

CALCULATOR APP DEVELOPMENT REPORT
at
Sathyabama Institute of Science and Technology
(Deemed to be University)

Submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering
Degree in Computer Science and Engineering specialization with Data Science

By

SANNE HRUTHIKA SHETTY (REGISTER NO: 40733025)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with Grade "A" by NAAC
JEPPIAAR NAGAR, RAJIV GANDHISALAI,
CHENNAI – 600119
NOVEMBER – 2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

(Established under Section 3 of UGC Act, 1956)

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI- 600119

www.sathyabamauniversity.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **SANNE HRUTHIKA SHETTY (40733025)** who carried out the project entitled "Calculator App using flutter" under my supervision from Aug 2022 to Oct 2022.

Internal Guide

Ms Nancy Kirupanithi.D

Head of the Department – Dr S Vigneshwari

Submitted for Viva voce Examination held on 12-09-2022

Internal Examiner

External Examiner

DECLARATION

I SANNE HRUTHIKA SHETTY hereby declare that the Project Report entitled_____

Done by me under the guidance Of Ms._____(Internal)and_____(External)

at_____ **Agloxfusion Pvt Ltd** _____(Company name

and address) is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE: 12-09-2022

PLACE: Chennai



SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph.D., Dean**, School of Computing , **Dr.S.Vigneshwari M.E., Ph.D., and Dr.L.Lakshmanan M.E., Ph.D.,** Heads of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide -----
-----for his valuable guidance, suggestions
and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

TRAINING CERTIFICATE



ABSTRACT

The objective of this documentation is to have overall view in Flutter Framework and Dart programming language which can be used to implement a calculator whilst learning the basics of Flutter. It should be functional but not overly complicated to develop.

Flutter SDK is an open-source software development kit for building beautiful UI which is natively compiled. In this we will build a Simple Calculator App that can perform basic arithmetic operations like addition, subtraction, multiplication or division depending upon the user input.

This basic calculator app will combine a simple user interface, a few standard operations and functions, and an output to display the result. So, it makes a great starter application, or template for another basic app.

This work was centered on the Design and implementation of a simple scientific calculator for education organization. The study traced calculator system as a tool to completely change mathematical knowledge and sophisticated problems solving strategies had advanced the field of simulated engine in mathematic. This project work also focused principally on numbers and arithmetic operation. This researcher investigated the manual system in detail with a view to finding out the need to automate the system. Interestingly, the end result of simple calculator system was its ability to process number and operators, and provides a useful result. Therefore, this project will help immensely in the following way. Easy calculating of tedious mathematical problems, easy to retrieval of errors and it will also be of a good assistance to any researcher on these topics.

TABLE OF CONTENTS

Chapter no.	Title	Page no.
1.	Abstract	
	List of fig.	
	INTRODUCTION	12-20
1.1	Calculator	12
1.2	ER Diagram	12-13
1.3	Flutter Basics	14-16
1.4	Flutter Application Architecture	16-20
2.	AIM AND SCOPE OF PRESENT INVESTIGATION	21 - 28
2.1	Aim	21
2.2	Scope	21
2.3	Feasibility Study	23
2.4	Existing System	24
2.5	Proposed System	25
2.6	Basic Operators	27
2.7	Data Preparation	27-28
3.	EXPERIMENTAL OR MATERIAL AND METHODS, WIDGETS AND ALGORITHMS USED	29-35
3.1	Creating a flutter app	29
3.2	Coding calculator app	29-34
3.3	Adding dependencies	35

Chapter no.	Title	Page no.
4.	FORMULA, LOGICS IMPLEMENTATION AND ACCURACY	30 - 41
	4.1 Implementations and packages used	31
5.	5.1 Summary And Conclusion	42

LIST OF FIGURES

Figure no.	Figure name	Page no.
1.2.1	ER Diagram	12
1.3.1	Flutter Basics	14
1.4.1	About the Platform	15
1.4.2	Dart Overview	15-16
1.5.1	Widgets	16-17
1.5.2	Layers	18
1.5.3	Gestures	19
1.5.4	Concept Of State	19
1.5.5	Common Widgets	20-21
2.3.1	Technical Feasibility	23
2.3.2	Economic Feasibility	23
2.3.3	Enviroment Feasibility	23
2.3.4	Feasibility In Operation	23
2.4.1	Disadvantages of Existing System	25

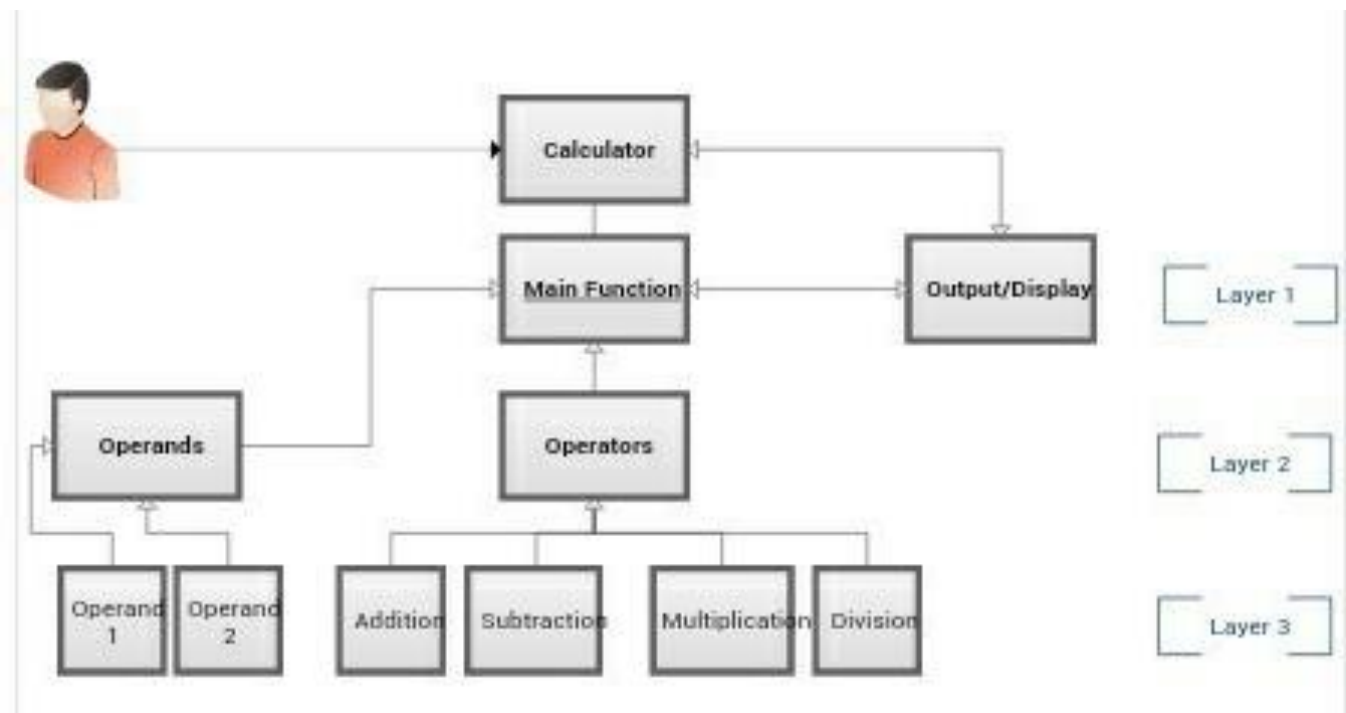
Chapter 1

Introduction

1.1 Calculator

A calculator is a small hand-held computer that performs mathematical calculations. Some calculators even permit simple text editing and programming. It is also a program on a computer that simulates a hand-held calculator. Calculator programs let you perform simple math calculations without leaving the computer. Basic calculators can do only addition, subtraction, multiplication and division mathematical calculations. However, more sophisticated calculators can handle exponential operations, square roots, logarithms, trigonometric functions and hyperbolic functions.

1.2 ER DIAGRAM



We'll look at how to design and build a simple calculator app with Dart and Flutter. The excellent features available in the Dart language and Flutter SDK provide the modern developer with great a great platform for quickly trying new ideas and building high performance apps.

This basic calculator app will combine a simple user interface, a few standard operations and functions, and an output to display the result. So it makes a great starter application, or template for another basic app.

The project for this app was initialized using `flutter create`, with no additional packages or dependencies. All features are implemented with plain Dart and Flutter in a clean and robust way.

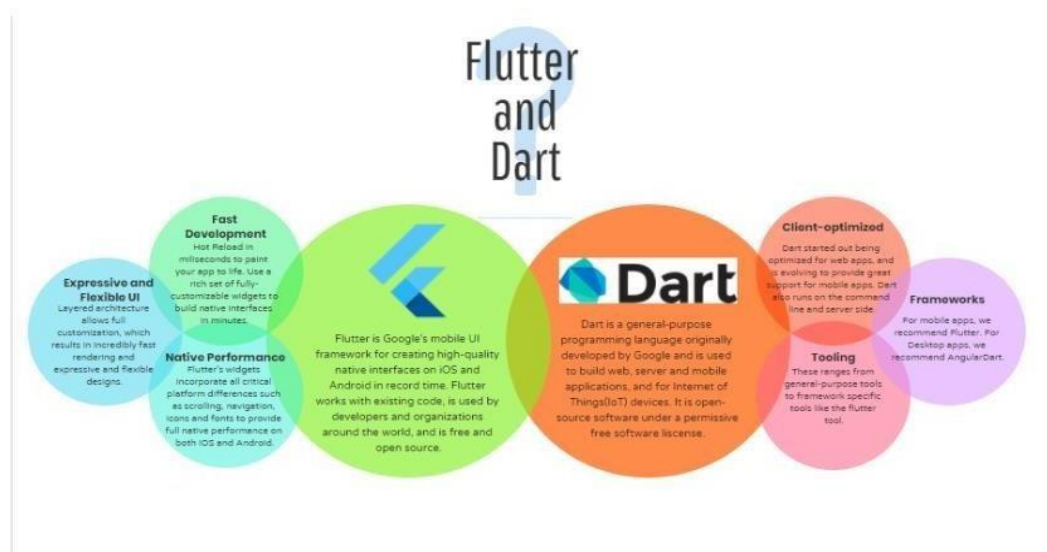
Leveraging built-in language and SDK features allow the app architecture to be kept as lean and efficient as possible. This translates into quicker project turnaround, faster app binaries, and less technical debt.

To build this project, or create one like it, the following tools are required:

- Dart
- Flutter SDK

1.3 Flutter Basics

Flutter is Google's Mobile SDK to build native iOS and Android, Desktop (Windows, Linux, macOS), and Web apps from a single codebase. When building applications with Flutter everything towards Widgets – the blocks with which the flutter apps are built. They are structural elements that ship with a bunch of material design-specific functionalities and new widgets can be composed out of existing ones too. The process of composing widgets together is called composition. The User Interface of the app is composed of many simple widgets, each of them handling one particular job. That is the reason why Flutter developers tend to think of their flutter app as a tree of widgets.



The biggest selling point of Flutter Tech are two things:

1. **High-Performance App:** The Apps developed using Flutter are highly expressive and have flexible UI. Its fast development due to hot reloading brings the app to life and its expressiveness provides features that are keened for native end-user experiences.
2. **Expressive and Flexible UI:** Flutter lets developers build beautiful-looking apps with ease by using prebuild material widgets. Even though many widgets are prebuilt still flutter enables full customization of the widget.
3. **Fast Development & Hot Reloading:** Hot Reloading refers to the injection of new versions of the files that you edited at runtime while keeping the app running.

About the platform:

Flutter is very new, but a promising platform, that has attracted the attention of large companies who've released their apps already. It is interesting because of its simplicity compared to developing web applications, and because of its speed as compared with native applications.

High performance and productivity in Flutter are achieved by using several techniques:

- Unlike many other popular mobile platforms, Flutter **doesn't use JavaScript in any way**. Dart is the programming language. It compiles to binary code, and that's why it runs with the native performance of Objective-C, Swift, Java or Kotlin.
- Flutter **doesn't use native UI components** because components are implemented in Flutter itself, there is no communication layer between the view and your code. Due to this, games hit the best speed for their graphics out of the smartphones.
- Flutter **uses a declarative approach** to build its UI based on widgets. To get more out of widgets, they are **rendered only when necessary**, usually when their state has been changed.
- The framework has **integrated Hot-Reload**. This allows the Flutter framework to automatically rebuild the widget tree, allowing you to quickly view the effects of your changes.

Dart Overview

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 for app frameworks.

Dart also forms the foundation of [Flutter](#). Dart provides the language and runtimes that power Flutter apps, but Dart also supports many core developer tasks like formatting, analyzing, and testing code.

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. 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, 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.

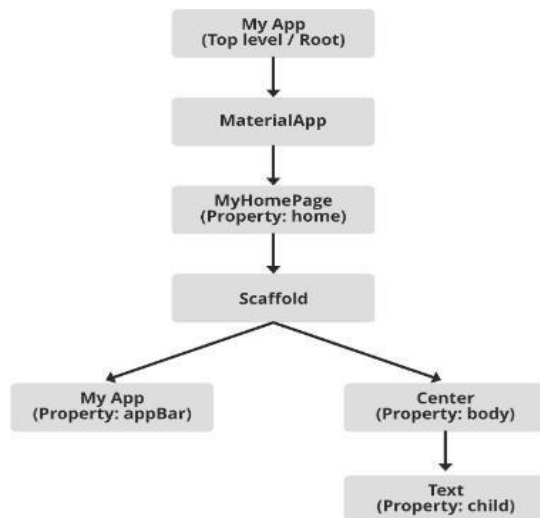
1.4 Flutter Application Architecture

Flutter architecture application mainly consists of:

- Widgets
- Gestures
- Concept of State
- Layers

Widgets

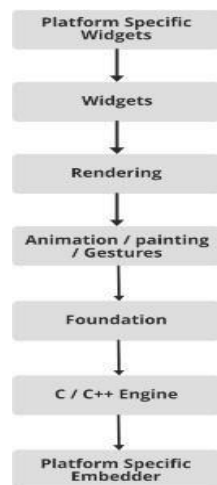
Widgets are the primary component of any flutter application. It acts as a UI for the user to interact with the application. Any flutter application is itself a widget that is made up of a combination of widgets. In a standard application, the root defines the structure of the application followed by a MaterialApp widget which basically holds its internal components in place. This is where the properties of the UI and the application itself is set. The MaterialApp has a Scaffold widget that consists of the visible components (widgets) of the application. The Scaffold has two primary properties namely the body and the appBar. It holds all the child widgets and this is where all its properties are defined.



Inside Scaffold, there is usually an AppBar widget, which as the name suggests defines the appBar of the application. The scaffold also has a body where all the component widgets are placed. This is where these widget's properties are set. All these widgets combined form the Homepage of the application itself. The Center widget has a property, Child, which refers to the actual content and it is built using the Text widget.

Layers

The Flutter framework is categorized based on its complexity and establishes a hierarchy based on the decreasing level of these complexities. These categories are often called Layers. These layers are built on top of one another. The topmost layer is a widget specific to the operating system of the device (ie, Android or iOS). The second layer consists of the native flutter widgets, which comprise structural UI components, gesture detectors, state management components, etc. This third layer is where all the Ui and state rendering occurs. It is the layer that includes all the visible components of the flutter application. The following layer consists of animations used in transitions, image flow, and gestures. These further go on to the very high level of system design that is not the target of this article. The below diagram gives an overview of the same:



Gestures

All physical form of interaction with a flutter application is done through pre-defined gestures. GestureDetector are used for the same. It is an invisible widget that is used to process physical interaction with the flutter application. The interaction includes gestures like tapping, dragging, and swiping, etc. These features can be used to creatively enhance the user experiences of the app by making it perform desired actions based on simple gestures.

Concept of State

If you have ever worked with React js, you might be familiar with the concept of a state. The states are nothing but data objects. Flutter also operates on similar turf. For the management of state in a Flutter application, StatefulWidget is used. Similar to the concept of state in React js, the re-rendering of widgets specific to the state occurs whenever the state changes. This also avoids the re-rendering of the entire application, every time the state of a widget changes.

The architecture of a Flutter app or the flutter framework generally consists of a combination of small and larger widgets interacting in conjuncture to build the application. All of its layers are integral to its design and functioning. As simple as building an application in flutter is, it is built with equally complex components at its core.

Common Widgets

Text()

Flutter cannot render text without knowing what's the preference for text direction. To render text, we have to set **Text.textDirection**.

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

main() => runApp(
  Center( // The widget that aligns content in the center
    child: Text(
      'Hello, World!',
      textDirection: TextDirection.ltr,
    ),
  ),
);
```

Centre()

Center(...) is the widget that aligns another widget given in **child** property in the center of itself. You'll often see **child** and **children** properties in Flutter applications, because almost all widgets are using them if they need one or several widgets to be rendered inside of them.

A composition of widgets in Flutter is used to represent an interface, to change its look, and to share data.

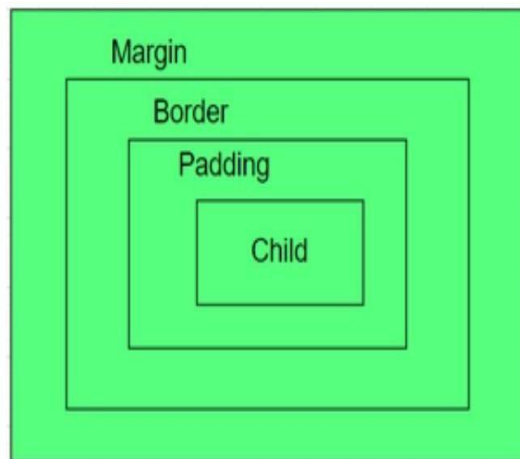
```
import 'package:flutter/widgets.dart';

main() => runApp(
  Directionality(
    textDirection: TextDirection.ltr,
    child: Center(
      child: Text('Hello, World!'),
    ),
  ),
);
```

Container()

Container class in flutter is a convenience widget that combines common painting, positioning, and sizing of widgets. A Container class can be used to store one or more widgets and position them on the screen according to our

convenience. Basically, a container is like a box to store contents. A basic container element that stores a widget has a **margin**, which separates the present container from other contents. The total container can be given a **border** of different shapes, for example, rounded rectangles, etc. A container surrounds its child with **padding** and then applies additional constraints to the padded extent (incorporating the width and height as constraints, if either is non-null).



Scaffold

Scaffold is a class in **flutter** which provides many widgets or we can say APIs like Drawer, SnackBar, BottomNavigationBar, FloatingActionButton, AppBar, etc. **Scaffold** will expand or occupy the whole device screen. It will occupy the available space. Scaffold will provide a framework to implement the basic material design layout of the application.

The class Hierarchy is as follows:

```
Object
└─ Diagnosticable
   └─ Diagnosticable Tree
      └─ Widget
         └─ Stateful Widget
            └─ Scaffold
```

Chapter 2:

AIM AND SCOPE OF PRESENT INVESTIGATION

2.1 AIM:

To calculate Calculator using the features and using the data of a user.

2.2 SCOPE

The idea is to implement a calculator whilst learning the basics of Flutter. It should be functional but not overly complicated to develop. For that reason, we delimit the feature set by the following:

- It should have the possibility to let the user perform all basic calculations: add, subtract, multiply, divide
- We do not work with float values, thus division only includes integers
- A clean button resets the state and enables the user to define a new calculation
- A bar at the top shows the current calculation with the first operand, the operator and the second operand
- After a tap on the equals sign the user is presented the result of the calculation



2.3A FEASIBILITY STUDY

The proposed system should be focus on three primary areas:

1. Technical Feasibility
2. Economic Feasibility
3. Environmental Feasibility
4. Feasibility in Operation

Technical Feasibility:

This research shows all of the technical details as well as the outcomes.

Because of the below-mentioned feature, the project "Calculator" is theoretically viable. The project was created in **Android Studio** utilising the **Flutter Framework and Dart programming language**, with a graphical user interface.

Economic Feasibility:

The organisation must determine its overall financial status based on the estimates acquired in the previous portions of the study. There are financial benefits to the project as a result of better information reliability and accuracy. The system has little hardware requirements, which reduces the cost of hardware acquisition.

Environmental Feasibility:

The method is simpler to use and does not necessitate any specific training. Any special events are alerted to the user through appropriate notifications and messages. The user will have little trouble adapting to the system. The system preserves exceptionally accurate records, and information may be accessible in a matter of seconds.

2.4 EXISTING SYSTEM:

The existing system is time-consuming and complex to calculate. Calculators are small electrical devices that can perform basic and advanced calculations in a fraction of a second. The concept of a unit is created using a Franchise, which was first used in 2000 BC; after a number of inventions and mechanical counting machines were developed. The scientific calculators were created to assist in performing scientific calculations. However, in the 21st century, when people started using personal computers, tablets, mobile phones and other electronic devices, so what to wear such as calculators?

2.4.1 DISADVANTAGES:

The existing system has the following disadvantages: It

- ☐ takes a long time.
- ☐ A great deal of time and effort is squandered.
- ☐ There is a waste of pages.
- ☐ It's difficult to keep outdated records up to date.
- ☐ Queries are difficult to implement.

2.5 PROPOSED SYSTEM

The Calculator Application is a software programme that eliminates the need for more manual hours to calculate and does basic operations with a single click. This application incorporates both American and Indian standards.

This application provides all of the information in both standards that is not available in any other application. BMI is calculated the same way for both adults and children. The calculation is based on the following formulas.



2.6 BASIC OPERATORS:

Addition +

Subtraction -

Multiplication *

Division /

Percentage %

2.7 DATA PREPARATION:

In this project as a backbone tool Dart is used to implement UI of the app and also the logical part of the application. With the help software Android Studio, a FlutterFramework is launched where Dart Language is used in backend.

The fastest developer tools for building market-leading apps and accelerating performance. With an intelligent code editor, flexible build system, realtime profilers and emulators. Building without limits. Code with confidence. Eliminate tiresome tasks.

User Enters the data using following ui:

To Select Operators:



To Select the numbers is used:

7	8	9
4	5	6
1	2	3
0	.	=

To Select Option:

C	+/-	%	DEL
---	-----	---	-----

CHAPTER-3:

EXPERIMENTAL OR MATERIAL AND METHODS, WIDGETS AND ALGORITHMS USED

Step 1: Creating a Flutter App

Open the Terminal /Command-prompt. Change Directory to your choice and run **flutter create calculatorApp**. Open the calculatorApp in VS Code or Android Studio.

```
D:\dev\Mini Projects\Flutter-Mini-Projects> flutter create calculatorApp
Creating project calculatorApp...
calculatorApp\.gitignore (created)
```

Step 2: Coding Calculator App

In the Lib folder, there is a main.dart file already present. And now in the same folder create a new file named **buttons.dart**. Starting with **main.dart** file. Create **MyApp** class and make it **StatelessWidget**. Add an array of buttons to be displayed. Set the background-color, text-color, functionality **onTapped** to the buttons. Write a function to calculate the Answers

- Dart

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

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

```
class MyApp extends StatelessWidget {
```

```

@override
Widget build(BuildContext context) {
  return MaterialApp(
    debugShowCheckedModeBanner: false,
    home: HomePage(),
  ); // MaterialApp
}
}

class HomePage extends StatefulWidget {
  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  var userInput = "";
  var answer = "";

  // Array of button
  final List<String> buttons = [
    'C',
    '+/-' ,
    '%',
    'DEL',
    '7',
    '8',
    '9',
    '/',
    '4',
    '5',
    '6',
    'x',
    '1',
    '2',
    '3',
    '-',
    '0',
    '.',
    '=',
    '+',
  ];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: new AppBar(
        title: new Text("Calculator"),
      ), //AppBar
      backgroundColor: Colors.white38,
      body: Column(

```

```

children: <Widget>[
  Expanded(
    child: Container(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: <Widget>[
          Container(
            padding: EdgeInsets.all(20),
            alignment: Alignment.centerRight,
            child: Text(
              userInput,
              style: TextStyle(fontSize: 18, color: Colors.white),
            ),
          ),
          Container(
            padding: EdgeInsets.all(15),
            alignment: Alignment.centerRight,
            child: Text(
              answer,
              style: TextStyle(
                fontSize: 30,
                color: Colors.white,
                fontWeight: FontWeight.bold),
            ),
          )
        ],
      ),
    ),
  Expanded(
    flex: 3,
    child: Container(
      child: GridView.builder(
        itemCount: buttons.length,
        gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: 4),
        itemBuilder: (BuildContext context, int index) {
          // Clear Button
          if (index == 0) {
            return MyButton(
              buttontapped: () {
                setState(() {
                  userInput = "";
                  answer = '0';
                });
              },
              buttonText: buttons[index],
              color: Colors.blue[50],
              textColor: Colors.black,
            );
          }
        }
      )
    )
  )
]

```

```

// +/- button
else if (index == 1) {
  return MyButton(
    buttonText: buttons[index],
    color: Colors.blue[50],
    textColor: Colors.black,
  );
}
// % Button
else if (index == 2) {
  return MyButton(
    buttontapped: () {
      setState(() {
        userInput += buttons[index];
      });
    },
    buttonText: buttons[index],
    color: Colors.blue[50],
    textColor: Colors.black,
  );
}
// Delete Button
else if (index == 3) {
  return MyButton(
    buttontapped: () {
      setState(() {
        userInput =
          userInput.substring(0, userInput.length - 1);
      });
    },
    buttonText: buttons[index],
    color: Colors.blue[50],
    textColor: Colors.black,
  );
}
// Equal_to Button
else if (index == 18) {
  return MyButton(
    buttontapped: () {
      setState(() {
        equalPressed();
      });
    },
    buttonText: buttons[index],
    color: Colors.orange[700],
    textColor: Colors.white,
  );
}

```

```

        // other buttons
    else {
        return MyButton(
            buttontapped: () {
                setState(() {
                    userInput += buttons[index];
                });
            },
            buttonText: buttons[index],
            color: isOperator(buttons[index])
                ? Colors.blueAccent
                : Colors.white,
            textColor: isOperator(buttons[index])
                ? Colors.white
                : Colors.black,
        );
    }
  }}, // GridView.builder
),
),
],
),
);
}

bool isOperator(String x) {
  if (x == '/' || x == 'x' || x == '-' || x == '+' || x == '=') {
    return true;
  }
  return false;
}

// function to calculate the input operation
void equalPressed() {
  String finaluserinput = userInput;
  finaluserinput = userInput.replaceAll('x', '*');

  Parser p = Parser();
  Expression exp = p.parse(finaluserinput);
  ContextModel cm = ContextModel();
  double eval = exp.evaluate(EvaluationType.REAL, cm);
  answer = eval.toString();
}
}

```

In Flutter `main.dart` file is the entry point from which the code starts to executing. In the `main.dart` file firstly material design package has been imported in addition to `math_expressions` and `buttons.dart` file. Then a function `runApp` has been created with parameter as `MyApp`. After the declaration of class `MyApp` which is a stateless widget, the state of class `MyApp` has been laid out.

Step 3: Building the buttons.dart

In `buttons.dart` which is already imported in `main.dart` file we are declaring variables that will be used throughout the program using a constructor. The color, text color, button text, and the functionality of the button on tapped will be implemented in `main.dart` file

- Dart

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

```
// creating StatelessWidget for buttons
```

```
class MyButton extends StatelessWidget {
```

```
  // declaring variables
```

```
  final color;
```

```
  final textColor;
```

```
  final String buttonText;
```

```
  final buttontapped;
```

```
  //Constructor
```

```
  MyButton(this.color, this.textColor, this.buttonText, this.buttontapped);
```

```
  @override
```

```
  Widget build(BuildContext context) {
```

```
    return GestureDetector(
```

```
      onTap: buttontapped,
```

```
      child: Padding(
```

```
        padding: const EdgeInsets.all(0.2),
```

```
        child: ClipRRect(
```

```
          // borderRadius: BorderRadius.circular(25),
```

```
          child: Container(
```

```
            color: color,
```

```
            child: Center(
```

```
              child: Text(
```

```
                buttonText,
```

```
                style: TextStyle(
```

```
                  color: textColor,
```

```
                  fontSize: 25,
```

```
                  fontWeight: FontWeight.bold,
```

```
              ),
```

```
            ),
```

```
          ),
```

```
        ),
```

```
    ),  
  ),  
);  
}  
}
```

Step 4: Adding dependencies in pubspec.yaml file

To make the process easier we are using **math_expressions: ^2.0.0** package which is imported in main.dart file to handle all the calculations and run time error exceptions.

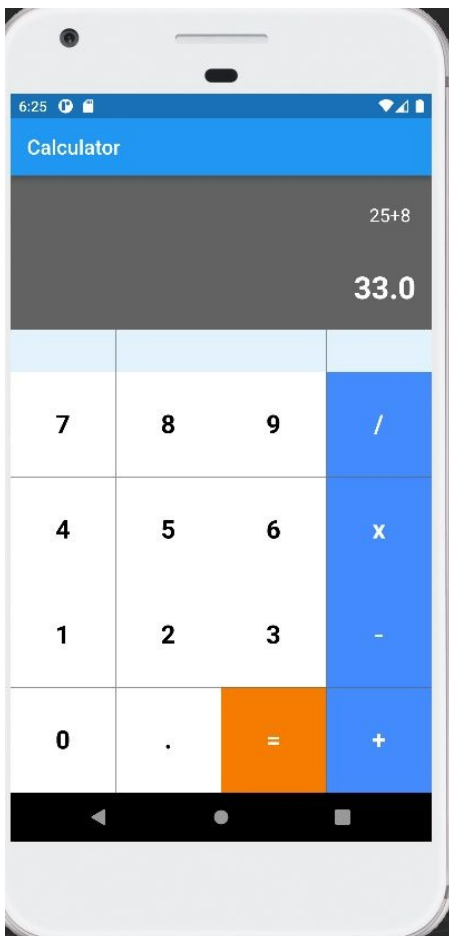
```
23 dependencies:      You, 2 weeks ago • Initial Commit  
24   flutter:  
25     sdk: flutter  
26  
27  
28   # The following adds the Cupertino Icons font to your application.  
29   # Use with the CupertinoIcons class for iOS style icons.  
30   cupertino_icons: ^0.1.3  
31   math_expressions: ^2.0.0  
32  
33 dev_dependencies:  
34   flutter_test:  
35     sdk: flutter  
36
```

Chapter 4:

IMPLEMENTATION

```
Widget calculatorButtonSymbols(String text) {  
  return Expanded(  
    child: MaterialButton(  
      onPressed: () {  
        setState(() {  
          if (text == "÷" || text == "×" || text == "+" || text == "-") {  
            } else if (text == "=") {}  
          });  
        },  
        padding: EdgeInsets.all(24),  
        child: Text(  
          text,  
          style: kCalculatorStyleSymbols,
```

1. ADDITION



2. SUBTRACTION



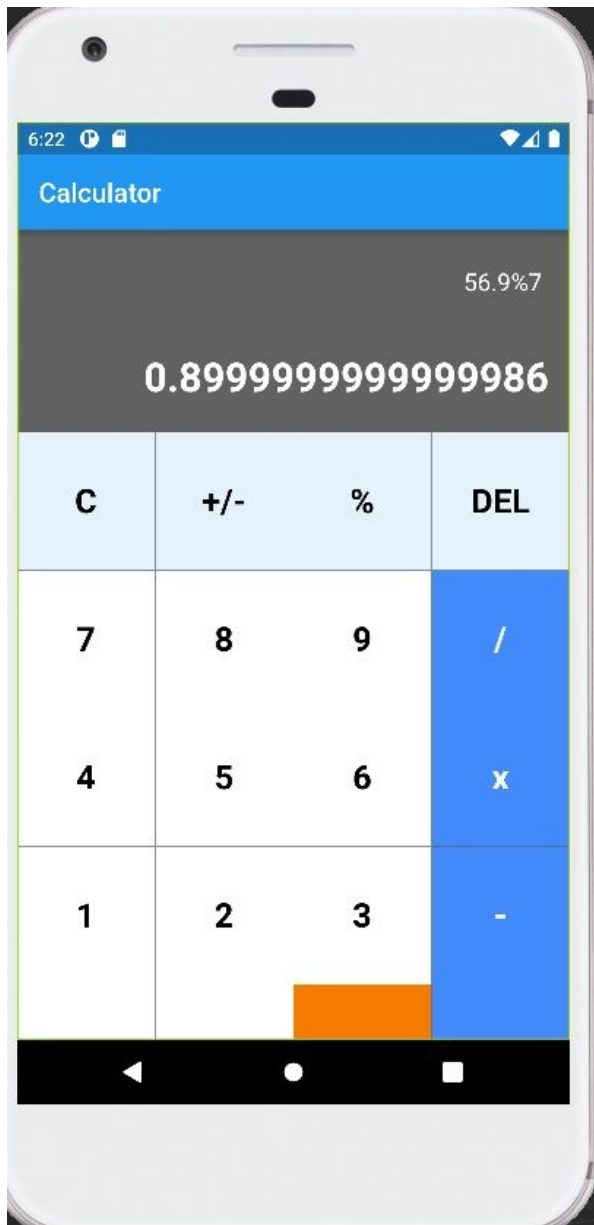
3. MULTIPLICATION



4. DIVISION



5. PERCENTAGE



Chapter 5:

SUMMARY AND CONCLUSION:

- This simple calculator was implemented in only a few hundred lines of code, and illustrates the kind of expressive power available when building mobile apps with Dart and Flutter, which encourage clean design patterns, efficient code reuse, explicit type safety, and other great development habits.
- These features make Flutter a great choice for building high-quality mobile applications that load and run fast, use only as much CPU and RAM as required, don't lag or hang unexpectedly, and can be easily updated and maintained in the future due to clean architecture and code.

Source Code

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

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

class myApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
        primarySwatch: Colors.deepOrange,
      ),
      home: HomePage(),
      debugShowCheckedModeBanner: false,
    );
  }
}

class HomePage extends StatefulWidget {
  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Calculator'),
      ),
      body: Container(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.end,
          children: <Widget>[
            Expanded(
              child: Container(
                padding: EdgeInsets.all(10.0),
```

```

        alignment: Alignment.bottomRight,
        child: Text(
          text,
          style: TextStyle(
            fontSize: 60.0,
            fontWeight: FontWeight.w500,
            color: Colors.deepOrange),
        ),
      ),
    ),
    Row(
      children: <Widget>[
        customOutlineButton("9"),
        customOutlineButton("8"),
        customOutlineButton("7"),
        customOutlineButton("+"),
      ],
    ),
    Row(
      children: <Widget>[
        customOutlineButton("6"),
        customOutlineButton("5"),
        customOutlineButton("4"),
        customOutlineButton("-"),
      ],
    ),
    Row(
      children: <Widget>[
        customOutlineButton("3"),
        customOutlineButton("2"),
        customOutlineButton("1"),
        customOutlineButton("x"),
      ],
    ),
    Row(
      children: <Widget>[
        customOutlineButton("C"),

```

```

        customOutlineButton("0"),
        customOutlineButton("="),
        customOutlineButton("/"),
    ],
),
],
),
),
);
}
Widget customOutlineButton(String val) {
    return Expanded(
        child: OutlineButton(
            padding: EdgeInsets.all(25.0),
            onPressed: () => btnClicked(val),
            child: Text(
                val,
                style: TextStyle(fontSize: 35.0, color: Colors.deepOrange),
            ),
        ),
    );
}
int first, second;
String res, text = "";
String opp;
void btnClicked(String btnText) {
    if (btnText == "C") {
        res = "";
        text = "";
        first = 0;
        second = 0;
    } else if (btnText == "+" ||
        btnText == "-" ||
        btnText == "x" ||
        btnText == "/") {
        first = int.parse(text);
        res = "";
    }
}

```

```

        opp = btnText;
    } else if (btnText == "=") {
        second = int.parse(text);
        if (opp == "+") {
            res = (first + second).toString();
        }
        if (opp == "-") {
            res = (first - second).toString();
        }
        if (opp == "x") {
            res = (first * second).toString();
        }
        if (opp == "/") {
            res = (first ~/ second).toString();
        }
    } else {
        res = int.parse(text + btnText).toString();
    }
    setState(() {
        text = res;
    });
}
}

```