :

```
<app-addcard (addClicked)="add($event)"></app-addcard>
```

- Dicho evento recibe los datos desde el hijo (nuevo objeto) a través de **$event** , reenviándolo de nuevo en su método **add( )** (a continuación se creará en su clase) .
- Por tanto para **app.component.ts** el código necesario teniendo en cuenta la primera parte sería:

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

import { Product } from './product.interface';

@Component({

  selector: 'app-root',
```

```
  standalone: false,
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'inout';
  productos:Array<Product>=[
                              {art:'Handbag', precio:120},
                              {art:'Dress', precio:60},
                              {art:'Car', precio:50000}
                            ];
  add(newCard:Product):void{
    console.log(newCard);
    this.productos.push(newCard);
  }
}
```

Donde se puede observar que el método **add( )** , añade un nuevo objeto al array. Comprobamos que Angular detecta el cambio en el array **productos** y actualiza la vista automáticamente.

IMPORTANTE: Debemos de copiar una imagen nombrada como **House.png** en **public** para poner a prueba los cambios.

Handbag
Precio: 120 €

Dress
Precio: 60 €

Car
Precio: 50000 €

House
Precio: 120000 €

Nombre: House
Precio: 120000
Add

Finalmente, con la versión **V17** de Angular podemos comunicar componentes de otra forma más rápida a través de un sistema de señales con **signal** y compartiendo un servicio, en el siguiente enlace de Github se muestra un ejemplo de uso entre componentes de igual a igual no relacionados, igualmente se podría usar entre componentes padre<>hijo.

https://github.com/jjjgallego/comunicacomponentessignal.git