





Angular 17

Tema 5. Componentes

Objetivos

-  Sintaxis completa de componentes
-  Diseño correcto de componentes complejos

Contenidos

- @Component. Definición y diseño de componentes
- Comunicación entre componentes. @Input y @Output
- Ciclo de vida
- Inyección de dependencia
- Signals
- Proyección de contenidos

@Component

Docenas de atributos. Los más habituales:

templateUrl	Ruta al fichero de plantilla.
template	Un texto que directamente define la plantilla.
styleUrl	Ruta al fichero de estilos
styleUrls	Array de rutas que permite indicar varios ficheros.
styles	Array de textos que definen directamente los estilos.
animations	Define animaciones de Angular
standalone	Valor booleano (false por defecto) que indica si el componente evita el uso de módulos.
imports	Sólo para componentes «standalone». Componentes, directivas y pipes que importa el componente actual. En ocasiones hace referencia a módulos para importar elementos «tradicionales».
providers	Los servicios que importa el componente.

Comunicación entre componentes

Diseñarás dos tipos de componentes:

Simple

- Usan un servicio
- Suelen contener otros componentes

Muy simple

- Dibujan lo que les dicen
- No suelen usar servicios
- Se limitan a recibir datos (@Input) del componente padre
- Y a avisarle (@Output) de lo que ha pasado



@Input. Decora campos del componente hijo. Permite modificarlo desde la plantilla del componente padre.

@Output. Decora campos de tipo EventEmitter. Emite eventos, ejecutando un método del componente padre.

```
export class ComponenteHijoComponent {
  @Input() texto='valor por defecto';
  @Output() aviso=new EventEmitter<number>();

  hanHechoClick() {
    this.aviso.emit(42);
  }
}
```

```
<h1>Componente padre</h1>
<componente-hijo [texto]="textoPadre" (aviso)="recibirAccion($event)"></componente-hijo>
```

Comunicación entre componentes

Podemos asignar «alias» a los campos o los eventos:

```
@Input('mensaje') textoRecibido='';  
@Output('cambioDeDatos') botonPulsado=new EventEmitter<string>();  
  
<componente-hijo [mensaje]="textoPadre" (cambioDeDatos)="recibirUnTexto($event)"></componente-hijo>
```

La definición de los campos puede ser compleja:

```
nombreCorregido='no ha llegado nada';  
@Input() set nombre(valor:string) {  
  | this.nombreCorregido=valor.toUpperCase();  
}  
  
@Input() accionRecibida !: (datos:any)=> void;
```

Ciclo de vida de un componente

Etapas por la que pasa un componente: Creación, actualización de datos, renderizado y destrucción. Podemos definir **métodos de enlace** para interceptar cada una de las etapas, preferiblemente mediante interfaces.

Interfaz	Método	Ejecución	Descripción
OnChanges	ngOnChanges	Múltiple	Cambios en los campos @Input
OnInit	ngOnInit	Única	Se ha iniciado el componente (campos, constructor)
DoCheck	ngDoCheck	Múltiple	Cambios en el componente
AfterContentInit	ngAfterContentInit	Única	Se ha iniciado el contenido proyectado
AfterContentChecked	ngAfterContentChecked	Múltiple	Cambios en el contenido proyectado
AfterViewInit	ngAfterViewInit	Única	El componente se ha renderizado y dibujado sus vistas
AfterViewChecked	ngAfterViewChecked	Múltiple	Cambios en las vistas del componente
OnDestroy	ngOnDestroy	Única	El componente va a ser destruido

Inyección de dependencia

Es un patrón de diseño que permite a un objeto recibir automáticamente las dependencias (otros objetos) que necesita para trabajar, generalmente cuando es creado.

En Angular se inyectan servicios a componentes, directivas, pipes u otros servicios.

El **injector** es el encargado de gestionar la inyección de dependencia. Sólo está disponible en el «contexto de inyección»



Dos modos de solicitar la inyección de dependencia.

Constructor:

```
constructor(private uno:ServicioUnoService,
             private dos:ServicioDosService){
}
```

Función Inject:

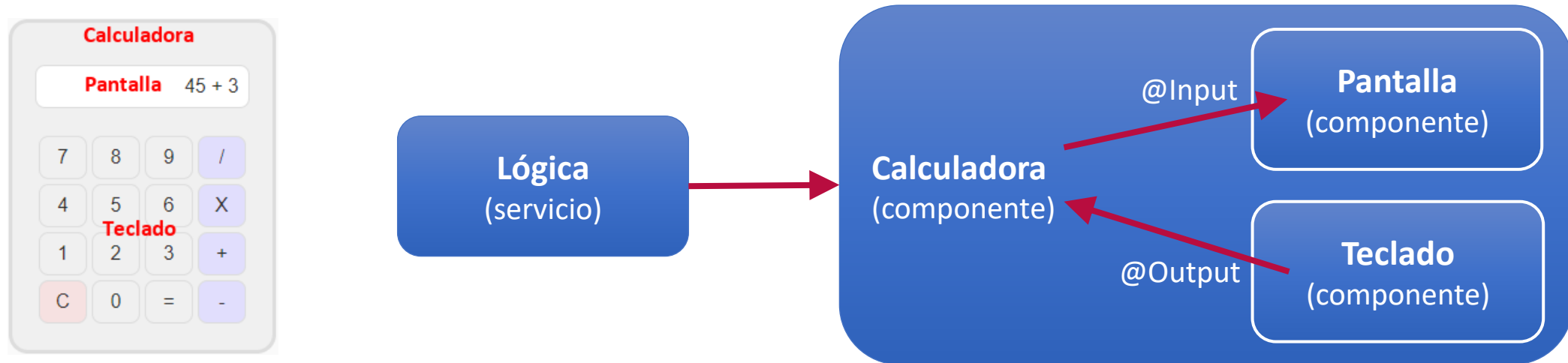
```
private uno=inject(ServicioUnoService);
private dos=inject(ServicioDosService);
```

En general «inject» es lo que se tiende a usar desde Angular 14.

Ejercicio 05 A

El objetivo del ejercicio es practicar la comunicación entre componentes. Mi código usará las interfaces de ciclo de vida e «inject()», como haré a partir de ahora en el resto de ejemplos.

Quiero que crees una calculadora, formada por los componentes «pantalla», «teclado», «calculadora» y el servicio «lógica»:



«Pantalla» se limita a escribir el texto que le pasan desde fuera, y «Teclado» a informar mediante un evento de la tecla que se ha pulsado. «Calculadora» es quien recibe el evento y rellena el texto, basándose en el servicio «Lógica».

El servicio se encarga de toda la lógica. Cuando le pasan la tecla pulsada decide qué hacer con ella, informando al componente de si se está escribiendo el primer operando, el segundo o es el resultado final.

La calculadora no permite escribir decimales ni negativos. Hazlo simple. Y da igual si el servicio no funciona del todo, lo que me interesa son los componentes.

Señales (signals)

Funciones que envuelven un valor (generalmente un campo) para optimizar su actualización en la interfaz de usuario

Permiten que Angular sepa con exactitud qué ha cambiado y que por tanto pueda optimizar la actualización del DOM.

- signal()** Define una señal.


```
valor=signal(42);
```

```
<p>El número vale {{valor()}}</p>
```

```
mensaje=signal("texto de prueba");
```

```
<p>El texto es {{mensaje()}}</p>
```
- set()** Asigna un valor nuevo.


```
this.mensaje.set('El mensaje ha cambiado');
```
- update()** Permite actualizar el valor a partir del actual.


```
this.valor.update(v=>v+1);
```
- computed()** Define una señal calculada a partir de otra. Sólo para tareas muy sencillas.


```
valorDoble=computed(()=>this.valor()*2);
```
- effect()** Se ejecuta cuando el valor de la señal cambia. Sólo está disponible durante el contexto de inyección. Diseñado para logs, actualizaciones personalizadas, localStorage, etc.


```
effect(()=>{
  | this.ejemploDeLog+= 'Valor ha cambiado: ' + this.valor() + '\n';
  | });
```

Proyección de contenido (content projection)

Técnica que permite insertar (proyectar) contenido de la plantilla del componente padre a la del componente hijo

Angular permite definir parte de la plantilla de un componente desde otro gracias a **<ng-content>**:

```
<p>Por favor, complete la pregunta</p>
<div>
  <ng-content></ng-content>
</div>

<div>
  <app-pregunta>
    <label>Nombre de cliente</label><br>
    <input type="text" name="nombre">
  </app-pregunta>
</div>
```

La directiva **<ng-content>** permite el uso de **selectores de CSS** para escoger el contenido a dibujar:

```
<p>Por favor, complete la pregunta</p>
<div>
  <label>
    <ng-content select="[pregunta]"></ng-content>
  </label> <br>
  <ng-content></ng-content>
</div>

<div>
  <app-pregunta>
    <span pregunta>Ingresos</span>
    <input type="radio" name="tipo" value="1"> Bajo<br>
    <input type="radio" name="tipo" value="2"> Medio<br>
    <input type="radio" name="tipo" value="3"> Alto
  </app-pregunta>
</div>
```

¿Qué hemos aprendido?

- Sintaxis completa de los componentes
- Comunicación entre componentes. Diseño
- Ciclo de vida e inyección
- Señales
- Proyección de contenidos

Resumen de comandos

npm

npm install [-g] paquete
npm install
npm uninstall paquete
npm -version

ng (1)

ng version
ng help
ng serve [-o]
ng build
ng new nombre_proyecto
 --routing false
 --skip-tests
 --skip-git
 --no-standalone

ng (2)

ng generate component nombre_componente / ng g c
ng generate service nombre_servicio / ng g s
ng generate module nombre_módulo / ng g m

Ejercicio 05 B. Parte uno

Basándote en los componentes de la calculadora y el servicio asociado del ejercicio 05 A, modifícalos para que los estados de las tres calculadoras no se pierdan al borrar el componente que las dibuja. Para ello necesitarás crear un nuevo servicio que recuerde sus estados durante toda la aplicación:

Ejercicio 05 B

Calculadoras

Ver almacén

Cuadros

5 X 2

7

8

9

/

4

5

6

X

1

2

3

+

C

0

=

-

Calculadora primera

4

7

8

9

/

4

5

6

X

1

2

3

+

C

0

=

-

Calculadora segunda

58 + 226

7

8

9

/

4

5

6

X

1

2

3

+

C

0

=

-

Calculadora tercera

Ejercicio 05 B

Calculadoras

Ver almacén

Cuadros

Listado del estado de las calculadoras

primera

Operando uno:

5

Operando dos:

2

Operación:

X

Estado:

1

segunda

Operando uno:

4

Operando dos:

0

Operación:

No

Estado:

0

tercera

Operando uno:

58

Operando dos:

226

Operación:

+

Estado:

1

Ejercicio 05 B. Parte dos

Quiero que uses la proyección de contenidos para dibujar un componente que simule un «cuadro de diálogo», que acepte cualquier contenido tanto en el «título» como en el «cuerpo» del cuadro.

He usado la directiva «ngClass» dentro del componente para que se oculte o se muestre, y «@Input()» y «@Output» para poder indicárselo desde el componente que lo dibuja:

No pierdas demasiado tiempo con los estilos. En un caso real es más simple aprender a usar Bootstrap o Material para conseguir este tipo de efectos.

