

# Angular

## *Componentes*

CertiDevs

#  ndice de contenidos

|   |    |
|---|----|
| 1. Componentes . . . . .  | 1  |
| 1.1. Creaci n y estructura de un componente . . . . .                                 | 1  |
| 1.2. Selector . . . . .   | 1  |
| 1.3. Plantilla . . . . .  | 1  |
| 1.4. Estilos . . . . .  | 2  |
| 1.5. Enlace de datos (Data Binding) . . . . .   | 2  |
| 1.6. Ciclo de vida de un componente . . . . .   | 3  |
| 1.7. Comunicaci n entre componentes . . . . .   | 3  |
| 2. Ejemplo 1: Crear un componente b sico . . . . .                                    | 4  |
| 2.1. Paso 1: Crear un nuevo proyecto Angular . . . . .                                | 4  |
| 2.2. Paso 2: Generar un nuevo componente . . . . .                                    | 4  |
| 2.3. Paso 3: Editar el archivo TypeScript del componente . . . . .                    | 4  |
| 2.4. Paso 4: Editar la plantilla del componente . . . . .                             | 5  |
| 2.5. Paso 5: Usar el componente en la aplicaci n . . . . .                            | 5  |
| 2.6. Paso 6: Ejecutar la aplicaci n . . . . .   | 5  |
| 3. Ejemplo 2: Contador . . . . .  | 5  |
| 3.1. Paso 1: Generar el componente de Contador . . . . .                              | 6  |
| 3.2. Paso 2: Editar el archivo TypeScript del componente de Contador . . . . .        | 6  |
| 3.3. Paso 3: Editar la plantilla del componente de Contador . . . . .                 | 6  |
| 3.4. Paso 4: Usar el componente de Contador en la aplicaci n . . . . .                | 7  |
| 3.5. Paso 5: Ejecutar la aplicaci n . . . . .   | 7  |
| 4. Ejemplo 3: Lista de tareas . . . . .   | 7  |
| 4.1. Paso 1: Generar el componente de Lista de Tareas . . . . .                       | 7  |
| 4.2. Paso 2: Editar el archivo TypeScript del componente de Lista de Tareas . . . . . | 8  |
| 4.3. Paso 3: Editar la plantilla del componente de Lista de Tareas . . . . .          | 8  |
| 4.4. Paso 4: Importar FormsModule en AppModule . . . . .                              | 9  |
| 4.5. Paso 5: Usar el componente de Lista de Tareas en la aplicaci n . . . . .         | 9  |
| 4.6. Paso 6: Estilos . . . . .  | 10 |
| 4.7. Paso 7: Ejecutar la aplicaci n . . . . .   | 10 |
| 5. Otros ejemplos . . . . .   | 10 |

# 1. Componentes

Los componentes son la unidad b sica de construcci n de la interfaz de usuario en una aplicaci n Angular.

Cada componente consta de una plantilla HTML, una clase TypeScript y, opcionalmente, un archivo de estilos CSS.

Los componentes son responsables de controlar una parte de la pantalla (vista) y de manejar la interacci n del usuario.

Los componentes se crean utilizando la clase `@Component`.

## 1.1. Creaci n y estructura de un componente

Un componente en Angular se crea utilizando la clase `@Component`. Esta clase es un decorador que acepta un objeto de metadatos como argumento.

El objeto de metadatos define las propiedades clave del componente, como el selector, la plantilla, los estilos y las animaciones.

A continuaci n se muestra un ejemplo b sico de un componente en Angular:

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

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Mi Aplicaci n Angular';
}
```

## 1.2. Selector

El `selector` es una propiedad en el objeto de metadatos que define c mo se utilizar  el componente en una plantilla HTML.

El selector puede ser un nombre de elemento, un nombre de atributo o una clase CSS. Cuando se utiliza el selector en una plantilla HTML, Angular crea una instancia del componente y lo renderiza en el lugar donde se encuentra el selector.

## 1.3. Plantilla

La plantilla es una propiedad en el objeto de metadatos que define la estructura HTML del componente.

La plantilla puede ser definida directamente en el objeto de metadatos utilizando la propiedad `template` o referenciando un archivo HTML externo utilizando la propiedad `templateUrl`.

La plantilla incluye enlaces de datos (data binding) y directivas para controlar el contenido y la estructura de la vista.

## 1.4. Estilos

Los estilos son una propiedad en el objeto de metadatos que define los estilos CSS para el componente.

Los estilos pueden ser definidos directamente en el objeto de metadatos utilizando la propiedad `styles` o referenciando un archivo CSS externo utilizando la propiedad `styleUrls`.

Los estilos de un componente se aplican únicamente a la vista del componente y no afectan a otros componentes o al DOM global.

## 1.5. Enlace de datos (Data Binding)

El enlace de datos o data binding es una característica clave en Angular que permite vincular las propiedades y eventos de los componentes con el DOM.

El enlace de datos facilita la actualización automática de la vista cuando cambian las propiedades del componente y la actualización de las propiedades del componente cuando ocurren eventos en la vista.

Hay varios tipos de enlaces de datos en Angular:

- ¥ Interpolación: Utiliza llaves dobles `{{ }}` para mostrar el valor de una propiedad en la plantilla.
- ¥ Enlace de propiedad: Utiliza corchetes `[]` para asignar el valor de una propiedad del componente a un atributo del DOM.
- ¥ Enlace de evento: Utiliza paréntesis `()` para vincular un evento del DOM a un método del componente.
- ¥ Enlace bidireccional: Utiliza la combinación `[]()` para vincular una propiedad del componente y un evento del DOM, permitiendo una comunicación bidireccional entre el componente y la vista.

El enlace de datos es una técnica utilizada en Angular para mantener la sincronización entre la vista (la representación visual en el navegador) y el modelo (los datos en typescript del componente) de una aplicación.

Permite que los cambios en los datos del modelo se reflejen automáticamente en la vista y viceversa.

## 1.6. Ciclo de vida de un componente

Los componentes en Angular tienen un ciclo de vida que comienza con la creaci3n del componente y termina con su destrucci3n.

Durante el ciclo de vida de un componente, Angular invoca varios m3todos del componente en momentos espec3ficos, conocidos como m3todos del ciclo de vida.

Estos m3todos permiten a los desarrolladores realizar acciones en diferentes etapas del ciclo de vida del componente. Los m3todos del ciclo de vida m3s comunes incluyen:

- ¥ **ngOnInit**: Se ejecuta despu3s de que Angular inicializa las propiedades vinculadas a datos y se utiliza com3nmente para realizar tareas de inicializaci3n y recuperaci3n de datos.
- ¥ **ngOnChanges**: Se ejecuta cuando Angular detecta cambios en las propiedades vinculadas a datos. Es 3til para reaccionar a los cambios en las propiedades de entrada (**@Input**) del componente.
- ¥ **ngDoCheck**: Se ejecuta cuando Angular ejecuta la detecci3n de cambios para el componente. Se puede utilizar para implementar verificaciones de cambio personalizadas.
- ¥ **ngAfterContentInit**: Se ejecuta despu3s de que Angular proyecta el contenido externo en la vista del componente. Es 3til para realizar acciones basadas en el contenido proyectado.
- ¥ **ngAfterContentChecked**: Se ejecuta despu3s de que Angular realiza la detecci3n de cambios en el contenido proyectado.
- ¥ **ngAfterViewInit**: Se ejecuta despu3s de que Angular inicializa las vistas y las plantillas de componentes hijos. Es 3til para realizar acciones que dependen de las vistas de los componentes hijos.
- ¥ **ngAfterViewChecked**: Se ejecuta despu3s de que Angular realiza la detecci3n de cambios en las vistas y las plantillas de componentes hijos.
- ¥ **ngOnDestroy**: Se ejecuta justo antes de que Angular destruya el componente. Se utiliza para realizar la limpieza y liberar recursos.

## 1.7. Comunicaci3n entre componentes

La comunicaci3n entre componentes es esencial en las aplicaciones Angular para mantener la coherencia y la actualizaci3n de datos entre diferentes partes de la aplicaci3n.

Angular proporciona varias formas de comunicaci3n entre componentes, como:

- ¥ Propiedades de entrada (**@Input**): Permiten que los componentes padres pasen datos a los componentes hijos.
- ¥ Propiedades de salida (**@Output**): Permiten que los componentes hijos emitan eventos a los componentes padres, generalmente utilizando la clase **EventEmitter**.
- ¥ Servicios: Facilitan la comunicaci3n entre componentes no relacionados jer3rquicamente, compartiendo datos y l3gica a trav3s de la inyecci3n de dependencias.
- ¥ Proyecci3n de contenido: Utiliza la etiqueta **<ng-content>** para insertar contenido din3mico en un componente desde su componente padre.

En resumen, los componentes en Angular son la base de la interfaz de usuario y desempeñan un papel crucial en la estructura y funcionalidad de las aplicaciones Angular.

Los componentes permiten crear vistas reutilizables, modularizar el código y facilitar la interacción del usuario con la aplicación.

Conocer cómo crear y utilizar componentes es fundamental para desarrollar aplicaciones Angular efectivas y escalables.

## 2. Ejemplo 1: Crear un componente básico

Para crear un componente desde cero, sigue estos pasos:

### 2.1. Paso 1: Crear un nuevo proyecto Angular

Si aún no tienes un proyecto Angular, crea uno ejecutando el siguiente comando en la línea de comandos:

```
ng new my-app
```

Reemplaza `my-app` con el nombre que prefieras para tu proyecto. A continuación, navega al directorio del proyecto con `cd my-app`.

### 2.2. Paso 2: Generar un nuevo componente

Utiliza Angular CLI para generar un nuevo componente ejecutando el siguiente comando:

```
ng generate component my-component
```

Reemplaza `my-component` con el nombre que prefieras para tu componente. Angular CLI creará los archivos necesarios y los importará automáticamente en el módulo principal.

### 2.3. Paso 3: Editar el archivo TypeScript del componente

Encuentra el archivo TypeScript generado automáticamente para tu componente (por ejemplo, `src/app/my-component/my-component.component.ts`) y reemplaza su contenido con lo siguiente:

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

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

```
})  
export class MyComponentComponent {  
  title = 'Mi primer componente en Angular';  
}
```

Aquí, estamos importando `Component` desde el paquete `@angular/core` y utilizando el decorador `@Component` para definir los metadatos del componente.

También estamos creando una propiedad llamada `title` en la clase `MyComponentComponent`.

## 2.4. Paso 4: Editar la plantilla del componente

Abre el archivo HTML de la plantilla del componente (por ejemplo, `src/app/my-component/my-component.component.html`) y reemplaza su contenido con lo siguiente:

```
<h1>{{ title }}</h1>  
<p>Este es mi primer componente en Angular.</p>
```

Aquí, estamos utilizando la interpolación de Angular `{{ title }}` para mostrar el valor de la propiedad `title` en la plantilla.

## 2.5. Paso 5: Usar el componente en la aplicación

Abre el archivo `src/app/app.component.html` y agrega el selector del componente que definimos en el paso 3 (en este caso, `app-my-component`):

```
<app-my-component></app-my-component>
```

Ahora, el componente `MyComponentComponent` deberá mostrarse en la página principal de la aplicación.

## 2.6. Paso 6: Ejecutar la aplicación

Finalmente, ejecuta la aplicación Angular utilizando el siguiente comando:

```
ng serve
```

Abre tu navegador web y navega a <http://localhost:4200> para ver tu componente en acción.

# 3. Ejemplo 2: Contador

En este ejemplo, crearemos un componente de contador que permita incrementar, decrementar y reiniciar un contador.

## 3.1. Paso 1: Generar el componente de Contador

En un nuevo proyecto utiliza Angular CLI para generar un nuevo componente llamado counter:

```
ng generate component counter
```

## 3.2. Paso 2: Editar el archivo TypeScript del componente de Contador

Abre el archivo TypeScript del componente de Contador (por ejemplo, `src/app/counter/counter.component.ts`) y reemplaza su contenido con lo siguiente:

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

@Component({
  selector: 'app-counter',
  templateUrl: './counter.component.html',
  styleUrls: ['./counter.component.css']
})
export class CounterComponent {
  count = 0;

  increment(): void {
    this.count++;
  }

  decrement(): void {
    this.count--;
  }

  reset(): void {
    this.count = 0;
  }
}
```

En este código, estamos importando `Component` desde `@angular/core` y creando un componente que maneja un contador simple con métodos para incrementar, decrementar y reiniciar el valor del contador.

## 3.3. Paso 3: Editar la plantilla del componente de Contador

Abre el archivo HTML de la plantilla del componente de Contador (por ejemplo, `src/app/counter/counter.component.html`) y reemplaza su contenido con lo siguiente:

```
<h1>Contador</h1>
```



```
<p>Valor actual: {{ count }}</p>
<button (click)="increment()">Incrementar</button>
<button (click)="decrement()">Decrementar</button>
<button (click)="reset()">Reiniciar</button>
```

En esta plantilla, estamos utilizando la interpolación `{{ count }}` para mostrar el valor actual del contador y las directivas `(click)` para manejar eventos de clic en los botones de incrementar, decrementar y reiniciar.

### 3.4. Paso 4: Usar el componente de Contador en la aplicación

Abre el archivo `src/app/app.component.html` y agrega el selector del componente de Contador (en este caso, `app-counter`):

```
<app-counter></app-counter>
```

Ahora, el componente de Contador deber'a mostrarse en la página principal de la aplicación.

### 3.5. Paso 5: Ejecutar la aplicación

Finalmente, ejecuta la aplicación Angular utilizando el siguiente comando:

```
ng serve
```

Abre tu navegador web y navega a <http://localhost:4200> para ver el componente de Contador en acción.

## 4. Ejemplo 3: Lista de tareas

En este ejemplo, crearemos un componente de lista de tareas que permita agregar y eliminar tareas.

### 4.1. Paso 1: Generar el componente de Lista de Tareas

Utiliza Angular CLI para generar un nuevo componente llamado `task-list`:

```
ng generate component task-list
```

## 4.2. Paso 2: Editar el archivo TypeScript del componente de Lista de Tareas

Abre el archivo TypeScript del componente de Lista de Tareas (por ejemplo, `src/app/task-list/task-list.component.ts`) y reemplaza su contenido con lo siguiente:

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

interface Task {
  title: string;
  done: boolean;
}

@Component({
  selector: 'app-task-list',
  templateUrl: './task-list.component.html',
  styleUrls: ['./task-list.component.css']
})
export class TaskListComponent {
  tasks: Task[] = [
    { title: 'Tarea 1', done: false },
    { title: 'Tarea 2', done: true },
  ];

  newTaskTitle = '';

  addTask(): void {
    if (this.newTaskTitle.trim()) {
      this.tasks.push({ title: this.newTaskTitle, done: false });
      this.newTaskTitle = '';
    }
  }

  removeTask(index: number): void {
    this.tasks.splice(index, 1);
  }
}
```

En este código, estamos importando `Component` desde `@angular/core` y utilizando la interfaz `Task` para definir la estructura de los objetos de tareas.

Luego, creamos un componente que maneja una lista de tareas y permite agregar y eliminar tareas.

## 4.3. Paso 3: Editar la plantilla del componente de Lista de Tareas

Abre el archivo HTML de la plantilla del componente de Lista de Tareas (por ejemplo, `src/app/task-`

`list/task-list.component.html`) y reemplaza su contenido con lo siguiente:

```
<h1>Lista de Tareas</h1>

<ul>
  <li *ngFor="let task of tasks; index as i">
    <input type="checkbox" [(ngModel)]="task.done" />
    {{ task.title }}
    <button (click)="removeTask(i)">Eliminar</button>
  </li>
</ul>

<h2>Agregar Tarea</h2>

<div>
  <label>Título de la tarea:</label>
  <input type="text" [(ngModel)]="newTaskTitle" />
  <button (click)="addTask()">Agregar</button>
</div>
```

En esta plantilla, estamos utilizando la directiva `*ngFor` para mostrar la lista de tareas y las directivas `(click)` y `[(ngModel)]` para manejar eventos de clic y vincular datos bidireccionalmente.

## 4.4. Paso 4: Importar FormsModule en AppModule

Para utilizar `[(ngModel)]`, debemos importar `FormsModule` en nuestro `AppModule`. Abre el archivo `src/app/app.module.ts`, importa `FormsModule` y agrégalo a la lista de imports del `@NgModule`:

```
import { FormsModule } from '@angular/forms';

@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
  ],
  // ...
})
```

## 4.5. Paso 5: Usar el componente de Lista de Tareas en la aplicación

Abre el archivo `src/app/app.component.html` y agrega el selector del componente de Lista de Tareas (en este caso, `app-task-list`):

```
<app-task-list></app-task-list>
```

Ahora, el componente de Lista de Tareas deber' a mostrarse en la p'gina principal de la aplicaci' n.

## 4.6. Paso 6: Estilos

Opcionalmente, puedes agregar algunos estilos al archivo `src/app/task-list/task-list.component.css` para mejorar la apariencia del componente de Lista de Tareas:

```
ul {  
  list-style-type: none;  
  padding: 0;  
}  
  
li {  
  display: flex;  
  align-items: center;  
  margin-bottom: 0.5rem;  
}  
  
input[type="checkbox"] {  
  margin-right: 0.5rem;  
}  
  
button {  
  margin-left: 1rem;  
}
```

## 4.7. Paso 7: Ejecutar la aplicaci' n

Finalmente, ejecuta la aplicaci' n Angular utilizando el siguiente comando:

```
ng serve
```

Abre tu navegador web y navega a <http://localhost:4200> para ver el componente de Lista de Tareas en acci' n.

## 5. Otros ejemplos

<https://angular.io/guide/what-is-angular>

Ver otros ejemplos:

¥ <https://angular.io/generated/live-examples/what-is-angular/stackblitz.html>

¥ <https://angular.io/generated/live-examples/getting-started-v0/stackblitz.html>

¥ <https://angular.io/generated/live-examples/component-overview/stackblitz.html>