DWC

(Desarrollo Web en entorno cliente)

Índice

1.- Anidamiento de componentes

Ya hemos visto que podemos llamar a un componente dentro de otro mediante la inclusión de la etiqueta del selector definida en el decorador del componente. De esta manera estamos incluyendo un componente dentro de otro.

Esto ya lo hemos realizado al principio viendo como un componente padre llamaba a otro componente hijo.

Por ejemplo, podíamos llamar en nuestro componente inicial a un componente que nos mostrará una barra de navegación. Para ello deberíamos crear un componente nuevo llamado header.

```
ng g c Components/header
```

Definimos el html para que nos muestre una barra de herramientas utilizando bootstrap

```
<nav class="navbar navbar-light bg-light">
    <a class="navbar-brand" href="#">
        <h2 class="text-danger">DWEC (Tienda on line con Angular) </h2>
    </a>
    <img src="assets/cart.png" height="60px">
</nav>
```

En principio no necesitamos definir nada en el controlador. Comprobamos el selector del decorador para ese componente.

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

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

}
```

Una vez definido el componente header, lo llamaríamos desde el componente principal, para ello, en el fichero componente app.component.html incluimos la etiqueta del selector, la etiqueta es <app-header>

```
<app-header></app-header>
```

Deberemos importar en ts de AppComponent el componente header

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { HeaderComponent } from './Components/header/header.component';

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

}
```

El resultado sería el siguiente

DWEC (Tienda on line con Angular) 

Este es un caso de anidamiento de componentes sencillo, ya que no existe comunicación entre los componentes, es decir un componente llama a otro, pero entre ellos no se pasan información.

En la mayoría de los casos nos interesara que un componente llame a otro y entre ellos se pasen información.

2.- Comunicación padre-hijo

En este caso un componente padre llamara a un componente hijo pasándole información que el hijo utilizara.

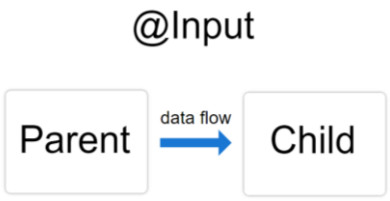
Para pasar datos del padre al hijo se utiliza el property binding, y en la llamada al hijo, dentro de la etiqueta del hijo se pasarán los datos.

La sintaxis es la siguiente:

<etiqueta-hijo [dato]=”valor”></etiqueta-hijo>

Lo que estamos realizando es llamar al componente hijo pasándole un valor en la propiedad dato. Para recibir el dato en el hijo deberemos utilizar el decorador **@Input** que lo que nos permite es recoger el valor asignado en la llamada del padre y poder utilizarlo en el componente hijo.

El esquema es el siguiente



Para utilizar el decorador en el componente hijo deberemos:

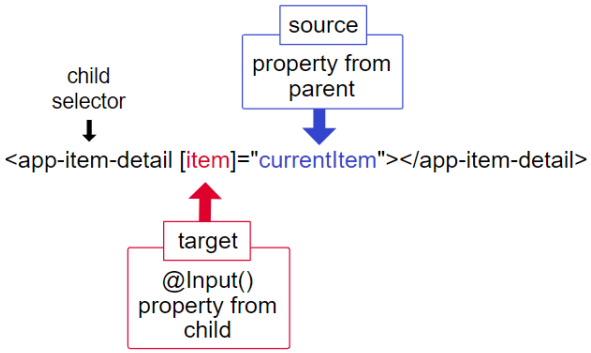
- Importar el elemento Input
- Una vez importado ya podemos utilizar el decorador @Input para recoger el dato que nos pasa el padre, para ello deberemos definir: **@Input() dato!:tipo_dato**

De esta manera estamos definiendo que en el componente hijo va a haber una variable llamada dato que viene desde el padre. El tipo del dato se define y el valor lo asignara @Input a la variable dato con el valor que nos pasa el padre.

En este caso hemos asignado al nombre de la variable dato el símbolo !, con esto lo que estamos haciendo es no tener que darle un valor inicial a la variable dato y evitar el error de no asignar valor inicial.

Desde este momento disponemos en el componente hijo una variable con los datos que nos ha pasado el padre.

Finalmente, el esquema que tendremos es el siguiente



Si quisiéramos pasar del componente padre al hijo un dato numérico con el valor 100 haríamos lo siguiente:

En el html del padre

```
<h3>Componente Padre</h3>

<app-hijo [dato]="100"></app-hijo>
```

En el ts del hijo

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

@Component({
  selector: 'app-hijo',
  standalone: true,
  imports: [],
  templateUrl: './hijo.component.html',
  styleUrl: './hijo.component.css'
})
export class HijoComponent {
  @Input() dato!:number

}
```

En el html del hijo

```
<h3>Componente Hijo</h3>
dato desde el padre: {{dato}}
```

El resultado

Componente Padre

Componente Hijo

dato desde el padre: 100

Podemos modificar la aplicación para que en vez de pasar un valor directamente, podamos pasar el valor de una variable que puede modificar su valor desde el padre y automáticamente se actualizara en el hijo.

Vamos a modificar el ejemplo anterior para poder pasar una variable al hijo e ir modificándola desde el padre.

En el ts del padre

```
import { Component } from '@angular/core';
import { HijoComponent } from '../hijo/hijo.component';

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

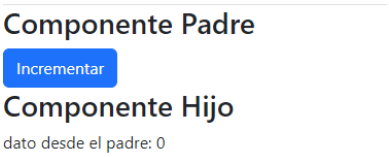
```
export class PadreComponent {
  variable:number=0

  incrementar(){
    this.variable++
  }
}
```

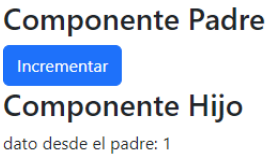
En el hrml del padre

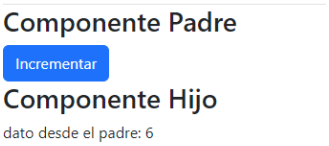
```
<h3>Componente Padre</h3>

<button class="btn btn-primary" (click)="incrementar()">Incrementar</button>

<app-hijo [dato]="variable"></app-hijo>
```

En el hijo no tendríamos que tocar nada.

El resultado





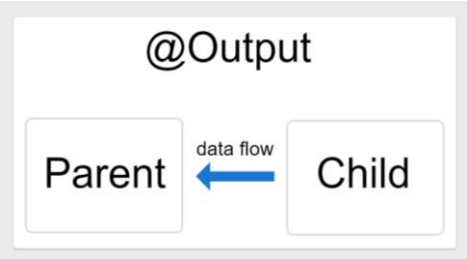


3.- Comunicación hijo-padre

La comunicación padre-hijo es muy sencilla, ya que para poder pintar al hijo lo tiene que llamar el padre incluyendo su etiqueta, en ese momento le pasa el dato mediante un property binding.

La comunicación hijo-padre es más compleja, ya que una vez pintado el hijo perdemos la referencia del padre y en cualquier momento se puede querer pasar información desde el hijo al padre. Es decir, cuando un padre pinta a un hijo le pasa información directamente en ese momento, pero cuando un hijo ya está pintado puede pasarle la información en cualquier momento o incluso nunca.

Para resolver este problema Angular recurre al event binding e incluye el decorador **@Output y EventEmitter** para poder crear eventos personalizados y mandar datos al padre, es decir podremos emitir eventos nuestros (no serán los eventos de HTML) para avisarle al padre que le vamos a pasar un dato desde el hijo.



La sintaxis es la siguiente

<etiqueta-hijo (eventoPersonalizado)=”metodoPadre($event)”> </etiqueta-hijo>

- **eventoPersonalizado** es el evento que utilizaremos desde el hijo para comunicarnos con el padre
- **metodoPadre($event)** es el método del padre que se ejecutará cuando suceda el eventoPersonalizado desde el hijo, en la variable $event estará el dato que pasamos desde el hijo al padre

Para utilizar el decorador en el componente hijo deberemos importar el elemento Output y el elemento EventEmitter

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

Una vez importado ya podemos utilizar el decorador @Output y EventEmitter para crear un evento personalizado. Para ello definimos el decorador de la siguiente forma:

```
@Output() eventoPersonalizado = new EventEmitter<string>();
```

Lo que estamos haciendo es definir un evento personalizado que se llamará eventoPersonalizado que cuando suceda devolverá un valor de tipo string en la variable event. Ahora en cualquier método del controlador del hijo podremos hacer que suceda ese evento, lo que hacemos es emitir el evento.

```
metodoHijo(value:string){
        this.eventoPersonalizado.emit(value);
}
```

En este caso hay un método en el hijo llamado metodoHijo que recibe un parámetro llamado value de tipo string. Dentro de ese método se lanza el evento eventoPersonalizado pasándole como parámetro el dato value de tipo string como se había definido en el decorador @Output

Una vez que se lanza el evento el padre lo recogerá, para ello en la etiqueta del hijo mediante event binding asignaremos al evento que ha sucedido el método del padre que se ejecutara.

```
<etiqueta-hijo (eventoPersonalizado)=”metodoPadre($event)”> </etiqueta-hijo>
```

El metodoPadre quedaría así:

```
metodoPadre(value: string) {
   alert(value);
 }
```

En este caso sacaríamos un alert con el valor pasado desde el hijo

Vamos a modificar el ejemplo anterior para añadir en el hijo la posibilidad de pasarle un mensaje al padre y que el padre lo muestre en su html.

En ts del hijo

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

@Component({
  selector: 'app-hijo',
  standalone: true,
  imports: [],
  templateUrl: './hijo.component.html',
  styleUrl: './hijo.component.css'
})
export class HijoComponent {
  @Input() dato!:number
  @Output() mandaMensaje=new EventEmitter<string>

  mensaje(){
    let mensaje:string | null
    mensaje=prompt("Dime mensaje")
    if (mensaje==null){
      mensaje=""
    }
    this.mandaMensaje.emit(mensaje)
  }
}
```

En html del hijo

```html
<h3>Componente Hijo</h3>
dato desde el padre: {{dato}}
<br>
<button class="btn btn-success" (click)="mensaje()">Mandar mensaje</button>
```

En el html del padre

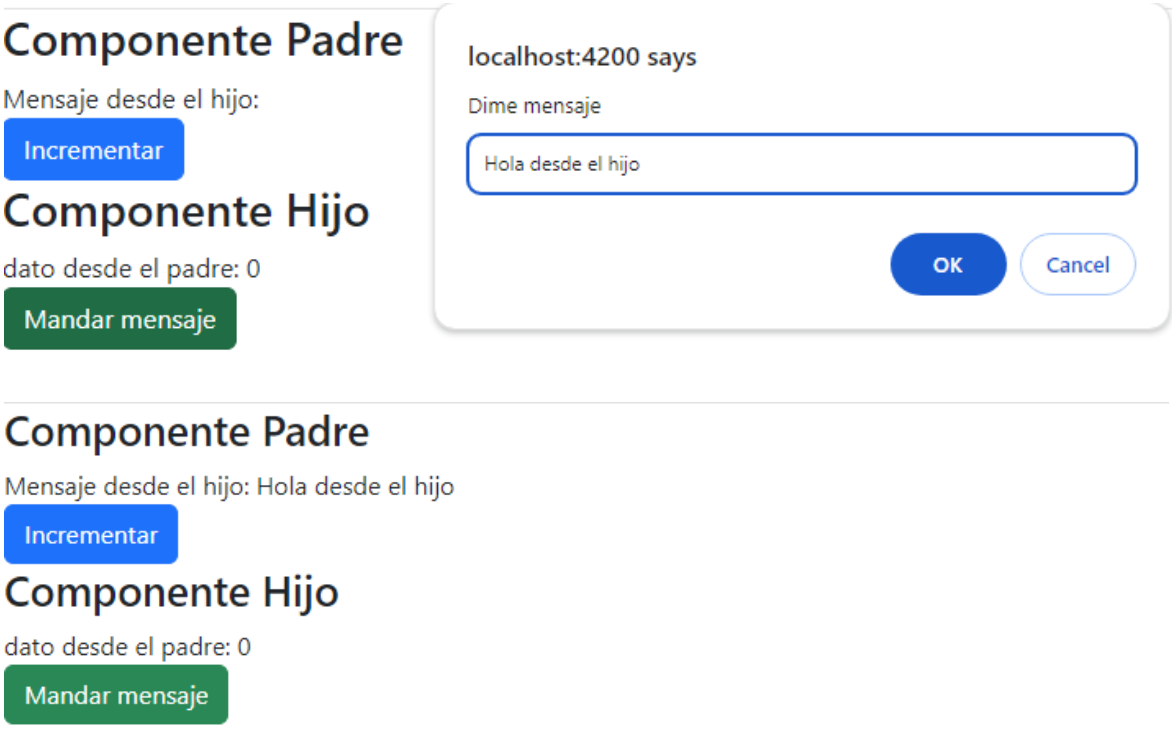
```
<h3>Componente Padre</h3>
Mensaje desde el hijo: {{mensaje}}
<br>

<button class="btn btn-primary" (click)="incrementar()">Incrementar</button>

<app-hijo [dato]="variable"
          (mandaMensaje)="muestraMensaje($event)" >
</app-hijo>
```

En el ts del padre

```
import { Component } from '@angular/core';
import { HijoComponent } from '../hijo/hijo.component';

@Component({
  selector: 'app-padre',
  standalone: true,
  imports: [HijoComponent],
  templateUrl: './padre.component.html',
  styleUrl: './padre.component.css'
})
export class PadreComponent {
  variable:number=0
  mensaje:string=""

  incrementar(){
    this.variable++
  }

  muestraMensaje(dato:string){
    this.mensaje=dato
  }
}
```

Ahora cada vez que pulsemos en el botón “Mandar mensaje” del hijo pediremos por un mensaje con un prompt y **emitiremos** el evento personalizado **“mandaMensaje”** con el string leído en el prompt. El padre cuando detecte que ha sucedido el evento mandaMensaje ejecutara su método muestraMensaje obteniendo el mensaje del objeto event

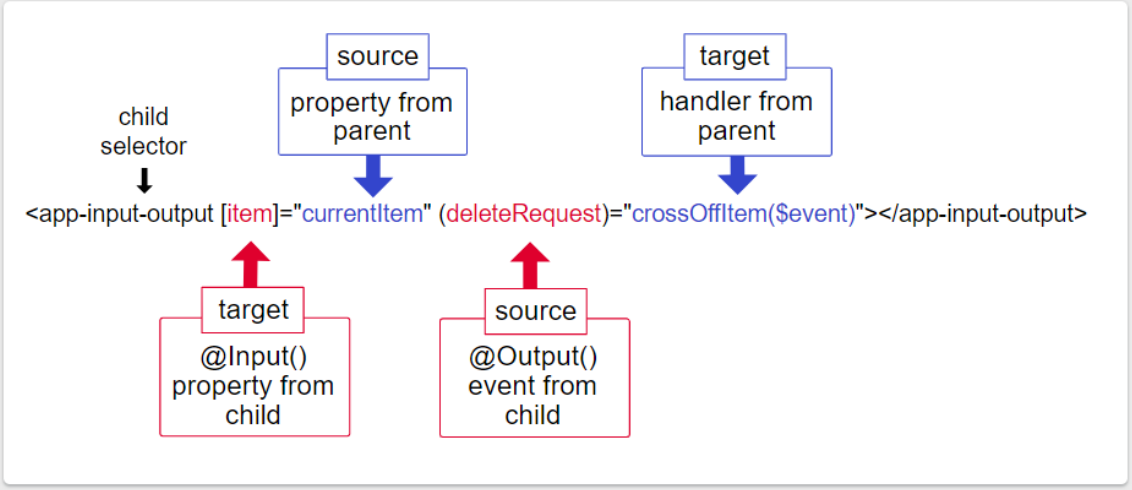
El resultado



4.- Comunicación bidireccional

Es muy habitual utilizar la comunicación bidireccional para mostrar datos de una colección en un componente hijo mediante la directiva @for y desde cada hijo realizar operaciones que pasen datos al padre.

En estos casos utilizaremos los dos métodos vistos anteriormente.



Vamos a utilizar un modelo de comunicación padre-hijo para poder mostrar cada artículo en un card, pero en este caso el card será un componente independiente, será el hijo. En este ejemplo utilizaremos un @for para llamar al componente hijo por cada elemento del array de artículos. En el card pondremos un botón para poder eliminar el artículo, como es un modelo padre-hijo, el hijo no podrá borra directamente, lo que tendrá que hacer es indicarle al padre mediante un evento personalizado que quiere borra ese artículo.

Creamos dos componentes cards (padre) y card(hijo)

```
ng g c /Components/cards
ng g c /Components/card
```

En el html de cards (padre)

```
<h2>Comunicación Padre-Hijo con *ngFor Bidereccional</h2>

<div id="contenedor" class="row row-cols-1 row-cols-md-6 g-4">
  @for(articulo of articulos; track articulo.id){
    <app-card [articulo]="articulo"
              (borrarArticulo)="borrar($event)">
    </app-card>
  }@empty{
    <div class="alert alert-danger" role="alert">
        No hay articulos
    </div>
  }
</div>
```

En el ts de cards (padre)

```
import { Component } from '@angular/core';
import { ARTICULOS, Articulo } from '../../Modelos/articulo';
import { CardComponent } from '../card/card.component';

@Component({
  selector: 'app-cards',
  standalone: true,
  imports: [CardComponent],
  templateUrl: './cards.component.html',
  styleUrl: './cards.component.css'
})
export class CardsComponent {
  articulos:Articulo[]=ARTICULOS

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

}
```

En el html de card (hijo)

```
<div class="card">
  <img src="assets/{{articulo.id}}.jpg" >
  <div class="card-body">
    <h5 class="card-title">{{articulo.nombre}}</h5>
    <p class="card-text">{{articulo.descripcion}}</p>
    <b><p class="card-text text-center">{{articulo.precio}}</p></b>
    <div class="card-text text-center">
      <button (click)="borrar(articulo.id)" class="btn btn-danger">
          Borrar
     </button>
   </div>
  </div>
</div>
```

En el ts de card (hijo)

```
import { Component, EventEmitter, Input, Output } from '@angular/core';
import { Articulo } from '../../Modelos/articulo';

@Component({
  selector: 'app-card',
  standalone: true,
  imports: [],
  templateUrl: './card.component.html',
  styleUrl: './card.component.css'
})
export class CardComponent {
  @Input () articulo!:Articulo
  @Output() borrarArticulo=new EventEmitter<string>

  borrar(id:string){
    this.borrarArticulo.emit(id)
  }
}
```