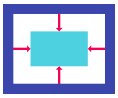
Flutter
Guia de atajos

{dev/talles}

Widgets
Posicionamiento y agrupadores

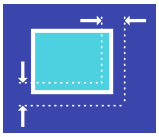


Center
Centra su hijo en relación al widget padre.

```
Center( child: Text('Widget') );
```

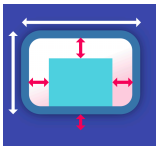
Align
Alinea su hijo dentro de si mismo,

```
Align(
  alignment: Alignment.topRight,
  child: FlutterLogo()
);
```



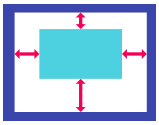
ConstrainedBox
Impone reglas adicionales de tamaño, no ser más grande de ciertas dimensiones, o más pequeño de ciertas dimensiones o que tome cierto espacio.

```
ConstrainedBox(
  constraints: const BoxConstraints.expand(),
  child: const Card(
    child: Text('Hello World!')
  ),
 );
```



Container
Un widget común que combina pintado personalizado, posicionamiento, padding y tamaño en uno sólo.

```
Center(
  child: Container(
     margin: const EdgeInsets.all(10.0),
     color: Colors.amber[600],
     width: 48.0,
     height: 48.0,
    ),
  );
```

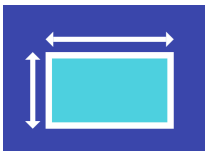


Padding
Este widget añade un relleno (padding), que permite al hijo acomodarse a ese nuevo espacio.

```
Padding(
   padding: EdgeInsets.all(16.0),
   child: Text('Hello World!'),
);
```

Transform
Permite realizar transformaciones, rotaciones y demás mutaciones visuales a su child.

```
Transform.scale(
    scale: 1.2,
    child: Text('Woohoo!'),
);
```



SizedBox
Una caja con un tamaño especifico, que también fuerza a su child a que respete su tamaño impuesto.

```
SizedBox(
   width: 200.0,
   height: 300.0,
   child: Container(),
 );
```

Widget con Múltiples hijos (Children)

Column
Ordena sus hijos de forma vertical, a la vez permite alinearlos internamente en su eje principal o secundario.

```
Column(
  children: const <Widget>[
     Text('Deliver features faster'),
     Text('Craft beautiful UIs'),
     Expanded(
       child: FittedBox(
         child: FlutterLogo(),
       ),
     ),
   ],
  );
```

Row
Similar al column, pero de forma horizontal, también permite alinear sus hijos en el eje principal y secundario.

```
Row(
  children: const <Widget>[
     FlutterLogo(),
      Text("Hot reload!!"),
       Icon(Icons.sentiment_very_satisfied),
    ],
  );
```

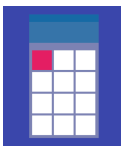
Flutter
Guia de atajos

{dev/talles}

Stack

Permite colocar sus hijos (Widgets) unos sobre otros y montarse básicamente y a la vez, se pueden alinear y posicionarse en relación al espacio que les da el padre.

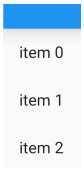
```
Stack(
   children: <Widget>[
      Container(
        width: 100,
        height: 100,
        color: Colors.red,
      ),
      Container(
        width: 90,
        height: 90,
        color: Colors.green,
      ),
    ],
  );
```



GridView

Un listado de widgets que siguen un patrón repetitivo de celdas de forma vertical.

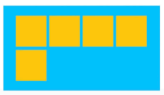
```
GridView.count(
    crossAxisCount: 2,
    children: <Widget>[
      Container(
        padding: const EdgeInsets.all(8),
        color: Colors.teal[100],
        child: const Text("Hi there!"),
      ),
      Container(
        padding: const EdgeInsets.all(8),
        color: Colors.teal[600],
        child: const Text('Revolution, they...'),
      ),
    ],
  );
```



ListView

Es una lista que permite scroll de Widgets que estarán alineados de forma lineal. Es un widget muy común para mostrar listados de widgets.

```
ListView(
   padding: const EdgeInsets.all(8),
   children: <Widget>[
      Container(),
      Container(),
      Container(),
   ],
 );
```



Wrap

Similar al Column y Row, pero permite ajustar sus hijos de forma vertical y horizontal acorde al número de hijos.

```
Wrap(
    children: <Widget>[
      Container(),
      Container(),
      Container(),
    ],
  );
```

Table

Permite ordenar sus hijos en un formato de tabla, filas y columnas de forma estricta.

```
Table(
    children: <TableRow>[
      TableRow(
        children: <Widget>[
          Container(
            height: 32,
            color: Colors.green,
          ),
          TableCell(
            child: Container(
              height: 32,
              width: 32,
              color: Colors.red,
            ),
          ),
          Container(
            height: 64,
            color: Colors.blue,
          ),
        ],
      )
    ]
  );
```

SingleChildScrollView

A diferencia de los anteriores, acepta un único hijo cuyo tamaño puede ser superior al tamaño de la pantalla del cliente, si ese es el caso, permite realizar el scroll de su contenido.

```
SingleChildScrollView(
      child: SizedBox(
        height: 4500,
        child: Container(),
      ),
    );
```

CustomScrollView

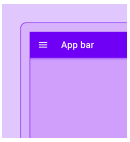
Similar al widget anterior, pero este está especializado a trabajar con Widgets que tienen el nombre de Slivers, los cuales están estrechamente relacionados a la posición del scroll para realizar su comportamiento visual requerido.

Flutter - Catálogo de Widgets

Flutter
Guia de atajos

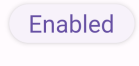
{dev/talles}

Widgets Comunes



AppBar
Este widget sigue el diseño de Material Design para crear una barra superior de herramientas.

```
AppBar(
   title: const Text('AppBar Demo'),
   actions: <Widget>[]
 );
```



Buttons
Flutter tiene muchos widgets que generan botones de diferentes tipos y formas.

Estos son los botones comunes:

```
ElevatedButton(
  onPressed: null,
  child: const Text('Elevated')),

FilledButton(
  onPressed: null,
  child: const Text('Filled')),

FilledButton.tonal(
  onPressed: null,
  child: const Text('Filled Tonal')),

OutlinedButton(
  onPressed: null,
  child: const Text('Outlined')),

TextButton(
  onPressed: null,
  child: const Text('Text Button')),

// Y sus variaciones con íconos
 ElevatedButton.icon(
   onPressed: () {},
   icon: const Icon(Icons.add_a_photo_outlined),
   label: const Text('Photo')),
```

Icon
Permite mostrar un ícono de material, se pueden usar otros iconos con paquetes externos. Aquí está el listado con buscador.

```
Icon(
    Icons.audiotrack,
    color: Colors.green,
    size: 30.0,
  );
```

Image
Diseñado para desplegar y mostrar imágenes, puedes obtenerla desde los assets, urls, bitarrays y archivos del dispositivo.

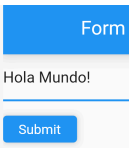
```
Image(
  image: NetworkImage(
    'https://flutter.github.io/assets-for-api-
docs/assets/widgets/owl.jpg'),
);
```

Scaffold
Implementa un diseño básico de material, y da las bases para colocar un menú lateral, snack-bars, appbars, bottom sheets y más elementos.

```
Scaffold(
  appBar: AppBar(
    title: const Text('Sample Code'),
   ),
   body: const Center(child: Text('5 clicks!')),
   floatingActionButton: FloatingActionButton(
     onPressed: (){},
     child: const Icon(Icons.add),
   ),
 );
```


Text
Diseñado para mostrar una sola línea de texto con un estilo específico.

```
Text(
 'Hello, $_name! How are you?',
 textAlign: TextAlign.center,
 overflow: TextOverflow.ellipsis,
 style: const TextStyle(fontWeight: FontWeight.bold),
);
```



Form y FormField
Un tipo de contenedor que da el seguimiento a FormFields y permite manejar las validaciones.

Pueden ver el ejemplo en la documentación aquí.

Theme
Permite aplicar un tema de estilos a todos los widgets hijos y sus descendientes.

Flutter cuenta con cientos de Widgets, es difícil agruparlos todos en una hoja pero les recomiendo esta playlist para conocerlos con ejemplos visuales.

Flutter Widget of the Week
Widget de la semana - Fernando Herrera
Catálogo de Widgets agrupados

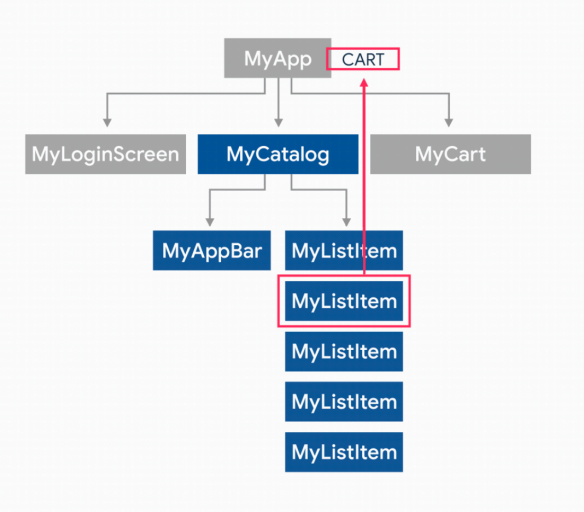
Flutter - Catálogo de Widgets

Flutter
Guia de atajos

{dev/talles}

BuildContext

Se puede ver como el objeto que tiene la referencia a cada widget en el árbol de Widgets.



Flutter CLI

Esta es la lista de algunos comandos útiles del CLI

CMD	Ejemplo	Descripción
build	`flutter build <DIRECTORY>`	Construye la aplicación
channel	`flutter channel <CHANNEL_NAME>`	Enumera o cambia los canales.
clean	`flutter clean`	Borra los `build/` y `.dart_tool/` directorios
create	`flutter create <DIRECTORY>`	Crea un nuevo proyecto
custom-devices	`flutter custom-devices list`	Añade, elimina, lista y resetea los dispositivos.
devices	`flutter devices -d <DEVICE_ID>`	Lista los dispositivos conectados
doctor	`flutter doctor`	Herramienta de auto-consulta y diagnostico.
downgrade	`flutter downgrade`	Hace un downgrade de Flutter a la versión anterior
drive	`flutter drive`	Ejecuta pruebas de Flutter Driver para el proyecto actual.
emulators	`flutter emulators`	Lista, lanza y crea emuladores.
format	`flutter format <DIRECTORY|DART_FILE>`	NO USAR - Formatea el código. Usar este en su lugar `dart format`.
gen-l10n	`flutter gen-l10n <DIRECTORY>`	Genera las localizations del proyecto de Flutter.
install	`flutter install -d <DEVICE_ID>`	Instala la aplicación en un dispositivo conectado.
logs	`flutter logs`	Muestra los logs de una aplicación corriendo.
run	`flutter run <DART_FILE>`	Ejecuta la aplicación.
screenshot	`flutter screenshot`	Toma una fotografía del dispositivo corriendo.
test	`flutter test [<DIRECTORY|DART_FILE>]`	Corre las pruebas de este paquete. Usar este en su lugar `dart test`.
upgrade	`flutter upgrade`	Actualiza a la siguiente versión de Flutter

Navegar a otra pantalla

Hay muchas formas de navegar a diferentes pantallas en Flutter, pero en general esta es la nativa:

```
onPressed: () {
 Navigator.push(
   context,
   MaterialPageRoute(
   builder: (context) => const SecondRoute()),
 );
}
```

Gestores de estado

Manejar el estado de la aplicación es algo de nivel intermedio, hay muchas formas de manejarlo, desde third-party y first-party.
Esta es la lista oficial de opciones según Flutter.dev.

A continuación una breve descripción sobre los más populares. **(En base a experiencia personal)**

Provider: Es el recomendado por Flutter, en lo personal lo recomiendo para empezar, es robusto, fácil de aprender y usar. **(Flutter Favorite)**

Riverpod: Es otro muy bueno y recomendado, fácilmente testeable.

InheritedWidget & InheritedModel: Esta es la opción nativa de Flutter, no dependencias de terceros, algo tediosa a mi parecer, pero es la solución oficial.

Flutter - Catálogo de Widgets

Flutter
Guia de atajos

{dev/talles}

BLoC / RX y Flutter BLoC + Cubits: Es un gestor de estado basado en Streams y Observables, no es el más fácil para empezar, pero da mucho control. A muchas personas no les gusta por que se escribe mucho código. (Flutter Farorite)

Get_it: Técnicamente no es un gestor de estado, pero es muy útil para ir puntualmente a un servicio u objeto sin necesidad del BuildContext.

MobX: Es popular y basado en observables y reacciones. **(Flutter Favorite)**

GetX: Actualmente el paquete y gestor de estado más popular, hace muchas cosas aparte de manejar estados, tiene navegación, snackbars, internacionalización, temas, validaciones. Es considerado un Framework dentro de Flutter.
Yo en lo personal **NO** lo recomiendo para aprender primero ya que oculta conceptos que son necesarios aprender (Como el BuildContext), pero es muy poderoso.

Pro Tip:
Prueben varios gestores de estado, miren cuál es el que más les gusta, sirve y resuelve su necesidad. Creen su propia opinion sobre ellos y escuchen a la comunidad, pero al final del día, ustedes tienen la última palabra, **NADIE** de la comunidad le dará mantenimiento a tu código, por eso deben de saber cuál es mejor para ti.

Glosario de palabras clave

Widget
Inspirado de los componentes de React, los widgets son como piezas de lego que son usadas para construir aplicaciones.

Child Widget
Un child widget no es más que cualquier otro Widget de Flutter, la mayoría de Widgets en Flutter aceptan un child widget, y así se va creando el árbol de Widgets

Context - BuildContext
El context es un enlace a la ubicación de widgets en el árbol de Widgets. El “context” es el nombre corto de BuildContext, el cual es pasado de Widget en Widget dejando su marca y posición.

Estado - Gestor de Estado
Son los valores actuales de las variables, clases y objetos modificados por la aplicación y el usuario.

Dash:
Es el nombre de la mascota de Flutter, es niña y originalmente era para Dart. Historia aquí

Stateless Widget
Es una pieza de lego que se construye muy rápido y no mantiene el estado por si mismo. Flutter sabe cuando se de be de volver a dibujar, es recomendado que en lugar de crear funciones y métodos que retornan Widget, es mejor crear Clases que extiendan de estos StatelesWidgets.

Stateful Widget
Es similar al stateless en cuanto a que es un Widget, pero este permite mantener un estado interno y ciclo de vida como su inicialización y destrucción. Muchos lo tachan de que jamás se deben de usar pero eso no es cierto, los stateful básicamente son el corazón de cualquier animación que suceda.

HotReload
Quizá lo más impresionante de Flutter es la capacidad que tiene para mantener el estado en pantalla mientras cambia la apariencia y estructura de los nuevos widgets en caliente. Es decir, no hay que bajar la aplicación y volverla a subir para ver los cambios reflejados en pantalla.

Hot Restart - Full-restart
Es el término que se usa para decir que hay que bajar la aplicación y volverla a subir, puede ser de dos formas, reiniciar toda la aplicación levemente o bajarla completamente y volverla a montar. Este proceso toma más tiempo que el HotReload.

Main
Toda aplicación de Dart, tiene una función main que puede ser asíncrona, todo inicia en este punto y desde ahí se desprende toda la aplicación de Flutter.

Método Build de los Widgets
Todo widget tiene un método build, el cual es lo que se ejecuta para construirlo, aquí se pueden definir variables, leer el contexto y definir las condiciones de construcción.
En caso de Stateful widgets, el build context está disponible de forma global (heredada) dentro de la subclase State.

CheatSheet - Última versión
Si esta guía te sirve o sirvió, por favor compártela con todo el mundo, eso nos ayudaría mucho!

Flutter - Catálogo de Widgets