

Angular Components

WEB APPLICATION
DEVELOPMENT

Introducción

- ¿Qué es un componente?
- Creación de componentes.
- Estructura de un componente.
- Comunicación entre componentes.

¿Qué es un componente?

Un componente Es el bloque fundamental de cualquier aplicación Angular. Cada componente en Angular está compuesto por cuatro elementos principales:

- Clase (TypeScript): Define la lógica y el comportamiento del componente. Se declara una clase TypeScript y contiene propiedades y métodos que manejan los datos y las interacciones del usuario.
- **Decorador (@Component):** Especifica metadatos sobre el componente, como su selector, plantilla y estilos.
- Plantilla (HTML): Define la estructura de la interfaz de usuario del componente. La plantilla puede incluir enlaces de datos, directivas y otros componentes.
- Estilos (CSS/SCSS): Define la apariencia visual del componente.

Estructura de un componente

• Clase TypeScript (ExampleComponent): Define una propiedad "title" y un método "onClick" que actualiza el valor de "title".

Decorador @Component:

- **selector:** Define el nombre del elemento HTML que representa este componente (**<app-example>**).
- **templateUrl:** Apunta al archivo HTML que contiene la plantilla de este componente.
- **styleUrls:** Apunta al archivo CSS que contiene los estilos de este componente.
- Plantilla (HTML): Usa la interpolación ({{title}})
 para mostrar el valor de la propiedad title.

Estructura de un Componente

```
@Component({ // Decorador @Component
  selector: 'app-root', //Nombre del Componente para llamarlo en html
  standalone: true, //Indicador Standalone
  imports: [RouterOutlet, ExampleComponent], // Imports de Componentes Anidados
  templateUrl: './app.component.html', //Vista HTML
  styleUrl: './app.component.css' // Estilos CSS
//Declaracion de la Clase, se incluye el comportamiento del Componente
export class AppComponent {
 title = 'my-app';
```

Creación de un Componente

- Puedes crear un componente en Angular utilizando el CLI de Angular.
 Por ejemplo, para crear un componente llamado "example":
 - ng generate component example
- Esto generará los archivos necesarios para el componente
 (example.component.ts, example.component.html,
 example.component.css, y example.component.spec.ts) y actualizará el
 módulo Angular (app.module.ts) para incluir el nuevo componente.

Uso de un Componente

 Para usar un componente en otro componente o en una plantilla, simplemente incluyes su selector en el HTML. Por ejemplo, para usar el componente ExampleComponent en el componente raíz (app.component.html), agregarías:

<app-example></app-example>

Los componentes en Angular son bloques de construcción esenciales para desarrollar aplicaciones web modulares y mantenibles. Cada componente encapsula una parte de la interfaz de usuario y su lógica asociada, lo que facilita la reutilización y el mantenimiento del código.

Uso de un Componente

```
<h1>Esto forma parte del componente Principal</h1>
<!-- aqui se inyecta el componente "example" -->
<app-example></app-example>
```

Ciclo de Vida de un Componente

• El ciclo de vida de una aplicación Angular se refiere a la serie de eventos o métodos que Angular llama durante la vida de un componente o directiva. Estos eventos permiten a los desarrolladores engancharse en diferentes momentos del ciclo de vida para ejecutar código, manejar eventos, o inicializar valores.

constructor

ngOnChanges

ngOnInit

ngDoCheck

ngAfterContentInit

ngAfterContentChecked

ngAfterViewInit

ngAfterViewChecked

ngOnDestroy

Ciclo de Vida

- ngOnChanges(): Es llamado antes de ngOnInit(), si las propiedades "Input" del componente cambian.
- ngOnInit(): Es llamado después de ngOnchanges(), si el no hay propiedades "Input" también es llamado.
- ngDoCheck(): Es llamado inmediatamente después de ngOnChanges() en cada detección de cambios e inmediatamente después de ngOnInit() en la primera ejecución.

Ciclo de Vida

- ngAfterContentInit(): Es llamado una vez después del primer ngDoCheck().
- ngAfterContentChecked(): Es llamado después de ngAfterContentInit() y subsecuente a cada ngDoCheck().
- ngAfterViewInit(): Es llamado una vez después del primer ngAfterContentChecked().

Ciclo de Vida

- ngAfterViewChecked(): Es llamado después de ngAfterViewInit() y después de cada ngAfterContentChecked().
- ngOnDestroy(): Es llamado inmediatamente antes de que angular destruya la directiva o el componente.

Directivas

01

¿Qué son las directivas?

02

Tipos de directivas.

03

Creación y uso de directivas personalizadas.

Directivas

- Las plantillas de Angular son dinámicas, cuando Angular las renderiza, transforma el DOM de acuerdo con las instrucciones dadas por las directivas.
- Cada Directiva que usamos tiene un nombre, y determina donde puede ser usada, sea en un elemento, atributo, componente o clase.
- Las directivas añaden comportamiento a los elementos de una aplicación Angular.
 (Extienden la funcionalidad del HTML)
- Existen 3 tipos:
 - Directivas de atributo
 - Directivas estructurales
 - Componentes

Directivas de Atributo

Las directivas de atributos alteran la apariencia o el comportamiento de un elemento del DOM y son usados como atributos de los elementos.

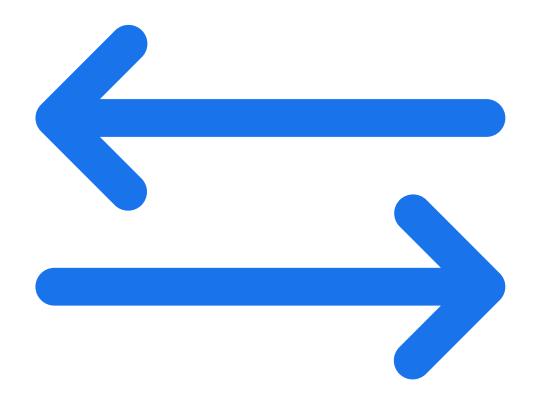
Ejemplo de directivas de atributo:

- ngModel: Implementa binding.
- ngClass: Permite añadir o eliminar varias clases.
- ngStyle: Permite asignar estilos inline

Directivas de Atributo

- Para crear una directiva de atributo, desde Angular CLI se utiliza el comando:
 - ng generate directive <nombre>
 - ng g d <nombre>
- Esto crea un archivo src/app/<nombre>.directive.ts y su correspondiente archivo de test.

Directivas
Estructurales
o de control
de Flujo



Directivas Estructurales o de Control de Flujo

Alteran la estructura del DOM, añadiendo, eliminando y manipulando los elementos a los que están unidos.

Ejemplo de directivas estructurales:

- *nglf: Nos permite incluir condicionales de lógica.
 - Evaluar sentencias.
 - Hacer comparaciones.
 - Mostrar/ocultar secciones de código.
- *ngFor: Directiva para iterar, permite ejecutar bucles.

Componente de Ejemplo

```
export class ExampleComponent {
  username = 'Fulano de Tal';
  isLoged = true;
  users = [
    { id: 0, name: 'Sarah' },
    { id: 1, name: 'Amy' },
    { id: 2, name: 'Rachel' },
    { id: 3, name: 'Jessica' },
     id: 4, name: 'Poornima' },
```

Directiva *nglf

```
<!-- Directiva Clasica de Angular 2 -->
<h1 *ngIf="isLoged; else other">Bienvenido {{username}}</h1>
<ng-template #other>
    <h1>Iniciar Sesion</h1>
</ng-template>
<!-- Nuevo Control de Flujo a partir de Angular 17 -->
@if (isLoged) {
    <h1>Bienvenido {{username}}</h1>
  @else {
    <h1>Iniciar Sesion</h1>
```

Directiva *ngFor

Pipes

- Los pipes sirven para realizar transformaciones de los datos, a la hora de mostrarlos en los templates de los componentes.
- Utilizar pipes para transformar cadenas, valores de monedas, fechas, y otros tipos de datos.
- Los pipes son simples funciones para usar en los templates que aceptan una entrada y regresan un valor transformado.
- Son muy útiles, ya que se pueden utilizar en toda una aplicación después de declararla una única vez.

Pipes

Angular tiene incorporados algunos pipes de uso habitual.

- DatePipe: Formatea una fecha de acuerdo con la regla local.
- **UpperCasePipe:** Transforma texto a todo Mayúsculas.
- LowerCasePipe: Transforma texto a todo Minúsculas.
- CurrencyPipe: Convierte un número a una moneda.
- **DecimalPipe:** Convierte un número a una cadena decimal.
- **PercentPipe:** Convierte un número a una cadena de porcentaje.

Es posible encadenar pipes para que la salida de una se convierta en la entrada de otra.

Pipes

```
<!-- Directiva Clasica de Angular 2 -->
{{user.name | lowercase }}
```