ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

502071

Cross-Platform Mobile Application Development

Getting Started to Flutter

1

Introduction to Flutter

Introduction to Flutter

- Flutter is an open source framework by Google for building beautiful, natively compiled, multi-platform applications from a single codebase.
- Flutter features:
  - **Fast**: Flutter code compiles to ARM or Intel machine code as well as JavaScript, for fast performance on any device.
  - **Productive**: Build and iterate quickly with **Hot Reload**. Update code and see changes almost instantly, without losing state.
  - **Flexible**: Control every pixel to create customized, adaptive designs that look and feel great on any screen.

Introduction to Flutter

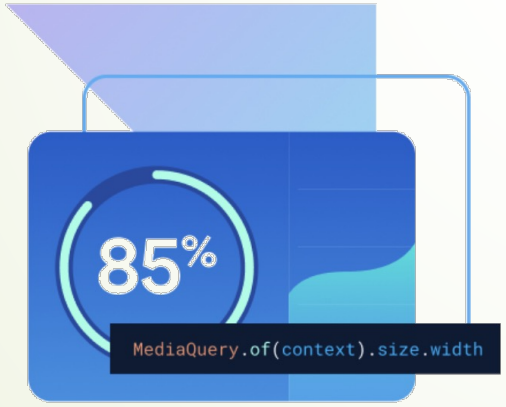
- **Multi-Platform:** Deploy to multiple devices from a single codebase: mobile, web, desktop, and embedded devices.
- **Stable & Reliable**: Flutter is supported and used by Google, trusted by well-known brands around the world, and maintained by a community of global developers.
- **Dart**: Flutter is powered by Dart, a language optimized for fast apps on any platform

Introduction to Flutter

- **Beautiful apps for every screen:** Flutter allows you to build apps for mobile, web, desktop, and embedded devices — all from a single codebase.

Single codebase

Maintain one codebase and deploy to multiple platforms, speeding up and simplifying workflows.

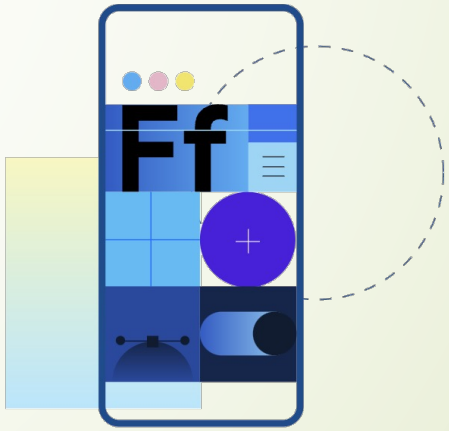


Performant by design

Flutter gives you the power of hardware-accelerated graphics for performant apps on any platform.

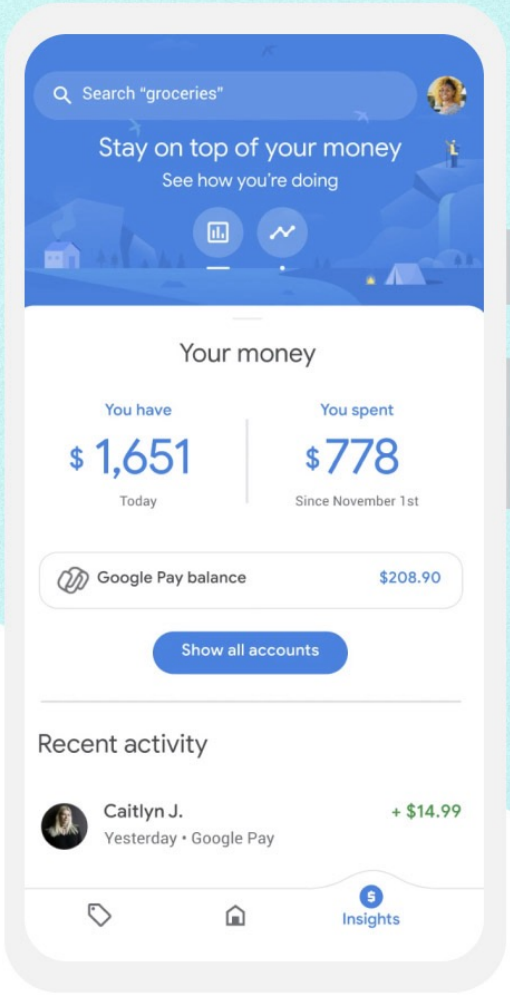
Customize every pixel

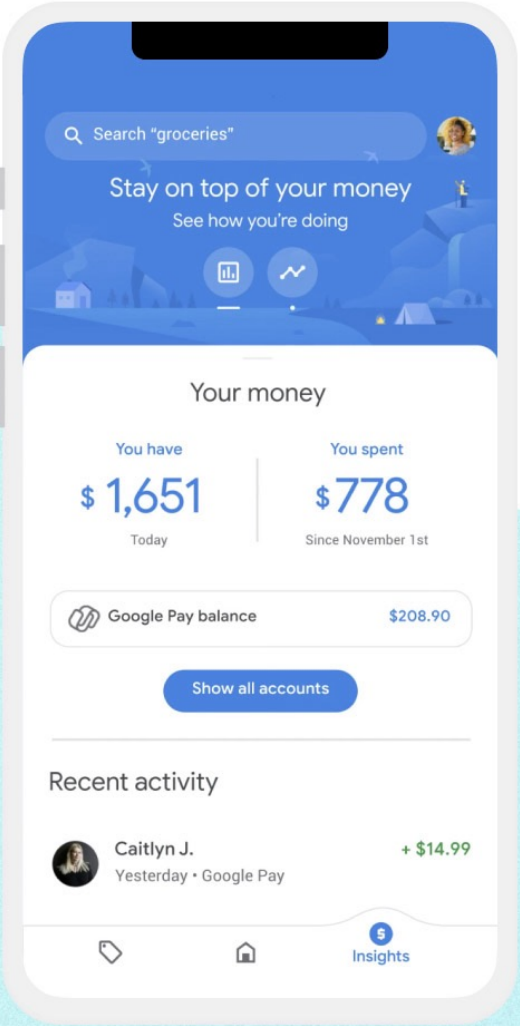
The Flutter rendering engine lets you control every pixel, and its widget library automatically adapts to any screen.



Introduction to Flutter

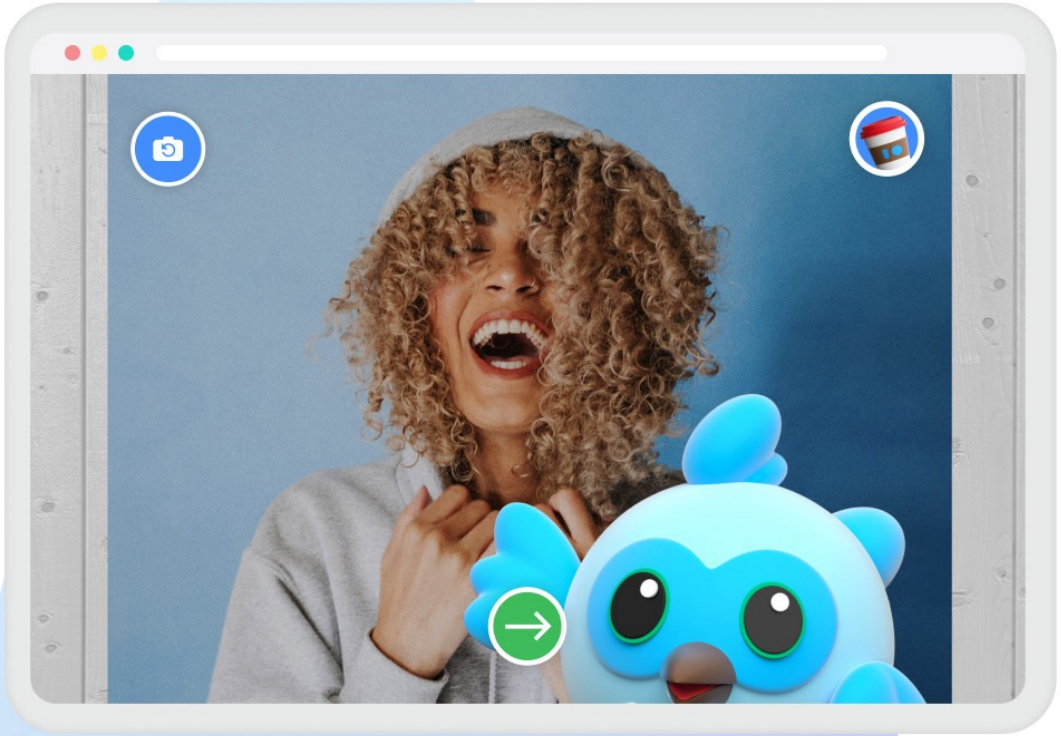
- **iOS and Android apps**
  - Build features once and deploy to both iOS and Android. Cupertino and Material designs are built into the Flutter framework, so your apps feel at home on both platforms.





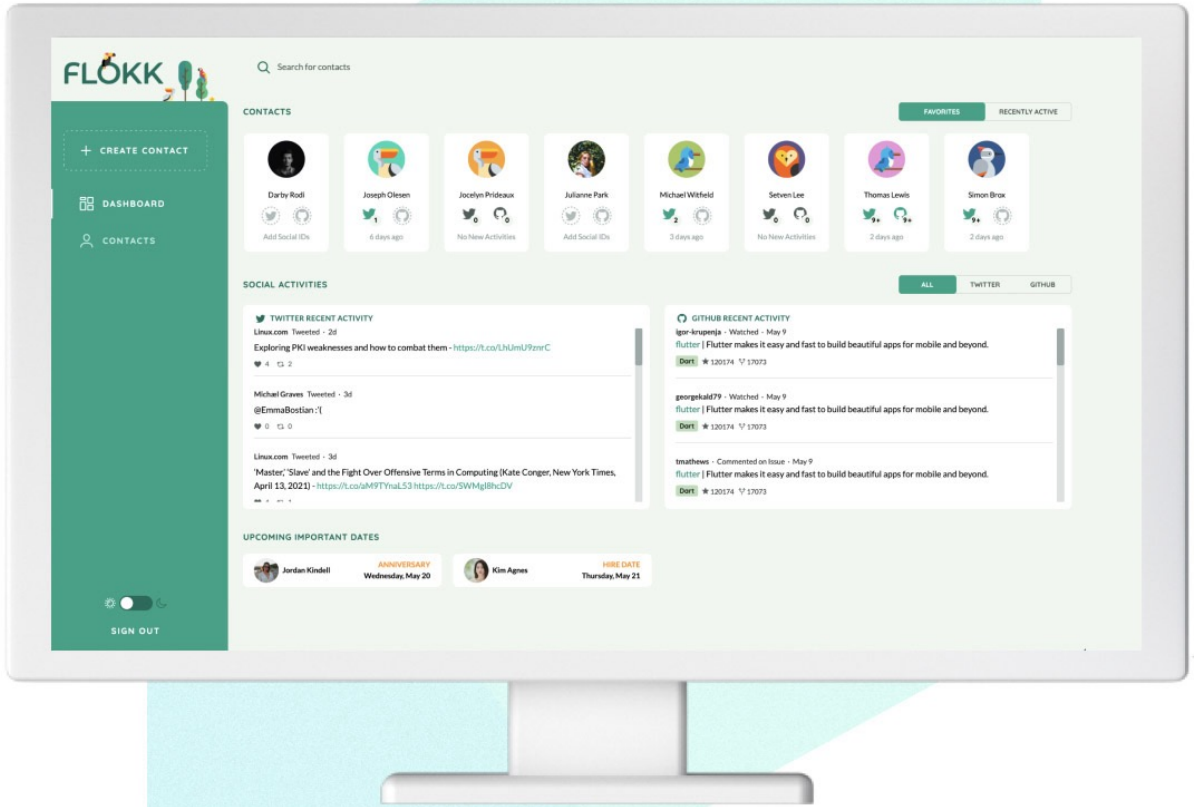
Introduction to Flutter

- **Web apps**
  - Reach users everywhere by deploying Flutter apps on the web. Build fast prototypes and deploy your mobile app to the web from the same codebase.



Introduction to Flutter

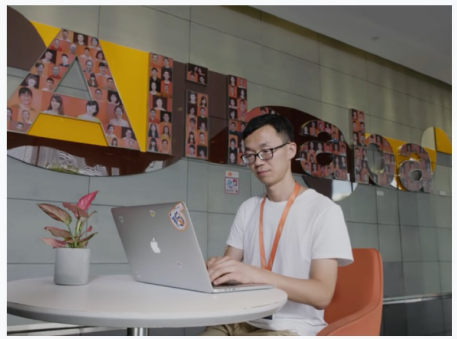
- **Windows, macOS, and Linux apps**
  - Transform your apps into desktop experiences with a single codebase and familiar tooling. Target Windows, macOS, and Linux without rewriting.



Introduction to Flutter

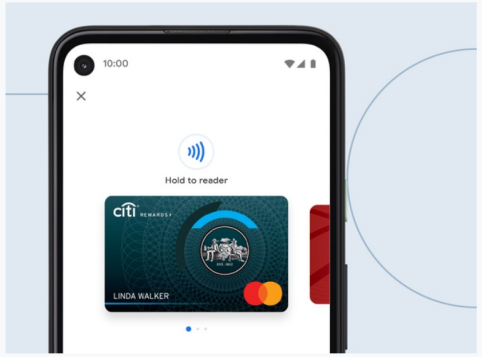
- **Flutter apps — anywhere**
  - Create custom solutions with the power and flexibility of Flutter. Deploy anywhere, including smart devices, cars, and more.

Apps that are built using Flutter



Alibaba Group

Alibaba scales China's largest second-hand marketplace with Flutter

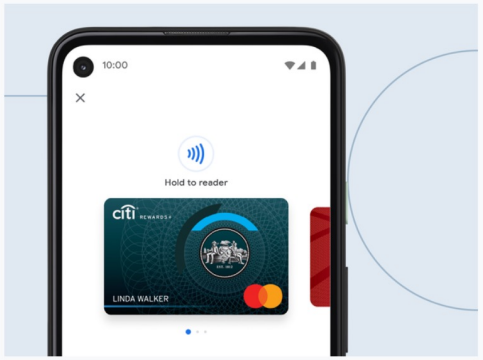


Google Pay

Going global at Google Pay with Flutter

ByteDance

Increasing productivity by 33% at ByteDance with Flutter



Google Pay

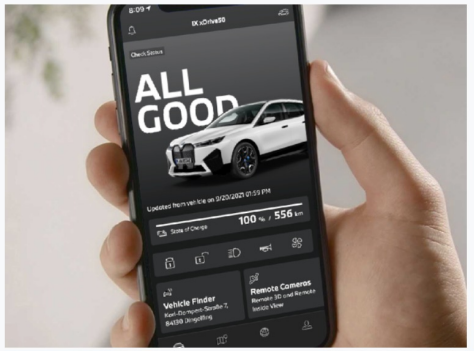
Going global at Google Pay with Flutter

Tencent

Tencent takes large, strategic bet on Flutter

Toyota

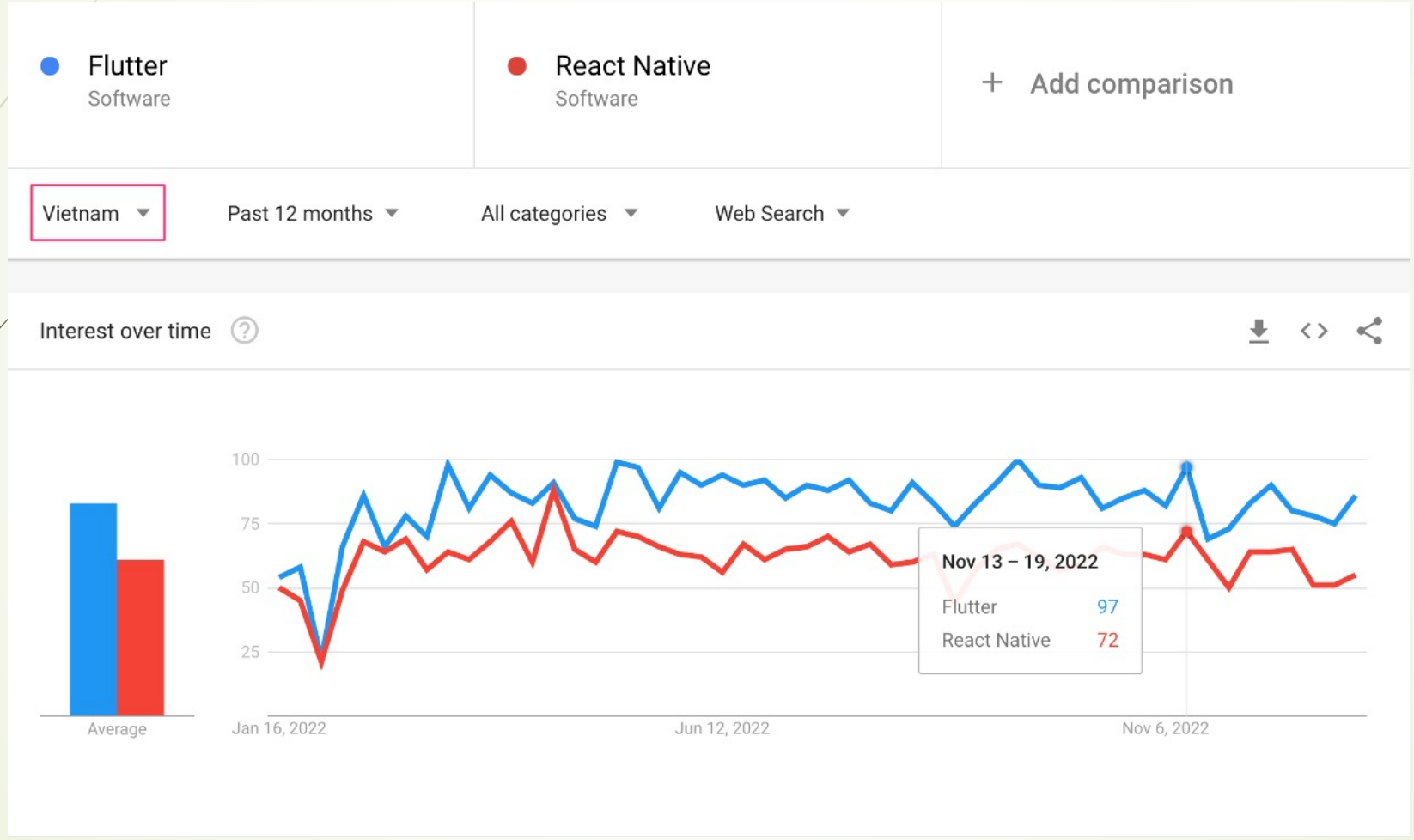
Improving infotainment systems at Toyota with Flutter



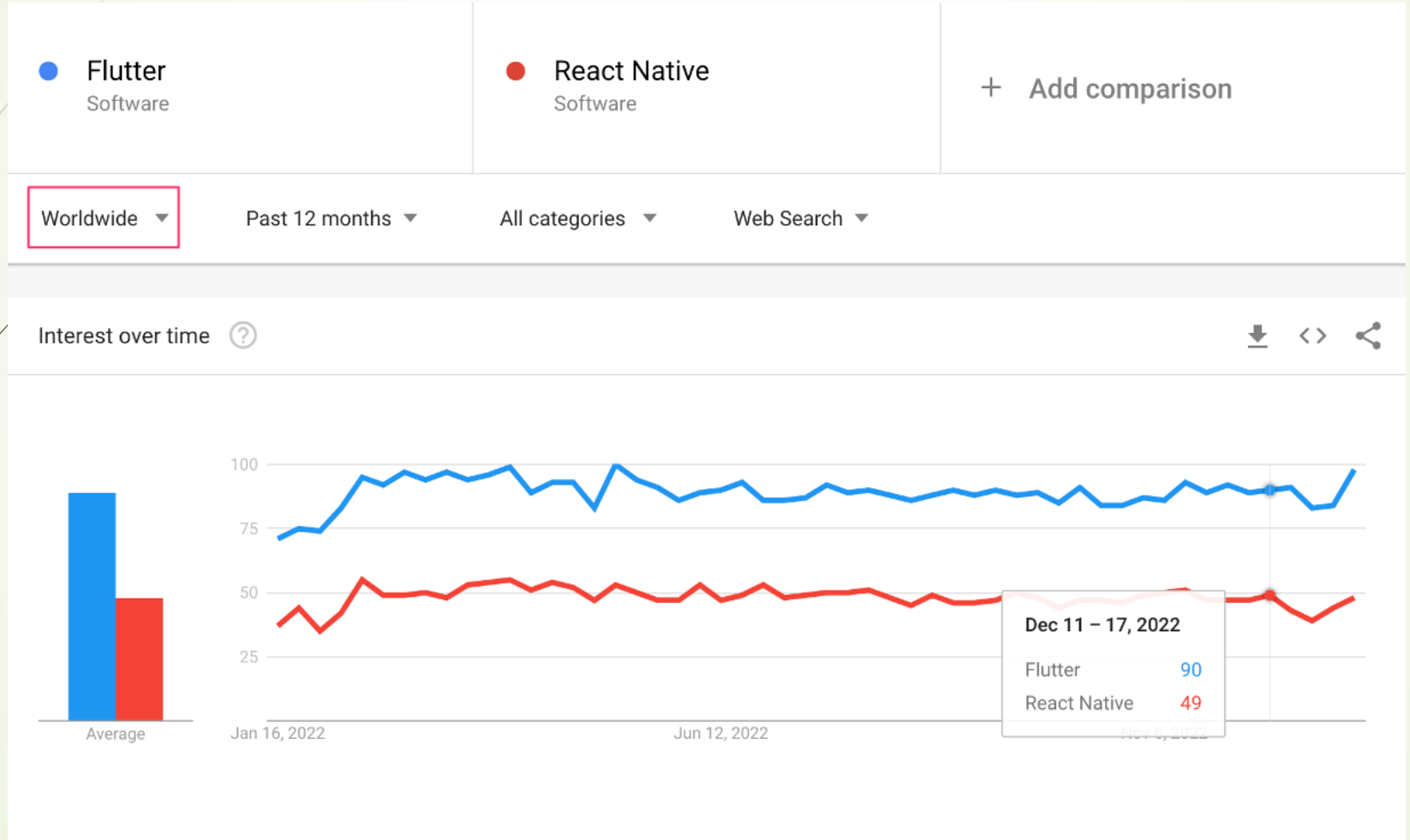
BMW

Scaling customer-centric product development at BMW Group with Flutter

Flutter vs React Native



Flutter vs React Native



Install Flutter

- Flutter can be installed in one of the following operating systems.
- System requirements
  - Operating Systems: macOS, Windows 10 or later (64-bit), x86-64 based.
  - Disk Space: 3 GB (does not include disk space for IDE/tools).
  - Tools: Flutter depends on these tools being available in your environment.
    - Windows PowerShell 5.0 or newer (this is pre-installed with Windows 10)
    - Xcode (macOS only)
    - Git for Windows 2.x, with the Use Git from the Windows Command Prompt option.

Install Flutter

1. Download the following installation bundle to get the latest stable release of the Flutter SDK:

   flutter_windows_3.3.10-stable.zip

   For other release channels, and older builds, see the SDK releases page.

2. Extract the zip file and place the contained `flutter` in the desired installation location for the Flutter SDK (for example, `C:\src\flutter`).

Update your path

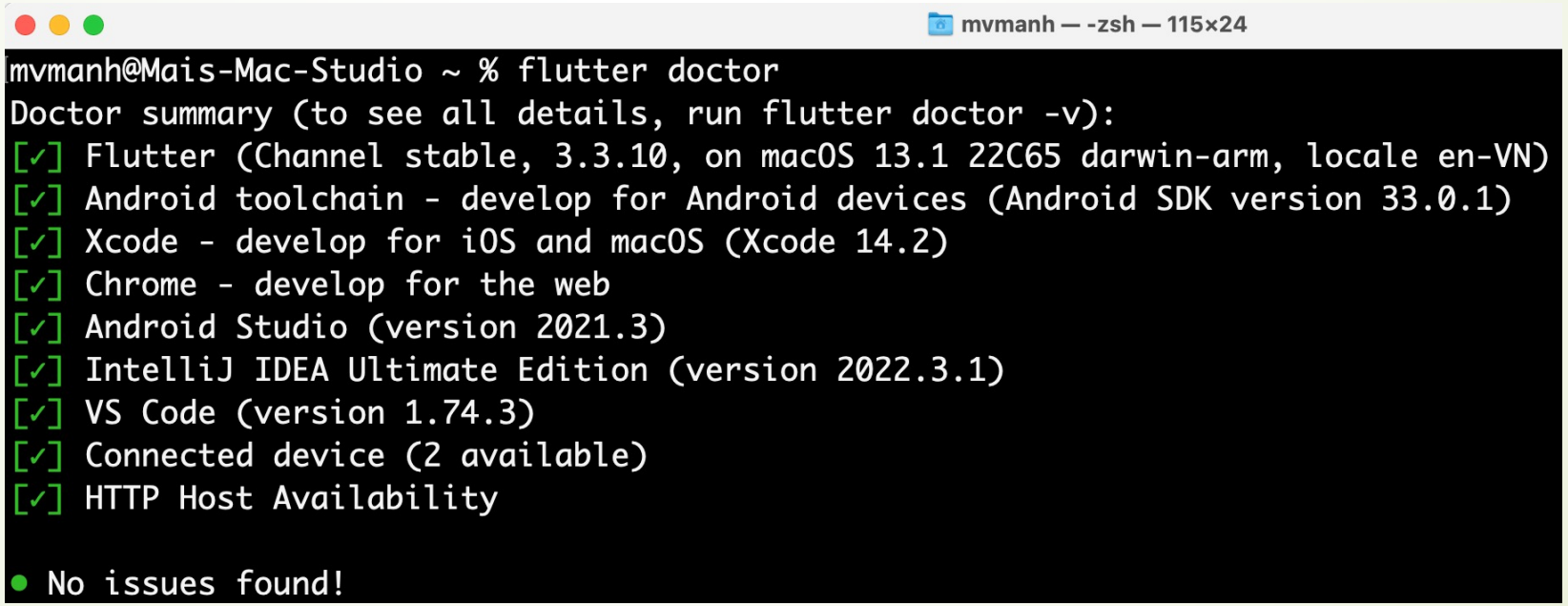
If you wish to run Flutter commands in the regular Windows console, take these steps to add Flutter to the `PATH` environment variable:

- From the Start search bar, enter 'env' and select **Edit environment variables for your account**.
- Under **User variables** check if there is an entry called **Path**:
  - If the entry exists, append the full path to `flutter\bin` using `;` as a separator from existing values.
  - If the entry doesn't exist, create a new user variable named `Path` with the full path to `flutter\bin` as its value.

You have to close and reopen any existing console windows for these changes to take effect.

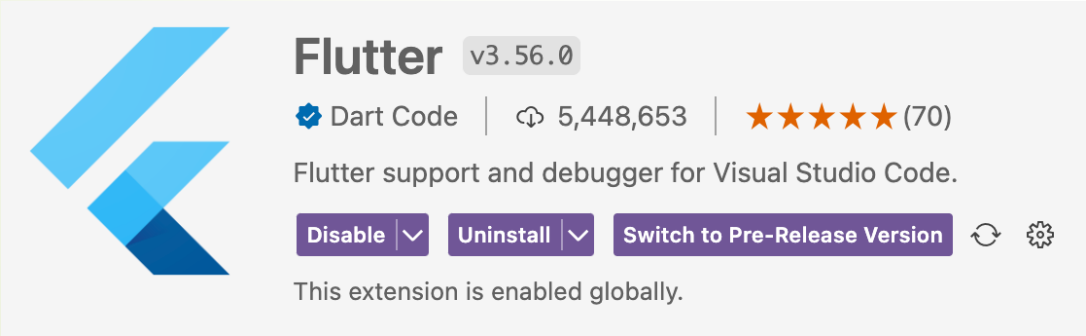
Install Flutter

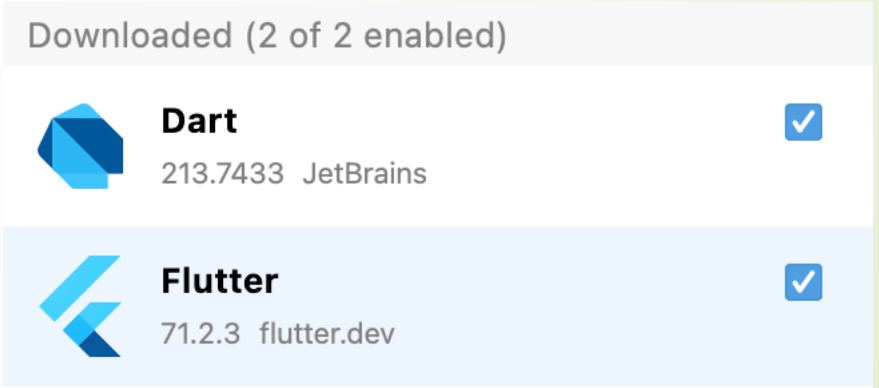
- Run flutter doctor
  - This command checks your environment and displays a report of the status of your Flutter installation. Check the output carefully for other software you might need to install or further tasks to perform (shown in **bold** text).



Set up an editor

- You can build apps with Flutter using any text editor combined with Flutter's command-line tools.
- However, using Visual Studio Code, Android Studio, IntelliJ, or Emacs is recommended for an even better experience. Install Flutter plugin on one of these code editor to get started.

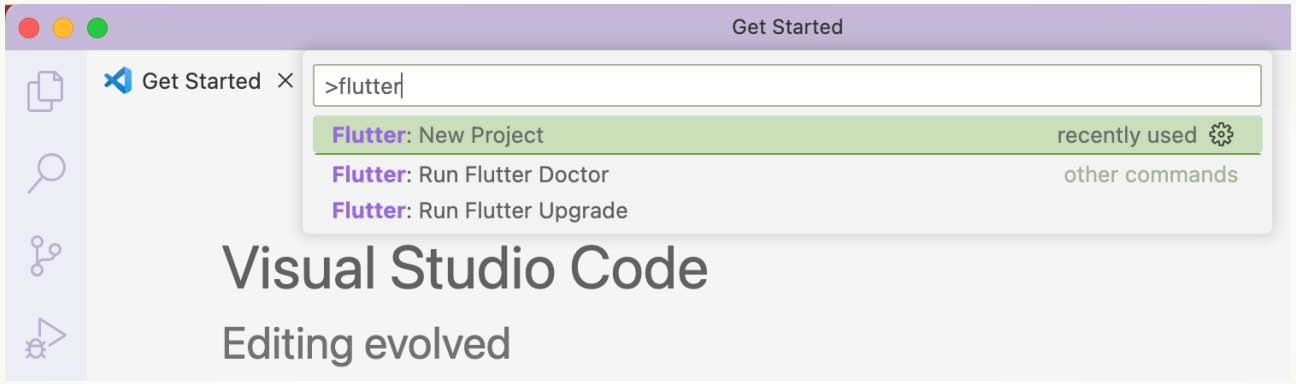




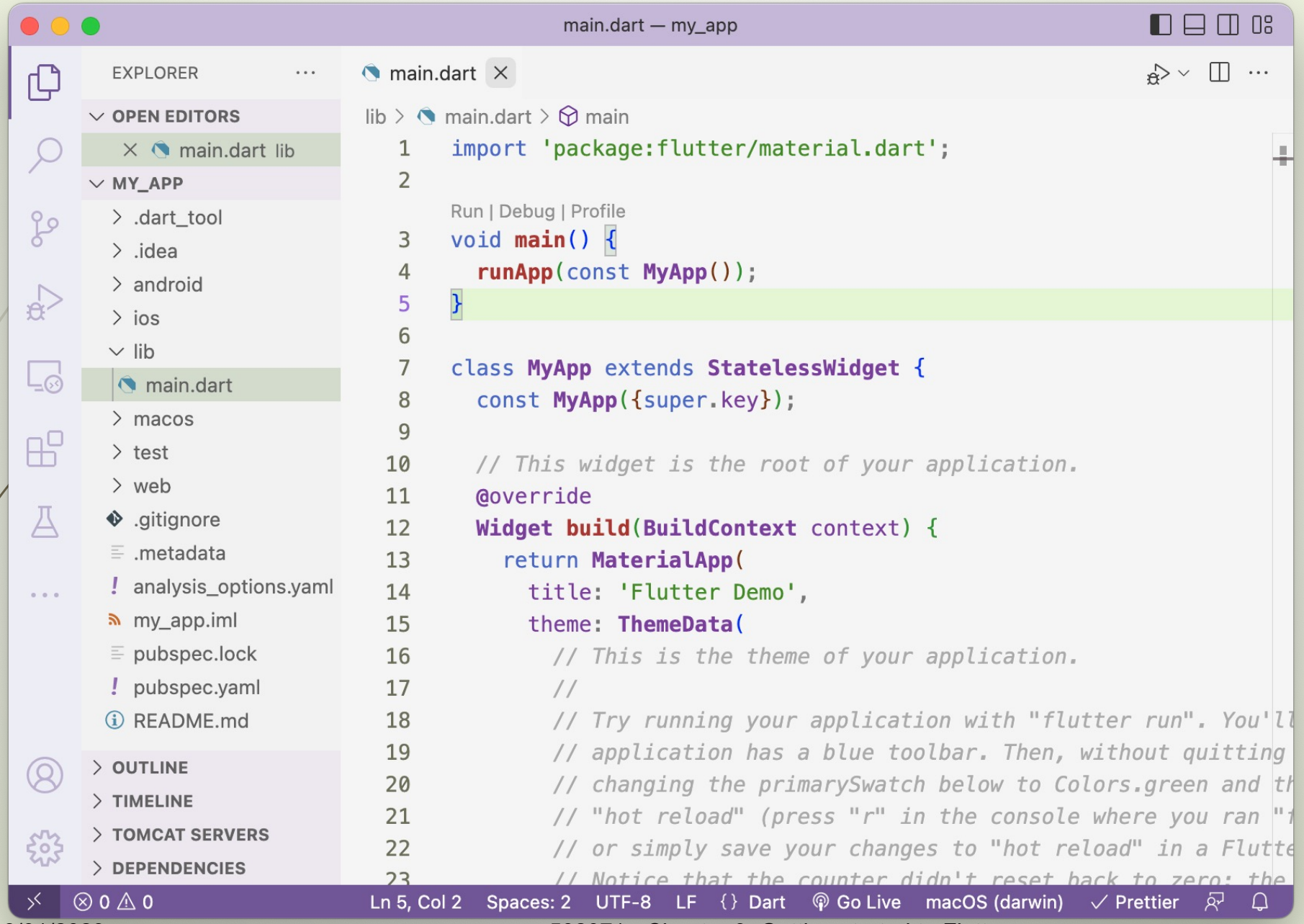
First Flutter App

First Flutter App

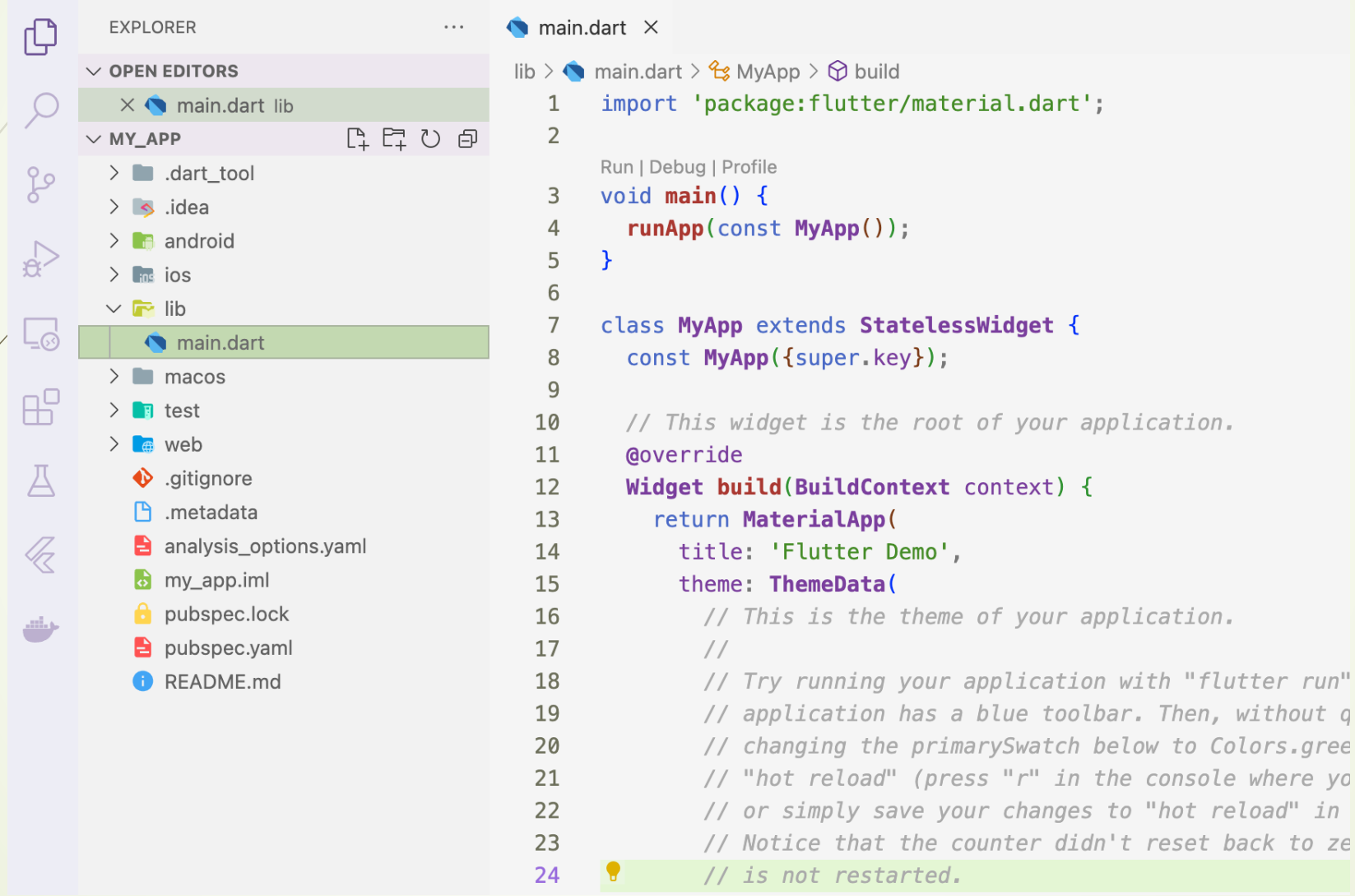
- Launch Visual Studio Code and open the command palette (with F1 or Ctrl+Shift+P or Shift+Cmd+P). Start typing "flutter new". Select the **Flutter: New Project** command.



- Next, select **Application** and then a folder in which to create your project. This could be your home directory, or something like C:\src\.
- Finally, name your project. Something like **namer_app** or **my_awesome_namer**.

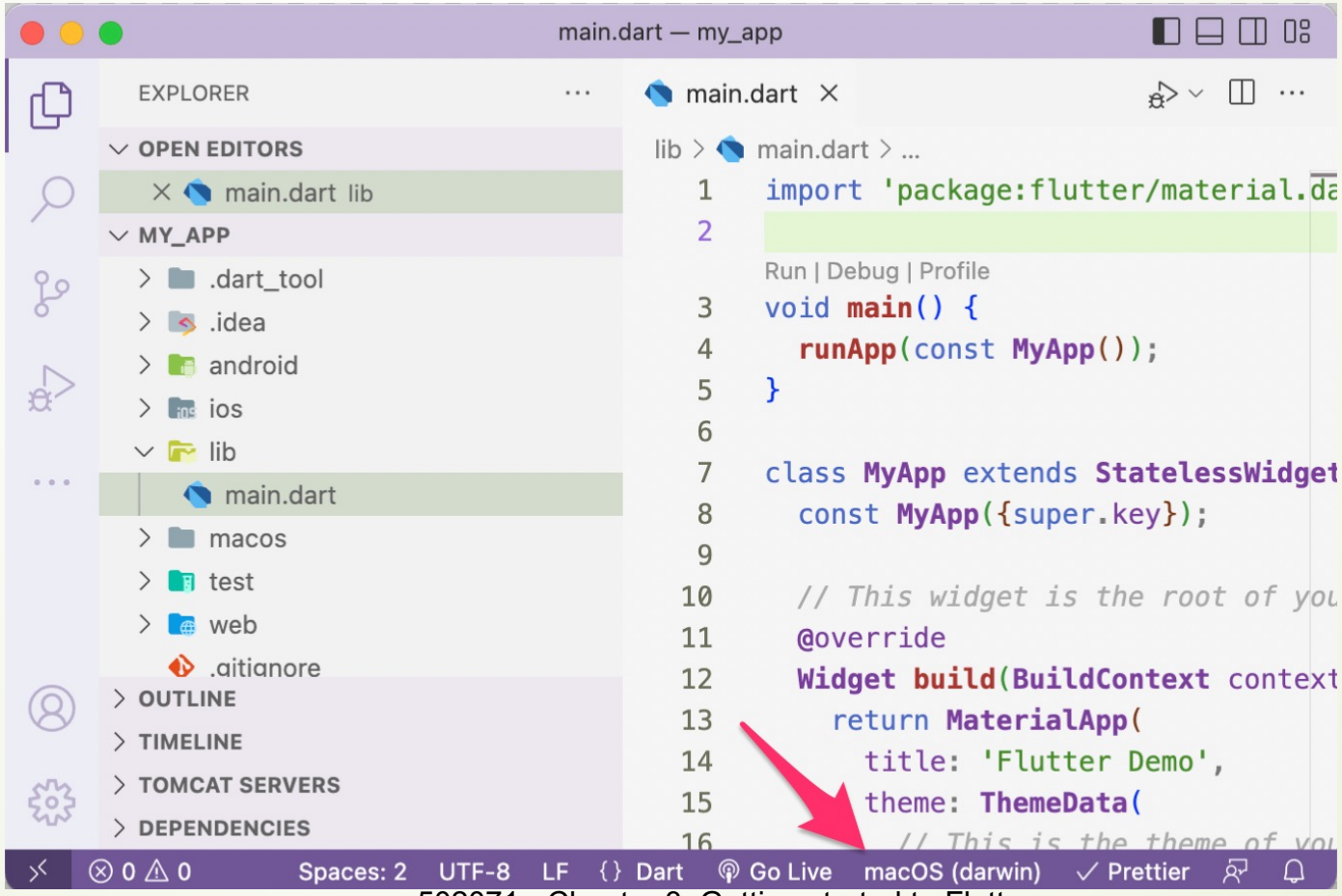

main.dart — my_app
EXPLORER
OPEN EDITORS
main.dart lib
MY_APP
.dart_tool
.idea
android
ios
lib
main.dart
macos
test
web
.gitignore
.metadata
analysis_options.yaml
my_app.iml
pubspec.lock
pubspec.yaml
README.md
OUTLINE
TIMELINE
TOMCAT SERVERS
DEPENDENCIES
main.dart
lib > main.dart > main
1 import 'package:flutter/material.dart';
2
Run | Debug | Profile
3 void main() {
4 runApp(const MyApp());
5 }
6
7 class MyApp extends StatelessWidget {
8 const MyApp({super.key});
9
10 // This widget is the root of your application.
11 @override
12 Widget build(BuildContext context) {
13 return MaterialApp(
14 title: 'Flutter Demo',
15 theme: ThemeData(
16 // This is the theme of your application.
17 //
18 // Try running your application with "flutter run". You'll
19 // application has a blue toolbar. Then, without quitting
20 // changing the primarySwatch below to Colors.green and th
21 // "hot reload" (press "r" in the console where you ran "f
22 // or simply save your changes to "hot reload" in a Flutte
23 // Notice that the counter didn't reset back to zero; the
0 0
Ln 5, Col 2
Spaces: 2
UTF-8
LF
Dart
Go Live
macOS (darwin)
Prettier

Flutter project structure



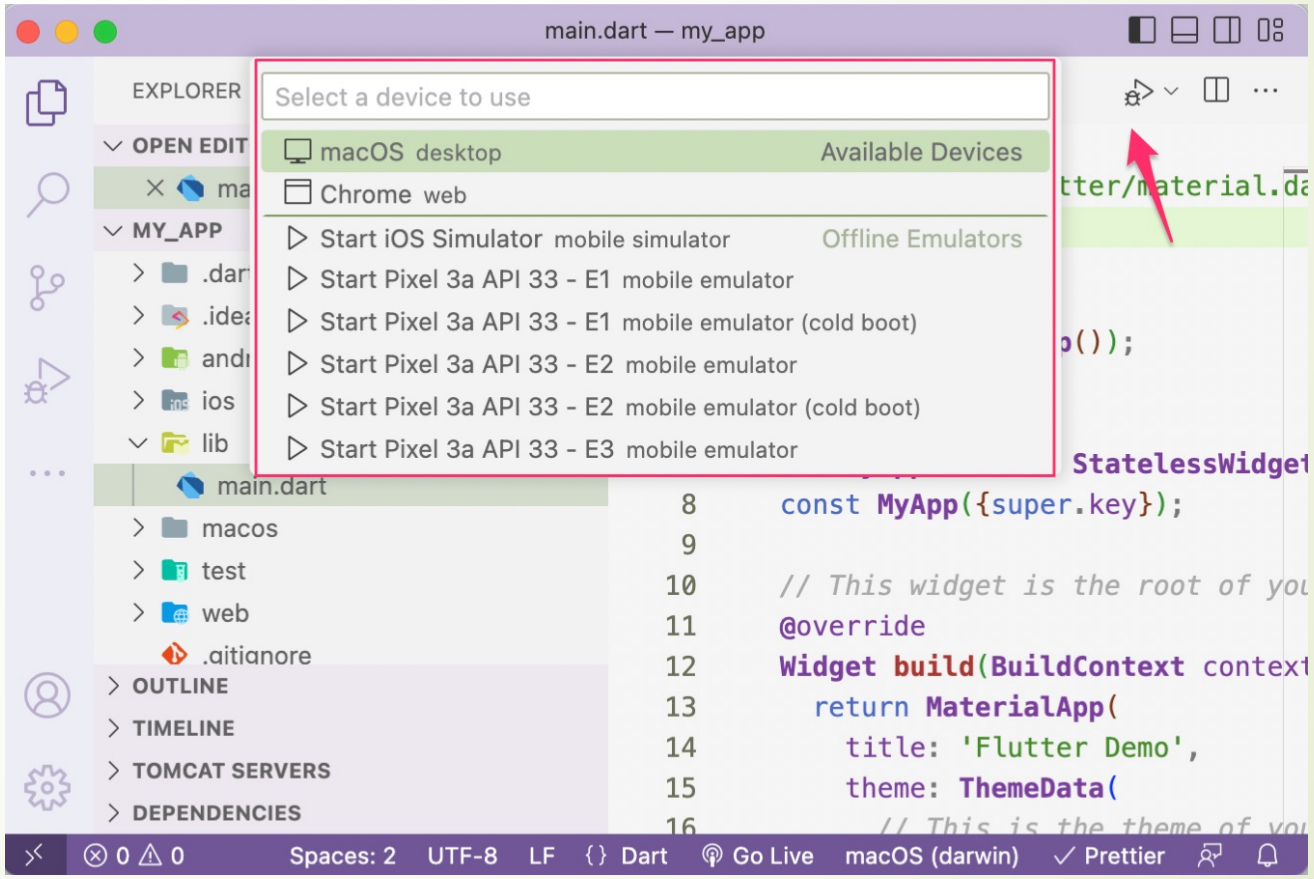
First Flutter App

- To run the project, you first choose the target device (macOS, Windows, Chrome or emulator) by clicking on the device section at the bottom of the editor.



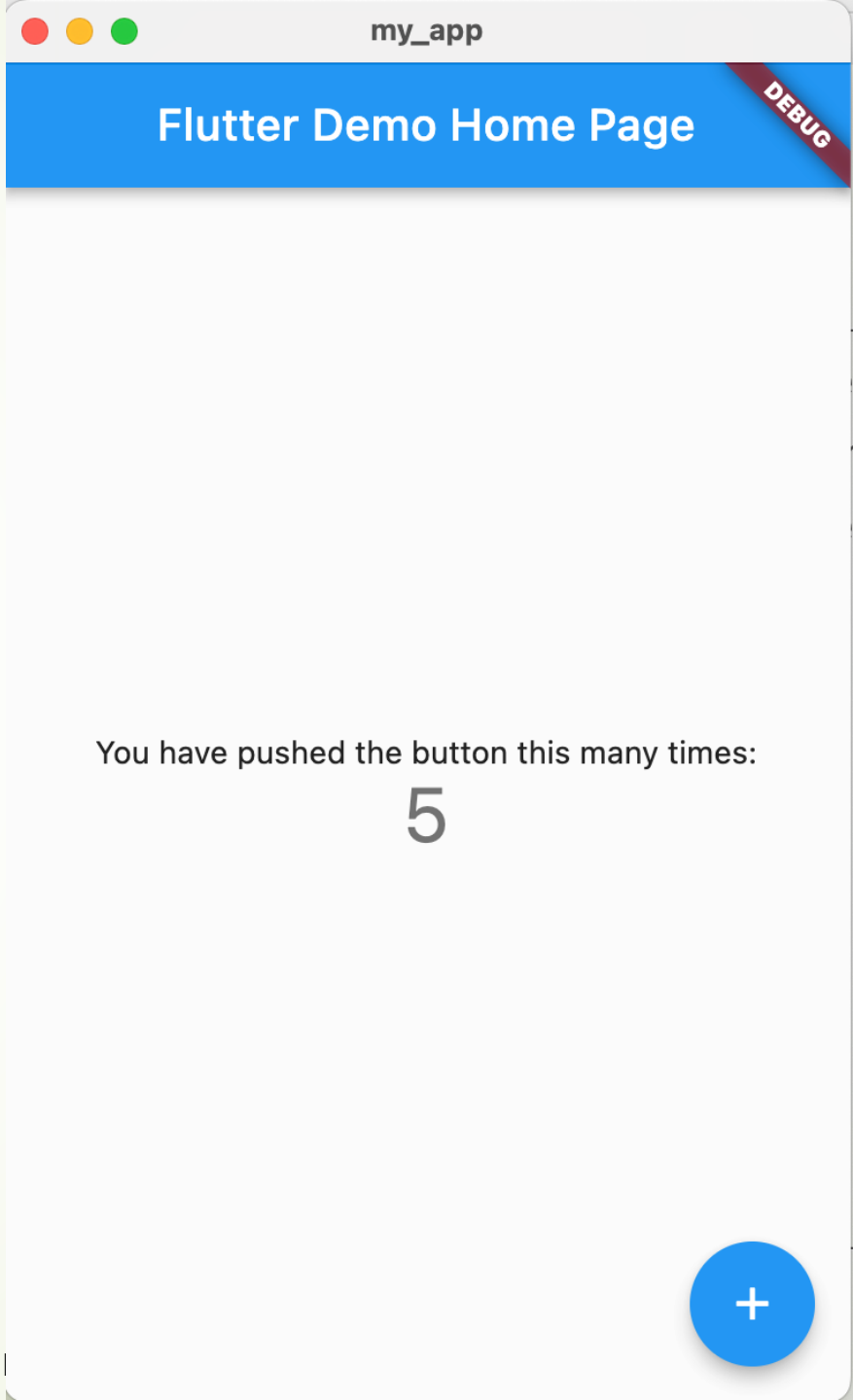
First Flutter App

- If you choose a desktop device (macOS, Windows or Linux) the app will be built and launched automatically. If you want to run an ios or android app, you should first launch the ios simulator android emulator device.



First Flutter App

- The result is a simple application using the Material Design theme with a Floating Action Button that counts the number of times the user has clicked on it.

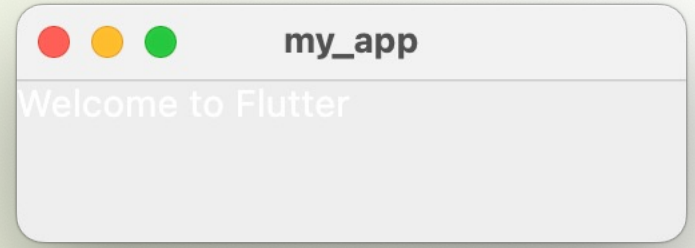


Minimalistic Flutter App

- Here is an example of one of the most minimalistic Flutter application which shows only one text on the screen at the top left corner.

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

void main() {

    var text = Text('Welcome to Flutter',
            textDirection: TextDirection.ltr);

    runApp(text);
}
```

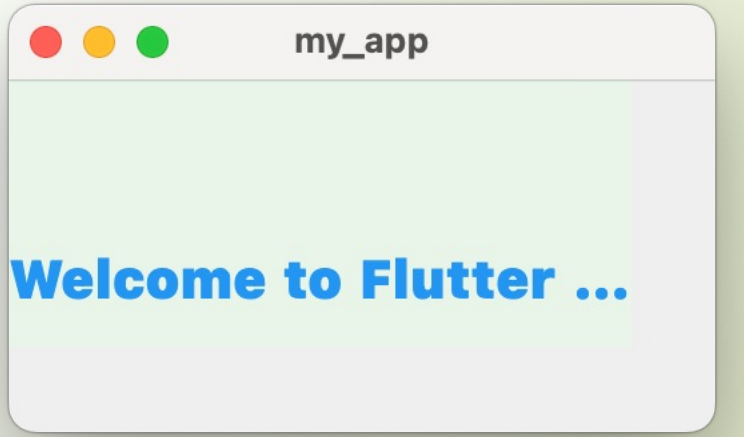
```dart
import 'package:flutter/material.dart';

void main() {

    runApp(Text('Welcome to Flutter',
            textDirection: TextDirection.ltr));
}
```



Minimalistic Flutter App

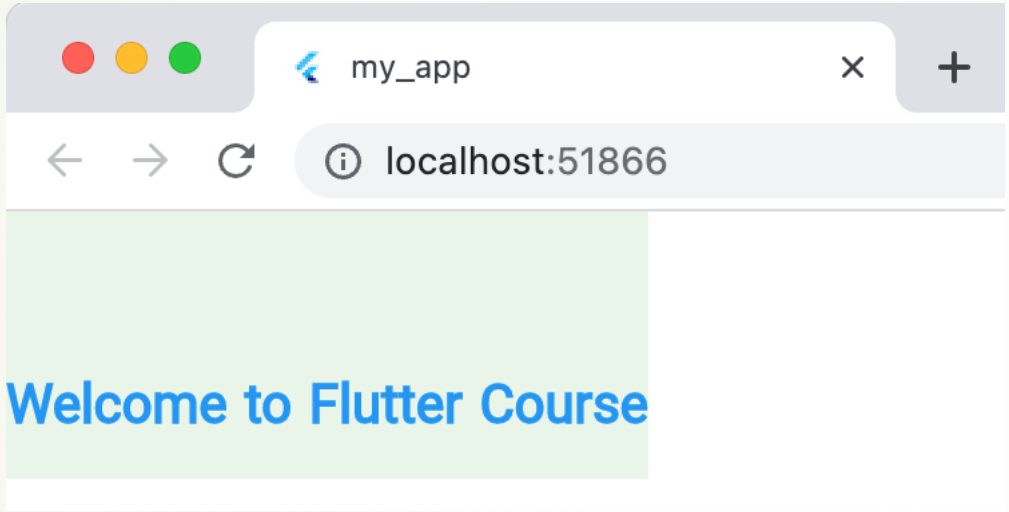
- Here is an example of one of the most minimalistic Flutter application which shows only one text on the screen at the top left corner.

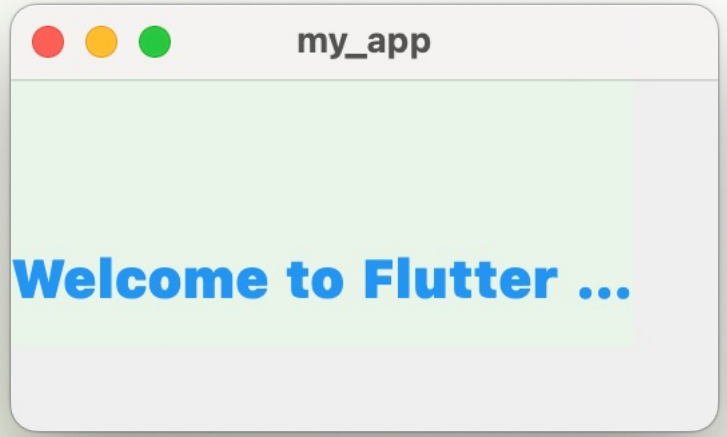
```
import 'package:flutter/material.dart';

void main() {

    var myStyle = TextStyle(color: Colors.blue,
        fontSize: 20, wordSpacing: 1.5,
        fontWeight: FontWeight.w900,
        height: 5, backgroundColor: Colors.green[50]);

    runApp(Text('Welcome to Flutter Course',
            textDirection: TextDirection.ltr,
            overflow: TextOverflow.ellipsis,
            style: myStyle,));
}
```

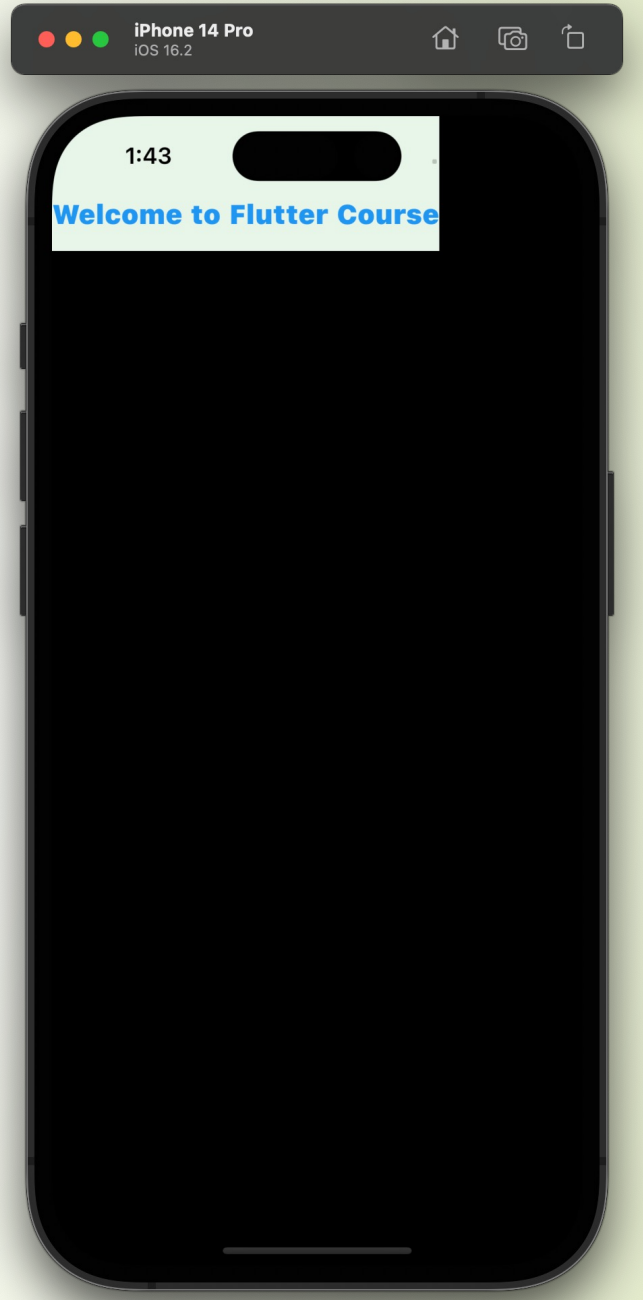


Minimalistic Flutter App

- Results may vary slightly between devices







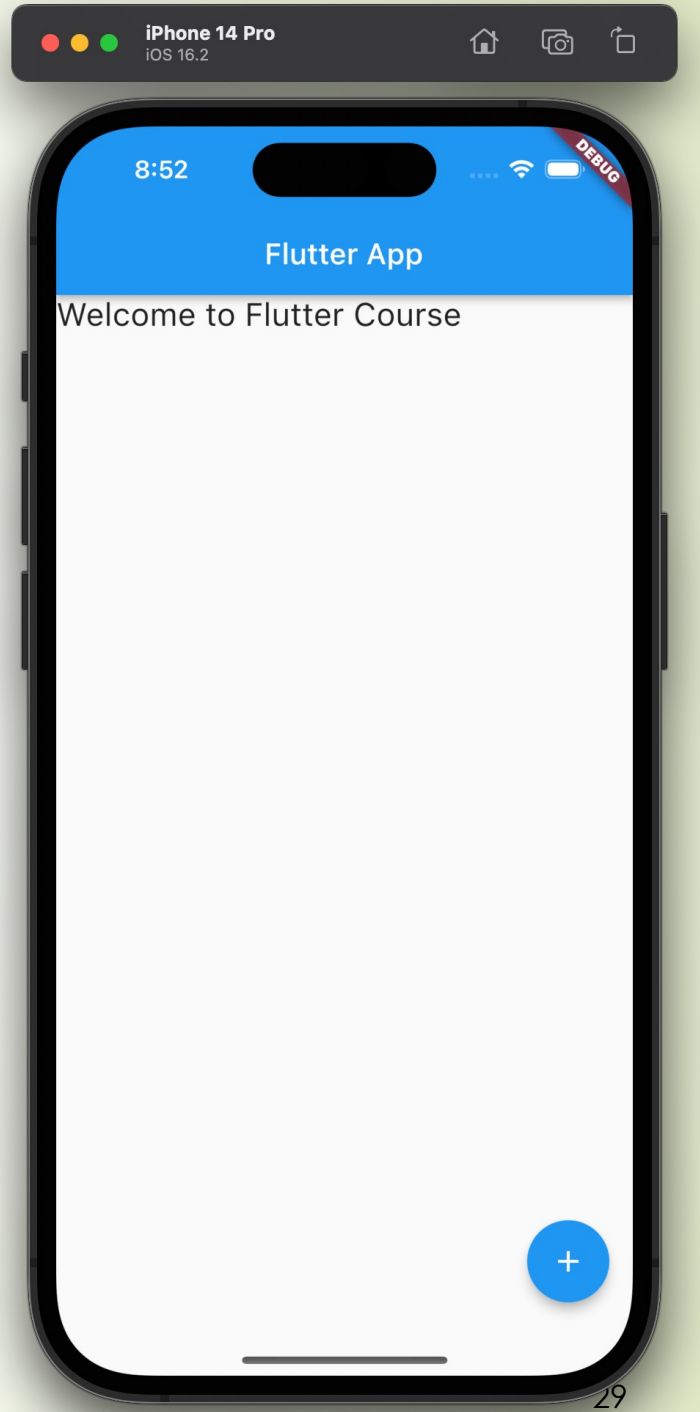
Widgets

Flutter Widget

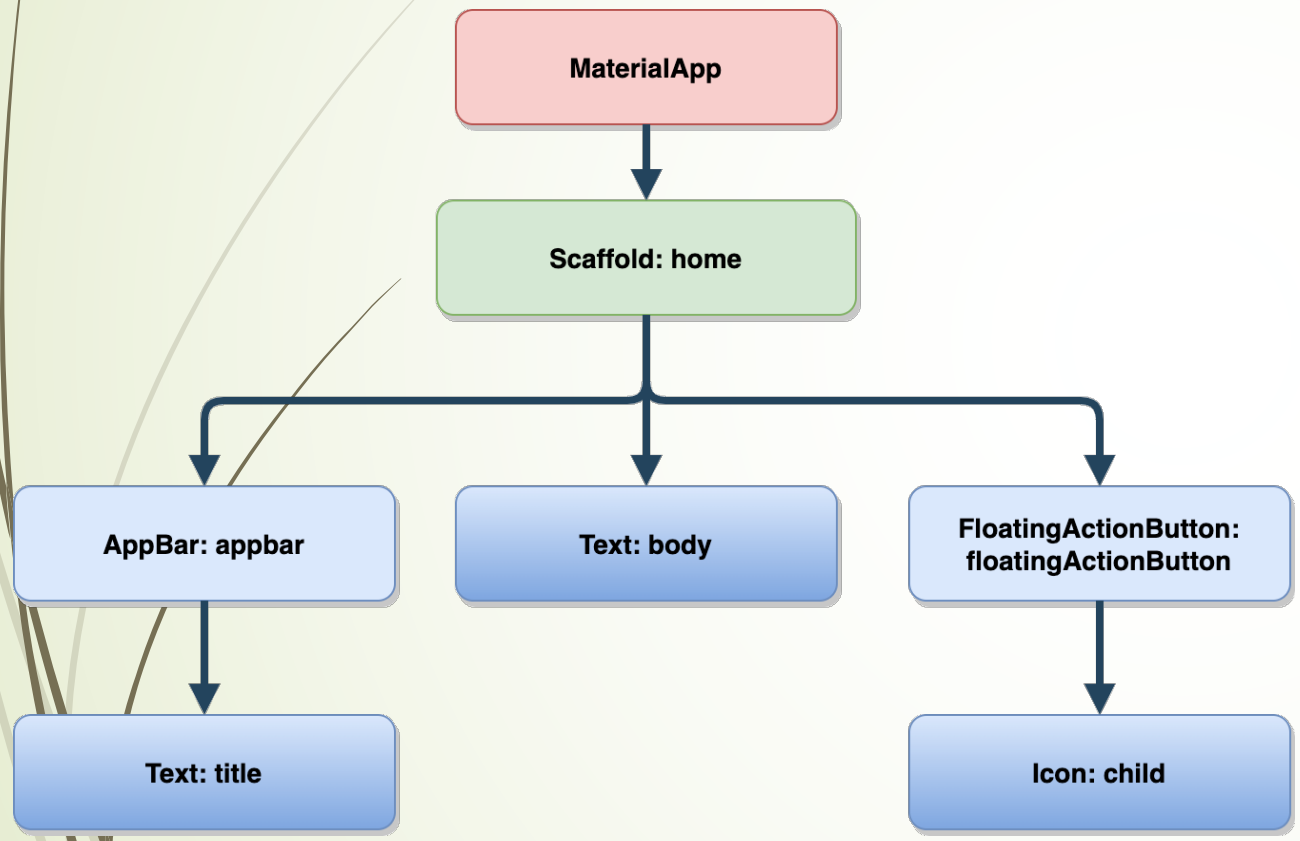
- The core concept of the Flutter framework is In Flutter, **Everything is a widget**. Widgets are basically user interface components used to create the user interface of the application.
- In Flutter, **the application is itself a widget**. The application is the top-level widget and its UI is build using one or more children (widgets), which again build using its children widgets.
- Flutter widgets are built using a modern framework that takes inspiration from React. The central idea is that you build your UI out of widgets.

Flutter Widget

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

var btn = FloatingActionButton(onPressed: () => {},
                               child: Icon(Icons.add));

var appBar = AppBar(title: Text('Flutter App'));

var scaffold = Scaffold(
    appBar: appBar,
    body: Text('Welcome to Flutter Course’),
    floatingActionButton: btn);

var app = MaterialApp(home: scaffold);

void main() {
    runApp(app);
}
```



Flutter Widget




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

var btn = FloatingActionButton(onPressed: () => {},
              child: Icon(Icons.add));

var appBar = AppBar(title: Text('Flutter App'));

var scaffold = Scaffold(
     appBar: appBar,
     body: Text('Welcome to Flutter Course'),
     floatingActionButton: btn);

var app = MaterialApp(home: scaffold);

void main() {
     runApp(app);
}
```

Flutter Widget

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

var btn = FloatingActionButton(onPressed: () => {},
                               child: Icon(Icons.add));

var appBar = AppBar(title: Text('Flutter App'));

var scaffold = Scaffold(
    appBar: appBar,
    body: Text('Welcome to Flutter Course’),
    floatingActionButton: btn);

var app = MaterialApp(home: scaffold);

void main() {
    runApp(app);
}
```

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

void main() {
runApp(MaterialApp(home: Scaffold(
     appBar: AppBar(title: Text('Flutter App’),),
     body: Text('Welcome to Flutter Course', style:
     TextStyle(fontSize: 22),),
     floatingActionButton: FloatingActionButton(onPressed:
     () => {}, child: Icon(Icons.add),),),),);
}
```

```dart
void main() {
    runApp(
        MaterialApp(
            home: Scaffold(
                appBar: AppBar(
                title: Text('Flutter App'),
            ), // AppBar
            body: Text(
                'Welcome to Flutter Course',
                style: TextStyle(fontSize: 22),
            ), // Text
            floatingActionButton: FloatingActionButton(
                    onPressed: () => {},
                    child: Icon(Icons.add),
            ), // FloatingActionButton),
        ), // Scaffold
    ); // MaterialApp
}
```

Basic Widgets

- Flutter comes with a suite of powerful basic widgets, the following are commonly used:
  - **Text**: The Text widget lets you create a run of styled text within your application.
  - **Row**, **Column**: These flex widgets let you create flexible layouts in both the horizontal (Row) and vertical (Column) directions. The design of these objects is based on the web's flexbox layout model.
  - **Stack**: Instead of being linearly oriented (either horizontally or vertically), a Stack widget lets you place widgets on top of each other in paint order. You can then use the Positioned widget on children of a Stack to position them relative to the top, right, bottom, or left edge of the stack. Stacks are based on the web's absolute positioning layout model.
  - **Container**: The Container widget lets you create a rectangular visual element. A container can be decorated with a BoxDecoration, such as a background, a border, or a shadow. A Container can also have margins, padding, and constraints applied to its size. In addition, a Container can be transformed in three dimensional space using a matrix.

Stateful and stateless widgets

- A widget is either **stateful** or **stateless**. If a widget can change — when a user interacts with it, for example—it's stateful.
- A stateless widget never changes. **Icon**, **IconButton**, and **Text** are examples of stateless widgets. Stateless widgets subclass StatelessWidget.
- **Checkbox**, **Radio**, **Slider**, **InkWell**, **Form**, and **TextField** are examples of stateful widgets. Stateful widgets subclass StatefulWidget.
- A widget's state is stored in a **State object**, separating the widget's state from its appearance. The state consists of values that can change, like a slider's current value or whether a checkbox is checked.

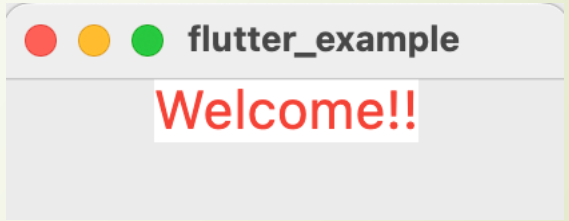
Text Widget

- The Text widget displays a string of text with single style. The string might break across multiple lines or might all be displayed on the same line depending on the layout constraints.

```
Text('Welcome to Flutter')
```

- The style argument is optional. When omitted, the text will use the style from the closest enclosing DefaultTextStyle.

```
Text('Welcome', textAlign: TextAlign.center,
    style: TextStyle(
        fontSize: 20,
        color: Colors.green,
        backgroundColor: Colors.grey),)
```



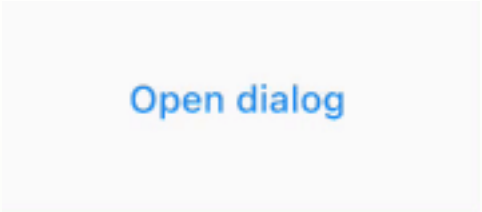
Button Widget

- Flutter provides several types of buttons that have different shapes, styles, and features.
- Following are the different types of button available in Flutter: **Flat Button**, **Raised Button**, **Floating Button**, **Drop Down Button**, **Icon Button**, **Inkwell Button**, **PopupMenu Button**, **Outlined Button**.
- Three of the most popular buttons are **TextButton**, **ElevatedButton**, and **OutlinedButton**.

Text Button

- TextButton is a built-in widget in Flutter which derives its design from Material Design Library.
- It is a simple Button without any border that listens for **onPressed** and **onLongPress** gestures

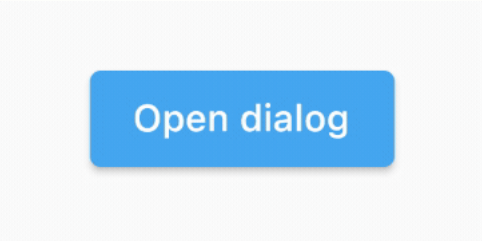
```
TextButton(
    child: const Text('Open dialog'),
    onPressed: () => print('Clicked'))
```



Elevated Button

- Elevated Button is a flutter component included inside the material package i.e. **“package:flutter/material.dart**“.
- The main characteristic these buttons hold is the slight elevation in their surface towards the screen on getting tapped by the user.

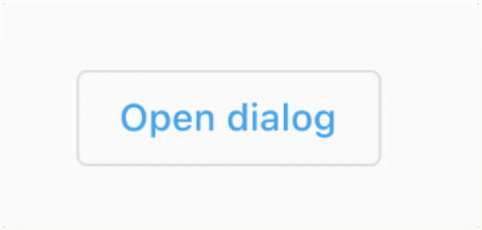
```
ElevatedButton(
    child: const Text('Open dialog’),
    onPressed: () => print('Clicked'))
```



Outlined Button

- **Outlined button** is in no way different from text buttons except for the special feature of the border this class provides.
- **Outlined buttons** have **child** as their label with text widgets or icons widgets as the child widget to this parent class.

```
OutlinedButton(
    child: const Text('Open dialog’),
    onPressed: () => print('Clicked'))
```



TextField

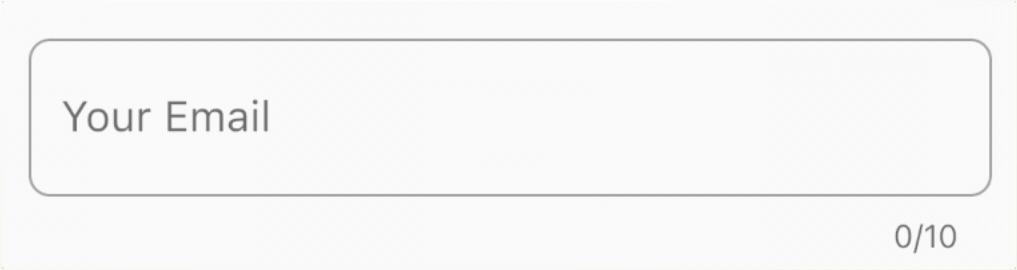
- **TextField** is used to get text input from user. The default behavior of TextField is that, when you press on it, it gets focus and a keyboard slides from the bottom of the screen.

```
TextField(
    maxLength: 10,
    maxLines: 1,
    textAlign: TextAlign.center,
    readOnly: false)
```

TextField

- By default, Text Field has no **border**. You can add borders by setting the **decoration** property.

```
TextField(
    decoration: InputDecoration(
                 border: OutlineInputBorder(
                              borderRadius: BorderRadius.circular(8)),
                              labelText: 'Your Email'),
    maxLength: 10, maxLines: 1,
    textAlign: TextAlign.center, readOnly: false)
```

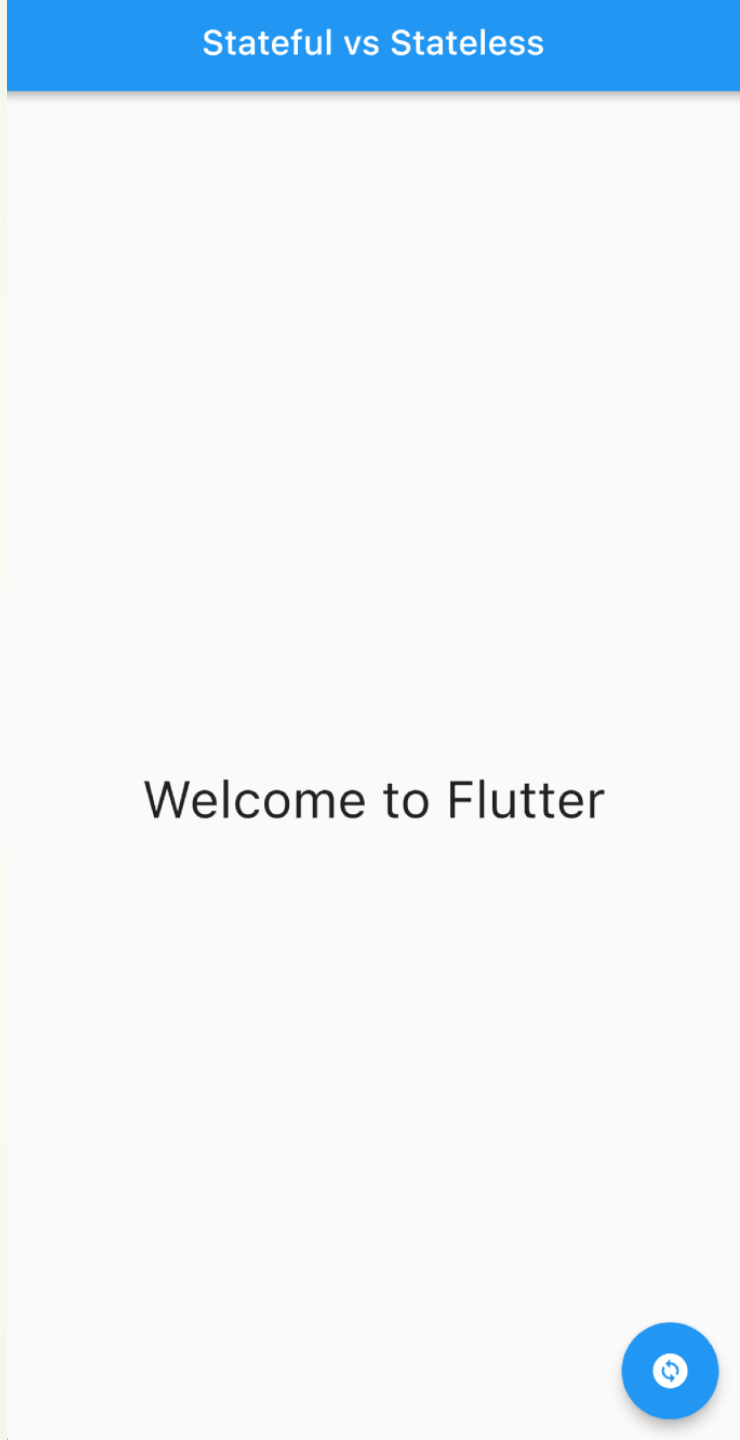


Stateless Widget

- **Immutable state**: The state of a StatelessWidget cannot be changed, making it easy to reason about and debug.
- **Easy to understand**: StatelessWidgets are straightforward and easy to understand, making them ideal for simple UI elements.
- **Fast creation**: Because StatelessWidgets have no mutable state, they can be created and drawn quickly, making them ideal for use in high-performance applications.
- **Predictable behavior**: Since StatelessWidgets have no mutable state, they always behave the same way, making them predictable and reliable. This is especially useful for creating UI elements that are not affected by user interactions.

Stateless Widget

```
class MyApp extends StatelessWidget {
        String message = 'Welcome to Flutter';

    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(
            title: const Text('Stateful vs Stateless')),
        body: Center(
            child: Text(message,
            style: const TextStyle(fontSize: 30))),
        floatingActionButton: FloatingActionButton(
            child: const Icon(Icons.change_circle),
            onPressed: () {
                message = 'You have clicked';
                print(message);
            },
        ),);
```



Stateful Widget

- **Mutable state**: StatefulWidgets can have mutable state, which allows them to change and respond to user interactions.
- **Dynamic updates**: Because StatefulWidgets can have mutable state, they can update dynamically in response to user interactions, allowing you to create rich and interactive user experiences.
- **Separate state management**: The mutable state of a StatefulWidget is managed in a separate class, making it easy to manage and update.
- **Improved performance**: With StatefulWidgets, you can selectively cache widgets that have a stable appearance, which can improve performance by reducing the number of widgets that need to be redrawn. Additionally, StatefulWidgets can be optimized for fast and efficient updates, allowing you to create high-performance applications.

Stateless Widget

- **Step 1**: Convert from stateless widget to stateful widget.

```
class MyApp extends StatefulWidget {
    @override
    State<MyApp> createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {

    String message = 'Welcome to Flutter';

    @override
    Widget build(BuildContext context) {
        return Scaffold(...);
    }
}
```

Stateless Widget

- **Step 2**: Put the changes inside the **setState** method.

```
floatingActionButton: FloatingActionButton(
    child: const Icon(Icons.change_circle),
    onPressed: () {
        setState(() {
            message = 'You have clicked';
            print(message);
        });
    },
)
```

You have clicked

Stateless Widget vs Stateful Widget

- Differences:
  - **State**: The main difference between StatelessWidget and StatefulWidget is that StatefulWidgets have mutable state, while StatelessWidgets do not.
  - **Reusability**: Because StatelessWidgets have no mutable state, they can be cached and reused, making them more efficient than StatefulWidgets.
  - **Dynamic updates**: StatefulWidgets can update dynamically in response to user interactions, while StatelessWidgets cannot.
  - **State management**: The mutable state of a StatefulWidget is managed in a separate class, making it easier to manage and update than StatelessWidgets.
  - **Performance**: StatefulWidgets can be optimized for fast and efficient updates, making them more suitable for high-performance applications than StatelessWidgets. However, StatelessWidgets are generally faster to create and render than StatefulWidgets.