# Componenetes

En Angular, los **componentes** son una parte fundamental de la arquitectura de la aplicación. Son bloques de construcción que permiten estructurar y organizar la interfaz de usuario (UI) en partes reutilizables e independientes. A continuación, te explico los **componentes principales** de un componente en Angular:

**1. Decorador @Component**

El decorador @Component es una función que se utiliza para definir metadatos del componente. Estos metadatos incluyen:

* **selector:** El nombre del componente que se usará en las plantillas HTML (por ejemplo, <app-mi-componente>).
* **templateUrl:** La ruta al archivo HTML que define la vista del componente.
* **styleUrls:** Un array de rutas a archivos CSS que definen los estilos del componente.
* **template:** Permite definir la plantilla HTML directamente en el archivo TypeScript (en lugar de usar templateUrl).
* **styles:** Permite definir estilos CSS directamente en el archivo TypeScript (en lugar de usar styleUrls).

Ejemplo:

@Component({

selector: 'app-mi-componente',

templateUrl: './mi-componente.component.html',

styleUrls: ['./mi-componente.component.css']

})

**2. Clase del Componente**

La clase del componente es donde se define la lógica del componente. Esta clase se escribe en TypeScript y contiene:

* **Propiedades:** Variables que almacenan datos y pueden ser utilizadas en la plantilla HTML.
* **Métodos:** Funciones que definen el comportamiento del componente.
* **Ciclo de vida:** Métodos especiales como ngOnInit(), ngOnDestroy(), etc., que se ejecutan en diferentes etapas del ciclo de vida del componente.

Ejemplo:

export class MiComponente {

titulo: string = 'Hola, soy un componente';

constructor() { }

ngOnInit(): void {

console.log('El componente se ha inicializado');

}

}

**3. Plantilla HTML (Template)**

La plantilla HTML define la estructura de la interfaz de usuario del componente. Aquí se utiliza la sintaxis de Angular para:

* **Interpolación:** Mostrar valores de las propiedades del componente (por ejemplo, {{ titulo }}).
* **Directivas:** Como \*ngIf, \*ngFor, etc., para manipular el DOM dinámicamente.
* **Enlace de datos:** Usar [(ngModel)] para enlazar datos entre la vista y el componente.
* **Eventos:** Usar (evento) para manejar acciones del usuario, como clics o cambios.

Ejemplo:

<h1>{{ titulo }}</h1>

<button (click)="cambiarTitulo()">Cambiar título</button>

Run HTML

**4. Estilos CSS**

Los estilos CSS definen la apariencia del componente. Pueden ser:

* **Globales:** Aplicados a toda la aplicación.
* **Específicos del componente:** Definidos en el archivo CSS del componente y aplicados solo a ese componente (gracias al encapsulamiento de Angular).

Ejemplo:

h1 {

color: blue;

}

**5. Ciclo de Vida del Componente**

Angular proporciona métodos de ciclo de vida que permiten ejecutar código en diferentes momentos, como:

* **ngOnInit():** Se ejecuta cuando el componente se inicializa.
* **ngOnChanges():** Se ejecuta cuando cambian las propiedades de entrada (@Input).
* **ngOnDestroy():** Se ejecuta cuando el componente se destruye.

Ejemplo:

ngOnInit(): void {

console.log('Componente inicializado');

}

**6. Entradas y Salidas (@Input y @Output)**

* **@Input:** Permite pasar datos de un componente padre a un componente hijo.
* **@Output:** Permite emitir eventos desde un componente hijo a un componente padre.

Ejemplo:

@Input() dato: string;

@Output() evento = new EventEmitter<void>();

emitirEvento(): void {

this.evento.emit();

}

**7. Servicios e Inyección de Dependencias**

Los componentes pueden utilizar **servicios** para compartir lógica o datos entre diferentes partes de la aplicación. Los servicios se inyectan en el constructor del componente.

Ejemplo:

constructor(private miServicio: MiServicio) { }

ngOnInit(): void {

this.miServicio.obtenerDatos().subscribe(datos => {

console.log(datos);

});

}

**Resumen de los Componentes en Angular:**

1. **Decorador @Component:** Define los metadatos del componente.
2. **Clase del Componente:** Contiene la lógica y propiedades del componente.
3. **Plantilla HTML:** Define la estructura de la interfaz de usuario.
4. **Estilos CSS:** Define la apariencia del componente.
5. **Ciclo de Vida:** Métodos que se ejecutan en diferentes etapas del componente.
6. **Entradas y Salidas (@Input y @Output):** Permiten la comunicación entre componentes.
7. **Servicios:** Para compartir lógica o datos entre componentes.

Los componentes en Angular son **reutilizables**, **modulares** y **encapsulados**, lo que facilita la creación de aplicaciones escalables y mantenibles.

## Generar nuevos componente desde Consola

El comando ng generate component counter (o su forma abreviada ng g c counter) es una instrucción de la CLI (Command Line Interface) de Angular que se utiliza para **generar un nuevo componente** llamado counter. Este comando crea automáticamente todos los archivos necesarios para el componente y los configura en el módulo correspondiente.

**¿Qué hace este comando?**

1. **Crea una carpeta llamada counter:**  
   Dentro de la carpeta src/app, se crea una nueva carpeta con el nombre del componente (counter).
2. **Genera los archivos del componente:**
   * counter.component.ts: Archivo TypeScript que contiene la lógica del componente.
   * counter.component.html: Archivo HTML que define la plantilla del componente.
   * counter.component.css: Archivo CSS para los estilos del componente.
   * counter.component.spec.ts: Archivo de pruebas unitarias (opcional).
3. **Registra el componente en el módulo:**  
   El componente se declara automáticamente en el módulo más cercano (por defecto, app.module.ts), lo que permite usarlo en la aplicación.

**Opciones adicionales del comando:**

Puedes personalizar la generación del componente con opciones como:

* **--skip-tests:** No genera el archivo de pruebas (spec.ts).  
  Ejemplo: ng g c counter --skip-tests
* **--inline-template:** Define la plantilla HTML directamente en el archivo TypeScript.  
  Ejemplo: ng g c counter --inline-template
* **--inline-style:** Define los estilos CSS directamente en el archivo TypeScript.  
  Ejemplo: ng g c counter --inline-style
* **--module:** Especifica en qué módulo se debe registrar el componente.  
  Ejemplo: ng g c counter --module=mi-modulo

**Resumen:**

El comando ng generate component counter es una herramienta poderosa de la CLI de Angular que te permite crear rápidamente un nuevo componente con toda su estructura básica. Esto facilita el desarrollo y mantiene la organización del código.

## SessiomStorage y LocalStorage

El **localStorage** y el **sessionStorage** son dos mecanismos de almacenamiento en el navegador que permiten guardar datos en el lado del cliente (es decir, en el navegador del usuario). Aunque ambos son muy similares en su uso, tienen diferencias clave en cuanto a la **persistencia de los datos** y el **ámbito de almacenamiento**. Aquí te explico las diferencias:

**1. Persistencia de los datos**

* **localStorage:**
  + Los datos almacenados en localStorage **persisten incluso después de cerrar el navegador**.
  + Los datos no tienen fecha de expiración y permanecen en el navegador hasta que se eliminan manualmente (por el usuario o mediante código).
  + Ejemplo: Ideal para guardar preferencias del usuario, como el tema de la aplicación (claro/oscuro).
* **sessionStorage:**
  + Los datos almacenados en sessionStorage **solo persisten durante la sesión actual del navegador**.
  + Los datos se eliminan automáticamente cuando el usuario cierra la pestaña o el navegador.
  + Ejemplo: Ideal para almacenar datos temporales, como el estado de un formulario durante una sesión.

**2. Ámbito de almacenamiento**

* **localStorage:**
  + Los datos están disponibles en **todas las pestañas y ventanas del mismo origen** (mismo protocolo, dominio y puerto).
  + Si abres una nueva pestaña o ventana, los datos de localStorage serán accesibles en todas ellas.
* **sessionStorage:**
  + Los datos están **limitados a la pestaña o ventana específica** donde se almacenaron.
  + Si abres una nueva pestaña o ventana, tendrá su propio sessionStorage independiente.

**3. Capacidad de almacenamiento**

* Ambos (localStorage y sessionStorage) tienen una capacidad de almacenamiento de **alrededor de 5 MB por origen** (dependiendo del navegador).
* Esta capacidad es compartida entre localStorage, sessionStorage y cookies en algunos navegadores.

**4. Uso en código**

Ambos se utilizan de manera similar, a través de los métodos:

* **setItem(key, value):** Almacena un valor bajo una clave específica.
* **getItem(key):** Obtiene el valor asociado a una clave.
* **removeItem(key):** Elimina un valor específico.
* **clear():** Elimina todos los datos almacenados.
* **key(index):** Obtiene la clave en una posición específica.
* **length:** Devuelve el número de elementos almacenados.

Ejemplo:

// Almacenar datos

localStorage.setItem('nombre', 'Juan');

sessionStorage.setItem('token', 'abc123');

// Obtener datos

const nombre = localStorage.getItem('nombre'); // "Juan"

const token = sessionStorage.getItem('token'); // "abc123"

// Eliminar datos

localStorage.removeItem('nombre');

sessionStorage.clear();

**5. Casos de uso comunes**

* **localStorage:**
  + Guardar preferencias del usuario (tema, idioma, etc.).
  + Almacenar datos que deben persistir entre sesiones, como el carrito de compras en un e-commerce.
  + Guardar tokens de autenticación (aunque no es recomendable por razones de seguridad).
* **sessionStorage:**
  + Almacenar datos temporales, como el estado de un formulario mientras el usuario navega.
  + Guardar información sensible que no debe persistir después de cerrar la pestaña.
  + Mantener datos específicos de una sesión, como el ID de sesión.

**Resumen de diferencias:**

| **Característica** | **localStorage** | **sessionStorage** |
| --- | --- | --- |
| **Persistencia** | Los datos persisten después de cerrar el navegador. | Los datos se eliminan al cerrar la pestaña o el navegador. |
| **Ámbito** | Disponible en todas las pestañas y ventanas del mismo origen. | Limitado a la pestaña o ventana específica. |
| **Capacidad** | ~5 MB por origen. | ~5 MB por origen. |
| **Casos de uso** | Preferencias del usuario, datos persistentes. | Datos temporales, información de sesión. |

**¿Cuándo usar cada uno?**

* Usa **localStorage** cuando necesites que los datos persistan entre sesiones.
* Usa **sessionStorage** cuando los datos solo sean relevantes durante la sesión actual.

Ambos son herramientas útiles, pero es importante elegir el adecuado según el caso de uso para garantizar un manejo eficiente y seguro de los datos.