Material 3

José M. Torresano
Noviembre 2025
buzondelprofesor@gmail.com

Sistema de color

Nuevo paradigma

Material 3 representa un salto cualitativo en la forma en que concebimos y aplicamos el color en el diseño de interfaces de usuario. Google ha introducido un sistema de color dinámico y expresivo, diseñado para adaptarse a las preferencias del usuario y a las necesidades de cada aplicación.

Este sistema se basa en la generación de esquemas de color **armónicos** a partir de **colores clave**, lo que permite crear interfaces visualmente coherentes y accesibles. Además, Material 3 introduce el concepto de "roles de color", que definen cómo se aplican los colores a los diferentes elementos de la interfaz, proporcionando una guía clara y sistemática para el uso del color.

Roles principales del color

primary

Este es el color principal de la aplicación. Se utiliza para elementos clave como botones principales, estados activos y barras de progreso. **Representa la identidad principal de la aplicación.**

secondary

Este color complementa al color primario y se utiliza para elementos menos destacados. Puede usarse para botones secundarios, selectores y otros elementos que requieren una diferenciación visual.

tertiary

Este color se utiliza para añadir contraste y variedad a la paleta de colores. Puede usarse para elementos flotantes, indicadores y otros elementos que requieren un alto nivel de diferenciación.

error

Este color se utiliza para indicar errores y estados de alerta. Se aplica a mensajes de error, indicadores de validación y otros elementos que requieren una atención especial.

Importancia de los roles de color

Los roles de color proporcionan una forma sistemática y coherente de aplicar colores en toda la aplicación.

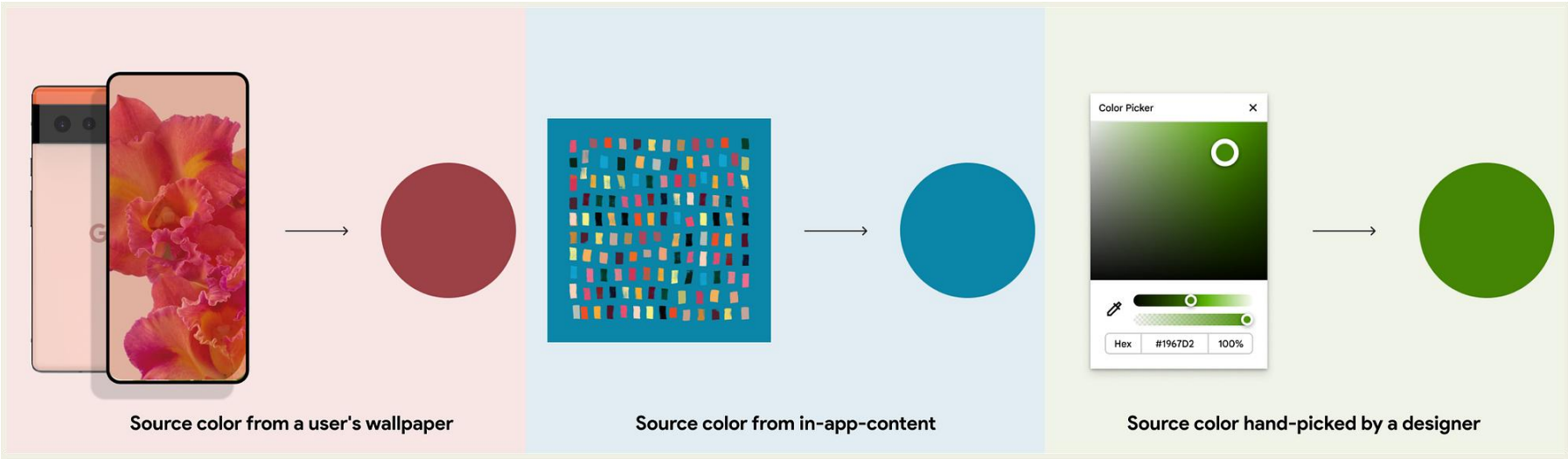
Aseguran que los colores se utilicen de manera lógica y predecible, lo que mejora la usabilidad y la accesibilidad.

Facilitan la creación de temas claros y oscuros, ya que los roles de color se adaptan automáticamente a diferentes modos.

Principios clave

Colores semilla (*seed colors*)

El sistema comienza con uno o más colores semilla, que sirven como base para generar toda la paleta de colores. Estos colores semilla pueden ser elegidos por el diseñador o extraídos dinámicamente del fondo de pantalla del usuario.



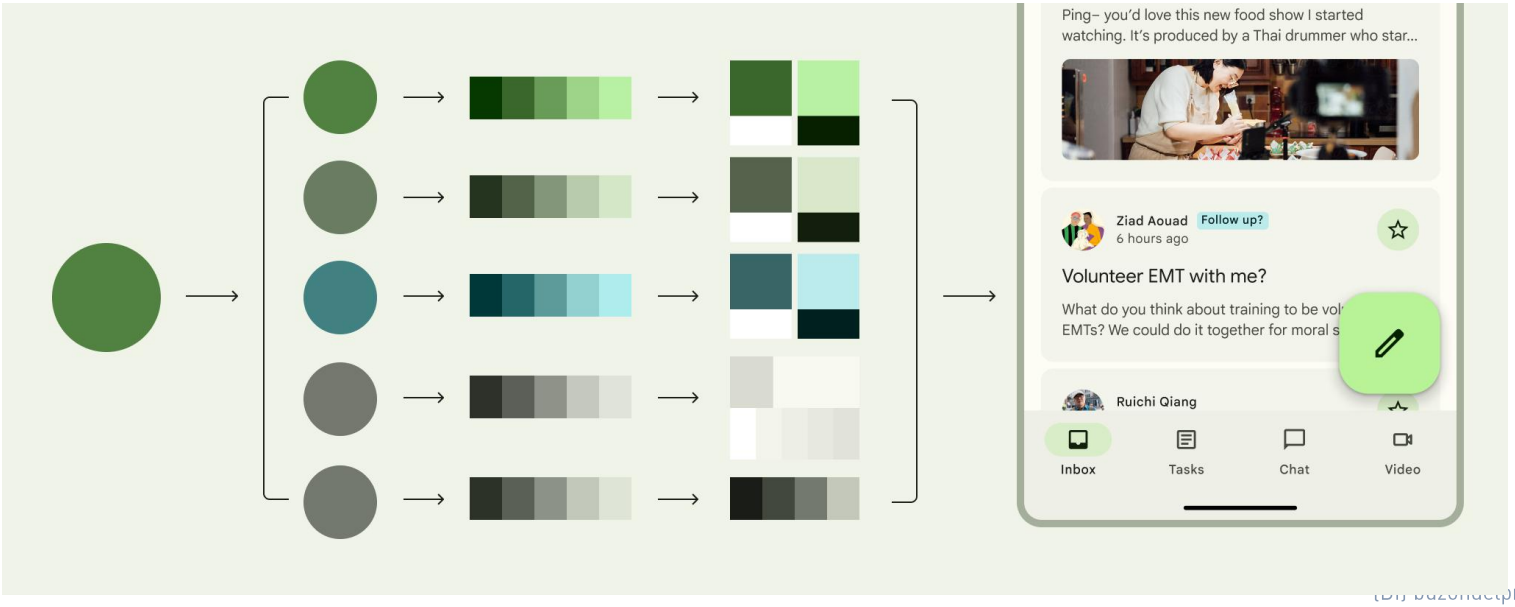
Principios clave

Generación de tonos (*tonal palettes*)

A partir de los colores semilla, se generan paletas tonales, que consisten en una serie de tonos y matices del color base que aseguran que los colores tengan suficiente contraste para la legibilidad y que funcionen bien juntos visualmente.

Esquemas de color (*color schemes*)

Material 3 utiliza algoritmos para generar esquemas de color armónicos a partir de las paletas tonales. Estos esquemas de color definen cómo se aplican los colores a los diferentes elementos de la interfaz de usuario, utilizando los roles de color mencionados anteriormente (primary, secondary, tertiary, etc.).



Principios clave

Adaptación a modos claro y oscuro

Material 3 genera esquemas de color separados para los modos claro y oscuro, asegurando que la interfaz de usuario sea legible y cómoda en cualquier condición de iluminación.

Beneficios de la generación armónica de colores

Coherencia visual

Material 3 garantiza que los colores se utilicen de manera coherente y armoniosa en toda la aplicación.

Accesibilidad

Las paletas tonales aseguran que los colores tengan suficiente contraste para la legibilidad, lo que mejora la accesibilidad para usuarios con discapacidades visuales.

Personalización

El sistema permite a los desarrolladores personalizar los colores semilla y los esquemas de color para crear aplicaciones con una identidad visual única.

Adaptabilidad

Material 3 genera esquemas de color separados para los modos claro y oscuro, asegurando que la interfaz de usuario sea legible y cómoda en cualquier condición de iluminación.

Roles principales del color

background

Este color se utiliza para el fondo principal de la aplicación.

surface

Se utiliza para superficies de componentes, como tarjetas, diálogos y menús.

onPrimary, onSecondary, onTertiary, onError, onBackground, onSurface

Estos colores se utilizan para el texto y los iconos que se muestran sobre los colores primario, secundario, terciario, de error, de fondo y de superficie, respectivamente. Aseguran un contraste adecuado para la legibilidad.

surfaceVariant

Un color variante de surface que sirve para dar otros contrastes dentro de la interfaz.

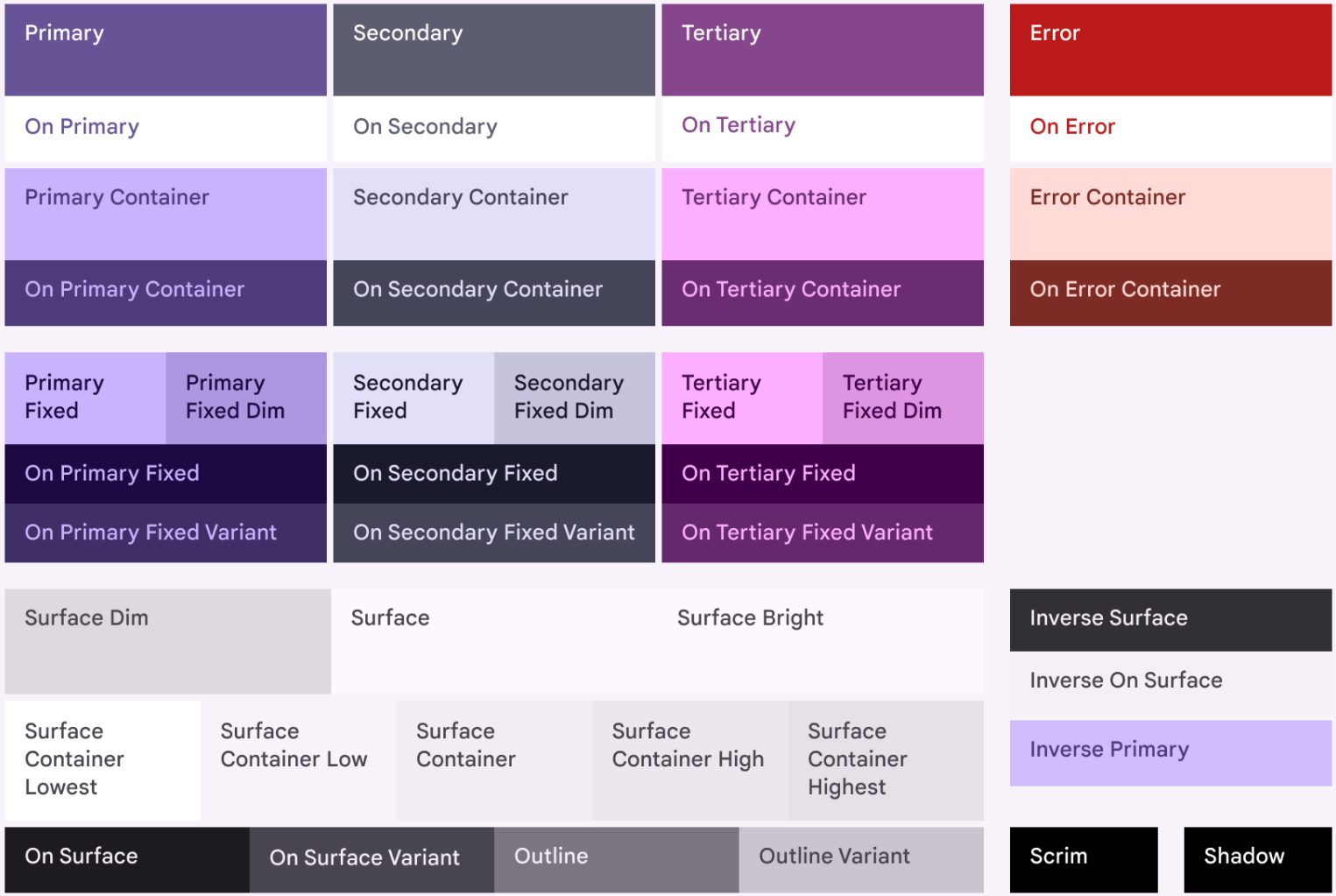
outline

Color del contorno de diversos componentes, como cajas de texto, botones, etc.

outlineVariant

Color variante del contorno, normalmente usado para contornos de menor énfasis.

Roles principales del color



ColorScheme

ColorScheme

En Flutter, `ColorScheme` es la clase que nos permite definir y aplicar esquemas de color de Material 3 a nuestra aplicación. Actúa como un contenedor para todos los roles de color mencionados anteriormente, y nos proporciona una forma fácil y centralizada de gestionar la paleta de colores de nuestra interfaz de usuario.

ColorScheme

1. Creación de un `ColorScheme`

- Podemos crear un `ColorScheme` utilizando uno de los constructores predefinidos, como `ColorScheme.fromSeed()`, que genera un esquema de color armónico a partir de un color semilla.
- También podemos crear un `ColorScheme` personalizado definiendo cada rol de color individualmente.

2. Aplicación del `ColorScheme` al `ThemeData`

- Una vez que tenemos un `ColorScheme`, podemos aplicarlo a nuestro `ThemeData` utilizando la propiedad `colorScheme`.
- El `ThemeData` resultante se utilizará para aplicar los colores a los diferentes widgets de nuestra aplicación.

3. Acceso a los colores del `ColorScheme` en los widgets

- En cualquier widget, podemos acceder al `ColorScheme` actual utilizando `Theme.of(context).colorScheme`.
- Luego, podemos utilizar los roles de color del `ColorScheme` para aplicar colores a los diferentes elementos del widget.

ColorScheme

```dart
MaterialApp(
  title: 'Demo Theme',
  theme: ThemeData(
    colorScheme: ColorScheme.fromSeed(
      seedColor: Colors.blue,
      brightness: Brightness.dark, // o Brightness.light
    ),
  ),
  home: HomeScreen(),
);
```

Con clases

```dart
import 'package:flutter/material.dart';

ThemeData lightTheme = ThemeData(
  colorScheme: ColorScheme.fromSeed(
    seedColor: Colors.blue,
    brightness: Brightness.light,
  ),
);
```

```dart
MaterialApp(
  title: 'Demo Theme',
  theme: ligthTheme,
  home: HomeScreen(),
);
```

ColorScheme

Podemos anular los valores de los roles de color individuales cuando se usa `ColorScheme.fromSeed()` especificándolos explícitamente.

```
MaterialApp(
  title: 'Demo Theme',
  theme: ThemeData(
    colorScheme: ColorScheme.fromSeed(
      seedColor: Colors.blue,
      secondary: Colors.teal,
      onSecondary: Colors.teal,
      secondaryContainer: Colors.teal.shade400,
      onSecondaryContainer: Colors.teal.shade600,
      secondaryFixed: Colors.teal.shade400,
      onSecondaryFixed: Colors.teal.shade800,
      onSecondaryFixedVariant: Colors.teal.shade600,
      secondaryFixedDim: Colors.teal.shade200
    ),
  ),
  home: HomeScreen(),
)
```

Anulaciones locales: Temas para widgets individuales

1. Anular el tema localmente

Envuelve el widget en un widget `Theme` y proporciona una nueva instancia `ThemeData`

El tema solo se aplicará al widget `FloatingActionButton()`

```
Theme(
  data: ThemeData(
    colorScheme: ColorScheme.fromSeed(seedColor: Colors.yellow)
  ),
  child: FloatingActionButton(onPressed: () {}, child: const Icon(Icons.add)),
);
```

2. Ampliar el tema existente

Cuando quieras reutilizar el tema actual de la aplicación, pero modificar valores específicos, utiliza `Theme.of(context).copyWith()`.

El tema solo se aplicará al widget `FloatingActionButton()`

```
Theme(
  data: Theme.of(context,).copyWith(
    colorScheme: ColorScheme.fromSeed(seedColor: Colors.pink)
  ),
  child: const FloatingActionButton(onPressed: null, child: Icon(Icons.add)),
);
```

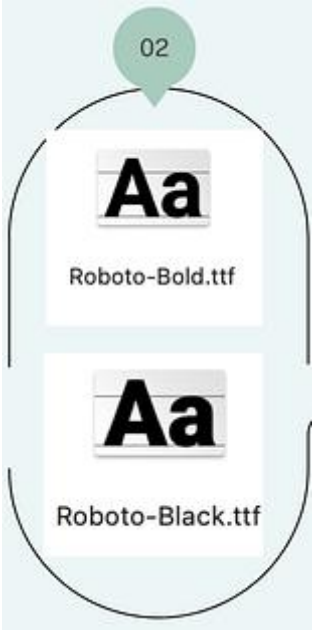
Fuentes

Typeface vs Font vs Typography

El **tipo de fuente** es el diseño de las letras: sus curvas, grosor, estilo y apariencia general. Es como la personalidad de las letras.

TYPEFACE

Una **fuente** es una versión específica de ese tipo de letra, que incluye peso, tamaño y estilo. Ejemplo: Roboto Bold Italic 16pt es una fuente de la familia Roboto.



FONT

La **tipografía** es la forma en que utilizas fuentes en tu diseño o aplicación, lo que incluye elegir tipos de letra, establecer tamaños, espaciado y altura de línea, alinear texto y combinar fuentes.

TYPOGRAPHY

{DI} buzondelprofesor@gmail.com

Como agregar fuentes en Flutter

1. Descargándolas y agregándolas manualmente

- Crea la carpeta assets/fonts y copia ahí tus archivos de fuentes
- Registra la fuente en pubspec.yalm

```
flutter:
  fonts:
    - family: MiFuente
      fonts:
        - asset: assets/fonts/MiFuente-Regular.ttf
        - asset: assets/fonts/MiFuente-Bold.ttf
          weight: 700
```

- Utiliza la fuente en el código

```
Text(
 'Hola Mundo',
 style: TextStyle( fontFamily: 'MiFuente-Regular'),
);
```

```
Text(
  'Hola mundo',
  style: TextStyle(fontFamily: 'MiFuente'),
)
```

Como agregar fuentes en Flutter

2. Utilizando el paquete `google_fonts`

- Ejecuta el comando `flutter pub get`
- Importa el paquete en tu código

```dart
import 'package:google_fonts/google_fonts.dart';
```

- Utiliza la fuente en el código

```dart
Text(
 'Hola Mundo',
 style: GoogleFonts.roboto(
   textStyle: TextStyle(fontSize: 32.0),
   ),
),
```

TextTheme

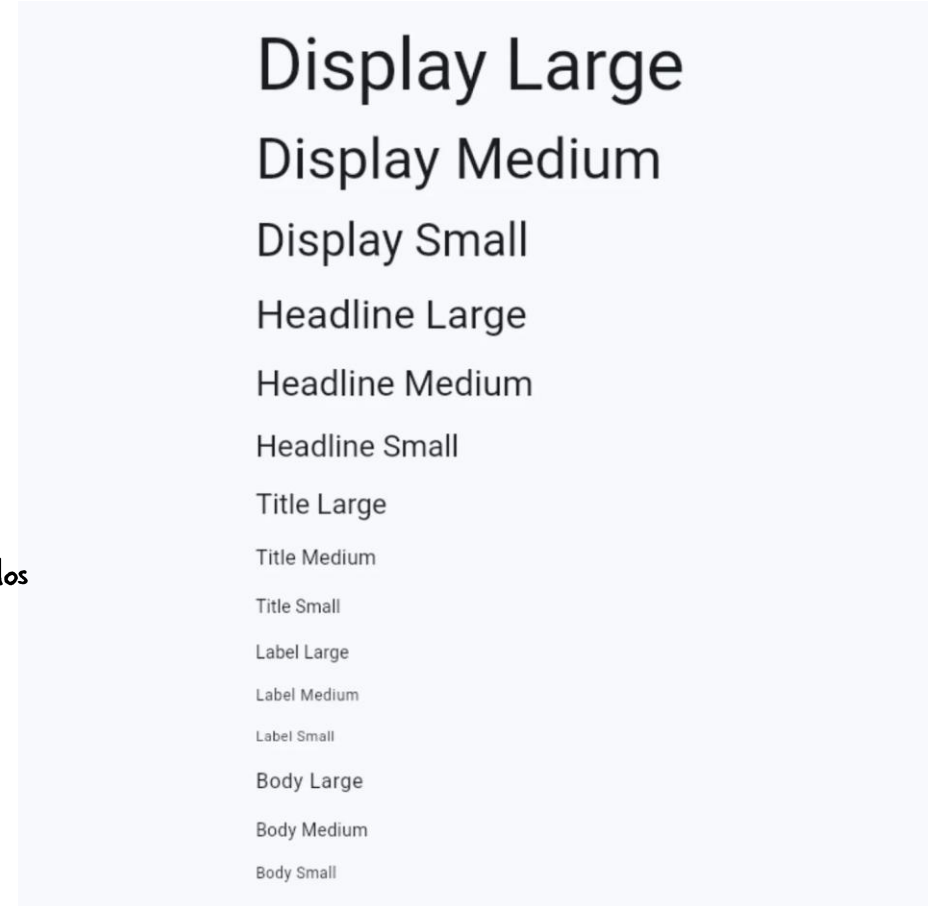
¿Qué es un TextTheme?

En nuestras aplicaciones, necesitamos diferentes estilos de texto para mostrar un título, una etiqueta y un texto. Cada uno de estos estilos tiene diferente tamaño de fuente, grosor, estilo, etc.

Flutter proporciona una escala tipográfica común en `TextTheme`: un conjunto estructurado de estilos de texto categorizados por función:

Categoría	Variantes
Display	Large, Medium, Small
Headline	Large, Medium, Small
Title	Large, Medium, Small
Label	Large, Medium, Small
Body	Large, Medium, Small

Esto nos da 15 `TextStyle` predefinidos



Definiendo un TextTheme a medida

Podemos definir cualquiera de los 15 estilos en ThemeDate.textTheme:

```dart
// Si utilizamos el paquete de Google Fonts
import 'package:google_fonts/google_fonts.dart';


textTheme: TextTheme(
  displayLarge: GoogleFonts.almendra(
    fontSize: 48,
    fontWeight: FontWeight.w700,
    fontStyle: FontStyle.italic,
),)


// Si agregamos las fuentes manualmente en el proyecto
textTheme: TextTheme(
  displayLarge: TextStyle(
    fontFamily: 'TikTok Sans',
    fontSize: 48,
    fontWeight: FontWeight.w700,
    fontStyle: FontStyle.italic,
),)
```

Uso

```dart
Text("Hola mundo",
  style: Theme.of(context).textTheme.bodyLarge,
)
```

Sobreescribiendo un TextTheme

A veces, es posible que desees que un widget Text específico tenga un estilo personalizado:

```dart
// 1.Google fonts Package
Text(
  'Awesome Gang',
  style: GoogleFonts.almendra(
    fontSize: 48,
    fontWeight: FontWeight.w700,
    fontStyle: FontStyle.italic,
  )
)
```

```dart
// 2.Fuentes agregadas manualmente
Text(
  'Awesome Gang',
  style: TextStyle(
    fontFamily: 'RobotoMono'
    fontSize: 48,
    fontWeight: FontWeight.w700,
    fontStyle: FontStyle.italic,
)
```

```dart
// O extendiendo un estilo de tema existente
Text("Hola mundo",
     style: Theme.of(context).textTheme.bodyLarge!.copyWith(
               fontSize: 24,
               fontStyle: FontStyle.normal),
)
```

KEEP CALM AND ASK QUESTIONS