

Diseño aplicado a interfaces web con Flutter

Unidad 05. Principales widgets en Flutter



Autor: Sergi García

Actualizado Noviembre 2025

Licencia



Reconocimiento - No comercial - CompartirIgual (BY-NC-SA): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se ha de hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán diferentes símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

Importante

Atención

Interesante

ÍNDICE

1. Introducción: todo es un widget en Flutter	3
2. Estructura básica de una app Flutter	4
3. Widgets en Flutter	6
4. Widgets con y sin estado	8
5. Creación de un widget personalizado	10
6. Resumen de principales Widgets	11
7. Custom Scaffold en Flutter	20
8. Recursos recomendados para aprender Flutter	23

UNIDAD 05. PRINCIPALES WIDGETS EN FLUTTER

1. INTRODUCCIÓN: TODO ES UN WIDGET EN FLUTTER

◆ Todo es un widget

En Flutter, absolutamente todo lo que ves en pantalla es un widget.

- Un texto es un widget.
- Un botón es un widget.
- La pantalla entera también es un widget.

Incluso elementos que no son visibles, como espaciadores (SizedBox), alineaciones (Align) o márgenes (Padding), se representan como widgets.

Esto significa que la UI se construye combinando widgets, de la misma forma que construyes un objeto más complejo a partir de piezas más simples.

◆ Analogía con LEGO

Piensa en Flutter como un juego de LEGO:

- Cada pieza pequeña (bloques de colores, ruedas, ventanas) es un widget simple (Text, Icon, Image).
- Al combinar piezas, puedes formar estructuras más complejas (una casa → Scaffold, un coche → Row con Icons).
- Igual que en LEGO, no necesitas reinventar la rueda: tienes piezas listas que puedes reutilizar y apilar.

📌 Ejemplo:

```
Scaffold(
  appBar: AppBar(title: Text("Analogía LEGO")),
  body: Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Icon(Icons.build, size: 50, color: Colors.blue), // Pieza 1
        SizedBox(height: 10), // Pieza 2
        Text("Todo es un widget", // Pieza 3
            style: TextStyle(fontSize: 20)),
      ],
    ),
  ),
)
```

👉 Igual que construir una figura con bloques LEGO, aquí apilamos piezas (widgets) para obtener la interfaz.

◆ Árbol de widgets y eficiencia en renderizado

Cuando Flutter dibuja la interfaz, organiza los widgets en un árbol jerárquico conocido como Widget Tree.

Ejemplo de código:

```
MaterialApp(
  home: Scaffold(
    body: Center(
      child: Text("Hola Flutter"),
```

```
  ),
),
);
```

Representación del árbol:

```
MaterialApp
└── Scaffold
    └── Center
        └── Text("Hola Flutter")
```

Importancia del árbol de widgets:

- Cada widget tiene un parent y puede tener hijos.
- Cuando algo cambia en la interfaz (ej: texto o color), Flutter no redibuja todo, solo los widgets afectados.
- Esto hace que el renderizado sea muy eficiente:
 - Si cambias solo un Text, no se vuelve a renderizar toda la pantalla, únicamente ese widget.
 - La eficiencia viene de este mecanismo de reutilización de widgets en el árbol.

2. ESTRUCTURA BÁSICA DE UNA APP FLUTTER

◆ Estructura básica de una aplicación Flutter

Toda app Flutter comienza en el archivo main.dart dentro del directorio lib/. Este es el punto de entrada:

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Mi primera app Flutter',
      home: HomePage(),
    );
  }
}

class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Inicio')),
      body: Center(child: Text('Hola Mundo')),
    );
  }
}
```

◆ Explicación del código

- `main()` → función principal que lanza la app con `runApp()`.
- `MyApp` → widget raíz que define el diseño global.
- `MaterialApp` → proporciona navegación, temas, rutas, etc.
- `Scaffold` → estructura visual estándar con `AppBar`, `Body`, `Drawer`, etc.
- `HomePage` → pantalla principal.

A continuación damos una explicación más detallada de como es una App de Flutter para principiantes:

🏗 Estructura básica de una app Flutter

Imagina que construir una app es como armar una casa. Necesitas planos (código) y materiales (widgets). Todo comienza en el archivo `main.dart` (la puerta de entrada!).

📦 Partes principales del código

```
import 'package:flutter/material.dart'; // 📦 Traemos Las "herramientas" de Flutter
```

👉 Esto es como: Abrir tu caja de herramientas antes de construir.
material.dart contiene todos los widgets básicos (botones, textos, diseños).

🚀 Función main() - El motor de la app

```
void main() {
    runApp(MyApp()); // 🏁 Inicia La app con el widget MyApp
}
```

👉 Así funciona:

- `main()` es como el interruptor de luz de tu casa (lo primero que se ejecuta).
- `runApp()` "enciende" la aplicación usando `MyApp` (el widget principal).

🌳 MyApp - La raíz de todo

```
class MyApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return MaterialApp( // 🎨 Define el "estilo" de La app
            title: 'Mi primera app', // Nombre (para el sistema)
            home: HomePage(), // 🏠 Primera pantalla al abrir
        );
    }
}
```

👉 Claves:

- `MaterialApp` es el "diseñador de interiores":
 - Configura temas, rutas y la pantalla inicial (`home`).
- Es `Stateless` porque no cambia después de crearse.

🏠 HomePage - La pantalla principal

```
class HomePage extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return Scaffold( // 🏠 Estructura básica de pantalla
```

```

        appBar: AppBar(title: Text('Inicio')), //  Barra superior
        body: Center(child: Text('Hola Mundo')), //  Contenido central
    );
}
}

```

Partes del Scaffold (andamiaje):

Widget	Función	Ejemplo real
AppBar	Barra superior con título	Como el nombre en WhatsApp
Body	Área principal de la pantalla	El chat en WhatsApp
(Opcionales) Drawer, FloatingActionButton	Menú lateral o botón flotante	El menú de Gmail



Consejos para recordar:

1. main.dart siempre es el punto de entrada.
2. MaterialApp envuelve toda la app (como un contenedor gigante).
3. Scaffold da estructura a cada pantalla (como paredes y techo).
4. Los widgets se anidan como muñecas rusas:
MaterialApp > Scaffold > Center > Text.

Ejemplo visual:

```

MaterialApp( //  La app completa
    home: Scaffold( //  Una pantalla
        body: Center( //  Centra contenido
            child: Text('Hola'), //  Widget final
        ),
    ),
)

```

3. WIDGETS EN FLUTTER

¿Qué es un widget en Flutter?

En Flutter, todo es un widget. Pero, ¿qué significa esto exactamente? Imagina que estás construyendo una casa de LEGO:

- **Widget = Pieza de LEGO**

Cada elemento visual o funcional en tu app (botones, textos, imágenes, pantallas, incluso la estructura completa) es un widget. Se combinan como bloques para crear interfaces complejas.

Definición técnica

Un widget es:

1. Componente reutilizable: Pequeña unidad de UI (Interfaz de Usuario) o lógica.
2. Configurable: Recibe parámetros (props) para personalizar su comportamiento/apariencia.
3. Anidable: Los widgets se componen unos dentro de otros (como un árbol).

Ejemplo visual:

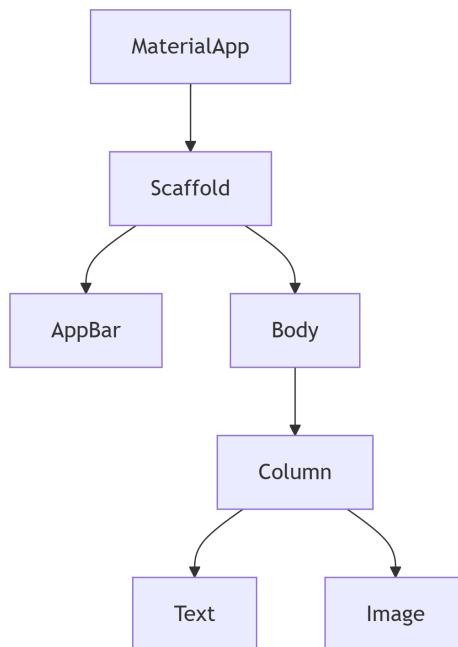
```
Column( // ← Widget padre
  children: [
    Text('Hola'), // ← Widget hijo
    Icon(Icons.star), // ← Widget hijo
  ],
)
```

🔍 Tipos de widgets

Tipo	Ejemplos comunes	¿Para qué sirve?
Layout	Row, Column, Stack	Organizar otros widgets en pantalla.
Visual	Text, Image, Icon	Mostrar contenido estático.
Interactivo	ElevatedButton, TextField	Responder a acciones del usuario.
Estructural	Scaffold, Container	Proporcionar "esqueleto" a la UI.

🌳 El árbol de widgets

Flutter organiza los widgets en una jerarquía de padres e hijos:



¿Por qué importa?

- Eficiencia: Flutter solo redibuja los widgets que cambian (no toda la pantalla!).
- Legibilidad: El código sigue la estructura visual.

¿Por qué Flutter usa widgets?

1. Modularidad: Puedes reusar widgets en cualquier parte.
2. Declarativo: Describe QUÉ quieres mostrar (no CÓMO hacerlo paso a paso).
3. Optimización: Flutter gestiona automáticamente su renderizado.

Regla de Oro

"En Flutter, si lo ves en pantalla, es un widget. Si no se ve, pero afecta a otros (como gestores de estado), también puede ser un widget."

4. WIDGETS CON Y SIN ESTADO

En Flutter, la interfaz de usuario se construye a partir de widgets, que se dividen en dos grandes categorías:

- **StatelessWidget** → No tienen estado interno.
- **StatefulWidget** → Mantienen un estado que puede cambiar durante la ejecución.

Comprender esta distinción es clave para diseñar interfaces eficientes y mantenibles.

Definición formal

◆ **StatelessWidget**

- Inmutables: No almacenan datos que cambien después de su creación.
- Renderizado: Solo se reconstruyen cuando reciben nuevos parámetros (desde el padre).
- Ciclo de vida: Muy simple → únicamente implementan el método build().

Ejemplo:

```
class MiWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Text('Soy un widget sin estado');
  }
}
```

◆ **StatefulWidget**

- Mutables: Mantienen un estado interno mediante la clase State.
- Renderizado: Se actualizan al invocar setState().
- Ciclo de vida: Más complejo, incluye métodos como:
 - initState() → inicialización.
 - dispose() → liberación de recursos.

Ejemplo:

```
class Contador extends StatefulWidget {
  @override
  _ContadorState createState() => _ContadorState();
}

class _ContadorState extends State<Contador> {
  int contador = 0;

  void incrementar() {
    setState(() {

```

```

        contador++;
    });
}

@Override
Widget build(BuildContext context) {
    return Column(
        children: [
            Text('Contador: $contador'),
            ElevatedButton(
                onPressed: incrementar,
                child: Text('Incrementar'),
            ),
        ],
    );
}
}

```

Analogía sencilla

Imagina una caja de herramientas:

- **StatelessWidget** → Como un martillo.
 - Siempre hace lo mismo: clavar.
 - No recuerda nada después de usarlo.
- **StatefulWidget** → Como un termómetro digital.
 - Muestra un valor que cambia con el tiempo.
 - Reacciona a la temperatura (estado) y actualiza la pantalla.

Tipos de widgets en Flutter

En Flutter todo es un widget, desde la estructura general hasta los detalles más pequeños.

Se dividen principalmente en:

- **StatelessWidget**
 - Sin estado interno.
 - Redibujados solo si su padre cambia.
 - Ejemplo típico: Text, Icon, Image.
- **StatefulWidget**
 - Guardan y actualizan un estado.
 - Usan `setState()` para refrescar la UI.
 - Ejemplo típico: Checkbox, TextField, Slider.

Resumen rápido

Tipo	Estado	Ejemplo	Ciclo de vida
 StatelessWidget	 No guarda estado	Text, Icon	Solo <code>build()</code>
 StatefulWidget	 Guarda estado	Checkbox, TextField	<code>initState()</code> , <code>setState()</code> , <code>dispose()</code>

👉 Buenas prácticas:

- Usa StatelessWidget cuando la UI no dependa de datos que cambian.
- Usa StatefulWidget cuando necesites dinamismo o interacción con el usuario.

5. CREACIÓN DE UN WIDGET PERSONALIZADO

◆ ¿Por qué crear widgets personalizados?

En Flutter, la interfaz se construye a base de widgets. Sin embargo, cuando un diseño se repite muchas veces en una app (por ejemplo, un mismo estilo de botón, tarjeta o cabecera), lo ideal es encapsularlo en un widget propio.

Esto nos da tres ventajas principales:

1. Reutilización → Evitamos repetir código en múltiples pantallas.
2. Legibilidad → El código queda más limpio y fácil de mantener.
3. Escalabilidad → Si necesitamos cambiar el diseño, lo hacemos en un solo lugar y se actualiza en toda la app.

◆ Ejemplo: BotonPersonalizado

Aquí creamos un widget que recibe texto y color como parámetros y renderiza un botón con un estilo definido:

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

// Definición del widget personalizado
class BotonPersonalizado extends StatelessWidget {
    final String texto;
    final Color color;

    // Constructor con parámetros obligatorios
    BotonPersonalizado(this.texto, this.color);

    @override
    Widget build(BuildContext context) {
        return Container(
            padding: EdgeInsets.all(12),
            decoration: BoxDecoration(
                color: color,
                borderRadius: BorderRadius.circular(8), // Bordes redondeados
            ),
            child: Text(
                texto,
                style: TextStyle(
                    color: Colors.white,
                    fontSize: 16,
                    fontWeight: FontWeight.bold,
                ),
                textAlign: TextAlign.center,
            ),
        );
    }
}
```

◆ **Uso en una pantalla del widget personalizado de ejemplo**

Podemos instanciar el botón en cualquier parte de nuestra app simplemente pasando el texto y color deseado:

```
class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Widget Personalizado")),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            BotonPersonalizado('Aceptar', Colors.green),
            SizedBox(height: 20),
            BotonPersonalizado('Cancelar', Colors.red),
          ],
        ),
      ),
    );
  }
}
```

◆ **Explicación paso a paso**

1. Creamos un StatelessWidget → porque el botón no necesita manejar estado interno.
2. Definimos parámetros en el constructor → texto y color permiten personalizar cada instancia.
3. Usamos un Container con padding, color de fondo y bordes redondeados para dar estilo.
4. Mostramos un Text centrado con estilo en blanco y negrita.
5. En el uso, simplemente llamamos BotonPersonalizado("Aceptar", Colors.green).

◆ **Analogía práctica**

Imagina que estás fabricando moldes en una fábrica:

- En lugar de diseñar cada pieza una y otra vez, creas un molde (widget personalizado).
- Cada vez que lo necesitas, rellenas el molde con distintos parámetros (texto, color, tamaño).
- Así obtienes botones consistentes, rápidos de producir y fáciles de modificar en masa.

 **Claves para recordar**

- Usa widgets personalizados para reutilizar código y mantener tu proyecto ordenado.
- Si no necesita estado → StatelessWidget.
- Si el widget cambia con interacción del usuario → StatefulWidget.
- Los parámetros en el constructor permiten configuración flexible de cada instancia.

6. RESUMEN DE PRINCIPALES WIDGETS

Widgets fundamentales en Flutter

Estos son los "ladrillos esenciales" para construir interfaces en Flutter. Cada uno resuelve necesidades específicas de diseño y funcionalidad.

Text

Descripción: Muestra texto en pantalla con estilo.

Propiedades clave:

- style: Fuente, tamaño, color (usar TextStyle).
- textAlign: Alineación (centro, izquierda, etc.).

Ejemplo:

```
Text(
  'Hola Flutter!',
  style: TextStyle(
    fontSize: 24,
    fontWeight: FontWeight.bold,
    color: Colors.blue,
  ),
)
```

Row / Column

Descripción:

- Row: Organiza widgets horizontalmente.
- Column: Organiza widgets verticalmente.

Propiedades clave:

- mainAxisAlignment: Alineación en el eje principal (ej: MainAxisAlignment.center).
- crossAxisAlignment: Alineación en el eje secundario.
- children: Lista de widgets hijos.

Ejemplo comparativo:

```
Row( // ←→
  children: [
    Icon(Icons.star),
    Text('Fila'),
  ],
)

Column( // ↪
  children: [
    Icon(Icons.star),
    Text('Columna'),
  ],
)
```

Container

Descripción: "Caja" personalizable con decoración y espaciado.

Propiedades clave:

- padding: Espacio interno.
- margin: Espacio externo.
- decoration: Color, bordes, sombras (BoxDecoration).
- width/height: Tamaño fijo.

Ejemplo:

```
Container(
  padding: EdgeInsets.all(16),
  margin: EdgeInsets.symmetric(vertical: 8),
  decoration: BoxDecoration(
    color: Colors.amber,
    borderRadius: BorderRadius.circular(10),
  ),
  child: Text('Contenedor'),
)
```

 **Padding**

Añade **espacio interno** alrededor de un widget.

```
Padding(
  padding: EdgeInsets.all(16),
  child: Text("Texto con padding"),
);
```

 Es como el “margen interno” en CSS.

 **Image**

Descripción: Muestra imágenes desde diferentes fuentes.

Tipos de carga:

- `Image.asset('ruta/local')`: Desde archivos del proyecto.
- `Image.network('URL')`: Desde internet.

Ejemplo:

```
Image.network(
  'https://example.com/imagen.jpg',
  width: 200,
  fit: BoxFit.cover, // Ajuste de la imagen
)
```

 **ElevatedButton**

Descripción: Botón con elevación visual (Material Design).

Propiedades clave:

- `onPressed`: Función al presionar (si es null, se deshabilita).
- `child`: Widget hijo (texto, icono, etc.).

Ejemplo:

```
ElevatedButton(
  onPressed: () {
    print('Botón presionado!');
  },
  child: Text('Presiona aquí'),
)
```

ListView

Descripción: Lista desplazable (vertical u horizontal).

Casos de uso:

Lista básica:

```
ListView(
  children: [
    ListTile(title: Text('Item 1')),
    ListTile(title: Text('Item 2')),
  ],
)
```

Dinámica (para muchos elementos):

```
ListView.builder(
  itemCount: 100,
  itemBuilder: (context, index) {
    return ListTile(title: Text('Item $index'));
  },
)
```

Stack

Descripción: Superpone widgets (útil para elementos en capas).

Ejemplo común: Texto sobre imagen.

```
Stack(
  children: [
    Image.network('https://example.com/fondo.jpg'),
    Positioned( // Posiciona un hijo relativo al Stack
      bottom: 10,
      child: Text('Texto superpuesto'),
    ),
  ],
)
```

Expanded

Descripción: Ocupa el espacio disponible en Row/Column.

Regla clave: Solo funciona dentro de Row o Column.

Ejemplo:

```
Row(
  children: [
    Expanded( // ← Ocupa 70% del espacio
      flex: 7,
      child: Container(color: Colors.red),
    ),
    Expanded( // ← Ocupa 30%
      flex: 3,
      child: Container(color: Colors.blue),
    ),
  ],
)
```

Flexible

El widget Flexible se usa dentro de un Row o Column para controlar cómo los hijos ocupan el espacio disponible.

- Permite que un widget crezca o se encoja según el espacio libre.
- Se diferencia de Expanded:
 - Expanded ocupa todo el espacio disponible.
 - Flexible ocupa solo lo necesario, pero puede estirarse o encogerse según sea necesario.

Ejemplo:

```
Row(
  children: [
    Flexible(
      flex: 2,
      child: Container(color: Colors.red, height: 50),
    ),
    Flexible(
      flex: 1,
      child: Container(color: Colors.blue, height: 50),
    ),
  ],
)
```

👉 Aquí el primer contenedor ocupa el doble de espacio que el segundo.

Caso de uso real: diseño responsive donde los elementos deben adaptarse a pantallas pequeñas sin romper la interfaz (ejemplo: columnas en una ficha de producto).

AnimatedSwitcher

Permite animar transiciones cuando cambia un widget hijo.

```
AnimatedSwitcher(
  duration: Duration(milliseconds: 400),
  child: Text(
    "Hola",
    key: ValueKey("texto1"),
  ),
);
```

👉 Muy útil para cambios de texto, iconos o pantallas pequeñas con animación suave.

FloatingActionButton (FAB)

Botón circular flotante, normalmente para la acción principal de la pantalla.

```
FloatingActionButton(
  onPressed: () => print("Añadir"),
  child: Icon(Icons.add),
);
```

DropdownButton

Crea un menú desplegable para seleccionar entre varias opciones.

```
DropdownButton<String>(
  value: "A",
  items: ["A", "B", "C"].map((e) {
    return DropdownMenuItem(value: e, child: Text(e));
  }).toList(),
  onChanged: (valor) => print("Seleccionado $valor"),
);
```

Ejemplo integrado

Combina varios widgets para crear una tarjeta de producto:

```
Container(
  margin: EdgeInsets.all(10),
  decoration: BoxDecoration(
    border: Border.all(color: Colors.grey),
  ),
  child: Column(
    children: [
      Image.network('https://example.com/producto.jpg'),
      Padding(
        padding: EdgeInsets.all(8),
        child: Row(
          children: [
            Expanded(
              child: Text('Zapatos deportivos'),
            ),
            ElevatedButton(
              onPressed: () {},
              child: Text('Comprar'),
            ),
          ],
        ),
      ),
    ],
  ),
)
```

Flexible

El widget Flexible se usa dentro de un Row o Column para controlar cómo los hijos ocupan el espacio disponible.

- Permite que un widget crezca o se encoja según el espacio libre.
- Se diferencia de Expanded:
 - Expanded ocupa todo el espacio disponible.
 - Flexible ocupa solo lo necesario, pero puede estirarse o encogerse según sea necesario.

Ejemplo:

```
Row(
  children: [
    Flexible(
      flex: 2,
      child: Container(color: Colors.red, height: 50),
    ),
    Flexible(
      flex: 1,
      child: Container(color: Colors.blue, height: 50),
    ),
  ],
)
```

👉 Aquí el primer contenedor ocupa el doble de espacio que el segundo.

Caso de uso real: diseño responsive donde los elementos deben adaptarse a pantallas pequeñas sin romper la interfaz (ejemplo: columnas en una ficha de producto).

 **Tabla resumen de casos de uso**

Widget	¿Cuándo usarlo?	Ejemplo real
Text	Mostrar información textual.	Nombre de usuario, títulos de sección.
Row / Column	Organizar widgets en fila o columna.	Fila de iconos de redes, formulario vertical.
Container	Agrupar y dar estilo (color, bordes, padding, tamaño).	Caja destacada en un dashboard, tarjeta simple.
ListView	Mostrar listas desplazables.	Chat de WhatsApp, catálogo de productos.
Stack	Superponer elementos.	Avatar con estado, imagen con texto encima.
ElevatedButton	Acción principal destacada.	Botón “Enviar”, “Comprar ahora”.
FloatingActionButton	Acción flotante única y principal.	Botón “+” en WhatsApp para nuevo chat.
AnimatedSwitcher	Animar transiciones de widgets hijos.	Cambio de contador, ícono de favorito animado.
Padding	Añadir espacio interno alrededor de un widget.	Margen en botones o textos.
Card	Mostrar información en un bloque con sombra y bordes.	Tarjeta de perfil en LinkedIn, producto en una tienda.

Image (asset/network)	Mostrar imágenes locales o de internet.	Logo de app, avatar remoto.
Icon	Representar acciones o estados con un ícono.	Ícono de búsqueda, estrella de favoritos.
DropdownButton	Elegir entre varias opciones desplegables.	Selección de país, filtro de orden en e-commerce.
TextField	Captura de texto por el usuario.	Búsqueda en Google, campo de contraseña.
Checkbox	Selección múltiple de opciones.	Lista de tareas, aceptar términos y condiciones.
Slider	Selección de un valor dentro de un rango.	Control de volumen, brillo de pantalla.
Switch	Activar o desactivar algo (booleano).	Modo oscuro, activar notificaciones.
AppBar	Barra superior con título y acciones.	Barra superior en Instagram con logo y notificaciones.
Scaffold	Estructura base de cada pantalla.	Pantallas principales con AppBar, FAB y Drawer.
Flexible	Ajustar cómo los widgets ocupan espacio en Row/Column.	Columnas de producto adaptables en distintas resoluciones.

Ejemplo real: Catálogo de productos

Ejemplo práctico que combina varios de los widgets clave:

- Scaffold como estructura base.
- AppBar con título.
- ListView para mostrar una lista de productos.
- Card con Row y Flexible para que los elementos se adapten al espacio.
- FloatingActionButton para añadir un producto.

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

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
```

```
return MaterialApp(  
    title: 'Catálogo Flutter',  
    theme: ThemeData(primarySwatch: Colors.blue),  
    home: ProductosPage(),  
)  
}  
  
}  
  
class ProductosPage extends StatelessWidget {  
    final List<Map<String, dynamic>> productos = [  
        {"nombre": "Camiseta Flutter", "precio": 20.0, "imagen": Icons.tshirt_rounded},  
        {"nombre": "Sudadera Dart", "precio": 35.0, "imagen": Icons.checkroom},  
        {"nombre": "Taza Dev", "precio": 10.0, "imagen": Icons.local_cafe},  
        {"nombre": "Pegatinas", "precio": 5.0, "imagen": Icons.emoji_emotions},  
    ];  
  
    @override  
    Widget build(BuildContext context) {  
        return Scaffold(  
            appBar: AppBar(title: Text("🛍 Catálogo de Productos")),  
            body: ListView.builder(  
                itemCount: productos.length,  
                itemBuilder: (context, index) {  
                    final producto = productos[index];  
                    return Card(  
                        margin: EdgeInsets.symmetric(horizontal: 12, vertical: 6),  
                        elevation: 3,  
                        child: Padding(  
                            padding: const EdgeInsets.all(12),  
                            child: Row(  
                                children: [  
                                    Icon(producto["imagen"], size: 50, color: Colors.blue),  
                                    SizedBox(width: 16),  
                                    Flexible(  
                                        flex: 3,  
                                        child: Column(  
                                            crossAxisAlignment: CrossAxisAlignment.start,  
                                            children: [  
                                                Text(producto["nombre"],  
                                                    style: TextStyle(  
                                                        fontSize: 18, fontWeight: FontWeight.bold)),  
                                                SizedBox(height: 4),  
                                                Text("\$${producto["precio"]}"),  
                                                style: TextStyle(  
                                                    fontSize: 16, color: Colors.grey[700])),  
                                            ],  
                                        ),  
                                    ),  
                                    Flexible(  
                                        flex: 1,  
                                        child: ElevatedButton(  
                                            onPressed: () {  
                                                ScaffoldMessenger.of(context).showSnackBar(  
                                            },  
                                        ),  
                                    ),  
                                ],  
                            ),  
                        ),  
                    );  
                },  
            ),  
        );  
    }  
}
```



Explicación del ejemplo

1. Scaffold → estructura base con AppBar, body y FloatingActionButton.
 2. ListView.builder → crea una lista de productos dinámica.
 3. Card → cada producto se muestra en una tarjeta con sombra.
 4. Row + Flexible →
 - Primer Flexible (3 partes) → para texto y detalles del producto.
 - Segundo Flexible (1 parte) → para el botón "Añadir".
 - Esto asegura que el diseño sea responsive.
 5. FloatingActionButton → acción principal → acceder al carrito.
 6. Snackbar → feedback inmediato cuando se pulsa un botón.

7. CUSTOM SCAFFOLD EN FLUTTER

◆ ¿Qué es un Custom Scaffold?

En Flutter, el widget Scaffold es la base para estructurar una pantalla: contiene la barra superior (AppBar), el contenido principal (body), botones flotantes (FAB), menús laterales y más.

Un Custom Scaffold es simplemente una versión personalizada del Scaffold que encapsula un diseño repetitivo en tu aplicación.

En lugar de repetir la misma configuración en todas las pantallas (colores, AppBar, pie de página, etc.), creamos un widget propio que extiende de StatelessWidget y usa un Scaffold en su interior.

◆ ¿Por qué merece la pena usarlo?

- Consistencia visual → Todas las pantallas comparten la misma estructura y estilo.
- Menos código repetido → Definimos el AppBar, Footer, colores o temas una sola vez.
- Escalabilidad → Si necesitas cambiar algo (ejemplo: logo, color del AppBar, mensaje del footer), lo haces en un solo lugar y se actualiza en toda la app.
- Mantenibilidad → El código es más limpio y las pantallas se enfocan solo en su contenido (body).

◆ Ejemplo: CustomScaffold

- Logo en el AppBar.
- Opciones dinámicas (actions, leading).
- Pie de página fijo con copyright.

```
import 'package:flutter/material.dart';
import 'package:juego_memoria/widgets/logo.dart';

class CustomScaffold extends StatelessWidget {
    final List<Widget>? actions;
    final Widget body;
    final Widget? floatingActionButton;
    final Widget? leading;

    const CustomScaffold({
        super.key,
        this.actions,
        required this.body,
        this.floatingActionButton,
        this.leading,
    });

    @override
    Widget build(BuildContext context) {
        final theme = Theme.of(context);

        return Scaffold(
            appBar: AppBar(
                title: const Logo(), // Widget de Logo centralizado
                centerTitle: true,
                backgroundColor: theme.colorScheme.primary,
                elevation: 0,
                actions: actions, // Botones extra (ej. perfil, Logout)
                leading: leading, // Botón en la izquierda (ej. menú Lateral)
                iconTheme: IconThemeData(color: theme.colorScheme.onPrimary),
            ),
            body: body, // Contenido principal dinámico
            floatingActionButton: floatingActionButton, // Botón flotante opcional
            bottomNavigationBar: Container(
                height: 50,
                color: theme.colorScheme.primary.withOpacity(0.1),
                child: Center(

```

```
        child: Text(
            'Copyright 2025 - Juego de Memoria',
            style: TextStyle(
                color: theme.colorScheme.onSurface.withOpacity(0.7),
                fontSize: 12,
            ),
        ),
    ),
),
),
),
);
}
}
```

◆ Uso en una pantalla

En lugar de escribir el Scaffold manualmente en cada pantalla, ahora usamos CustomScaffold y pasamos solo lo necesario:

```
class HomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return CustomScaffold(
      body: Center(
        child: Text(
          'Bienvenido al Juego de Memoria',
          style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
        ),
      ),
      actions: [
        IconButton(
          icon: Icon(Icons.info),
          onPressed: () {
            ScaffoldMessenger.of(context).showSnackBar(
              SnackBar(content: Text('Versión 1.0 del juego')),
            );
          },
        ),
      ],
      floatingActionButton: FloatingActionButton(
        onPressed: () {},
        child: Icon(Icons.play_arrow),
      ),
    );
  }
}
```

◆ Explicación paso a paso

1. Definimos parámetros flexibles (actions, body, leading, FAB) → permiten personalizar cada pantalla sin perder la estructura.
 2. Centralizamos el estilo → el AppBar siempre tendrá el logo, colores y tipografía predefinidos.
 3. Añadimos un footer con un mensaje común a toda la app.

4. El body cambia según la pantalla, pero todo lo demás se mantiene.
LayoutBuilder → Similar a MediaQuery, pero se ajusta dinámicamente cuando cambia el espacio disponible.

8. RECURSOS RECOMENDADOS PARA APRENDER FLUTTER

Documentación Oficial

- Flutter Docs
 <https://docs.flutter.dev>
- Dart Language
 <https://dart.dev/language>

Cursos Gratuitos

1. Flutter Crash Course (Google)
 <https://docs.flutter.dev/get-started/codelab>

Paquetes y bibliotecas

- Pub.dev (Repositorio Oficial)
 <https://pub.dev>
- Flutter Awesome (Inspiración UI)
 <https://flutterawesome.com>

Libros

1. "Flutter in Action" (Manning)
 <https://www.manning.com/books/flutter-in-action>
2. "Dart Apprentice" (Ray Wenderlich)
 <https://www.raywenderlich.com/books/dart-apprentice>

Comunidad

- Stack Overflow (Flutter Tag)
 <https://stackoverflow.com/questions/tagged/flutter>
- Reddit r/FlutterDev
 <https://www.reddit.com/r/FlutterDev>
- Flutter Community en Medium
 <https://medium.com/flutter-community>

Herramientas Clave

1. Flutter DevTools (Debugging)
 <https://docs.flutter.dev/tools/devtools>
2. Firebase para Flutter
 <https://firebase.flutter.dev>
3. Riverpod (Gestión de Estado)
 <https://riverpod.dev>
4. VS Code + Extensión Flutter
 <https://marketplace.visualstudio.com/items?itemName=Dart-Code.flutter>

Extra: Proyectos Open-Source

- Repositorio Oficial de Flutter
 <https://github.com/flutter/flutter>
- Ejemplos de Apps en GitHub
 <https://github.com/flutter/samples>