Flutter

Caspian Consultancy Services

Disclaimer

The information contained in this presentation may be confidential, proprietary and/or subject to privileged. If you are not the intended recipient, you are hereby notified that any use, reliance on, reference to, review, disclosure or copying of this presentation for any purpose is strictly prohibited.

The D'Factor Trainer

Abang Caspian Abang Thairani

25 Years Experience in IT

12 Years of Mobile Development

ISO7014 Android & iOS Cert

Master Trainer MDeC ICON Program (Mobile Apps)

Train Over 1,000 Developers

Flutter Development 10 Months

Our Training Studio

Our Corporate Training

Our Corporate Training

Our Corporate Training

Let’s Flutter…

WHAT IS FLUTTER?

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

What is Dart Language?

What is Dart for?

Flutter

To develop mobile apps, get the Flutter SDK.

Web

To develop web apps, get the Dart SDK.

Server

To develop command-line or server-side apps, get the Dart SDK.

WHY FLUTTER?

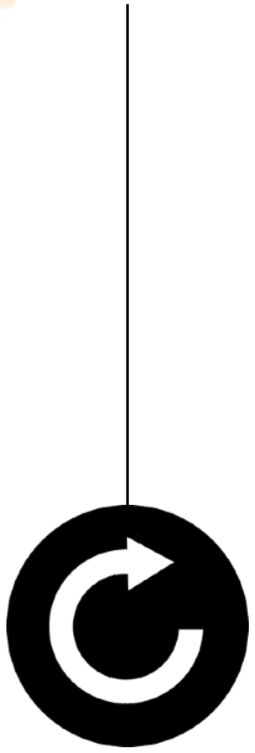
With
Flutter, you can quickly and easily develop beautiful, powerful apps for both Android & iOS, Web and Desktop without the need to learn multiple programming languages or juggle more than one code base.

1 FAST DEVELOPMENT
- Hot Sync
- Full-customisable Re-useable Widgets
- Build Native UI in minutes!

2 EXPRESSIVE & FLEXIBLE UI
- Support Native Material Design
- Layered Architecture
- Fast Rendering
- Flexible Designs

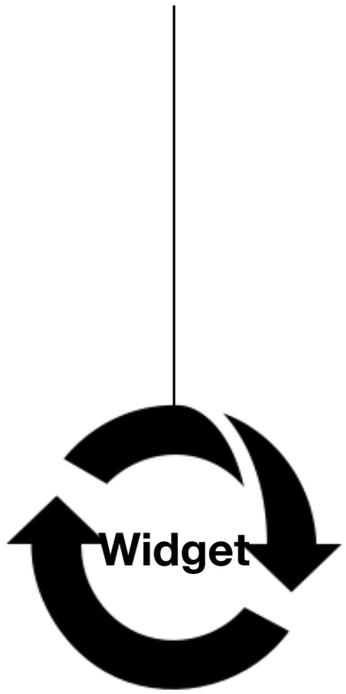
3 NATIVE PERFORMANCE
- Native Speed on iOS & Android

Fast Development



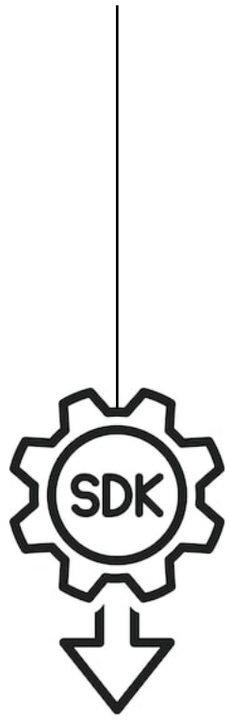
HOT RELOAD

Hot Sync!
Compile changes almost instantly!



REUSABLE WIDGETS

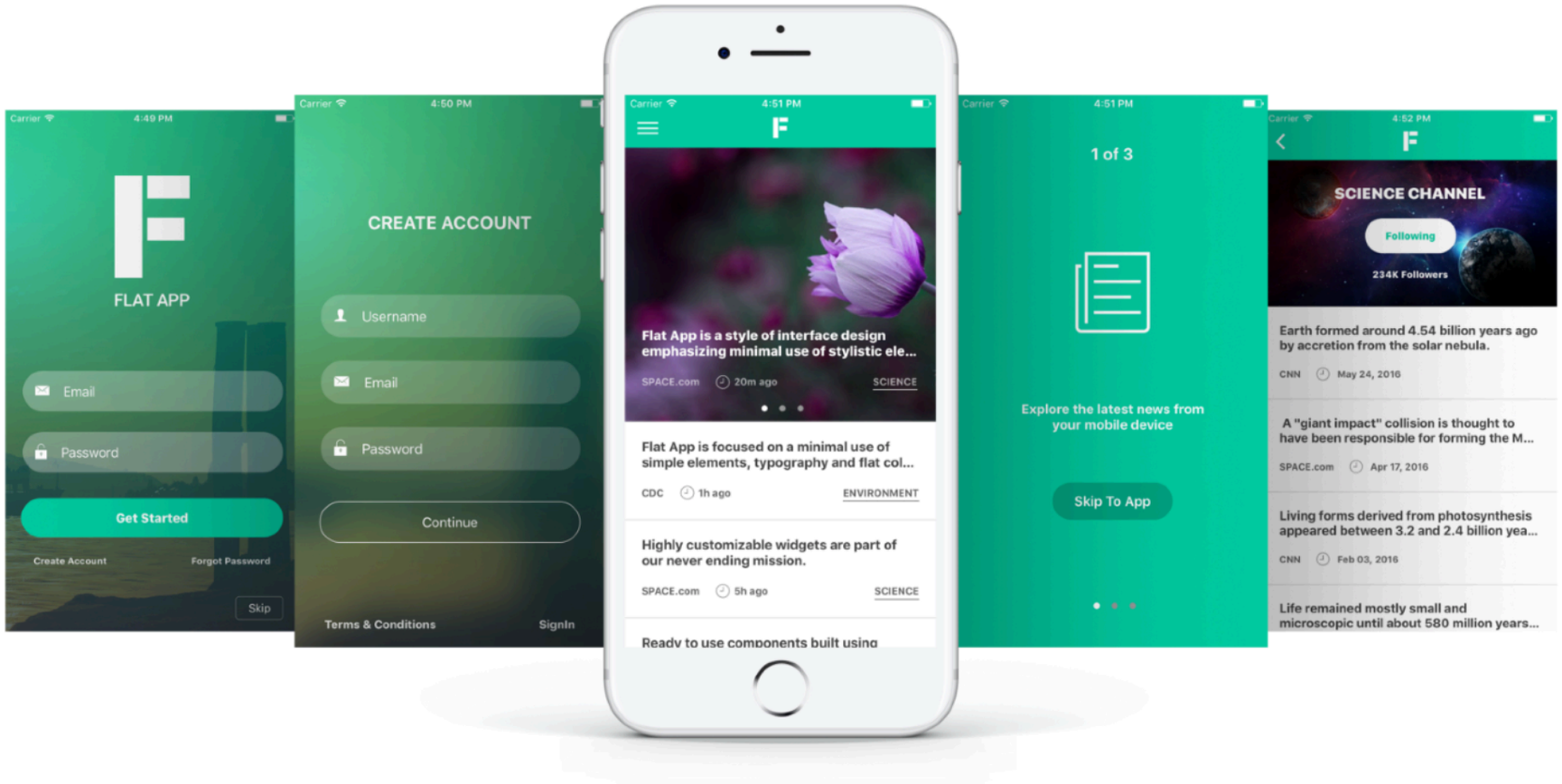
Build Native UI in minutes!
Full-customisable!
Create components called widgets!



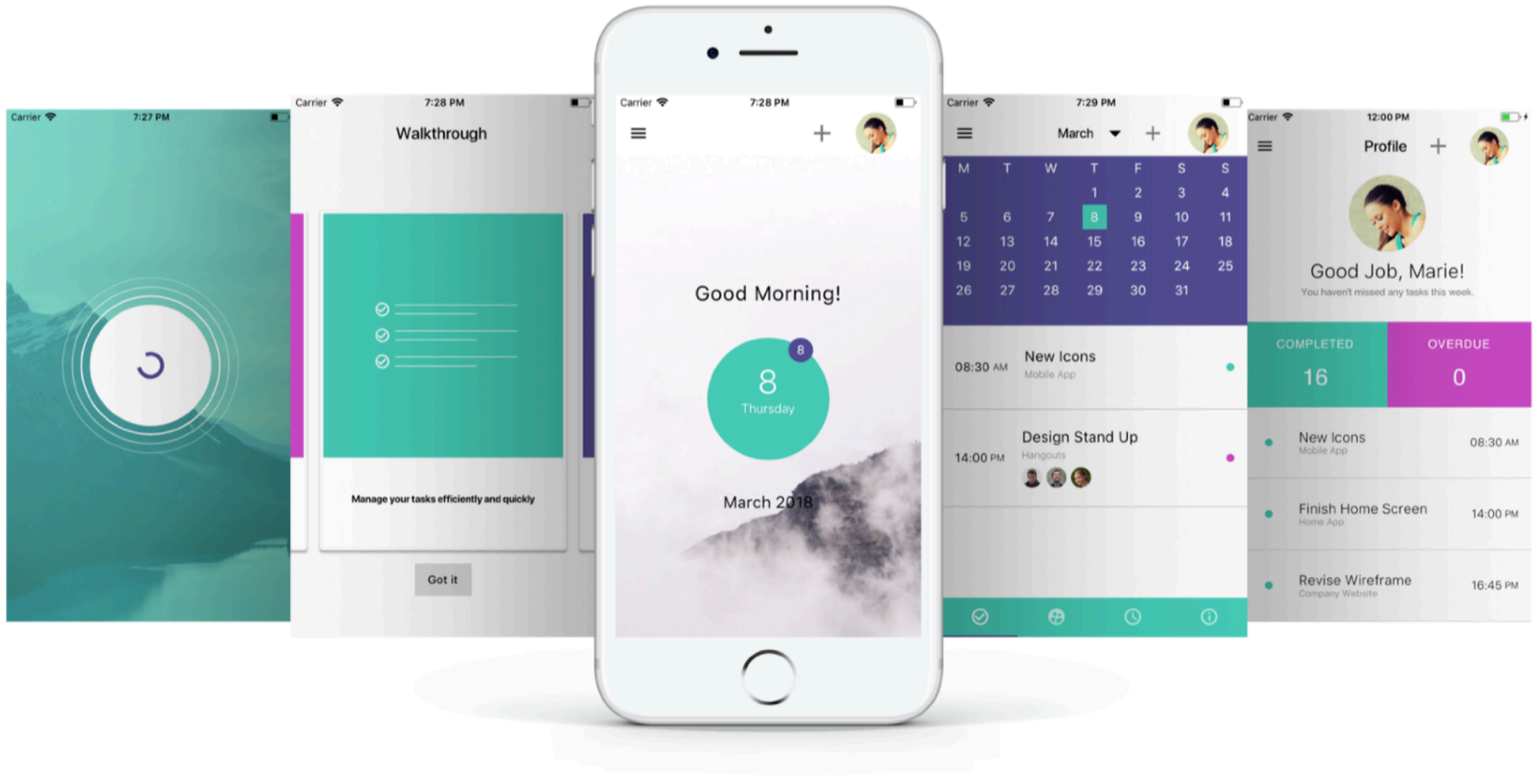
NATIVE FEATURES & SDKs

Make your app come to life with platform APIs, 3rd party SDKs, and native code.
Modern, reactive framework.
Unified app development

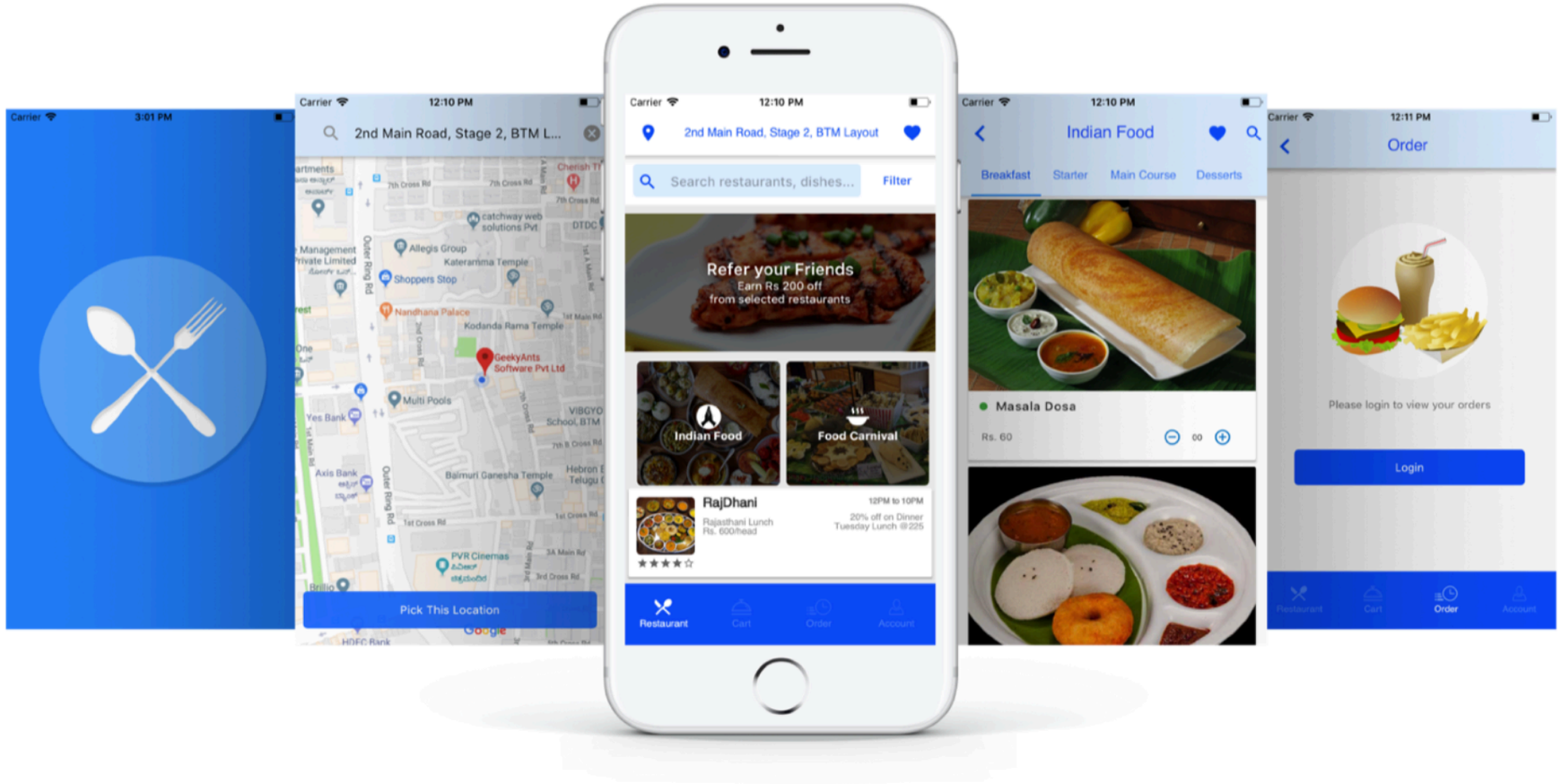
Expressive, Beautiful UIs



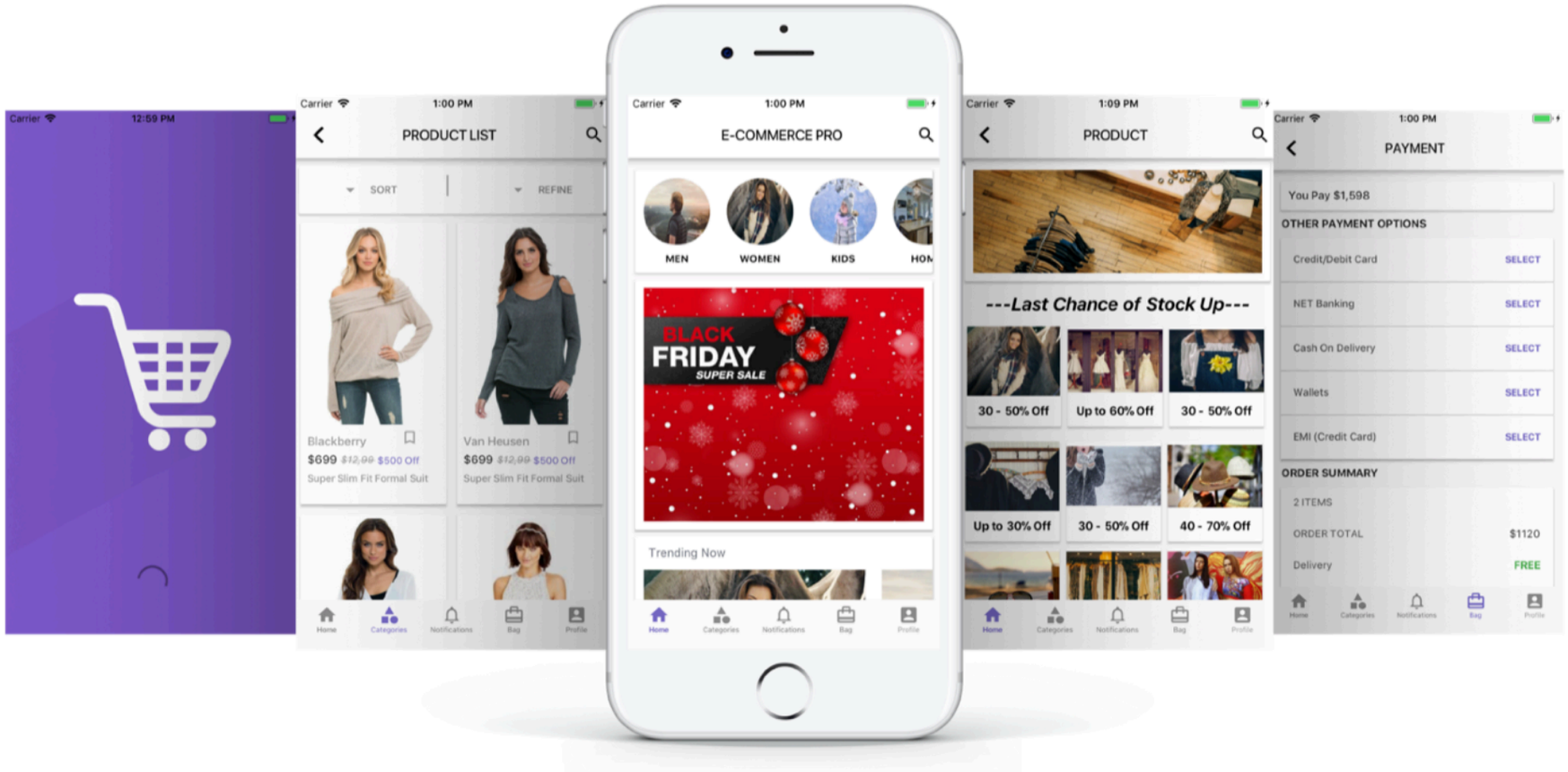
Expressive, Beautiful UIs



Expressive, Beautiful UIs



Expressive, Beautiful UIs



Installation

Windows Installation

Follow the Installation instructions in Flutter.dev website

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

Note that you will have to close and reopen any existing console windows for these changes to take effect.

Installation

Windows Installation

Run `flutter doctor`

From a console window which has the Flutter directory in the path (see above), run the following command to see if there are any platform dependencies you need to complete the setup:

```
C:\src\flutter>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 may need to install or further tasks to perform (shown in **bold** text).

For example:

```
[-] Android toolchain - develop for Android devices
    • Android SDK at D:\Android\sdk
    ✗ Android SDK is missing command line tools; download from https://goo.gl/XxQghQ
    • Try re-installing or updating your Android SDK,
      visit https://flutter.dev/setup/#android-setup for detailed instructions.
```

The following sections describe how to perform these tasks and finish the setup process. Once you have installed any missing dependencies, you can run the `flutter doctor` command again to verify that you've set everything up correctly.

Make sure no errors, Then you're good to go!

Installation

Checking Environment Setup

Run flutter doctor, make sure no errors, Then you're good to go!
if [x] Connected device (0 available) that's normal for 1st setup

```
Last login: Sun Apr 21 23:52:40 on console
You have new mail.
MacBook-Pro:~ Caspian$ flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[✓] Flutter (Channel stable, v1.2.1, on Mac OS X 10.14.4 18E226, locale en-GB)
[✓] Android toolchain - develop for Android devices (Android SDK version 28.0.3)
[✓] iOS toolchain - develop for iOS devices (Xcode 10.2.1)
[✓] Android Studio (version 3.3)
[✓] VS Code (version 1.33.1)
[✓] Connected device (1 available)

• No issues found!
MacBook-Pro:~ Caspian$
```

Fixing PC Installation

Sometimes you encounter errors

Common Error 1:

Android Emulator Does not Start!

Open SDK Manager and Download Intel x86 Emulator Accelerator (HAXM installer) if you haven't. ... In case you get an error like "Intel virtualization technology (vt,vt-x) is not enabled". Go to your BIOS settings and enable Hardware Virtualization. 3) Restart Android Studio and then try to start the AVD again.

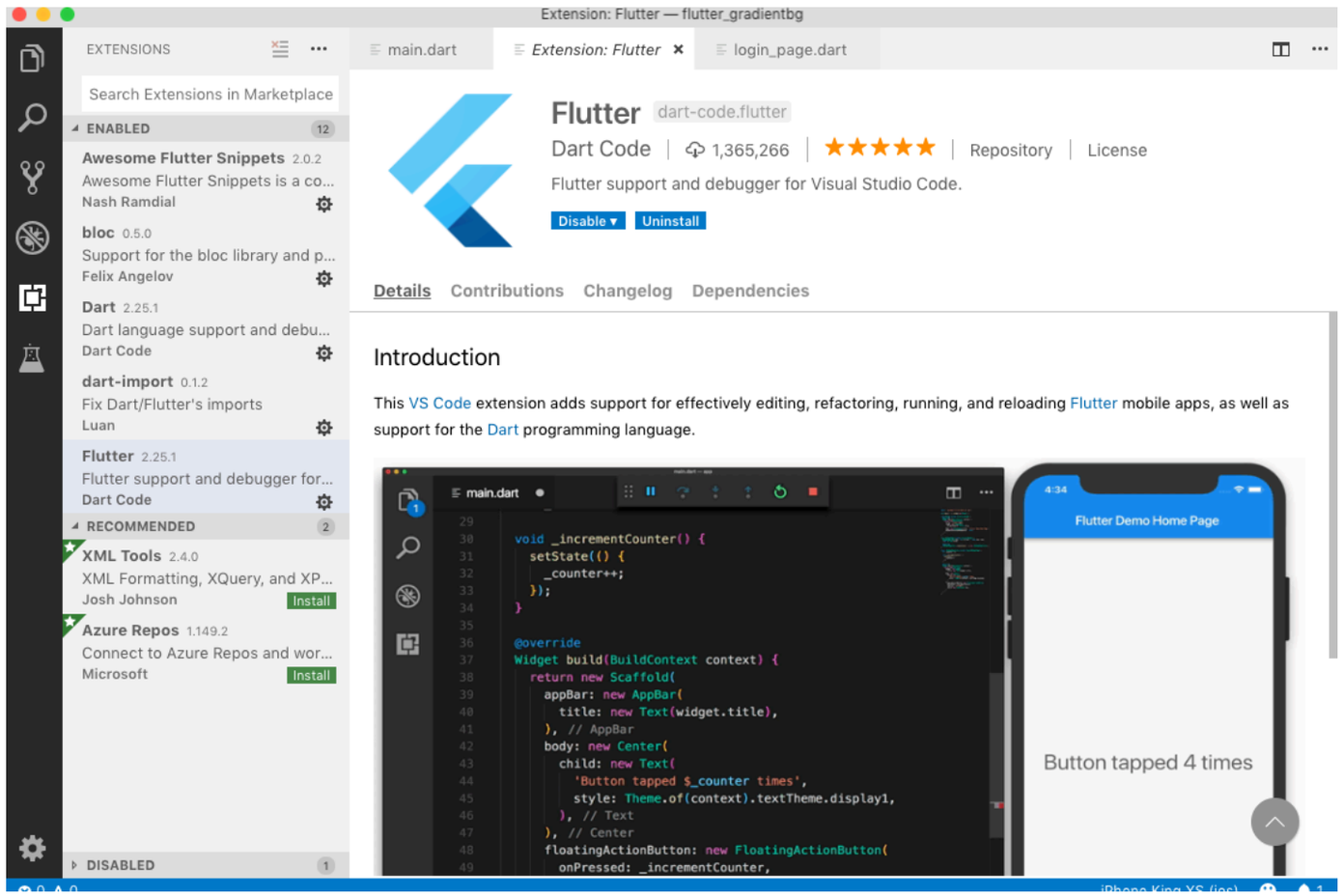
Common Error 2:

My Editor looking for Flutter SDK When creating a new Project

Normally your Flutter SDK Window’s Path is not correctly setup. Please set it go to Start and search “env” and set your path. Simply open **Environment Variables**, add new PATH where your flutter package is. For example: c:\src\flutter\bin
Restart visual studio code, open CMD and type flutter
That will check for all files and download some if missing.
Reboot your PC and try creating new project using Visual Studio Code again.

Installation

Visual Studio Code Plugins

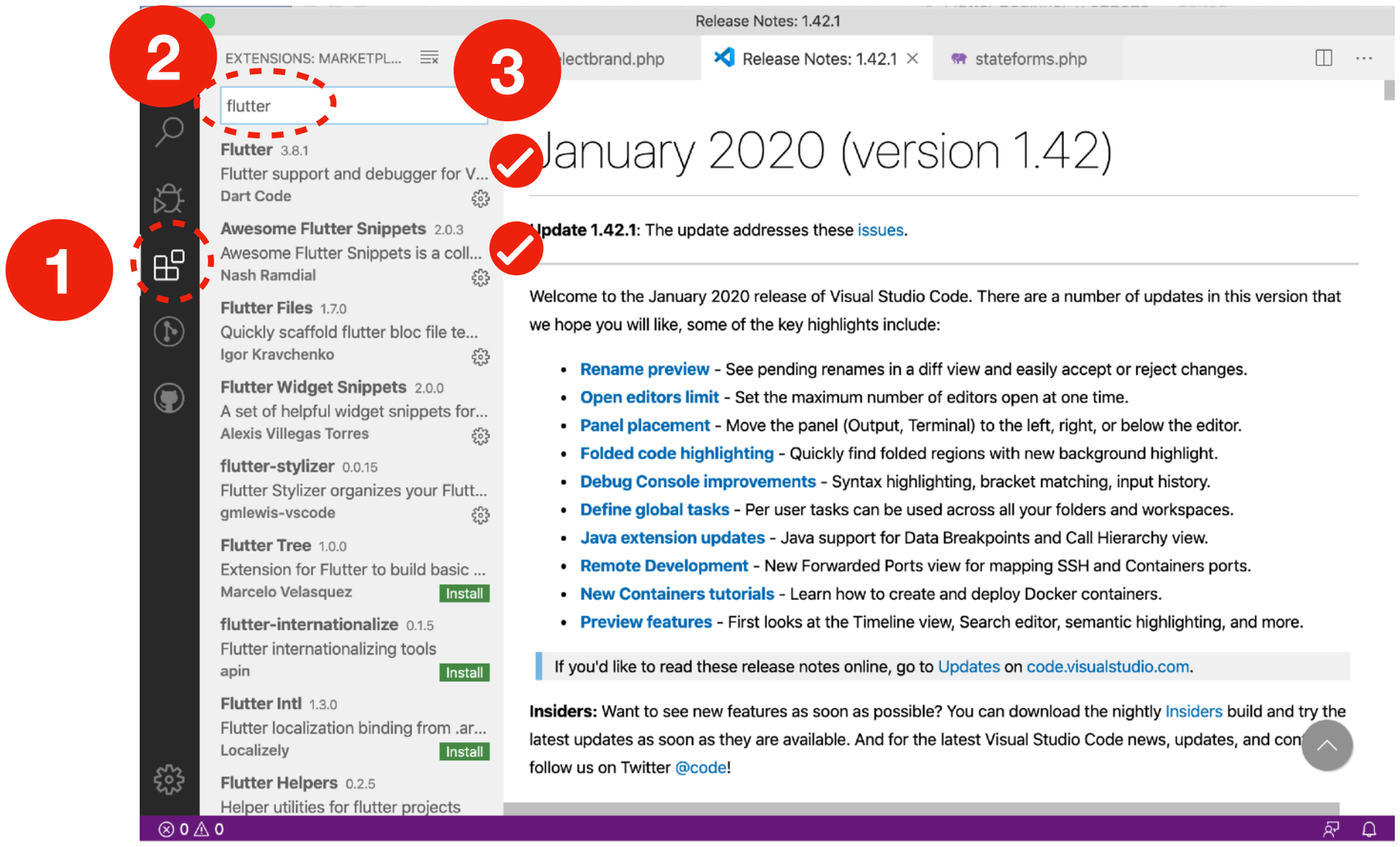


Download at: https://code.visualstudio.com/download

Install Plugins > Plugin, Search: Flutter

Install Extensions

Visual Studio Code



Favorite Extensions

Awesome Flutter Snippets nash.awesome-flutter-snippets

Neevash Ramdial | 313,405 | ★★★★★ | Repository | License | v2.0.4

Awesome Flutter Snippets is a collection snippets and shortcuts for commonly used Flutter functions and classes

Disable Uninstall *This extension is enabled globally.*

Bracket Pair Colorizer coenraads.bracket-pair-colorizer

CoenraadS | 3,464,090 | ★★★★½ | Repository | License | v1.0.61

A customizable extension for colorizing matching brackets

Disable Uninstall *This extension is enabled globally.*

Pubspec Assist jeroen-meijer.pubspec-assist

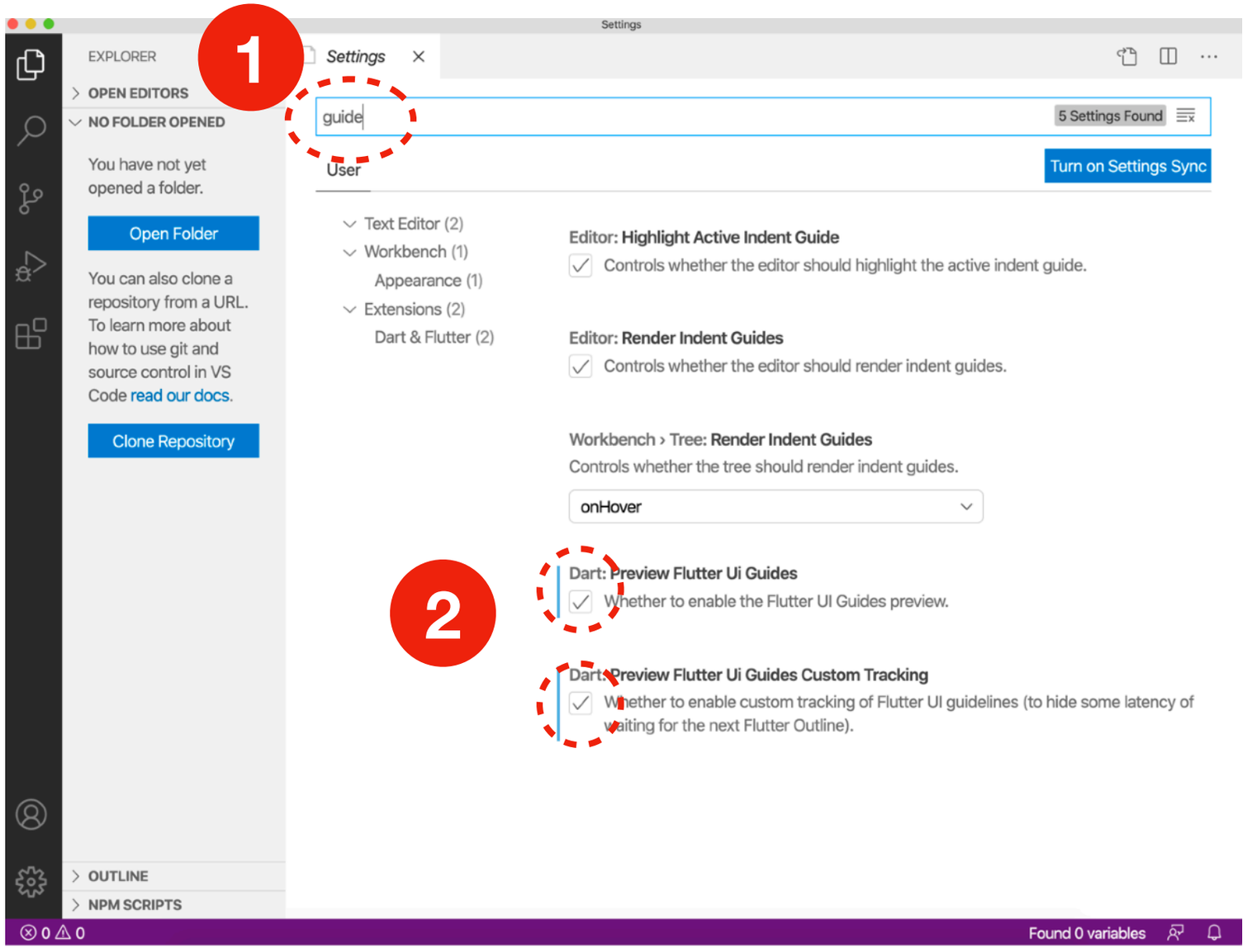
Jeroen Meijer | 83,279 | ★★★★★ | Repository | v2.0.0

Easily add and update dependencies to your Dart and Flutter project.

Disable Uninstall *This extension is enabled globally.*

Guide

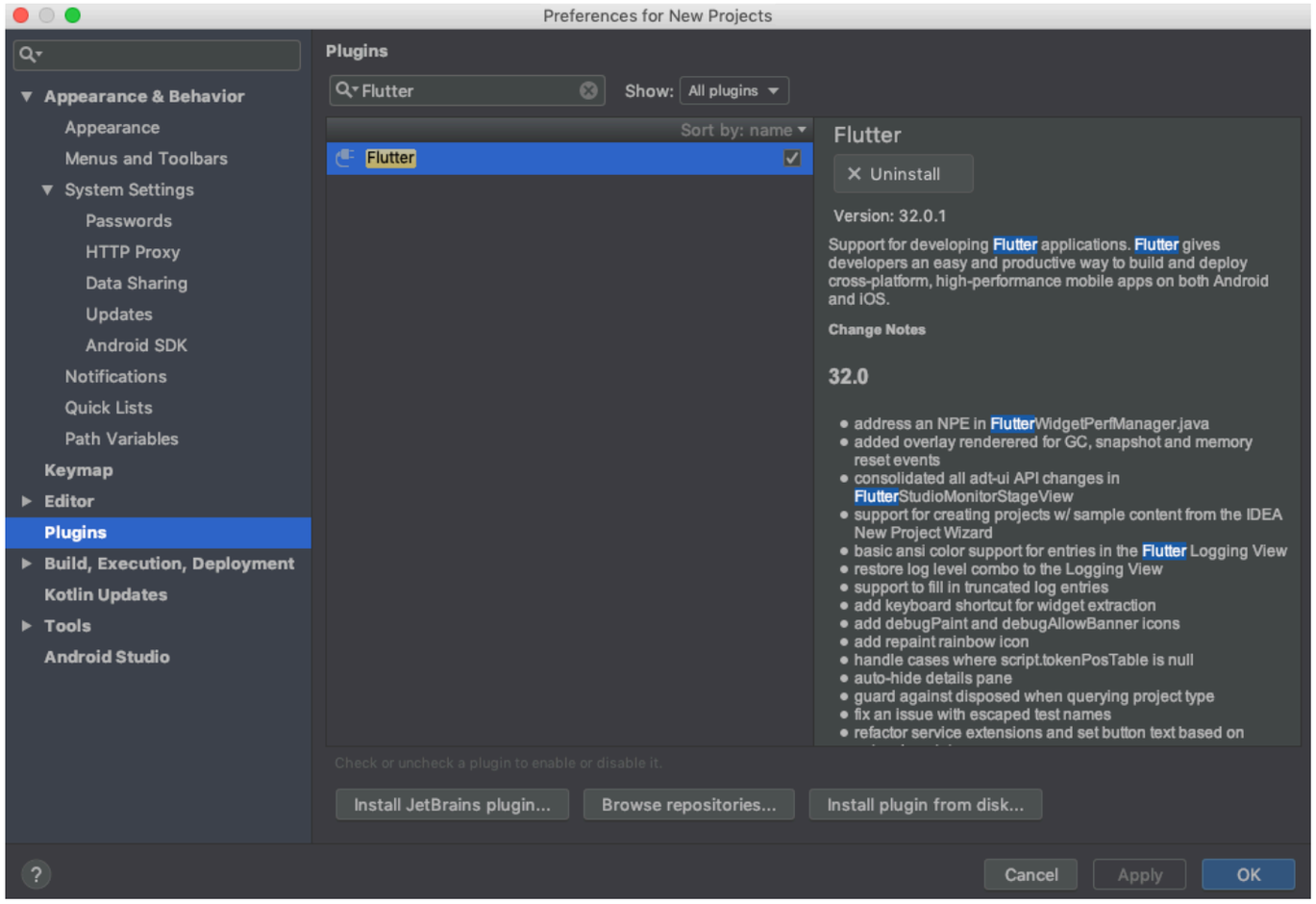
Activate Guides so that you can view better while coding



Installation

Plugin

Install Android Studio > SDK > Plugin, Search: Flutter

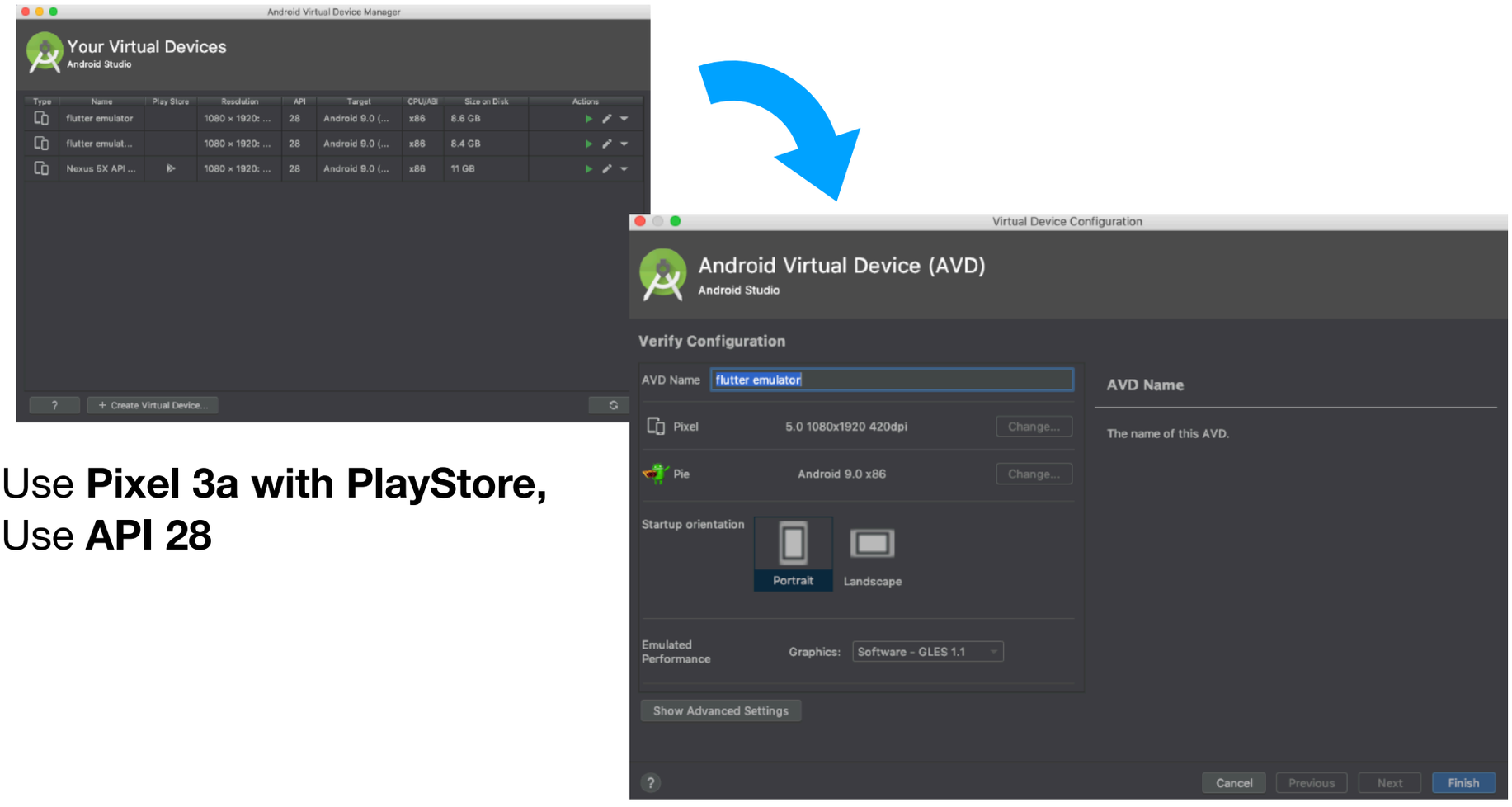


Installation

Create Emulator

You need to create Emulators or Simulator (Mac only) to preview your Mobile apps

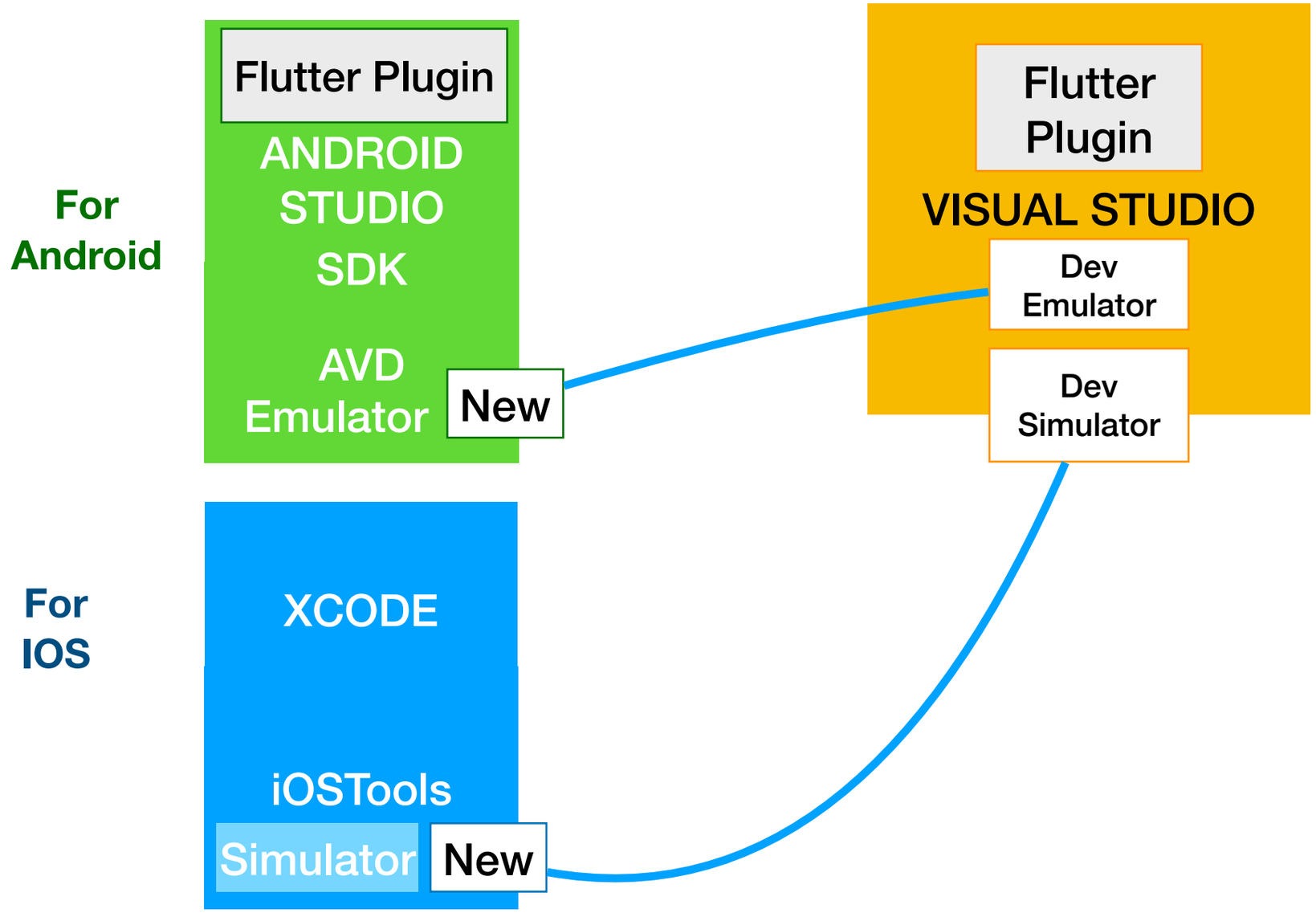
Android Studio > AVD > Create Emulator



Use **Pixel 3a with PlayStore,**
Use **API 28**

Flutter, Android (SDK & AVD), Xcode (Simulator)

Why you need to install



Editors/IDE

Editors for Flutter

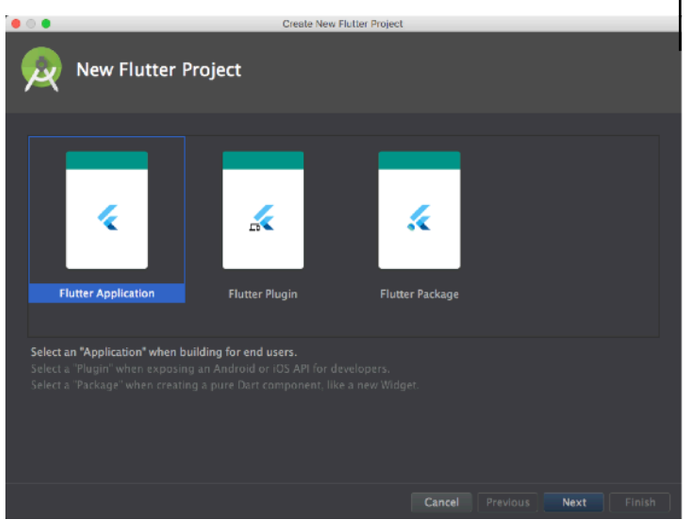
You can use anyone of these

IntelliJ IDEA

Some Java Developers using this

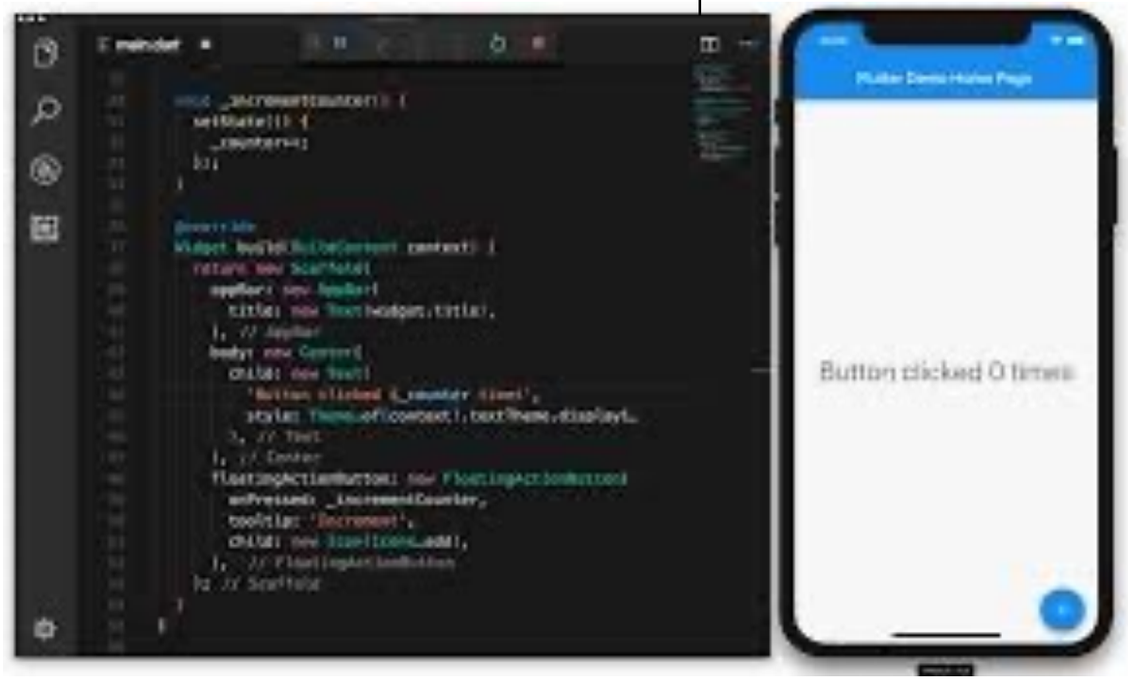
Android Studio

You can use this too



Visual Studio Code

Our Default Editor



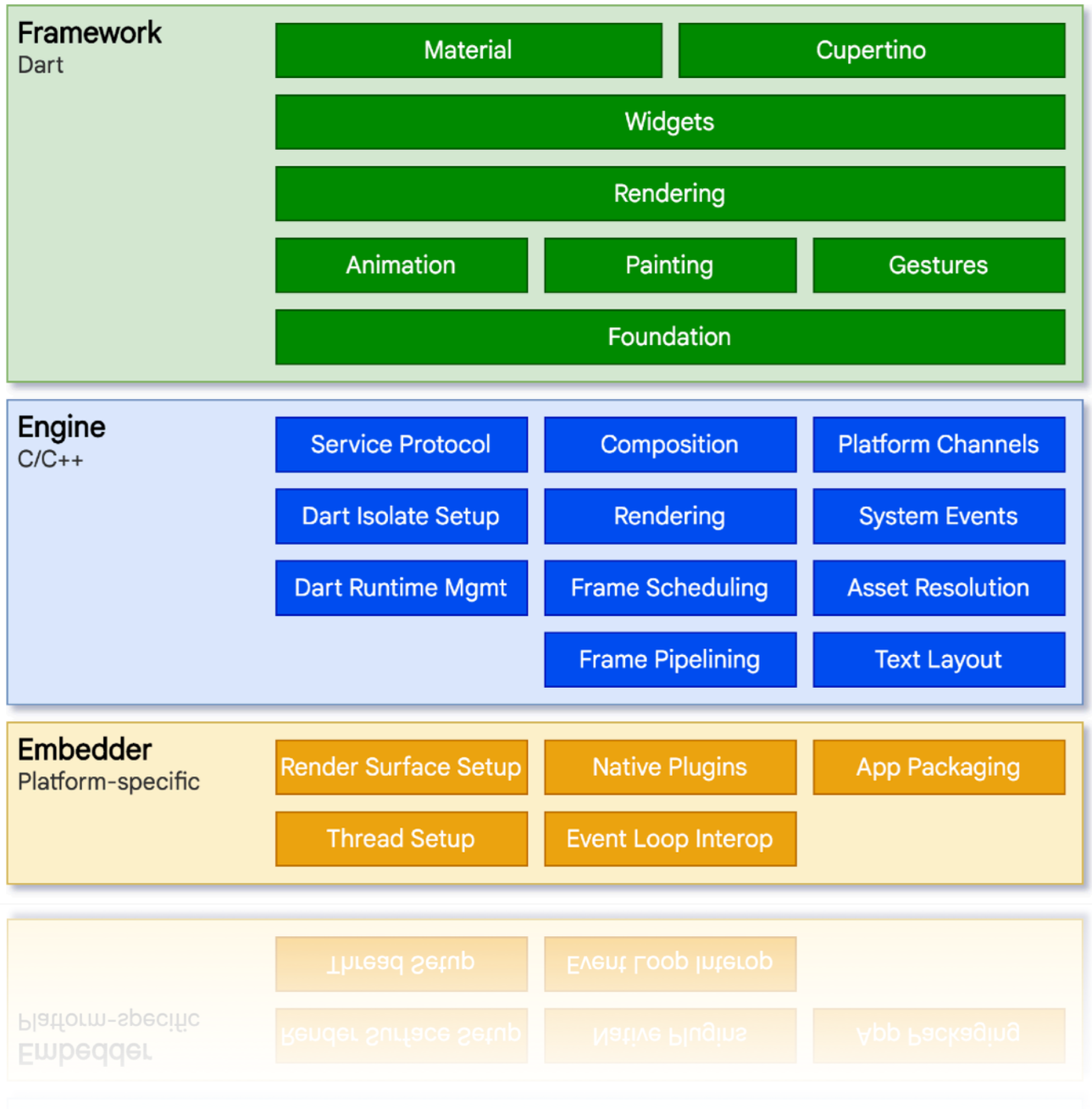
Architecture

Flutter Architecture

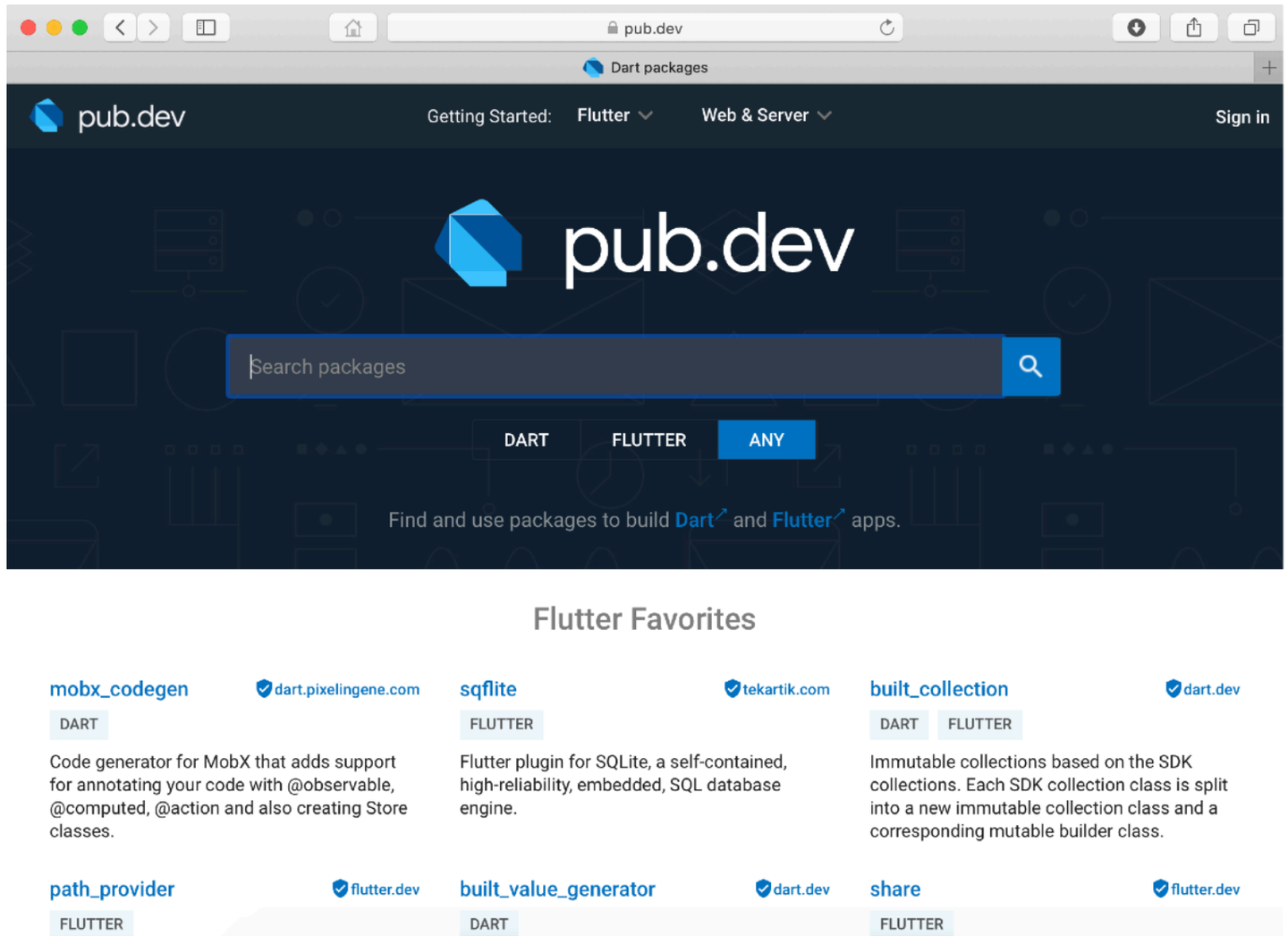
During development, Flutter apps run in a VM that offers stateful hot reload of changes without needing a full recompile.

For release, Flutter apps are compiled directly to machine code, whether Intel x64 or ARM instructions, or to JavaScript if targeting the web.

The framework is open source, with a permissive BSD license, and has a thriving ecosystem of third-party packages that supplement the core library functionality.



Pub.Dev


pub.dev
Dart packages
pub.dev
Getting Started:
Flutter
Web & Server
Sign in
pub.dev
Search packages
DART
FLUTTER
ANY
Find and use packages to build Dart and Flutter apps.
Flutter Favorites
mobx_codegen
dart.pixelingene.com
DART
Code generator for MobX that adds support for annotating your code with @observable, @computed, @action and also creating Store classes.
sqflite
tekartik.com
FLUTTER
Flutter plugin for SQLite, a self-contained, high-reliability, embedded, SQL database engine.
built_collection
dart.dev
DART
FLUTTER
Immutable collections based on the SDK collections. Each SDK collection class is split into a new immutable collection class and a corresponding mutable builder class.
path_provider
flutter.dev
FLUTTER
built_value_generator
dart.dev
DART
share
flutter.dev
FLUTTER

Widgets

Widgets

Flutter emphasizes widgets as a unit of composition.

Widgets are the building blocks of a Flutter app's user interface.

Widgets form a hierarchy based on composition.

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('My Home Page')),
        body: Center(
          child: Builder(
            builder: (BuildContext context) {
              return Column(
                children: [
                  Text('Hello World'),
                  SizedBox(height: 20),
                  RaisedButton(
                    onPressed: () {
                      print('Click!');
                    },
                    child: Text('A button'),
                  ),
                ],
              );
            },
          ),
        ),
      ),
    );
  }
}
```

Widgets

Stateless widgets are immutable, meaning that their properties can't change—all values are final.

Stateful widgets maintain state that might change during the lifetime of the widget. Implementing a stateful widget requires at least two classes:

i) a **StatefulWidget** class that creates an instance of

ii) a **State** class. The StatefulWidget class is, itself, immutable, but the State class persists over the lifetime of the widget.

Inherited Widget a special kind of widget that defines a *context* at the root of a sub-tree.

Stateless Widgets

How A Stateless Widget looks like

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

class ThirdScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Screen 3"),
      ), // AppBar
      body: Center(
        child: RaisedButton(
          child: Text('Go back Screen 1'),
          onPressed: () {
            //Guna ne utk balik ke screen 1
            Navigator.of(context).pushNamedAndRemoveUntil('/screen1',
            (Route<dynamic> route) => false);
            //----
          },
        ), // RaisedButton
      ), // Center
    ); // Scaffold
  }
}

```

A **stateless widget** has no internal state to manage. Icon, IconButton, and Text are examples of stateless widgets, which subclass StatelessWidget.

Stateful Widgets

How a Stateful Widget has 2 classes

State class
Create State

StatefulWidget class
Immutable. More to Creating Widgets

A **stateful widget** is dynamic. The user can interact with a stateful widget (by typing into a form, or moving a slider, for example), or it changes over time (perhaps a data feed causes the UI to update).

Checkbox, Radio, Slider, InkWell, Form, and TextField are examples of stateful widgets, which subclass StatefulWidget.

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

class FirstScreen extends StatefulWidget {
  @override
  _FirstScreenState createState() => new _FirstScreenState();

}

class _FirstScreenState extends State<FirstScreen> {
  var _textController = new TextEditingController();

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Screen 1'),
      ), // AppBar
      body: new ListView(
        children: <Widget>[
          new ListTile(title: new TextField(controller: _textController,),),
          new ListTile(title: new RaisedButton(
            child: new Text("Next"),
            onPressed: () {
              //do something when pressed

              //original kaedah just navigate
              //Navigator.of(context).push(route);

              // Kaedah 1
              //var route = new MaterialPageRoute(builder: (BuildContext context)

              //Kaedah 2
              Navigator.push(context, MaterialPageRoute(builder:(context) => new S
);
            }
          ),) // RaisedButton // ListTile
```

Lifecycles

Android Lifecycle

onCreate -

onStart -

onResume -

onPause -

onRestart -

onStop -

onDestroy -

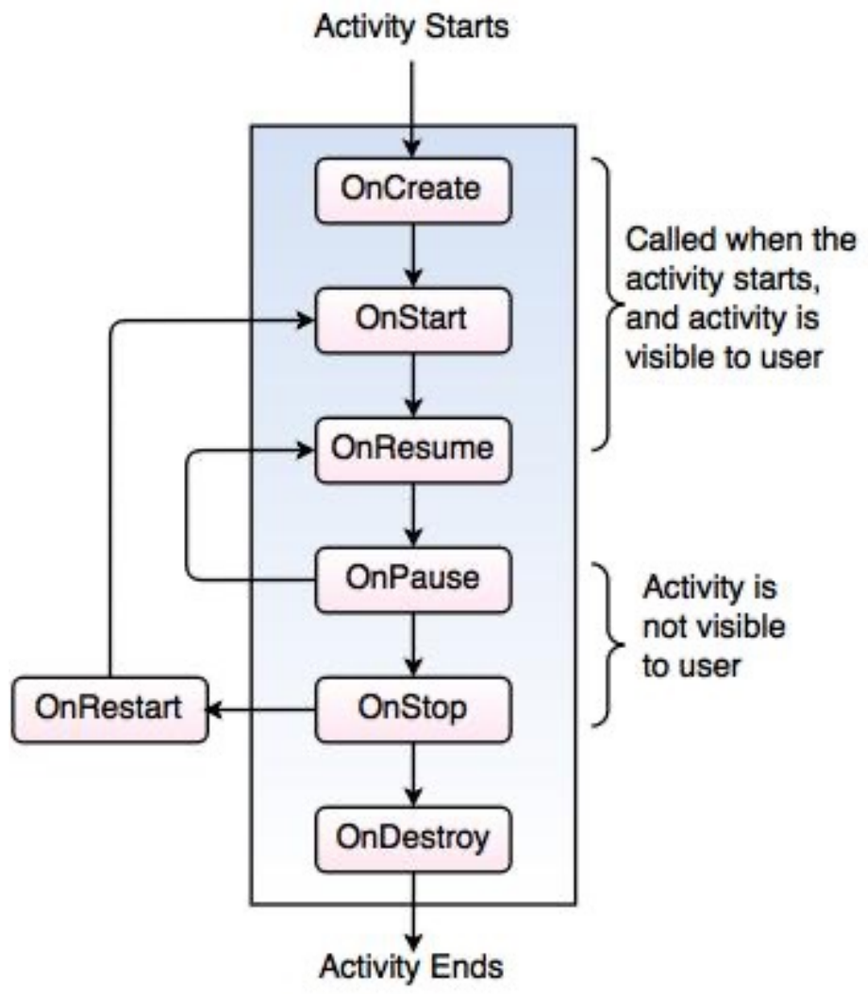


Fig. Android Activity Lifecycle Methods

iOS Lifecycle

didFinishLaunchingWithOptions

viewDidLoad -

viewWillAppear -

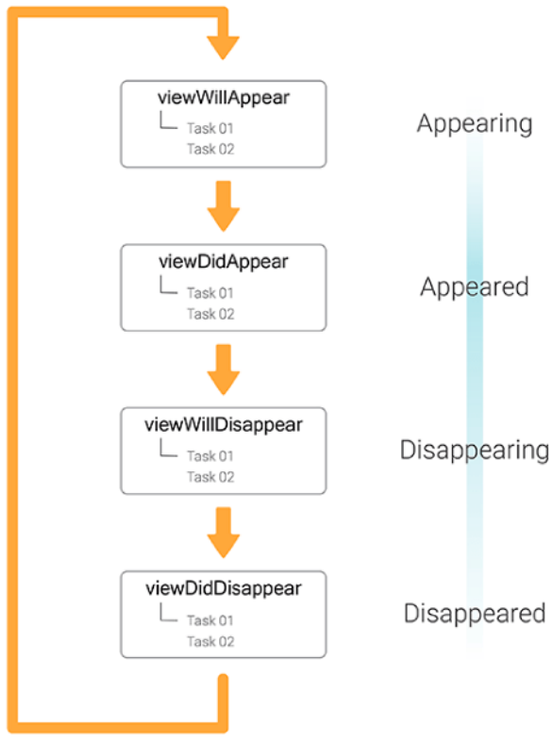
viewDidAppear -

viewWillDisappear -

viewDidDisappear -

viewDidUnload -

didDiscardSceneSessions



Flutter Lifecycle

createState -

initState -

didChangeDependencies -

build -

didUpdateWidget -

deactivate -

dispose -

Flutter Lifecycle

createState()
When the Framework is instructed to build a StatefulWidget, it immediately calls createState()

mounted is true
When createState creates your state class, a buildContext is assigned to that state. BuildContext is, overly simplified, the place in the widget tree in which this widget is placed. Here's a longer explanation. All widgets have a bool this.mounted property. It is turned true when the buildContext is assigned. It is an error to call setState when a widget is unmounted.

initState()
This is the first method called when the widget is created (after the class constructor, of course.) initState is called once and only once. It must called super.initState().

Flutter Lifecycle

didChangeDependencies()
This method is called immediately after initState on the first time the widget is built.

build()
This method is called often. It is required, and it must return a Widget.

didUpdateWidget(Widget oldWidget)
If the parent widget changes and has to rebuild this widget (because it needs to give it different data), but it's being rebuilt with the same runtimeType, then this method is called. This is because Flutter is re-using the state, which is long lived. In this case, you may want to initialize some data again, as you would in initState.

setState()
This method is called often from the framework itself and from the developer. Its used to notify the framework that data has changed

Flutter Lifecycle

deactivate()

Deactivate is called when State is removed from the tree, but it might be reinserted before the current frame change is finished. This method exists basically because State objects can be moved from one point in a tree to another.

dispose()

Dispose is called when the State object is removed, which is permanent. This method is where you should unsubscribe and cancel all animations, streams, etc.

mounted is false

The state object can never remount, and an error is thrown is setState is called.

Note: For LifeCycle, you need to use **WidgetsBindingObserver** it works when the app goes on foreground and background.

Example Flutter Lifecycle

```
import 'package:flutter/widgets.dart';
  class YourWidgetState extends State<YourWidget> with
WidgetsBindingObserver {

      @override
     void initState() {
       WidgetsBinding.instance.addObserver(this);
       super.initState();
     }


   @override
   void dispose() {
     WidgetsBinding.instance.removeObserver(this);
     super.dispose();
   }


    @override
   void didChangeAppLifecycleState(AppLifecycleState state) {
     if (state == AppLifecycleState.resumed) {
        //do your stuff
     }
   }
}
```

Types of State Management

There are several ways for State Management in Flutter

1.setState

setState is the State Management approach used in the default Flutter counter app. Easy to use. Great way to handle for local State. The default Project make use of setState to update value of the counter.

2. Provider

Provider is a state management package built by the community, not by Google. It Quite easy to use.

3. BLoC

BLoC implements the Observer pattern, with it your events are fed into a Stream that is the input into a logic block. Can get out of hand if large Projects.

4. Redux

Redux uses a lot of Boilerplate and can be quite difficult to understand. The State is contained in something called a Store.

Command Flutter Commands

```
Available commands:
  analyze                Analyze the project's Dart code.
  attach                 Attach to a running application.
  bash-completion        Output command line shell completion setup scripts.
  build                  Flutter build commands.
  channel                List or switch flutter channels.
  clean                  Delete the build/ and .dart_tool/ directories.
  config                 Configure Flutter settings.
  create                 Create a new Flutter project.
  devices                List all connected devices.
  doctor                 Show information about the installed tooling.
  drive                  Runs Flutter Driver tests for the current project.
  emulators              List, launch and create emulators.
  format                 Format one or more dart files.
  help                   Display help information for flutter.
  install                Install a Flutter app on an attached device.
  logs                   Show log output for running Flutter apps.
  make-host-app-editable   Moves host apps from generated directories to
                         non-generated directories so that they can be edited
                         by developers.
  precache               Populates the Flutter tool's cache of binary
                         artifacts.
  pub                    Commands for managing Flutter packages.
  run                    Run your Flutter app on an attached device.
  screenshot             Take a screenshot from a connected device.
  test                   Run Flutter unit tests for the current project.
  upgrade                Upgrade your copy of Flutter.
  version                List or switch flutter versions.
```

Language

What is Dart Language?

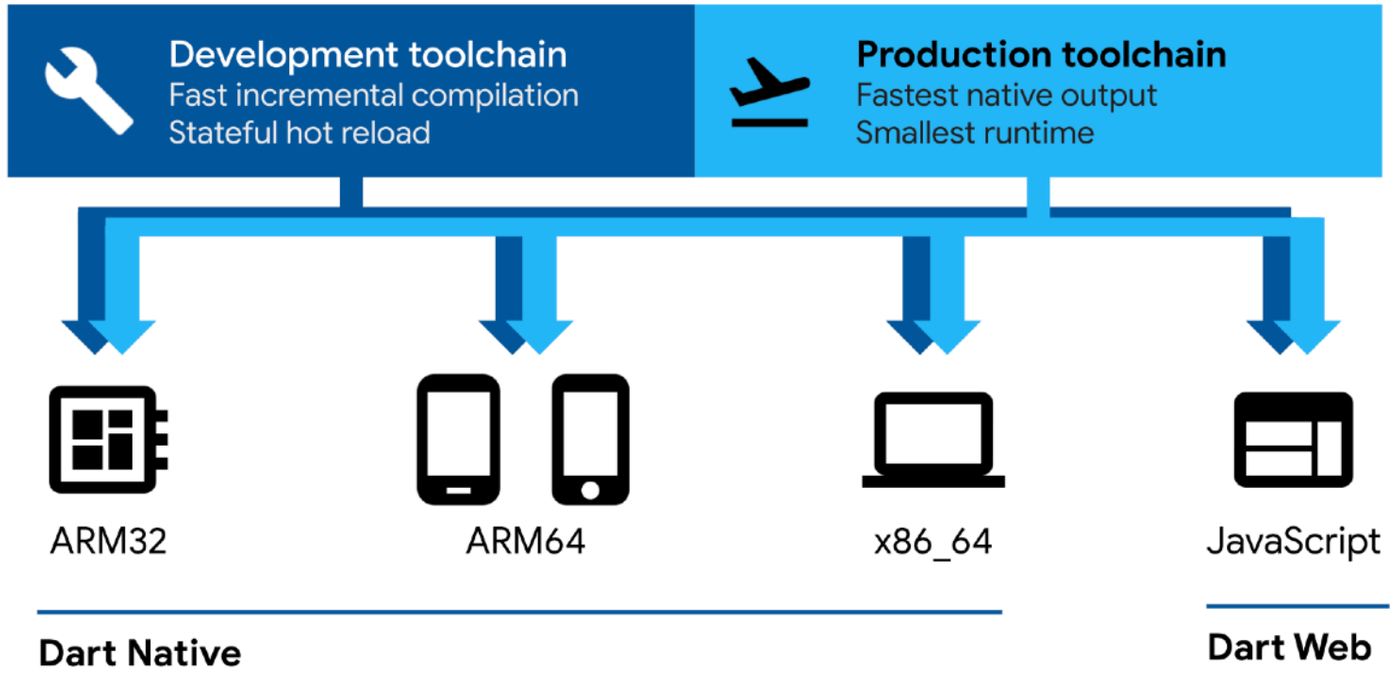
What is Dart for?

Dart Native:

For programs targeting devices (mobile, desktop, server, and more), Dart Native includes both a Dart VM with JIT (just-in-time) compilation and an AOT (ahead-of-time) compiler for producing machine code.

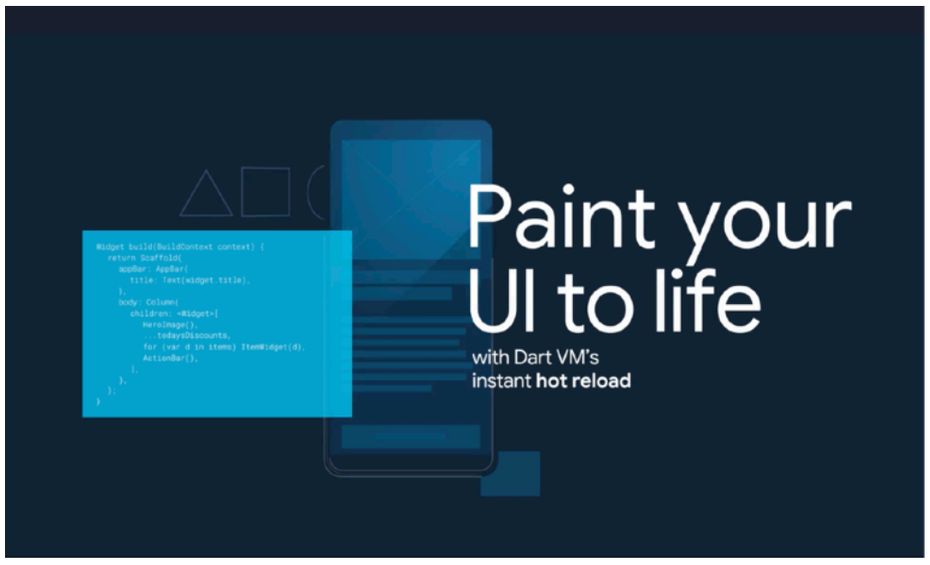
Dart Web:

For programs targeting the web, Dart Web includes both a development time compiler (dartdevc) and a production time compiler (dart2js)



Dart Language

Reference: https://www.dartlang.org



Dart documentation

Welcome to the Dart documentation! Here are some of the most visited pages:

Language samples
A brief, example-based introduction to the Dart language.

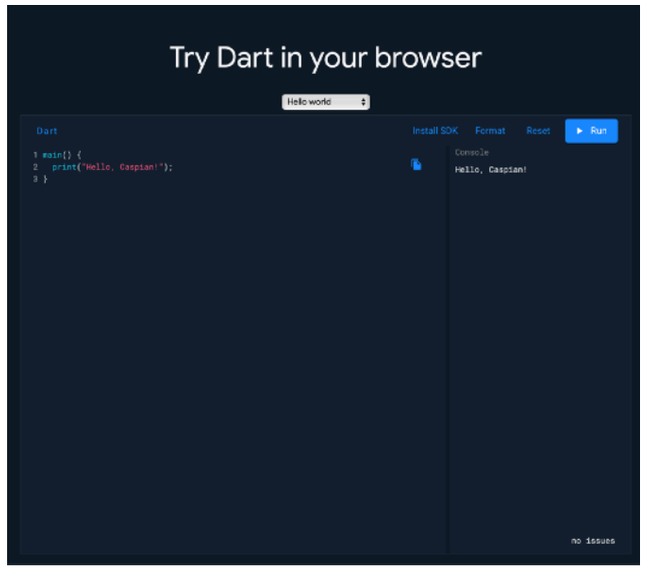
Language tour
A more thorough (yet still example-based) introduction to the Dart language.

Effective Dart
Best practices for building consistent, maintainable, efficient Dart code.

Library tour
An example-based introduction to the major features in the Dart SDK's core libraries.

Dart SDK
What's in the SDK, and how to install it.

Futures, async, await
How to write asynchronous Dart code that uses futures and the async and await keywords.



mportant concepts

- Everything you can place in a variable is an object, and every object is an instance of a class.
- Type annotations are optional
- Dart supports generic types, like List<int> (a list of integers) or List<dynamic> (a list of objects of any type).
- You can also create nested functions.
- Instance variables are sometimes known as fields or properties.
- Dart doesn’t have the keywords public, protected, and private. If an identifier starts with an underscore (_), it’s private to its library.
- Dart has both expressions (which have runtime values) and statements (which don’t). For example, the conditional expression condition ? expr1 : expr2 has a value of expr1 or expr2.
- Dart tools can report two kinds of problems: warnings and errors.

Dart Libraries

dart:core
Built-in types, collections, and other core functionality. This library is automatically imported into every Dart program.

dart:async
Support for asynchronous programming, with classes such as Future and Stream.

dart:math
Mathematical constants and functions, plus a random number generator.

dart:convert
Encoders and decoders for converting between different data representations, including JSON and UTF-8.

Dart Sample Codes

Hello World

Every app must have a main() function. To display text on the console, you can use print() function:

```
void main() {
  print('Hello, World!');
}
```

Variables

Even in type-safe Dart code, most variables don’t need explicit types, thanks to type inference:

```
var name = 'Voyager I';
var year = 1977;
var antennaDiameter = 3.7;
var flybyObjects = ['Jupiter', 'Saturn', 'Uranus', 'Neptune'];
var image = {
  'tags': ['saturn'],
  'url': '//path/to/saturn.jpg'
};
```

Dart Sample Codes

Build-In Types

The Dart language has special support for the following types:

- numbers
- strings
- booleans
- lists (also known as arrays)
- sets
- maps
- runes (for expressing Unicode characters in a string)
- symbols

Numbers

The Dart language has 2 flavors:

- int
- double

Dart Sample Codes

Conversion

Here’s how you turn a string into a number, or vice versa

```dart
// String -> int
var one = int.parse('1');
assert(one == 1);

// String -> double
var onePointOne = double.parse('1.1');
assert(onePointOne == 1.1);

// int -> String
String oneAsString = 1.toString();
assert(oneAsString == '1');

// double -> String
String piAsString = 3.14159.toStringAsFixed(2);
assert(piAsString == '3.14');
```

Dart Sample Codes

Control flow statements

Dart supports the usual control flow statements:

```
if (year >= 2001) {
  print('21st century');
} else if (year >= 1901) {
  print('20th century');
}

for (var object in flybyObjects) {
  print(object);
}

for (int month = 1; month <= 12; month++) {
  print(month);
}

while (year < 2016) {
  year += 1;
}
```

Dart Sample Codes

Functions

Recommend specifying the types of each function's arguments and return value:

```dart
int fibonacci(int n) {
  if (n == 0 || n == 1) return n;
  return fibonacci(n - 1) + fibonacci(n - 2);
}

var result = fibonacci(20);
```

A shorthand => (arrow) syntax is handy for functions that contain a single statement. This syntax is especially useful when passing anonymous functions as arguments:

Normally

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

shorthand

```dart
void main() => runApp(new MyApp());
```

Dart Sample Codes

Comments

Dart comments usually start with //.

```
// This is a normal, one-line comment.

/// This is a documentation comment, used to document libraries,
/// classes, and their members. Tools like IDEs and dartdoc treat
/// doc comments specially.

/* Comments like these are also supported. */
```

Imports

To access APIs defined in other libraries, use import.

```
// Importing core libraries
import 'dart:math';

// Importing libraries from external packages
import 'package:test/test.dart';

// Importing files
import 'path/to/my_other_file.dart';
```

Dart Sample Codes

Classes

An example shows how you can use getters and setters in a Dart class. A getter has no parameters and returns a value, and the setter has one parameter and does not return a value.

```dart
class Student {
   String name;
   int age;

   String get stud_name {
      return name;
   }

   void set stud_name(String name) {
      this.name = name;
   }

   void set stud_age(int age) {
      if(age<= 0) {
        print("Age should be greater than 5");
      }  else {
         this.age = age;
      }
   }

   int get stud_age {
      return age;
   }
}
void main() {
   Student s1 = new Student();
   s1.stud_name = 'MARK';
   s1.stud_age = 0;
   print(s1.stud_name);
   print(s1.stud_age);
}
```

Variables

getter

Setter

getter

Note: Can also use Constructor, for assignment to members.

```
Student(this.name, this.age) {
// Initialization code goes here.
}
```

Student s1 = new Student('MARK', 20);

Dart Sample Codes

Inheritance

Dart has single inheritance. Dart doesn't support multiple inheritance. It uses Mixins.

Uses the word **extends** to inherit

```
class Orbiter extends Spacecraft {
  num altitude;
  Orbiter(String name, DateTime launchDate, this.altitude)
      : super(name, launchDate);
}
```

Dart Sample Codes

Mixins

Mixins are a way of reusing code in multiple class hierarchies. The following class can act as a mixin

```dart
class Piloted {
  int astronauts = 1;
  void describeCrew() {
    print('Number of astronauts: $astronauts');
  }
}
```

Uses the word **with** to extend the class with the mixins

```dart
class PilotedCraft extends Spacecraft with Piloted {
  // ...
}
```

PilotedCraft now has the **astronauts** field as well as the **describeCrew()** method.

To implement a **mixin**, create a class that extends Object and declares no constructors. Unless you want your **mixin** to be usable as a regular class, use the **mixin** keyword instead of class.

Dart Sample Codes

Async

async and **await** help make asynchronous code easy to read. Use **Future<>** in front of a function that uses async

```
Future<void> createDescriptions(Iterable<String> objects) async {
  for (var object in objects) {
    try {
      var file = File('$object.txt');
      if (await file.exists()) {
        var modified = await file.lastModified();
        print(
            'File for $object already exists. It was modified on $modified.');
        continue;
      }
      await file.create();
      await file.writeAsString('Start describing $object in this file.');
    } on IOException catch (e) {
      print('Cannot create description for $object: $e');
    }
  }
}
```

Dart Sample Codes

Async*

Use **async*** to give a nice, readable way to build streams. Stream means data is continuous streaming. Use the word **Stream<>** in front of a function that does streaming.

```
Stream<String> report(Spacecraft craft, Iterable<String> objects) async* {
  for (var object in objects) {
    await Future.delayed(oneSecond);
    yield '${craft.name} flies by $object';
  }
}
```

Dart Sample Codes

static, final, and const

"static", "final", and "const" mean entirely distinct things in Dart:

"**static**" means a member is available on the class itself instead of on instances of the class. That's all it means, and it isn't used for anything else.

"**final**" means single-assignment: a final variable or field *must* have an initializer. Once assigned a value, a final variable's value cannot be changed. final modifies *variables*.

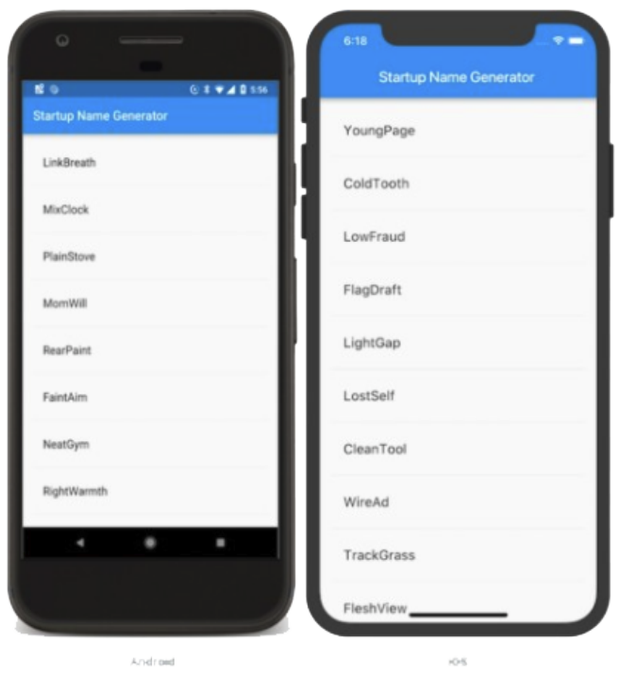
“**const**" means that the object's entire deep state can be determined entirely at compile time and that the object will be frozen and completely immutable.

Layouts

UI Design

- Flutter using **Material Design** in Android & Using '**Cupertino**' Design in iOS.
- Slight differences in rendering UIs.
- Not all designs can be same.

Material Cupertino



Material Cupertino

Flutter Layout

Basic Layout in Flutter makes up of widgets:

- Container
- Row

} **layout elements**

- Column
- Image
- Text
- Icon

} **UI elements**

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

class selamatpagiumt extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return new Container(
        padding: const EdgeInsets.all(5.0),
        child: new Column(
          children: <Widget>[
            new Text('Selamat'),
            new Text('Pagi'),
            new Text('UMT'),
          ], // <Widget>[]
        ) // Column
    ); // Container
  }
}
```

REMEMBER:

- Widgets are used for both layout and UI elements.
- Compose simple widgets to build complex widgets.

Flutter Layout

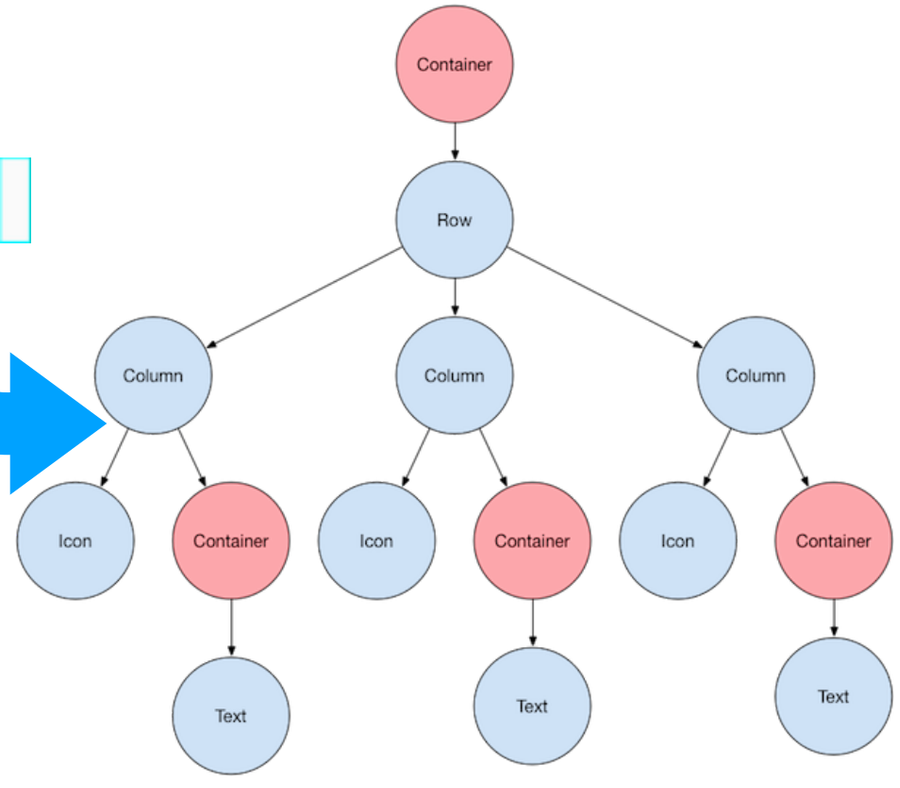
Final UI

Made up of Widgets:

Understand This in Container, Row, Column & Widgets

In Flutter, a common way of laying out widgets is composing them into groups of columns and row—this allows us to easily layout our components in a vertical / horizontal manner. When you wish to layout components in a **Horizontal** manner, then you will use a **Row** and for **Vertical** layout you will use a **Column**. There is also a Stack Widget for overlapping content

When you code it in Flutter



Common Layout Widgets

Standard widgets:

- **Container**: Adds padding, margins, borders, background color, or other decorations to a widget.
- **GridView**: Lays widgets out as a scrollable grid.
- **ListView**: Lays widgets out as a scrollable list.
- **Stack**: Overlaps a widget on top of another.

Common Layout Widgets

Material widgets:

- **Card:** Organizes related info into a box with rounded corners and a drop shadow.

- **ListTile:** Organizes up to 3 lines of text, and optional leading and trailing icons, into a row.

Common Layout Widgets

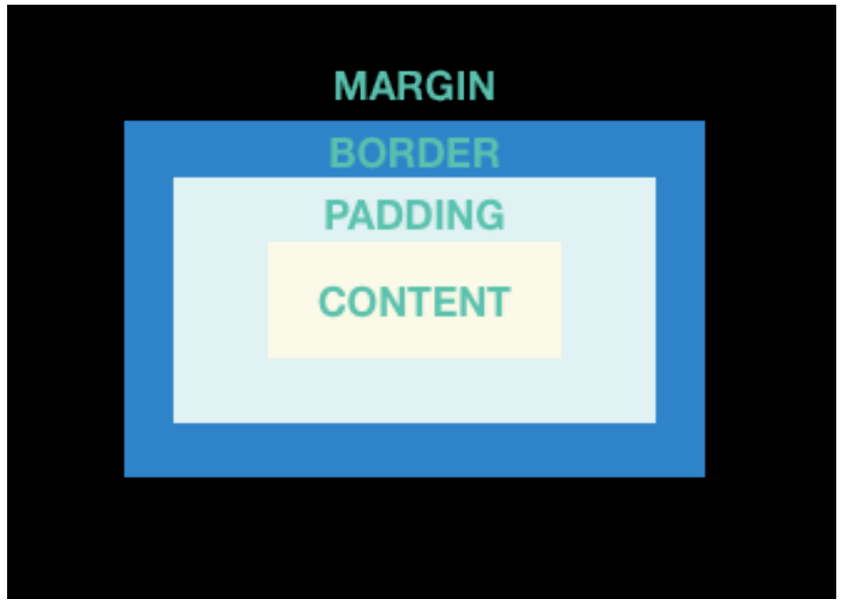
Container:

- You can embed other Widgets in a Container to make them compartmentalised. Container is Optional

- Add padding, margins, borders

- Change background color or image

- Contains a single child widget, but that child can be a Row, Column, or even the root of a widget tree



Common Layout Widgets

There are two basic approaches to creating Flutter apps with responsive design:

- **Use the LayoutBuilder class**
- **Use the MediaQuery.of() method in your build functions**

Other useful widgets and classes for creating a responsive UI:

- AspectRatio
- CustomSingleChildLayout
- CustomMultiChildLayout
- FittedBox
- FractionallySizedBox
- LayoutBuilder
- MediaQuery
- MediaQueryData
- OrientationBuilder

Navigation

Navigation

To create a View, You create a new Class

To create multiple Views, you create many Classes

To navigate, you use Routes

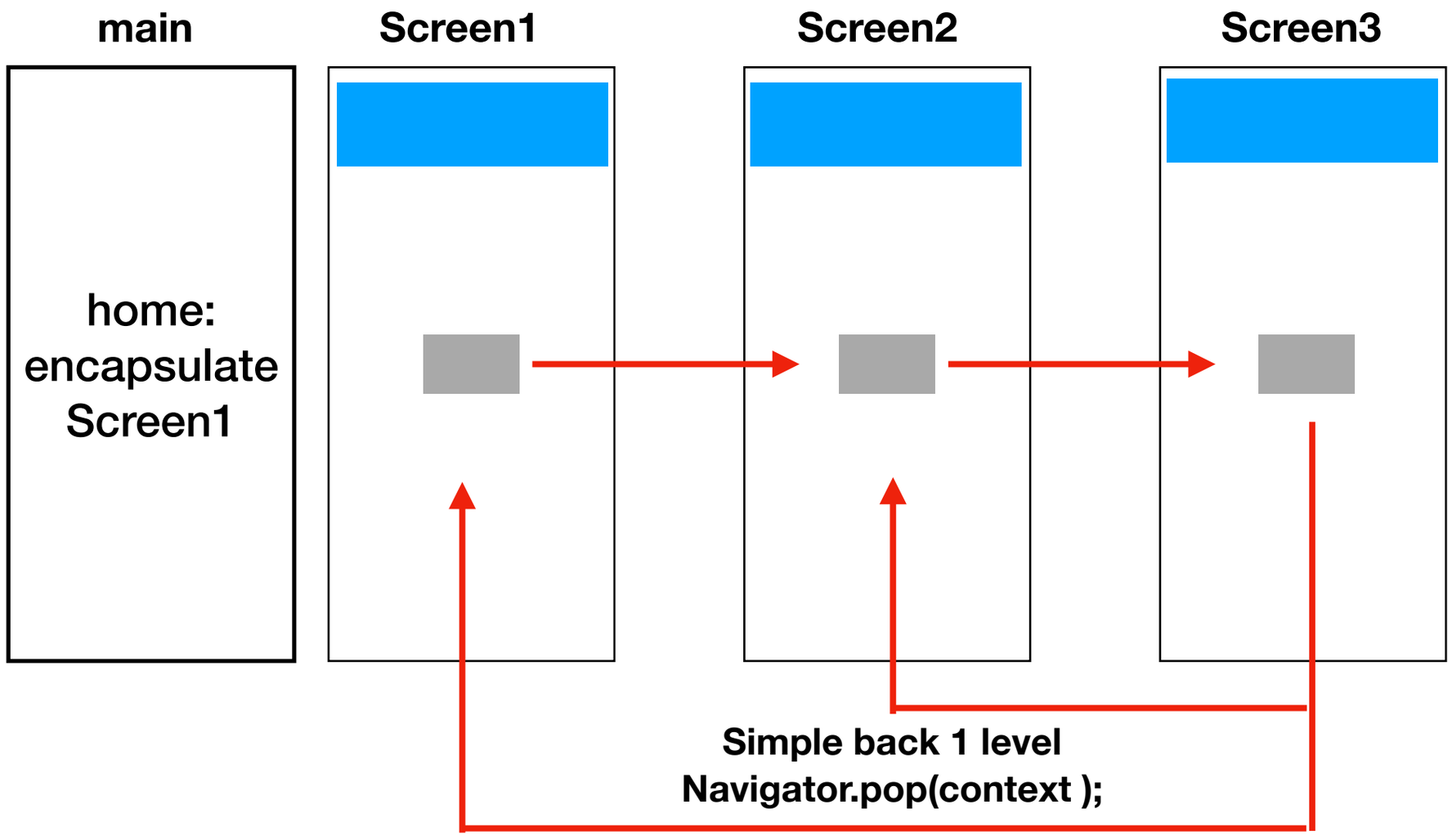
There are 2 versions of Navigation as of Flutter Version 1.22

- **Imperative (Version 1.0) - Easier**
- **Declarative (Version 2.0) - More options**

You can use Both

Navigation

Multipage Navigation using Route

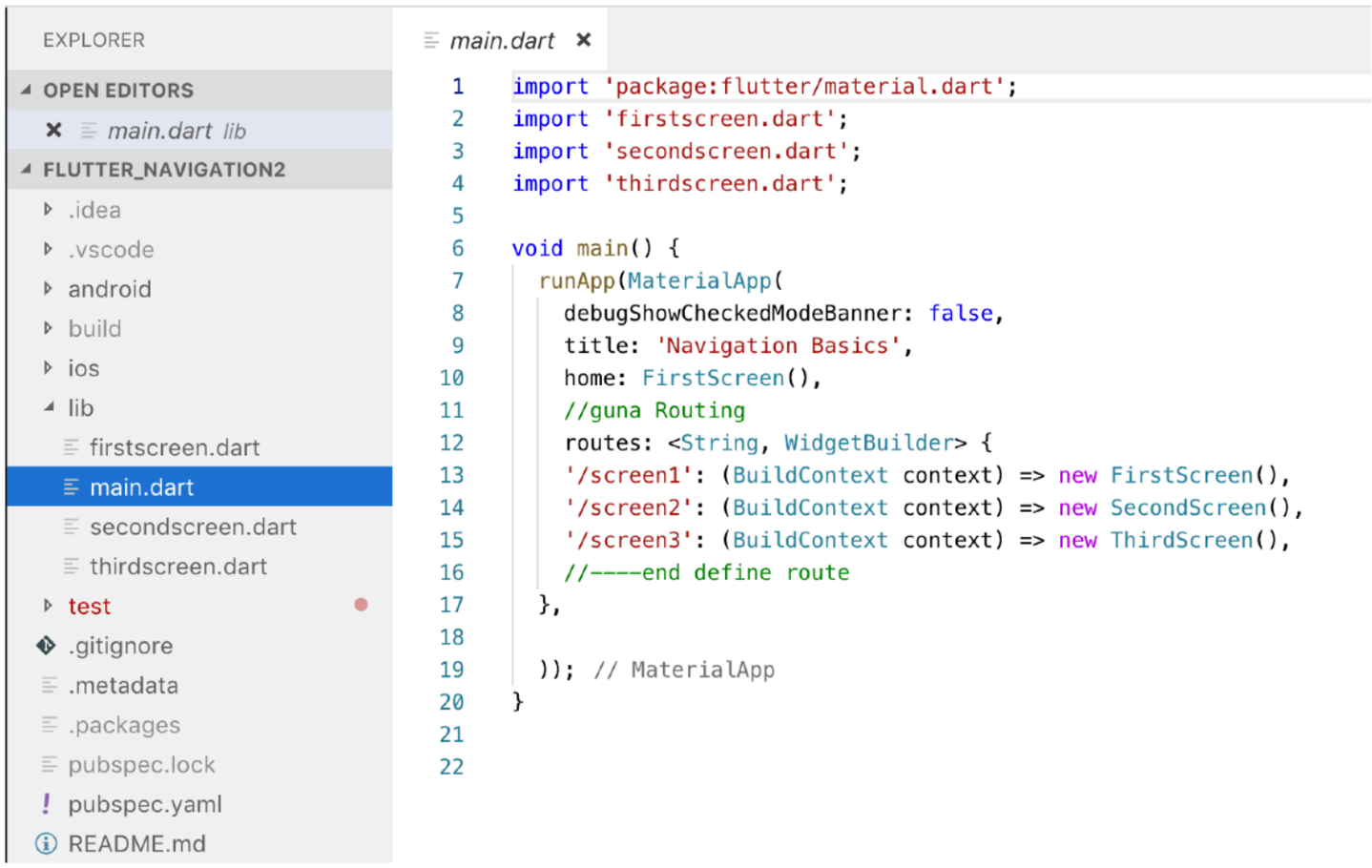


Use This for going back multiple levels
Navigator.of(context).pushNamedAndRemoveUntil('/screen1', (Route<dynamic> route) => false);

Navigation

Multipage Navigation using Route

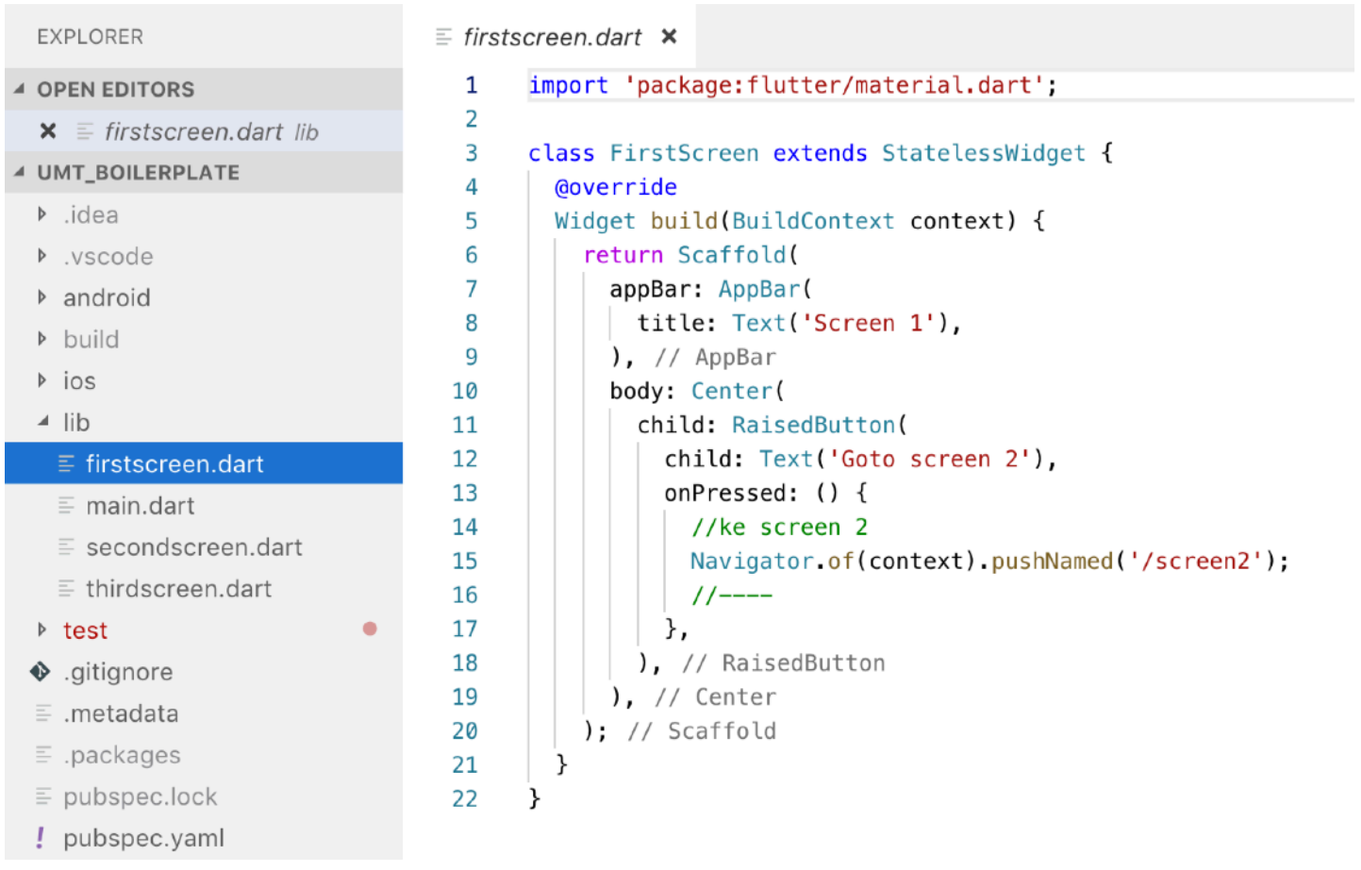
Step 1: main.dart



Navigation

Multipage Navigation using Route

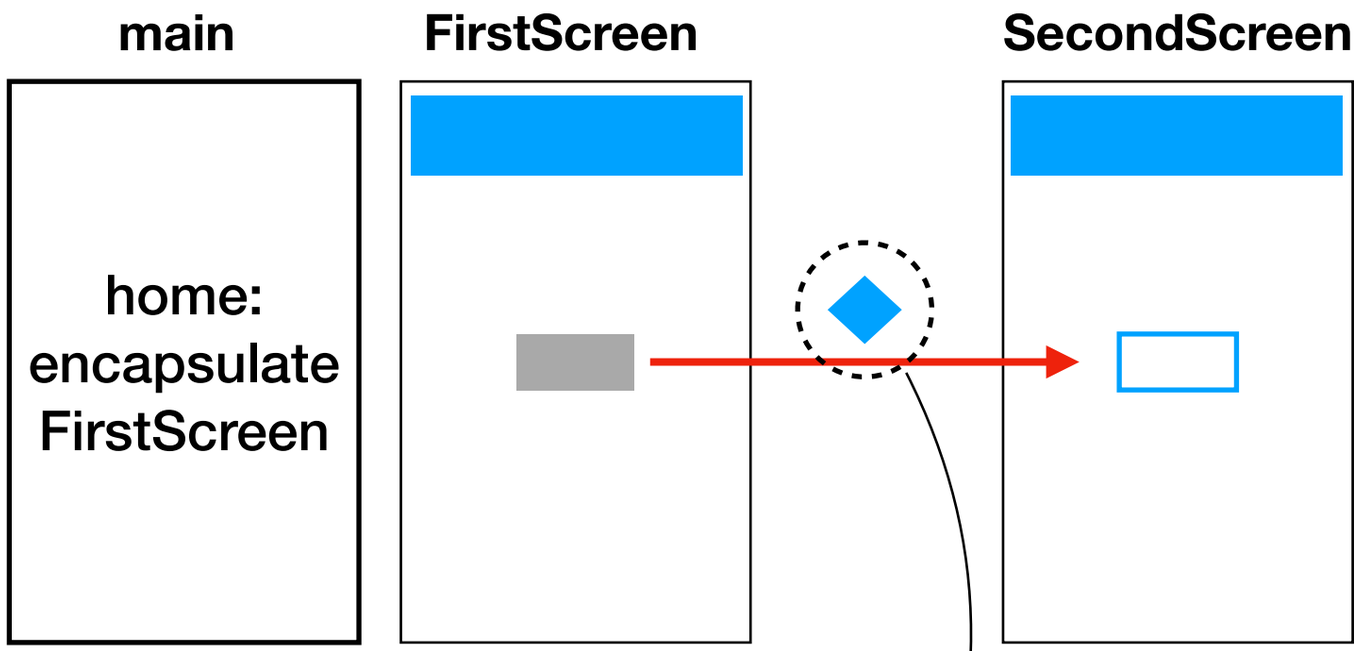
Step 2: firstscreen.dart



Passing Data

Passing data

Using Route



Passing single data, Can use parameter

Passing Multiple data, then create a Class
and define the data as a Class Object

Animations

Animations

1. Implicit Animations

- Use in a View

2. Animation Explicit (Hero)

- Use between View aka Transition

Implicit Animation

Implicitly Animated Widgets

To create Basic animations, Use Animated Widgets:

Align ⇢ AnimatedAlign

Container ⇢ AnimatedContainer

DefaultTextStyle ⇢ AnimatedDefaultTextStyle

Opacity ⇢ AnimatedOpacity

Padding ⇢ AnimatedPadding

PhysicalModel ⇢ AnimatedPhysicalModel

Positioned ⇢ AnimatedPositioned

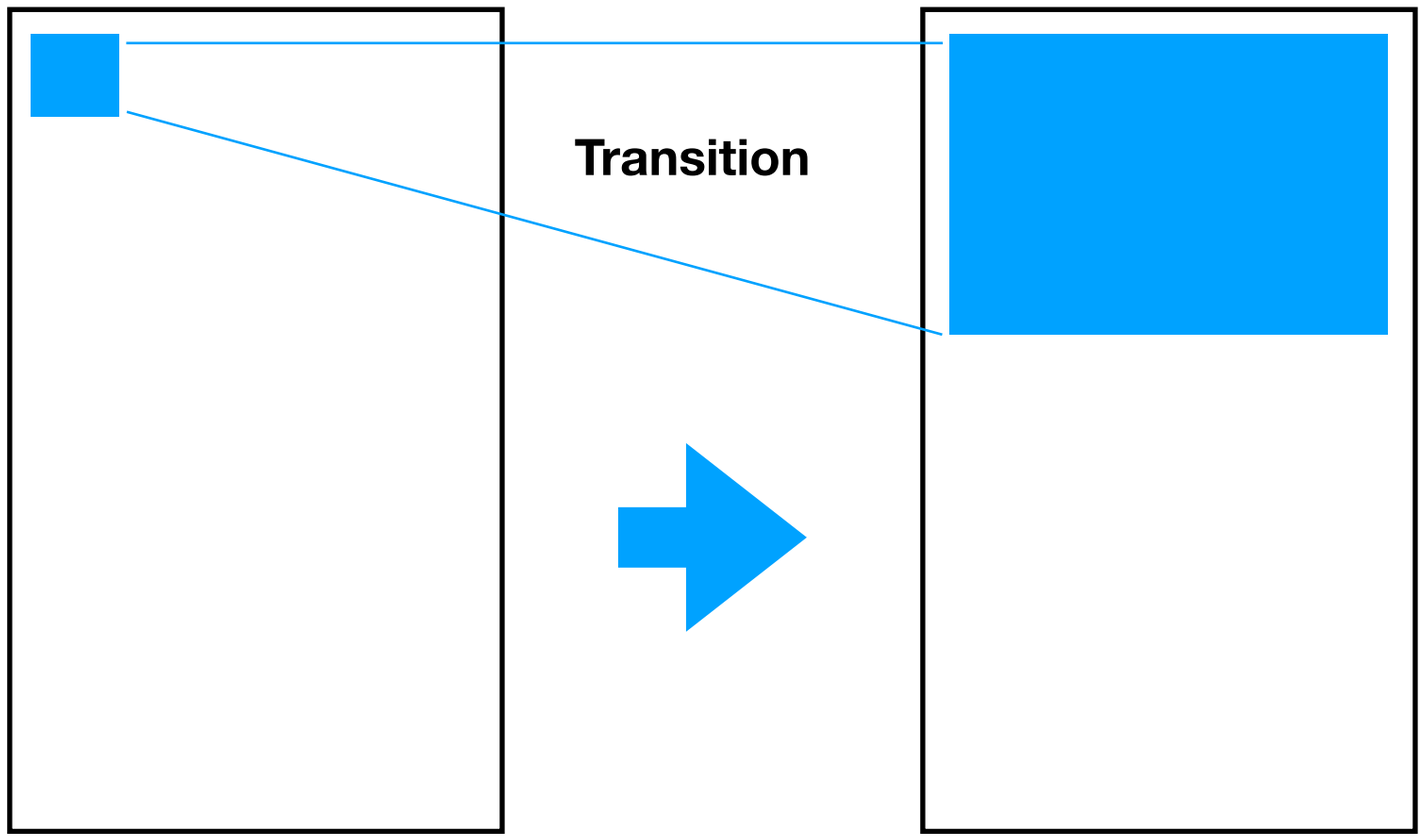
PositionedDirectional ⇢ Animated PositionedDirectional

Theme ⇢ AnimatedThemeSize ⇢ AnimatedSize

Hero Animation

- **The hero refers to the widget that flies between screens.**
- **Create a hero animation using Flutter's Hero widget.**
- **Fly the hero from one screen to another.**
- **The Hero widget implements a style of animation commonly known as shared element transitions or shared element animations.**

Hero Animation



Storage

Type of Storage Options

- **SharedPreferences (key-Value)**
- **SQLite (local DB)**
- **File**
- **Cloud-Based (Online & offline)**

SharedPreferences

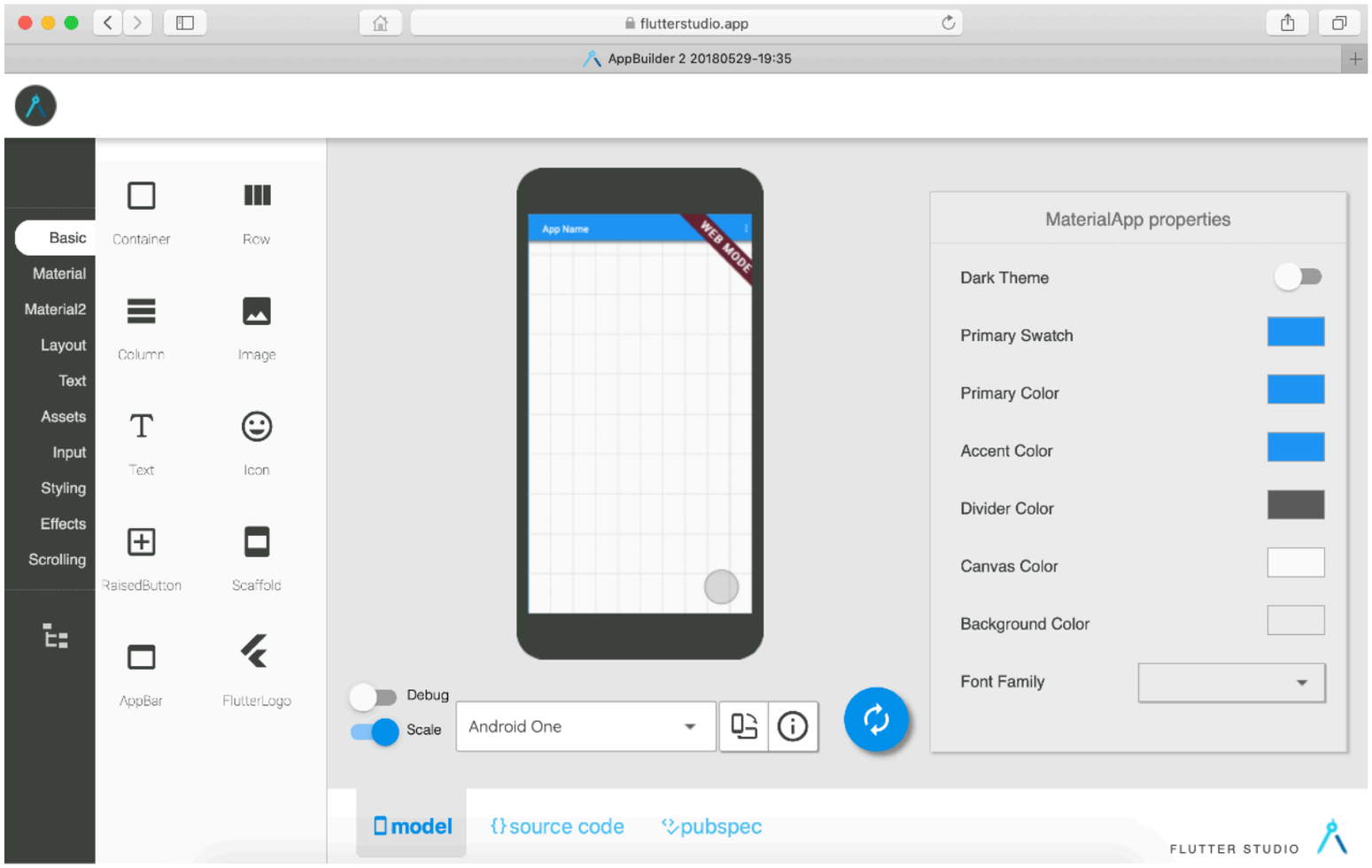
- **Simple Storage to Store data in Local**
- **Very easy to code**

Drag & Drop Tools

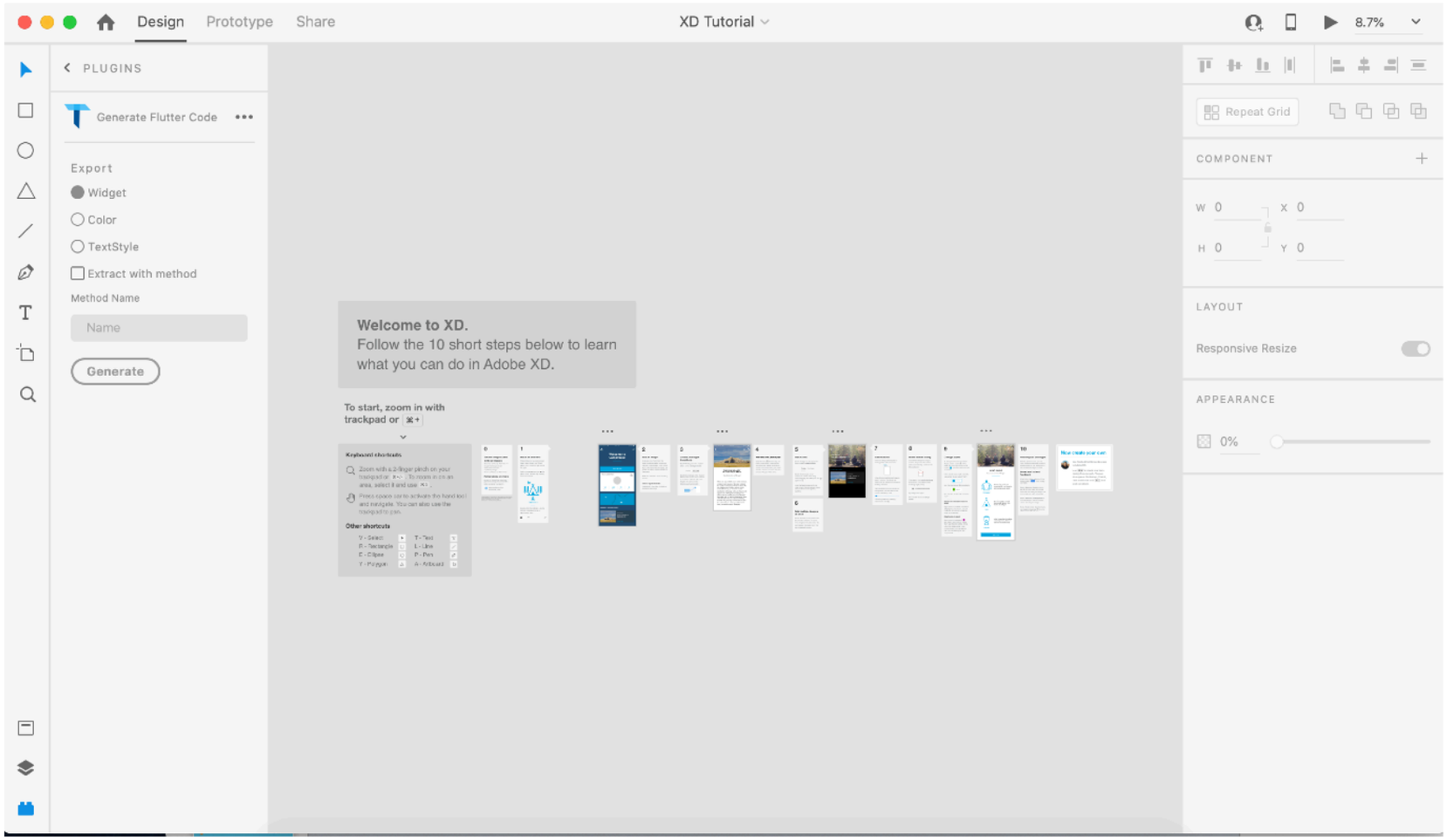
Tools

	Windows	Mac	Linux
Flutter Studio	✓	✓	✓
Adobe XD	✓	✓	
Lunacy	✓		
Sketch App		✓	
Supernova		✓	

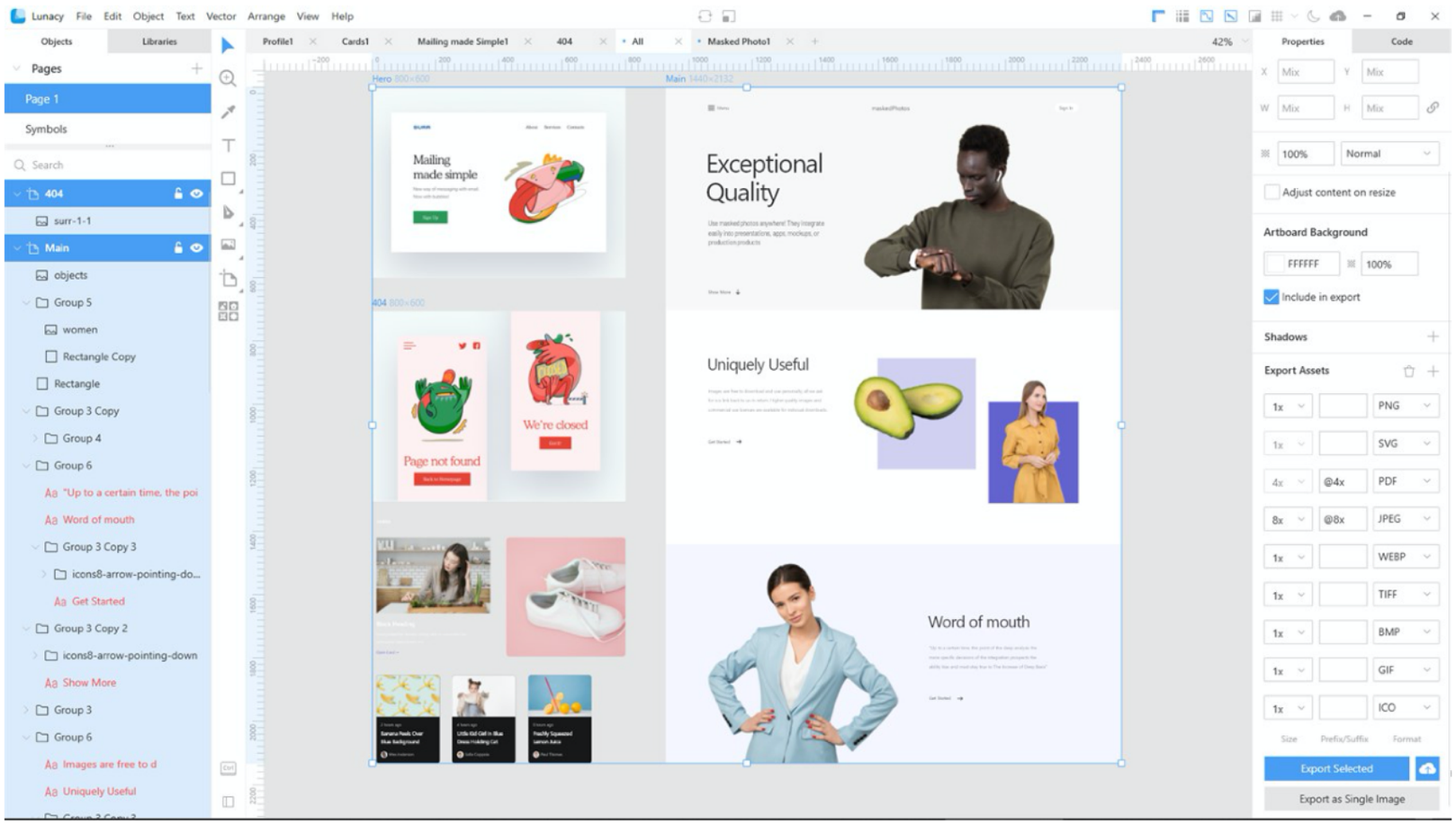
Flutter Studio



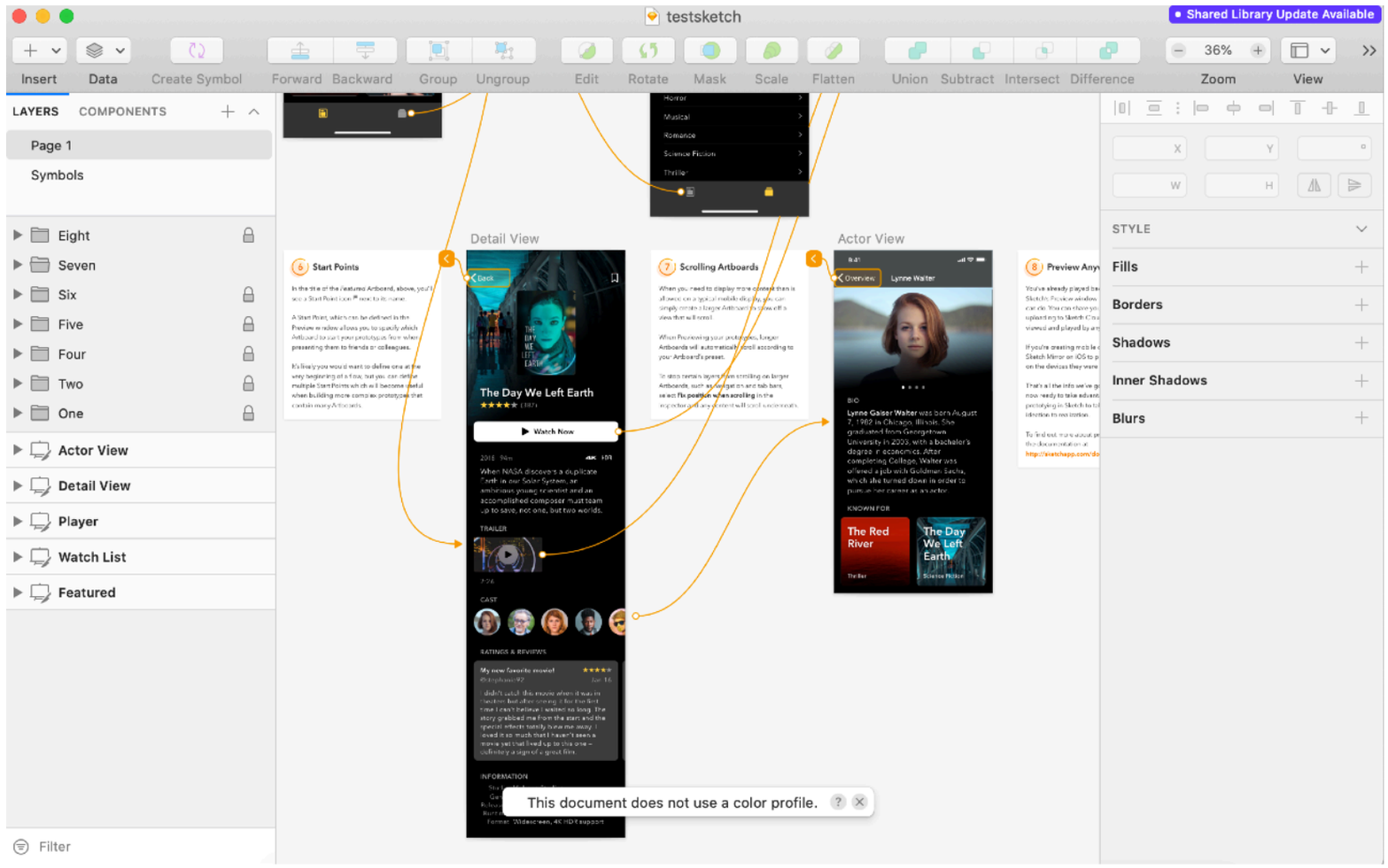
Adobe XD


Design
Prototype
Share
XD Tutorial
8.7%
PLUGINS
Generate Flutter Code
Export
Widget
Color
TextStyle
Extract with method
Method Name
Name
Generate
Welcome to XD.
Follow the 10 short steps below to learn what you can do in Adobe XD.
To start, zoom in with trackpad or ⌘+
Repeat Grid
COMPONENT
W 0
X 0
H 0
Y 0
LAYOUT
Responsive Resize
APPEARANCE
0%

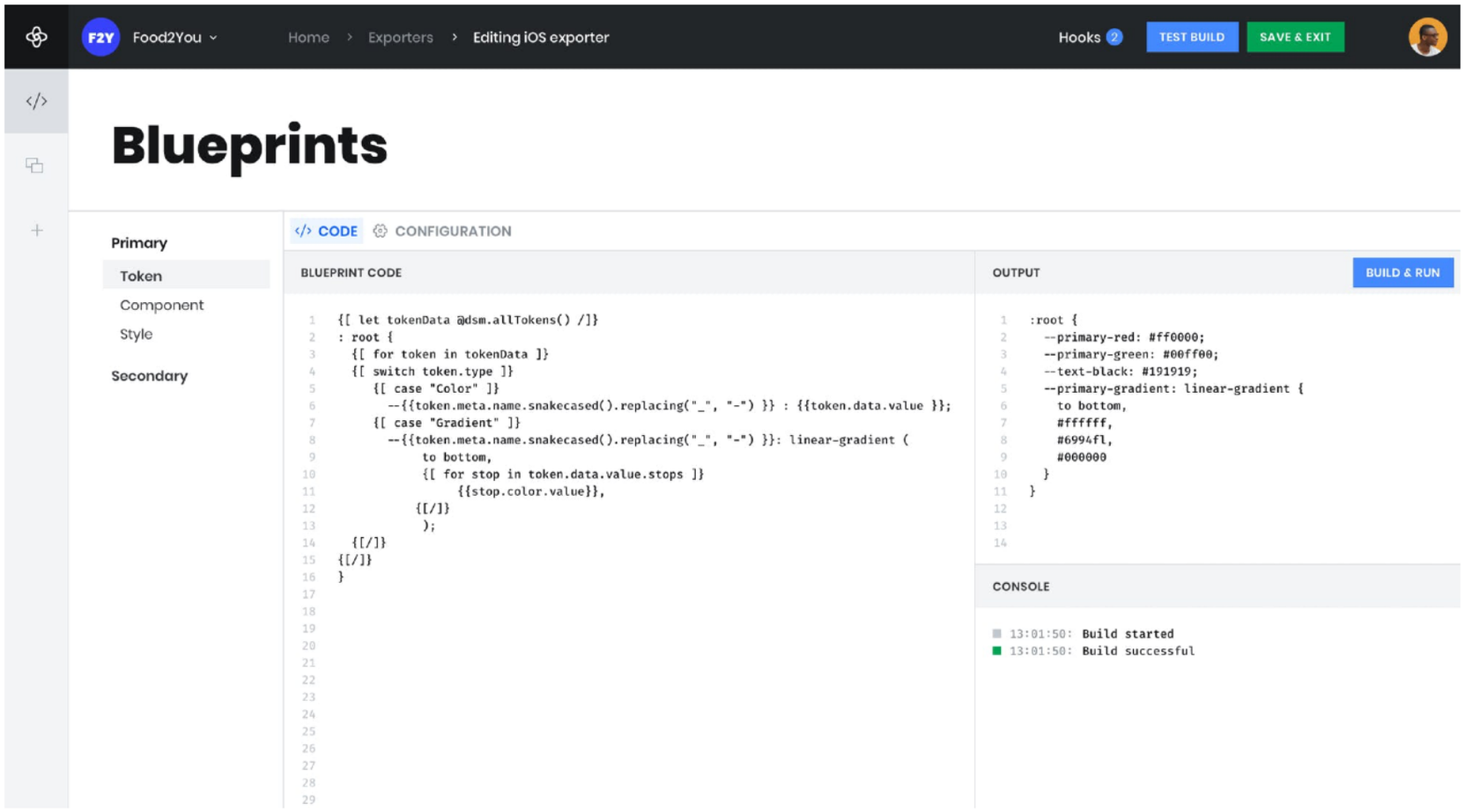
Lunacy


Lunacy File Edit Object Text Vector Arrange View Help
Objects
Libraries
Pages
Page 1
Symbols
Search
404
surr-1-1
Main
objects
Group 5
women
Rectangle Copy
Rectangle
Group 3 Copy
Group 4
Group 6
"Up to a certain time, the poi
Word of mouth
Group 3 Copy 3
icons8-arrow-pointing-do...
Get Started
Group 3 Copy 2
icons8-arrow-pointing-down
Show More
Group 3
Group 6
Images are free to d
Uniquely Useful
Profile1
Cards1
Mailing made Simple1
404
All
Masked Photo1
42%
Hero 800×600
Main 1440×2132
Mailing made simple
Exceptional Quality
404 800×600
Page not found
We're closed
Uniquely Useful
Word of mouth
Properties
Code
Mix
100%
Normal
Adjust content on resize
Artboard Background
FFFFFF
100%
Include in export
Shadows
Export Assets
1x
PNG
SVG
4x
@4x
PDF
8x
@8x
JPEG
WEBP
TIFF
BMP
GIF
ICO
Size
Prefix/Suffix
Format
Export Selected
Export as Single Image

Sketch


testsketch
Shared Library Update Available
Insert
Data
Create Symbol
Forward
Backward
Group
Ungroup
Edit
Rotate
Mask
Scale
Flatten
Union
Subtract
Intersect
Difference
Zoom
View
LAYERS
COMPONENTS
Page 1
Symbols
Eight
Seven
Six
Five
Four
Two
One
Actor View
Detail View
Player
Watch List
Featured
Filter
Detail View
Actor View
Start Points
Scrolling Artboards
The Day We Left Earth
Watch Now
Lynne Gaiser Walter
The Red River
The Day We Left Earth
STYLE
Fills
Borders
Shadows
Inner Shadows
Blurs
This document does not use a color profile.

Supernova


F2Y
Food2You
Home
Exporters
Editing iOS exporter
Hooks 2
TEST BUILD
SAVE & EXIT
Blueprints
Primary
Token
Component
Style
Secondary
CODE
CONFIGURATION
BLUEPRINT CODE
OUTPUT
BUILD & RUN
{[let tokenData @dsm.allTokens() /]}
: root {
{[for token in tokenData]}
{[switch token.type]}
{[case "Color"]}
--{{token.meta.name.snakecased().replacing("_", "-") }} : {{token.data.value }};
{[case "Gradient"]}
--{{token.meta.name.snakecased().replacing("_", "-") }}: linear-gradient (
to bottom,
{[for stop in token.data.value.stops]}
{{stop.color.value}},
{[/]}
);
{[/]}
{[/]}
}
:root {
--primary-red: #ff0000;
--primary-green: #00ff00;
--text-black: #191919;
--primary-gradient: linear-gradient {
to bottom,
#ffffff,
#6994fl,
#000000
}
}
CONSOLE
13:01:50: Build started
13:01:50: Build successful

Lessons Day 1

1. **Lesson: Environment**
   Hello World, Images

2. **Lesson: Creating UI & Layouts**
   Simple Layout, Reusable widgets

3. **Lesson: Interactions**
   Buttons, Alerts, SnackBar

4. **Lesson: Navigation**
   Using Navigation, Passing Data

5. **Lesson: Listings**
   ListView, ListBuilder, JSON with ListBuilder