

# Flutter ADI Documentación

**Rafael Vicente Llorca Cortes**

**Dario Guerrero Montero**

**Dario Zorrilla Berenguer**

**Jordi Cardona Merino**

<b>Introducción</b>	<b>3</b>
<b>Origen de Flutter</b>	<b>3</b>
<b>Comparativa con otros frameworks de uso más comunes</b>	<b>4</b>
<b>Fuentes</b>	<b>8</b>

# Introducción

Flutter es un framework multiplataforma gratuito creado por Google que permite crear una aplicación móvil nativa con una sola base de código.

Flutter utiliza Dart, un lenguaje orientado a objetos también desarrollado por Google que se lanzó como alternativa a JavaScript con la intención de reemplazarlo aunque todavía se encuentra en proceso de mejora.

Con Dart, las aplicaciones se pueden compilar directamente en código máquina nativo lo que permite que Flutter consuma menos recursos del dispositivo, es decir, un mejor rendimiento y un inicio más rápido de las aplicaciones.

Flutter cuenta con una enorme cantidad de funcionalidades sin necesidad de añadir dependencias lo que reduce la necesidad de programar funcionalidades específicas en nativo.

Cuenta con un amplio catálogo de widgets para utilizar interfaces modernas y transiciones fluidas aunque también permite la creación de Widgets propios y modificar los existentes de forma que tienes un control total de la interfaz.

Además, Flutter busca el desarrollo rápido haciendo uso del hot reload, lo que permite experimentar con el código, modificar UIs, funcionalidades y corregir bugs con pocos tiempos de recarga y sin perder el estado de los emuladores, simuladores o dispositivos para iOS y Android

# Origen de Flutter

Creado en 2015 pero lanzado en mayo de 2017, fue primeramente conocido como sky en android en la cumbre de desarrolladores de Dart declarando que era capaz de hacer render a 120 fotogramas por segundo.

Más tarde, en septiembre de 2018, Google anunció su lanzamiento de pre-release, esta sería la última antes de su lanzamiento completo con la 1.0, la cual fue lanzada el 4 de diciembre de ese mismo año.

En mayo de 2020 se lanzó la versión 2.8 del SDK de Dart y la versión 1.17 de flutter, esta versión venía con soporte para "meta" un api que mejora el rendimiento en dispositivos iOS entre otras cosas.

En marzo de 2021 se anunció la versión 2.0 de flutter la cual trae soporte oficial para aplicaciones basadas en la web con nuevo renderizador , widgets y apis mejoradas. Al hacer uso de Dart 2.0 el soporte de flutter tuvo que hacer cambios y añadir instrucciones y herramientas para ayudar con los problemas que surgían de esta versión en seguridad.

Con la versión 2.5 de flutter , se mejoró el modo de pantalla completa de android y el añadido de "material you" la cual es la última iteración de Material Design.

Con la versión 3.0 se añadió la admisión de hasta 6 plataformas , más tarde se añadió la interoperabilidad de objective C y swift

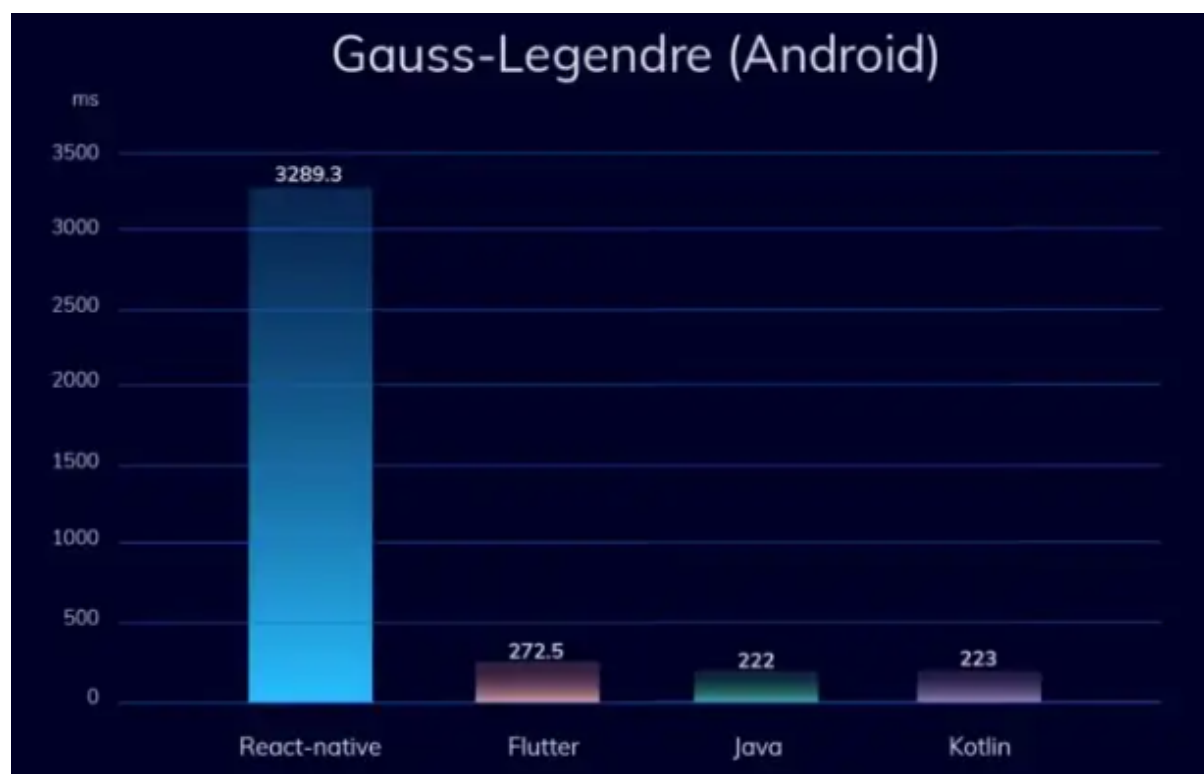
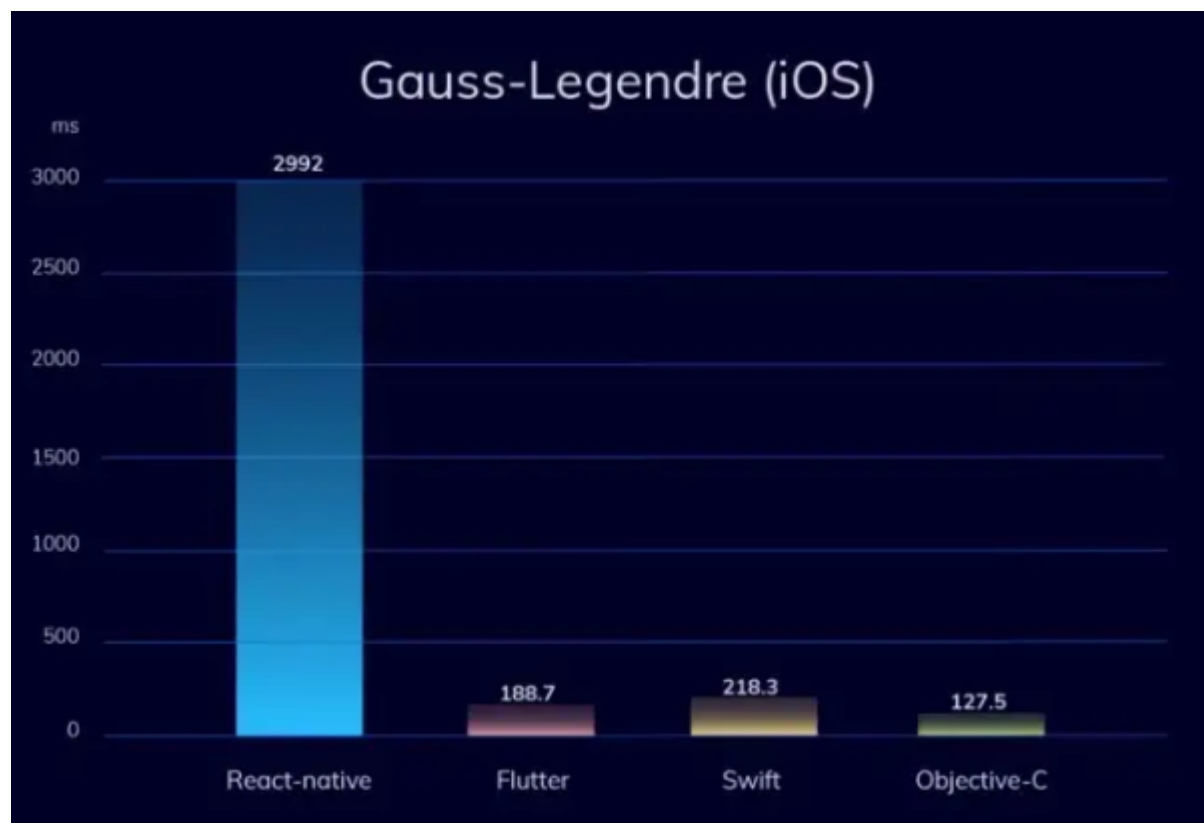
## Comparativa con otros frameworks de uso más comunes

El aprendizaje de flutter no es difícil, la cantidad de apoyo por parte de su página oficial es lo suficientemente grande como para garantizar un aprendizaje rápido y sencillo.

Debido a fallas que tuvieron con Dart, este se considera menos seguro que sus competidores, además de que tiene un sdk no tan amplio como el que ofrece IOS. En cambio , gracias a Google este se puede desarrollar en android studio, visual studio o intel idj además de hacer precompilados son necesidad de hacer un build para cada cambio menor por lo que para proyectos grandes viene muy bien.

El tamaño de lo desarrollado no es algo pequeño (5MB mínimo) debido a los asserts y frameworks que este necesita ya que no son nativos y son necesarios.

En cuanto al testing, tiene una gran cantidad de benchmarks por lo que se puede comparar sin necesidad de hacer todo el testeo uno mismo.



# Ejemplo contador

Para el primer ejemplo realizaremos un contador simple.

```
1  import 'package:flutter/material.dart';
2
3  void main() {
4    runApp(MyApp());
5  }
6
7  class MyApp extends StatefulWidget {
8    @override
9    _MyAppState createState() => _MyAppState();
10 }
11
12 class _MyAppState extends State<MyApp> {
13   int _counter = 0;
14
15   void _incrementCounter() {
16     setState(() {
17       _counter++;
18     });
19   }
20
21   @override
22   Widget build(BuildContext context) {
23     return MaterialApp(
24       home: Scaffold(
25         appBar: AppBar(
26           title: const Text('Contador'),
27         ),
28         body: Center(
29           child: Column(
30             mainAxisAlignment: MainAxisAlignment.center,
31             children: <Widget>[
32               const Text('Has presionado el botón este número de veces:'),
33               Text(
34                 '$_counter',
35                 style: Theme.of(context).textTheme.headline4,
36               ),
37             ],
38           ),
39         ),
40         floatingActionButton: FloatingActionButton(
41           onPressed: _incrementCounter,
42           tooltip: 'Incrementar',
43           child: const Icon(Icons.add),
44         ),
45       ),
46     );
47   }
48 }
```

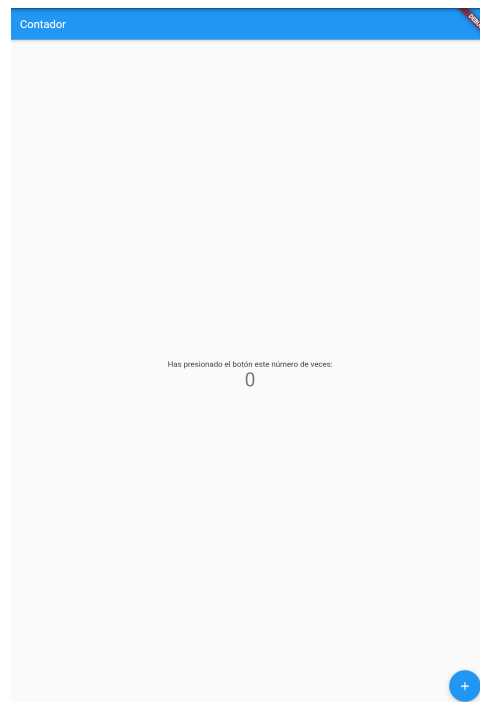
Para la realización del contador primero crearemos el main que llamara a la clase que hereda de la clase StatefulWidget la cual es la estructura base sobre la que llamaremos a la función de creación de nuestra interfaz. Para ello llamaremos a otra clase que heredará a su vez de la clase State<Myapp> Por tanto generamos un estado que tiene una función de construcción que generará el widget.

El widget devuelve la app de manera que dentro de la app rellenaremos los campos como el appBar que será nuestra barra superior y nuestro body que será el esqueleto.

En el body tendremos los diferentes elementos como sucesores del body, tiene una jerarquía de descendencia de los elementos, por último se agregaría un elemento externo **"FloatingActionButton"** como el botón de suma el cual llamará a la función de suma que creamos.

Por último en nuestra clase de la app tendremos una variable que se encargará de almacenar el contador llamada **\_counter**.

En nuestra función de suma utilizaremos el setState para indicar que el estado ha cambiado y debe actualizar la interfaz.



## Lista de la compra

Hemos realizado una demo en Flutter de la lista de la compra que hemos realizado durante este curso. Es una lista un tanto más sencilla, pero sirve para poder enseñaros cómo funciona flutter.

```

return Scaffold(
  appBar: AppBar(
    title: Row(children: [
      Expanded(
        child: TextField(
          controller: textController,
          style: const TextStyle(color: Colors.white),
          decoration: const InputDecoration(
            hintStyle: TextStyle(color: Colors.white),
            hintText: 'Añadir nuevo elemento',
            border: OutlineInputBorder(), // InputDecoration
          ), // TextField // Expanded
        ),
        child: ElevatedButton(
          onPressed: () {
            setState(() {
              if (groceryIndex != null) {
                groceries[groceryIndex!] = textController.text;
                groceryIndex = null;
                buttonIcon = Icons.add;
              } else {
                groceries.add(textController.text);
                comprado.add(false);
              }
              textController.clear();
            });
          },
          child: Icon(buttonIcon) // ElevatedButton
        ), // Row
      ), // AppBar
    ),
  ),
);

```

En la primera parte de la aplicación, tenemos el `appBar`, que es la barra de la parte superior de nuestra app. En esta barra, hemos añadido un *input* de texto donde podremos añadir o editar elementos de nuestra lista de la compra. Si vemos con más detalle el *ElevatedButton*, podemos ver que tenemos dos casos. El primero, representa el caso en el que queremos editar un elemento ya existente en la lista. Para ello, almacenamos el índice en el que se encuentra el elemento y lo modificamos. Una vez que se haya modificado, cambiamos el icono para que vuelva a ser un +. El segundo caso es añadir un nuevo elemento, que simplemente será añadir el texto introducido en el input a nuestro array de *groceries*.

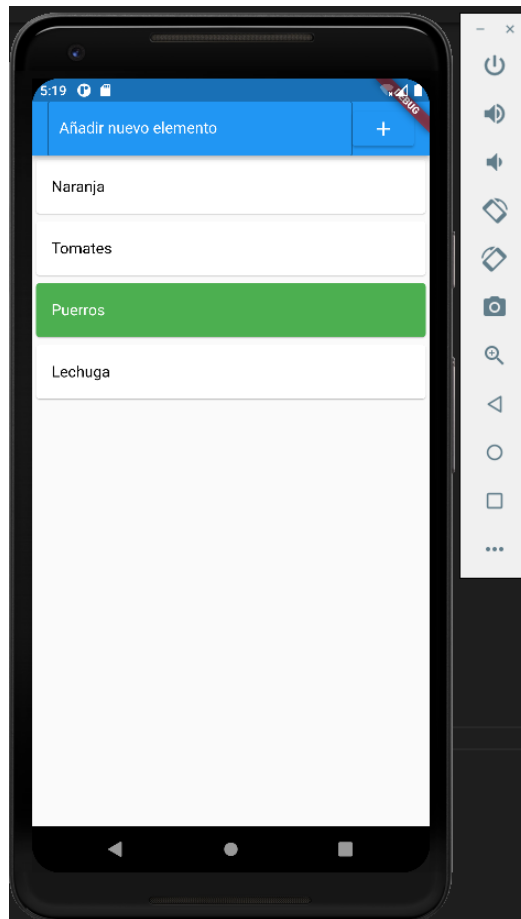


```

body: ListView.builder(
  itemCount: groceries.length,
  itemBuilder: (BuildContext context, int index) {
    return GestureDetector(
      onTap: () {
        groceryIndex = index;
        setState(() {
          textController.text = groceries[index];
          buttonIcon = Icons.save;
        });
      },
      child: Card(
        color: comprado[index] ? Colors.green : Colors.white,
        child: ListTile(
          title: Text(
            groceries[index],
            style: TextStyle(
              color: comprado[index] ? Colors.white : Colors.black), // TextStyle
          ), // Text
          onTap: () {
            setState(() {
              comprado[index] = !comprado[index];
            });
          },
          onLongPress: () {
            setState(() {
              groceries.removeAt(index);
            });
          },
        ), // ListTile // Card
      ), // GestureDetector
    ); // ListView.builder // Scaffold
  },
); // ListView.builder // Scaffold



```

Después, en el cuerpo, podemos ver como se renderiza la lista de la compra. Utilizamos un **ListView**, que nos permite crear una lista de elementos con un mismo formato. El equivalente en Vue es cuando hacemos un *v-for* de un array o lista. Le indicamos el número de elementos y el constructor que es el que se encarga de crear la estructura para cada uno de los elementos del array. A continuación, podemos ver el widget *GestureDetector*. Este widget nos permite capturar gestos que ocurren en ciertas partes de la pantalla. En este caso, cada una de las tarjetas captura el gesto *double tap* para poder editar el elemento seleccionado. Finalmente, cada tarjeta incluye un elemento de nuestra lista y un estilo dinámico en caso de que el elemento haya sido ya comprado o no. También, contamos con otros gestos que nos permiten cambiar el estado de un elemento o eliminarlos.



## Fuentes

- [▶ Reflectly App \(Flutter Developer Story\)](#)
- [▶ ¿Qué es Flutter? | Historia de Flutter](#)
- [https://profile.es/blog/que-es-flutter-sdk/#:~:text=Creado%20por%20Google%2C%20se%20presentó.aplicaciones%20híbridas\)%20con%20rendimiento%20nativo.](https://profile.es/blog/que-es-flutter-sdk/#:~:text=Creado%20por%20Google%2C%20se%20presentó.aplicaciones%20híbridas)%20con%20rendimiento%20nativo.)
- [https://es.wikipedia.org/wiki/Flutter\\_\(software\)](https://es.wikipedia.org/wiki/Flutter_(software))
- <https://medium.com/comunidad-flutter/lo-revolucionario-de-flutter-425d4e43de8d>
- [▶ Flutter vs Native iOS](#)
- <https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980>
- <https://medium.com/flutter-community/flutter-understanding-counter-app-ca89de564170>
- <https://esflutter.dev/>
- <https://flutter.dev/>
- <https://www.hiberus.com/crecemos-contigo/que-es-el-lenguaje-de-programacion-dart/>
- <https://talently.tech/blog/que-es-flutter/>
- [▶ Flutter App From Scratch: Grocery list App \(Episode 1, Hello World!\)](#)
- [▶ GestureDetector \(Widget of the Week\)](#)

-  Theme (Flutter Hallowidget of the Week)
-  MediaQuery (Flutter Widget of the Week)