Ex .No : 1 Date :	Study and installation of Flutter/Kotlin multi-platform environment .	Page No : 1

Aim:

The aim of this guide is to help you set up a Flutter multi-platform development environment using Android Studio. This includes installing Flutter, configuring Android Studio, and creating a basic Flutter project that can be run on both Android and iOS platforms.

1. Install Flutter SDK:

- Download the Flutter SDK from the official website: Flutter SDK
- Extract the downloaded zip file to a location on your machine.
- Add the Flutter bin directory to your system PATH. This step is crucial for running Flutter commands from the terminal.

2. Install Dart SDK:

- Flutter requires Dart SDK. Download it from the Dart SDK website: Dart SDK
- Extract the Dart SDK and add its bin directory to your system PATH.

3. Verify Flutter Installation

- Open a terminal and run the following command to verify Flutter is correctly installed:
  $ flutter doctor
- Fix any issues reported by flutter doctor until all checks pass.

4. Install Android Studio:

- Download and install Android Studio from the official website: Android Studio
- Open Android Studio, and install the Flutter and Dart plugins from the marketplace.

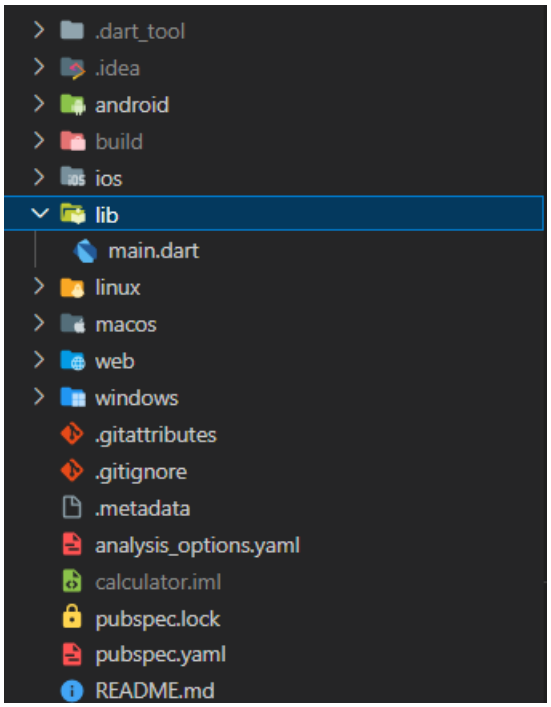
5. Configure Flutter in Android Studio:

- Open Android Studio, go to Preferences on macOS or Settings on Windows/Linux.
- Navigate to Languages & Frameworks > Flutter.
- Set the Flutter SDK path to the location where you extracted the Flutter SDK.

5. Create a Flutter Project:

- Open Android Studio and click on **File > New > New Flutter Project.**
- Choose a Flutter application template.
- Set the **project name, location**, and other details.
- Click Finish to create the project.

Project Structure :



- **android/:** Android-specific code and configurations.
- **build/:** Auto-generated build files.
- **ios/:** iOS-specific code and configurations.
- **lib/:** Dart code for your Flutter application.
  main.dart: The entry point of your Flutter app.
- **test/:** Folder for unit tests.
- **.gitignore:** File to specify files and directories to ignore in version control.
- **.metadata:** Flutter-specific metadata file.
- **.packages:** Flutter package dependencies.
- **.vscode/:** Configuration files for Visual Studio Code (if used).
- **android.iml:** Android Studio project file.
- **pubspec.lock:** Lock file specifying exact versions of dependencies.
- **pubspec.yaml:** YAML file for project configuration, including dependencies.

7. Run on Android Device:

- Connect an Android device or start an emulator.
- Open the terminal in Android Studio and navigate to your project directory.
- Run flutter devices to see the available devices.
- Run flutter run to build and run the Flutter app on the selected device.

8. Run on iOS Simulator (macOS only):

- Open the project in Android Studio.
- Open a terminal and navigate to your project directory.
- Run flutter devices to ensure an iOS simulator is available.
- Run flutter run with the target device set to the iOS simulator.

9. Study Notes:

- Understand the Flutter project structure, especially the lib directory where your Dart code resides.
- Explore the **pubspec.yaml** file for managing dependencies.
- Study Flutter widgets and their properties.
- Learn how to navigate between screens using **Navigator**.
- Understand the concept of **Stateful** and **Stateless** widgets.

Result:

Thus the Installation Of Flutter Multi-Platform Environment was successfully installed and verified

Ex .No : 2 Date :	Develop an application that uses Widgets, GUI components, Fonts, and Colors.	Page No : 4

Aim:

To Develop an application that uses Widgets, GUI components, Fonts, and Colors.

Algorithm :

Widget Tree Structure:

- The program begins with the main function, which calls the runApp method to start the Flutter application.
- The MyApp class is a stateless widget representing the entire application.
- MyApp creates a MaterialApp with a custom theme and sets the home page to an instance of MyHomePage.

Home Page Widget (MyHomePage):

- MyHomePage is a stateful widget that holds the mutable state of the counter.
- It has a corresponding state class _MyHomePageState that extends State<MyHomePage>.

State Class (_MyHomePageState):

- The state class _MyHomePageState contains the mutable state for the counter.
- It includes an integer variable _counter initialized to 0.
- There are two methods, _incrementCounter and _decrementCounter, to handle the increment and decrement operations, respectively.
- The setState method is used in both methods to trigger a rebuild of the UI when the counter changes.

Build Method (build):

- The build method is responsible for creating the widget tree.
- It returns a Scaffold widget, which provides the basic structure of the app, including an AppBar and a body.
- The body contains a Center widget with a Column of child widgets.
- The first child is a text widget displaying the label "Counter" with a specified style.
- The second child is another text widget displaying the current counter value, using a larger font size and a specific color.
- A SizedBox is used to add some spacing between the text and the buttons.
- The third child is a Row containing two ElevatedButton widgets with icons for increment and decrement operations.
- Each button has an onPressed callback linked to _incrementCounter and _decrementCounter methods.

Increment and Decrement Methods:

- incrementCounter and _decrementCounter methods modify the _counter variable using the setState function to trigger a rebuild of the UI.

UI Update:

- When the user taps the increment or decrement buttons, the corresponding _incrementCounter or decrementCounter method is called.

- setState is used to notify Flutter that the internal state has changed, triggering a rebuild of the widget tree.
- The updated counter value is reflected in the UI.

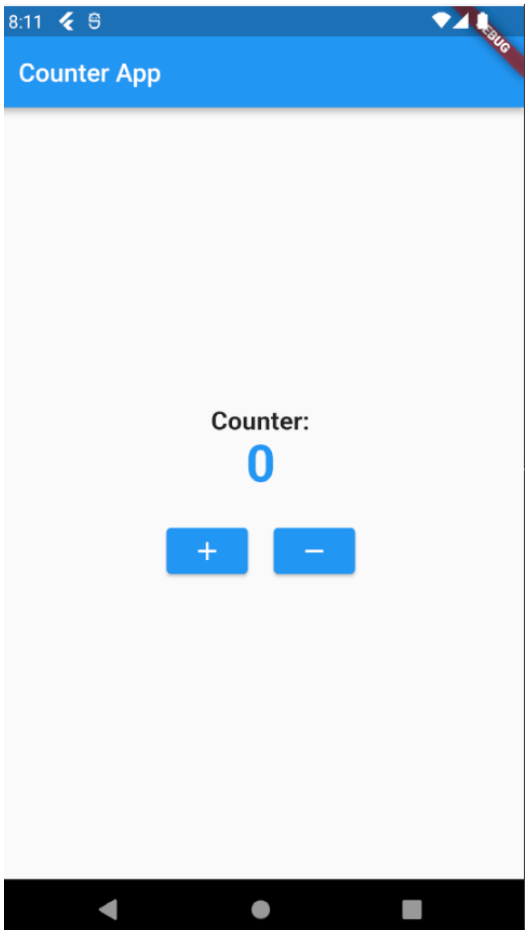
Program :

main.dart

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) { return
    MaterialApp(
      title: 'Flutter Counter App', theme:
      ThemeData(
        primarySwatch: Colors.blue,
        fontFamily: 'Roboto', // Setting a custom font
      ),
      home: MyHomePage(),
    );
  }
}

class MyHomePage extends StatefulWidget { @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> { int
  _counter = 0;

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

  void _decrementCounter() {
    setState(() {
      _counter--;
    });
  }

  @override
  Widget build(BuildContext context) { return
    Scaffold(
      appBar: AppBar(
        title: Text('Counter App'),
      ),
      body: Center( child:
        Column(
          mainAxisAlignment: MainAxisAlignment.center, children:
          <Widget>[
```

```
                Text(
                  'Counter:', style:
                  TextStyle(
                    fontSize: 20.0,
                    fontWeight: FontWeight.bold,
                  ),
                ),
                Text(
                  '$_counter', style:
                  TextStyle(
                    fontSize: 40.0, color:
                    Colors.blue,
                    fontWeight: FontWeight.bold,
                  ),
                ),
                SizedBox(height: 20.0), Row(
                  mainAxisAlignment: MainAxisAlignment.center, children: [
                    ElevatedButton(
                      onPressed: _incrementCounter,
                      child: Icon(Icons.add),
                    ),
                    SizedBox(width: 20.0),
                    ElevatedButton(
                      onPressed: _decrementCounter,
                      child: Icon(Icons.remove),
                    ),
                  ],
                ),
              ],
            ),
          ),
        );
    }
}
```

Output:



Result:

Thus the given program was executed and verified succesfully