Introducing Flutter

Mobile App Development

Lecture Set – 03

Engr. Ali Asghar Manjotho, Assistant Professor, CSE-MUET

What is Flutter?

Flutter is an opensource framework that allows you to build native cross-platform apps with one programming language and codebase.

Flutter SDK
(Software Development Kit)

Tools to compile your code to native machine language.

A Framework / Widget Library

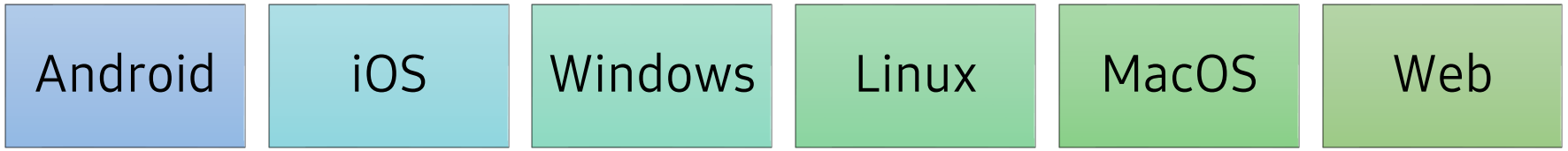
Reusable UI building blocks, utility functions, and packages.

What is Flutter?

- Flutter is Google's open-source technology for creating mobile, desktop, and web apps with a single codebase. Unlike other popular solutions, Flutter is not a framework or library; it's a complete SDK – software development kit.
- A library is basically a reusable piece of code that you put in your application to perform a particular function.
- A framework is a structure that provides you with a skeleton architecture for building software. It's a set of tools that serves as a foundation for your app, requiring you to fill in the blanks with your code to complete the entire structure and get the desired functionality.
- An SDK has a much wider scope as it's a collection of tools, including libraries, documentation, APIs, sometimes frameworks, and more, giving you all you need for software development. And that's the case with Flutter — it already contains everything necessary to build cross-platform applications.

Flutter Target Platforms

Flutter allows you to build apps for multiple platforms from single codebase using Dart programming language.



Dart Programming Language

Dart is client-optimized, object-oriented programming language by Google.

Object-Oriented
&
Strongly Typed

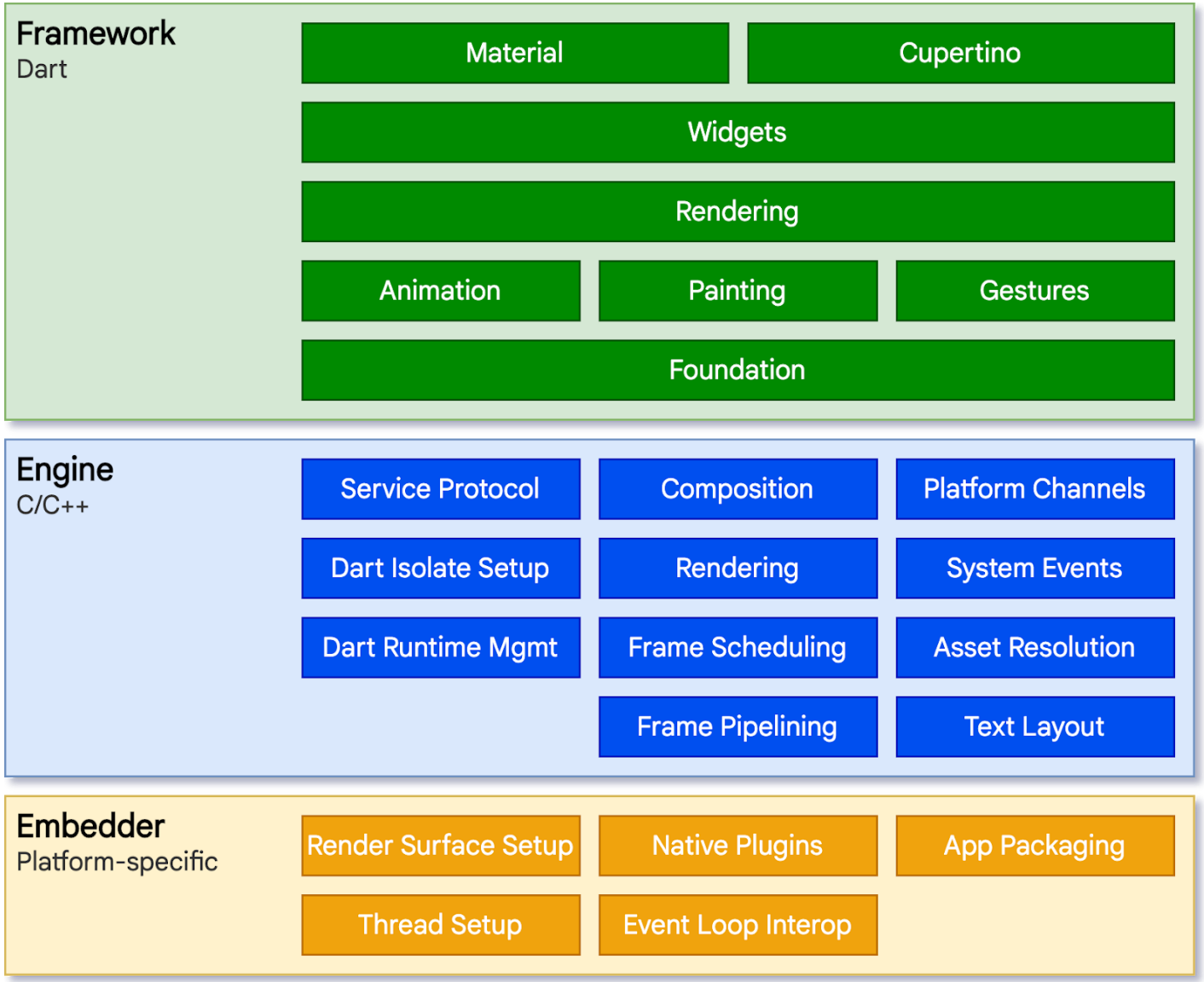
Syntax is like a mixture of Java, C#, and JavaScript.

Dart is capable of compiling into native code for mobile and desktop, as well as into JavaScript

Flutter SDK

Since Flutter is a full-fledged SDK, it includes:

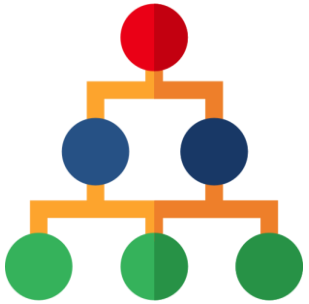
- a rendering engine
- ready-made widgets
- testing APIs,
- integration APIs, etc.



Flutter SDK

- The three main architectural layers of Flutter are:
  - An embedder that uses a platform-specific language and makes the app run on any OS;
  - An engine written in C/C++ that provides a low-level implementation of Flutter's core APIs. That includes graphics (through Skia 2D graphics library), text layout, file and network I/O, accessibility support, plugin architecture, and a Dart runtime and compile toolchain; and
  - A framework based on the Dart programming language. Its implementation is optional, but it provides a rich set of libraries that can be divided into layers: basic foundational classes, rendering layer, widget layer, and Material/Cupertino libraries.

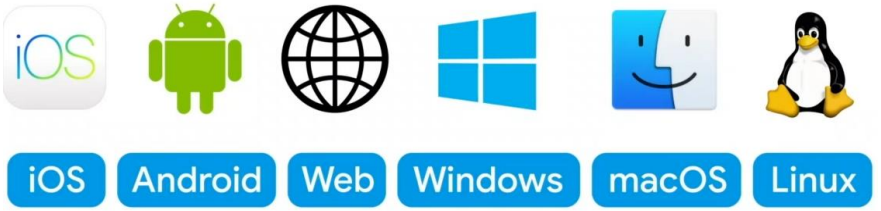
Flutter Architecture



UI as code:

Build a Widget Tree

No visual editor, code only.



Embraces platform differences

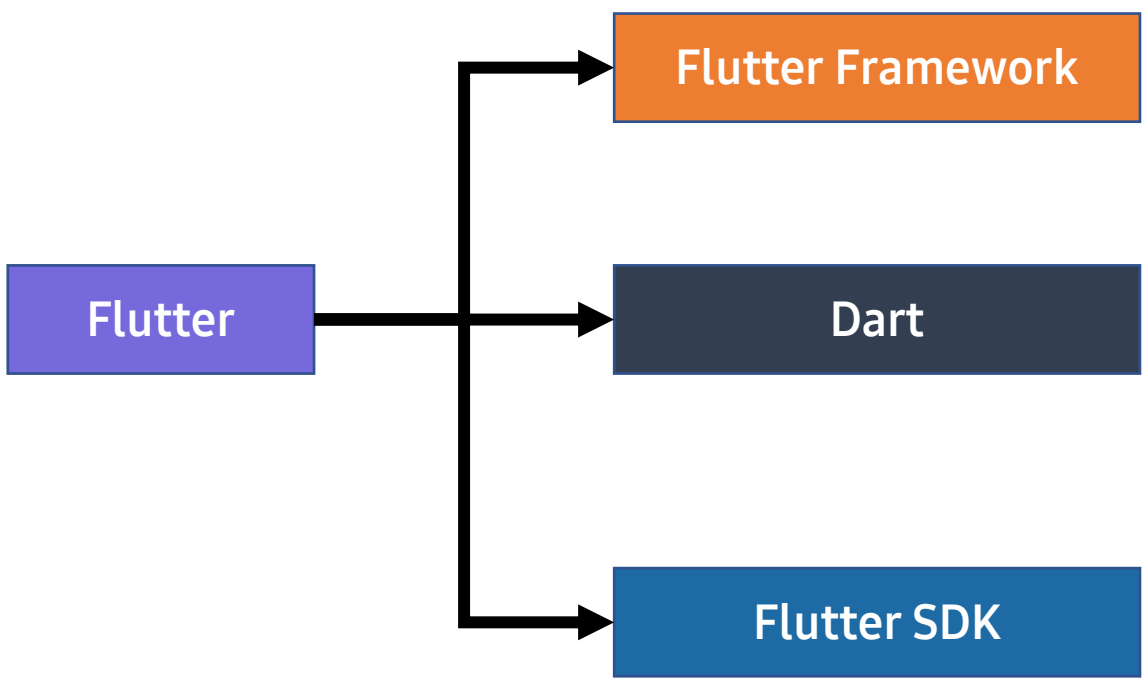
Checks the target platform and compile to native code.

Single codebase

Single code, single development team for every platform.

Flutter vs Dart

Flutter is a framework; dart is a language.

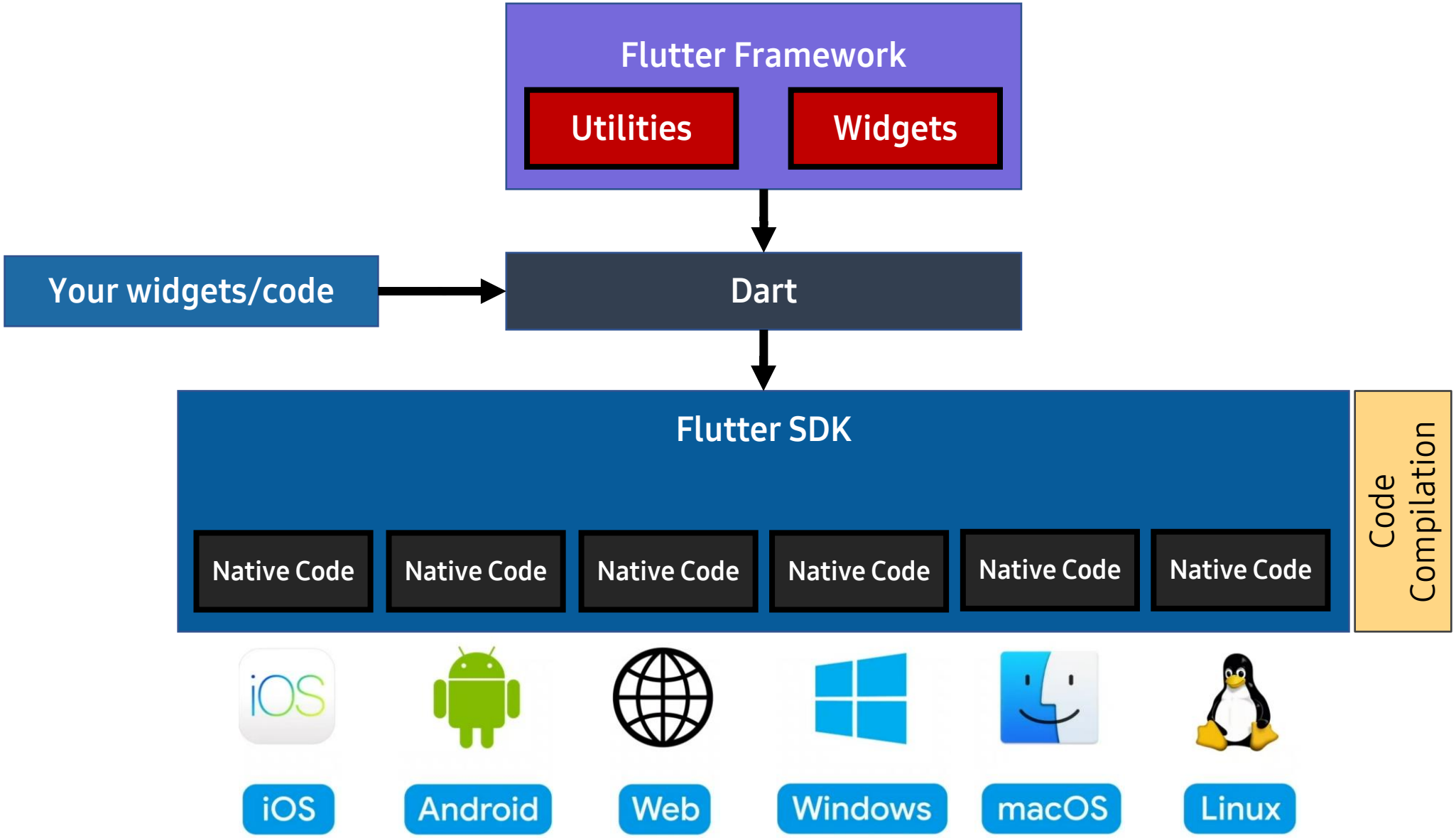


is a framework (set of features), provides UI elements (widgets) and utility functions.

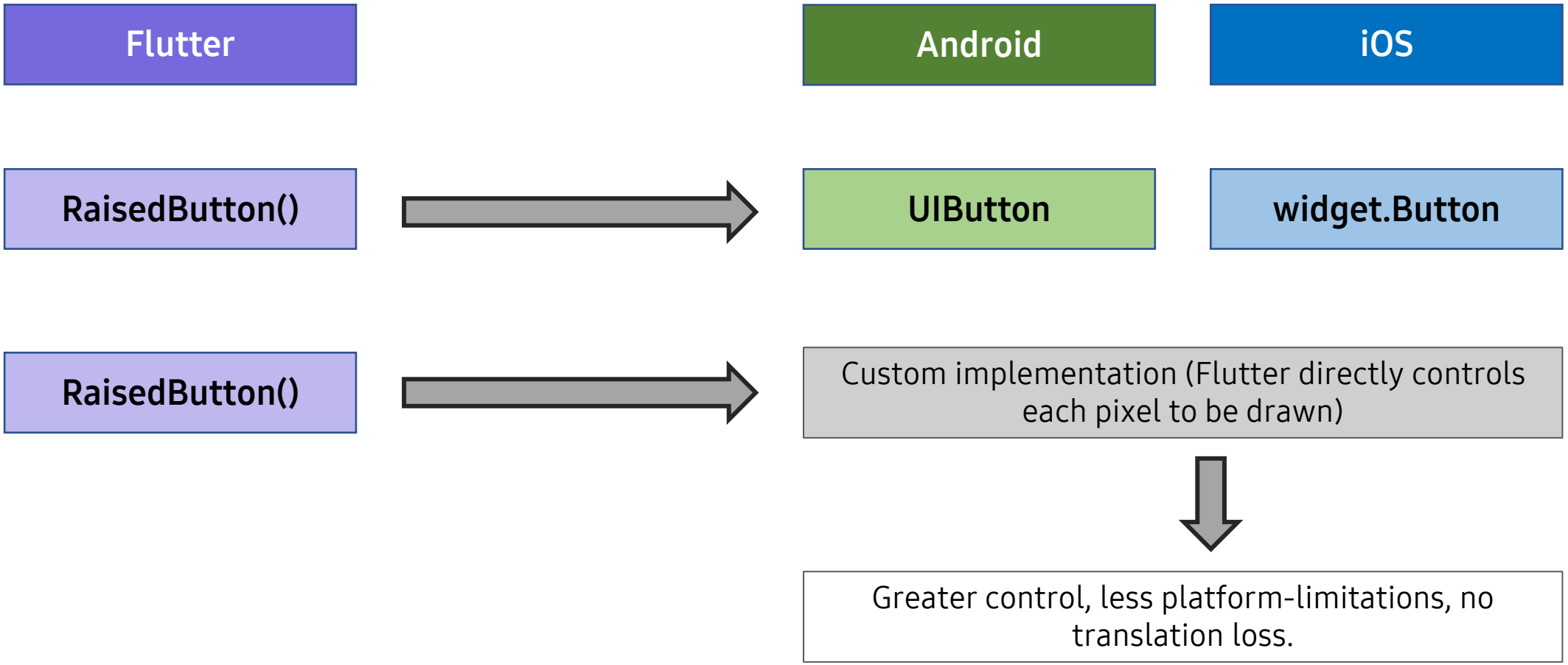
is a programming language

provides set of tools for compilation.

Flutter Code Compilation



Flutter does NOT use platform primitives



Flutter Pros

- Flutter widgets for fast UI coding.
- Simple and effective tool targeted at Java programmers (Dart).
- Ahead-of-Time (AOT) and Just-in-Time (JIT) compilation types.
- No need for XML files (Android Studio).
- No need for intermediate bridges (Dart directly compiles to native code).
- Mildest learning curve for an easy start.
- Great documentation and learning resources.
- Flutter developers community for knowledge sharing.
- Google as a guarantee of long-term support.
- Hot reload function for instantaneous updates.
- High performance and Portability.
- Internationalization and accessibility

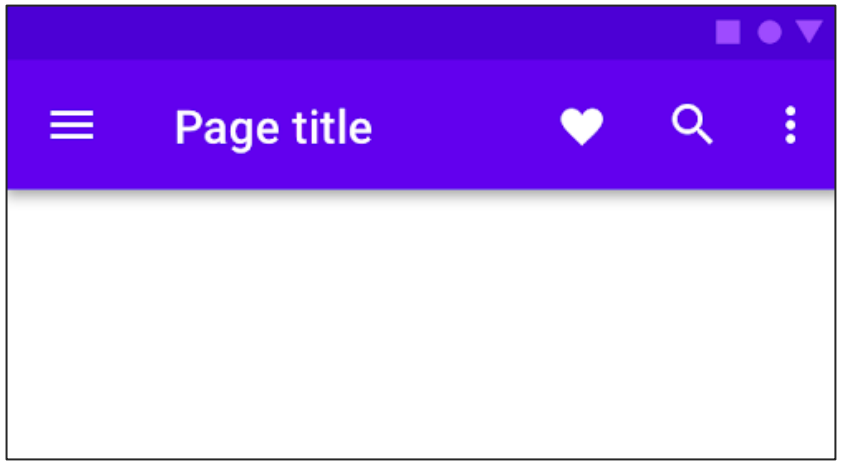
Flutter Cons

- Lack of third-party libraries.
- Relatively low adoption of Dart.
- Flutter app size.

- Since Flutter has the built-in widgets, a minimum app size exceeds 4MB, which is definitely bigger than native Java (539KB) and Kotlin (550KB) apps – and that's for the bare minimum app. To be fair, its competitors share the same problem, and probably even more so – the release version in Xamarin will take almost 16MB and 7MB in React Native.

Material Design

Flutter by default uses material design system



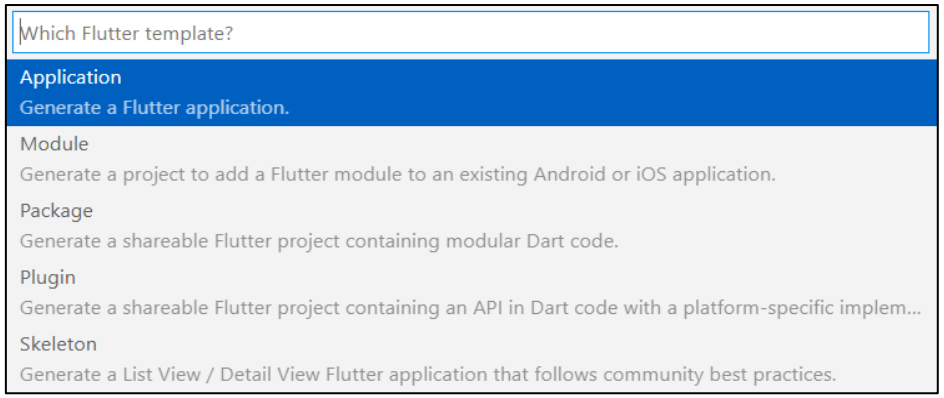
Material is a Design System created (and heavily used) by Google.

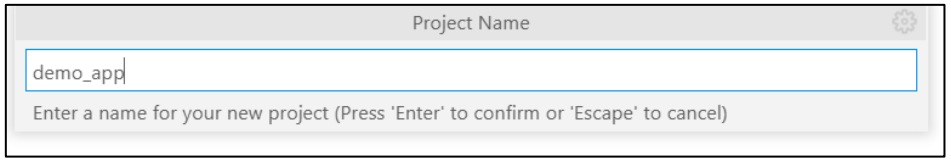
It is NOT Google's style for Everyone! It is indeed highly customizable (and works on iOS devices, too)

Material Design is built into Flutter, but you also find Apple-styled (Cupertino) widgets

Creating New Flutter Project

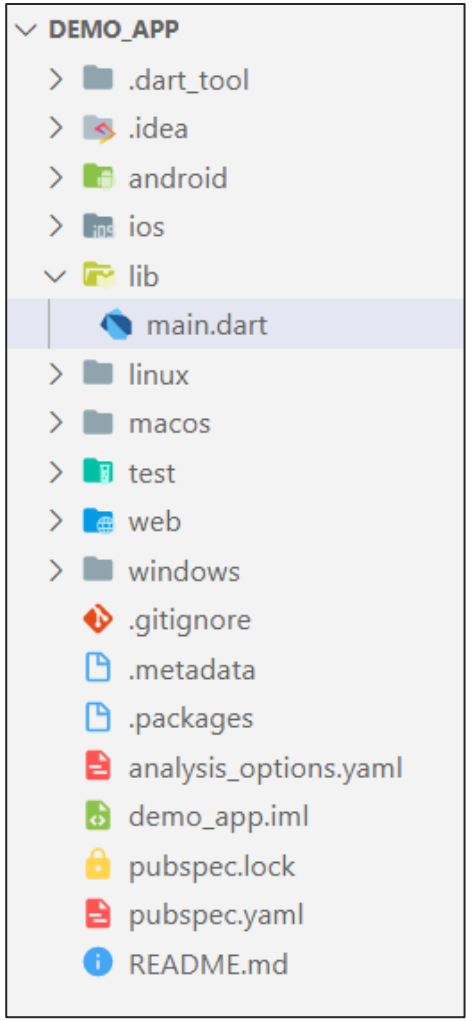
- Launch Visual Studio Code.
- Press Ctlr Shift P to open command palette.
- Type flutter and select Flutter: New Project.
- Select Application.
- Select the destination location for the project.
- Enter the project name e.g. “demo_app”.



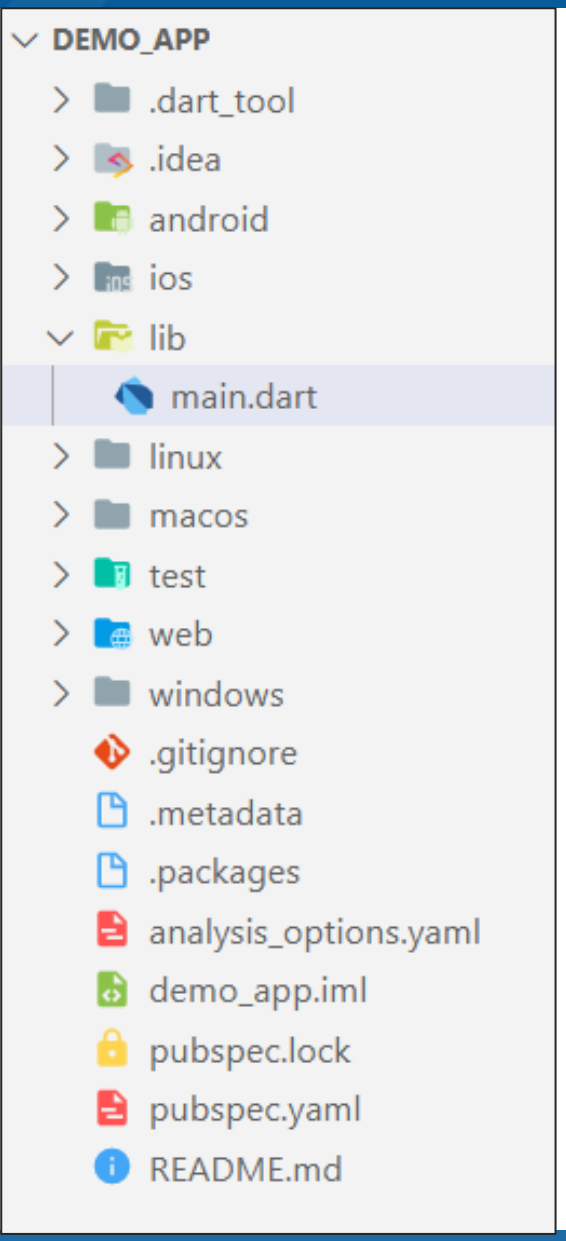


Flutter Project Folder Structure

Whenever we create a new flutter project, the following default directory structure is generated.



Flutter Project Folder Structure



contains files used by various Dart tools like pub.
holds configuration for Android Studio.
contains an Android native app project and is used when building your Flutter application for Android.
contains an XCode iOS native project and is used when building your Flutter application for iOS.
directory containing all the user code in dart language.
entry point dart code file containing main function.
contains a Linux native app project and is used when building your Flutter application for Linux OS.
contains XCode macOS native project and is used when building your Flutter application for macOS.
dart files for managing flutter test code.
contains a browser native web app project and is used when building your Flutter application as web application.
contains a windows native app project and is used when building your Flutter application for Windows OS.
text file containing a list of files, file extensions, and folders which should be ignored when working with Git.
managed by Flutter automatically and is used to track properties of the Flutter project.
file contains automatically generated content by the Flutter and contains a list of dependencies for your Flutter project.
contains set of rules that encourage good coding practices for applications, packages, and plugins in flutter project.
file is always named according to the project's name and contains further settings of the Flutter project.
contains the version of each dependency and packages used in the flutter application.
project's configuration file you'll use a lot when working with your Flutter project.
markdown file is used to describe your application in the GitHub repository.

pubspec.yaml File

- Every pub package needs some metadata so it can specify its dependencies.
- Pub packages that are shared with others also need to provide some other information so users can discover them.
- All of this metadata goes in the package's pubspec: a file named pubspec.yaml that's written in the YAML language.

Flutter Project Folder Structure

- All our dart code resides under the lib folder.
- When building large Flutter apps, one of the first things we should decide is how to structure our project.
- This ensures that the entire team can follow a clear convention and add features in a consistent manner.
- Two of the common approaches for structuring our project are:

Feature-First

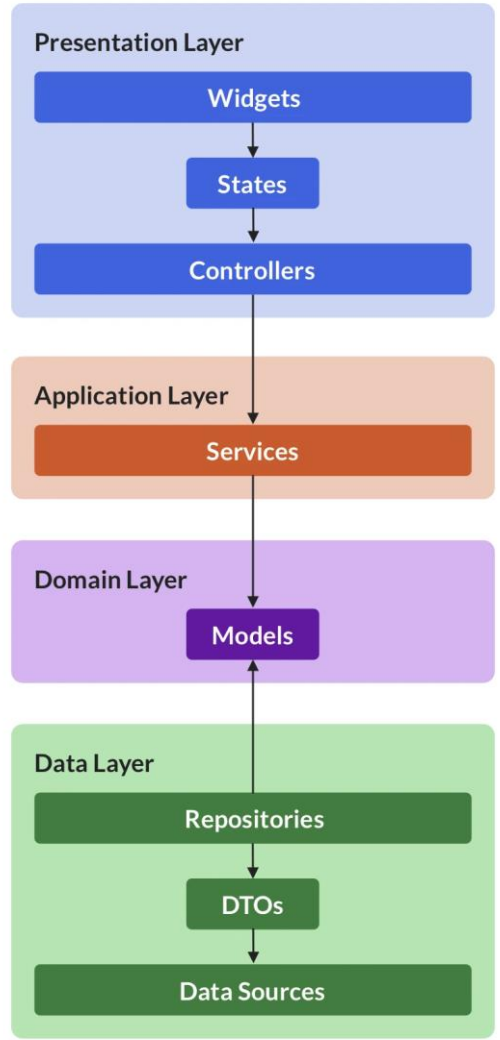
Layer-First

Flutter App Architecture

- In practice, we can only choose a project structure once we have decided what app architecture to use.
- This is because app architecture forces us to define separate layers with clear boundaries. And those layers will show up somewhere as folders in our project.
- Flutter app architecture is made of four distinct layers, each containing the components that our app needs:

Flutter App Architecture

Presentation	• Widgets, states, and controllers
Application	• Services
Domain	• Models
Data	• Repositories, data sources, and DTOs (data transfer objects)

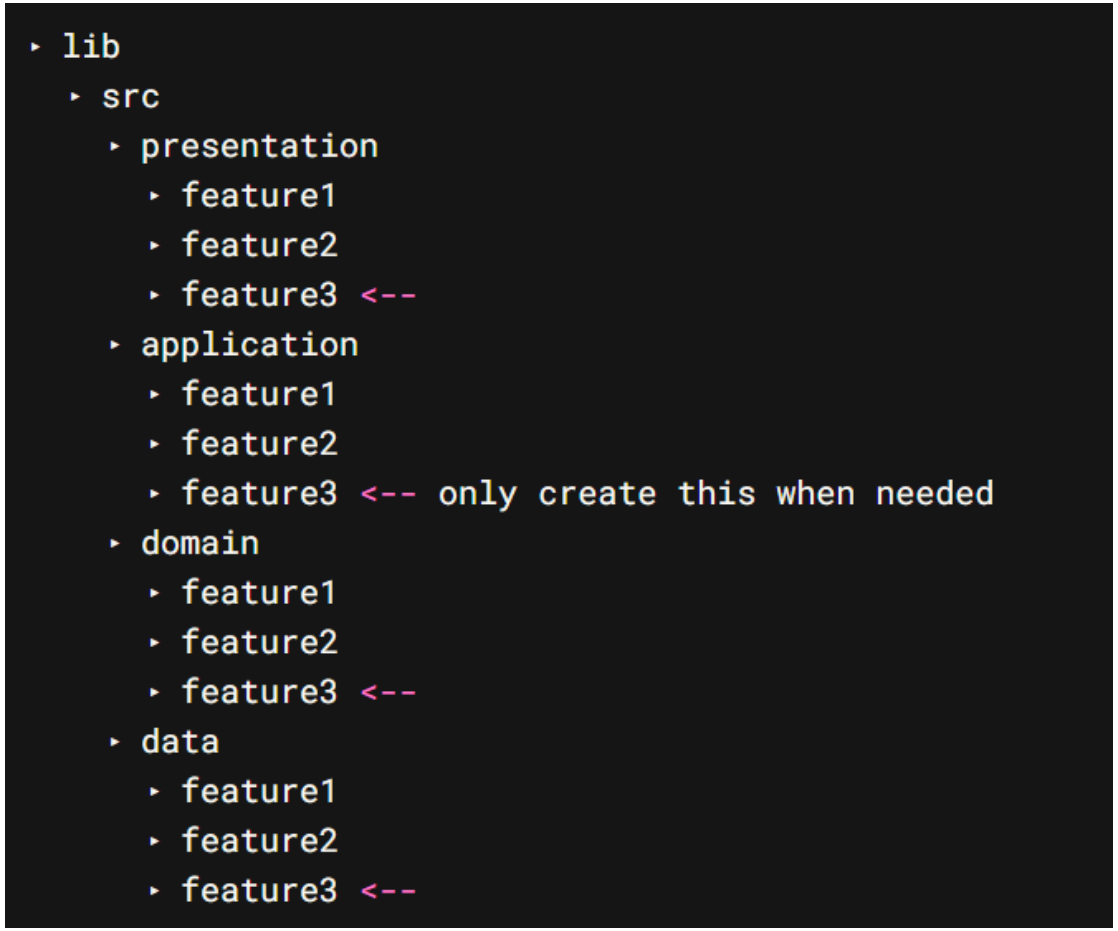


Layer-First Approach

- In layer-first approach we maintain only single folder for each layer.
- Each feature is placed separately as subfolder within each layer directory.
- Suppose we have only two features in the app, our project structure may look like:

Layer-First Approach

- if we want to add feature3, we need to add a feature3 folder inside each layer and repeat the process.



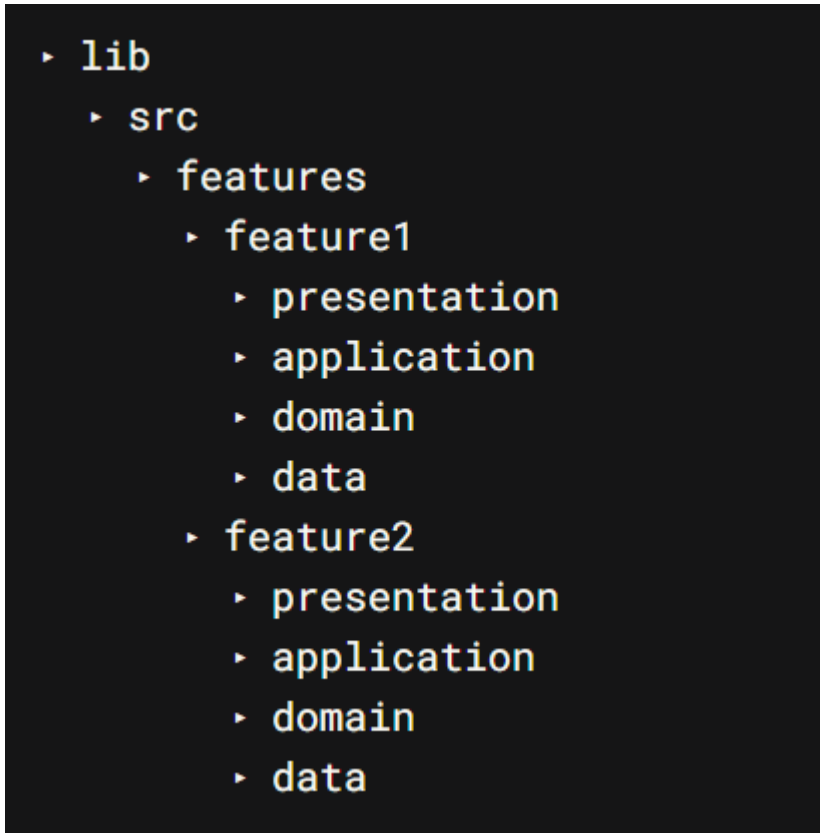
Layer-First Approach (Drawbacks)

Drawbacks

- This approach is easy to use in practice but doesn't scale very well as the app grows.
- For any given feature, files that belong to different layers are far away from each other. And this makes it harder to work on individual features because we have to keep jumping to different parts of the project.
- If we decide that we want to delete a feature, it's far too easy to forget certain files, because they are all organized by layer.
- For these reasons, the feature-first approach is often a better choice when building medium/large apps.

Feature-First Approach

- The feature-first approach demands that we create a new folder for every new feature that we add to our app.
- Inside that folder, we can add the layers themselves as sub-folders.
- Using the same example, we would organize our project (as to right).
- This approach is more logical because we can easily see all the files that belong to a certain feature, grouped by layer.



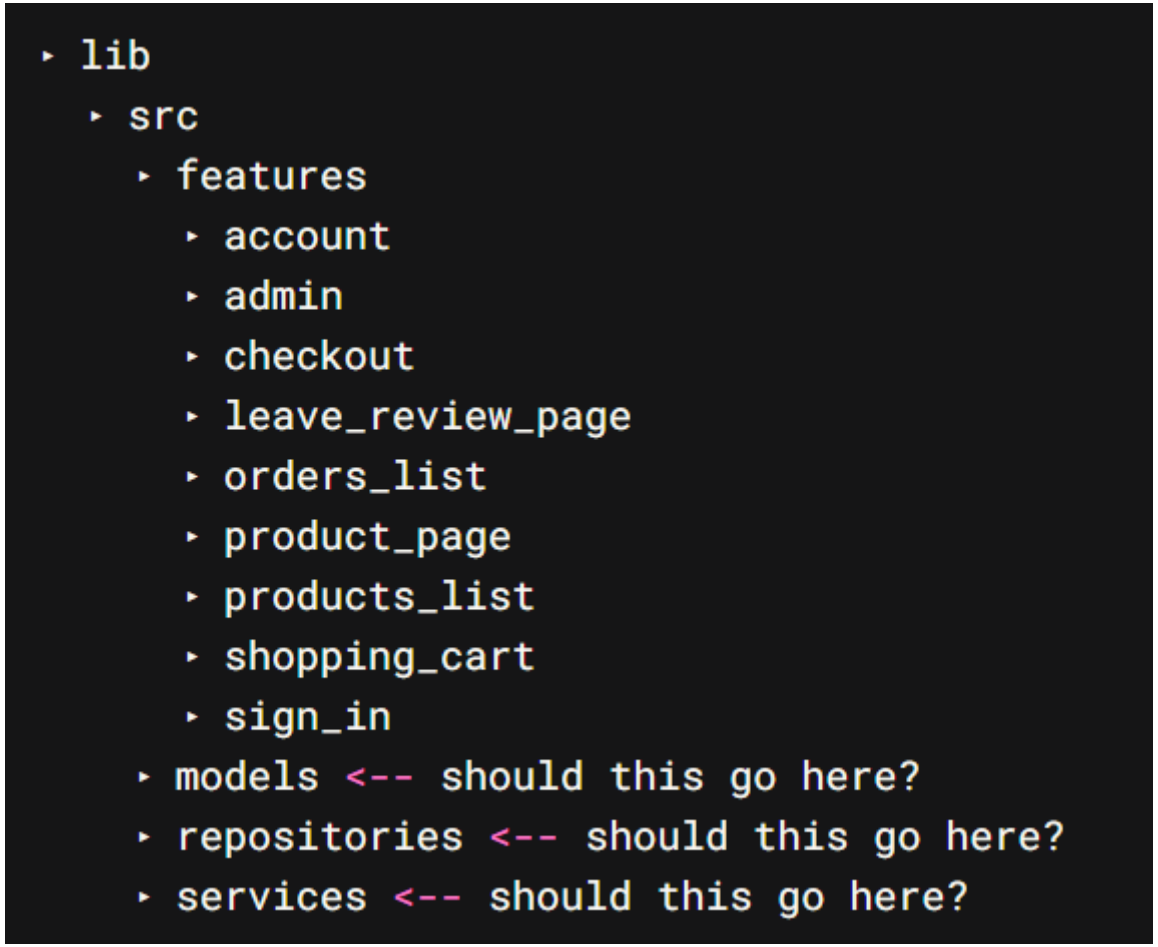
Feature-First Approach

- In comparison to the layer-first approach, there are some advantages:
  - Whenever we want to add a new feature or modify an existing one, we can focus on just one folder.
  - If we want to delete a feature, there's only one folder to remove (two if we count the corresponding tests folder)
- So, it would appear that the feature-first approach wins hands down!

Shared Components in Features

- What if two or more separate features need to share some widgets or model classes?
- In these cases, it's easy to end up with folders called shared or common, or utils.
- If your app has 20 features and has some code that needs to be shared by only two of them, should it really belong to a top-level shared folder?
- What if it's shared among 5 features? Or 10?
- In this scenario, there is no right or wrong answer, and you have to use your best judgement on a case-by-case basis.

Shared Components in Features



What is Feature?

Feature is not about what the user sees, but what the user does

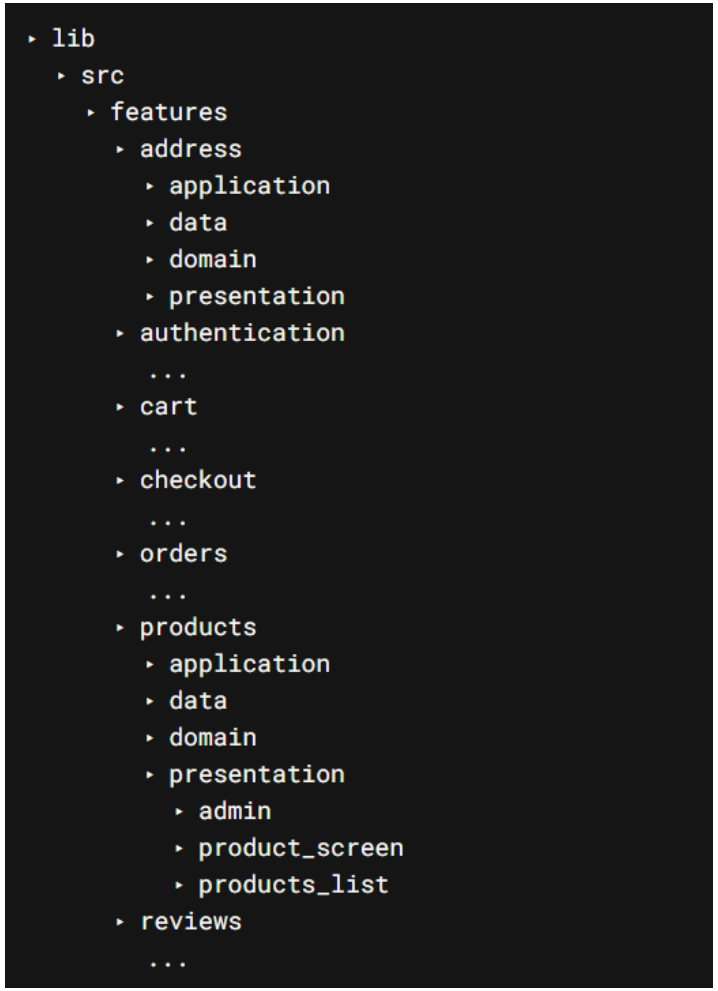
- Feature-first is not about the UI!
- Do not attempt to apply a feature-first approach by looking at the UI. This will result in an "unbalanced" project structure that will bite you later on.

Feature examples:

- Authenticate
- Manage the shopping cart
- Checkout
- View all past orders
- Leave a review

What is Feature?

- A feature is a functional requirement that helps the user complete a given task.



Flutter Project from Scratch

- **Step 01:** Launch VS Code and create a new flutter project.
- **Step 02:** Open the `main.dart` file.
- **Step 03:** Remove everything from the file.

- **Step 04:** Write the import statement at the top, so that we may use flutter SDK classes and libraries.

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

Flutter Project from Scratch

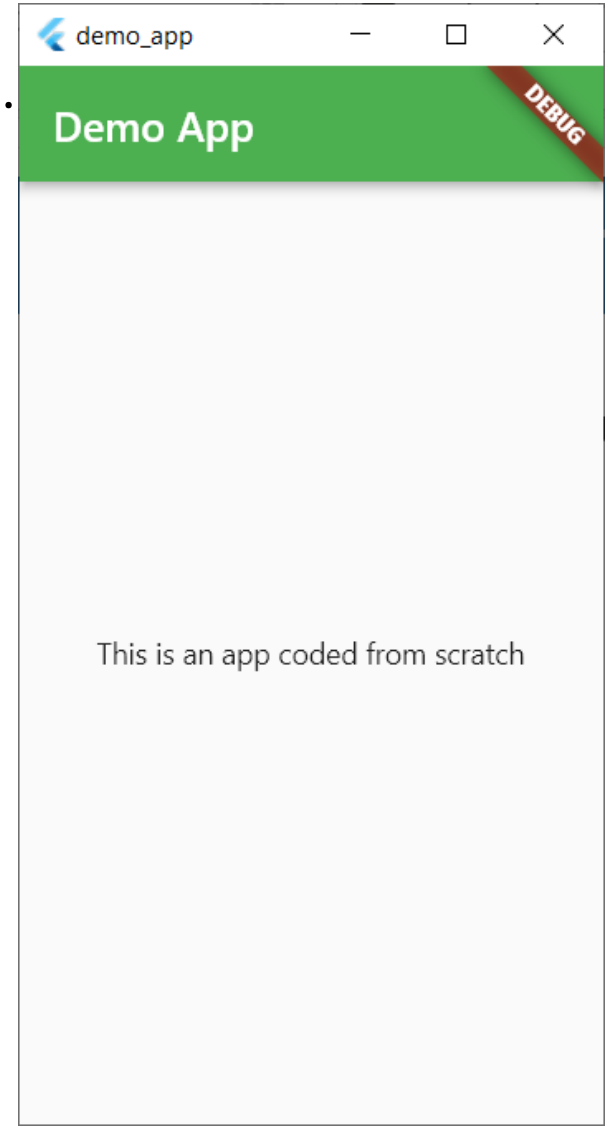
- **Step 05:** In flutter, everything is a widget, create a new widget `MyApp` by extending from a base class widget **`StatelessWidget`**.
- `MyApp` widget overrides the `build` method.
- The build method return a custom widget.

```
class MyApp extends StatelessWidget {

  @override
  Widget build(BuildContext context) {

    return ;

  }
}
```

Flutter Project from Scratch

- **Step 06:** Return a material app widget within the build method.

```
Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Demo App',
      theme: ThemeData(
        primarySwatch: Colors.green,
      ),
      home: Scaffold(
        appBar: AppBar(
          title: Text('Demo App'),
        ),
        body: Center(
          child: Text('This is an app coded from scratch'),
        ),
      ),
    );
  }
```



Flutter Project from Scratch

- **Step 07:** To execute the app, create a main method within `main.dart` file and call `runApp` method along with new instance of `MyApp` widget.
- The main function is the entry point for the flutter applications.

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