DWEC - ANGULAR

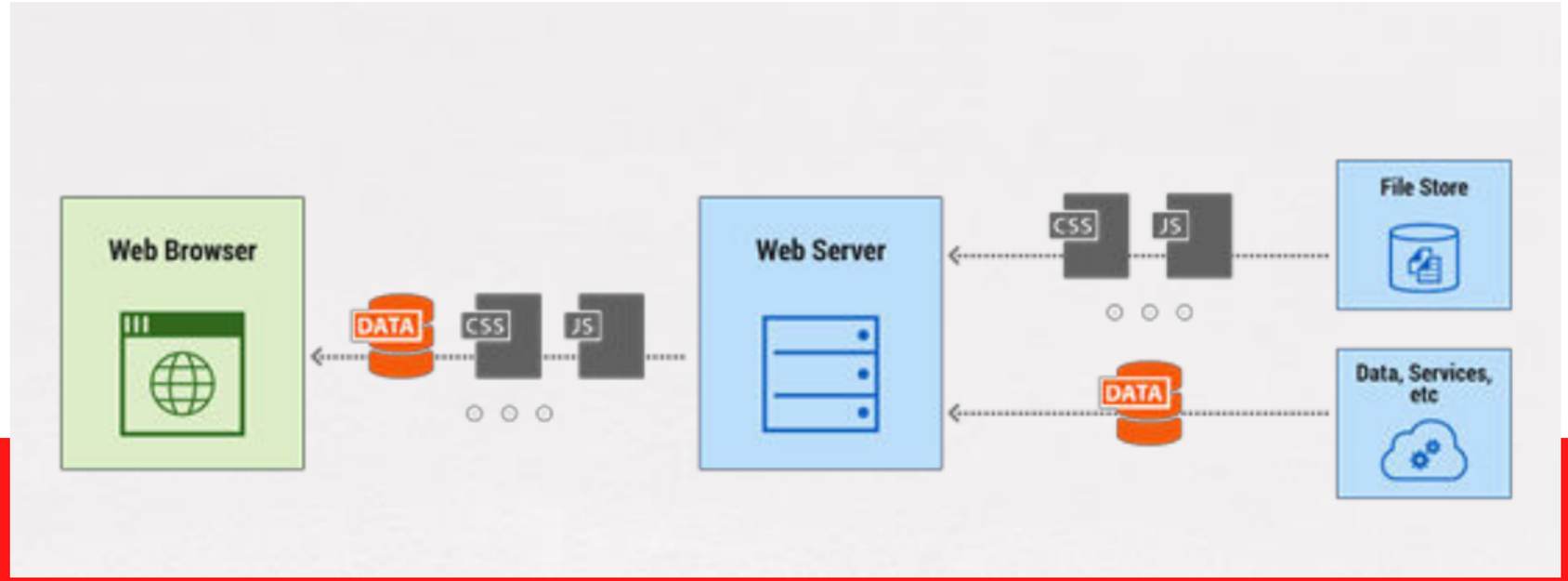
ANGULAR

Angular es un marco de trabajo (Framework) que facilita el desarrollo de aplicaciones web, esta basado en componentes.

- Desarrollado por Google, recibe actualizaciones cada 6 meses
- Extensa comunidad
- Gran demanda en el mercado laboral
- Multiplataforma
- Velocidad y rendimiento
- Framework modular

ANGULAR - SPA (SINGLE PAGE APPLICATION)

Aplicación web donde todo el contenido se carga en la primera petición.
La primera carga puede ser más lenta pero luego la navegación del usuario es mucho más fluida.
El propósito principal de Angular es separar el back-end completamente del front-end evitando que escribas código repetitivo y todo se mantendrá mas ordenado por el patrón de diseño MVC (Modelo, Vista, Controlador).



Ejemplos:
- Gmail
- Twitter
- Facebook (95%)

ANGULAR VS ANGULAR JS

Antes de nada hay tener claro que, a pesar de compartir el nombre, AngularJS y Angular no tienen nada que ver. El mal llamado "Angular 2", que se llama solamente Angular, no es una nueva versión de AngularJS (también denominado Angular 1.x). Es un nuevo framework, escrito desde cero y con conceptos y formas de trabajar completamente distintos.

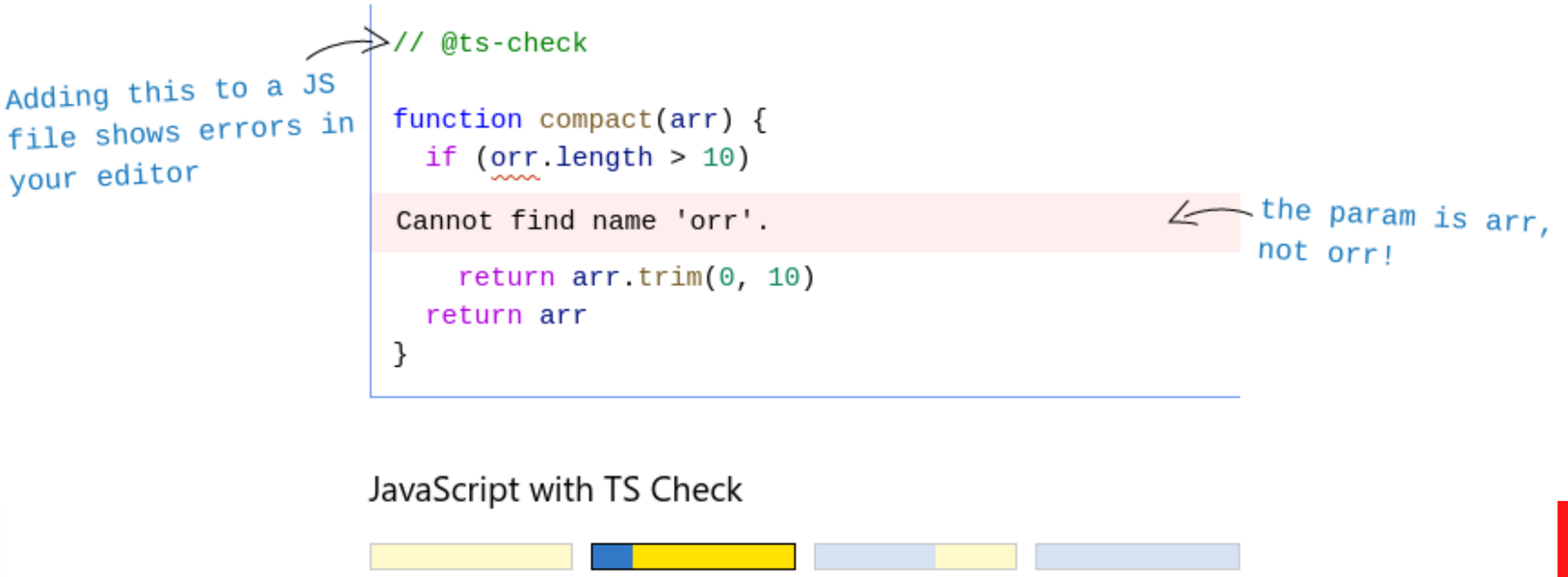
Hay muchos cambios a nivel interno y de funcionamiento que no explicaremos ya que partimos de que tampoco conocemos AngularJS.

El equipo de Angular lanza una nueva versión de este cada 6 meses, actualmente va por la versión 13, pero los cambios entre estas no suelen afectar a nivel de código o de como lo utilizamos los programadores. Afectan más a nivel interno, rendimiento etc.
Por lo que aprender Angular 8, 10 o 12 va a ser prácticamente igual que aprender Angular 13 o 14.

ANGULAR - TYPESCRIPT

TypeScript is JavaScript with syntax for types.

TypeScript añade una sintaxis adicional a JavaScript para permitir una mayor integración con el editor. **Detecta los errores antes de tiempo en el editor.**



ANGULAR - TYPESCRIPT - TIPOS

```typescript
let edad: number = 32;

let edad2 = 32;


const nombre: string = "Antonio"

const nombre2 = "Antonio"


let activo: boolean;
let activo2;

let edades: number[];
let edades2;

let numerosAleatorios: number[] = [1,2,3,4,5];
let numerosAleatorios2;
```

ANGULAR - TYPESCRIPT - TIPOS

String	Number	Array
Tuple	Enum (como en C#)	Any
Void	Boolean	Null / Undefined

https://www.typescriptlang.org/play

Que tipo de dato usarias para:

- Una serie de números la cual no sabes la longitud que puede tener.
- 3 valores, nombre, email, edad. Para 5 personas que estarán en el equipo organizativo.

ANGULAR - TYPESCRIPT - TIPOS - ANY

```
let  notSure: any = 4;
notSure = "Nueva cadena de texto"; // Ahora pasará a ser un string
notSure = false; // Por último es de un tipo boolean
let  lista: any[] = [1, true, "Cadena"];
lista[1] = 100;
```

ANGULAR - TYPESCRIPT - TIPOS - ARRAY

Method	Description
pop()	Removes the last element of the array and return that element
push()	Adds new elements to the array and returns the new array length
sort()	Sorts all the elements of the array
concat()	Joins two arrays and returns the combined result
indexOf()	Returns the index of the first match of a value in the array (-1 if not found)
copyWithin()	Copies a sequence of elements within the array
fill()	Fills the array with a static value from the provided start index to the end index
shift()	Removes and returns the first element of the array
splice()	Adds or removes elements from the array
unshift()	Adds one or more elements to the beginning of the array
includes()	Checks whether the array contains a certain element
join()	Joins all elements of the array into a string
lastIndexOf()	Returns the last index of an element in the array
slice()	Extracts a section of the array and returns the new array
toString()	Returns a string representation of the array
toLocaleString()	Returns a localized string representing the array

```typescript
const listaNumeros: Array<Number> = [1];

const listaFrutas: string[] = ['Apple', 'Orange', 'Banana'];

const lista: Array<string | number> = ['Apple', 2, 3, 4, 'Banana'];
```

https://jsfiddle.net/n7efhqzk/

ANGULAR - TYPESCRIPT - EJERCICIO ARRAYS

- Crea un array con 20 números aleatorios entre el 1 y el 100.
- Muestra el número con máximo valor, número con el mínimo y la posición que estos ocupan en el array.
- Muestra la media de los números de todo el array.
- Muestra el array ordenado.
- Genera un nuevo array con los numeros primos que haya en el array anterior y muestralo.

- Abrimos terminal en vscode:
- npm install –g typescript
- tsc archivo.ts
- node archivo.js

https://jsfiddle.net/n7efhqzk/

ANGULAR - TYPESCRIPT - CLASES - FUNCIONES - PARAMETROS OP

```typescript
function construirNombre(nombre: string, apellido?: string): string{
    if (apellido) return nombre + apellido
    else return nombre
}

class  Persona {
    private  nombre: string;
    private  edad: number;
    constructor(nombre: string, edad: number) {
        this.nombre = nombre;
        this.edad = edad;
    }
    public  saludar(): void {
        console.log("Hola, mi nombre es "+this.nombre+"y tengo "+this.edad+" años.");
    }
}


let  persona = new  Persona("Jonatan", 32);
persona.saludar();
// Hola, mi nombre es Jonatan y tengo 32 años.
```

```typescript
enum Medal {
  Gold = 1,
  Silver,
  Bronze
}
```

```typescript
enum Position {
  First = 'First',
  Second = 'Second',
  Other = 'Other'
}
```

https://jsfiddle.net/nxduj21t/1/

ANGULAR - TYPESCRIPT - EJERCICIO CLASES

- Crea una nueva persona con tu nombre y edad
- Crea una función que reciba como parámetro 2 personas y te responda cual es mayor de las dos imprimiendolo en la consola
- Crea una lista de profesiones haciendo uso de enum ("Pintor", "Programador", "Panadero")
- Añade un nuevo atributo profesion a la clase persona
- Añade como parámetro opcional la profesión en el constructor de persona
- Crea una nueva persona con una profesion
- Modifica el método saludar para que muestre también la profesion de la persona si la tiene

https://jsfiddle.net/nxduj21t/1/

ANGULAR - TYPESCRIPT - EJERCICIO CLASES

- Crea una nueva clase Perro
- Esta clase tendrá un atributo raza y otro color y un método ladrar que imprimirá "guau!"
- Crea 2 perros diferentes
- Crea una interfaz Mamifero con un método caminar y un atributo velocidad.
- Tanto la clase Perro como la clase Persona deben implementar la interfaz Mamimero.

- Dada una distancia de 100 metros (paŕametro que le enviaremos a la función caminar), calcula quien la recorre más rápido si un perro o una persona.
- Para esto la función caminar nos debe devolver el tiempo que tarda en recorrer una distancia en función a la velocidad que se tenga.

https://jsfiddle.net/nxduj21t/1/

ANGULAR - ENTORNO DE TRABAJO

- Visual Studio Code
- Git
- NodeJS
- NVM
- Angular Essentials (Version 12)
- Debugger for firefox

ANGULAR - INSTALACIÓN - ANGULAR CLI

- npm install –g @angular/cli
- ng ––version

ng new	ng build	ng config
ng generate	ng help	ng add
ng serve	ng test	ng update

https://angular.io/cli

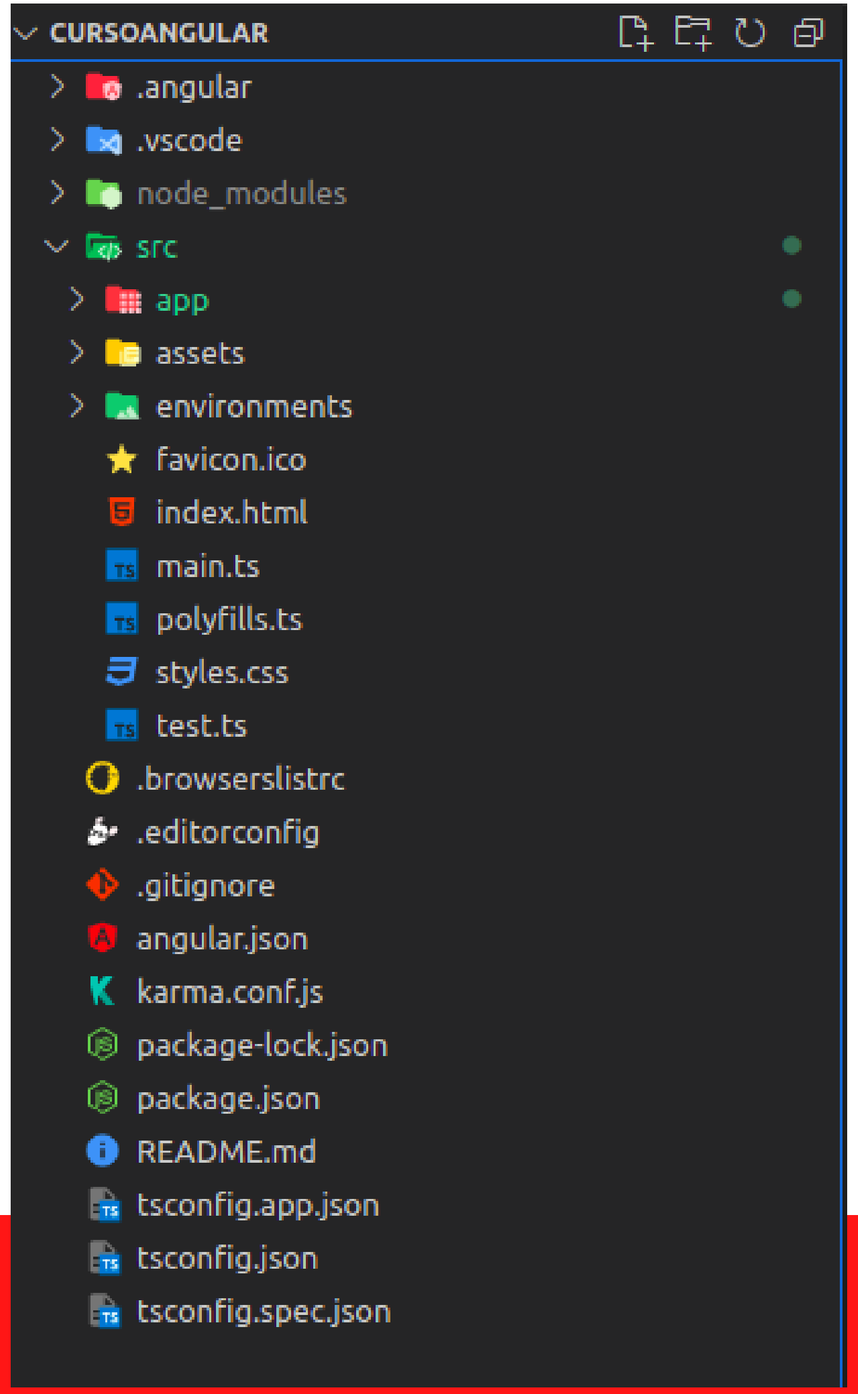
ANGULAR - CREACIÓN PRIMER PROYECTO

- ng new nombreProyecto
- ng serve
- ng serve –o
- ng serve –o ––port 4100

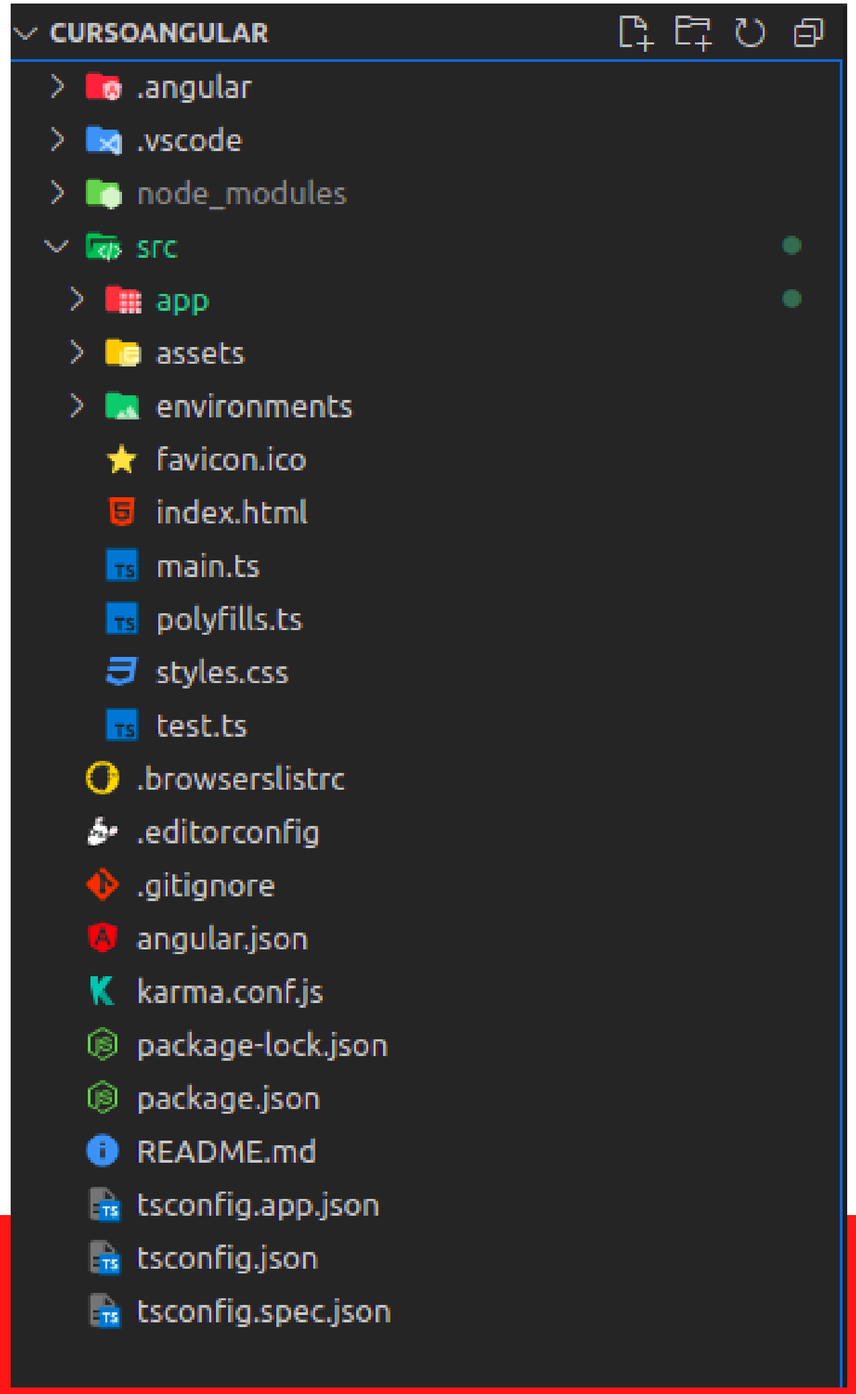
Windows Powersell:

Set-ExecutionPolicy RemoteSigned –Force

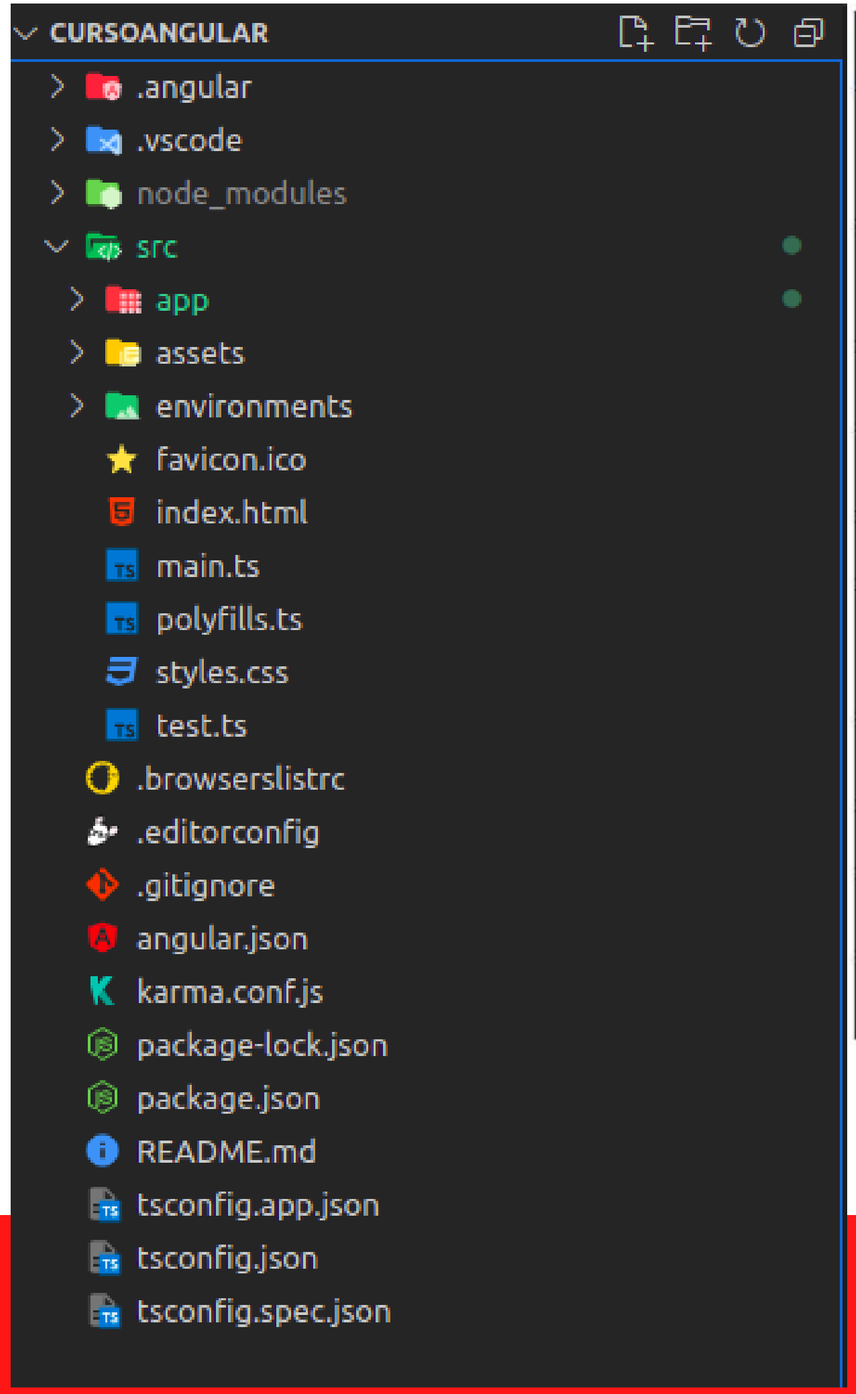
https://angular.io/cli



ANGULAR - ESTRUCTURA

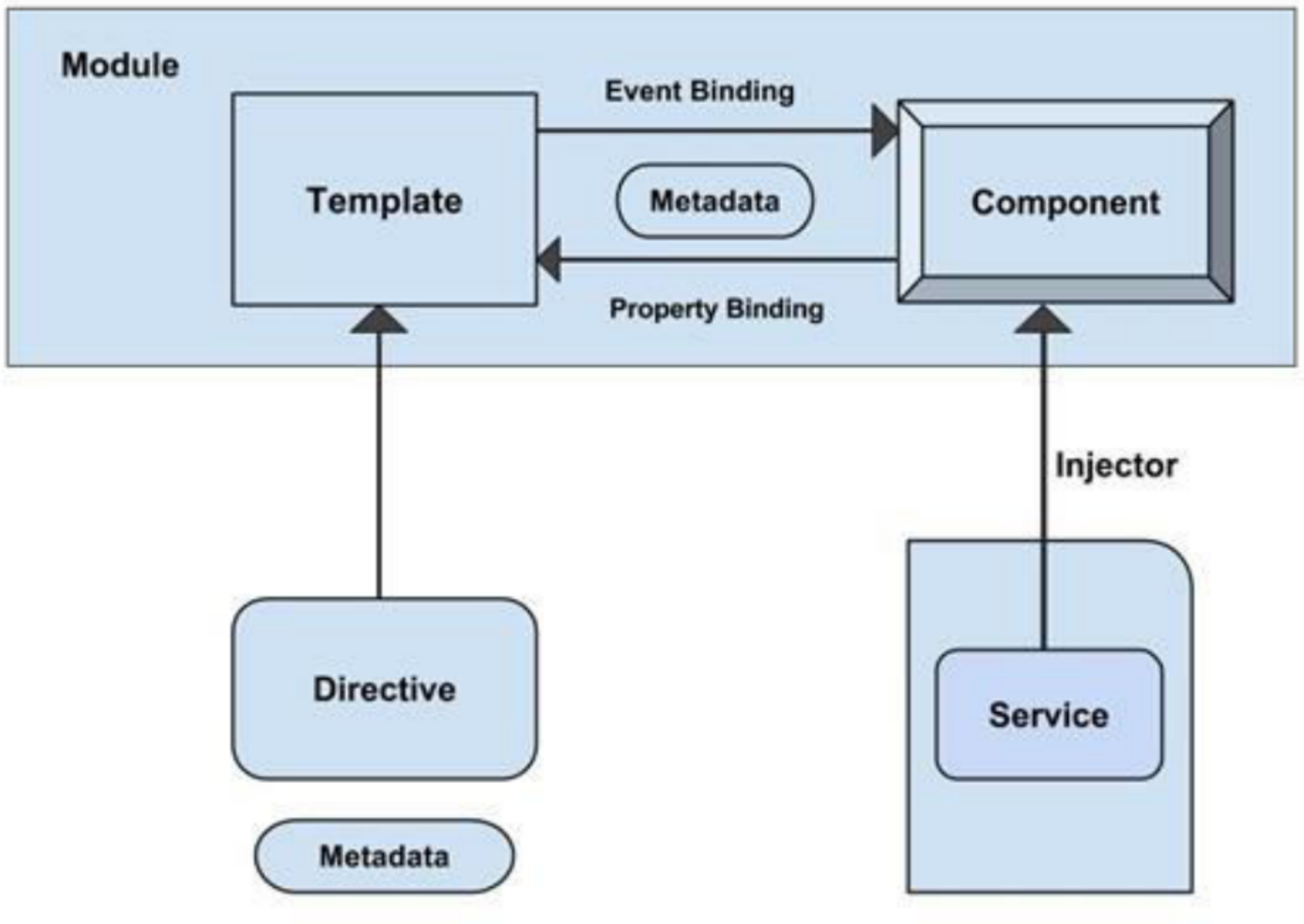


Elemento	Comentario
.git	Contiene los archivos necesarios para la gestión con GIT.
e2e	Carpeta que contiene los archivos de prueba, denominados end to end (de extremo a extremo) que se ejecutan con Jasmine.
node_modules	Contiene todos los paquetes y dependencias de Node.js de nuestro proyecto.
src	Carpeta que contiene los archivos fuentes raíz de la aplicación.
.editorconfig	Contiene la configuración de nuestro editor.
.gitignore	Permite indicar archivos que queremos ignorar de cara al control de versiones git.
angular-cli.json	Contiene la configuración de CLI.
karma.conf.js	Contiene la configuración para los test.
package.json	Describe las dependencias necesarias para ejecutar la aplicación.
protractor.conf.js	Configuración de test con Jasmine.
README.md	'Readme' clásico de cualquier proyecto compartido en GitHub.
tslint.json	Contiene la configuración del Linter para Typescript.



Elemento	Comentario
app	Carpeta raíz que contendrá los componentes, servicios y resto de elementos que constituyen nuestra aplicación.
assets	Imágenes, vídeos y archivos en general que no son propiedad de ningún componente.
environments	Contiene características relativas al entorno de trabajo.
favicon.icon	Imagen que permite identificar nuestra aplicación.
index.html	Archivo raíz donde se inicia la aplicación
main.ts	Es el primer archivo que se ejecuta y contiene todos los parámetros de la aplicación.
polyfills.ts	Asegura la compatibilidad con los navegadores.
styles.css	Estilos del proyecto.
test.ts	Puede contener test unitarios.
tsconfig.json	Configuración de TypeScript.

ANGULAR - ESTRUCTURA



ANGULAR - ESTRUCTURA - MODULOS

- Modular, cada módulo tiene una función concreta
- Cada módulo puede contener componentes, servicios, directivas...
- Todos los proyectos angular tienen un módulo raíz
- Sistema de almacenamiento principal, todo se engloba dentro de módulos

ANGULAR - COMPONENTES

Puede mostrar vistas completas o partes de una vista.

Definen la lógica de una plantilla dentro de una clase a través de una API de atributos/propiedades y métodos

ng generate component nombre
ng g c nombre

ANGULAR - ESTRUCTURA - COMPONENTES - VISTAS

- Son partes que definen vistas o pequeños trozos de nuestro HMTL
- Parte lógica de la vista
- Se comunica con la template/vista/plantilla a la que le envía la información

```
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html',
    styleUrls: ['./app.component.css']
})
```

ANGULAR - TEMPLATING -INTERPOLATION

```
Estas en la página de
{{ title }}
<br>
Este es el número aleatorio {{numeroAleatorio}}
```

```
export class AppComponent {
  title = 'ejemplos';
  numeroAleatorio:Number = 10;
}
```

Estas en la página de ejemplos
Este es el número aleatorio 10

ANGULAR - TEMPLATING - PROPERTY BINDING

```
//Property binding
isExampleSelected:Boolean = false;
isExample:Boolean = true;
```

```
<option [selected]="isExampleSelected" value="Rainbow">Rainbow</option>
<option [selected]="isExample" value="Rainbow">Selected Rainbow</option>
<input type="radio" [checked]="isExampleSelected" />Option 1
<input type="radio" [checked]="isExample" />Option 2
```

```
<div [hidden]="isHidden">Hidden or not</div>
```

```
<p [style.color]="foreground">Friendship is Magic</p>
```

ANGULAR - TEMPLATING - EVENT BINDING

- Otro de los conceptos básicos en angular es la captura de eventos, al hacer clic sobre un botón, presionar una tecla o movimientos con el mouse se pueden capturar mediante los eventos de angular.

- Uno de los eventos mas comunes que ejecutaras al desarrollar un aplicación en angular es el evento CLICK (click) lo puedes poner en un botón, input, etc.

ANGULAR - TEMPLATING - EVENT BINDING

Otro de los conceptos básicos en angular es la captura de eventos, al hacer clic sobre un botón, presionar una tecla o movimientos con el mouse se pueden capturar mediante los eventos de angular.

- -blur
- -change
- -click
- -copy
- -cut
- -dblclick
- -focus
- -keydown
- -keypress
- -keyup
- -mousedown
- -mouseenter
- -mouseleave
- -mousemove
- -mouseover
- -mouseup
- -paste

```
<button (click)="onSave()">Save</button>
```

target event name

template statement

```
OPCION: {{ opcionMarcada }}<br>
<input type="radio" value="opcion1" name="input1" (click)="setOption($event)"/>Option 1
<br>
<input type="radio" value="opcion2" name="input1" (click)="setOption($event)"/>Option 2
```

```
//Event binding
opcionMarcada:String = "";

setOption(event:Event){
  alert(event.target);
  //Casting del event
  if((<HTMLInputElement>event.target).value=="opcion1") this.opcionMarcada = "Has marcado la opcion 1"
  else this.opcionMarcada = "Has marcado la opcion 2"

}
```

ANGULAR - TEMPLATING - EVENT BINDING - BURBUJA

```
<div (click)="onButtonClick()">
  <button>Click me!</button>
</div>
```

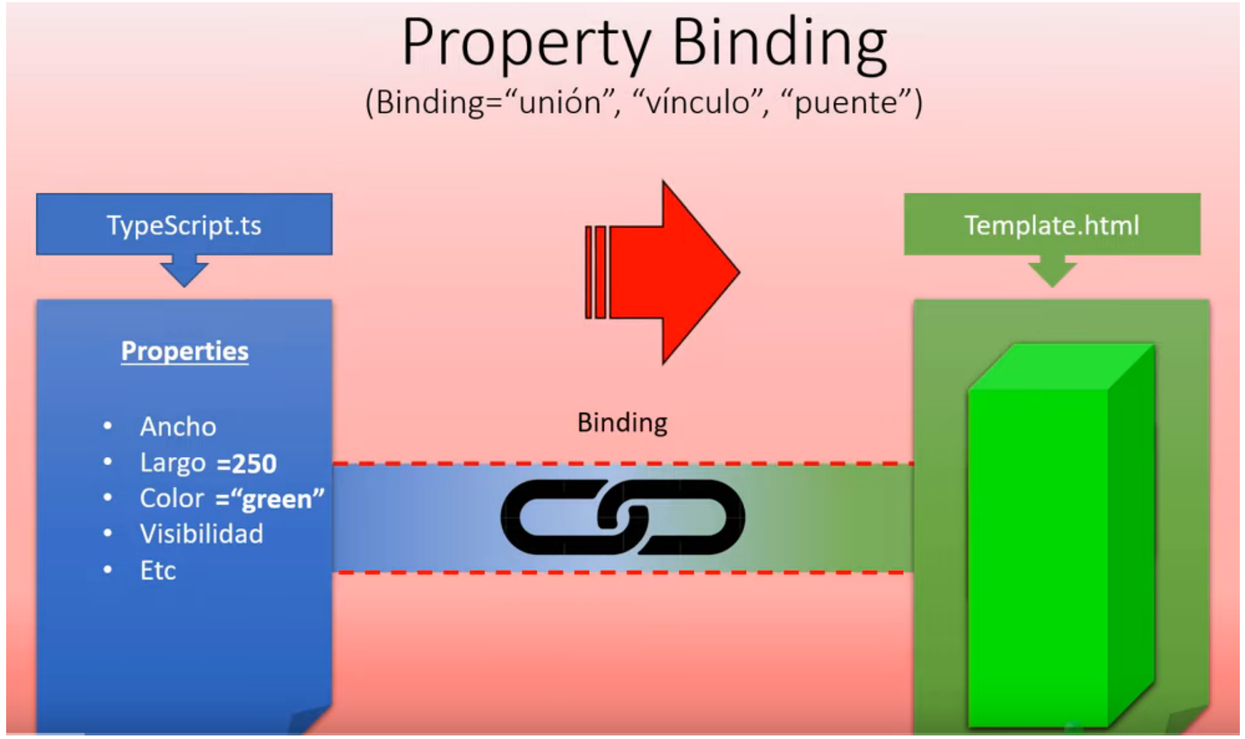
```
<div (click)="onButtonClick($event)">
  <button>Click me!</button>
</div>
```

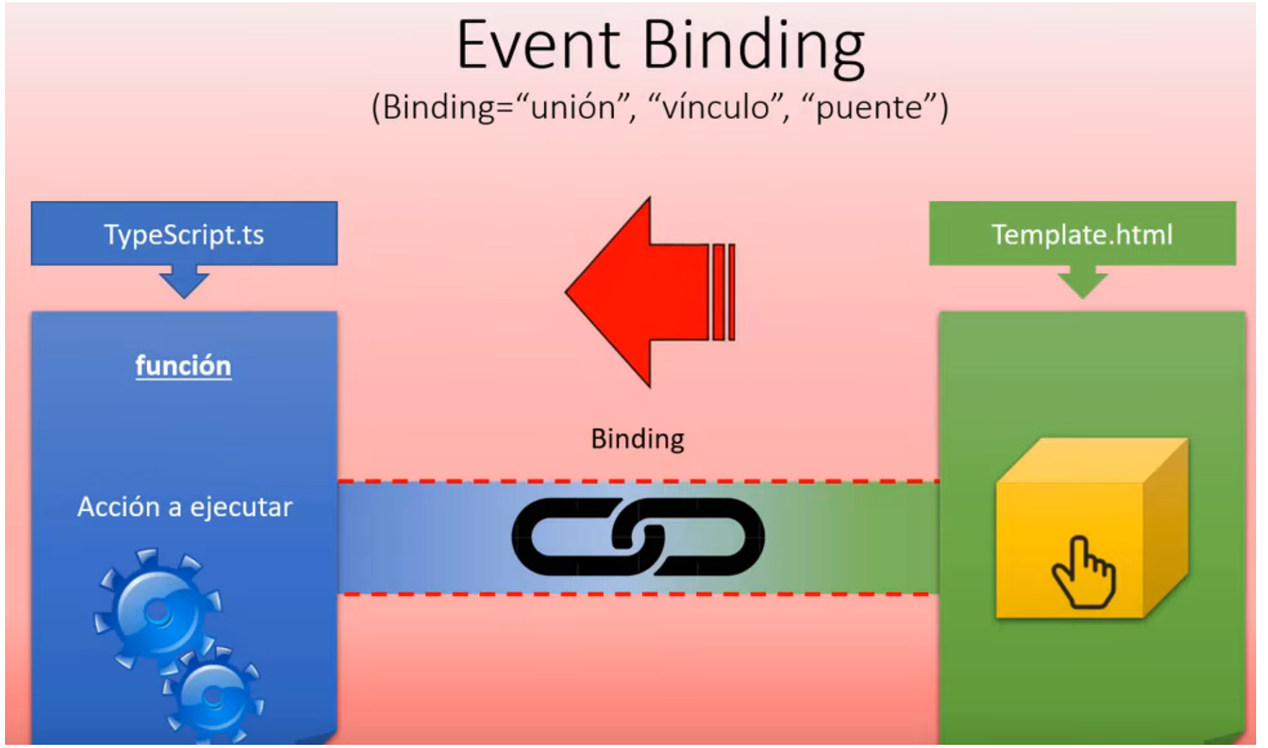
```
onButtonClick(event) {
  event.preventDefault();
  event.stopPropagation();
}
```

ANGULAR - TEMPLATING -VARIABLE DE PLANTILLA

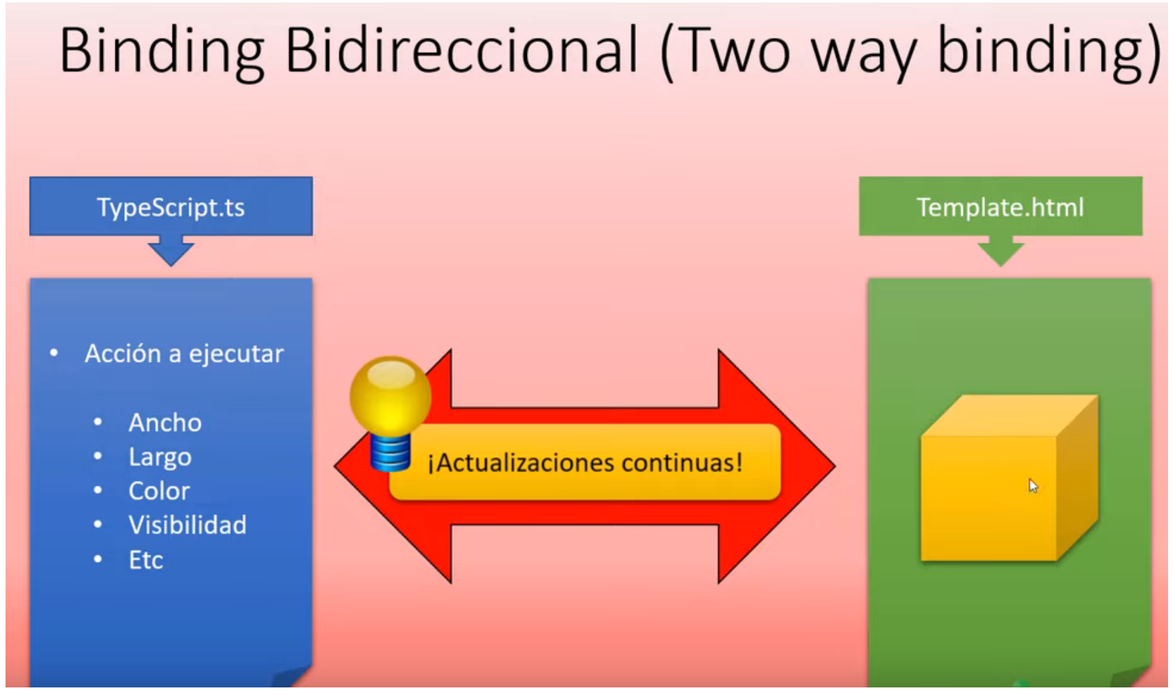
```
<!--VARIABLE DE PLANTILLA-->

<hr><br><br>
<input #inputPrueba />
<br>
Este es el título 2 : {{saludo}}



<br>
<br>
<button (click)="saludar(inputPrueba.value)">Saludar</button>
```

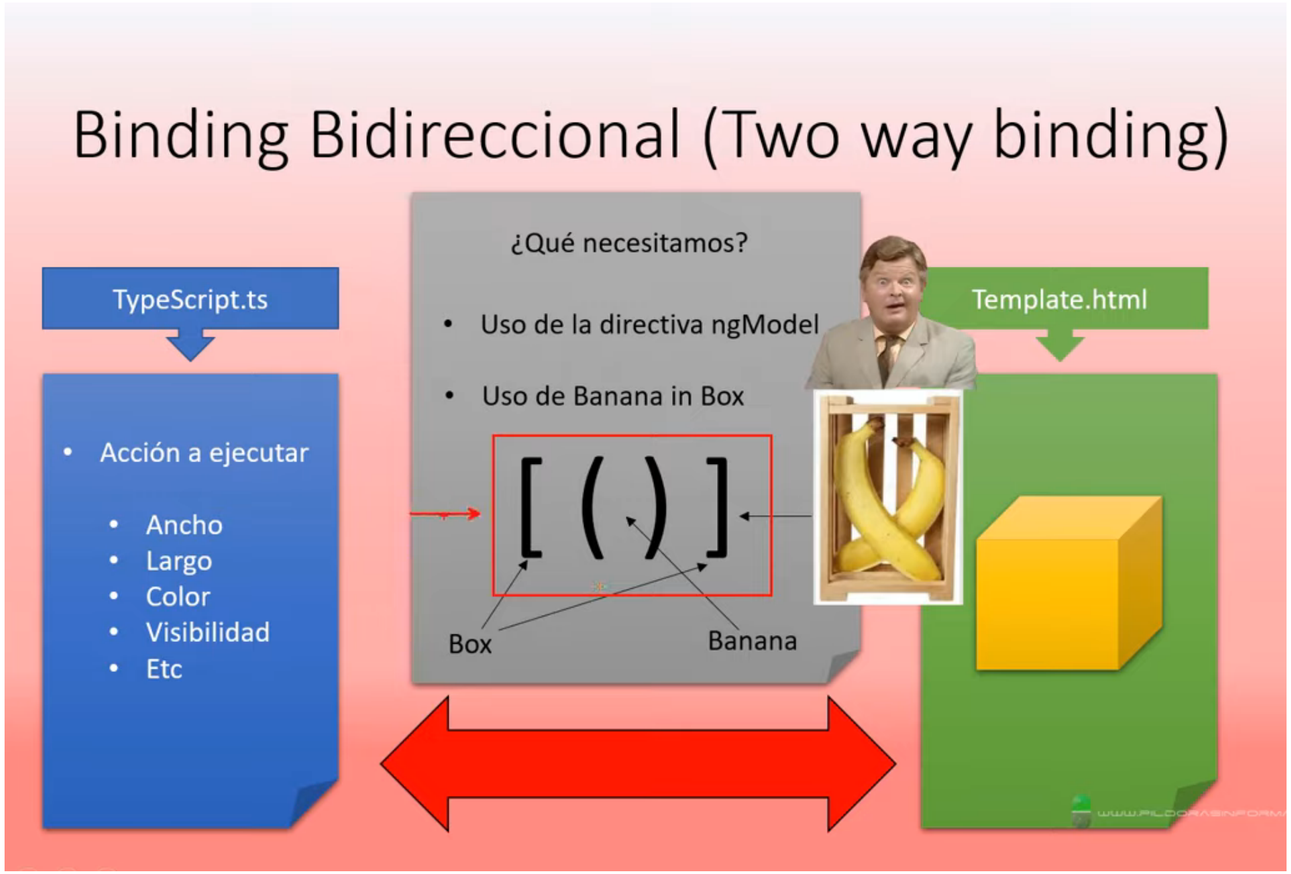
ANGULAR - TEMPLATING - PROPERTY/EVENT BINDING





ANGULAR - TEMPLATING - TWO WAY BINDING - BIDIRECCIONAL






```
imports: [
  BrowserModule,
  FormsModule
],
```

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

ANGULAR - TEMPLATING - EJERCICIO CALCULADORA

- Crea un nueva aplicacion en Angular que se llamara MiniCalculadora.
- La calculadora deberá tener 2 input para introducir números y un tercer bloque donde se mostrará el resultado.
- Deberá tener 3 botones: sumar, restar y multiplicar.

Instala Bootstrap en tu proyecto para darle estilo a los botones.
Para ello deberás ejecutar el siguiente comando en la terminal de tu proyecto:
npm install bootstrap --save
npm install jquery --save

Y modificar el archivo angular.json

- `node_modules/bootstrap/dist/css/bootstrap.css` in the `projects->architect->build->styles` array,
- `node_modules/bootstrap/dist/js/bootstrap.js` in the `projects->architect->build->scripts` array,
- `node_modules/bootstrap/dist/js/bootstrap.js` in the `projects->architect->build->scripts` array,

ANGULAR - DIRECTIVAS

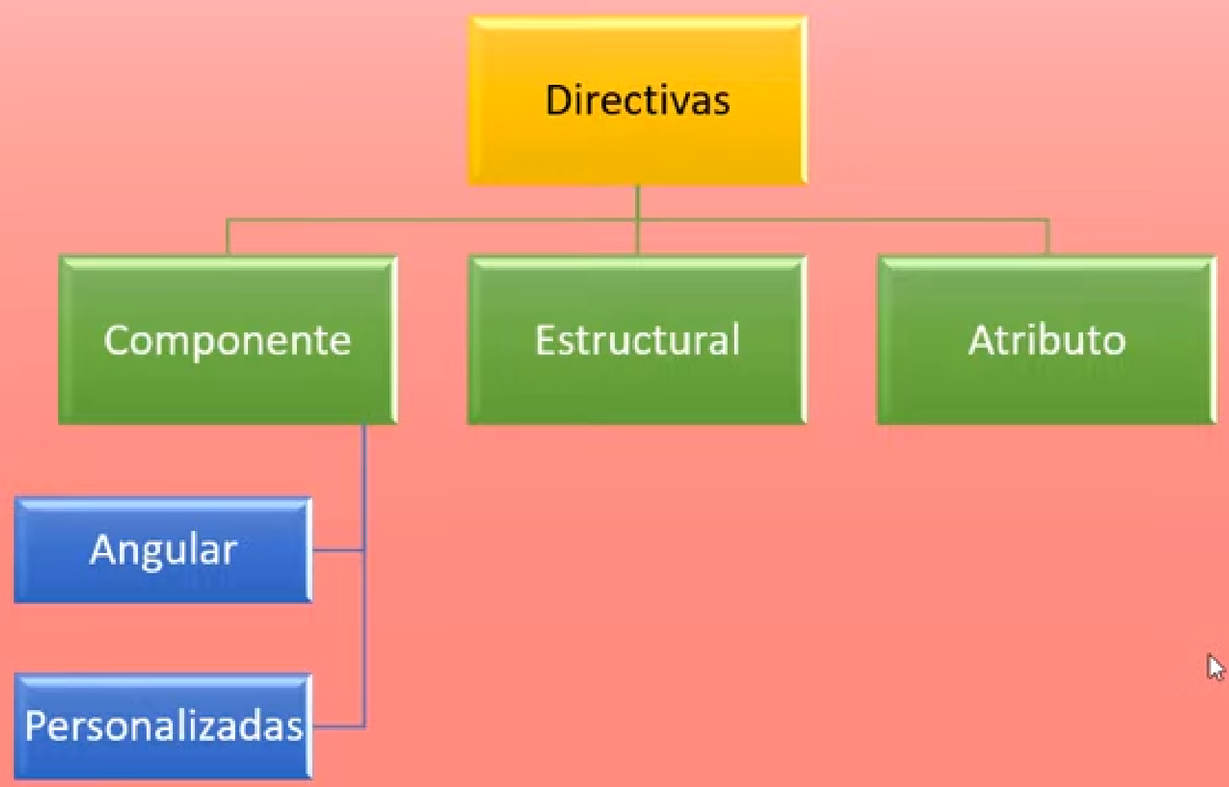
Property y Event Binding son funcionalidades geniales pero no nos permiten cambiar la estructura del DOM, como iterar sobre una colección y añadir un elemento por elemento.

Para hacerlo, necesitamos usar directivas estructurales.

Una directiva en Angular se utiliza para añadir comportamiento a un elemento.

ANGULAR - DIRECTIVAS

Elementos que se aplican a etiquetas HTML y añaden funcionalidad a las etiquetas donde se aplican.



- Atributo:
  - ngClass
  - ngStyle
  - ngModel
- Estructural:
  - ngIf
  - ngFor
  - ngSwitch
  - ngPlural
  - ngTemplate
  - ngComponentOutlet
- Componente
  - Angular
    - @Component
    - @Module
  - Personalizadas (Custom)

https://tinyurl.com/directivasAngular

ANGULAR - DIRECTIVAS - NGIF

***ngIf**

```
<div *ngIf="races.length > 0"><h2>Races</h2></div>
```

Se utiliza * para mostrar que es una directiva estructural. El ngIf mostrará o no el div siempre que el valor de las carreras cambie: si no hay más "races", el div desaparecerá.

```
  <div *ngIf="races.length > 0; else empty"><h2>Races</h2></div>
  <ng-template #empty><h2>No races.</h2></ng-template>
`
```

ANGULAR - DIRECTIVAS - *NGFOR

***ngFor**

Permite instanciar un template por elemento de una colección.

```
export class RacesComponent {
  races: Array<any> = [{ name: 'London' }, { name: 'Lyon' }];
}
```

```
<ul>
  <li *ngFor="let race of races">{{ race.name }}</li>
</ul>
```

```
<ul>
  <li *ngFor="let race of races; index as i">{{ i }} - {{ race.name }}</li>
</ul>
```

```
      <li *ngFor="let pony of ponies; even as isEven" [style.color]="isEven ? 'green' : 'black'">
        {{ pony.name }}
      </li>
```

- `even`,
- `odd`, a
- `first`
- `last`,

ANGULAR - DIRECTIVAS - *NGSWITCH

***ngSwitch**

```
<div [ngSwitch]="messageCount">
  <p *ngSwitchCase="0">You have no message</p>
  <p *ngSwitchCase="1">You have a message</p>
  <p *ngSwitchDefault>You have some messages</p>
</div>
```

ANGULAR - DIRECTIVAS

*ngStyle

```
<p [style.color]="foreground">Friendship is Magic</p>
```

```HTML
<div [ngStyle]="{fontWeight: fontWeight, color: color}">I've got style</div>
```

*ngClass

```
<div [class.awesome-div]="isAnAwesomeDiv()">I've got style</div>
```

```
<div [ngClass]="{'awesome-div': isAnAwesomeDiv(), 'colored-div': isAColoredDiv()}">I've got
style</div>
```

ANGULAR - EJERCICIO MONSTERAPP

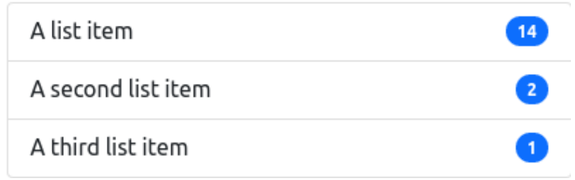
- Crea una nueva app de Angular monsterApp
- Crea una nueva clase dentro de la app monster.model.ts
- Nuestro monster deberá tener nom, raza, fuerza y vida además de dos métodos "atacar" y "ataqueEspecial" (Que por ahora estarán vacíos).
- Crea un enum Raza que contenga 4 razas ("Orco", "Trasgo", "Enano", "Elfo").
- Crea un formulario donde podremos crear monstruos. Desde este formulario escribiremos el nombre, seleccionaremos la raza y le daremos un valor a ataque que sea menor de 100, y otro valor a vida que sea menor de 1000.

- Muestra todos los montruos que se han creado debajo de este formulario.
- Utiliza bootstrap para darle un estilo simple al formulario, botones y la lista.

ANGULAR - EJERCICIO MONSTERAPP

- Si no hay monstruos creados deberemos mostrar un mensaje utilizando un alert de bootstrap diciendo que aun no hay ningun monstruo creado.

- Utiliza listas de bootstrap para mostrar el nombre de monstruo, junto con su raza y su vida, el ataque se mostrará con un badge en dentro de la celda.



- Si la raza es Orco o Trasgo usaremos la clase list-group-item-warning para mostrar la celda en rojo. Si la raza es Enano o Elfo usaremos list-group-item-primary para mostrar la celda en azul.

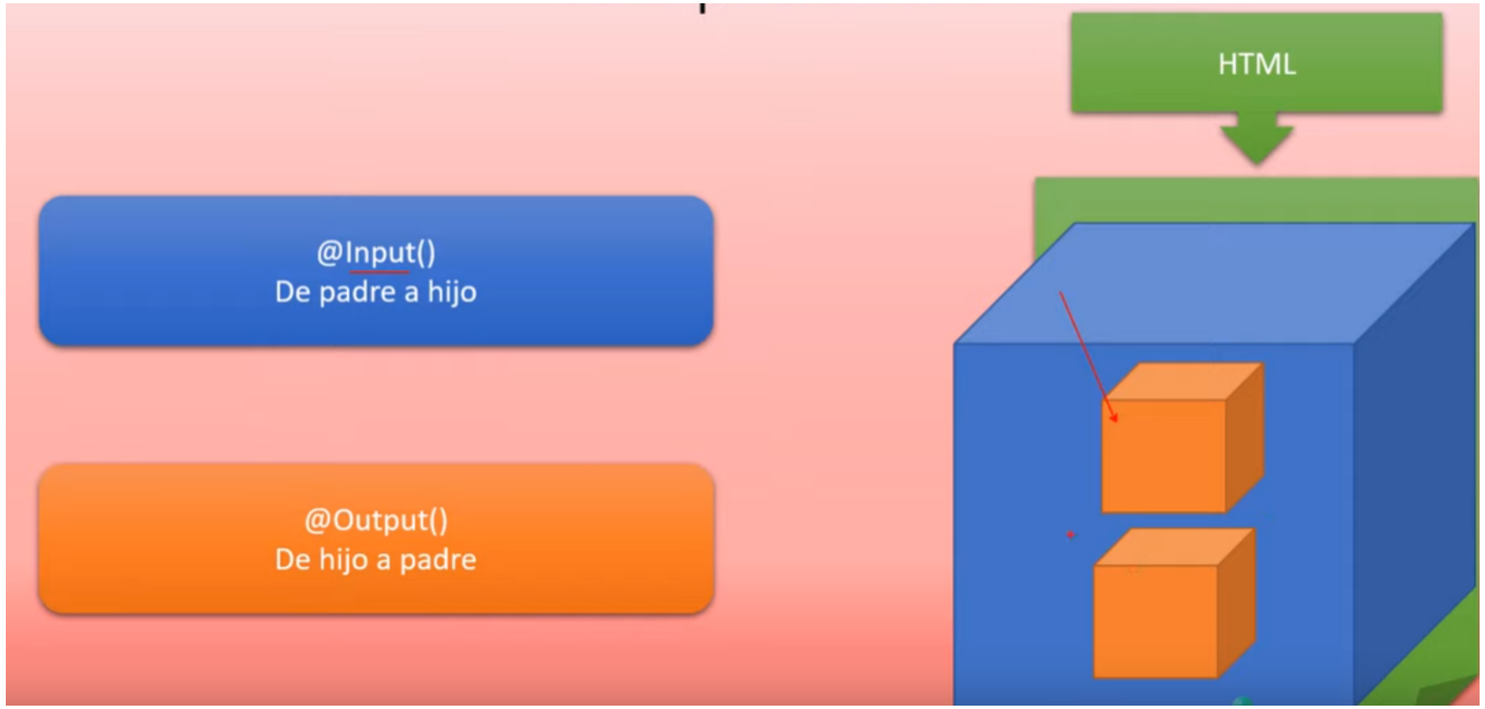
ANGULAR - COMPONENTES

Puede mostrar vistas completas o partes de una vista.

Definen la lógica de una plantilla dentro de una clase a través de una API de atributos/propiedades y métodos

ng generate component nombre
ng g c nombre

ANGULAR - COMUNICACIÓN ENTRE COMPONENTES



Muchas veces será necesario enviar información entre componentes padre-hijo, hijo-padre. Para esto necesitamos de la comunicación entre componentes.

ANGULAR - COMUNICACIÓN ENTRE COMPONENTES

Consiste en usar la etiqueta **@Input** de Angular. Ésta etiqueta se pone en el componente hijo para indicar que esa variable proviene desde fuera del componente, decir desde el componente padre.

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

@Component({
  selector: 'app-child',
  template: `
      Message from parent:
  `,
  styleUrls: ['./child.component.css']
})
export class ChildComponent {

  @Input() childMessage: string;

  constructor() { }
```

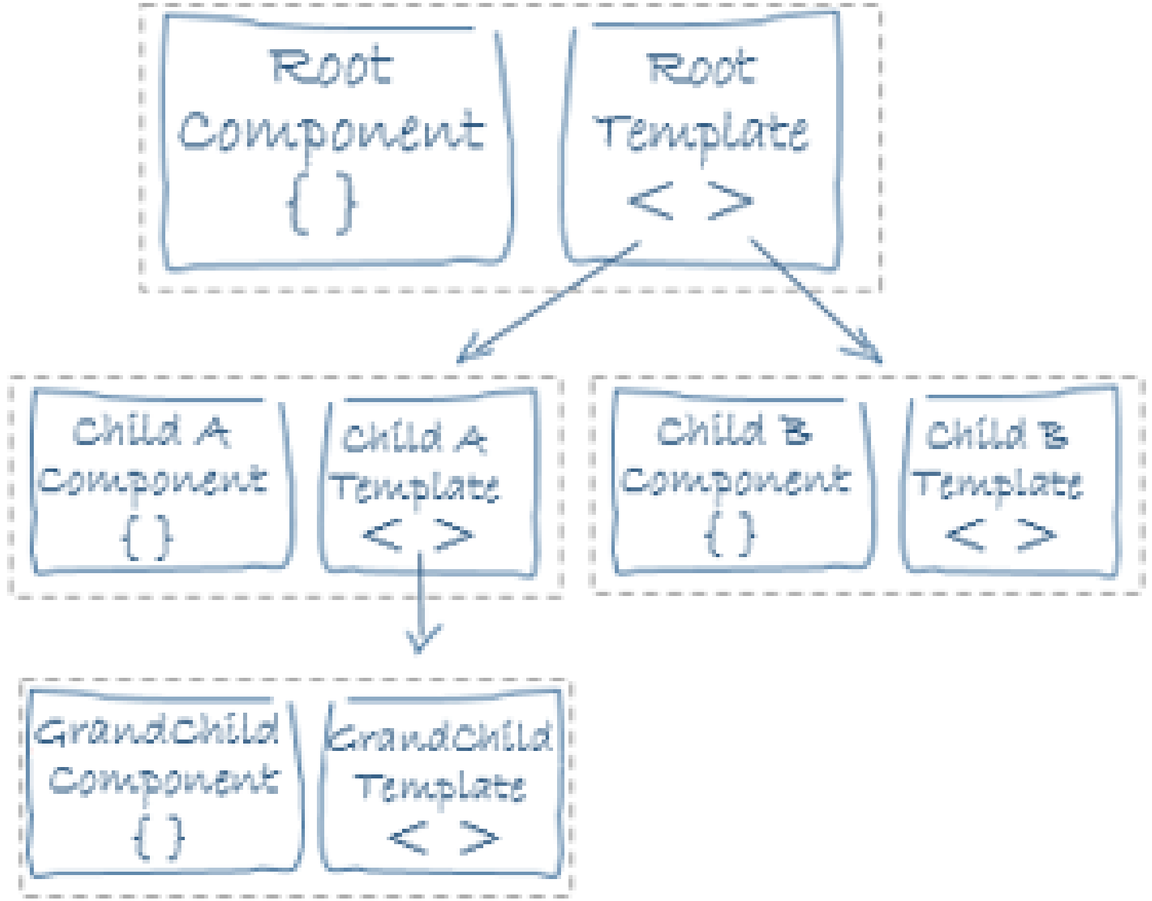
Hijo

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

@Component({
  selector: 'app-parent',
  template: `
    <app-child [childMessage]="parentMessage"></app-child>
  `,
  styleUrls: ['./parent.component.css']
})
export class ParentComponent{
  parentMessage = "message from parent"
  constructor() { }
}
```

Padre

ANGULAR - CREACIÓN DE COMPONENTES

- Crea un nuevo componente que se llamará monsterC
    - ng generate component monsterC
    - ng g c monsterC
- Este componente va a tener toda la informacion referente a cada monster.
- Añade el componente recién creado en tu componente principal.



ANGULAR - NGONINIT

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

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.css']
})

export class HomeComponent implements OnInit {
  constructor() { }
  ngOnInit() {
    console.log("En este instante el componente ha cargado")
  }
}
```

Angular, como otros frameworks Javascript tiene una forma de poder ejecutar código cuando el componente carga por primera vez, en Angular es un método que se llama ngOnInit.

Diferencias entre ngOnInit() y el constructor() en Angular:

Como otros lenguajes, typescript también tiene un constructor de clase, en este caso el constructor se ejecuta antes que el ngOnInit().

Normalmente se usa el constructor para inicializar variables, y el ngOnInit para inicializar o ejecutar tareras que tienen que ver con Angular. Todo esto lo podemos poner directamente en el constructor y funcionaría de la misma manera, pero no está de más tener más separado el código para que sea más mantenible.

Es decir, mi recomendación es que en el constructor solo incialices variables y el ngOnInit el resto de cosas: llamadas a backend, preparación de los datos, filtrado, etc.

ANGULAR - SERVICIOS - INJECCION DE DEPENDENCIAS

Los componenetes cada vez se hacen más y más grandes a medida vamos escribiendo código en nuestra app.
Llegará un punto en el que 2 o más componentes tengan que hacer la misma tarea o una muy similar. En este punto en lugar de escribir 2 veces lo mismo en componentes diferentes es cuando haremos uso de un servicio.

Así evitamos repetir código para una misma tarea. Concepto que se llama modularización en programación orientada a objetos. Esto en Angular se hace creando un **servicio**.

Lo que haremos será programar el servicio y injectarlo a los servicios que lo necesiten.

ng g s nombreServicio

ANGULAR - SERVICIOS - INJECCION DE DEPENDENCIAS

```
ng generate service nombre
```

Agregar en providers en el app.module.ts

Código de un servicio

```
import { Injectable } from "@angular/core";

@Injectable({
  providedIn: "root"
})
export class MessageService {
  messages: string[] = [];

  add(message: string) {
    this.messages.push(message);
  }

  clear() {
    this.messages = [];
  }
}
```

Para usar un servicio dentro de un componente:

```
import { ServiceName } from "../serviceName.service";
```

```
export class MessagesComponent implements OnInit {
  constructor(public messageService: MessageService) {}

  ngOnInit() {
    this.messageService.add("prueba");
  }
}
```

ANGULAR - SERVICIOS

- Crea un nuevo servicio al que llamaremos monsterDataService.
- En este servicio cargaremos todos los datos de nuestos monstruos.
- Inicializa el array de monsters con 4 monstruos en el servicio.
- Carga desde el componente los monsters creados en el servicio.

ng g s monsterDataService

ANGULAR - SERVICIOS - LLAMADAS HTTP

Hasta ahora con lo que sabemos podemos usar los servicios para leer y escribir datos, pero simulados o guardados en la memoria del navegador, para leer o escribir de una API REST tenemos que hacer llamadas HTTP.

Por cierto, una API es un conjunto de endpoints (o rutas) que provee un servidor (backend) que permiten acceder a datos o realizar operaciones. Normalmente lo que se suele hacer es primero generar una API en un servidor usando un lenguaje como Java o Python (ojo porque no puedes generar una API con Angular, tiene que estar creada en un servidor).

Nosotros usaremos la siguiente API de ejemplo para no tener que crear una:

https://reqres.in

ANGULAR - SERVICIOS - LLAMADAS HTTP

Antes de empezar con las llamadas hay que conocer que Angular tiene un modulo que facilita esta tarea, el modulo es **HttpClient**. Con este modulo no necesitas usar fetch ni ajax ni nada.

Para usar **HttpClient** de Angular en cualquier parte, tenemos que importar el módulo **HttpClientModule**, en la sección imports de el **app.module.ts**:

```
import { HttpClientModule } from "@angular/common/http";
```

HttpClient usa **Observables** de RxJS. Los observables son una colección de futuros eventos que llegan de forma asíncrona. Si quieres aprender más de RxJS puedes visitar su web oficial: **http://reactivex.io/rxjs/**

ANGULAR - SERVICIOS - LLAMADAS HTTP

Aunque lo hayamos importado en la página de forma global, también tenemos que importarlo y inyectarlo en los constructores de los servicios desde los que vayamos a realizar llamadas HTTP. Por ejemplo para este nuevo servicio que he creado para listar los usuarios de que vienen de la API:

```
import { HttpClient } from "@angular/common/http";

@Injectable({
  providedIn: "root"
})
export class UsersService {
  constructor(private http: HttpClient) {}
}
```

ANGULAR - SERVICIOS - HTTP GET

```
getUsers(){
  this.http.get('https://reqres.in/api/users?page=2').subscribe(data => {
    console.log(data);
  });
  console.log("Esto se ejecutará antes que el console log de arriba");
}
```

```
import { HttpClient } from "@angular/common/http";

@Injectable({
  providedIn: "root"
})
export class UsersService {
  constructor(private http: HttpClient) {}

  getUsers() {
    this.http.get("https://reqres.in/api/users?page=2").subscribe(data =>
      console.log(data);
    });
  }
}
```

Fíjate que después de la llamada al método **get()** usamos **subscribe**. Esto funciona como las promesas de Javascript, con ese método te esperas a que la petición termine. Dentro de ese método se ejecuta una función (usando arrow functions) en la que se devuelve el objeto data que contiene la respuesta a la petición de la API.

El console.log de abajo se ejecuta antes incluso de que termine la petición. Esto pasa porque **el código es asíncrono** y por tanto lo que pongas debajo no va a esperar a que la petición termine. Dentro del subscribe si que tienes la certeza de que la petición ha terminado y por tanto tienes la respuesta.

ANGULAR - SERVICIOS - HTTP GET

Al inyectar el servicio en el componente, quedaría asi:

```
// users/users.component.ts

import { Component, OnInit } from "@angular/core";
import { UsersService } from "./users.service";

@Component({
  selector: "app-users",
  templateUrl: "./users.component.html",
  styleUrls: ["./users.component.css"]
})
export class UsersComponent implements OnInit {
  constructor(public usersService: UsersService) {}

  ngOnInit() {
    this.usersService.getUsers();
  }
}
```

Con esto podemos realizar llamadas desde el servicio pero los datos que se devuelven todavía no los tenemos en el componente para poder mostrarlos.

Para hacer poder mostrar los datos en el HTML del componente, en lugar de hacer el subscribe en el servicio, tenemos que devolver un **Observable** de la llamada http, es decir:

```
getUsers(): Observable<any>{
  return this.http.get('https://reqres.in/api/users?page=2');
}
```

```
import { Observable } from "rxjs/Observable";
```

ANGULAR - SERVICIOS - HTTP GET

```
import { Component, OnInit } from "@angular/core";
import { UsersService } from "../users.service";

@Component({
  selector: "app-users",
  templateUrl: "./users.component.html",
  styleUrls: ["./users.component.css"]
})
export class UsersComponent implements OnInit {
  users: any;

  constructor(public usersService: UsersService) {}

  ngOnInit() {
    this.usersService.getUsers().subscribe(data => {
      this.users = data;
    });
  }
}
```

Para este ejemplo he puesto que el Observable sea de tipo Any (cualquier tipo de objeto) pero lo suyo, en un futuro, sería usar una interfaz para poder tener un modelo para los datos.

Ahora, en el componente, cuando queramos llamar al servicio (siempre y cuando lo hayamos inyectado en el controlador), tenemos que subscribirnos para recibir la información.

IMPORTANTE Cuando carga un componente, si mostramos una variable que viene de una petición HTTP, no se cargará y tirará error porque en el instante en el que se abre la página, la petición aún no se ha realizado. Para arreglar esto tenemos que poner un ngIf a la variable que viene desde la petición antes de mostrarla en la vista:

```
<div *ngIf="users">
  {{ users }}
</div>
```

ANGULAR - SERVICIOS - HTTP GET

```
ngOnInit() {
  this.usersService.getUsers().subscribe(data => {
    this.users = data;
  });
```

- Crea un nuevo componente usersPage.
- Muesta con un ngFor todos los usuarios recibidos de la peticion http que hemos realizado. Deberás mostrar su id, email, first_name y last_name.

- Ahora en lugar de trabajar con users vamos a trabajar con los resources que nos ofrece Reqres.in
- Crea una nueva función getResources en el servicio para obtener todos los resources.
- Lista todos los resources con un ngFor mostrando los campos "name", "year" y "color".
- Muestra el resource con el mismo color que tenga en el campo "color".

ANGULAR - ROUTING - CREAR RUTAS EN ANGULAR

Si ya sabes cómo se crean los Componentes en Angular lo siguiente que puedes hacer es asignarlos a ciertas rutas para que el usuario pueda navegar por la web.
Por ejemplo, imagina que quieres crear la página /users. Pues lo primero que tienes que hacer es crear el componente UsersComponent (o como quieras llamarlo).

Para asignar este componente a una ruta necesitamos hacer uso del routing de Angular, necesitamos crear las rutas en **app.module.ts**

ANGULAR - ROUTING - CREAR RUTAS EN ANGULAR

Ahora, creamos una variable que guardará la lista de todas las rutas de la web:

```
const appRoutes = [
  { path: '', component: UsersComponent,  pathMatch: 'full'}
];
```

- **path**: La ruta a que queremos configurar
- **component**: Componente asociado a esa ruta. Para que funcione tienes que importar el componente en la parte de arriba.
- **pathMatch**: Esto es opcional, significa que toda la ruta URL tiene que coincidir (no solo cierta parte).

```
const appRoutes:Routes=[
  {path:'', component:HomePageComponent},
  {path:'users', component:UsersPageComponent},
  {path:'contact', component:ContactPageComponent},

];
```

ANGULAR - ROUTING - CREAR RUTAS EN ANGULAR

app.component.html

```
<router-outlet></router-outlet>
```

ANGULAR - ROUTING - CREAR RUTAS EN ANGULAR

- Nuestro componente padre se encargará del routing por lo que debemos crear un nuevo componente para mostrar la paǵina inicial.
- Crea un nuevo componente homePage que se cargará en la ruta /
    - En este componente tendremos todo el código referente a la creación de monstruos
- Crea un nuevo componente usersPage que se cargará en la ruta /users donde tendremos todo el código referente a users y resources.
- Crea un nuevo componente contactPage donde mostrarás tu nombre y apellidos junto con un link a tu repositorio de git donde tengas este proyecto.

ANGULAR - CREANDO RUTAS CON PARTES DINÁMICAS

Ahora, imaginemos que queremos crear una página para mostrar en detalle items de una lista. Por ejemplo queremos tener /item/1, /item/2, /item/3, etc. En lugar de crear cada una de esas rutas, puedes configurar que cierta parte de la ruta sea dinámica, es decir, que Angular genere todas las rutas por tí. Para ello tienes que poner dentro del path, a continuación de la barra /, dos puntos y nombre que quieras para esa variable:

```
const appRoutes = [
 { path: 'items', component: ItemListComponent,  pathMatch: 'full'},
 { path: 'item/:id', component: ItemDetailComponent }
];
```

ANGULAR - CREANDO RUTAS CON PARTES DINÁMICAS

Con esto se crean todas las rutas posibles que empiecen por /item. El nombre id sirve para guardar en esa variable la ruta específica en la que estamos, es decir, luego dentro del componente ItemDetailComponent puedes recoger ese valor para saber en qué ruta estás. Para ello dentro del constructor:

```
// item-detail.component.ts

import { ActivatedRoute } from '@angular/router';

...

constructor(private route: ActivatedRoute) {
  console.log(route.snapshot.params['id'];)
}
```

ANGULAR - CREAR RUTAS EN ANGULAR - SISTEMA DE ROUTING

- Crea una ruta dinámica para cargar un monster en función de su nombre
  - La ruta sera /monster/name
  - Deberá mostrar todos los detalles del monster

ANGULAR - PIPES

Las pipes son filtros o funciones que pones directamente en la vista para el dar formato a un dato que estés pintando.

Por ejemplo, imagina que quieres pintar una fecha, 'Fri Apr 15 1988 00:00:00 GMT-0700'. Si quieres pintarla en formato MM/DD/YYYY, con lo que sabes hasta ahora, vas a tener que crear una función en el componente que se ejecuta cuando se pinte la vista. Pues con las pipes vas a poder crear funciones que hacen esto automáticamente y que puedes reutilizar a lo largo de tu aplicación.

La ventaja de las pipes es que te vas a ahorrar mucho código en los componentes, lo que facilitará su lectura y mantenimiento.

ANGULAR - PIPES

```
{{ 'BBBBB' | lowercase }}
```

Y lo que imprimirá será: bbbbb

```
{{ 459.67 | currency: 'USD' | lowercase }}
```

Lo que imprimirá: $459.67 (se han aplicado las dos pipes, pero en este caso, no hay letras para ponerlas en minúsculas).

ANGULAR - PIPES

- AsyncPipe
- CurrencyPipe
- DatePipe
- DecimalPipe
- I18nPluralPipe
- I18nSelectPipe
- JsonPipe
- KeyValuePipe
- LowerCasePipe
- PercentPipe
- SlicePipe
- TitleCasePipe
- UpperCasePipe

```
{{ 345.76 | currency: 'EUR' }}
```

```
<p>{{ myVal | json }}</p>
```

```
<p>{{ [1,2,3,4,5,6] | slice:1:3 }}</p>
```

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

```
<p>{{ 'prueba' | uppercase }}</p>
```

```
{{myVar | date: 'M/d/yy'}}
```

```
<p>{{ 'PRUEBA' | lowercase }}</p>
```

https://angular.io/api?type=pipe

ANGULAR - PIPES PERSONALIZADAS

```
ng generate pipe
```

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({name: 'reversed'})
export class ReverseStr implements PipeTransform {
  transform(value: string): string {
    let newStr: string = "";
    for (var i = value.length - 1; i >= 0; i--) {
      newStr += value.charAt(i);
    }
    return newStr;
  }
}
```

Para usar esta pipe que acabamos de crear debemos importarla en la sección **imports** del **app.module.ts**
Por último para usarla en cualquier vista, simplemente creamos la pipe con el nombre que le dimos y listo:

```
{{ variable | reversed }}
```

ANGULAR - ANGULAR MATERIAL

Angular Material, es una de las Mejores librerías para Angular .
https://material.angular.io

```
ng add @angular/material
```

Para habilitar las animaciones y que puedan funcionar, tenemos que importar las de Angular en el app.module.ts.

```
imports: [BrowserAnimationsModule],
```

El siguiente paso es importar los componentes que vayamos a utilizar.

ANGULAR - ANGULAR MATERIAL

Revisa los diferentes componentes que nos ofrece Angular Material y siguiendo la documentación oficial, instala Angular Material y uno de sus componentes e integralo en tu proyecyo.

https://material.angular.io