Build an Android App with Flutter: A Beginner’s Guide for Windows Users

By Dora Cheng

Google launched Flutter, an open-source UI SDK, in 2017 and Flutter has been growing in popularity ever since. According to the 2020 Stack Overflow survey, **Dart, the object-oriented language used to develop apps in Flutter, is ranked 7th in ‘the most loved programming language by developers’ category**, just above C#, Swift, and JavaScript. You can check out Quincy Larson’s analysis on the Stack Overflow survey here.

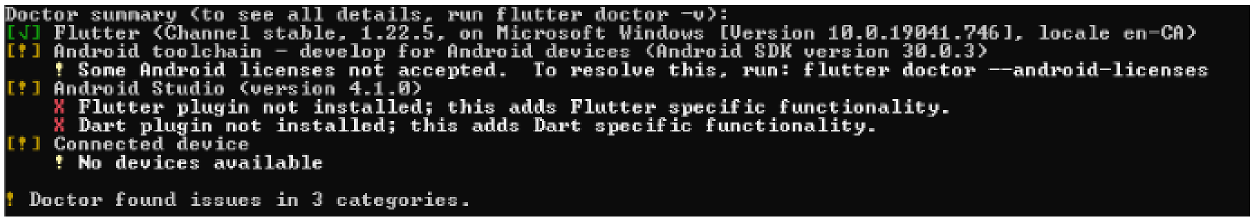
The winning trait of Flutter is of course it’s capability in both Android and IOS development using one codebase, but the emphasis on simplicity is much to be applauded as well. And what I mean by that is the concept of wrapping widgets within widgets to piece together an App that is both beautiful to look at and friendly to use.

Follow along if you want to get started with Flutter and Dart.

Installation

1. Install Flutter here. Remember where in the PC you installed it.

2. Run Flutter doctor in the flutter console with the command `flutter doctor`. This will complete your download and tell you there are issues with set up which has to do with Android Studio not being installed.

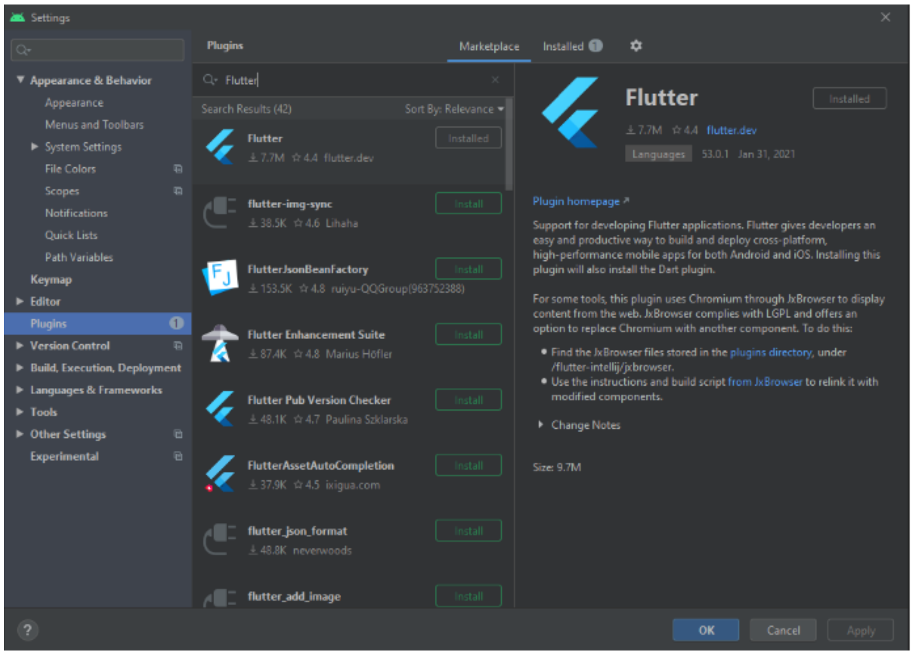


```
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 1.22.5, on Microsoft Windows [Version 10.0.19041.746], locale en-CA)
[!] Android toolchain - develop for Android devices (Android SDK version 30.0.3)
    ! Some Android licenses not accepted.  To resolve this, run: flutter doctor --android-licenses
[!] Android Studio (version 4.1.0)
    X Flutter plugin not installed; this adds Flutter specific functionality.
    X Dart plugin not installed; this adds Dart specific functionality.
[!] Connected device
    ! No devices available

! Doctor found issues in 3 categories.
```

3. Install Android Studio here.

4. Run Android Studio. Go to *File> Settings> Plugins* and search for Flutter. Install Flutter plugin, which includes Dart, making it very convenient for users since there is no need to install Dart as a separate SDK.

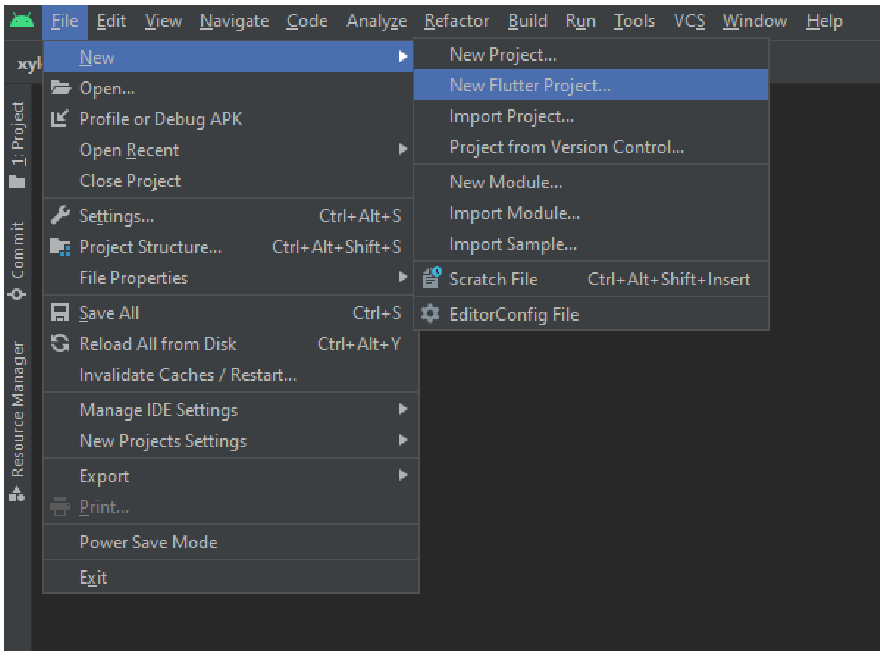

Settings
Appearance & Behavior
Appearance
Menus and Toolbars
System Settings
File Colors
Scopes
Notifications
Quick Lists
Path Variables
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Other Settings
Experimental
Plugins
Marketplace
Installed
Flutter
Search Results (42)
Sort By: Relevance
Flutter 7.7M 4.4 flutter.dev Installed
flutter-img-sync 38.5K 4.6 Lihaha Install
FlutterJsonBeanFactory 153.5K 4.8 ruiyu-QQGroup(963752388) Install
Flutter Enhancement Suite 87.4K 4.8 Marius Höfler Install
Flutter Pub Version Checker 48.1K 4.7 Paulina Szklarska Install
FlutterAssetAutoCompletion 37.9K 4.5 ixigua.com Install
flutter_json_format 48.8K neverwoods Install
flutter_add_image Install
Flutter
Installed
7.7M 4.4 flutter.dev
Languages 53.0.1 Jan 31, 2021
Plugin homepage
Change Notes
Size: 9.7M
OK
Cancel
Apply

Support for developing Flutter applications. Flutter gives developers an easy and productive way to build and deploy cross-platform, high-performance mobile apps for both Android and iOS. Installing this plugin will also install the Dart plugin.

For some tools, this plugin uses Chromium through JxBrowser to display content from the web. JxBrowser complies with LGPL and offers an option to replace Chromium with another component. To do this:

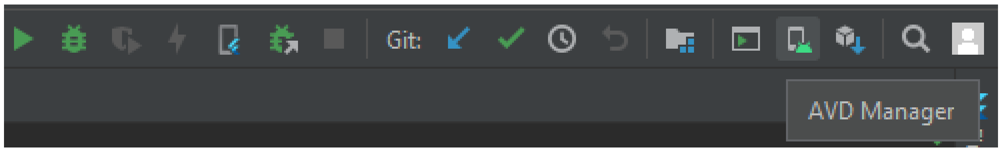
- Find the JxBrowser files stored in the plugins directory, under /flutter-intellij/jxbrowser.
- Use the instructions and build script from JxBrowser to relink it with modified components.

5. Start a new Flutter project. Go to *File>New>New Flutter Project*.



Select Flutter Application when prompted. Next, name your project. If the Flutter SDK path is empty, you will have to input it by locating where you installed Flutter in Step 1.
Then, set your package name or leave it as *com.example.flutterapp*.

6. Ready your Android emulator, which will allow you to see what your app looks like when rendered on a mobile device. Click AVD Manager icon (the 4th icon counting from the right) inside Android Studio:



You can choose which device and OS you want to emulate. **Alternatively, you could also use your real phone as the emulator.** Connect your device with a USB cord to your computer. Find ‘Build number’ in your device and tap it several times until a prompt tells you that you have unlocked ‘Developer options’.

For Samsung, go to *Settings>About phone>Software information>Build number.* When ‘Developer options’ has unlocked, it will show up in *Settings*.

If prompted whether to allow Computer access, click ‘OK’. After you should be able to see your mobile device as one of the choices to render in on the top of Android Studio.

Set up is done!

Building an App

We need to understand what MaterialApp and widgets are.

MaterialApp is a component that gives us access to elements that beautifies our app such as color palettes. Implementing the MaterialApp class allows us to use Material Design, a Google UI design system.

Widgets are the pieces that form the backbone of our app and have properties that further alternates their functionality or appearance. In the end you will have something called a widget tree, which is just a diagram that describes the network of widgets.

Let’s get started on making a page of an app.

1. New Flutter projects always come with a boilerplate code inside the `main.dart` file for you to use. Replace the code with:

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

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

We import `material.dart` to use the MaterialApp class. `Void main` indicates where the computer should start running the program.

2. Add a `Scaffold()` widget within the home location of the app. The Scaffold widget helps us form a layout structure.

3. To add an app bar (the header of an app) with text, we first add the property `appbar` with the widget `Appbar()`. Then add the property `title` with the widget `Text()`. Input any string you want within the Text widget. In this case, I added ‘Our App’.

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

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Our App'),),),),
  );
}
```

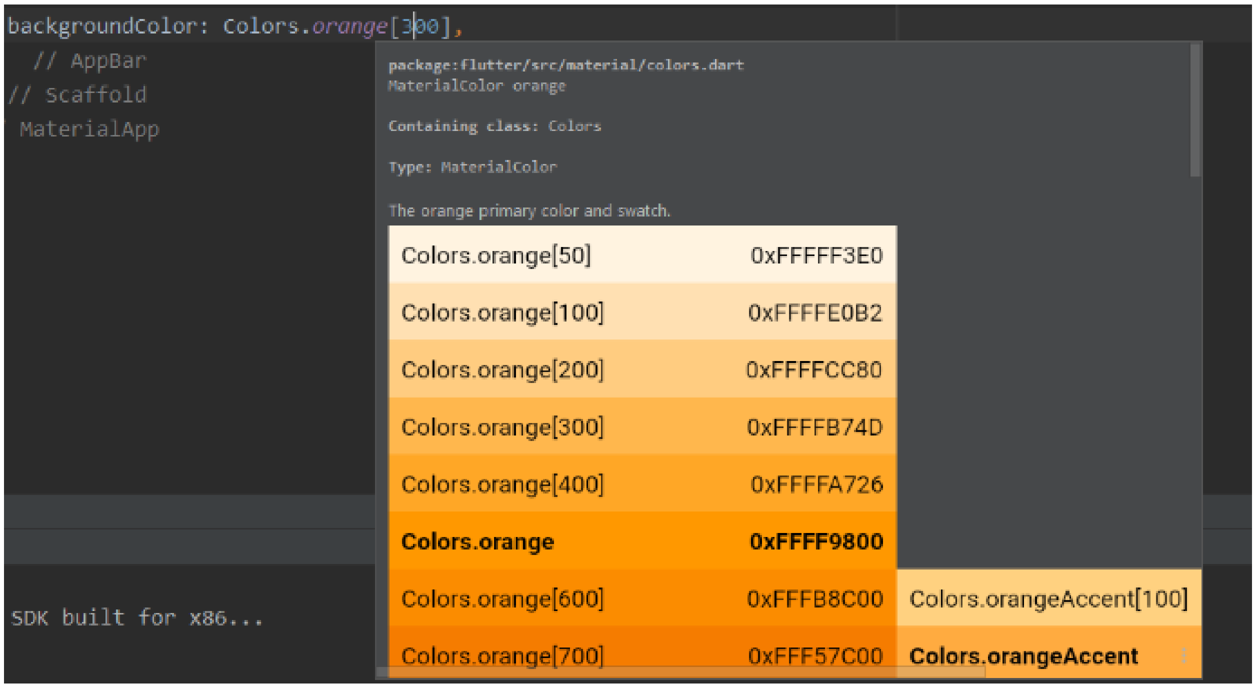
Our code looks very confusing. Flutter has a function to help reformat our code. Put a comma after each backward parentheses and right click to select ‘Reformat Code with dartfmt’.

After reformatting:

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

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Our App'),
        ),
      ),
    ),
  );
}
```

4. Let’s add color to our app bar. After the comma that follows the Text widget, use the property `backgroundColor` with the value `Colors.` with the period. This period will open a color palette in which you can choose a color. Hover over the chosen color, and different shades of the selected color will appear for further customization.



```
backgroundColor: Colors.orange[300],
  // AppBar
// Scaffold
 MaterialApp

SDK built for x86...
```

package:flutter/src/material/colors.dart
MaterialColor orange

Containing class: Colors

Type: MaterialColor

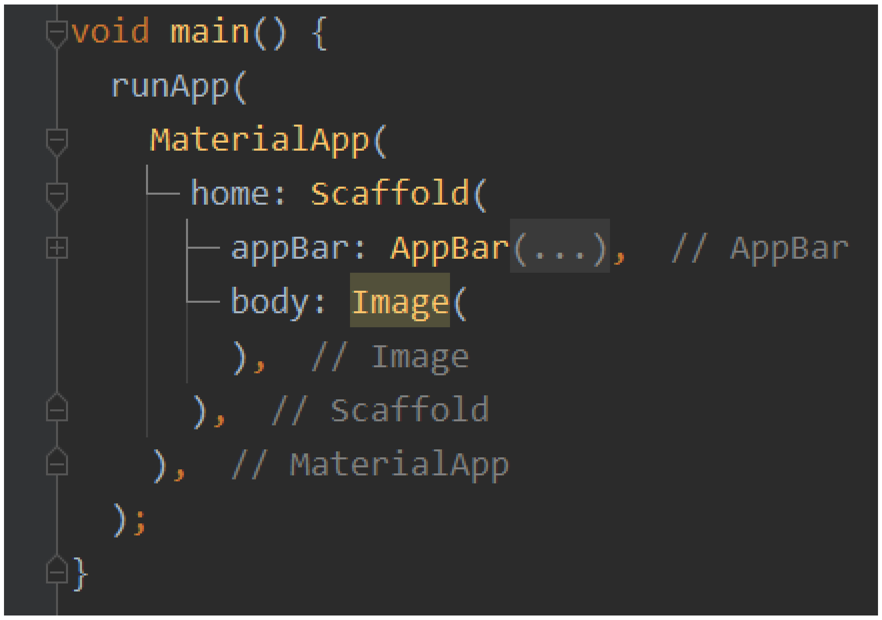
The orange primary color and swatch.

Colors.orange[50]	0xFFFFF3E0
Colors.orange[100]	0xFFFFE0B2
Colors.orange[200]	0xFFFFCC80
Colors.orange[300]	0xFFFFB74D
Colors.orange[400]	0xFFFFA726
Colors.orange	**0xFFFF9800**
Colors.orange[600]	0xFFFB8C00
Colors.orange[700]	0xFFF57C00

Colors.orangeAccent[100]
Colors.orangeAccent

Type the number within the square brackets beside your color. For example, `Colors.orange[300]`.

5. Add an image to the body of the app. First collapse the appbar row. You can do that by pressing the minus sign beside the line. Then, add a `body` property and an `Image()` widget.

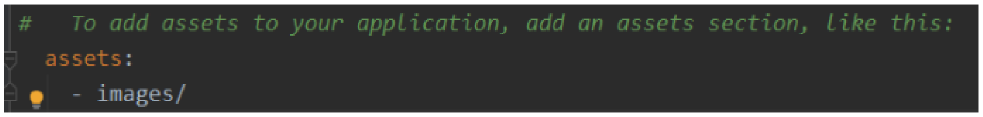


```
void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(...),  // AppBar
        body: Image(
        ),  // Image
      ),  // Scaffold
    ),  // MaterialApp
  );
}
```

Images can be from a local file or from the internet. Use the property `image` and:

- **From internet**: the value `NetworkImage()`. Inside the brackets, input a string with the source url

- **From local file**: the value `Image.asset()`. Make a directory under your project directory named images. Add your image file into the image directory. Go to the `pubspec.yaml` file and input your image path underneath 'assets'. Ensure 2 spaces before the word 'assets' and two spaces before the dash before the image path. Inside the brackets of `Image.asset()`, input a string with the image path.

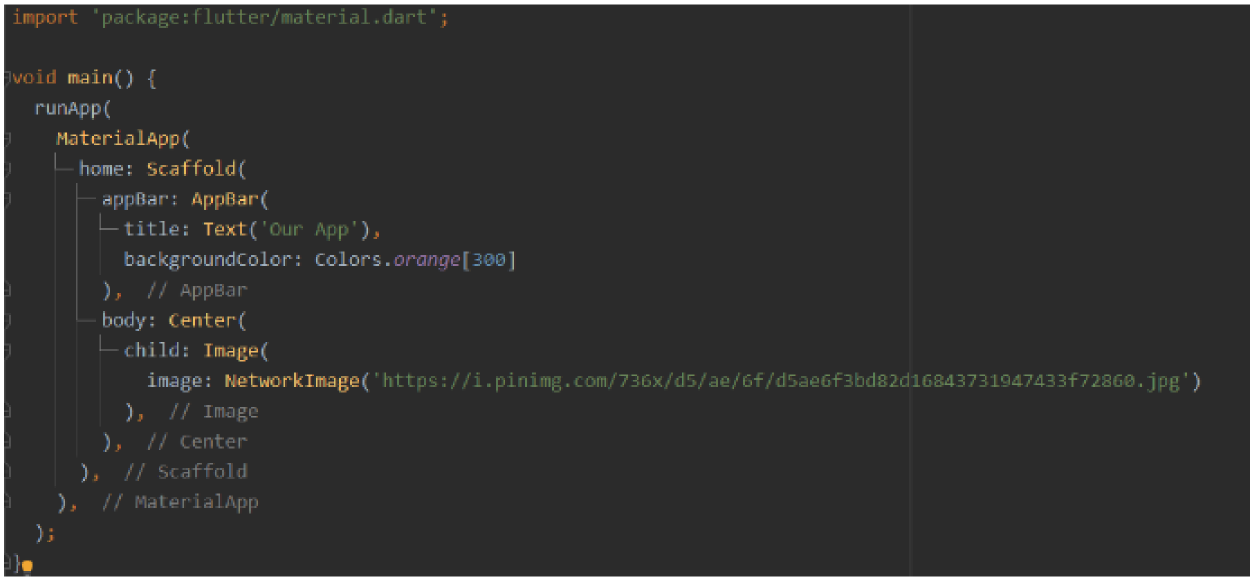


```
#   To add assets to your application, add an assets section, like this:
  assets:
    - images/
```

6. Let's center the image. The quick way to do this is to move your insertion point, a.k.a. typing cursor, to the middle of the `Image()` widget.

Press ALT + Enter. A list of intentions should show up. Select 'Wrap with Widget' then type `Center` to use the `Center()` widget.

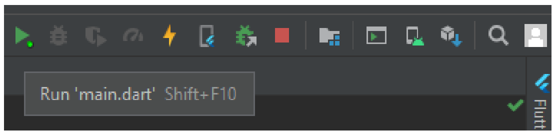
7. Our final code should look something like this depending on the customization:



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

void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Our App'),
          backgroundColor: Colors.orange[300]
        ), // AppBar
        body: Center(
          child: Image(
            image: NetworkImage('https://i.pinimg.com/736x/d5/ae/6f/d5ae6f3bd82d16843731947433f72860.jpg')
          ), // Image
        ), // Center
      ), // Scaffold
    ), // MaterialApp
  );
}
```

8. Run this on the emulator to see how it looks! If you want to run this in your physical device then make sure to pick your mobile device in the drop down at the top of your window. Click the green triangle icon to run.



Congratulations! Your first page is done!