

DWC

(Desarrollo Web en entorno cliente)



Angular

Tema 02

Módulos y Componentes

Índice

1.- Módulos	1
2.- Componentes	2
2.1.- El Controlador	2
2.2.- La vista.....	4
2.3.- Creación de componentes	4
2.4.- Uso de componentes.....	5
2.5.- Ciclo de vida de los componentes	7
3.- Proceso de arranque	9

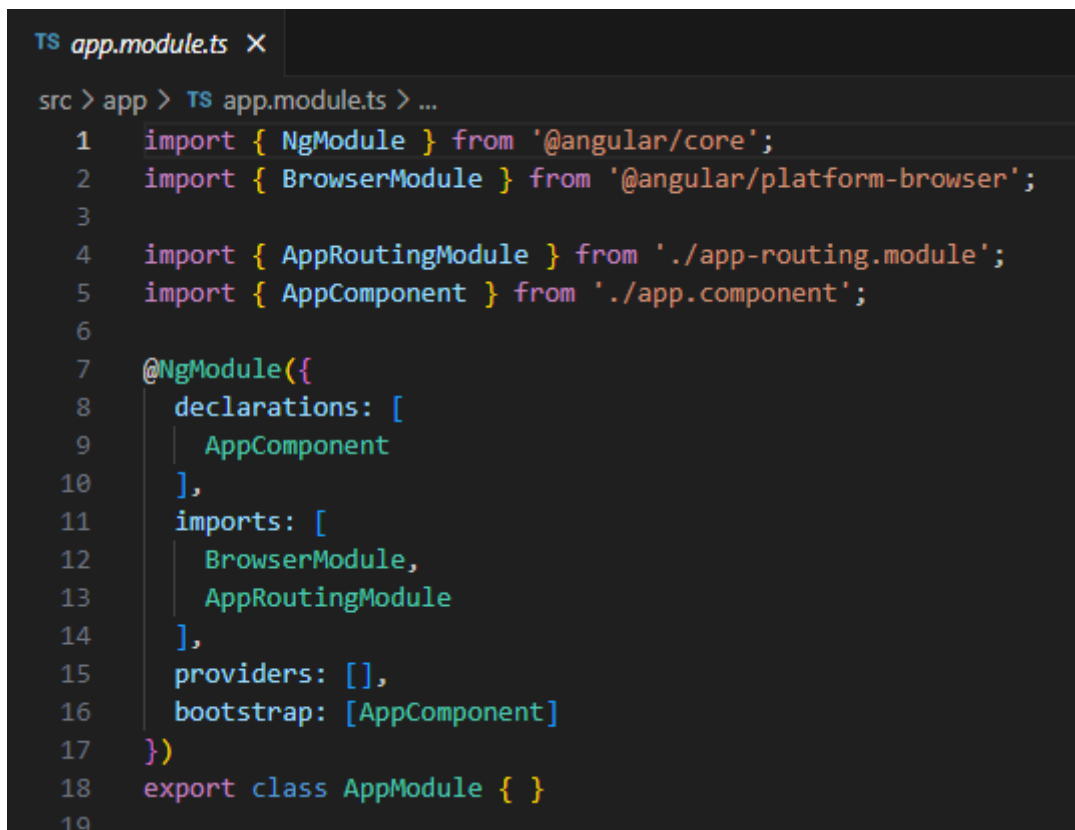
1.- Módulos

En Angular, un módulo es una clase con el decorador **@NgModule** que agrupa componentes, directivas, pipes y otros servicios relacionados.

Un decorador en Angular es una función especial que se utiliza para agregar metadatos a una clase, propiedad o método. Los decoradores se colocan justo encima de la declaración de la clase, propiedad o método que se desea decorar, y se denotan con el símbolo @ seguido del nombre del decorador.

Los módulos son fundamentales para organizar la aplicación en bloques cohesivos de código, facilitando la gestión y el mantenimiento del proyecto. Son **contenedores para almacenar los componentes y servicios** de una aplicación. En Angular cada programa se puede ver como un árbol de módulos jerárquico. A partir de un módulo raíz se enlazan otros módulos mediante la importación.

En versiones anteriores a la 17 cualquier aplicación creada con Angular generaba un módulo principal llamado **app.module.ts** en el cual se definían todos los elementos del proyecto



```
TS app.module.ts X
src > app > TS app.module.ts > ...
1  import { NgModule } from '@angular/core';
2  import { BrowserModule } from '@angular/platform-browser';
3
4  import { AppRoutingModule } from './app-routing.module';
5  import { AppComponent } from './app.component';
6
7  @NgModule({
8    declarations: [
9      AppComponent
10   ],
11   imports: [
12     BrowserModule,
13     AppRoutingModule
14   ],
15   providers: [],
16   bootstrap: [AppComponent]
17 })
18 export class AppModule { }
19
```

A partir de la versión 17 Angular recomienda utilizar los **componentes standalone**, los cuales no necesitan de ningún módulo. Es en el propio componente donde se incluirán los elementos que se necesitan, para ello se amplía la funcionalidad del decorador **@Component**

Al crear un proyecto en Angular ya no se crea ningún módulo, se pueden crear de forma manual utilizando el CLI. El comando es **ng g m nombre_modulo**

```
D:\my-app>ng g m /Modules/modulo1  
CREATE src/app/Modules/modulo1/modulo1.module.ts (205 bytes)
```

Se nos habrá generado el fichero **modulo1.module.ts** en la carpeta Modules

```
import { NgModule } from '@angular/core';  
import { CommonModule } from '@angular/common';  
  
@NgModule({  
  declarations: [],  
  imports: [CommonModule]  
})  
export class Modulo1Module { }
```

2.- Componentes

Los componentes son los bloques básicos de construcción de las páginas web en Angular. Contienen una parte visual en html (la Vista) y una funcional en Typescript (el Controlador). La aplicación original que crea el CLI nos incluye un primer componente de ejemplo en el fichero app.component.ts.

Según la configuración del CLI este componente puede haber sido creado en un sólo fichero o hasta cuatro: (el controlador, con la vista y los estilos en ficheros propios y fichero extra para pruebas unitarias).

2.1.- El Controlador (archivo .component.ts)

Los componentes, como el resto de elementos en Angular, serán **clases TypeScript decoradas** con funciones específicas. En este caso la función es **@Component()** que recibe un objeto de definición de componente. Igual que en el caso de los módulos contiene las propiedades en las que configurar el componente.

Al crear un proyecto en Angular se crea el componente AppComponent

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';

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

Los componentes definen nuevas etiquetas HTML para ser usados dentro de otros componentes. Excepcionalmente en este caso por ser el componente raíz se consume en la página index.html.

Al igual que el resto de elementos de Angular los componentes deberán importar los elementos que vayan a utilizar. En este caso se están importando:

- **Component:** Es la clase que utilizaremos en el decorador `@Component`, esto se deberá importar en todos los componentes.
- **RouterOutlet:** Es la clase que nos permite utilizar el router en nuestra aplicación.

Una vez realizadas las importaciones necesarias pasamos al **decorador** `@Component` que es el encargado de incluir los metadatos de cada componente, es decir, es donde se definirán las propiedades de la clase del objeto componente que estamos utilizando. Las propiedades más importantes son:

- **selector:** Define el nombre del elemento HTML que se utilizará para renderizar el componente.
- **standalone:** Se incluye en la versión 17 e indica que ese componente no depende de ningún módulo.
- **imports:** Se utiliza para importar módulos, componentes, directivas y pipes que se necesitan en el componente.

- **templateUrl:** La ruta al archivo HTML que contiene la plantilla del componente. Se puede definir directamente el html en el decorador utilizando la propiedad template.
- **styleUrls:** La ruta al archivo CSS que contiene los estilos del componente.

Una vez definido el decorador pasamos a definir la clase del componente, esta clase se exportará para que pueda ser importada por los elementos que la necesiten.

Es aquí donde definiremos la lógica de nuestro componente, es decir donde incluiremos el código que implemente la funcionalidad de nuestro componente. Este código estar formado:

- **Propiedades y variables** que definen el estado del componente.
- **Métodos** que manipulan el estado del componente y responden a eventos.

2.2.- La vista (archivo .component.html)

En este archivo es donde se define la vista del componente, para ello se utilizará html. Además del html se pueden utilizar el data binding y las directivas de Angular. Aquí se define la estructura visual del componente utilizando:

- Plantillas HTML que definen la estructura del DOM.
- Enlaces de datos que conectan las propiedades del componente con la plantilla HTML.
- Directivas que añaden funcionalidades a la plantilla.

Angular cada vez que crea un componente en la plantilla solo incluye el html mínimo para indicar que ese componente funciona.

2.3.- Creación de componentes

Para crear un componente utilizaremos el comando **ng g c nombre_componente**.

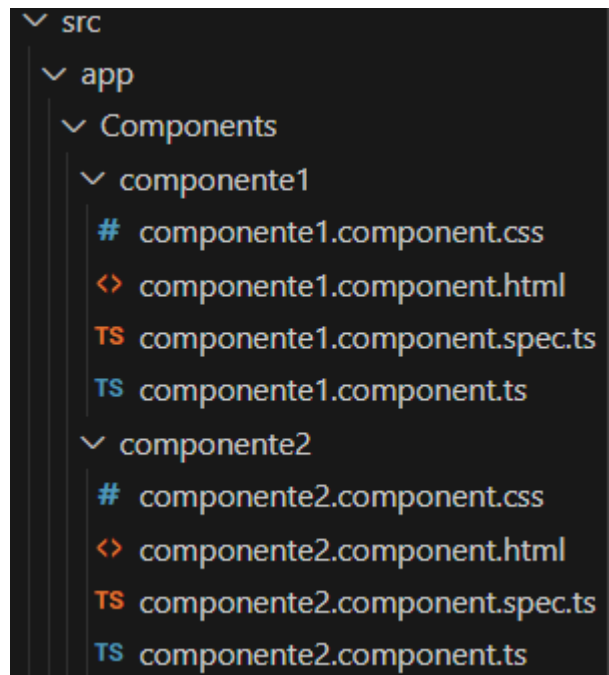
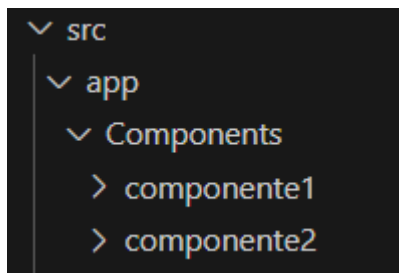
Es una buena práctica incluir una carpeta para cada tipo de elemento que creemos, en el caso de los componentes los vamos a crear todos dentro de la carpeta Components

Vamos a crear dos componentes llamados componente1 y componente2, los crearemos en una carpeta llamada components para separar los componentes de la estructura de carpeta principal src.

```
D:\my-app>ng g c /Components/componente1
CREATE src/app/Components/componente1/componente1.component.html (27 bytes)
CREATE src/app/Components/componente1/componente1.component.spec.ts (654 bytes)
CREATE src/app/Components/componente1/componente1.component.ts (266 bytes)
CREATE src/app/Components/componente1/componente1.component.css (0 bytes)

D:\my-app>ng g c /Components/componente2
CREATE src/app/Components/componente2/componente2.component.html (27 bytes)
CREATE src/app/Components/componente2/componente2.component.spec.ts (654 bytes)
CREATE src/app/Components/componente2/componente2.component.ts (266 bytes)
CREATE src/app/Components/componente2/componente2.component.css (0 bytes)
```

Esto nos ha creado dos componentes dentro de la carpeta Components dentro de src. Y en cada componente los cuatro archivos



2.4.- Uso de componentes

Una vez que ya hemos creado nuestro componente estará listo para poder utilizarse, para ello en el decorador se ha definido la propiedad selector.

El **componente app.component** es el que inicia la aplicación. En este caso la propiedad selector: **"app-root"** permite el uso de este componente dentro de otro con esta invocación **<app-root></app-root>**.

Particularidades del componente raíz:

- Su nombre es AppComponent, y su selector se llama **app-root**
- Está destinado a ser usado en la página principal, en el **index.html**.

```

<> index.html X
src > <> index.html > ...
  1  <!doctype html>
  2  <html lang="en">
  3  <head>
  4    <meta charset="utf-8">
  5    <title>MyApp</title>
  6    <base href="/">
  7    <meta name="viewport" content="width=device-width, initial-scale=1">
  8    <link rel="icon" type="image/x-icon" href="favicon.ico">
  9  </head>
 10  <body>
 11    <app-root></app-root>
 12  </body>
 13  </html>
 14

```

Para cualquier otro componente podemos hacer uso de él mediante la inclusión de su selector en html, de esta manera lo que estamos haciendo es llamar a ese componente para que se cargue en el lugar que le indicamos

Vamos a llamar a los componentes 1 y 2 en el html del app.component, para ello será necesario que en el ts de app.component importemos los dos componentes.

```

import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { Componente1Component } from
'./Components/componente1/componente1.component';
import { Componente2Component } from
'./Components/componente2/componente2.component';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet, Componente1Component, Componente2Component],
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
}

```


Ahora en el html ya podemos incluir los selectores de los dos componentes.

```
<h2>App Component</h2>
<app-componente1></app-componente1>
<app-componente2></app-componente2>
```

Si no importamos los componentes en el decorador no se reconocerán las etiquetas de los selectores de los componentes 1 y 2

El resultado es:

App Component

componente1 works!

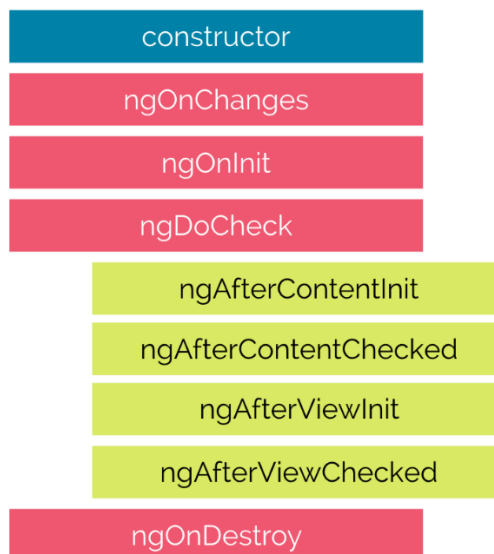
componente2 works!

De esta manera estamos anidando componentes y manteniendo una relación padre hijo entre componentes.

2.5.- Ciclo de vida de los componentes

En Angular, los componentes no son solo bloques de construcción de la interfaz, sino que también tienen un ciclo de vida similar al de un ser vivo. Este ciclo abarca desde su creación hasta su destrucción, pasando por diferentes etapas que podemos aprovechar para controlar su comportamiento.

El ciclo de vida de un componente en Angular se compone de 8 etapas:



1. **ngOnChanges:** Se ejecuta cuando se detecta un cambio en las propiedades de entrada del componente.
2. **ngOnInit:** Se ejecuta una vez, después de que el componente ha sido inicializado y sus propiedades han sido establecidas.
3. **ngDoCheck:** Se ejecuta después de cada detección de cambios, lo que permite realizar comprobaciones personalizadas.
4. **ngAfterContentInit:** Se ejecuta una vez, después de que el contenido del componente (plantillas hijas) haya sido inicializado.
5. **ngAfterContentChecked:** Se ejecuta después de cada detección de cambios en el contenido del componente.
6. **ngAfterViewInit:** Se ejecuta una vez, después de que la vista del componente (DOM) haya sido renderizada.
7. **ngAfterViewChecked:** Se ejecuta después de cada detección de cambios en la vista del componente.
8. **ngOnDestroy:** Se ejecuta antes de que el componente sea destruido, lo que permite realizar tareas de limpieza.

Cada etapa del ciclo de vida se asocia a un **hook** "gancho" (método especial) que podemos implementar en nuestro componente. Estos hooks nos permiten ejecutar código en momentos específicos del ciclo de vida, como:

- **ngOnChanges:** Este evento se ejecuta cada vez que se cambia un valor de un input control dentro de un componente. Se activa primero cuando se cambia el valor de una propiedad vinculada. Siempre recibe un change data map o mapa de datos de cambio, que contiene el valor actual y anterior de la propiedad vinculada envuelta en un SimpleChange
- **ngOnInit:** Se ejecuta una vez que Angular ha desplegado los data-bound properties(variables vinculadas a datos) o cuando el componente ha sido inicializado, una vez que ngOnChanges se haya ejecutado. Este evento es utilizado principalmente para inicializar la data en el componente.
- **ngDoCheck:** Se activa cada vez que se verifican las propiedades de entrada de un componente. Este método nos permite implementar nuestra propia lógica o algoritmo de detección de cambios personalizado para cualquier componente.

- **ngAfterContentInit:** Se ejecuta cuando Angular realiza cualquier muestra de contenido dentro de las vistas de componentes y justo después de ngDoCheck. Actuando una vez que todas las vinculaciones del componente deban verificarse por primera vez. Está vinculado con las inicializaciones del componente hijo.
- **ngAfterContentChecked:** Se ejecuta cada vez que el contenido del componente ha sido verificado por el mecanismo de detección de cambios de Angular; se llama después del método ngAfterContentInit. Este también se invoca en cada ejecución posterior de ngDoCheck y está relacionado principalmente con las inicializaciones del componente hijo.
- **ngAfterViewInit:** Se ejecuta cuando la vista del componente se ha inicializado por completo. Este método se inicializa después de que Angular ha inicializado la vista del componente y las vistas secundarias. Se llama después de ngAfterContentChecked. Solo se aplica a los componentes.
- **ngAfterViewChecked:** Se ejecuta después del método ngAfterViewInit y cada vez que la vista del componente verifique cambios. También se ejecuta cuando se ha modificado cualquier enlace de las directivas secundarias. Por lo tanto, es muy útil cuando el componente espera algún valor que proviene de sus componentes secundarios.
- **ngOnDestroy:** Este método se ejecutará justo antes de que Angular destruya los componentes. Es muy útil para darse de baja de los observables y desconectar los event handlers para evitar memory leaks o fugas de memoria.

Lo más habitual suele ser utilizar el constructor para realizar la inyección de dependencias (servicios, router...) en el componente y los hooks ngOnInit para utilizar inicialmente esa inyección y el hook ngOnDestroy para dar de baja los observables

Si queremos usar estos hooks los deberemos de implementar en el fichero ts del componente

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

@Component({
  selector: 'app-componente1',
  standalone: true,
  imports: [],
  templateUrl: './componente1.component.html',
  styleUrls: ['./componente1.component.css']
})
export class Componente1Component {
  constructor(){
    console.log("Se ejecuta constructor")
  }
  ngOnInit(){
    console.log("Se ejecuta ngOnInit")
  }
}
```

3.- Proceso de arranque

A partir de la versión 17 de Angular y no utilizar módulos el arranque se realiza indicando en el fichero main.ts cuál es el componente que arranca la aplicación

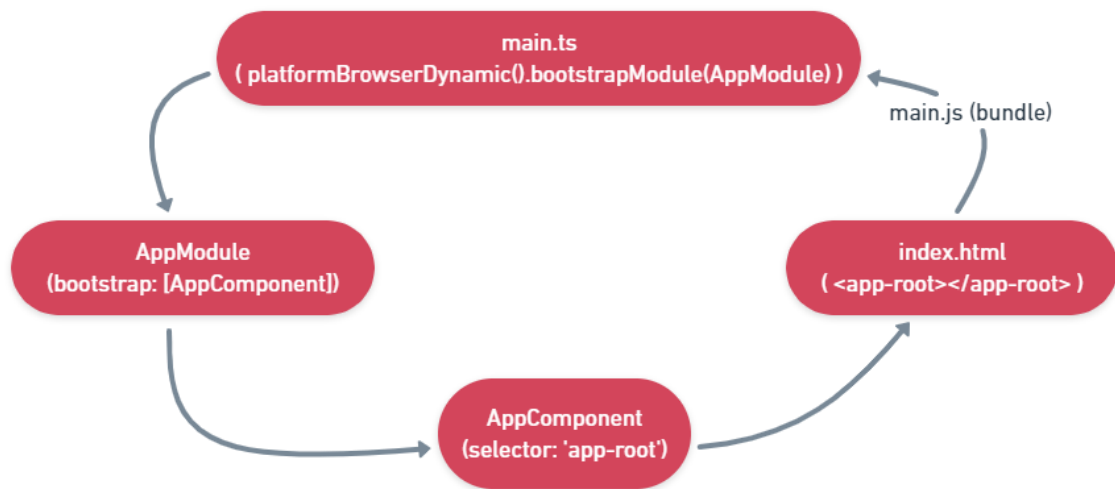
```
import { bootstrapApplication } from '@angular/platform-browser';
import { appConfig } from './app/app.config';
import { AppComponent } from './app/app.component';

bootstrapApplication(AppComponent, appConfig)
  .catch((err) => console.error(err));
```

Y añadiendo ese componente en fichero index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>MyApp</title>
  <base href="/">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>
  <app-root></app-root>
</body>
</html>
```

En versiones anteriores a la 17 se sigue el siguiente esquema



En este caso al existir un módulo principal AppModule en el proceso de arranque se indica que el modulo que arranca el proyecto es AppModule. Dentro de AppModule se indica que el componente que arranca la app es AppComponent.