DWC

(Desarrollo Web en entorno cliente)

Índice

1.- Directivas

Cuando realizamos un desarrollo de una aplicación en Angular es muy habitual trabajar constantemente con directivas de Angular y ver que estás llegan hasta en puntos donde no nos lo esperaríamos debido a que son fáciles de leer y sencillas y rápidas de añadir.

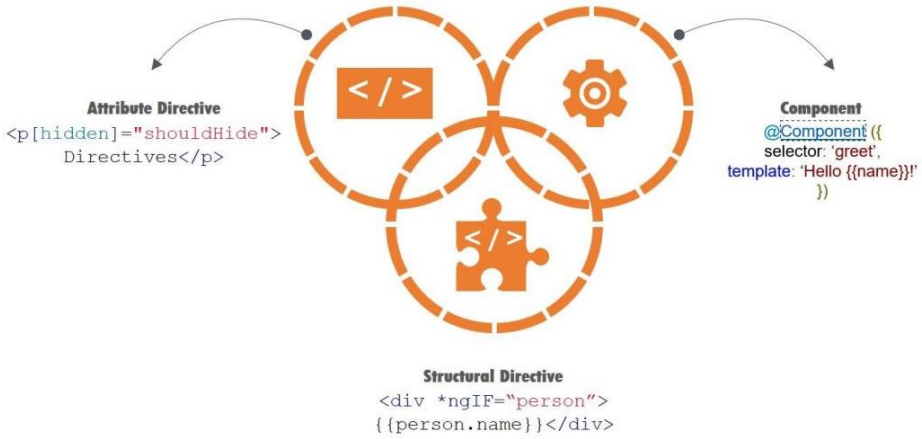
¿Qué son las directivas de Angular?

Las directivas son una serie de elementos que aplicaremos a nuestro código HTML como si de un atributo se tratara con el fin de añadir una nueva funcionalidad a las etiquetas HTML.

Estás directivas, por tanto, les otorgan una nueva funcionalidad y un nuevo comportamiento a los elementos de nuestro template (páginas HTML).

Algunas de las funcionalidades que podemos añadir en función de la directiva que utilicemos nos permitirán entre otras cosas:

- **Modificar la estructura del DOM:** añadiendo, manipulando o eliminado directamente desde la vista HTML
- **Modificar la apariencia general:** modificar clases, o una propiedad en concreto, etc.



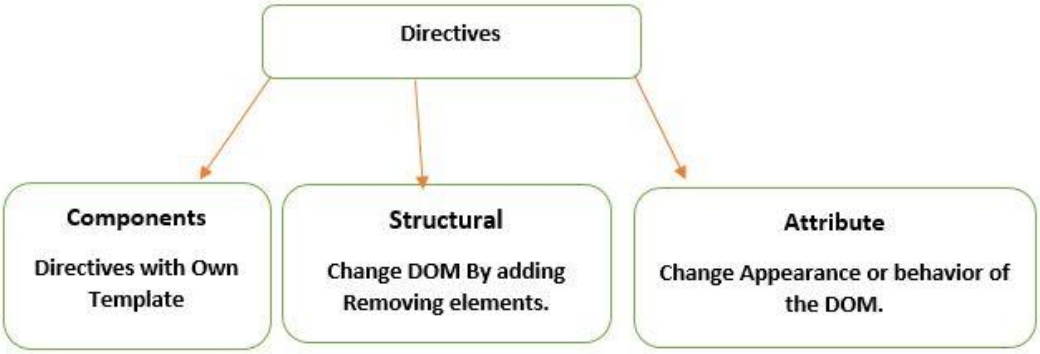
Hay que matizar que las directivas son simplemente instrucciones interpretadas por el compilador (como @Component, pero aplicadas sobre nuestras templates, y más concretamente sobre las etiquetas html) mediante a las cuales le indicamos a Angular que tiene que hacer en ciertos lugares de nuestro código.

Las directivas, por tanto, junto a los componentes **son elementos claves para renderizar (generar) nuestras vistas de nuestros templates**.

2.- Tipos de directivas en Angular

Cada directiva que usamos tiene un nombre, y determina su funcionalidad y puede ser usada, sea en un elemento, atributo, componente o clase.

Las directivas se subdividen en:



2.1 Directivas de Componentes

Son directivas especiales que se utilizan para crear bloques de código reutilizables con su propia plantilla HTML, lógica y estilos. Las Directivas de Componente son directivas con un Template. Los componentes tienen decoradores "**@Component"**, el componente es un decorador @**Directive** que es extendido con características propias de los ***templates***.

Las directivas de componentes se subdividen en:

- **Directivas de Angular (Angular Directives):** @Component, @Module, etc.
- **Directivas personalizadas (Custom Directives):** creadas por los desarrolladores mediante a la instrucción: ng g directives [nombre]

2.2 Directivas de Atributo

Son aquellas que se aplican como atributos a los elementos HTML. Modifican el DOM pero sin ser capaces de crear y destruir el elemento html sobre el que se aplican.

Las directivas de tipo atributo están formadas por:

- **[ngStyle]:** permitirán cambiar las propiedades del elemento HTML seleccionado.
- **[ngClass]:** permitirán agregar clases dinámicamente sobre el elemento HTML seleccionado.
- **[ngModel]:** nos permite realizar el two way data binding

2.3 Directivas Estructurales

Son aquellas que cambian el DOM, y el elemento HTML al que se le apliquen. Y nos permiten analizar una condición y en función de si dicha condición es verdadera o falsa nos va a permitir mostrar / ocultar elementos del DOM. También nos van a permitir realizar iteraciones modificando elementos del DOM

Las directivas estructurales están formadas por:

- ***ngIf o @if:** permite evaluar de forma condicional una condición y dependiendo del resultado normalmente se suele utilizar con la finalidad de mostrar u ocultar una información.
- ***ngSwitch o @switch:** permite evaluar una expresión mediante a una sucesión de condiciones principalmente se utiliza cunado existen casuísticas múltiples y solo queremos ejecutar una en específico o todas a partir de dicha condición.
- ***ngFor o @for:** permite iterar una array, objeto, etc. normalmente con el fin de insertar cada uno de los elementos contenidos en su interior.

A partir de la versión 17 se incluye una nueva sintaxis para este tipo de directivas que ha cambiado es el control de flujo para las plantillas. Ahora tenemos una estructura más parecida a JavaScript que nos permitirá tener esta misma funcionalidad en nuestras plantillas, esta es una estructura con sintaxis de block. Con esta estructura, el compilador de angular transforma esta sintaxis en instrucciones de JavaScript.

2.3.1.- Directiva *ngIf y @if

La directiva estructural *ngIf y @If permiten mostrar u ocultar elementos del DOM. Esta directiva analiza una condición y en función de la condición el elemento se mostrará o no se mostrará. Sería equivalente a la condicional simple o doble de JavaScript

2.3.1.1- Directiva *ngIf

Hasta la versión 17 **la directiva *ngIf iba dentro de un elemento del DOM** o de un contenedor de Angular, no teníamos que importar nada. Ahora si queremos seguir utilizando esta sintaxis, al trabajar con componentes standalone, al poner *ngIf en la plantilla nos aparecerá este error.

```
The `*ngIf` directive was used in the template, but neither the `NgIf` directive nor the `CommonModule` was imported. Use Angular's built-in control flow @if or make sure that either the `NgIf` directive or the `CommonModule` is included in the `@Component.imports` array of this component. ngtsc(-998103)
```

Para solucionarlo deberemos importar en el controlador el módulo **CommonModule**

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

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

}
```

Una vez importado la directiva ya es reconocida y no da problemas

La directiva *ngIf funciona de la siguiente forma:

```
<elemento *ngIf=”condición”>…</elemento>
```

Vamos a mostrar u ocultar un div en función de una variable que nos indique si el usuario está logueado. La variable se definirá en el controlador con un valor inicial de true

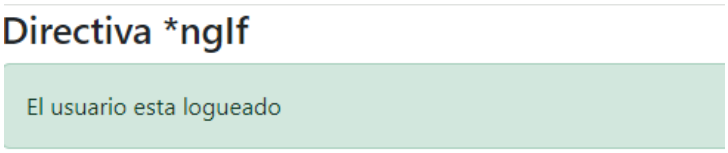
```
isLogged:boolean=true
```

En la plantilla definimos el html

```
<h2>Directiva *ngIf</h2>
<!-- Ejemplo simple con condición -->

<div class="alert alert-danger" role="alert" *ngIf="isLogged">
    A simple primary alert—check it out!
</div>
```

El resultado será que el div se visualizará



Si modificamos la condición a false el div ya no se verá

Para poder evaluar la condición con acciones para true o false deberemos utilizar else dentro de la directiva llamando a un **ng-template.**

La estructura es:

```
<elemento *ngIf=”condición; else plantilla”>…</elemento>
<ng-template #plantilla>
        HTML
</ng-template>
```

En Angular, ng-template es una directiva que nos permite definir un bloque de contenido HTML que no se renderiza por defecto. Funciona como una plantilla dentro de una plantilla, proporcionando una forma de almacenar fragmentos reutilizables de interfaz de usuario (UI) que se pueden insertar dinámicamente en la vista de nuestra aplicación según condiciones específicas.

Modificamos el html

```
<div class="alert alert-success" role="alert" *ngIf="isLogged; else
noLogged">
    El usuario esta logueado
  </div>

<ng-template #noLogged>
  <div class="alert alert-danger" role="alert">
      El usuario no esta logueado
  </div>
</ng-template>
```

Ahora en función del valor de isLogged se mostrará un div rojo de no logueado o un div verde de logueado.

Directiva *ngIf

El usuario no esta logueado

Podemos incluir dos botones para modificar el valor de la variable isLogged y comprobar cómo se seguirá realizando el renderizado condicional

En el HTML

```
<button class="btn btn-primary" (click)="logIn()">Login</button>
<button class="btn btn-danger"  (click)="logOut()">No Login</button>
```

En el ts

```
logIn(){
    alert("Simulamos que está logueado...")
    this.isLogged=true
  }

  logOut(){
    alert("Simulamos que no está logueado...")
    this.isLogged=false
  }
```

Ahora cada vez que pulsemos en un botón u otro se modificará el valor de la variable y se mostrará el div correspondiente

También se puede utilizar la sintaxis **[ngIf]** en vez de *ngIf. Es lo que se conoce como azúcar sintáctico

2.3.1.2- Directiva @if

Ya hemos visto la estructura *ngIf para el renderizado condicional en nuestras plantillas, **a partir de la versión 17 tenemos una estructura más parecida a JavaScript** que nos permitirá tener esta misma funcionalidad en nuestras plantillas, esta es una estructura **con sintaxis de block**.

Ahora se define la estructura condicional similar a JavaScript y dentro de ella se define el HTML a renderizar.

```
@if(condición){
      HTML (true)
}@else{
      HTML (false)
}
```

El ejemplo anterior quedaría así:

```
@if (isLogged) {
  <div class="alert alert-success" role="alert" *ngIf="isLogged">
    El usuario esta logueado
</div>
}@else {
  <div class="alert alert-danger" role="alert">
    El usuario no esta logueado
</div>
```

El resultado será el mismo, lo que cambia es la sintaxis. Además, no tenemos que usar un ng-template

2.3.2- Directiva *ngSwitch y @switch

La directiva estructural *ngSwitch y @switch permiten mostrar u ocultar elementos del DOM. Esta directiva analiza un valor y en función del valor del elemento se mostrará un template u otro. Además, disponemos de un valor por defecto para cuando no se cumpla ningún valor de los anteriores.

Sería equivalente a la condicional simple o doble de JavaScript

2.3.2.1- Directiva *ngSwitch

Hasta la versión 17 **la directiva *ngSwitch iba dentro de un elemento del DOM** o de un contenedor de Angular, no teníamos que importar nada. Al igual que sucede con el *ngIf deberemos importar el CommonModule.

```
<elemento [ngSwitch]=”variable”>
    <elemento *ngSwitchCase=”valor1”> HTML 1</elemento>
    <elemento *ngSwitchCase=”valor2”> HTML 2</elemento>
    <elemento *ngSwitchCase=”valorn”> HTML n</elemento>
    <elemento *ngSwitchDefault”> HTML otro valor</elemento>
</elemento>
```

Vamos a ver un ejemplo en el que en función del valor de la variable numero mostremos el numero en texto, esto lo haremos para los valores 1,2 y 3, para el resto de valores mostraremos un mensaje

```
<div [ngSwitch]="numero">
 <div *ngSwitchCase="1">Uno</div>
 <div *ngSwitchCase="2">Dos</div>
 <div *ngSwitchCase="3">Tres</div>
 <div *ngSwitchDefault class="alert alert-danger">Fuera de [1-3]</div>
 </div>
```

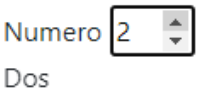
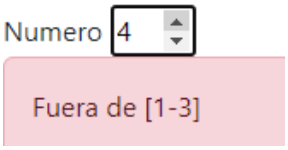
Si modificamos el valor de la variable número se mostrará el valor numérico en texto o el valor “Otro valor fuera de [1-3]”

Para probarlo modificando los valores de la variable número, podemos añadir un cuadro de texto con ngModel a la variable número, así cada vez que cambiemos el valor del cuadro de texto cambiara el resultado a mostrar.

```
Numero <input type="number" size="3" [(ngModel)]="numero" id="number">

<div [ngSwitch]="numero">
 <div *ngSwitchCase="1">Uno</div>
 <div *ngSwitchCase="2">Dos</div>
 <div *ngSwitchCase="3">Tres</div>
 <div *ngSwitchDefault class="alert alert-danger">Fuera de [1-3]</div>
</div>
```

Ahora cada vez que cambiemos el valor del input en el div se mostrara el número correspondiente.

 Valores [1-3]	 Valores distintos [1-3]

2.3.2.2- Directiva @switch

A partir de la versión 17 se modifica esta estructura para que sea más similar a la estructura de javaScript.

```
@switch(variable){
  @case(valor1){
    HTML 1
  }
  @case(valor2){
    HTML 2
  }
  @case(valorn){
    HTML n
  }
  @default{
    HTML otro valor
  }
}
```

El ejemplo anterior quedaría así

```
@switch(numero){
 @case(1){
  <div>Uno</div>
  }
 @case(2){
  <div>Dos</div>
  }
  @case(3){
   <div>Tres</div>
  }
  @default{
   <div class="alert alert-danger">Fuera de [1-3]</div>
  }
}
```

2.3.3.- Directiva *ngFor y @for

Son directivas estructurales en Angular que nos permite iterar sobre una colección de datos y renderizar un bloque de HTML para cada elemento de la colección. Son los equivalentes a los bucles de JavaScript.

2.3.3.1- Directiva *ngFor

En versiones anteriores a la 17 se incluye la estructura *ngFor dentro de un elemento de HTML. Deberemos importar el CommonModule

Funcionamiento:

1. **Se define una variable** que contiene la colección de datos a iterar.
2. **Se aplica la directiva ngFor** a un elemento HTML.
3. **Se indica la variable** a iterar dentro de la directiva *ngFor*.
4. **Se define el bloque de HTML** que se renderizará para cada elemento de la colección.

La estructura es:

```
<elemento *ngFor=”let item of items”>
        HTML {{item}}
</elemento>
```

Vamos trabajar con el array de artículos de la clase Articulo que utilizamos en el tema anterior y vamos mostrar el nombre de cada artículo en una lista. Para ello deberemos importar la lista de datos a nuestro componente.

En el ts

```
import { Component } from '@angular/core';
import { ARTICULOS } from '../Modelos/articulo';

@Component({
  selector: 'app-ejemplo-for',
  standalone: true,
  imports: [],
  templateUrl: './ejemplo-for.component.html',
  styleUrl: './ejemplo-for.component.css'
})
export class EjemploForComponent {
  articulos=ARTICULOS
}
```

En el HTML

```
<ul>
    <li *ngFor="let articulo of articulos">
        {{ articulo.nombre }}
    </li>
</ul>
```

Resultado

Directiva *ngFor

- Galaxy A32
- Oppo A94
- Galaxy S22
- Apple iPhone
- Galaxy Z Flip4
- Note 11
- Realme 9
- Asus Zen
- HP Pavillion
- MacBook
- Xiaomi P1E

Se nos presenta un problema para cuando queremos controlar el caso de que el array de datos este vacío. Si utilizamos el array articulos2 que este vacío.

```
articulos2:Articulo[]=[]
```

Realizamos lo mismo, pero con el array articulos2

```
<ul>
 <li *ngFor="let articulo of articulos2">
    {{ articulo.nombre }}
 </li>
</ul>
```

El resultado sería

Ejemplo con *ngFor y array vacio

No se vería nada, porque como no hay elementos no se muestra ningún li. Sería interesante que en este caso se nos mostrara algún mensaje de advertencia.
La solución para poder controlar este caso sería utilizar un *ngIf dentro del elemento li.

```
<ul>
  <li *ngFor="let articulo of articulos2" *ngIf="articulos2.length==0">
    {{ articulo.nombre }}
  </li>
</ul>
```

Esto nos devolverá un error

```
Can't have multiple template bindings on one element. Use only one
attribute prefixed with * ngtsc(-995002)

ejemplo-for.component.ts(12, 36): Error occurs in the template of
component EjemploForComponent.
```

Este error es debido a que Angular sólo permite utilizar una directiva estructural por elemento. Para evitar este problema Angular proporciona **ng-container**

En Angular, ng-container es un elemento especial que funciona como contenedor para otros elementos pero no introduce ninguna estructura HTML adicional en el DOM (Modelo de Objeto del Documento). Esto significa que no aparece como una etiqueta separada en el HTML renderizado final. **Es un elemento de sintaxis especial reconocido por Angular**.

En nuestro caso podríamos utilizar:

```
<ul>
    <ng-container *ngFor="let articulo of articulos2">
            <li>{{ articulo.nombre }}</li>
    </ng-container>
    <ng-container *ngIf="articulos2.length === 0">
      <div class="alert alert-danger">No hay articulos</div>
    </ng-container>
</ul>
```

El resultado sería

Ejemplo con *ngFor y array vacio con ng-container

No hay articulos

2.3.3.2- Directiva @for

A partir de la versión 17 se incluye la estructura @for que incluye algunos cambios. Nos encontramos con una estructura similar a JavaScript, que, además, incluye la propiedad obligatoria **track**. Con esto se tiene un rendimiento superior en la diferenciación de elementos, con esta propiedad Angular rastrea cada elemento ante cualquier cambio, así que identificamos de forma única a cada uno de los elementos usando la propiedad track

También se incluye **@empty** para poder comprobar si el array está vacío y en ese caso poder mostrar el HTML adecuado

```
@for (item of items; track item.id) {
  <elemento> {{ item }} </elemento>
}
@empty {
   HTML ítems vacio
}
```

El ejemplo anterior quedaría

```
<ul>
    @for(articulo of articulos; track articulo.id){
        <li> {{articulo.nombre}}</li>
    }
    @empty {
        <div class="alert alert-danger">No hay articulos</div>
    }
</ul>
```

Ahora el resultado sería.

Ejemplo con @for

- Galaxy A32
- Oppo A94
- Galaxy S22
- Apple iPhone
- Galaxy Z Flip4
- Note 11
- Realme 9
- Asus Zen
- HP Pavillion
- MacBook
- Xiaomi P1E

Ejemplo con @for y array vacio

No hay articulos