

DWC

(Desarrollo Web en entorno cliente)



Angular

Práctica 03

Data Binding

Índice

1.- Objetivos.....	1
2.- Añadir Bootstrap al proyecto.....	1
3.- Angular CLI: Interfaces y Componentes	2
4.- Tipado y Mock de datos.....	3
5.- Data Binding	6
5.1.- Comprar artículos con unidades.....	6
5.2.- Seleccionar el artículo.....	6
5.3.- Modificar las unidades	7

1.- Objetivos

En esta práctica vamos a trabajar con los siguientes conceptos:

- Añadir Bootstrap a nuestro proyecto
- Trabaja con Angular CLI para la creación de interfaces y componentes.
- Tipado y Mock de datos.
- Trabajar con el data binding

2.- Añadir Bootstrap a nuestro proyecto

Hemos visto que Angular permite definir un estilo global para la aplicación y un estilo para cada componente. Es habitual utilizar librerías de terceros para utilizar componentes avanzados de interface de usuario o simplemente para utilizar estilos y componentes definidos.

Vamos a instalar Bootstrap para trabajar con estilos y componentes, para ello deberemos instalar la librería y configurar los estilos en nuestra aplicación.

Para instalar

```
D:\my-app>npm install bootstrap jquery @popperjs/core
added 3 packages, and audited 916 packages in 3s

120 packages are looking for funding
  run `npm fund` for details

2 high severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
```

Se instalarán las dependencias **bootstrap**, **jquery** y **@popperjs/core** que son necesarios para darle mejor potencial al funcionamiento del framework.

Después de la instalación aparecerá en node_modules una carpeta para Bootstrap

```
✓ bootstrap
  > dist
  > js
  > scss
  📄 LICENSE
  {} package.json
  ⓘ README.md
```

Para configurarlo

Modificamos el archivo **angular.json** y colocamos las siguientes instrucciones en las propiedades **styles** y **scripts** en donde llamaremos a las propiedades css del bootstrap y las dependencias scripts correspondientes a las interacciones de cada uno.

```
"styles": [
  "node_modules/bootstrap/dist/css/bootstrap.min.css",
  "src/styles.css"
],
"scripts": [
  "node_modules/jquery/dist/jquery.min.js",
  "node_modules/@popperjs/core/dist/umd/popper.min.js",
  "node_modules/bootstrap/dist/js/bootstrap.min.js"
]
```

Ya tendremos disponibles todas las clases de bootstrap para utilizar en nuestro proyecto. Podríamos probarlo poniendo en el HTML un botón con la clase que queramos.

3.- Trabaja con Angular CLI: Interfaces y Componentes

Hemos visto que Angular utilizado el tipado, es muy habitual en Angular el uso de interfaces.

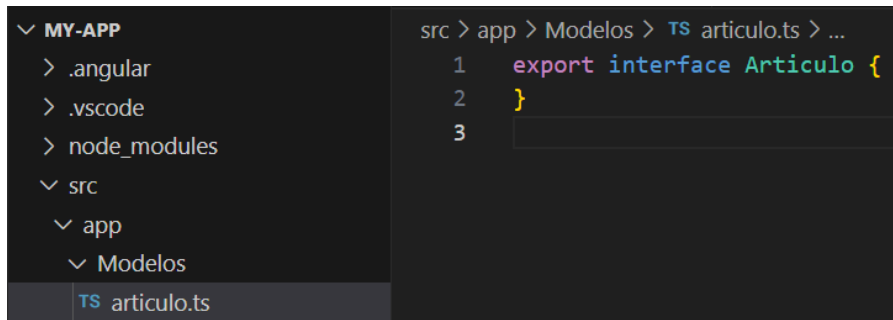
Una interfaz describe las propiedades que un objeto debe tener y los tipos de datos que deben contener esas propiedades. No define la implementación real del objeto, solo su forma.

Las interfaces **se utilizan para tipar objetos con una estructura determinada, no para crear los objetos**. Para crear objetos utilizaríamos las clases.

Vamos a trabajar con artículos que tengan un id, nombre, descripción, unidades y precio. Para ello nos definiremos una interface llamada Artículo, **usaremos una carpeta dentro del proyecto para las interfaces llamada Modelos**.

Para crear la interface:

```
D:\my-app>ng g i Modelos/Articulo
CREATE src/app/Modelos/articulo.ts (32 bytes)
```



Se habrá creado la interface dentro de la carpeta Modelos. Se ha creado un archivo ts que solo exporta la interface Articulo. Ahora deberemos de crear la interface definiendo las propiedades y sus tipos de datos.

```
export interface Articulo {
  id:string,
  nombre:string,
  descripcion:string,
  unidades:number,
  precio:number
}
```

4.- Tipado y Mock de datos

Como vamos a trabajar con muchos artículos podemos definir un array de los datos que vamos a usar dentro de ese fichero. Como vamos a utilizar esos datos en nuestros componentes deberemos exportar el array.

```
export const ARTICULOS:Array<Articulo>=[]
```

También podemos definir el array de esta forma

```
export const ARTICULOS:Articulo[]=[]
```

Ahora ya completamos el array con los datos

```
export const ARTICULOS:Array<Articulo>=[
  {
    'id':'m1',
    'nombre':'Galaxy A32',
    'descripcion':'4GB + 128GB libre',
    'unidades':10,
    'precio':229
  },
  {
    'id':'m2',
    'nombre':'Oppo A94',
    'descripcion':'8GB + 128GB libre',
    'unidades':10,
    'precio':269,
  },
]
```

Ahora ya disponemos de un array de artículos para poder utilizarlo en nuestros componentes.

Una vez creado la interface y los datos vamos a probarlos en un componente. Creamos un componente llamado artículos, lo crearemos dentro de una carpeta llamada Components.

```
D:\my-app>ng g c Components/articulos
CREATE src/app/Components/articulos/articulos.component.html (25 bytes)
CREATE src/app/Components/articulos/articulos.component.spec.ts (640 bytes)
CREATE src/app/Components/articulos/articulos.component.ts (258 bytes)
CREATE src/app/Components/articulos/articulos.component.css (0 bytes)
```

Una vez creado el componente vamos a interpolar un artículo y simular su compra mediante un botón de comprar.

Para ello deberemos:

- Importar la interface Artículo
- Importar los datos del array de artículos
- Utilizar una variable del componente para asignar los datos del array de artículos
- Usar una variable del componente para un artículo en concreto, por ejemplo, el 3.
- En la vista interpolar ese artículo, usaremos un card de Bootstrap
- Asignar un método al click del botón para simular su compra

En el ts

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

@Component({
  selector: 'app-articulos',
  standalone: true,
  imports: [],
  templateUrl: './articulos.component.html',
  styleUrls: ['./articulos.component.css']
})
export class ArticulosComponent {

  articulos:Array<Artículo>=ARTICULOS
  articulo:Artículo=this.articulos[3]

  comprar(articulo:Artículo){
    console.log(articulo)
  }
}
```

En el html

```
<div class="card" style="width: 18rem;">
  
  <div class="card-body">
    <h5 class="card-title">{{articulo.nombre}}</h5>
    <p class="card-text">{{articulo.descripcion}}</p>
    <b>
      <p class="card-text text-center">{{articulo.precio}}</p>
    </b>
  </div>
  <button class="btn btn-primary" (click)="comprar(articulo)">comprar</button>
</div>
```

Para poder probar el proyecto deberemos incluir el selector del componente que hemos creado en el componente principal.

app.component.ts

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { ArticulosComponent } from './Components/articulos/articulos.component';

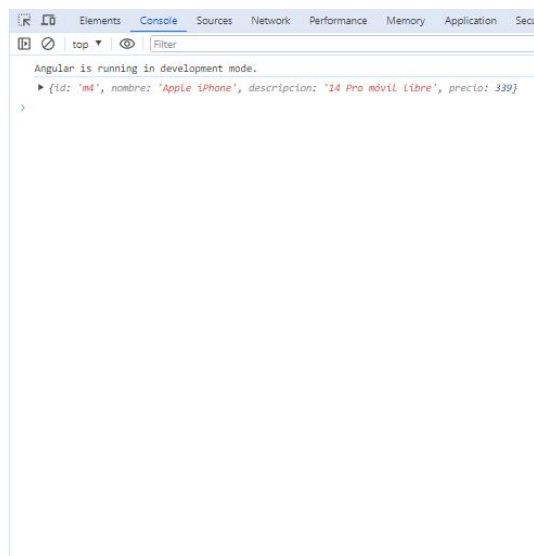
@Component({
  selector: 'app-root',
  standalone: true,
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  imports: [RouterOutlet, ArticulosComponent]
})
export class AppComponent {
  title = 'Practica 1';
}
```

app.component.html

```
<h2>{{title}}</h2>
<app-articulos></app-articulos>
```

El resultado

Practica 1



5.- El Data Binding

Hasta ahora hemos trabajado con la **interpolación** para poder mostrar un artículo, también hemos utilizado el **event binding** para gestionar el evento clic del botón comprar.

5.1.- Comprar artículos con unidades

Vamos a modificar el ejemplo para que solo podamos comprar cuando el artículo tenga unidades. Para ello podríamos utilizar un **property binding** de la propiedad disabled con el número de unidades del artículo. De esta manera el botón comprar solo estará activo si tienen unidades.

Deberemos modificar el html para añadir el property binding

```
<button class="btn btn-primary"
  (click)="comprar(articulo)"
  [disabled]="articulo.unidades==0">comprar
</button>
```

Como el artículo 3 no tiene unidades no se podrá comprar porque el botón estará desactivado.

Practica 1



5.2.- Seleccionar el artículo

Vamos a añadir un cuadro de texto para que podamos introducir el código del artículo que queramos visualizar y al pulsar en un botón que se vea ese artículo. Si el artículo no existe que muestre un error.

Modificamos el html

Añadimos encima del card el input y el button. Vamos a usar el two way data binding, para ello usaremos una variable llamada id para vincularla con ngModel al input


```
Articulo: <input [(ngModel)]="id" size="3">
<button class="btn btn-success" (click)="ver()">Ver</button>
```

En el ts

Añadimos la variable id y el método para gestionar el click del button


```
id:string=""
```

```
ver(){
  let art=this.articulos.find(a=>a.id==this.id)
  if (art)
    this.articulo=art
  else
    alert("El articulo no existe !!!")
}
```

El resultado

Practica 1

Articulo: Ver



Galaxy A32 (10)
4GB + 128GB libre


229

Comprar

Existe artículo

Practica 1

Articulo: Ver



Galaxy A32 (10)
4GB + 128GB libre

229

Comprar

No existe el artículo

localhost:4200 says

El articulo no existe !!!

OK

5.3.- Modificar las unidades

Vamos a añadir dos botones para poder aumentar o disminuir el número de unidades del artículo que se muestra.

En el html

Añadimos los dos botones al final del card

```
<button class="btn btn-danger"
  (click)="aumentar(articulo)">Aumentar
</button>
<button class="btn btn-danger"
  (click)="disminuir(articulo)">Disminuir
</button>
```

En el ts

Añadimos los métodos para gestionar los clicks de los buttons

```

aumentar(articulo:Articulo){
  let art=this.articulos.find(a=>a.id==articulo.id)
  if (art)
    art.unidades++
}

disminuir(articulo:Articulo){
  let art=this.articulos.find(a=>a.id==articulo.id)
  if (art)
    if (art.unidades>0)
      art.unidades--
}

```

Podríamos hacer un único método al que le pasáramos un artículo y la operación.

En html

```

<button class="btn btn-success"
  (click)="modificarUnidades(articulo, '+')">Aumentar
</button>
<button class="btn btn-danger"
  (click)="modificarUnidades(articulo, '-')">Disminuir
</button>

```

En el ts

```

modificarUnidades(articulo:Articulo, operacion:string){
  let art=this.articulos.find(a=>a.id==articulo.id)
  if (art)
    if(operacion=="+")
      art.unidades++
    else
      if (art.unidades>0)
        art.unidades--
}

```

El resultado

