Flutter

Themes

Carmelo Villegas Cruz

Themes

En la mayoría de las Apps existe un diseño común que se repite a lo largo de todas sus pantallas.

Por ejemplo, puede pasar que todos los AppBar tengas estilos en común como el color.

¿Es necesario repetir todo la implementación de estos estilos en cada una de las pantallas?
Por suerte, la respuesta es NO
Gracias al uso de Themes.

Theme widget

En Flutter todo es un widget, y podemos desarrollar las interfaces, anidando widgets usando child y children para conseguirlo.

Themes, se comporta como cualquier otro widget y por lo tanto podría tener hijos, **child**.

Themes además trabaja con la técnica **InheritedWidget**, que consiste en que cada descendiente puede acceder al tema usando **Theme.of(context)**

Explore InheritedWidget

De manera sencilla es como hablar de un "mensajero" que lleva información a muchas partes de tu aplicación Flutter sin necesidad de pasarla manualmente por cada widget.

Vamos a verlo con un ejemplo.

Explore Inherited Widget

Imagina que tienes una casa (**tu aplicación**) con varias habitaciones (**tus widgets**).

Ahora, en una habitación colocas una pizarra con información importante, como la hora o el nombre de una persona.

Las demás habitaciones pueden ver esa pizarra para obtener esa información, en lugar de que cada habitación tenga que pedir la hora por separado.

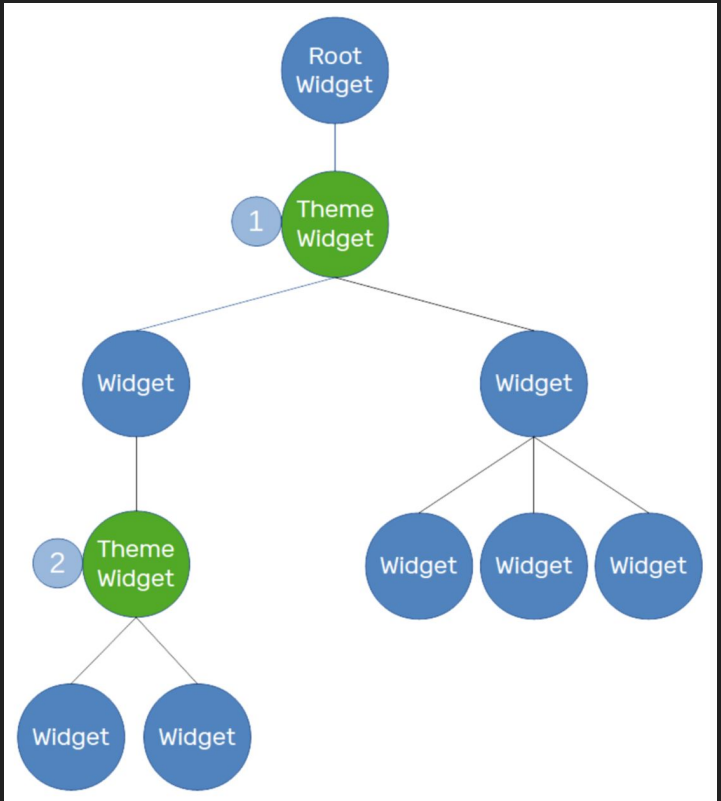
En este caso, la pizarra es el Inherited Widget.

Explore InheritedWidget

De una forma más técnica.

Un Inherited Widget permite compartir datos entre widgets que están dentro de su árbol de widgets (sus descendientes) sin necesidad de pasar explícitamente esos datos como parámetros por cada widget intermedio. Los widgets hijos pueden acceder a estos datos utilizando el método of(context).

Explore Inherited Widget



El tema 1 es aplicado a sus descendientes.

El tema 2 sobreescribe el 1 y el subárbol a partir de 2 heredan el tema 2.

Podemos acceder a los temas definidos a través de Theme.of(context) y hacer uso tanto de los temas que hayamos definidos como aquellos definidos por Material Design.

Explore Inherited Widget

¿Y esto tiene alguna ventaja?

- Hace que los datos sean fáciles de compartir en el árbol de widgets.
- Cuando los datos cambian, solo los widgets que los usan se reconstruyen, no todo el árbol.

ThemeData

Widget que ayuda a **Theme** a aplicar estilos.

Usando las propiedades de la clase ThemeData podemos personalizar los estilos relacionados con la aplicación como colores, tipografía y componentes específicos.

Si decidimos seguir el estilo iOS tanto Theme como ThemeData tiene su correspondiente widget, CupertinoTheme y CupertinoThemeData.

Theming in practice - style

```
Container(
 color: Colors.grey,
 child: Center(
   child: Text("Text",
   textDirection:TextDirection.ltr,
   style: Theme.of(context).textTheme.headlineMedium,),
   //style recibe un TextStyle
 ),
);
```

Haciendo uso de la propiedad style hemos conseguido el estilo headlineMedium definido en Material design a no ser que este estilo hubiera sido sobreescrito en la propiedad Theme y este sería el que se aplicaría.

Theming in practice - MaterialApp theme

```
MaterialApp(
 theme: ThemeData(
     textTheme: TextTheme(headlineMedium: TextStyle(color:
Colors.amber)),
 home: Ejemplo(),
));
```

Al sobrescribir el estilo de **headlineMedium** cuando hagamos referencia a él como vimos en la diapositiva anterior el color del texto debería cambiar.

Theming in practice - Theme

```
Container(
 color: Colors.blue,
 child: Center(
   child: Theme(
     data: Theme.of(context).copyWith(
       textTheme: Theme.of(context).textTheme.copyWith(
            headlineMedium:  TextStyle(color: Colors.green,),
          ),
     ),
     child: Text("Text",
       textDirection: TextDirection.ltr,
       style: Theme.of(context).textTheme.headlineMedium,),
   ),
 ),
);
```

Añadimos en la jerarquía el widget Theme para personalizar **headlineMedium**

Ahora los estilos para headlineMedium son los definidos en el padre.

Theming in practice - Theme

Si ejecutamos el código anterior, texto se debería de ver en verde pero eso no ocurre, ¿por qué?

La respuesta es **Contexto**, otra vez nos está rondando el contexto pero veamos qué hacer para solucionarlo.

Theming in practice - Theme

```
Builder(
 builder: (context) => Text("Texto",
   textDirection: TextDirection.ltr,
   style: Theme.of(context).textTheme.headline4,
 ),
),
```

Funciona con Builder dado que este widget delega el método **build** y puede ocurrir en niveles inferiores del árbol.

Platform class - dart:io

A veces cuando estamos desarrollando una App necesitamos diferenciar en qué plataforma se está ejecutando la App, por ejemplo para aplicar unos estilos diferentes o bien para definir unas navegación diferente.

Para ello usamos **Platform** y algunos de sus getters: isAndroid, isFuchsia, isIOS …

Platform class - dart:io

```
Platform.isAndroid
    ? MaterialApp(
 theme: ThemeData(primaryColor: Colors.grey),
)
    : CupertinoApp(
 theme: CupertinoThemeData(primaryColor:
 CupertinoColors.lightBackgroundGray),
);
```