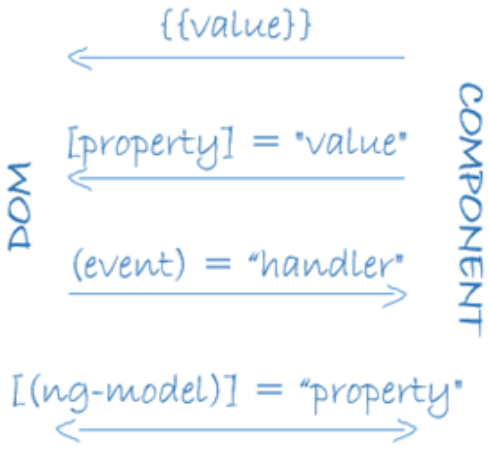
4.- Conceptos avanzados de componentes

4.1.- Sintáxis binding

Data-Binding es una mecánica para coordinar que vé el usuario, especialmente con los valores de la aplicación. Angular posee diferentes tipos de data-binding, se podrían agrupar en tres categorías:

- *Source-to-view*. Desde el componente a la vista.
- *View-to-source*. Desde la vista al componente.
- *Two-way*. Doble binding.



Categoría	Tipo	Sintáxis
Source-to-view	Interpolacion Propiedades Atributos 1. Clases Estilos	{{expression}} [target]=”expression” bind-target=”expression”
View-to-source	Eventos	(target)=”statement” on-target=”statement”
Two-way		[(target)]=”expression” bindon-target=”expression”

Ejemplos de uso

Tipo	Target	Ejemplo
Propiedad	Elemento Componente Directiva	<img [src]="heroImageUrl"> <app-hero-detail [hero]="currentHero"></app-hero-detail> <div [ngClass]="{'special': isSpecial}">
Evento	Elemento Componente Directiva	<button (click)="onSave()">Save</button> <app-hero-detail (deleteRequest)="deleteHero()"></app-hero-detail> <div (myClick)="clicked=$event" clickable>click me
Two-way	Eventos Propiedades	<input [(ngModel)]="name">
Atributos	Atributos	<button [attr.aria-label]="help">help</button>
Clases	Clases	<div [class.special]="isSpecial">Special
Estilo	Estilo	<button [style.color]="isSpecial ? 'red' : 'green'">

Posible error a tener en cuenta

Cuando intentamos modificar una clase, atributo o estilo Angular nos puede bloquear ese trozo de código por seguridad. Para `saltarnos' esa seguridad y poder bindear información se debe de importar la clase *DomSanitizer* y utilizarla como en el siguiente ejemplo:

Elemento en el HTML

```
[style.background-color]="data.color"
```

Elemento en el componente

```
this.sanitization.bypassSecurityTrustStyle('#666');
```

De esta forma le estamos diciendo a Angular que el código que queremos inyectar no es peligroso.

4.2.- Two-way binding [(...)]

Two-way binding hace dos cosas:

- Colecciona o captura el valor de una variable.
- Permanece ‘escuchando’ por los cambios de esa variable.

Size.component.ts

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

@Component({
  selector: 'app-sizer',
  templateUrl: './sizer.component.html',
  styleUrls: ['./sizer.component.css']
})
export class SizerComponent {


  @Input()  size: number | string;
  @Output() sizeChange = new EventEmitter<number>();

  dec() { this.resize(-1); }
  inc() { this.resize(+1); }

  resize(delta: number) {
    this.size = Math.min(40, Math.max(8, +this.size + delta));
    this.sizeChange.emit(this.size);
  }

}
```

app.component.html

```
<app-sizer [(size)]="fontSizePx"></app-sizer>
<div [style.font-size.px]="fontSizePx">Resizable Text</div>
```

Mismo ejemplo con diferente opción:

```
<app-sizer [size]="fontSizePx" (sizeChange)="fontSizePx=$event"></app-sizer>
```

4.3.- NgClass y NgStyle

NgClass

Con NgClass podremos añadir o eliminar todas las clases para CSS que queramos y actualizar nuestro diseño sin problema con esta directiva.

```
<!-- toggle the "special" class on/off with a property -->
<div [ngClass]="isSpecial ? 'special' : """>This div is special</div>
```

Con varias clases a la vez:

```
currentClasses: {};
setCurrentClasses() {
  // CSS classes: added/removed per current state of component properties
  this.currentClasses =  {
    'saveable': this.canSave,
    'modified': !this.isUnchanged,
    'special':  this.isSpecial
  };
}
```

```
<div [ngClass]="currentClasses">This div is initially saveable, unchanged, and special.</div>
```

La directiva NgClass coge cada una de esas variables y si son true las coloca.

NgStyle

Al igual que NgClass, NgStyle añade o elimina estilos.

```
<div [style.font-size]="isSpecial ? 'x-large' : 'smaller'">
  This div is x-large or smaller.
</div>
```

Usando NgStyle:

```
currentStyles: {};
setCurrentStyles() {
  // CSS styles: set per current state of component properties
  this.currentStyles = {
    'font-style':  this.canSave      ? 'italic' : 'normal',
    'font-weight': !this.isUnchanged ? 'bold'   : 'normal',
    'font-size':   this.isSpecial    ? '24px'   : '12px'
  };
}
```

```
<div [ngStyle]="currentStyles">
  This div is initially italic, normal weight, and extra large (24px).
</div>
```

Lo que hace NgStyle es implementar un valor u otro dependiendo del booleano.

4.4.- [(ngModel)]: Two-way binding

Esta directiva nos permite mostrar el valor de una variable o de un objeto y a la vez actualizar su contenido. Su uso más frecuente es en la etiqueta <input>

```
<label for="example-ngModel">[(ngModel)]:</label>
<input [(ngModel)]="currentItem.name" id="example-ngModel">
```

Es importante no olvidar importar la clase *FormsModule* en el módulo principal o submodulo donde estemos usando esta directiva para que no de fallo el navegador.

Un ejemplo de esta directiva pero sin usarla sería:

```
<label for="example-change">(ngModelChange)="...name=$event":</label>
<input [ngModel]="currentItem.name" (ngModelChange)="currentItem.name=$event"
id="example-change">
```

4.5.- NgSwitch

```
<div [ngSwitch]="currentItem.feature">
  <app-stout-item    *ngSwitchCase="'stout'"    [item]="currentItem"></app-stout-item>
  <app-device-item   *ngSwitchCase="'slim'"     [item]="currentItem"></app-device-item>
  <app-lost-item     *ngSwitchCase="'vintage'"  [item]="currentItem"></app-lost-item>
  <app-best-item     *ngSwitchCase="'bright'"   [item]="currentItem"></app-best-item>
<!-- . . . -->
  <app-unknown-item  *ngSwitchDefault           [item]="currentItem"></app-unknown-item>
</div>
```

Al igual que la condición *switch() case* en ciertos lenguajes mostraremos un elemento según el resultado de una variable. Esta directiva añade o elimina el componente según su valor como se puede ver en el ejemplo anterior.

Esta directiva trabaja tanto con componentes como elementos nativos.

```
<div *ngSwitchCase="'bright'"> Are you as bright as {{currentItem.name}}?</div>
```

4.6.- Variable #var para referenciar elementos en una plantilla

Usaremos el símbolo # para declarar una variable en una plantilla. Esta lógica funciona del mismo modo que declarar una variable en en un componente y después trabajar con ella. En el siguiente ejemplo podemos ver como se declara una variable y luego trabajamos con ella:

```
<input #phone placeholder="phone number" />

<!-- lots of other elements -->

<!-- phone refers to the input element; pass its `value` to an event handler -->
<button (click)="callPhone(phone.value)">Call</button>
```

El ejemplo más usual para esta lógica es su uso en formularios:

```
<form #itemForm="ngForm" (ngSubmit)="onSubmit(itemForm)">
  <label for="name"
    >Name <input class="form-control" name="name" ngModel required />
  </label>
  <button type="submit">Submit</button>
</form>

<div [hidden]="!itemForm.form.valid">
  <p>{{ submitMessage }}</p>
</div>
```

4.7.- Operaciones especiales

4.7.1.- Pipes (|)

El valor de cualquier variable a veces necesita ser transformada para mostrarla en la plantilla, por ejemplo, un texto en mayúsculas o filtrar una lista. Para casos como estos y más tenemos los *Pipes*, son simples funciones que aceptan un valor y lo devuelve transformado.

```
<p>Title through uppercase pipe: {{title | uppercase}}</p>
```

Podemos también aplicar varios filtros:

```
<!-- convert title to uppercase, then to lowercase -->
<p>Title through a pipe chain: {{title | uppercase | lowercase}}</p>
```

Aplicar parámetros:

```
<!-- pipe with configuration argument => "February 25, 1980" -->
<p>Manufacture date with date format pipe: {{item.manufactureDate | date:'longDate'}}</p>
```

O mostrar la información de un Json:

```
<p>Item json pipe: {{item | json}}</p>
```

Para saber más acerca de los *Pipes* pincha aquí.

4.7.2.- El operador (?)

```
<p>The item name is: {{item?.name}}</p>
```

Esta opción se utiliza para variables que son usadas en la plantilla pero que esperan respuesta de un servidor para ser inicializadas por lo que durante un breve momento son *null* o *undefined* y dan un error en el navegador. Dicho error no es grave y la aplicación continua su ejecución pero en algunos casos sí que hace que falle la plantilla y no se muestre la información.

Con este operador hacemos que la plantilla se renderice y una vez que se devuelva el valor del servidor mostrarlo.

4.8.- Tipos de eventos

Algunos de los eventos más usados en Angular son:

click	blur	change
focus	keypress	submit

En la siguiente web se pueden ver una lista completa de todos los eventos:

https://www.w3schools.com/jsref/dom_obj_event.asp