Flutter

MINI APUNTES FLUTTER 1

Desarrollo de Interfaces

3 DE NOVIEMBRE DE 2024

ERIK AMO TOQUERO

Contenido

Introducción a Flutter

Flutter es un framework desarrollado por Google para crear aplicaciones multiplataformas con una base de código única. Permite la compilación Just-In-Time para desarrollo y utiliza Dart como lenguaje de programación.

Tiene una estructura algo compleja, ya que los Widgets de Flutter se van introduciendo como hijas unas de otras, dando una jerarquía algo compleja frente a la estructura lineal de la programación normalmente vista.

Ciclo de Vida

El ciclo de vida de una aplicación es la siguiente:

- main() -> Punto de entrada de la aplicación
- runApp() -> Inicio de la aplicación
- MaterialApp -> Configuración de la aplicación
- Scaffold -> Estructura base de una pantalla en Flutter

El ciclo de vida de un widget depende de su tipo de estado:

- **StatefulWidget:** Tiene distintos cambios de estado a lo largo de su ciclo de vida.
- **StatelessWidget:** No tiene cambios de estado

```
class MainApp extends StatelessWidget {
  const MainApp({super.key});

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: Scaffold(
        body: Center(
          child: Text('Hello World!'),
        ),
      ),
    );
  }
}
```

Widgets

Los widgets son bloques de construcción visual que definen parte de la interfaz gráfica. Cada elemento que aparece dentro de una aplicación es un Widget.

La jerarquía de la interfaz gráfica de una aplicación de Flutter es en un árbol jerárquico.
Es decir:

1. Widget principal
    a. Widget secundario
        i. Widget terciario 1
            1. Widget cuaternario 1
            2. Widget cuaternario 2
        ii. Widget terciario 2

Cada Widget puede ser Stateful o Stateless, depende de si tiene cambios de estado (como es el caso de los botones)

Método build()

Se define la interfaz gráfica de un widget y se ejecuta cada vez que su estado cambia. **Se tiene que tener en cuenta que build() no debe tener muchos cálculos costosos para que no tarde en cargar la aplicación**.

StatefulWidget

Los StatefulWidget guardan su información en otro objeto, de tipo State. Este objeto tendrá el método build():

Esto se puede utilizer para cambiar estados como se va a enseñar a continuación: Al dar un botón, se añade un 1 a un contador.

```
class MainApp extends StatefulWidget {
  const MainApp({super.key});

  @override
  State<StatefulWidget> createState() {
    return Estado();
  }
}

class Estado extends State<MainApp> {
  int incremento = 0;
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        body: Column(children: [
          Text('$incremento'),
          ElevatedButton(
            child: Text("Pulsa"),
            onPressed: () {setState(() { //Se edita el estado
              incremento++; //Incrementa el valor
            });},
          )
        ]),
      ),
    );
  }
}
```

Con el método setState() se actualiza el estado del widget que se esté editando, en este caso es la pantalla principal.

Scaffold

Es la estructura básica de una pantalla de flutter. Tiene los siguientes widgets:

- **AppBar**: Barra superior
- **Drawer**: Menú lateral
- **BottomNavigationBar**: Menú inferior
- **FloatingActionButton**: Botón flotante

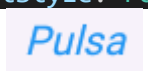
Listado de Widgets que se pueden utilizar:

Text – Texto en pantalla

Simplemente texto, el cual contiene las siguientes variables:

- **data:** El string que va a mostrar
- **style:** El estilo del texto (TextStyle())
    - **Color:** Color de texto
    - **FontSize (double):** Tamaño de fuente
    - **FontFamily (String):** Tipo de fuente
    - **FontStyle:** Tipo de fuente (Itálico, Negrita, normal…)

```
Text("Pulsa",style: TextStyle(color: Colors.blue, fontSize: 20,
fontFamily: "Comic Sans MS", fontStyle: FontStyle.italic),),
```

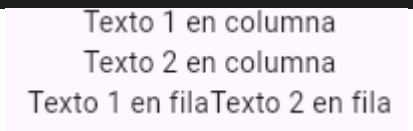


Column y Row – Contenedores que alinean widgets

Column alinea los widgets de forma vertical y Row horizontalmente. Ambos tienen las mismas propiedades principales:

- **mainAxisAlignment:** La alineación del eje principal (center, right, left)
- **mainAxisSize:** Tamaño máximo de la alineación
- **children:** Todos los widgets que tienen

```
Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [Text("Texto 1 en columna"), Text("Texto 2 en columna")],
),
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [Text("Texto 1 en fila"), Text("Texto 2 en fila")],
)
```



SizedBox – Caja en blanco

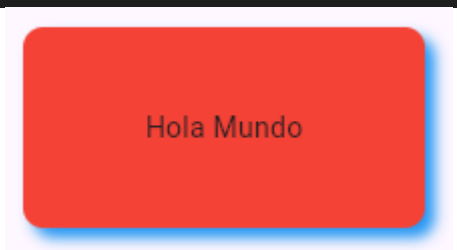
Se suele utilizar para dejar espacios en blanco entre contenedores de fila y columna. Tiene las medidas de altura (height) y longitud (width)

Container – Contenedor de elementos

Un contenedor con un solo hijo (child) que permite aplicar estilos. Tiene las siguientes propiedades:

- **color:** Color de la caja
- **padding:** Espacio dentro del contenedor
- **margin:** Espacio fuera del contenedor
- **alignment:** Alineación del hijo
- **decoration (BoxDecoration()):** Decoración que sigue el contenedor. Si tiene esta propiedad no puede tener la propiedad color:
    - **color:** Color de la caja (Dentro del BoxDecoration())
    - **decorationImage:** Imagen de fondo
    - **boxShadow (List<BoxShadow>):** Sombras de la caja
        - **color:** Color de la sombra
        - **offset:** Desplazamiento de la sombra
        - **blurRadius:** Radio de desenfoque
    - **borderRadius:** Curvatura del borde
    - **boxShape:** Forma geométrica de la caja
    - **border:** Otros ajustes del borde
- **child:** Widget hijo
- **width y height:** Parámetros de tamaño

```
Container(
  margin: EdgeInsets.all(100),
  alignment: Alignment.center,
  decoration: const BoxDecoration(
    boxShadow: [BoxShadow(
      color: Colors.blue,
      offset: Offset(5, 5),
      blurRadius: 5
    )],
    color: Colors.red,
    borderRadius: BorderRadius.all(Radius.circular(10)),
  ),
  child: Text("Hola Mundo"),
  width: 200,
  height: 100,
)
```



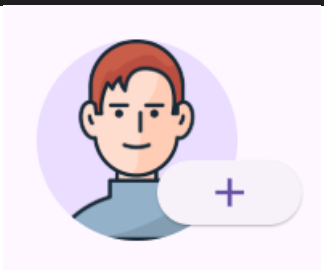
Padding – Espacio dentro de un widget

Además de ser un parámetro en contenedores, también existe el Padding como widget. Sirve para separar widgets sin más

Stack – Contenedor de widgets superpuestos

Funciona con una arquitectura por capas y se utiliza el widget Positioned para definir la posición exacta de un widget

```
Stack(
  children: [
    Positioned(
      left: 20,
      top: 20,
      child: const CircleAvatar(
        radius: 50,
        backgroundImage: NetworkImage("https://cdn-icons-
png.flaticon.com/512/5556/5556499.png"),
      ),
    ),
    Positioned(
        top: 80,
        left: 80,
        child: ElevatedButton(onPressed: () {}, child: Icon(Icons.add)))
  ],
),
```



Flex, Expanded y Flexible – Modificadores del contenido de contenedores

- **Flex** es una clase para Row y Column para gestionar la disposición de sus hijos
- **Expanded** es un widget que toma todo el espacio disponible de su contenedor. Por ejemplo, si hay 2 widgets con expanded, cada uno ocupará la mitad
- **Flexible** funciona como Expanded, pero tiene más control sobre el tamaño que puede tomar un widget

ListView – Listas

Es un widget que muestra listas de elementos que se desplazan verticalmente

Hay varios tipos de ListViews además del normal (poco útil para elementos dinámicos):

- ListView.builder: utiliza una cuenta para repetir el mismo patrón tantas veces como el programa lo requiera. Muy útil en el uso de colecciones debido a que permite crear listas dinámicas muy fáciles de editar
- ListView.separated: Es una lista pero que agrega separadores sin más

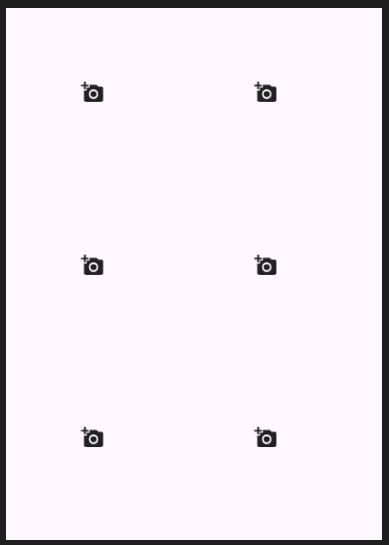
```
ListView( //Ejemplo con ListView
  children: [
    ListTile(
      leading: Icon(Icons.abc),
      title: Text("Titulo 1"),
    ),
    ListTile(
      leading: Icon(Icons.abc),
      title: Text("Titulo 2"),
    ),
    ListTile(
      leading: Icon(Icons.abc),
      title: Text("Titulo 3"),
    ),
  ],
)
ListView.builder(//Ejemplo con builder
  itemCount: 3, //Cantidad de iteraciones
  itemBuilder: (context, count) {//count = itemCount
    return ListTile( //Devuelve count listTiles
      leading: Icon(Icons.abc),
      title: Text("Título ${count + 1}"), //Count empieza en 0
    );
  }
)
ListView.separated(
  separatorBuilder: (context, index) => Divider(), //Divisor,
  itemCount: 12,
  itemBuilder: (context, count) {
    return ListTile(//Devuelve count listTiles
      leading: Icon(Icons.abc),
      title: Text("Título ${count + 1}"), //Count empieza en 0
    );
  }
)
```

GridView – Información en cuadrícula

Además de que utiliza cosas raras que no nos interesa, no se utiliza GridView directamente, sino que se utiliza GridView.count y GridView.builder

- GridView.count permite definir el número de columnas que queremos y define lo demás de forma “Automática”
- GridView.builder crea elementos de forma dinámica, como ListView.builder

```
GridView.count(//Caso de count
  crossAxisCount: 2, //Cantidad de columnas
  children: [
    Icon(
      Icons.add_a_photo,
      size: 20,
    ),
    Icon(
      Icons.add_a_photo,
      size: 20,
    ),
    Icon(
      Icons.add_a_photo,
      size: 20,
    ),
    Icon(
      Icons.add_a_photo,
      size: 20,
    ),
    Icon(
      Icons.add_a_photo,
      size: 20,
    ),
    Icon(
      Icons.add_a_photo,
      size: 20,
    ),
    Icon(
      Icons.add_a_photo,
      size: 20,
    ),
  ],
)
GridView.builder(//Caso de builder
  itemCount: 6, //Cantidad de items
  gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(crossAxisCount:
2), //gridDelegate: Sistema de Grid que va a seguir
  itemBuilder: (context, count) { //ItemBuilder como el de listView
    return Icon(
      Icons.add_a_photo,
      size: 20,
    );
  }
)
```

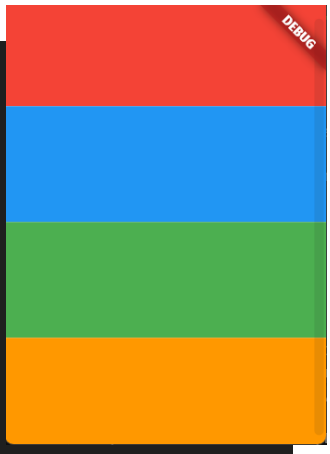


ScrollView – Scroller

Este widget permite desplazar los distintos widgets cuando el contenido es mayor al tamaño de la pantalla. Existen dos tipos:

- **SingleChildScrollView:** Desplaza una columna de widgets
- **ScrollController:** Maneja el desplazamiento de la pantalla en casos de listas dinámicas. **No es un widget, es un Controller**

```dart
SingleChildScrollView(
  child: Column(
    children: <Widget>[
      Container(height: 100, color: Colors.red),
      Container(height: 100, color: Colors.blue),
      Container(height: 100, color: Colors.green),
      Container(height: 100, color: Colors.orange),
    ],
  ),
)
```




```dart
ListView.builder(//Ejemplo de ScrollController con listView
  controller: ScrollController(),
  itemCount: 3,
  itemBuilder: (context, count) {
    return ListTile(
      leading: Icon(Icons.abc),
      title: Text("Título ${count + 1}"),
    );
  }
)
```

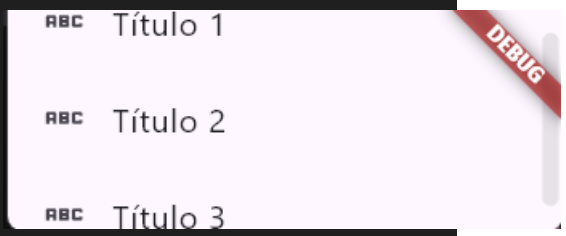


Image – Imágenes

Como su nombre indica, muestra imágenes como contenido. Estas imágenes pueden ser de dos tipos:

- **AssertImage:** Imágenes locales del proyecto (*Importante asignar una carpeta para los assets en pubspec.yaml*)
- **NetworkImage:** Imágenes en la red cuya única necesidad es poner su url

```
GridView.count(crossAxisCount: 2, children: <Widget>[
            Image(image: AssetImage("assets/aaaaa.jpg")),
            Image(
                image: NetworkImage(
                    "https://media.istockphoto.com/id/1296242195/es/vector/mano-agitada-dibujos-animados-moviendo-la-mano-humana-gesto-de-saludo-o-despedida-
signo.jpg?s=612x612&w=0&k=20&c=2Sp3dfkXHelfYGaeKZwHg_SaeVZ6mTo0nRXYwcqQz2k="))
          ])
```

Form – Formulario

Los widgets de Form permiten agrupar varios campos de entrada de información y darles una validación. Para dicha validación, hay que crear un **FormKey**

Más adelante se verá un ejemplo, mientras tanto aquí hay elementos de los formularios:

TextField/TextFormField – Bloques de entrada de texto

Permite al usuario escribir texto para luego usarlo.

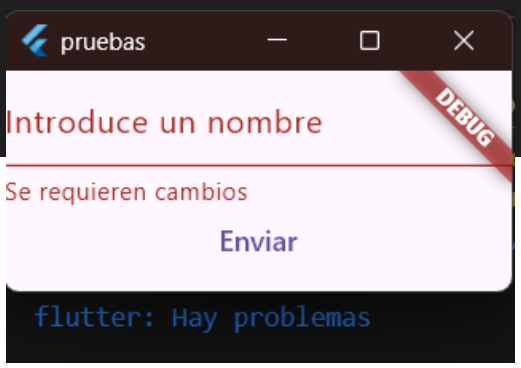
La diferencia entre TextField y TextFormField es que **TextFormField permite hacer validaciones en el propio widget**.

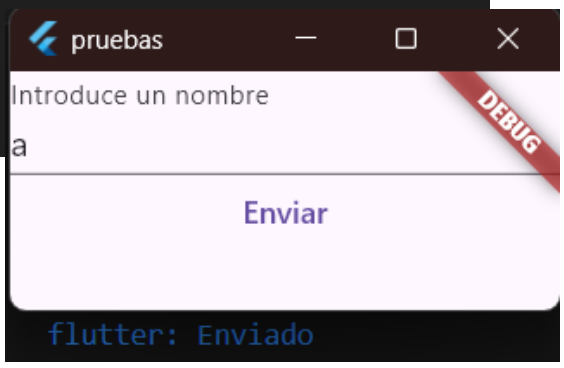
Botones

Sirven para realizar acciones. Se utilizan para validar y enviar Forms. Hay 3 tipos de botones:

- **FloatingActionButton**: Botón flotante
- **TextButton**: Botón con texto
- **IconButton**: Botón con icono

```dart
class Estado extends State<MainApp> {
  final _formkey =
      GlobalKey<FormState>(); //Creo y guardo una clave para el
formulario
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
        home: Scaffold(
            body: Column(
      children: [
        Form(
          key: _formkey,
          //Formulario
          child: Column(
            children: [
              TextFormField(
                decoration: InputDecoration(
                    labelText: "Introduce un nombre"), //Añadir texto
                validator: (value) {
                  if (value == null || value.isEmpty) {
                    return 'Se requieren cambios';
                  } else {
                    return null; //No hay fallos
                  }
                },
              )
            ],
          ),
        ),
        TextButton(
            onPressed: () {
              if (_formkey.currentState!.validate()) {
                print("Enviado");
              } else {
                print("Hay problemas");
              }
            },
            child: Text("Enviar"))
      ],
    )
  )
);
  }
}
```





Objetos en Flutter

Una vez se ha visto toda la teoría de Widgets solo queda aclarar una cosa: Los widgets son y pueden ser tratados como objetos. ¿Esto en qué beneficia? Pues en que se pueden reutilizar fragmentos de código con gran facilidad.

Por ejemplo, queremos hacer contenedores de imágenes exactamente iguales salvo porque tienen distintas fotos. Pues se hace un container de x altura y anchura como objeto predefinido, que tenga como variable la url que se le pase:

```
class ImageContenedor extends StatelessWidget {
  const ImageContenedor({super.key, required this.url});

  final String url;

  @override
  Widget build(BuildContext context) {
    return Container(
      width: 100,
      height: 100,
      decoration: BoxDecoration(
        borderRadius: BorderRadius.all(Radius.circular(10)),
        image: DecorationImage(
          image: NetworkImage(url),
          fit: BoxFit.cover,
        ),
      ),
    );
  }
}
```

Y ahora lo utilizamos fuera de aquí:

```
GridView.count(crossAxisCount: 2, children: <Widget>[
            ImageContenedor(
                url:"https://d7hftxdivxxvm.cloudfront.net/?quality=80&res
ize_to=width&src=https%3A%2F%2Fartsy-media-
uploads.s3.amazonaws.com%2F2RNK1P0BYVrSCZEy_Sd1Ew%252F3417757448_4a6bdf36
ce_o.jpg&width=910"),
            ImageContenedor(url:
"https://www.clarin.com/2021/05/21/xNdTyR1R5_1200x0__1.jpg")
          ])
```

Este es un caso sencillo de para qué sirve, y la complejidad que ganan los objetos de Flutter es tanta como la que quieras dar a tu programa