

# Unidad 4: Angular avanzado

## Librerías visuales

### Introducción a Angular Material

Angular Material es una biblioteca de componentes para Angular basada en los principios de diseño de **Material Design**, un sistema desarrollado por Google que busca unificar diseño visual y usabilidad en aplicaciones web y móviles. Su propósito principal es proporcionar herramientas que permitan crear interfaces modernas, consistentes y accesibles de forma rápida y eficiente.

#### ¿Por qué usar Angular Material?

1. **Componentes preconstruidos:** Ofrece una amplia gama de elementos como botones, tarjetas, cuadros de diálogo, menús, barras de navegación y más.
2. **Consistencia visual:** Garantiza que la interfaz de usuario mantenga un diseño profesional y uniforme, siguiendo las pautas de Material Design.
3. **Accesibilidad integrada:** Todos los componentes están optimizados para cumplir con las mejores prácticas de accesibilidad.
4. **Compatibilidad multiplataforma:** Las aplicaciones funcionan correctamente en dispositivos móviles y de escritorio.
5. **Optimización del tiempo de desarrollo:** Facilita el proceso de diseño y reduce la necesidad de desarrollar componentes desde cero.

#### Ejemplo:

Un botón en Angular Material puede incluir funcionalidades como efectos de sombreado o resaltado al interactuar con el usuario:

```
<button mat-raised-button color="primary">Click aquí</button>
```

Angular Material es una elección ideal para desarrolladores que buscan una solución ágil para el diseño de aplicaciones modernas.

# Ejemplos de componentes

Angular Material incluye numerosos componentes que pueden implementarse fácilmente en cualquier proyecto Angular. A continuación, se explican tres ejemplos populares:

## Botones

Los botones permiten a los usuarios interactuar con la aplicación a través de acciones como enviar formularios o navegar.

### Tipos de botones:

- Básico
- Raised (elevado)
- Stroked (contorno)
- Flat (plano)

### Características principales:

- Personalización de estilos (elevados, planos, iconos).
- Admite efectos visuales como sombras o cambios de color al pasar el cursor.

### Ejemplo de uso básico:

```
<button mat-raised-button color="accent">Guardar</button>
```

## Barras de herramientas

Las barras de herramientas sirven para agrupar elementos como menús, títulos o botones.

### Ejemplo de uso básico:

```
<mat-toolbar color="primary">
  <span>Mi Aplicación</span>
  <button mat-button>Inicio</button>
</mat-toolbar>
```

## Cuadrículas (Grid)

El sistema de cuadrículas organiza contenido en un diseño de columnas y filas.

### Ejemplo de uso básico:

```
<mat-grid-list cols="2" rowHeight="100px">
  <mat-grid-tile>Item 1</mat-grid-tile>
  <mat-grid-tile>Item 2</mat-grid-tile>
```

```
</mat-grid-list>
```

## Cards

Las Material Cards permiten mostrar datos con una estructura de cabecera, cuerpo y pie.

### Importación:

```
import { MatCardModule } from '@angular/material/card';
```

### Uso en la vista:

```
<mat-card>

  <mat-card-header>
    <mat-card-title>Título de la Tarjeta</mat-card-title>
    <mat-card-subtitle>Subtítulo</mat-card-subtitle>
  </mat-card-header>
  <mat-card-content>
    <p>Este es el contenido principal de la tarjeta.</p>
  </mat-card-content>
  <mat-card-actions>
    <button mat-button>Acción 1</button>
    <button mat-button>Acción 2</button>
  </mat-card-actions>
</mat-card>
```

## Íconos

Permiten usar íconos SVG o personalizados en la aplicación.

### Importación:

```
import { MatIconModule } from '@angular/material/icon';
```

### Uso en la vista:

```
<mat-icon>home</mat-icon>
```

## Formularios

Facilitan la creación de formularios con campos de entrada de texto, listas desplegables, áreas de texto, entre otros.

### Importación:

```
import { MatFormFieldModule } from '@angular/material/form-field';
import { MatInputModule } from '@angular/material/input';
import { MatSelectModule } from '@angular/material/select';
```

### Uso en la vista:

```
<mat-form-field>
  <mat-label>Nombre</mat-label>
  <input matInput placeholder="Escribe tu nombre">
</mat-form-field>
```

## Diálogos

Los cuadros de diálogo permiten mostrar mensajes emergentes o formularios.

### Uso en la vista:

```
<button mat-button (click)="abrirDialogo()">Abrir diálogo</button>
```

### Componente de diálogo:

```
import { MatDialog } from '@angular/material/dialog';

export class ComponentePadre {
  constructor(public dialog: MatDialog) {}

  abrirDialogo(): void {
    this.dialog.open(DialogEjemploComponent);
  }
}
```

## Tablas

Permiten mostrar listas de datos con soporte para paginación, ordenamiento y filtrado.

### Importación:

```
import { MatTableModule } from '@angular/material/table';
```

## Uso en la vista:

```
<table mat-table [dataSource]="dataSource">
  <!-- Definición de columnas -->
  <ng-container matColumnDef="nombre">
    <th mat-header-cell *matHeaderCellDef> Nombre </th>
    <td mat-cell *matCellDef="let element"> {{element.nombre}} </td>
  </ng-container>

  <!-- Fila -->
  <tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
  <tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
</table>
```

## Configuración de temas y personalización de estilos

Permite personalizar la apariencia de la aplicación mediante la configuración de temas.

**Acción:** Configurar el archivo `theme.scss` para definir la paleta de colores.

### Ejemplo:

```
@import '~@angular/material/theming';
$mi-paleta: (
  primary: #2196F3,
  accent: #FFC107,
  warn: #F44336
);
```

## Configuración de breakpoints (Responsive Design)

Permite adaptar la visualización de la aplicación a diferentes tamaños de pantalla.

**Acción:** Usar `mat-grid-list` para crear una cuadrícula responsiva.

### Ejemplo:

```
<mat-grid-list cols="2" rowHeight="100px">
  <mat-grid-tile>1</mat-grid-tile>
  <mat-grid-tile>2</mat-grid-tile>
</mat-grid-list>
```

# Pipes en Angular

## Uso de Pipes integrados

Los **pipes** en Angular son herramientas que transforman datos en las vistas, facilitando su formato o manipulación sin necesidad de modificar el modelo.

### Pipes más comunes

#### 1. Uppercase y Lowercase

- Transforman cadenas de texto a mayúsculas o minúsculas.

```
<p>{{ 'Texto Ejemplo' | uppercase }}</p>
<!-- Salida: TEXTO EJEMPLO -->
```

#### 2. Date

- Formatea fechas de acuerdo con un patrón especificado.

```
<p>{{ today | date:'fullDate' }}</p>
<!-- Salida: Tuesday, December 5, 2024 -->
```

#### 3. Json

- Convierte objetos en formato JSON para visualizarlos en la vista.

```
<pre>{{ objeto | json }}</pre>
```

# Directivas personalizadas

## ¿Qué son las directivas personalizadas?

En Angular, las **directivas personalizadas** son una de las herramientas más potentes para personalizar el comportamiento de los elementos HTML y manipular el DOM de manera eficiente. Se utilizan para encapsular lógica específica que se puede reutilizar en varios componentes, permitiendo que el código sea más modular y fácil de mantener.

Una directiva personalizada puede alterar aspectos visuales, como el estilo de un elemento, o añadir comportamientos dinámicos, como manejar eventos del usuario o interactuar con el DOM. Esto las convierte en una solución ideal para abordar requerimientos únicos de un proyecto.

## Clasificación de Directivas en Angular

Antes de profundizar en las personalizadas, es importante entender que Angular divide las directivas en tres categorías:

### 1. Directivas de Atributo:

Estas alteran la apariencia o el comportamiento de un elemento, atributo o componente existente. Ejemplo: `ngClass` o `ngStyle`.

### 2. Directivas Estructurales:

Modifican la estructura del DOM añadiendo, eliminando o manipulando elementos.

Ejemplo: `*ngIf`, `*ngFor`.

### 3. Directivas Personalizadas:

Creadas por los desarrolladores, estas directivas pueden ser de atributo o estructurales y están diseñadas para resolver necesidades específicas no cubiertas por las directivas integradas.

## Diferencias entre Directivas Integradas y Personalizadas

Aspecto	Directivas Integradas	Directivas Personalizadas
Definición	Predefinidas por Angular, listas para su uso.	Creadas por desarrolladores según necesidades.
Propósito	Resolver casos comunes, como bucles o estilos.	Resolver necesidades específicas del proyecto.
Ejemplos	<code>*ngIf</code> , <code>*ngFor</code> , <code>ngClass</code> , <code>ngStyle</code> .	Directivas para resaltar texto o gestionar eventos personalizados.
Flexibilidad	Limitadas a los casos previstos por Angular.	Totalmente personalizables para cualquier requerimiento.

## ¿Cómo funcionan las Directivas Personalizadas?

Las directivas personalizadas utilizan el decorador `@Directive` para definir su comportamiento. Este decorador indica a Angular que el código asociado debe ejecutarse en un elemento del DOM. Además, pueden interactuar con elementos HTML a través de:

1. **ElementRef**: Para acceder al elemento DOM directamente.
2. **Renderer2**: Para modificar el DOM de manera segura y evitar conflictos en entornos de servidor.
3. **Decoradores como @Input y @HostListener**: Para pasar parámetros a la directiva o escuchar eventos del usuario.

## Ejemplo Conceptual: Directiva para Resaltar Texto

Una directiva personalizada puede resaltar un texto al pasar el mouse sobre él. Aquí se explica su implementación paso a paso:

### 1. Generar la directiva personalizada:

Usamos Angular CLI para crear la estructura básica.

```
ng generate directive Highlight
```

### 2. Definir la lógica de la directiva:

En el archivo `.ts` generado, se define el comportamiento deseado.

```
import { Directive, ElementRef, Renderer2, HostListener } from
'@angular/core';

@Directive({
  selector: '[appHighlight]'
})
export class HighlightDirective {
  constructor(private el: ElementRef, private renderer:
  Renderer2) {}

  @HostListener('mouseenter') onMouseEnter() {
    this.renderer.setStyle(this.el.nativeElement,
    'background-color', 'yellow');
  }

  @HostListener('mouseleave') onMouseLeave() {
    this.renderer.removeStyle(this.el.nativeElement,
    'background-color');
  }
}
```

### 3. Usar la directiva en un componente:

Se aplica el selector de la directiva como un atributo en el HTML.

```
<p appHighlight>Este texto se resalta al pasar el mouse.</p>
```

### 4. Resultado esperado:

Cuando el usuario pasa el cursor sobre el texto, el fondo cambia a amarillo. Al salir, el estilo vuelve al estado original.

## Beneficios de las Directivas Personalizadas

- Reutilización:** Una vez creada, la directiva puede aplicarse en cualquier parte del proyecto.
- Mantenimiento:** Encapsular lógica específica en directivas reduce el código redundante.
- Personalización:** Permite implementar comportamientos únicos y dinámicos que mejoran la experiencia del usuario.
- Consistencia:** Asegura que el mismo comportamiento se aplique de manera uniforme en diferentes componentes.

Este enfoque modular es especialmente útil en aplicaciones grandes y complejas, donde la reutilización y el mantenimiento del código son fundamentales

### Directivas de atributo vs. directivas estructurales

Criterio	Directivas de Atributo	Directivas Estructurales
Definición	Modifican la apariencia o el comportamiento de un elemento, atributo o componente existente.	Modifican la estructura del DOM, agregando, eliminando o manipulando elementos.
Sintaxis	Usan corchetes [ ] para enlazar propiedades o clases	Usan asterisco * para indicar el control de la estructura.
Decorador	@Directive	@Directive
Efecto en el DOM	Cambian el estilo, clase o propiedades de un elemento.	Añaden o eliminan elementos completos en el DOM.
Control de flujo	No. No afectan la estructura del DOM.	Sí. Pueden crear, eliminar o manipular elementos.
Ejemplos	ngClass, ngStyle, appHighlight	*ngIf, *ngFor, *ngSwitch
Uso común	Aplicación de clases dinámicas, estilos o atributos.	Mostrar u ocultar elementos, listas dinámicas, contenido condicional.
Reutilización	Se reutilizan para modificar la apariencia de cualquier componente HTML.	Se utilizan para controlar la lógica de flujo de la interfaz.
Impacto visual	Cambios de estilo, colores, visibilidad o clases CSS.	Cambio completo de la estructura del DOM.
Ejemplo de uso	<p [ngClass]="{'activo': esActivo}">Texto</p>	<div *ngIf="mostrarContenido"> Contenido</div>
Complejidad	Menor, ya que solo afecta el estilo o atributos de un elemento.	Mayor, ya que implica la manipulación de la estructura DOM.

# Deploy en Producción

## ¿Qué es el Deploy en Producción?

El **Deploy en Producción** es el proceso mediante el cual una aplicación web se prepara y se sube a un servidor para que pueda ser utilizada por los usuarios finales. Este proceso permite transformar la aplicación de su estado de **desarrollo a producción**, optimizando los archivos, mejorando el rendimiento y eliminando archivos innecesarios.

Angular permite realizar este proceso de forma sencilla usando **Angular CLI** y generando la carpeta de producción `dist/`, que contiene los archivos optimizados que se subirán a un servidor web.

## Proceso de Deploy paso a paso

### Paso 1: Revisión de la aplicación

Antes de generar la versión de producción, es importante verificar que no existan errores de sintaxis, dependencias faltantes o fallos en la aplicación.

#### Comando a ejecutar:

```
ng build
```

Este comando compila la aplicación y muestra cualquier error que pueda existir. Aunque no es obligatorio, se recomienda para garantizar la correcta generación del paquete de producción.

### Paso 2: Generación del paquete final

Para generar el paquete de producción, se utiliza el siguiente comando:

#### Comando a ejecutar:

```
ng build --prod
```

#### ¿Qué hace este comando?

- **Optimización de archivos:** Minimiza el código HTML, CSS y JavaScript.
- **Reducción de archivos:** Se eliminan comentarios y se reducen los nombres de variables.
- **Mejora del rendimiento:** Se habilita la compresión y se genera el código necesario para producción.

El resultado de este comando es la creación de una carpeta llamada `dist/` en el directorio raíz del proyecto. Esta carpeta contiene los archivos necesarios para la distribución.

### Paso 3: Configuración de variables de entorno

En una aplicación Angular, a menudo necesitamos variables de entorno que cambian entre desarrollo y producción, como la **URL de la API**.

Angular permite definir archivos de entorno para diferenciar entre desarrollo y producción:

- **Archivo para desarrollo:** `environment.ts`
- **Archivo para producción:** `environment.prod.ts`

#### Ejemplo de configuración del archivo `environment.prod.ts`:

```
export const environment = {  
  production: true,  
  apiUrl: 'https://mi-api-produccion.com'  
};
```

De esta forma, cuando ejecutamos `ng build --prod`, Angular usará la configuración de `environment.prod.ts`, asegurando que la aplicación se conecte a la URL de producción correcta.

#### Paso 4: Creación de la carpeta de producción (dist/)

Después de ejecutar el comando `ng build --prod`, se genera una carpeta `dist/` en la raíz del proyecto.

#### ¿Qué contiene esta carpeta?

- Archivos **HTML, CSS y JavaScript** optimizados.
- Todos los archivos necesarios para ejecutar la aplicación en un servidor.

Esta carpeta es la que se debe subir a un servidor para que la aplicación esté disponible para los usuarios finales.

#### Paso 5: Publicación de la aplicación

La carpeta `dist/` se puede subir a diferentes servidores para que la aplicación esté disponible en línea. Las opciones más comunes son:

Plataforma	Descripción	Costo
GitHub Pages	Subida de archivos de la carpeta <code>dist/</code> a un repositorio GitHub.	Gratis.
Netlify	Plataforma de despliegue automático. Permite actualizaciones automáticas.	Gratis y pagos avanzados.
Vercel	Similar a Netlify, pero con mayor integración para aplicaciones frontend.	Gratis y pagos avanzados.

Firebase Hosting	Servicio de Google que permite desplegar aplicaciones Angular fácilmente.	Gratis hasta cierto límite.
Heroku	Permite desplegar aplicaciones de backend y frontend completas.	Gratis y pagos avanzados.

### ¿Cómo subir la aplicación a Netlify?

1. Crear una cuenta en [Netlify](#).
2. Arrastrar la carpeta `dist/` al panel de Netlify.
3. ¡Listo! La aplicación estará en línea.

### Buenas prácticas para el Deploy

Para asegurar una correcta implementación de la aplicación en producción, se recomiendan las siguientes buenas prácticas:

1. **Validar antes de subir:** Ejecuta `ng build` antes de `ng build --prod` para evitar errores.
2. **Utilizar archivos de entorno:** Usa los archivos `environment.ts` y `environment.prod.ts` para mantener URLs separadas.
3. **Minimizar y optimizar:** Angular CLI ya se encarga de la optimización, pero asegúrate de que las imágenes estén optimizadas.
4. **Configurar la seguridad:** Revisa los encabezados de seguridad del servidor web (CORS, CSP).
5. **Pruebas de rendimiento:** Usa herramientas como **Lighthouse** para comprobar la velocidad de la aplicación en producción.

### Comandos útiles para Deploy

Comando	Descripción
<code>ng build</code>	Verifica la compilación de la aplicación.
<code>ng build --prod</code>	Genera la versión optimizada de la aplicación.
<code>ng serve</code>	Sirve la aplicación localmente para pruebas.
<code>ng build --base-href /mi-app/</code>	Define la URL base para aplicaciones SPA.

El **Deploy en Producción** es la última fase del ciclo de desarrollo de una aplicación. Con Angular CLI, este proceso se simplifica, permitiendo optimizar el rendimiento de la aplicación y asegurar su correcto funcionamiento en un entorno de producción. La correcta configuración de variables de entorno y la optimización de archivos garantizan que la aplicación sea ligera, rápida y fácil de usar para los usuarios finales.

# Actividades de la clase práctica en vivo - Unidad 4

## Actividad 1. Implementación de Angular Material

**Duración:** 20 minutos

**Objetivo:** Maquetar la vista de un componente utilizando **Angular Material**.

**Instrucciones:**

1. Utilizar **Angular CLI** para crear un nuevo componente.
2. Aplicar **Angular Material** para maquetar la vista de este componente.
3. Utilizar componentes de Angular Material, especialmente una **tabla**, para representar la información.

**Resolución:**

```
// componente-material.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-componente-material',
  templateUrl: './componente-material.component.html',
  styleUrls: ['./componente-material.component.css']
})
export class ComponenteMaterialComponent {
  displayedColumns: string[] = ['nombre', 'edad', 'ocupacion'];
  dataSource = [
    { nombre: 'Juan', edad: 30, ocupacion: 'Desarrollador' },
    { nombre: 'Ana', edad: 25, ocupacion: 'Diseñadora' },
    { nombre: 'Pedro', edad: 35, ocupacion: 'Analista' }
  ];
}
```

---

```
<!-- componente-material.component.html -->
<table mat-table [dataSource]="dataSource" class="mat-elevation-z8">
  <ng-container matColumnDef="nombre">
    <th mat-header-cell *matHeaderCellDef> Nombre </th>
    <td mat-cell *matCellDef="let element"> {{ element.nombre }} </td>
  </ng-container>

  <ng-container matColumnDef="edad">
    <th mat-header-cell *matHeaderCellDef> Edad </th>
    <td mat-cell *matCellDef="let element"> {{ element.edad }} </td>
  </ng-container>
```

```

<ng-container matColumnDef="ocupacion">
  <th mat-header-cell *matHeaderCellDef> Ocupación </th>
  <td mat-cell *matCellDef="let element"> {{ element.ocupacion }}</td>
</ng-container>

<tr mat-header-row *matHeaderRowDef="displayedColumns"></tr>
<tr mat-row *matRowDef="let row; columns: displayedColumns;"></tr>
</table>

```

## Actividad 2. Creación de un formulario con Angular Material

**Duración:** 20 minutos

**Objetivo:** Crear un formulario con componentes de **Angular Material** y mostrar un diálogo de confirmación.

**Instrucciones:**

1. Crear un formulario en un proyecto Angular utilizando componentes de Angular Material.
2. Incluir un **diálogo de confirmación** para la confirmación de envío de datos.

**Resolución:**

```

// formulario.component.ts
import { Component } from '@angular/core';
import { MatDialog } from '@angular/material/dialog';
import { FormBuilder, FormGroup } from '@angular/forms';

@Component({
  selector: 'app-formulario',
  templateUrl: './formulario.component.html',
  styleUrls: ['./formulario.component.css']
})
export class FormularioComponent {
  form: FormGroup;

  constructor(private fb: FormBuilder, public dialog: MatDialog) {
    this.form = this.fb.group({
      nombre: [''],
      email: ['']
    });
  }

  enviarFormulario() {
    this.dialog.open(DialogoConfirmacion);
  }
}

```

```

}

@Component({
  selector: 'dialogo-confirmacion',
  template: `<h1>Confirmación</h1><p>¡Formulario enviado con éxito!</p>`
})
export class DialogoConfirmacion {}

```

---

```

<!-- formulario.component.html -->
<form [formGroup]="form">
  <mat-form-field>
    <input matInput placeholder="Nombre" formControlName="nombre">
  </mat-form-field>

  <mat-form-field>
    <input matInput placeholder="Email" formControlName="email">
  </mat-form-field>

  <button mat-raised-button color="primary"
(click)="enviarFormulario()">Enviar</button>
</form>

```

## Actividad 3. Creación de un Pipe Personalizado

**Duración:** 10 minutos

**Objetivo:** Crear un **pipe personalizado** que formatee un número como moneda.

**Instrucciones:**

1. Crear un pipe llamado **moneda**.
2. El pipe debe recibir un número y un string como parámetro para definir la moneda (**euro**, **dólar**, etc.).

**Resolución:**

```

// moneda.pipe.ts
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'moneda'
})
export class MonedaPipe implements PipeTransform {
  transform(value: number, tipo: string): string {
    switch(tipo) {

```

```

        case 'euro':
            return `${value} €`;
        case 'dolar':
            return `$$${value}`;
        default:
            return value.toString();
    }
}
}

```

---

```

<!-- ejemplo de uso en un componente -->
<p>Euro: {{ 2000 | moneda:'euro' }}</p> <!-- Muestra: 2000 € -->
<p>Dólar: {{ 1000 | moneda:'dolar' }}</p> <!-- Muestra: $1000 -->

```

## Actividad 4. Creación de una Directiva Personalizada

**Duración:** 10 minutos

**Objetivo:** Crear una **directiva personalizada** para cambiar el tamaño de la fuente y el color de fondo.

**Instrucciones:**

1. Crear la directiva **appResaltado**.
2. La directiva debe permitir modificar tanto el **color de fondo** como el **tamaño de la fuente**.

**Resolución:**

```

// resaltado.directive.ts
import { Directive, ElementRef, Input, OnChanges } from
'@angular/core';

@Directive({
    selector: '[appResaltado]'
})
export class ResaltadoDirective implements OnChanges {
    @Input('appResaltado') colorResaltado!: string;
    @Input() tamano!: string;

    constructor(private elemento: ElementRef) {}

    ngOnChanges() {
        this.elemento.nativeElement.style.backgroundColor =
this.colorResaltado || 'yellow';
    }
}

```

```
    this.elemento.nativeElement.style.fontSize = this.tamano ?  
` ${this.tamano}px` : '16px';  
}  
}
```

---

```
<!-- ejemplo de uso -->  
<p [appResaltado]="'red'" [tamano]="'20'">Este texto está  
resaltado</p>
```

## Actividad 5. Integración de componentes y directivas

**Duración:** 30 minutos

**Objetivo:** Combinar **componentes de Angular Material** con una **directiva personalizada**.

**Instrucciones:**

1. Crear una **página principal** con:
  - Una barra de herramientas (**Toolbar**) con el título de la aplicación.
  - Tres botones interactivos que cambien su color de fondo de forma aleatoria.
  - Una lista generada dinámicamente a partir de un arreglo de datos.
2. Crear una **directiva personalizada** que cambie el color de fondo de los botones de forma aleatoria.

**Resolución:**

```
// main.component.ts  
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-main',  
  templateUrl: './main.component.html',  
  styleUrls: ['./main.component.css']  
)  
export class MainComponent {  
  elementos = ['Elemento 1', 'Elemento 2', 'Elemento 3'];  
  
  cambiarColor(elemento: HTMLElement) {  
    const colores = ['red', 'blue', 'green', 'orange'];  
    const colorAleatorio = colores[Math.floor(Math.random() *  
      colores.length)];  
    elemento.style.backgroundColor = colorAleatorio;  
  }  
}
```

---

```
<!-- main.component.html -->
<mat-toolbar color="primary">Mi Aplicación</mat-toolbar>

<button mat-raised-button (click)="cambiarColor($event.target)">Botón
1</button>
<button mat-raised-button (click)="cambiarColor($event.target)">Botón
2</button>
<button mat-raised-button (click)="cambiarColor($event.target)">Botón
3</button>

<ul>
  <li *ngFor="let item of elementos">{{ item }}</li>
</ul>
```