DWC

(Desarrollo Web en entorno cliente)

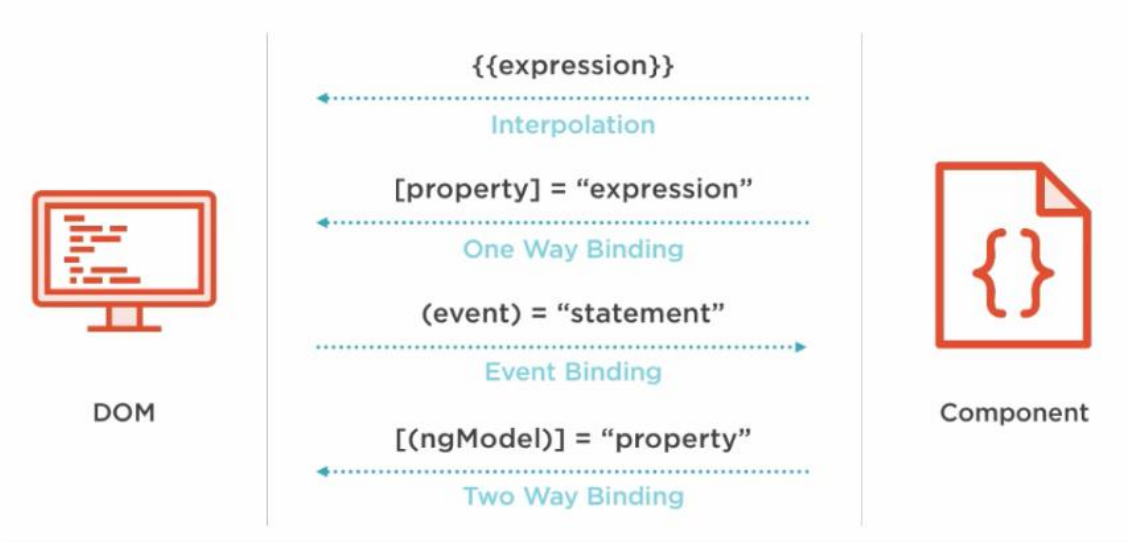
Índice

1.- Data Binding

El Data Binding (enlace de datos) es un concepto fundamental en Angular que establece una conexión dinámica entre los datos de nuestro componente (el controlador) y la vista (la plantilla HTML). Garantiza que los cambios en el modelo se reflejen automáticamente en la vista, y viceversa, manteniendo la interfaz de usuario de nuestra aplicación actualizada y reactiva a las interacciones del usuario.

Los tipos de enlace se pueden agrupar en tres categorías que se distinguen de acuerdo a la dirección del flujo de datos:

- Desde el *modelo-hacia-vista*
- Desde la *vista-hacia-modelo*
- Secuencia Bidireccional: *vista-hacia-modelo-hacia-vista*



2.- Interpolación

La interpolación es el mecanismo para sustituir una expresión por un valor de tipo *string* en la plantilla (*template*).

Cuando **Angular** se encuentra una expresión entre {{}} en la plantilla lo evalúa y trata de **sustituir el contenido por el valor de la propiedad del componente con el mismo nombre**.

Este tipo de vinculación o *binding* es unidireccional, vincula el valor del elemento a la propiedad del componente.

Definimos la variable titulo en el controlador con el valor “Data Binding”

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

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

Hacemos uso de ella en la vista mediante interpolación.

```
<h2>Estamos viendo el {{titulo}}</h2>
```

El resultado es:

Estamos viendo el Data Binding

Al ser un **enlace de datos unidireccional** sólo podemos mostrar datos desde el controlador a la vista, esto quiere decir que **no podemos modificar el valor del controlador desde la vista**.

3.- Property Binding

El **property binding** en Angular ayuda a establecer valores para las propiedades de los elementos o directivas HTML. Este es un enlace unidireccional del controlador a la vista y nos permite establecer propiedades de los elementos del HTML con el valor de una variable del controlador.

Para vincular la propiedad de un elemento, escribimos entre corchetes, **[], l**o que identifica la propiedad como una propiedad del controlador. Una propiedad de destino es la propiedad DOM a la que deseamos asignar un valor.

Para asignar un valor a la propiedad src del elemento de imagen deberíamos utilizar.

En el html

```
<img [src]="imagen">
```

En el controlador

```
export class AppComponent {
  titulo:string="Data Binding"
  imagen:string="../assets/logo.png"
}
```

El resultado

De esta manera estamos asignando a la propiedad src de la imagen la ruta que esta almacenada en la variable imagen que se encuentra en el archivo ts del componente.

Otro ejemplo podría ser determinar el valor de la propiedad **disabled** de un botón mediante una variable del controlador que almacene el valor true o false

En html

```
<button [disabled]="desactivado">Boton</button>
```

En el controlador

```
desactivado:boolean=false
```

En función de la variable el botón estará activo o desactivado. Si aplicamos un timer y modificamos el valor del a variable a los 5 segundos podremos ver cómo cambia el comportamiento del botón.

```
ngOnInit(){
 setTimeout(() => {console.log("Desactivamos boton...");
                   this.desactivado = true
                   }, 5000)
}
```

Hemos añadido en el hook ngOnInit la función SetTimeOut que al pasar 5 segundos cambiara el valor de la propiedad desactivado a true y en ese momento se aplicara el property binding sobre la propiedad disabled del botón cambiando el valor.

También está permitido el uso de expresiones en el property binding, de esta manera podemos asignar condicionales que determinarán el valor de la propiedad que estamos enlazando.
Definimos en el css las clases activado y desactivado

```
.activado{background-color:green}
.desactivado{background-color:red}
```

Definimos en el html el estilo del botón en función del estado.

```
<button [class]="desactivado ? 'desactivado' : 'activado'"
        [disabled]="desactivado">Boton</button>
```

En este caso estamos asignando **la propiedad class** del elemento botón en función del estado del botón, para ello utilizamos una condicional con el operador ternario que asignara la clase “desactivado” si la variable desactivado está a true y la clase activado si la variable desactivado está a false.

Para evitar problemas con el DOM, si hemos creado un proyecto con SSR, añadiremos en el decorador de @Component la propieda host con el valor {ngSkipHydration: 'true'}

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

Las propiedades de estilos y clases se suelen modificar mediante las **directivas ngStyle y ngClass**, que veremos más adelante, pero también se pueden realizar como acabamos de ver desde el property binding de la propiedad class

4.- Event Binding

El **event bindig** permite pasar datos desde la vista al controlador. El enlace se realiza a través de eventos que nos permite escuchar y responder a las acciones del usuario, como pulsaciones de teclas, movimientos del mouse, clicks...

Parea realizar el event binding deberemos incluir el evento que queremos gestionar **mediante los () y sin el on** (como en el addEventListener) y asignarle una función de nuestro controlador, realmente será un método de la clase del componente que habremos implementado en fichero ts

En el html

```
<button (click)="onSave()">Save</button>
```

En el controlador

```
onSave(){
    alert("Has pulsado en grabar...")
  }
```

El mecanismo de asignación para el event binding del evento click del botón es el siguiente:

De esta manera cada vez que hagamos click sobre el botón se ejecutara la función onSave de nuestro componente.

También podemos pasar parámetros a nuestra función del controlador, de hecho, es bastante habitual cuando trabajemos con directivas **ngFor** o comunicación entre componentes

Vamos a pasar un parámetro de tipo string al método onSaveCliente.

En el html

```
<button (click)="onSaveCliente('Pepe')">Save Cliente</button>
```

En el ts

```
onSaveCliente(cliente:string){
    alert("Grabamos a " + cliente)
  }
```

Como parámetro también podemos pasar el **objeto event**, que permitirá pasar a la función con la que trabajemos el objeto event y que se pueda obtener toda la información del evento.

En el html

```
<button (click)="onSaveEvent($event)">Save Event</button>
```

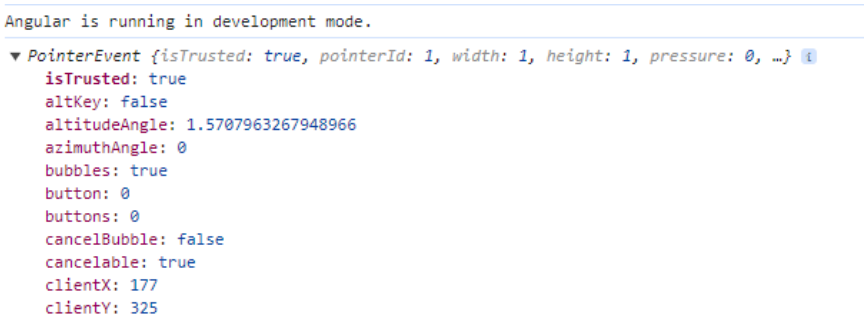
En este caso hemos puesto que al hacer click se llame al método onSaveEvent pasándole el objeto event que se genera

El método onSaveEvent recibirá el objeto event y en nuestro caso lo mostrará por la consola.

En el ts

```
onSaveEvent($event:Event){
    console.log($event)
  }
```

El resultado es el objeto event generado



Two Way Data Binding

El **two way data binding** permite pasar información de forma bidireccional, es decir del controlador a la vista y de la vista al controlador.

Este tipo de data binding combina el property binding y el event binding, por ello utiliza la sintaxis conocida como **banana in a box** y los elementos se incluyen con **[()]**

Se suele utilizar mayoritariamente en los elementos de los formularios, para ello hay que utilizar la directiva **ngModel.** Esta directiva permite mostrar datos de una propiedad del controlador en el elemento del formulario y a su vez, cuando cambia el valor del elemento del formulario se actualiza el valor de la propiedad en el controlador.

Para ello en el elemento del formulario realizaremos lo siguiente:

En el html

```
<input type="text" [(ngModel)]="nombre">
```

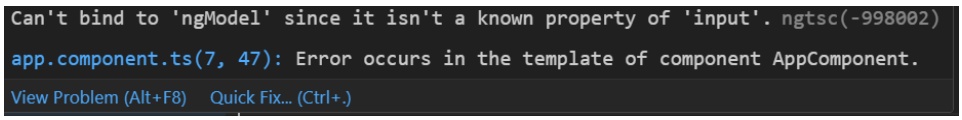
En el ts

```
nombre:string="Pepe Perez"
```

De esta manera estamos enlazando el valor del cuadro de texto a la propiedad nombre del controlador.

Ahora deberíamos ver inicialmente en el cuadro de texto el valor Pepe Perez y cuando vayamos modificando en el cuadro de texto la propiedad nombre ira cambiando.

Esto no va a funcionar y nos mostrara el siguiente error

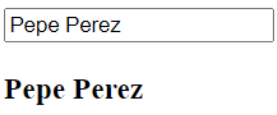


Esto es debido a que **Angular no reconoce la directiva ngModel**.

Para poder utilizar esta directiva en los formularios deberemos importar en nuestro componente el módulo de angular llamado **FormsModule** y añadirlo al array de imports del mismo. De esta manera ya podremos utilizar la directiva ngModel en nuestros formularios.

```
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet, FormsModule],
  host:{ngSkipHydration: 'true'},
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
```

Para probar de verdad los cambios en ambas direcciones vamos interpolar la variable nombre en el html y veremos que cada vez que cambiemos el valor en el cuadro de texto cambiara en la variable que se interpola



Ahora podremos comprobar que:

- Inicialmente en el HTML saldrá el valor de la variable nombre interpolada, es decir Pepe Perez, y además en el cuadro de texto también aparecerá el valor de Pepe Perez por el two way data binding. **En este caso se verá el enlace del controlador a la vista**.
- Al modificar el valor del cuadro de texto, se modificará el valor de la variable y se verá reflejado a través de la interpolación el nuevo valor de la variable. **En este caso se verá el enlace de la vista al controlador**

También podemos añadir un botón que al pulsar sobre el modifique el valor de la variable nombre pidiéndonos con un prompt por el nuevo valor.

En el html

```
<input type="text" [(ngModel)]="nombre">
<h3>{{nombre}}</h3>
<button (click)="cambiarValor()">Cambiar valor</button>
```

En el ts

```
cambiarValor(){
    let nuevoNombre:string | null =prompt("Nuevo valor")
    this.nombre=nuevoNombre==null ? this.nombre : nuevoNombre
  }
```

Ahora cuando pulsemos en el botón pediremos por un dato nuevo y se lo asignaremos a la variable nombre, en ese momento se modificará el valor en el cuadro de texto y en la interpolación.

A tener en cuenta el tipado de la variable nuevoNombre, es string o null, esto es porque la ventana modal del prompt se puede cerrar con el **botón ok** o con el **botón cancelar**, en el primer caso devolverá un string y en el segundo un valor null. Como la variable nombre está definida como string deberemos de comprobar que valor le asignamos para que no sea null