# Angular

Instalación

Tenemos que tener disponible la última versión de Node.js, que nos permitirá descargar Angular-cli. Node.js viene con npm. Angluar-cli es un herramienta en línea de comandos bastante útil para trabajar con proyectos angular2 o angular4. Angular-cli nos creará un proyecto básico o esqueleto, incluye el compilador typescript y más.

npm install –g @angular/cli@latest

Si ya lo tenemos instalado:

npm uninstall –g @angular/cli

npm cache clean

npm install –g @angular/cli@latest

Crear proyecto con angular-cli

ng new proyecto1

cd new proyecto1

ls, para ver las carpetas y ficheros generados pro angular-cli para el proyecto. 

Fichero package.json

En este fichero podemos ver la versión de Node.js, del compilador /transpilador typescript.

Este fichero se encuentra ubicado en la carpeta raíz del proyecto, en nuestro caso en proyecto1.

Fichero main.ts

Ubicado en /src.

En este fichero es donde se indicará el módulo que será el punto de entrada de la aplicación.

/src/main.ts:

import { enableProdMode } from '@angular/core';

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';

import { environment } from './environments/environment';

platformBrowserDynamic().bootstrapModule(AppModule)

.catch(err => console.log(err));

Fichero index.html

Ubicado en /src.

Como cualquier aplicación o sitio web, los proyectos Angular disponen de un archivo index.html, ubicado en la carpeta src. Además del código habitual de un archivo html, este archivo se caracteriza por incluir la etiqueta <app-root></app-root>, que será la etiqueta del web component donde se ‘renderice’ todo el código de la aplicación. Dentro de la etiqueta raíz anterior, podemos añadir un texto que indique que la aplicación está cargando o bien un spinner, que se mostrará durante la carga de la aplicación si esta se demora. Por ejemplo: src/index.html … <app-root><p>Cargando aplicación…</p></app-root> … Si la aplicación es muy ligera, cargará tan rápido el componente raíz, que ese texto de carga, directamente no se mostrará. En la etiqueta <head></head> de este archivo, podemos añadir como en cualquier otra aplicación, CDN de librerías de fuentes e iconos así como frameworks de estilo como por ejemplo Bootstrap 4. Por ejemplo, nosotros vamos a emplear las fuentes Google y Bootstrap 4, para lo cual añadimos, dentro del <head></head>:

Fichero styles.css

Ubicado en /src.

Aquí podemos incluir todos los estilos globales, es decir para toda la aplicación. No hay que indicarlo en index.html.

Directorio assets

Ubicado en /src.

Aquí ponemos todos los fichero estáticos de la aplicación como imágnes, videos, etc..



En todas las aplicaciones Angular, existe al menos un módulo raíz, que se encuentra ubicado en el archivo app.module.ts generado por Angular CLI en el directorio src/app.

Módulos y componentes.

Módulos

Un módulo en angular es el conjunto de código dedicado a un ámbito concreto de la aplicación o funcionalidad.

En Angular, los módulos se definen mediante una clase decorada con @NgModule. Toda aplicación de Angular tendrá al menos un módulo, el llamado módulo principal o raíz (root module), que, para el caso de aplicaciones pequeñas será único. Los módulos, se definen en archivos TypeScript y podemos decir que están compuestos de tres bloques o apartados de código.

Apartado para importación

Aquí se importan librerías y paquetes de angular así como elementos externos que se emplearán en el módulo.

Apartado para el decorador

Se emplea el decorador @NgModule, que recibe un objeto mediante el cual se configuran los metadatos:

Declarations

Las declaraciones son las llamadas vistas de un módulo. Hay 3 tipos de vistas o declaraciones, los componentes, las directivas y los pipes.

Imports

En este apartado se indican las dependencias o paquetes que empleará este módulo, cuyo origen se define en las importaciones al inicio del archivo.

Providers

Son los servicios utilizados por el módulo, disponibles para todos los componentes, y que centralizan la gestión de datos o funciones para inyectarlos en los componentes.

Bootstrap

Este metadato define la vista raíz de la aplicación y es utilizado solo por el módulo raíz. No confundir con el popular framework de estilos del mismo nombre.

Apartado para la exportación

Para dar nombre al módulo y ser exportable.

Ejemplo fichero modulo

Este ejemplo ha sido generado por angular-cli para proyecto1 (/src/app/app.module.ts):

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

import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

@NgModule({

declarations: [

AppComponent

],

imports: [

BrowserModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

Componentes

Un componente controla una zona de espacio de la pantalla.

Un componente es una clase estándar de ES6 decorada con @Component.

Es equivalente en angularjs a un controlador.

En los componentes se controla la lógica de la aplicación, su vista HTML y el enlace con otros componentes.

Apartado para importación

Aquí se importan requeridos por el componente.

Apartado para el decorador

Se emplea el decorador @Component, que recibe un objeto mediante el cual se configuran los metadatos:

Selector

Define la etiqueta HTML donde se renderizará el componente.

Template

Archivo html con la vista del componente que no tiene la estructura de un archivo html, se incrustará, en este caso en el archivo index.html.

Style

Define el archivo css con los estilos del componente.

Apartado para la exportación

Para dar nombre al componente y ser exportable.

Ejemplo fichero componente

Fichero generado por angular-cli para proyecto1 /src/app/app.component.ts:

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

@Component({

selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

title = 'app';

}

Arranque del proyecto generado

npm start o ng server

Que arrancará en el puerto 4200

Para arrancar en puerto 2700: ng server –port 2700

El primer fichero typescript en ejecutarse es /src/main.ts:

import { enableProdMode } from '@angular/core';

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import { AppModule } from './app/app.module';

import { environment } from './environments/environment';

if (environment.production) {

enableProdMode();

}

platformBrowserDynamic().bootstrapModule(AppModule)

.catch(err => console.log(err));

Creación de components con angular-cli

Desde la carpeta raíz del proyecto:

ng generate component <nombrecomponente>

o de forma abreviada:

ng g c <nombrecomponente>

Este comando actualiza el fichero app.module.ts, incluyendo el componente en la sección import y en el decorador @NgModule y en concreto en el metadato ‘declaration’.

app.module.ts, después de añadir el componente por ejemplo copyright quedaría así:

ng generate component copyright

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

import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

import { CopyrightComponent } from './copyright/copyright.component';

@NgModule({

declarations: [

AppComponent,

CopyrightComponent

],

imports: [

BrowserModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

Data Binding

Es la comunicación o enlace de datos entre el componente y la vista html (su template).

Este modo de comunicación también tiene el nombre de “String Interpolation”.

Para nuestros ejemplos, vamos a declarar un archivo de clase, y los ts crearan los objetos para el paso a la vista y viceversa

Creamos el directorio “modelo” y la clase la definimos en /modelo/desarrolador.modelo.ts:

export class Desarrollador { public id: number; public nombre: string;

public apellidos: string;

public ciudad: string;

constructor (id: number, nombre: string, apellidos: string, ciudad: string){

this.id = id;

this.nombre = nombre;

this.apellidos = apellidos;

this.ciudad = ciudad;

}

}

Interpolación

Comunicación de la fuente (ts) hacia la vista (html)

Su sintaxis, en la que se puede incluir una propiedad, un objeto o una expresión JavaScritp entre dobles llaves, conocidad como sintaxis moustache:

{{propiedad/objeto/expresión}}

Ejemplo 1:

creamos el componente viewmodelo

ng g c viewmodelo -spec false

En el ficheor viewmodelo.component.ts, importamos la clase desarrollador, y se crea el objeto desarrollador. Se emplea el constructor para asignar los atributos:

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

import { Desarrollador } from '../modelo/desarrollador.modelo';

@Component({

selector: 'app-viewmodelo',

templateUrl: './viewmodelo.component.html',

styleUrls: ['./viewmodelo.component.css']

})

export class ViewmodeloComponent implements OnInit{

desarrollador = new Desarrollador(1, 'Juan', 'Gutiérrez', 'Madrid');

constructor() { }

ngOnInit() {

}

}

En el fichero viewmodelo.component.html, es donde utilizamos la expresión mustage para mostrar el contenido del objeto desarrollador:

<div class="container"> <h4>Información del Desarrollador</h4>

<hr>

<h5>id: {{desarrollador.id }}</h5>

<h5>Nombre: {{desarrollador.nombre }}</h5>

<h5>Apellidos: {{desarrollador.apellidos }}</h5>

<h5>Ciudad: {{desarrollador.ciudad }}</h5>

</div>

Property Binding

Comunicación de la fuente (ts) hacia la vista (html)

En este caso, se trata de un enlace que relaciona un atributo con una expresión, con la siguiente sintaxis: [atributodelelementoHTML] = “ expresión “

Creamos el componente ejpropertybinding

ng g c ejpropertybinding --spec false

y en ejpropertybindind.component.ts:

texto = 'Escribe algo';

constructor() { setTimeout(() => { this.texto = 'por favor'; }, 3000);

}

y en fichero template:

<input type="text" [placeholder]="texto">

y en fichero app.component.html renderizamos con:

<div class="container" >

<h3>Ejemplo de Property Binding</h3>

<hr>

<app-ejpropertybinding></app-ejpropertybinding>

</div>

Event Binding

Comunicacion de la vista (html) hacia la fuente (ts)

Se trata de un movimiento de datos dela vista hacia el modelo y se desencadena por un evento en el cliente web.

La sintaxis tiene la siguiente forma:

evento=”nombreMetodo()”;

Creamos el component ejeventbuilding

Y en el fichero ts:

texto = 'Originalmente el texto se carga así';

modTexto() {

if (this.texto === 'arriba')

this.texto = 'abajo';

else

this.texto = 'arriba';

}

Y en la vista:

<button class="btn btn-success"

(click)="modTexto()">Modificar Texto</button>

<h3> {{ texto }} </h3>

Two-way Binding

Es posible la comunicación de la vista a la fuente y viceversa.

Se enlaza el valor de un elemento HTML de la vista con la propiedad de un componente:

[(directiva)] = “nombredelapropiedad”

Creamos el componente ej2waybinding

ng g c ej2waybinding - - spec false

En ts:

export class Ej2waybindingComponent implements OnInit {

texto = 'Texto original a cargar';

constructor() { }

ngOnInit() {

}

En la vista:

<label>Introduce un valor</label>

<input type="text" class="form-control" [(ngModel)]="texto">

<h3>{{texto}}</h3>

Directivas

Las directivas son clases angular para crear, configurar e interactuar con elementos html del DOM.

Las directivas llevan el decorador @Directive en las clases angular.

Hay tres tipos de directivas:

Componentes

Se puede decir que los componentes con su directiva @Component, es una aplicación de las directivas.

De atributo

Modifican el aspecto o comportamiento de los elemento del DOM. Tienen el prefijo ng.

Directiva ngStyle

Sirve para establecer de manera dinámicas los atributos de un elemento del DOM.

[ngStyle]=”{ expresión/propiedad/método }”

Creamos componente ejdirectivangstyle

ng g c ejdirectivangstyle --spec false

En .html:

<h4>Introduzca la puntuación del Alumno</h4>

<input type="number" class="form-control" [(ngModel)]="puntuacion">

<hr>

<h4 style="display: inline-block;">Puntuación obtenida:&nbsp; </h4>

<h4 style="display: inline-block;" [ngStyle]="{color:setColor()}">{{puntuacion}}</h4>

En .ts:

puntuacion:number;

setColor() {

return this.puntuacion >= 5? 'green': 'red';

}

Directiva ngClass

Esta directiva es similar a ngStyle y sirve para establecer la clase (css) de un elemento del HTML.

Creamos el componente para demo:

ng g c ejdirectivangclass --spec false

en .html:

<h4>Introduzca la puntuación del Alumno</h4>

<input type="text" class="form-control" [(ngModel)]="puntuacion">

<div \*ngIf="puntuacion">

<h4 \*ngIf="puntuacion >= 0 && puntuacion <= 10; else aviso"

[ngClass]="{ aprobado: puntuacion >= 5, suspenso: puntuacion <5 }"> Puntuación obtenida: {{puntuacion }}

</h4>

<ng-template #aviso>

<h4 \*ngIf="puntuacion > 10" class="advertencia"> Introduzca una puntuación menor a 10</h4>

<h4 \*ngIf="puntuacion < 0" class="advertencia"> Introduzca una puntuación mayor o igual a 0</h4>

</ng-template>

En .ts:

puntuación:number;

En .css:

.advertencia { color: white; background-color: orange; padding: 10px; }

.aprobado { color: white; background-color: green; padding: 10px; }

.suspenso { color: white; background-color: red; padding: 10px; }

Estructurales

Añaden, reemplazan o eliminan elementos del DOM.

Estas directivas van precedidas por el carácter “\*”.

Directiva \*ngif

Con esta directiva podemos controlar que un elemento HTML se muestre o no dependiendo de una condición definida mediante una propiedad o método.

\*ngIf=”expresión/propiedad/metodo”

lo veremos con un ejemplo, para ello creamos :

ng g c ejdirectivangif --spec false

En ejdirectivangif.component.ts:

definimos dos propiedades tipo string nombre1 y nombre2

nombre1:string;

nombre2:string;

setResultado(){

return this.capital === "Madrid"? true: false;

}

y en ejdirectivangif.component.html:

<label>Nombre y Apellidos</label>

<input type="text" class="form-control" [(ngModel)]="nombre1" placeholder="Complete su nombre y apellidos">

<button type="submit" class="btn btn-primary" \*ngIf="nombre1">Enviar</button

Con las línea de arriba, si el atributo nombre1 tiene valor, se muestra el boton “Enviar”.

<label>Nombre y Apellidos</label>

<input type="text" class="form-control" [(ngModel)]="nombre2" placeholder="Complete su nombre y apellidos">

<button type="submit" class="btn btn-primary" \*ngIf="nombre2; else desactivado">Enviar</button>

<ng-template #desactivado>

<button type="submit" class="btn btn-primary" **disabled**>Enviar</button>

</ng-template>

Con las líneas de arriba, si el atributo nombre2 tiene valor se muestra el botón “Enviar” activado en caso contrario se muestra desactivado

<h3>¿Cuál es la capital de España?</h3>

<input type="text" class="form-control" [(ngModel)]="capital">

<p \*ngIf="!capital; else tienecontenido">Complete la pregunta</p>

<ng-template #tienecontenido>

<h4 \*ngIf="setResultado(); else incorrecto">¡Correcto!</h4>

</ng-template>

<ng-template #incorrecto>

<h4>Lo siento, inténtelo de nuevo</h4>

</ng-template>

En las líneas de arriba, si la propiedad capital no tiene valor muestra “Complete la pregunta”, en caso contrario comprueba si el contenido es Madrid mediante el método setResultado() y en ese caso muestra “¡Correcto!”, en caso contrario muestra “Lo siento, inténtelo de nuevo”.

Directiva \*ngFor

Para realizar iteraciones y presentar listados.

\*ngFor=”let objeto/propiedad of objetos/propiedades”

Crea dentro del elemento html una variable local con let que recorrerá el array definido por of y proveniente del componente.

ng g c ejdirectivangfor -spec

En ts:

cursos:string[];

constructor() {

this.cursos = ['java', 'cobol', 'angular'];

}

En .html:

<h3>Cursos Disponibles</h3>

<ul>

<li \*ngFor="let curso of cursos">

<h4>{{curso}}</h4>

</li>

</ul>