Contents

- Background
- Specific Problems
- Use Case
- Real World Examples
- How it Works
- Code Snippet
Background

Flutter is Google’s UI toolkit for building beautiful, natively compiled applications for mobile, web, and desktop from a single codebase.

Flutter apps are written in **Dart language** and make use of the language's advanced features.

Written primarily in C++, provides **low level rendering** support using Google's graphics library.

Written in Dart, provides basic **classes and functions** which are used to construct applications.

Material Design widgets implement Google's design language, and Cupertino widgets implement iOS design.
Specific Problems & Use Case

- Fast Development
- Expressive, Beautiful UIs
- Native Performance
Specific Problems & Use Case

Fast Development
Expressive, Beautiful UIs
Native Performance

Flutter's **hot reload** helps you quickly and easily experiment, build UIs, add features, and fix bugs faster.
Specific Problems & Use Case

Fast Development

Expressive, Beautiful UIs

Native Performance

Flutter's built-in beautiful Material Design and Cupertino (iOS-flavor) widgets, rich motion APIs, smooth natural scrolling, and platform awareness.

Reflectly
An award winning mindfulness app built with Flutter.

Download: iOS, Android
Learn more
Specific Problems & Use Case

- **Fast Development**
- **Expressive, Beautiful UIs**
- **Native Performance**

Flutter’s widgets incorporate all critical platform differences such as scrolling, navigation, icons and fonts to provide full native performance on both iOS and Android.
Real World Examples
How it Works

| Platform | Engine | Framework | Rendering Pipeline |

Starting at the platform level

Flutter provides a **Shell**, that hosts the **Dart VM**.

Shell gives access to the native platform APIs.

Shell hosts the establishing platform and relevant canvas.
How it Works

<table>
<thead>
<tr>
<th>Platform</th>
<th>Engine</th>
<th>Framework</th>
<th>Rendering Pipeline</th>
</tr>
</thead>
</table>

The engine is the next layer up

Provides Dart Runtime

Provides Skia

Provides Platform Channels
How it Works

Framework

Dart

Framework is the most relevant to the developer. It contains everything you will interact with, when developing your app.

Themes

Cupertino

Material

Widgets

Rendering

Painting

Gestures

Foundation

Engine

C/C++

Skia

Dart Runtime

Platform Channels

And more...

Platform

iOS Shell

Android Shell

Embedder API
Flutter works more like a gaming engine. The UI is built and rendered on a Skia Canvas as it changes. Flutter updates the UI at 60fps, and uses the GPU for most of the work.

App is composed of Widgets, that are rendered onto a Skia canvas.

The platform shows the canvas, and sends events back as required.
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        // This is the theme of your application.
        // Try running your application with "flutter run". You'll see the
        // application has a blue toolbar. Then, without quitting the app, try
        // changing the primarySwatch below to Colors.green and then invoke
        // "hot reload" (press "r" in the console where you ran "flutter run",
        // or simply save your changes to "hot reload" in a Flutter IDE).
        // Notice that the counter didn't reset back to zero; the application
        // is not restarted.
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}

class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  // This widget is the home page of your application. It is
  // stateful, meaning
  // that it has a State object (defined below) that
  // contains fields that affect
  // how it looks.

  // This class is the configuration for the state. It holds
  // the values (in this
  // case the title) provided by the parent (in this case the
  // App widget) and
  // used by the build method of the State. Fields in a
  // Widget subclass are
  // always marked "final".

  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {

}
class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    // This method is rerun every time setState is called, for instance as done
    // by the _incrementCounter method above.
    return Scaffold(
      appBar: AppBar(
        // Here we take the value from the MyHomePage object that was created by
        // the App.build method, and use it to set our
        appBar title.
        title: Text(widget.title),
      ),
      body: Center(
        // Center is a layout widget. It takes a single child and positions it
        // in the middle of the parent.
        child: Column(
          // Column is also a layout widget. It takes a list of children and
          // arranges them vertically. By default, it sizes itself to fit its
          // children horizontally, and tries to be as tall as its parent.
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'You have pushed the button this many times:
            ),
            Text(
              '${_counter}',
            style: Theme.of(context).textTheme.display1,
            ),
          ],
        ),
        floatingActionButton: FloatingActionButton(
          onPressed: _incrementCounter,
          tooltip: 'Increment',
          child: Icon(Icons.add),
        ),
      ),
    );
  }
}