江西理工大学 信息工程学院

JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY SCHOOL OF INFORMATION ENGINEERING

Mobile application development

移动应用开发

Flutter

Lecture 26:

Flutter_Section

Dr Ata Jahangir Moshayedi

Prof Associate ,

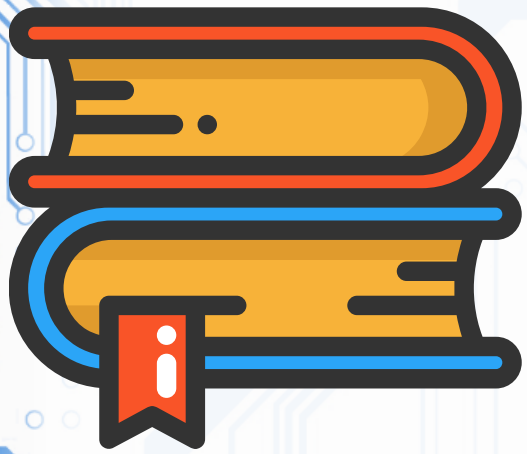
School of information engineering Jiangxi university of science and technology, China

EMAIL: ajm@jxust.edu.cn

Autumn _2021

江西理工大學 信息工程学院

JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY SCHOOL OF INFORMATION ENGINEERING



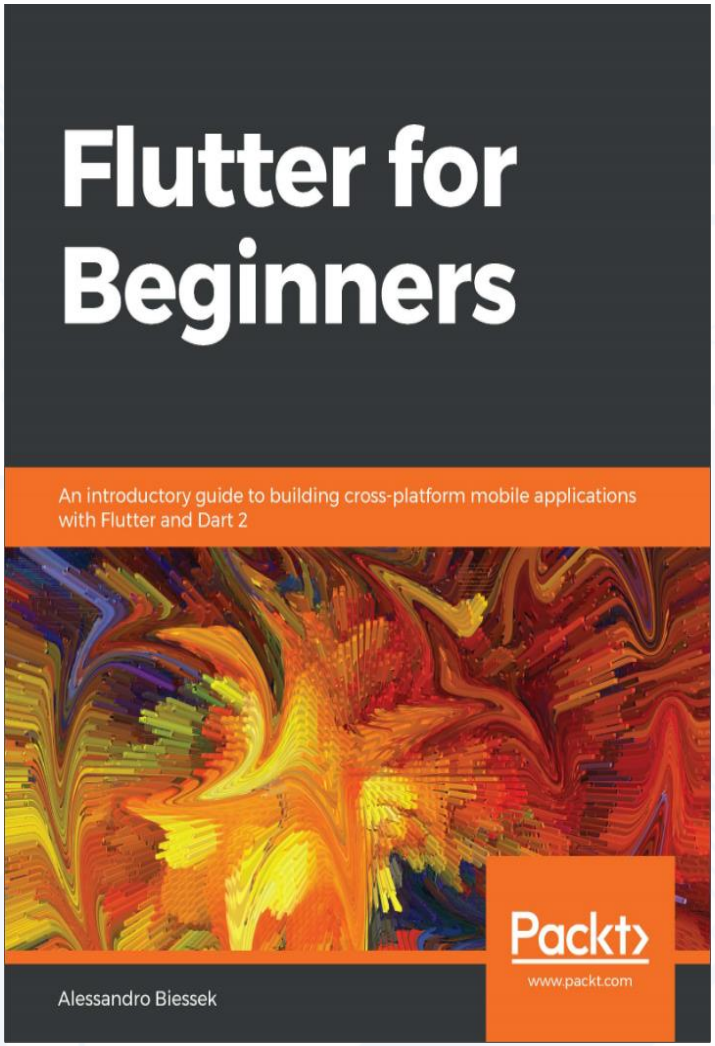
Mobile application development

移动应用开发

LECTURE 26: Flutter short review and Project

江西理工大學 信息工程学院

JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY SCHOOL OF INFORMATION ENGINEERING



Flutter for Beginners

An introductory guide to building cross-platform mobile applications with Flutter and Dart 2

Alessandro Biessek

What is Flutter?

Flutter

- Flutter is a free and open-source mobile UI framework created by Google and released in May 2017.
- *In a few words,* ***it allows you to create a native mobile application with only one codebase.***
- *This means that you can use one programming language and one codebase to create two different apps (for iOS and Android).*
- ***Flutter is Google's mobile app SDK for crafting high-quality native interfaces on iOS and Android in record time.***
- ***Flutter works with existing code, is used by developers and organisations around the world, and is free and open source.***
- **Flutter** is used to develop cross platform applications for Android, iOS, Linux, Mac, Windows, Google Fuchsia, and the web from a single codebase.

Flutter History in breif

- **Initial release date:May 2017**
  - Preview release: Release Preview 1(v0.6.0) / July 2018
  - Developer(s):Google and community
  - Original author(s):Google Platform
  - Development: Windows, MacOS and Linux, Target:Android, iOS and Google Fuchsia

    Written in: C, C++, Dart, Skia Graphics Engine https://hackr.io/tutorials/learn-flutter

History

Flutter

- The first version of Flutter was known by the codename "Sky" and ran on the Android operating system. It was unveiled at the 2015 Dart developer summit with the stated intent of being able to render consistently at 120 frames per second.
- During the keynote of Google Developer Days in Shanghai in September 2018, Google announced Flutter Release Preview 2, which is the last big release before Flutter 1.0. On December 4th of that year, Flutter 1.0 was released at the Flutter Live event, denoting the first "stable" version of the Framework. On December 11, 2019, Flutter 1.12 was released at the Flutter Interactive event.
- On May 6, 2020, the Dart software development kit (SDK) in version 2.8 and the Flutter in version 1.17.0 were released, where support was added to the Metal API, improving performance on iOS devices (approximately 50%), new Material widgets, and new network tracking.

History

Flutter

- **On March 3, 2021, Google released Flutter 2 during an online Flutter Engage event.** This major update brought official support for web-based applications with new CanvasKit renderer and web specific widgets, early-access desktop application support for Windows, macOS, and Linux and improved Add-to-App APIs.
- This release included sound null-safety, which caused many breaking changes and issues with many external packages, but the Flutter team included instructions to mitigate these changes as well.
- On September 8th, 2021, the Dart SDK in version 2.14 and Flutter version 2.5 were released by Google. The update brought improvements to the Android Full-Screen mode and the latest version of Google's Material Design called Material You. Dart received two new updates, the newest lint conditions have been standardized and preset as the default conditions as well Dart for Apple Silicon is now stable.

Framework architecture

- The major components of Flutter include:
  1. **Dart platform**
  2. **Flutter engine**
  3. **Foundation library**
  4. **Design-specific widgets**
  5. **Flutter Development Tools (DevTools)**

Framework architecture

Flutter

1. Dart platform

- Flutter apps are written in the Dart language and make use of many of the language's more advanced features.
- On Windows, macOS, and Linux Flutter runs in the Dart virtual machine, which features a just-in-time execution engine. While writing and debugging an app, Flutter uses Just In Time compilation, allowing for "hot reload", with which modifications to source files can be injected into a running application.
- Flutter extends this with support for stateful hot reload, where in most cases changes to source code are reflected immediately in the running app without requiring a restart or any loss of state.
- For better performance, release versions of Flutter apps targeting Android and iOS are compiled with ahead-of-time (AOT) compilation.

Framework architecture

Flutter

2. Flutter engine

- Flutter's engine, written primarily in C++, provides low-level rendering support using Google's Skia graphics library. Additionally, it interfaces with platform-specific SDKs such as those provided by Android and iOS.

- The Flutter Engine is a portable runtime for hosting Flutter applications. It implements Flutter's core libraries, including animation and graphics, file and network I/O, accessibility support, plugin architecture, and a Dart runtime and compile toolchain.

- Most developers interact with Flutter via the Flutter Framework, which provides a reactive framework and a set of platform, layout, and foundation widgets.

Framework architecture

3. Foundation library

- The Foundation library, written in Dart, provides basic classes and functions that are used to construct applications using Flutter, such as APIs to communicate with the engine.

Framework architecture

Flutter

4. Design-specific widgets

- The Flutter framework contains two sets of widgets that conform to specific design languages:
- Material Design widgets implement Google's design language of the same name, and *Cupertino* widgets implement Apple's iOS Human interface guidelines.

Widgets

- Flutter uses a variety of widgets to deliver a fully functioning application.
- These widgets are Flutter's framework architecture.
- Flutter's Widget Catalog provides a full explanation and API on the framework.

Basics of Widgets in Flutter

- Widgets are generally defined in three basic types: Stateful widgets, Stateless widgets, and Inherited widgets. Being the central class hierarchy in the Flutter framework the three basic types of widgets are used in the construction of every Flutter application.
- Although all the instances of a widget are immutable, the Stateful widget allows the interaction between user and application.
- By giving access to the method setState, the state can be maintained in separate state objects. Alternatively, the Stateless widget acts as a constant, and before anything displayed can be changed, the widget has to be recreated.
- The Inherited widget works by allowing another widget to subscribe to the Inherited widget's state allowing the state to be passed down to its children.

Flutter important parts:

Flutter

Flutter consists of two important parts:

- An SDK (Software Development Kit):

  A collection of tools that are going to help you develop your applications. This includes tools to compile your code into native machine code (code for iOS and Android).

- A Framework (UI Library based on widgets):

  A collection of reusable UI elements (buttons, text inputs, sliders, and so on) that you can personalize for your own needs.

Flutter important parts:

- To develop with Flutter, you will use a programming language called Dart. The language was created by Google in October 2011, but it has improved a lot over these past years.
- Dart focuses on front-end development, and you can use it to create mobile and web applications.
- *If you know a bit of programming, Dart is a typed object programming language. You can compare Dart's syntax to JavaScript.*

Why We Should Learn Flutter

Simple to learn and use

- Flutter is a modern framework, and you can feel it! It’s way simpler to create mobile applications with it. If you have used Java, Swift, or React Native, you'll notice how Flutter is different.
- You can create a real native application without a bunch of code.
- Quick compilation: maximum productivity
- You can change your code and see the results in real-time. It’s called Hot-Reload. It only takes a short amount of time after you save to update the application itself.
- Significant modifications force you to reload the app. But if you do work like design, for example, and change the size of an element, it’s in real-time!

Why You Should Learn Flutter

A growing community

- **Flutter has a robust community, and it's only the beginning!**

1. **Flutter Awesome:** An awesome list that curates the best Flutter libraries and tools. This website publishes daily content with lots of examples, application templates, advice, and so on.
2. **Awesome Flutter:** A GitHub repository (linked to Flutter Awesome) with a list of articles, videos, components, utilities, and so on.
3. **It's all widgets!:** An open list of apps built with Flutter.
4. **Flutter Community:** A Medium publication where you can find articles, tutorials, and much more.

Top reasons to use it:

Flutter

- **It’s cheaper to develop** a mobile application with Flutter, because you don’t need to create and maintain two mobile apps (one for iOS and one for Android).
- **It’s performant** – you won't notice the difference between a native application and a Flutter app.
- **It’s beautiful** – you can easily use widgets provided by Flutter and personalize it to create a valuable UI for your customers.

Top reasons to use it:

Flutter

- Supported by Android Studio and VS Code Flutter is available on different IDEs.
- *The two main code editors for developing with this technology are Android Studio (IntelliJ) and VS Code.*
- **Android Studio is a complete software with everything already integrated. You have to download Flutter and Dart plugins to start.**
- **VS Code is a lightweight tool, and everything is configurable through plugins from the marketplace.**
- You are free to choose your preferred IDE!

The Essential Programming Features Of Dart

Why Learn Dart?

- Flutter has now become one of the best cross platform mobile app development kit.
- We can build powerful, high performant, beautiful and highly customizable native mobile apps with Flutter.
- Furthermore, it is now also moving towards reaching the web development community.
- So, one will also be able to build for the mobile, desktop and web with the same code base in the near future with the help of Flutter.

How Much Dart Should I Know To Learn Flutter?

- *The answer to this question is obviously, the more the better!*
- *But with that being said, do I have to be an expert on Dart to start out with Flutter?*
- *The answer for the second question is "No".*

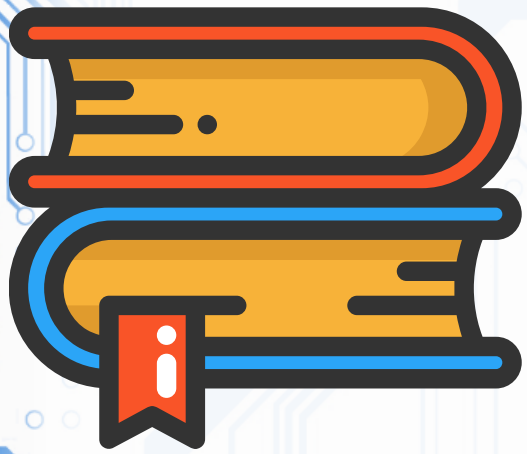
So, here's a bare minimum list of features you should be aware of when working with Dart and Flutter.

- **Dart Keywords**
- **Dart Data Types**
  - Numbers, Strings, DateTime, Bool etc.
  - Dart Collection Types: Lists, Sets And Maps
- **Control Statements**
  - If/Else
  - List Iterations With For Each, Map And For Loops
  - Switch Statements
- **Object Oriented Programming in Dart**
  - Classes
  - Functions
  - Mixins
- **Working With Futures, Streams and Async Operations In Dart**
- **Working With JSON Data**

https://stacksecrets.com/dart/for-loop-for-each-loop-and-map-in-dart

江西理工大學 信息工程学院

JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY SCHOOL OF INFORMATION ENGINEERING



Mobile application development

移动应用开发

Flutter_Installation

江西理工大學 信息工程学院

JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY SCHOOL OF INFORMATION ENGINEERING

Flutter_Installation

Using Flutter in China

If you'd like to install Flutter using an installation bundle (SDK), you can replace the domain of the original URL with a trusted mirror to speed it up.

For example:

1.Original URL:

https://storage.googleapis.com/flutter_infra_release/releases/stable/windows/flutter_windows_v1.0.0-stable.zip

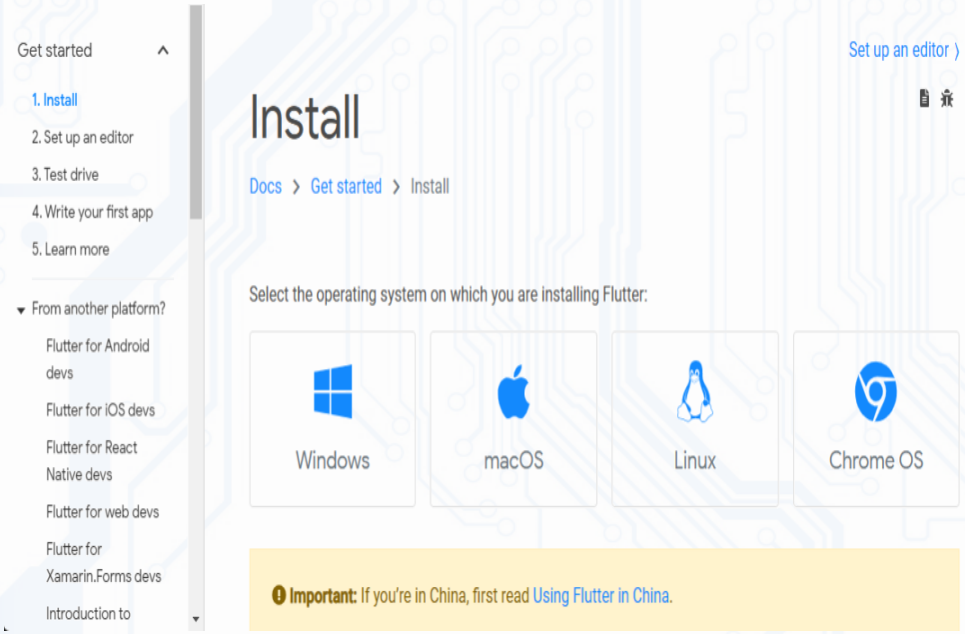
1.Mirrored URL:

https://storage.flutter-io.cn/flutter_infra_release/releases/stable/windows/flutter_windows_v1.0.0-stable.zip

Step 1: **Installation**

Installing Flutter Step 1:

Navigate to https://flutter.dev/docs/get-started/install and select your operating system
After installation send to Documents folder



System requirements

To install and run Flutter, your development environment must meet these minimum requirements:

Operating Systems: Windows 7 SP1 or later (64-bit), x86-64 based.

Disk Space: 1.64 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)**
- **Git for Windows 2.x, with the Use Git from the Windows Command Prompt option.**

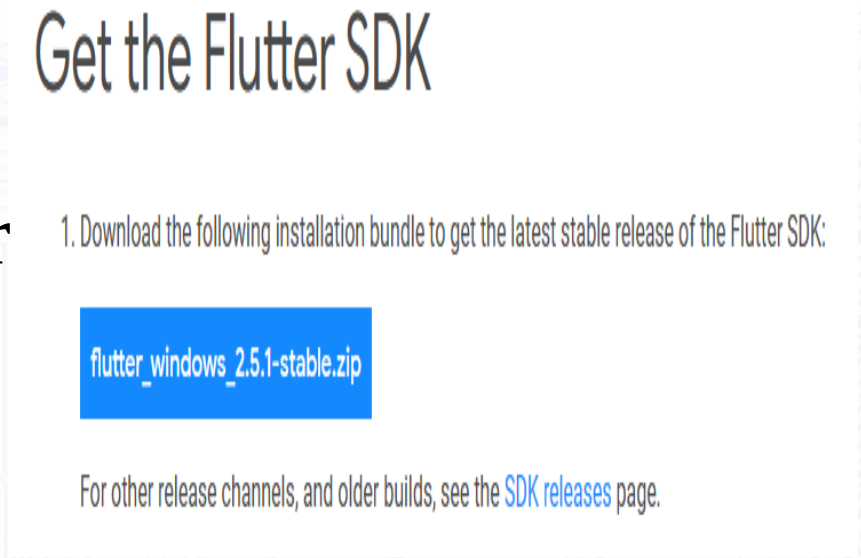
If Git for Windows is already installed, make sure you can run git commands from the command prompt or PowerShell.

Step 2: **Installation**

Flutter

Step 2:

Download the latest stable build of the Flutter SDK for your OS

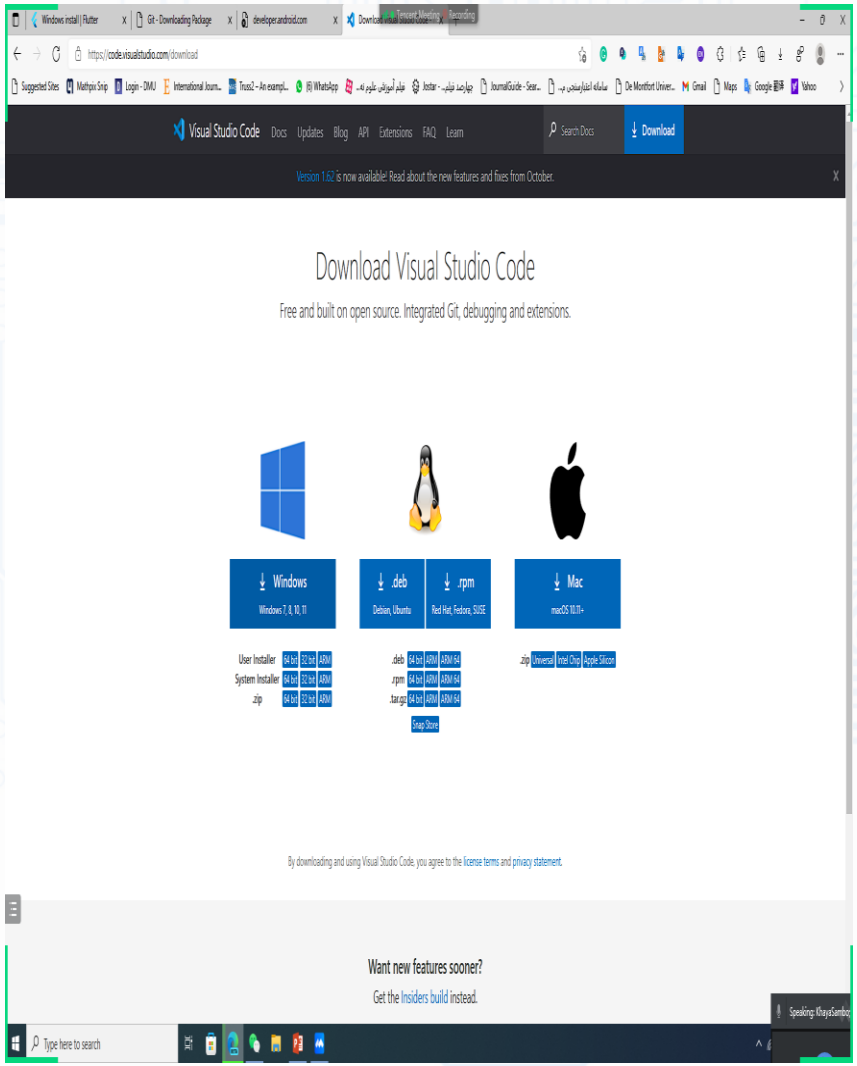


Step 3: **Installation**

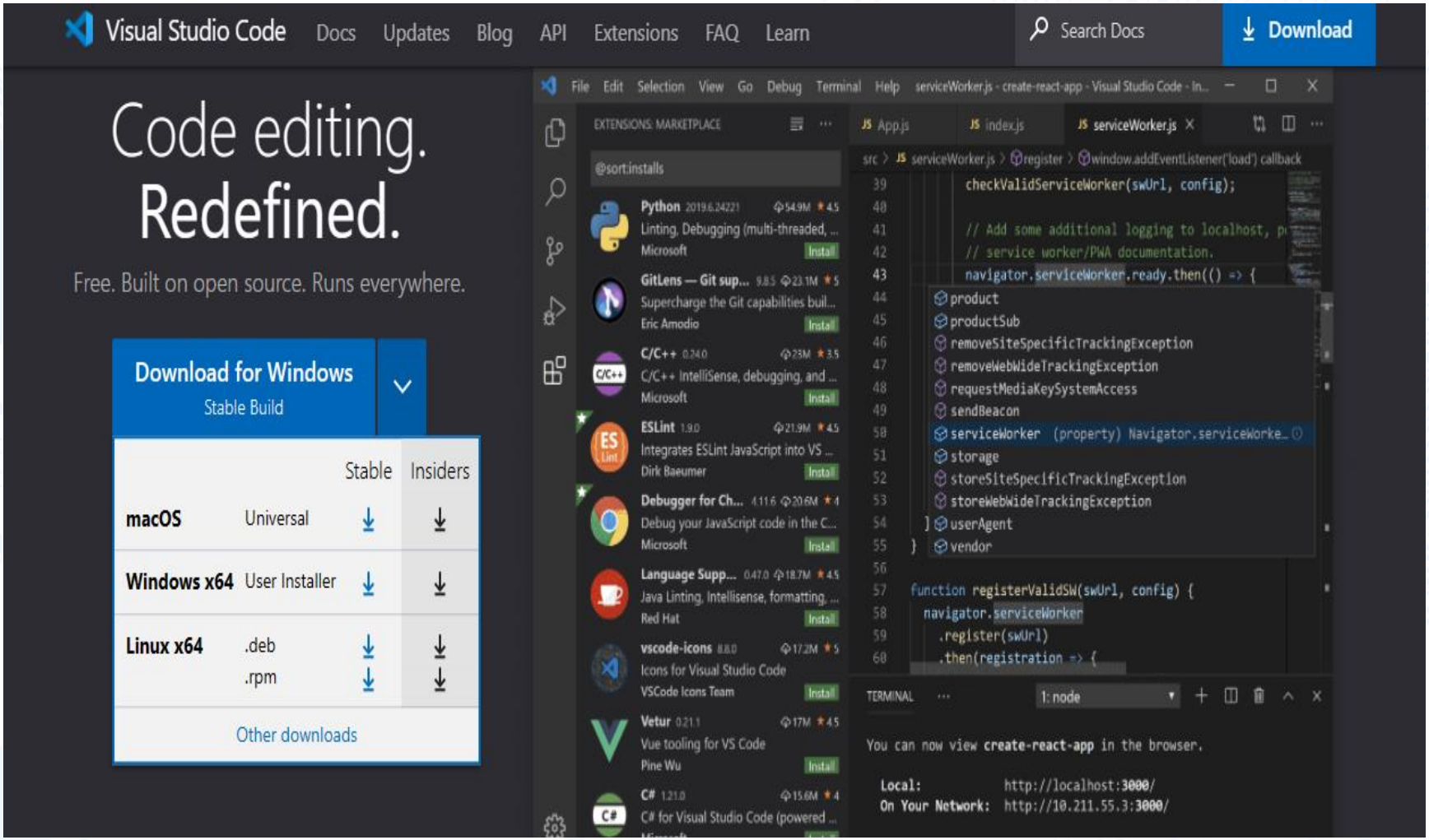
code.visualstudio link:

https://code.visualstudio.com/download

Install Visual Studio Code by following this link https://code.visualstudio.com/ or Use Android Studio as your IDE

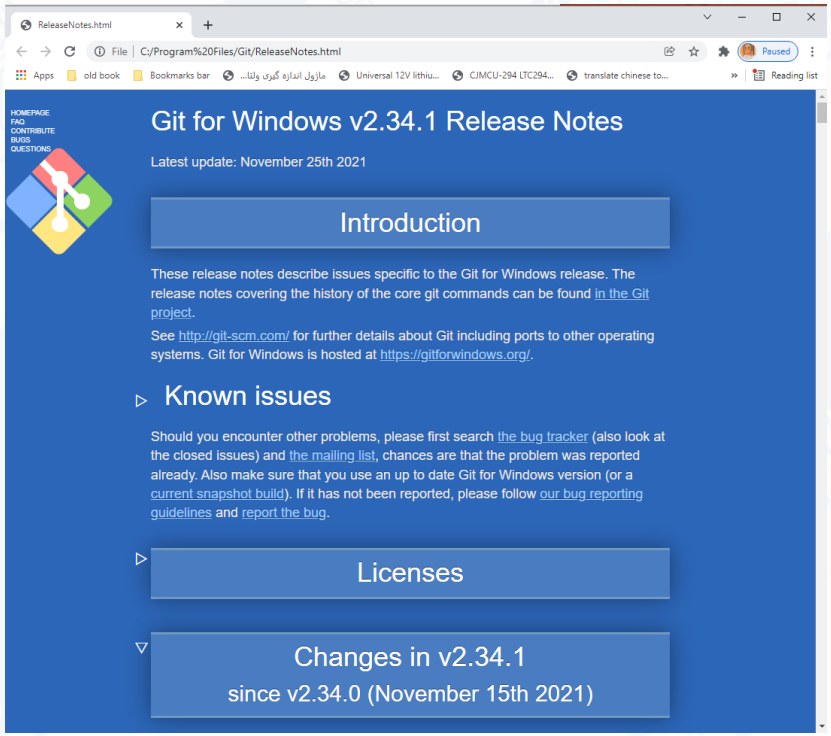


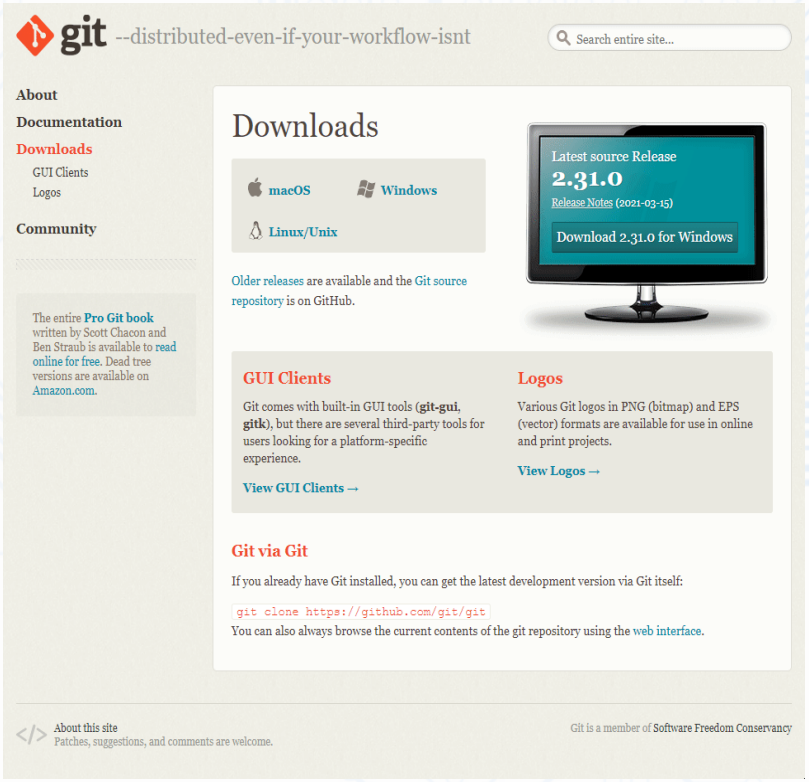
Step 3: **Installation**



Step 4: **Installation**

- Step 4:
- Download Git from this link https://git-scm.com/downloads

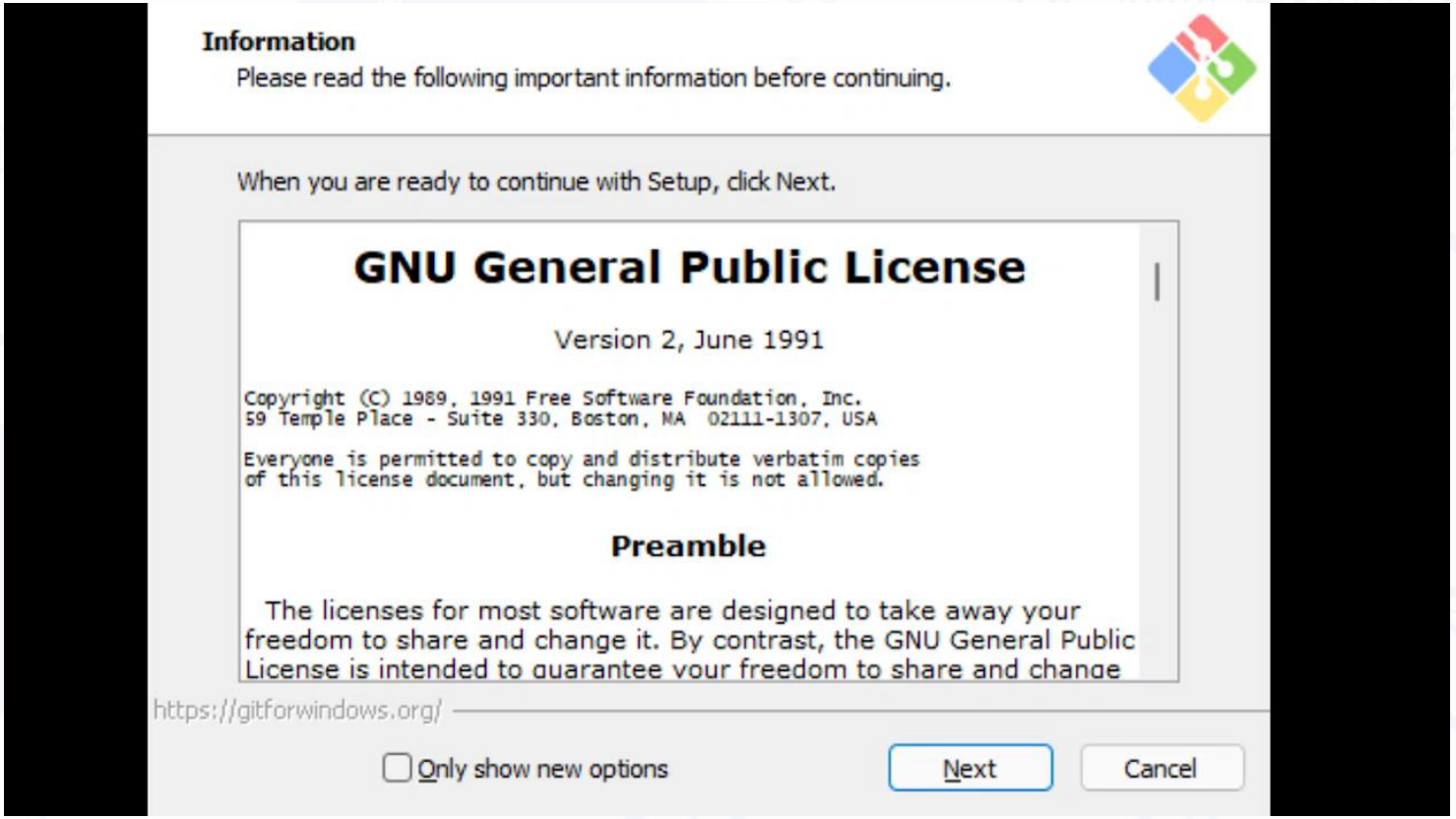




Step 4: **Installation**

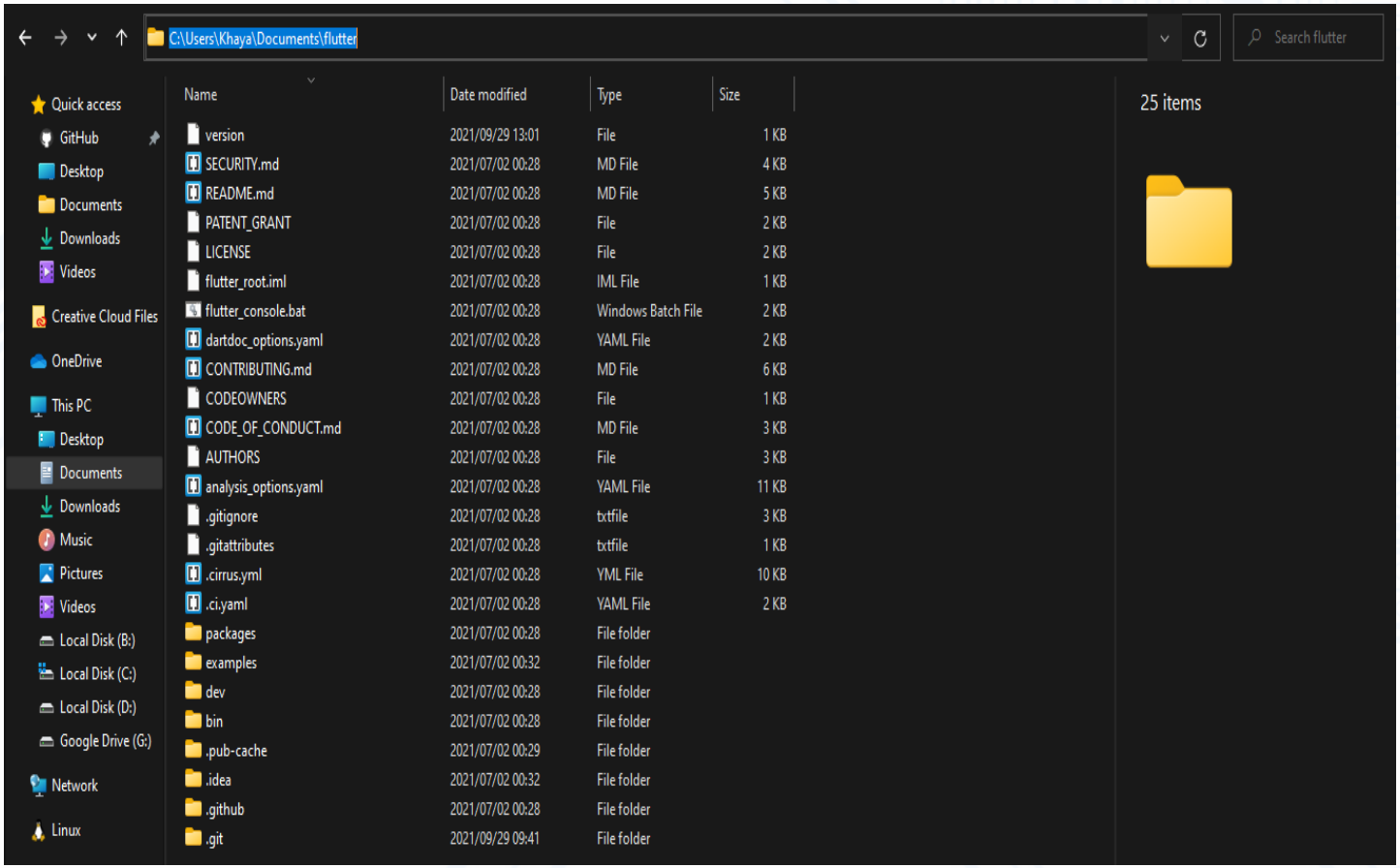
Flutter

GIT Installation



Step 4: Installation

Install Git in its default configuration



Step 5: **Installation**

Flutter

Download Android Studio from this link:

https://developer.android.com/studio?gclid=CjwKCAjwndCKBhAkEiwAgSDKQVjz8zKywgnQbflC4DBAgCEfJAzmFOogXn4J8nASHuYZJs1iLCaaEBoCCWQQAvD_BwE&gclsrc=aw.ds#downloads

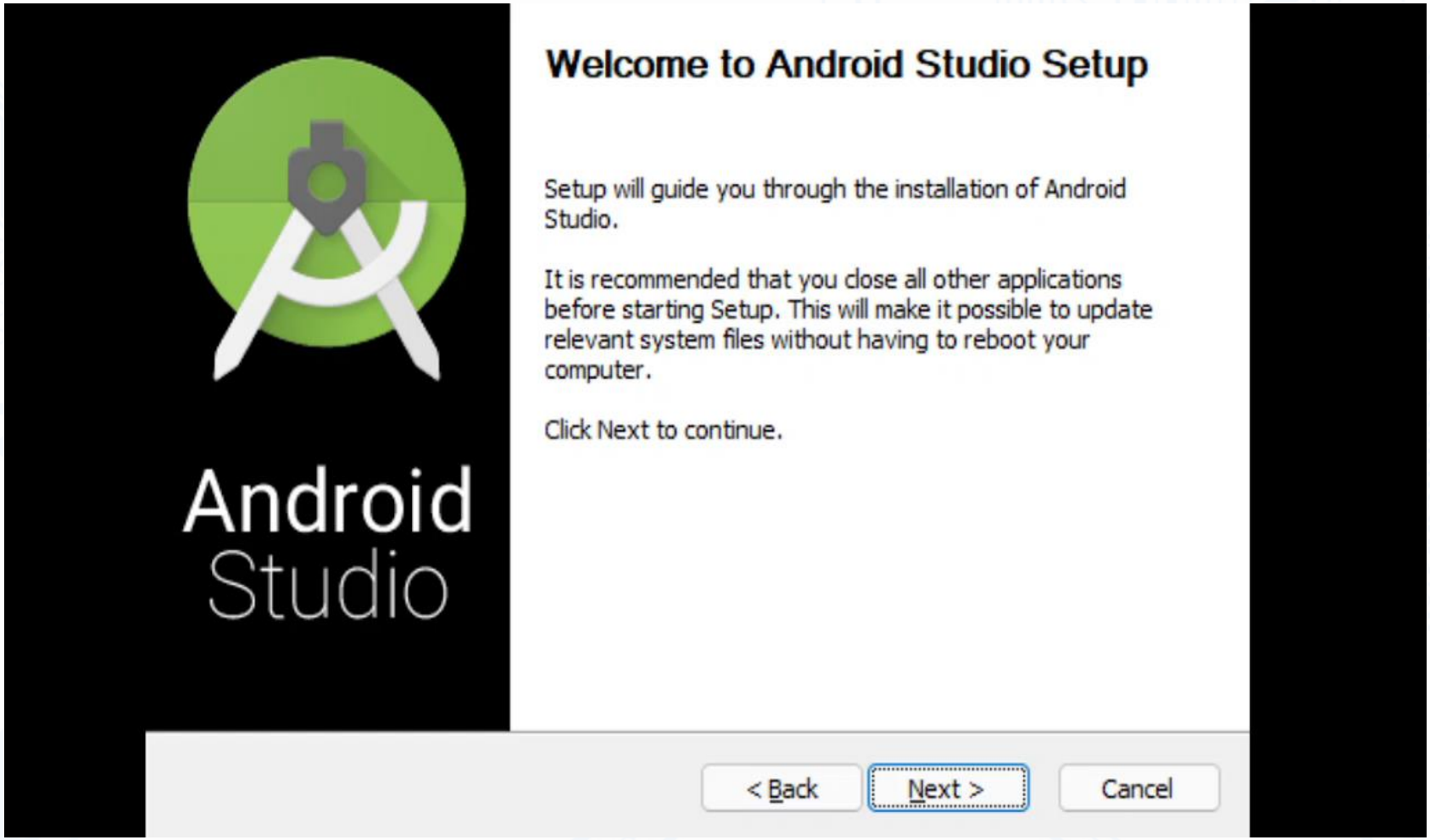
Step 6: **Installation**

Step 6: Create the folder

C:\Users\<your-pc-name>\Documents\flutter and unzip the Flutter into this directory

Step 6: **Installation**

Flutter

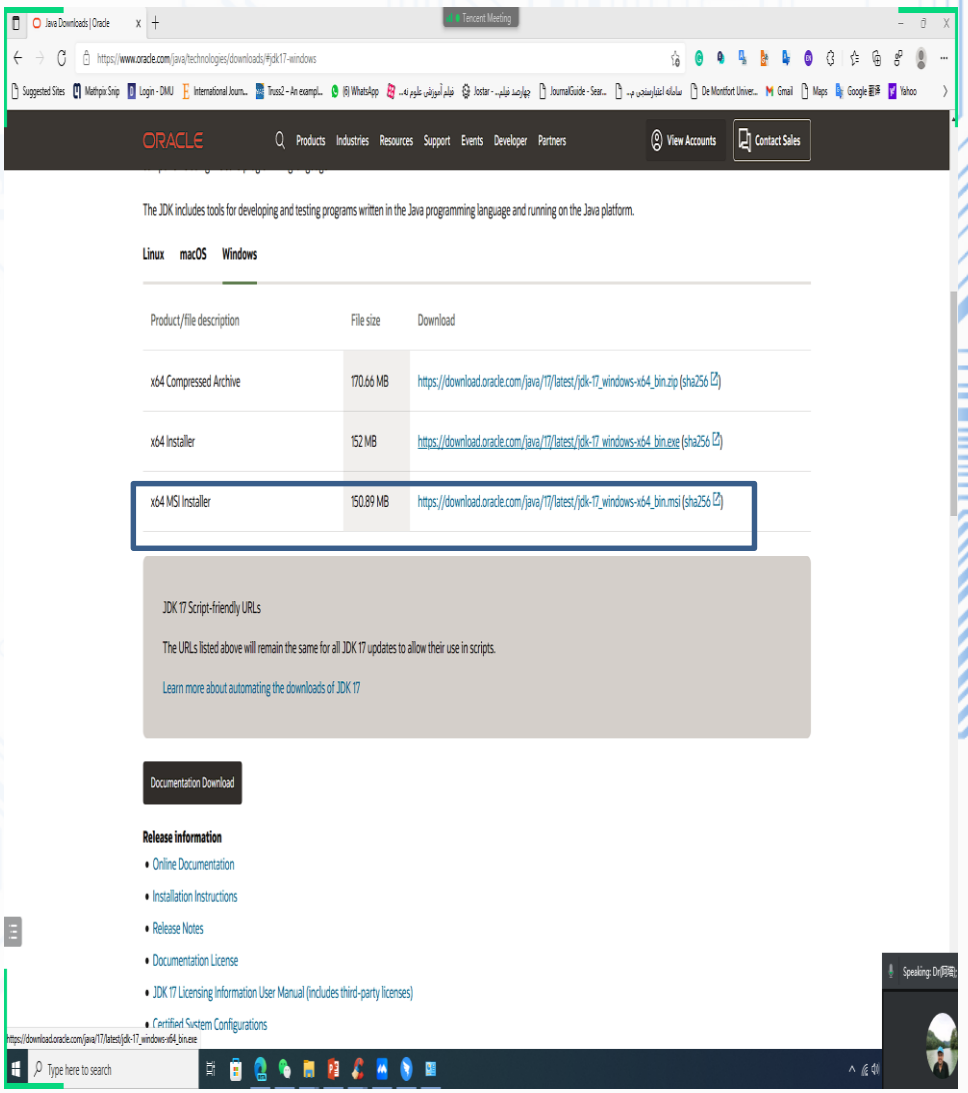


Step 7: **Installation**

Flutter

Step7: Java installation

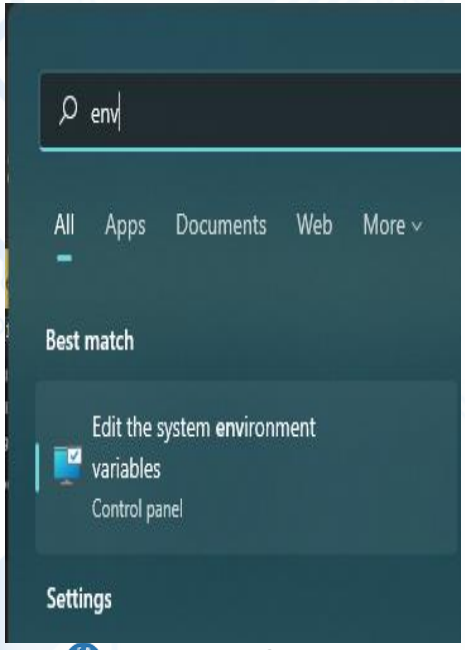
- https://www.oracle.com/java/technologies/downloads/#jdk17-windows

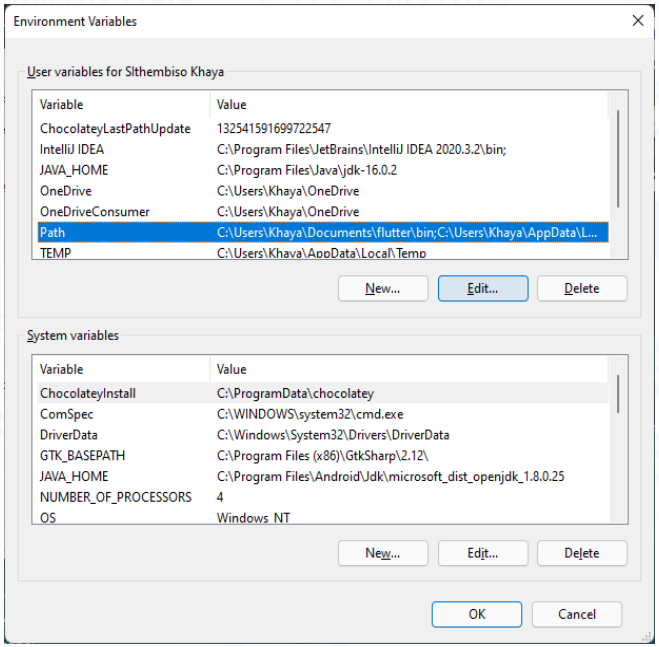


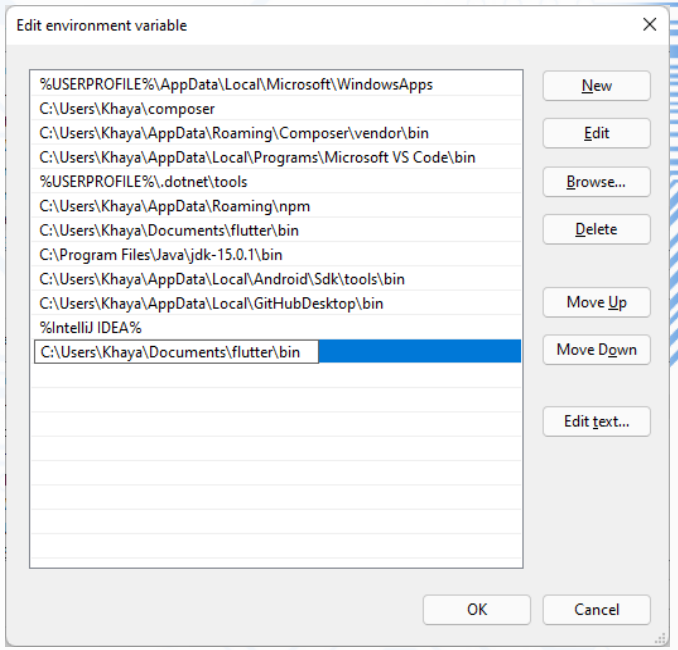
Step 8: **Installation**

Step 8: Copy the path

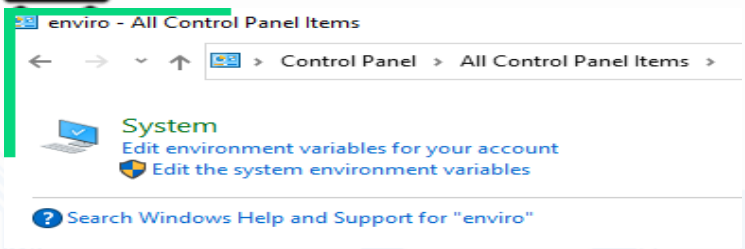
C:\Users\<your-pc-name>\Documents\flutter\bin Save the Flutter path in the environment variables by navigating to search and typing env and select Edit System Environment Variables > Environment Variables > select the “Path” Variable and press the “Edit…” button > Press “New” button and add paste the path that you copied > press OK

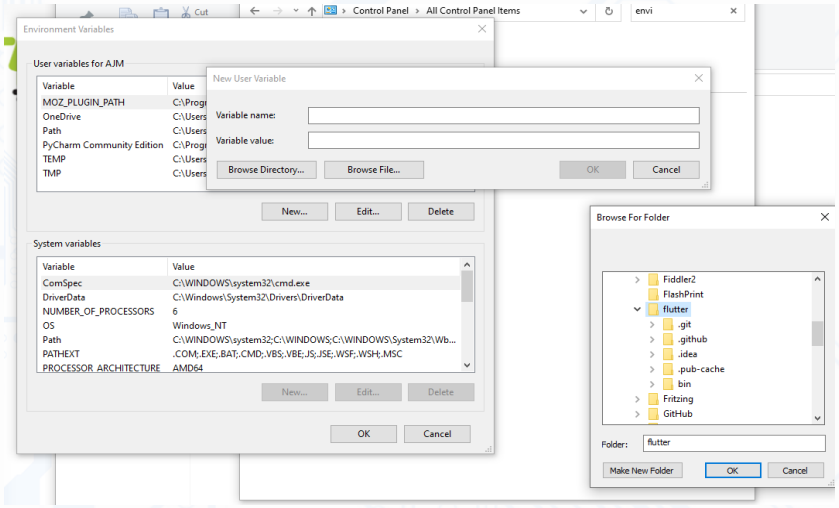


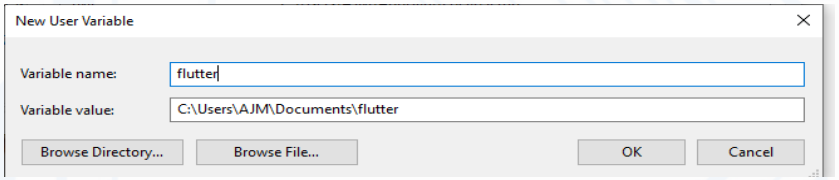


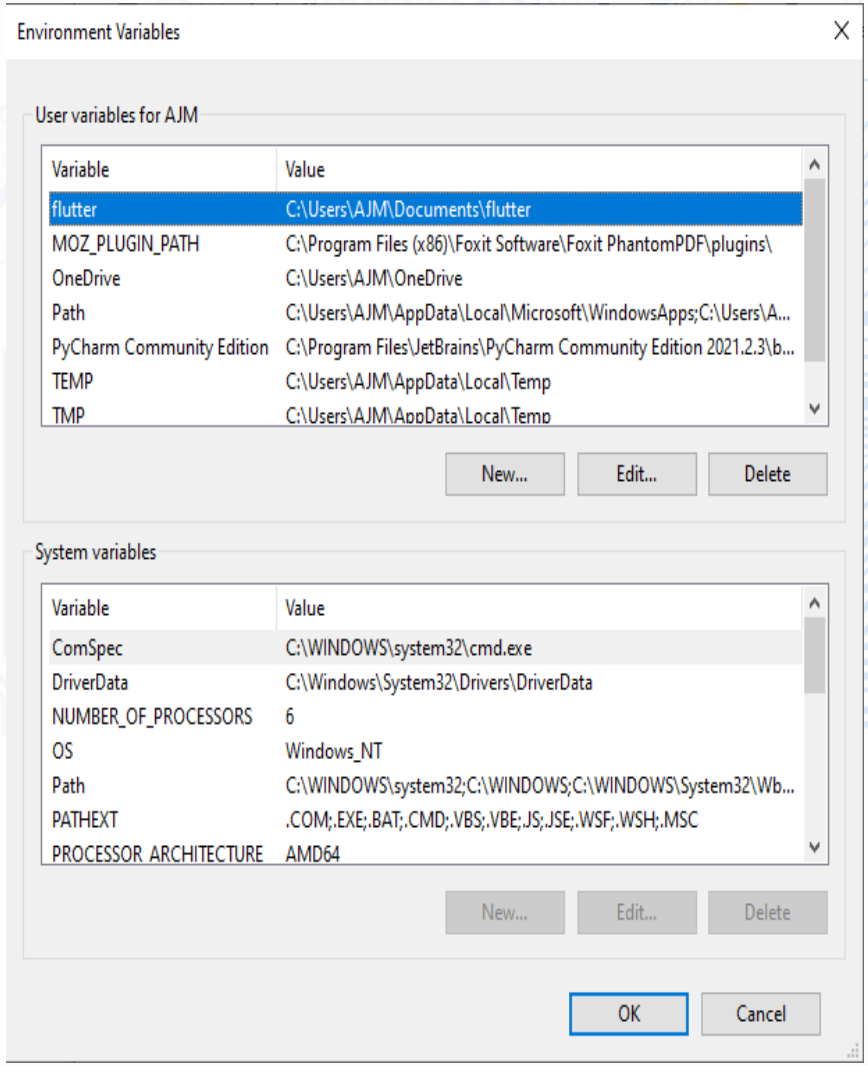


On error in Installation

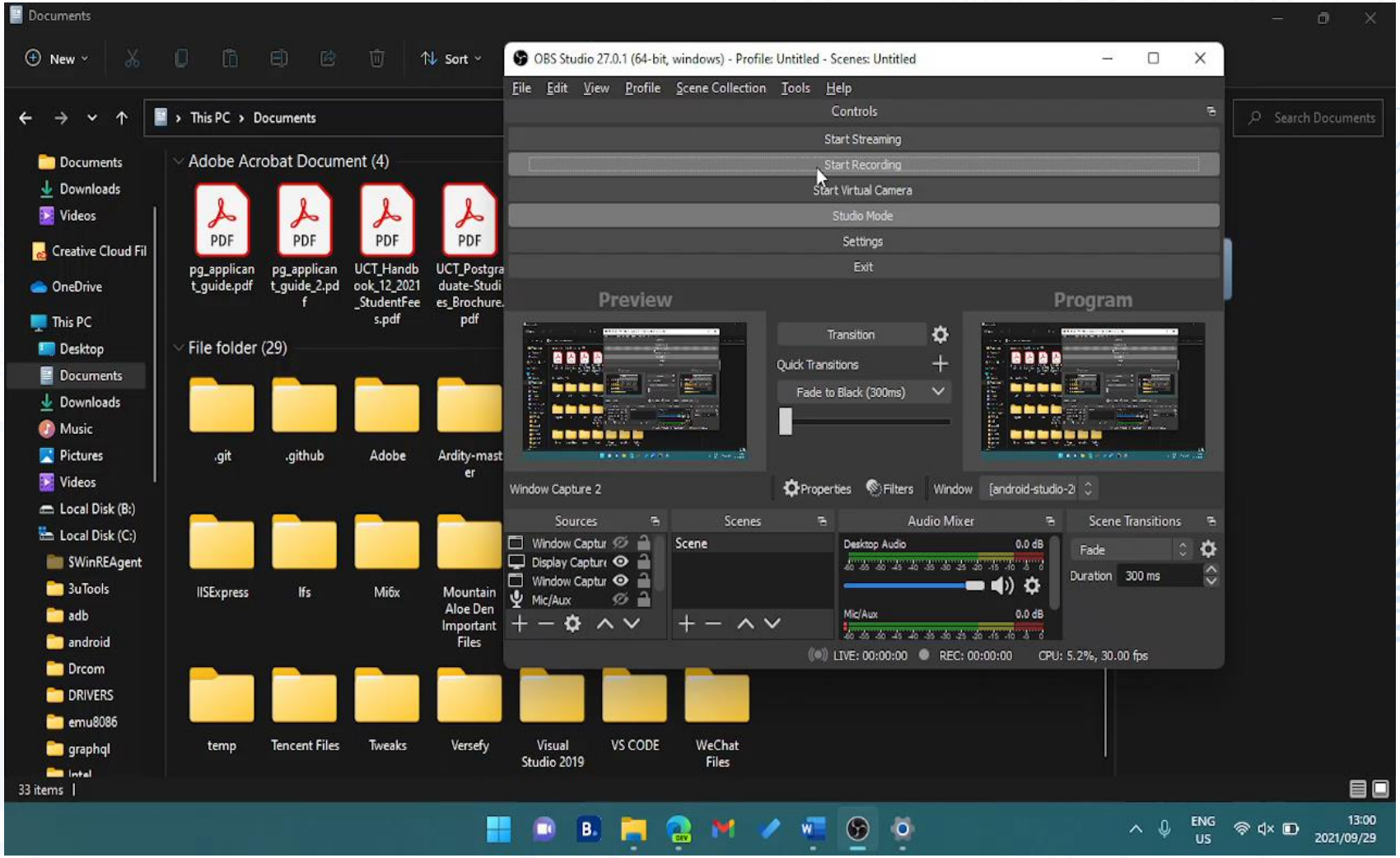








On error in Installation

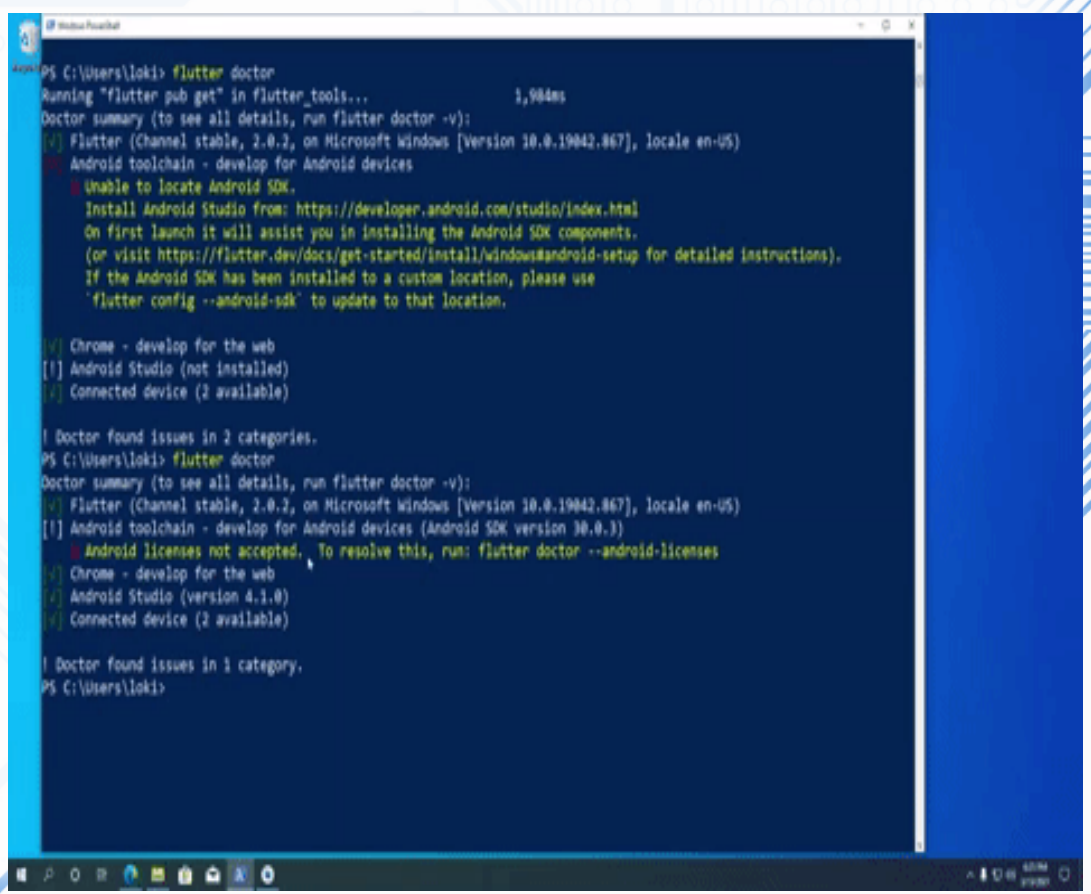


Step 9: **Installation**

Step 9: Open your command prompt or Powershell and type

Flutter doctor --android-licences

Once typed in press enter and accept all the licenses by pressing “y”.

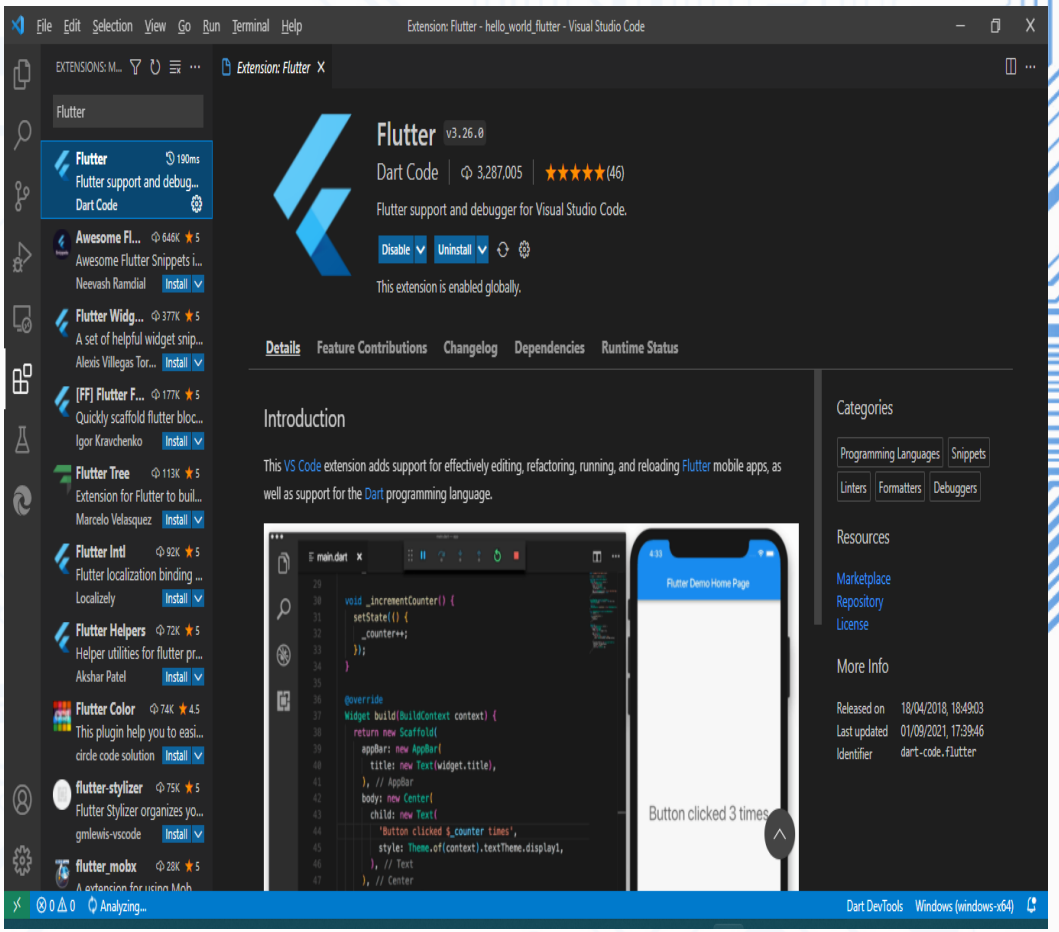


Step 10: **Installation**

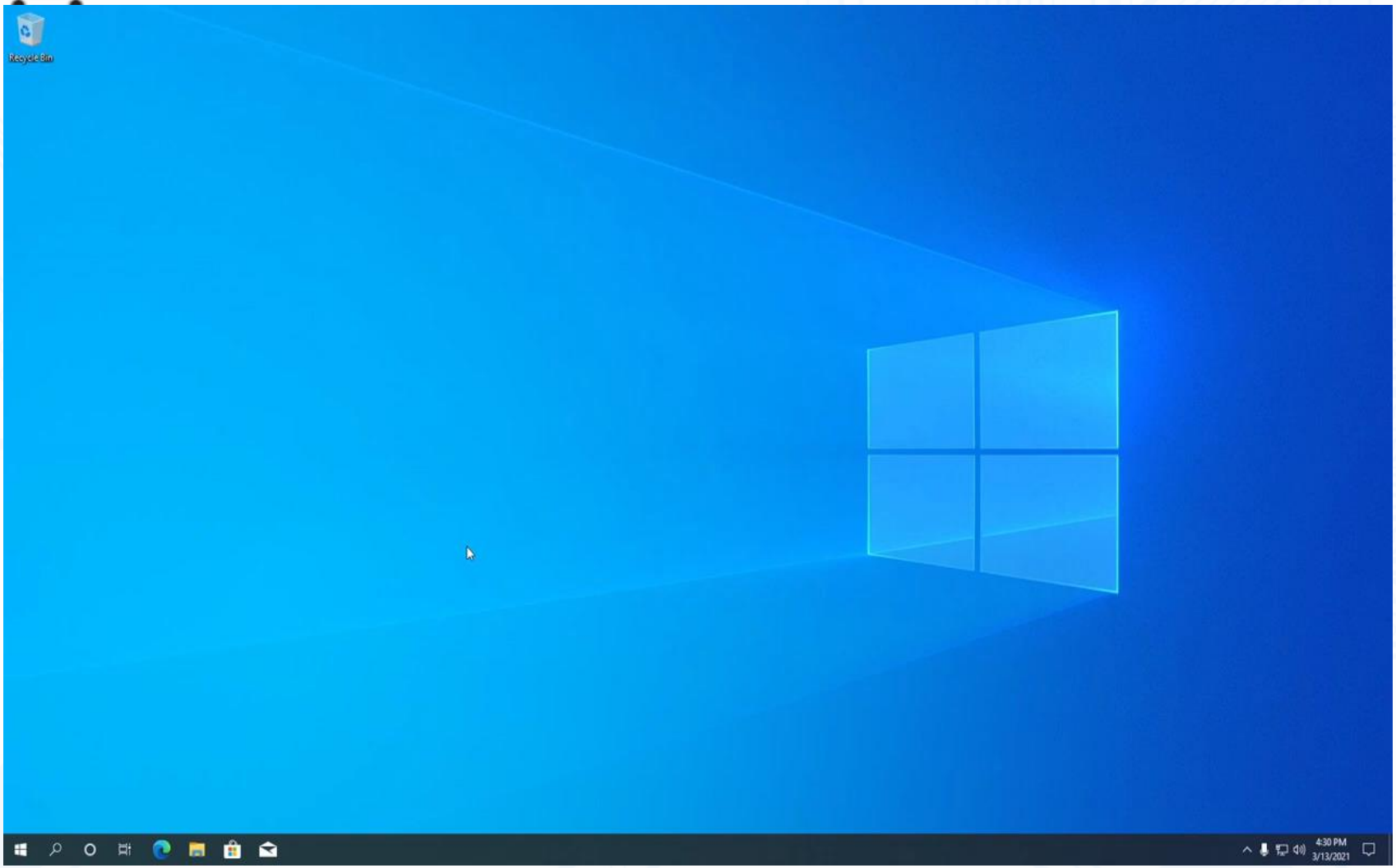
Flutter

Step 10:

Install Flutter extension by navigating to the extensions tab and searching for Flutter this will install both Flutter and Dart programming Language



Let us see ALL installation process



Step 12: **Installation**

Android Emulator

- **Open your project on VSCode.**
- **If you would like to use your phone as a test device then connect it with your USB cable and enable USB Debugging in your developer settings.**

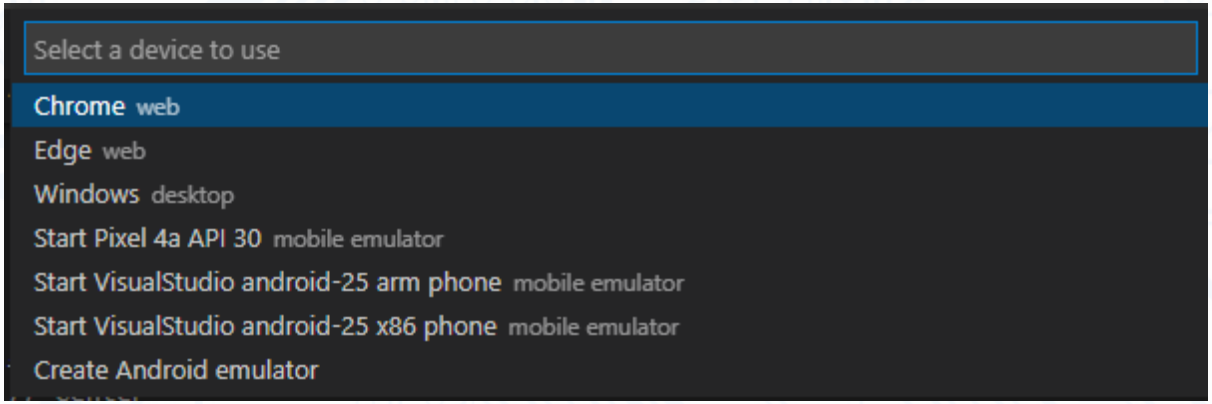
Navigate to the bottom right corner the editor. Select the devices box

Step 12: **Installation**

Flutter

Emulator…..

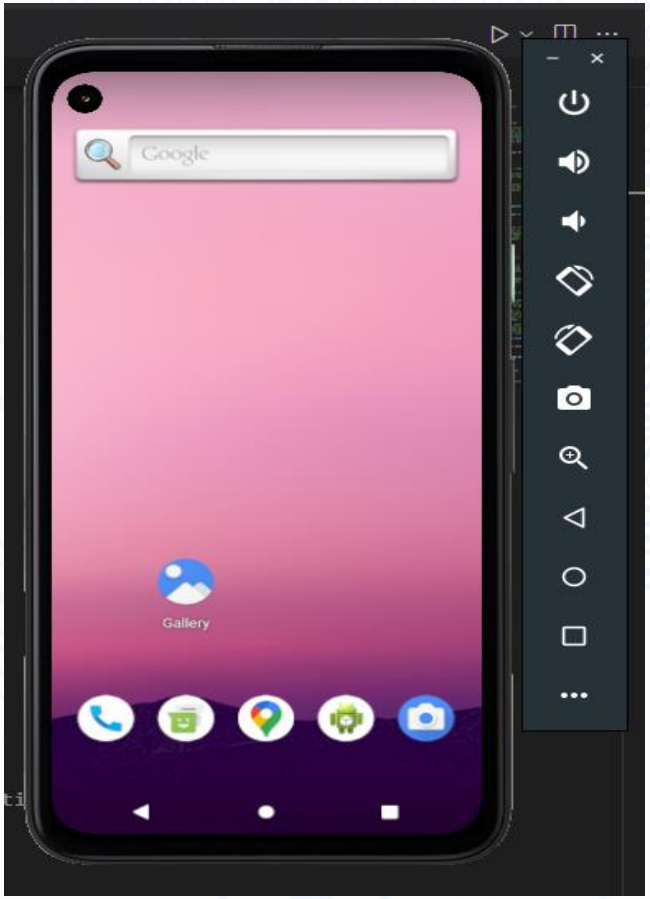
- **Select your desired device or emulator.**



Step 12: **Installation**

Emulator…..

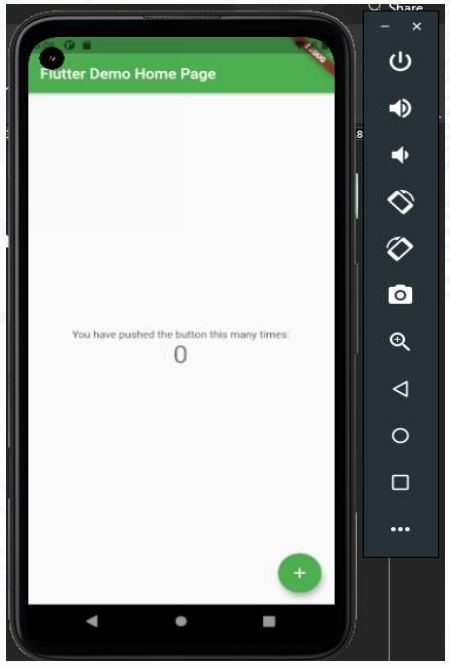
- If an emulator is selected then it should pop up and start to load, loading should take less than 5 minutes, if not try to open it from Android Studio



Step 12: **Installation**

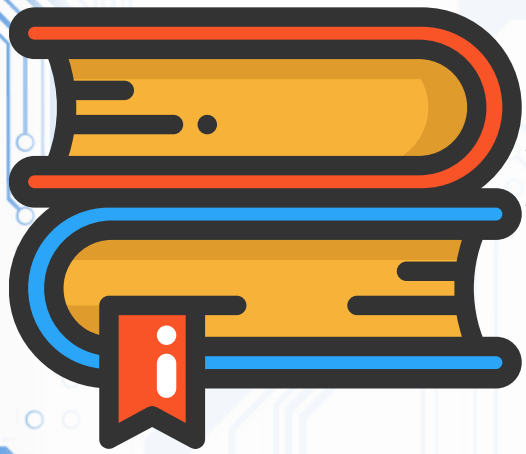
Emulator…..

- To run your project onto the phone press ctrl + F5.
- This will then activate Hot Reload Settings which will be necessary for debugging and development



江西理工大學 信息工程学院

JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY SCHOOL OF INFORMATION ENGINEERING



Mobile application development

移动应用开发

LECTURE 01: Flutter_DART

Dart Programming Language

- **Dart is the programming language used by Flutter.**
- The first hurdle for most would-be Flutter developers is being convinced to learn a new, sparsely used Language: Dart.
  - In order to write Flutter apps, it's important that you also learn Dart.
  - The fundamentals of Dart are similar to all higher-level languages.
  - You'll find familiarity in Dart syntax if you're coming from JavaScript, Java, or any other C-like language.
  - You'll feel comfortable with Dart's object-oriented design if you're coming from Ruby or Python.
  - Dart excels at being a "safe" language to learn. Google didn't set out to create anything innovative with Dart. Google wanted to make a language that was simple and productive and that could be compiled into JavaScript. What Google came up with, it turns out, works well for writing UIs.

Documentation and Dart Packages

- The documentation for Dart and Flutter are vast and clear.
- If you need a quick reference, or additional examples to compliment Flutter by Example, you can find Dart documentation at dart.dev and Flutter documentation at flutter.dev.

Dart's dependency management system

- **Pub** is Dart's dependency management system and library.
- If you're looking for an open-sourced library to solve a problem for any Dart (and Flutter) program, It can be accessed at pub.dev.

The main function

Flutter

- All dart and Flutter programs start with the main function

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

- This is the program which will tell Dart where the program starts and it's found in "main.dart" file.
- Only one main function can exist in a program,

Main Function in Flutter


```
void main() {
// any preprocessing can be done here, such
as determining a device location //
// runApp is a Flutter function that runs
your Flutter app runApp(MyApp());
 }
```

Widgets

Flutter

- *Widgets* are building blocks which you, the developer, will compose together to make a UI. When learning Flutter, you'll hear the phrase "everything is a widget" quite often.
- Every object in your Flutter application's UI is a widget.
- Structure is defined with widgets, styles are defined with widgets, even animations and routing is handled by widgets.
- Widgets are just Dart classes that know how to describe their view.
- The basic widgets are:
  - Text
  - Button
  - Column

Built-In Widgets

Flutter

- Flutter comes with a set of built in widgets which combined can make a beautiful UI. These are:
- Layout
  - Row, Column, Scaffold, Stack
- Structures
  - Button, Toast, MenuDrawer
- Styles
  - TextStyle, Color
- Animations
  - FadeInPhoto, transformations
- Positioning and alignment
  - Center, Padding

Widget types: Stateless

- A StatelessWidget is a widget that you (as the developer) are okay with being destroyed.
- Meant to display information and UI.
- Writing a StatelessWidget requires extending the correct class and including a build method.

```
class TitleText extends StatelessWidget {
final String text;
TitleText(this.text);
@override
Widget build(BuildContext context) {
return Center(
child: Text(text)
);
 }
 }
```

StatefulWidget

Flutter

- **A Stateful Widget is actually two classes: a State object and the widget itself.**
- The purpose of this class is to persist state when Flutter rebuilds widgets.
- The State object is special in that it has several methods that interact with Flutter in different ways. The most important of which is setState.
  - setState is used to tell Flutter that it needs to rebuild, usually because something has changed and the screen needs to reflect that. In actuality, after setState is called, Flutter knows that it needs to call the build method again.
  - The StatefulWidget class must implement the createState method.
  - The State object must implement the build method.
  - The StatefulWidget is immutable.
  - The State object is mutable.

Flutter

```
class Counter extends StatefulWidget {
Counter({Key key, this.title})
 : super(key: key); // Stateful Widgets don't have build methods.
// They have createState() methods.
 // Create State returns a class that extends Flutters State class.
@override _MyHomePageState createState() => new _MyHomePageState(); //
Stateful Widgets are rarely more complicated than this.
 } class _MyHomePageState extends State<MyHomePage> {
int counter = 0; void increaseCount() {
// setState is a special method that tells Flutter to repaint // the
view because state has been updated!
 setState(() { this.counter++; } } // gotta have that build method!
Widget build(context)
 { return new RaisedButton( onPressed: increaseCount, child:
new Text('Tap to Increase'), ); } }
```

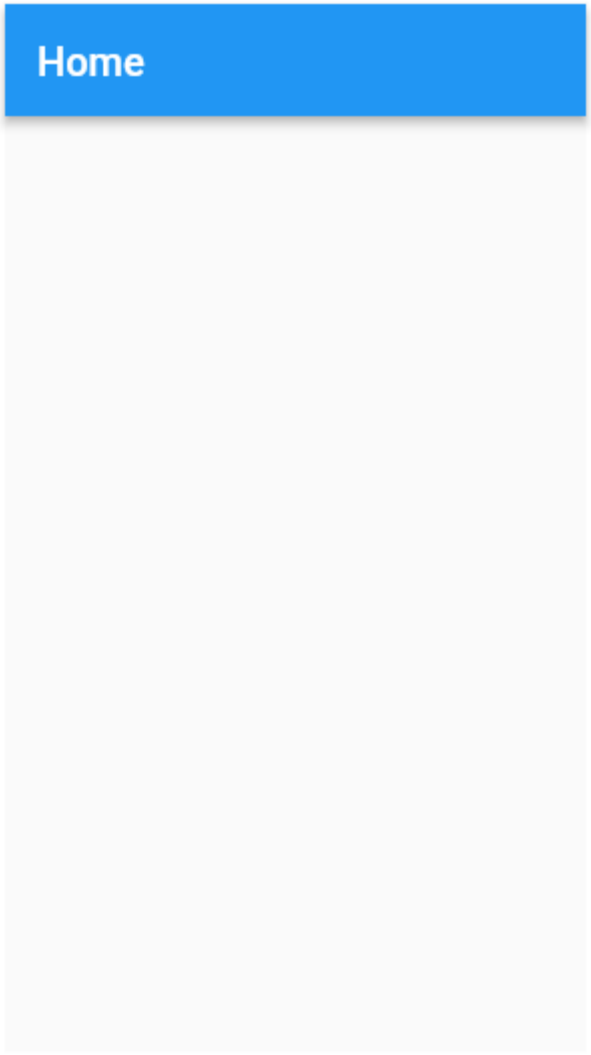
Show this in real environment

Material App Widget

- A convenience widget that wraps a number of widgets that are commonly required for material design applications.
- It builds upon a WidgetsApp by adding material-design specific functionality, such as AnimatedTheme and GridPaper.
- The MaterialApp configures the top-level Navigator to search for routes in the following order:
  - ***For the / route, the home property, if non-null, is used.***
  - ***Otherwise, the routes table is used, if it has an entry for the route.***
  - ***Otherwise, onGenerateRoute is called, if provided. It should return a non-null value for any valid route not handled by home and routes.***
  - ***Finally if all else fails onUnknownRoute is called.***

Flutter

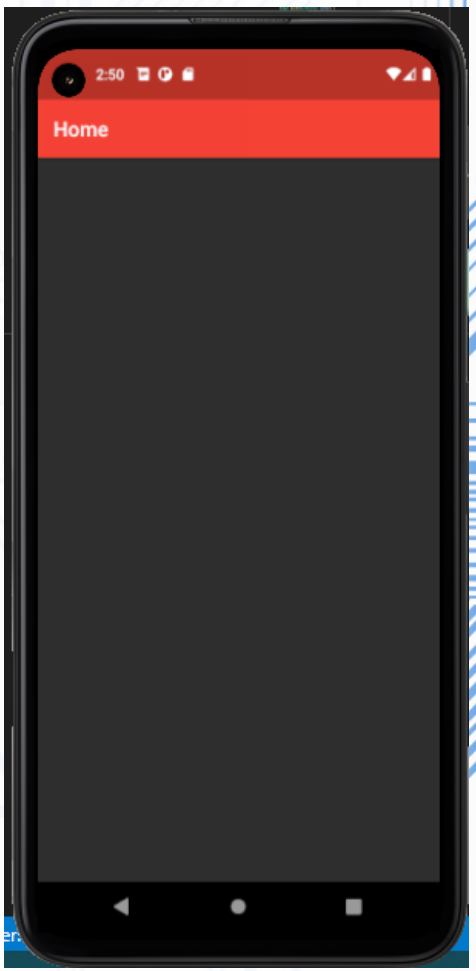
```
MaterialApp(
home: Scaffold(
appBar: AppBar( title: const Text('Home’),), ),
debugShowCheckedModeBanner: false, )
```



Show this in real environment

Theme

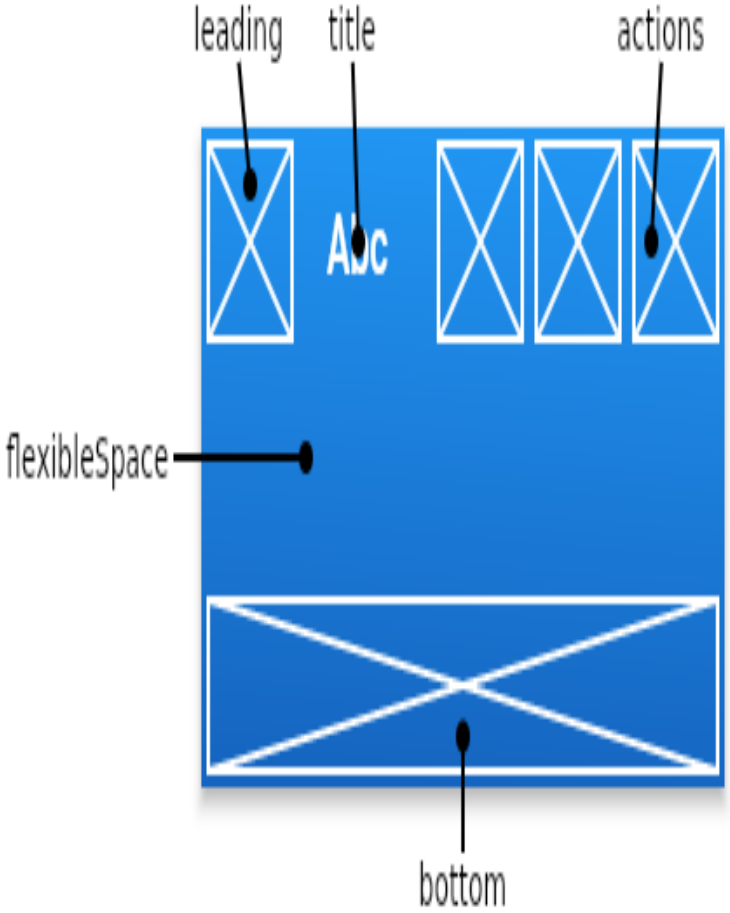
- The theme of the app can be changed by changing the theme property on the Material App Widget.
- You can change the theme of the app to the way to see fit.
- You may need to restart the app in order to see changes




```
MaterialApp(
  debugShowCheckedModeBanner: false,
  theme: ThemeData(brightness: Brightness.dark, primaryColor: Colors.red),
  )); // MaterialApp
```

App Bar Widget

- The AppBar displays the toolbar widgets, leading, title, and actions, above the bottom (if any).
- The bottom is usually used for a TabBar.
- If a flexibleSpace widget is specified then it is stacked behind the toolbar and the bottom widget.
- The following diagram shows where each of these slots appears in the toolbar when the writing language is left-to-right (e.g. English):
- The AppBar insets its content based on the ambient MediaQuery's padding, to avoid system UI intrusions. It's taken care of by Scaffold when used in the Scaffold.appBar property.
- When animating an AppBar, unexpected MediaQuery changes (as is common in Hero animations) may cause the content to suddenly jump. Wrap the AppBar in a MediaQuery widget, and adjust its padding such that the animation is smooth.



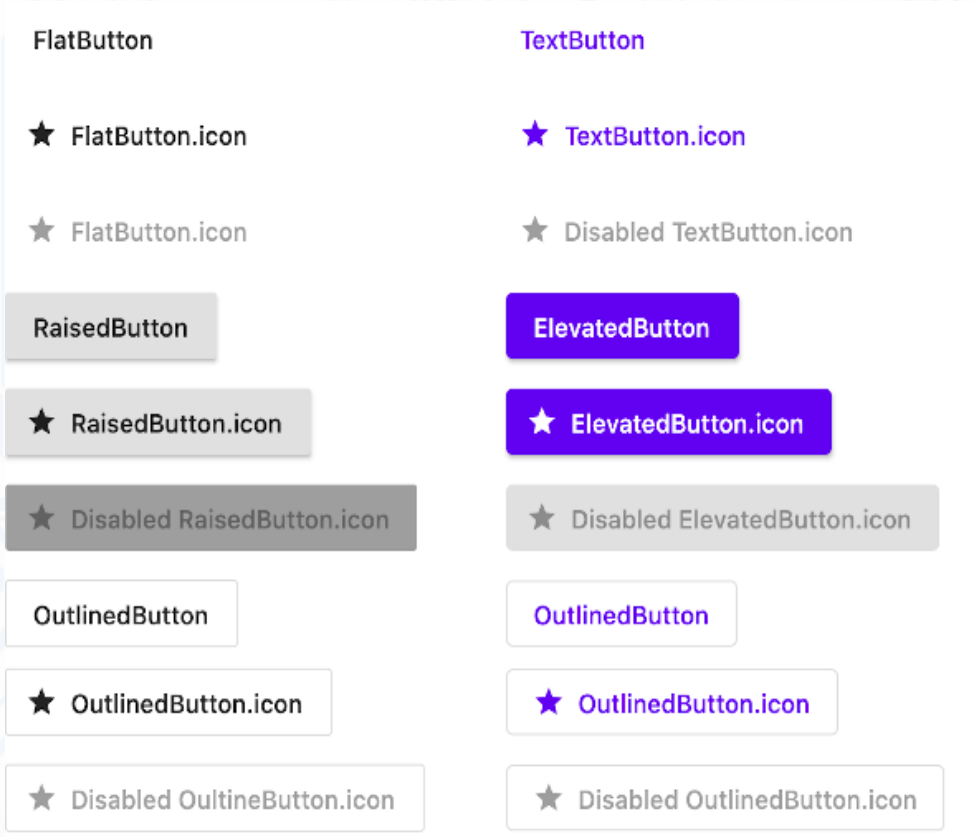
Flutter

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

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

/// This is the main application widget.
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  static const String _title = 'Flutter Code Sample';

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      title: _title,
      home: MyStatelessWidget(),
    );
  }
}

/// This is the stateless widget that the main application instantiates.
class MyStatelessWidget extends StatelessWidget {
  const MyStatelessWidget({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('AppBar Demo'),

      ),
    );
  }
}
```

Show and test this in real environment

Button Widget

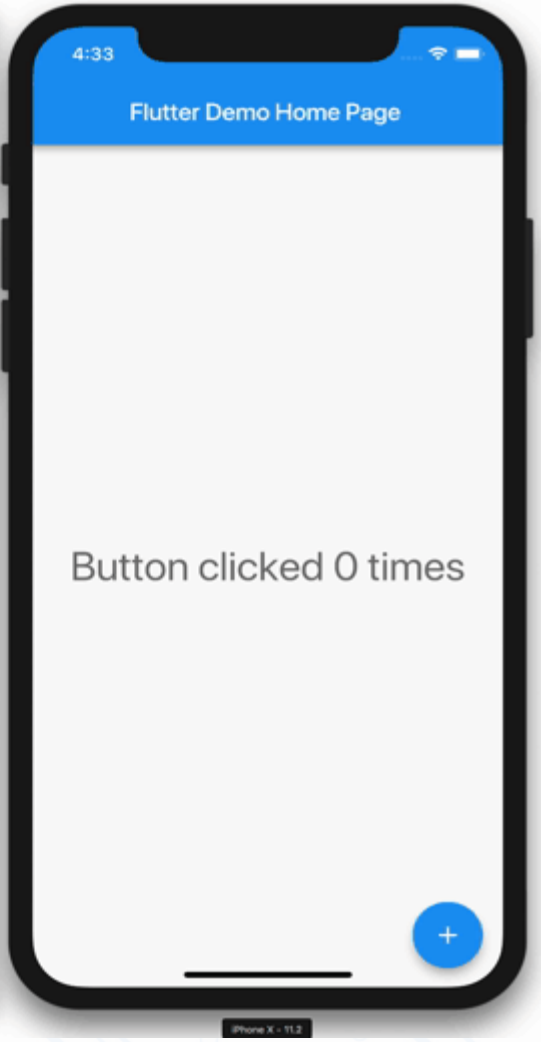
- Flutter has a number of buttons which you can use. These include:
  - Text Button
  - Elevated Button
  - Outlined Button
  - Floating Button
  - Drop Down Button
  - Icon Button
  - PopupMenu Button



Flutter

main.dart

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

@override
Widget build(BuildContext context) {
  return new Scaffold(
    appBar: new AppBar(
      title: new Text(widget.title),
    ), // AppBar
    body: new Center(
      child: new Text(
        'Button clicked $_counter times',
        style: Theme.of(context).textTheme.display1,
      ), // Text
    ), // Center
    floatingActionButton: new FloatingActionButton(
      onPressed: _incrementCounter,
      tooltip: 'Increment',
      child: new Icon(Icons.add),
    ),  // FloatingActionButton
  ); // Scaffold
}
}
```



Show and test this in real environment

Example:Buttons

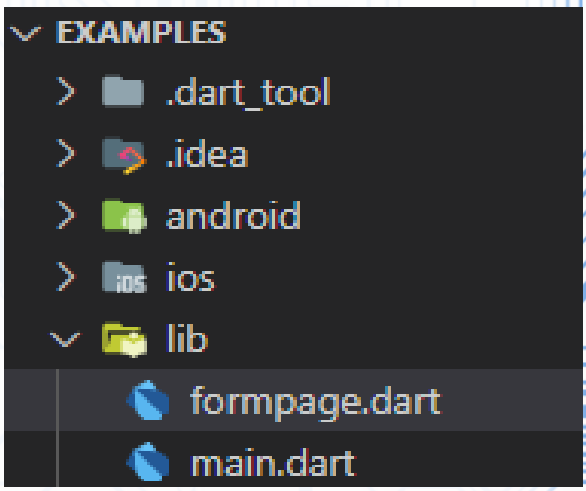
Navigation

Flutter

- Most apps contain several screens for displaying different types of information.
- For example, an app might have a screen that displays products. When the user taps the image of a product, a new screen displays details about the product.
- In Flutter, screens and pages are called routes.
- In our example we will navigate to a new page using a button.

Navigation

- Make a new flutter project and name it examples
- Create a new page called formpage.dart in the lib folder.
- Import the material package.
- Create a stateless widget called named FormPage.
- open main.dart and delete all the widgets.
- In runApp() add the following code:




```
runApp(MaterialApp(
      debugShowCheckedModeBanner: false,
      initialRoute: '/',
      routes: {
        '/': (context) => Home(),
        '/form': (context) =>
      })); // MaterialApp
}
```

Show and test this in real environment

Formpage

- In the form stateless widget add a scaffold and appbar.

```dart
class FormPage extends StatelessWidget {
  const FormPage({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Form"),
      ), // AppBar
    ); // Scaffold
  }
}
```

Show and test this in real environment

Floating Action Send Button

- TO add a bit more flare to our Form, we will create a new Floating Action Button which will later be set to submit the form data
- All we need to do is add the following code to the Scaffold

```
floatingActionButton:
FloatingActionButton(
        onPressed: () {
          Navigator.pop(context);
        },
        child: Icon(Icons.send),
      ),
```

Flutter

- In main.dart create a stateless widget named Home which returns a Scaffold Widget.
- In the scaffold widget add an appbar with the tile “Home”
- Add a body to the scaffold which will have a Center widget with the child of a column widget.
- The children of the column widget will be an elevated button with the child of a Text widget with the text “Form”.

```dart
class Home extends StatelessWidget {
  const Home({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Home"),
      ), // AppBar
      body: Center(
        child: Column(
          children: [ElevatedButton(onPressed: () {}, child: Text("Form"))],
        ), // Column
      ), // Center
    ); // Scaffold
  }
}
```

Show and test this in real environment

OnPressed

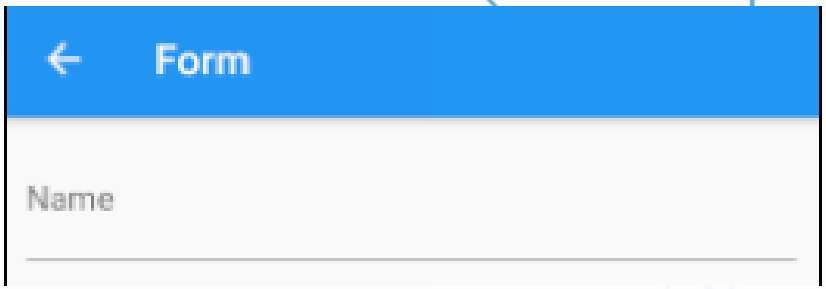
Flutter

- To execute a function when button is pressed, use onPressed() property of the button.
- We will configure the onPressed(){} function in our Elevated Button to open the FormPage().
- We do this by adding a Navigator with the context of current page and we'll push it to the route that we defined in the MaterialApp function.
- To test if it works we'll need to restart the app by pressing the green restart button on the hot reload panel.

```
child: Column(
  children: [
    ElevatedButton(
        onPressed: () {
          Navigator.pushNamed(context, '/form');
        },
        child: Text("Form")) // ElevatedButton
  ],
), // Column
```

Form

- Flutter provides a Form widget to create a form. The form widget acts as a container, which allows us to group and validate the multiple form fields.
- The Form widget acts as a container for grouping and validating multiple form fields.
- When creating the form, provide a GlobalKey. This uniquely identifies the Form, and allows validation of the form in a later step.



- Open the formpage.dart file and underneath the stateless widget create a new stateful widget named FormData.
- Once it's created add final `_formKey = GlobalKey<FormState>();` underneath class `_FormDataState extends State<FormData> {`

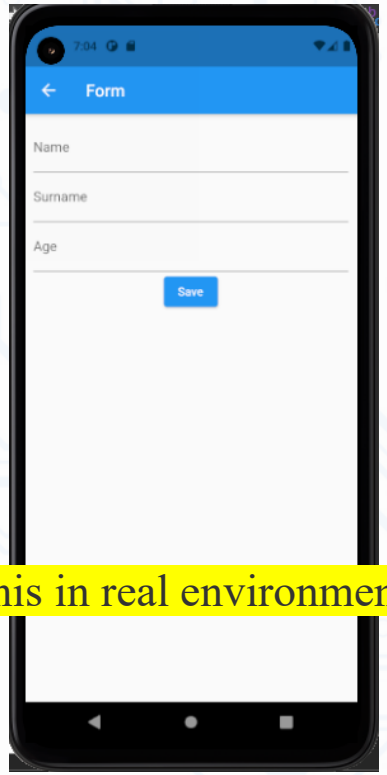
```
class _FormDataState extends State<FormData> {
  final _formKey = GlobalKey<FormState>();
```

- In the stateless widget add a body property to the scaffold and in the body property type the name of the widget you just created "FormData()".
- Back in the FormData() widget delete the Container and replace it with the Form() widget and add a key property which will be `_formKey.`
- Inside the form widget's child property there should be a Padding() widget with a padding property which has const EdgeInsets.all(10.0).
- Inside the Padding widget's child property there should be a Column() widget.
- In the column widget's children property add a TextFormField() widget.
- In the TextFormField type `(decoration: InputDecoration(labelText: 'Name')`

Flutter

```dart
class FormData extends StatefulWidget {
  const FormData({Key? key}) : super(key: key);

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

class _FormDataState extends State<FormData> {
  final _formKey = GlobalKey<FormState>();
  @override
  Widget build(BuildContext context) {
    return Form(
      key: _formKey,
      child: Padding(
        padding: const EdgeInsets.all(10.0),
        child: Column(children: [
          TextFormField(decoration: InputDecoration(labelText: 'Name')),
          TextFormField(decoration: InputDecoration(labelText: 'Surname')),
          TextFormField(decoration: InputDecoration(labelText: 'Age')),
          ElevatedButton(
            child: Text('Save'),
            onPressed: () {},
          ), // ElevatedButton
        ]), // Column
      ), // Padding
    ); // Form
  }
}
```

- Duplicate the Name Text form field to make more Fields.
- Add an Elevated Button with the child Text("Save").
- The code for the FormData Widget should look like this:



Show and test this in real environment

Form With Validation

- Apps often require users to enter information into a text field. For example, you might require users to log in with an email address and password combination.
- To make apps secure and easy to use, check whether the information the user has provided is valid. If the user has correctly filled out the form, process the information. If the user submits incorrect information, display a friendly error message letting them know what went wrong.

Validator Property

- The validator property is used to check validation for users.
- To add validation to your form start by adding the validator property to TextFormField() with the following code:

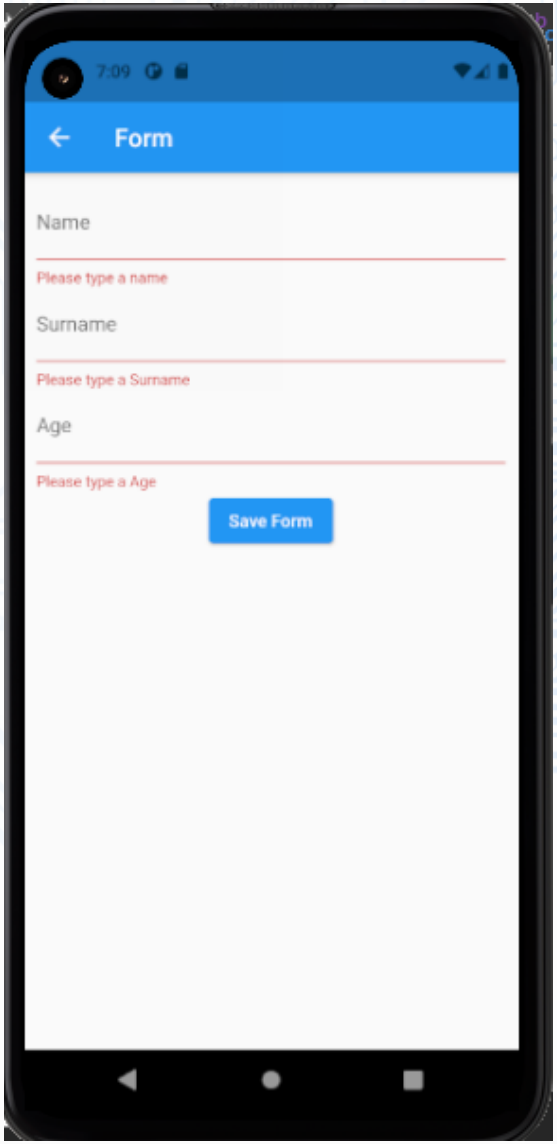
```
validator: (value) {
  if (value != null && value.isEmpty) {
    return 'Please type a name';
  }
  return null;
},
```

- Repeat for the other text fields

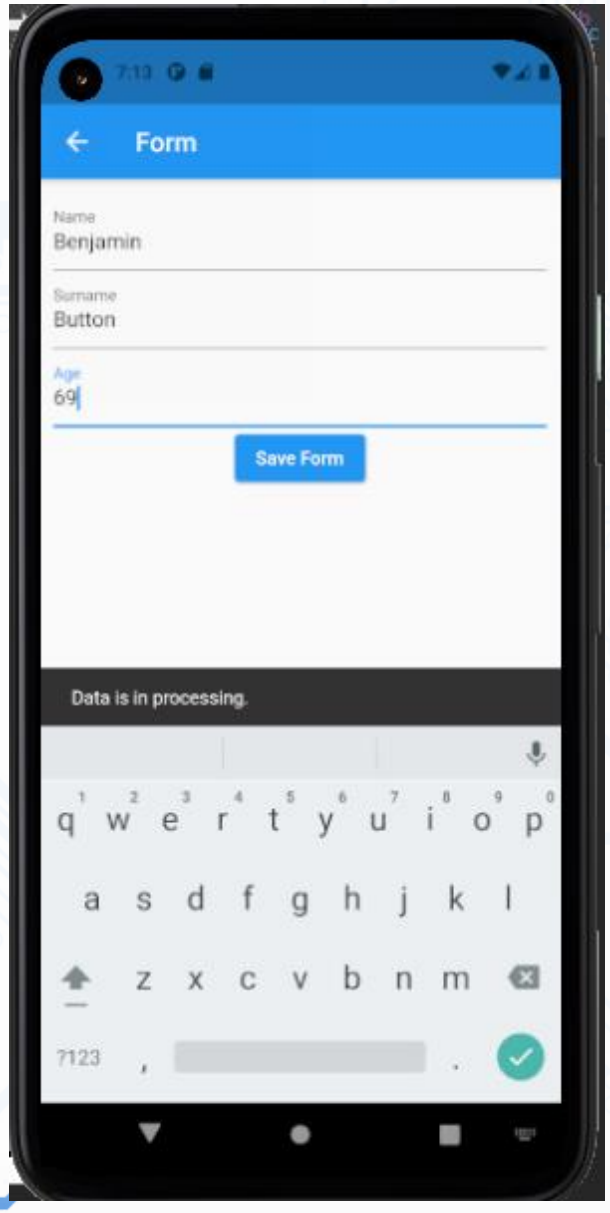
Flutter

- In the ElevatedButton's onPressed property add the following code and your app should validate each time you press the Save button and if validated successfully a snackbar will pop up.

```
if (_formKey.currentState!.validate()) {
                    // If the form is
valid, display a Snackbar.

ScaffoldMessenger.of(context).showSnackBa
r(
                        SnackBar(content:
Text('Data is in processing.')));
```

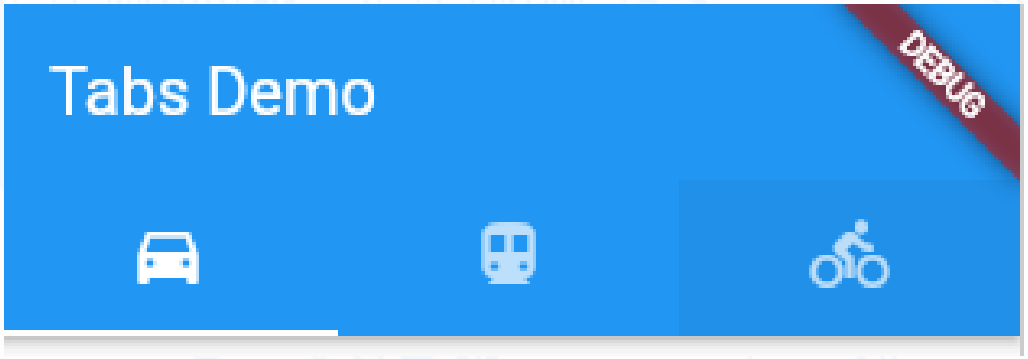


Snackbar


Form
Name
Benjamin
Surname
Button
Age
69
Save Form
Data is in processing.

Appbar Tabs

- Working with tabs is a common pattern in apps that follow the Material Design guidelines. Flutter includes a convenient way to create tab layouts as part of the material library.



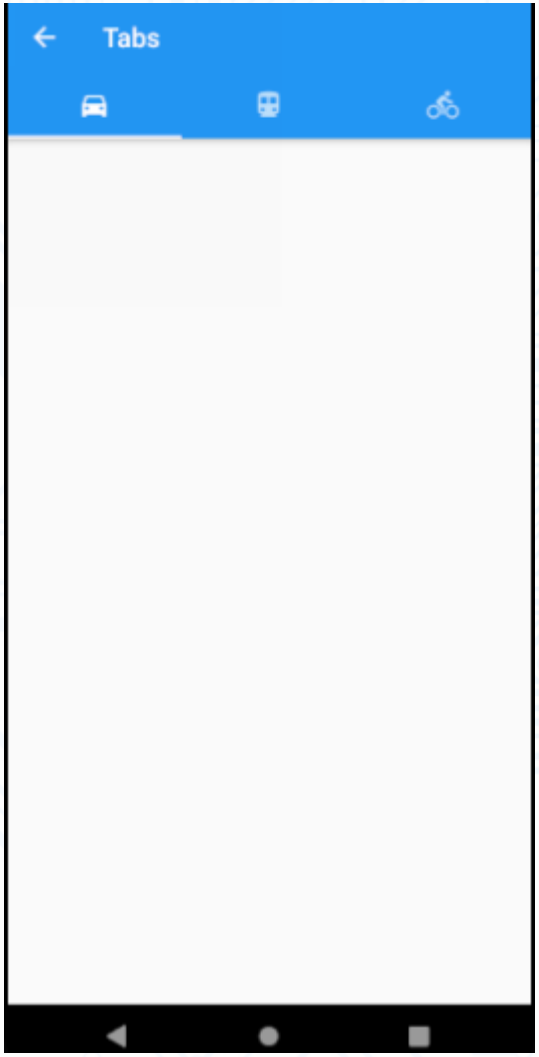
Let’s make Tabs

- To make an App Bar tabs we’ll need create a new file named tabpage.dart in the lib folder.
- Create a stateless widget named TabPage.
- To add a new button and route to our main.dart page. We do this by adding the following code to the routes list in Material App :
- `'/tab': (context) => TabPage(),`
- Then simply duplicate the code for the previous Elevated button:
- `ElevatedButton(`
- `onPressed: () {`
  `Navigator.pushNamed(context, '/tab');`
  `},`
- `child: Text("App Bar Tabs"))`

Tabpage

- Replace the Container widget with the following code:

```
return DefaultTabController(
      length: 3,
      child: Scaffold(
        appBar: AppBar(
          title: Text("Tabs"),
          bottom: TabBar(
            tabs: [
              Tab(icon:
Icon(Icons.directions_car)),
              Tab(icon:
Icon(Icons.directions_transit)),
              Tab(icon:
Icon(Icons.directions_bike)),
            ],
          ),
        ),
      ),
    );
```



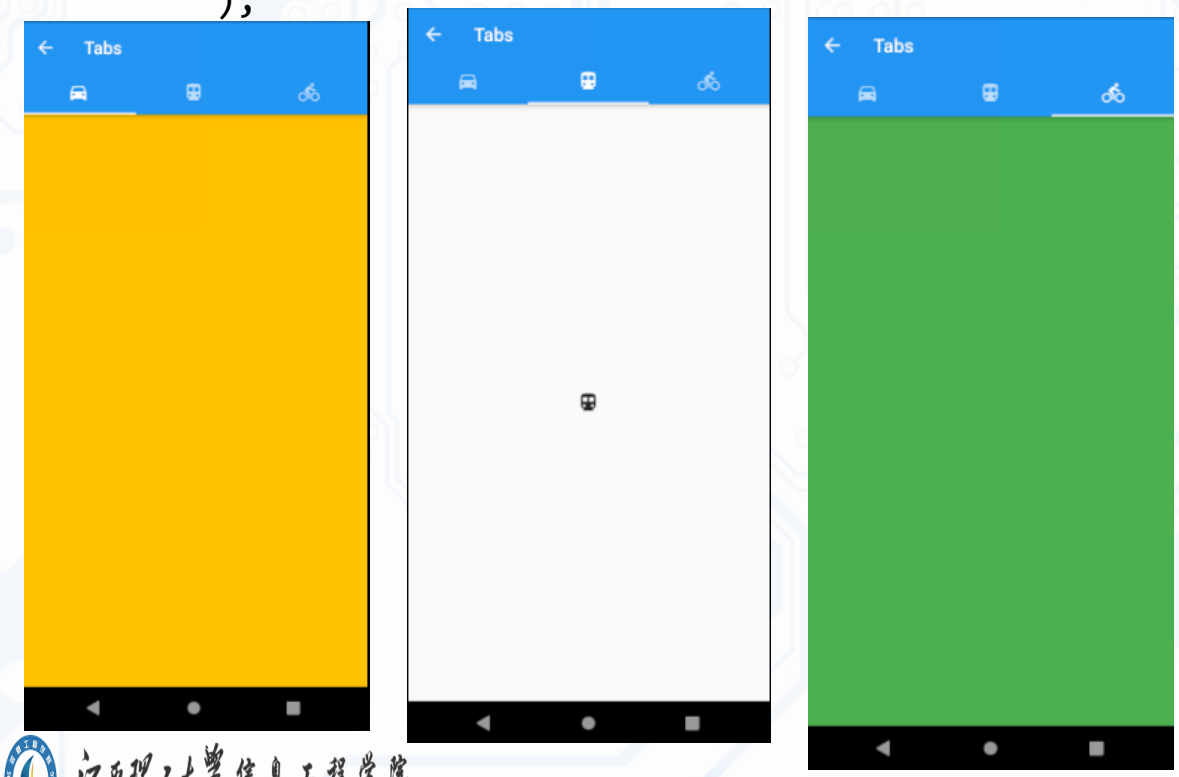
Content for each tab

- Content can be added for each tab using the TabView using the body property in Scaffold

```
body: const TabBarView(
            children: [

Icon(Icons.directions_car),

Icon(Icons.directions_transit)
,

Icon(Icons.directions_bike),
            ],
```

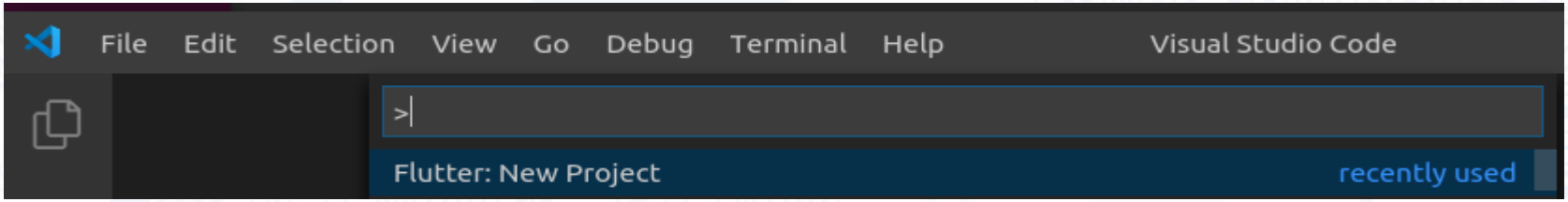
Different content Tabs

```
TabBarView(
          children: [
            Container(color: Colors.amber),
            Icon(Icons.directions_transit),
            Container(color: Colors.green),
          ],
        ),
```



Creating a Flutter App

- You can create a new Flutter application in app:
- Open VSCode and invoke the command palette via View > Command Palette or ctrl + shift + p and type flutter to see the new project option



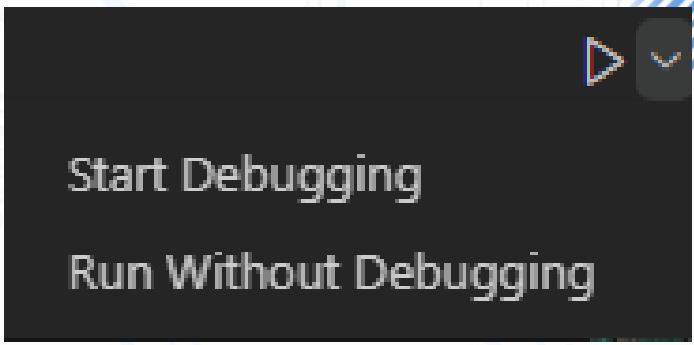
- Select a folder to save it in and name the project appropriately.
- Or by using the terminal
  - Type Flutter create mynewapp
  - Then open the new app folder in VSCode

```
\Desktop\Flutter_Projects\examples> flutter create newflutterapp
```

Running a Flutter App

- There are a couple of ways to start running a flutter app you could run it by:
- Using the terminal
  - Flutter start

- Using keyboard Shortcuts
  - Ctrl + F5
- Using UI
  - Pressing the play icon on the top left corner of Visual Studio Code.
  - To start without debugging press the drop down arrow.



Debug Toolkit

- You can press the lightning icon to invoke Hot Reload manually or you could press ctrl +s to save your work and invoke hot reload.
- The green button is to restart the app.
- The red square is meant to stop the app.
- The blue arrow icons are meant for debugging and skipping over debug points.
- The debug toolkit pops up if the app is run by pressing ctrl + F5 or by pressing the UI start button.

Properties

Flutter

- Flutter widgets come with properties.
- These properties essentially define what a widget is able to do.
- For instance a container widget has alignment child, clipbehaviour and colour properties amongst others. These properties can be changed to edit the appearance or position of the container.
- Visual Studio gives more information about a widget and it's properties when you hover over it.

```
Container(
```

```
(new) Container Container({Key? key, AlignmentGeometry?
alignment, EdgeInsetsGeometry? padding, Color? color,
Decoration? decoration, Decoration? foregroundDecoration,
double? width, double? height, BoxConstraints? constraints,
EdgeInsetsGeometry? margin, Matrix4? transform,
AlignmentGeometry? transformAlignment, Widget? child, Clip
clipBehavior = Clip.none})
```

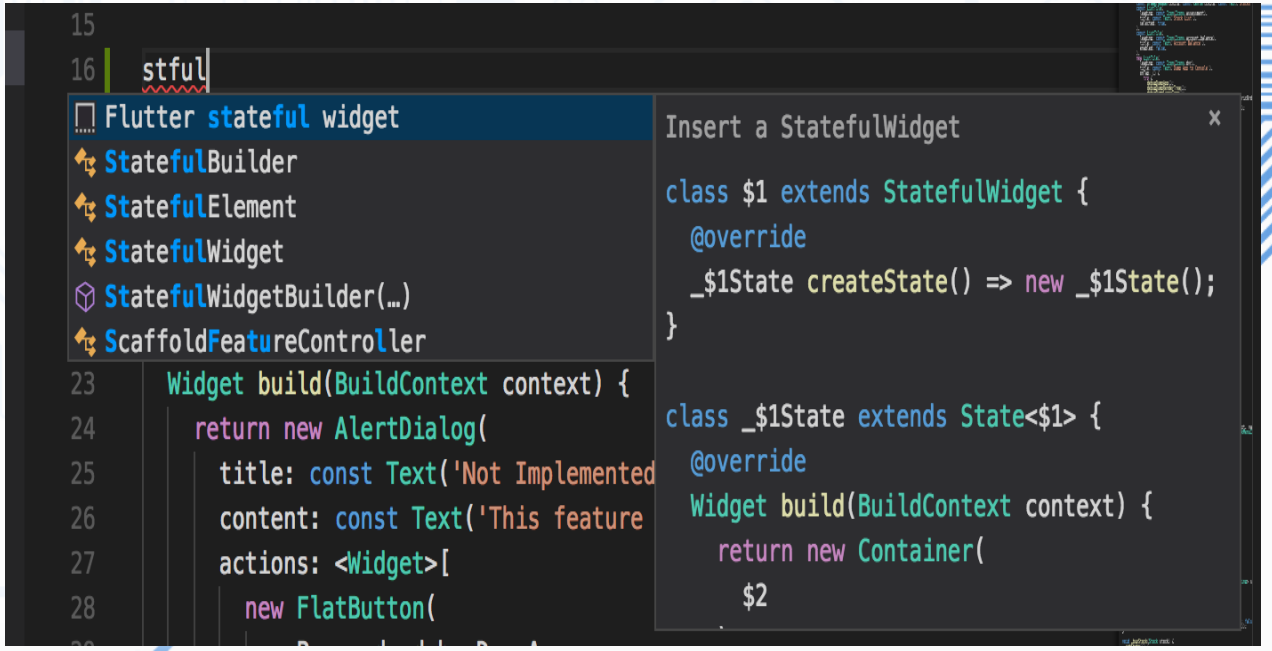
package:flutter/src/widgets/container.dart

Creates a widget that combines common painting, positioning, and sizing widgets.

The `height` and `width` values include the padding.

Snippets

- Snippets can be used to speed up entering typical code structures. They are built in but more can be downloaded in the extension library.
- They are invoked by typing their prefix, and then selecting from the code completion window:



Pubspec File

- Every Flutter project includes a pubspec.yaml file, often referred to as the pubspec. A basic pubspec is generated when you create a new Flutter project. It’s located at the top of the project tree and contains metadata about the project that the Dart and Flutter tooling needs to know. The pubspec is written in YAML, which is human readable, but be aware that white space (tabs v spaces) matters.

- The pubspec file specifies dependencies that the project requires, such as particular packages (and their versions), fonts, or image files. It also specifies other requirements, such as dependencies on developer packages (like testing or mocking packages), or particular constraints on the version of the Flutter SDK.

- Fields common to both Dart and Flutter projects are described in the pubspec file on dart.dev. This page lists Flutter-specific fields that are only valid for a Flutter project.

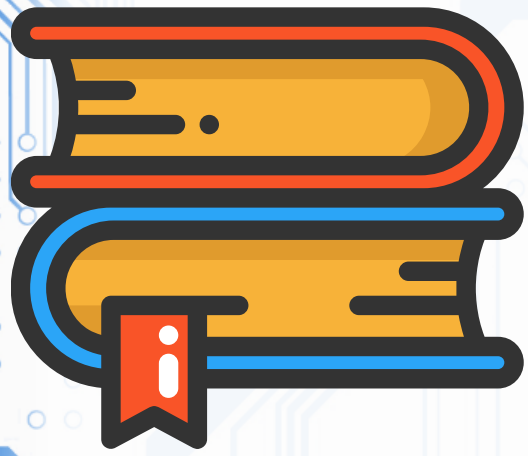

```
name: <project name>
description: A new Flutter project.
publish_to: 'none'

version: 1.0.0+1
environment:
  sdk: ">=2.7.0 <3.0.0"
dependencies:
  flutter:       # Required for every Flutter project
    sdk: flutter # Required for every Flutter project
  cupertino_icons: ^1.0.2 # Only required if you use Cupertino (iOS style) icons
dev_dependencies:
  flutter_test:
    sdk: flutter # Required for a Flutter project that includes tests

flutter:
  uses-material-design: true # Required if you use the Material icon font
  assets:  # Lists assets, such as image files
    - images/a_dot_burr.jpeg
    - images/a_dot_ham.jpeg

  fonts:             # Required if your app uses custom fonts
    - family: Schyler
      fonts:
        - asset: fonts/Schyler-Regular.ttf
        - asset: fonts/Schyler-Italic.ttf
          style: italic
    - family: Trajan Pro
      fonts:
        - asset: fonts/TrajanPro.ttf
        - asset: fonts/TrajanPro_Bold.ttf
          weight: 700
```

Assets

- Common types of assets include static data (for example, JSON files), configuration files, icons, and images (JPEG, WebP, GIF, animated WebP/GIF, PNG, BMP, and WBMP).
- Before you can use an asset it should be listed inthe pubspec.yaml file.
- Besides listing the images that are included in the app package, an image asset can also refer to one or more resolution-specific “variants”.

江西理工大學 信息工程学院

JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY SCHOOL OF INFORMATION ENGINEERING



5 concepts every Flutter dev should know

江西理工大學 信息工程学院

JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY SCHOOL OF INFORMATION ENGINEERING

5 concepts every Flutter dev should know

1. State management architecture

- This is one of the most important topics in the flutter community, it is quite important if you want to maintain a mid or large-size flutter project. It will help to create a project smoothly and add new features flawlessly.

1. State management architecture

- What do we need from architecture?

1. **Simplicity to understand:-** I prefer architecture that is quite simple to understand and has a very low learning curve, so any new dev can grasp it quickly and start doing what matters.
2. **Total separation of business logic:-** In flutter as we know, UI and business logic got mixed up very frequently and this can later cause problems and it becomes quite cumbersome to maintain this type of project.
3. **Effortless testing:-** Testing the codebase is as important as code, we need an architecture that lets us do straightforward testing.

2. Testing

Flutter

- This is a single topic that I did not understand why it is important earlier in my career, but as I go ahead in my career and got experience with multiple projects and issues that came in a production environment.
- I have realized in a hard way, why this is as important.

3. IDE Shortcuts

- IDE shortcuts really? this is one of the most important thing that I need to learn, you must be thinking about it. Let me explain why, we're devs we spend most of our time writing code and debugging, which of course include a keyboard and we want to use it as efficiently as possible. Studies calculate that using **keyboard shortcuts** allows working 10 times faster than working with the mouse.
- **Benefits:-**
  - It saves you a ton of time.
  - It makes us productive and doesn't let us lose focus.
  - And of course, it makes you look cool (in movies have you noticed actors never use mouse)

4. Platform channel

- In mobile development even though we are using a cross-platform framework, we need to interact with multiple SDKs and libraries which are only available for the native platforms.
- Platform channel gives you the superpower to interact with native SDK. And I want you to have this superpower, if you are a beginner flutter dev almost in all interviews you will be asked if you've worked with native SDK using platform channel or not.

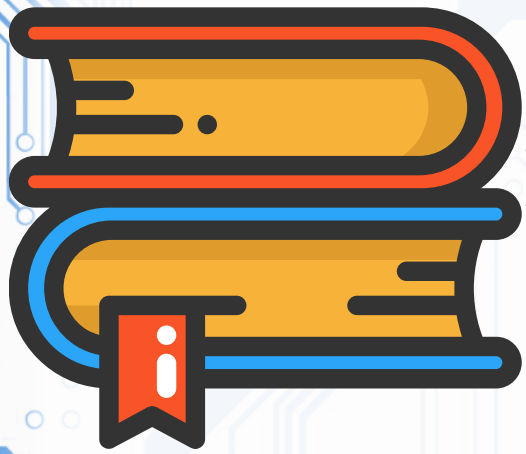
5. Maintaining a project

Flutter

- You have worked hard and made a full-fledged application, You published it in-store and there you go, you start to get feedback and issues from users that need to be solved or a feature that needs to be in your app these are common things that will come upon daily basis while you're maintaining a project.
- suggestions for better app maintenance:-
  - Always write tests, it'll save you from issues and headaches in long run.
  - Focus on code coverage, with the coverage report you'll know which code your tests are specifically covering and which part is not covered and cause issues later on.
  - We reiterating here, choose a good architecture for your project and follow it.
  - Write strings, color codes, styles on separate files as constant and reuse them.
  - There are my top five suggestions for anyone who is into flutter development.

江西理工大學 信息工程学院

JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY SCHOOL OF INFORMATION ENGINEERING



Mobile application development

移动应用开发

LECTURE 01: Flutter Examples

江西理工大學 信息工程学院

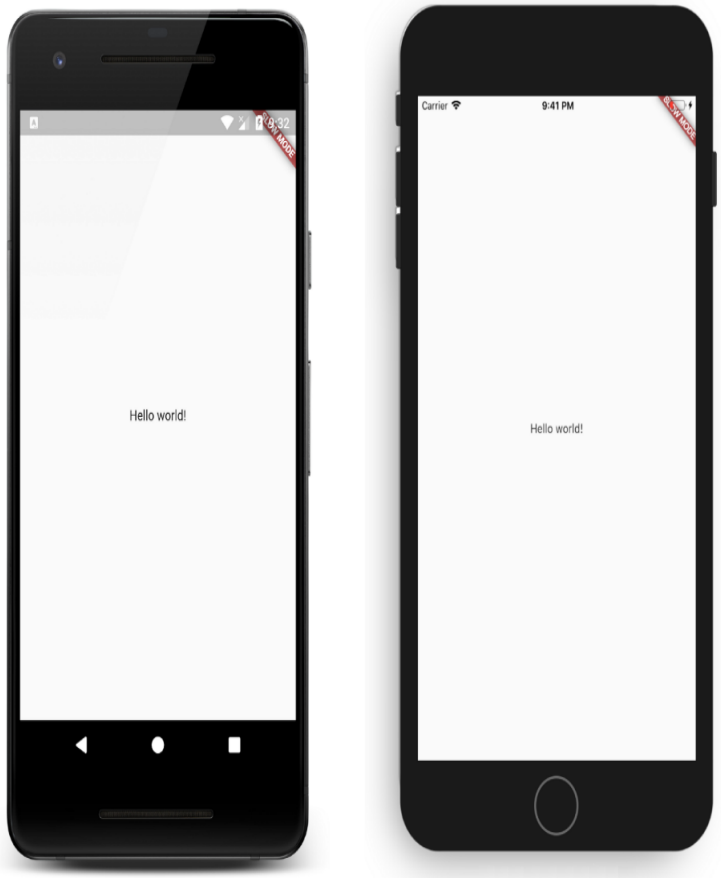
JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY SCHOOL OF INFORMATION ENGINEERING

Example 1: Hello World

Flutter

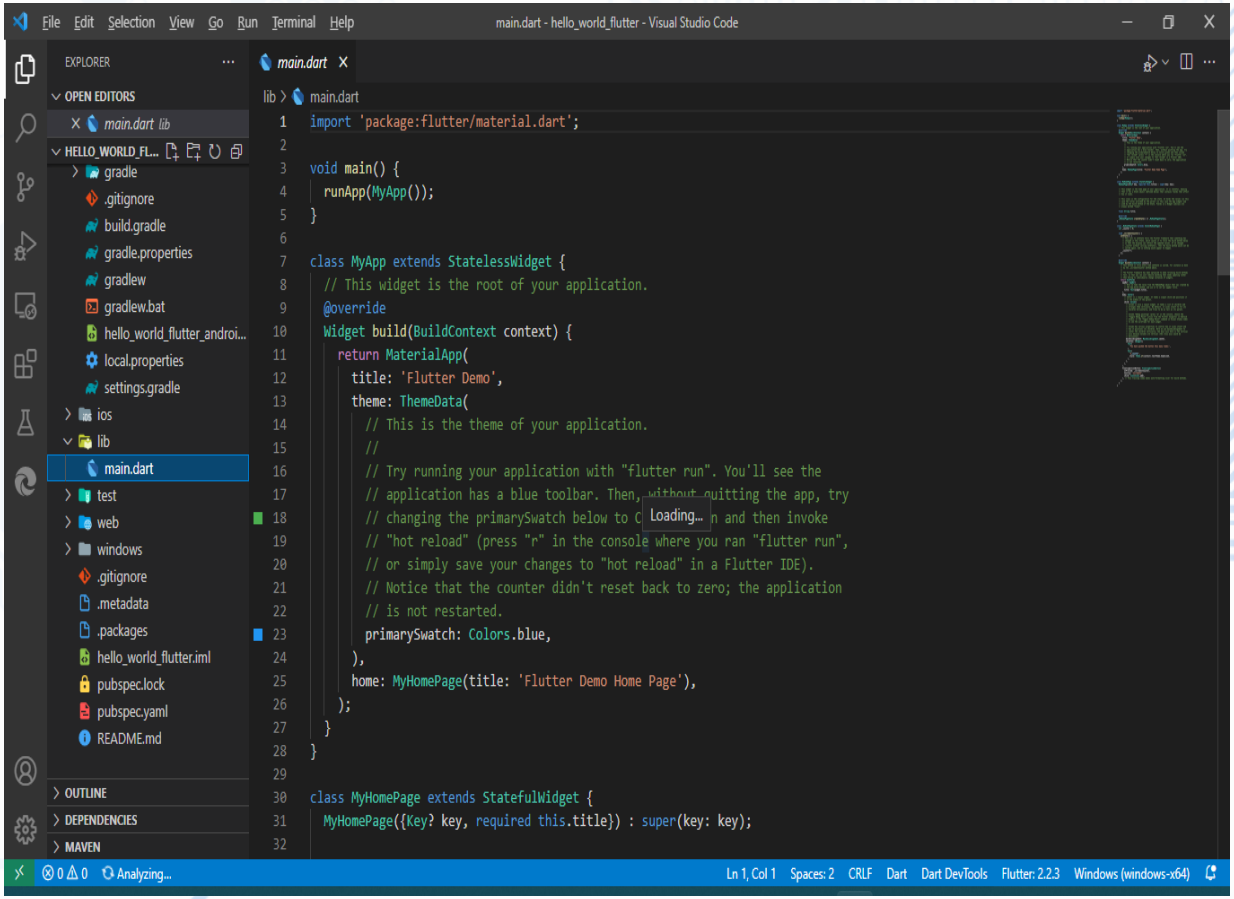
- Create a new folder named Flutter_Projects
- Open your preferred Terminal and navigate to Flutter_Projects and type the following command: flutter create hello_world_flutter

```
C:\Users\Khaya\Desktop\Flutter_Projects>flutter create hello_world_flutter
Creating project hello_world_flutter...
```



Example 1: Hello World

- **Open the project up in your IDE and navigate to lib\main.dart**



Example 1: Hello World

Flutter

Delete the code in main.dart and enter this code.

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: const Text('Welcome to Flutter'),
        ),
        body: const Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

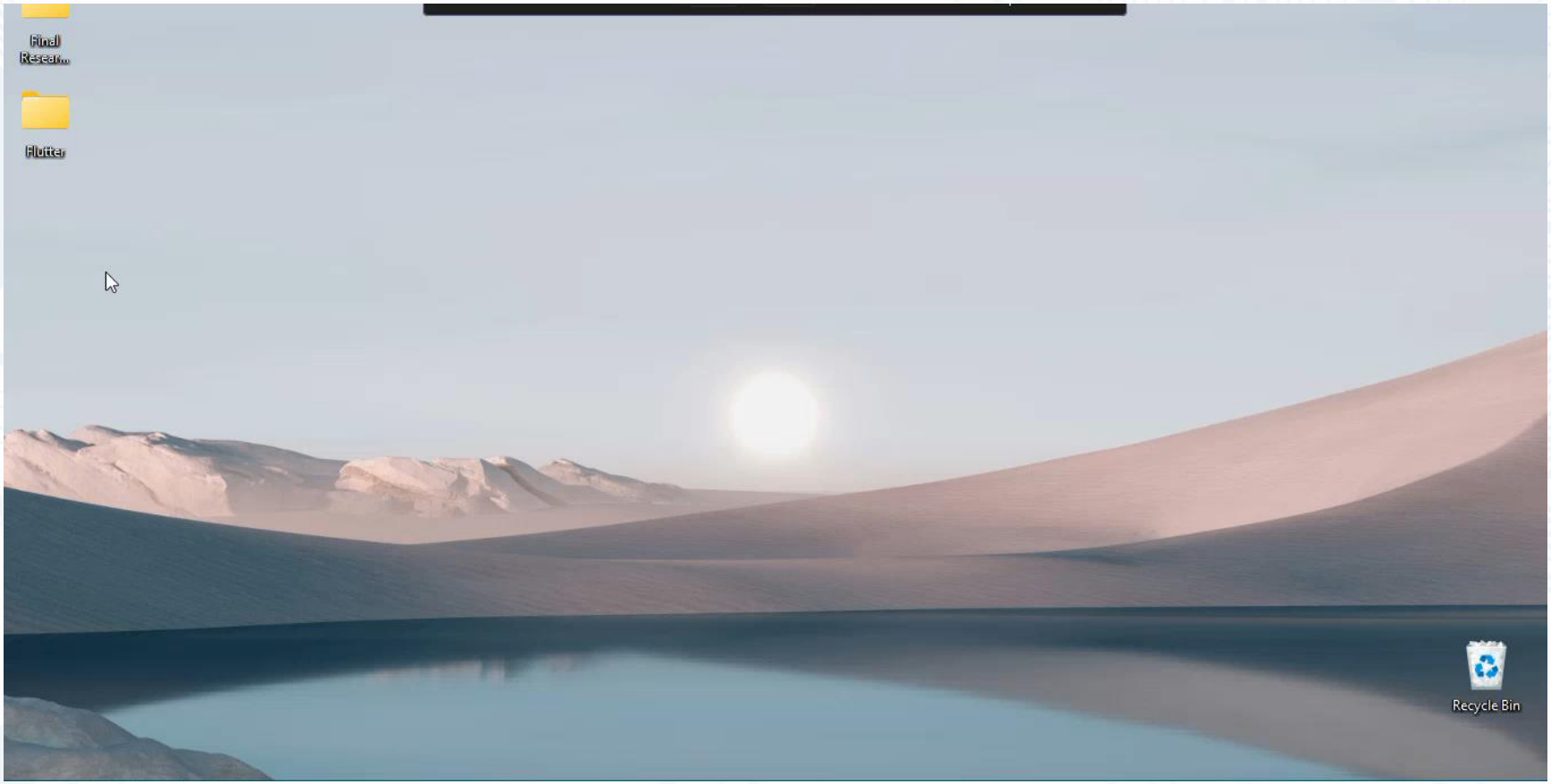
Example 1: Hello World

Flutter

Run the app by selecting your device and the pressing ctrl F5

Example 1: Hello World

Flutter

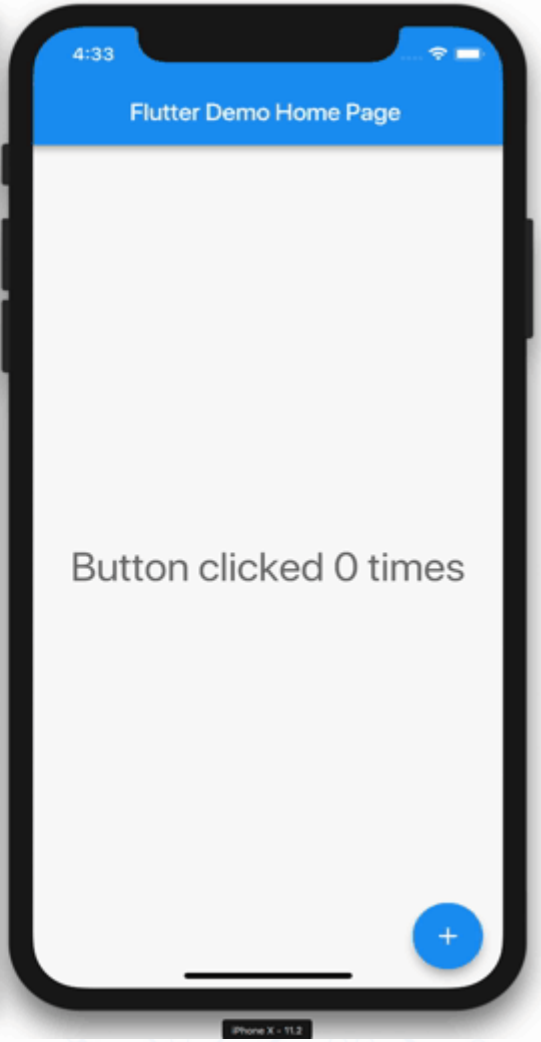


Example 2: Button

Flutter

main.dart

```
29
30    void _incrementCounter() {
31      setState(() {
32        _counter++;
33      });
34    }
35
36    @override
37    Widget build(BuildContext context) {
38      return new Scaffold(
39        appBar: new AppBar(
40          title: new Text(widget.title),
41        ), // AppBar
42        body: new Center(
43          child: new Text(
44            'Button clicked $_counter times',
45            style: Theme.of(context).textTheme.display1,
46          ), // Text
47        ), // Center
48        floatingActionButton: new FloatingActionButton(
49          onPressed: _incrementCounter,
50          tooltip: 'Increment',
51          child: new Icon(Icons.add),
52        ),  // FloatingActionButton
53      ); // Scaffold
54    }
55  }
56
```



Where to Start Learning Flutter?

1. https://flutter.dev/
2. learning Flutter? https://hackr.io/tutorials/learn-flutter
3. https://hackr.io/tutorials/learn-flutter
4. https://flutter.dev/
5. learning Flutter? https://hackr.io/tutorials/learn-flutter
6. Take a look at the following tutorials: https://hackr.io/tutorials/learn-flutter
7. **FragmentedCast**
8. Tutorial Links:
9. http://fragmentedpodcast.com/episodes/118/
10. http://fragmentedpodcast.com/episodes/119/
11. https://hackr.io/tutorials/learn-flutter

12. **Flutter Docs**
13. Tutorial Links:
    1. •Flutter for Android devs
    2. • Flutter for iOS devs
    3. • Flutter for React Native devs
    4. • Flutter for Web devs https://hackr.io/tutorials/learn-flutter
14. **Google odelabs:** https://hackr.io/tutorials/learn-flutter
15. **Github repository:** https://hackr.io/tutorials/learn-flutter
16. **Udacity Course :** https://hackr.io/tutorials/learn-flutter
17. **MTechViral: YouTube :** https://hackr.io/tutorials/learn-flutter

Flutter

- **You just saw the top 8 Flutter Tutorials/Programming**

Community and courses but do you know which of the above ones is “the best”? No!
https://hackr.io/tutorials/learn-flutter

- **Hackr.io is the solution. Presenting you the best**

programming courses and tutorials recommended by the programming community.
Click here to check out the best Flutter Tutorials https://hackr.io/tutorials/learn-flutter

- **Introduction of Flutter by Google**

https://hackr.io/tutorials/learn-flutter

- **You can also subscribe to Flutter Weekly**

https://hackr.io/tutorials/learn-flutter

- **Here is Flutter (@r_FlutterDev) on Twitter**

https://hackr.io/tutorials/learn-flutter

Student Task_Extra

- The student who repeat the example in this ppt have extra mark
- GOOD chance for student who didn’t send the GTASK

- **JUST MOOC**
- **Your file should have this format of name**

  <Task number><student name><Student ID>.ppt

Reference

- https://flutterbyexample.com/lesson/about-dart
- https://medium.com/app-dev-community/flutter-button-types-with-examples-10ae487621a3
- https://www.javatpoint.com/flutter-forms
- https://www.quytech.com/blog/top-11-key-features-of-a-successful-mobile-app-development/
- http://www.appinventor.org/
- Flutter Docs Tutorial Links: • Flutter for Android devs • Flutter for iOS devs • Flutter for React Native devs • Flutter for Web devs https://hackr.io/tutorials/learn-flutter
- Google Codelabs https://hackr.io/tutorials/learn-flutter
- Github repository https://hackr.io/tutorials/learn-flutter
- Udacity Course https://hackr.io/tutorials/learn-flutter
- MTechViral: YouTube https://hackr.io/tutorials/learn-flutter
- You just saw the top 8 Flutter Tutorials/Programming Community and courses but do you know which of the above ones is “the best”? No! https://hackr.io/tutorials/learn-flutter
- Hackr.io is the solution. Presenting you the best programming courses and tutorials recommended by the programming community. Click here to check out the best Flutter Tutorials https://hackr.io/tutorials/learn-flutter
- Introduction of Flutter by Google https://hackr.io/tutorials/learn-flutter
- You can also subscribe to Flutter Weekly https://hackr.io/tutorials/learn-flutter
- Here is Flutter (@r_FlutterDev) on Twitter https://hackr.io/tutorials/learn-flutter
- https://flutter.dev/docs/development/tools/vs-code#:~:text=To%20perform%20a%20hot%20restart,Shift%20%2B%20F5%20on%20macOS).
- https://flutterbyexample.com/lesson/about-dart
- https://medium.com/app-dev-community/flutter-button-types-with-examples-10ae487621a3
- https://www.javatpoint.com/flutter-forms

江西理工大学
Jiangxi University of Science and Technology
信息工程学院
School of information engineering

Digital Image Processing

THANK YOU

"The beauty of research is that you never know where it's going to lead."

RICHARD ROBERTS
Nobel Prize in Physiology or Medicine 1993

Flutter

"BE HUMBLE. BE HUNGRY. AND ALWAYS BE THE HARDEST WORKER IN THE ROOM."