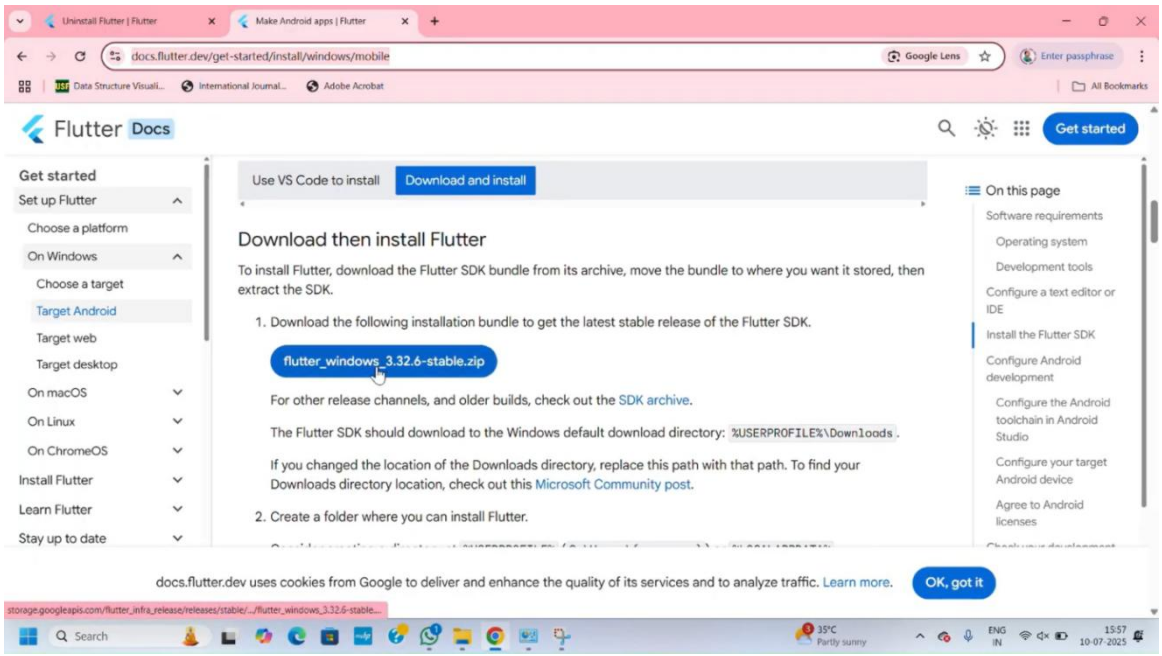
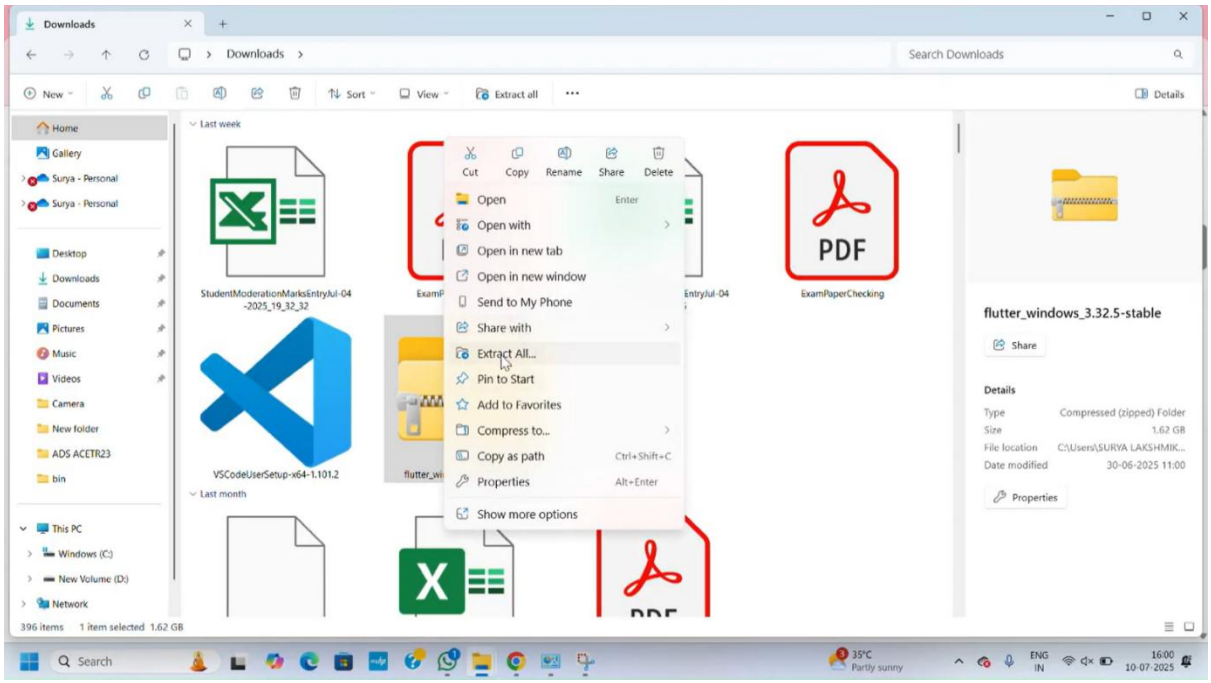
1.) Install Flutter and Dart SDK.

STEP-1: Download flutter SDK for windows OS from

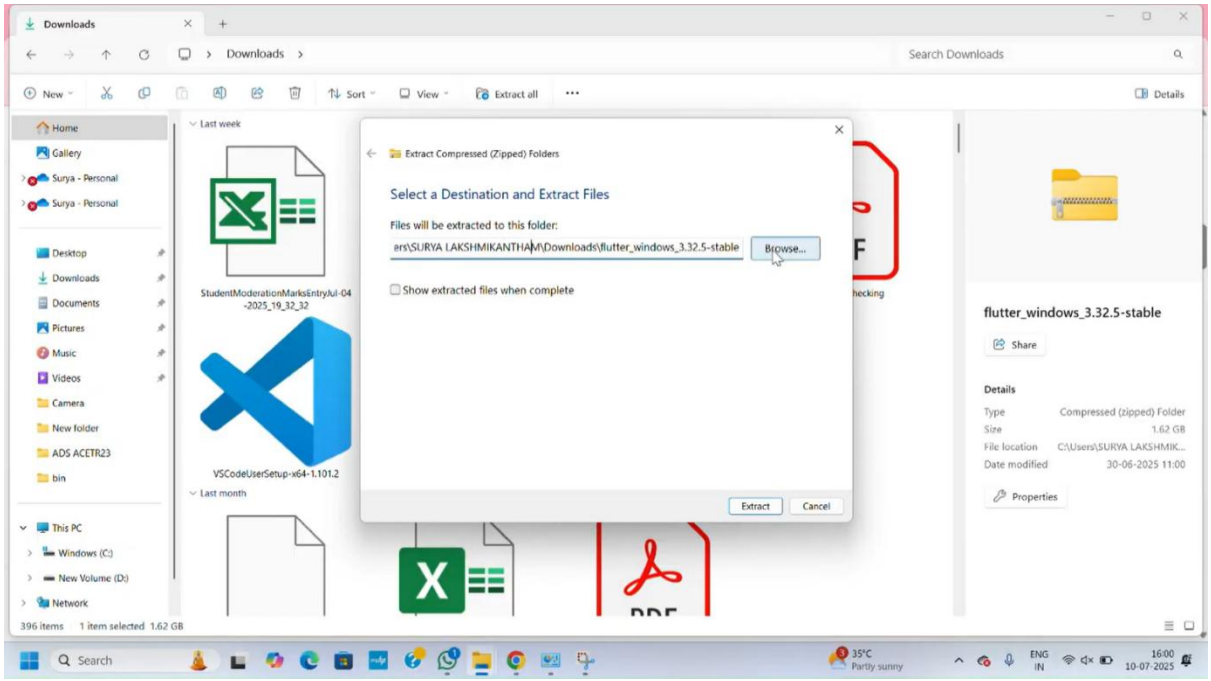
[https://docs.flutter.dev/install/manual] URL.



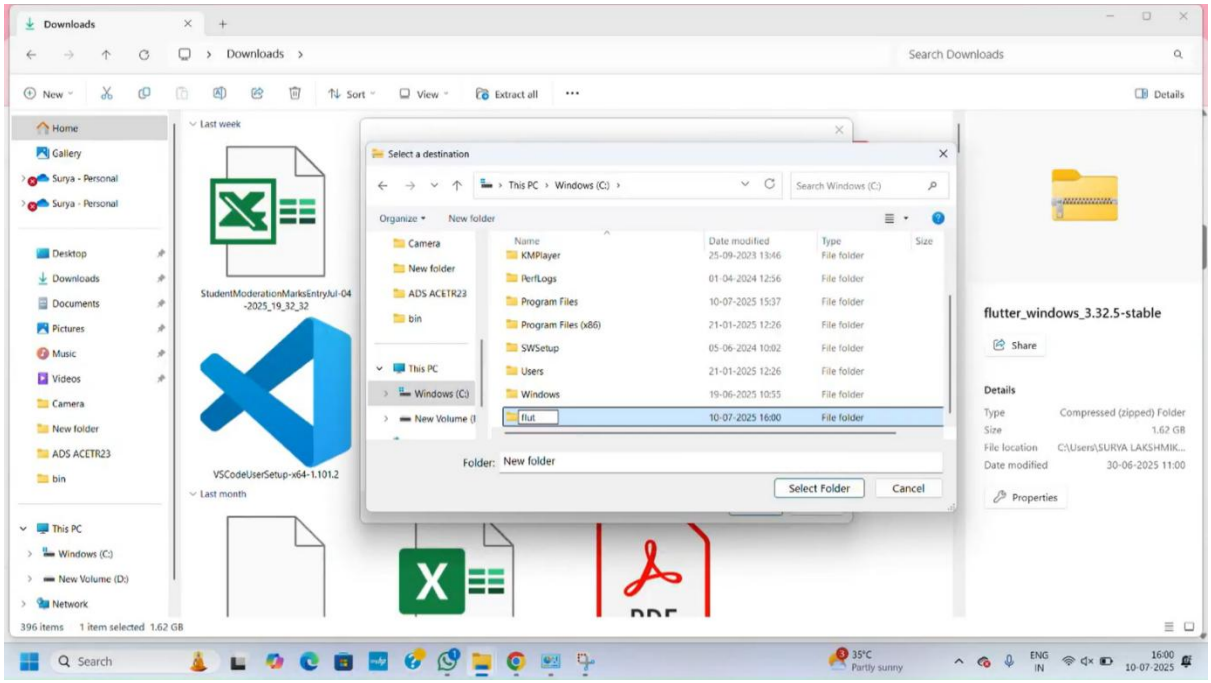
STEP-2: Extraction of flutter SDK into a specific folder.



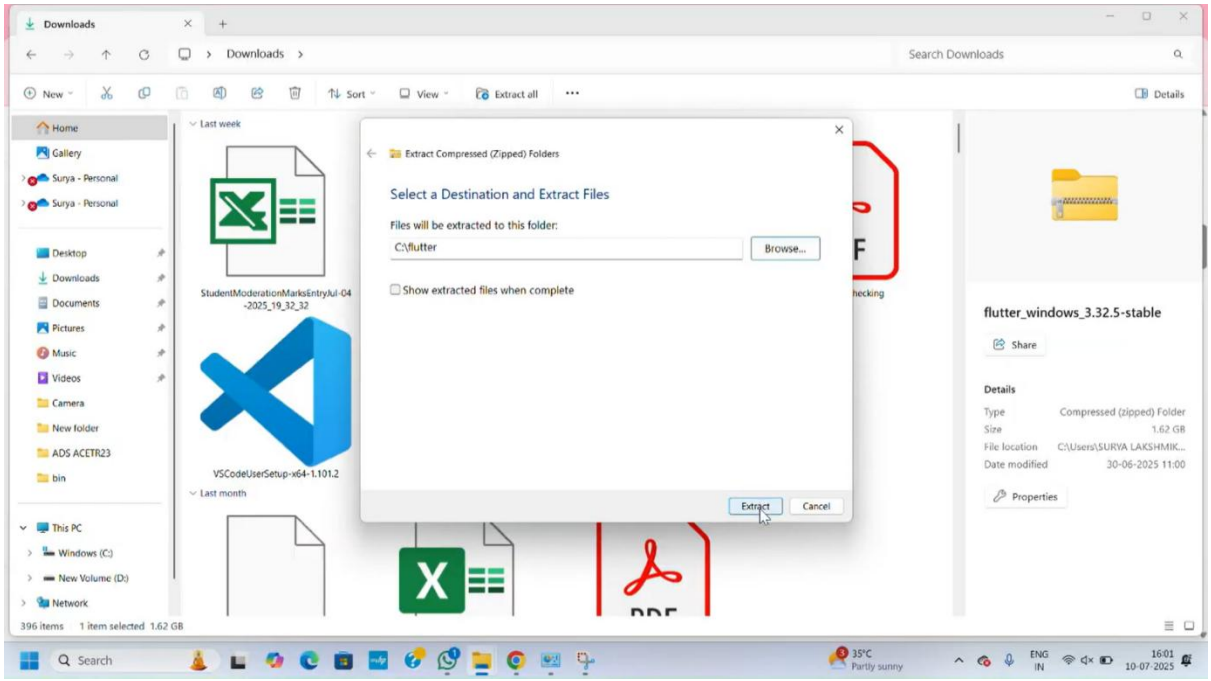
STEP-3: Once it is downloaded, extraction must be done. Click on the Zip file to extract..



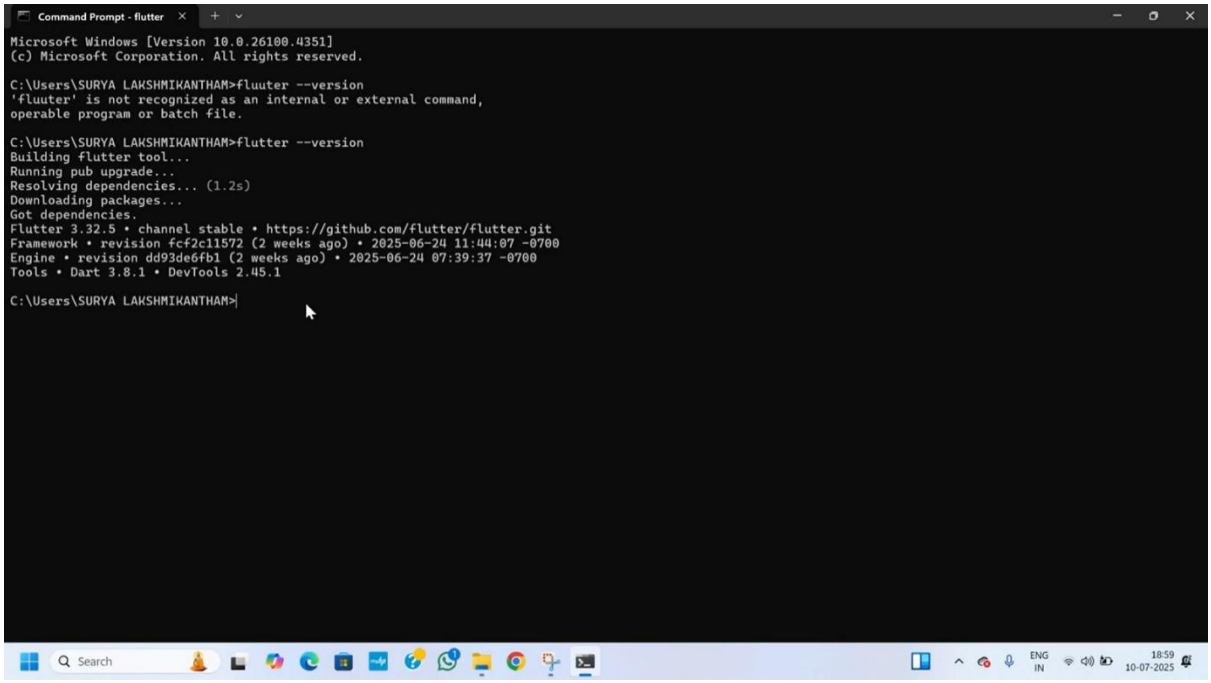
STEP-4: we have to select the folder



STEP-5: Once, the folder is created, extract the contents, by selecting the folder.



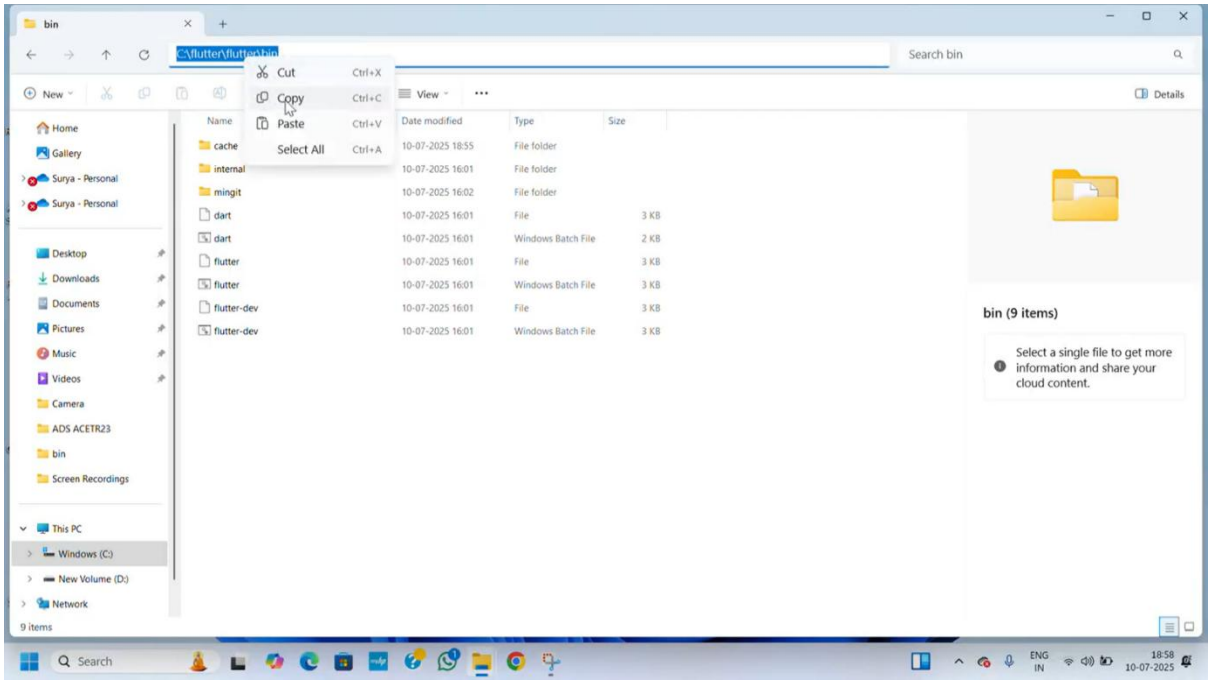
STEP-6: to check whether the flutter SDK is successfully installed, go to command prompt and type **flutter –version** and click enter



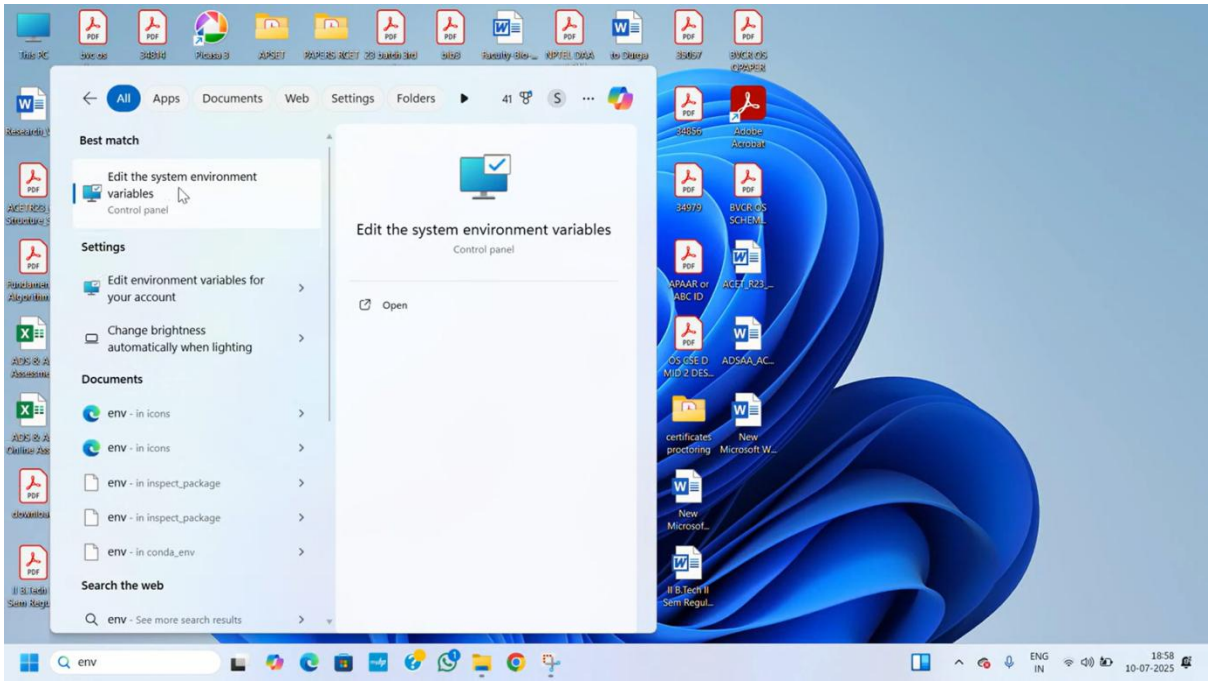
Setting Environmental Variable

STEP-1: To set up the path, go to folder created on c-drive and then move to the flutter and then to the bin.

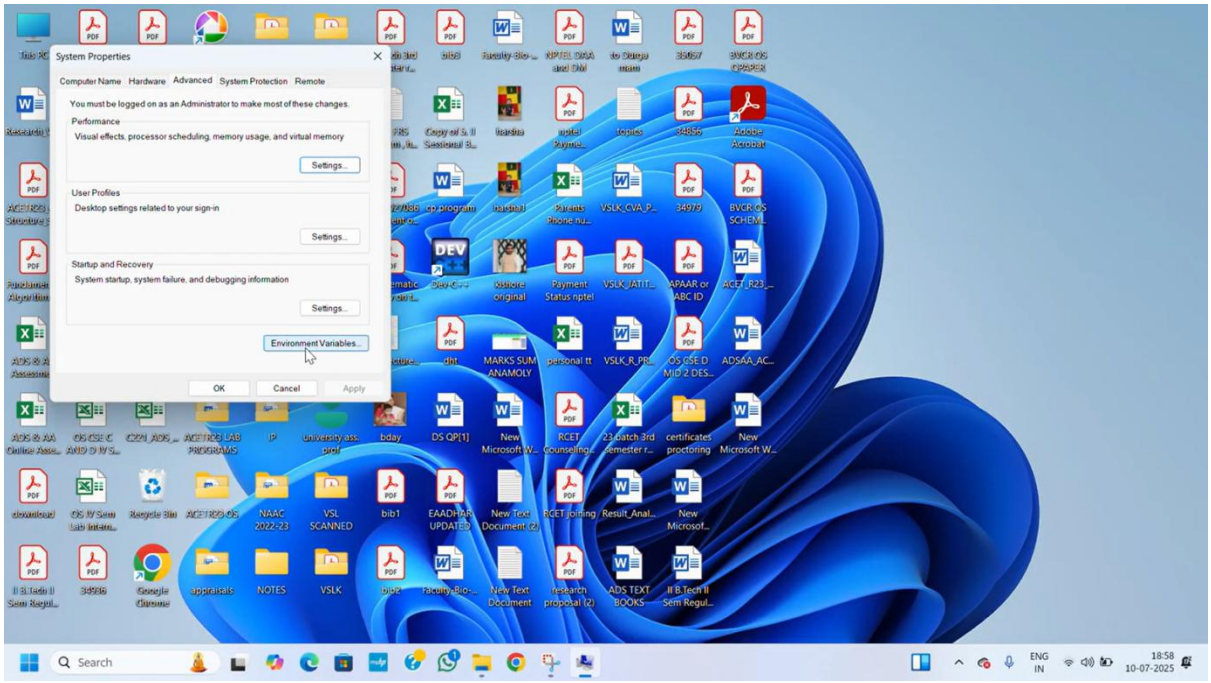
Now copy the path **c:\FLUTTER\flutter\bin**



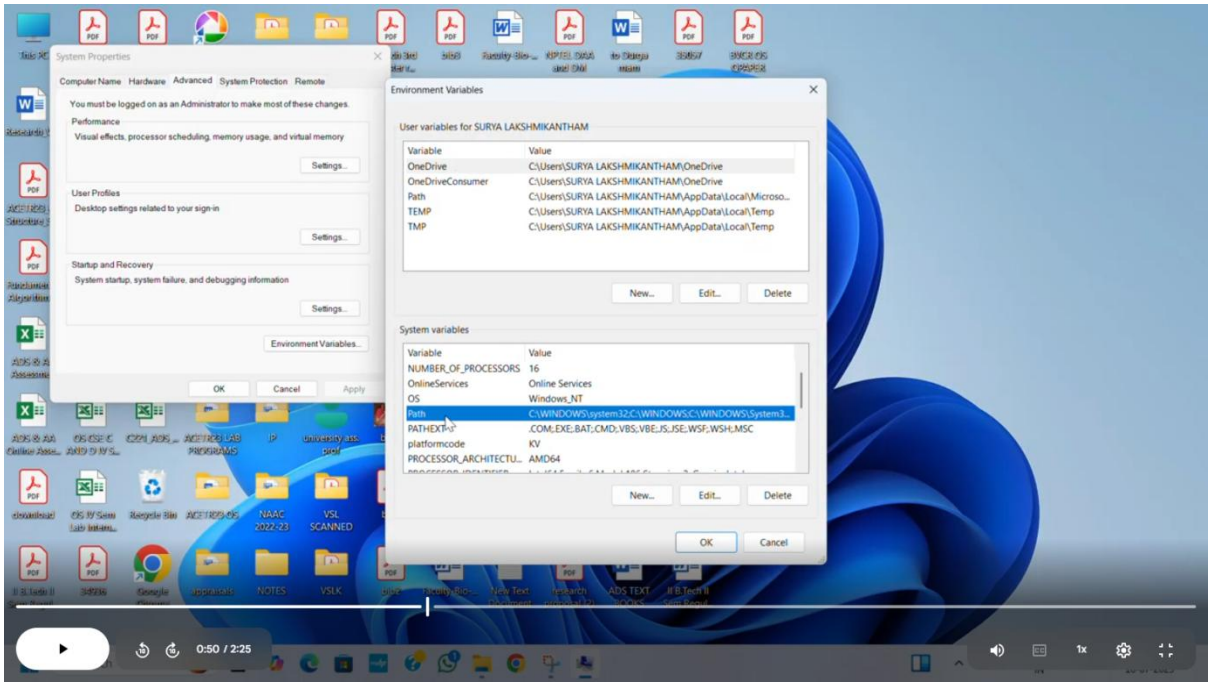
STEP-2: Then go to "Edit the system Environment Variables" control panel



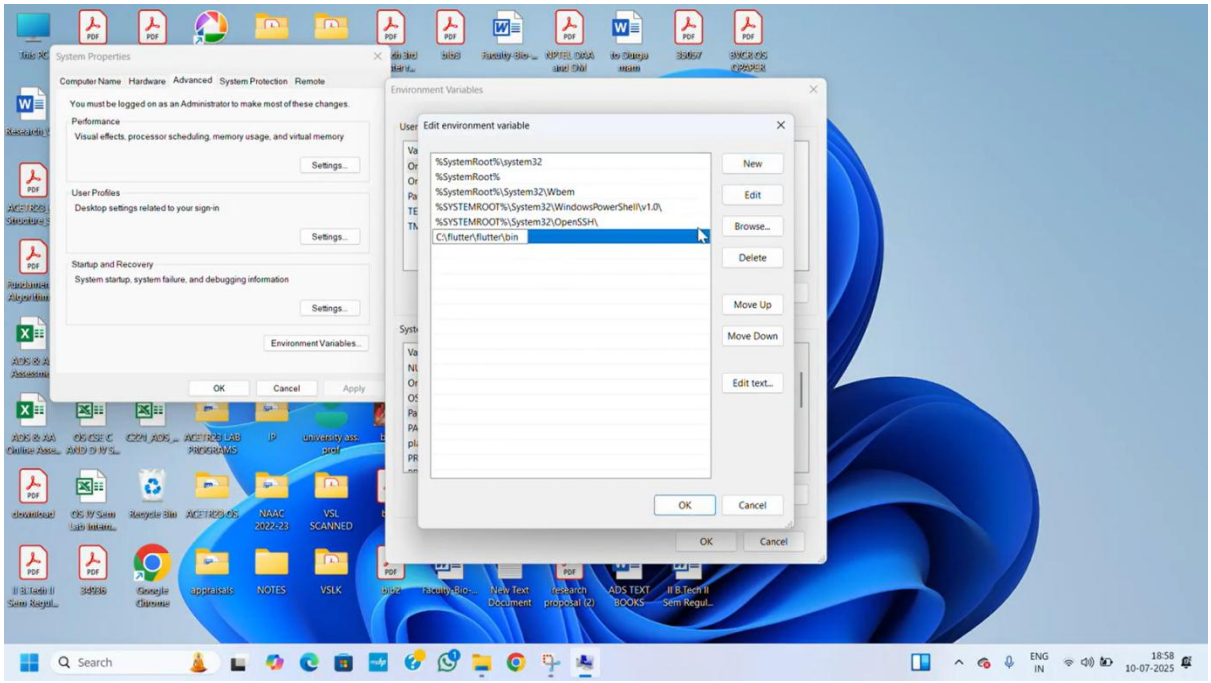
STEP-3: Then click on Environment Variables.



STEP-4: Then move to System variables. Double click on Path and there copy the path link.



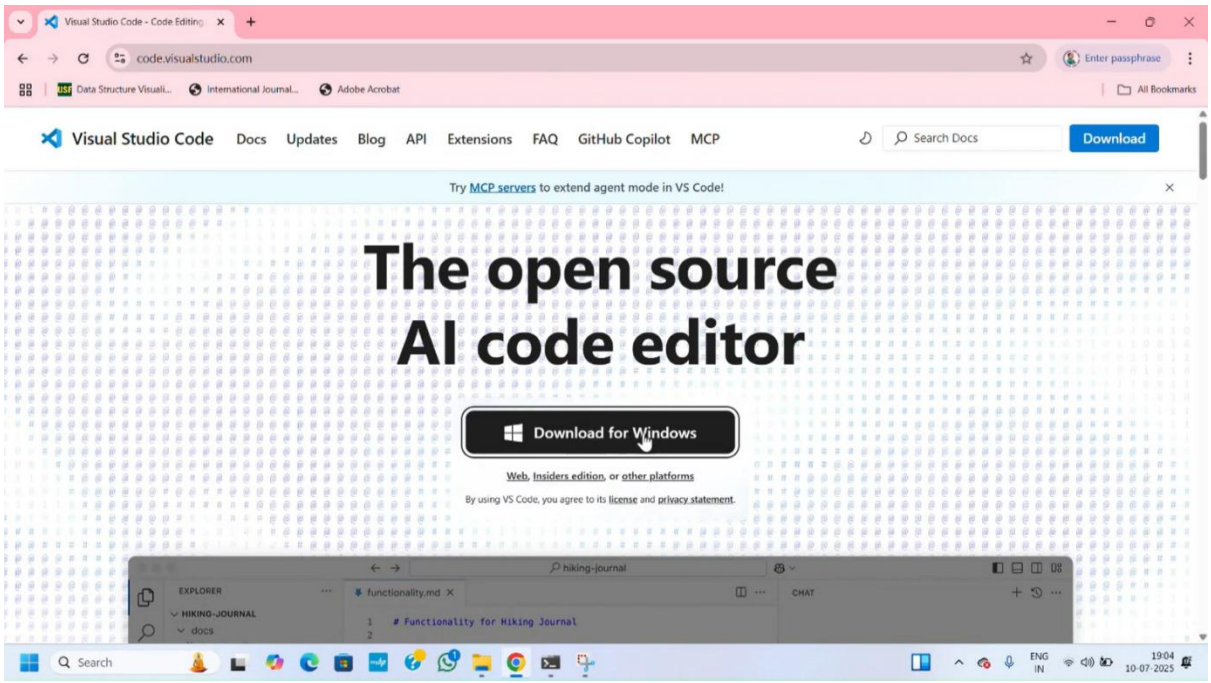
STEP-5: Click on new, then copy the path link.



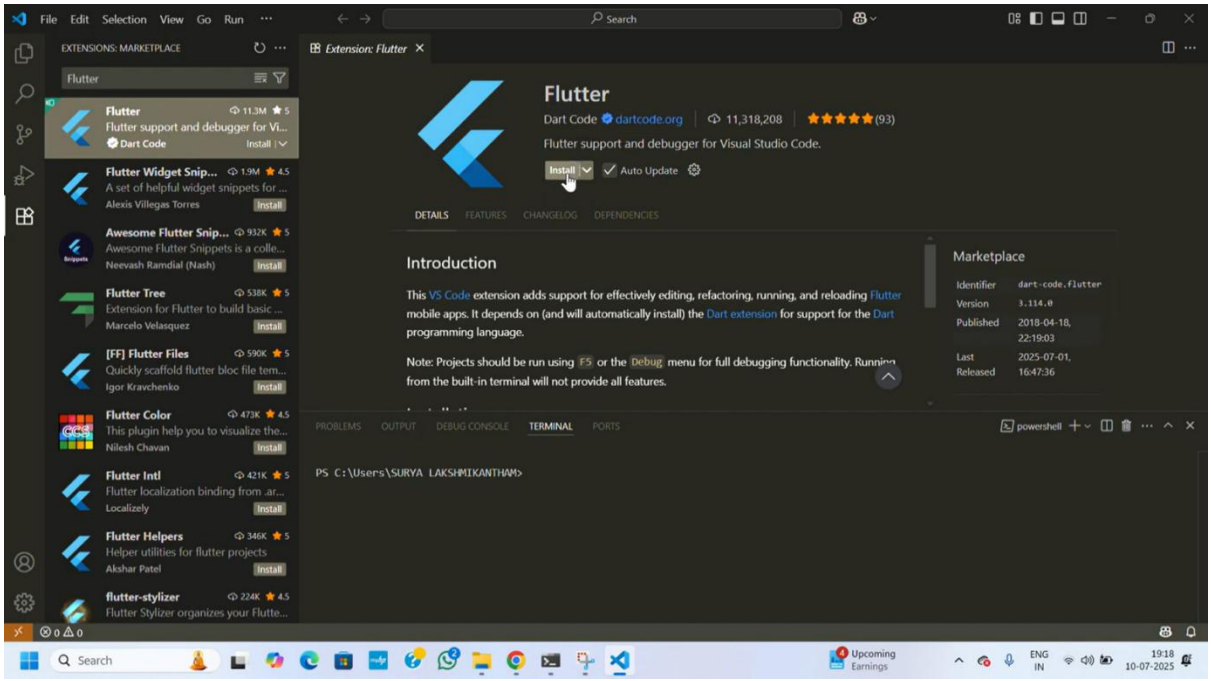
STEP-6: Come back by clicking OK everywhere.

Downloading Flutter and Dart Extensions

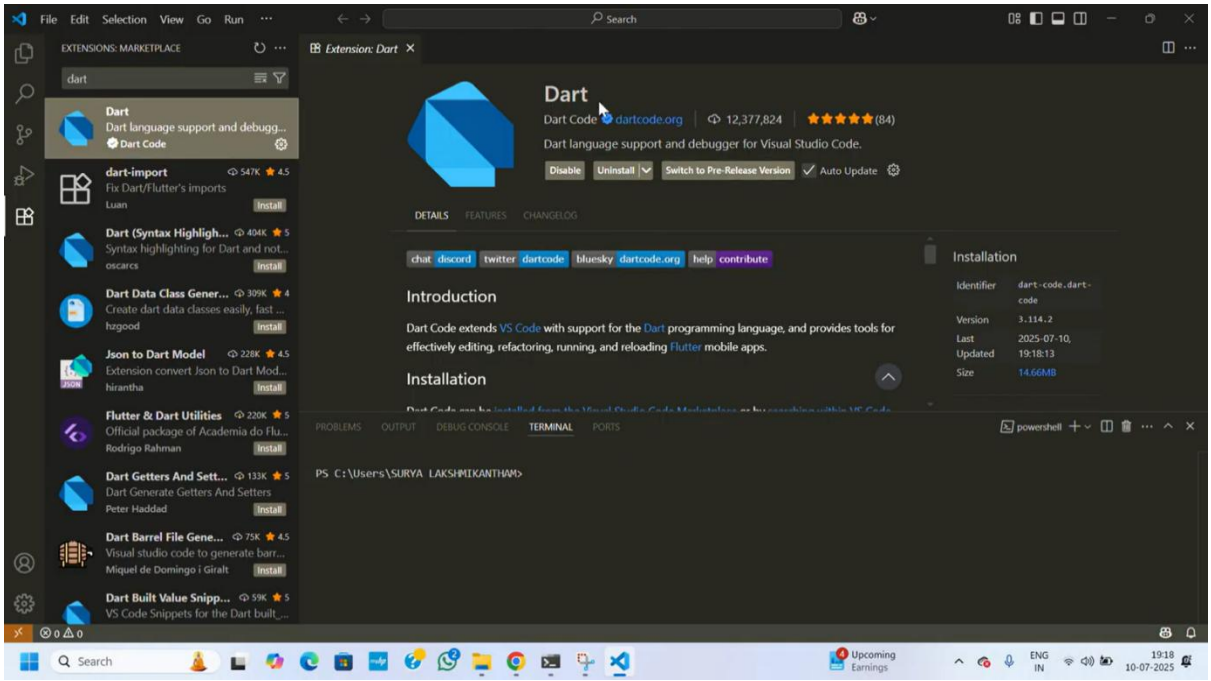
STEP-1: First install the VISUAL STUDIO CODE from user search engine.



STEP-2: Once it is installed, search for the Flutter extension and install it.



STEP-3: Search Dart extension and install it.



1b)

Write a dart program to illustrate basic data types and print function.

Code:

```
void main() {
// Integer variable
 int count = 9;
// Boolean variable
bool flag = true;
// Double (floating point) variable
double marks = 55.78;
// String variable
String name = 'Amar';
// var automatically takes String type
  var hai = "Good Morning";
// Printing using string interpolation
 print("$hai! My name is $name with aggregate $marks");
print("flag is $flag and count is $count");
}
```

OUTPUT:

```
Good Morning! My name is Amar with aggregate 55.78
flag is true and count is 9
```

Write a dart program to read multiple variables of different data types from keyboard and print them.

Code:

```
import 'dart:io';
void main() {
  print('Enter age, height, and name :');
  // Take a line of input, split by spaces, store as List<String>
  List<String> inputs = stdin.readLineSync()!.split(' ');
  // Convert to proper data types
  int age = int.parse(inputs[0]);       // First input → int
  double height = double.parse(inputs[1]);   // Second input → double
  String name = inputs[2];              // Third input → String

  // Print values
  print('Age: $age, Height: $height, Name: $name');
}
```

OUTPUT

```
Enter age, height, and name :
30 180 Ram
Age:30, Height:180.0, Name:Ram
```

Write a dart program on logical operators

code:

```
void main() {
  bool a = true;
  bool b = false;
  // logical AND (&&)
  print('a && b = ${a && b}');
  // logical OR (||)
  print('a || b = ${a || b}');
  // logical NOT (!)
  print('!a = ${!a}');
  print('!b = ${!b}');
  // combining expressions
  int x = 10;
  int y = 20;
  print('(x > 5) && (y > 30) = ${(x > 5) && (y > 30)}');
  print('(x > 5) && (y < 30) = ${(x > 5) && (y < 30)}');
  print('(x < 5) && (y < 30) = ${(x < 5) && (y < 30)}');
  print('(x < 5) || (y > 15) = ${(x < 5) || (y > 15)}');
}
```

OUTPUT:

```
a && b = false
a || b = true
!a = false
!b = true
(x>5) && (y>30) = false
(x>5) && (y<30) = true
(x<5) && (y<30) = false
(x<5) || (y>15) = true
```

Write a Dart program to read number from user, and find whether it is negative or positive. If positive , check whether it is even or odd.

PROGRAM :

```
import 'dart:io';

void main() {
  print("Enter a number:");
  int number = int.parse(stdin.readLineSync()!);
  // Simple if statement
  if (number > 0) {
    print("The number is positive.");
    if (number % 2 == 0) {
      print("The number is even.");
    } else {
      print("The number is odd.");
    }
  } else if (number < 0) {
    print("Negative number");
  } else {
    print("Zero");
  }
}
```

OUTPUT :

```
PS D:\CSE C FLUTTER\dart_application_3\bin> dart .\dart_application_3.dart
Enter a number:
9
The number is positive.
The number is odd.
PS D:\CSE C FLUTTER\dart_application_3\bin> dart .\dart_application_3.dart
Enter a number:
0
Zero
```

WRITE A DART PROGRAM TO SIMULATE SIMPLE CALCULATOR

```
import 'dart:io';

void main() {

/ //Simple Calculator
  print("\n--- Simple Calculator ---");
  print("Enter first number:");
  double num1 = double.parse(stdin.readLineSync()!);

  print("Enter operator (+, -, *, /):");
  String op = stdin.readLineSync()!;

  print("Enter second number:");
  double num2 = double.parse(stdin.readLineSync()!);

  double result;

  switch (op) {
    case '+':
      result = num1 + num2;
      print("Result: $result");
      break;
    case '-':
      result = num1 - num2;
      print("Result: $result");
      break;
    case '*':
      result = num1 * num2;
      print("Result: $result");
      break;
    case '/':
      if (num2 != 0) {
        result = num1 / num2;
        print("Result: $result");
      } else {
        print("Error: Division by zero");
      }
      break;
    default:
      print("Invalid operator");
  }
}
```

OUTPUT :

```
--- Simple Calculator ---
Enter first number:
5
Enter operator (+, -, *, /):
+
Enter second number:
3
Result: 8.0
```

2 a) Explore various Flutter widgets (Text, Image, Container, etc.).

Creating Flutter Widget Text :

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text("This is AppBar"),
          titleTextStyle: TextStyle(
            fontSize: 45,
            fontWeight: FontWeight.w600,
            color: Colors.pink,
          ),
          backgroundColor: Colors.amberAccent,
        ),
      ),
    );
  }
}
```

- runApp(MyApp()): Launches your app.
- MaterialApp: Wraps your app in Material Design.
- Scaffold: Provides basic layout like appBar, body.
- AppBar: Top bar with title.
- Text: Displays a string on screen.
- Center: Centers the text on the screen.

OUTPUT :

Creating Flutter Widget Image:

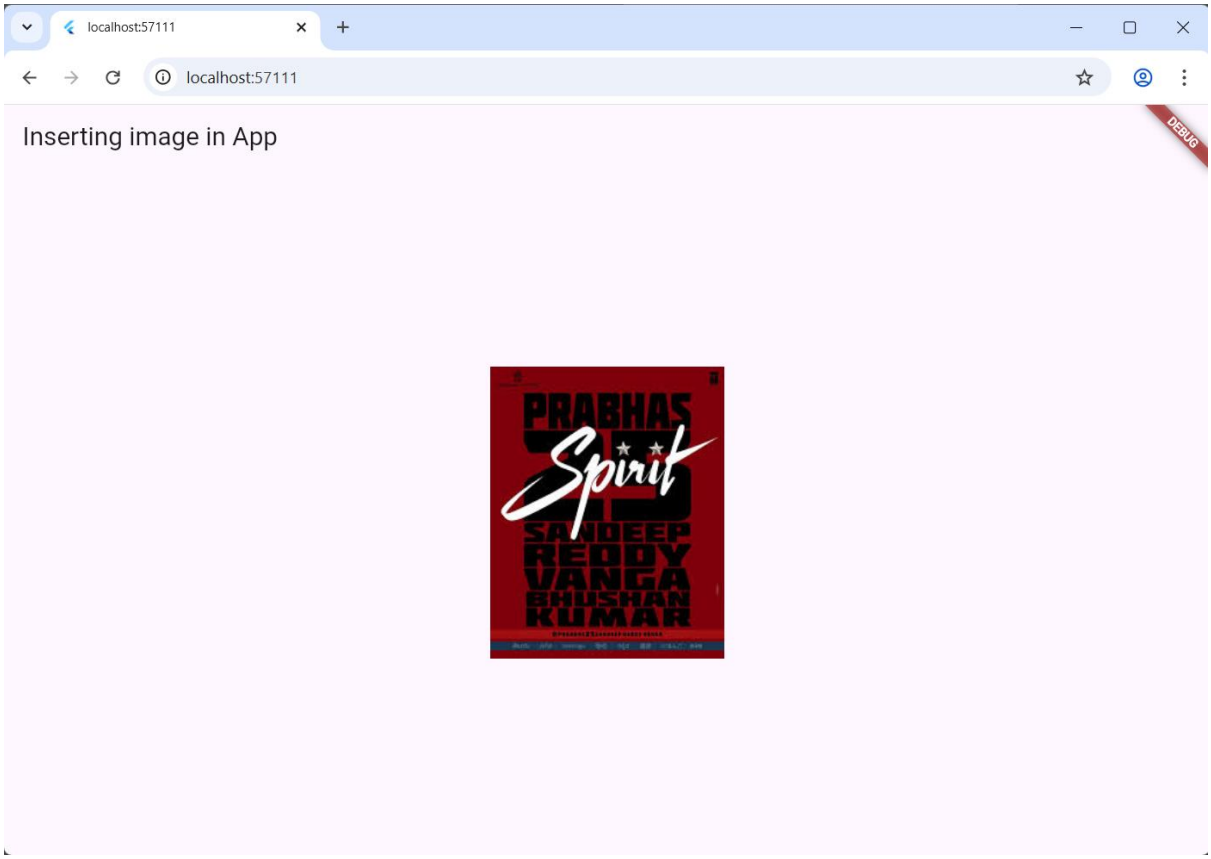
```
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text("Inserting image in App")),
        body: Center(child: Image.asset('assets/vangod_spirit.jpeg')),
      ),
    );
  }
}
```

Steps to Add Local Image:

1. Create a folder in your project with name **assets**
2. Place your image (e.g., vangod_spirit.jpeg) inside that folder.
3. Edit your **pubspec.yaml** file in the project folder :

```
flutter:
  assets:
    - assets/vangod_spirit.jpeg
```

OUTPUT :


localhost:57111
localhost:57111
Inserting image in App
DEBUG
PRABHAS
Spirit
SANDEEP
REDDY
VANGA
BHUSHAN
KUMAR

Creating Flutter Widget Container:

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Container Example')),
        body: Center(
          child: Container(
            width: 150,
            height: 150,
            padding: EdgeInsets.all(50),
            margin: EdgeInsets.all(50),
            decoration: BoxDecoration(
              color: Colors.blue[200],
              borderRadius: BorderRadius.circular(22),
              border: Border.all(color: Colors.blue, width: 12),
            ),
            child: Center(
              child: Container(color: Colors.orange, width: 100, height: 100),
```

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

Attribute	Purpose
width	Sets the container's width (in logical pixels)
height	Sets the container's height
color	Background color (use directly only if no decoration is used)
alignment	Aligns the child widget inside the container (e.g., Alignment.center)
padding	Space **inside** the container, around its child
margin	Space **outside** the container, around its border
child	The single widget placed inside the container
decoration	Used for styling like background image, gradient, border, shadows

OUTPUT :

2 b) Implement different layout structures using Row, Column, and Stack widgets.

Implementing Row (layout Structure) Widget :

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Demonstrating row Layout')),
        body: Center(
          child: Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            crossAxisAlignment: CrossAxisAlignment.center,
            children: [
              Container(
                height: 80,
                width: 80,
                color: Colors.blueAccent,
                child: Text("Box1"),
              ),
              Container(
```

```
            height: 80,
            width: 80,
            color: Colors.yellow,
            child: Text("Box2"),
          ),
          Container(
            height: 80,
            width: 80,
            color: Colors.lightGreen,
            child: Text("Box3"),
          ),
        ],
      ),
    ),
  );
 }
}
```

- Row: Lays out its children **horizontally**
- mainAxisAlignment.spaceEvenly: Distributes the children **with equal spacing**
- crossAxisAlignment.center: Aligns all boxes **vertically in the center**
- Three Containers with different background colors and labels

OUTPUT :


localhost:62131
localhost:62131
Demonstrating row Layout
DEBUG
Box1
Box2
Box3

Implementing Row (layout Structure) Widget :

```
import 'package:flutter/material.dart';
void main() {
  runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text("Demonstrating Column Layout")),
        body: Column(
          mainAxisAlignment: MainAxisAlignment.start,
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Container(
              height: 55,
              width: 55,
              child: Text("Box1"),
              color: Colors.blueAccent,
            ),
            Container(
              height: 55,
              width: 55,
```

```
            child: Text("Box2"),
            color: Colors.pinkAccent,
          ),
          Container(
            height: 55,
            width: 55,
            child: Text("Box3"),
            color: Colors.tealAccent,
          ),
        ],
      ),
    ),
  );
 }
}
```

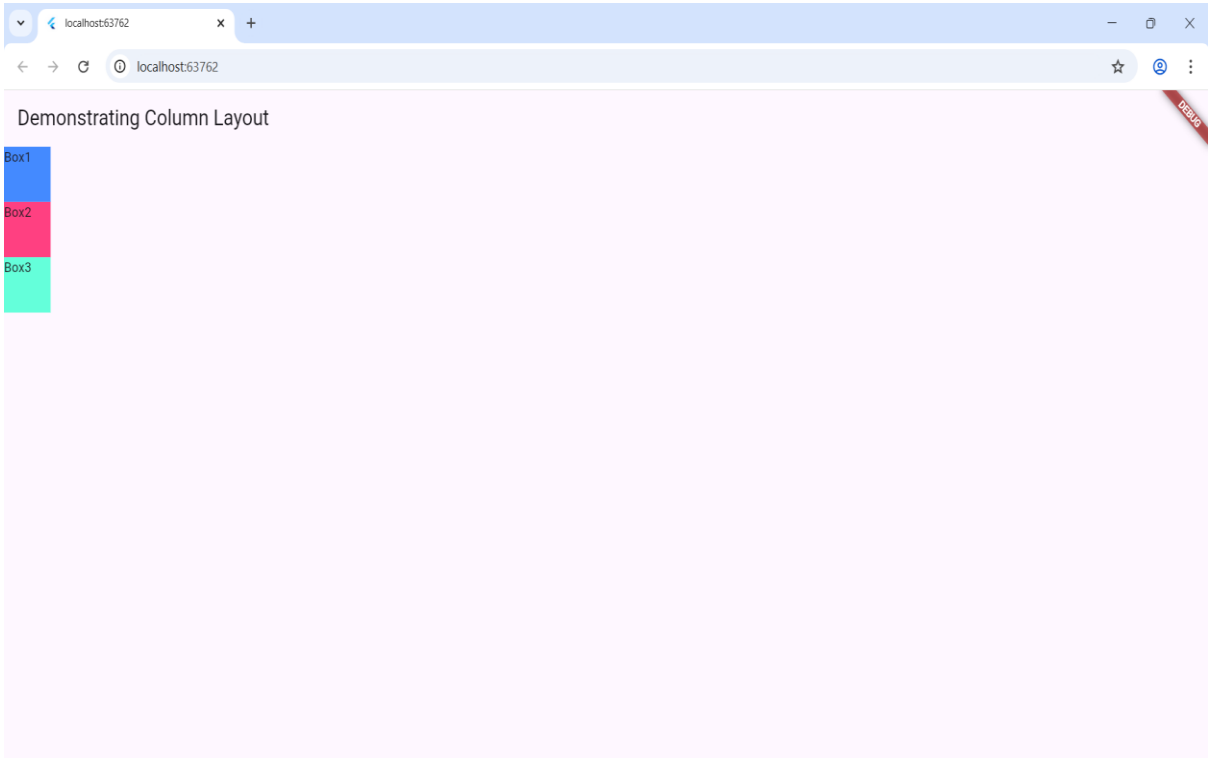
mainAxisAlignment

- **Defines how children are placed along the main axis.**
- For a Row, the main axis is **horizontal**.
- For a Column, the main axis is **vertical**.]

crossAxisAlignment

- **Defines how children are placed along the cross axis.**
- For a Row, the cross axis is **vertical**.
- For a Column, the cross axis is **horizontal**.

OUTPUT :

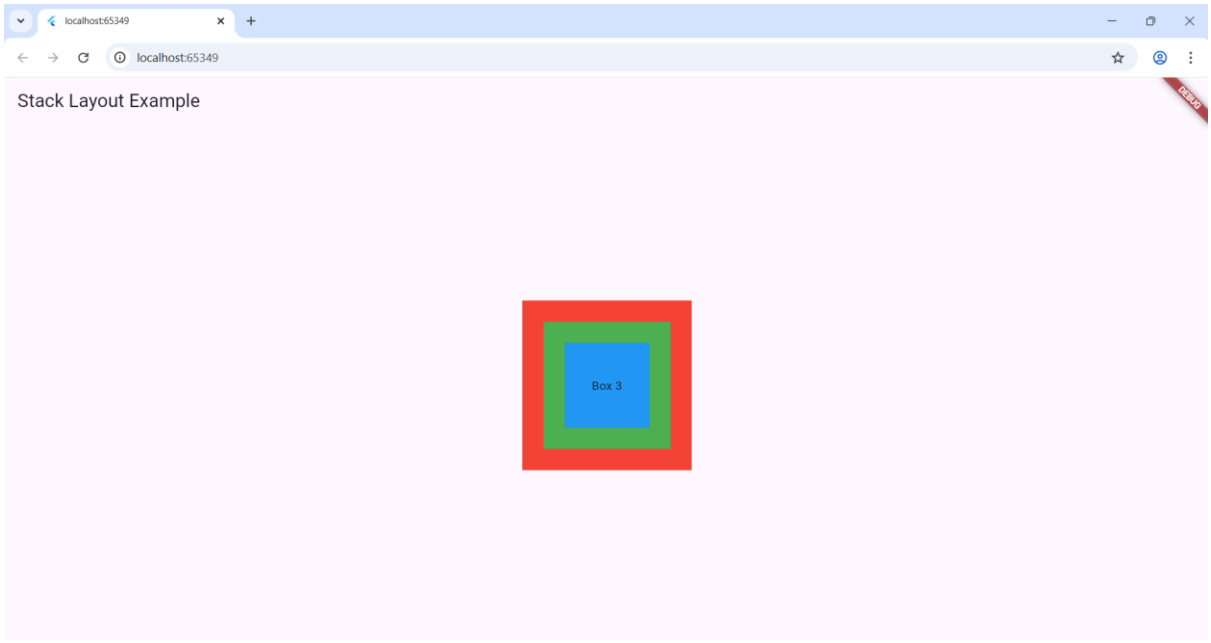

localhost:63762
localhost:63762
Demonstrating Column Layout
DEBUG
Box1
Box2
Box3

Implementing Row (layout Structure) Widget :

```
import 'package:flutter/material.dart';
void main()
{
runApp(MyApp());
}
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Stack Layout Example')),
        body: Center(
          child: Stack(
            alignment: Alignment.center,
            children: [
              Container(
                width: 200,
                height: 200,
                color: Colors.red,
                child: Center(child: Text('Box 1')),
              ),
              Container(
                width: 150,
```

```
            height: 150,
            color: Colors.green,
            child: Center(child: Text('Box 2')),
          ),
          Container(
            width: 100,
            height: 100,
            color: Colors.blue,
            child: Center(child: Text('Box 3')),
          ),
        ],
      ),
    ),
  ),
 );
 }
}
```

OUTPUT :


localhost:65349
localhost:65349
Stack Layout Example
DEBUG
Box 3

3 a) Design a responsive UI that adapts to different screen sizes.

B**eginner-friendly responsive UI design in Flutter** that adapts gracefully to different screen sizes (like mobile and tablet). It uses:

- MediaQuery to detect screen width
- LayoutBuilder to switch layout dynamically
- Flexible for resizing elements
- Clean UI using Containers and Text widgets

PROGRAM :

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

void main()

{

 runApp(MyResponsiveApp());

}

class MyResponsiveApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      title: 'Responsive UI',

      theme: ThemeData(primarySwatch: Colors.blue),

      home: ResponsiveHomePage(),

    );

  }

}

class ResponsiveHomePage extends StatelessWidget {

  @override
```

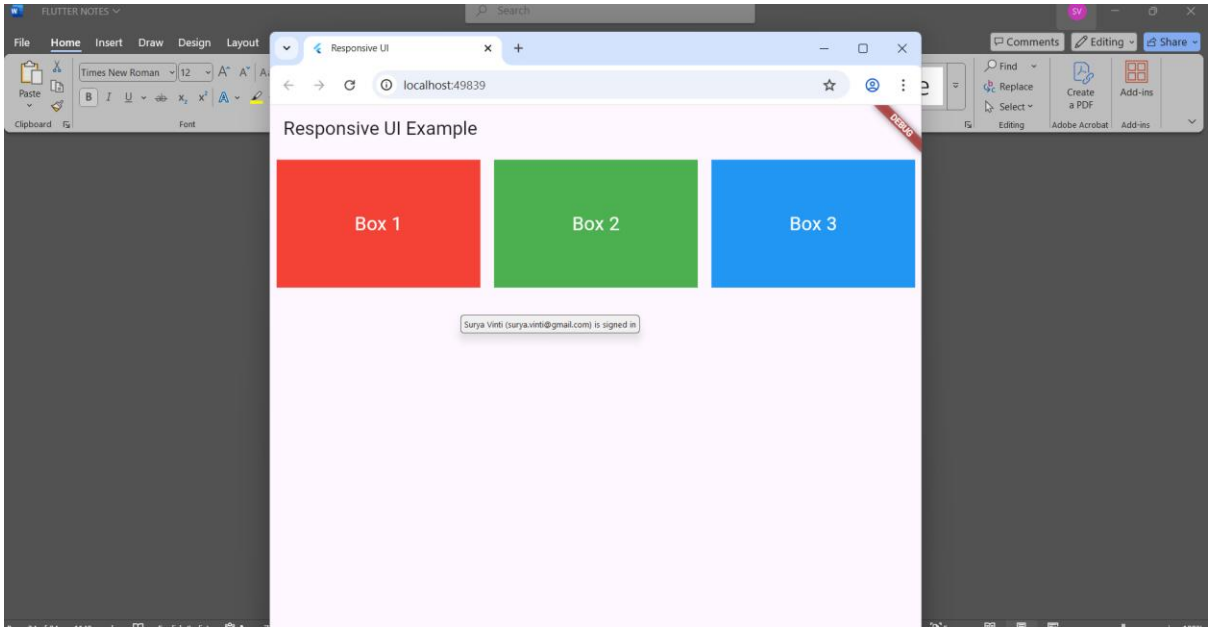
```
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: Text('Responsive UI Example')),
    body: LayoutBuilder(
      builder: (context, constraints) {
        if (constraints.maxWidth < 600) {
          //Small screen: use vertical layout
          return _buildVerticalLayout();
        } else {
          //Large screen: use horizontal layout
          return _buildHorizontalLayout();
        }
      },
    ),
  );
}

//Mobile View
Widget _buildVerticalLayout() {
  return Column(
    children: [
      _buildBox(Colors.red, 'Box 1'),
      SizedBox(height: 10),
      _buildBox(Colors.green, 'Box 2'),
```

```
      SizedBox(height: 10),
      _buildBox(Colors.blue, 'Box 3'),
    ],
  );
}

// 💻 Tablet/Desktop View
Widget _buildHorizontalLayout() {
  return Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: [
      Flexible(child: _buildBox(Colors.red, 'Box 1')),
      Flexible(child: _buildBox(Colors.green, 'Box 2')),
      Flexible(child: _buildBox(Colors.blue, 'Box 3')),
    ],
  );
}

// Reusable UI block
Widget _buildBox(Color color, String label) {
  return Container(
    height: 150,
    margin: EdgeInsets.all(8),
    color: color,
```

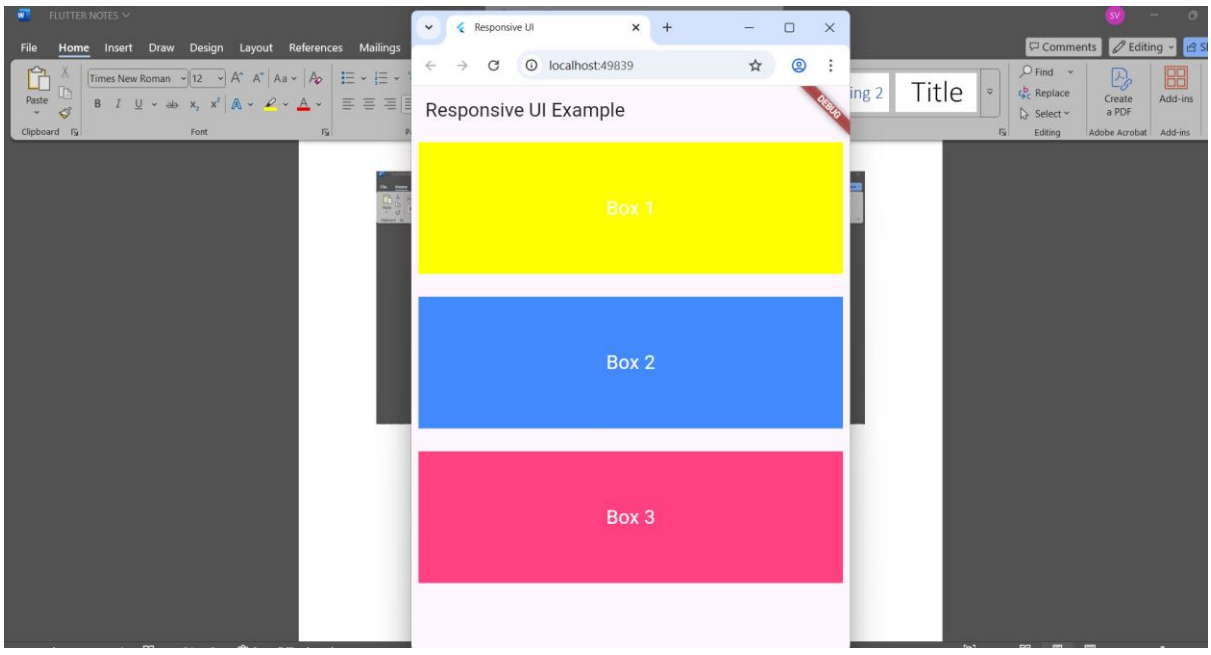
```
      child: Center(
        child: Text(
          label,
          style: TextStyle(fontSize: 22, color: Colors.white),
        ),
      ),
    );
  }
}
```

OUTPUT :

DESKTOP/TAB VIEW



DESKTOP/TAB VIEW



MOBILE VIEW

3 b) Implement media queries and breakpoints for responsiveness.

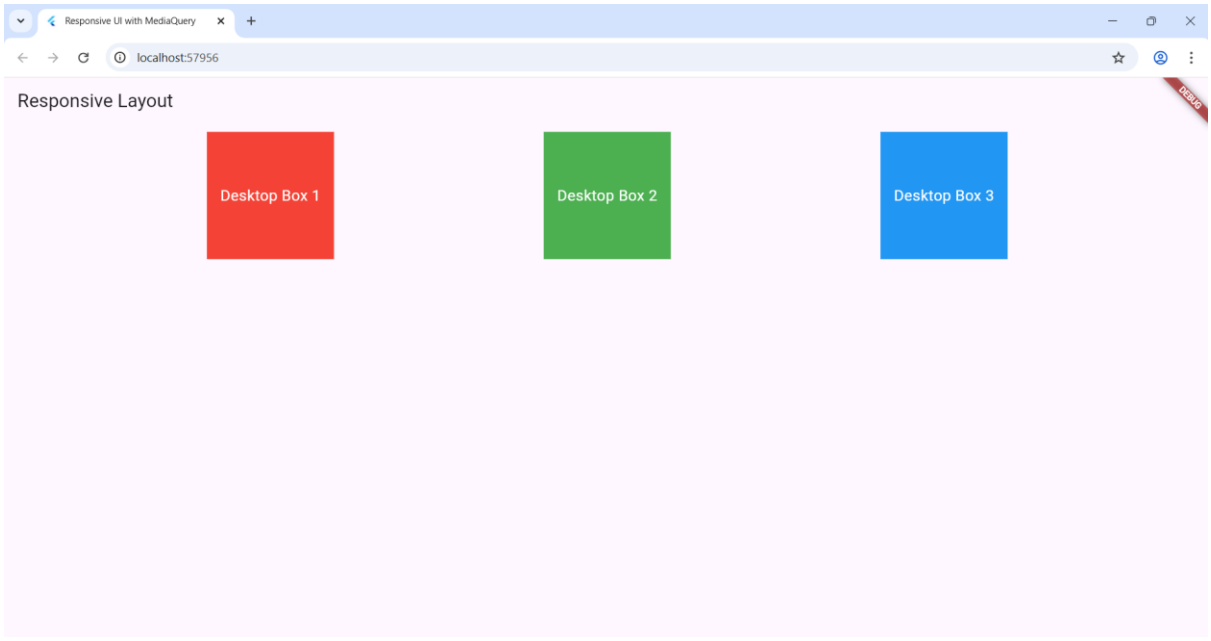
```
import 'package:flutter/material.dart';
void main()
{
runApp(MyResponsiveApp());
}
class MyResponsiveApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Responsive UI with MediaQuery',
      theme: ThemeData(primarySwatch: Colors.blue),
      home: ResponsiveHomePage(),
    );
  }
}

class ResponsiveHomePage extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    // 📏 Screen width from MediaQuery
    double screenWidth = MediaQuery.of(context).size.width;

    Widget content;

    // 🔁 Check breakpoints
    if (screenWidth < 600) {
      content = _buildMobileLayout();
    } else if (screenWidth >= 600 && screenWidth < 1024) {
      content = _buildTabletLayout();
```

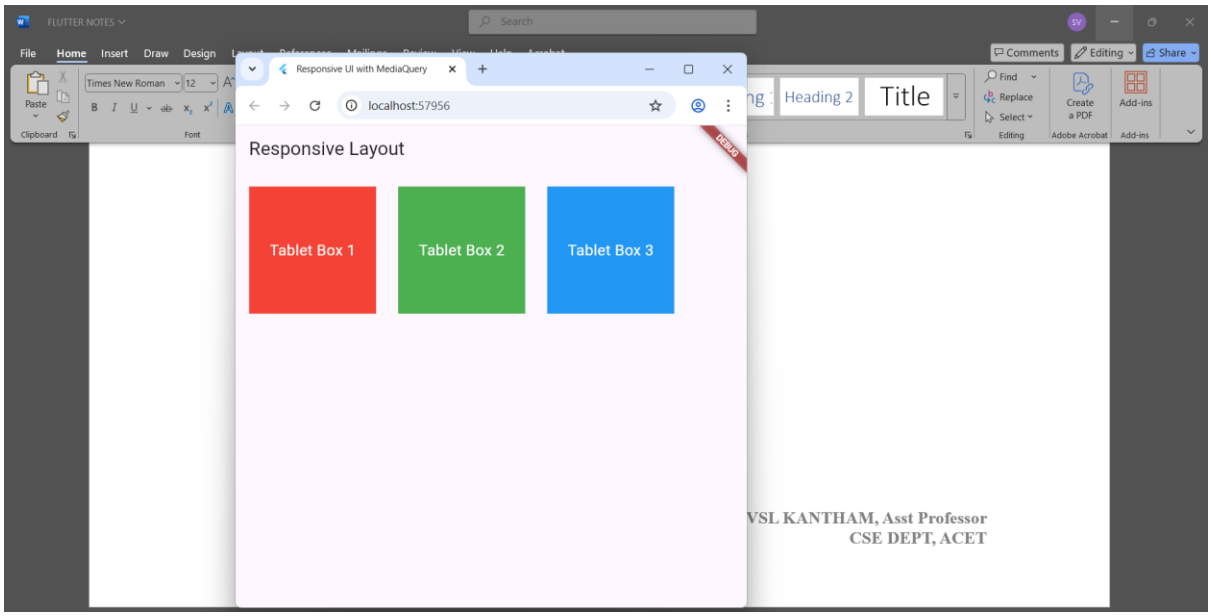
```
  } else {
    content = _buildDesktopLayout();
  }

  return Scaffold(
    appBar: AppBar(title: Text('Responsive Layout')),
    body: content,
  );
}

// 📱 Mobile layout (column)
Widget _buildMobileLayout() {
  return Column(
    children: [
      _buildBox(Colors.red, 'Mobile Box 1'),
      _buildBox(Colors.green, 'Mobile Box 2'),
      _buildBox(Colors.blue, 'Mobile Box 3'),
    ],
  );
}

// 📲 Tablet layout (grid-like)
Widget _buildTabletLayout() {
  return Padding(
    padding: const EdgeInsets.all(8.0),
    child: Wrap(
      spacing: 10,
      runSpacing: 10,
      children: [
        _buildBox(Colors.red, 'Tablet Box 1'),
```

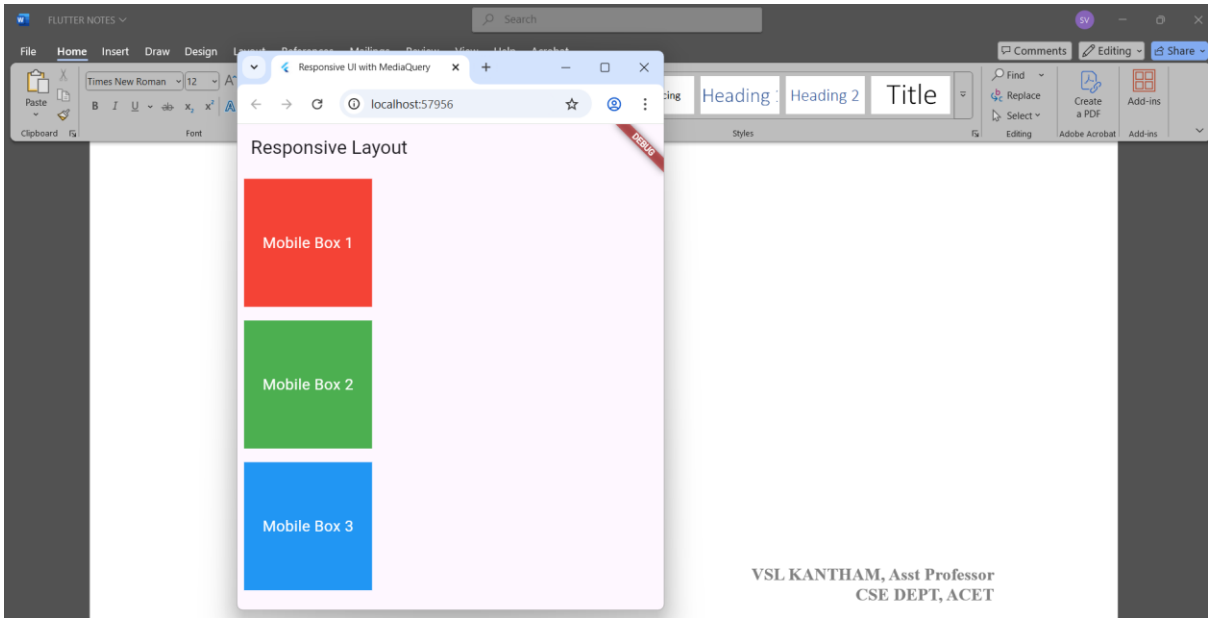
```
        _buildBox(Colors.green, 'Tablet Box 2'),
        _buildBox(Colors.blue, 'Tablet Box 3'),
      ],
    ),
  );
}

// 💻 Desktop layout (row)
Widget _buildDesktopLayout() {
  return Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: [
      _buildBox(Colors.red, 'Desktop Box 1'),
      _buildBox(Colors.green, 'Desktop Box 2'),
      _buildBox(Colors.blue, 'Desktop Box 3'),
    ],
  );
}

// 🔧 Reusable widget
Widget _buildBox(Color color, String label) {
  return Container(
    width: 150,
    height: 150,
    margin: EdgeInsets.all(8),
    color: color,
    child: Center(
      child: Text(
        label,
        style: TextStyle(color: Colors.white, fontSize: 18),
```

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

OUTPUT:


Responsive UI with MediaQuery
localhost:57956
Responsive Layout
Desktop Box 1
Desktop Box 2
Desktop Box 3


Responsive UI with MediaQuery
localhost:57956
Responsive Layout
Tablet Box 1
Tablet Box 2
Tablet Box 3
VSL KANTHAM, Asst Professor
CSE DEPT, ACET


FLUTTER NOTES
Search
File
Home
Insert
Draw
Design
Comments
Editing
Share
Responsive UI with MediaQuery
localhost:57956
Heading
Heading 2
Title
Responsive Layout
DEBUG
Mobile Box 1
Mobile Box 2
Mobile Box 3
VSL KANTHAM, Asst Professor
CSE DEPT, ACET

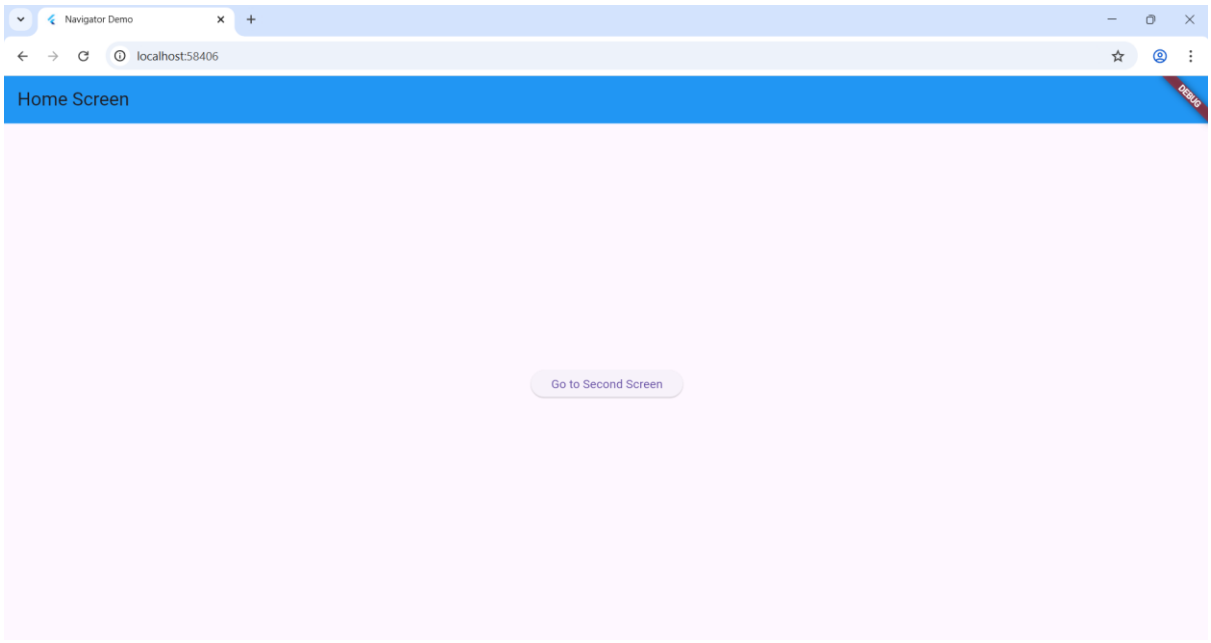
4 a) Set up navigation between different screens using Navigator.

- Navigator.push moves to a new screen
- Navigator.pop returns to the previous screen

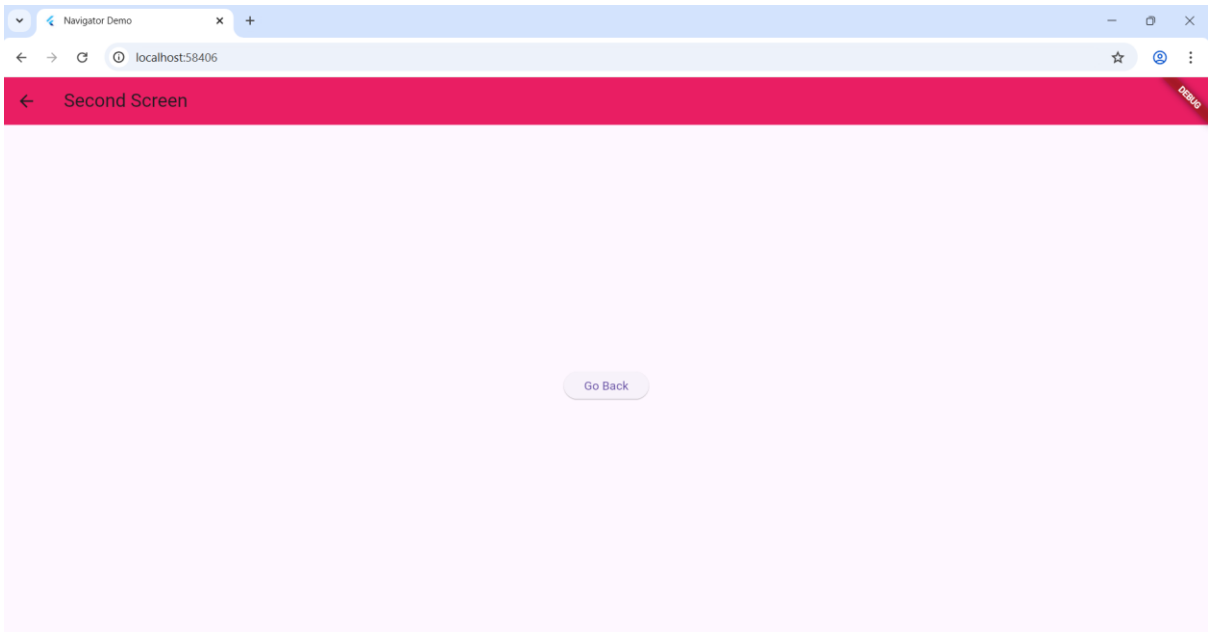
```
import 'package:flutter/material.dart';

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

class MyNavigationApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Navigator Demo',
      theme: ThemeData(primarySwatch: Colors.pink),
      home: HomeScreen(),
    );
  }
}

// 🏠 Home Screen
class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Home Screen'), backgroundColor: Colors.blue),
      body: Center(
        child: ElevatedButton(
          child: Text('Go to Second Screen'),
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => SecondScreen()),
            );
          },
        ),
```

```
      ),
    );
  }
}

// 📄 Second Screen
class SecondScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Second Screen'),
        backgroundColor: Colors.pink,
      ),
      body: Center(
        child: ElevatedButton(
          child: Text('Go Back'),
          onPressed: () {
            Navigator.pop(context);
          },
        ),
      ),
    );
  }
}
```

OUTPUT:



FIRST SCREEN



SECOND SCREEN

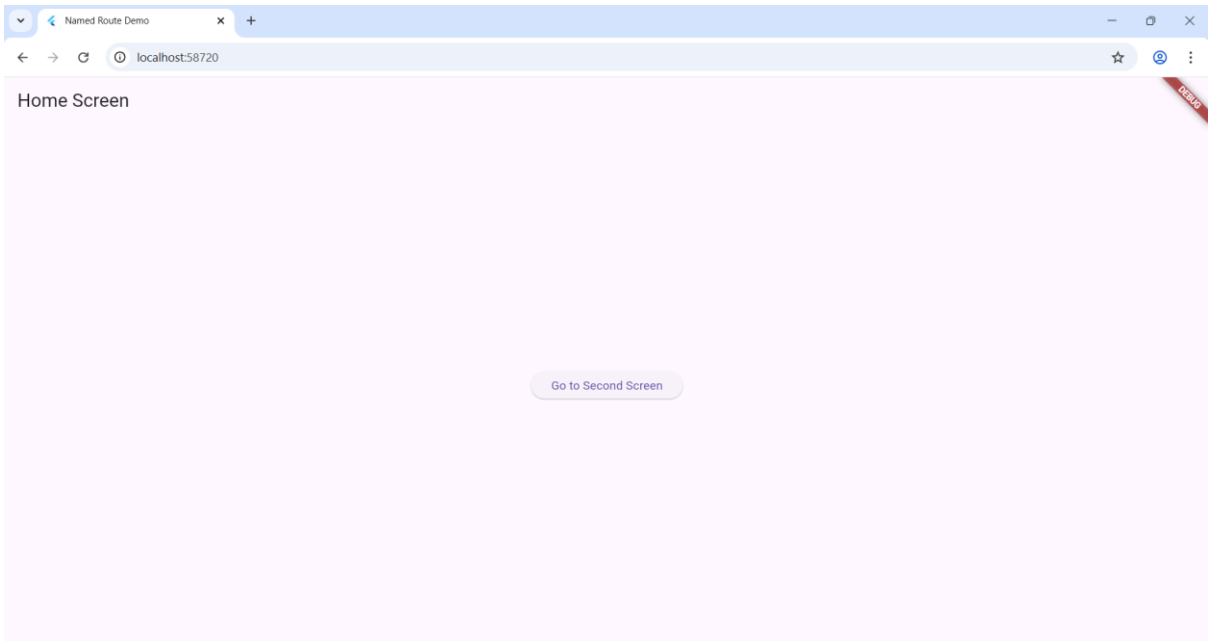
b) Implement navigation with named routes.

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

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

class MyNamedRouteApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Named Route Demo',
      // ✅ Define named routes here
      initialRoute: '/',
      routes: {
        '/': (context) => HomeScreen(),
        '/second': (context) => SecondScreen(),
      },
    );
  }
}

// 🏠 Home Screen
class HomeScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Home Screen')),
      body: Center(
        child: ElevatedButton(
          child: Text('Go to Second Screen'),
          onPressed: () {
            // 🚀 Navigate using named route
            Navigator.pushNamed(context, '/second');
          },
```

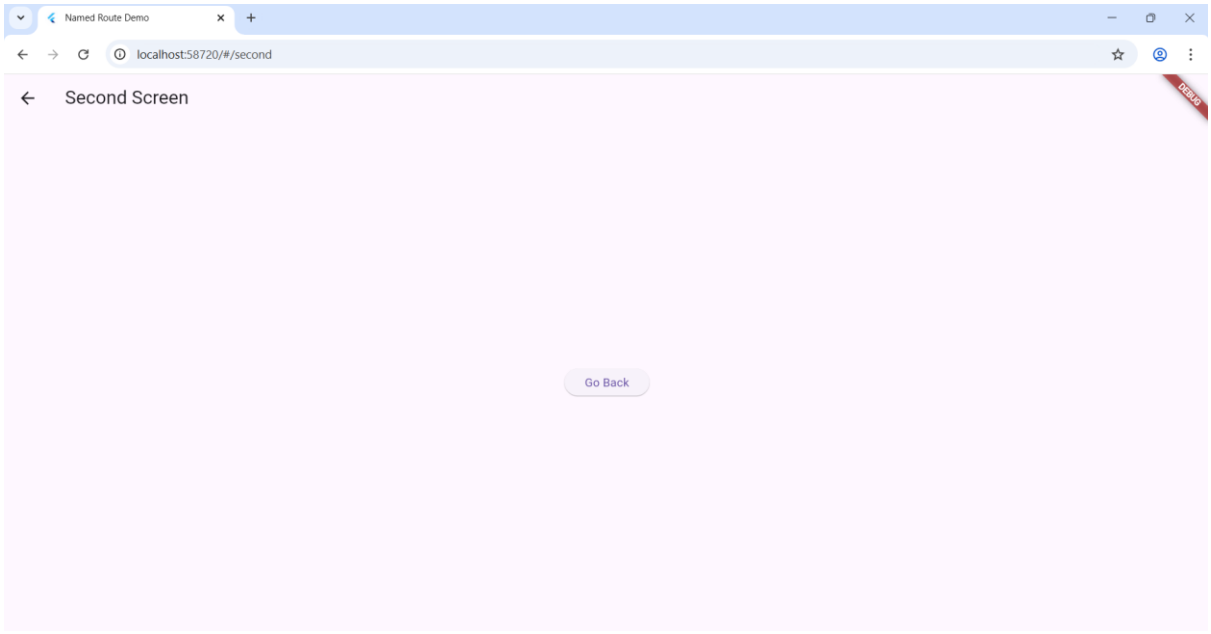
```
        ),
      ),
    );
  }
}

// 📄 Second Screen
class SecondScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Second Screen')),
      body: Center(
        child: ElevatedButton(
          child: Text('Go Back'),
          onPressed: () {
            Navigator.pop(context); // ⬅️ Go back
          },
        ),
      ),
    );
  }
}
```

Concept	Description
routes	A map of route names to widget builders
initialRoute	The first screen shown when the app starts
Navigator.pushNamed	Navigates using a route name
Navigator.pop	Returns to the previous screen

OUTPUT :



FIRST SCREEN



SECOND SCREEN

5 a) Learn about stateful and stateless widgets.

Stateless Widgets

A Stateless Widget is a widget that does not hold any state. This means that the widget's properties are immutable and cannot change during its lifecycle. Stateless widgets are ideal for displaying static content, such as text or images. This means these widgets are ideal for situations where the UI does not need to change dynamically in response to user interactions or other events.

Characteristics of Stateless Widgets:

- Immutability: Once created, the properties cannot be changed.
- No State Management: They do not manage any internal state.
- Rebuild on Parent Change: They rebuild only when their parent widget changes.

Benefits of Stateless Widgets:

- Simpler to code: Due to their immutable nature, stateless widgets require less code to implement.
- Faster performance: Stateless widgets are rebuilt efficiently, leading to smoother UI rendering.
- Easier to reason about: Their predictable behavior makes them easier to understand and maintain.

Stateful Widgets

A Stateful Widget is a widget that can change its state over time. This means that a Stateful Widget can rebuild itself when its state changes. They are suitable for scenarios where the UI needs to reflect changes based on user interactions, animations, or network responses. A stateful widget consists of two classes: the widget itself and a state class that holds the state. The state class is where you define the mutable state and the logic to manage it.

Characteristics of Stateful Widgets:

- Mutable State: They maintain an internal state that can change during the widget's lifecycle.
- State Management: They manage their state using a separate class that extends state.

- Rebuild on State Change: They rebuild when their internal state changes.

Advantages of Stateful Widgets:

- Dynamic UI: They provide the power to create UIs that respond to user actions and data changes.
- Complex interactions: They are well-suited for handling intricate user interactions and state management.

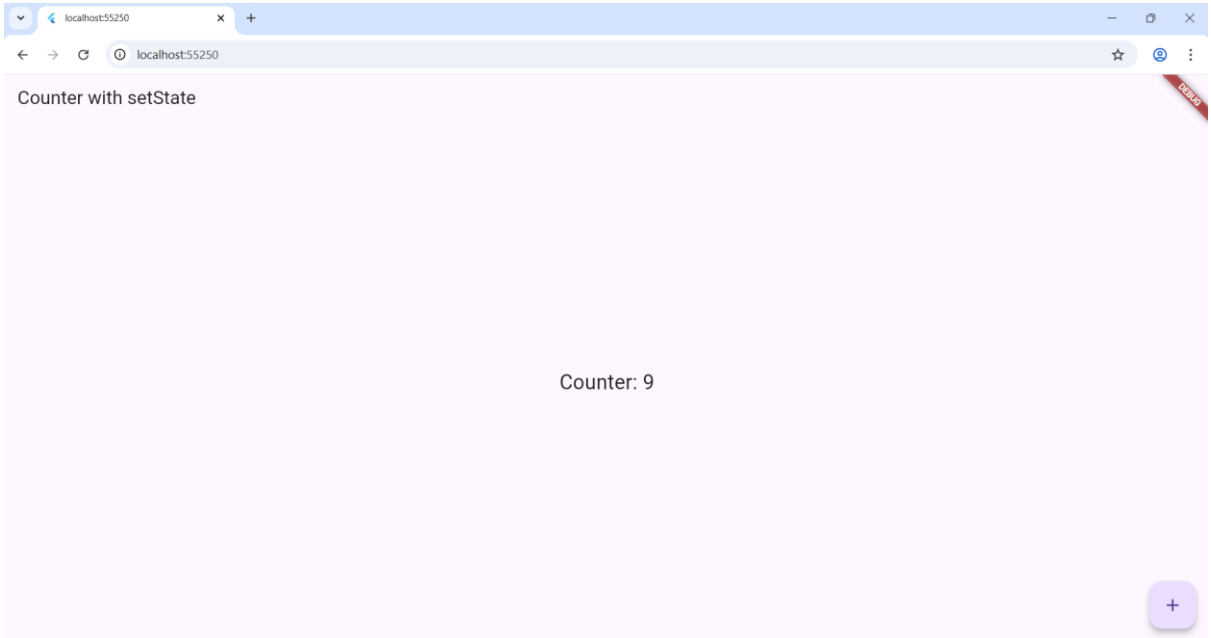
5b) Implement state management using set State and Provider.

State Management with setState() :

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


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


class MySetStateApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: CounterScreen(),
    );
  }
}


class CounterScreen extends StatefulWidget {
  @override
  _CounterScreenState createState() => _CounterScreenState();
}


class _CounterScreenState extends State<CounterScreen> {
  int counter = 0;
```

```
  void _incrementCounter() {
    setState(() {
      counter++; // updates UI
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Counter with setState")),
      body: Center(
        child: Text(
          "Counter: $counter",
          style: TextStyle(fontSize: 24),
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        child: Icon(Icons.add),
      ),
    );
  }
}
```

OUTPUT:



State Management with Provider:

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

// Step 1: Create ChangeNotifier class
class CounterProvider extends ChangeNotifier {
  int _counter = 0;

  int get counter => _counter;

  void increment() {
    _counter++;
    notifyListeners(); // tells widgets to rebuild
  }
```

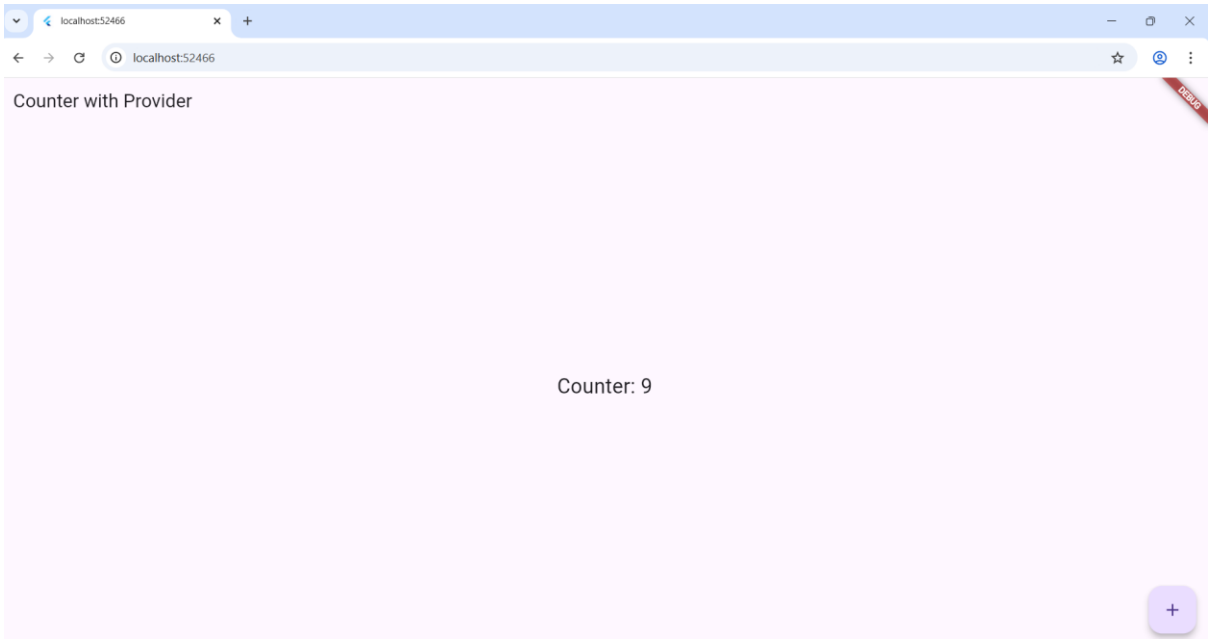
```
}

// Step 2: Wrap app with ChangeNotifierProvider
void main() {
  runApp(
    ChangeNotifierProvider(
      create: (_) => CounterProvider(),
      child: MyProviderApp(),
    ),
  );
}

class MyProviderApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: CounterScreen(),
    );
  }
}

// Step 3: Use Provider in UI
class CounterScreen extends StatelessWidget {
  @override
```

```
  Widget build(BuildContext context) {
    final counterProvider = Provider.of<CounterProvider>(context);

    return Scaffold(
      appBar: AppBar(title: Text("Counter with Provider")),
      body: Center(
        child: Text(
          "Counter: ${counterProvider.counter}",
          style: TextStyle(fontSize: 24),
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: counterProvider.increment,
        child: Icon(Icons.add),
      ),
    );
  }
}
```

Add dependency in pubspec.yaml:

```
dependencies:
  flutter:
    sdk: flutter
  provider: ^6.0.5
```

OUTPUT:


localhost:52466
localhost:52466
Counter with Provider
DEBUG
Counter: 9
+