

Contents

[**Introduction** 6](#_Toc119325957)

[**Flutter:** 6](#_Toc119325958)

[**Framework architecture** 6](#_Toc119325959)

[**Dart Language** 6](#_Toc119325960)

[**IDE support** 7](#_Toc119325961)

[**Widgets** 7](#_Toc119325962)

[**INSTALLATION** 8](#_Toc119325963)

[**Windows install** 8](#_Toc119325964)

[**System requirements** 8](#_Toc119325965)

[**Get the Flutter SDK** 8](#_Toc119325966)

[**Update your path** 9](#_Toc119325967)

[**Run  flutter doctor** 9](#_Toc119325968)

[**Android setup** 10](#_Toc119325969)

[**Install Android Studio** 10](#_Toc119325970)

[**Set up your Android device** 10](#_Toc119325971)

[**Set up the Android emulator** 11](#_Toc119325972)

[**Agree to Android Licenses** 11](#_Toc119325973)

[**Windows setup** 12](#_Toc119325974)

[**Web setup** 12](#_Toc119325975)

[**Set up an editor** 12](#_Toc119325976)

[**Install VS Code** 13](#_Toc119325977)

[**Install the Flutter and Dart plugins** 13](#_Toc119325978)

[**Validate your setup with the Flutter Doctor** 13](#_Toc119325979)

[**Test drive** 14](#_Toc119325980)

[**Create the app** 14](#_Toc119325981)

[**Run the app** 14](#_Toc119325982)

[**Try hot reload** 16](#_Toc119325983)

[**Dart Basics** 17](#_Toc119325984)

[**Dart intro** 17](#_Toc119325985)

[**Dart: The language** 17](#_Toc119325986)

[**Dart Object-Oriented Concepts** 19](#_Toc119325987)

[**Variables** 19](#_Toc119325988)

[Rules 20](#_Toc119325989)

[**final and const** 21](#_Toc119325990)

[**Types** 22](#_Toc119325991)

[**Numbers** 22](#_Toc119325992)

[**Strings** 23](#_Toc119325993)

[**Booleans** 23](#_Toc119325994)

[**Lists** 23](#_Toc119325995)

[**Maps** 24](#_Toc119325996)

[**Functions** 24](#_Toc119325997)

[**The main() function** 33](#_Toc119325998)

[**Dart Recursive Function** 35](#_Toc119325999)

[**Named Parameters in Function** 38](#_Toc119326000)

[**More Functions** 38](#_Toc119326001)

[**Function Assignment** 38](#_Toc119326002)

[**Classes and Object Part 1** 38](#_Toc119326003)

[**Classes and Object Part 2** 40](#_Toc119326004)

[**Constructors** 43](#_Toc119326005)

[**Dart Constructor with Syntactic sugar** 44](#_Toc119326006)

[**Multiple constructors in Dart/Flutter** 44](#_Toc119326007)

[**More about Constructors** 46](#_Toc119326008)

[**Named constructors** 50](#_Toc119326009)

[**String Interpolation** 52](#_Toc119326010)

[**Example** 53](#_Toc119326011)

[**Car Assignment** 53](#_Toc119326012)

[**Installing Dart** 53](#_Toc119326013)

[**Moving Code to IDE** 53](#_Toc119326014)

[**Modularity** 53](#_Toc119326015)

[**Private / Public** 54](#_Toc119326016)

[**Getter Functions** 58](#_Toc119326017)

[**Getters and Setters** 58](#_Toc119326018)

[**Lists** 60](#_Toc119326019)

[**List Function** 77](#_Toc119326020)

[**List Assignment** 77](#_Toc119326021)

[**Maps Start** 77](#_Toc119326022)

[**Using Map Constructor** 79](#_Toc119326023)

[**More about Maps** 84](#_Toc119326024)

[**Iterations Intro** 84](#_Toc119326025)

[**If Statement Intro** 84](#_Toc119326026)

[**More Ifs** 85](#_Toc119326027)

[**More about Iterations** 86](#_Toc119326028)

[**Iterations Final** 94](#_Toc119326029)

[**Repository Assignment** 94](#_Toc119326030)

[**Cascade Operator** 94](#_Toc119326031)

[**Cascade notation: part 2** 95](#_Toc119326032)

[**Inheritance part 1** 96](#_Toc119326033)

[**Inheritance part 2** 98](#_Toc119326034)

[**Debugging** 103](#_Toc119326035)

[**“Too many open files” exception (MacOS)** 107](#_Toc119326036)

[**Override** 108](#_Toc119326037)

[**Abstract Classes** 113](#_Toc119326038)

[**Assignment Doctor** 115](#_Toc119326039)

[**Generic Types** 115](#_Toc119326040)

[**Passing Function Callback** 121](#_Toc119326041)

[**Custom Foreach Callback** 124](#_Toc119326042)

[**Get Input From User** 124](#_Toc119326043)

[**Check User Options + Switch** 125](#_Toc119326044)

[**Improve Display Users** 127](#_Toc119326045)

[**Futures** 127](#_Toc119326046)

[**Async and Await** 130](#_Toc119326047)

[**Static** 134](#_Toc119326048)

[**Safe Operators** 138](#_Toc119326049)

[**Constructors:** 149](#_Toc119326050)

[**Scaffold:** 149](#_Toc119326051)

[**MaterialApp** 151](#_Toc119326052)

[**AppBar** 153](#_Toc119326053)

[**AppBarTheme** **class** 156](#_Toc119326054)

[**Body** 157](#_Toc119326055)

[**Container** 157](#_Toc119326056)

[**Constructor (Custom made):** 160](#_Toc119326057)

[**Components:** 160](#_Toc119326058)

[**Clay Container:** 160](#_Toc119326059)

[**ListView** 162](#_Toc119326060)

[**GridView** 166](#_Toc119326061)

[**Advance Deployment** 171](#_Toc119326062)

[**Neumorphism** 171](#_Toc119326063)

[**Drawer:** 174](#_Toc119326064)

[**ReorderableListView** 180](#_Toc119326065)

[**Database:** 188](#_Toc119326066)

[**Online Database** 188](#_Toc119326067)

[**Firebase** 188](#_Toc119326068)

[**Local Database** 188](#_Toc119326069)

[**Sqflite:** 188](#_Toc119326070)

[**MySql:** 188](#_Toc119326071)

[**Provider** 188](#_Toc119326072)

[**Project:** 188](#_Toc119326073)

[**First Steps** 188](#_Toc119326074)

[**Next Steps** 188](#_Toc119326075)

[**Possible Projects can be achieved:** 188](#_Toc119326076)

[**Reference** 189](#_Toc119326077)

# **Introduction**

## **Flutter:**

Flutter is an open-source UI software development kit created by Google. It is used to develop cross platform applications for Android, iOS, Linux, macOS, Windows, Google Fuchsia, and the web from a single codebase. [Wikipedia]

## **Framework architecture**

The major components of Flutter include:

* Dart platform
* Flutter engine
* Foundation library
* Design-specific widgets
* Flutter Development Tools (DevTools)

## **Dart Language**

Flutter apps are written in the Dart language and make use of many of the language's more advanced features.

While writing and debugging an application, Flutter runs in the Dart virtual machine, which features a just-in-time execution engine. This allows for fast compilation times as well as "hot reload", with which modifications to source files can be injected into a running application. Flutter extends this further 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 on all platforms use ahead-of-time (AOT) compilation, except for on the Web where code is transpiled to JavaScript.

Flutter inherits Dart's Pub package manager and software repository, which allows users to publish and use custom packages as well as Flutter-specific plugins.

## **IDE support**

Flutter maintains official support for the following IDEs and editors via plugins:

* IntelliJ IDEA
* Android Studio
* Visual Studio Code
* Emacs

Other IDEs can be used with community-supported plugins, or by using Flutter tools from the command line.

## **Widgets**

The basic component in a Flutter program is a "widget", which can in turn consist of other widgets. A widget describes the logic, interaction, and design of a UI element with an implementation similar to React. Unlike other cross-platform toolkits such as React Native and Xamarin which draw widgets using native platform components, Flutter renders widgets itself on a per-pixel basis.

There are two fundamental types of widgets: stateless and stateful. Stateless widgets only update if their inputs change, making them very efficient, while stateful widgets can call the setState() method to update an internal state and redraw.

Although widgets are the primary method of constructing Flutter applications, they can also be bypassed in favor of directly drawing on a canvas. This feature has been occasionally used to implement game engines in Flutter.

# **INSTALLATION**

## **Windows install**

### **System requirements**

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

* **Operating Systems**: Windows 10 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](https://docs.microsoft.com/en-us/powershell/scripting/install/installing-windows-powershell) or newer (this is pre-installed with Windows 10)
  + [Git for Windows](https://git-scm.com/download/win) 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.

### **Get the Flutter SDK**

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

<https://storage.googleapis.com/flutter_infra_release/releases/stable/windows/flutter_windows_3.3.7-stable.zip>

For other release channels, and older builds, see the [SDK releases](https://docs.flutter.dev/development/tools/sdk/releases) page.

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

**Warning:**Do not install Flutter to a path that contains special characters or spaces.

**Warning:**Do not install Flutter in a directory like C:\Program Files\ that requires elevated privileges.

1. If you don’t want to install a fixed version of the installation bundle, you can skip steps 1 and 2. Instead, get the source code from the [Flutter repo](https://github.com/flutter/flutter) on GitHub, and change branches or tags as needed. For example:

git clone https://github.com/flutter/flutter.git -b stable

### **Update your path**

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

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

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

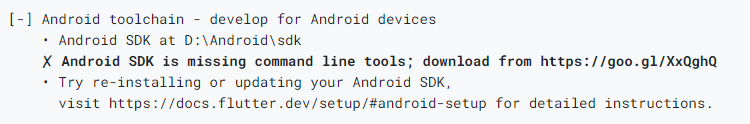
### **Run  flutter doctor**

From a console window that has the Flutter directory in the path, 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 might need to install or further tasks to perform (shown in **bold** text).

For example:



Once you have installed any missing dependencies, you can run the flutter doctor command again to verify that you’ve set everything up correctly.

## **Android setup**

**Note:** Flutter relies on a full installation of Android Studio to supply its Android platform dependencies. However, you can write your Flutter apps in a number of editors; a later step discusses that.

### **Install Android Studio**

1. Download and install [**Android Studio**](https://developer.android.com/studio).
2. Start Android Studio, and go through the ‘Android Studio Setup Wizard’. This installs the latest Android SDK, Android SDK Command-line Tools, and Android SDK Build-Tools, which are required by Flutter when developing for Android.
3. Run flutter doctor to confirm that Flutter has located your installation of Android Studio. If Flutter cannot locate it, run flutter config --android-studio-dir <directory> to set the directory that Android Studio is installed to.

### **Set up your Android device**

To prepare to run and test your Flutter app on an Android device, you need an Android device running Android 4.1 (API level 16) or higher.

1. Enable **Developer options** and **USB debugging** on your device. Detailed instructions are available in the [Android documentation](https://developer.android.com/studio/debug/dev-options).
2. Windows-only: Install the [Google USB Driver](https://developer.android.com/studio/run/win-usb).
3. Using a USB cable, plug your phone into your computer. If prompted on your device, authorize your computer to access your device.
4. In the terminal, run the flutter devices command to verify that Flutter recognizes your connected Android device. By default, Flutter uses the version of the Android SDK where your adb tool is based. If you want Flutter to use a different installation of the Android SDK, you must set the ANDROID\_SDK\_ROOT environment variable to that installation directory.

### **Set up the Android emulator**

To prepare to run and test your Flutter app on the Android emulator, follow these steps:

1. Enable [VM acceleration](https://developer.android.com/studio/run/emulator-acceleration#accel-vm) on your machine.
2. Launch **Android Studio**, click the **AVD Manager** icon, and select **Create Virtual Device…**
   * In older versions of Android Studio, you should instead launch **Android Studio > Tools > Android > AVD Manager** and select **Create Virtual Device…**. (The **Android** submenu is only present when inside an Android project.)
   * If you do not have a project open, you can choose **Configure > AVD Manager** and select **Create Virtual Device…**
3. Choose a device definition and select **Next**.
4. Select one or more system images for the Android versions you want to emulate, and select **Next**. An *x86* or *x86\_64* image is recommended.
5. Under Emulated Performance, select **Hardware - GLES 2.0** to enable [hardware acceleration](https://developer.android.com/studio/run/emulator-acceleration).
6. Verify the AVD configuration is correct, and select **Finish**.

For details on the above steps, see [Managing AVDs](https://developer.android.com/studio/run/managing-avds).

1. In Android Virtual Device Manager, click **Run** in the toolbar. The emulator starts up and displays the default canvas for your selected OS version and device.

### **Agree to Android Licenses**

Before you can use Flutter, you must agree to the licenses of the Android SDK platform. This step should be done after you have installed the tools listed above.

1. Make sure that you have a version of Java 8 installed and that your JAVA\_HOME environment variable is set to the JDK’s folder.

Android Studio versions 2.2 and higher come with a JDK, so this should already be done.

1. Open an elevated console window and run the following command to begin signing licenses.

$ flutter doctor --android-licenses

1. Review the terms of each license carefully before agreeing to them.
2. Once you are done agreeing with licenses, run flutter doctor again to confirm that you are ready to use Flutter.

## **Windows setup**

For Windows desktop development, you need the following in addition to the Flutter SDK:

* [Visual Studio 2022](https://visualstudio.microsoft.com/downloads/) or [Visual Studio Build Tools 2022](https://visualstudio.microsoft.com/downloads/#build-tools-for-visual-studio-2022) When installing Visual Studio or only the Build Tools, you need the “Desktop development with C++” workload installed for building windows, including all of its default components.

**Note:** **Visual Studio** is different than Visual Studio *Code*.

For more information, see [Desktop support for Flutter](https://docs.flutter.dev/desktop)

## **Web setup**

Flutter has support for building web applications in the stable channel. Any app created in Flutter 2 automatically builds for the web. To add web support to an app created before web was in stable, follow the instructions on [Building a web application with Flutter](https://docs.flutter.dev/get-started/web) when you’ve completed the setup above.

## **Set up an editor**

You can build apps with Flutter using any text editor combined with Flutter’s command-line tools. However, we recommend using one of our editor plugins for an even better experience. These plugins provide you with code completion, syntax highlighting, widget editing assists, run & debug support, and more.

### **Install VS Code**

VS Code is a lightweight editor with complete Flutter app execution and debug support.

* [VS Code](https://code.visualstudio.com/), latest stable version.

### **Install the Flutter and Dart plugins**

1. Start VS Code.
2. Invoke **View > Command Palette…**.
3. Type “install”, and select **Extensions: Install Extensions**.
4. Type “flutter” in the extensions search field, select **Flutter** in the list, and click **Install**. This also installs the required Dart plugin.

### **Validate your setup with the Flutter Doctor**

1. Invoke **View > Command Palette…**.
2. Type “doctor”, and select the **Flutter: Run Flutter Doctor**.
3. Review the output in the **OUTPUT** pane for any issues. Make sure to select Flutter from the dropdown in the different Output Options.

## **Test drive**

This page describes how to create a new Flutter app from templates, run it, and experience “hot reload” after you make changes to the app.

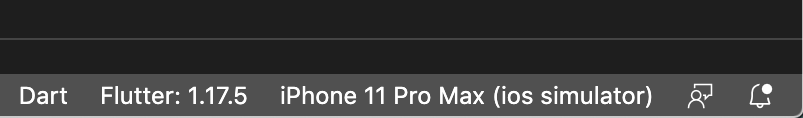
### **Create the app**

1. Invoke **View > Command Palette**.
2. Type “flutter”, and select the **Flutter: New Project**.
3. Select **Application**.
4. Create or select the parent directory for the new project folder.
5. Enter a project name, such as my\_app, and press **Enter**.
6. Wait for project creation to complete and the main.dart file to appear.

The above commands create a Flutter project directory called my\_app that contains a simple demo app that uses [Material Components](https://material.io/guidelines).

**Warning:** If VS Code was running during your initial Flutter setup, you might need to restart it for VS Code’s Flutter plugin to detect the Flutter SDK.

### **Run the app**

1. Locate the VS Code status bar (the blue bar at the bottom of the window):  
   
2. Select a device from the **Device Selector** area. For details, see [Quickly switching between Flutter devices](https://dartcode.org/docs/quickly-switching-between-flutter-devices).
   * If no device is available, and you want to use a device simulator, click **No Devices** and click **Start iOS Simulator** to launch a simulator.

**Warning:** You might not see **Start iOS Simulator** option when you click **No Devices** in VS Code. If you’re on Mac, then you might have to run following command in terminal to launch a simulator.

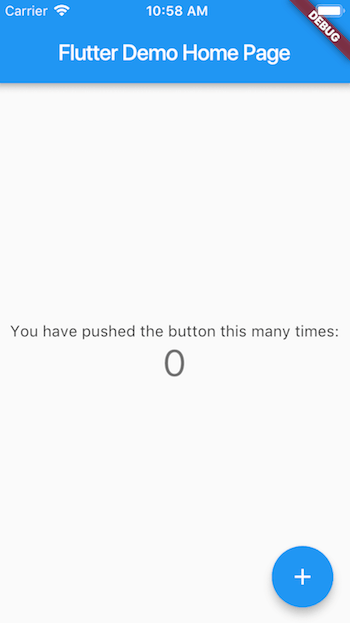
$ open -a simulator

On Windows or Linux, it’s not possible to launch an iOS simulator.

* + To setup a real device, follow the device-specific instructions on the [Install](https://docs.flutter.dev/get-started/install) page for your OS.

1. Invoke **Run > Start Debugging** or press F5.
2. Wait for the app to launch—progress is printed in the **Debug Console** view.

After the app build completes, you’ll see the starter app on your device.



### **Try hot reload**

Flutter offers a fast development cycle with *Stateful Hot Reload*, the ability to reload the code of a live running app without restarting or losing app state. Make a change to app source, tell your IDE or command-line tool that you want to hot reload, and see the change in your simulator, emulator, or device.

1. Open lib/main.dart.
2. Change the string

'You have pushed the button this many times'

to

'You have clicked the button this many times'

**Important:** Do *not* stop your app. Let your app run.

1. Save your changes: invoke **Save All**, or click **Hot Reload**  .

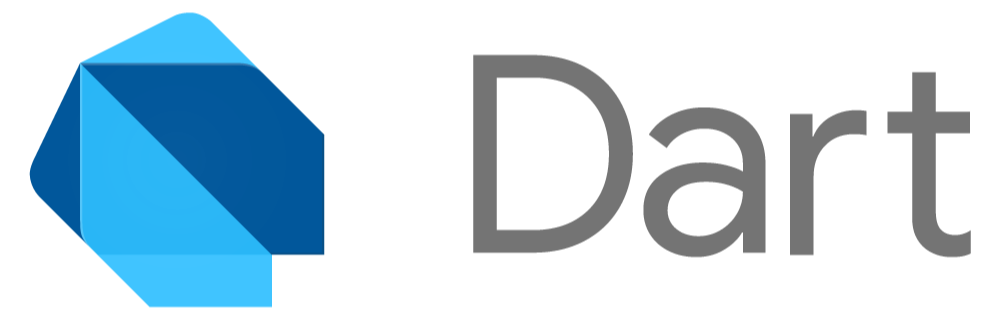
You’ll see the updated string in the running app almost immediately.

So, that’s all the basic requirements you need to start your first flutter application. For more detail and info please visit flutter official website. Link is given bellow:

<https://docs.flutter.dev/>

# **Dart Basics**

## **Dart intro**



Dart is a client-optimized language for developing fast apps on any platform. Its goal is to offer the most productive programming language for multi-platform development, paired with a [flexible execution runtime platform](https://dart.dev/overview#platform) for app frameworks.

Languages are defined by their technical envelope—the choices made during development that shape the capabilities and strengths of a language. Dart is designed for a technical envelope that is particularly suited to client development, prioritizing both development (sub-second stateful hot reload) and high-quality production experiences across a wide variety of compilation targets (web, mobile, and desktop).

Dart also forms the foundation of [Flutter](https://flutter.dev/). Dart provides the language and runtimes that power Flutter apps, but Dart also supports many cores developer tasks like formatting, analyzing, and testing code.

## **Dart: The language**

The Dart language is type safe; it uses static type checking to ensure that a variable’s value always matches the variable’s static type. Sometimes, this is referred to as sound typing. Although types are mandatory, type annotations are optional because of type inference. The Dart typing system is also flexible, allowing the use of a dynamic type combined with runtime checks, which can be useful during experimentation or for code that needs to be especially dynamic.

Dart offers [sound null safety](https://dart.dev/null-safety), meaning that values can’t be null unless you say they can be. With sound null safety, Dart can protect you from null exceptions at runtime through static code analysis. Unlike many other null-safe languages, when Dart determines that a variable is non-nullable, that variable is always non-nullable. If you inspect your running code in the debugger, you’ll see that non-nullability is retained at runtime (hence sound null safety).

The following code sample showcases several Dart language features, including libraries, async calls, nullable and non-nullable types, arrow syntax, generators, streams, and getters. To find examples of using additional Dart features, see the [samples page](https://dart.dev/samples). To learn more about the language, take the [Dart language tour](https://dart.dev/guides/language/language-tour).

Example:

import 'dart:math' show Random;

void main() async {

  print('Compute π using the Monte Carlo method.');

  await for (final estimate in computePi().take(100)) {

    print('π ≅ $estimate');

  }

}

/// Generates a stream of increasingly accurate estimates of π.

Stream<double> computePi({int batch = 100000}) async\* {

  var total = 0; // Inferred to be of type int

  var count = 0;

  while (true) {

    final points = generateRandom().take(batch);

    final inside = points.where((p) => p.isInsideUnitCircle);

    total += batch;

    count += inside.length;

    final ratio = count / total;

    // Area of a circle is A = π⋅r², therefore π = A/r².

    // So, when given random points with x ∈ <0,1>,

    // y ∈ <0,1>, the ratio of those inside a unit circle

    // should approach π / 4. Therefore, the value of π

    // should be:

    yield ratio \* 4;

  }

}

Iterable<Point> generateRandom([int? seed]) sync\* {

  final random = Random(seed);

  while (true) {

    yield Point(random.nextDouble(), random.nextDouble());

  }

}

class Point {

  final double x;

  final double y;

  const Point(this.x, this.y);

  bool get isInsideUnitCircle => x \* x + y \* y <= 1;

}

## **Dart Object-Oriented Concepts**

[8]

Dart is an object-oriented programming language, and it supports all the concepts of object-oriented programming such as classes, object, inheritance, mixin, and abstract classes. As the name suggests, it focuses on the object and objects are the real-life entities. The Object-oriented programming approach is used to implement the concept like polymorphism, data-hiding, etc. The main goal of oops is to reduce programming complexity and do several tasks simultaneously. The oops concepts are given below.

* Class
* Object
* Inheritance
* Polymorphism
* Interfaces
* Abstract class

## **Variables**

[1]

Variables is a type of identifier that refers to a memory address in a computer’s memory that maintains a value for that variable. This value can be updated throughout the program’s execution. When you declare a variable in Dart, you are allocating memory for that variable. The size of the memory space allocated and the type of the value it holds entirely depend on the type of variable.

**Declaration**

Syntax: datatype variable\_name;

Note: Before a variable can be utilized in a program, it must first be declared.

### Rules

Variables or the identifier must conform to the following rules when declared:

1. The keyword cannot be a variable name or an identifier.
2. A variable name or identifier can contain alphabets and integers
3. Except for the underscore\_ and the dollar ＄ symbol, variable names or identifiers cannot contain spaces or special characters.
4. A number cannot be the first character in a variable name or the identifier.

Note: Dart supports type-checking, which means it determines whether the data type and data that are stored by a variable are unique to that data.

**Example**

//declaring and assigning  
String name = 'Maria';   
  
int a = "Maria"; // typeError  
// A string can't be assigned to a variable of type int

**Dynamic type variables**

dynamic creates a specific variable called dynamic. During the program’s execution, the variable declared with this data type can store any value implicitly. It’s comparable to the var datatype in Dart. However, the distinction is that when you assign data to a variable using the var keyword, the appropriate data type is changed.

Syntax: dynamic variable\_name;

**Code**

void main() {

  // Assigning value to variable

  dynamic myType = "Dynamic type";

  print(myType);

  // Reassigning  value to variable

  myType = 22.0691;

  print(myType);

}

Note: If we use var instead of dynamic in the code above, it will throw a **typeError:** a variable of the double type that cannot be assigned to a variable of the String type.

### **final and const**

In Dart, the keywords final and const are used to define constant variables. This means that once a variable is defined using these keywords, its value cannot be modified throughout the code. These keywords can be combined with or without the name of a data type.

#### **final declaration**

Syntax: final data\_type variable\_name //with datatype  
  
Syntax: variable\_name //without datatype

#### **const declaration**

Syntax: const data\_type variable\_name //with datatype  
  
Syntax: variable\_name //without datatype

#### **Code**

The code below shows the use of the final and const keywords in Dart programming:

void main() {

  // const variable

  const myType = "const type";

  print(myType);

  // final variable

  final myType1 = "final type";

  final String myType2 = "final type !!!";

  print(myType1);

  print(myType2);

}

Note: A final or const variable can’t be reassigned.

## **Types**

[2]

When a variable is created in Dart, it has an associated data type, just like in other languages (C, Python, Java).

Data types for a variable specifies the following:

1. The amount of space to be allocated
2. Possible values
3. The operation to be performed on the variable

Dart is a programming language that is statically typed; this means that variables always have the same type, which cannot be changed.

Data types in Dart programming are given below:

1. Numbers
2. Strings
3. Booleans
4. Lists
5. Maps

### **Numbers**

In Dart, numbers are used to represent numeric literals. Dart numbers are grouped into two types:

1. **Integer**: represents non-fractional numbers (whole numbers). Integers can be declared with the int keyword.
2. **Double**: represents fractional numbers (floating-point numbers). Doubles can be declared with the double keyword.

void main() {

    //declare a integer value

    int num1 = 1;

    // declare a double value

    double num2 = 1.5;

    print(num1);

    print(num2);

}

### **Strings**

A **string** represents string literals and is a sequence of characters. A string is declared with the String keyword.

void main() {

    String str = 'Educative';

    print(str);

}

### **Booleans**

A **boolean** represents true and false values. It is declared with the bool keyword.

void main() {

  bool val = true;

  print(val);

}

### **Lists**

A **list** is used to represent a collection of objects. It is similar to the concept of an array in other programming languages. A list is a group of ordered objects.

void main()

{

    List shot = List(3);

    shot[0] = 'Data types';

    shot[1] = 'in';

    shot[2] = 'Dart';

    print(shot);

    // access the first index

    print(shot[0]);

}

### **Maps**

A **map** is a dynamic collection that represents a set of values as key-value pairs. Keys and values on a map may be of any type.

void main()

{

    Map shot =  Map();

    shot['0'] = 'Data types';

    shot['Second'] = 'in';

    shot['a'] = 'Dart';

    print(shot);

}

## **Functions**

[3]

Dart function is a set of codes that together perform a specific task. It is used to break the large code into smaller modules and reuse it when needed. Functions make the program more readable and easier to debug. It improves the modular approach and enhances the code reusability.

Suppose, we write a simple calculator program where we need to perform operations number of times when the user enters the values. We can create different functions for each calculator operator. By using the functions, we don't need to write code for adding, subtracting, multiplying, and divide again and again. We can use the functions multiple times by calling.

The function provides the flexibility to run a code several times with different values. A function can be called anytime as its parameter and returns some value to where it called.

**Defining a Function**

A function can be defined by providing the name of the function with the appropriate parameter and return type. A function contains a set of statements which are called function body. The syntax is given below.

Example:

return\_type func\_name (parameter\_list):

{

    //statement(s)

   return value;

}

Let's understand the general syntax of the defining function.

* **return\_type -** It can be any data type such as void, integer, float, etc. The return type must be matched with the returned value of the function.
* **func\_name -** It should be an appropriate and valid identifier.
* **parameter\_list -** It denotes the list of the parameters, which is necessary when we called a function.
* **return value -** A function returns a value after complete its execution.

Let's understand the following example.

**Example – 1**

int mul(int a, int b){

     int c;

     c = a+b;

     print("The sum is:${c}");

}

**Calling a Function**

After creating a function, we can call or invoke the defined function inside the [main() function](https://www.javatpoint.com/dart-main-function) body. A function is invoked simply by its name with a parameter list, if any. The syntax is given below.

**Syntax**:

fun\_name(<argument\_list>);

or

variable = function\_name(argument);

#### **Note - Calling function must be ended with semicolon (;).**

When we call a function, the control is transferred to the called function. Then the called function executes all defined statements and returns the result to the calling function. The control returns to the main() function..

mul(10,20);

**Passing Arguments to Function**

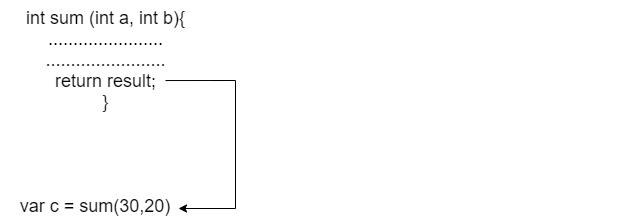
When a function is called, it may have some information as per the function prototype is known as a parameter (argument). The number of parameters passed and data type while the function call must be matched with the number of parameters during function declaration. Otherwise, it will throw an error. Parameter passing is also optional, which means it is not compulsory to pass during function declaration. The parameter can be two types.

**Actual Parameter -** A parameter which is passed during a function definition is called the actual parameter.

**Formal Parameter -** A parameter which is passed during a function call is called the formal parameter.

We will learn more about the parameter in the next tutorial.

A function always returns some value as a result to the point where it is called. The **return** keyword is used to return a value. The return statement is optional. A function can have only one return statement. The syntax is given below.



**Syntax:**

return <expression/values>

**Example -**

return result;

**Function Examples**

Let's understand the functions by using a program of adding two numbers using functions.

**Dart Function with parameter and return value**

In the following example, we are creating a sum() function to add two number.

**Example - 1**

void main() {

  print("Example of add two number using the function");

  // Creating a Function

  int sum(int a, int b){

            // function Body

            int result;

            result = a+b;

            return result;

}

// We are calling a function and storing a result in variable c

var c = sum(30,20);

print("The sum of two numbers is: ${c}");

}

**Output**

Example of add two number using the function

The sum of two numbers is: 50

**Explanation:**

In the above example, we declared a function named **sum()** and passed two integer variables as actual parameters. In the function body, we declared a **result** variable to store the sum of two numbers and returned the result.

In order to add two values, we called a function with the same name, passed formal parameters 30 and 20. The **sum()** returned a value which we stored in the variable c and printed the sum on the console.

**Dart Function with No Parameter and Return Value**

As we discussed earlier, the parameters are optional to pass while defining a function. We can create a function without parameter return value. The syntax is given below.

**Syntax:**

return\_type func\_name()

{

       //Statement(s);

       return value;

}

Let's understand the following example.

**Example - 2**

void main(){

// Creating a function without argument

String greetings(){

   return "Welcome to JavaTpoint";

}

// Calling function inside print statement

print(greetings());

}

**Output**

Welcome to JavaTpoint

**Explanation:**

In the above example, we created a function named **greetings()** without argument and returned the string value to the calling function. Then, we called the greeting() function inside the print statement and printed the result to the console.

**Dart Function with No Parameter and without a Return Value**

We can declare a function without parameter and no return value. The syntax is given below.

**Syntax:**

func\_name() {

 //statement

}

Or

void fun\_name() {

  //statement(s)

}

In the above general syntax-

**void -** It represents the function has no return type.

**fun\_name -** It represents the function name.

Let's understand the following example.

**Example – 3**

void greetings()

{

   print("Welcome to JavaTpoint");

}

void main() {

  print("The example of Dart Function");

  // function callling

  greetings();

}

**Output**

The example of Dart Function

Welcome to JavaTpoint

**Explanation:**

In the above example, we created a function called **greeting()** outside the **main()** function and writing the print statement. Inside the **main()** function, we called the defined function and printed the output to console.

Dart Function with Parameter and without a Return Value

We are creating a function to find the given number is even or odd. Let's understand the following example.

**Example - 4**

void main()

{

  void number(int n){

           // Check the given number is even or odd

           if (n%2 ==0){

                   print("The given number is even");

             }

           else {

                  print("The given number is odd");

            }

}

   number(20);

}

**Output**

The given number is even

**Dart Anonymous Function**

We have learned the Dart Function, which is defined by using a user-define name. Dart also provides the facility to specify a nameless function or function without a name. This type of function is known as an **anonymous function, lambda, or closure**. An anonymous function behaves the same as a regular function, but it does not have a name with it. It can have zero or any number of arguments with an optional type annotation.

We can assign the anonymous function to a variable, and then we can retrieve or access the value of the closure based on our requirement.

An Anonymous function contains an independent block of the code, and that can be passed around in our code as function parameters. The syntax is as follows.

**Syntax:**

(parameter\_list) {

   statement(s)

}

Let's consider the following example.

**Example –**

void main() {

  var list = ["James","Patrick","Mathew","Tom"];

  print("Example of anonymous function");

  list.forEach((item) {

      print('${list.indexOf(item)}: $item');

});

}

**Output:**

Example of anonymous function

0: James

1: Patrick

2: Mathew

3: Tom

**Explanation:**

In the above example, we defined an anonymous function with an untype argument item. The function called for each item in the list and printed the strings with its specified index value.

If the function consists of one statement, then we can also write the above code in the following way.

list.forEach(

(item) => print("${list.indexOf(item)}: $item"));

It is equivalent to the previous code. You can verify it by paste in your dart pad and run.

**Lexical Scope**

As we have discussed in the [Dart](https://www.javatpoint.com/dart-programming) introduction, it is a lexical scope language which means the variable's scope is decided at compile-time. The scope of the variable is determined when code is compiled. The variable behaves differently if they defined in the different curly braces. Let's understand the following example.

**Example -**

bool topVariable = true;

void main() {

  var inside\_Main = true;

 // Defining Nested Function

 void myFunction() {

    var inside\_Function = true;

 void nestedFunction() {

      var inside\_NestedFunction = true;

      // This function is using all variable of the previous functions.

      assert(topVariable);

      assert(inside\_Main);

      assert(inside\_Function);

      assert(inside\_NestedFunction);

    }

  }

}

Observe the above code, the **nestedFunction()** used the variables of the previous function.

**Lexical Closure**

A lexical closure is referred to as a closure, is a function object that has access to variables in its lexical scope even when the function is used of its original scope. In other words, it provides access to an outer function's scope from inner function. Let's understand the following example.

**Example** -

void main() {

 String initial() {

     var name = 'Will Smith'; // name is a local variable created by init

     void disp\_Name() { // displayName() is the inner function, a closure

           print(name); // use variable declared in the parent function

  }

  disp\_Name();

}

init();

**Output**

Will Smith

**Explanation:**

In the above code, the **initial()** function created a local variable called **name** and function called **disp\_Name()**. The **disp\_Name()** function defined inside the **initial()** function and hence **disp\_Name()** function has no local variable its own.

The inner function can access the variable of the outer functions. The function disp\_Name() can access the name variable which is declared in the outer function, **initial()**.

## **The main() function**

[4]

The main() function is the top-level function of the Dart. It is the most important and vital function of the [Dart](https://www.javatpoint.com/dart-programming) programming language. The execution of the programming starts with the **main()** function. The **main()** function can be used only once in a program.

It is responsible for all types of execution such as user-defined statements, functions, and libraries function. The program begins with main() function and we declares variable, and user defined executable statements inside it. The main function returns void and can have an optional List<String> parameter as arguments. The general syntax of the main() function is given below.

**Syntax:**

void main() {

  // main function body

}

**Example - 1**

void main()

{

  print("Welcome To JavaTpoint");

 }

**Output**

Welcome To Flutter Course

**Dart Return Value**

Sometimes the function returns a value after evaluating the function statements to the point where it is called from. The return statement holds the result of the function, and it is transferred to the function call. The **return** keyword is used to represent the return statement. If the return statement not specified, then the function returns null. The return statement is optional to specify in function, but there can be only one return statement in a function.

**Syntax:**

return <expression/value>;

**Dart value with Return Value**

Below is given syntax of return value.

**Syntax:**

return\_type function\_name()

{

   //statement(s);

  return value;

}

Here is the description of the above syntax.

**function\_name -** It represents the function name, which can be any valid identifier.

return type - It denotes the return type of the function. It can be any valid data type. The return must be matched with the return type of the function.

Let's understand the following example -

**Example –**

void main() {

  int mul(int a, int b){

        int c = a\*b;

        return c;

}

print("The multiplication of two numbers: ${mul(10,20)}");

}

**Output**

The multiplication of two numbers: 200

## **Dart Recursive Function**

Recursive functions are quite similar to the other functions, but difference is to calling itself recursively. A recursive function repeats multiple times until it returns the final output. It allows programmers to solve complex problems with minimal code.

But before we dive deep into recursive function let’s first look into what is recursion.

Dart Recursion is the method where a function calls itself as its subroutine. It is used to solve the complex problem by dividing it into sub-part. A function which is called itself again and again or recursively, then this process is called recursion.

The iterators can be an option to solve problems, but recursion is recommended to the programmers to deal with complex problems because it is an effective approach of problem-solving technique. It requires less time and code to evaluate the same complex task.

Recursion makes many calls to the same function; however, there should be a base case to terminate the recursion.

Recursion uses the divide and conquers technique to solve a complex mathematical computation task. It divides the large task into small chunks.

Recursion is not recommended to solve all types of problems. However, it is best for a few questions such as searching, sorting, Inorder/Preorder/Postorder, Tree Traversal, and DFS of Graph algorithms. But, while using recursion, it must be implemented carefully; otherwise, it turns into the infinite loop.

**What is base condition in recursion?**

void main() {

   int factorial(int num){

  if(num<=1) { // base case

        return 1;

  else{

         return n\*fact(n-1);

 }

}

}

}

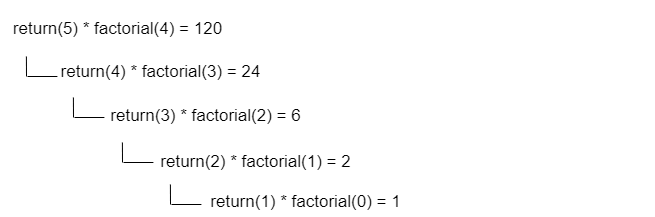
In the above example, the base case is defined as n<=1, and a larger value of a number can be solved by changing to a lesser one till the base case is matched.

#### **Note - The base case or valid terminating condition is required in recursion function; otherwise, it will turn into an infinite loop.**

**How does recursion works?**

Let's understand the concept of the recursion of the example of a factorial of a given number. In the following example, we will evaluate the factorial of n numbers. It is the series of the multiplication as follows.

Factorial of n (n!) = n\*(n-1)\*(n-2)........1



**Characteristics of Recursive Function**

The characteristics of the recursive function are given below.

A recursive function is a unique form of a function where function calls itself.

A valid base case is required to terminate the recursive function.

It is slower than the iteration because of stack overheads.

Let's have a look at recursion syntax:

**Syntax:**

void recurse() {

  //statement(s)

 recurse();

//statement(s);

}

void main(){

   //statement(s)

  recurse();

 //statement(s)

}

Let's understand the following example.

**Example - 1**

int factorial(int num){

  //base case of recursion.

  if(num<=1) { // base case

        return 1;

}

  else{

         return num\*factorial(num-1);    //function call itself.

 }

}

void main() {

  var num = 5;

  // Storing function call result in fact variable.

  var fact = factorial(num);

  print("Factorial Of 5 is: ${fact}");

}

**Output:**

Factorial Of 10 is: 120

**Explanation:**

In the above example, the **factorial()** is a recursive function as it call itself. When we called the **factorial()** function by passing the integer value 5, it will recursively call itself by decreasing the number.

The factorial() function will be called every time until it matched the base condition, or it is equal to one. It multiplied the number with factorial of the number. Consider the following explanation of the recursive call.

factorial(5)              # 1st call with 5

5 \* factorial(4)          # 2nd call with 4

5 \* 4 \* factorial(3)      # 3rd call with 3

5 \* 4 \* 3 \* factorial(2)  # 4th call with 2

5 \* 4 \* 3 \* 2 \* 1         # return from 2nd call

120                    # return from 1st call

The recursion is ended when the number reduced to 1, and it is the base condition of recursion.

A recursion function must have a base condition to avoid to infinite call.

**Disadvantage of Recursion**

The recursive calls consume a lot of memory; that's why these are inefficient.

The recursive functions are difficult to debug.

Sometimes, It is hard to follow the logic behind the recursion.

## **Named Parameters in Function**

## **More Functions**

## **Function Assignment**

## **Classes and Object Part 1**

Dart classes are the blueprint of the object, or it can be called object constructors. A class can contain fields, functions, constructors, etc. It is a wrapper that binds/encapsulates the data and functions together; that can be accessed by creating an object. A class can refer to as user-define data type which defines characteristics by its all objects.

We can assume a class as a sketch (prototype) or a car. It contains all the details about model name, year, features, price, etc. Based on these properties of the car, we can build the car. Here the car is an object. There can be many cars so we can create many objects of cars to access all the properties.

**Defining a Class in Dart**

Dart provides class keyword followed by a class name is used to define a class; all fields and functions are enclosed by the pair of curly braces ({}). The syntax is given below.

**Syntax:**

class ClassName {

   <fields>

   <getters/setters>

  <constructor>

 <functions>

}

Here, the ClassName represents the actual name of the class, which is defined by the user. In curly braces, we provide a class definition. A class can consist of fields, constructors, getters setters, and methods.

#### **Note - According to the naming convention rule of identifiers, the first letter of the class name must be capital and use no separators.**

Let's understand the following example.

**Example -**

void main() {

   // Defining class

 class Student {

   var stdName;

   var stdAge;

   var stdRoll\_nu;

   // Class Function

    showStdInfo() {

        print("Student Name is : ${stdName}");

        print("Student Age is : ${stdAge}");

        print("Student Roll Number is : ${stdRoll\_nu}")

}

In the above example of class, we declared a class called **Student**. This class has three fields **stdName, stdAge**, and **stdRoll\_nu**. The **showStdInfo()** is a class function which prints the fields of class. To access the properties of the class, we need to create its object.

## **Classes and Object Part 2**

**Dart Object**

Dart is object-oriented programming, and everything is treated as an object in Dart. An object is a variable or instance of the class used to access the class's properties. Objects have two features - state and behavior. Suppose a man is an object with a state (name, age, health) and behavior (walking, running, and sleeping). Programming objects are theoretically similar to the real-life objects; they also have state and behavior. An object is created from a template which is known as class.

The fields of the classes are stored as object states, whereas the method represents an object's behavior

**Creating Class Objects in Dart**

After creating the class, we can create an instance or object of that class which we want to access its fields and functions. The **new** keyword is used to declare class followed by the class name. The general syntax of creating an object of a class is given below.

**Syntax:**

var object\_name  = new class\_name(<constructor\_arguments>);

Here, object\_name and class\_name signifies as the actual object name and class name respectively. If the class constructor is parameterized then constructor arguments must be passed value.

Let's understand the following example.

**Example –**

 class Student {

   var stdName;

   var stdAge;

   var stdRoll\_nu;

   // Class Function

    showStdInfo() {

        print("Student Name is : ${stdName}");

        print("Student Age is : ${stdAge}");

        print("Student Roll Number is : ${stdRoll\_nu}")

}

}

void main () {

 // Creating Object called std

  var std = new Student();

}

We have created the object called **std** of the class **Student** but only creating an object not enough. We have to access the properties by using the newly created object.

**Assessing Instance Variable and Function**

After creating an object, we can access the fields and methods of the class. The class property name is separated by the (.) operator with the instance name. The syntax is given below.

**Syntax:**

  objectName.propName or objectName.methoName()

Let's understand the following example.

**Example -**

// Defining class

 class Student {

   var stdName;

   var stdAge;

   var stdRoll\_nu;

   // defining class function

    showStdInfo() {

        print("Student Name is : ${stdName}");

        print("Student Age is : ${stdAge}");

        print("Student Roll Number is : ${stdRoll\_nu}");

               }

}

void main () {

  // Creating object called std

  var std = new Student();

  std.stdName = "Peter";

  std.stdAge =24;

  std.stdRoll\_nu = 90001;

// Accessing class Function

 std.showStdInfo();

}

**Output**

Student Name is: Peter

Student Age is: 24

Student Roll Number is: 90001

**Explanation:**

In the above example, we created a class called **Student**, which consisted of the student name, age, roll number, and **showStdInfo()** function to show the student details.

Then, we created a Student class object and assigned the values to each field by using the (.) operator. We called the showStdInfo() function that displayed the details as an output to screen.

**Benefit of Objects**

There are various benefits of using object-oriented programming. Below are the few benefits.

**Modularity:** The source code of an object can be maintained individually and can hide from the other object's source code.

**Data - hiding:** Using oops programming, the details of the internal functionality of code are hidden from the others. For example - Users only interact with the application, but they don't familiar with the internal implementation.

**Reusability -** We don't need to write the same code again and again. We can use the object of class multiple times in our program.

**Pluggability and debugging easy -** If any object is creating a problem in our program, and then we can replace it in our program and plug the new object as its replacement. The oops code can be easy to debug.

## **Constructors**

[5]

Dart Constructor methods:

Constructor is a special method of Dart class which is automatically called when the object is created. The constructor is like a function with/without parameter but it doesn’t have a return type.

For example, this is Customer class with constructor that has the same name:

class Customer {

String name;

int age;

String location;

// constructor

Customer(String name, int age, String location) {

this.name = name;

this.age = age;

this.location = location;

}

}

Now you can create new object using a constructor.

var customer = Customer("Naruto", 26, "US");

If we don’t define any constructor, the default constructor below will be created.

Customer() {

}

### **Dart Constructor with Syntactic sugar**

If you use constructor with normal syntax above, you need to write boilerplate to assign each argument to an instance variable.

Dart supports syntactic sugar to make it easy.

class Customer {

String name;

int age;

String location;

Customer(this.name, this.age, this.location);

}

Dart supports syntactic sugar to make it easy.

class Customer {

String name;

int age;

String location;

Customer(this.name, this.age, this.location);

}

### **Multiple constructors in Dart/Flutter**

How about the case you want to have more than one constructor. For example, there are 2 constructors you desire to use:

Customer(String name, int age, String location) {

this.name = name;

this.age = age;

this. location = location;

}

Customer(this.name, this.age) {

this.name = name;

this.age = age;

}

But if you define both of them in a class, there will be a compiler error.

Dart provides Named constructor that helps you implement multiple constructors with more clarity:

class Customer {

  // ...

  Customer(String name, int age, String location) {

    this.name = name;

    this.age = age;

    this.location = location;

  }

  // Named constructor - for multiple constructors

  Customer.withoutLocation(this.name, this.age) {

    this.name = name;

    this.age = age;

  }

  Customer.empty() {

    name = "";

    age = 0;

    location = "";

  }

  @override

  String toString() {

    return "Customer [name=${this.name},age=${this.age},location=${this.location}]";

  }

}

You can write it more simply with Syntactic sugar:

Customer(this.name, this.age, this.location);

Customer.withoutLocation(this.name, this.age);

Customer.empty() {

  name = "";

  age = 0;

  location = "";

}

Now we can create new Customer object by these methods.

var customer = Customer("Flutter", 26, "US");

print(customer);

// Customer [name=Flutter,age=26,location=US]

var customer1 = Customer.withoutLocation("Flutter", 26);

print(customer1);

// Customer [name=flutter,age=26,location=null]

var customer2 = Customer.empty();

print(customer2);

// Customer [name=,age=0,location=]

So, is there any way to make Customer.empty() neat?

And how to initialize an empty value for location field when calling Customer.withoutLocation() instead of null?

Yes, we can.

Let’s take a look at Redirecting Constructor.

## **More about Constructors**

[5]

**Redirecting Constructor**

We can redirect a constructor to another constructor in the same class by using a colon (:). Remember that body of Redirecting Constructor is empty.

For example, I will rewrite Customer.empty() & Customer.withoutLocation() above.

class Customer {

  String name;

  int age;

  String location;

  Customer(this.name, this.age, this.location);

  // Redirecting constructors

  Customer.empty() : this("", 0, "");

  Customer.withoutLocation(String name, int age) : this(name, age, "");

}

Let’s run and check again:

var customer1 = Customer.empty();

print(customer1);

// Customer [name=,age=0,location=]

var customer2 = Customer.withoutLocation("flutter", 26);

print(customer2);

// Customer [name=flutter,age=26,location=]

**Factory Constructor in Dart/Flutter**

We can use the factory keyword for a constructor that return an object instead of creating a new instance.

class Customer {

  String name;

  int age;

  String location;

  static final Customer origin = Customer("", 0, "");

  // factory constructor

  factory Customer.create() {

    return origin;

  }

  @override

  String toString() { ... }

}

var customer = Customer.create();

print(customer);

// Customer [name=,age=0,location=]

**Dart/Flutter Constructor with Optional parameters**

We can define constructor with two types of parameters: required and optional. The required parameters (which we used in sections above) are listed first, followed by any optional parameters.

Optional parameters can be **Named** or **Positional**.

**Dart Constructor using Square brackets: Positional optional parameters**

You can wrap one or more parameters in square brackets [ ] to make them as optional positional parameters.

class Customer {

  String name;

  int age;

  String location;

  // Positional optional parameters

  Customer(this.name, [this.age, this.location]);

  @override

  String toString() {

    return "Customer [name=${this.name},age=${this.age},location=${this.location}]";

  }

}

Let’s make some test by calling constructor without the optional parameters:

var customer = Customer("flutter", 26, "US");

print(customer);

// Customer [name=bezkoder,age=26,location=US]

var customer1 = Customer("flutter", 26);

print(customer1);

// Customer [name=bezkoder,age=26,location=null]

var customer2 = Customer("flutter");

print(customer2);

// Customer [name=zkoder,age=null,location=null]

**Dart Constructor using Curly braces: Named optional parameters**

We can also definie a constructor using curly braces { } to specify named parameters.

class Customer {

  String name;

  int age;

  String location;

  // Named optional parameters

  Customer(this.name, {this.age, this.location});

  @override

  String toString() {

    return "Customer [name=${this.name},age=${this.age},location=${this.location}]";

  }

}

When calling the constructor, we have to use parameter name to assign a value which separated with colan paramName: value.

var customer = Customer("flutter", location: "US", age: 26);

print(customer);

// Customer [name=flutter,age=26,location=US]

var customer1 = Customer("flutter", age: 26);

print(customer1);

// Customer [name=flutter,age=26,location=null]

var customer2 = Customer("fulter");

print(customer2);

// Customer [name=zkoder,age=null,location=null]

You can see that the order of parameters does not matter. It can avoid confusion while passing value for the constructor which has many parameter.

**Dart/Flutter Constructor default value**

For the constructors with either Named or Positional parameters, we can use = to define default values.

The default values must be compile-time constants. If we don’t provide value, the default value is null.

**Positional optional parameters**

class Customer {

  String name;

  int age;

  String location;

  Customer(this.name, [this.age, this.location = "US"]);

  @override

  String toString() {

    return "Customer [name=${this.name},age=${this.age},location=${this.location}]";

  }

}

Now create some Customer objects, you can see that default value for age is null and for location is "US".

var customer = Customer("Flutter", 26, "US");

print(customer);

// Customer [name=Flutter,age=26,location=US]

var customer1 = Customer("Flutter", 26);

print(customer1);

// Customer [name=Flutter,age=26,location=US]

var customer2 = Customer("flutter");

print(customer2);

// Customer [name=zkoder,age=null,location=US]

## **Named constructors**

[5]

**Named optional parameters**

class Customer {

  String name;

  int age;

  String location;

  Customer(this.name, {this.age, this.location = "US"});

  @override

  String toString() {

    return "Customer [name=${this.name},age=${this.age},location=${this.location}]";

  }

}

Let’s run to check default values for age & location.

var customer = Customer("Flutter", age: 26, location: "US");

print(customer);

// Customer [name=Flutter,age=26,location=US]

var customer1 = Customer("Flutter", age: 26);

print(customer1);

// Customer [name=Flutter,age=26,location=US]

var customer2 = Customer("flutter");

print(customer2);

// Customer [name=flutter,age=null,location=US]

**Constant constructor**

If we want all instances of our class will never change, we can define a const constructor in which all fields are final.

class ImmutableCustomer {

  final String name;

  final int age;

  final String location;

  // Constant constructor

  const ImmutableCustomer(this.name, this.age, this.location);

}

Now we can put the const keyword before the constructor name:

var immutableCustomer = const ImmutableCustomer("zkoder", 26, "US");

// immutableCustomer.name = ... // compile error

## **String Interpolation**

[6]

**String interpolation** is the process of inserting variable values into placeholders in a string literal.

To concatenate strings in Dart, we can utilize string interpolation. We use the ＄{} symbol to implement string interpolation in your code.

**Code**

void main() {

  // Assigning values to the variable

  String shot1 = "String";

  String shot2 = "interpolation";

  String shot3 = "in";

  String shot4 = "Dart programming";

  // Concatenate all values using

  // string interpolation without space

  print('$shot1$shot2$shot3$shot4');

  // Concatenate all values using

  // string interpolation with space

  print('\n');

  print('Now, include space between each value');

  print('\n');

  print('$shot1 $shot2 $shot3 $shot4');

}

To interpolate the value of Dart expressions within strings, use ＄{}. The curly braces {} are skipped if the expression is an identifier.

void main() {

  String text = 'Educative';

  // text is an identifier

  // so the {} can be omitted

  print('The word $text has ${text.length} letters');

}

### **Example**

The example below uses string interpolation in a list and map operation.

void main(){

  List mylist = [10, 20, 30, 40, 50, 60];

  // Using map on list items

  List newList = [mylist.map((i) => i \* i)];

  print("The list $mylist was squared to give a new list $newList");

  // Creating  Map items

  Map myMap = {"1": "String", "2": "literals"};

  print("This is a shot on ${myMap['1']} Interpolation");

}

## **Car Assignment**

## **Installing Dart**

Please refer to the installation section.

## **Moving Code to IDE**

Please refer to the installation section.

## **Modularity**

[27]

The Dart language was designed by keeping the modules in mind. Modularity in Dart is realized through packages, libraries, and classes.

A **library** exposes functionality as a set of interfaces and hides the implementation from the rest of the world. As a concept, it's very similar to the separation of concern between objects in **object-oriented programming** (**OOP**). Separating an application into libraries helps minimize tight coupling and makes it easier to maintain the code. A library can be implemented as a simple function, a single class, several classes, or a collection of parts representing the entire API of a library. The Dart application is a library as well.

A **package** is simply a directory that contains a pubspec.yaml file and may include any number of libraries and resources. The pubspec.yaml file contains significant information about the package, its authors, and its dependencies on other packages.

 name: animation\_library

version: 0.1.0

author: Sergey Akopkokhyants

description: Animation library for Web application

dependencies:

  browser: any

## **Private / Public**

[8]

In Java, we can use public, protected, and private keywords to control the access scope for a property or method. However, Dart doesn't provide that kind of keywords. Instead, you can use \_ (underscore) at the start of the name to make a data member of a class becomes private.

In Dart, the privacy is at library level rather than class level. It means other classes and functions in the same library still have the access. So, a data member is either public (if not preceded by \_) or private (if preceded by \_)

**Usage on Fields**

For example, there is a library a.dart with a class named A. The class has two fields: first (public) and \_second (private).

a.dart

  class A {

    String first;

    String \_second;

  }

  void main() {

    A a = new A();

    a.first = 'New first';

    a.\_second = 'New second';

    print('${a.first}: ${a.\_second}');

  }

It works fine when \_second is accessed within the same library, even from a function (main) outside the class. What will happen if it's accessed outside the library? Take a look at the example below.

other.dart

import 'a.dart';

  void main() {

    A a = new A();

    a.first = 'New first';

    a.\_second = 'New second'; // The setter \_second is not defined for the class 'A'

    print('${a.first}: ${a.\_second}'); // The getter \_second is not defined for the class 'A'

  }

If you try to run the code, you will get compile error that the setter or getter is not defined for the class. You should have got warned if you are using IDE. If you need to access \_second outside a.dart, you have to create a getter and setter for it which must be non-private.

a.dart

  class A {

    String first;

    String \_second;

    String get second {

      return \_second;

    }

    void set second(String second) {

      this.\_second = second;

    }

  }

In the other file, instead of accessing the field directly, use the getter and setter.

other.dart

  import 'a.dart';

  void main() {

    A a = new A();

    a.first = 'New first';

    a.second = 'New second';

    print('${a.first}: ${a.second}');

  }

**Usage on Functions**

The same thing also applies on class function (which is usually referred to as method). A function with \_ modifier can only be accessed within the library.

a.dart

  class A {

    // Fields

    // Getters and setters

    void \_printFields() {

      print('First: $first, second: $\_second');

    }

  }

If you call that function outside the library.

other.dart

  import 'a.dart';

  void main() {

    A a = new A();

    a.first = 'New first';

    a.second = 'New second';

    a.\_printFields(); // The getter \_printFields is not defined for the class 'A'

  }

You'll get the similar error when compiling the code. Removing the \_ modifier, makes it becomes public which means you can access the function outside the library.

a.dart

   class A {

    // Fields

    // Getters and setters

    void printFields() {

      print('First: $first, second: $\_second');

    }

  }

other.dart

  import 'a.dart';

  void main() {

    A a = new A();

    a.first = 'New first';

    a.second = 'New second';

    a.printFields();

  }

**Inheritance**

The same rule also appllies for inheritance. For example, we have a class B which is the sub-class of A.

  class B extends A {

    printA() {

      \_printFields();

    }

  }

B can only access private fields and attributes of A only if it's placed in the same library. Moving the code above outside the library, it will cause error with reason the method is not defined for class B.

Tips: Don't use **\_** on local variable

The private concept is not for local variable. Therefore, avoid using \_ for local variables.

## **Getter Functions**

**What is a getter in Dart?**

A getter method is used to get and save a specific fieldName class in a variable. A default getter method exists in all classes, but it can be overridden explicitly. The get keyword can be used to define the getter method. Syntax: returnType get fieldName {}

## 

## **Getters and Setters**

[7]

Getters and setters are special methods that provide read and write access to an object’s properties. Each instance variable of your class has an implicit getter, and a setter if needed. In dart, you can take this even further by implementing your own getters and setters. If you've had any experience in Object-Oriented Programming you'll feel right at home. Let's get started.

In OOP a class acts as an Abstract Data Type(ADT) for an instance of that class(Object). In dart, this is also the case. The basic syntax for a class is:

class className {

 fields;

 getters/setters

 constructor

 methods/functions

}

The getters and setters can also be placed after the constructor. Now let's create a class and instantiate it.

class Vehicle {

  String make;

  String model;

  int manufactureYear;

  int vehicleAge;

  String color;

  int get age {

    return vehicleAge;

  }

  void set age(int currentYear) {

    vehicleAge = currentYear - manufactureYear;

  }

  // We can also eliminate the setter and just use a getter.

  //int get age {

  //  return DateTime.now().year - manufactureYear;

  //}

  Vehicle({this.make,this.model,this.manufactureYear,this.color,});

}

Age here is both a getter and a setter. Let's see how we can use it.

void main() {

 Vehicle car =

 Vehicle(make:"Honda",model:"Civic",manufactureYear:2010,color:"red");

  print(car.make); // output - Honda

  print(car.model); // output - Civic

  car.age = 2019;

  print(car.age); // output - 9

}

One of my favourite ways of using getters is getting a Map from an object.

void main() {

 Vehicle car = Vehicle(make:"Honda",model:"Civic",manufactureYear:2010,color:"red");

  print(car.map); // output - {make: Honda, model: Civic, manufactureYear: 2010, color: red}

}

class Vehicle {

  String make;

  String model;

  int manufactureYear;

  int vehicleAge;

  String color;

  Map<String,dynamic> get map {

    return {

      "make": make,

      "model": model,

      "manufactureYear":manufactureYear,

      "color": color,

    };

  }

  int get age {

    return DateTime.now().year - manufactureYear;

  }

  void set age(int currentYear) {

    vehicleAge = currentYear - manufactureYear;

  }

  Vehicle({this.make,this.model,this.manufactureYear,this.color,});

}

This covers basic usage of getter and setters in Dart.

## **Lists**

[9]

One of the most popular data structures in OOP is List. In this tutorial, we’ll show you many methods and functions to work with a List in Dart (also in [Flutter](https://bezkoder.com/tag/flutter/)). At the end, you’re going to know:

* Introduction to [Dart](https://bezkoder.com/tag/dart/) List
* How to create, initialize, access, modify, remove items in a List
* Ways to iterate, find, filter, transform items of a List in Dart/Flutter
* How to create List of objects in Dart/Flutter
* Ways to sort a List (of objects) in Dart/Flutter
* Initialize, iterate, flatten list of Lists

**Important points about Dart List**

These are some important information you should know before working with Dart List:

There are kinds of List: fixed-length list (list’s length cannot be changed) & growable list (size can be changed to accommodate new items or remove items)

Dart List is an ordered collection which maintains the insertion order of the items.

Dart List allows duplicates and null values.

While an operation on the list is being performed, modifying the list’s length (adding or removing items) will break the operation.

**Create a List in Dart/Flutter**

The example shows you:

How to create a List using List() constructor or literal syntax.

How to add new items to a List.

**Create fixed-length list in Dart/Flutter**

List<String> myList = List<String>(3);

myList[0] = 'one';

myList[1] = 'two';

myList[2] = 'three';

// myList.add('four');

/\* throw UnsupportedError

(Unsupported operation: Cannot add to a fixed-length list) \*/

print(myList);

**Output:**

[one, two, three]

Dart also allows literal syntax and null value:

var myList = List(3);

myList[0] = 'one';

myList[1] = 2;

myList[2] = null;

print(myList);

**Output:**

[one, 2, null]

**Create growable list in Dart/Flutter**

We can create growable list by not specify the length of the List:

List<int> myList = List<int>();

myList.add(42);

myList.add(2018);

print(myList);

print(myList.length);

myList.add(2019);

print(myList);

print(myList.length);

**Output:**

[42, 2018]

2

[42, 2018, 2019]

3

For growable list, Dart also allows literal syntax and null value:

var myList = List(); // var myList = [];

myList.add(42);

myList.add(null);

print(myList);

print(myList.length);

myList.add('year 2019');

print(myList);

print(myList.length);

**Output:**

[42, null]

2

[42, null, year 2019]

3

Dart/Flutter initialize List with values

The examples show you how to:

* initialize list in simple way using operator [].
* create and fill a list with specified value using [filled()](https://api.dartlang.org/stable/dart-core/List/List.filled.html) constructor.
* create a list containing all specified itemsusing [from()](https://api.dartlang.org/stable/dart-core/List/List.from.html) constructor.
* create a ‘const’ list using [unmodifiable()](https://api.dartlang.org/stable/dart-core/List/List.unmodifiable.html) constructor.
* create and fill a list with values by a generator function using [generate()](https://api.dartlang.org/stable/dart-core/List/List.generate.html) constructor.
* List<int> intList = [1, 2, 3];
* print(intList);
* var myList = ['one', 2];
* print(myList);

**Output:**

[1, 2, 3]

[one, 2]

// by default, growable: false

var fixedList = List.filled(3, 1);

print(fixedList);

// fixedList.add(42);

/\*

UnsupportedError (Unsupported operation: Cannot add to a fixed-length list)

\*/

var growableList = List.filled(3, 2, growable: true);

growableList.add(42);

print(growableList);

**Output:**

[1, 1, 1]

[2, 2, 2, 42]

// by default, growable: true

var fixedList = List.from([1, 2, 3], growable: false);

print(fixedList);

// fixedList.add(42);

/\*

UnsupportedError (Unsupported operation: Cannot add to a fixed-length list)

\*/

var growableList = List.from([1, 2, 3]);

growableList.add(42);

print(growableList);

**Output:**

[1, 2, 3]

[1, 2, 3, 42]

var unmodifiableList = List.unmodifiable([1, 2, 3]);

print(unmodifiableList);

// unmodifiableList.add(42);

/\*

UnsupportedError (Unsupported operation: Cannot add to an unmodifiable list)

\*/

**Output:**

[1, 2, 3]

// by default, growable: true

var myList = List.generate(5, (index) => index \* 2);

print(myList);

**Output**:

[0, 2, 4, 6, 8]

**Combine Lists in Dart/Flutter**

The examples show how to combine Lists using:

[from()](https://api.dartlang.org/stable/dart-core/List/List.from.html) and [addAll()](https://api.dartlang.org/stable/dart-core/List/addAll.html) method

expand() method

operator +

spread operator ... or null-aware spread operator ...?

var list1 = [1, 2, 3];

var list2 = [4, 5];

var list3 = [6, 7, 8];

// from() and addAll() method

var combinedList1= List.from(list1)..addAll(list2)..addAll(list3);

// expand() method

var combinedList2 = [list1, list2, list3].expand((x) => x).toList();

// operator +

var combinedList3  = list1 + list2 + list3;

// spread operator

var combinedList4 = [...list1, ...list2, ...list3];

**Output:**

[1, 2, 3, 4, 5, 6, 7, 8]

Now, what if there is one of 3 lists above is a null list:

var list1 = [1, 2, 3];

var list2 = null;

var list3 = [6, 7, 8];

If we use any methods above to combine these lists, the program will throw an Exception:  
– NoSuchMethodError: The getter 'iterator' was called on null.  
– or: NoSuchMethodError: The getter 'length' was called on null.

This is why null-aware spread operator ...? came to us. The operator check null list automatically with only one more ? symbol to be added:

var combinedList5 = [...?list1, ...?list2, ...?list3];

**Output:**

[1, 2, 3, 6, 7, 8]

**Access items from List in Dart/Flutter**

The examples show you how to:

find the size of a List using .length getter.

check if a List is empty or not using the getters: .isEmpty or .isNotEmpty. DON’T use .length.

access the item at specified index in a List using elementAt() method or operator [].

modify the item at specified index in a List using operator [].

get a group of items by specifying the range in List using [getRange()](https://api.dartlang.org/stable/dart-core/List/getRange.html) method.

get the first count items of a List using [take(count)](https://api.dartlang.org/stable/dart-core/Iterable/take.html)

var myList = [0, 'one', 'two', 'three', 'four', 'five'];

myList.isEmpty;                       // false

myList.isNotEmpty;                    // true

myList.length;                        // 6

myList.elementAt(2);                  // two

myList[2];                            // two

myList[myList.length - 1];            // five

myList[3] = 3;                        // myList: [0, one, two, 3, four, five]

myList.getRange(1, 3).toList();       // [one, two]

myList.take(3).toList()               // [0, one, two]

**Remove items from List in Dart/Flutter**

The examples show you how to:

remove the item at a given index in a List | [removeAt(int index)](https://api.dartlang.org/stable/dart-core/List/removeAt.html)

remove a item from a List by its value | [remove(Object value)](https://api.dartlang.org/stable/dart-core/List/remove.html)

remove all the items that match a given condition | [removeWhere(bool test)](https://api.dartlang.org/stable/dart-core/List/removeWhere.html)

remove all the items in the range of a List | [removeRange(int start, int end)](https://api.dartlang.org/stable/dart-core/List/removeRange.html)

clear a List | [clear()](https://api.dartlang.org/stable/dart-core/List/clear.html)

var myList = [0, 'one', 'two', 'three', 'four', 'five'];

// remove the item at index '3'

myList.removeAt(3);

/\* myList:

[0, one, two, four, five]

\*/

// remove() returns false if the item does not exist in the List

bool isRemoved = myList.remove('three');

/\* isRemoved:

false

\*/

bool isRemoved4thItem = myList.remove('four');

/\* isRemoved4thItem :

true

myList:

[0, one, two, five]

\*/

// remove all items which length > 3

myList.removeWhere((item) => item.toString().length > 3);

/\* myList:

[0, one, two]

\*/

// remove all items in the List

myList.clear();

/\* myList:

[]

\*/

var anotherList = [0, 'one', 'two', 'three', 'four', 'five'];

// remove items from index 2 to 4

anotherList.removeRange(2, 5);

/\* myList:

[0, one, five]

\*/

**Update List item in Dart/Flutter**

You can also update one or some items in a List using:

the item’s index

[replaceRange()](https://api.dartlang.org/stable/dart-core/List/replaceRange.html) method to remove the objects in a range, then insert others

var myList = [0, 'one', 'two', 'three', 'four', 'five'];

// replace the item at index '3'

myList[3] = 3;

/\* myList:

[0, one, two, 3, four, five]

\*/

// replace the item at index '1'

myList.replaceRange(1, 2, [1]);

/\* myList:

[0, 1, two, 3, four, five]

\*/

// replace the items from index 2 to 4

myList.replaceRange(2, 5, ['new 2', '3 and 4']);

/\* myList:

[0, 1, new 2, 3 and 4, five]

\*/

**Iterate over List in Dart/Flutter**

The examples show you how to iterate over a Dart List using:

* forEach() and lambda expression.
* [iterator](https://api.dartlang.org/stable/dart-core/Iterable/iterator.html) property to get Iterator that allows iterating.
* [every()](https://api.dartlang.org/stable/dart-core/Iterable/every.html) method
* simple for-each loop
* for loop with item index
* var myList = [0, 'one', 'two', 'three', 'four', 'five'];
* // use forEach()
* myList.forEach((item) => print(item));
* // or
* myList.forEach(print);
* // use iterator
* var listIterator = myList.iterator;
* while (listIterator.moveNext()) {
* print(listIterator.current);
* }
* // use every()
* myList.every((item) {
* print(item);
* return true;
* });
* // simple for-each
* for (var item in myList) {
* print(item);
* }
* // for loop with item index
* for (var i = 0; i < myList.length; i++) {
* print(myList[i]);
* }

**Dart/Flutter find elements in List**

The examples show how to:

* check if a List contains an element or not | [contains()](https://api.dartlang.org/stable/dart-core/Iterable/contains.html)
* find the index of the first occurrence of an element | [indexOf()](https://api.dartlang.org/stable/dart-core/List/indexOf.html)
* find the index of the last occurrence of an element | [lastIndexOf()](https://api.dartlang.org/stable/dart-core/List/lastIndexOf.html)
* find the index of the first occurrence of an element that matches a condition | [indexWhere()](https://api.dartlang.org/stable/dart-core/List/indexWhere.html)
* find the index of the last occurrence of an element that matches a condition | [lastIndexWhere()](https://api.dartlang.org/stable/dart-core/List/lastIndexWhere.html)

var myList = [0, 2, 4, 6, 8, 2, 8];

myList.contains(2);                        // true

myList.contains(5);                        // false

myList.indexOf(2);                         // 1

myList.lastIndexOf(2);                     // 5

myList.indexWhere((item) => item > 5);     // 3

myList.lastIndexWhere((item) => item > 5); // 6

**Filter items for a List in Dart/Flutter**

The examples show how to:

* filter all items in List that match the condition | [where()](https://api.dartlang.org/stable/dart-core/Iterable/where.html)
* get the first item in List that matches the condition | [firstWhere()](https://api.dartlang.org/stable/dart-core/Iterable/firstWhere.html)
* get the last item in List that matches the condition | [lastWhere()](https://api.dartlang.org/stable/dart-core/Iterable/lastWhere.html)
* var myList = [0, 2, 4, 6, 8, 2, 7];
* myList.where((item) => item > 5).toList();   // [6, 8, 7]
* myList.firstWhere((item) => item > 5);       // 6
* myList.lastWhere((item) => item > 5);        // 7

**Dart/Flutter List every**

We can verify if all items in a List satisfy a condition using [every()](https://api.dart.dev/stable/dart-core/Iterable/every.html) method.

var intList = [5, 8, 17, 11];

if (intList.every((n) => n > 4)) {

  print('All numbers > 4');

}

**Dart/Flutter List any**

We can verify if at least one item in a List satisfies a condition using [any()](https://api.dart.dev/stable/dart-core/Iterable/any.html) method.

var intList = [5, 8, 17, 11];

if (intList.any((n) => n > 10)) {

  print('At least one number > 10');

}

**Dart/Flutter List map items into new List**

We can map each item in a Dart List to new form using [map()](https://api.dartlang.org/stable/dart-core/Iterable/map.html) method:

var myList = ['zero', 'one', 'two', 'three', 'four', 'five'];

var uppers = myList.map((item) => item.toUpperCase()).toList();

/\*

myList: [zero, one, two, three, four, five]

uppers: [ZERO, ONE, TWO, THREE, FOUR, FIVE]

\*/

**User defined objects List in Dart/Flutter**

In Dart, we can create a List of any type, from int, double, String, to complex types like a List, Map, or any user defined objects.

The example show how to create a List of user defined object:

class Customer {

  String name;

  int age;

  Customer(this.name, this.age);

  @override

  String toString() {

    return '{ ${this.name}, ${this.age} }';

  }

}

main() {

  List customers = [];

  customers.add(Customer('Jack', 23));

  customers.add(Customer('Adam', 27));

  customers.add(Customer('Katherin', 25));

  print(customers);

  print(customers.length);

}

**Output:**

[{ Jack, 23 }, { Adam, 27 }, { Katherin, 25 }]

3

**Dart/Flutter List collection if and collection for**

With the collection if and collection for, we can dynamically create lists using conditionals (if) and repetition (for).

var mobile = true;

var web = false;

var tringList = ['kotlin', 'dart', if (mobile) 'flutter', if (web) 'react'];

// [kotlin, dart, flutter]

var intList = [for (var i = 1; i < 10; i++) i];

// [1, 2, 3, 4, 5, 6, 7, 8, 9]

var evenList = [

  for (var i = 1; i < 10; i++)

    if (i % 2 == 0) i

];

// [2, 4, 6, 8]

**Sort List in Dart/Flutter**

The examples show you how to:

* sort a List using [sort()](https://api.dartlang.org/stable/dart-core/List/sort.html) method.
* sort a List of objects using custom compare function.
* sort a List of objects by extending Comparable abstract class.

**Sort List using sort() method**

var intList = [0, 5, 2, 3, 8, 17, 11];

intList.sort();

print(intList);

var tringList = ['vue', 'kotlin','dart', 'angular', 'flutter'];

tringList.sort();

print(tringList);

**Output:**

[0, 2, 3, 5, 8, 11, 17]

[angular, dart, flutter, kotlin, vue]

**Sort a List of objects in Dart/Flutter**

For more details, please visit: [Dart/Flutter – Sort list of Objects](https://www.bezkoder.com/dart-sort-list-of-objects/)

- use custom compare function:

class Customer {

  String name;

  int age;

  Customer(this.name, this.age);

  @override

  String toString() {

    return '{ ${this.name}, ${this.age} }';

  }

}

main() {

  List customers = [];

  customers.add(Customer('Jack', 23));

  customers.add(Customer('Adam', 27));

  customers.add(Customer('Katherin', 25));

  customers.sort((a, b) => a.age.compareTo(b.age));

  print('Sort by Age: ' + customers.toString());

  customers.sort((a, b) => a.name.compareTo(b.name));

  print('Sort by Name: ' + customers.toString());

}

**Output:**

Sort by Age: [{ Jack, 23 }, { Katherin, 25 }, { Adam, 27 }]

Sort by Name: [{ Adam, 27 }, { Jack, 23 }, { Katherin, 25 }]

- The second approach is to extend Comparable abstract class and override compareTo() method. Now we don't need to pass compare function, we just call list.sort() instead of list.sort(compare).

class Customer extends Comparable {

  String name;

  int age;

  Customer(this.name, this.age);

  @override

  String toString() {

    return '{ ${this.name}, ${this.age} }';

  }

  // sort by Name (asc), then age (desc)

  @override

  int compareTo(other) {

    int nameComp = this.name.compareTo(other.name);

    if (nameComp == 0) {

      return -this.age.compareTo(other.age); // '-' for descending

    }

    return nameComp;

  }

}

main() {

 List customers = [];

  customers.add(Customer('Jack', 23));

  customers.add(Customer('Adam', 27));

  customers.add(Customer('Katherin', 25));

  customers.add(Customer('Jack', 32));

  customers.sort();

  print(customers);

}

**Output:**

[{ Adam, 27 }, { Jack, 32 }, { Jack, 23 }, { Katherin, 25 }]

**Dart/Flutter List reverse**

We can reverse a list using [reversed](https://api.dart.dev/stable/dart-core/List/reversed.html) property. It returns an Iterable of the objects in the list.

var tringList = ['vue', 'kotlin','dart', 'angular', 'flutter'];

var reversed = List.of(tringList.reversed);

// [flutter, angular, dart, kotlin, vue]

**Dart/Flutter List of Lists**

**Initialize List of Lists**

We will:

* initialize list of existing lists using operator [].
* create and fill list of lists with values by a generator function using [generate()](https://api.dartlang.org/stable/dart-core/List/List.generate.html) constructor.
* var list1 = [1, 2];
* var list2 = [3, 4];
* var list3 = [5, 6];
* var listOfLists = [list1, list2, list3];
* // [[1, 2], [3, 4], [5, 6]]
* List> listOfNumbers =
* List.generate(3, (i) => [i \* 2 + 1, i \* 2 + 2]);
* // [[1, 2], [3, 4], [5, 6]]

**Iterate List of Lists**

The examples show you how to iterate over a Dart List using:

* forEach() and lambda expression.
* every() method
* simple for-each loop
* for loop with item index

var listOfNumbers = [[1, 2], [3, 4, 5], [6, 7, 8]];

listOfNumbers.forEach((nums) => nums.forEach((number) => print(number)));

listOfNumbers.every((nums) {

  nums.forEach((number) => print(number));

  return true;

});

for (var nums in listOfNumbers) {

  for (var number in nums) {

    print(number);

  }

}

for (var i = 0; i < listOfNumbers.length; i++) {

  for (var j = 0; j < listOfNumbers[i].length; j++) {

    print(listOfNumbers[i][j]);

  }

}

/\* Result:

1

2

3

4

5

6

7

8

\*/

**Flatten List of Lists**

The examples show you how to flatten a Dart list of lists using:

* combination of forEach() and addAll() method.
* Iterator [expand()](https://api.dartlang.org/stable/dart-core/Iterable/expand.html) method.
* var listOfNumbers = [[1, 2], [3, 4, 5], [6, 7, 8]];
* var flattenList1 = [];
* listOfNumbers
* .forEach((nums) => nums.forEach((number) => flattenList1.add(number)));
* // [1, 2, 3, 4, 5, 6, 7, 8]
* var flattenList2 = [];
* flattenList2 = listOfNumbers.expand((number) => number).toList();
* // [1, 2, 3, 4, 5, 6, 7, 8]

**Conclusion**

In this tutorial, we've learned overview of a Dart List, how to create a List, how to add, modify and remove items from a List, how to iterate over a List, how to combine Lists, transform, find, filter, sort items in a List along with [List of objects](https://www.bezkoder.com/dart-sort-list-of-objects/).

## **List Function**

## **List Assignment**

## **Maps Start**

[9]

Dart Map is an object that stores data in the form of a key-value pair. Each value is associated with its key, and it is used to access its corresponding value. Both keys and values can be any type. In Dart Map, each key must be unique, but the same value can occur multiple times. The Map representation is quite similar to Python Dictionary. The Map can be declared by using curly braces {} ,and each key-value pair is separated by the commas(,). The value of the key can be accessed by using a square bracket([]).

**Declaring a Dart Map**

Dart Map can be defined in two methods.

* Using Map Literal
* Using Map Constructor

The syntax of declaring Dart Map is given below.

**Using Map Literals**

To declare a Map using map literal, the key-value pairs are enclosed within the curly braces "{}" and separated by the commas. The syntax is given below.

**Syntax –**

var map\_name = {key1:value1, key2:value2 [.......,key\_n: value\_n]}

**Example - 1:**

void main() {

   var student = {'name':'Tom','age':'23'};

   print(student);

}

**Output:**

{name: Tom, age: 23}

**Example - 2: Adding value at runtime**

void main() {

   var student = {'name':' tom', 'age':23};

   student['course'] = 'B.tech';

   print(student);

}

**Output:**

{name: tom, age: 23, course: B.tech}

**Explanation -**

In the above example, we declared a Map of a **student** name. We added the value at runtime by using a square bracket and passed the new key as a **course** associated with its value.

### **Using Map Constructor**

To declare the [Dart](https://www.javatpoint.com/dart-programming) Map using map constructor can be done in two ways. First, declare a map using **map()** constructor. Second, initialize the map. The syntax is given below.

**Syntax -**

var map\_name = new map();

After that, initialize the values.

map\_name[key] = value

**Example - 1: Map constructor**

void main() {

   var student = new Map();

   student['name'] = 'Tom';

   student['age'] = 23;

   student['course'] = 'B.tech';

   student['Branch'] = 'Computer Science';

   print(student);

}

**Output:**

{name: Tom, age: 23, course: B.tech, Branch: Computer Science}

#### **Note - A map value can be any object including NULL.**

**Map Properties**

The dart:core:package has Map class which defines following properties.

|  |  |
| --- | --- |
| **Properties** | **Explanation** |
| Keys | It is used to get all keys as an iterable object. |
| values | It is used to get all values as an iterable object. |
| Length | It returns the length of the Map object. |
| isEmpty | If the Map object contains no value, it returns true. |
| isNotEmpty | If the Map object contains at least one value, it returns true. |

**Example -**

void main() {

   var student = new Map();

   student['name'] = 'Tom';

   student['age'] = 23;

   student['course'] = 'B.tech';

   student['Branch'] = 'Computer Science';

   print(student);

  // Get all Keys

  print("The keys are : ${student.keys}");

 // Get all values

 print("The values are : ${student.values}");

 // Length of Map

 print("The length is : ${student.length}");

//isEmpty function

print(student.isEmpty);

//isNotEmpty function

print(student.isNotEmpty);

}

**Output:**

{name: Tom, age: 23, course: B.tech, Branch: Computer Science}

The keys are : (name, age, course, Branch)

The values are : (Tom, 23, B.tech, Computer Science)

The length is : 4

false

true

**Map Methods**

The commonly used methods are given below.

**addAll() -** It adds multiple key-value pairs of other. The syntax is given below.

**Syntax -**

Map.addAll(Map<Key, Value> other)

**Parameter:**

* **other -** It denotes a key-value pair. It returns a void type.

Let's understand the following example.

**Example -**

void main() {

   Map student = {'name':'Tom','age': 23};

   print('Map :${student}');

   student.addAll({'dept':'Civil','email':'tom@xyz.com'});

   print('Map after adding  key-values :${student}');

}

**Output:**

Map :{name: Tom, age: 23}

Map after adding  key-values :{name: Tom, age: 23, dept: Civil, email: tom@xyz.com}

**remove()** - It eliminates all pairs from the map. The syntax is given below.

**Syntax -**

Map.clear()

Let's have a look at following example.

**Example -**

void main() {

   Map student = {'name':'Tom','age': 23};

   print('Map :${student}');

   student.clear();

   print('Map after removing all key-values :${student}');

}

**Output:**

Map :{name: Tom, age: 23}

Map after removing all key-values :{}

**remove() -** It removes the key and its associated value if it exists in the given map. The syntax is given below.

**Syntax -**

Map.remove(Object key)

**Parameter -**

* **Keys -** It deletes the given entries. It returns the value associated with the specified key.

Let's understand the following example.

**Example -**

void main() {

   Map student = {'name':'Tom','age': 23};

   print('Map :${student}');

   student.remove('age');

   print('Map after removing given key :${student}');

}

**Output:**

Map :{name: Tom, age: 23}

Map after removing given key :{name: Tom}

**forEach() -** It is used to iterate the Map's entries. The syntax is given below.

**Syntax -**

Map.forEach(void f(K key, V value));

</pre></div>

<p><strong>Parameter -</strong></p>

<ul class="points">

<li><strong>f(K key, V value) -</strong> It denotes the key-value pair of the map.</li>

</ul>

<p>Let's understand the following example.</p>

<p><strong>Example -</strong></p>

<div class="codeblock"><textarea name="code" class="java">

void main() {

   Map student = {'name':'Tom','age': 23};

   print('Map :${student}');

   student.forEach((k,v) => print('${k}: ${v}'));

}

**Output:**

Map :{name: Tom, age: 23}

name: Tom

age: 23

For more info please visit:

* <https://zetcode.com/dart/map/>
* <https://www.tutorialspoint.com/dart_programming/dart_programming_map.htm>

## **More about Maps**

## **Iterations Intro**

Iteration refers to the process in which the code segment is executed once. One iteration refers to 1-time execution of a loop. A loop can undergo many iterations.

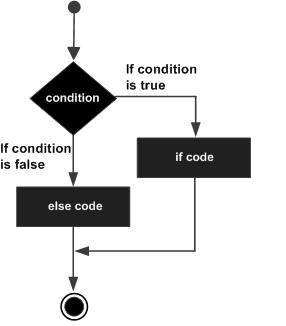
## **If Statement Intro**

[11]

In Dart, if-else statement is used to execute one of the two blocks: if-block or else-block; based on the result of a given condition.

If the Boolean expression evaluates to be true, then the if block of code will be executed, otherwise else block of code will be executed.

The following illustration shows the flowchart of the if…else statement.



The **if** block guards the conditional expression. The block associated with the **if** statement is executed if the Boolean expression evaluates to true. The **if** block may be followed by an optional **else** statement. The instruction block associated with the **else** block is executed if the expression evaluates to false.

## **More Ifs**

[12]

Dart If-Else statement contains two blocks: if-block and else-block.

If the boolean\_expression next to if keyword evaluates to true, then the code inside if-block is executed.

If the boolean\_expression next to if keyword evaluates to false, then the code inside else-block is executed.

**Syntax**

The syntax of if-else statement in Dart is given in the following.

if (boolean\_expression) {

     //if block statement(s)

 } else {

     //else block statement(s)

 }

where boolean\_expression evaluates to or promotes to boolean value true or false.

**Example**

In the following program, we take an integer value in x, and check if the number in x is even or odd using if-else statement.

void main(){

    int x = 13;

    if(x%2==0){

        print('$x is even number.');

    } else{

        print('$x is odd number.');

    }

}

**Output**

13 is odd number.

## **More about Iterations**

[14]

**General loop structure**

There are two types of loops:

1. **Definite Loops**: These refer to loops where we know the number of times we want to execute the code.
2. **Indefinite Loops**: These refer to loops where we do not know the number of times we want to execute the code.

All the loop types mentioned above have the following characteristics:

* Counter Variable - Also known as the Initializer. It keeps track of the number of times a loop is executed.
* Increment or Decrement Counter Variable - This refers to the number in which counter variable increases or decreases after each iteration.
* Condition Check - Every loop will have a condition that will be checked on each Iteration. If the condition is evaluated to true, then the next iteration will get executed.

Loops can be classified into two types based on condition checking:

1. Entry controlled loops - These are loops in which the condition is checked first and then the iteration is executed.
2. Exit controlled loops - These are loops in which an iteration is executed and then the condition is checked afterwards.

**Loops in Dart**

The syntax of Dart loops is similar to the ones in Java Programming Language.

There are three loops in Dart:

**For loop**

**Syntax of For loop:**

void main(){

for (initialize counter variable; condition; increment/decrement){

//put your code here;

}

}

**How the For loop works:**

**Initialize counter - Condition check - Execute code - Increment**.

1. The counter variable is initialized. The initialization occurs only once, it won’t be executed on every iteration.
2. Then the condition is checked. If the condition is evaluated as true, then the code block will be executed. If not, the code block will not be executed.
3. The counter variable will be incremented and the condition is evaluated again with the new value of the counter variable. This process repeats itself until the condition isn’t met.

***Example:***

void main(){

for(var i = 0; i<4; i++){

print ("Hello");

}

}

**Output :**

Hello

Hello

Hello

Hello

Here the value of the counter variable will increase by 1 for every iteration. If the value of i becomes equal to 4 or more, the loop will be terminated since the condition states that the value of i should be less than 4. The code block prints Hello, hence the output is Hello written four times for (0,1,2,3) < 4.

**While loop**

**Syntax of a While loop:**

void Main(){

// initialize counter variable;

while(condition){

//code;

//increment/decrement;

}

}

**How the While loop works:**

**Initialize counter - Condition check - Execute code - Increment**.

1. We Initialize the counter variable outside the while block.
2. It begins with the condition check, if it’s true, then the code block will be executed.
3. The value of the counter variable is incremented or decremented, and the new value of the counter variable goes through the condition check.
4. This process will continue until the condition check is false.

NOTE: We must always increment/decrement the counter variable inside the while block. Failure to do so will create a never-ending loop.

***Example:***

void main(){

int i = 0;

while(i<3){

print("Hello");

i++;

}

}

**Output:**

Hello

Hello

Hello

Our condition check will begin with the statement 0 < 3 which is true. Therefore the code executes and prints out the first hello. Then, the value of i increases by one to 1.

Now, the condition 1 < 3 is still true, so the code executes. This goes on until the value of the counter variable is 3. Once the condition is 3 < 3 which is false. The loop terminates and three Hellos get printed.

**Do-While Loop**

**Syntax of Do-While Loop:**

void main(){

// initialize counter variable

do {

// code;

// increment/decrement;

} while(condition);

}

**How the Do-While loop works:**

**Initialize counter - Execute code - Increment - Condition check**.

1. The counter variable will be initialized outside the do-while block and the code block will be executed regardless of the condition for the first time.
2. Then, the condition will be checked. If it’s evaluated to true, the loop will proceed to the next iteration.
3. The iterations will continue until the condition is evaluated as false.

Note: The code block in the do-while loop is executed at least once.

***Example:***

void main(){

int i = 0;

do {

print ("Hello");

i++;

} while (i<5);

}

**Output**:

Hello

Hello

Hello

Hello

Hello

The code is executed for the first time and then the counter variable is incremented by one. The loop then checks the condition 1 < 5. Since this is true, the loop will proceed to the next iteration.

In the next iteration, the counter variable is incremented by one. The loop then checks the condition 2 < 5. Since this is true, the loop will proceed to the next iteration. This will continue until the value of the counter variable is 5. This time the condition check 5 < 5 will be evaluated as false, so the loop ends after printing 5 Hellos.

**Other key concepts**

**Break keyword**

We use the break keyword to forcefully end a loop. It comes in handy when you want a partial output or to terminate the loop when a certain condition is met.

***Example:***

void main(){

for (int i = 1; i<=10; i++){

print ("Hello");

if(i >= 6){

break;

}

}

}

**Output:**

Hello

Hello

Hello

Hello

Hello

Hello

This code will terminate the loop after printing six Hellos because of the break statement.

**Continue keyword**

We use the continue keyword to skip some code and proceed to the next iteration.

***Example:***

void main(){

for (int i = 1; i<=8; i++){

if(i == 5){

continue;

}

print (i);

}

}

**Output:**

1

2

3

4

6

7

8

In the example, when i becomes 5, the loop is forced to the next iteration, thus skipping the print statement.

**Applications of loops**

Here are some positive and negative applications of loops:

* **Writing a Music Player**: A music player has a loop that enables it to play from one song to the next systematically.
* **Cycling through values**: Loops are used to print out values in an extensive list. For example, the names of students in a school.

void main(){

List grade\_one\_students = ["John", "Mark", "Alex"];

for(String student in grade\_one\_students){

print(student);

}

}

**Output:**

John

Mark

Alex

It loops over all the elements stored in the list and prints them out.

* **Create computer viruses**: Malicious programmers use loops to create [computer viruses](https://en.wikipedia.org/wiki/Computer_virus) by using the While loop and embedding it to software downloads. If we write a while loop without the increment or decrement counter variable, it creates a never-ending loop which leads to unnecessary storage and memory consumption on devices. This causes the devices to crash or behave abnormally.

***Caution: DO NOT RUN THIS CODE!!!***

void main(){

int i = 0;

while(i<3){

print("Hello");

}

}

The code will print out an endless string of the word Hello. Note that we are not incrementing the counter variable.

**Conclusion**

Loops and iteration may seem complicated and a bit challenging to new developers while learning a new language. The goal of this article was to make it easier to understand them in Dart programming language. Practice will increase your skill and understanding of these loops.

## **Iterations Final**

## **Repository Assignment**

## **Cascade Operator**

[14]

The **cascade notation** (**. .**) in Dart allows you to make a sequence of operations on the same object (including function calls and field access). This notation helps keep Dart code compact and removes the need to create temporary variables to store data.

**Code**

In the example below, we have created an Example class with two methods:

* bSetter
* printValues

In line 24-27, we are using dart cascade notation for operations on Example class object eg1.

import 'dart:convert';

//An Example class with member attributes and methods

class Example{

  var a;

  var b;

  void bSetter(b)

  {

    this.b = b;

  }

  void printValues(){

    print(this.a);

    print(this.b);

  }

}

void main() {

    //Instantiating two Example objects

    Example eg1 = new Example();

    Example eg2 = new Example();

    //Using the .. operator for operations on Example object

    print("Example 1 results:");

    eg1

    ..a = 88

    ..bSetter(53)

    ..printValues();

    //The same operations as above but without the .. operator

    print("Example 2 results:");

    eg2.a = 88;

    eg2.bSetter(53);

    eg2.printValues();

}

## **Cascade notation: part 2**

[15]

Cascade notation is used when we want to operate a sequence of operations on the same object. The cascade notation is denoted by the (..) symbol.

It is similar to method chaining that we have in other programming languages and it does save us plenty of steps and need of temporary variable.

**Example**

Consider the following example for a representation of how the cascade notation works in Dart.

class Sample{

   var a;

   var b;

   void showA(x){

      this.a = x;

   }

   void showB(y){

      this.b = y;

   }

   void printValues(){

      print(this.a);

      print(this.b);

   }

}

void main(){

   Sample sampleOne = new Sample();

   sampleOne.showA(2);

   sampleOne.showB(3);

   sampleOne.printValues();

   Sample sampleTwo = new Sample();

   sampleTwo..showA(2)

   ..showB(3)

   ..printValues();

}

In the above example, we have two objects created of a single class and we are calling three methods that are present in the above class. When we call the three methods on the object instance sampleOne, we are explicitly typing the name of the object three times followed by the dot notation and the call to methods.

In the second case, the sampleTwo is written only once and we made use of the cascade operator to call the methods we want.

It should be noted that we can't have semicolons(;) between the consecutive method calls as it will confuse the compiler and we will get an error.

**Output**

2

3

2

3

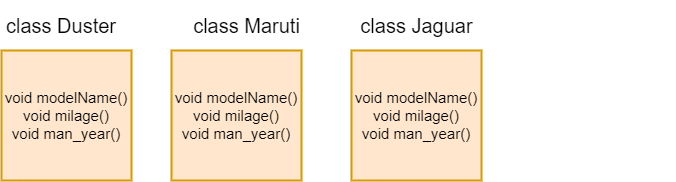
## **Inheritance part 1**

[16]

Dart inheritance is defined as the process of deriving the properties and characteristics of another class. It provides the ability to create a new class from an existing class. It is the most essential concept of the oops(Object-Oriented programming approach). We can reuse the all the behavior and characteristics of the previous class in the new class.

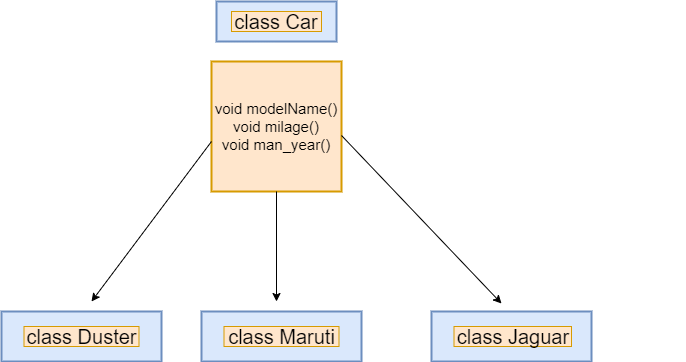
* **Parent Class -** A class which is inherited by the other class is called **superclass** or **parent class**. It is also known as a **base class**.
* **Child Class -** A class which inherits properties from other class is called the child class. It is also known as the **derived class** or **subclass**.

Suppose we have a fleet of cars, and we create three classes as Duster, Maruti, and Jaguar. The methods modelName(), milage(), and man\_year() will be same for all of the three classes. By using the inheritance, we don't need to write these functions in each of the three classes.



As you can see in the above figure, if we create class Car and write the common function in each of the classes. Then, it will increase duplication and data redundancy in the program. The inheritance is used to avoid this type of situation.

We can avoid data redundancy by defining the class Car with these functions in it and inheriting in the other classes from the Car class. It enhances the re-usability of code. We just need to write function one time instead of multiple times. Let's have a look at the following image.



The syntax is given below.

**Syntax -**

class child\_class extends parent\_class {

    //body of child class

}

The child class inherits functions and variables, or properties of parent class using the extends keyword. It cannot inherit the parent class constructor; we will discuss this concept later.

## **Inheritance part 2**

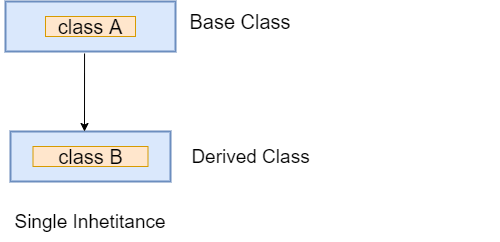
**Types of Inheritance**

The inheritance can be mainly four types. These are given below.

* Single Inheritance
* Multiple Inheritance
* Multilevel Inheritance
* Hierarchical Inheritance

**Single Level Inheritance**

In the single inheritance, a class is inherited by a single class or subclass is inherited by one parent class. In the following example, we create Person which inherits Human class.



Let's understand the following example.

**Example -**

class Bird{

      void fly()

         {

            print("The bird can fly");

          }

   }

      // Inherits the super class

class Parrot extends Bird{

         //child class function

         void speak(){

             print("The parrot can speak");

                 }

}

void main() {

      // Creating object of the child class

      Parrot p=new Parrot();

      p.speak();

      p.fly();

}

**Output**

The parrot can speak

The bird can fly

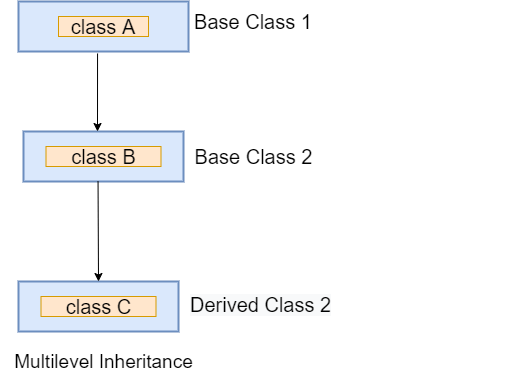
**Explanation:**

In the above code, we create parent class Bird and declared the fly() function in it. Then, we created the child class called Parrot, which inherited the parent class's property using the extends keyword. The child class has its own function speak().

Now the child class has two functions fly() and speak(). So we created the object of child class and access both functions. It printed the result to the console.

**Multilevel Inheritance**

In the multiple inheritance, a subclass is inherited by another subclass or creates the chaining of inheritance. Let's understand the following example.



**Example -**

class Bird{

      void fly()

         {

            print("The bird can fly");

          }

   }

      // Inherits the super class

class Parrot extends Bird{

         void speak(){

             print("The parrot can speak");

                 }

}

// Inherits the Parror base class

class Eagle extends Parrot {

          void vision(){

             print("The eagle has a sharp vision");

                 }

}

void main() {

      // Creating object of the child class

      Eagle e=new Eagle();

      e.speak();

      e.fly();

      e.vision();

}

**Output**

The parrot can speak

The bird can fly

The eagle has a sharp vision

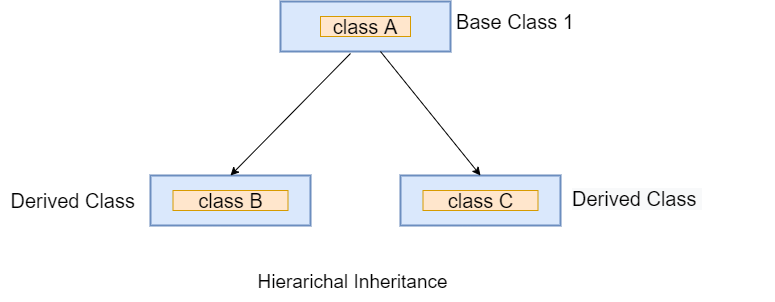
**Explanation:**

In the above example, we created another new class Eagle and inherited the Parrot class. Now the parrot is the parent class of Eagle, and class Eagle acquired all functions of both parent classes. We created the object of the child class and accessed all properties. It printed the output to the screen.

#### **Note - Dart doesn't support multiple inheritance because it creates complexity in the program.**

**Hierarchical Inheritance**

In the hierarchical inherence, two or more classes inherit a single class. In the following example, the two-child classes Peter and James inherit the Person class.



**Example -**

// Parent Class

class Person {

  void dispName(String name) {

    print(name);

  }

  void dispAge(int age) {

    print(age);

  }

}

class Peter extends Person {

  void dispBranch(String nationality) {

    print(nationality);

  }

}

//Derived class created from another derived class.

class James extends Person {

          void result(String result){

              print(result);

}

}

void main() {

      // Creating Object of James class

      James j = new James();

      j.dispName("James");

      j.dispAge(24);

      j.result("Passed");

    // Creating Object of Peter class

      Peter p = new Peter();

      p.dispName("Peter");

      p.dispAge(21);

      p.dispBranch("Computer Science");

}

**Output**

James

24

Passed

Peter

21

Computer Science

## **Debugging**

[17]

**Debugging Flutter apps**

There’s a wide variety of tools and features to help debug Flutter applications. Here are some of the available tools:

* [DevTools](https://docs.flutter.dev/development/tools/devtools), a suite of performance and profiling tools that run in a browser.
* [Android Studio/IntelliJ](https://docs.flutter.dev/development/tools/android-studio#run-app-with-breakpoints), and [VS Code](https://docs.flutter.dev/development/tools/vs-code#run-app-with-breakpoints) (enabled with the Flutter and Dart plugins) support a built-in source-level debugger with the ability to set breakpoints, step through code, and examine values.
* [Flutter inspector](https://docs.flutter.dev/development/tools/devtools/inspector), a widget inspector available in DevTools, and also directly from Android Studio and IntelliJ (enabled with the Flutter plugin). The inspector allows you to examine a visual representation of the widget tree, inspect individual widgets and their property values, enable the performance overlay, and more.

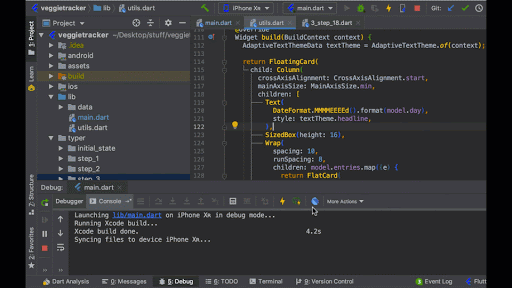
**DevTools**

For debugging and profiling apps, DevTools might be the first tool you reach for. DevTools runs in a browser and supports a variety of features:

* source-level debugger
* widget inspector that displays a visual widget tree, and “widget select” mode where you select a widget in the app and it drills down to that widget in the tree
* memory profiler
* timeline view that supports tracing, and importing and exporting trace information
* logging view

If you run your application in [debug mode](https://docs.flutter.dev/testing/build-modes#debug) or [profile mode](https://docs.flutter.dev/testing/build-modes#profile), while it’s running you can open DevTools in the browser to connect to your app. DevTools doesn’t work well with an app compiled to [release mode](https://docs.flutter.dev/testing/build-modes#release), as the debugging and profiling information has been stripped away.

If you use DevTools for profiling, make sure to run your application in [profile mode](https://docs.flutter.dev/testing/build-modes#profile). Otherwise, the main output that appears on your profile are the debug asserts verifying the framework’s various invariants (see [Debug mode assertions](https://docs.flutter.dev/testing/debugging#debug-mode-assertions)).



For more information, see the [DevTools](https://docs.flutter.dev/development/tools/devtools) documentation.

**Setting breakpoints**

You can set breakpoints directly in your IDE/editor (such as [Android Studio/IntelliJ](https://docs.flutter.dev/development/tools/android-studio#run-app-with-breakpoints) and [VS Code](https://docs.flutter.dev/development/tools/vs-code#run-app-with-breakpoints)), in the [DevTools debugger](https://docs.flutter.dev/development/tools/devtools/debugger), or [programmatically](https://docs.flutter.dev/testing/code-debugging#setting-breakpoints).

**The Dart analyzer**

If you’re using a [Flutter enabled IDE/editor](https://docs.flutter.dev/get-started/editor), the Dart analyzer is already checking your code and looking for possible mistakes.

If you run from the command line, test your code with flutter analyze.

The Dart analyzer makes heavy use of type annotations that you put in your code to help track problems down. You are encouraged to use them everywhere (avoiding var, untyped arguments, untyped list literals, and so on) as this is the quickest and least painful way of tracking down problems.

For more information, see [Using the Dart analyzer](https://github.com/flutter/flutter/wiki/Using-the-Dart-analyzer).

**Logging**

Another useful debugging tool is logging. You set logging up [programmatically](https://docs.flutter.dev/testing/code-debugging#logging) then view the output in the DevTools [logging view](https://docs.flutter.dev/development/tools/devtools/logging), or in the console.

**Debugging application layers**

Flutter was designed with a layered architecture that includes widget, rendering, and painting layers. For links to more information and videos, see [The Framework architecture](https://github.com/flutter/flutter/wiki/The-Framework-architecture) on the [GitHub wiki](https://github.com/flutter/flutter/wiki/), and the community article, [The Layer Cake](https://medium.com/flutter-community/the-layer-cake-widgets-elements-renderobjects-7644c3142401).

The Flutter widget inspector provides a visual representation of the widget tree, but if you want a greater level of detail, or you want a verbose text-based dump of the widget, layer, or render trees, see [Debug flags: application layers](https://docs.flutter.dev/testing/code-debugging#debug-flags-application-layers) in the [Debugging Flutter apps programmatically](https://docs.flutter.dev/testing/code-debugging) page.

**Debug mode assertions**

During development, you are highly encouraged to use Flutter’s [debug mode](https://docs.flutter.dev/testing/build-modes#debug). This is the default if you use bug icon in Android Studio, or flutter run at the command line. Some tools support assert statements through the command-line flag --enable-asserts.

In this mode, Dart assert statements are enabled, and the Flutter framework evaluates the argument to each assert statement encountered during execution, throwing an exception if the result is false. This allows developers to enable or disable invariant checking, such that the associated performance cost is only paid during debugging sessions.

When an invariant is violated, it’s reported to the console, with some context information to help track down the source of the problem.

For more information, see [Assert](https://dart.dev/guides/language/language-tour#assert) in the [Dart language tour](https://dart.dev/guides/language/language-tour).

**Debugging animations**

The easiest way to debug animations is to slow them down. The [Flutter inspector](https://docs.flutter.dev/development/tools/devtools/inspector) provides a **Slow Animations** button, or you can [slow the animations programmatically](https://docs.flutter.dev/testing/code-debugging#debugging-animations).

For more information on debugging janky (non-smooth) applications, see [Flutter performance profiling](https://docs.flutter.dev/perf/ui-performance).

**Measuring app startup time**

To gather detailed information about the time it takes for your Flutter app to start, you can run the flutter run command with the trace-startup and profile options.

***content\_copy***

$ flutter run --trace-startup --profile

The trace output is saved as a JSON file called start\_up\_info.json under the build directory of your Flutter project. The output lists the elapsed time from app startup to these trace events (captured in microseconds):

* Time to enter the Flutter engine code.
* Time to render the first frame of the app.
* Time to initialize the Flutter framework.
* Time to complete the Flutter framework initialization.

For example:

***content\_copy***

{

"engineEnterTimestampMicros": 96025565262,

"timeToFirstFrameMicros": 2171978,

"timeToFrameworkInitMicros": 514585,

"timeAfterFrameworkInitMicros": 1657393

}

**Tracing Dart code**

To perform a performance trace, you can use the DevTools [Timeline view](https://docs.flutter.dev/development/tools/devtools/performance). The Timeline view also supports importing and exporting trace files. For more information, see the [Timeline view](https://docs.flutter.dev/development/tools/devtools/performance) docs.

You can also [perform traces programmatically](https://docs.flutter.dev/testing/code-debugging#tracing-dart-code-performance), though these traces can’t be imported into DevTool’s Timeline view.

Be sure to use run your app in [profile mode](https://docs.flutter.dev/testing/build-modes#profile) before tracing to ensure that the runtime performance characteristics closely matches that of your final product.

**Performance overlay**

To get a graphical view of the performance of your application, turn on the performance overlay. You can do this in the by clicking the **Performance Overlay** button in the [Flutter inspector](https://docs.flutter.dev/development/tools/devtools/inspector).

You can also turn on the overlay [programmatically](https://docs.flutter.dev/testing/code-debugging#performance-overlay).

For information on how to interpret the graphs in the overlay, see [The performance overlay](https://docs.flutter.dev/perf/ui-performance#the-performance-overlay) in the [Flutter performance profiling](https://docs.flutter.dev/perf/ui-performance) guide.

**Debug flags**

In most cases, you won’t need to use the debug flags directly, as you’ll find the most useful debugging functionality in the [DevTools](https://docs.flutter.dev/development/tools/devtools) suite. But if you prefer to use the debug flags directly, see [Debug flags: performance](https://docs.flutter.dev/testing/code-debugging#debug-flags-performance) in the [Debugging Flutter apps programmatically](https://docs.flutter.dev/testing/code-debugging) page.

**Common problems**

The following is a problem that some have encountered on MacOS.

### **“Too many open files” exception (MacOS)**

The default limit for Mac OS on how many files it can have open at a time is rather low. If you run into this limit, increase the number of available file handlers using the ulimit command:

***content\_copy***

ulimit -S -n 2048

If you use Travis or Cirrus for testing, increase the number of available file handlers that they can open by adding the same line to flutter/.travis.yml, or flutter/.cirrus.yml, respectively.

**Widgets marked const that should be equal to each other, aren’t**

In debug mode, you might find that two const widgets that should to all appearances be equal (because of Dart’s constant deduplication) are not.

For example, this code should print 1:

***content\_copy***

print(<Widget>{

// this is the syntax for a Set<Widget> literal

const SizedBox(),

const SizedBox(),

}.length);

It should print 1 (rather than 2) because the two constants are the same and sets coalesce duplicate values (and indeed the analyzer complains that “Two elements in a set literal shouldn’t be equal”). As expected, in release builds, it does print 1. However, in debug builds it prints 2. This is because the flutter tool injects the source location of Widget constructors into the code at compile time, so the code is effectively:

***content\_copy***

print(<Widget>{

const SizedBox(/\* location: Location(file: 'foo.dart', line: 12) \*/),

const SizedBox(/\* location: Location(file: 'foo.dart', line: 13) \*/),

}.length);

This results in the instances being different, and so they are not deduplicated by the set. We use this injected information to make the error messages clearer when a widget is involved in an exception, by reporting where the relevant widget was created. Unfortunately, it has the visible side-effect of making otherwise-identical constants be different at compile time.

To disable this behavior, pass --no-track-widget-creation to the flutter run command. With that flag set, the code above prints “1” in debug and release builds, and error messages include a message saying that they cannot provide all the information that they would otherwise be able to provide if widget creation tracking was enabled.

## **Override**

[18]

Before looking into Dart Override let’s first look into what is Polymorphism,

The polymorphism is a combination of the two Greek words **poly,** which means **many** and morph means **morphing into different forms or shapes**. Together, polymorphism means the same entity can be used in various forms. In the programming aspect, the same method can be used in different classes. This technique makes programming more intuitive and more accessible.

For example - We have Shape class to define the shape of the object. The shape can be the circle, rectangle, square, straight line, etc. So here the goal is common, but the approach is different.

The method overriding is a technique to achieve polymorphism. Sometimes, we want a subclass object to give different results for the same method when subclass object invokes it. This can be done by defining the same method again in subclass. The method has the same name, same arguments, and the same return type. When that method is called, the subclass's method is executed instead of the method defined in the superclass.

**Method Overriding**

When we declare the same method in the subclass, which is previously defined in the superclass is known as the method overriding. The subclass can define the same method by providing its own implementation, which is already exists in the superclass. The method in the superclass is called **method overridden,** and method in the subclass is **called method overriding**. Let's understand the method overriding in the following example.

**Method Overriding Example**

We define two classes; first, is a subclass called **Human,** and the second is a superclass **Boy**. The Boy subclass inherits the Human superclass. The same method **void showInfo()** in both classes is defined with the different implementation. The subclass has its own definition of the void **showInfo().** Let's have a look at the following code snippet.

**Example -**

class Human{

   //Overridden method

    void run()

   {

      print("Human is running");

   }

}

class Man extends Human{

   //Overriding method

    void run(){

      print("Boy is running");

   }

}

void main(){

      Man m = new Man();

      //This will call the child class version of run()

      m.run();

}

**Output:**

Boy is running

**Explanation:**

In the above example, we defined a method with the same name in both, subclass and superclass. The purpose of method overriding is to give the own implementation of subclass method. When we created the object of the Boy subclass, it executed the subclass method and printed the Man is running instead of Human is running.

If we create the object of a parent class, then it will be always invoked the parent class method.

Let's take another example where we create two Classes called College and Student with common method void student\_details(). Let's have a look at the following code snippet.

**Example – 2**

class College{

 // Declaring variables

           String name;

           int rollno;

// Overriden Method

void stu\_details(name,rollno){

         this.name = name;

         this.rollno = rollno;

}

void display(){

         print("The student name:${name}");

         print("The student rollno: ${rollno}");

         print("The result is passed");

      }

 }

class Student extends College{

// Overriding Method

void stu\_details(name,rollno){

         this.name = name;

         this.rollno = rollno;

}

void show(){

         print("The student name:${name}");

         print("The student rollno: ${rollno}");

         print("The result is failed");

}

}

void main(){

//Creating object of subclass

Student  st = new Student();

st.stu\_details("Joseph",101);

st.show();

// Creating object of superclass

College cg = new College();

cg.stu\_details("Jason",102);

cg.display();

}

**Output:**

The student name: Joseph

The student rollno: 101

The result is failed

The student name:Peter

The student rollno: 102

The result is passed

**Explanation:**

In the above example, we create two classes - College as a parent class and Student as a child class. The method **stu\_details** defined in both classes with the same parameters and same return types.

Now, the College superclass is inherited by the Student subclass and the **stu\_details()** method is overridden in the subclass.

We created the object of Student and to invoked the **stu\_details()** with suitable arguments. It executed the subclass method, and then it printed the result.

Same as we created the object of **College superclass** object invoked its methods and printed the different results.

**Method Overriding using super Keyword**

We can invoke the parent class method without creating its object. It can be done by using the super keyword in the subclass. The parent class data member can be accessed in the subclass by using the super keyword. Let's understand the following example.

**Example -**

class Human{

   //Overridden method

    void run()

   {

      print("Human is running");

   }

}

class Man extends Human{

   //Overriding method

    void run(){

       // Accessing Parent class run() method in child class

       super.run();

      print("Boy is running");

   }

}

void main(){

      Man m = new Man();

      //This will call the child class version of eat()

      m.run();

}

**Output:**

Human is running

Boy is running

**Explanation:**

In the above program, we accessed the Human class method in child class using the super keyword. Now, we don't need to instantiate the parent class. We only created the object of subclass, which invoked the **run()** method of child class and parent class.

Note - When we created the child class object and invoked the method, it executes the parent class (if accessed by super keyword) method first, then the child class method.

**Advantage of method overriding**

The main benefit of the method overriding is that the subclass can provide its own implementation to the same method as per requirement without making any changes in the superclass method. This technique is much when we want to subclass method to behave differently also with the same name.

**Rules of Method overriding in Dart**

The few rules of method overriding are given below. These points must be kept in mind while declaring the same method in subclass.

1. The overriding method (the child class method) must be declared with the same configuration as the overridden method (the superclass method). The return type, list of arguments and its sequence must be the same as the parent class method.
2. The overriding method must be defined in the subclass, not in the same class.
3. The static and final method cannot be inherited in the subclass as they are accessible in their own class
4. The constructor of the superclass cannot be inherited in a subclass.
5. A method that cannot be inherited, then it cannot be overridden.

For more info please visit –

* <https://www.darttutorial.org/dart-tutorial/dart-method-overriding/>
* <https://api.flutter.dev/flutter/dart-core/override-constant.html>

## **Abstract Classes**

Abstract classes are the classes in Dart that has one or more abstract method. Abstraction is a part of the data encapsulation where the actual internal working of the function hides from the users. They interact only with external functionality. We can declare the abstract class by using the abstract keyword. There is a possibility that an abstract class may or may not have abstract methods.

Abstract methods are those methods, which are declared without implementation. The concrete methods or normal methods are declared with implementation. An abstract class can contain both types of methods, but a normal class is not allowed to have abstract methods.

We cannot create the instance of an abstract class that means it can't be instantiated. It can only be extended by the subclass, and the subclass must be provided the implantation to the abstract methods which are present in the present class. Then it is necessary to declare abstract subclass.

**Rules for Abstract classes:**

The rules of the abstract are given below.

1. An abstract class can have an abstract method (method without implementation), or not.
2. If there is at least one abstract method, then the class must be declared abstract.
3. The object of the abstract class cannot be created, but it can be extended.
4. An abstract keyword is used to declare the abstract class.
5. An abstract class can also include normal or concrete (method with the body) methods.
6. All abstract methods of parent class must be implemented in the subclass.

**Declaring Abstract Class**

An abstract keyword followed by a class name is used to declare the abstract class. An abstract class mostly used to offer a base for the subclass to extends and implement the abstract method.

**Syntax:**

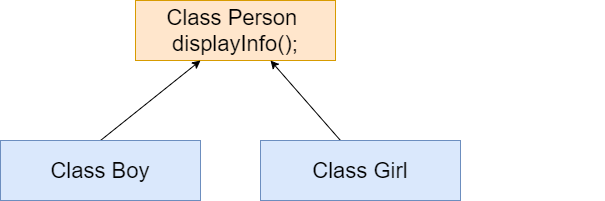
abstract class ClassName {

 // Body of abstract class

}

**Usage of Abstract class**

Let's suppose we have a class **Person** that has method **displayInfo()**, and we have to sub classes of it **Boy** and **Girl**. Each of the person information varies from the other person, so there is no benefit to implementing the **displayInfo()** in the parent class. Because every subclass must override the parent class method by provides its own implementation. Thus, we can force the subclass to provide implementation to that method, so that is the benefit to make method abstract. We don't require the give implementation in the parent class.



Let's understand the above scenario through the following code.

**Example -**

abstract class Person {

//declaring abstract method

void displayInfo();  //abstract method

}

class Boy extends Person

{

// Overriding method

void displayInfo() {

    print("My name is Johnathon");

       }

}

class Girl extends Person

{

// Overriding method

void displayInfo() {

    print("My name is Grecia");

       }

}

void main() {

Boy b = new Boy();  // Creating Object of Boy class

Girl g = new Girl();  // Creating Object of Girl class

b.displayInfo();

g.displayInfo();

}

**Output**

My name is Johnathon

My name is Grecia

**Explanation:**

As we can see that in the above code, we implemented the abstract method in two subclasses according to its requirement and then we called the **displayInfo()** method using the object of the both class's object.

## **Assignment Doctor**

## **Generic Types**

[20]

**Generics** are utilized to apply more grounded type checks at the compile time. They implement type-safety in code. For instance, in collections, the type-safety is authorized by holding a similar kind of information. Generics help composes reusable classes, methods/functions for various information types.

**Collections**, **streams**, and **futures** are the center library highlights you’ll use with generics frequently as you compose Flutter applications with Dart. It’s a positive routine to exploit generics any place they’re accessible. It’s additionally critical to have the option to believe that information that emerging from prospects or streams has the correct construction, and Dart’s generics include permits you to indicate what that design ought to be.

**Generics with Collections:**

Collection generics can assist you with being sure every component inside a collection is of the normal kind. You can proclaim collection factors without generics like this:

List myList;

Map myMap;

That code is identical to the accompanying:

List<dynamic> myList;

Map<dynamic, dynamic> myMap;

This ought to possibly be done when you truly need a collection containing a wide range of types. If you know the expected kind of list’s components, you ought to indicate that type inside the angle brackets, which will permit the Dart analyzer to assist you with keeping away from mistakes:

List<String> myList;

Likewise, in the event that you expect for a map to contain keys and values of a specific sort, remember them for the revelation:

Map<String, dynamic> jsonData;

Map<int, String> myMap;

With maps, the primary type inside the angle brackets obliges the map’s keys while the second does likewise for the **guide’s** qualities. It ought to be noticed that Dart permits you to utilize any sort of map keys, while in certain languages just strings are permitted.

**Generics with Asynchronous:**

We utilize **asynchronous** activities to permit an application to stay responsive while trusting that moderately extensive tasks will finish. Instances of tasks that require some time in this manner may be getting information over a network, working with the document framework, or getting to a database. Dart’s essential development supporting asynchronous programs are the **Future** and the **Stream**.

It’s a best practice to incorporate kinds when managing **futures** and **streams**. This holds them back from returning information of some unacceptable kind. As in different circumstances, if you neglect to incorporate a particular kind, dynamic is expected, and any sort will be permitted.

**=> Futures:**

It addresses the consequence of asynchronous activity. At the point when at first made, a future is uncompleted. When the activity is finished, what’s to come is finished either with a worth or an error. Utilizing generics, we can indicate the normal type of significant value that is created.

This function returns a Future, yet a bool is at last delivered when the future finishes:

Future<bool> someData() {

  return Future.delayed(const Duration(seconds: 2 ), () => true);

}

**=> Streams:**

They resemble an asynchronous list, or an information pipe, conveying an asynchronous grouping of information. As values become accessible, they are embedded into the stream. **Listeners** on the stream get the values in a similar request they were embedded.

A typical method to permit a class to communicate with outside code is to utilize a **StreamController** joined with a Stream. Adding generic sort assignments to these is a decent method to ensure they don’t convey unforeseen outcomes:

final \_onData= StreamController<Data>.broadcast();

Stream<Data> get onData => \_onData.stream;

This code makes a StreamController that can be utilized to send “Data” objects out on a Stream asynchronously.

**Generics in Flutter:**

The most widely recognized spots you’ll utilize generics in Flutter are in collections and stateful widgets. Stateful widgets have both a StatefulWidget class and a going with State class. The State class utilizes generics to ensure it manages the StatefulWidget it has a place with, utilizing the syntax structure **State<MyApp>**. A State case is conventional, composed to work with any StatefulWidget, and for this situation, we’re making a state explicitly for our MyApp class.

class MyApp extends StatefulWidget {

  @override

  \_MyAppState createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

  @override

  Widget build(BuildContext context) {

    return Row(

      children: <Widget>[

        const Text("Hello, "),

        const Text("Flutter Dev's"),

      ],

    );

  }

}

If you somehow managed to leave off the angle brackets and the MyApp symbol, the analyzer wouldn’t say anything negative, yet you would have made a State attached to the default dynamic type. In addition to the fact that this is not type-safe, however, it could make issues for the framework as it attempts to coordinate with state cases to the right widgets.

The List strict passed to children for the Row widget is comparatively composed, this time as a list of Widget objects: <Widget>[]. This assignment assists Dart with getting issues at configuration time. If you attempt to put a non-widget into that collection, you will get alerts.

A Row doesn’t have the foggiest idea of how to manage objects that aren’t widgets, so getting this sort of issue before your code runs is helpful. The generics additionally serve to make code that is more self-reporting, clarifying what’s generally anticipated inside the collection.

**Generic Methods & Functions:**

Dart upholds the utilization of generics on **methods** and **functions**. This can prove to be useful when you have an activity to perform and you would prefer not to compose a few unique renditions of that activity to help different types.

Assume you need to make a generic function that can change over string esteem into an **enum**. Generics can assist you with abstaining from utilizing **dynamic**, guarding your return type safe:

enum Size {

  small,

  medium,

  large

}

T stringToEnum<T>(String str, Iterable<T> values) {

  return values.firstWhere(

    (value) => value.toString().split('.')[1] == str,

    orElse: () => null,

  );

}

Size size = stringToEnum<Size>("large", Size.values);

In the above code, T addresses the type to be given by the caller of **stringToEnum()**. That type will be utilized as the function’s return type, so when we call the function on the last line, the size will be securely composed. By chance, **T** will be given to the values boundary type, guaranteeing that only the right sorts will be acknowledged in the Iterable collection. The stringToEnum() function will work in a type-safe path for any **enum**. The gave string doesn’t coordinate with any of the enum values, and null will be returned.

**Generic Classes:**

You will likewise need to try not to make separate classes for the sole motivation behind dealing with various information types. Maybe you need to make a specific collection class, and still, keep up type safety.

class Data<T> {

  List<T> \_data = [];

void push(T item) => \_data.add(item);

  T pop() => \_data.removeLast();

}

This **class** furnishes you with a collection that will never really push things onto data and pop them off. It’s difficult to get to the data’s values straightforwardly from outside a case, as the \_data property is private.

final data = Data<String>();

data.push("A string.");  // works

data.push(5);            // errors

This data won’t permit a value of some unacceptable type to be added. Furthermore, the **pop()** technique will create a value with a coordinating with the return type.

For more information please visit:

* <https://www.geeksforgeeks.org/dart-generics/>
* <https://www.educative.io/answers/generic-methods-in-dart>
* <https://www.tutorialspoint.com/dart_programming/dart_programming_generics.htm>
* <https://dart.academy/generics-in-dart-and-flutter/>

## **Passing Function Callback**

[21]

In this Section, we will see how we can use callback functions in flutter. We will learn about different methods to implement callback functions in flutter. **Callback** is basically a function or a method that we pass as an argument into another function or a method to perform an action. In the simplest words, we can say that Callback or VoidCallback are used while sending data from one method to another and vice-versa. It is very important to maintain a continuous flow of data throughout the flutter app.

Let’s assume that you are working on an app.  This app displays some sort of data.  Now to alter the values in the application, there are 2 approaches that you can take, either change the state using various state-altering techniques or change the value using a Callback. If we are to work with the Callback function there are 2 possible methods that we can use as shown below:

**Method 1:** Directly writing the callback

In this approach, we just define the function that is supposed to trigger a callback when a specific event occurs.

**Example:**

import 'package:flutter/material.dart';

// function to trigger the app build process

void main() {

runApp(MaterialApp(

  home: Scaffold(

  // appbar

  appBar: AppBar(

    title: const Text('GeeksForGeeks'),

    backgroundColor: const Color.fromRGBO(15, 157, 88, 1),

  ),

  body: const MyApp(),

  ),

));

}

class MyApp extends StatefulWidget {

const MyApp({Key? key}) : super(key: key);

@override

// ignore: library\_private\_types\_in\_public\_api

\_MyAppState createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

int count = 0;

@override

Widget build(BuildContext context) {

  return Center(

  child: Column(

    mainAxisAlignment: MainAxisAlignment.center,

    children: [

    Text(

      '$count',

      style: const TextStyle(fontSize: 50.0),

    ),

    ElevatedButton(

      style: ButtonStyle(

        backgroundColor: MaterialStateProperty.all(Colors.green)),

      onPressed: () {

      setState(() {

        count++;

      });

      },

      child: const Text(''

        'increase'),

    ),

    ],

  ),

  );

}

}

**Method 2: Passing the callback function**

**In this approach, the callback is directly passed to the event. As shown in the below example, the onPressed action makes the direct callback function defined in the earlier part of the code.**

**Example:**

import 'package:flutter/material.dart';

void main() {

runApp(MaterialApp(

  home: Scaffold(

  // appbar

  appBar: AppBar(

    title: const Text('GeeksForGeeks'),

    backgroundColor: const Color.fromRGBO(15, 157, 88, 1),

  ),

  body: const MyApp(),

  ),

));

}

class MyApp extends StatefulWidget {

const MyApp({Key? key}) : super(key: key);

@override

// ignore: library\_private\_types\_in\_public\_api

\_MyAppState createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

int count = 0;

// callback function

callBack() {

  setState(() {

  count++;

  });

}

@override

Widget build(BuildContext context) {

  return Center(

  child: Column(

    mainAxisAlignment: MainAxisAlignment.center,

    children: [

    Text(

      '$count',

      style: const TextStyle(fontSize: 50.0),

    ),

    ElevatedButton(

      style: ButtonStyle(

        backgroundColor: MaterialStateProperty.all(Colors.green)),

      onPressed: callBack,

      child: const Text('increase'),

    ),

    ],

  ),

  );

}

}

**For more information please visit :**

* <https://medium.com/@dnkibere/passing-a-function-as-an-argument-flutter-e011ad2afd86>

## **Custom Foreach Callback**

## **Get Input From User**

[22]

**How to take input in Dart**

To take input from a user in Dart, you need to import the dart:io library. The input is taken through the console using the .readLineSync() function.

**Code**

In the code below, enter your name in stdin column first and then press the run button:

import 'dart:io';

void main()

{

    String name = stdin.readLineSync();

    print("Hello, $name");

}

This was how you can take a string as input in the console. Below, we will see how we can take an integer value as input in the console:

import 'dart:io';

void main()

{

    int number = int.parse(stdin.readLineSync());

    print("The number is $number");

}

For more information please visit :

* <https://www.geeksforgeeks.org/dart-standard-input-output/>

## **Check User Options + Switch**

The switch statement evaluates an expression, matches the expression’s value to a case clause and executes the statements associated with that case.

Following is the syntax.

switch(variable\_expression) {

   case constant\_expr1: {

      // statements;

   }

   break;

   case constant\_expr2: {

      //statements;

   }

   break;

   default: {

      //statements;

   }

   break;

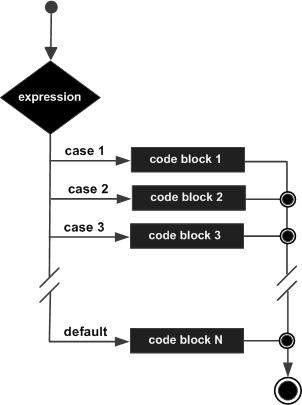
}

The value of the **variable\_expression** is tested against all cases in the switch. If the variable matches one of the cases, the corresponding code block is executed. If no case expression matches the value of the variable\_expression, the code within the default block is associated.

The following rules apply to a switch statement −

* There can be any number of case statements within a switch.
* The case statements can include only constants. It cannot be a variable or an expression.
* The data type of the variable\_expression and the constant expression must match.
* Unless you put a break after each block of code, the execution flows into the next block.
* The case expression must be unique.
* The default block is optional.

The flow diagram of the **switch…case** statement is as follows −



**Example - switch…case**

void main() {

   var grade = "A";

   switch(grade) {

      case "A": {  print("Excellent"); }

      break;

      case "B": {  print("Good"); }

      break;

      case "C": {  print("Fair"); }

      break;

      case "D": {  print("Poor"); }

      break;

      default: { print("Invalid choice"); }

      break;

   }

}

The example verifies the value of the variable grade against the set of constants (A, B, C, D, and E) and executes the corresponding blocks. If the value in the variable doesn’t match any of the constants mentioned above, the default block will be executed.

The following **output** is displayed on successful execution on the above code.

Excellent

## **Improve Display Users**

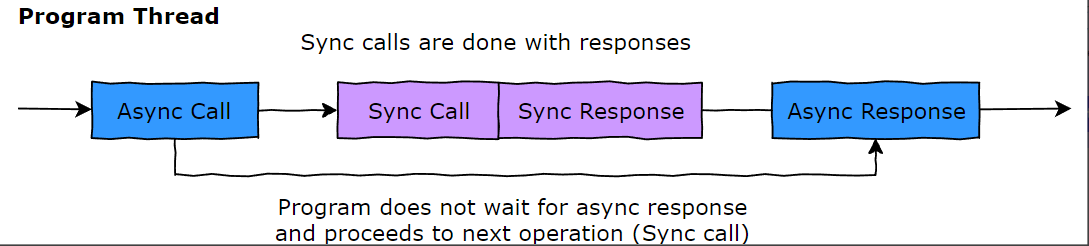
There are many tools in flutter which can be used to improve the display for the users. For this please refer to the components section and the Advance section. Apart from this please refer to the dart official link: <https://dart.dev/guides/language/effective-dart/design>

## **Futures**

[26]

A **future** is a valuable construct that allows asynchronous programming in Dart. **Asynchronous programming** is programming that caters to delayed operations. These delayed operations happen when you need to fetch data over a server, write to a database, or read a file on the system (commonly referred to as I/O operations).

With asynchronous programming, you can tell your computer that it is free to perform other operations while the asynchronous operation completes.



// Function that replicates a slow operation

Future<void> getOrder(){

    Future.delayed(

    Duration(seconds: 2), () =>

      print("Vanilla Shot"));

}

void main() {

  getOrder();

  //Notice how the following appears on screen first

  print("Getting order...");

}

**What is a future?**

A future is an instance of the Future class. It can have two states: uncompleted or completed.

**Uncompleted**

When a future is called, but has not returned a value, the future is in the uncompleted state.

Future<String> getOrder(){

  Future.delayed(

    Duration(seconds: 2), ()

    => 'vanilla shot');

}

void main() {

  var order = getOrder();

  print("Getting order...");

  print(order);

}

**Completed**

If the asynchronous call is successful, the future completes with a value. Otherwise, it completes with an error.

Future<String> getOrder(){

  return Future.delayed(

    Duration(seconds: 2), ()

    => 'vanilla shot');

}

void main() async {

  var order = getOrder();

  print("Getting order...");

  print(await order);

}

Read more about getting a program to wait using async/await, in the upcoming section

**a. Completing with a value**

If successful, a future of type Future<String> will return a String value. Similarly, Future<T> (a future with any valid type T) will return T.

**b. Completing with an error**

If unsuccessful, a future will return an error.

// Function that replicates an error

Future<void> getOrder(){

  Future.delayed(

    Duration(seconds: 2), ()

    => throw Exception('Some Error'));

}

void main() {

  getOrder();

  print("Getting order...");

}

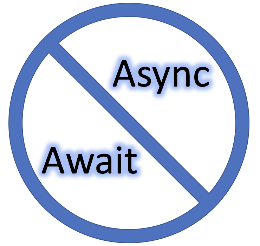
Read more about asynchronous programming in Dart in the [official documentation](https://dart.dev/codelabs/async-await).

For more information please click on the following websites:

* <https://www.kodeco.com/32851541-dart-futures-and-streams>
* <https://medium.com/dartlang/dart-asynchronous-programming-futures-96937f831137>
* <https://zetcode.com/dart/future/>
* <https://api.flutter.dev/flutter/dart-async/Future-class.html>

## **Async and Await**

[25]



The async and await approaches in Dart are very similar to other languages (looking at you C#), which makes it a comfortable topic to grasp for those who have used this pattern before. However, even if you don’t have experience with asynchronous programming using async/await, you should find it easy to follow along here. Do keep in mind that this article assumes at least a basic familiarity with asynchronous programming.

In a nutshell, you have two keywords to understand - async and await. Any functions you want to run asynchronously need to have the async modifier added to it. This modifier comes right after the function signature, like this:

void hello() async {

  print('something exciting is going to happen here...');

}

This is, of course, a contrived example. Typically, the function you want to run asynchronously would have some expensive operation in it like file I/O (an API call to a RESTful service)or some sort of more common computation. Don’t worry, we’ll cover more complex scenarios in a bit…

First, we come to await. The await part basically says  -  go ahead and run this function asynchronously and, when it is done, continue on to the next line of code. This is the best part of using async/await, you can write code that is very easy to follow, like this:

void main() async {

  await hello();

  print('all done');

}

There are two important things to grasp concerning the block of code above. First off, we use the async modifier on the main method because we are going to run the hello() function asynchronously.

Secondly, we place the await modifier directly in front of our asynchronous function. Hence, this is frequently referred to as the async/await pattern.

Just remember, if you are going to use await, make sure that both the caller function and any functions you call within that function all use the async modifier.

**How does this work under the hood?**

So, how exactly does Dart make all this work? To really grasp this, you need to understand [**Dart futures**](https://www.educative.io/edpresso/having-fun-with-futures-in-dart). You can skip this section if you want and still use the pattern discussed above, but it does help to have the knowledge of how and why it works. So here are the finer points of how async and await work in Dart.

When you await an asynchronous function, the execution of the code within the caller suspends while the async operation is executed. When the operation is completed, the value of what was awaited is contained within a Future object.

Take a look at the simple program below. We assign the result of the asynchronous function four() to the variable x. Then, we print out its number to prove we got the expected result.

import 'dart:async';

void main() async {

  var x = await four();

  print(x);

}

Future<int> four() async {

 return 4;

}

**A more realistic example**

So far, I have shown you contrived examples that never really needed to be asynchronous. This was done to keep things as simple as possible. Now, let’s do some realistic async/await work.

Typically, the reason you would want to do async programming is if you knew that you were going to perform a long-running operation, and you didn’t want your program to be unresponsive while said operation was running. Let’s create a long-running operation (2 seconds) and do it async.

import 'dart:async';

class Employee {

  int id;

  String firstName;

  String lastName;

  Employee(this.id, this.firstName, this.lastName);

}

void main() async {

  print("getting employee...");

  var x = await getEmployee(33);

  print("Got back ${x.firstName} ${x.lastName} with id# ${x.id}");

}

Future<Employee> getEmployee(int id) async {

  //Simluate what a real service call delay may look like by delaying 2 seconds

  await Future<Employee>.delayed(const Duration(seconds: 2));

  //and then return an employee - lets pretend we grabbed this out of a database 🙂

  var e = new Employee(id, "Joe", "Coder");

  return e;

}

If you run the above code, you will see the message “getting employee…” print out immediately. Then, two seconds later, the employee arrives back and the details of that employee are printed out.

**Multiple async calls**

In some languages, having to make multiple async calls can be a real headache if they don’t support the async/await pattern. This is because, typically, you would make the first async call and, within its callback, nest another async call, and so on… This is referred to as “**callback hell**”. However, with async/await, you make the calls linearly and without nesting - just like you would in any non-async code.

Consider the example below, we have three async methods and we want to asynchronously call them, one at a time, in order.

import 'dart:async';

Future<String> firstAsync() async {

  await Future<String>.delayed(const Duration(seconds: 2));

  return "First!";

}

Future<String> secondAsync() async {

  await Future<String>.delayed(const Duration(seconds: 2));

  return "Second!";

}

Future<String> thirdAsync() async {

  await Future<String>.delayed(const Duration(seconds: 2));

  return "Third!";

}

void main() async {

  var f = await firstAsync();

  print(f);

  var s = await secondAsync();

  print(s);

  var t = await thirdAsync();

  print(t);

  print('done');

}

If you were to run the above program, you would see the following result shown in your console (note the two second delay as well between each call):

First!

Second!

Third!

done

**For more information please visit the following links:**

* [**https://www.tutorialspoint.com/async-and-await-in-dart-programming**](https://www.tutorialspoint.com/async-and-await-in-dart-programming)
* [**https://www.javatpoint.com/dart-async**](https://www.javatpoint.com/dart-async)
* [**https://sarunw.com/posts/how-to-use-async-await-in-flutter/**](https://sarunw.com/posts/how-to-use-async-await-in-flutter/)
* [**https://www.geeksforgeeks.org/using-await-async-in-dart/**](https://www.geeksforgeeks.org/using-await-async-in-dart/)

## **Static**

[24]

The static keyword is used to declare the class variable and method. It generally manages the memory for the global data variable. The static variables and methods are the member of the class instead of an individual instance. The static variable or methods are the same for every instance of the class, so if we declare the data member as static then we can access it without creating an object. The class object is not required to access the static method or variable we can access it by putting the class name before the static variable or method. Using the class name, we can call the class method from the other classes.

**Dart Static Variable**

A variable which is declared using the static keyword inside the class is called [Dart](https://www.javatpoint.com/dart-programming) static keyword. These are the member of the class instead of a specific instance. The static variables are treated the same for all instances of the class; it means a single copy of the static variable is shared among all instances of classes. It allocates memory once and at the class loading and uses throughout the program.

**Point to Remember -**

* The static variable is also identified as a class variable.
* Single copy of the static variable is shared among the instance of a class.
* It can be accessed using the class name. We don't need to create an object of that class they belong to.
* The static variables can be accessed directly in the static methods.

**Declaring Static Variable**

Dart provides the static keyword to declare the static variable. It is declared by using the static keyword followed by the variable name. The syntax is given below.

**Syntax:**

static [data\_type] [variable\_name];

**Accessing Static Variable**

We can access the static variable by using the class name itself instead of creating an object of it. The syntax is given below.

**Syntax:**

ClassName.staticVariableName;

**Dart Static Method**

The concept of the static method is also similar to static variable. The static methods are the member of the class instead of the class instance. The static methods can use only static variables and can invoke the static method of the class. We don't need to create an instance of class the access it. The static method is useful when we want to use it in other classes.

**Points to Remember**

* The static methods are the member class instead of its object.
* Static methods are also identifies as class methods.
* We can access static methods using the class name.
* A particular copy of the static method is distributed among all the instances of a class.

**Declaring Static Methods**

We can declare the static method by using the static keyword followed by the method name with the return type. The syntax is given below.

**Syntax:**

static return\_type method\_name() {

 //statement(s)

}

**Calling Static Method**

The static methods can be called by using the class name, which they belong to instead of creating an object.

**Syntax:**

className.staticMethod();

Let's understand the following example.

**Example -**

class Student {

   static String stdBranch;  // Declaring static variable

   String stdName;

   int roll\_num;

   showStdInfo() {

     print("Student's name is: ${empName}");

     print("Student's salary is: ${roll\_num}");

     print("Student's branch name is: ${stdBranch}");

      }

}

void main() {

  Student std1 = new Student();  // Creating instances of student class

  Student std2 = new Student();

  // Assigning value of static variable using class name

  Student.stdBranch = "Computer Science";

  std1.stdName = "Ben Cutting";

  std1.roll\_num = 90013

  std1.showStdInfo();

  std2.stdName = "Peter Handscomb";

  std2.roll\_num = 90014

  std2.showStdInfo();

}

**Output**

Student's name is: Ben Cutting

Student's salary is: 90013

Student's branch name is: Computer Science

Student's name is: Peter Handscomb

Student's salary is: 90014

Student's branch name is: Computer Science

**Explanation:**

In the above code, we declared the class called **Student**, which has three fields including static variable **stdBranch** and one method **showStdInfo()**. We created two instances of class Student and assigned values to the class variables.

The static variable **stdBranch** accessed by using the class name and assigned value. Then, we called the showStdInfo() function by objects std1 and stu2. It printed details of the student as an output.

For more info please visit:

* <https://www.geeksforgeeks.org/dart-static-keyword/>

## **Safe Operators**

[23]

The introduction of Null Safety marks a major milestone for the Dart language. Null Safety helps you avoid an entire class of problems by **catching null errors during development rather than at runtime**.

This Section outlines what's changed, and shows how to use the new Null Safety features by example.

**Some Context**

Null References were first introduced in 1965 in the ALGOL programming language, and since then they have been adopted by most mainstream programming languages.

However, null errors are so common that null references have been called the The Billion Dollar Mistake.



So let's see what's changed in Dart to address this.

**Dart Type System**

Before addressing Null Safety, let's talk about the Dart type system.

Dart is said to have a sound type system. When we write Dart code, the type checker makes sure that we can't write something like this:

int age = "hello world"; // A value of type `String` can't be assigned to a variable of type `int`

This code produces an error telling us that "a *String* value can't be assigned to a variable of type *int*".

Similarly, when we write a function in Dart, we can specify a return **type**:

int square(int value) {

  return value \* value;

}

Because of **type safety**, Dart can guarantee with 100% confidence that this function **always** returns an int.

Type safety help us write safer programs, and more easily reason about the code.

But type safety alone can't guarantee that a variable (or return value) is not null. As a result this code compiles, but generates an exception **at runtime**:

square(null);

// Unhandled Exception: NoSuchMethodError: The method '\*' was called on null.

In this example it's easy enough to spot the problem. But in large codebases it's hard to keep track of what can and cannot be null.

Runtime null checks can mitigate the problem, but they add more noise:

int square(int value) {

  assert(value != null); // for debugging

  if (value == null) throw Exception();

  return value \* value;

}

What we really want here is to tell Dart that the value argument should **never** be null.

A better solution is needed - and now we have it.

[**Dart Null Safety: Benefits**](https://codewithandrea.com/videos/dart-null-safety-ultimate-guide-non-nullable-types/#dart-null-safety-benefits)

Dart 2.12 enables Sound Null Safety **by default** and brings three main benefits:

* We can write null-safe code with strong **compile-time** guarantees. This makes us productive because Dart can tell us when we're doing something wrong.
* We can more easily declare our **intent**. This leads to APIs that are self-documenting and easier to use.
* The Dart compiler can optimise our code, resulting in smaller and faster programs.

So, let's see how Null Safety works in practice.

[**Declaring Non-Nullable Variables**](https://codewithandrea.com/videos/dart-null-safety-ultimate-guide-non-nullable-types/#declaring-non-nullable-variables)

The main language change is that all types are now non-nullable **by default**. This means that this code doesn't compile:

void main() {

  int age; // non-nullable

  age = null; // A value of type `Null` can't be assigned to a variable of type 'int'

}

When using non-nullable variables, we must follow one important rule:

Non-nullable variables must always be initialized with non-null values.

If you reason along these lines, it will be easier to understand all the new syntax changes.

Let's revisit this example:

int square(int value) {

  return value \* value;

}

Here both the value argument and the return value are now **guaranteed** to be not null.

As a result **runtime** null checks are no longer necessary, and this code now produces a **compile-time** error:

square(null);

// The argument type 'Null' can't be assigned to the parameter type 'int'

But if all types are now **non-nullable** by default, how can we declare **nullable** variables?

[**Declaring Nullable Variables**](https://codewithandrea.com/videos/dart-null-safety-ultimate-guide-non-nullable-types/#declaring-nullable-variables)

The ? symbol is what we need:

String? name; // initialized to null by default

int? age = 36; // initialized to non-null

age = null; // can be re-assigned to null

Note: You don't need to initialize a nullable variable before using it. It is initialized to null by default.

Here are some other ways of declaring nullable variables:

// nullable function argument

void openSocket(int? port) {

  // port can be null

}

// nullable return type

String? lastName(String fullName) {

  final components = fullName.split(' ');

  return components.length > 1 ? components.last : null;

}

// using generics

T? firstNonNull<T>(List<T?> items) {

  // returns first non null element in list if any

  return items.firstWhere((item) => item != null);

}

Take away: you can declare nullable variables **anywhere** in your code with the ? syntax.

Nullable variables are a good way of expressing the **absence** of a value, and this is useful in many APIs.

When you design an API, ask yourself if a variable should be nullable or not, and declare it accordingly.

But there are cases where we know that something can't be null, but we can't **prove** it to the compiler. In these cases, the assertion operator can help.

**The assertion operator**

We can use the assertion operator ! to assign a nullable expression to a non-nullable variable:

int? maybeValue = 42;

int value = maybeValue!; // valid, value is non-nullable

By doing this, we're **telling** Dart that maybeValue is not null, and it's safe to assign it to a non-nullable variable.

Note that applying the assertion operator to a null value will throw a runtime exception:

String? name;

print(name!); // NoSuchMethodError: '<Unexpected Null Value>'

print(null!); // NoSuchMethodError: '<Unexpected Null Value>'

When your assumptions are wrong, the ! operator leads to runtime exceptions.

Sometimes we need to work with APIs that return nullable values. Let's revisit the lastName function:

String? lastName(String fullName) {

  final components = fullName.split(' ');

  return components.length > 1 ? components.last : null;

}

Here the type system can't help. If we **know** that the function **will** return a non-null value for a given argument, we should assign it to a non-nullable variable **as soon as possible**.

This is done with the ! operator:

// prefer this:

String last = lastName('Andrea Bizzotto')!;

// to this:

String? last = lastName('Andrea Bizzotto');

**In summary:**

* Try to create non-nullable variables when possible, as these will be **guaranteed** to be not null at **compile time**.
* If you know that a nullable expression won't be null, you can assign it to a non-nullable variable with the ! operator.

[**Flow Analysis: Promotion**](https://codewithandrea.com/videos/dart-null-safety-ultimate-guide-non-nullable-types/#flow-analysis-promotion)

Dart can make your life easier by taking into account null checks on nullable variables:

int absoluteValue(int? value) {

  if (value == null) {

    return 0;

  }

  // if we reach this point, value is non-null

  return value.abs();

}

Here we use an if statement to return early if the value argument is null.

Beyond that point, value cannot be null and is treated (or **promoted**) to a non-nullable value. Hence we can safely use value.abs() rather than value?.abs() (with the null-aware operator).

Similarly, we could throw an exception if the value is null:

int absoluteValue(int? value) {

  if (value == null) {

    throw Exception();

  }

  // if we reach this point, value is non-null

  return value.abs();

}

Once again, value is promoted to a non-nullable value, and the null-aware operator ?. is not needed.

In summary:

* Use **upfront** null checks to return early or throw exceptions
* After null checks, nullable variables are **promoted** to be non-nullable

And after a nullable variable has been null checked, Dart lets you use it as a non-nullable variable, which is quite nice.

[**Flow Analysis: Definite Assignment**](https://codewithandrea.com/videos/dart-null-safety-ultimate-guide-non-nullable-types/#flow-analysis-definite-assignment)

Dart knows where variables are **assigned** and where they're **read**.

This example shows how to initialize a non-nullable variable **after** checking for a condition:

int sign(int x) {

  int result; // non-nullable

  print(result.abs()); // invalid: 'result' must be assigned before it can be used

  if (x >= 0) {

    result = 1;

  } else {

    result = -1;

  }

  print(result.abs()); // ok now

  return result;

}

As long as a non-nullable variable is given a value **before** it's used, Dart is happy.

[**Using non-nullable variables with classes**](https://codewithandrea.com/videos/dart-null-safety-ultimate-guide-non-nullable-types/#using-non-nullable-variables-with-classes)

Instance variables in classes must be initialized if they are non-nullable:

class BaseUrl {

  String hostName; // Non-nullable instance field 'hostName' must be initialized

  int port = 80; // ok

}

If a non-nullable instance variable can't be initialized with a default value, set it with a constructor:

class BaseUrl {

  BaseUrl(this.hostName);

  String hostName; // now valid

  int port = 80; // ok

}

[**Non-nullable named and positional arguments**](https://codewithandrea.com/videos/dart-null-safety-ultimate-guide-non-nullable-types/#non-nullable-named-and-positional-arguments)

With Null Safety, non-nullable **named** arguments must always be **required** or have a **default value**.

This applies to regular methods as well as class constructors:

void printAbs({int value}) {  // 'value' can't have a value of null because of its type, and no non-null default value is provided

  print(value.abs());

}

class Host {

  Host({this.hostName}); // 'hostName' can't have a value of null because of its type, and no non-null default value is provided

  final String hostName;

}

We can fix the code above with the new required **modifier**, which replaces the old @required **annotation**:

void printAbs({required int value}) {

  print(value.abs());

}

class Host {

  Host({required this.hostName});

  final String hostName;

}

And when we use the above APIs, Dart can tell us if we're doing something wrong:

printAbs(); // The named parameter 'value' is required, but there's no corresponding argument

printAbs(value: null); // The argument type 'Null' can't be assigned to the parameter type 'int'

printAbs(value: -5); // ok

final host1 = Host(); // The named parameter 'hostName' is required, but there's no corresponding argument

final host2 = Host(hostName: null); // The argument type 'Null' can't be assigned to the parameter type 'String'

final host3 = Host(hostName: "example.com"); // ok

On the flip side, if we use **nullable** instance variables we can omit the required modifier (or the default value):

class Host {

  Host({this.hostName});

  final String? hostName; // nullable, initialized to `null` by default

}

// all valid cases

final host1 = Host(); // hostName is null

final host2 = Host(hostName: null); // hostName is null

final host3 = Host(hostName: "example.com"); // hostName is non-null

**Positional** parameters are subject to the same rules:

class Host {

  Host(this.hostName); // ok

  final String hostName;

}

class Host {

  Host([this.hostName]); // The parameter 'hostName' can't have a value of 'null' because of its type, and no non-null default value is provided

  final String hostName;

}

class Host {

  Host([this.hostName = "www.codewithandrea.com"]); // ok

  final String hostName;

}

class Host {

  Host([this.hostName]); // ok

  final String? hostName;

}

Between nullable and non-nullable variables, named and positional arguments, required and default values, there's a lot to take in. If you're confused, remember the golden rule:

Non-nullable variables must always be initialized with non-null values.

To fully understand all the Null Safety features, practice using them with Dartpad. Dart will tell if you're doing something wrong - so read the error messages carefully.

[**Null-aware cascade operator**](https://codewithandrea.com/videos/dart-null-safety-ultimate-guide-non-nullable-types/#null-aware-cascade-operator)

To deal with Null Safety, the cascade operator now gains a new null-aware variant: ?... Example:

Path? path;

// will not do anything if path is null

path

  ?..moveTo(0, 0)

  ..lineTo(0, 2)

  ..lineTo(2, 2)

  ..lineTo(2, 0)

  ..lineTo(0, 0);

The cascade operations above will only be executed if path is not null.

The null-aware cascade operator can **short-circuit**, so only one ?.. operator is needed at the beginning of the sequence.

[**Null-aware subscript operator**](https://codewithandrea.com/videos/dart-null-safety-ultimate-guide-non-nullable-types/#null-aware-subscript-operator)

Up until now, checking if a collection was null before using the subscript operator was verbose:

int? first(List<int>? items) {

  return items != null ? items[0] : null; // null check to prevent runtime null errors

}

Dart 2.9 introduces the null aware operator ?[], which makes this a lot easier:

int? first(List<int>? items) {

  return items?[0];

}

[**The late keyword**](https://codewithandrea.com/videos/dart-null-safety-ultimate-guide-non-nullable-types/#the-late-keyword)

Use the late keyword to initialize a variable when it is **first read**, rather than when it's **created**.

A good example is when initializing variables in initState():

class ExampleState extends State {

  late final TextEditingController textEditingController;

  @override

  void initState() {

    super.initState();

    textEditingController = TextEditingController();

  }

}

Even better, initState() can be removed altogether:

class ExampleState extends State {

  // late - will be initialized when first used (in the build method)

  late final textEditingController = TextEditingController();

}

It's common to use late in combination with final, to **defer** the creation of **read-only** variables to when they are first read.

This is ideal when creating variables whose initializer does some heavy work:

late final taskResult = doHeavyComputation();

When used within a function body, late and final can be used like this:

void foo() {

  late final int x;

  x = 5; // ok

  x = 6; // The late final local variable is already definitely initialized

}

Though I don't recomment using late variables this way. Because this style can result in non-obvious runtime errors. Example:

class X {

  late final int x;

  void set1() => x = 1;

  void set2() => x = 2;

}

void main() {

  X x = X();

  x.set1();

  print(x.x);

  x.set2(); // LateInitializationError: Field 'x' has already been initialized.

  print(x.x);

}

By declaring a non-nullable late variable, we **promise** that it will be non-null at runtime, and Dart helps us with some compile-time guarantees.

But I recommend to only use late sparingly, and to always initialize late variables when they are declared.

[**Static and global variables**](https://codewithandrea.com/videos/dart-null-safety-ultimate-guide-non-nullable-types/#static-and-global-variables)

All global variables **must now be initialized when they are declared** unless they are late:

int global1 = 42; // ok

int global2; // The non-nullable variable 'global2' must be initialized

late int global3; // ok

The same applies to static class variables:

class Constants {

  static int x = 10; // ok

  static int y; // The non-nullable variable 'y' must be initialized

  static late int z; // ok

}

But as I said before, I do not recommend using late this way as it can lead to runtime errors.

For more information please visit the following links:

* <https://flutterbyexample.com/lesson/null-aware-operators>
* <https://dart.dev/null-safety/understanding-null-safety>

# **Constructors:**

## **Scaffold:**

Implements the basic Material Design visual layout structure.

This class provides APIs for showing drawers and bottom sheets.

To display a persistent bottom sheet, obtain the [ScaffoldState](https://api.flutter.dev/flutter/material/ScaffoldState-class.html) for the current [BuildContext](https://api.flutter.dev/flutter/widgets/BuildContext-class.html) via [Scaffold.of](https://api.flutter.dev/flutter/material/Scaffold/of.html) and use the [ScaffoldState.showBottomSheet](https://api.flutter.dev/flutter/material/ScaffoldState/showBottomSheet.html) function.

Example:

import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  static const String \_title = 'Flutter Code Sample';

  @override

  Widget build(BuildContext context) {

    return const MaterialApp(

    debugShowCheckedModeBanner: false,

      title: \_title,

      home: MyStatefulWidget(),

    );

  }

}

class MyStatefulWidget extends StatefulWidget {

  const MyStatefulWidget({super.key});

  @override

  State<MyStatefulWidget> createState() => \_MyStatefulWidgetState();

}

class \_MyStatefulWidgetState extends State<MyStatefulWidget> {

  int \_count = 0;

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: const Text('Sample Code'),

      ),

      body: Center(child: Text('You have pressed the button $\_count times.')),

      floatingActionButton: FloatingActionButton(

        onPressed: () => setState(() => \_count++),

        tooltip: 'Increment Counter',

        child: const Icon(Icons.add),

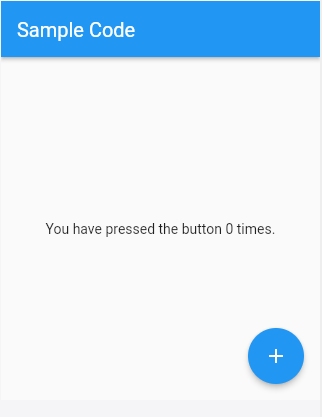
      ),

    );

  }

}

Output:



## **MaterialApp**

An application that uses Material Design.

A convenience widget that wraps a number of widgets that are commonly required for Material Design applications. It builds upon a [WidgetsApp](https://api.flutter.dev/flutter/widgets/WidgetsApp-class.html) by adding material-design specific functionality, such as [AnimatedTheme](https://api.flutter.dev/flutter/material/AnimatedTheme-class.html) and [GridPaper](https://api.flutter.dev/flutter/widgets/GridPaper-class.html).

[MaterialApp](https://api.flutter.dev/flutter/material/MaterialApp-class.html) configures its [WidgetsApp.textStyle](https://api.flutter.dev/flutter/widgets/WidgetsApp/textStyle.html) with an ugly red/yellow text style that's intended to warn the developer that their app hasn't defined a default text style. Typically the app's [Scaffold](https://api.flutter.dev/flutter/material/Scaffold-class.html) builds a [Material](https://api.flutter.dev/flutter/material/Material-class.html) widget whose default [Material.textStyle](https://api.flutter.dev/flutter/material/Material/textStyle.html) defines the text style for the entire scaffold.

The [MaterialApp](https://api.flutter.dev/flutter/material/MaterialApp-class.html) configures the top-level [Navigator](https://api.flutter.dev/flutter/widgets/Navigator-class.html) to search for routes in the following order:

1. For the / route, the [home](https://api.flutter.dev/flutter/material/MaterialApp/home.html) property, if non-null, is used.
2. Otherwise, the [routes](https://api.flutter.dev/flutter/material/MaterialApp/routes.html) table is used, if it has an entry for the route.
3. Otherwise, [onGenerateRoute](https://api.flutter.dev/flutter/material/MaterialApp/onGenerateRoute.html) is called, if provided. It should return a non-null value for any valid route not handled by [home](https://api.flutter.dev/flutter/material/MaterialApp/home.html) and [routes](https://api.flutter.dev/flutter/material/MaterialApp/routes.html).
4. Finally if all else fails [onUnknownRoute](https://api.flutter.dev/flutter/material/MaterialApp/onUnknownRoute.html) is called.

If a [Navigator](https://api.flutter.dev/flutter/widgets/Navigator-class.html) is created, at least one of these options must handle the / route, since it is used when an invalid [initialRoute](https://api.flutter.dev/flutter/material/MaterialApp/initialRoute.html) is specified on startup (e.g. by another application launching this one with an intent on Android; see [dart:ui.PlatformDispatcher.defaultRouteName](https://api.flutter.dev/flutter/dart-ui/PlatformDispatcher/defaultRouteName.html)).

This widget also configures the observer of the top-level [Navigator](https://api.flutter.dev/flutter/widgets/Navigator-class.html) (if any) to perform [Hero](https://api.flutter.dev/flutter/widgets/Hero-class.html) animations.

The [MaterialApp](https://api.flutter.dev/flutter/material/MaterialApp-class.html) automatically creates a [DefaultSelectionStyle](https://api.flutter.dev/flutter/widgets/DefaultSelectionStyle-class.html). It uses the colors in the [ThemeData.textSelectionTheme](https://api.flutter.dev/flutter/material/ThemeData/textSelectionTheme.html) if they are not null; otherwise, the [MaterialApp](https://api.flutter.dev/flutter/material/MaterialApp-class.html) sets [DefaultSelectionStyle.selectionColor](https://api.flutter.dev/flutter/widgets/DefaultSelectionStyle/selectionColor.html) to [ColorScheme.primary](https://api.flutter.dev/flutter/material/ColorScheme/primary.html) with 0.4 opacity and [DefaultSelectionStyle.cursorColor](https://api.flutter.dev/flutter/widgets/DefaultSelectionStyle/cursorColor.html) to [ColorScheme.primary](https://api.flutter.dev/flutter/material/ColorScheme/primary.html).

If [home](https://api.flutter.dev/flutter/material/MaterialApp/home.html), [routes](https://api.flutter.dev/flutter/material/MaterialApp/routes.html), [onGenerateRoute](https://api.flutter.dev/flutter/material/MaterialApp/onGenerateRoute.html), and [onUnknownRoute](https://api.flutter.dev/flutter/material/MaterialApp/onUnknownRoute.html) are all null, and [builder](https://api.flutter.dev/flutter/material/MaterialApp/builder.html) is not null, then no [Navigator](https://api.flutter.dev/flutter/widgets/Navigator-class.html) is created.

Example:

import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  static const String \_title = 'Flutter Code Sample';

  @override

  Widget build(BuildContext context) {

    return const MaterialApp(

      debugShowCheckedModeBanner: false,

      title: \_title,

      home: MyStatefulWidget(),

    );

  }

}

class MyStatefulWidget extends StatefulWidget {

  const MyStatefulWidget({super.key});

  @override

  State<MyStatefulWidget> createState() => \_MyStatefulWidgetState();

}

class \_MyStatefulWidgetState extends State<MyStatefulWidget> {

  int \_count = 0;

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      theme:

          ThemeData(brightness: Brightness.dark, primaryColor: Colors.blueGrey),

      home: Scaffold(

        appBar: AppBar(

          title: const Text('MaterialApp Theme'),

        ),

      ),

    );

  }

}

Each application page can be divided into three section which are:

* 1. Appbar
  2. Body
  3. BottomNavigationBar

## **AppBar**

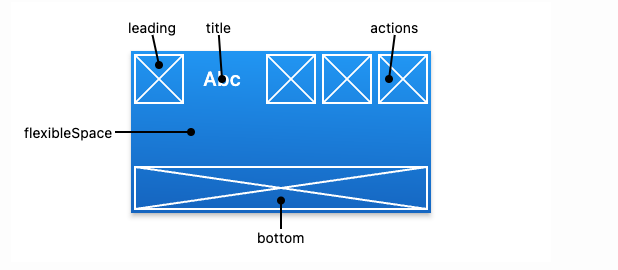
A Material Design app bar.

An app bar consists of a toolbar and potentially other widgets, such as a [TabBar](https://api.flutter.dev/flutter/material/TabBar-class.html) and a [FlexibleSpaceBar](https://api.flutter.dev/flutter/material/FlexibleSpaceBar-class.html). App bars typically expose one or more common [actions](https://api.flutter.dev/flutter/material/AppBar/actions.html) with [IconButton](https://api.flutter.dev/flutter/material/IconButton-class.html)s which are optionally followed by a [PopupMenuButton](https://api.flutter.dev/flutter/material/PopupMenuButton-class.html) for less common operations (sometimes called the "overflow menu").

App bars are typically used in the [Scaffold.appBar](https://api.flutter.dev/flutter/material/Scaffold/appBar.html) property, which places the app bar as a fixed-height widget at the top of the screen. For a scrollable app bar, see [SliverAppBar](https://api.flutter.dev/flutter/material/SliverAppBar-class.html), which embeds an [AppBar](https://api.flutter.dev/flutter/material/AppBar-class.html) in a sliver for use in a [CustomScrollView](https://api.flutter.dev/flutter/widgets/CustomScrollView-class.html).

The AppBar displays the toolbar widgets, [leading](https://api.flutter.dev/flutter/material/AppBar/leading.html), [title](https://api.flutter.dev/flutter/material/AppBar/title.html), and [actions](https://api.flutter.dev/flutter/material/AppBar/actions.html), above the [bottom](https://api.flutter.dev/flutter/material/AppBar/bottom.html) (if any). The [bottom](https://api.flutter.dev/flutter/material/AppBar/bottom.html) is usually used for a [TabBar](https://api.flutter.dev/flutter/material/TabBar-class.html). If a [flexibleSpace](https://api.flutter.dev/flutter/material/AppBar/flexibleSpace.html) 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](https://api.flutter.dev/flutter/material/AppBar-class.html) insets its content based on the ambient [MediaQuery](https://api.flutter.dev/flutter/widgets/MediaQuery-class.html)'s padding, to avoid system UI intrusions. It's taken care of by [Scaffold](https://api.flutter.dev/flutter/material/Scaffold-class.html) when used in the [Scaffold.appBar](https://api.flutter.dev/flutter/material/Scaffold/appBar.html) property. When animating an [AppBar](https://api.flutter.dev/flutter/material/AppBar-class.html), unexpected [MediaQuery](https://api.flutter.dev/flutter/widgets/MediaQuery-class.html) changes (as is common in [Hero](https://api.flutter.dev/flutter/widgets/Hero-class.html) animations) may cause the content to suddenly jump. Wrap the [AppBar](https://api.flutter.dev/flutter/material/AppBar-class.html) in a [MediaQuery](https://api.flutter.dev/flutter/widgets/MediaQuery-class.html) widget, and adjust its padding such that the animation is smooth.



If the [leading](https://api.flutter.dev/flutter/material/AppBar/leading.html) widget is omitted, but the [AppBar](https://api.flutter.dev/flutter/material/AppBar-class.html) is in a [Scaffold](https://api.flutter.dev/flutter/material/Scaffold-class.html) with a [Drawer](https://api.flutter.dev/flutter/material/Drawer-class.html), then a button will be inserted to open the drawer. Otherwise, if the nearest [Navigator](https://api.flutter.dev/flutter/widgets/Navigator-class.html) has any previous routes, a [BackButton](https://api.flutter.dev/flutter/material/BackButton-class.html) is inserted instead. This behavior can be turned off by setting the [automaticallyImplyLeading](https://api.flutter.dev/flutter/material/AppBar/automaticallyImplyLeading.html) to false. In that case a null leading widget will result in the middle/title widget stretching to start.

Example:

import 'package:flutter/material.dart';

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

class AppBarApp extends StatelessWidget {

  const AppBarApp({super.key});

  @override

  Widget build(BuildContext context) {

    return const MaterialApp(

      debugShowCheckedModeBanner: false,

      home: AppBarExample(),

    );

  }

}

class AppBarExample extends StatelessWidget {

  const AppBarExample({super.key});

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: const Text('AppBar Demo'),

        actions: <Widget>[

          IconButton(

            icon: const Icon(Icons.add\_alert),

            tooltip: 'Show Snackbar',

            onPressed: () {

              ScaffoldMessenger.of(context).showSnackBar(

                  const SnackBar(content: Text('This is a snackbar')));

            },

          ),

          IconButton(

            icon: const Icon(Icons.navigate\_next),

            tooltip: 'Go to the next page',

            onPressed: () {

              Navigator.push(context, MaterialPageRoute<void>(

                builder: (BuildContext context) {

                  return Scaffold(

                    appBar: AppBar(

                      title: const Text('Next page'),

                    ),

                    body: const Center(

                      child: Text(

                        'This is the next page',

                        style: TextStyle(fontSize: 24),

                      ),

                    ),

                  );

                },

              ));

            },

          ),

        ],

      ),

      body: const Center(

        child: Text(

          'This is the home page',

          style: TextStyle(fontSize: 24),

        ),

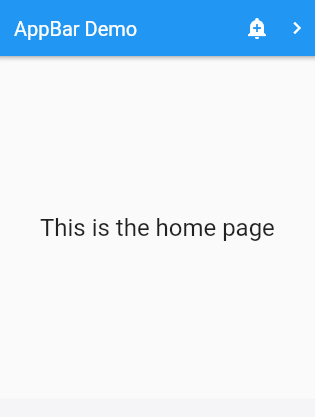
      ),

    );

  }

}

OutPut:



## **AppBarTheme** **class**

Overrides the default values of visual properties for descendant [AppBar](https://api.flutter.dev/flutter/material/AppBar-class.html) widgets.

Descendant widgets obtain the current [AppBarTheme](https://api.flutter.dev/flutter/material/AppBarTheme-class.html) object with AppBarTheme.of(context). Instances of [AppBarTheme](https://api.flutter.dev/flutter/material/AppBarTheme-class.html) can be customized with [AppBarTheme.copyWith](https://api.flutter.dev/flutter/material/AppBarTheme/copyWith.html).

Typically an [AppBarTheme](https://api.flutter.dev/flutter/material/AppBarTheme-class.html) is specified as part of the overall [Theme](https://api.flutter.dev/flutter/material/Theme-class.html) with [ThemeData.appBarTheme](https://api.flutter.dev/flutter/material/ThemeData/appBarTheme.html).

All [AppBarTheme](https://api.flutter.dev/flutter/material/AppBarTheme-class.html) properties are null by default. When null, the [AppBar](https://api.flutter.dev/flutter/material/AppBar-class.html) compute its own default values, typically based on the overall theme's [ThemeData.colorScheme](https://api.flutter.dev/flutter/material/ThemeData/colorScheme.html), [ThemeData.textTheme](https://api.flutter.dev/flutter/material/ThemeData/textTheme.html), and [ThemeData.iconTheme](https://api.flutter.dev/flutter/material/ThemeData/iconTheme.html).

Example:

import 'package:flutter/material.dart';

class MyWidget extends StatelessWidget {

  const MyWidget({super.key});

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      title: 'Theme Widget',

      debugShowCheckedModeBanner: false,

      theme: ThemeData(

        colorSchemeSeed: Colors.red,

        brightness: Brightness.dark,

        scaffoldBackgroundColor: Colors.black,

        fontFamily: 'Anton',

        appBarTheme: const AppBarTheme(

          backgroundColor: Colors.indigo,

          centerTitle: true,

          toolbarHeight: 90,

        ),

      ),

    );

  }

}

For more info please visit: <https://api.flutter.dev/flutter/material/AppBar-class.html>

## **Body**

This is the most important section of the flutter application as it can contain different types of constructors and components. Most of the time the application can be build only with the body section. It can consist:

* Container
* Colum
* Row
* Elevated button

For more constructors, please read the **Components** Section.

## **Container**

A convenience widget that combines common painting, positioning, and sizing widgets.

A container first surrounds the child with [padding](https://api.flutter.dev/flutter/widgets/Container/padding.html) (inflated by any borders present in the [decoration](https://api.flutter.dev/flutter/widgets/Container/decoration.html)) and then applies additional [constraints](https://api.flutter.dev/flutter/widgets/Container/constraints.html) to the padded extent (incorporating the width and height as constraints, if either is non-null). The container is then surrounded by additional empty space described from the [margin](https://api.flutter.dev/flutter/widgets/Container/margin.html).

During painting, the container first applies the given [transform](https://api.flutter.dev/flutter/widgets/Container/transform.html), then paints the [decoration](https://api.flutter.dev/flutter/widgets/Container/decoration.html) to fill the padded extent, then it paints the child, and finally paints the [foregroundDecoration](https://api.flutter.dev/flutter/widgets/Container/foregroundDecoration.html), also filling the padded extent.

Containers with no children try to be as big as possible unless the incoming constraints are unbounded, in which case they try to be as small as possible. Containers with children size themselves to their children. The width, height, and [constraints](https://api.flutter.dev/flutter/widgets/Container/constraints.html) arguments to the constructor override this.

Example:

Center(

child: Container (

margin: **const** EdgeInsets.all(10.0),

color: Colors.amber[600],

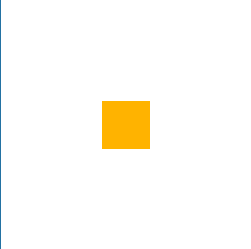
width: 48.0,

height: 48.0,

),

)

Output



For more detail please visit: <https://api.flutter.dev/flutter/widgets/Container-class.html>

## **Constructor (Custom made):**

# **Components:**

Flutter have a lot of amazing inbuild packages which can be used to upgrade the user display and easily add different kind of components with minimal effort. For our upcoming projects we have used these packages as components to achieve our target of the project.

## **Clay Container:**

**Step 1:**

Create a new project in the VS code. To do that press ctrl + shift + P

Then click on Flutter: new project

Then choose Application

Now choose a place to save the project and then name the project.

**Step 2:**

After creating a new project, you will a have a default template.

Now go to the VS code terminal and run the following command:

flutter pub add clay\_containers

it will add the necessary package in your project’s pubspec.yaml file’s dependency.

**Step 3:**

Now delete the default flutter template from the main.dart and type the following code:

import 'package:flutter/material.dart';

import 'package:clay\_containers/clay\_containers.dart';

void main() {

  runApp(MyApp());

}

class MyApp extends StatelessWidget {

  MyApp({super.key});

  Color baseColor = Color(0xFFF2F2F2);

  @override

  Widget build(BuildContext context) {

    return Container(

      color: baseColor,

      child: Center(

        child: ClayContainer(

          color: baseColor,

          height: 200,

          width: 200,

        ),

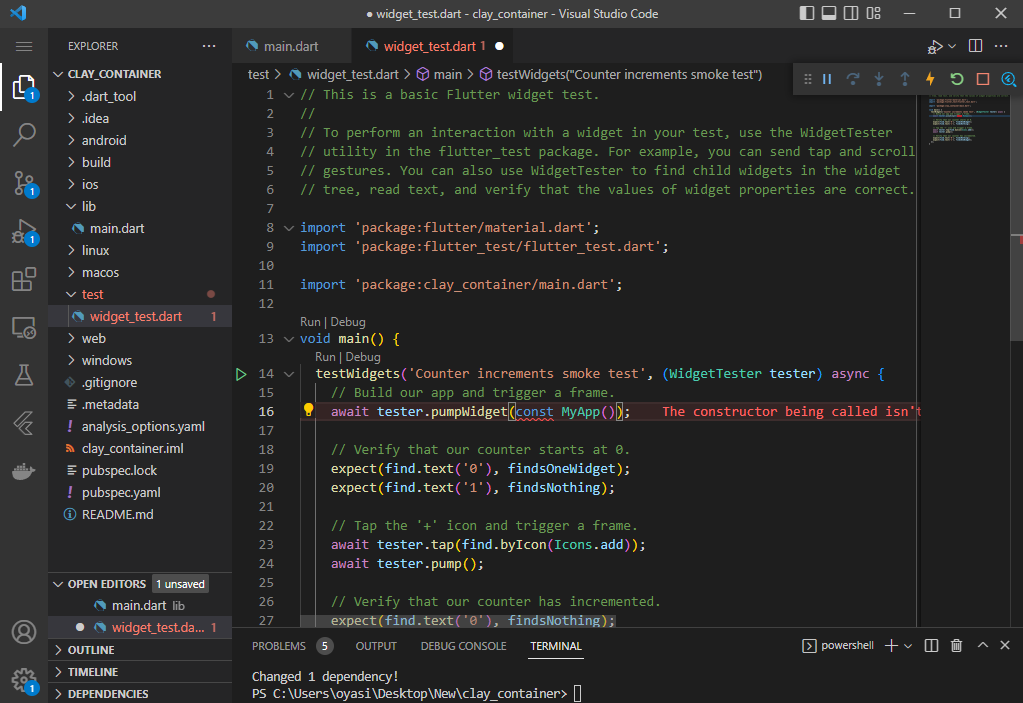
      ),

    );

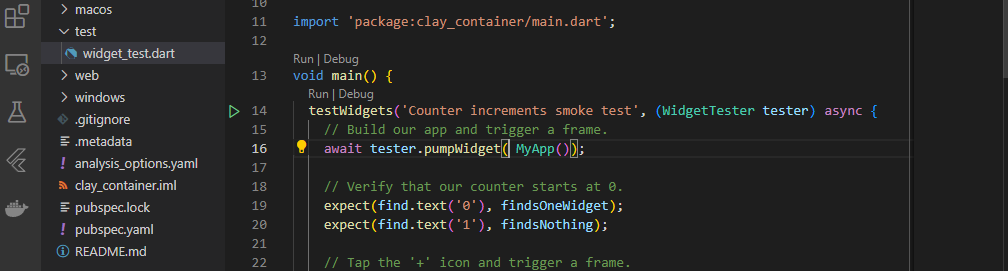
  }

}

After this you will find an error in your widget\_test.dart file it is

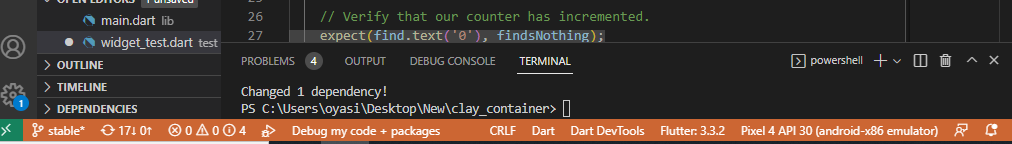


It is because you can’t use the constant any more. The reason is MyApp constructor is not a constant anymore in the app. Remove const and it will solve the issue.



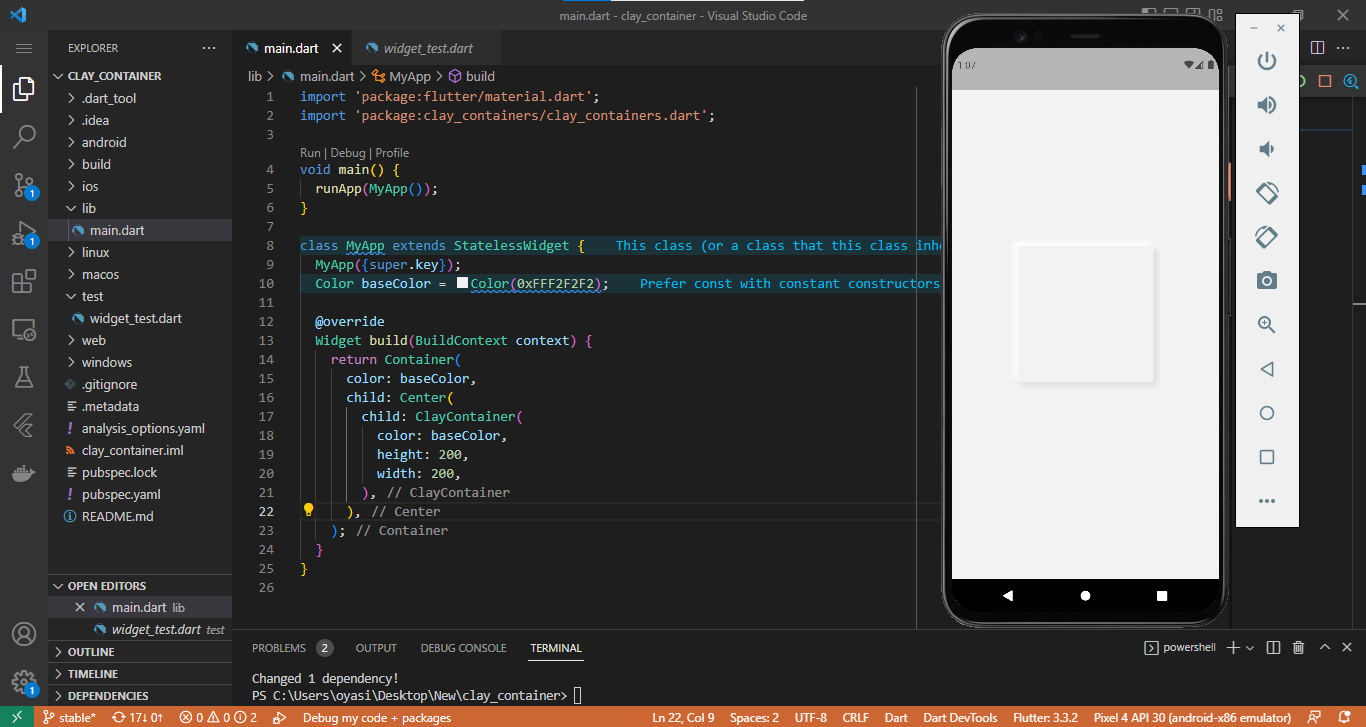
**Step 4:**

Now choose a device in your VS code



Now run the project by pressing ctrl + f5

Output :



You will be able to see this output in your emulator.

## **ListView**

**Step 1:**

Create a new project in the VS code. To do that press ctrl + shift + P

Then click on Flutter: new project

Then choose Application

Now choose a place to save the project and then name the project.

**Step 2:**

Now delete the default flutter template from the main.dart and type the following code:

import 'package:flutter/material.dart';

void main() {

  runApp(const MyApp());

}

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  @override

  Widget build(BuildContext context) {

    return const MaterialApp(

      home: HomePage(),

    );

  }

}

class HomePage extends StatelessWidget {

  const HomePage({super.key});

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      body: ListView(

        children: [

          Padding(

            padding: const EdgeInsets.all(8.0),

            child: Container(

              height: 200,

              color: Colors.lightBlueAccent,

              child: Center(

                child: Text(

                  '1st box',

                  style: TextStyle(

                      color: Colors.white,

                      fontSize: 32,

                      fontWeight: FontWeight.w600),

                ),

              ),

            ),

          ),

          Padding(

            padding: const EdgeInsets.all(8.0),

            child: Container(

              height: 200,

              color: Colors.deepOrangeAccent,

              child: const Center(

                child: Text(

                  '2nd box',

                  style: TextStyle(

                      color: Colors.white,

                      fontSize: 32,

                      fontWeight: FontWeight.w600),

                ),

              ),

            ),

          ),

          Padding(

            padding: const EdgeInsets.all(8.0),

            child: Container(

              height: 200,

              color: Colors.deepPurpleAccent,

              child: const Center(

                child: Text(

                  '3rd box',

                  style: TextStyle(

                      color: Colors.white,

                      fontSize: 32,

                      fontWeight: FontWeight.w600),

                ),

              ),

            ),

          ),

          Padding(

            padding: const EdgeInsets.all(8.0),

            child: Container(

              height: 200,

              color: Colors.black12,

              child: const Center(

                child: Text(

                  '4th box',

                  style: TextStyle(

                      color: Colors.white,

                      fontSize: 32,

                      fontWeight: FontWeight.w600),

                ),

              ),

            ),

          ),

        ],

      ),

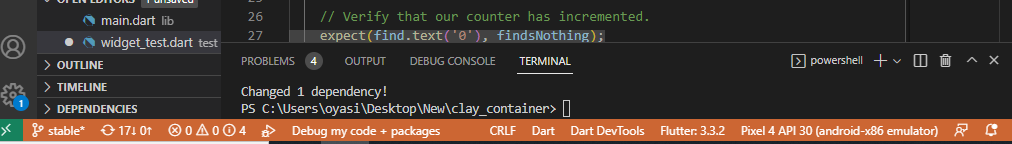
    );

  }

}

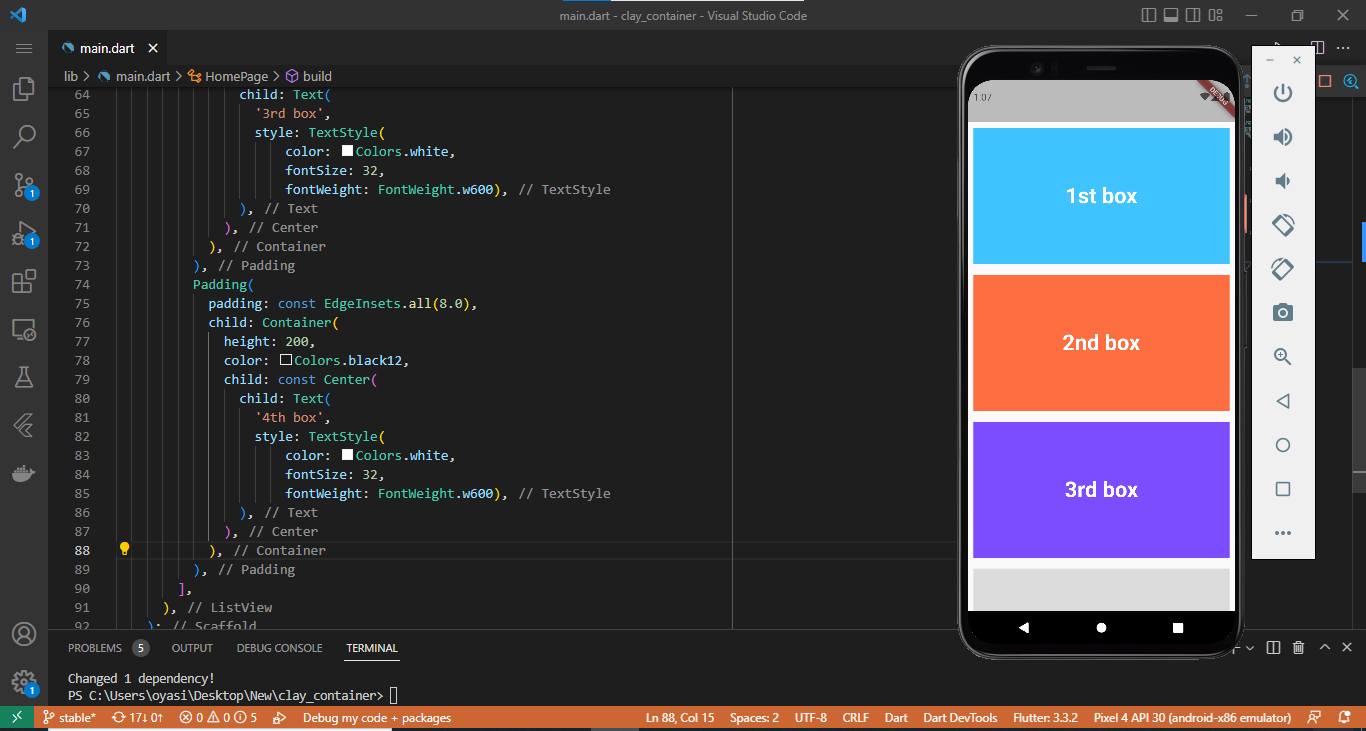
**Step 3:**

Now choose a device in your VS code



Now run the project by pressing ctrl + f5

**Output:**



## **GridView**

**Step 1:**

Create a new project in the VS code. To do that press ctrl + shift + P

Then click on Flutter: new project

Then choose Application

Now choose a place to save the project and then name the project.

**Step 2:**

Now delete the default flutter template from the main.dart and type the following code:

import 'package:flutter/material.dart';

void main() {

  runApp(const MyApp());

}

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  @override

  Widget build(BuildContext context) {

    return const MaterialApp(

      home: HomePage(),

    );

  }

}

class HomePage extends StatelessWidget {

  const HomePage({super.key});

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      body: GridView.count(

        crossAxisCount: 2,

        children: [

          Padding(

            padding: const EdgeInsets.all(8.0),

            child: Container(

              height: 200,

              color: Colors.lightBlueAccent,

              child: Center(

                child: Text(

                  '1st box',

                  style: TextStyle(

                      color: Colors.white,

                      fontSize: 32,

                      fontWeight: FontWeight.w600),

                ),

              ),

            ),

          ),

          Padding(

            padding: const EdgeInsets.all(8.0),

            child: Container(

              height: 200,

              color: Colors.deepOrangeAccent,

              child: const Center(

                child: Text(

                  '2nd box',

                  style: TextStyle(

                      color: Colors.white,

                      fontSize: 32,

                      fontWeight: FontWeight.w600),

                ),

              ),

            ),

          ),

          Padding(

            padding: const EdgeInsets.all(8.0),

            child: Container(

              height: 200,

              color: Colors.deepPurpleAccent,

              child: const Center(

                child: Text(

                  '3rd box',

                  style: TextStyle(

                      color: Colors.white,

                      fontSize: 32,

                      fontWeight: FontWeight.w600),

                ),

              ),

            ),

          ),

          Padding(

            padding: const EdgeInsets.all(8.0),

            child: Container(

              height: 200,

              color: Colors.black12,

              child: const Center(

                child: Text(

                  '4th box',

                  style: TextStyle(

                      color: Colors.white,

                      fontSize: 32,

                      fontWeight: FontWeight.w600),

                ),

              ),

            ),

          ),

          Padding(

            padding: const EdgeInsets.all(8.0),

            child: Container(

              height: 200,

              color: Colors.amberAccent,

              child: const Center(

                child: Text(

                  '5th box',

                  style: TextStyle(

                      color: Colors.white,

                      fontSize: 32,

                      fontWeight: FontWeight.w600),

                ),

              ),

            ),

          ),

          Padding(

            padding: const EdgeInsets.all(8.0),

            child: Container(

              height: 200,

              color: Colors.orangeAccent,

              child: const Center(

                child: Text(

                  '6th box',

                  style: TextStyle(

                      color: Colors.white,

                      fontSize: 32,

                      fontWeight: FontWeight.w600),

                ),

              ),

            ),

          ),

          Padding(

            padding: const EdgeInsets.all(8.0),

            child: Container(

              height: 200,

              color: Colors.blueGrey,

              child: const Center(

                child: Text(

                  '7th box',

                  style: TextStyle(

                      color: Colors.white,

                      fontSize: 32,

                      fontWeight: FontWeight.w600),

                ),

              ),

            ),

          ),

          Padding(

            padding: const EdgeInsets.all(8.0),

            child: Container(

              height: 200,

              color: Colors.yellowAccent,

              child: const Center(

                child: Text(

                  '8th box',

                  style: TextStyle(

                      color: Colors.white,

                      fontSize: 32,

                      fontWeight: FontWeight.w600),

                ),

              ),

            ),

          ),

        ],

      ),

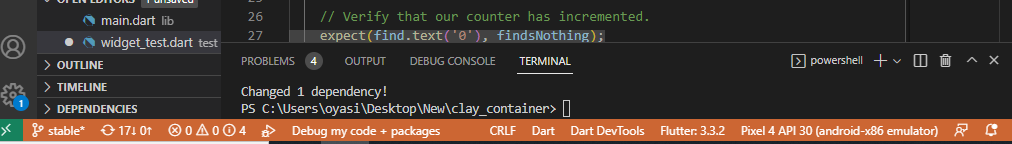
    );

  }

}

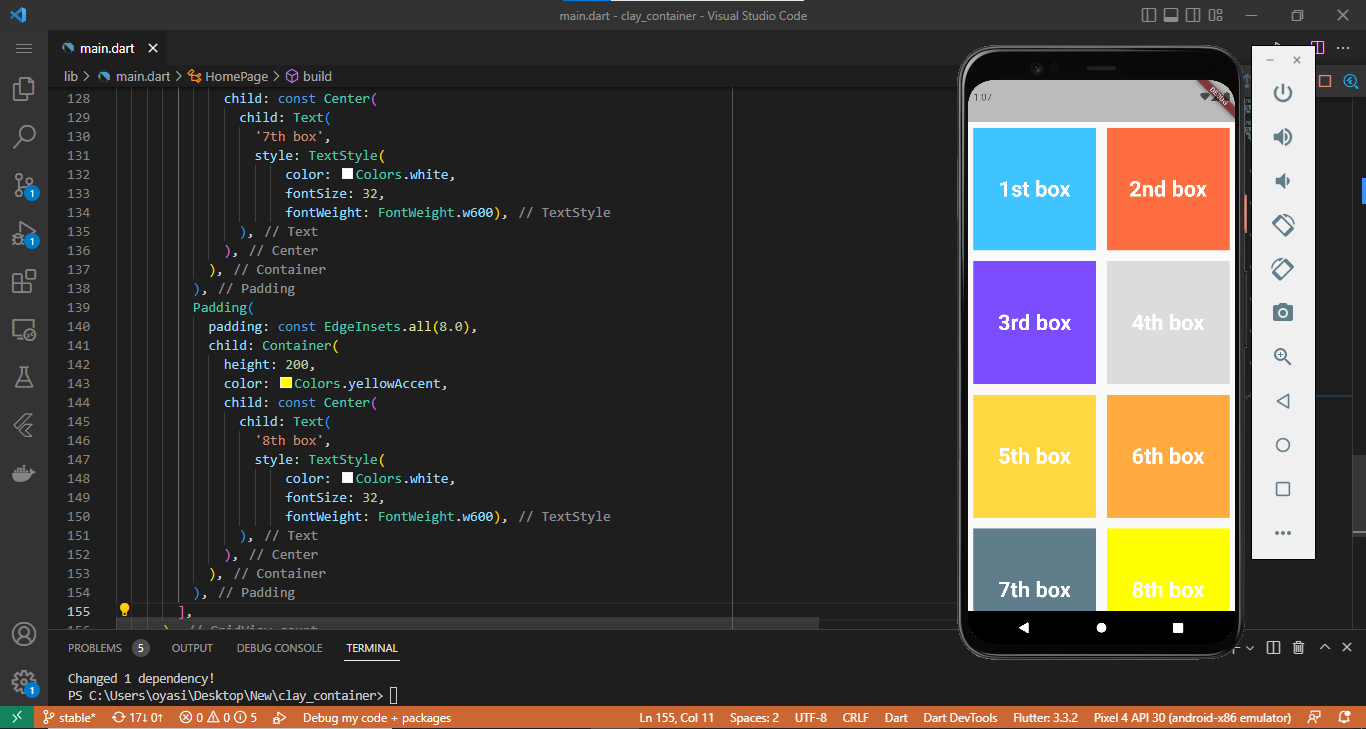
**Step 3:**

Now choose a device in your VS code



Now run the project by pressing ctrl + f5

**Output:**



# **Advance Deployment**

## **Neumorphism**

**Step 1:**

Create a new project in the VS code. To do that press ctrl + shift + P

Then click on Flutter: new project

Then choose Application

Now choose a place to save the project and then name the project.

**Step 2:**

Now delete the default flutter template from the main.dart and type the following code:

import 'package:flutter/material.dart' hide BoxDecoration, BoxShadow;

import 'package:flutter\_inset\_box\_shadow/flutter\_inset\_box\_shadow.dart';

void main() {

  runApp(const MyApp());

}

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  // This widget is the root of your application.

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      title: 'Flutter Demo',

      debugShowCheckedModeBanner: false,

      theme: ThemeData(

        primarySwatch: Colors.blue,

      ),

      home: const homepage(),

    );

  }

}

class homepage extends StatefulWidget {

  const homepage({super.key});

  @override

  State<homepage> createState() => \_homepageState();

}

class \_homepageState extends State<homepage> {

  bool isPressed = true;

  bool isDarkMode = false;

  @override

  Widget build(BuildContext context) {

    final backgroundColor =

        isDarkMode ? const Color(0xFF2E3239) : const Color(0xFFE7ECEF);

    Offset distance = isPressed ? Offset(10, 10) : Offset(28, 28);

    double blur = isPressed ? 5.0 : 30.0;

    return Scaffold(

      backgroundColor: backgroundColor,

      body: Center(

        child: Listener(

          onPointerUp: (\_) => setState(() => isPressed = false),

          onPointerDown: (\_) => setState(() => isPressed = true),

          child: AnimatedContainer(

            duration: const Duration(microseconds: 100),

            decoration: BoxDecoration(

                borderRadius: BorderRadius.circular(30),

                color: backgroundColor,

                boxShadow: isPressed

                    ? []

                    : [

                        BoxShadow(

                          blurRadius: blur,

                          offset: -distance,

                          color: isDarkMode ? Color(0xFF35393F) : Colors.white,

                          inset: isPressed,

                        ),

                        BoxShadow(

                          blurRadius: blur,

                          offset: distance,

                          color: isDarkMode

                              ? Color(0xFF23262A)

                              : Color(0xFFA7A9AF),

                          inset: isPressed,

                        )

                      ]),

            child: SizedBox(

              height: 200,

              width: 200,

            ),

          ),

        ),

      ),

    );

  }

}

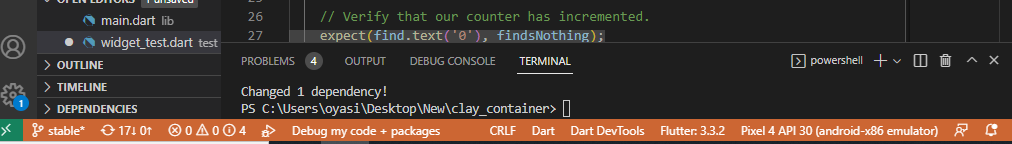
**Step 3 :**

Run the following command in the Vs code terminal to add the necessary dependency in your pubspec.yaml.

flutter pub add flutter\_inset\_box\_shadow

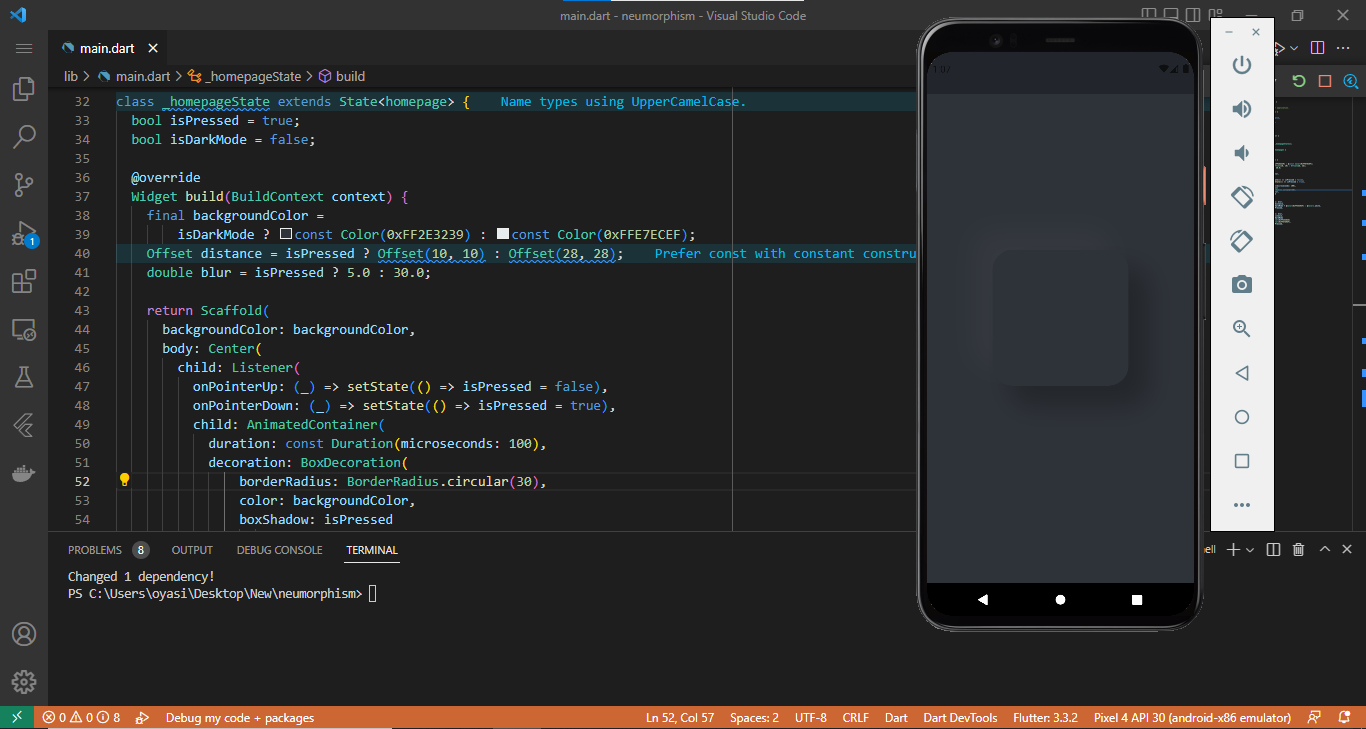
**Step 4:**

Now choose a device in your VS code



Now run the project by pressing ctrl + f5

**Output:**



## **Drawer:**

**Step 1:**

Create a new project in the VS code. To do that press ctrl + shift + P

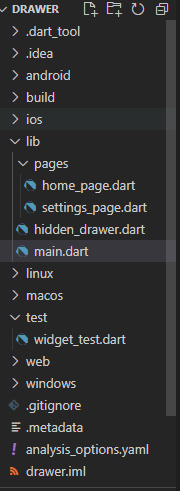
Then click on Flutter: new project

Then choose Application

Now choose a place to save the project and then name the project.

Ste 2:

Now to organize our project well we need to create some files and organize it as shown below:



**Step 3:**

Now delete the default flutter template from the main.dart and type the following code:

import 'package:drawer/hidden\_drawer.dart';

import 'package:flutter/material.dart';

void main() {

  runApp(const MyApp());

}

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      home: const HiddenDrawer(),

      theme: ThemeData(primarySwatch: Colors.deepPurple),

    );

  }

}

Now include the code below in the hidden\_drawer.dart :

import 'package:drawer/pages/home\_page.dart';

import 'package:drawer/pages/settings\_page.dart';

import 'package:flutter/material.dart';

import 'package:hidden\_drawer\_menu/hidden\_drawer\_menu.dart';

class HiddenDrawer extends StatefulWidget {

  const HiddenDrawer({super.key});

  @override

  State<HiddenDrawer> createState() => \_HiddenDrawerState();

}

class \_HiddenDrawerState extends State<HiddenDrawer> {

  List<ScreenHiddenDrawer> \_pages = [];

  final mystyle = TextStyle(

    fontWeight: FontWeight.bold,

    fontSize: 18,

    color: Colors.white,

  );

  @override

  void initState() {

    super.initState();

    \_pages = [

      ScreenHiddenDrawer(

        ItemHiddenMenu(

          name: 'Homepage',

          baseStyle: mystyle,

          selectedStyle: TextStyle(),

        ),

        HomePage(),

      ),

      ScreenHiddenDrawer(

        ItemHiddenMenu(

          name: 'Settings',

          baseStyle: mystyle,

          selectedStyle: TextStyle(),

        ),

        SettingsPage(),

      ),

    ];

  }

  @override

  Widget build(BuildContext context) {

    return HiddenDrawerMenu(

      backgroundColorMenu: Colors.deepPurple.shade200,

      screens: \_pages,

      initPositionSelected: 0,

      slidePercent: 50,

    );

  }

}

Now include the code below in the home\_page.dart :

import 'package:flutter/material.dart';

class HomePage extends StatefulWidget {

  const HomePage({super.key});

  @override

  State<HomePage> createState() => \_HomePageState();

}

class \_HomePageState extends State<HomePage> {

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      backgroundColor: Colors.blueAccent,

    );

  }

}

Now include the code below in the settings\_page.dart :

import 'package:flutter/material.dart';

class SettingsPage extends StatefulWidget {

  const SettingsPage({super.key});

  @override

  State<SettingsPage> createState() => \_SettingsPageState();

}

class \_SettingsPageState extends State<SettingsPage> {

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      backgroundColor: Colors.amberAccent,

    );

  }

}

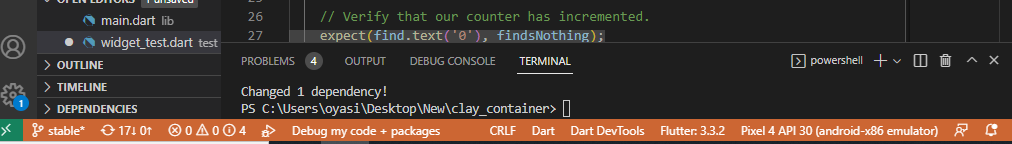
**Step 4:**

Run the following command in the Vs code terminal to add the necessary dependency in your pubspec.yaml.

flutter pub add hidden\_drawer\_menu

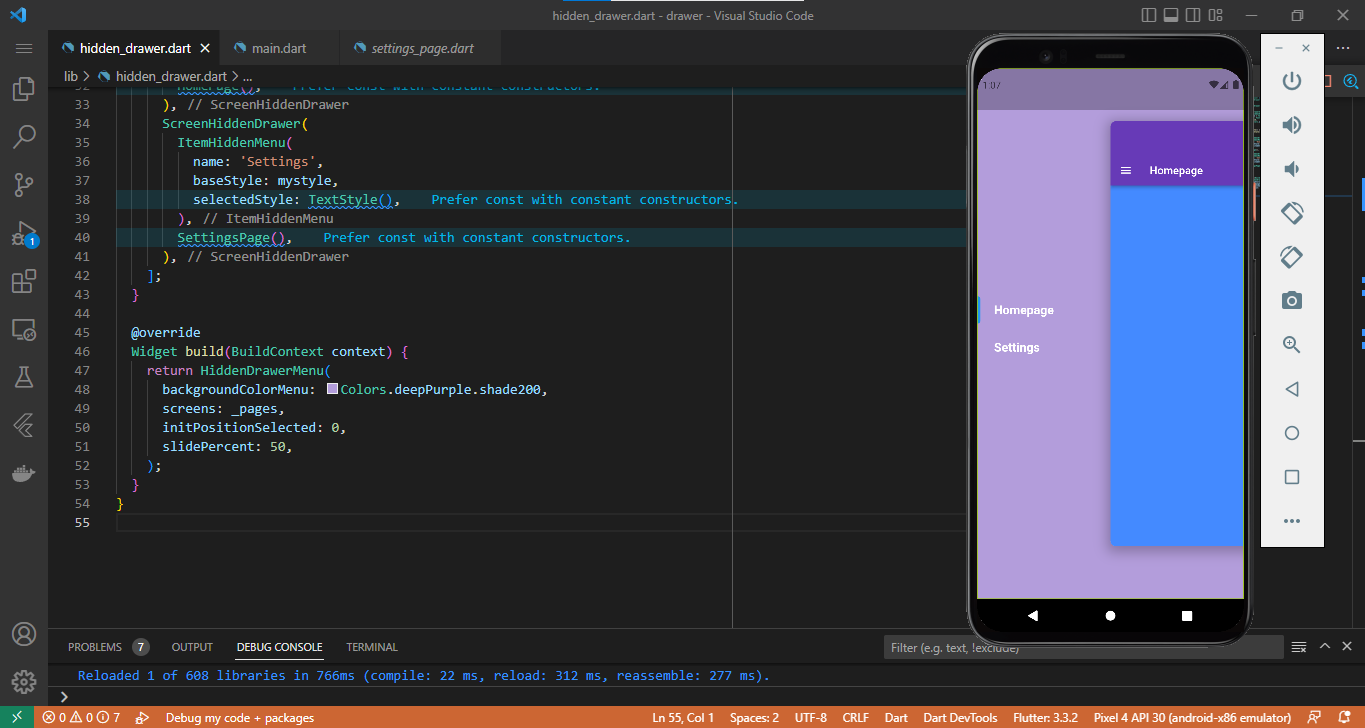
**Step 5:**

Now choose a device in your VS code



Now run the project by pressing ctrl + f5

**Output:**



## **ReorderableListView**

**Step 1:**

Create a new project in the VS code. To do that press ctrl + shift + P

Then click on Flutter: new project

Then choose Application

Now choose a place to save the project and then name the project.

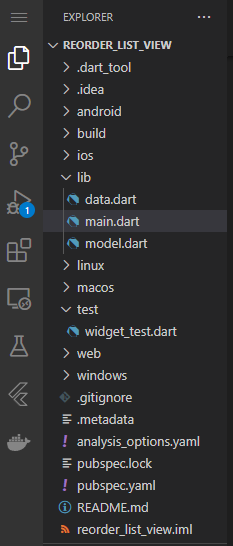
**Step 2:**

Run the following command in the Vs code terminal to add the necessary dependency in your pubspec.yaml.

flutter pub add drag\_and\_drop\_lists

**Step 3:**

The file structure for this project should be like:



**Step 4:**

Now delete the default flutter template from the main.dart and type the following code:

import 'package:drag\_and\_drop\_lists/drag\_and\_drop\_lists.dart';

import 'data.dart';

import 'package:flutter/material.dart';

import 'package:flutter/services.dart';

import 'model.dart';

Future main() async {

  WidgetsFlutterBinding.ensureInitialized();

  await SystemChrome.setPreferredOrientations([

    DeviceOrientation.portraitUp,

    DeviceOrientation.portraitDown,

  ]);

  runApp(const MyApp());

}

class MyApp extends StatelessWidget {

  static const String title = 'Drag & Drop ListView';

  const MyApp({super.key});

  @override

  Widget build(BuildContext context) => MaterialApp(

        debugShowCheckedModeBanner: false,

        title: title,

        theme: ThemeData(primarySwatch: Colors.red),

        home: MainPage(),

      );

}

class MainPage extends StatefulWidget {

  const MainPage({super.key});

  @override

  \_MainPage createState() => \_MainPage();

}

class \_MainPage extends State<MainPage> {

  late List<DragAndDropList> lists;

  @override

  void initState() {

    super.initState();

    lists = allLists.map(buildList).toList();

  }

  @override

  Widget build(BuildContext context) {

    const backgroundColor = Color.fromARGB(255, 243, 242, 248);

    return Scaffold(

      backgroundColor: backgroundColor,

      appBar: AppBar(

        title: const Text(MyApp.title),

        centerTitle: true,

      ),

      body: DragAndDropLists(

        // lastItemTargetHeight: 50,

        // addLastItemTargetHeightToTop: true,

        // lastListTargetSize: 30,

        listPadding: const EdgeInsets.all(16),

        listInnerDecoration: BoxDecoration(

          color: Theme.of(context).canvasColor,

          borderRadius: BorderRadius.circular(10),

        ),

        children: lists,

        itemDivider:

            const Divider(thickness: 2, height: 2, color: backgroundColor),

        itemDecorationWhileDragging: const BoxDecoration(

          color: Colors.white,

          boxShadow: [BoxShadow(color: Colors.black12, blurRadius: 4)],

        ),

        listDragHandle: buildDragHandle(isList: true),

        itemDragHandle: buildDragHandle(),

        onItemReorder: onReorderListItem,

        onListReorder: onReorderList,

      ),

    );

  }

  DragHandle buildDragHandle({bool isList = false}) {

    final verticalAlignment = isList

        ? DragHandleVerticalAlignment.top

        : DragHandleVerticalAlignment.center;

    final color = isList ? Colors.blueGrey : Colors.black26;

    return DragHandle(

      verticalAlignment: verticalAlignment,

      child: Container(

        padding: const EdgeInsets.only(right: 10),

        child: Icon(Icons.menu, color: color),

      ),

    );

  }

  DragAndDropList buildList(DraggableList list) => DragAndDropList(

        header: Container(

          padding: const EdgeInsets.all(8),

          child: Text(

            list.header,

            style: const TextStyle(fontWeight: FontWeight.bold, fontSize: 16),

          ),

        ),

        children: list.items

            .map((item) => DragAndDropItem(

                  child: ListTile(

                    leading: Image.network(

                      item.urlImage,

                      width: 40,

                      height: 40,

                      fit: BoxFit.cover,

                    ),

                    title: Text(item.title),

                  ),

                ))

            .toList(),

      );

  void onReorderListItem(

    int oldItemIndex,

    int oldListIndex,

    int newItemIndex,

    int newListIndex,

  ) {

    setState(() {

      final oldListItems = lists[oldListIndex].children;

      final newListItems = lists[newListIndex].children;

      final movedItem = oldListItems.removeAt(oldItemIndex);

      newListItems.insert(newItemIndex, movedItem);

    });

  }

  void onReorderList(

    int oldListIndex,

    int newListIndex,

  ) {

    setState(() {

      final movedList = lists.removeAt(oldListIndex);

      lists.insert(newListIndex, movedList);

    });

  }

}

Now we have to include the following code in the data.dart file.

import 'model.dart';

List<DraggableList> allLists = [

  const DraggableList(

    header: 'Best Fruits',

    items: [

      DraggableListItem(

        title: 'Orange',

        urlImage:

            'https://images.unsplash.com/photo-1582979512210-99b6a53386f9?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=934&q=80',

      ),

      DraggableListItem(

        title: 'Apple',

        urlImage:

            'https://images.unsplash.com/photo-1560806887-1e4cd0b6cbd6?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=3367&q=80',

      ),

      DraggableListItem(

        title: 'Blueberries',

        urlImage:

            'https://images.unsplash.com/photo-1595231776515-ddffb1f4eb73?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=1950&q=80',

      ),

    ],

  ),

  const DraggableList(

    header: 'Good Fruits',

    items: [

      DraggableListItem(

        title: 'Lemon',

        urlImage:

            'https://images.unsplash.com/photo-1590502593747-42a996133562?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=975&q=80',

      ),

      DraggableListItem(

        title: 'Melon',

        urlImage:

            'https://images.unsplash.com/photo-1571575173700-afb9492e6a50?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=976&q=80',

      ),

      DraggableListItem(

        title: 'Papaya',

        urlImage:

            'https://images.unsplash.com/photo-1617112848923-cc2234396a8d?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=1567&q=80',

      ),

    ],

  ),

  const DraggableList(

    header: 'Disliked Fruits',

    items: [

      DraggableListItem(

        title: 'Banana',

        urlImage:

            'https://images.unsplash.com/photo-1543218024-57a70143c369?ixlib=rb-1.2.1&ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w=975&q=80',

      ),

      DraggableListItem(

        title: 'Strawberries',

        urlImage:

            'https://images.unsplash.com/photo-1464965911861-746a04b4bca6?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=1950&q=80',

      ),

      DraggableListItem(

        title: 'Grapefruit',

        urlImage:

            'https://images.unsplash.com/photo-1577234286642-fc512a5f8f11?ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=975&q=80',

      ),

    ],

  ),

];

Finally lets include the following code in the model.dart file

class DraggableList {

  final String header;

  final List<DraggableListItem> items;

  const DraggableList({

    required this.header,

    required this.items,

  });

}

class DraggableListItem {

  final String title;

  final String urlImage;

  const DraggableListItem({

    required this.title,

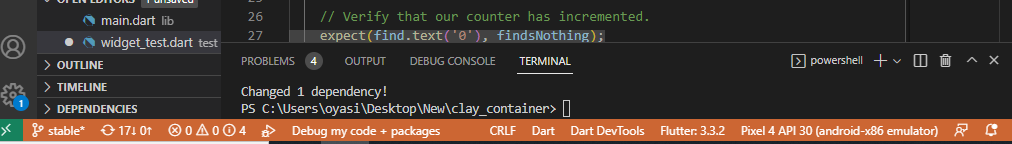
    required this.urlImage,

  });

}

**Step 5:**

Now choose a device in your VS code



Now run the project by pressing ctrl + f5

**Output:**



## **Reorderable Grid View:**

**Step 1:**

Create a new project in the VS code. To do that press ctrl + shift + P

Then click on Flutter: new project

Then choose Application

Now choose a place to save the project and then name the project.

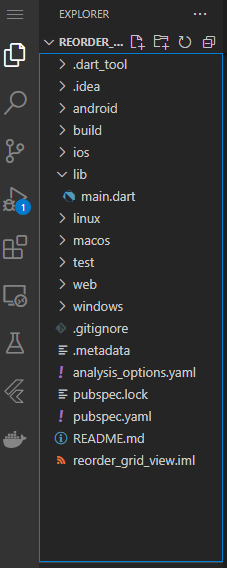
**Step 2:**

Run the following command in the Vs code terminal to add the necessary dependency in your pubspec.yaml.

flutter pub add reorderable\_grid\_view

**Step 3:**

The file structure for this project should be like:



**Step 4:**

Now we have to type the following code in the main.dart after deleting the default template of flutter

import 'package:flutter/material.dart';

import 'package:reorderable\_grid\_view/reorderable\_grid\_view.dart';

void main() {

  runApp(const MyApp());

}

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  // This widget is the root of your application.

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      title: 'Flutter Demo',

      theme: ThemeData(

        // This is the theme of your application.

        //

        // Try running your application with "flutter run". You'll see the

        // application has a blue toolbar. Then, without quitting the app, try

        // changing the primarySwatch below to Colors.green and then invoke

        // "hot reload" (press "r" in the console where you ran "flutter run",

        // or simply save your changes to "hot reload" in a Flutter IDE).

        // Notice that the counter didn't reset back to zero; the application

        // is not restarted.

        primarySwatch: Colors.blue,

      ),

      home: const MyHomePage(),

    );

  }

}

class MyHomePage extends StatefulWidget {

  const MyHomePage({super.key});

  @override

  State<MyHomePage> createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

  final data = [1, 2, 3, 4, 5];

  @override

  Widget build(BuildContext context) {

    Widget buildItem(String text) {

      return Card(

        key: ValueKey(text),

        child: Text(text),

      );

    }

    return Scaffold(

      appBar: AppBar(),

      body: Center(

        // use ReorderableGridView.count() when version >= 2.0.0

        // else use ReorderableGridView()

        child: ReorderableGridView.count(

          crossAxisSpacing: 10,

          mainAxisSpacing: 10,

          crossAxisCount: 3,

          children: this.data.map((e) => buildItem("$e")).toList(),

          onReorder: (oldIndex, newIndex) {

            setState(() {

              final element = data.removeAt(oldIndex);

              data.insert(newIndex, element);

            });

          },

          footer: [

            Card(

              child: Center(

                child: Icon(Icons.add),

              ),

            ),

          ],

        ),

      ),

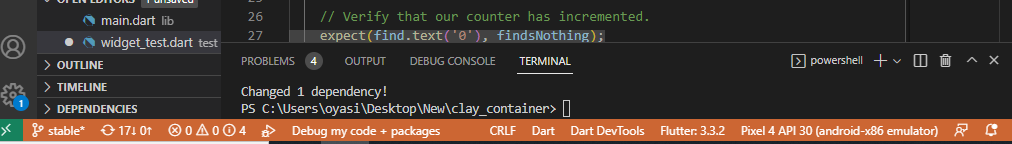
    );

  }

}

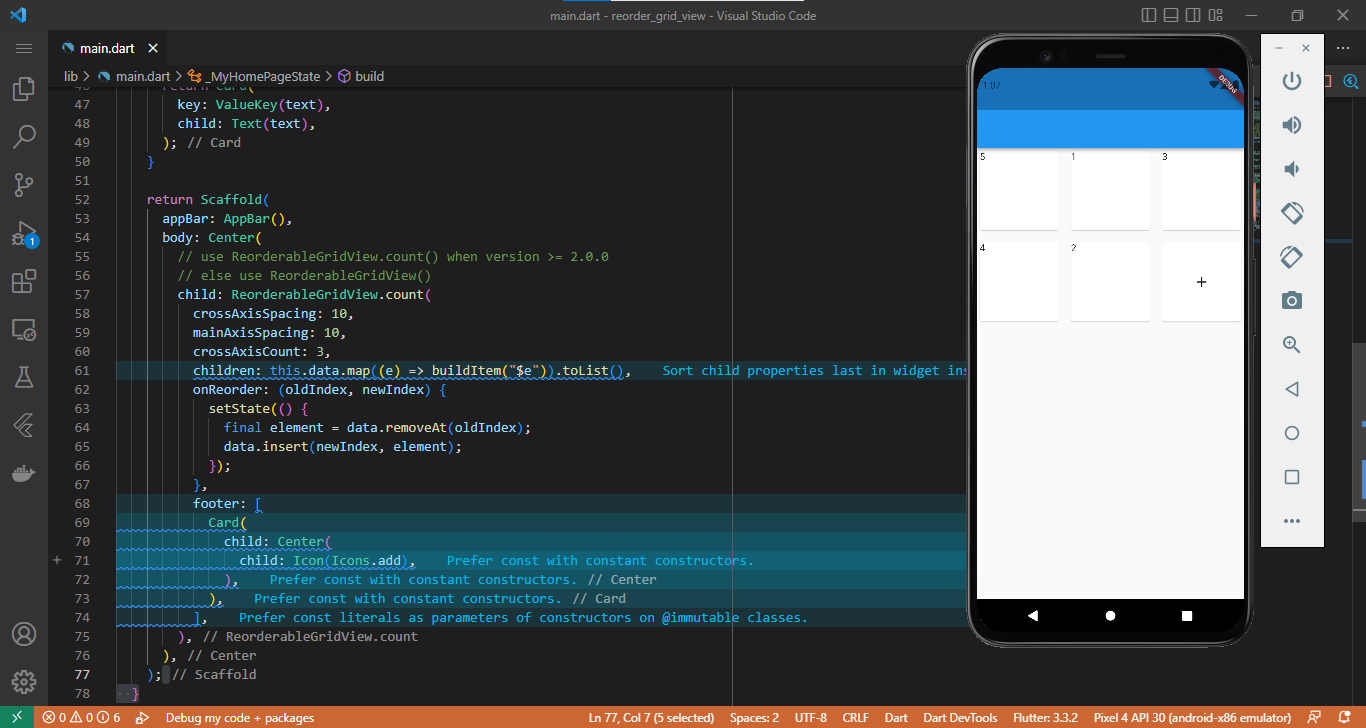
**Step 5:**

Now choose a device in your VS code



Now run the project by pressing ctrl + f5

**Output:**

****

## **Light & Dark Mode**

**Step 1:**

Create a new project in the VS code. To do that press ctrl + shift + P

Then click on Flutter: new project

Then choose Application

Now choose a place to save the project and then name the project.

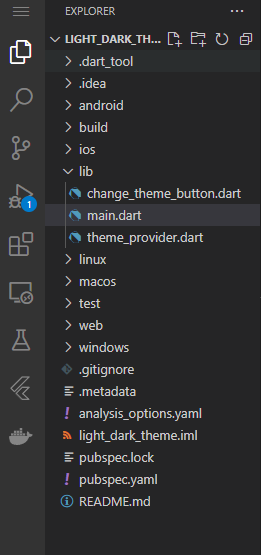
**Step 2:**

Run the following command in the Vs code terminal to add the necessary dependencies in your pubspec.yaml.

flutter pub add provider

**Step 3:**

The file structure for this project should be like:



**Step 4:**

Now we have to type the following code in the main.dart after deleting the default template of flutter.

import 'package:flutter/material.dart';

import 'package:light\_dark\_theme/change\_theme\_button.dart';

import 'package:light\_dark\_theme/theme\_provider.dart';

import 'package:provider/provider.dart';

void main() {

  runApp(const MyApp());

}

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  // This widget is the root of your application.

  @override

  Widget build(BuildContext context) => ChangeNotifierProvider(

      create: (context) => ThemeProvider(),

      builder: (context, \_) {

        final themeProvider = Provider.of<ThemeProvider>(context);

        return MaterialApp(

          title: 'Flutter Demo',

          themeMode: themeProvider.themeMode,

          theme: Mythemes.lightTheme,

          darkTheme: Mythemes.darkTheme,

          home: const homepage(),

          debugShowCheckedModeBanner: false,

        );

      });

}

class homepage extends StatefulWidget {

  const homepage({super.key});

  @override

  State<homepage> createState() => \_homepageState();

}

class \_homepageState extends State<homepage> {

  @override

  Widget build(BuildContext context) {

    final text = Provider.of<ThemeProvider>(context).themeMode == ThemeMode.dark

        ? 'DarkMode'

        : 'LightTheme';

    return Scaffold(

      appBar: AppBar(

        backgroundColor: Colors.lightBlueAccent,

        actions: [

          ChangeThemeButton(),

        ],

      ),

      body: Center(

        child: Text(

          '$text',

          style: TextStyle(

            fontSize: 32,

            fontWeight: FontWeight.bold,

          ),

        ),

      ),

    );

  }

}

Then to we have to decide which color should it take upon changing the theme to dark to light or light to dark in the theme\_provider.dart file.

import 'package:flutter/material.dart';

class ThemeProvider extends ChangeNotifier {

  ThemeMode themeMode = ThemeMode.dark;

  bool get isDarkMode => themeMode == ThemeMode.dark;

  void toggleTheme(bool isOn) {

    themeMode = isOn ? ThemeMode.dark : ThemeMode.light;

    notifyListeners();

  }

}

class Mythemes {

  static final darkTheme = ThemeData(

    scaffoldBackgroundColor: Colors.grey.shade900,

    primaryColor: Colors.black,

    colorScheme: const ColorScheme.dark(),

  );

  static final lightTheme = ThemeData(

    scaffoldBackgroundColor: Colors.white,

    primaryColor: Colors.white,

    colorScheme: const ColorScheme.light(),

  );

}

Now we have to set the theme for the button and connect the theme provider with the button in the change\_theme\_button.dart file.

import 'package:flutter/material.dart';

import 'package:light\_dark\_theme/theme\_provider.dart';

import 'package:provider/provider.dart';

class ChangeThemeButton extends StatelessWidget {

  const ChangeThemeButton({super.key});

  @override

  Widget build(BuildContext context) {

    final themeProvider = Provider.of<ThemeProvider>(context);

    return Switch.adaptive(

      value: themeProvider.isDarkMode,

      onChanged: (value) {

        final provider = Provider.of<ThemeProvider>(context, listen: false);

        provider.toggleTheme(value);

      },

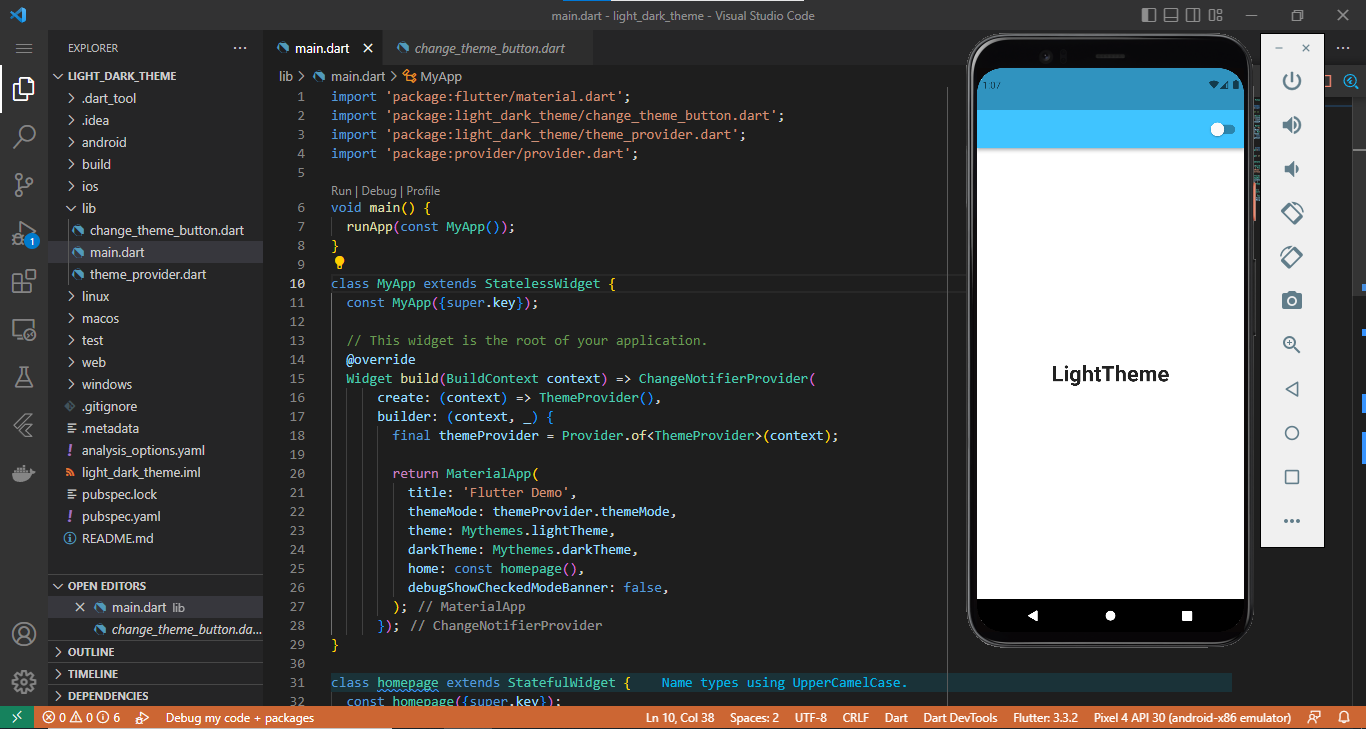
      activeColor: Colors.deepPurple,

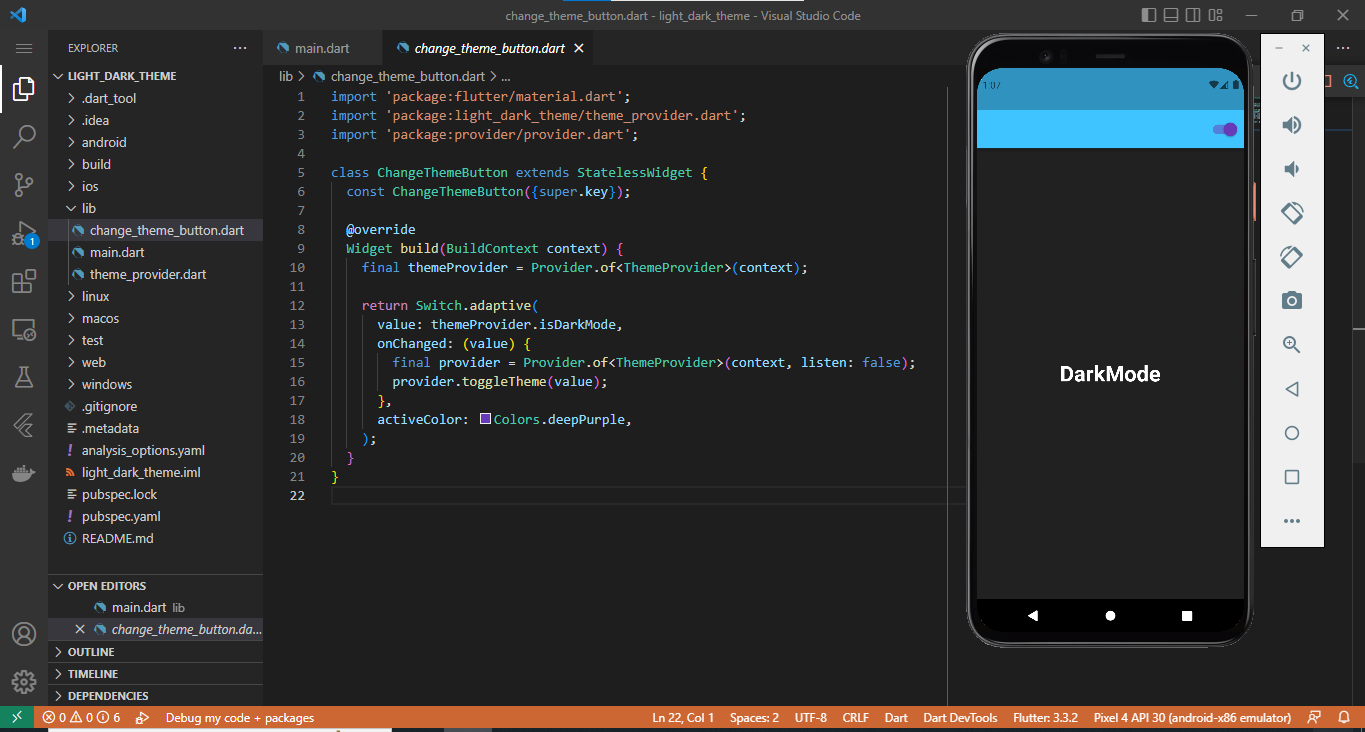
    );

  }

}

**Output:**

****



# **Database:**

## **Local Database**

### **Hive:**

**Step 1:**

Create a new project in the VS code. To do that press ctrl + shift + P

Then click on Flutter: new project

Then choose Application

Now choose a place to save the project and then name the project.

**Step 2:**

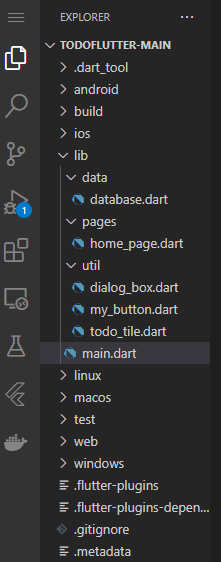
Run the following command in the Vs code terminal to add the necessary dependencies in your pubspec.yaml.

flutter pub add hive\_generator

flutter pub add build\_runner

**Step 3:**

The Project structure should be like the following image:



**Step 4:**

Now we have to delete the default flutter template from main.dart and add the following code in the main.dart file.

import 'package:flutter/material.dart';

import 'package:hive\_flutter/hive\_flutter.dart';

import 'pages/home\_page.dart';

void main() async {

  // init the hive

  await Hive.initFlutter();

  // open a box

  var box = await Hive.openBox('mybox');

  runApp(const MyApp());

}

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      home: HomePage(),

      theme: ThemeData(primarySwatch: Colors.yellow),

    );

  }

}

Now let’s create home\_page.dart file and add the following code:

import 'package:flutter/material.dart';

import 'package:hive\_flutter/hive\_flutter.dart';

import '../data/database.dart';

import '../util/dialog\_box.dart';

import '../util/todo\_tile.dart';

class HomePage extends StatefulWidget {

  const HomePage({super.key});

  @override

  State<HomePage> createState() => \_HomePageState();

}

class \_HomePageState extends State<HomePage> {

  // reference the hive box

  final \_myBox = Hive.box('mybox');

  ToDoDataBase db = ToDoDataBase();

  @override

  void initState() {

    // if this is the 1st time ever openin the app, then create default data

    if (\_myBox.get("TODOLIST") == null) {

      db.createInitialData();

    } else {

      // there already exists data

      db.loadData();

    }

    super.initState();

  }

  // text controller

  final \_controller = TextEditingController();

  // checkbox was tapped

  void checkBoxChanged(bool? value, int index) {

    setState(() {

      db.toDoList[index][1] = !db.toDoList[index][1];

    });

    db.updateDataBase();

  }

  // save new task

  void saveNewTask() {

    setState(() {

      db.toDoList.add([\_controller.text, false]);

      \_controller.clear();

    });

    Navigator.of(context).pop();

    db.updateDataBase();

  }

  // create a new task

  void createNewTask() {

    showDialog(

      context: context,

      builder: (context) {

        return DialogBox(

          controller: \_controller,

          onSave: saveNewTask,

          onCancel: () => Navigator.of(context).pop(),

        );

      },

    );

  }

  // delete task

  void deleteTask(int index) {

    setState(() {

      db.toDoList.removeAt(index);

    });

    db.updateDataBase();

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      backgroundColor: Colors.yellow[200],

      appBar: AppBar(

        title: Text('TO DO'),

        elevation: 0,

      ),

      floatingActionButton: FloatingActionButton(

        onPressed: createNewTask,

        child: Icon(Icons.add),

      ),

      body: ListView.builder(

        itemCount: db.toDoList.length,

        itemBuilder: (context, index) {

          return ToDoTile(

            taskName: db.toDoList[index][0],

            taskCompleted: db.toDoList[index][1],

            onChanged: (value) => checkBoxChanged(value, index),

            deleteFunction: (context) => deleteTask(index),

          );

        },

      ),

    );

  }

}

database.dart

import 'package:hive\_flutter/hive\_flutter.dart';

class ToDoDataBase {

  List toDoList = [];

  // reference our box

  final \_myBox = Hive.box('mybox');

  // run this method if this is the 1st time ever opening this app

  void createInitialData() {

    toDoList = [

      ["Make Tutorial", false],

      ["Do Exercise", false],

    ];

  }

  // load the data from database

  void loadData() {

    toDoList = \_myBox.get("TODOLIST");

  }

  // update the database

  void updateDataBase() {

    \_myBox.put("TODOLIST", toDoList);

  }

}

**my\_button.dart**

import 'package:flutter/material.dart';

class MyButton extends StatelessWidget {

  final String text;

  VoidCallback onPressed;

  MyButton({

    super.key,

    required this.text,

    required this.onPressed,

  });

  @override

  Widget build(BuildContext context) {

    return MaterialButton(

      onPressed: onPressed,

      color: Theme.of(context).primaryColor,

      child: Text(text),

    );

  }

}

**dialog\_box.dart**

import 'package:flutter/material.dart';

import 'my\_button.dart';

class DialogBox extends StatelessWidget {

  final controller;

  VoidCallback onSave;

  VoidCallback onCancel;

  DialogBox({

    super.key,

    required this.controller,

    required this.onSave,

    required this.onCancel,

  });

  @override

  Widget build(BuildContext context) {

    return AlertDialog(

      backgroundColor: Colors.yellow[300],

      content: Container(

        height: 120,

        child: Column(

          mainAxisAlignment: MainAxisAlignment.spaceEvenly,

          children: [

            // get user input

            TextField(

              controller: controller,

              decoration: InputDecoration(

                border: OutlineInputBorder(),

                hintText: "Add a new task",

              ),

            ),

            // buttons -> save + cancel

            Row(

              mainAxisAlignment: MainAxisAlignment.end,

              children: [

                // save button

                MyButton(text: "Save", onPressed: onSave),

                const SizedBox(width: 8),

                // cancel button

                MyButton(text: "Cancel", onPressed: onCancel),

              ],

            ),

          ],

        ),

      ),

    );

  }

}

**todo\_tile.dart**

import 'package:flutter/material.dart';

import 'package:flutter\_slidable/flutter\_slidable.dart';

class ToDoTile extends StatelessWidget {

  final String taskName;

  final bool taskCompleted;

  Function(bool?)? onChanged;

  Function(BuildContext)? deleteFunction;

  ToDoTile({

    super.key,

    required this.taskName,

    required this.taskCompleted,

    required this.onChanged,

    required this.deleteFunction,

  });

  @override

  Widget build(BuildContext context) {

    return Padding(

      padding: const EdgeInsets.only(left: 25.0, right: 25, top: 25),

      child: Slidable(

        endActionPane: ActionPane(

          motion: StretchMotion(),

          children: [

            SlidableAction(

              onPressed: deleteFunction,

              icon: Icons.delete,

              backgroundColor: Colors.red.shade300,

              borderRadius: BorderRadius.circular(12),

            )

          ],

        ),

        child: Container(

          padding: EdgeInsets.all(24),

          decoration: BoxDecoration(

            color: Colors.yellow,

            borderRadius: BorderRadius.circular(12),

          ),

          child: Row(

            children: [

              // checkbox

              Checkbox(

                value: taskCompleted,

                onChanged: onChanged,

                activeColor: Colors.black,

              ),

              // task name

              Text(

                taskName,

                style: TextStyle(

                  decoration: taskCompleted

                      ? TextDecoration.lineThrough

                      : TextDecoration.none,

                ),

              ),

            ],

          ),

        ),

      ),

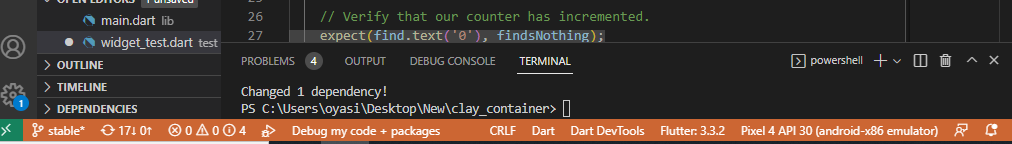
    );

  }

}

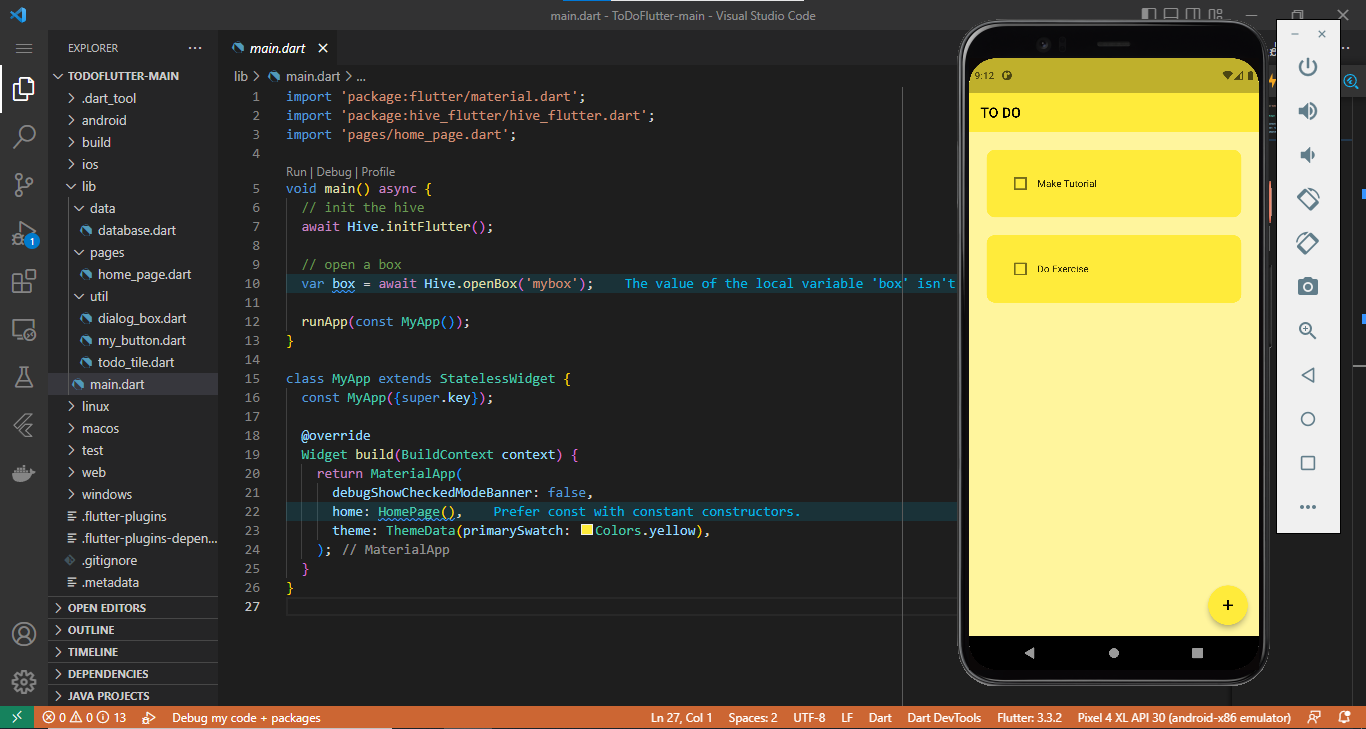
**Step 5:**

Now choose a device in your VS code



Now run the project by pressing ctrl + f5

**Output:**



### **Sqflite:**

**Step 1:**

Create a new project in the VS code. To do that press ctrl + shift + P

Then click on Flutter: new project

Then choose Application

Now choose a place to save the project and then name the project.

**Step 2:**

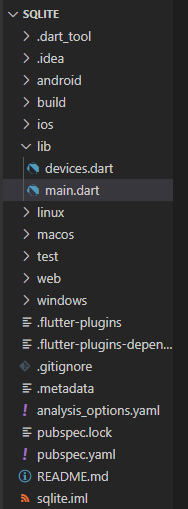
Run the following command in the Vs code terminal to add the necessary dependencies in your pubspec.yaml.

flutter pub add sqflite

flutter pub add path\_provider

**Step 3:**

The Project structure should be like the following image:



**Step 4:**

Now we have to delete the default flutter template from main.dart and add the following code in the main.dart file.

import 'package:flutter/material.dart';

import 'package:sqlite/devices.dart';

void main() {

  runApp(const MyApp());

}

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  // This widget is the root of your application.

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      title: 'Flutter Demo',

      theme: ThemeData(

        primarySwatch: Colors.blue,

      ),

      home: const dashboard(),

    );

  }

}

Now let’s create the devices.dart file and type the following code:

import 'dart:async';

import 'package:flutter/material.dart';

import 'package:path\_provider/path\_provider.dart';

import 'package:sqflite/sqflite.dart';

class Device implements Comparable {

  final int id;

  final String firstName;

  final String lastName;

  const Device({

    required this.id,

    required this.firstName,

    required this.lastName,

  });

  String get fullName => '$firstName $lastName';

  Device.fromRow(Map<String, Object?> row)

      : id = row['ID'] as int,

        firstName = row['FIRST\_NAME'] as String,

        lastName = row['LAST\_NAME'] as String;

  @override

  int compareTo(covariant other) => other.id.compareTo(id);

  @override

  bool operator ==(covariant Device other) => id == other.id;

  @override

  int get hashCode => id.hashCode;

  @override

  String toString() =>

      'Device, id = $id, firstName: $firstName, lastname: $lastName';

}

class DeviceDB {

  final String dbName;

  Database? \_db;

  List<Device> \_devices = [];

  final \_streamController = StreamController<List<Device>>.broadcast();

  DeviceDB({required this.dbName});

  Future<List<Device>> \_fetchTool() async {

    final db = \_db;

    if (db == null) {

      return [];

    }

    try {

      final read = await db.query(

        'DEVICE',

        distinct: true,

        columns: [

          'ID',

          'FIRST\_NAME',

          'LAST\_NAME',

        ],

        orderBy: 'ID',

      );

      final tool = read.map((row) => Device.fromRow(row)).toList();

      return tool;

    } catch (e) {

      print('Error fetching devices = $e');

      return [];

    }

  }

// C in CRUD

  Future<bool> create(String firstName, String lastName) async {

    final db = \_db;

    if (db == null) {

      return false;

    }

    try {

      final id = await db.insert('DEVICE', {

        'FIRST\_NAME': firstName,

        'LAST\_NAME': lastName,

      });

      final device = Device(

        id: id,

        firstName: firstName,

        lastName: lastName,

      );

      \_devices.add(device);

      \_streamController.add(\_devices);

      return true;

    } catch (e) {

      print('Error in creating device = $e');

      return false;

    }

  }

  Future<bool> delete(Device device) async {

    final db = \_db;

    if (db == null) {

      return false;

    }

    try {

      final deleteCount =

          await db.delete('DEVICE', where: 'ID = ?', whereArgs: [device.id]);

      if (deleteCount == 1) {

        \_devices.remove(device);

        \_streamController.add(\_devices);

        return true;

      } else {

        return false;

      }

    } catch (e) {

      print('Deletion failed with error $e');

      return false;

    }

  }

  Future<bool> close() async {

    final db = \_db;

    if (db == null) {

      return false;

    }

    await db.close();

    return true;

  }

  Future<bool> open() async {

    if (\_db != null) {

      return true;

    }

    final directory = await getApplicationDocumentsDirectory();

    final path = '${directory.path}/$dbName';

    try {

      final db = await openDatabase(path);

      \_db = db;

      // create table

      final create = '''CREATE TABLE IF NOT EXISTS DEVICE (

        ID INTEGER PRIMARY KEY AUTOINCREMENT,

        FIRST\_NAME STRING NOT NULL,

        LAST\_NAME STRING NOT NULL

      )''';

      await db.execute(create);

      // read all existing Device objects from the db

      \_devices = await \_fetchTool();

      \_streamController.add(\_devices);

      return true;

    } catch (e) {

      print('Error = $e');

      return false;

    }

  }

  Future<bool> update(Device device) async {

    final db = \_db;

    if (db == null) {

      return false;

    }

    try {

      final updateCount = await db.update(

        'DEVICE',

        {

          'FIRST\_NAME': device.firstName,

          'LAST\_NAME': device.lastName,

        },

        where: 'ID = ?',

        whereArgs: [device.id],

      );

      if (updateCount == 1) {

        \_devices.removeWhere((other) => other.id == device.id);

        \_devices.add(device);

        \_streamController.add(\_devices);

        return true;

      } else {

        return false;

      }

    } catch (e) {

      print('failed to update device, error = $e');

      return false;

    }

  }

  Stream<List<Device>> all() =>

      \_streamController.stream.map((devices) => devices..sort());

}

class dashboard extends StatefulWidget {

  const dashboard({super.key});

  @override

  State<dashboard> createState() => \_dashboardState();

}

class \_dashboardState extends State<dashboard> {

  late final DeviceDB \_crudStorage;

  @override

  void initState() {

    \_crudStorage = DeviceDB(dbName: 'db.sqlite');

    \_crudStorage.open();

    super.initState();

  }

  @override

  void dispose() {

    \_crudStorage.close();

    super.dispose();

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        actions: [Icon(Icons.notification\_add)],

        title: const Text('CRUD'),

      ),

      body: StreamBuilder(

        stream: \_crudStorage.all(),

        builder: (context, snapshot) {

          print(snapshot);

          switch (snapshot.connectionState) {

            case ConnectionState.active:

            case ConnectionState.waiting:

              if (snapshot.data == null) {

                return const Center(child: CircularProgressIndicator());

              }

              final tool = snapshot.data as List<Device>;

              return Column(

                children: [

                  ComposeWidget(

                    onCompose: (firstName, lastName) {

                      \_crudStorage.create(firstName, lastName);

                    },

                  ),

                  Expanded(

                    child: ListView.builder(

                        itemCount: tool.length,

                        itemBuilder: (context, index) {

                          final device = tool[index];

                          return ListTile(

                            onTap: () async {

                              final editedDevice = await showUpdateDialog(

                                context,

                                device,

                              );

                              if (editedDevice != null) {

                                await \_crudStorage.update(editedDevice);

                              }

                            },

                            // title: Text(device.firstName),

                            // subtitle: Text(device.lastName),

                            title: Text(device.fullName),

                            subtitle: Text('ID: ${device.id}'),

                            trailing: TextButton(

                                onPressed: () async {

                                  final shouldDelete =

                                      await showDeleteDialog(context);

                                  print(shouldDelete);

                                  if (shouldDelete) {

                                    await \_crudStorage.delete(device);

                                  }

                                },

                                child: const Icon(

                                  Icons.delete\_forever\_rounded,

                                  color: Colors.red,

                                )),

                          );

                        }),

                  ),

                ],

              );

            default:

              return const Center(child: CircularProgressIndicator());

          }

        },

      ),

    );

  }

}

Future<bool> showDeleteDialog(BuildContext context) {

  return showDialog(

    context: context,

    builder: (context) {

      return AlertDialog(

        content: const Text('Are you sure you want to delete this item?'),

        actions: [

          TextButton(

            onPressed: () {

              Navigator.of(context).pop(false);

            },

            child: const Text('No'),

          ),

          TextButton(

            onPressed: () {

              Navigator.of(context).pop(true);

            },

            child: const Text('Delete'),

          ),

        ],

      );

    },

  ).then((value) {

    if (value is bool) {

      return value;

    } else {

      return false;

    }

  });

}

final \_firstNameController = TextEditingController();

final \_lastNameController = TextEditingController();

Future<Device?> showUpdateDialog(BuildContext context, Device device) {

  \_firstNameController.text = device.firstName;

  \_lastNameController.text = device.lastName;

  return showDialog(

    context: context,

    builder: (context) {

      return AlertDialog(

        content: Column(

          mainAxisSize: MainAxisSize.min,

          children: [

            const Text('Enter your updated Values here:'),

            TextField(

              controller: \_firstNameController,

            ),

            TextField(

              controller: \_lastNameController,

            ),

          ],

        ),

        actions: [

          TextButton(

            onPressed: () {

              Navigator.of(context).pop(null);

            },

            child: const Text('Cancel'),

          ),

          TextButton(

            onPressed: () {

              final editedDevice = Device(

                id: device.id,

                firstName: \_firstNameController.text,

                lastName: \_lastNameController.text,

              );

              Navigator.of(context).pop(editedDevice);

            },

            child: const Text('Save'),

          ),

        ],

      );

    },

  ).then((value) {

    if (value is Device) {

      return value;

    } else {

      return null;

    }

  });

}

typedef OnCompose = void Function(String firstname, String lastName);

class ComposeWidget extends StatefulWidget {

  final OnCompose onCompose;

  const ComposeWidget({super.key, required this.onCompose});

  @override

  State<ComposeWidget> createState() => \_ComposeWidgetState();

}

class \_ComposeWidgetState extends State<ComposeWidget> {

  late final TextEditingController \_firstNameController;

  late final TextEditingController \_lastNameController;

  @override

  void initState() {

    \_firstNameController = TextEditingController();

    \_lastNameController = TextEditingController();

    super.initState();

  }

  @override

  void dispose() {

    \_firstNameController.dispose();

    \_lastNameController.dispose();

    super.dispose();

  }

  @override

  Widget build(BuildContext context) {

    return Padding(

      padding: const EdgeInsets.all(8.0),

      child: Column(

        children: [

          TextField(

            controller: \_firstNameController,

            decoration: InputDecoration(

              hintText: 'Enter first name',

            ),

          ),

          TextField(

            controller: \_lastNameController,

            decoration: InputDecoration(

              hintText: 'Enter last name',

            ),

          ),

          TextButton(

            onPressed: () {

              final firstName = \_firstNameController.text;

              final lastName = \_lastNameController.text;

              widget.onCompose(firstName, lastName);

              \_firstNameController.text = '';

              \_lastNameController.text = '';

            },

            child: Text(

              'Add to list',

              style: TextStyle(fontSize: 24),

            ),

          ),

        ],

      ),

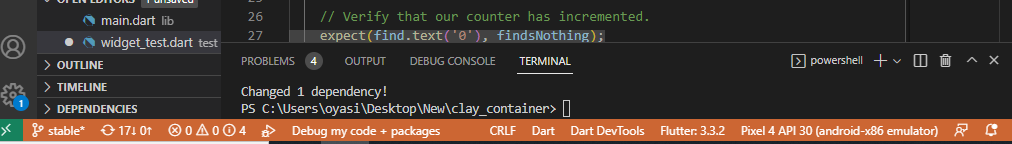
    );

  }

}

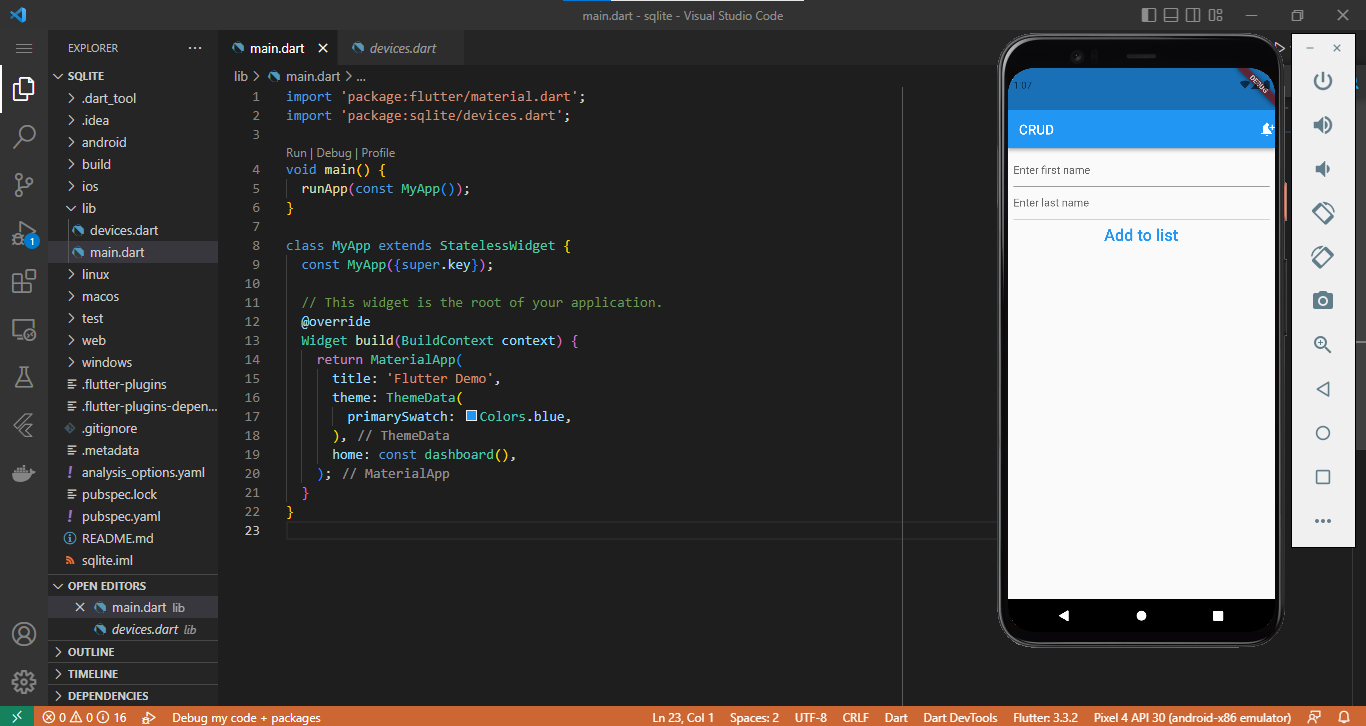
**Step 5:**

Now choose a device in your VS code



Now run the project by pressing ctrl + f5

**Output:**



### **MySql:**

**Requirements:**

* Google chrome
* XAMPP
* Desktop

**XAMPP:**

XAMPP is the title used for a **compilation of free software**. The name is an acronym, with each letter representing one of the five key components. The software packet contains the web server **A**pache, the relational database management system **M**ySQL (or **M**ariaDB), and the scripting languages **P**erl and **P**HP. The initial**X** stands for the operating systems that it works with: Linux, Windows, and Mac OS X.

* **Apache:** ihe open source web server Apache is the most widely used server worldwide for delivery of web content. The server application is made available as a free software by the Apache Software Foundation.
* **MySQL/MariaDB:**in MySQL, XAMPP contains one of the most popular relational database management systems in the world. In combination with the web server Apache and the scripting language PHP, MySQL offers data storage for web services. Current XAMPP versions have replaced MySQL with MariaDB (a community-developed fork of the MySQL project, made by the original developers).
* **PHP:** the server-side programming language [PHP](https://www.ionos.com/digitalguide/websites/website-creation/learn-php-our-all-encompassing-php-tutorial-for-beginners/) enables users to create dynamic websites or applications. PHP can be installed on all platforms and supports a number of diverse database systems.
* **Perl:** the scripting language Perl is used in system administration, web development, and network programming. Like PHP, Perl also enables users to program dynamic web applications.

Alongside these core components, this free-to-use Apache distribution contains some other useful tools, which vary depending on your operating system. These tools include the mail server **Mercury**, the database administration tool **phpMyAdmin**, the web analytics software solutions [Webalizer](https://www.ionos.com/digitalguide/server/configuration/use-the-webalizer-to-analyze-apache-web-logs/" \o "Use The Webalizer to Analyze Apache Web Logs), **OpenSSL**, and **Apache Tomcat**, and the FTP servers **FileZilla** or **ProFTPd**.

**XAMPP installation process:**

Our XAMPP tutorial will take you through the installation process for the software package on Windows. If you’re using Linux or Mac OS X, then the steps listed below for the installation process may differ.

#### **Step 1: Download**

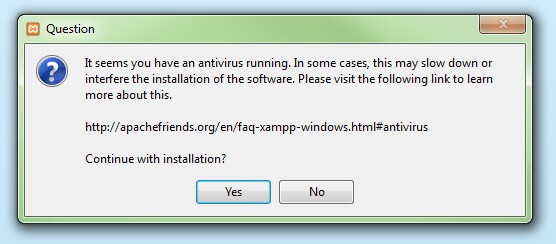
XAMPP is a release made available by the non-profit project Apache Friends. Versions with PHP 5.5, 5.6, or 7 are available for download on the [Apache Friends](http://www.apachefriends.org/de/download.html) website.

#### **Step 2: Run .exe file**

Once the software bundle has been downloaded, you can start the installation by double clicking on the file with the ending [.exe](https://www.ionos.com/digitalguide/server/know-how/exe-file/).

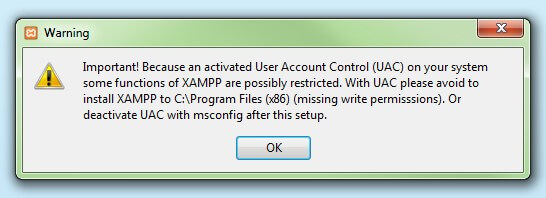
#### **Step 3: Deactivate any antivirus software**

Since an active antivirus program can negatively affect the installation process, it’s recommended to temporarily pause any antivirus software until all XAMPP components have successfully been installed.

[](https://www.ionos.com/digitalguide/fileadmin/DigitalGuide/Screenshots/XAMPP_01.jpg)Before installing XAMPP, it is advisable to disable the anti-virus program temporarily

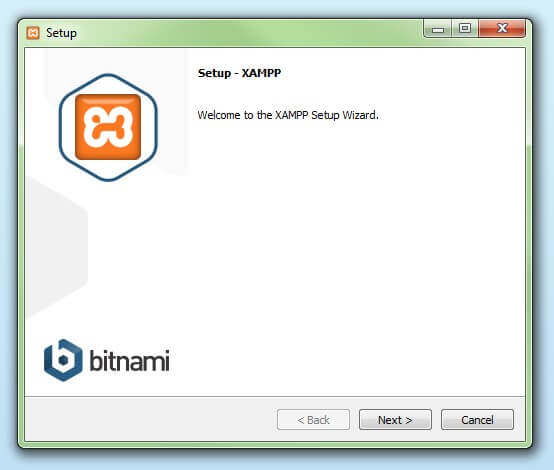
#### **Step 4: Deactivate UAC**

User Account Control (UAC) can interfere with the XAMPP installation because it limits writing access to the C: drive, so we recommend you deactivate this too for the duration of the installation process. To find out how to turn off your UAC, head to the [Microsoft Windows support pages](https://support.microsoft.com/en-us/help/17228/windows-protect-my-pc-from-viruses).

[](https://www.ionos.com/digitalguide/fileadmin/DigitalGuide/Screenshots/XAMPP_02.jpg)User account control can affect the installation of XAMPP

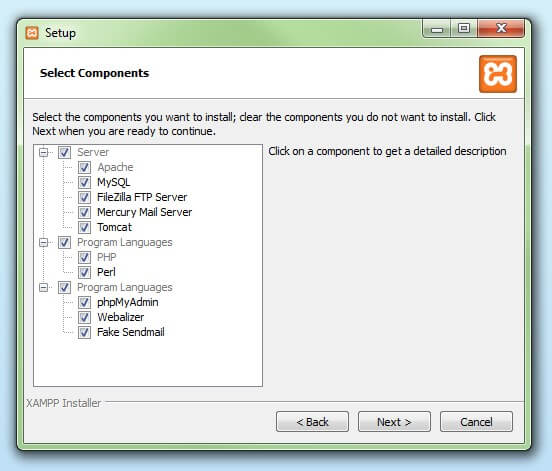
#### **Step 5: Start the setup wizard**

After you’ve opened the .exe file (after deactivating your antivirus program(s) and taken note of the User Account Control, the start screen of the XAMPP setup wizard should appear automatically. Click on ‘Next’ to configure the installation settings.

[](https://www.ionos.com/digitalguide/fileadmin/DigitalGuide/Screenshots/XAMPP_03.jpg)You can start the setup on the startup screen

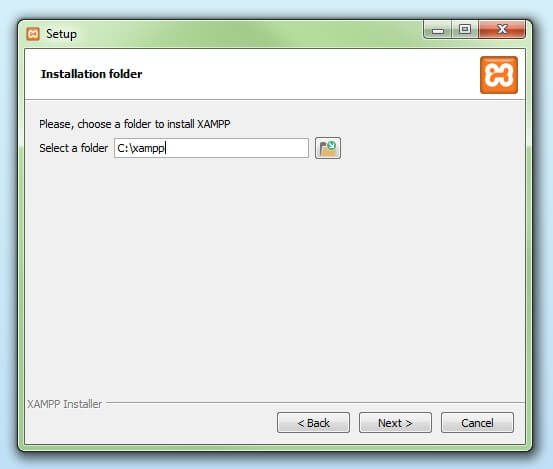
#### **Step 6: Choose software components**

Under ‘Select Components’, you have the option to exclude individual components of the XAMPP software bundle from the installation. But for a full local test server, we recommend you install using the standard setup and all available components. After making your choice, click ‘Next’.

[](https://www.ionos.com/digitalguide/fileadmin/DigitalGuide/Screenshots/XAMPP_04.jpg)In the dialog window entitled 'select components', you can choose the software components before installation

#### **Step 7: Choose the installation directory**

In this next step, you have the chance to choose where you’d like the XAMPP software packet to be installed. If you opt for the standard setup, then a folder with the name XAMPP will be created under C:\ for you. After you’ve chosen a location, click ‘Next’.

[](https://www.ionos.com/digitalguide/fileadmin/DigitalGuide/Screenshots/XAMPP_05.jpg)For the next step, you need to select the directory where XAMPP should be installed

#### **Step 8: Start the installation process**

Once all the aforementioned preferences have been decided, click to start the installation. The setup wizard will unpack and install the selected components and save them to the designated directory. This process can take several minutes in total. You can follow the progress of this installation by keeping an eye on the green loading bar in the middle of the screen.

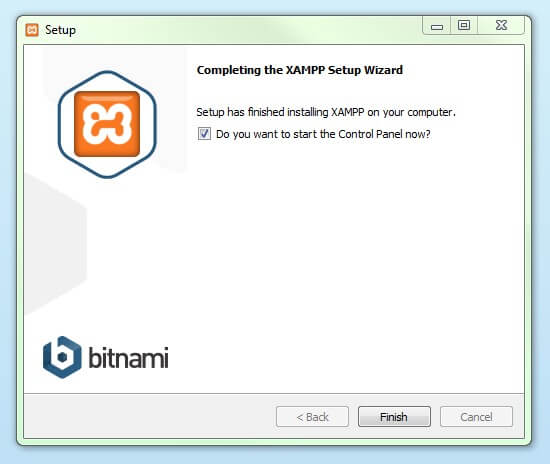
[](https://www.ionos.com/digitalguide/fileadmin/DigitalGuide/Screenshots/XAMPP_06.jpg)According to the default settings, the selected software components are unpacked and installed in the target folder

#### **Step 9: Windows Firewall blocking**

Your Firewall may interrupt the installation process to block the some components of the XAMPP. Use the corresponding check box to enable communication between the Apache server and your private network or work network. Remember that making your XAMPP server available for public networks isn’t recommended.

#### **Step 10: Complete installation**

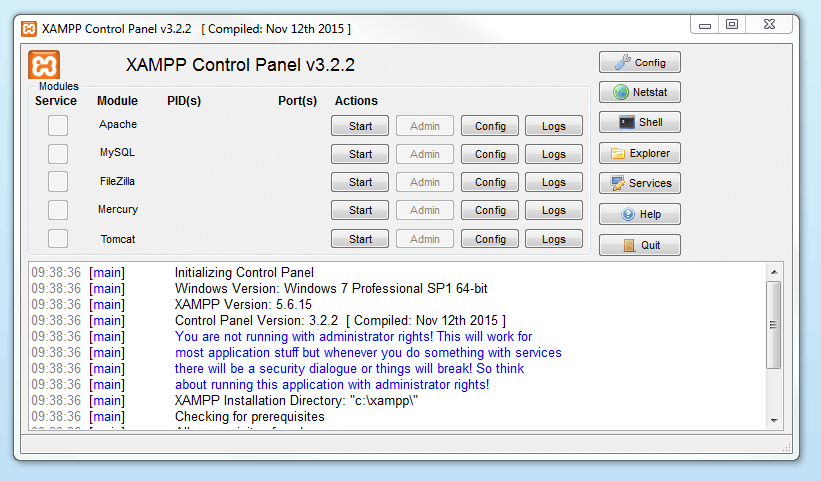
Once all the components are unpacked and installed, you can close the setup wizard by clicking on ‘Finish’. Click to tick the corresponding check box and open the XAMPP Control Panel once the installation process is finished.

[](https://www.ionos.com/digitalguide/fileadmin/DigitalGuide/Screenshots/XAMPP_08.jpg)By clicking on 'finish', the XAMPP Setup Wizard is completed

## **The XAMPP Control Panel**

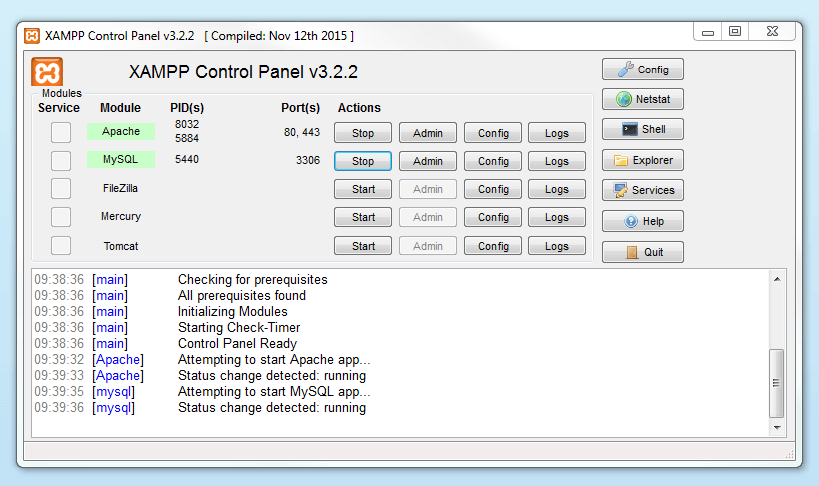
Controls for the individual components of your test server can be reached through the XAMPP Control Panel. **The clear user interface**logs all actions and allows you to start or stop individual modules with a single. The XAMPP Control Panel also offers you various other buttons, including:

* **Config:** allows you to configure the XAMPP as well as the individual components
* [Netstat](https://www.ionos.com/digitalguide/server/tools/introduction-to-netstat/)**:** shows all running processes on the local computer
* **Shell:** opens a UNIX shell
* **Explorer:** opens the XAMPP folder in Windows Explorer
* **Services:** shows all services currently running in the background
* **Help:**offers links to user forums
* **Quit:**closes the XAMPP Control Panel

[](https://www.ionos.com/digitalguide/fileadmin/DigitalGuide/Screenshots/EN_XAMPP_Control_Panel_1.PNG)In the Control Panel, you can start and stop individual modules

### **Starting modules**

Individual modules can be started or stopped on the XAMPP Control Panel through the corresponding buttons under ‘Actions’. You can see which modules have been started because their names are highlighted green under the ‘Module’ title.

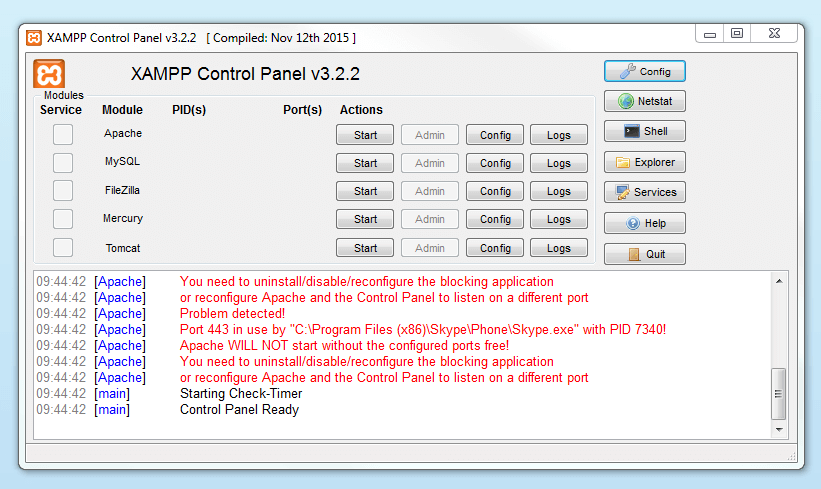
[](https://www.ionos.com/digitalguide/fileadmin/DigitalGuide/Screenshots/EN_XAMPP_Control_Panel_2.PNG)An active module is marked in green in the Control Panel

If a module can’t be started as a result of an error, you’ll be informed of this straight away in red font. A **detailed error report** can help you identify the cause of the issue.

### **Setting up XAMPP**

A common source of error connected with Apache is **blocked ports**. If you’re using the standard setup, then XAMPP will assign the web server to main port 80 and the SSL port 443. The latter of these particularly is often blocked by other programs. In the example above, it’s likely that the Tomcat port is being blocked, meaning the web server can’t be started. There are three ways to solve this issue:

* **Change the conflicting port:**Let’s assume for the sake of example that the instant messenger program Skype is blocking SSL port 443 (this is a common problem). One way to deal with this issue is to change Skype’s port settings. To do this, open the program and navigate via ‘Actions’, ‘Options’, and ‘Advanced’, until you reach the ‘Connections’ menu. You should find a box checked to allow Skype access to ports 80 and 443. Deselect this checkbox now.
* **Change the XAMPP module port settings**: Click the Config button for the module in question and open the files httpd.conf and httpd-ssl.conf. Replace port number 80 in httpd.conf and port number 443 in httpd-ssl.conf with any free ports, before saving the file data. Now click on the general Config button on the right-hand side and select ‘Services and Ports Settings’. Customize the ports for the module server to reflect the changes in the conf files.
* **End the conflicting program:**The simplest way to avoid port conflicts in the short term is to end the conflicting program (Skype in this case). If you restart Skype after your XAMPP module servers are already running, it will select a different port and your issue will be resolved.

[](https://www.ionos.com/digitalguide/fileadmin/DigitalGuide/Screenshots/EN_XAMPP_Control_Panel_3.PNG)Modules that can’t be started will be shown in red. The user will also receive an error report to help solve the problem

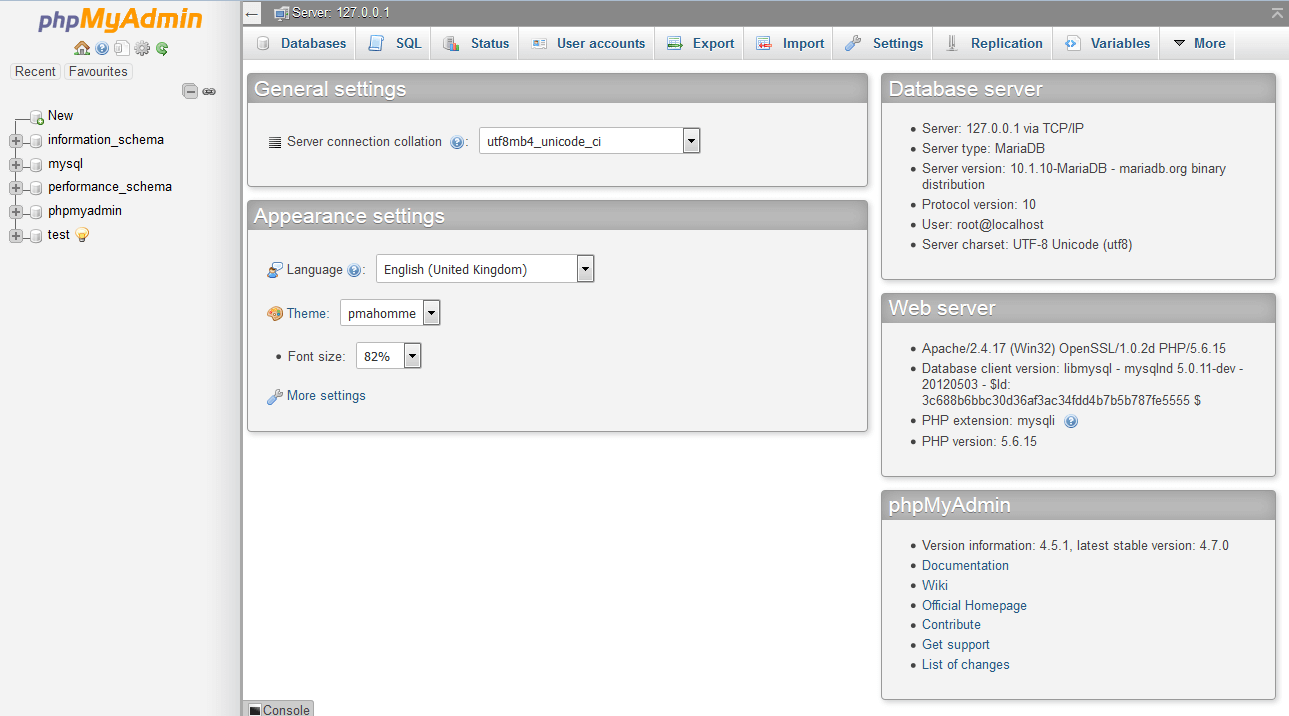
### **Module administration**

You have an ‘Admin’ option located on the Control Panel for every module in your XAMPP.

* Click on the Admin button of your Apache server to go to the web address of your web server. The Control Panel will now start in your standard browser, and you’ll be led to the **dashboard of your XAMPP’s local host**. The dashboard features numerous links to websites for useful information as well as the open source project [BitNami](https://bitnami.com/stack/xampp" \o "Link to homepage for BitNami" \t "_blank), which offers you many different applications for your XAMPP, like WordPress or other content management systems. Alternatively, you can reach the dashboard through localhost/dashboard/.

[](https://www.ionos.com/digitalguide/fileadmin/DigitalGuide/Screenshots/XAMPP_12.jpg)By clicking on the 'admin' button of the Apache module, the user will be redirected to the local dashboard of XAMPP

* You can use the Admin button of your database module to open **phpMyAdmin**. Here, you can manage the databases of your web projects that you’re testing on your XAMPP. Alternatively, you can reach the administration section of your MySQL database via [localhost](https://www.ionos.com/digitalguide/server/know-how/localhost/)/phpmyadmin/.

[](https://www.ionos.com/digitalguide/fileadmin/DigitalGuide/Screenshots/EN_XAMPP_phpMyAdmin.PNG)The web project’s databases are managed by the user in phpMyAdmin (accessible via the 'Admin' button in the database module)

**Now let’s create a simple login application using MySQL.**

**Step 1 (Create Project):**

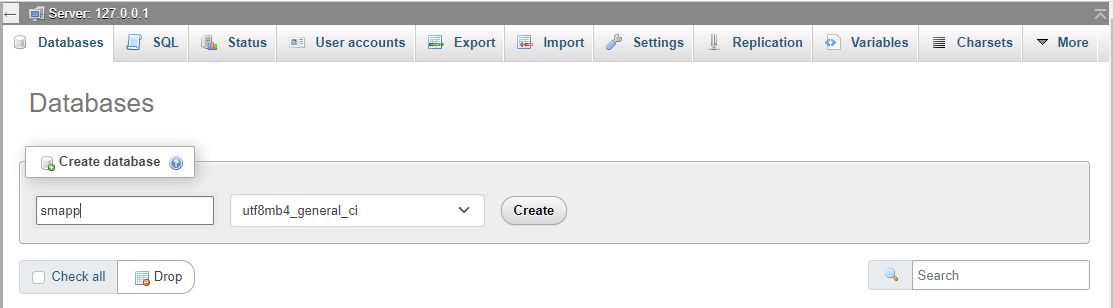
Create a new project in the VS code. To do that press ctrl + shift + P

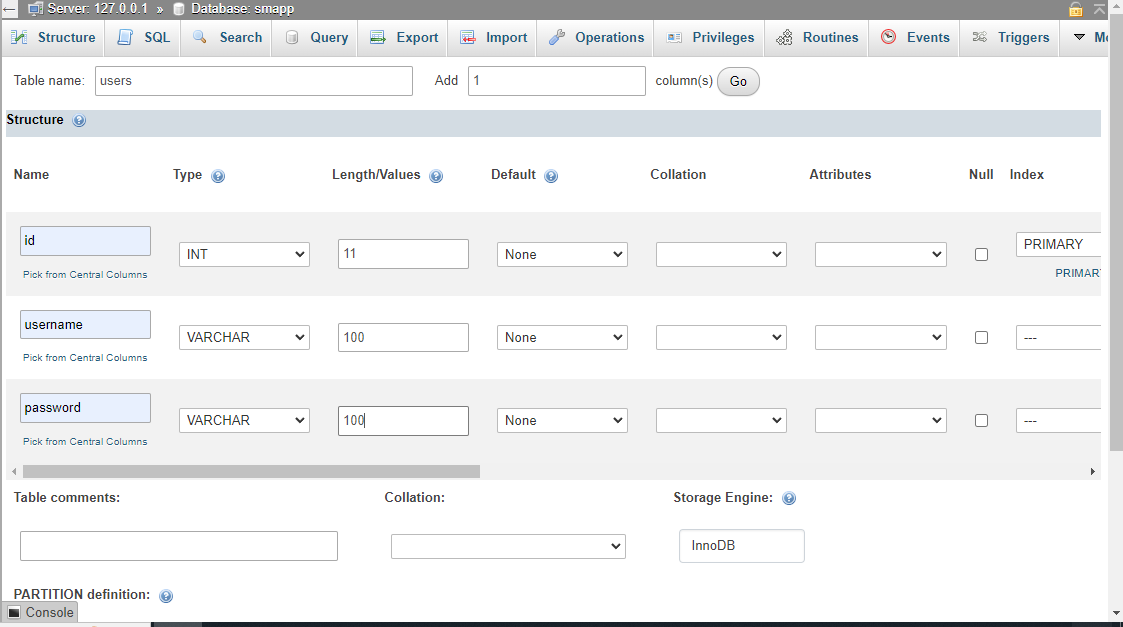
Then click on Flutter: new project

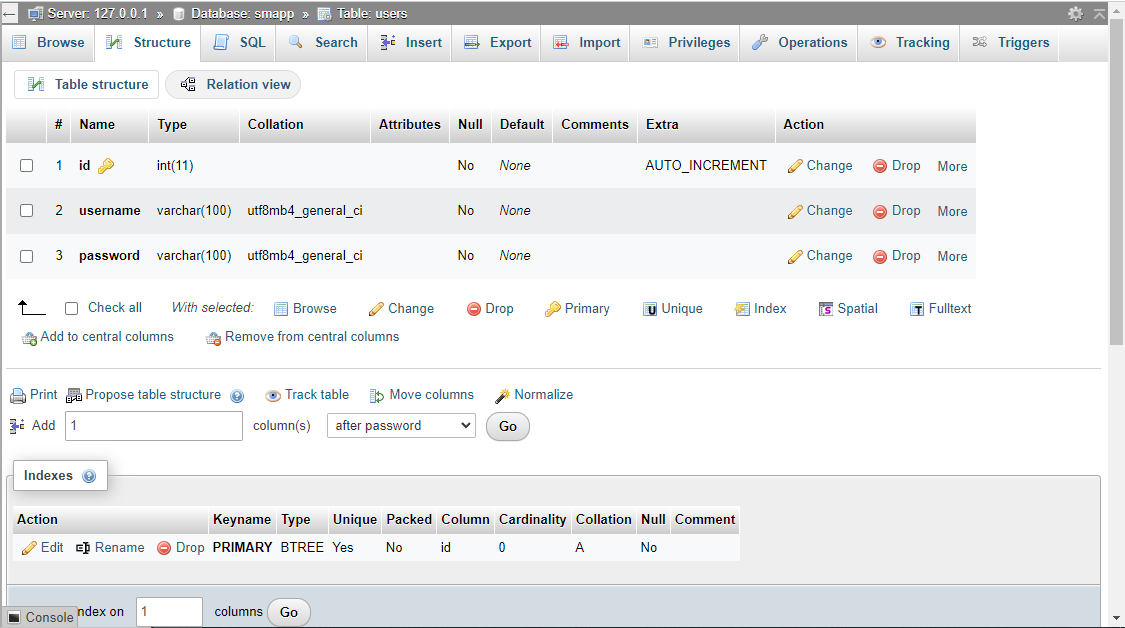
Then choose Application

Now choose a place to save the project and then name the project.

**Step 2 (database setup):**

****

****

****

**Step 3 (file structure):**

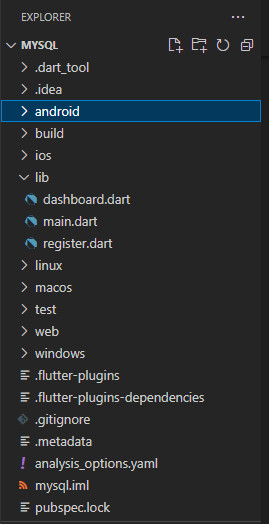
File structure:

Create a folder named smapp\_db:

and create 2 file named login.php & register.php



And in the flutter project the file structure will as below:



**Step 4 (Dependencies & permission):**

Run the following command in the Vs code terminal to add the necessary dependencies in your pubspec.yaml.

flutter pub add http

flutter pub add fluttertoast

flutter pub add provider

**Step 5 (Code):**

To access the database we have to create 2 file login.php & register.php

register.php will add the data to the database and on the other hand login.php will verify the data if it is already in the database or not.

**register.php**

<?php

$db = mysqli\_connect('localhost','root','','smapp');

if(!$db)

{

    echo "Database connection failed";

}

$username = $\_POST['username'];

$password = $\_POST['password'];

$sql = "SELECT username FROM users WHERE username = '".$username."'";

$result = mysqli\_query($db,$sql);

$count = mysqli\_num\_rows($result);

if($count == 1){

    echo json\_encode("Error");

}else{

    $insert = "INSERT INTO users(username,password) VALUES ('".$username."','".$password."')";

        $query = mysqli\_query($db,$insert);

        if($query){

            echo json\_encode("Success");

        }

}

?>

**login.php**

<?php

 $db = mysqli\_connect('localhost','root','','smapp');

 $username = $\_POST['username'];

 $password = $\_POST['password'];

 $sql = "SELECT \* FROM users WHERE username = '".$username."' AND password = '".$password."'";

 $result = mysqli\_query($db,$sql);

 $count = mysqli\_num\_rows($result);

 if($count == 1){

    echo json\_encode("Success");

 }

 else{

    echo json\_encode("Error");

 }

?>

**Now we need to add the database files in the flutter project**

**register.dart**

import 'dart:convert';

import 'package:flutter/material.dart';

import 'package:fluttertoast/fluttertoast.dart';

import 'package:http/http.dart' as http;

import 'package:mysql/DashBoard.dart';

import 'main.dart';

class Register extends StatefulWidget {

  const Register({Key? key}) : super(key: key);

  @override

  \_RegisterState createState() => \_RegisterState();

}

class \_RegisterState extends State<Register> {

  TextEditingController user = TextEditingController();

  TextEditingController pass = TextEditingController();

  Future register() async {

    var url =

        Uri.http("192.168.0.26", '/smapp\_db/register.php', {'q': '{http}'});

    var response = await http.post(url, body: {

      "username": user.text.toString(),

      "password": pass.text.toString(),

    });

    var data = json.decode(response.body);

    if (data == "Error") {

      Fluttertoast.showToast(

        backgroundColor: Colors.orange,

        textColor: Colors.white,

        msg: 'User already exit!',

        toastLength: Toast.LENGTH\_SHORT,

      );

    } else {

      Fluttertoast.showToast(

        backgroundColor: Colors.green,

        textColor: Colors.white,

        msg: 'Registration Successful',

        toastLength: Toast.LENGTH\_SHORT,

      );

      Navigator.push(

        context,

        MaterialPageRoute(

          builder: (context) => DashBoard(),

        ),

      );

    }

  }

  bool isPasswordVisible = false;

  @override

  Widget build(BuildContext context) {

    return SafeArea(

      child: Scaffold(

        body: Container(

          height: double.infinity,

          width: double.infinity,

          alignment: Alignment.center,

          decoration: BoxDecoration(

              gradient: LinearGradient(

                  colors: [Colors.teal.shade200, Colors.purple.shade900])),

          child: SingleChildScrollView(

            child: Column(

              children: [

                Align(

                  alignment: Alignment.topRight,

                  child: Container(

                    height: 100,

                    width: 300,

                    decoration: const BoxDecoration(

                        gradient:

                            LinearGradient(colors: [Colors.red, Colors.yellow]),

                        boxShadow: [

                          BoxShadow(

                              blurRadius: 4,

                              spreadRadius: 3,

                              color: Colors.black12)

                        ],

                        borderRadius: BorderRadius.only(

                            topLeft: Radius.circular(200),

                            bottomRight: Radius.circular(200))),

                    child: Padding(

                      padding: const EdgeInsets.only(bottom: 35, left: 65),

                      child: Row(

                        crossAxisAlignment: CrossAxisAlignment.end,

                        children: [

                          const Text(

                            'Let\'s',

                            style: TextStyle(

                                fontSize: 30,

                                fontWeight: FontWeight.bold,

                                color: Colors.white,

                                shadows: [

                                  Shadow(

                                      color: Colors.black45,

                                      offset: Offset(1, 1),

                                      blurRadius: 5)

                                ]),

                          ),

                          Text(

                            ' Register',

                            style: TextStyle(

                                fontSize: 30,

                                fontWeight: FontWeight.bold,

                                color: Colors.pink.shade600,

                                shadows: const [

                                  Shadow(

                                      color: Colors.black45,

                                      offset: Offset(1, 1),

                                      blurRadius: 5)

                                ]),

                          ),

                        ],

                      ),

                    ),

                  ),

                ),

                const SizedBox(

                  height: 40,

                ),

                Padding(

                  padding: const EdgeInsets.symmetric(horizontal: 30)

                      .copyWith(bottom: 10),

                  child: TextField(

                    controller: user,

                    style: const TextStyle(color: Colors.white, fontSize: 14.5),

                    decoration: InputDecoration(

                        prefixIconConstraints:

                            const BoxConstraints(minWidth: 45),

                        prefixIcon: const Icon(

                          Icons.alternate\_email\_outlined,

                          color: Colors.white70,

                          size: 22,

                        ),

                        border: InputBorder.none,

                        hintText: 'Enter Username',

                        hintStyle: const TextStyle(

                            color: Colors.white60, fontSize: 14.5),

                        enabledBorder: OutlineInputBorder(

                            borderRadius: BorderRadius.circular(100).copyWith(

                                bottomRight: const Radius.circular(0)),

                            borderSide:

                                const BorderSide(color: Colors.white38)),

                        focusedBorder: OutlineInputBorder(

                            borderRadius: BorderRadius.circular(100).copyWith(

                                bottomRight: const Radius.circular(0)),

                            borderSide:

                                const BorderSide(color: Colors.white70))),

                  ),

                ),

                const SizedBox(

                  height: 20,

                ),

                Padding(

                  padding: const EdgeInsets.symmetric(horizontal: 30)

                      .copyWith(bottom: 10),

                  child: TextField(

                    controller: pass,

                    style: const TextStyle(color: Colors.white, fontSize: 14.5),

                    obscureText: isPasswordVisible ? false : true,

                    decoration: InputDecoration(

                        prefixIconConstraints:

                            const BoxConstraints(minWidth: 45),

                        prefixIcon: const Icon(

                          Icons.lock,

                          color: Colors.white70,

                          size: 22,

                        ),

                        suffixIconConstraints:

                            const BoxConstraints(minWidth: 45, maxWidth: 46),

                        suffixIcon: GestureDetector(

                          onTap: () {

                            setState(() {

                              isPasswordVisible = !isPasswordVisible;

                            });

                          },

                          child: Icon(

                            isPasswordVisible

                                ? Icons.visibility

                                : Icons.visibility\_off,

                            color: Colors.white70,

                            size: 22,

                          ),

                        ),

                        border: InputBorder.none,

                        hintText: 'Enter Password',

                        hintStyle: const TextStyle(

                            color: Colors.white60, fontSize: 14.5),

                        enabledBorder: OutlineInputBorder(

                            borderRadius: BorderRadius.circular(100).copyWith(

                                bottomRight: const Radius.circular(0)),

                            borderSide:

                                const BorderSide(color: Colors.white38)),

                        focusedBorder: OutlineInputBorder(

                            borderRadius: BorderRadius.circular(100).copyWith(

                                bottomRight: const Radius.circular(0)),

                            borderSide:

                                const BorderSide(color: Colors.white70))),

                  ),

                ),

                const SizedBox(

                  height: 50,

                ),

                GestureDetector(

                  onTap: () {

                    register();

                  },

                  child: Container(

                    height: 53,

                    width: double.infinity,

                    margin: const EdgeInsets.symmetric(horizontal: 30),

                    alignment: Alignment.center,

                    decoration: BoxDecoration(

                        boxShadow: [

                          BoxShadow(

                              blurRadius: 4,

                              color: Colors.black12.withOpacity(.2),

                              offset: const Offset(2, 2))

                        ],

                        borderRadius: BorderRadius.circular(100)

                            .copyWith(bottomRight: const Radius.circular(0)),

                        gradient: LinearGradient(colors: [

                          Colors.purple.shade600,

                          Colors.amber.shade900

                        ])),

                    child: Text('Signup',

                        style: TextStyle(

                            color: Colors.white.withOpacity(.8),

                            fontSize: 15,

                            fontWeight: FontWeight.bold)),

                  ),

                ),

                const SizedBox(

                  height: 50,

                ),

                const Text('Already have an account?',

                    style: TextStyle(color: Colors.white70, fontSize: 13)),

                const SizedBox(

                  height: 20,

                ),

                GestureDetector(

                  onTap: () {

                    Navigator.push(

                      context,

                      MaterialPageRoute(

                          builder: (context) => const MyHomePage()),

                    );

                  },

                  child: Container(

                    height: 53,

                    width: double.infinity,

                    margin: const EdgeInsets.symmetric(horizontal: 30),

                    alignment: Alignment.center,

                    decoration: BoxDecoration(

                      border: Border.all(color: Colors.white60),

                      borderRadius: BorderRadius.circular(100)

                          .copyWith(bottomRight: const Radius.circular(0)),

                    ),

                    child: Text('Login',

                        style: TextStyle(

                            color: Colors.white.withOpacity(.8),

                            fontSize: 15,

                            fontWeight: FontWeight.bold)),

                  ),

                ),

                const SizedBox(

                  height: 20,

                )

              ],

            ),

          ),

        ),

      ),

    );

  }

}

**main.dart**

import 'package:flutter/material.dart';

import 'dart:convert';

import 'package:fluttertoast/fluttertoast.dart';

import 'package:mysql/DashBoard.dart';

import 'register.dart';

import 'package:http/http.dart' as http;

// import 'DashBoard.dart';

void main() {

  runApp(const MyApp());

}

class MyApp extends StatelessWidget {

  const MyApp({Key? key}) : super(key: key);

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      title: 'Flutter Demo',

      theme: ThemeData(

        primarySwatch: Colors.blue,

        visualDensity: VisualDensity.adaptivePlatformDensity,

      ),

      home: const MyHomePage(),

    );

  }

}

class MyHomePage extends StatefulWidget {

  const MyHomePage({Key? key}) : super(key: key);

  @override

  \_MyHomePageState createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

  TextEditingController user = TextEditingController();

  TextEditingController pass = TextEditingController();

  Future login() async {

    var url = Uri.http("192.168.0.26", '/smapp\_db/login.php', {'q': '{http}'});

    var response = await http.post(url, body: {

      "username": user.text,

      "password": pass.text,

    });

    var data = json.decode(response.body);

    if (data.toString() == "Success") {

      Fluttertoast.showToast(

        msg: 'Login Successful',

        backgroundColor: Colors.green,

        textColor: Colors.white,

        toastLength: Toast.LENGTH\_SHORT,

      );

      Navigator.push(

        context,

        MaterialPageRoute(

          builder: (context) => DashBoard(),

        ),

      );

    } else {

      Fluttertoast.showToast(

        backgroundColor: Colors.red,

        textColor: Colors.white,

        msg: 'Username and password invalid',

        toastLength: Toast.LENGTH\_SHORT,

      );

    }

  }

  bool isPasswordVisible = false;

  @override

  Widget build(BuildContext context) {

    return SafeArea(

      child: Scaffold(

        body: Container(

          height: double.infinity,

          width: double.infinity,

          alignment: Alignment.center,

          decoration: BoxDecoration(

              gradient: LinearGradient(

                  colors: [Colors.teal.shade200, Colors.purple.shade900])),

          child: SingleChildScrollView(

            child: Column(

              children: [

                Align(

                  alignment: Alignment.topRight,

                  child: Container(

                    height: 100,

                    width: 300,

                    decoration: const BoxDecoration(

                        gradient:

                            LinearGradient(colors: [Colors.red, Colors.yellow]),

                        boxShadow: [

                          BoxShadow(

                              blurRadius: 4,

                              spreadRadius: 3,

                              color: Colors.black12)

                        ],

                        borderRadius: BorderRadius.only(

                            topLeft: Radius.circular(200),

                            bottomRight: Radius.circular(200))),

                    child: Padding(

                      padding: const EdgeInsets.only(bottom: 35, left: 65),

                      child: Row(

                        crossAxisAlignment: CrossAxisAlignment.end,

                        children: [

                          const Text(

                            'Let\'s',

                            style: TextStyle(

                                fontSize: 30,

                                fontWeight: FontWeight.bold,

                                color: Colors.white,

                                shadows: [

                                  Shadow(

                                      color: Colors.black45,

                                      offset: Offset(1, 1),

                                      blurRadius: 5)

                                ]),

                          ),

                          Text(

                            ' Login',

                            style: TextStyle(

                                fontSize: 30,

                                fontWeight: FontWeight.bold,

                                color: Colors.pink.shade600,

                                shadows: const [

                                  Shadow(

                                      color: Colors.black45,

                                      offset: Offset(1, 1),

                                      blurRadius: 5)

                                ]),

                          ),

                        ],

                      ),

                    ),

                  ),

                ),

                const SizedBox(

                  height: 40,

                ),

                Padding(

                  padding: const EdgeInsets.symmetric(horizontal: 30)

                      .copyWith(bottom: 10),

                  child: TextField(

                    controller: user,

                    style: const TextStyle(color: Colors.white, fontSize: 14.5),

                    decoration: InputDecoration(

                        prefixIconConstraints:

                            const BoxConstraints(minWidth: 45),

                        prefixIcon: const Icon(

                          Icons.alternate\_email\_outlined,

                          color: Colors.white70,

                          size: 22,

                        ),

                        border: InputBorder.none,

                        hintText: 'Enter Username',

                        hintStyle: const TextStyle(

                            color: Colors.white60, fontSize: 14.5),

                        enabledBorder: OutlineInputBorder(

                            borderRadius: BorderRadius.circular(100).copyWith(

                                bottomRight: const Radius.circular(0)),

                            borderSide:

                                const BorderSide(color: Colors.white38)),

                        focusedBorder: OutlineInputBorder(

                            borderRadius: BorderRadius.circular(100).copyWith(

                                bottomRight: const Radius.circular(0)),

                            borderSide:

                                const BorderSide(color: Colors.white70))),

                  ),

                ),

                const SizedBox(

                  height: 20,

                ),

                Padding(

                  padding: const EdgeInsets.symmetric(horizontal: 30)

                      .copyWith(bottom: 10),

                  child: TextField(

                    controller: pass,

                    style: const TextStyle(color: Colors.white, fontSize: 14.5),

                    obscureText: isPasswordVisible ? false : true,

                    decoration: InputDecoration(

                        prefixIconConstraints:

                            const BoxConstraints(minWidth: 45),

                        prefixIcon: const Icon(

                          Icons.lock,

                          color: Colors.white70,

                          size: 22,

                        ),

                        suffixIconConstraints:

                            const BoxConstraints(minWidth: 45, maxWidth: 46),

                        suffixIcon: GestureDetector(

                          onTap: () {

                            setState(() {

                              isPasswordVisible = !isPasswordVisible;

                            });

                          },

                          child: Icon(

                            isPasswordVisible

                                ? Icons.visibility

                                : Icons.visibility\_off,

                            color: Colors.white70,

                            size: 22,

                          ),

                        ),

                        border: InputBorder.none,

                        hintText: 'Enter Password',

                        hintStyle: const TextStyle(

                            color: Colors.white60, fontSize: 14.5),

                        enabledBorder: OutlineInputBorder(

                            borderRadius: BorderRadius.circular(100).copyWith(

                                bottomRight: const Radius.circular(0)),

                            borderSide:

                                const BorderSide(color: Colors.white38)),

                        focusedBorder: OutlineInputBorder(

                            borderRadius: BorderRadius.circular(100).copyWith(

                                bottomRight: const Radius.circular(0)),

                            borderSide:

                                const BorderSide(color: Colors.white70))),

                  ),

                ),

                const SizedBox(

                  height: 50,

                ),

                GestureDetector(

                  onTap: () {

                    login();

                  },

                  child: Container(

                    height: 53,

                    width: double.infinity,

                    margin: const EdgeInsets.symmetric(horizontal: 30),

                    alignment: Alignment.center,

                    decoration: BoxDecoration(

                        boxShadow: [

                          BoxShadow(

                              blurRadius: 4,

                              color: Colors.black12.withOpacity(.2),

                              offset: const Offset(2, 2))

                        ],

                        borderRadius: BorderRadius.circular(100)

                            .copyWith(bottomRight: const Radius.circular(0)),

                        gradient: LinearGradient(colors: [

                          Colors.purple.shade600,

                          Colors.amber.shade900

                        ])),

                    child: Text('Login',

                        style: TextStyle(

                            color: Colors.white.withOpacity(.8),

                            fontSize: 15,

                            fontWeight: FontWeight.bold)),

                  ),

                ),

                const SizedBox(

                  height: 50,

                ),

                const Text('Don\'t have an account?',

                    style: TextStyle(color: Colors.white70, fontSize: 13)),

                const SizedBox(

                  height: 20,

                ),

                GestureDetector(

                  onTap: () {

                    Navigator.push(

                      context,

                      MaterialPageRoute(builder: (context) => const Register()),

                    );

                  },

                  child: Container(

                    height: 53,

                    width: double.infinity,

                    margin: const EdgeInsets.symmetric(horizontal: 30),

                    alignment: Alignment.center,

                    decoration: BoxDecoration(

                      border: Border.all(color: Colors.white60),

                      borderRadius: BorderRadius.circular(100)

                          .copyWith(bottomRight: const Radius.circular(0)),

                    ),

                    child: Text('Sign Up',

                        style: TextStyle(

                            color: Colors.white.withOpacity(.8),

                            fontSize: 15,

                            fontWeight: FontWeight.bold)),

                  ),

                ),

                const SizedBox(

                  height: 20,

                )

              ],

            ),

          ),

        ),

      ),

    );

  }

}

**Dashboard.dart**

import 'package:flutter/material.dart';

class DashBoard extends StatefulWidget {

  const DashBoard({super.key});

  @override

  State<DashBoard> createState() => \_DashBoardState();

}

class \_DashBoardState extends State<DashBoard> {

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(),

      body: homepage(),

    );

  }

}

class homepage extends StatefulWidget {

  const homepage({super.key});

  @override

  State<homepage> createState() => \_homepageState();

}

class \_homepageState extends State<homepage> {

  @override

  Widget build(BuildContext context) {

    return Container(

      decoration: const BoxDecoration(color: Colors.amberAccent),

      child: const Center(

          child: Text(

        'Welcome',

        style: TextStyle(fontSize: 30),

      )),

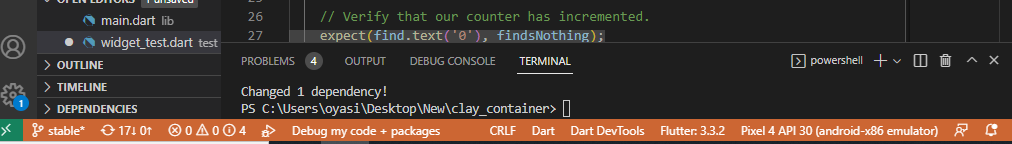
    );

  }

}

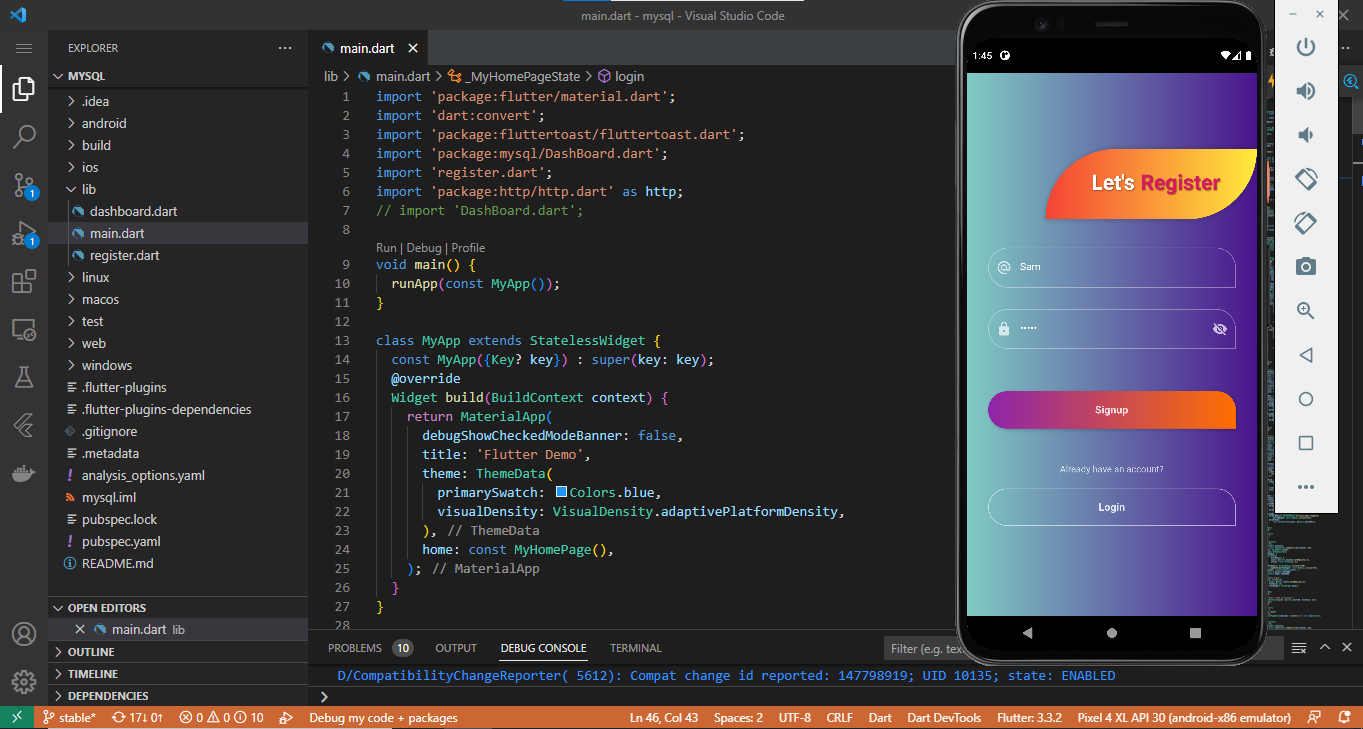
**Step 6 (Device):**

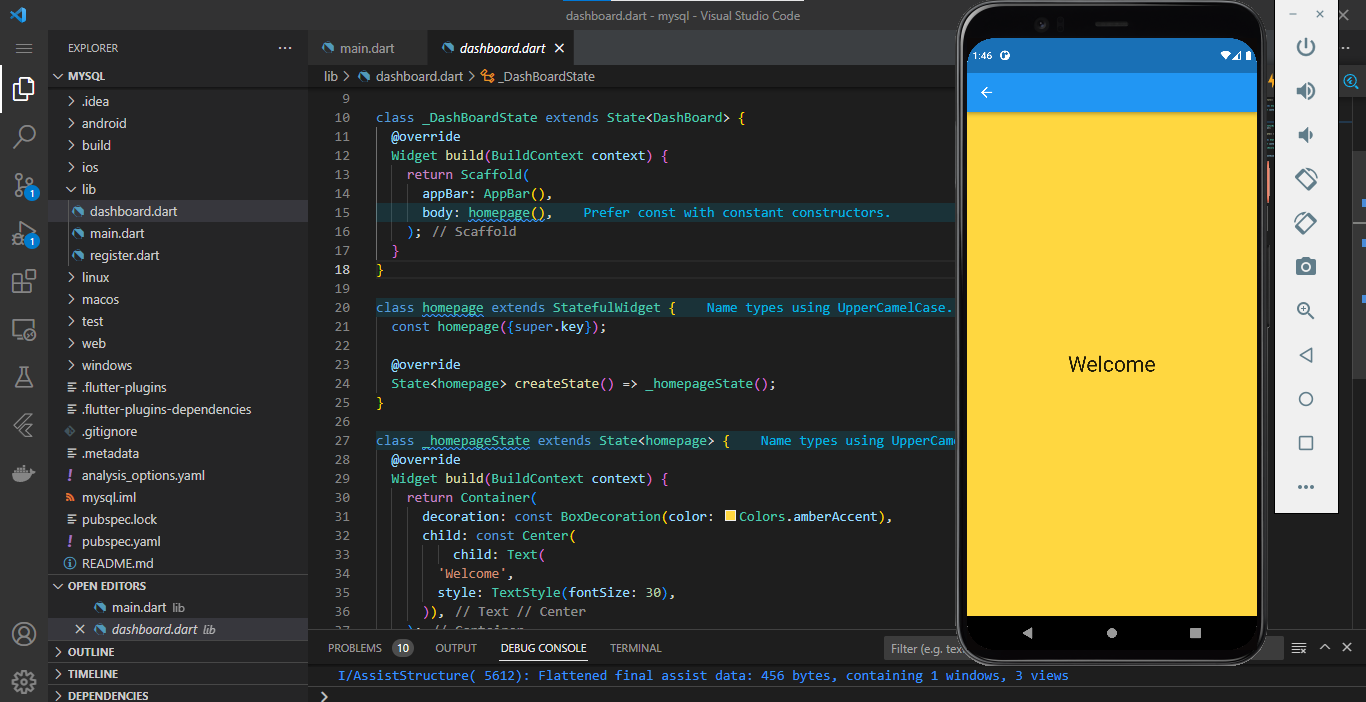
Now choose a device in your VS code



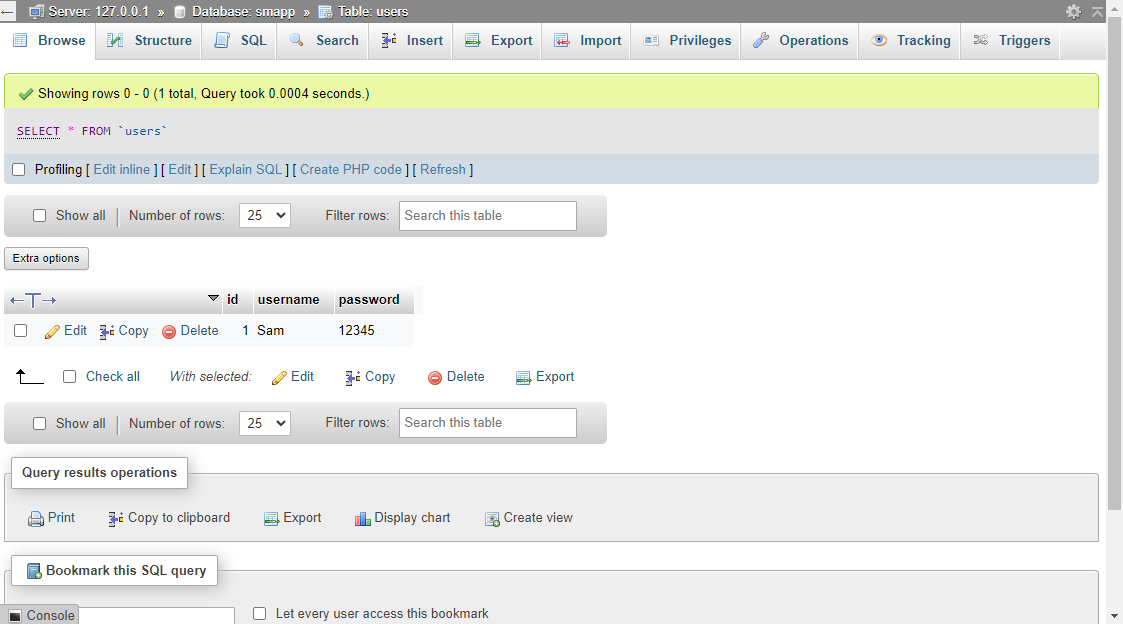
Now run the project by pressing ctrl + f5

**Output:**

****



**Database:**



Warning: Make sure your XAMPP is on and it should be like :



# **API Development:**

**Rest API integration:**

**Step 1:**

Create a new project in the VS code. To do that press ctrl + shift + P

Then click on Flutter: new project

Then choose Application

Now choose a place to save the project and then name the project.

**Step 2:**

Run the following command in the Vs code terminal to add the necessary dependencies in your pubspec.yaml.

flutter pub add http

**Step 3:**

the file structure for this project will the like the image below:



**Step 4:**

Now Let’s delete the default template from flutter main.dart file. And replace the code with the code below

import 'package:flutter/material.dart';

import 'package:rest\_api/home\_page.dart';

void main() {

  runApp(const MyApp());

}

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      title: 'Flutter Demo',

      theme: ThemeData(

        primarySwatch: Colors.amber,

      ),

      home: HomePage(),

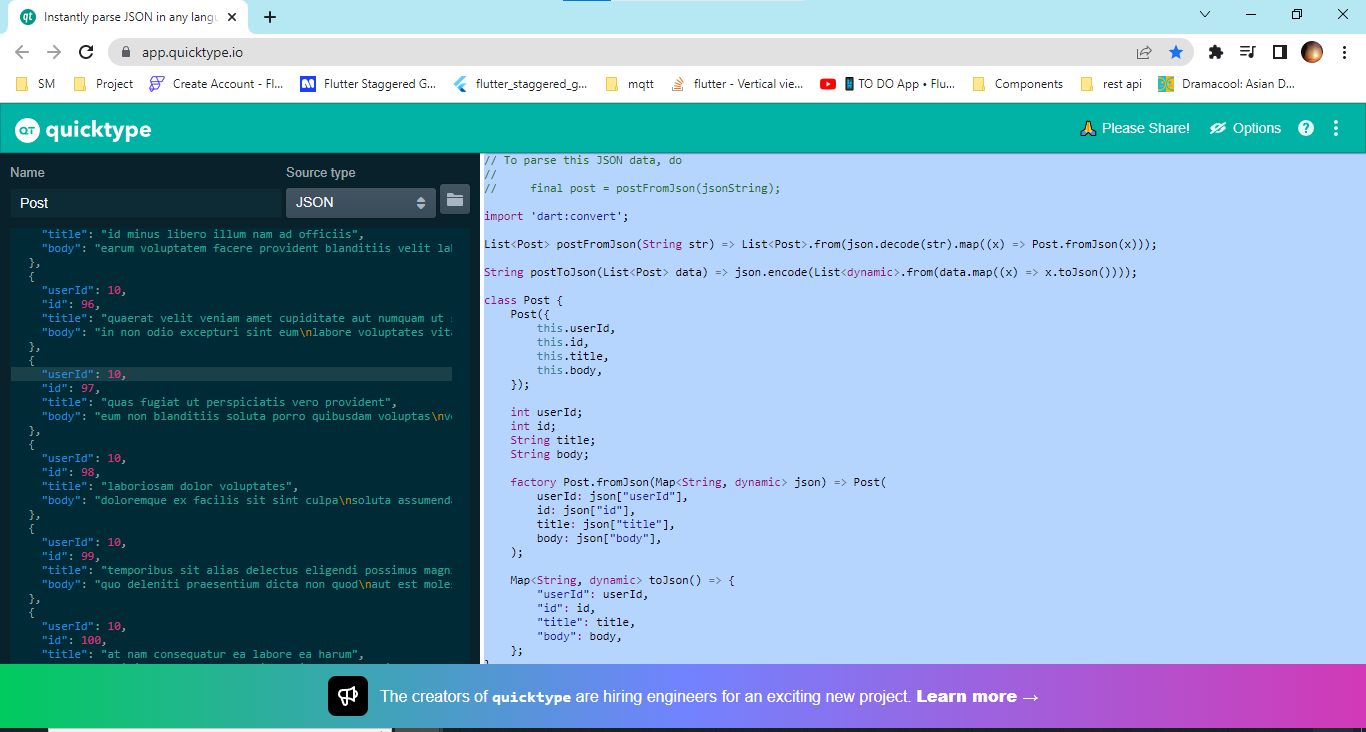
    );

  }

}

Now we have to get an API Jason file from a website but for this project we are going to use an API Jason file from <https://jsonplaceholder.typicode.com/posts> here.

Next, we have to convert this Jason to Dart. You can use <https://app.quicktype.io/> this online converter to convert this file. We have to change the name to “Post”.



Now we have to create a file naming post.dart and paste the code here. Even though it is already in dart we have to change some configuration for now we have to change if the data type is required or not and if the data is nullable we can just use ? to let the dart know that the variable can be nullable(this is to provide the null safety). The final code is given bellow.

// To parse this JSON data, do

//

//     final post = postFromJson(jsonString);

import 'dart:convert';

List<Post> postFromJson(String str) =>

    List<Post>.from(json.decode(str).map((x) => Post.fromJson(x)));

String postToJson(List<Post> data) =>

    json.encode(List<dynamic>.from(data.map((x) => x.toJson())));

class Post {

  Post({

    required this.userId,

    required this.id,

    required this.title,

    this.body,

  });

  int userId;

  int id;

  String title;

  String? body;

  factory Post.fromJson(Map<String, dynamic> json) => Post(

        userId: json["userId"],

        id: json["id"],

        title: json["title"],

        body: json["body"],

      );

  Map<String, dynamic> toJson() => {

        "userId": userId,

        "id": id,

        "title": title,

        "body": body,

      };

}

Now let’s create the home\_page.dart file where we will structure the view of the application and add the code given below:

import 'package:flutter/material.dart';

import 'package:flutter/src/widgets/container.dart';

import 'package:flutter/src/widgets/framework.dart';

import 'package:rest\_api/post.dart';

import 'package:rest\_api/remote\_services.dart';

class HomePage extends StatefulWidget {

  const HomePage({super.key});

  @override

  State<HomePage> createState() => \_HomePageState();

}

class \_HomePageState extends State<HomePage> {

  List<Post>? posts;

  var isLoaded = false;

  @override

  void initState() {

    super.initState();

    //fetch data from API

    getData();

  }

  getData() async {

    posts = await RemoteService().getPosts();

    if (posts != null) {

      setState(() {

        isLoaded = true;

      });

    }

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        centerTitle: true,

        title: const Text('Posts'),

      ),

      body: Visibility(

        visible: isLoaded,

        replacement: const Center(

          child: CircularProgressIndicator(),

        ),

        child: ListView.builder(

          itemCount: posts?.length,

          itemBuilder: (context, index) {

            return Container(

              padding: const EdgeInsets.all(16),

              child: Row(

                children: [

                  Container(

                    height: 50,

                    width: 50,

                    decoration: BoxDecoration(

                      borderRadius: BorderRadius.circular(12),

                      color: Colors.grey[300],

                    ),

                  ),

                  const SizedBox(

                    width: 16,

                  ),

                  Expanded(

                    child: Column(

                      crossAxisAlignment: CrossAxisAlignment.start,

                      children: [

                        Text(

                          posts![index].title,

                          maxLines: 2,

                          overflow: TextOverflow.ellipsis,

                          style: const TextStyle(

                            fontSize: 24,

                            fontWeight: FontWeight.bold,

                          ),

                        ),

                        Text(

                          posts![index].body ?? '',

                          maxLines: 3,

                          overflow: TextOverflow.ellipsis,

                        ),

                      ],

                    ),

                  ),

                ],

              ),

            );

          },

        ),

      ),

    );

  }

}

Now we have to connect the API into the dart file. So for that we have to create another file named remote\_services.dart and place the below code in the file.

import 'package:rest\_api/post.dart';

import 'package:http/http.dart' as http;

class RemoteService {

  Future<List<Post>?> getPosts() async {

    var client = http.Client();

    var uri = Uri.parse('https://jsonplaceholder.typicode.com/posts');

    var response = await client.get(uri);

    if (response.statusCode == 200) {

      var jason = response.body;

      return postFromJson(jason);

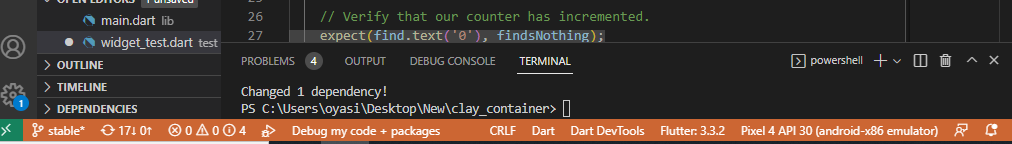
    }

  }

}

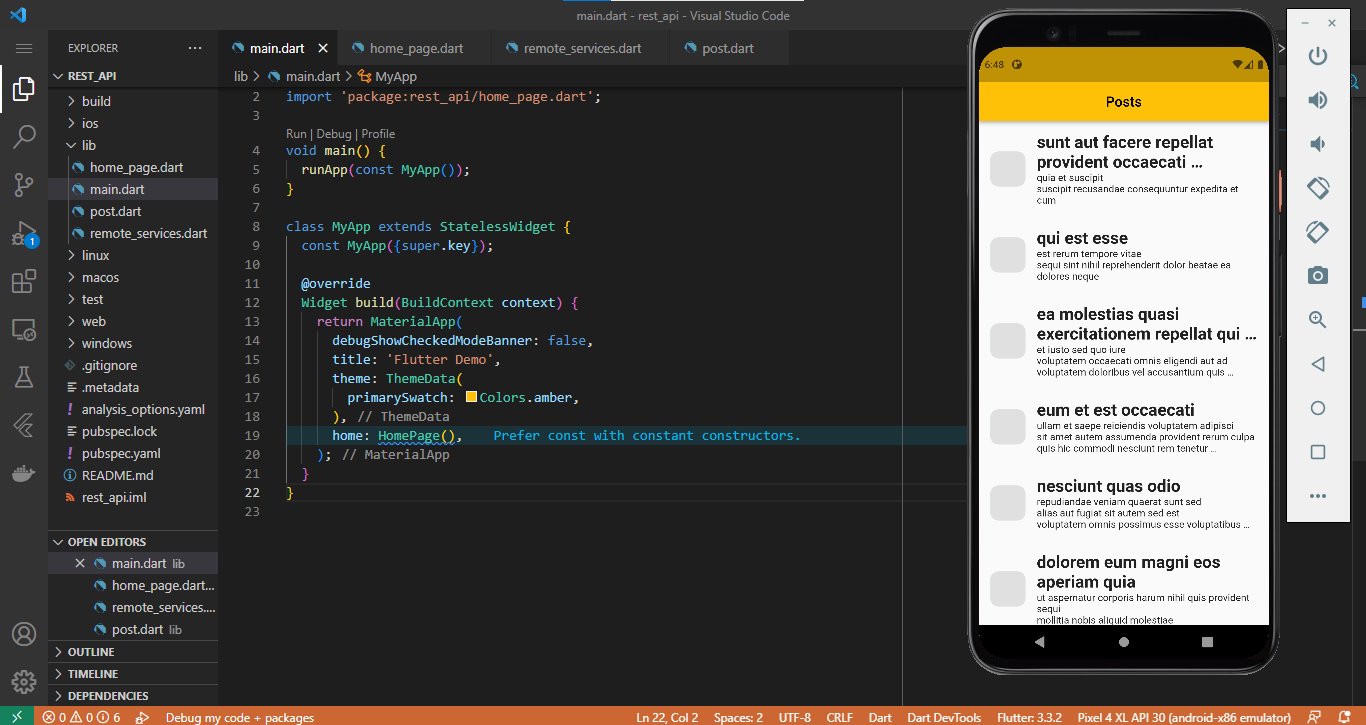
**Step 5:**

Now choose a device in your VS code



Now run the project by pressing ctrl + f5

**Output:**



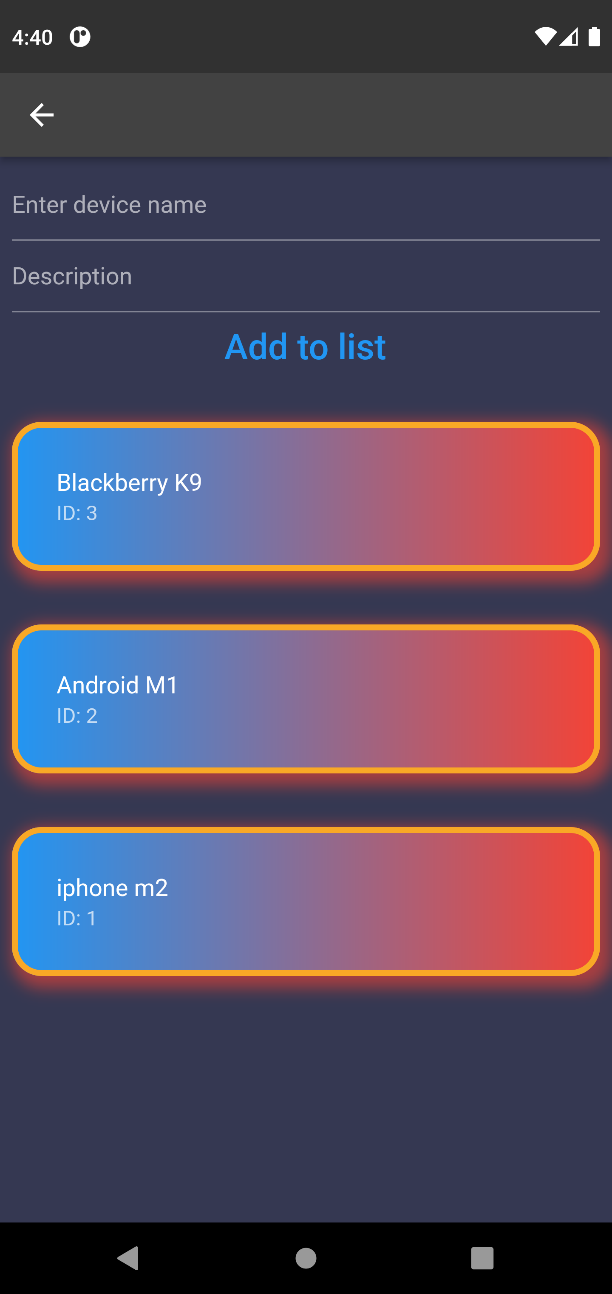
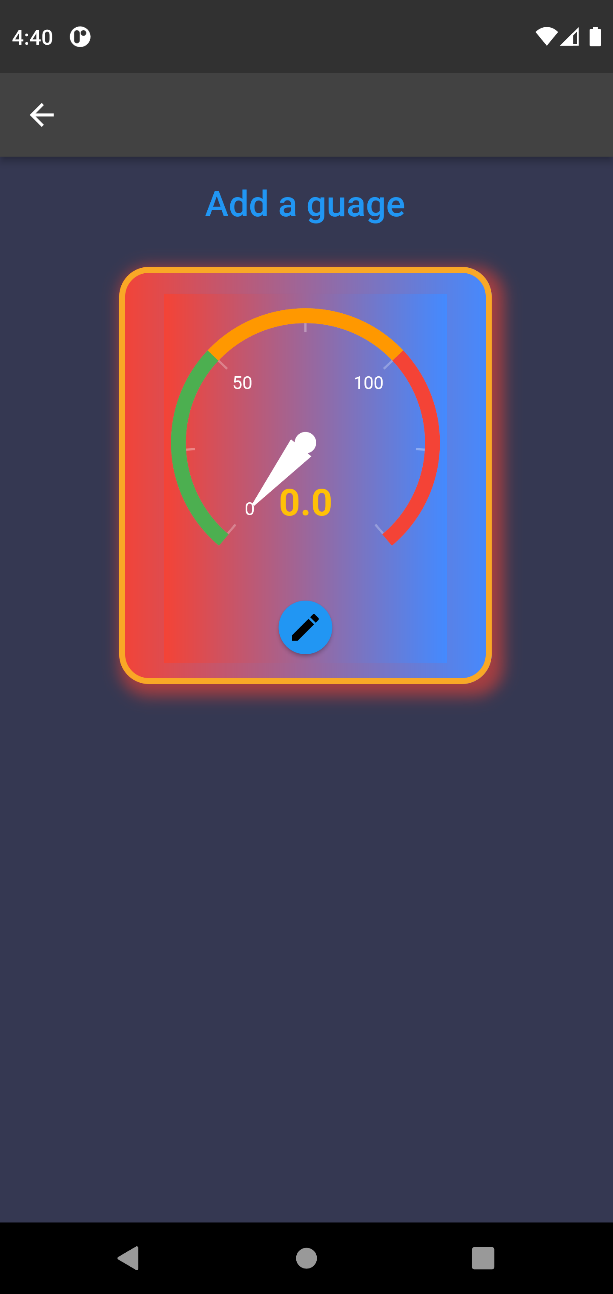
# **Project:**

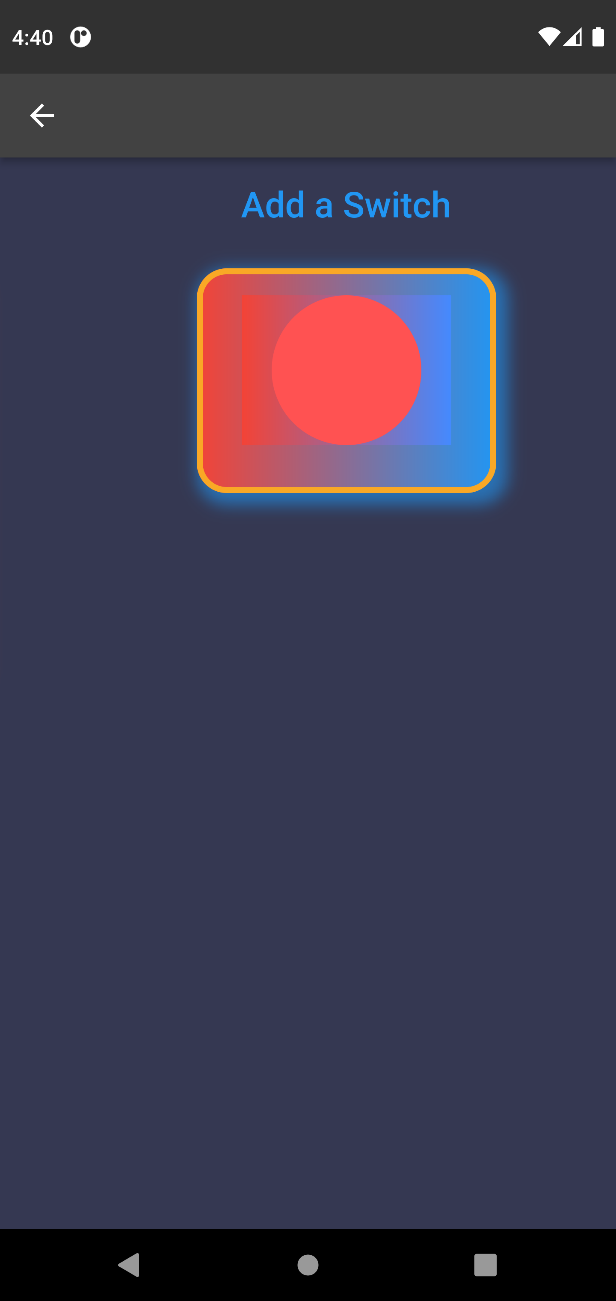
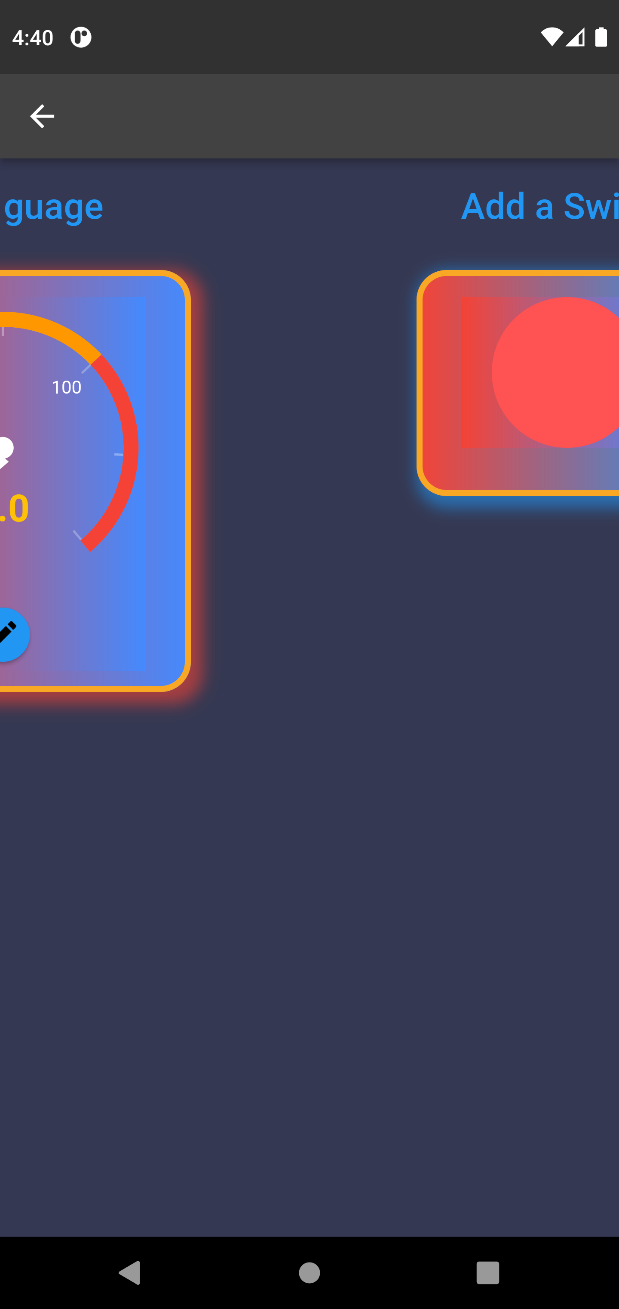
In this section of our Flutter course, we will be creating a flutter application using Mqtt concept.

## **First Steps:**

The first step of starting any project is to plan and decide what are the necessary things and the objective of the project. After it is decided then we can proceed to the designing part. To design the interface, we can use figma which is a very good UI designing platform.

As it is out first project we will go with a simple UI design. The interface should look like the following

After the designing part is done we will advance to the coding part.

## **Next Steps**

**Step 1:**

Create a new project in the VS code. To do that press ctrl + shift + P

Then click on Flutter: new project

Then choose Application

Now choose a place to save the project and then name the project.

**Step 2:**

Now we need to add the following dependencies in the pubspec.yaml file.

  provider: ^6.0.3

  dotted\_border: ^2.0.0+2

  sqflite: ^2.1.0+1

  reorderable\_grid\_view: ^2.2.5

  mqtt\_client: ^9.7.2

  flutter\_inset\_box\_shadow: ^1.0.8

  flutter\_gauge: ^1.0.8

  syncfusion\_flutter\_gauges: ^20.3.49

  lite\_rolling\_switch: ^1.0.0

  flutter\_colorpicker: ^1.0.3

  path\_provider: ^2.0.11

  flutter\_slidable: ^2.0.0

  get\_it: ^7.2.0

  sleek\_circular\_slider: ^2.0.1

  clay\_containers: ^0.3.2

  joystick: ^1.1.0

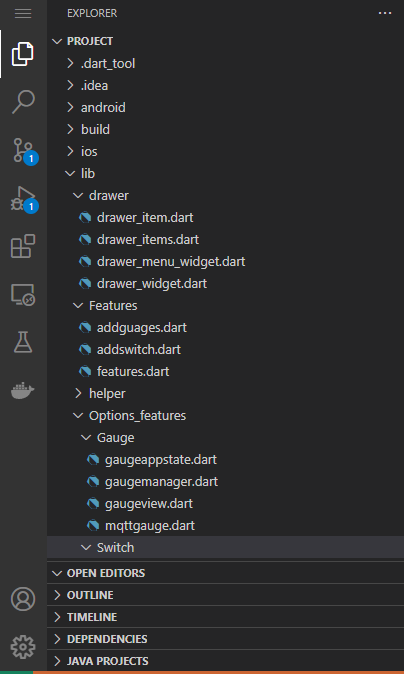
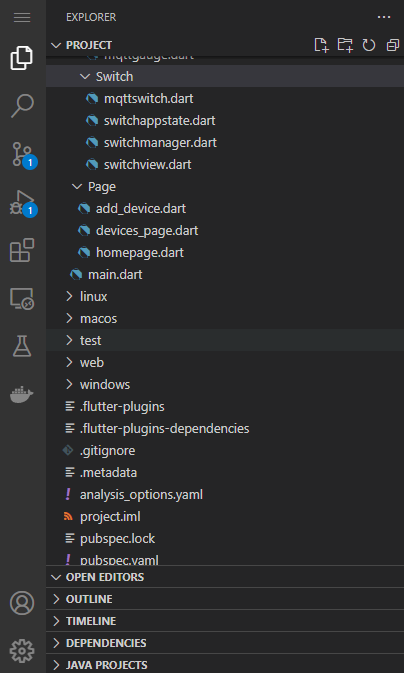
  flutter\_joystick: ^0.0.2

  get: ^4.6.5

  font\_awesome\_flutter: ^10.2.1

**Step 3:**

the file structure for this project will the like the image below:

**Step 4:**

Now Let’s delete the default template from flutter main.dart file. And replace the code with the code below

Notification: Please refer to the comment lines in the code for explanation

import 'package:flutter/material.dart';

import 'package:project/Page/devices\_page.dart';

import 'package:project/helper/theme\_data.dart';

import 'package:project/drawer/drawer\_item.dart';

import 'package:project/drawer/drawer\_items.dart';

import 'package:project/drawer/drawer\_widget.dart';

void main() {

  runApp(const MyApp());

}

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  // This widget is the root of your application.

  @override

  Widget build(BuildContext context) => MaterialApp(

        debugShowCheckedModeBanner: false,

        title: 'Flutter Demo',

        theme: ThemeData.dark().copyWith(

          scaffoldBackgroundColor: Color.fromARGB(255, 53, 56, 82),

          primaryColor: CustomColors.clockBG,

        ),

        home: MainPage(),

      );

}

class MainPage extends StatefulWidget {

  const MainPage({super.key});

  @override

  State<MainPage> createState() => \_MainPageState();

}

// here we will decalre the variable inside a class which will decide the

class \_MainPageState extends State<MainPage> {

  late double xOffset;

  late double yOffset;

  late double scaleFactor;

  late bool isDrawerOpen;

  DrawerItem item = DrawerItems.home;

  bool isDragging = false;

  // Now we will have to set the initial state of the drawer for opeening drawer and closing drawer

  @override

  void initState() {

    super.initState();

    closeDrawer();

  }

  void closeDrawer() => setState(() {

        xOffset = 0;

        yOffset = 0;

        scaleFactor = 1;

        isDrawerOpen = false;

      });

  void openDrawer() => setState(() {

        xOffset = 230;

        yOffset = 150;

        scaleFactor = 0.6;

        isDrawerOpen = true;

      });

// Here we will buld Drawer option in the app page

  @override

  Widget build(BuildContext context) => Scaffold(

      backgroundColor: Theme.of(context).primaryColor,

      body: Stack(

        children: [

          buildDrawer(),

          buildPage(),

        ],

      ));

  Widget buildDrawer() => SafeArea(

        child: Container(

          width: xOffset,

          child: DrawerWidget(

            onSelectedItem: (item) {

              setState(() => this.item = item);

              closeDrawer();

            },

          ),

        ),

      );

  Widget buildPage() {

    return WillPopScope(

      onWillPop: () async {

        if (isDrawerOpen) {

          closeDrawer();

          return false;

        } else {

          return true;

        }

      },

// Here we will declare the gesture to what to happen when we tap on the draewer icon.

      child: GestureDetector(

        onTap: closeDrawer,

        onHorizontalDragStart: (details) => isDragging = true,

        onHorizontalDragUpdate: (details) {

          if (!isDragging) return;

          const delta = 1;

          if (details.delta.dx > delta) {

            openDrawer();

          } else if (details.delta.dx < -delta) {

            closeDrawer();

          }

          isDragging = false;

        },

        child: AnimatedContainer(

          duration: Duration(milliseconds: 250),

          transform: Matrix4.translationValues(xOffset, yOffset, 0)

            ..scale(scaleFactor),

          child: AbsorbPointer(

              absorbing: isDrawerOpen,

              child: ClipRRect(

                  borderRadius: BorderRadius.circular(isDrawerOpen ? 70 : 0),

                  child: Container(

                    color: isDrawerOpen

                        ? Colors

                            .transparent //-------------------------------- Can be changed if needed ---------------------------------//

                        : Theme.of(context).primaryColor,

                    child: getDrawerPage(),

                  ))),

        ),

      ),

    );

  }

// here we will name and link the pages and name the pages. But here we are using only Device page because this the only page for now.

  Widget getDrawerPage() {

    switch (item) {

      case DrawerItems.device:

        return DevicesPage(openDrawer: openDrawer);

      case DrawerItems.message:

        return DevicesPage(openDrawer: openDrawer);

      case DrawerItems.gauge:

        return DevicesPage(openDrawer: openDrawer);

      case DrawerItems.remote:

        return DevicesPage(openDrawer: openDrawer);

      case DrawerItems.home:

      default:

        return DevicesPage(openDrawer: openDrawer);

    }

  }

}

Now, in the drawer\_item.dart file we are going to declare two variable in the DrawerItem class which we need to use to make the drawer and we have to make them required.

import 'package:flutter/material.dart';

class DrawerItem {

  final String title;

  final IconData icon;

  const DrawerItem({

    required this.title,

    required this.icon,

  });

}

Now let’s declare Drawer pages title name and we have to decide the icon for each option. We also have to declare the option list a list so that we can use it in the main.dart for listing all the options.

drawer\_items.dart

import 'dart:developer';

import 'package:flutter/material.dart';

import 'package:font\_awesome\_flutter/font\_awesome\_flutter.dart';

import 'package:project/drawer/drawer\_item.dart';

class DrawerItems {

  static const home = DrawerItem(title: 'Home', icon: FontAwesomeIcons.house);

  static const device =

      DrawerItem(title: 'Device', icon: FontAwesomeIcons.toolbox);

  static const message =

      DrawerItem(title: 'Message', icon: FontAwesomeIcons.message);

  static const gauge = DrawerItem(title: 'Gauge', icon: FontAwesomeIcons.gauge);

  static const remote =

      DrawerItem(title: 'Switch', icon: FontAwesomeIcons.toggleOn);

  static final List<DrawerItem> all = [

    home,

    device,

    message,

    gauge,

    remote,

  ];

}

In drawer\_menu\_widget.dart, we will declare onclicked method.

import 'package:flutter/material.dart';

import 'package:font\_awesome\_flutter/font\_awesome\_flutter.dart';

class DrawerMenuWidget extends StatelessWidget {

  final VoidCallback onClicked;

  const DrawerMenuWidget({

    Key? key,

    required this.onClicked,

  }) : super(key: key);

  @override

  Widget build(BuildContext context) => IconButton(

      onPressed: onClicked, icon: FaIcon(FontAwesomeIcons.alignLeft));

}

drawer\_widget.dart file will be used to map the data from the list that declared previously to the drawer.

import 'package:flutter/material.dart';

import 'package:project/drawer/drawer\_item.dart';

import 'package:project/drawer/drawer\_items.dart';

class DrawerWidget extends StatelessWidget {

  final ValueChanged<DrawerItem> onSelectedItem;

  const DrawerWidget({

    Key? key,

    required this.onSelectedItem,

  }) : super(key: key);

  @override

  Widget build(BuildContext context) => Container(

        padding: EdgeInsets.fromLTRB(16, 160, 16, 0),

        child: SingleChildScrollView(

          child: Column(

            children: [

              buildDrawerItems(context),

            ],

          ),

        ),

      );

  Widget buildDrawerItems(BuildContext context) => Column(

        children: DrawerItems.all

            .map(

              (item) => ListTile(

                contentPadding:

                    EdgeInsets.symmetric(horizontal: 24, vertical: 8),

                leading: Icon(item.icon, color: Colors.white),

                title: Text(

                  item.title,

                  style: TextStyle(color: Colors.white, fontSize: 20),

                ),

                onTap: () => onSelectedItem(item),

              ),

            )

            .toList(),

      );

}

Now we need 2 pages where one is for displaying the devices and another one is for displaying the features. But first we need to create the page to add the devices with the SQLite package. For the device page we are going to divide coding into 2 pages add\_device.dart & device\_page.dart .

device\_page.dart is for building the appbar of the device page and add\_device.dart is for adding new device and saving the data into the device local storage.

device\_page.dart

import 'package:flutter/material.dart';

import 'package:project/drawer/drawer\_menu\_widget.dart';

import 'package:project/Page/add\_device.dart';

class DevicesPage extends StatefulWidget {

  final VoidCallback openDrawer;

  const DevicesPage({

    Key? key,

    required this.openDrawer,

  }) : super(key: key);

  @override

  State<DevicesPage> createState() => \_DevicesPageState();

}

class \_DevicesPageState extends State<DevicesPage> {

  @override

  Widget build(BuildContext context) => Scaffold(

        backgroundColor: Color.fromARGB(

            255, 53, 56, 82), //------------body color------------//

        //--------------------------- App Bar Text & icons --------------------------- //

        appBar: AppBar(

          leading: DrawerMenuWidget(onClicked: widget.openDrawer),

          title: const Text('SmApp'),

          backgroundColor: Colors.transparent,

          centerTitle: true,

          //------------------ App bar color customizatuion ------------------//

          shadowColor: const Color.fromARGB(255, 89, 241, 168),

          flexibleSpace: Container(

            decoration: const BoxDecoration(

                gradient: LinearGradient(

                    begin: Alignment.centerLeft,

                    end: Alignment.centerRight,

                    colors: <Color>[

                      Color.fromARGB(255, 76, 39, 176),

                      Color.fromARGB(255, 113, 228, 248)

                    ]),

                boxShadow: [

                  BoxShadow(

                    color: Color.fromARGB(255, 15, 4, 80),

                    offset: Offset(

                      5.0,

                      5.0,

                    ),

                    blurRadius: 20.0,

                    spreadRadius: 2.0,

                  )

                ]),

          ),

        ),

        body: const dashboard(),

      );

}

add\_device.dart

import 'dart:async';

import 'package:flutter/material.dart';

import 'package:flutter\_slidable/flutter\_slidable.dart';

import 'package:path\_provider/path\_provider.dart';

import 'package:project/Page/features.dart';

import 'package:sqflite/sqflite.dart';

class Device implements Comparable {

  final int id;

  final String firstName;

  final String lastName;

  const Device({

    required this.id,

    required this.firstName,

    required this.lastName,

  });

  String get fullName => '$firstName $lastName';

  Device.fromRow(Map<String, Object?> row)

      : id = row['ID'] as int,

        firstName = row['FIRST\_NAME'] as String,

        lastName = row['LAST\_NAME'] as String;

  @override

  int compareTo(covariant other) => other.id.compareTo(id);

  @override

  bool operator ==(covariant Device other) => id == other.id;

  @override

  int get hashCode => id.hashCode;

  @override

  String toString() =>

      'Device, id = $id, firstName: $firstName, lastname: $lastName';

  static removeAt(int oldIndex) {

    Device;

  }

  static insert(int newIndex) {

    Device;

  }

}

class DeviceDB {

  final String dbName;

  Database? \_db;

  List<Device> \_devices = [];

  final \_streamController = StreamController<List<Device>>.broadcast();

  DeviceDB({required this.dbName});

  Future<List<Device>> \_fetchTool() async {

    final db = \_db;

    if (db == null) {

      return [];

    }

    try {

      final read = await db.query(

        'DEVICE',

        distinct: true,

        columns: [

          'ID',

          'FIRST\_NAME',

          'LAST\_NAME',

        ],

        orderBy: 'ID',

      );

      final tool = read.map((row) => Device.fromRow(row)).toList();

      return tool;

    } catch (e) {

      print('Error fetching devices = $e');

      return [];

    }

  }

// C in CRUD

  Future<bool> create(String firstName, String lastName) async {

    final db = \_db;

    if (db == null) {

      return false;

    }

    try {

      final id = await db.insert('DEVICE', {

        'FIRST\_NAME': firstName,

        'LAST\_NAME': lastName,

      });

      final device = Device(

        id: id,

        firstName: firstName,

        lastName: lastName,

      );

      \_devices.add(device);

      \_streamController.add(\_devices);

      return true;

    } catch (e) {

      print('Error in creating device = $e');

      return false;

    }

  }

  Future<bool> delete(Device device) async {

    final db = \_db;

    if (db == null) {

      return false;

    }

    try {

      final deleteCount =

          await db.delete('DEVICE', where: 'ID = ?', whereArgs: [device.id]);

      if (deleteCount == 1) {

        \_devices.remove(device);

        \_streamController.add(\_devices);

        return true;

      } else {

        return false;

      }

    } catch (e) {

      print('Deletion failed with error $e');

      return false;

    }

  }

  Future<bool> close() async {

    final db = \_db;

    if (db == null) {

      return false;

    }

    await db.close();

    return true;

  }

  Future<bool> open() async {

    if (\_db != null) {

      return true;

    }

    final directory = await getApplicationDocumentsDirectory();

    final path = '${directory.path}/$dbName';

    try {

      final db = await openDatabase(path);

      \_db = db;

      // create table

      final create = '''CREATE TABLE IF NOT EXISTS DEVICE (

        ID INTEGER PRIMARY KEY AUTOINCREMENT,

        FIRST\_NAME STRING NOT NULL,

        LAST\_NAME STRING NOT NULL

      )''';

      await db.execute(create);

      // read all existing Device objects from the db

      \_devices = await \_fetchTool();

      \_streamController.add(\_devices);

      return true;

    } catch (e) {

      print('Error = $e');

      return false;

    }

  }

  Future<bool> update(Device device) async {

    final db = \_db;

    if (db == null) {

      return false;

    }

    try {

      final updateCount = await db.update(

        'DEVICE',

        {

          'FIRST\_NAME': device.firstName,

          'LAST\_NAME': device.lastName,

        },

        where: 'ID = ?',

        whereArgs: [device.id],

      );

      if (updateCount == 1) {

        \_devices.removeWhere((other) => other.id == device.id);

        \_devices.add(device);

        \_streamController.add(\_devices);

        return true;

      } else {

        return false;

      }

    } catch (e) {

      print('failed to update device, error = $e');

      return false;

    }

  }

  Stream<List<Device>> all() =>

      \_streamController.stream.map((devices) => devices..sort());

}

class dashboard extends StatefulWidget {

  const dashboard({super.key});

  @override

  State<dashboard> createState() => \_dashboardState();

}

class \_dashboardState extends State<dashboard> {

  late final DeviceDB \_crudStorage;

  List<Device> \_devices = [];

  Color \_color = Color(0xfff9a826);

  @override

  void initState() {

    \_crudStorage = DeviceDB(dbName: 'db.sqlite');

    \_crudStorage.open();

    super.initState();

  }

  @override

  void dispose() {

    \_crudStorage.close();

    super.dispose();

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(),

      body: StreamBuilder(

        stream: \_crudStorage.all(),

        builder: (context, snapshot) {

          print(snapshot);

          switch (snapshot.connectionState) {

            case ConnectionState.active:

            case ConnectionState.waiting:

              if (snapshot.data == null) {

                return const Center(child: CircularProgressIndicator());

              }

              final tool = snapshot.data as List<Device>;

              return Column(

                children: [

                  ComposeWidget(

                    onCompose: (firstName, lastName) {

                      \_crudStorage.create(firstName, lastName);

                    },

                  ),

                  Expanded(

                    child: ReorderableListView.builder(

                      itemCount: tool.length,

                      onReorder: (oldIndex, newIndex) => setState(() {

                        final index =

                            newIndex > oldIndex ? newIndex - 1 : newIndex;

                        final device = tool.removeAt(oldIndex);

                        tool.insert(index, device);

                      }),

                      itemBuilder: (context, index) {

                        final device = tool[index];

                        return Slidable(

                          key: Key('$index'),

                          startActionPane: ActionPane(

                            motion: const StretchMotion(),

                            children: [

                              SlidableAction(

                                backgroundColor: Colors.yellowAccent,

                                icon: Icons.edit,

                                label: 'Edit',

                                onPressed: (context) async {

                                  final editedDevice = await showUpdateDialog(

                                    context,

                                    device,

                                  );

                                  if (editedDevice != null) {

                                    await \_crudStorage.update(editedDevice);

                                  }

                                },

                              )

                            ],

                          ),

                          endActionPane: ActionPane(

                            motion: const StretchMotion(),

                            children: [

                              SlidableAction(

                                  backgroundColor: Colors.redAccent,

                                  icon: Icons.delete\_forever\_rounded,

                                  label: 'Delete',

                                  onPressed: (context) async {

                                    final shouldDelete =

                                        await showDeleteDialog(context);

                                    print(shouldDelete);

                                    if (shouldDelete) {

                                      await \_crudStorage.delete(device);

                                    }

                                  })

                            ],

                          ),

                          child: Padding(

                            padding: const EdgeInsets.all(8.0),

                            child: Container(

                              padding: EdgeInsets.all(10),

                              margin: EdgeInsets.symmetric(vertical: 10),

                              decoration: BoxDecoration(

                                gradient: LinearGradient(

                                  begin: Alignment.centerLeft,

                                  end: Alignment.centerRight,

                                  colors: [Colors.blue, Colors.red],

                                ),

                                boxShadow: [

                                  BoxShadow(

                                    color: [Colors.blue, Colors.red]

                                        .last

                                        .withOpacity(0.6),

                                    blurRadius: 8,

                                    spreadRadius: 4,

                                    offset: Offset(4, 4),

                                  ),

                                ],

                                borderRadius: BorderRadius.circular(20),

                                border: Border.all(color: \_color, width: 4),

                              ),

                              child: ListTile(

                                key: Key('$index'),

                                onTap: () {

                                  Navigator.push(

                                      context,

                                      MaterialPageRoute(

                                          builder: ((context) => feature())));

                                },

                                // title: Text(device.firstName),

                                // subtitle: Text(device.lastName),

                                title: Text(device.fullName),

                                subtitle: Text('ID: ${device.id}'),

                                // title: Container(

                                //     height: 100, width: 200, child: mqttgauge()),

                              ),

                            ),

                          ),

                        );

                      },

                    ),

                  ),

                ],

              );

            default:

              return const Center(child: CircularProgressIndicator());

          }

        },

      ),

    );

  }

}

Future<bool> showDeleteDialog(BuildContext context) {

  return showDialog(

    context: context,

    builder: (context) {

      return AlertDialog(

        content: const Text('Are you sure you want to delete this item?'),

        actions: [

          TextButton(

            onPressed: () {

              Navigator.of(context).pop(false);

            },

            child: const Text('No'),

          ),

          TextButton(

            onPressed: () {

              Navigator.of(context).pop(true);

            },

            child: const Text('Delete'),

          ),

        ],

      );

    },

  ).then((value) {

    if (value is bool) {

      return value;

    } else {

      return false;

    }

  });

}

final \_firstNameController = TextEditingController();

final \_lastNameController = TextEditingController();

Future<Device?> showUpdateDialog(BuildContext context, Device device) {

  \_firstNameController.text = device.firstName;

  \_lastNameController.text = device.lastName;

  return showDialog(

    context: context,

    builder: (context) {

      return AlertDialog(

        content: Column(

          mainAxisSize: MainAxisSize.min,

          children: [

            const Text('Enter your updated Values here:'),

            TextField(

              controller: \_firstNameController,

            ),

            TextField(

              controller: \_lastNameController,

            ),

          ],

        ),

        actions: [

          TextButton(

            onPressed: () {

              Navigator.of(context).pop(null);

            },

            child: const Text('Cancel'),

          ),

          TextButton(

            onPressed: () {

              final editedDevice = Device(

                id: device.id,

                firstName: \_firstNameController.text,

                lastName: \_lastNameController.text,

              );

              Navigator.of(context).pop(editedDevice);

            },

            child: const Text('Save'),

          ),

        ],

      );

    },

  ).then((value) {

    if (value is Device) {

      return value;

    } else {

      return null;

    }

  });

}

typedef OnCompose = void Function(String firstname, String lastName);

class ComposeWidget extends StatefulWidget {

  final OnCompose onCompose;

  const ComposeWidget({super.key, required this.onCompose});

  @override

  State<ComposeWidget> createState() => \_ComposeWidgetState();

}

class \_ComposeWidgetState extends State<ComposeWidget> {

  late final TextEditingController \_firstNameController;

  late final TextEditingController \_lastNameController;

  @override

  void initState() {

    \_firstNameController = TextEditingController();

    \_lastNameController = TextEditingController();

    super.initState();

  }

  @override

  void dispose() {

    \_firstNameController.dispose();

    \_lastNameController.dispose();

    super.dispose();

  }

  @override

  Widget build(BuildContext context) {

    return Padding(

      padding: const EdgeInsets.all(8.0),

      child: Column(

        children: [

          TextField(

            controller: \_firstNameController,

            decoration: InputDecoration(

              hintText: 'Enter device name',

            ),

          ),

          TextField(

            controller: \_lastNameController,

            decoration: InputDecoration(

              hintText: 'Description',

            ),

          ),

          TextButton(

            onPressed: () {

              final firstName = \_firstNameController.text;

              final lastName = \_lastNameController.text;

              widget.onCompose(firstName, lastName);

              \_firstNameController.text = '';

              \_lastNameController.text = '';

            },

            child: Text(

              'Add to list',

              style: TextStyle(fontSize: 24),

            ),

          ),

        ],

      ),

    );

  }

}

In the same way we will create two separate database table for gauge and the switch and then add it into the feature page.

For the coding Please refer to the github link given below:

<https://github.com/oyasizaki/Flutter_Mqtt_Project>

Now we have to connect with the Mqtt broker to send and also to fetch data from the broker for this we will be using flutter mqtt client package. To achieve this task we will be dividing the code into 3 part one is for maintain the appstate another is for managing the connection and the last one is for viewing the data in whatever from we want to connect it.

Here we will discus only about the switch because sending data to guage is also the same.

To Maintain the appstate we have to code as below.

Please refer to the comments in the coding for explanation.

// ignore: file\_names

import 'package:flutter/cupertino.dart';

enum MQTTAppConnectionState { connected, disconnected, connecting }

class MQTTAppState with ChangeNotifier {

  MQTTAppConnectionState \_appConnectionState =

      MQTTAppConnectionState.disconnected;

  // String \_receivedText = '';

  // String \_historyText = '';

  int \_receivedText = 0;

  int \_historyText = 0;

  void setReceivedText(String text) {

    // \_receivedText = text;

    print(text[0]);

    \_receivedText = int.parse(text);

    // \_historyText = '$\_historyText\n$\_receivedText'; // to list the message

    \_historyText = \_receivedText; // view single and latest message

    // temp = double.parse(text);

    // receivetemp = double.parse(\_receivedText);

    // histemp = receivetemp;

    notifyListeners();

  }

  void setAppConnectionState(MQTTAppConnectionState state) {

    \_appConnectionState = state;

    notifyListeners();

  }

  // String get getReceivedText => \_receivedText;

  // String get getHistoryText => \_historyText;

  int get getReceivedText => \_receivedText;

  int get getHistoryText => \_historyText;

  MQTTAppConnectionState get getAppConnectionState => \_appConnectionState;

}

For managing the app connection :

// ignore: file\_names

import 'package:mqtt\_client/mqtt\_client.dart';

import 'package:mqtt\_client/mqtt\_server\_client.dart';

import 'package:project/Options\_features/Switch/switchappstate.dart';

class MQTTManager {

  // Private instance of client

  final MQTTAppState \_currentState;

  MqttServerClient? \_client;

  final String \_identifier;

  final String \_host;

  final String \_topic;

  // Constructor

  // ignore: sort\_constructors\_first

  MQTTManager(

      {required String host,

      required String topic,

      required String identifier,

      required MQTTAppState state})

      : \_identifier = identifier,

        \_host = host,

        \_topic = topic,

        \_currentState = state;

  void initializeMQTTClient() {

    \_client = MqttServerClient(\_host, \_identifier);

    \_client!.port = 1883;

    \_client!.keepAlivePeriod = 20;

    \_client!.onDisconnected = onDisconnected;

    \_client!.secure = false;

    \_client!.logging(on: true);

    /// Add the successful connection callback

    \_client!.onConnected = onConnected;

    \_client!.onSubscribed = onSubscribed;

    final MqttConnectMessage connMess = MqttConnectMessage()

        .withClientIdentifier(\_identifier)

        .withWillTopic(

            'willtopic') // If you set this you must set a will message

        .withWillMessage('My Will message')

        .startClean() // Non persistent session for testing

        .withWillQos(MqttQos.atLeastOnce);

    \_client!.connectionMessage = connMess;

  }

  // Connect to the host

  // ignore: avoid\_void\_async

  void connect() async {

    assert(\_client != null);

    try {

      \_currentState.setAppConnectionState(MQTTAppConnectionState.connecting);

      await \_client!.connect();

    } on Exception {

      disconnect();

    }

  }

  void disconnect() {

    \_client!.disconnect();

  }

  void publish(String message) {

    final MqttClientPayloadBuilder builder = MqttClientPayloadBuilder();

    builder.addString(message);

    \_client!.publishMessage(\_topic, MqttQos.exactlyOnce, builder.payload!);

  }

  /// The subscribed callback

  void onSubscribed(String topic) {}

  /// The unsolicited disconnect callback

  void onDisconnected() {

    if (\_client!.connectionStatus!.returnCode ==

        MqttConnectReturnCode.noneSpecified) {}

    \_currentState.setAppConnectionState(MQTTAppConnectionState.disconnected);

  }

  /// The successful connect callback

  void onConnected() {

    \_currentState.setAppConnectionState(MQTTAppConnectionState.connected);

    \_client!.subscribe(\_topic, MqttQos.atLeastOnce);

    \_client!.updates!.listen((List<MqttReceivedMessage<MqttMessage?>>? c) {

      // ignore: avoid\_as

      final MqttPublishMessage recMess = c![0].payload as MqttPublishMessage;

      // final MqttPublishMessage recMess = c![0].payload;

      final String pt =

          MqttPublishPayload.bytesToStringAsString(recMess.payload.message);

      \_currentState.setReceivedText(pt);

//       var pa =

//           MqttPublishPayload.fromByteBuffer(

//     this.header, this.variableHeader, MqttByteBuffer payloadStream) {

//   readFrom(payloadStream);

// };

    });

  }

}

To view to mqtt data and connect it with the broker the code will be something like

import 'dart:io' show Platform;

import 'package:flutter/material.dart';

import 'package:provider/provider.dart';

import 'package:project/Options\_features/Switch/switchappstate.dart';

import 'package:project/Options\_features/Switch/switchmanager.dart';

import 'package:syncfusion\_flutter\_gauges/gauges.dart';

class SwitchView extends StatefulWidget {

  @override

  State<StatefulWidget> createState() {

    return \_SwitchViewState();

  }

}

class \_SwitchViewState extends State<SwitchView> {

  final TextEditingController \_hostTextController = TextEditingController();

  final TextEditingController \_messageTextController = TextEditingController();

  final TextEditingController \_topicTextController = TextEditingController();

  late MQTTAppState currentAppState;

  late MQTTManager manager;

  bool \_isShow = false;

  @override

  void initState() {

    super.initState();

    /\*

    \_hostTextController.addListener(\_printLatestValue);

    \_messageTextController.addListener(\_printLatestValue);

    \_topicTextController.addListener(\_printLatestValue);

     \*/

  }

  @override

  void dispose() {

    \_hostTextController.dispose();

    \_messageTextController.dispose();

    \_topicTextController.dispose();

    super.dispose();

  }

  /\*

  \_printLatestValue() {

    print("Second text field: ${\_hostTextController.text}");

    print("Second text field: ${\_messageTextController.text}");

    print("Second text field: ${\_topicTextController.text}");

  }

   \*/

  @override

  Widget build(BuildContext context) {

    final MQTTAppState appState = Provider.of<MQTTAppState>(context);

    // Keep a reference to the app state.

    currentAppState = appState;

    final Scaffold scaffold = Scaffold(body: \_buildColumn());

    return scaffold;

  }

  Widget \_buildColumn() {

    return Scaffold(

      body: SingleChildScrollView(

        child: Container(

          width: 600,

          decoration: BoxDecoration(

            gradient: LinearGradient(

              begin: Alignment.centerLeft,

              end: Alignment.centerRight,

              colors: [Colors.red, Colors.blueAccent],

            ),

          ),

          child: Column(

            children: <Widget>[

              Visibility(

                  visible: !\_isShow,

                  child:

                      \_buildScrollableTextWith(currentAppState.getHistoryText)),

              Visibility(

                visible: \_isShow,

                child: Column(

                  children: [

                    \_buildConnectionStateText(\_prepareStateMessageFrom(

                        currentAppState.getAppConnectionState)),

                    \_buildEditableColumn(),

                  ],

                ),

              ),

              ElevatedButton(

                style: ElevatedButton.styleFrom(

                  foregroundColor: Colors.black,

                  shape: CircleBorder(),

                ),

                onPressed: () {

                  setState(

                    () {

                      \_isShow = !\_isShow;

                    },

                  );

                },

                child: Icon(

                  \_isShow ? Icons.hide\_source : Icons.edit,

                ),

              )

            ],

          ),

        ),

      ),

    );

  }

  Widget \_buildEditableColumn() {

    return Padding(

      padding: const EdgeInsets.all(2.0),

      child: Column(

        children: <Widget>[

          Padding(padding: EdgeInsets.symmetric(vertical: 20, horizontal: 10)),

          \_buildTextFieldWith(\_hostTextController, 'Enter broker address',

              currentAppState.getAppConnectionState),

          const SizedBox(height: 10),

          \_buildTextFieldWith(

              \_topicTextController,

              'Enter a topic to subscribe or listen',

              currentAppState.getAppConnectionState),

          const SizedBox(height: 10),

          // \_buildPublishMessageRow(),  //-------------------------------------- message sending box-------------//

          const SizedBox(height: 10),

          \_buildConnecteButtonFrom(currentAppState.getAppConnectionState)

        ],

      ),

    );

  }

  // Widget \_buildPublishMessageRow() {    //-------------------------------------- message sending box-------------//

  //   return Row(

  //     mainAxisAlignment: MainAxisAlignment.spaceEvenly,

  //     children: <Widget>[

  //       Expanded(

  //         child: \_buildTextFieldWith(\_messageTextController, 'Enter a message',

  //             currentAppState.getAppConnectionState),

  //       ),

  //       \_buildSendButtonFrom(currentAppState.getAppConnectionState)

  //     ],

  //   );

  // }

  Widget \_buildConnectionStateText(String status) {

    return Row(

      children: <Widget>[

        Expanded(

          child: Container(

              color: Colors.deepOrangeAccent,

              child: Text(status, textAlign: TextAlign.center)),

        ),

      ],

    );

  }

  Widget \_buildTextFieldWith(TextEditingController controller, String hintText,

      MQTTAppConnectionState state) {

    bool shouldEnable = false;

    if (controller == \_messageTextController &&

        state == MQTTAppConnectionState.connected) {

      shouldEnable = true;

    } else if ((controller == \_hostTextController &&

            state == MQTTAppConnectionState.disconnected) ||

        (controller == \_topicTextController &&

            state == MQTTAppConnectionState.disconnected)) {

      shouldEnable = true;

    }

    return TextField(

        enabled: shouldEnable,

        controller: controller,

        decoration: InputDecoration(

          contentPadding:

              const EdgeInsets.only(left: 0, bottom: 0, top: 0, right: 0),

          labelText: hintText,

        ));

  }

  Widget \_buildScrollableTextWith(int text) {

    //-------------------------------- main box-------------------------//

    return SingleChildScrollView(

      child: Padding(

        padding: const EdgeInsets.only(bottom: 15),

        child: Container(

          height: 100,

          width: 100,

          decoration: BoxDecoration(

              color: text < 1 ? Colors.redAccent : Colors.green,

              borderRadius: BorderRadius.circular(300)),

        ),

      ),

    );

  }

  Widget \_buildConnecteButtonFrom(MQTTAppConnectionState state) {

    return Row(

      children: <Widget>[

        Expanded(

          // ignore: deprecated\_member\_use

          child: ElevatedButton(

            style: const ButtonStyle(

              backgroundColor: MaterialStatePropertyAll<Color>(Colors.green),

            ),

            onPressed: state == MQTTAppConnectionState.disconnected

                ? \_configureAndConnect

                : null,

            child: const Text('Connect'), //

          ),

        ),

        const SizedBox(width: 10),

        Expanded(

          // ignore: deprecated\_member\_use

          child: ElevatedButton(

            style: const ButtonStyle(

              backgroundColor:

                  MaterialStatePropertyAll<Color>(Colors.redAccent),

            ),

            onPressed:

                state == MQTTAppConnectionState.connected ? \_disconnect : null,

            child: const Text('Disconnect'), //

          ),

        ),

      ],

    );

  }

  Widget \_buildSendButtonFrom(MQTTAppConnectionState state) {

    // ignore: deprecated\_member\_use

    return ElevatedButton(

      style: const ButtonStyle(

        backgroundColor: MaterialStatePropertyAll<Color>(Colors.green),

      ),

      child: const Text('Send'),

      onPressed: state == MQTTAppConnectionState.connected

          ? () {

              \_publishMessage(\_messageTextController.text);

            }

          : null, //

    );

  }

  // Utility functions

  String \_prepareStateMessageFrom(MQTTAppConnectionState state) {

    switch (state) {

      case MQTTAppConnectionState.connected:

        return 'Connected';

      case MQTTAppConnectionState.connecting:

        return 'Connecting';

      case MQTTAppConnectionState.disconnected:

        return 'Disconnected';

    }

  }

  void \_configureAndConnect() {

    // ignore: flutter\_style\_todos

    // ignore: todo

    // TODO: Use UUID

    String osPrefix = 'U1';

    if (Platform.isAndroid) {

      osPrefix = 'U2';

    }

    manager = MQTTManager(

        host: \_hostTextController.text,

        topic: \_topicTextController.text,

        identifier: osPrefix,

        state: currentAppState);

    manager.initializeMQTTClient();

    manager.connect();

  }

  void \_disconnect() {

    manager.disconnect();

  }

  void \_publishMessage(String text) {

    String osPrefix = 'U11234567890poiuytrewqasdfghjklm';

    if (Platform.isAndroid) {

      osPrefix = 'U2mnbvcxzasdfghjklpoiuytrewq1234';

    }

    final String message = '$osPrefix>$text';

    manager.publish(message);

    \_messageTextController.clear();

  }

}

Now for desiging we have declared all the theme in theme\_data.dart and placed it into the helper folder

We have created a class named CustomColors and placed all the colors that was used through out the project.

For the full code please visit given link: <https://github.com/oyasizaki/Flutter_Mqtt_Project>

# **Possible Projects can be achieved:**

Mqtt can be used in many projects for example:

* Creating a flutter app to connect with the camera and displaying the video
* Cratering a Chatting application like WhatsApp.
* Creating an application to add with Arduino.

Flutter can be used to make shopping applications. For an example, please visit the following link:

* <https://github.com/oyasizaki/Flutter_Shopping_Cart>

For Flutter basic Components and its uses please refer to the following link:

* <https://github.com/oyasizaki/Flutter>

# **Reference**

1. <https://www.educative.io/answers/what-are-variables-in-dart-programming>
2. <https://www.educative.io/answers/what-are-the-data-types-in-dart>
3. <https://www.javatpoint.com/dart-function#:~:text=Dart%20function%20is%20a%20set,and%20enhances%20the%20code%20reusability>.
4. <https://www.javatpoint.com/dart-main-function>
5. <https://www.bezkoder.com/dart-flutter-constructors/>
6. <https://www.educative.io/answers/what-is-string-interpolation-in-dart>
7. <https://dev.to/newtonmunene_yg/dart-getters-and-setters-1c8f>
8. <https://www.woolha.com/tutorials/dart-using-access-modifiers-private-public>
9. <https://www.bezkoder.com/dart-list/>
10. <https://www.javatpoint.com/dart-map>
11. <https://www.tutorialspoint.com/dart_programming/dart_programming_if_else_statement.htm>
12. <https://www.tutorialkart.com/dart/dart-if-else/>
13. <https://www.section.io/engineering-education/understanding-loops-and-iteration-in-dart/#:~:text=Iteration%20refers%20to%20the%20process,loop%20can%20undergo%20many%20iterations>.
14. <https://www.educative.io/answers/what-is-dart-cascade-notation>
15. <https://www.tutorialspoint.com/cascade-notation-in-dart-programming>
16. <https://www.javatpoint.com/dart-inheritance>
17. <https://docs.flutter.dev/testing/debugging>
18. <https://www.javatpoint.com/dart-method-overriding>
19. <https://www.javatpoint.com/dart-abstract-classes>
20. <https://medium.flutterdevs.com/explore-generics-in-dart-flutter-6dd62b6f3ed4>
21. <https://www.geeksforgeeks.org/flutter-working-with-callback-functions/>
22. <https://www.educative.io/answers/how-to-get-input-from-users-in-dart>
23. <https://codewithandrea.com/videos/dart-null-safety-ultimate-guide-non-nullable-types/>
24. <https://www.javatpoint.com/dart-static-keyword>
25. <https://www.educative.io/answers/darts-async-await-in-flutter>
26. <https://www.educative.io/answers/what-are-dart-futures>
27. <https://subscription.packtpub.com/book/web-development/9781783989560/1/ch01lvl1sec08/modularity-and-a-namespace>