Temas de angular

Johan Silva Alvares
55222030

Ing.Johan Mauricio Fonseca

Taller de aplicaciones móviles
2025

Temas de Angular necesarios para el taller

Introducción

Angular es un framework moderno de desarrollo web creado y mantenido por Google. Está basado en TypeScript y se centra en una arquitectura por componentes que permite crear aplicaciones dinámicas, modulares y escalables (Angular, s.f.-a). La lógica, la vista y los estilos se agrupan en unidades llamadas *componentes*, que constituyen los bloques básicos de toda aplicación.

Componentes básicos

¿Qué es un componente y su estructura?

Un **componente** en Angular es una clase de TypeScript que define la lógica y se asocia a un template (HTML) y estilos (CSS/SCSS). Se declara con el decorador @Component, el cual incluye propiedades como:

- **selector**: el nombre de la etiqueta personalizada que representa al componente en HTML.
- **template**: el código HTML que define la vista.
- **style/styleUrls**: estilos que afectan al componente.

Por ejemplo:

```
@Component({
  selector: 'app-product-card',
  templateUrl: './product-card.component.html',
  styleUrls: ['./product-card.component.css']
})
export class ProductCardComponent { }
```

Como señalan los documentos oficiales de Angular (s.f.-a), esta estructura es la base para la construcción de interfaces.

Diferencia entre componente de página y componente reutilizable

Los **componentes de página** representan una vista completa dentro de la aplicación, usualmente asociada a rutas, como *Home* o *Login*. En contraste, los **componentes reutilizables** son piezas pequeñas que pueden insertarse en diferentes vistas, como botones personalizados o tarjetas de producto (Angular, s.f.-a).

Standalone Components (Angular 15+)

Sin NgModules

Tradicionalmente, cada componente debía declararse dentro de un NgModule. Sin embargo, a partir de Angular 15, los desarrolladores pueden crear **componentes standalone**, es decir, sin necesidad de módulos (Angular, s.f.-b).

Propiedad standalone: true

La clave es la propiedad standalone: true dentro del decorador @Component. Esto convierte al componente en independiente:

```
@Component({
  selector: 'app-hello',
  standalone: true,
  template: `<h1>Hola Angular</h1>`
})
export class HelloComponent {}
```

Esto simplifica la arquitectura y acelera el desarrollo (Angular University, 2025).

Importaciones en componentes standalone

Los componentes standalone pueden importar otros componentes, directivas o módulos usando la propiedad imports. Esto hace posible incluir directivas como *ngFor o *ngIf desde CommonModule (Angular, s.f.-b).

Ciclo de vida de los componentes

Angular proporciona varios **hooks de ciclo de vida** que permiten ejecutar lógica en distintos momentos:

- ngOnInit(): se ejecuta una vez al inicializar el componente.
- ngOnChanges(): responde a cambios en propiedades marcadas con @Input.
- ngOnDestroy(): se usa para limpiar recursos antes de destruir el componente (Angular, s.f.-c).

Estos métodos permiten controlar con precisión cómo y cuándo se ejecuta la lógica interna de cada componente.

Data Binding

El *data binding* conecta la lógica de los componentes con la interfaz gráfica. Según Angular (s.f.-a), existen varias formas:

- **Interpolación** ({{ }}): inserta valores de variables en el HTML.
- **Property Binding** ([prop]="value"): enlaza propiedades de elementos del DOM.
- **Event Binding** ((event)="handler()"): escucha eventos del usuario, como clics o entradas de texto.
- **Two-way Binding** ([(ngModel)]): combina property y event binding para sincronización bidireccional.

Directivas estructurales

Angular dispone de **directivas estructurales** que manipulan el DOM:

- *ngIf: muestra u oculta un elemento según una condición.
- *ngFor: repite un elemento para cada valor de una lista.

Estas directivas forman parte del módulo común (CommonModule) y son ampliamente utilizadas en plantillas dinámicas (Angular, s.f.-a).

Decoradores @Input y @Output

La comunicación entre componentes se logra mediante decoradores:

- **@Input()**: permite que un componente hijo reciba datos de su padre.
- **@Output()**: permite que el hijo emita eventos al padre, generalmente usando EventEmitter (Angular, s.f.-d).

Ejemplo:

```
@Input() producto!: string;
```

@Output() comprado = new EventEmitter<string>();

Esto posibilita construir aplicaciones reactivas y modulares.

Conclusión

Angular es un framework robusto que combina arquitectura por componentes, *data binding* flexible y directivas poderosas para construir aplicaciones modernas. La introducción de standalone components en Angular 15 marca una evolución significativa al simplificar la estructura y eliminar la necesidad de NgModules. Además, los decoradores @Input y @Output aseguran una comunicación clara entre componentes, mientras que las directivas estructurales permiten gestionar dinámicamente el contenido de las vistas.

Referencias

Angular. (s.f.-a). *Component overview*. Angular. Recuperado el 16 de septiembre de 2025, de https://angular.io/guide/component-overview

Angular. (s.f.-b). *Standalone components*. Angular. Recuperado el 16 de septiembre de 2025, de https://angular.io/guide/standalone-components

Angular. (s.f.-c). *Component lifecycle*. Angular. Recuperado el 16 de septiembre de 2025, de https://angular.dev/guide/components/lifecycle

Angular. (s.f.-d). *Accepting data with input properties*. Angular. Recuperado el 16 de septiembre de 2025, de https://angular.dev/guide/components/inputs

Angular. (s.f.-e). *Anatomy of components*. Angular. Recuperado el 16 de septiembre de 2025, de https://angular.dev/guide/components

Angular University. (2025, marzo). *Angular standalone components: Complete guide*. Angular University. Recuperado el 16 de septiembre de 2025, de https://blog.angular-university.io/angular-standalone-components/