Angular 17

Tema 4. Plantillas y Data Binding

Objetivos

- Diseño de plantillas y sintaxis de bloques de Angular 17
- Conexión entre plantillas y componentes
- Introducción a tareas habituales

Contenidos

- Data Binding
- Sintaxis de control de flujo: @if, @for, @switch
- Introducción a @defer
- Variables de plantilla
- Rutas (introducción)

Data Binding

Comunicación entre la clase de TypeScript y la plantilla de un componente.

Se definen en la plantilla del componente. Cuatro tipos.

Interpolación de textos (string interpolation). Muestra el valor de una expresión de TypeScript en la plantilla.

```
<p>El valor del mensaje es <b>{{mensaje}}</p>
```

Enlace de eventos (event binding). Asocia un evento de HTML con la ejecución de un método de la clase.

```
<button (click)="botonPulsado()">Pulsar</button>
<input type="text" ... (change)="valorCambiado($event)">
```

Enlace de propiedades (property binding). Asigna el valor de una expresión de TypeScript a una propiedad de HTML o una directiva.

```
<input type="text" [value]="mensaje" ... >
```

Enlace bidireccional (two-way binding). Truco de sintaxis para cambiar un valor desde TypeScript y también desde HTML.

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

Sintaxis de control de flujo

La nueva sintaxis reemplaza a las antiguas directivas estructurales. No necesita importaciones, y es mucho más clara.

@if / @else if / @else

```
15  @if (valor < 100) {
16      <p>Es menor de cien</p>
17  }
18  @else if  (valor<=200) {
19      <p>Vale entre cien y doscientos</p>
20  }
21  @else {
22      <p>Vale más de doscientos</p>
23  }
```

@swicth / @case / @default

```
25  @switch (valor) {
26      @case (10) {
27          <p>Vale diez</p>
28      }
29      @case (20) {
30          <p>Vale veinte</p>
31      }
32      @default {
33          <p>No sé cuánto vale</p>
34      }
35  }
```

@for (of;track) / @empty. Variables predefinidas: $index, $first, $last, $even, $odd, $count

```
37  @for(texto of lista; track texto) {
38    <p>el valor {{$index+1}} es {{texto}}</p>
39  }
40  @empty {
41      <p>La lista está vacía</p>
42  }
```

Ejercicio 04 A (Planteamiento)

Este ejercicio es un resumen de todo lo que hemos visto durante el curso, por lo que te llevará más tiempo de lo habitual. Te vas a basar en el código «planteamiento ejercicio 04 A» que tienes disponible en la sección de ficheros del tema actual:

Código	Nombre completo	Gasto medio	Origen
100	Javier Rodríguez Diez	213,45	(A) Nacional
200	Felisa Ramos Pérez	400,00	(A) Nacional
300	Marcos Gonaga Sunaga	1,57	(C) Extracomunitario
400	Lucas Arriaga Ramirez	**2.133,48**	(B) Comunitario
500	Juan López Ruíz	**923,10**	(A) Nacional
600	Mateo Sanz Sanchez	**993,00**	(B) Comunitario
700	Elvira Rodríguez Asteguieta	118,00	(C) Extracomunitario
800	Alejandara Fenaz Carlet	93,00	(C) Extracomunitario

He creado una aplicación con rutas, que dibujan tres componentes. El que voy a explicarte como ejemplo es «listado por tipo», que permite filtrar los clientes en función de su «tipo» (‘A’, ‘B’ o ‘C’). Como ves en la imagen, he implementado el filtro de dos maneras distintas.

Como recordatorio, he añadido pipes, directivas y un tipo compuesto, aunque éste último sólo ha servido para complicar el código.

Ejercicio 04 A (listado por gasto)

Clientes

Listado por tipo Listado por gasto Detalle

Listado de clientes (gasto)

límite mínimo límite máximo

Código	Nombre completo	Gasto medio	Origen
100	Javier Rodríguez Diez	213,45	(A) Nacional
200	Felisa Ramos Pérez	400,00	(A) Nacional
300	Marcos Gonaga Sunaga	1,57	(C) Extracomunitario
400	Lucas Arriaga Ramirez	**2.133,48**	(B) Comunitario
500	Juan López Ruíz	**923,10**	(A) Nacional
600	Mateo Sanz Sanchez	**993,00**	(B) Comunitario
700	Elvira Rodríguez Asteguieta	118,00	(C) Extracomunitario
800	Alejandara Fenaz Carlet	93,00	(C) Extracomunitario

Este componente es similar al anterior, salvo que filtra los clientes en función del gasto. Permite indicar un gasto mínimo, máximo o ambos.

Debe comprobar si el texto introducido se puede convertir a número y corrige el contenido del cuadro de texto si la cantidad mínima es superior a la máxima.

Cuidado, TypeScript convierte textos a números de forma distinta a como lo hace JavaScript. No deberías usar las funciones «parse»:

```
let unTexto='123';
let elValor=Number(unTexto);
//esta función SÓLO comprueba si el número vale "NaN"
if (Number.isNaN(elValor)) console.log('El texto no era una cifra');
else console.log('Se ha convertido correctamente');
```

Ejercicio 04 A (detalle)

Él tercer componente muestra un único cliente por pantalla. Los botones «anterior» y «siguiente» permiten cambiar el cliente actual.

En este caso quiero que sea el componente quien se encargue del gestionar los botones. Al comenzar, el componente solicitará al servicio la lista completa de clientes y después la usará como le parezca, sin volver a molestarlo.

El componente debe mostrar el número de cliente actual y cuántos hay en total. También debe mostrar el mensaje «importante» (con algún estilo aplicado) cuando el gasto sea mayor de 800.

Variables de plantilla

Son identificadores asociados a una etiqueta de HTML. Permiten hacer referencia a la etiqueta desde la plantilla y desde la clase de TypeScript del componente.

Se define con el símbolo «#» en cualquier etiqueta de HTML:

```
<input type="text" #elControl> <br>
El contenido del control es <b>{{elControl.value}}</b><br>
<button (click)="procesar(elControl)">Usar datos</button>
```

El método de TypeScript:

```
procesar(elControl:HTMLInputElement) {
  console.log(elControl.value);
}

procesar(elControl:any) {
  console.log(elControl.value);
}
```

Y la referencia directa desde la clase:

```
@ViewChild('elControl') control!:ElementRef<HTMLInputElement>;
```

@defer. Vistas diferidas

ANGULAR

Bloque diseñado para aplazar la carga y el renderizado de componentes, directivas y pipes

- Define los elementos en bloques de JavaScript independientes
- Sólo para elementos «standalone»
- Bloques auxilares adicionales: @placeholder, @loading, @error
- Disparadores:
  - on (idle, viewport, interaction, hover, immediate, timer)
  - when (condición)
- Permite distinguir entre renderizado y carga (prefetch)

```
@defer (on timer(5s)) {
    <p>Ejemplo de carga diferida</p>
    <app-dos></app-dos>
}
@placeholder {
    <p>El componente no se ha dibujado</p>
}
```

```
@defer (on viewport; when valor > 200) {
    <p>Ejemplo de carga diferida</p>
    <app-dos></app-dos>
}
@loading {
    <p>Cargando...</p>
}
@placeholder {
    <p>El componente no se ha dibujado</p>
}
```

¿Qué hemos aprendido?

- Sintaxis y uso de Data Binding
- Sentencias de control de flujo de Angular 17
- Variables de plantilla
- Carga diferida de Angular 17: @defer
- Introducción a rutas

Resumen de comandos

npm

npm install [-g] paquete

npm install

npm unistall 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 04 B

Quiero gestionar «conceptos de gasto», para una hipotética lista de gastos. Necesito una página en la que pueda ver, añadir o eliminar dichos conceptos:

Conceptos

Código	Concepto de gasto	
10	Kilometraje	Eliminar
20	Dieta comida	Eliminar
30	Dieta cena	Eliminar
40	Dieta alojamiento	Eliminar
		Añadir

```
1  export class Concepto {
2      constructor(
3          public idConcepto:number,
4          public nombre:string
5      ){}
6  }
```

Sólo es necesario que escribas el nombre del concepto, el código lo debe «deducir» el servicio. Utiliza variables de plantilla en vez de «ngModel» o eventos para referirte al cuadro de texto.

Ejercicio 04 B (opcional)

Si lo anterior te ha parecido fácil y no tienes nada mejor que hacer, añade esa lista de gastos:

Listado de gastos

Código	Concepto de gasto	Cantidad	Precio unidad	Total	
100	Kilometraje / 10	120	34	4080	Eliminar
200	Dieta comida / 20	2	9.9	19.8	Eliminar
300	Dieta alojamiento / 40	2	56	112	Eliminar
					Añadir

```
export class Gasto {
    constructor(
        public idGasto:number,
        public idConcepto:number,
        public cantidad:number,
        public precio:number
    ) {}
}
```

Usa «ngModel» para leer el «select» y los cuadros de texto. Crea un segundo servicio para los gastos, no metas todo esto en el servicio de conceptos. Y tendrás que inyectar ambos servicios en el componente, ya que necesitas convertir los «códigos de concepto» a «nombres de concepto», tanto para los textos de la tabla como para el «select».

Como los datos se los inventan los servicios (no tenemos un servidor), cada vez que uses el menú el programa se olvidará de todo lo que hayas añadido. El motivo lo veremos en el tema siguiente, pero si quieres que el ejemplo funcione mejor te adelanto cómo tienes que definir las dependencias de los servicios:

```
@Component({
  ...
  providers: [ConceptosService]
})
export class ConceptosComponent {
```

```
@Component({
  ...
  providers: [ConceptosService, GastosService]
})
export class GastosComponent {
```

```
export const appConfig: ApplicationConfig = {
  providers: [provideRouter(routes),
              ConceptosService,
              GastosService]
};
```