

Angular

IFCD65

Introducción

Angular es una plataforma que permite desarrollar aplicaciones web en la sección cliente utilizando HTML y JavaScript para que el cliente asuma la mayor parte de la lógica y descargue al servidor con la finalidad de que las aplicaciones ejecutadas a través de Internet sean más rápidas.



Introducción

Permite la creación de aplicaciones web de una sola página (SPA: single-page application) realizando la carga de datos de forma asíncrona

Angular está orientado a objetos, trabaja con clases y favorece el uso del patrón MVC Modelo Vista Controlador.

Permite el uso de TypeScript Microsoft (lenguaje desarrollado por) con las ventajas que supone poder disponer de un tipado estático y objetos basados en clases. Gracias a un compilador se traduce el código escrito en TypeScript a JavaScript.

Introducción

Permite la creación de aplicaciones web de una sola página (SPA: single-page application) realizando la carga de datos de forma asíncrona

Angular está orientado a objetos, trabaja con clases y favorece el uso del patrón MVC Modelo Vista Controlador.

Permite el uso de TypeScript Microsoft (lenguaje desarrollado por) con las ventajas que supone poder disponer de un tipado estático y objetos basados en clases. Gracias a un compilador se traduce el código escrito en TypeScript a JavaScript.

Introducción SPA

Patrón de diseño single-page application (SPA) es una evolución del patrón de diseño MPA + AJAX, pero llevando al extremo el uso de AJAX. Hasta el punto de que en el cliente se carga una única página que se modifica desde el propio cliente (navegador) según las acciones de usuario. Por tanto, toda la navegación por las distintas pantallas o interfaces de la aplicación se realizará sin salir de esa única página.

Introducción SPA

Una de las principales ventajas de las aplicaciones SPA respecto las MPA es la mejora de experiencia de usuario debido a la reducción en el tiempo de respuesta ante las acciones del usuario. Esto se consigue gracias a que:

- Ya no se crean páginas completas por cada acción del usuario.
- Solo se intercambia la información necesaria con el servidor.

Introducción SPA

La responsabilidad del aspecto de la aplicación recae principalmente en la parte cliente, mientras que el servidor tiene la función de ofrecer al cliente una API de servicios para dar acceso a la base de datos de la cual se alimenta la aplicación. Los datos intercambiados entre cliente y servidor suelen estar en formato JSON, que es un formato más óptimo que el tradicional XML.

Instalación

PASO 1

Instalación de Node.js

<https://nodejs.org/es>

(descargar la versión estable)

Probar en la línea de comandos:

`node -v`

PASO 2

Instalamos Angular CLI

Escribimos en la línea de comandos:

`npm install -g @angular/cli`

(necesitamos permisos admin, (sudo en Mac)

`ng version`

Instalación

PASO 3

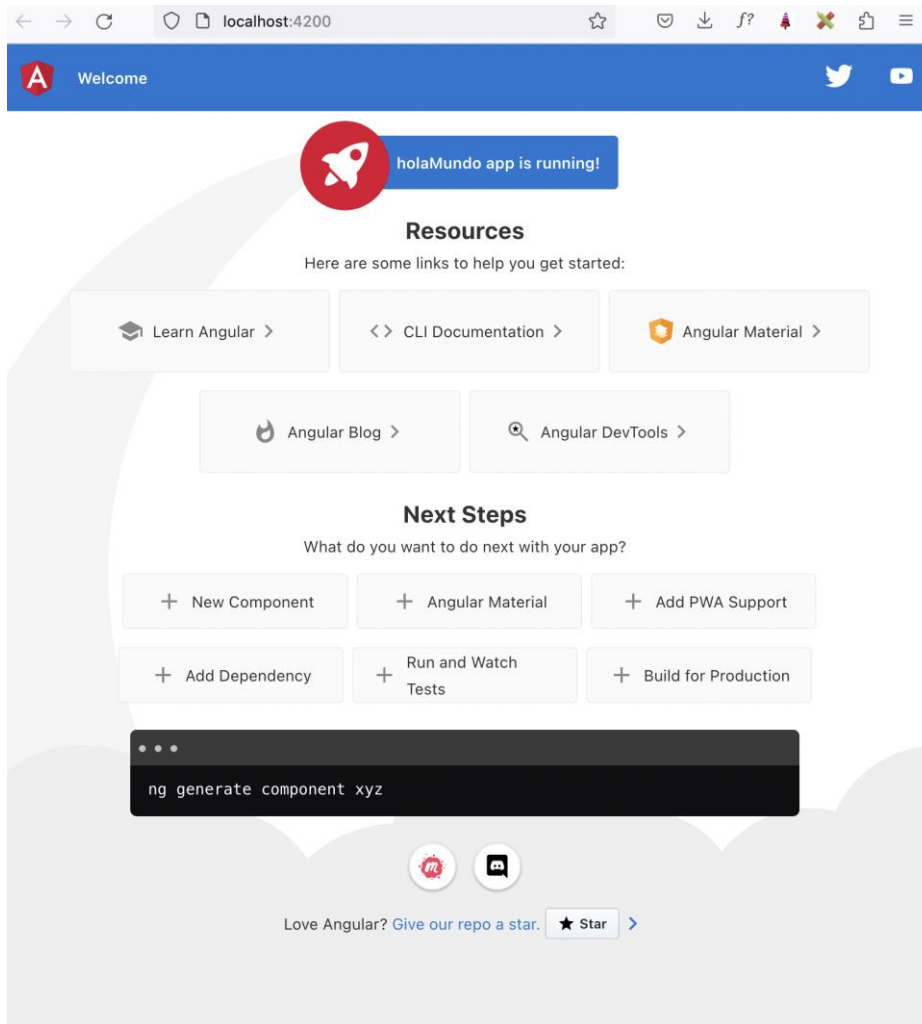
Creamos una carpeta donde
programaremos nuestros proyectos

Accedemos a ella desde la línea de comandos y ejecutamos:
`ng new holaMundo`

No angular routing
Elegimos CSS

OJO, tuve que añadir
`npm config set legacy-peer-deps true`
`sudo chown -R 501:20 "/Users/borjamulleras/.npm"`

Instalación



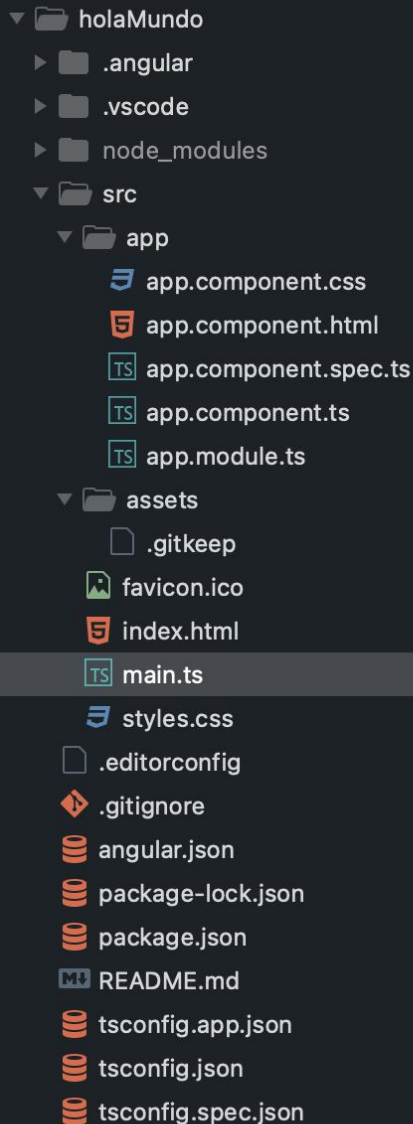
PASO 4

Accedemos al proyecto holaMundo en la línea de comandos
Ejecutamos:
`ng serve --open`

El comando inicia el servidor, observa tus archivos, y reconstruye la aplicación a medida que realizas cambios en esos archivos.

Se abre automáticamente tu navegador en <http://localhost:4200/>.

FOLDERS



Primeros pasos

Carpeta node_modules es el corazón de angular. No nos interesa para desarrollar proyectos.

Cuando se suba el repositorio sino la tenemos podemos ejecutar `npm install`

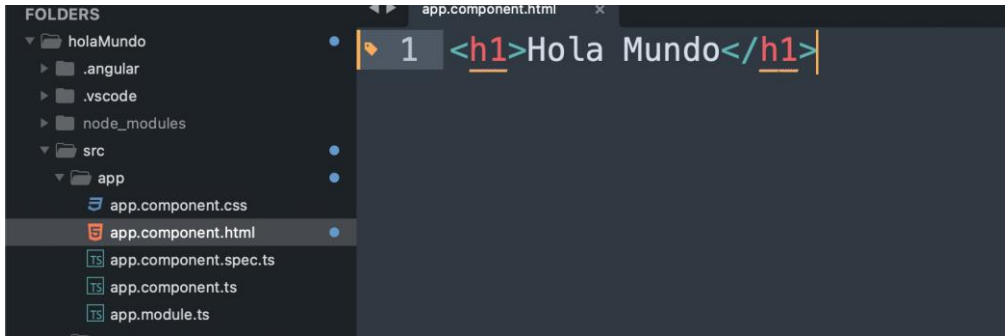
Carpeta SRC es la que se trabaja

app es donde se encuentran los componentes que son las partes de las páginas.

Podemos borrar el contenido del archivo:

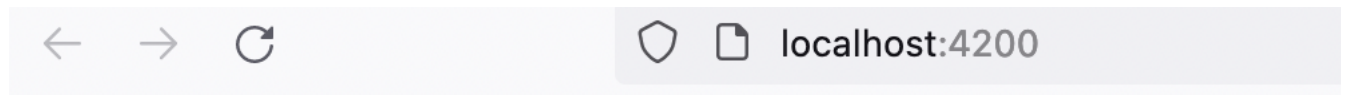
App.component.html y ponerle
`<h1>Hola Mundo</h1>`

Primeros pasos



```
1 <h1>Hola Mundo</h1>
```

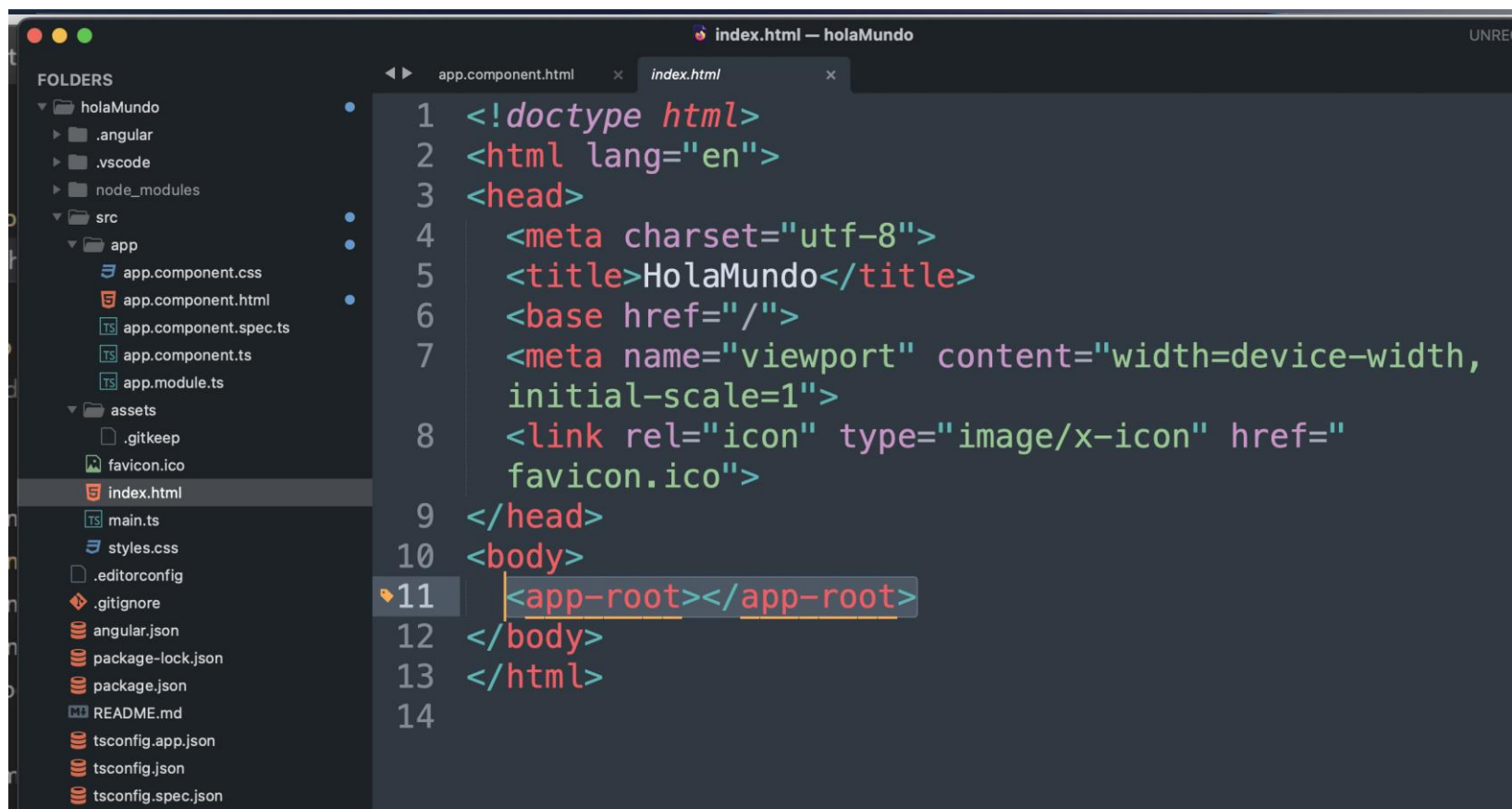
Solo se pone el contenido del componente, no la estructura de la Página (html, head, body)



Hola Mundo

Primeros pasos

En src/index.html está la estructura del documento y en <app-root> es donde se coloca el componente



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

FOLDERS

- ▼ holaMundo
 - ▶ .angular
 - ▶ .vscode
 - ▶ node_modules
 - ▼ src
 - ▼ app
 - app.component.css
 - app.component.html
 - app.component.spec.ts
 - app.component.ts
 - app.module.ts
 - ▼ assets
 - .gitkeep
 - favicon.ico
 - index.html
 - main.ts
 - styles.css
 - .editorconfig
 - .gitignore
 - angular.json
 - package-lock.json
 - package.json
 - README.md
 - tsconfig.app.json
 - tsconfig.json
 - tsconfig.spec.json

Primeros pasos

app/app.component.css

Archivo de CSS del componente

app/app.component.spec.ts

Archivo para pruebas unitarias, de momento no lo utilizamos

app/app.component.ts

Archivo de TypeScript que contiene elementos necesarios para la programación del componente

app/app.module.ts

Datos que definirán un módulo.

Cuando añadamos más componentes tendremos que tocarlo.

(más adelante lo vemos).

Primeros pasos

app/app.component.ts

Ponemos el siguiente código

```
import { Component } from '@angular/core';
//Decorador de Angular
@Component({
  //La etiqueta de index.html donde se inyecta el contenido.
  selector: 'app-root',
  //Plantilla del componente de HTML
  templateUrl: './app.component.html',
  //CSS del componente. Es una lista por si hay más CSS
  styleUrls: ['./app.component.css']
})
//Si usamos el componente fuera hay que exportarlo.
//Es un objeto de JS y podemos definir las propiedades que queramos
export class AppComponent {
  //Propiedades
  nombre='Borja';
  apellidos="Mulleras";
}
```

Primeros pasos

angular data-binding



```
1 <h1>Hola Mundo</h1>
2 <ul>
3   <li>Nombre: {{nombre}}</li>
4   <li>Apellido: {{apellido}}</li>
5 </ul>
```



Hola Mundo

- Nombre: Borja
- Apellido: Mulleras

Primeros pasos

Muy Importante

Pondemos el nombre de los archivos en minúsculas:

`app.component.ts`

y la clase se llamará:

```
export class AppComponent {  
  //Propiedades  
  nombre='Borja';  
  apellido1="Mulleras";  
}
```

La A y la C son mayúsculas
El nombre es igual que el archivo

Primeros pasos

Definición de la clase con TypeScript

```
export class AppComponent {  
  //definición de las propiedades  
  public nombre: string;  
  public apellido1: string;  
  //constructor  
  constructor() {  
    this.nombre = 'Borja';  
    this.apellido1='Mulleras';  
  }  
}
```

Primeros pasos

Main.ts

Indica a Angular que módulo es el que tiene que cargar inicialmente

Style.css

Estilos generales a todos los componentes

Los CSS de los componentes solo afectan a los componentes

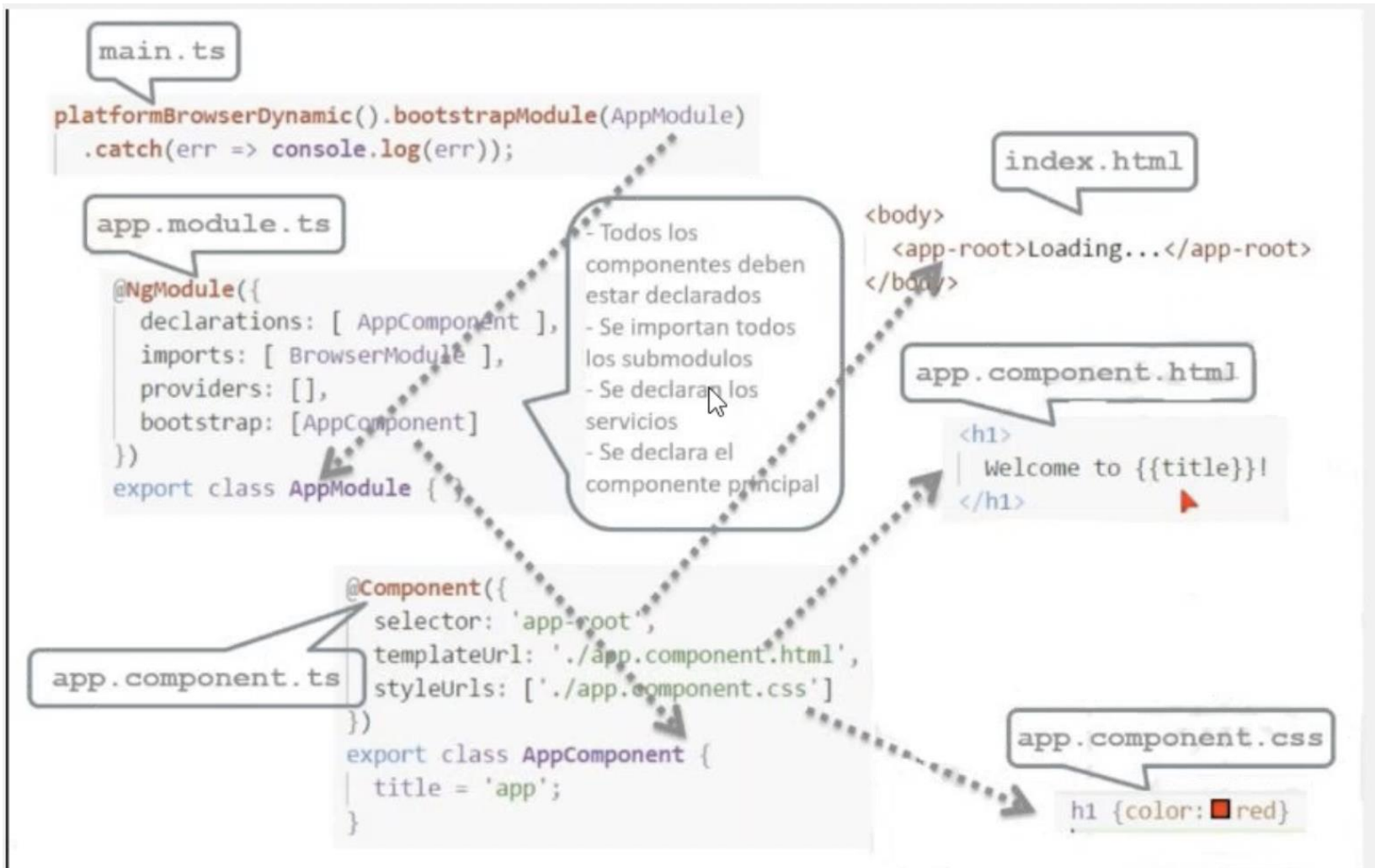
Test.ts

Sirve para hacer pruebas unitarias.

Carpeta assets

Es donde guardaremos las imágenes, videos, fuentes de letra, etc..

Esquema arranque de un proyecto



Angular 1

Crear y modificar el proyecto holaMundo añadiendo css, html, imagenes y modificando las propiedades del objeto de JavaScript usando siempre el componente inicial.

Componentes

- Angular esta basado en componentes.
- Los componentes son reutilizables en otros proyectos a bajo coste de mantenimiento.
- Un componente consite en un conjunto de archivos interconectados entre ellos y que tienen un propósito común.
- Todas las aplicaciones tienen como mínimo un componente y un módulo.
- El componente por defecto es AppComponent y se encuentra en src->app

Componentes

Los componentes están formados por los siguientes archivos:

- `app.component.css` (opcional)
- `app.component.html` (opcional)
- `app.component.spec.ts` (opcional): Código para el test unitario del componente
- `app.component.ts`: describe el comportamiento del componente en TypeScript. Se trata de una clase que sigue el paradigma de la programación orientada a objetos. (Hay componentes que solo tienen este archivos, son los servicios)

`app` es el nombre del componente.

Componentes

Definición componente

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

//Decorador de Angular
@Component({
  //La etiqueta de index.html donde se inyecta el contenido.
  selector: 'app-root',
  //Plantilla del componente de HTML
  templateUrl: './app.component.html',
  //CSS del componente. Es una lista por si hay más CSS
  styleUrls: ['./app.component.css']
})
```


Componentes

Lógica de negocio

```
//Si usamos el comonente fuera hay que exportarlo.  
//Es un objeto de JS y podemos definir las propiedades que queramos  
export class AppComponent {  
  //definición de las propiedades  
  public nombre: string;  
  public apellido1: string;  
  //constructor  
  constructor() {  
    this.nombre = 'Borja';  
    this.apellido1='Mulleras';  
  }  
}
```

Módulos

- Un módulo no es más que un fichero de TypeScript (una clase) que utiliza un decorador (annotation) especial (`@NgModule`) que lo identifica como módulo.
- Estructurar la aplicación mediante módulos es muy buena práctica ya que nos ayuda con el escalado de las aplicaciones, manteniendo la complejidad bajo control, facilitando un bajo acoplamiento y una alta cohesión.
- No obstante, nuestra idea inicial es que es un elemento que aumenta la complejidad, pero con el paso del tiempo veremos su utilidad.

Módulos

En un módulo encontramos

- Componentes, servicios, pipes y demás elementos que pertenecen al módulo. (src->app.modules.ts declarations)
- Aquello que queremos que sea visible des de otros módulos (exportable)
- Lo que el módulo necesita de otros módulos (importable)

Módulos

```
//componente para definie el propio módulo  
import { NgModule } from '@angular/core';
```

```
//componentes para visualizar el código en el browser  
import { BrowserModule } from '@angular/platform-browser';
```

```
//componentes del propio módulo  
import { AppComponent } from './app.component';
```

Módulos

El decorador @NgModule es una función que recibe un objeto como parámetro, con las siguientes propiedades:

- declarations: array con los componentes que conforman el módulo.
- imports: array con los módulos requeridos en el presente módulo.
- Providers: array con los servicios requeridos en el presente módulo
- Bootstrap: componente inicial del módulo.

Realizamos el export para que se pueda ser accesibles desde otros puntos del proyecto.

```
@NgModule({  
  
  declarations: [  
    AppComponent  
  ],  
  imports: [  
    BrowserModule  
  ],  
  providers: [],  
  bootstrap: [AppComponent]  
})
```

```
export class AppModule { }
```

Módulos

En el fichero main.ts se indica cual es el módulo de arranque de la aplicación.

Todas las aplicaciones de angular tienen como mínimo un módulo y el de por defecto es app.module.

Generar un componente

En la línea de comando en proyecto determinado ponemos:
ng generate component ruta\nombre_componente

Lo vamos a generar manualmente para ver lo que hace esta línea de comandos.

PASO 1

Dentro de la carpeta src/**app** de nuestro proyecto creamos la carpeta **componentes**

PASO2

Dentro de la carpeta componentes creamos la carpeta **cabecera**

PASO 3

Dentro de la carpeta header vamos a crear el archivo **header.components.ts**

Generar un componente

PASO 4

Con el editor de texto
escribimos en
header.components.ts

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

//Decorador de Angular
@Component({
  //La etiqueta de index.html donde se inyecta el contenido.
  selector: 'app-cabecera',
  //Plantilla del componente de HTML. Hay dos opciones
  templateUrl: './cabecera.component.html',
  //o bien podemos poner directamente el código html
  //template: `<h2>Cabecera</h2>`
})

export class CabeceraComponent { }
```

PASO 5

Añadir en el app.component.html la etiqueta app-cabecera

Importante: no se coloca en el index.html, allí solo va la etiqueta inicial.

Generar un componente

PASO 6

Los componentes tienen que estar asociados a un módulo. En este caso lo asociamos a `app.module.ts`

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

import { AppComponent } from './app.component';
import { CabeceraComponent } from
'./componentes/cabecera/cabecera.component';

@NgModule({
  declarations: [
    AppComponent,CabeceraComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Angular 2

Generar 2 componentes extra, un body y un footer.

A lmenos el footer hacerlo con un archivo
footer.component.html y otro footer.component.css

Componentes

ng generate component componentes/pie

ng g c componentes/pie

Queremos poner un
@2023 todos los derechos reservados

PASO1

Modificamos el archivo pie.component.html

<p>@{{anyo}} Todos los derechos reservados </p>

Componentes

PASO 2

Añadimos la propiedad y el cálculo en pie.component.ts

```
export class PieComponent {  
  public anyo:number;  
  
  constructor() {  
    this.anyo = new Date().getFullYear();  
  }  
}
```

Directivas estructurales

Las directivas estructurales *ngIf y *ngFor

Una directiva es un componente, pero que carece de html asociado

Gracias a estos componentes podemos crear aplicaciones que realicen cierta lógica en la capa de presentación (plantilla)

PASO 1

Creamos un proyecto nuevo
ng new directivas

(OJO!!! Ponerlos fuera del proyecto holaMundo)

Directivas estructurales

PASO 2

Borramos el contenido de app.component.html

PASO 3

Creamos los componentes:

ng g c componentes/cabecera

ng g c componentes/cuerpo

ng g c componentes/pie

PASO 4

Borramos los archivos de test de los 3 componentes

PASO 5

Añadir en app.component.html

<app-cabecera></app-cabecera>

<app-cuerpo></app-cuerpo>

<app-pie></app-pie>

Directivas estructurales

PASO 6

Podemos poner el html que queramos en cabecera y en el pie.
En cuerpo.component.html vamos a añadir un código con 2 columnas:

cuerpo.component.html

```
<div class="cuerpo">
<div>
<h2>Directiva *ngIf</h2>
  <div *ngIf="mostrar">
    <h3>{{titulo}}</h3>
    <p>{{contenido}}</p>
  </div>
  <button (click)="mostrarOcultar();">
    Mostrar/Ocultar</button>
</div>
<div>
<h2>Directiva *ngFor</h2>
</div> </div>
```

cuerpo.component.css

```
.cuerpo {
  display: grid;
  grid-template-columns: 50% 50%;
  max-width: 1200px;
  margin: 0 auto;
}
```

Directivas estructurales

***ngIf**

PASO 7

Añadimos en cuerpo.component.ts en el objeto las propiedades titulo y contenido (hay que poner el data-binding en el html correspondiente)

```
....  
export class CuerpoComponent {  
  public titulo:string;  
  public contenido:string;  
  
  constructor(){  
    this.titulo='titulo';  
    this.contenido='contenido';  
  }  
.....
```


Directivas estructurales

***ngIf**

PASO 8

La directiva *ngIf funciona que si vale true se muestra la caja de html donde se encuentra si vale false no se muestra.

```
<div *ngIf="true">                </div>
```

PASO 9

Añadimos una propiedad mostrar en cuerpo.component.ts

```
export class CuerpoComponent {  
  public titulo:string;  
  public contenido:string;  
  
  public mostrar:boolean;  
  
  constructor(){  
    this.titulo='titulo';  
    this.contenido='contenido';  
    this.mostrar=true;  
  }  
}
```

Formador: Borja Mulleras Vinzia

Directivas estructurales

***ngIf**

PASO 10

Ponemos el nombre de la propiedad en cuerpo.component.html
OJO!!! No es data-binding, es sin las llaves {{}}

```
<div *ngIf="mostrar">      </div>
```

Queremos que la directiva interprete el valor de mostrar

PASO 11

En el botón vamos a incluir el evento onclick.

Con Angular los eventos se definen:

```
<button (click)="mostrarOcultar();">Mostrar/esconder</button>
```

Directivas estructurales

***ngIf**

PASO 12

Añadimos en cuerpo.component.ts el método public mostrarOcultar() en la clase CuerpoComponent

```
public mostrarOcultar() {  
    /* if (this.mostrar) {  
        this.mostrar=false;  
    } else {  
        this.mostrar=true;  
    } */  
    this.mostrar = !this.mostrar;  
}
```

Directivas estructurales

***ngIf**

PASO 13

Modificamos el texto del botón para que muestre Mostrar o Ocultar en función de cada caso.

```
<button (click)="mostrarOcultar();">
  <span *ngIf="!mostrar">Mostrar</span>
  <span *ngIf="mostrar">Ocultar</span>
</button>
```

Angular 3

***ngIf**

Realizar un botón que muestre o esconda una imagen en la página web.

Directivas estructurales

Directiva *ngFor

PASO 1

Añadimos en cuerpo.component.html una lista de elementos

```
<div>  
  <h2>Directiva *ngFor</h2>  
  <ul>  
    <li>Lorem Ipsum</li>  
    <li>Lorem Ipsum</li>  
    <li>Lorem Ipsum</li>  
    <li>Lorem Ipsum</li>  
    <li>Lorem Ipsum</li>  
    <li>Lorem Ipsum</li>  
  </ul>  
</div>
```

Directivas estructurales

Directiva *ngFor

PASO 2

En la clase CuerpoComponent de cuerpo.component.ts añadimos:

```
public colores:string[];
```

```
    constructor(){  
        this.titulo='titulo';  
        this.contenido='contenido';  
        this.mostrar=true;
```

```
this.colores=['rojo','azul','verde','amarillo','naranja','rosa'];  
    }
```

Directivas estructurales

Directiva *ngFor

PASO 3

En cuerpo.component.html sustituimos los li's por

```
<li *ngFor="let color of colores">{{color}}</li>
```

Let color of colores es como un for of

Y si queremos mostrar el índice:

```
<li *ngFor="let color of colores; let i=index;">{{i}}: {{color}}</li>
```


Angular 4

***ngFor**

Mostrar un listado de coches en la pantalla

Directivas estructurales

Directiva *ngIf combinada con ngTemplate

PASO 1

En la clase CuerpoComponent de cuerpo.component.ts añadimos:

```
public numero:number;  
como propiedad y en el constructor:  
this.numero: 7;
```

PASO 2

En cuerpo.component.html ponemos:

```
<div>  
  <h2>Directiva *ngIf combinada con ngTemplate</h2>  
  <span *ngIf="numero % 2==0">Es numero par</span>  
  <span *ngIf="numero % 2==1">Es numero impar</span>  
</div>
```

*El *ngIf debe dar un resultado de true o false*

Directivas estructurales

Directiva *ngIf combinada con ngTemplate

PASO 3

Cambiamos el código de cuerpo.component.html por:

```
<span *ngIf="numero % 2==0; then par else impar"></span>  
<ng-template #par>Es par</ng-template>  
<ng-template #impar>Es impar</ng-template>
```

Directivas no estructurales

ngStyle

Nos permite cambiar estilos de forma dinámica.

```
<p [ngStyle]="{'font-size':'32px','color':'red'}">Hola</p>
```

Los corchetes en la directiva se ponen porque asignamos un valor a la directiva

Y ponemos las distintas propiedades entre llaves ya que estamos definiendo un objeto

PASO 1

Definimos en cuerpo.component.ts y el objeto CuerpoComponent la propiedad tam e inicializamos el valor a 40px en el constructor.

PASO 2 – Opción 1

Cambiamos el código de arriba por:

```
<p [ngStyle]="{'font-size': tam+'px','color':'red'}">Opción 1</p>
```

Directivas no estructurales

ngStyle

PASO 3 – Opción 2

Añadimos:

```
<p [style.font-size]="tam +'px'">Opción 2</p>
```

PASO 4 – Opción 3

```
<p [style.font-size.px]="tam">Opción 3</p>
```

Angular 5

ngStyle

Crear dos botnoes + y – que aumente la letra de 2 en 2 o lo disminuya de 2 en 2.

Directivas no estructurales

ngClass

Sirve para activar/desactivar clases

```
<div [ngClass]=" ' letraArial azul ' "> ... </div>
```

```
<div [ngClass]="['letraArial','azul']">...</div>
```

```
<div [ngClass]="{'letraArial': true, 'Azul': true}">...</div>
```

PASO 1

Añadimos clases en cuerpo.component.css

```
.letraArial {font-family: arial;}  
.azul {color: blue;}  
.rojo {color: red;}
```

Directivas no estructurales

ngClass

PASO 2

Añadimos el siguiente código en cuerpo.components.html

Donde color es el nombre de una variable que contiene el nombre de la clase.
Esta variable la controlamos en cuerpo.component.ts

```
<div>  
  <h2>Directiva ngClass</h2>  
  <div class="letraArial" [ngClass]="color">  
    Texto de prueba  
  </div>  
</div>
```


Angular 6

ngClass

Añadir tres botones que nos cambie el color de un texto, con los valores rojo y azul

Podéis probar realizar la orden directa en el botón, sin necesidad de crear un método

```
<button (click)="color = 'azul' ">Azul</button>
```

Procesos asincronos

Vamos a hacer que el color del botón cambie cuando lo apretemos

```
cuerpo.component.html
<div>
<h2>Procesos asincronos</h2>
<button (click)="guardarCambios()" [ngClass]="{'rojo':!guardar,'azul':guardar}">
    Guardar cambios
</button>
</div>
```

Procesos asincronos

Añadimos la propiedad guardar = true en el objeto CuerpoComponent.

Y añadimos el método guardarCambios

```
public guardarCambios() {  
    this.guardar=!this.guardar;  
}
```

Procesos asincronos

Vamos a simular que hacemos un paron de 3 segundos.

```
public guardarCambios() {  
    this.guardar=!this.guardar;  
    //simulamos 3 segundos de consulta en el servidor  
    setTimeout(() => {this.guardar=!this.guardar; },3000);  
}
```

Angular 7

Cambiar el texto de Guardar por Guardando durante los 3 segundos y después ponga Cambios guardados

Pipes

Los *pipes* son una herramienta de Angular que nos permite **transformar** visualmente la información, por ejemplo, cambiar un texto a mayúsculas o minúsculas, o darle formato de fecha y hora.

Angular trae sus propias pipes, pero también permite crear pipes nuevas a los desarrolladores.

```
<div>{{ 'mensaje' | uppercase }}</div>
```

Vamos a crear un componente nuevo que llamaremos pipes en el proyecto

Listado de Pipes

pipes.component.ts

```
export class PipesComponent {  
  mensaje='Buenos días don José';  
  hoy = new Date();  
  numero = 2780.67890;  
  vector = [1,2,3,4,5,6,7,8];  
  porcentaje =0.3545;  
  sueldo = 1266.10;  
}
```

Creamos estas variables en el objeto PipesComponent

Listado de Pipes

pipes.component.html

```
<hr>
<h2>Uso de Pipes</h2>
<h3>uppercase</h3>
<div>{{ 'mensaje' | uppercase }}</div>
<div>uppercase -> {{ mensaje |
uppercase }}</div>
<div>slice:2 ->{{ mensaje | slice:2 }}</div>
<div>slice:2:5 ->{{ mensaje | slice:2:5 }}</div>
<div>Slice aplicado a arrays<br>
array {{vector}} Posiciones del 3 al 5-> {{ vector |
slice:3:5 }}
</div>
<h3>Dates</h3>
<div>date -> {{ hoy | date }}</div>
<div>date:"dd/MM/yy" -> {{hoy |
date:"dd/MM/yy"}}</div>
```

uppercase

MENSAJE

uppercase -> BUENOS DÍAS DON JOSÉ

slice:2 ->enos días don José

slice:2:5 ->eno

Slice aplicado a arrays

array 1,2,3,4,5,6,7,8 Posiciones del 3 al 5-> 4,5

Dates

date -> 18 may 2023

date:"dd/MM/yy" -> 18/05/23

Listado de Pipes

pipes.component.html

<h3>Números</h3>

<div>numero | number -> {{numero | number}}

<div>numero:'N1:N2-N3' | number -> {{numero |
number:'1.2-2'}}

<hr>

N1 es la cantidad de números enteros. Por defecto 1

N2 es la cantidad mínima de números decimales. Por defecto 0

N3 es la cantidad máxima de números decimales. Por defecto 0

</div>

<h3>Percent</h3>

<div>porcentaje:'N1:N2-N3' | percent -> {{porcentaje | percent:'1.2-2'}}

Números

numero | number -> 2.780,679

numero:'N1:N2-N3' | number -> 2.780,68

N1 es la cantidad de números enteros. Por defecto 1

N2 es la cantidad mínima de números decimales. Por defecto 0

N3 es la cantidad máxima de números decimales. Por defecto 0

Percent

porcentaje:'N1:N2-N3' | percent -> 35,45 %

Listado de Pipes

pipes.component.html

```
<h3>Divisas</h3>
```

```
<div>Currency {{sueldo | currency}}</div>
```

Divisas

Currency \$1,266.10

Los datos y los números salen en formato americano

Hay que cambiar el app.module.ts para añadir la configuración del módulo europeo

Configuración locale ES

app.module.ts

```
import { LOCALE_ID, NgModule } from '@angular/core';  
import { BrowserModule } from '@angular/platform-browser';  
import { AppComponent } from './app.component';  
import localeEs from '@angular/common/locales/es';  
import { registerLocaleData } from '@angular/common';  
registerLocaleData(localeEs);
```

...

```
@NgModule({  
  declarations: ...,  
  imports: ...  
  providers: [{provide:LOCALE_ID,useValue: 'es'}],  
  bootstrap: [AppComponent]  
})
```

Listado de Pipes

pipes.component.html

```
<h3>Divisas</h3>
<div>Currency {{sueldo | currency}}</div>
<div>Currency {{sueldo | currency:'EUR'}}</div>
<div>Currency {{sueldo | currency:'EUR':false:'1.2-2'}}</div>
```

Divisas

Currency 1.266,10 US\$

Currency 1.266,10 €

Currency 1.266,10 EUR

Listado de Pipes

pipes.component.js

...

```
public personas = [{"nombre":"Borja",  
                    "apellido1":"Mulleras",  
                    "apellido2":"Vinzia",  
                    "sexo":"Hombre"},  
                    {"nombre":"Rosa",  
                    "apellido1":"Berlanga",  
                    "apellido2":"Muñoz",  
                    "sexo":"Mujer"},  
                    {"nombre":"Ramon",  
                    "apellido1":"Lopez",  
                    "apellido2":"Martinez",  
                    "sexo":"Hombre"}]
```

...

pipes.component.html

```
<h3>JSON</h3>  
<div>JSON {{personas}} {{personas |  
json}}
```

JSON

```
JSON [object Object],[object Object],[object Object] [ { "nombre": "Borja",  
"apellido1": "Mulleras", "apellido2": "Vinzia", "sexo": "Hombre" }, { "nombre":  
"Rosa", "apellido1": "Berlanga", "apellido2": "Muñoz", "sexo": "Mujer" }, {  
"nombre": "Ramon", "apellido1": "Lopez", "apellido2": "Martinez", "sexo":  
"Hombre" } ]
```

Pipes

Vamos a desarrollar un pipe propio

PASO 1

Vamos a la carpeta del proyecto y ejecutamos

```
ng g pipe carpeta_pipes/nombre_pipe  
ng g pipe pipes/saludo
```

Esto nos crea el archivo app/pipes/saludo.pipes.ts

```
import { Pipe, PipeTransform } from '@angular/core';  
@Pipe({  
  name: 'saludo'  
})  
export class SaludoPipe implements PipeTransform {  
  transform(value: unknown, ...args: unknown[]): unknown {  
    return null;  
  }  
}
```

Pipes

PASO 2

Cambiamos el método transform del objeto SaludoPipe

```
export class SaludoPipe implements
PipeTransform {

    transform(value: any, ...args: any[]): string
    {
        return 'Buenas '+value;
    }

}
```

Pipes

PASO 3

En pipe.component.html escribimos:

```
<h3>Pipes a medida</h3>  
<div>Mensaje en de saludo {{ nombre | saludo}}</div>
```

PASO 4

Le pasamos si es masculino o femenino

La llamada al objeto transform del objeto SaludoPipe cambiará:

Pipes

```
saludo.pipe.ts
export class SaludoPipe implements PipeTransform {

  transform(value: any, isMale:boolean=true): string {
    if (isMale==true) {
      return 'Buenos días señor '+value;
    } else {
      return 'Buenos días señora '+value;
    }
  }
}
```

Pipes

PASO 3

En pipe.component.html escribimos:

```
<h3>Pipes a medida</h3>  
<div>Mensaje en de saludo {{ nombre | saludo }}</div>
```

PASO 4

Le pasamos si es masculino o femenino

La llamada al objeto transform del objeto SaludoPipe cambiará:

pipes.component.html

```
<div>Mensaje en de saludo {{ nombre | saludo:isMale }}</div>  
<button (click)="isMale=!isMale">Cambiar sexo</button>
```

Pipes

PASO 5

Añadir en pipes.component.ts
public isMale = false;

Angular 8

Crear una pipe personalizada que nos muestre el texto entre paréntesis.

Angular IFCD65