

CREANDO UN NUEVO COMPONENTE

»» COMPONENT

Componente. ¿qué es?

- ▶ Elemento básico para construir la GUI de la app.
- ▶ Controla un fragmento de la pantalla

example.component.ts

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

```
@Component({  
  selector: 'example',  
  template: `  
    <div>  
      <h1>{{msg}}</h1>  
    </div>  
  `
```

```
})  
export class ExampleComponent {  
  msg: string = 'Hola mundo!!!';  
}
```

Metadata

Configuración
(decorador)

Template
< >

Vista

Component
{ }

Controlador
(clase TypeScript)

Componente. Convenciones

- ▶ Nombre fichero de un componente:

nombre_del_componente.component.ts

Ej: example.component.ts

- ▶ Nombres de los ficheros **SIEMPRE** en MINÚSCULAS

- *usando guiones si hace falta espacios*

Ej: barra-navegacion.component.ts

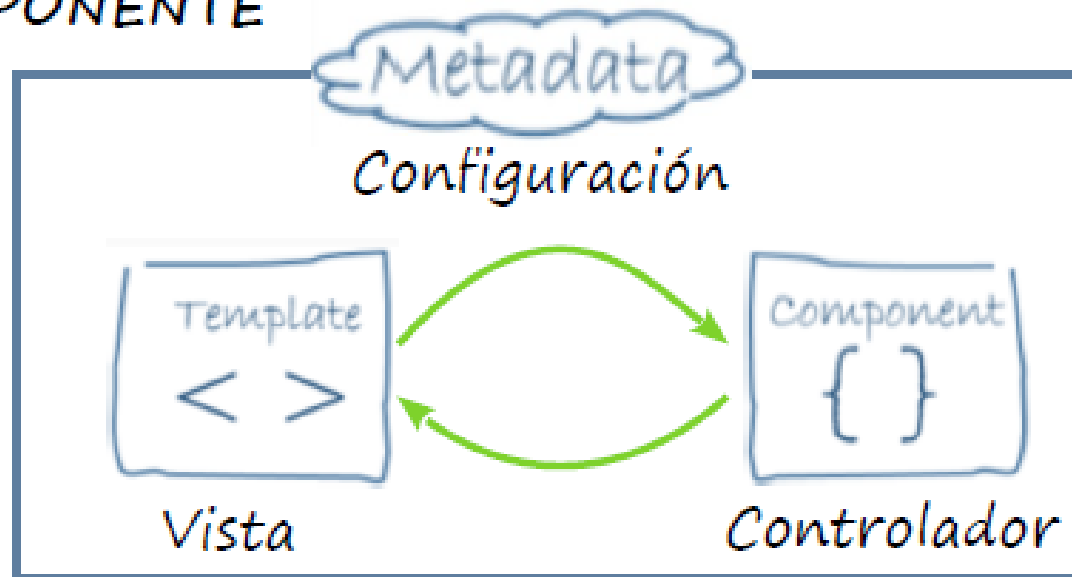
- ▶ Nombres de la **clase** que representa el componente usa **NOMENCLATURA CAMEL CASE**

*Ej: export class BarraNavegacionComponent {
...}*

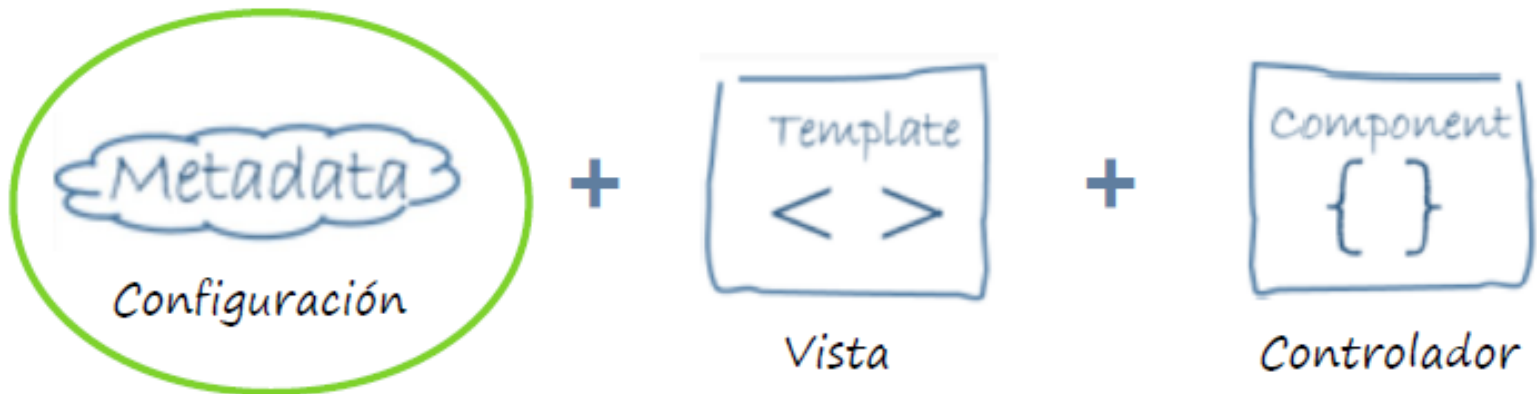
Componente. Resumen

- ▶ La **vista** construye la GUI e interacciona con el usuario (HTML)
 - GENERA EVENTOS al interactuar con el usuario
- ▶ El **controlador** contiene la lógica y reacciona a eventos.
 - **Comunicaremos** la vista con el controlador en ambos sentidos
- ▶ El componente se **configura** con metadatos

COMPONENTE



Componente

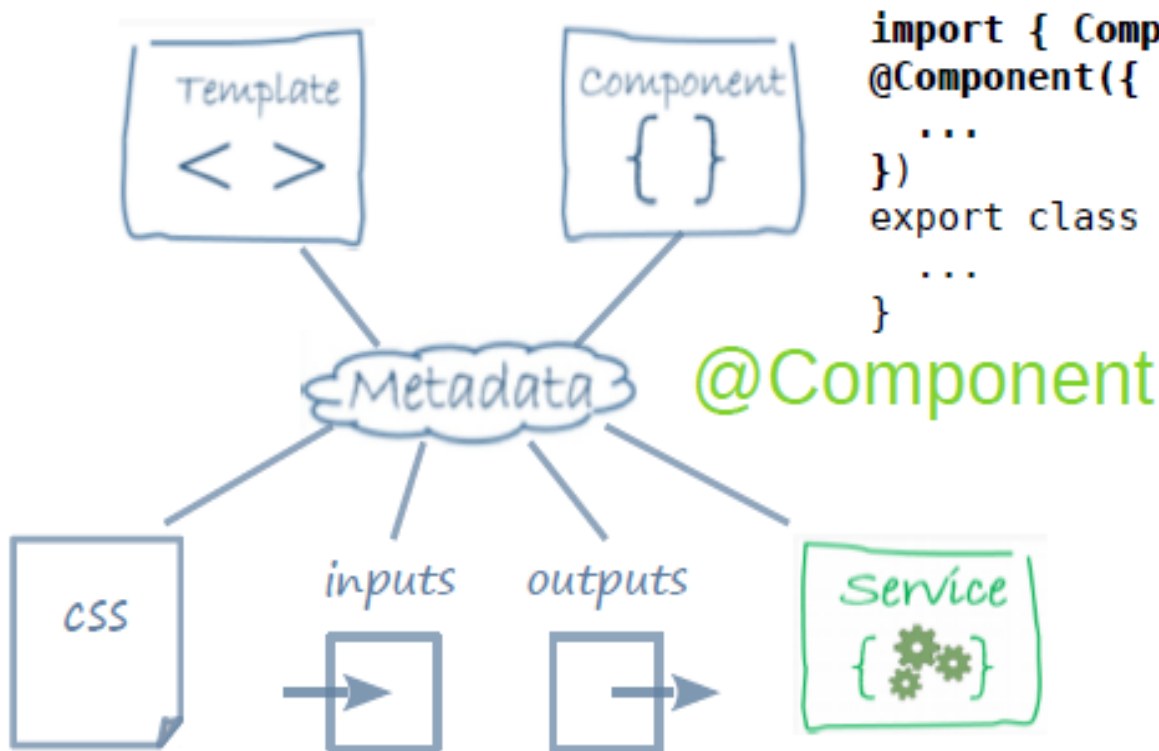


Componente

Configuración

Metadata

- ▶ **@Component** contiene la configuración del componente.
- ▶ Es el pegamento que lo une todo!!!!



En la configuración indicaré:

- **Servicios** que usa el componente
- **Entradas y salidas** del componente
- **Hoja estilos** del componente
- Su **plantilla html**

Ejemplo de creación de un componente

- ▶ Vamos a crear un nuevo componente:
 - Cuando esté creado, podremos usarlo en nuestro HTML mediante la etiqueta:

`<app-hello-world></app-hello-world>`

- Comando para crearlo:

`$ ng generate component hello-word`

```
create src/app/hello-word/hello-word.component.html (29 bytes)
create src/app/hello-word/hello-word.component.spec.ts (650 bytes)
create src/app/hello-word/hello-word.component.ts (284 bytes)
create src/app/hello-word/hello-word.component.css (0 bytes)
update src/app/app.module.ts (410 bytes)
```

- Si ahora vamos al editor Visual Code, veremos que se han creado esos 4 ficheros. Si abrimos el que tiene extensión .ts

¿Cómo se define un componente?

- ▶ Tiene dos partes:
 - a) Un decorador
 - b) Una clase que define el componente.
- ▶ Se define en un fichero TypeScript (*.ts*)

```
TS hello-word.component.ts x
1  import { Component, OnInit } from '@angular/core';
2
3  @Component({
4    selector: 'app-hello-word',
5    templateUrl: './hello-word.component.html',
6    styleUrls: ['./hello-word.component.css']
7  })
8  export class HelloWorldComponent implements OnInit {
9
10     constructor() { }
11
12     ngOnInit() {
13     }
14
15 }
16
```

- Importa el **decorador Component** de la librería de Angular (módulo) “@angular/core”.
- Importa **OnInit** de la misma librería (módulo)

- Método de la **interfaz OnInit** que pertenece al ciclo de vida del componente (lo veremos más adelante).

¿Cómo se define un componente?:

DECORADOR

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

Con la propiedad **styleUrls** estamos indicando qué *hojas de estilos CSS* (puede ser una o varias), se aplicarán a este componente

Con la propiedad **selector** estamos definiendo una nueva “tag” para usar en nuestro HTML de la APP y que estará *ligada a la clase **HelloWordComponent*** definida más abajo.

Con la propiedad **templateUrl** estamos definiendo qué HTML se verá cuando se cargue este componente. En este ejemplo, será el fichero: “hello-word.component.html”. La **VISTA**.

¿Cómo se define un componente?:

DECORADOR

```
@Component({
  selector: 'app-hello-word',
  template: `
    <p>
      hello-world works!!!!
    </p>
  `,
  styleUrls: ['./hello-word.component.css']
})
```

Con la propiedad **template** estamos definiendo directamente el HTML que se mostrará cuando se cargue el componente. INCRUSTADO

IMPORTANTE: Fijaros en las comillas del string que contiene “template”. Esas comillas son estas: ` ` .
Estas comillas permiten *definir un string MULTILINEA*.

Cargando el nuevo componente

- ▶ Para cargarlo necesitamos añadir la “tag” de nuestro componente (que en nuestro caso es `<app-hello-world></app-hello-world>`) a una plantilla HTML que ya esté siendo renderizada (en nuestro caso a [app.component.html](#)):

```
1  <!--The content below is only a placeholder and can be replaced.-->
2  <div style="text-align:center">
3      <h1>
4          Welcome to {{title}}!
5
6          <app-hello-world></app-hello-world>
7      </h1>
8  </div>
```

Si ahora recargamos nuestra app en el navegador, veremos:

Cargando el nuevo componente

Welcome to app!

hello-world works!



¿Qué pasa con el resultado de la compilación?

▶ **ng serve**

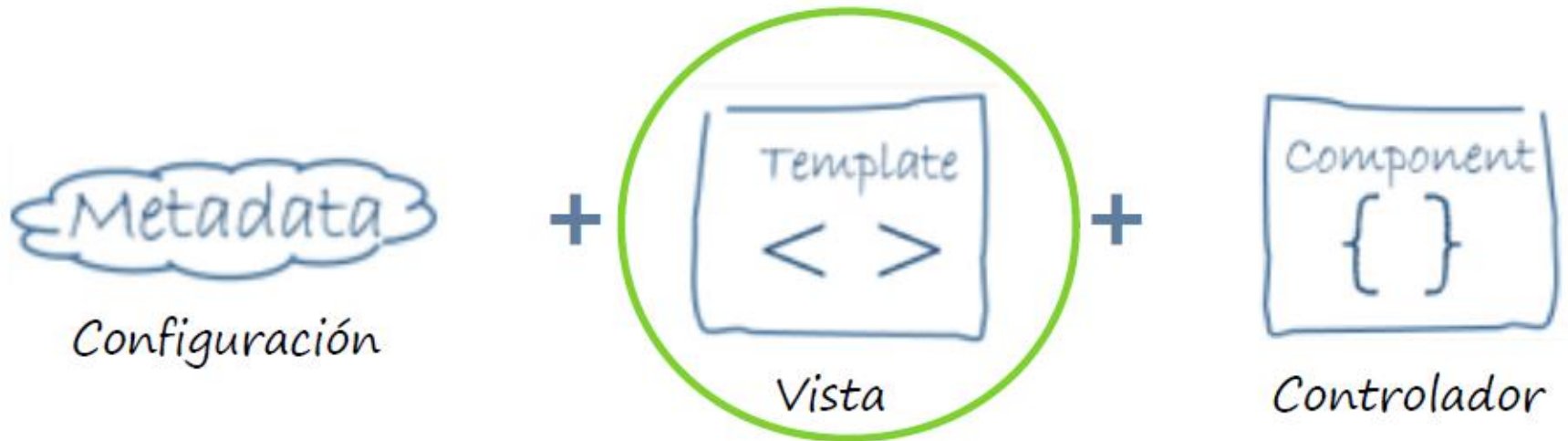
- *Compila, empaqueta y lo mantiene todo en memoria*
- *No lo guarda en disco*
- *Sirve en el proceso de desarrollo, no en producción*

▶ Si queremos **generar todos los javascript y recursos estáticos para colgarlos en un servidor web**:

◦ **ng build**

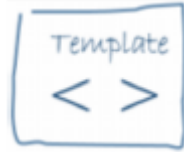
- *Este comando compila y lo **guarda en disco**, pero no arranca ningún servidor*
- *Crea una nueva carpeta “**dist**” en nuestro proyecto que contiene el resultado de la compilación:*
 - **index.html**
 - fichero principal, lo ha cambiado, si os fijais: añade una serie de **scripts** (son los ficheros javascript que han sido generados automáticamente por la herramienta cliente de angular)
 - **main.bundle.js**
 - resultado de compilar todos nuestros componentes y meterlos en un fichero.
 - Usa webpack para empaquetarlo todo en un único fichero.

Componente



Componentes

Vista



- ▶ Representa la GUI y recibe la **interacción del usuario**.
- ▶ Plantilla HTML puede incluir: **expresiones**, **directivas** y **componentes**
- ▶ **Se refresca automáticamente si detecta cambios en el modelo**

example.component.html

expresión

Hello {{ name }}, your tasks are:

directiva

componente

<li *ngFor="let u of users">

<user-details></user-details>

user-details.component.ts

```
@Component({
  selector: 'user-details',
  templateUrl: './user-details.component.html',
  styleUrls: ['./user-details.component.css']
})
export class UserDetailsComponent { ... }
```

Componentes

Vista > Expresiones

- ▶ Interpolación de datos con `{{ ... }}` → se llama *Template expressions*

- Permite incrustar datos (variables, arrays, constantes, llamar funciones,...) en la plantilla

Hello `{{ user }}`

Hello `{{ user.name }}`

Hello `{{ 1+1 }}`


- ▶ El **contexto de la expresión** (**¿dónde está?**) es el **componente** (su **controlador**, su clase). Ej:

example.component.html

```
<p>Hello {{ user }}</p>
```

example.component.ts

```
@Component({ ... })  
export class ExampleComponent {  
  user: string;  
  constructor() {  
    this.user = 'Pepe';  
  }  
}
```



- ▶ **IMPORTANTE**: si “user” cambia en el controlador, *automáticamente Angular refresca la plantilla.*

SINCRONIZACIÓN *fuente → plantilla*

- ▶ La sintaxis es un subconjunto de JS (**eliminando objetos globales, modificación de datos como =, ++, new,...**)

Componentes

Vista > Expresiones

[enlace documentación pipes](#)

- ▶ Transformaciones con **tuberías** `{{ ... | <pipe> }}`

- Permite modificar cómo se visualiza ese dato


Hello `{{ user | uppercase }}`

Today is `{{ today | date: 'longDate' }}`

- ▶ Tuberías **built-in**: uppercase, lowercase, date, ...
- ▶ Se pueden **encadenar**: `{{ ... | <pipe1> | <pipe2> | <pipe...> }}`
- ▶ Una tubería es una clase con **@Pipe**
- ▶ Una tubería **admite parámetros** que van con ":". Ver ejemplo "date" arriba.

mypipe.pipe.ts

```
import { Pipe, PipeTransform } from '@angular/core';
@Pipe({name: 'mypipe'})
export class MyPipe implements PipeTransform {
  transform(value: number, arg: number): number {
    return value + arg;
  }
}
```

 The number is: `{{ 5 | mypipe: 5 }}`

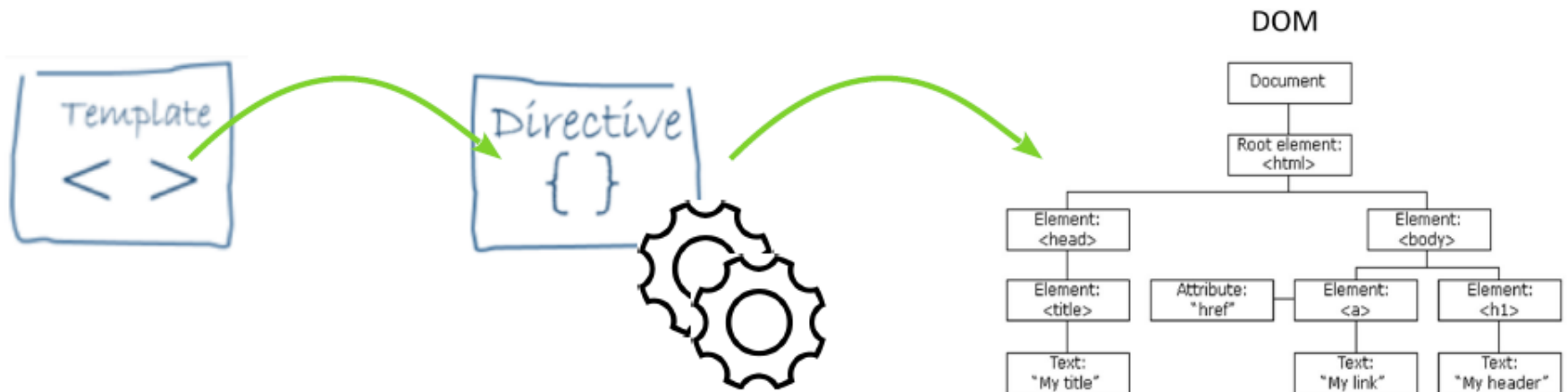
Componentes

Vista > Directivas

- ▶ Angular está basado en directivas muy potentes. Todo son directivas.
- ▶ *Extienden el lenguaje HTML usado en las plantillas*

Ej: Tarea enviar email ` completada `

- ▶ A partir de la plantilla, **manipulan el DOM**, generando la VISTA.



- ▶ **Las veremos más adelante con más detalle.**


Componentes

Vista > Directivas

- ▶ Una directiva es una clase con **@Directive**

myhighlight.directive.ts

```
import { Directive, ElementRef } from '@angular/core';
@Directive({
  selector: '[myHighlight]'
})
export class MyHighlightDirective {
  constructor(el: ElementRef) {
    el.nativeElement.style.backgroundColor = 'yellow';
  }
}
```

 <p myHighlight>Highlight me!</p>

- ▶ Este ejemplo lo que hace es:

- Todos los elementos del DOM que tengan un atributo “myHighlight” harán que se active la directiva. En el ejemplo, sería <p>
- ¿por qué el atributo: “myHighlight”? Porque es el **selector** que he puesto a mi directiva y lo he puesto entre []. Esos [] representan un **atributo en los selectores CSS**.
- *¿qué hace esta directiva* mía?: coge el elemento del DOM que tenga ese atributo “myHighlight” y le cambia el color de fondo a “yellow”.

- ▶ NOSOTROS USAREMOS **DIRECTIVAS built-in** (*ya creadas por Angular*).

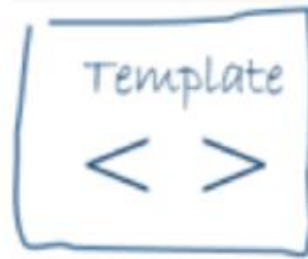
Componente



Metadata

Configuración

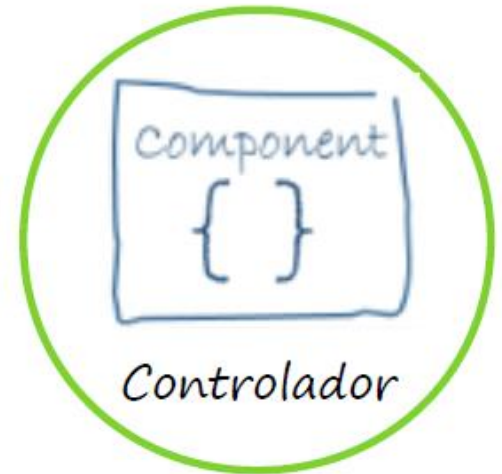
+



Template

Vista

+

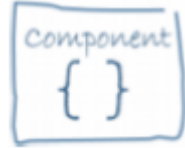


Component

Controlador

Componentes

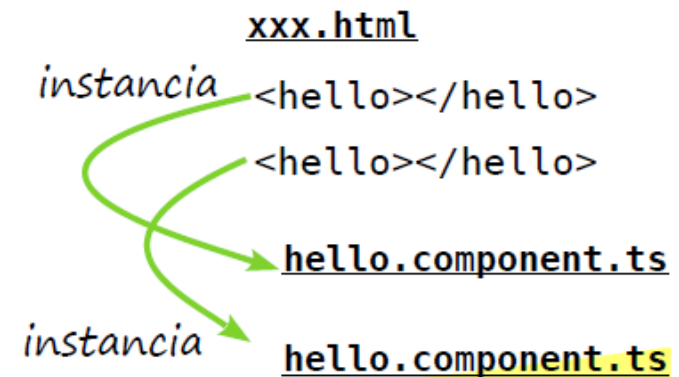
Controlador



- Clase que contiene la lógica del componente

```
@Component({ ... })
export class TasksListComponent {
  tasks: Task[];
  constructor() { this.tasks = [{ name: 'task1' }, { name: 'task2' }]; }
  selectTask(t: Task) { console.log('Task ' + t.name + ' selected!'); }
}
```

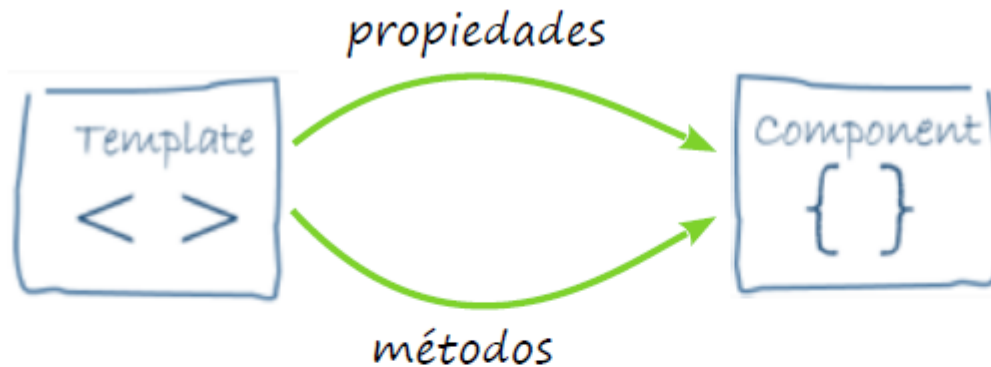
- Cuando se crea un componente, Angular instancia su controlador



Componentes

Controlador > Interacción con la vista

- ▶ El controlador interactúa con la vista a través de **propiedades y métodos**: *es el **contexto de la plantilla**.*
 - Normalmente, desde la plantilla accedemos a elementos que están en el controlador:



hello.component.html

```
<div>
  Hello {{ msg }},
</div>
Today is {{ getDate() }}
```

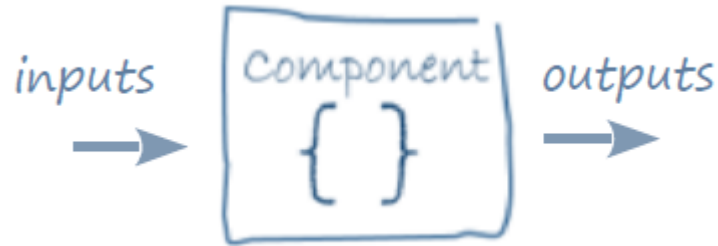
hello.component.ts

```
@Component({ ... })
export class HelloComponent {
  msg: string;
  getDate(): string { ... }
}
```

Componentes

Controlador > API

- ▶ A veces, el controlador necesita datos (**entradas**) y genera eventos (**salidas**): es el **API del componente**.



- ▶ En Angular, **todo son componentes** (con API):
 - Incluso los elementos del DOM de HTML son tratados como COMPONENTES. Ej: un botón, un input, un <p>,...
 - Así:
 - las propiedades de un elemento del DOM → en Angular = **ENTRADA**
 - los eventos que genera un elemento del DOM → en Angular = **SALIDA**

