Elementos Básicos.

- **Módulos:** Permiten organizar y dividir la aplicación en funcionalidades más pequeñas.
- **Componentes:** Partes específicas de la interfaz de usuario.
- **Decoradores:** Se utilizan para definir y configurar componentes, servicios y otros elementos: @Component, @NgModule, @Injectable, ...
- **Directivas:** Extienden el comportamiento de los elementos HTML.
- **Tuberías / Pipes:** Permite transformar datos en la vista antes de mostrarlos. Pueden usarse para formatear números, fechas, cadenas de texto: date, uppercase, lowercase,slice ...
- **Servicios:** Objetos reutilizables que contienen la lógica de negocio.
- **Rutas**: sistema de enrutamiento para navegar entre las vistas.

Módulos

Toda aplicación de Angular tiene al menos un módulo de Angular, el módulo principal (o root module) llamado AppModule (App.module.ts). Pero podemos dividir la aplicación en varios módulos para organizar y modularizar el código.

¿Cómo creamos un módulo?

ng generate module mi_modulo
o también:
ng g m mi_modulo

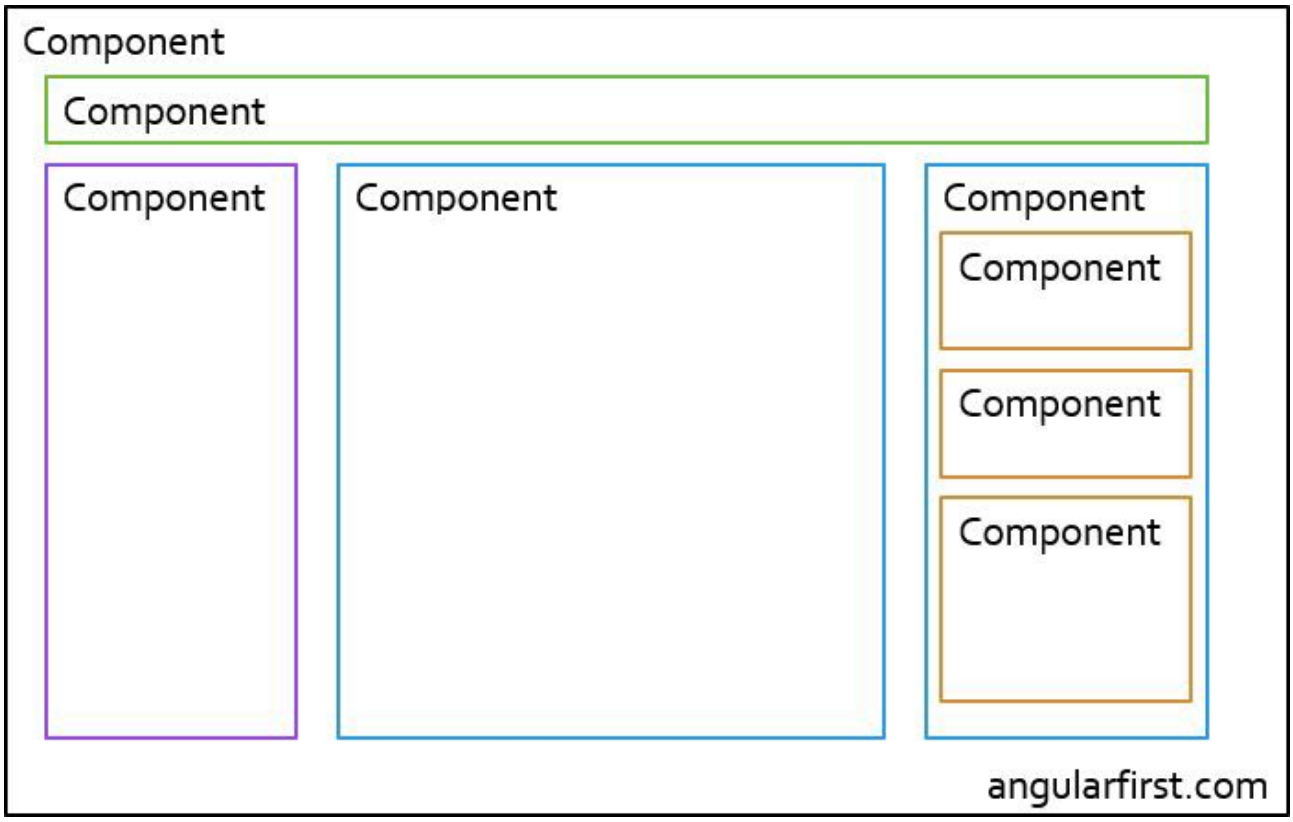
Ejemplo:

ng generate module usuarios
ng generate module productos

En cada módulo se declaran los componentes y si es necesario se importan otros módulos necesarios.

Componentes

Los componentes son partes de la pantalla que queremos que tenga una lógica del programa. Cada componente gestiona una pequeña parte de UI.



IMP: Cada componente tendrá su parte HTML (template) y su parte de lógica (Typescript)

¿Cómo creamos un componente?

ng generate componente mi_componente
o también:
ng g c mi_componente

Cuando creamos un componente siempre se crea:

- La vista asociada al componente: Template o View: **mi_componente.component.html**
  Es la parte HTML del componente.

- La clase del componente: **mi_componente.component.ts**
  Es una clase typescript donde se programa la lógica que contiene **metadatos**. Los metadatos contienen información adicional del componente. Se definen mediante **decoradores**: @Nombre_del_decorator.

- La hoja de estilos asociado al componente: **mi_componente.component.css**

- Un fichero para pruebas: **mi_componente.component.spec.ts**

Importante: Para que un componente pueda ser utilizado en la aplicación, es necesario registrar dicho componente en el módulo: app.module.ts

Vamos a practicar: Sobre el proyecto que ya tenemos creado vamos a añadir un componente:

- Crear un componente con el nombre UserProfile: ¿Cómo se genera?

  Vamos a la carpeta src/app/
  Vemos que se ha creado una carpeta user-profile:

  - user-profile.component.css
  - user-profile.component.html
  - user-profile.component.spec.css
  - user-profile.component.ts

- **Comprobamos que el componente se ha registrado en app.module.ts**.

- En la vista de user-profile vamos a escribir un párrafo con el texto: Componente UserProfile.
  ¿Lo vemos en nuestra página? **¿Por qué?**

- Incluimos el selector del componente : app-user-profile y lo incluimos en el componente principal: app.component.html

- ¿Podríamos poner el **selector del componente** varias veces? Probemos!

- En la **vista** del componente user-profile, creamos un div con dos etiquetas p, una para escribir un nombre de una persona y el otro para escribir el rol de dicha persona.

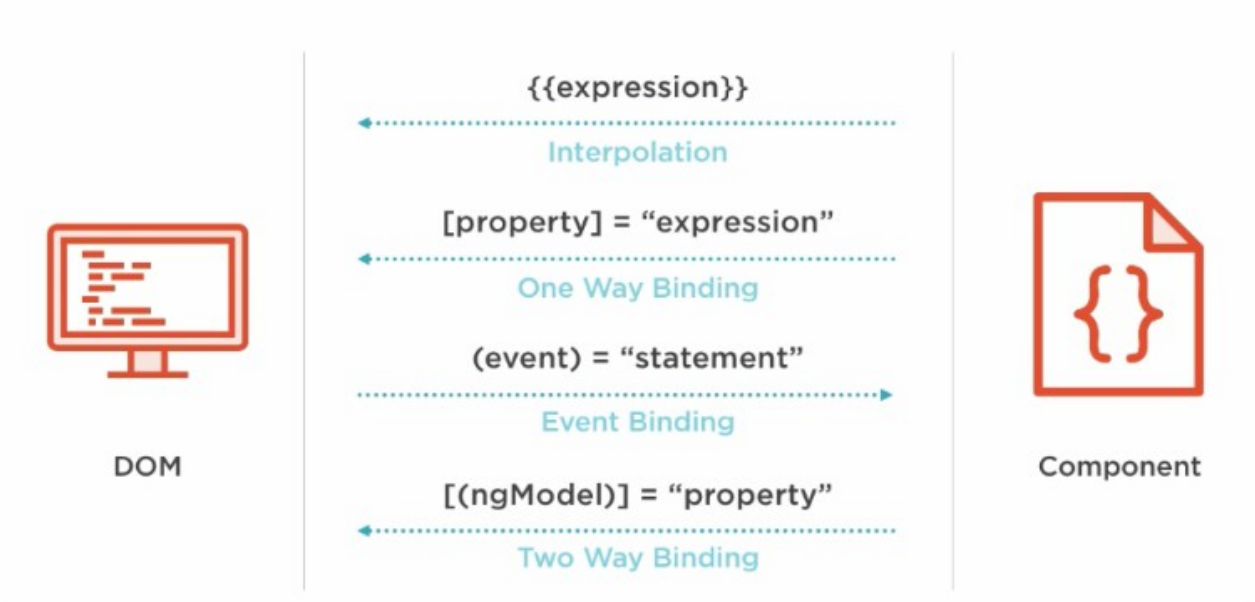
- Le damos estilos al div del componente con una class=”comp-user” que tenga: border 1 solid negro, padding 2em, margin 1em y width 20em.

- Le damos estilo al nombre, con una class=”name” y lo ponemos en negrita (font-weight:bold) y color azul.

- Le damos estilo al rol, con una class=”role” y lo ponemos en itálica (font-style: italic)

Parametrización o Data Binding

Los datos hay que parametrizarlos para que no estén en las plantillas o vistas de manera estática.
Hay 4 formas de pasar parámetros de la vista al componente y/o viceversa para que siempre estén los datos sincronizados entre ambos, de tal manera que siempre que el usuario actualice datos en la vista, Angular actualiza el componente y cuando el componente obtiene nuevos datos, angular actualiza la vista.



- Interpolación: Modelo a vista del componente al DOM.
- Enlazado de propiedades: Modelo a vista.
- Event binding
- Two way binding

1.- DataBinding – Interpolación

La información va desde el componente a la vista.

Se utilizan las llaves {{ *expresion*}}, de tal forma que Angular evalúa la expresión, la convierte a cadena y reemplaza su valor. Si *expresion* se actualiza, Angular actualiza la cadena.

Necesitaremos crear tantas propiedades como datos queremos mostrar en el componente.

Ejemplo: En el ejemplo anterior, vamos a duplicar otro div, pero esta vez los datos no serán estáticos. En el componente UserProfileComponent, en la clase, me creo dos propiedades:

- name: string = 'Tom Hanks';
- role: string = 'Usuario';

Y en la vista tendremos que utilizar esas propiedades con doble llave**:{{ nombre_parametro }}**

```
<span class="name">{{ name }}</span>
<span class="role"> {{ role }}</span>
```

Practica: Poner una propiedad titulo con el valor 'Componente User' en el componente y mostrarlo en la vista con un h1. Poner un color de letra purple a todas las etiquetas h1.

2.- DataBinding – Property binding o One way Binding

Permite asociar una propiedad (atributo) de un elemento HTML (class, href, src, etc..) con una propiedad del componente. Se utilizan los corchetes: **[propiedadHtml]** =”*expresion*”

Cuando cambia el valor del componente, Angular actualiza la vista.

Ejemplos:

```
<h1 [innerHTML]=”title” > </h1>
<input type="text" [disabled]="isDisabled">
```

Practica: Duplicar el titulo con h1 del componente UserProfile utilizando ahora property binding.

Practica: Crear ahora un input de tipo text que muestre el valor de nombre con property binding.

IMPORTANTE: Para mostrar propiedades en la vista desde el componente hay dos maneras:

1.- **Interpolación**: {{ expression}}
2.- **Property Binding**: [propiedad] =”expresion”

```
<img alt=”item” src={{itemImageUrl}}>
<img alt="item" [src]="itemImageUrl">
```

Binding de clases y estilos:

- Para usar clases de estilos definidos, podemos usar binding con la propiedad class de la siguiente forma:

```
<p class="miclase">En este párrafo usamos el atributo "class"</p>       // miclase es un estilo

<p [class]=”miclase”>En este párrafo usamos el atributo "class"</p> // miclase es una propiedad del componente
```

Angular evalúa **la expresión miclase**, y le aplica el valor a la propiedad class.

- También podemos aplicar un estilo si se cumple una condición:

```
<p [class.aviso]=”num==0” ></p>
```

- Para configurar el atributo **style** hacemos binding sobre la propiedad [style] seguido de la propiedad css que queremos modificar dinámicamente:

```
<p [style.width]=”width”>En este párrafo cambiamos la propiedad width</p>
<p [style.color]=”color”>En este párrafo cambiamos la propiedad  color</p>
<p [style.color]="isActive ? 'blue' : 'red' "></p>
```

Ejercicios.

- Sobre el componente creado **app.component**, definir dos parámetros y mostrarlos en la vista. Uno que se llame *name* con vuestro nombre y otro que se llame *imgUrl* y tenga la URL de la imagen que ya está en el componente. Crear la imagen en la vista para que la url sea ese valor que le estáis pasando desde el componente. Hacerlo de las dos formas vistas.

- Sobre el componente creado **app.component**, crear un párrafo con vuestro nombre, con una clase de estilo "naranja" que ponga el color naranja. Hacedlo usando el método **Property Binding**.

- Sobre la vista **app.component.html,** crear un checkbox que inicialmente esté marcado. Para ello crear una propiedad *marcado* = true en el componente y que desde la vista pueda leer dicha propiedad.

- Sobre la vista **app.component.html,** crear un input de type text que muestre vuestro nombre y que esté deshabilitado. Creando una propiedad que se llame *deshabilitado*.

- Crear en el componente un registro (objeto literal) que se llame *asignatura*. Tendrá los campos: nombre y codigo. Darles valores y mostrarlos desde la vista dentro de dos div.

- Crear un componente nuevo **alumno (carpeta alumno) e invocarlo desde el componente ppal. Crear propiedades para todo lo que se pide:**
  - Mostrar un título: 'Componente Alumno' de color rojo.
  - Que tenga las siguientes propiedades: Nombre, apellido, edad y dni. Dar valores y mostrarlos en la vista o plantilla (html) con etiquetas <p>.
  - Mostrar además en la vista otro párrafo p con la edad del alumno multiplicado por 5.
  - Además, si el alumno es mayor de edad, mostrar el mensaje 'Mayor de edad' y si es menor de edad 'Menor de edad'. Pista: usar operador ternario en la interpolación.
  - A continuación creamos un texto: 'Introducir DNI' y un input de tipo de texto de forma que cuando levantemos la tecla, el valor introducido en el input se vea un nuevo campo div dni.
    Usar el siguiente elemento: # hace que se pueda referenciar ese elemento.
    <input type="text" #nuevoDni (keyup) ="(0)" />
  - Por defecto el cuadro de texto estará deshabilitado(disabled). Usar property binding.
  - Crear en la vista un chexbox que esté marcado si la edad del alumno es menor de 18 años. Probad cambiando el valor de la edad del componente.
  - Crear en la vista otro checkbox que inicialmente esté desmarcado y cuando se marque, se muestre un mensaje alert con un mensaje. Probad a cambiar el valor del booleano en el componente.
  - Crear una propiedad en el componente con el texto 'antes de pulsar'. Y que cuando el checkbox anterior sea pulsado, el texto cambie a: 'check pulsado'
  - Crear dos radios. (sólo puede haber uno marcado, con el mismo name). Un radio que ponga Hombre y otro Mujer con value H y M respectivamente. Crear una función, de forma que si se pulsa el radio Hombre el texto anterior debe mostrar 'Pulsado Hombre' y si se pulsa Mujer, mostrará 'Pulsada Mujer'.

Pero, ¿ y si queremos pasarle los valores desde la vista al componente?

3.- DataBinding – Event binding

Para pasar valores desde la plantilla al componente usamos eventos. Se conoce como **Event Binding**.
Permite asociar eventos del DOM (click, keydown, mouseup, etc.) con un método del componente.
Se utiliza los paréntesis para el evento. La función debe definirse en el componente, dentro de la clase.

(evento)=”expresion o funcion”

Listado de eventos:

```
(click)="myFunction()"
(dblclick)="myFunction()"
(submit)="myFunction()"
(blur)="myFunction()"
(focus)="myFunction()"
(scroll)="myFunction()"
(cut)="myFunction()"
(copy)="myFunction()"
(paste)="myFunction()"
(keyup)="myFunction()"
(keypress)="myFunction()"
(keydown)="myFunction()"
(mouseup)="myFunction()"
(mousedown)="myFunction()"
(mouseenter)="myFunction()"
(drag)="myFunction()"
(drop)="myFunction()"
(dragover)="myFunction()"
```

A la función asociada al evento, se le puede pasar el objeto ‘**$event**’, para obtener el objeto que ha provocado el evento y todos sus atributos: Ej: (click)=mostrar($event)

Ejercicios en app.component:

- Crear un botón y un input de tipo text. Al pulsar el botón (click), muestre en el input, el nombre de vuestro compañero. Pista: Crear un método y un atributo en la clase para guardar el nombre de vuestro compañero.

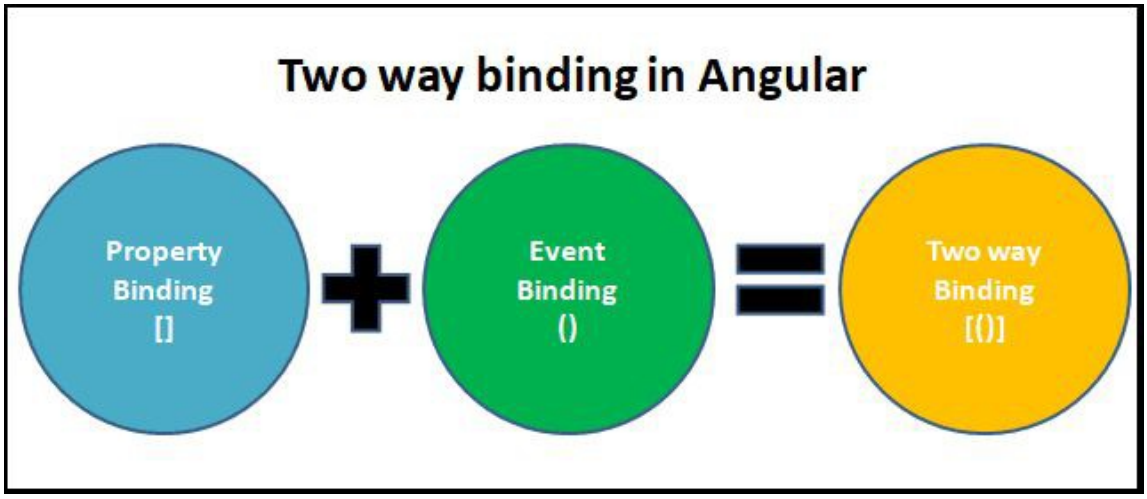
- Crear un input de tipo text. Cuando termine de escribir en él, con el evento keyup, mostrará una etiqueta con lo que haya escrito. Cada vez que lance el evento, los valores se irán concatenando con una coma. Pista: crear un método asociado al evento keyup y un atributo key para concatenar su valor al nuevo valor del input. Para acceder a dicho valor: event.target.value

- Crear un checkbox y que cuando lo marque o desmarque avise al componente y muestre en una etiqueta p de la vista, si está o no marcado.

Pero, ¿y si necesitamos la información tanto en el componente como en la vista?

4.- DataBinding – Two Way data binding

Se usan fundamentalmente en formularios. Establece una conexión bidireccional entre el valor del elemento de la vista y una propiedad en el modelo del componente. Los cambios hechos en el componente se propagan a la vista y los cambios hechos en la vista se actualizan en el componente.



Se utilizan los corchetes y paréntesis combinados:

```
[(ngModel)]="gender"
```

```
<input type="text" [(ngModel)]="nombre">
<p>El nombre es: {{ nombre }} </p>
```

En este caso, `[(ngModel)]` está vinculado a una propiedad llamada `nombre` en el componente. Si el usuario escribe algo en el cuadro de texto, el valor se actualiza en la propiedad `nombre` del componente automáticamente. Del mismo modo, si el valor de `nombre` en el componente cambia por alguna razón (por ejemplo, mediante la lógica del componente), la vista se actualiza automáticamente para reflejar ese cambio.

Importante: Hay que importar en app.module.ts FormsModule:

```
import { FormsModule } from '@angular/forms';
   imports: [BrowserModule, FormsModule],
```

Ejercicios:

- Crear un input de tipo text y un div, de tal forma que el contenido del input se muestre en el div.
  Primero usar la forma de One way Binding y event Binding.
  Después usar la forma de Two Way data binding

- Crear dos input de tipo text. De tal forma que lo que escribamos en el primer input se escriba en el segundo en mayúsculas . Y lo que escribamos en el segundo input se escriba en el primero pero en minúsculas.

- Crear una tabla con cuatro filas y dos columnas:
  - Fila 1: en la primera columna tiene un checkbox y en la segunda columna una capa div con el mensaje Hola. El funcionamiento debe ser el siguiente: Si se marca el checkbox, la capa div tendrá el estilo 'fondo':
    ```
    .fondo { background: yellow; }
    ```
    Si se desmarca, la capa div, no tendrá dicho estilo.

  - Fila 2: crear una columna con colspan=2 para que ocupe las dos columnas. Tiene un enlace <a> y queremos que el atributo href se le pase con un valor desde el componente, por ejemplo: "https://ceuandalucia.es"

  - Fila 3: En la primera columna crear un botón que inicialmente muestre el valor 'Mostrar' y cuando se pulse, cambie el texto a 'Ocultar'. Cuando esté en 'Ocultar', cambiará a 'Mostrar'. En la segunda columna, hay una capa div, con un texto 'Visible' que se ve cuando el botón de la primera columna muestre 'Mostrar'. Cuando se pulse el botón con texto 'Ocultar', la capa se oculta (hidden).
    Pista: crear un atributo mostrarCapa en el componente que inicialmente será true. Esto hará que el botón muestre 'Mostrar' y la capa estará hidden.

  - Fila 4: En la primera columna hay un botón que podrá tener los valores 'Habilitar' o 'Deshabilitar'. Inicialmente mostrará 'Habilitar'. Cuando se pulse 'Habilitar', el botón de la segunda columna con texto 'Hola que tal' se habilitará. Cuando el primer botón muestre 'Deshabilitar', el segundo botón se deshabilitará.

- Crear un componente nuevo para mostrar los datos de un libro. El componente se llama datos-libro.

  - Cambiamos el titulo de datos-libro.component.html. ¿Se ve en nuestra página?

  - Incluir la etiqueta del nuevo componente en la página app.component.html
  - En el componente nuevo datos-libro.component.ts vamos a crearnos un atributo libro, que será un objeto literal con los valores: titulo, autor, precio, stock, cantidad e imagen.

  - Mostrar todos los datos del libro en datos-libro.component.html en una lista (li)

  - En el caso de que el stock fuera cero, aplicar un estilo de color rojo a las unidades disponibles.
    Nota: crear `.aviso{ color: red; }` en la hoja de estilo del componente.
    Probarlo, poniendo dicho valor a 0 en el componente.

  - Añadimos dos botones: para añadir y quitar unidades del libro y una etiqueta para ver la cantidad de unidades del libro. Si pulsamos el botón para quitar una unidad, la cantidad se disminuye en una unidad, y si pulsamos el botón para añadir una unidad, la cantidad se aumenta en una unidad.

- Para evitar que puedan salir números negativos, vamos a poner que el botón que quita unidades se deshabilite si no hay cantidad disponible.

- Escribir por consola la posición del ratón cuando el puntero pase por encima de la imagen.

- Ahora vamos a permitir modificar el valor de la cantidad de unidades a través de un input. Sustituimos la etiqueta con las unidades de los libros por un input de tipo text.
  Acordaros que hay que tener importado el modulo FormsModule.

- Crear un proyecto nuevo con nombre Calculadora. Crear un componente reloj que se invocará desde el componente principal. Estará formado por dos input donde insertaremos dos valores y 4 botones: sumar, restar, multiplicar y dividir. Debajo de los botones mostraremos un texto con el resultado de la operación. Inicialmente los input mostraran el valor 0.

- Crear un proyecto nuevo con nombre Contador. Crear un componente de nombre Counter para poder sumar y restar una unidad a un contador que inicialmente tendrá el valor 10. El componente debe mostrar un título en h1 y el valor del contador en una etiqueta h3. Tiene además 3 botones, “+1”, “Resetear” y “-1”. Cuando se pulse el primer botón se incrementa el valor del contador en uno . Resetear lo volverá a poner a 10, y el último botón, resta uno al contador.

Ejercicios para practicar:

Crear un proyecto cuyo componente principal llame a otra llamado reloj. Este componente reloj permitrá al usuario cambiar la hora utilizando input de texto. Por defecto mostrará: ‘12:00:00’. Cuando se pulse un botón, se mostrará en un etiqueta p, la hora insertada en el input.
Además, tendremos otra etiqueta p donde se mostrará la hora actual (actualizándose cada 1000 ms)

Paso de parámetros de un componente padre a un hijo

Angular nos permite pasar valor de un componente padre a uno hijo. Por ejemplo: Vamos a recuperar el componente user-profile. Si en app.component.html incluimos:

```
<app-user-profile name="Harry Potter" rol="Mago"></app-user-profile>
                                  o
<app-user-profile [name]="variable" [rol]="Variable"></app-user-profile>
```

Vemos que no esos datos no se están mostrando. Falta un pequeño detalle, comunicar en el componente hijo que esos datos se pueden obtener desde el padre, para eso hay que **decorar (darle información específica)** esos **atributos. ¿Cómo? Diciendo que son datos input (de entrada).**

En el componente hijo, en userProfileComponent, importamos “Input” y en los atributos los decoramos con **@Input**():

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

Y en los atributos name y role, anteponerle el decorador @Input().

```
@Input()
 name: string = '';
```

Paso de parámetros de un componente hijo al padre

Sobre la vista del componente UserProfile vamos a crearnos un botón, con un evento click. Al pulsarlo, vamos a crear un método seleccionar que mostrará por consola el nombre del usuario. Pero además, quiero que el el componente padre sepa que ha lanzado el evento en el componente hijo y a quien he pulsado. Para que el componente padre se entere, tenemos que hacer varias cosas.

Primero, tenemos que tener en el hijo un evento que "avise" al padre y dicho evento debe tener un decorador de salida: @Output:

```
@Output()
selected = new EvenEmitter<string>();  // de angular core
```

NOTA: selected es el atributo que le voy a pasar al padre.

En el componente hijo, en userProfileComponent, **importamos "Output"**

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

Segundo, tengo que decir qué voy a emitir. En este caso, quiero emitirle al padre el nombre del usuario que he pulsado. En el método seleccionar(), emito el nombre pulsado con **emit():**

```
this.selected.emit(this.name);
```

Tenemos que importar "EventEmitter":

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

Tercero, falta que el padre se entere del valor que el hijo le está pasando. En la vista del padre, hay que utlizar dicho evento: selected:

```
<app-user-profile name="Harry Potter" rol="Mago" (selected)="SeleccionarPadre($event)"></app-user-profile>
```

$event le pasa al padre lo que viene del hijo.
Ahora definimos en el componente padre mi nueva función SeleccionarPadre, que se encargará de pintar el valor recibido.

Ejercicios

- En el componente padre, escribir un texto y muestra el nombre del usuario asociado al botón pulsado.

- En el componente **datos-libro**, crear un array de dos objetos literales libros.
  En el componente padre, crear un input de tipo text para escribir numero del libro que queremos 1 ó 2. Pasarle al componente hijo el número de libro.
  El componente hijo tiene que recuperar ese valor y tendrá un botón que al pulsarlo, devolverá el título del libro solicitado.

- ¿Y si quiero pasarle al componente padre un objeto? Repetir el ejercicio anterior, pero el evento al padre debe de enviar el objeto completo del libro, no sólo el titulo. Pista: crear una interfaz libro.

Interface

Las interfaces se utilizan para definir estructuras. Puedes declarar la forma que debe tener un objeto, incluyendo sus propiedades y métodos, pero no puedes proporcionar implementación.

```
export interface Persona {
nombre: string;
edad: number;
saludar(): void;
}
```

Ejercicio para practicar

Crear un proyecto llamado mensajes. El componente principal tendrá un título:"Mensaje Padre", y un input de type textarea. Llamará a un componente hijo: mensajeHijo.
Cuando en el componente padrese escriba el mensaje, será enviado al hijo. El componente hijo mostrará dicho mensaje en una etiqueta <p>. Y tendrá un input de type textarea para enviar la respuesta al padre, cuando pulse un botón. Dicha respuesta será visualizada en el padre en una etiqueta p.

Instalar Bootstrap

En el proyecto instalamos las dependencias de bootstrap:

```
npm install bootstrap
```

En styles.css dentro de src:

```
@import "~bootstrap/dist/css/bootstrap.min.css";
```

Ciclo de vida de un componente

Angular comienza cargando main.ts, que levanta el módulo principal. En el módulo principal vemos cúal es el componente principal, AppComponent. En dicha clase, crea el objeto de la clase, mira el selector, app-root y busca en index.html ese selector para poder reemplazar el código del componente.

Por lo tanto, en la clase del componente hay un constructor que permite crear el objeto de dicha clase.

```
export class AppComponent {
        constructor {
                console.log('constructor de appComponent');
        }
```

Los componentes poseen un ciclo de vida, de tal manera que se crean, se actualizan y se destruyen. Podremos utilizar dichos ciclos mediante eventos anteponiendo el prefijo ng:

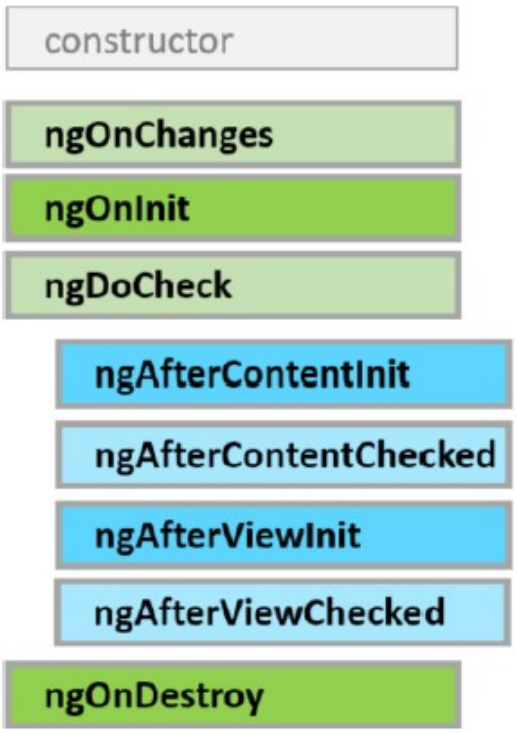
https://angular.io/guide/lifecycle-hooks

ngOnChanges: Cuando las propiedades de entrada detectan cambios.

ngOnInit: Se llama solo una vez justo después de la construcción del componente. Se suele usar para cargar datos.

ngDoCheck: para actuar sobre cambios que angular no captura

ngOnDestroy: antes de destruir el componente. Se usa para limpiar recursos que no se hacen automaticamente.



Ejercicio

Crear una aplicación para crear una traza y monitorizar la secuencia de los momentos del ciclo de vida de un componente. Tendremos dos componentes, el principal y otro hijo llamado desde el principal: comp1

- El componente principal tiene un título y input de tipo text. El valor que introduzcamos en el input, lo pasaremos al componente hijo.

- El componente hijo tiene un input que mostrará el valor del input del padre y un div con el texto: Hola valor_input.

- Importar lo siguiente en la clase de comp1:
import { Component, OnInit, SimpleChanges, Input, OnChanges, ViewChild } from '@angular/core';

- Copiar lo siguiente en la clase de comp1:

```
contador: number = 0;
constructor() { }

ngOnInit() { this.mostrar("pasa por ngOnInit"); }

ngOnChanges(cambios: SimpleChanges) {
        for (let propiedad in cambios) {
        let cambio = cambios[propiedad];
        let actual = JSON.stringify(cambio.currentValue);
        let anterior = JSON.stringify(cambio.previousValue);
        this.mostrar("Pasa por ngOnChanges. Propiedad (" + propiedad + ") valor actual (" + actual + ")
        valor anterior (" + anterior + ")");
        }
}

ngDoCheck() { this.mostrar("pasa por ngDoCheck"); }

ngAfterContentInit() { this.mostrar("pasa por ngAfterContentInit"); }

ngAfterContentChecked() { this.mostrar("pasa por ngAfterContentChecked"); }

ngAfterViewInit() { this.mostrar("pasa por ngAfterViewInit");}

ngAfterViewChecked() { this.mostrar("pasa por ngAfterViewChecked"); }

public mostrar(texto: string) {
        this.contador += 1;
        console.log(this.contador + " - " + texto);
}
```

- Abrimos la consola F12, y vemos la traza.

Directivas

Permiten añadir a HTML comportamiento dinámico mediante una etiqueta o selector. Podremos añadir o eliminar elementos del DOM. Ya hemos visto la directiva NgModel.

- ***ngFor**: itera sobre una **lista de elementos**

```
<div *ngFor="let hero of heroes">
   {{hero.name}} {{hero.lastname}}
</div>
```

***ngIf**: crea o elimina físicamente elementos según se cumple una condición.

```
<div *ngIf="condicion">
  {{contenido}}
</div>

<div *ngIf="obj">
  <p>Hola {{obj.name}}</p>
</div>

<div *ngIf="obj"; else otraRama>
  <p>Hola {{obj.name}}</p>
</div>
<ng-template #otraRama>
</ng-template>
```

- ***unless:** es la inversa de if.

```
<div *unless=“data">
        No data found
</div>
```

- ***ngSwitch**: evalúa una expresión contra N casos, eliminando los que no cumplan una condición.

Opción 1:

```
<elemento [ngSwitch]=”expresion”>
      <elemento *ngSwitchCase=”’valor1’”>…</elemento>
      <elemento *ngSwitchCase=”’valor2’”>…</elemento>
      <elemento *ngSwitchDefault>…</elemento>
</elemento>
```

Opción 2:

```
<div [ngSwitch]="status">
     <template [ngSwitchCase]="'in-mission'">In Mission</template>
     <template [ngSwitchCase]="'ready'">Ready</template>
     <template ngSwitchDefault>Unknown</template>
</div>
```

- **ngStyle y ngClass.** Permite dinámicamente modificar los estilos:
  Ejemplo: [ngStyle]=”{color:’red’}”

  Si quisieramos tener muchas propiedades, usamos ngClass.
  Ejemplo: [ngClass] = “{menor: edad<18}”

```
. menor{
color:red;
font-weight: bold;
text-decoration: underline;
}
```

Ejercicios

- Crear un nuevo proyecto ‘directivas’.

- Crear un formulario en el componente principal con el título “Alta de usuario”. Tendrá dos input text: para el nombre y los apellidos, y un botón ‘Añadir’. Cuando se pulse el botón , debajo se mostrará un párrafo con el mensaje: ‘Alta correcta del usuario XXX’. OJO: el párrafo sólo debe aparecer cuando se haya pulsado el botón. Usar directiva *ngIf.

- Modificarlo para que si no se ha pulsado el bóton muestre el mensaje: ‘Ninguna alta’

- En el componente principal, crear una interfaz Persona con : nombre, apellidos y edad.
  Crear un atributo de la clase: personas que es un array de la interfaz Persona.
  Inicializar el array con dos objetos Persona.
  Pintar en el html del componente una lista con los datos del array. Usar directiva *ngFor

- Mostrar la lista con los datos en negrita. Añadimos en cada entrada de la lista el indice de cada objeto (index).

- A continuación mostrar sólo las personas cuya edad sea >=30

- Crear 3 botones con los valores A, B y C. Mostrar debajo del botón qué opción ha pulsado.
  Además, si ha pulsado el botón A que indique el texto “Pulsado el botón A”, si ha pulsado el botón B: “Ha pulsado el botón B” , si ha pulsado el botón C: “Ha pulsado el botón C” y si no pulsa nada, por defecto que muestre, “No ha seleccionado ningún botón”

- Crear un componente articulo. En la vista incluiremos un formulario con los campos: nombre, precio y unidades. Tendrá un botón Comprar.

  Debajo del formulario veremos la lista de artículos que tenemos (nombre, precio y unidades) más el artículo nuevo que hemos introducido en el formulario: para ello dentro del método, crear un objeto literal con los campos y valores que se han insertado en el formulario.
  Nos vamos a crear esta vez una interfaz artículo en una carpeta app/model/articulo.model.ts con los 3 atributos que necesitamos almacenar.
  Inicialmente, en el componente artículo crear una lista con 3 artículos.
  En el listado de artículo mostrar el índice de cada registro.

- Crear un componente hijo: listado-articulos. De forma que el componente articulo sólo tenga el formulario con los campos, nombre, precio y unidades y el botón. El listado de los articulos añadidos deben estar en este componente hijo listado-articulos. Desde el componente padre *articulo* debemos llamar al component *listado-articulos*.

- Crear un componente llamado caracteristicas-articulo. Será un componente hijo de listado-articulo. Este nuevo componente tendrá un input para introducir un nuevo dato, una caracteristica: y cuando se pulse un botón, se añadirá esta información al componente padre: listado-articulos.

- Crear un componente hijo: listado-personas. Desde el componente principal, llamamos al componente hijo tantas veces como personas haya en el array pasándole la "persona" El componente hijo, listará nombre, apellidos y edad. Además:
  Si la persona es menor de edad, se muestre un mensaje "Menor de edad" en color rojo. En caso contrario, mostrar el mensaje: "Mayor de edad".

- Repetir el apartado anterior, pero creando una propiedad esMenor()

- Al lado de cada persona crear un radio button. De forma que al pulsar uno, envíe la información al componente padre y sepa que persona ha sido marcada. Escribir en el componente padre la información de la persona seleccionada.

- Incluir en el componente hijo tres input (text) para mostrar el nombre , apellidos y edad de cada persona. Si algún valor de la persona es modificado en el input, el cambio debe verse en todas las referencias de dicho valor.

Pipes

Se usan para formatear y filtrar datos. Se pueden concatenar: {{data | pipe1:arg1 | pipe2 | pipe3}}

Ejemplo: Pasar a mayúsculas

```
{{texto | uppercase}}
```

Ejemplo: Pasar a minúsculas

```
{{texto | lowercase}}
```

Pipe para fechas:

```
{{ d1 | date: 'shortDate' }}
```

Para crear un nuevo pipe

```
ng g pipe pipes/nombrePipe
```

Ejercicio Repaso

- Crear una aplicación cuyo componente principal tenga un título de bienvenida centrado de color azul. Debe tener los siguientes componentes:

  - counter
    - La vista debe mostrar:
      - Un título h1 de color verde.
      - El valor de una propiedad 'counter'
      - 3 botones: +1, resetear y +1.
    - El componente debe funcionar de la siguiente manera:
      La propiedad counter debe estar inicializada con el valor 10.
      Al pulsar el boton +1 el valor de counter debe aumentar en 1. Al pulsar el botón reset, el valor de counter debe volver a tomar el valor inicial y al pulsar el boton -1 el valor de counter debe disminuir en 1.

  - hero
    - La vista debe mostrar:
      - Un título h1 de color naranja
      - Los siguientes datos de un heroe: Nombre, edad, el nombre en mayúsculas, la concatenación del nombre y la edad.
      - Botón que nos permita cambiar la edad del heroe a la edad de 18 por defecto.
      - Botón que nos permita cambiar el nombre a 'spiderman' por defecto. Este botón será solo visible si el nombre del heroe es ironman.

    - El componente debe tener las siguientes propiedades:
      - Nombre del heroe inicializado al valor que queramos
      - Edad del heroe inicializado al valor que queramos

  - list
    - La vista debe mostrar:
      - Un listado de un array con nombres de heroes.
      - Un botón 'Borrar el último heroe'.
      - El texto con el heroe eliminado: El héroe borrado es ___
        Inicialmente mostrará el texto: No se ha borrado nada (Hacerlo con *ngIf)
    - El componente debe tener:
      - Un array de string con 4 nombres de heroes
      - Piensa qué funcionalidad debe tener...
        NOTA:¿Qué pasa si ya no me quedan elementos que borrar?