

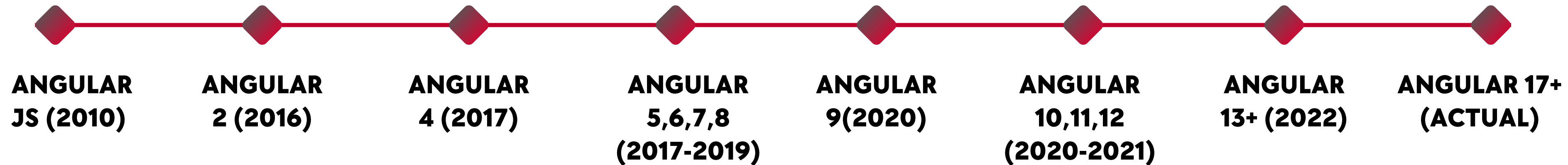
ANGULAR JS



INTRODUCCIÓN A ANGULAR

HISTORIA Y EVOLUCIÓN

Versiones de Angular



CARACTERÍSTICAS PRINCIPALES

1

SEPARACIÓN FRONTEND Y BACKEND

Enfoca claramente la separación entre frontend y backend de una aplicación.

2

DATA BINDING

Angular usa un enfoque de datos bidireccional, esto significa que los cambios del modelo se ven directamente reflejados en la vista y viciversa.

3

SIMPLICIDAD EN EL CÓDIGO

Permite realizar múltiples tareas con un código reducido. Se basa en el modelo vista controlador y utiliza componentes para simplificar el desarrollo.

4

USO DE PLANTILLAS

Utiliza plantillas para definir la interfaz de usuario. Combina plantillas con el código del componente para renderizar la vista final.

5

CONSTRUIDO CON TYPESCRIPT

Desarrollado con TypeScript para mejorar la mantenibilidad y la detección de errores en el código.

6

INYECCIÓN DE DEPENDENCIAS

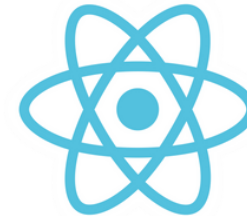
Implementa un sistema de inyección de dependencias para gestionar y organizar componentes. Facilita la creación, prueba y mantenimiento de aplicaciones mediante una mejor modularidad.

DIFERENCIAS ENTRE TECNOLOGÍAS



Angular

- Framework completo
- Trabaja con DOM real (manipulación directa de los elementos HTML)
- Enlaza los datos de forma bidireccional
- Utiliza plantillas HTML + TypeScript
- Tiene libertad limitada



React

- Biblioteca JavaScript
- Trabaja con DOM virtual (menos manipulación del DOM real)
- Enlaza los datos de forma unidireccional
- Utiliza plantillas JSX+JS(ES5/ES6)
- Permite elegir bibliotecas, arquitecturas y herramientas

CONFIGURACIÓN DEL ENTORNO DE DESARROLLO

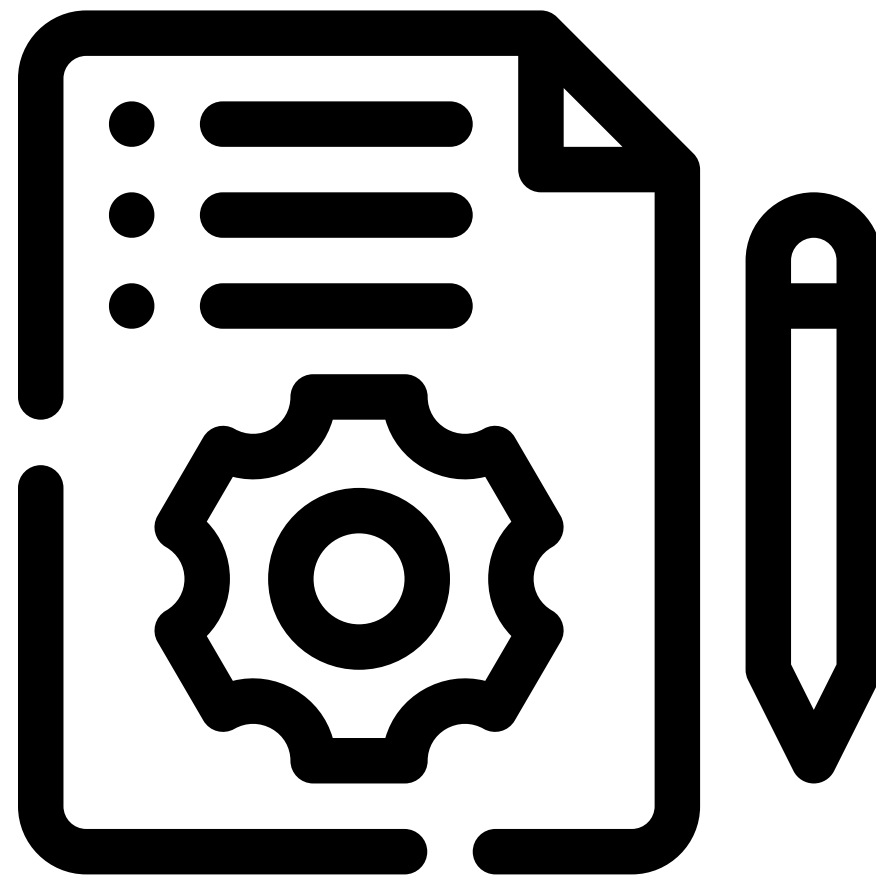


NODE JS



ANGULAR CLI

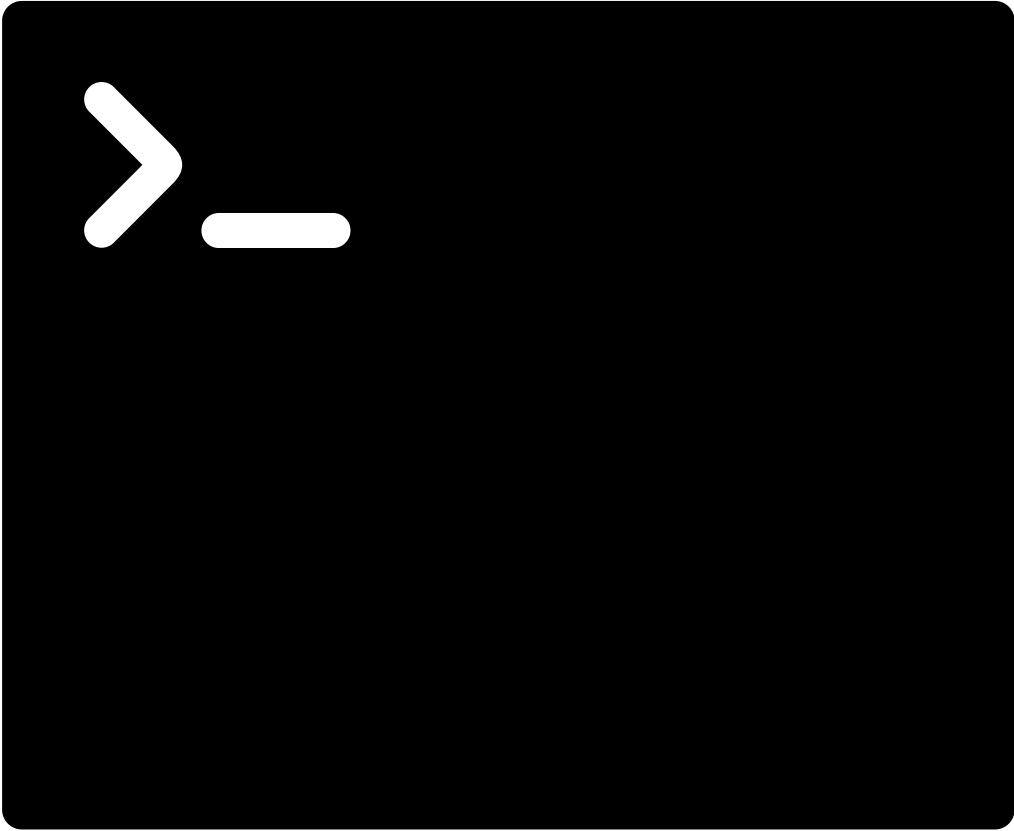
CREACIÓN DE UN NUEVO PROYECTO





ESTRUCTURA DEL PROYECTO

USO BÁSICO DE ANGULAR CLI

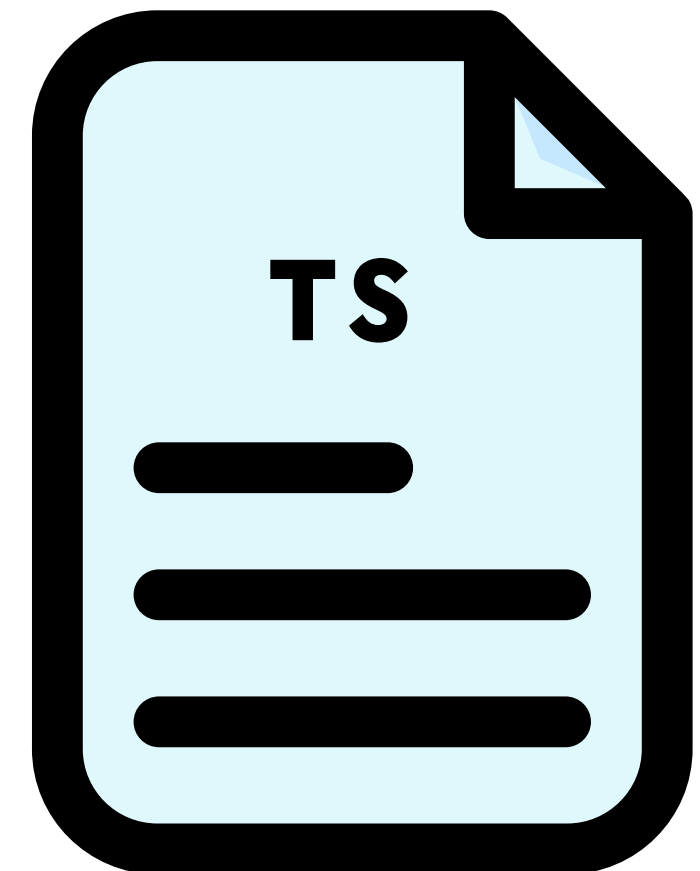
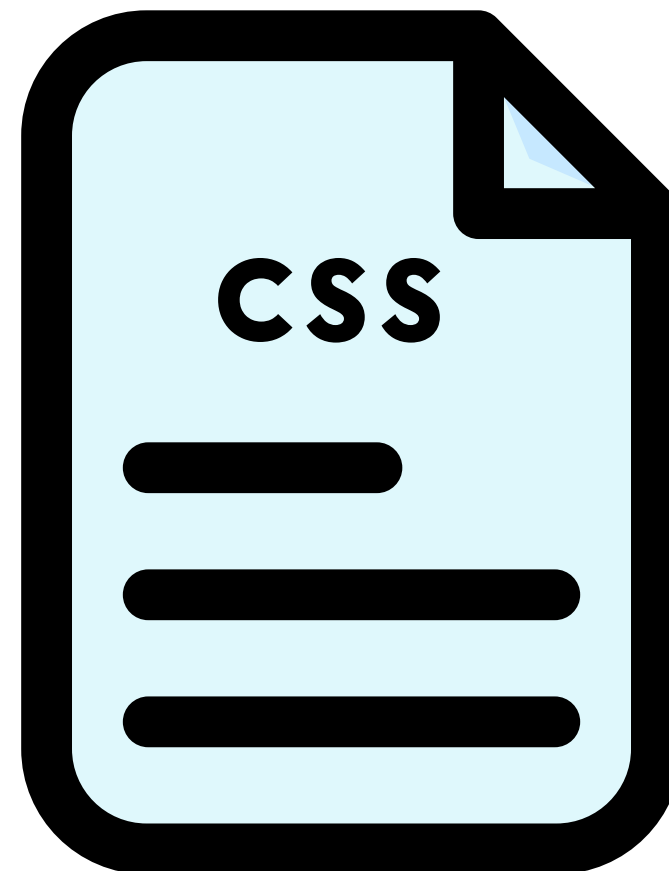
A black rounded rectangle representing a terminal window, containing a white prompt character '>_'.

```
>_
```



CONCEPTOS BÁSICOS

COMPONENTES EN ANGULAR



PROPIEDADES Y EVENTOS

Propiedades

Tipos de propiedades

Input properties

Son propiedades que permiten la comunicación entre componentes, decorándolos con el decorador `@Input()` en el componente hijo.

Además, permiten la transferencia de datos desde el componente padre al componente hijo. Ejemplo:

```
@Input() nombre: string;
```

Output properties

Se utiliza para emitir eventos desde un componente hijo al componente padre. Se suelen decorar con el decorador `@Output()` y son instancias de la clase `EventEmitter`.

```
@Output() onEvento =  
new EventEmitter<string>();
```

ngModel

Se utiliza para implementar enlaces bidireccionales en los formularios, permitiendo la sincronización de datos entre el modelo y la vista. Por ejemplo:

```
<input [(ngModel)]= "nombre" />
```

PROPIEDADES Y EVENTOS

Eventos

Tipos de eventos

Event Binding

Permite la respuesta a eventos del usuario, como clics de botones, cambios en cuadros de texto, etc. Además, se utiliza el paréntesis `()` para vincular un evento a un método en el componente.

```
<button (click)="onClick()">Click</button>
```

Eventos personalizados

Eventos que puedes definir y emitir desde un componente hijo para ser capturados por un componente padre. Estos eventos son útiles cuando necesitas comunicación entre componentes. Por ejemplo:

```
@Output()  
propagar = new EventEmitter<string>();
```

```
this.propagar.emit('Este dato viajará hacia el padre');
```

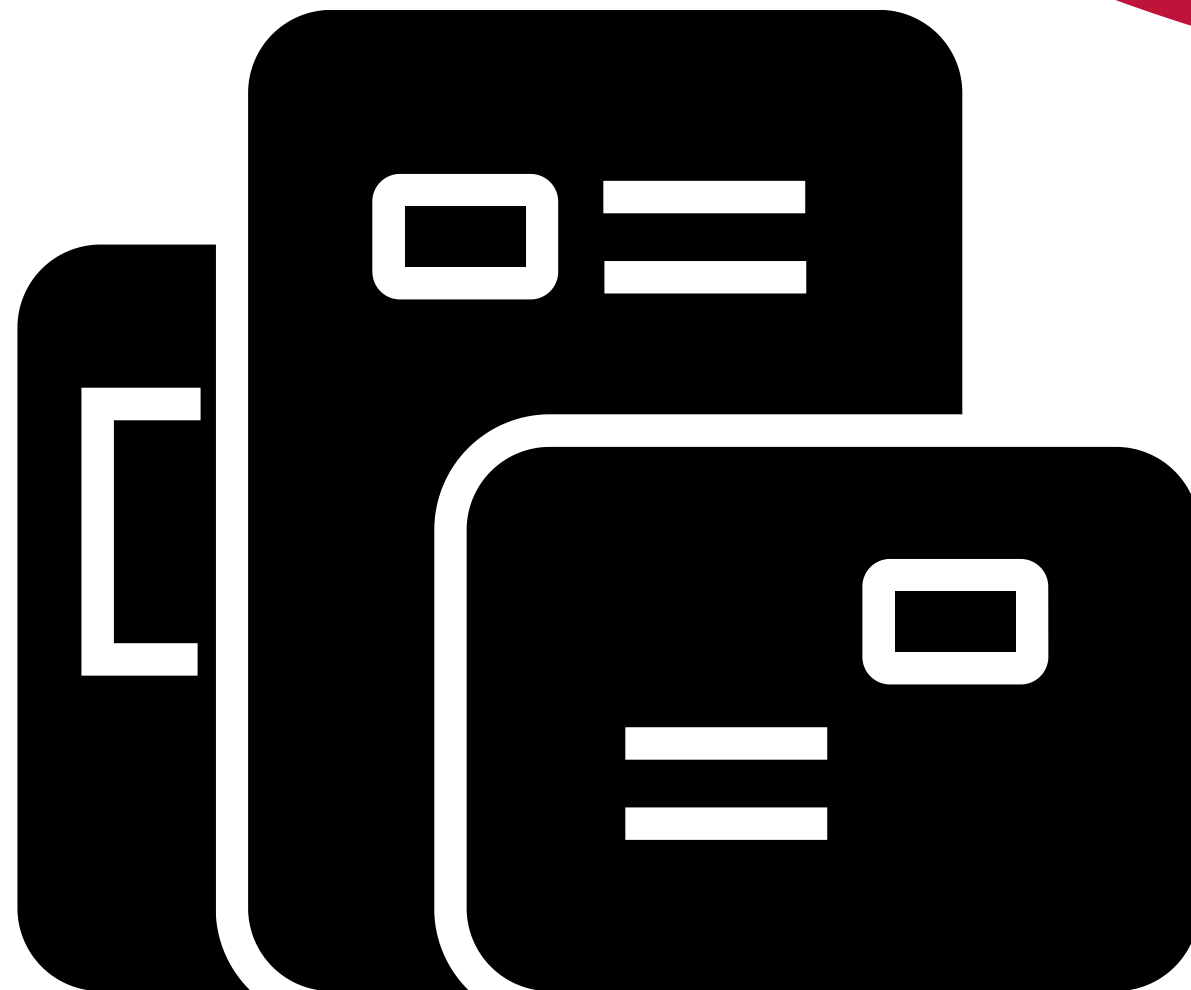
```
<p>  
  <input type="text" [(ngModel)]="mensaje">  
  <button (click)="onPropagar()">Propagar</button>  
</p>
```

Eventos de ciclo de vida

Se refieren a una serie de ganchos o métodos específicos que son invocados en momentos clave durante el ciclo de vida de un componente. Estos eventos permiten a los desarrolladores ejecutar código en momentos específicos, como la inicialización, la actualización y la destrucción del componente. Ejemplo:

```
export class EjemploComponente implements OnInit {  
  
  ngOnInit() {  
    alert('Hola, mundo');  
  }  
}
```

EVENT BINDING



DIRECTIVAS EN ANGULAR





DIRECTIVAS ESTRUCTURALES

DIRECTIVA @IF

EL USUARIOEXISTENTE ES UNA PROPIEDAD DE LA CLASE QUE POR DEFECTO ESTÁ EN TRUE.

```
@if (usuarioExistente) {  
    <p> El usuario existe </p>  
}  
@else {  
    <p> El usuario no existe </p>  
}
```

EN LA CLASE: USUARIOEXISTENTE:BOOLEAN = TRUE;

DIRECTIVA @FOR

Teniendo en cuenta que en la clase tenemos una propiedad 'animales' que es un array de objetos (de animales), con 2 atributos cada animal, el id y la especie. El track es para identificar de manera única cada elemento de la lista para realizar un seguimiento de los cambios.

```
<ul>
```

```
  @for (a of animales; track a.id) {
```

```
    <li> {{a.especie}} </li>
```

```
  }
```

```
</ul>
```

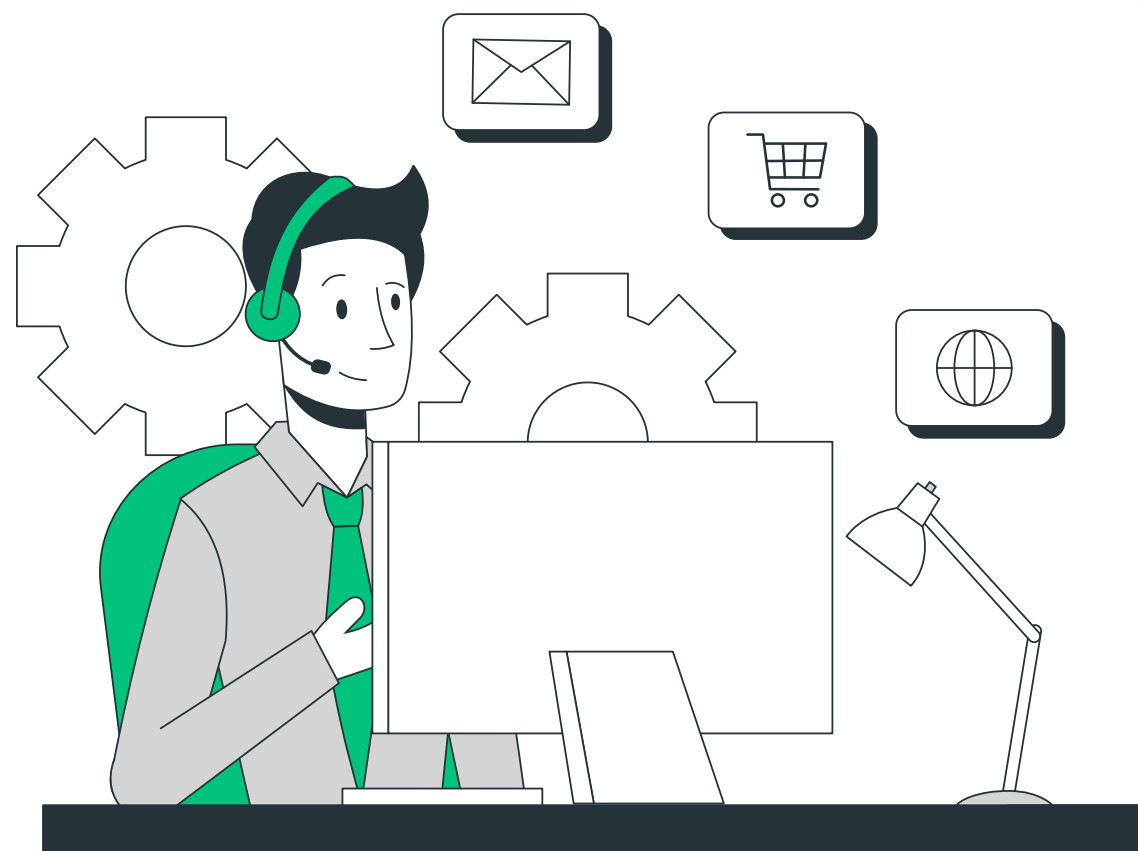
EN LA CLASE: ANIMALES = [{ID=1, ESPECIE="PERRO"}, {ETC}]

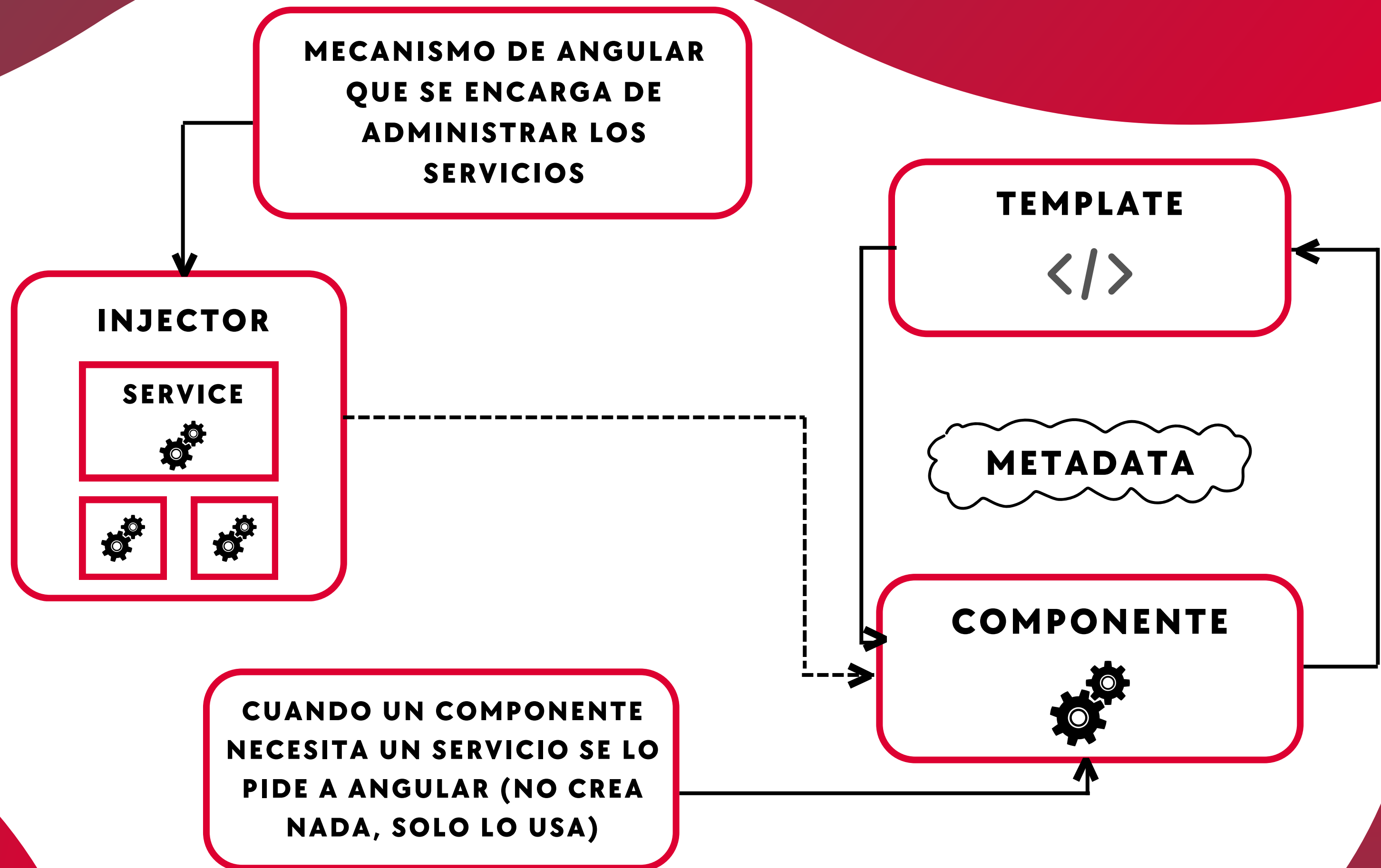
DIRECTIVAS DE ATRIBUTO

TIPOS

- **ngClass:** Se usa para añadir o eliminar varias clases CSS, para cambiar la apariencia.
- **ngModel:** Se utiliza para implementar binding.
- **ngStyle:** Se usa para añadir elementos que cambien la apariencia.

SERVICIOS Y DEPENDENCIAS







FUNCIONALIDAD DE LOS SERVICIOS



CREAR Y USAR UN SERVICIO

INYECCIÓN DE DEPENDENCIAS

PRIVATE INDICA QUE LA VARIABLE "DATASERVICE" VA A SER ACCESIBLE SÓLO EN ESTA CLASE

EL NOMBRE DE LA VARIABLE QUE SE USA EN ESTE COMPONENTE

":" INDICA LA SEPARACIÓN ENTRE EL NOMBRE DE LA VARIABLE Y LA CLASE

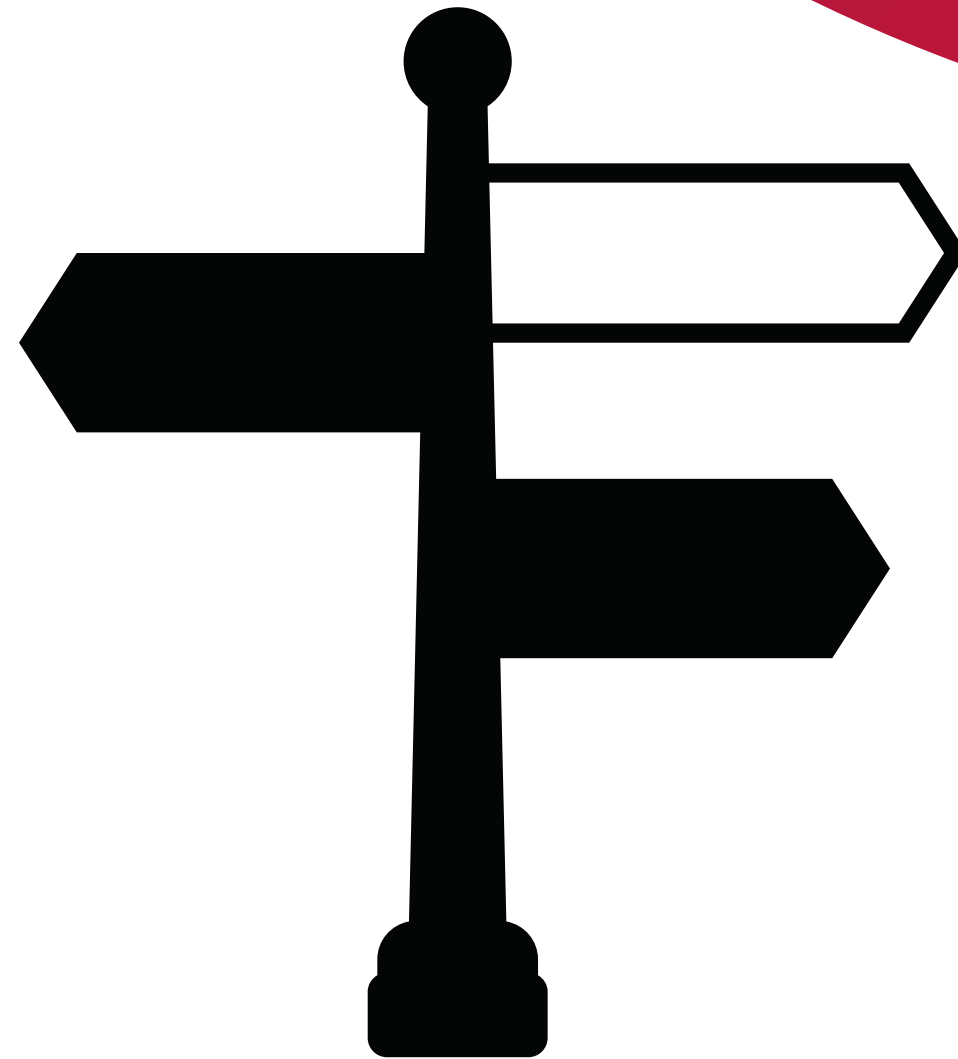
CLASE QUE SE LE ASIGNA A LA VARIABLE "DATASERVICE"

```
constructor(private dataService: DataService) {}
```

ADEMAS ESTA CLASE ES EL IDENTIFICADOR EN LA "BOLSA DE SERVICIOS" QUE USA ANGULAR (ESTO LO HACE ANGULAR SÓLO SI ESTÁ EN UN CONSTRUCTOR)

NAVEGACIÓN EN ANGULAR

CONFIGURACIÓN DE RUTAS



NAVEGACIÓN ENTRE COMPONENTES

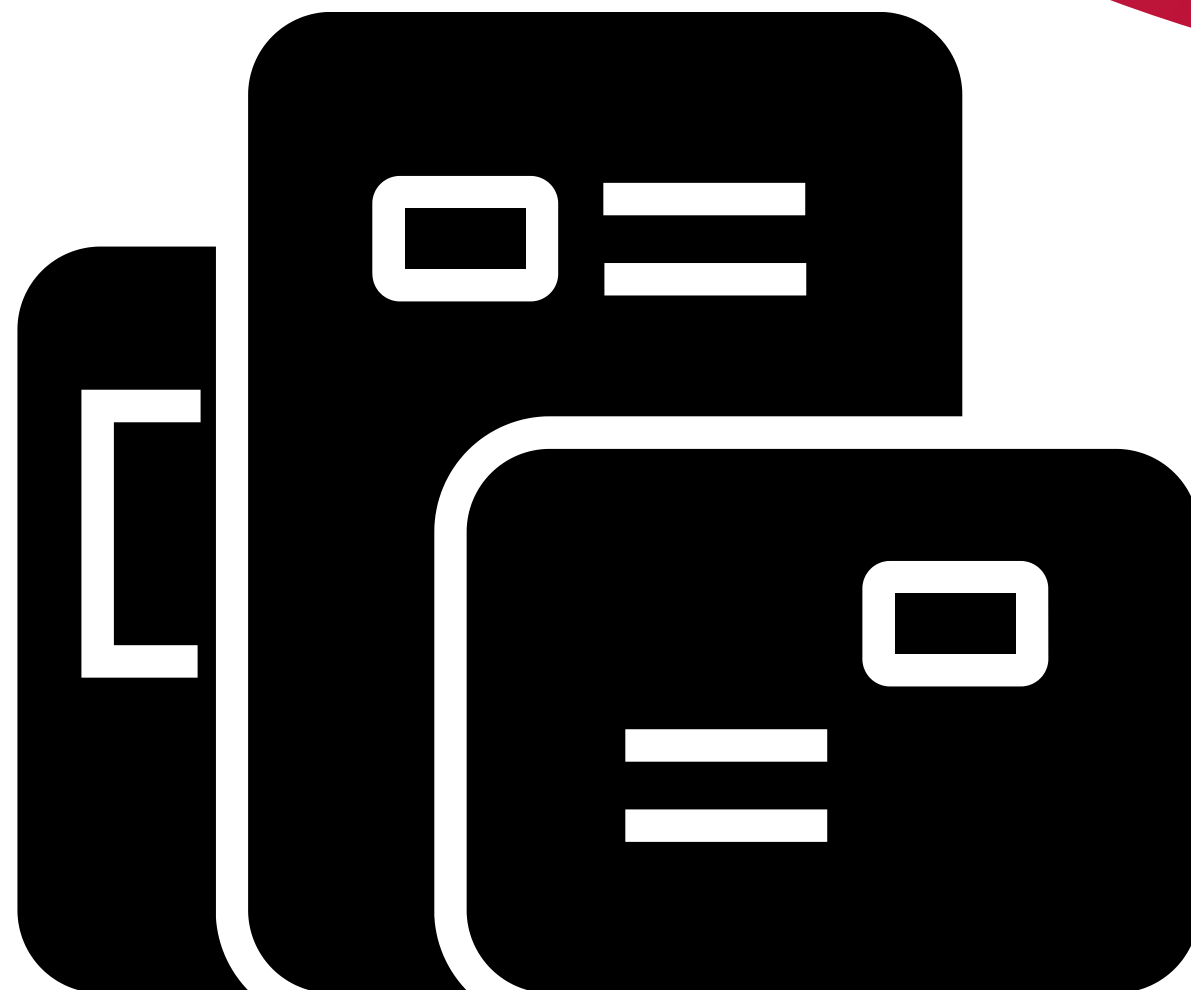


FORMULARIOS EN ANGULAR

¿QUÉ SON LOS FORMULARIOS EN ANGULAR?



FORMULARIOS TEMPLATE-DRIVEN



FORMULARIOS REACTIVOS



USO DE FORMULARIOS DINÁMICOS

1

DEFINICIÓN

Un formulario dinámico permite a los usuarios generar campos mediante acciones específicas.

2

CARACTERÍSTICAS CLAVES

El formulario no tiene un límite predefinido de campos. El usuario tiene el control total sobre la cantidad que se genera.

3

FLEXIBILIDAD

El usuario puede añadir o eliminar elementos según sus necesidades, otorgando mayor versatilidad y adaptabilidad al formulario.

4

CONTROL DEL USUARIO

En un contexto dinámico, el usuario tiene la capacidad de personalizar la cantidad de campos, proporcionando una experiencia más interactiva y centrada en sus preferencias.

USO DE FORMULARIOS DINÁMICOS

```
componente1.component.ts U  componente2.component.html U  TS componente2.component.ts U X
Formularios > src > app > componente2 > TS componente2.component.ts > Componente2Component

11  export class Componente2Component {
12
13      reactForm: FormGroup;
14
15      constructor(private formBuilder: FormBuilder){
16          this.reactForm = this.formBuilder.group({
17              nombre: new FormControl('', [Validators.required, Validators.minLength(3)]),
18              apellidos: new FormControl('', [Validators.required]),
19              direcciones: this.formBuilder.array([])
20          });
21      }
22
23      getDireccion(){
24          return this.reactForm.get('direcciones') as FormArray;
25      }
26
27      agregarFormularioDireccion(){
28          const direccionFormGroup = this.formBuilder.group({
29              calle: '',
30              ciudad: '',
31              codigoPostal: ''
32          });
33          this.getDireccion().push(direccionFormGroup);
34      }
35
36      imprimePorConsola(){
37          console.log(this.reactForm.value);
38      }
39  }
```

USO DE FORMULARIOS DINÁMICOS

```
component.html U  TS componente1.component.ts U  componente2.component.html U X  TS component
Formularios > src > app > componente2 > componente2.component.html > div > form
26
27     <br> <br>
28
29     <button type="button"
30     (click)="agregarFormularioDireccion()">Agregar dirección</button>
31     <br><br>
32
33     <form formArrayName="direcciones">
34
35         @for (dir of getDireccion().controls; track dir) {
36
37             <div formGroupName="{{ $index }}">
38                 <label for="'calle' + {{ $index }}">Calle:</label>
39                 <input type="text" id="'calle' + {{ $index }}"
40                 formControlName="calle"> <br>
41
42                 <label for="'ciudad' + {{ $index }}">Ciudad:</label>
43                 <input type="text" id="'ciudad' + {{ $index }}"
44                 formControlName="ciudad"> <br>
45
46                 <label for="'codigoPostal' + {{ $index }}">Codigo postal:</label>
47                 <input type="text" id="'codigoPostal' + {{ $index }}"
48                 formControlName="codigoPostal"><br><br>
49
50             </div>
51         }
52
53     </form>
54
55
```

USO DE FORMULARIOS DINÁMICOS

Formulario reactivo:

Nombre: (El nombre no puede estar vacío)

Apellidos: (Los apellidos no puede estar vacío)

Formulario reactivo:

Nombre: (El nombre no puede estar vacío)

Apellidos: (Los apellidos no puede estar vacío)

Calle:

Ciudad:

Código postal:

Angular is running in development mode.

Angular hydrated 2 component(s) and 28 node(s), 0 component(s) were skipped. Learn more at <https://angular.io>.

[componente](#)

```
▼ {nombre: '', apellidos: '', direcciones: Array(1)} |
  apellidos: ""
  ▼ direcciones: Array(1)
    ▼ 0:
      calle: "Cisne"
      ciudad: "Madrid"
      codigoPostal: "28001"
      ► [[Prototype]]: Object
      length: 1
      ► [[Prototype]]: Array(0)
      nombre: ""
      ► [[Prototype]]: Object
```

>



CONSUMO DE DATOS

ACCESO A DATOS

¿Qué es el acceso a datos?

¿Qué son las funciones asíncronas?

Tipos de peticiones:

- GET
- POST
- PUT
- DELETE



INTEGRACIÓN DE DATOS EN COMPONENTES

INTEGRANTES



**DAVID DORANTE
LUCAS**



**ÁLVARO SÁNCHEZ
MORENO**



**JUAN MARTÍN
CANDELA**



**JOSE MARÍA
MAÑERO BRENES**



**ANTONIO JESÚS
LEÓN FERNÁNDEZ**

¡GRACIAS!