INDEX

6	a) Create custom widgets for specific UI elements. b)Apply styling using themes and custom styles.	CO3
7	a) Design a form with various input fields. b)Implement form validation and error handling	CO4
8	a) Add animations to UI elements using Flutter's animation framework. b)Experiment with different types of animations (fade, slide, etc.).	CO4
9	a) Fetch data from a REST API. b)Display the fetched data in a meaningful way in the UI.	CO5
10	a) Write unit tests for UI components. b)Use Flutter's debugging tools to identify and fix issues.	CO5

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that this manual is a Bonafide record of practical work carried out in the **UI-FLUTTER** for the **B.Tech (Computer Science and Engineering) V Semester** Programme during the academic year **2025–2026.**

This manual has been prepared by **Mrs. M.Pallavi (Assistant Professor), Mr. A. Satchidanandam (Assistant Professor), Mr. K.Vivek(Assistant Professor)and Mrs. B.Niveditha(Assistant Professor)**, Department of Computer Science and Engineering, with my/our own efforts and to the best of our knowledge.

Signature of Lab Faculty **Signature of HOD**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

PREFACE

This laboratory plays the foundation for the Computer Science and Engineering students during Third year of their course.

This lab is designed to provide a hands-on introduction to Flutter, Google's UI toolkit for building natively compiled applications for mobile, web, and desktop from a single codebase.

Flutter has gained popularity due to its speed, expressiveness, and ability to create visually appealing and high-performance applications. In this lab, you'll explore the core concepts of Flutter development like Widget Composition, State Management and Responsive Design.

By,

Mrs.M Pallavi

Mr. A.Satchidanandam

Mr. K. Vivek

Mrs. B.Niveditha

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ACKNOWLEDGEMENT

It was really a good experience, working at Flutter-UI Lab. First, I would like to thank Mrs. M. Pallavi, Assistant Professor, Department of Computer Science and Engineering, Marri Laxman Reddy Institute of technology & Management for giving the technical support in preparing the document.

I express my sincere thanks to Dr.K.Abdul Basith, Head of the Department of Computer Science and Engineering, Marri Laxman Reddy Institute of technology & Management, for his concern towards me and gave me opportunity to prepare Flutter-UI laboratory manual.

I am deeply indebted and gratefully acknowledge the constant support and valuable patronage of Dr.Ravi Prasad, Dean Academics, Marri Laxman Reddy Institute of technology & Management. I am unboundedly grateful to him for timely corrections and scholarly guidance.

I express my heartful thanks to Dr.P.Sridhar, Director, and Dr. R.Murali Prasad, Principal, Marri Laxman Reddy Institute of technology & Management, for giving me this wonderful opportunity for preparing the Flutter-UI laboratory manual.

At last, but not the least I would like to thank the entire Computer Science Department faculties those who had inspired and helped me to achieve my goal.

By,

Mrs.M Pallavi, Assistant Professor,

Department of Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GENERAL INSTRUCTIONS

1. Students are instructed to come to Flutter-UI laboratory on time. Late comers are not entertained in the lab.

2. Students should be punctual to the lab. If not, conducted experiments will not be repeated.

3. Students are expected to come prepared at home with the experiments which are going to performed.

4. Students are instructed to display their identity cards and apron before entering into the lab.

5. Students are instructed not to bring mobile phones to the lab.

6. The equipment's and other accessories used in Flutter-UI lab should be handled with care and responsibility.

7. Any damage to the equipment's during the lab session is student's responsibility and penalty or fine will be collected from the student.

8. Students should update the records and lab observation books session wise. Before leaving the lab, the student should get his lab observation book signed by the faculty.

9. Students should submit the lab records 2/3 days in advance to the concerned faculty members in the staffroom for their correction and return.

10. Students should not move around the lab during the lab session.

11. If any emergency arises, the student should take the permission from faculty member concerned in written format.

12. The faculty members may suspend any student from the lab session on disciplinary grounds.

MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT
(AN AUTONOMOUS INSTITUTION)
(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)
Accredited by NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SAFETY MEASURES

1. While working in the laboratory suitable precautions should be observed to prevent accidents.
2. Always follow the experimental instructions strictly.
3. Use the first aid box in case of any accident/mishap.
4. Never work in the laboratory unless a demonstrator or teaching assistant in present.
5. When the experiment is completed, students should disconnect the setup made by them, and should return all the components/instruments taken for the purpose.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VISION & MISSION OF THE INSTITUTE

Vision of the Institute:

To be a globally recognized institution that fosters innovation, excellence, and leadership in education, research, and technology development, empowering students to create sustainable solutions for the advancement of society.

Mission of the Institute:

To foster a transformative learning environment that empowers students to excel in engineering, innovation, and leadership.

To produce skilled, ethical, and socially responsible engineers who contribute to sustainable technological advancements and address global challenges.

To shape future leaders through cutting-edge research, industry collaboration, and community engagement.

VISION & MISSION OF THE DEPARTMENT

Vision of the Department:

To empower the students to be technologically adept, innovative, self-motivated and responsible global citizen possessing human values and contribute significantlytowards high quality technical education with ever changing world.

Mission of the Department:

- To offer high-quality education in the computing fields by providing an environment where the knowledge is gained and applied to participate inresearch, for both students and faculty.
- To develop the problem solving skills in the students to be ready to deal with cutting edge technologies of the industry.
- To make the students and faculty excel in their professional fields by inculcating the communication skills, leadership skills, team building skills with the organization of various co-curricular and extra-curricular programmes.
- To provide the students with theoretical and applied knowledge, and adopt an education approach that promotes lifelong learning and ethical growth.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Program Educational Objectives (PEOs)

PEO 1	To induce strong foundation in mathematical and core concepts, which enable them to participate in research, in the field of computer science.
PEO 2	To be able to become the part of application development and problem solving by learning the computer programming methods, of the industry and related domains.
PEO 3	To gain the multidisciplinary knowledge by understanding the scope of association of computer science engineering discipline with other engineering disciplines.
PEO 4	To improve the communication skills, soft skills, organizing skills which build the professional qualities, there by understanding the social responsibilities and ethical attitude.

R22 B.TECH CSE SYLLABUS

CS506PC: UI DESIGN - FLUTTER

B.Tech. III Year I Sem. **L T P C**

0 0 2 1

Course Objectives:

- Learns to Implement Flutter Widgets and Layouts
- Understands Responsive UI Design and with Navigation in Flutter
- Knowledge on Widges and customize widgets for specific UI elements, Themes
- Understand to include animation apart from fetching data

Course Outcomes:

- Implements Flutter Widgets and Layouts
- Responsive UI Design and with Navigation in Flutter
- Create custom widgets for specific UI elements and also Apply styling using themes and custom styles.
- Design a form with various input fields, along with validation and error handling
- Fetches data and write code for unit Test for UI components and also animation

List of Experiments: Students need to implement the following experiments

1. a) Install Flutter and Dart SDK.
   b) Write a simple Dart program to understand the language basics.

2. a) Explore various Flutter widgets (Text, Image, Container, etc.).
   b) Implement different layout structures using Row, Column, and Stack widgets.

3. a) Design a responsive UI that adapts to different screen sizes.
   b) Implement media queries and breakpoints for responsiveness.

4. a) Set up navigation between different screens using Navigator.
   b) Implement navigation with named routes.

5. a) Learn about stateful and stateless widgets.
   b) Implement state management using set State and Provider.

6. a) Create custom widgets for specific UI elements.

b) Apply styling using themes and custom styles.

7. a) Design a form with various input fields.
   b) Implement form validation and error handling.

8. a) Add animations to UI elements using Flutter's animation framework.
   b) Experiment with different types of animations (fade, slide, etc.).

9. a) Fetch data from a REST API.
   b) Display the fetched data in a meaningful way in the UI.
10. a) Write unit tests for UI components.
    b) Use Flutter's debugging tools to identify and fix issues.

TEXT BOOK:

11. 1. Marco L. Napoli, Beginning Flutter: A Hands-on Guide to App Development.

MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT
(AN AUTONOMOUS INSTITUTION)
(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)
Accredited by NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FLUTTER-UI LABORATORY

LAB PLANNER

S.No	Experiment	CO	Virtual Lab Availabilty	Date planned	Date conducted
1	a) Install Flutter and Dart SDK. b)Write a simple Dart program to understand the language basics.	CO1	NA		
2	a) Explore various Flutter widgets (Text, Image, Container, etc.). b)Implement different layout structures using Row, Column, and Stack widgets.	CO2	NA		
3	a) Design a responsive UI that adapts to different screen sizes. b) Implement media queries and breakpoints for responsiveness	CO2	NA		
4	a) Set up navigation between different screens using Navigator. b) Implement navigation with named routes.	CO2	NA		

5	a) Learn about stateful and stateless widgets. b)Implement state management using set State and Provider.	CO3	NA		
6	a) Create custom widgets for specific UI elements. b)Apply styling using themes and custom styles.	CO3	NA		
7	MID-I				
8	a) Create custom widgets for specific UI elements. b)Apply styling using themes and custom styles.	CO3	NA		
9	a) Design a form with various input fields. b)Implement form validation and error handling	CO4	NA		
10	a) Add animations to UI elements using Flutter's animation framework. b)Experiment with different types of animations (fade, slide, etc.).	CO4	NA		
11	a) Fetch data from a REST API.	CO5	NA		

	b)Display the fetched data in a meaningful way in the UI.				
12	a) Write unit tests for UI components. b)Use Flutter's debugging tools to identify and fix issues.	CO5	NA		
13	Revision				
14	MID-II				

MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT
(AN AUTONOMOUS INSTITUTION)
(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)
Accredited by NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FLUTTER-UI LABORATORY

LAB PLANNER

VIRTUAL LAB :NA

MARRI LAXMAN REDDY
INSTITUTE OF TECHNOLOGY AND MANAGEMENT
(AN AUTONOMOUS INSTITUTION)
(Approved by AICTE, New Delhi & Affiliated to JNTUH, Hyderabad)
Accredited by NAAC with 'A' Grade & Recognized Under Section2(f) & 12(B)of the UGC act,1956

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FLUTTER-UI LABORATORY

RUBRICS USED TO ASSESS LEARNINGS IN LABORATORIES

1. RUBRICS FOR DAY TO DAY EVALUATION

Parameter	Max Marks	Level-1 (Very Poor)	Level-2 (Poor)	Level-3 (Average)	Level-4 (Good)	Level-5 (Excellent)
Observation Book	05	No observations or irrelevant data. (0–1)	Incomplete or incorrect data. (2)	Basic values with some errors. (3)	Mostly correct with good format. (4)	Fully correct, clear, and well-formatted. (5)
Record Writing	05	Not submitted. (0-1)	Submitted but mostly incomplete. (2)	Submitted with some missing/wrong parts. (3)	Submitted with minor issues. (4)	Fully complete, correct algorithm & flowchart. (5)
Result	05	No result or major errors. (0–1)	Result partially obtained. (2)	Acceptable result with limited error. (3)	Near-correct result and reasonable error. (4)	Accurate result. (5)
Viva-Voce	05	Did not answer any questions. (1)	Answered very few questions. (2)	Answered some questions with help. (3)	Answered most questions correctly. (4)	Answered all questions accurately. (5)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

FLUTTER-UI LABORATORY

2.RUBRICS FOR INTERNAL EVALUATION

Criterion	Max Marks	Level-1 *(Very Poor)*	Level-2 *(Poor)*	Level-3 *(Average)*	Level-4 *(Good)*	Level-5 *(Excellent)*
Design/Tool/Apparatus Selection	2 Marks	Incorrect tool/design and no reasoning. **(0)**	Tool/design selection attempted with unclear logic. **(0.5)**	Satisfactory selection with partial justification. **(1)**	Correct selection and proper analysis with few errors. **(1.5)**	Smart selection with accurate, relevant analysis. **(2)**
Execution (Code/Debug/Run) /Analysis/Method Used	4 Marks	Did not attempt or completely failed to execute. **(0)**	Attempted but unable to proceed or with major errors. **(1)**	Partial execution with some logic/syntax errors. **(2)**	Mostly correct execution with minimal help. **(3)**	Fully correct and independently executed program. **(4)**
Results& Documentation	2 Marks	Incomplete or poorly presented. **(0)**	Basic structure but lacks clarity or formatting. **(0.5)**	Complete but generic or with formatting issues. **(1)**	Well-structured and mostly clear. **(1.5)**	Well-organized, professional, and engaging documentation. **(2)**
Viva-Voce (Understanding of Concepts)	2 Marks	No understanding; could not answer questions. **(0)**	Answered a few with difficulty. **(0.5)**	Answered half the questions with basic clarity. **(1)**	Good understanding with confident answers. **(1.5)**	Answered all questions with clarity and depth. **(2)**

FLUTTER-UI LABORATORY

3. RUBRICS FOR SEMESTER END EXAMINATIONS

Criterion	Max Marks	Level-1 (Very Poor *(0–2 marks)*	Level-2 (Poor) *(3–4 marks)*	Level-3 (Average) *(5–6 marks)*	Level-4 (Good) *(7–9 marks)*	Level-5 (Excellent) *(10–12 marks)*
Preparedness for the Experiment	12 marks	No clarity on objective or procedure. Unable to explain basics.	Limited idea of the objective/procedure. Needed prompting.	Has basic understanding; minor gaps in concept or preparation.	Well-prepared, with clear understanding of steps and background.	Fully prepared with strong conceptual clarity and confident explanation.
Performance in the Laboratory	12 marks	Unable to perform experiment. Relied entirely on examiner's help.	Performed with multiple errors and constant support.	Performed with some errors; required occasional help.	Performed mostly independently with minimal support.	Performed independently, efficiently, and with precision.
Calculations & Graphs	12 marks	No or incorrect calculations. Graphs missing or irrelevant.	Multiple calculation errors. Graphs/plots inaccurate or poorly labeled.	Calculations partially correct. Graphs present but with some flaws.	Correct calculations and graphs with minor errors.	Accurate calculations and well-labeled graphs with proper nterpretation
Results & Error Analysis	12 marks	No result or invalid result. No error analysis attempted.	Incorrect result with vague or no error discussion.	Acceptable result. Error analysis attempted but limited.	Correct result with sound error discussion.	Accurate result with detailed and relevant error analysis.
Viva-Voce (Subject Knowledge)	12 marks	Unable to answer any questions. No. conceptual nderstanding.	Answered few questions with poor logic.	Answered half of the questions with average understanding.	Answered most questions with clarity and confidence.	Answered all questions with depth, clarity, and reasoning.

Experiment 1:

1. a) Aim: Install Flutter and Dart SDK.

System Requirements

- Operating System: Windows 10 or later (64-bit)
- Disk Space: 1.64 GB (does not include disk space for IDE/tools)
- Tools: Git for Windows 2.x, PowerShell 5.0, and a compatible IDE (VS Code, Android Studio, IntelliJ)

Step 1: Get the Flutter SDK

1. Download the Flutter SDK:
    - Visit the Flutter SDK releases page.
    - Download the latest stable release of the Flutter SDK (the .zip file).
2. Extract the Flutter SDK:
    - Extract the downloaded .zip file and place the contained flutter directory in a desired installation location (e.g., C:\src\flutter).

Step 2: Update Your Path

1. Locate Your System Path:
    - Open the Start Search, type in env, and select Edit the system environment variables.
    - In the System Properties window, click on the Environment Variables button.
2. Update Path:
    - In the User variables section, find the Path variable and click Edit.
    - Click New and add the full path to the flutter\bin directory (e.g., C:\src\flutter\bin).
    - Click OK to close all windows.
3. Verify Flutter is Added:
    - Open a new Command Prompt or PowerShell window and run flutter doctor.

Step 3: Run flutter doctor

1. Open Command Prompt or PowerShell:
    - Type cmd or powershell in the Start menu and open it.
2. Run flutter doctor:
    - In the terminal, type flutter doctor and press Enter.

- This command checks your environment and displays a report of the status of your Flutter installation.

Step 4: Install Android Studio

1. Download Android Studio:
   - Visit the Android Studio download page and download the installer.
2. Install Android Studio:
   - Run the installer and follow the setup wizard to complete the installation.
   - During installation, make sure the boxes for the following are checked:
     - Android SDK
     - Android SDK Platform
     - Android Virtual Device
3. Setup Android Studio:
   - Open Android Studio.
   - Complete the Android Studio Setup Wizard, which includes downloading the Android SDK components.

Step 5: Set Up the Android Emulator

1. Open Android Studio:
   - Go to Tools > AVD Manager.
2. Create a Virtual Device:
   - Click on Create Virtual Device, select a hardware profile, and click Next.
   - Select a system image (e.g., x86 Images tab), download if necessary, and click Next.
   - Click Finish to create the AVD.

Step 6: Set Up the IDE (Visual Studio Code)

1. Download VS Code:
   - Visit the Visual Studio Code download page and download the installer.
2. Install VS Code:
   - Run the installer and follow the setup wizard to complete the installation.
3. Install Flutter and Dart Plugins:
   - Open VS Code.
   - Go to Extensions (Ctrl+Shift+X).
   - Search for and install the Flutter and Dart extensions.

Step 7: Set Up the Flutter Device

1. Enable Developer Mode on Your Device:
    - Go to Settings > About phone and tap the Build number 7 times to unlock developer options.
    - Go to Settings > System > Developer options and enable USB debugging.
2. Connect Your Device:
    - Connect your Android device to your computer via USB.
    - Run flutter devices in the terminal to verify that Flutter recognizes your connected device.

Step 8: Create and Run a New Flutter Project

1. Create a New Flutter Project:
    - In VS Code, open the command palette (Ctrl+Shift+P).
    - Type Flutter: New Project, then press Enter.
    - Select a project name and location to save the project.
2. Run the Flutter Project:
    - Open the main.dart file in your new project.
    - Press F5 to start debugging and run your app.

That's it! You now have Flutter installed and set up on your Windows machine. You can start building Flutter applications. If you encounter any issues, the flutter doctor command can provide helpful diagnostics.

1 b) Aim:Write a simple Dart program to understand the language basics.

```
import 'package:flutter/material.dart';
void main() {
 runApp(Abc());
}
class Abc extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     home: Def(),
   ); }}
class Def extends StatelessWidget {
 const Def({super.key});
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       title: Text("Welcome"),
       backgroundColor: Colors.purple,
     ),
     body: Column(
       children: [
         //Widgets
```

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

OUTPUT:


12:39
DEBUG
Welcome

VIVA QUESTIONS

S.No	Question	CO	Blooms Taxonomy
1	What are the minimum system requirements for installing Flutter and Dart SDK on Windows/macOS/Linux?	CO1	Remember
2	Name two command-line tools you would typically use after installing Flutter.	CO1	Remember
3	What is the purpose of the flutter doctor command?	CO1	Remember
4	Why is it important to set up your environment variables correctly after installing Flutter?	CO1	Understand
5	How would you explain the role of a package manager (like pub for Dart) in the context of Flutter development?	CO1	Understand
6	You've just installed Flutter, but flutter doctor shows some issues. What are some common issues you might encounter, and how would you go about resolving them?	CO1	Apply
7	Show where you would find the flutter executable on your system after installation.	CO1	Apply
8	Analyze the output of flutter doctor. What information does it provide, and how can you use it to identify problems?	CO1	Analyze
9	Discuss the advantages and disadvantages of using different installation methods for Flutter (e.g., direct download vs. using a version manager like fvm).	CO1	Analyze
10	Which installation method for Flutter do you think is most efficient for a team development environment, and why?	CO1	Evaluate
11	Assess the importance of proper SDK installation for the overall development workflow. What are the potential consequences of a faulty installation?	CO1	Evaluate
12	Evaluate the helpfulness of the flutter doctor command. Are there any improvements you would suggest?	CO1	Evaluate
13	What keyword is used to declare a variable in Dart?	CO1	Remember
14	How do you print output to the console in Dart?	CO1	Remember
15	What is the entry point of a Dart program?	CO1	Remember
16	Explain the difference between var and final in Dart.	CO1	Understand
17	Describe the purpose of the main() function in a Dart program.	CO1	Understand

18	Write a Dart program that declares two integer variables, adds them, and prints the result.	CO1	Analyze
19	Compare and contrast for loops and while loops in Dart. When would you use one over the other?	CO1	Analyze
20	Which Dart data type would be most appropriate for representing a person's age, and why?	CO1	Evaluate
21	Assess the importance of comments in a simple Dart program. When are they most valuable?	CO1	Evaluate

Note :Each experiment should contain Minimum 20 Viva Questions

Experiment 2:

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

Text Widget:

```
import 'package:flutter/material.dart';
void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     home: Abc();
   );}}
class Abc extends StatelessWidget {
 const Abc({super.key});
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(title: Text('Text Widget Example')),
     body: Center(
       child: Text(
         'Hello, Flutter!',
         style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
```

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


12:43
DEBUG
Text Widget Example
Hello, Flutter!

Image Widgets

- Network Image

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

void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     home: Abc(),
   );
 }
}
class Abc extends StatelessWidget {
 const Abc({super.key});
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       title: Text("Image Widget"),
```

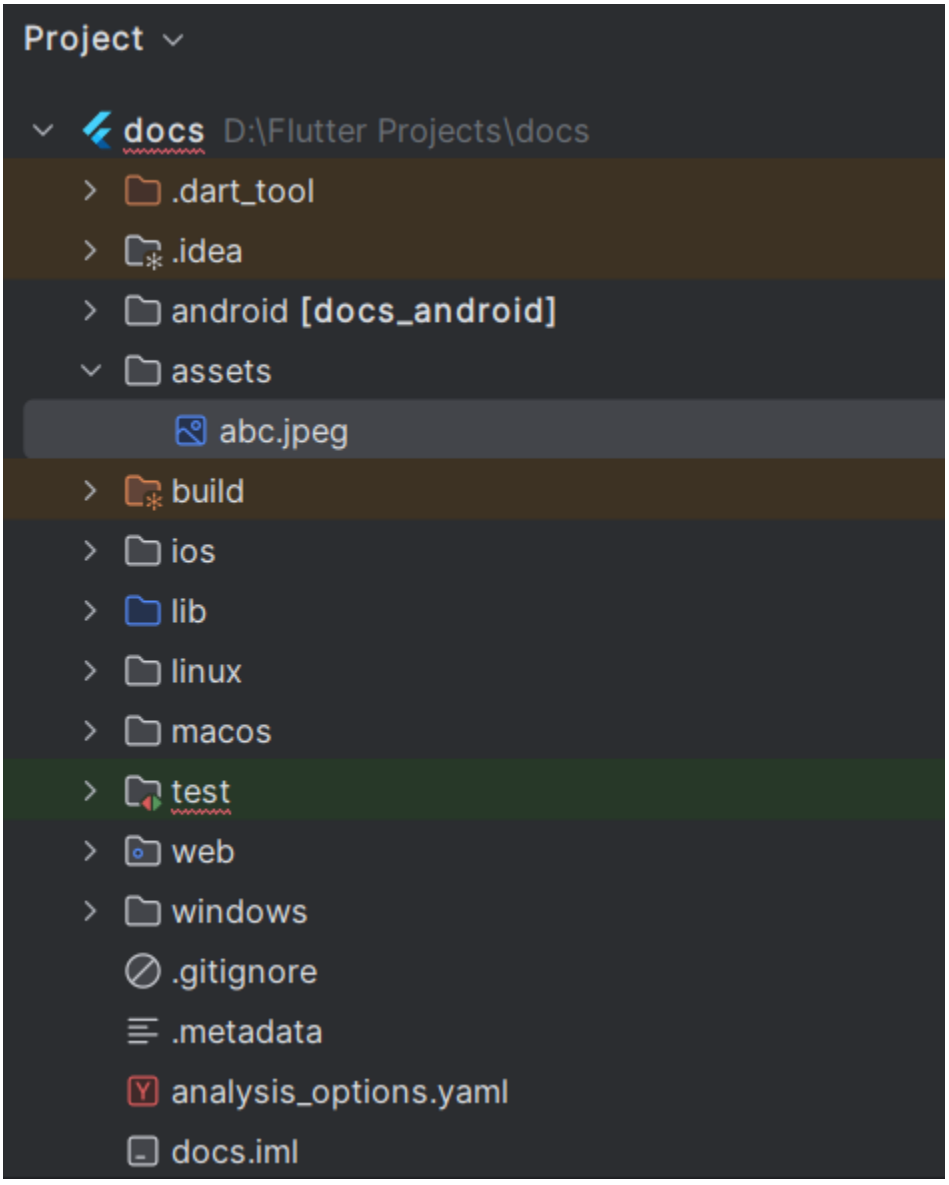
```
),

 body: Image.network('https://picsum.photos/250?image=9'),


);}}
```

Note: Must Add the Internet Tag in AndoridManifiest.yaml which is inside Android Folder for giving the Internet access to the App.

- **Asset Image or Local image**

- **Add Images to Your Project**

1. Create an assets directory: In your Flutter project root, create a directory named assets (or any other name you prefer). And copy the image of your requirement inside the assets folder.



2. Declare the assets: Open your pubspec.yaml file and add the assets under the flutter section.

   assets:

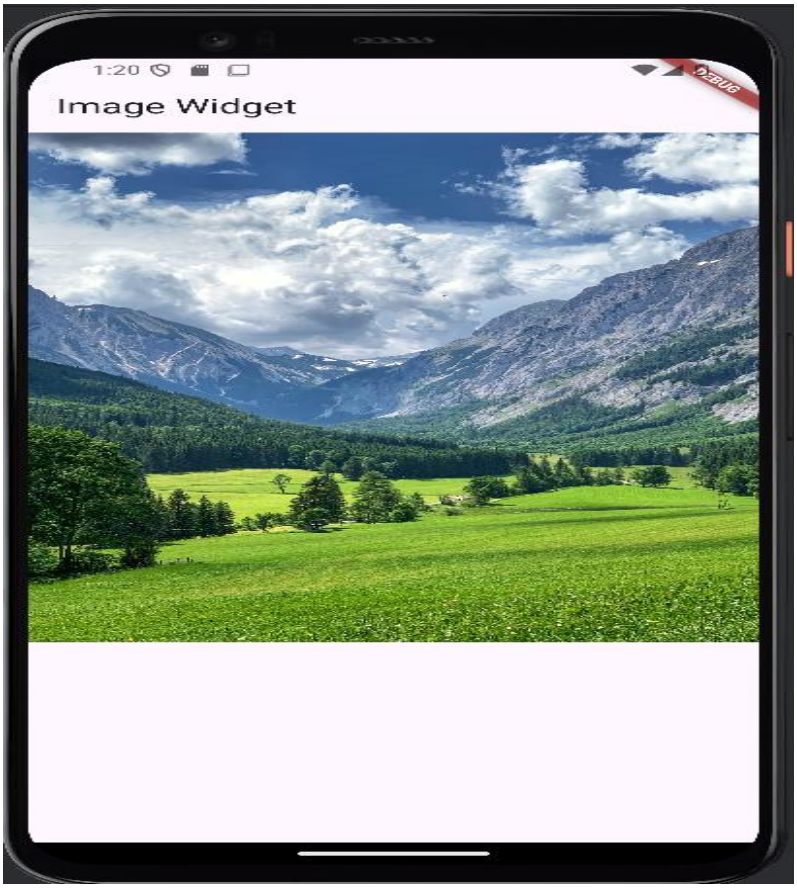
   - assets/abc.jpeg

3. Open Terminal: give the command 'flutter pub get'.
4. To Display Images in Your Flutter App, we will be using 'Image.asset' widget.

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

void main() {

 runApp(MyApp());

}

class MyApp extends StatelessWidget {

 @override

 Widget build(BuildContext context) {

   return MaterialApp(

     home: Abc(),

);}}

class Abc extends StatelessWidget {

 const Abc({super.key});

 @override

 Widget build(BuildContext context) {

   return Scaffold(

     appBar: AppBar(

       title: Text("Image Widget"),
```

```
),
body: Image.asset('assets/abc.jpeg'),   );}}
```

OUTPUT



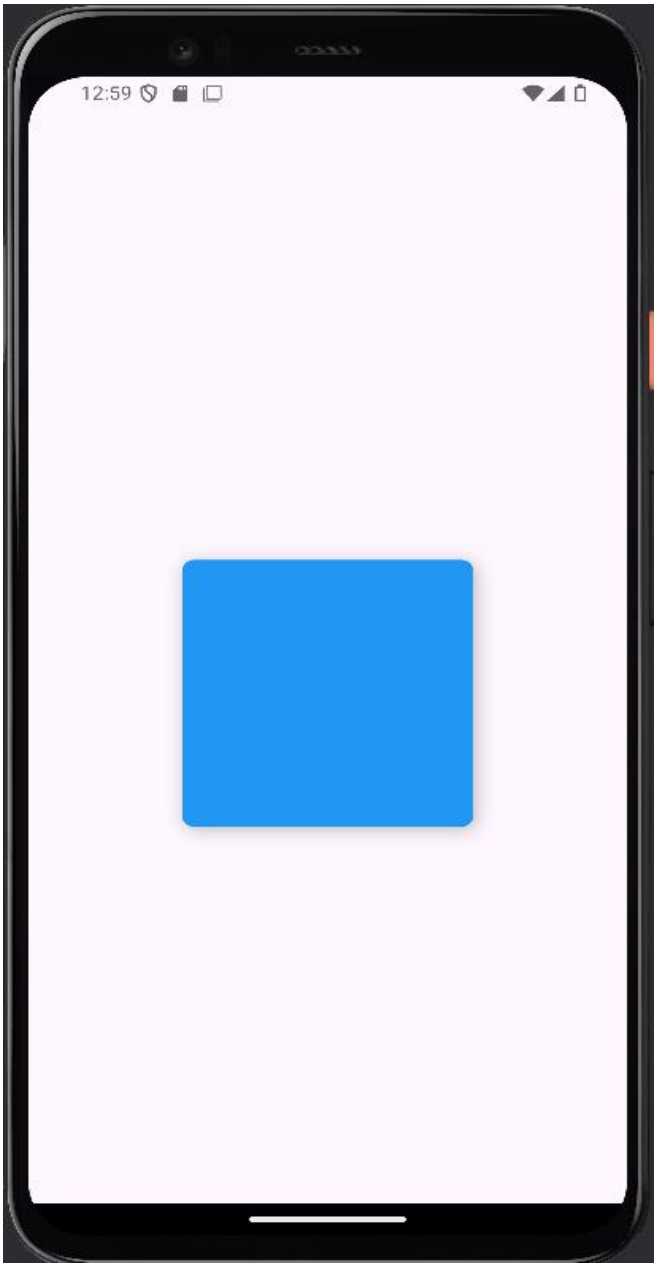
Container Widget

```
import 'package:flutter/material.dart';
void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
```

```
@override
Widget build(BuildContext context) {
  return MaterialApp(
    home: Abc(),
  );
 }
}

class Abc extends StatelessWidget {
 const Abc({super.key});

 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: Text('Container Widget Example')),
    body: Center(
      child: Container(
        width: 200,
        height: 200,
        padding: EdgeInsets.all(16),
        margin: EdgeInsets.all(16),
        decoration: BoxDecoration(
          color: Colors.blue,
          borderRadius: BorderRadius.circular(8),
```

```
          boxShadow: [
            BoxShadow(
              color: Colors.black26,
              blurRadius: 10,
              offset: Offset(2, 2),
            ),
          ],
        ),
        child: Center(
          child: Text(
            'Container',
            style: TextStyle(color: Colors.white, fontSize: 24),
          ),
        ),
      ),
    ),
  );
 }
}
```

OUTPUT



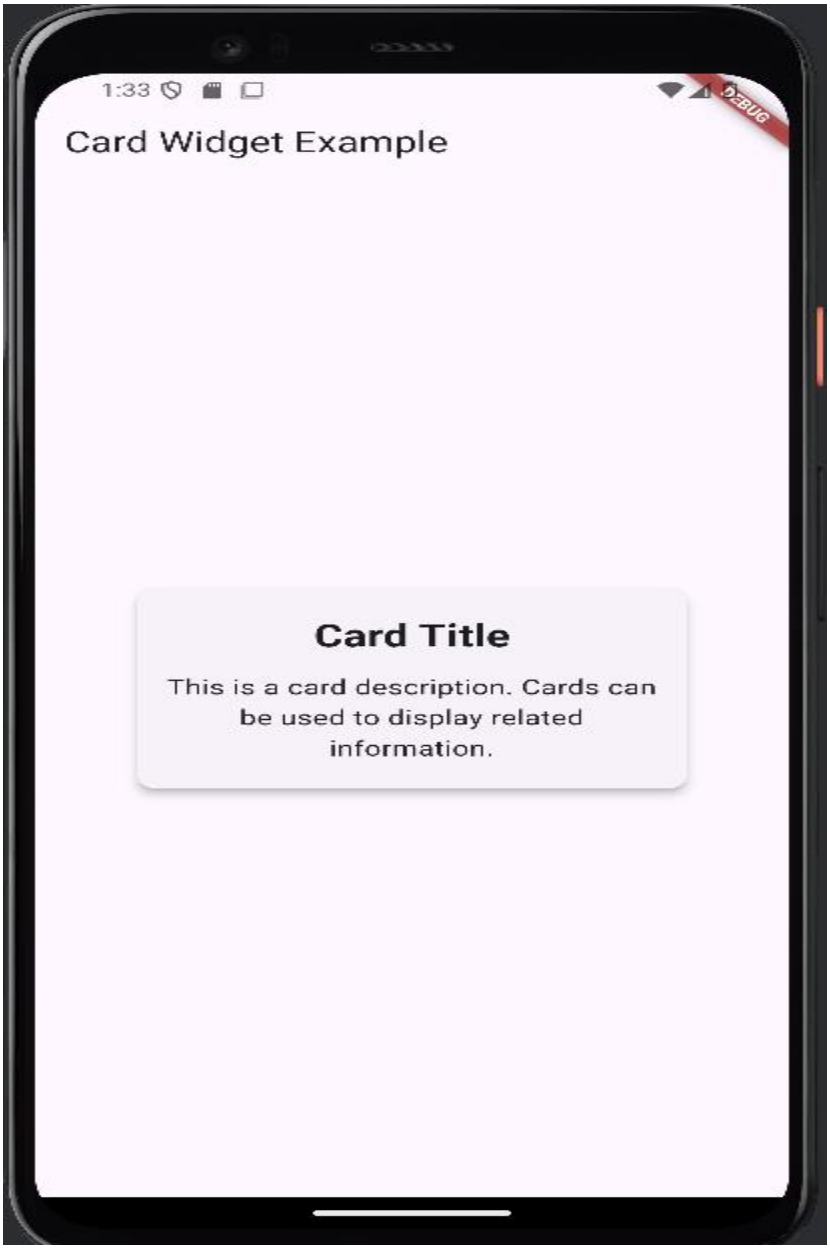
Card Widget

```
import 'package:flutter/material.dart';
void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     home: Abc();
   );
 }
}
class Abc extends StatelessWidget {
 const Abc({super.key});
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(title: Text('Card Widget Example')),
     body: Center(
       child: Card(
         elevation: 5, // The shadow effect of the card
         shape: RoundedRectangleBorder(
```

```
      borderRadius: BorderRadius.circular(10), // Rounded corners
    ),
    child: Container(
      width: 300,
      height: 150,
      padding: EdgeInsets.all(16),
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Text(
            'Card Title',
            style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
          ),
          SizedBox(height: 10),
          Text(
            'This is a card description. Cards can be used to display related information.',
            textAlign: TextAlign.center,
            style: TextStyle(fontSize: 16),
          ),
        ],
      ),
    ),
  ),
),
```

```
  );
 }
}
```

OUTPUT


1:33
DEBUG
Card Widget Example
Card Title
This is a card description. Cards can be used to display related information.

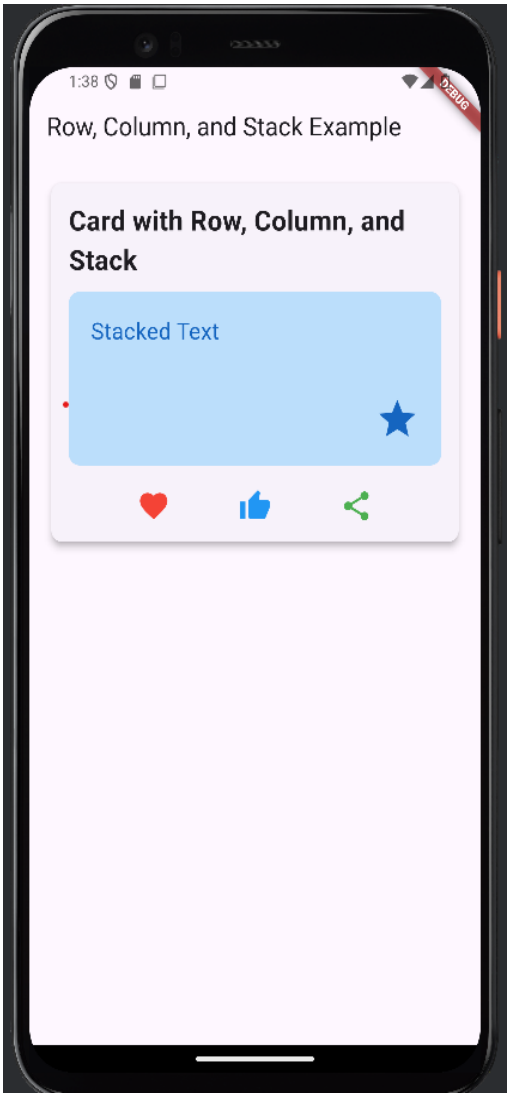
2 b) Implement different layout structures using Row, Column, and Stack widgets.

```
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('Row, Column, and Stack Example')),
      body: Padding(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          children: [
            Card(
              elevation: 5,
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(10),
              ),
              child: Container(
                width: double.infinity,
                padding: EdgeInsets.all(16),
```

```
child: Column(
 crossAxisAlignment: CrossAxisAlignment.start,
 children: [
  Text(
   'Card with Row, Column, and Stack',
   style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
  ),
  SizedBox(height: 10),
  Stack(
   children: [
    Container(
     width: double.infinity,
     height: 150,
     decoration: BoxDecoration(
      color: Colors.blue[100],
      borderRadius: BorderRadius.circular(10),
     ),
    ),
    Positioned(
     top: 20,
     left: 20,
     child: Text(
      'Stacked Text',
      style: TextStyle(fontSize: 20, color: Colors.blue[800]),),),
```

```
    Positioned(
      bottom: 20,
      right: 20,
      child: Icon(Icons.star, size: 40, color: Colors.blue[800]),
    ),
  ],
),
SizedBox(height: 20),
Row(
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,
  children: [
    Icon(Icons.favorite, size: 30, color: Colors.red),
    Icon(Icons.thumb_up, size: 30, color: Colors.blue),
    Icon(Icons.share, size: 30, color: Colors.green),
  ],),],),),),],),),),);}}
```

OUTPUT


1:38
DEBUG
Row, Column, and Stack Example
Card with Row, Column, and Stack
Stacked Text

VIVA QUESTIONS

S.No	Question	CO	Blooms Taxonomy
1	What is the primary purpose of the Text widget in Flutter?	CO2	Remember
2	Which widget is used to display images from the network or assets?	CO2	Remember
3	What is a Container widget typically used for?	CO2	Remember
4	Explain the difference between Image.asset() and Image.network() constructors. When would you use each?	CO2	Understand
5	Describe how you would add padding to a Text widget.	CO2	Understand
6	How does the Container widget help in customizing the appearance and positioning of its child?	CO2	Understand
7	Write a simple Flutter code snippet that displays "Hello, Flutter!" in red color with a font size of 20.	CO2	Apply
8	Demonstrate how to display an image from your project's assets folder.	CO2	Apply
9	Discuss the concept of "implicit sizing" in Container. How does a Container determine its size if width and height are not explicitly set?	CO2	Analyze
10	When would you consider using Decorated Box instead of Box Decoration directly within a Container?	CO2	Analyze
11	What is the primary purpose of the Stack widget?	CO2	Remember
12	List any two main axes of a Row widget?	CO2	Remember
13	What property is used to align children along the main axis in Row or Column?	CO2	Remember
14	Explain the difference between main Axis Alignment and cross Axis Alignment for Row and Column widgets.	CO2	Understand
15	Describe how Stack widgets allow children to overlap.	CO2	Understand
16	Demonstrate how to place a small icon on top of a larger Image using a Stack.	CO2	Apply
17	Implement a Row where one child takes up all available space, and the others take their natural size.	CO2	Apply
18	Analyze the behavior of Row and Column when their children exceed the available space. What is the typical visual indication of an overflow, and how can it be resolved?	CO2	Analyze

19	Compare and contrast the Stack widget with Row and Column. In what scenarios would Stack be the preferred layout widget?	CO2	Analyze
20	Assess the importance of Expanded and Flexible for creating responsive layouts in Flutter.	CO2	Evaluate

Note :Each experiment should contain Minimum 20 Viva Questions

Experiment 3:

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

```
import 'package:flutter/material.dart';
void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     home: ResponsiveLayout(),
   );
 }
}
class ResponsiveLayout extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(title: Text('Responsive UI Example')),
     body: Column(
       children: [
         Expanded(
           flex: 2,
```

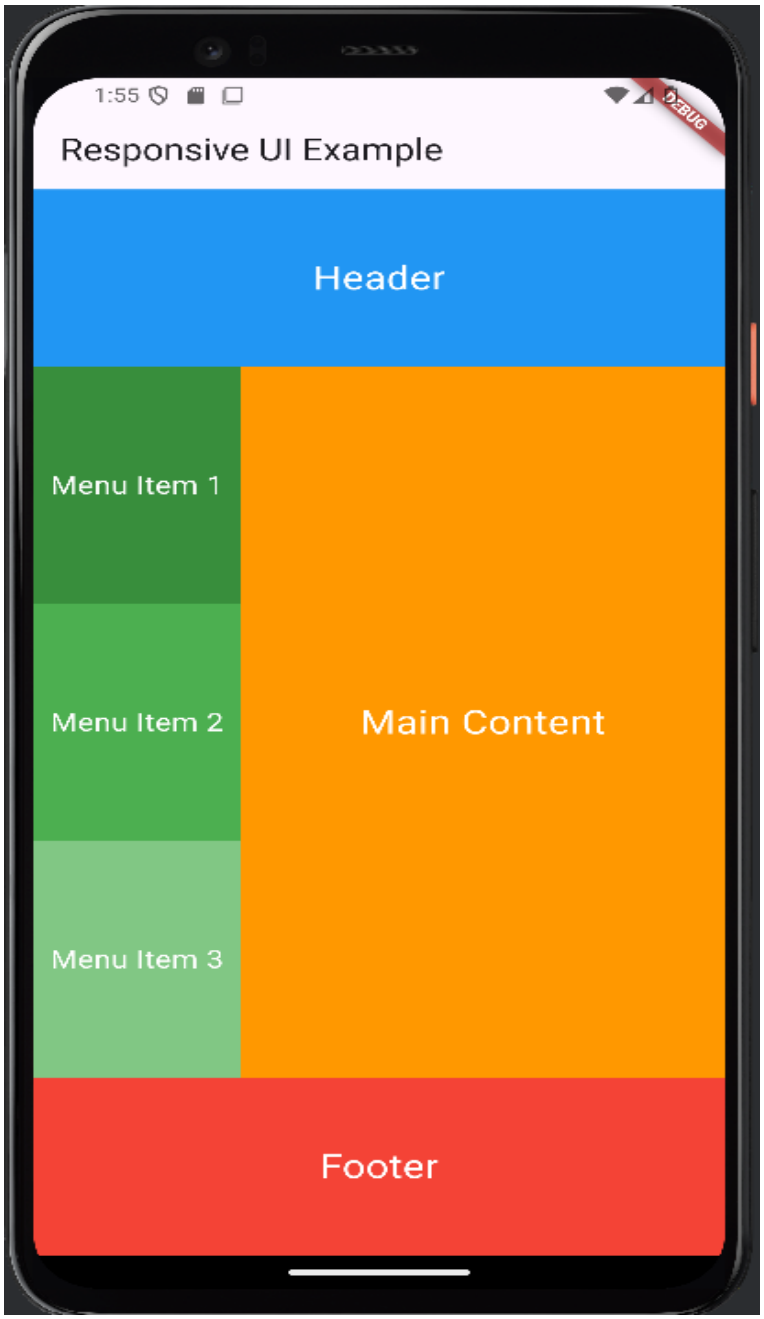
```
    child: Container(
      color: Colors.blue,
      child: Center(
        child: Text(
          'Header',
          style: TextStyle(color: Colors.white, fontSize: 24),
        ),
      ),
    ),
  ),
  Expanded(
    flex: 8,
    child: Row(
      children: [
        Expanded(
          flex: 3,
          child: Container(
            color: Colors.green,
            child: Column(
              children: [
                Expanded(
                  flex: 1,
                  child: Container(
                    color: Colors.green[700],
```

```
    child: Center(
     child: Text(
       'Menu Item 1',
       style: TextStyle(color: Colors.white, fontSize: 18),
     ),
    ),
   ),
  ),
  Expanded(
   flex: 1,
   child: Container(
    color: Colors.green[500],
    child: Center(
     child: Text(
       'Menu Item 2',
       style: TextStyle(color:Colors.white,fontSize: 18),
     ),
    ),
   ),
  ),
  Expanded(
   flex: 1,
   child: Container(
    color: Colors.green[300],
```

```
            child: Center(
              child: Text(
                'Menu Item 3',
                style: TextStyle(color: Colors.white, fontSize: 18),
              ),
            ),
          ),
        ),
      ],
    ),
  ),
),
Expanded(
  flex: 7,
  child: Container(
    color: Colors.orange,
    child: Center(
      child: Text(
        'Main Content',
        style: TextStyle(color: Colors.white, fontSize: 24),
      ),
    ),
  ),
),
```

```
      ],
    ),
  ),
  Expanded(
    flex: 2,
    child: Container(
      color: Colors.red,
      child: Center(
        child: Text(
          'Footer',
          style: TextStyle(color: Colors.white, fontSize: 24),
        ),
      ),
    ),],),);}}
```

OUTPUT

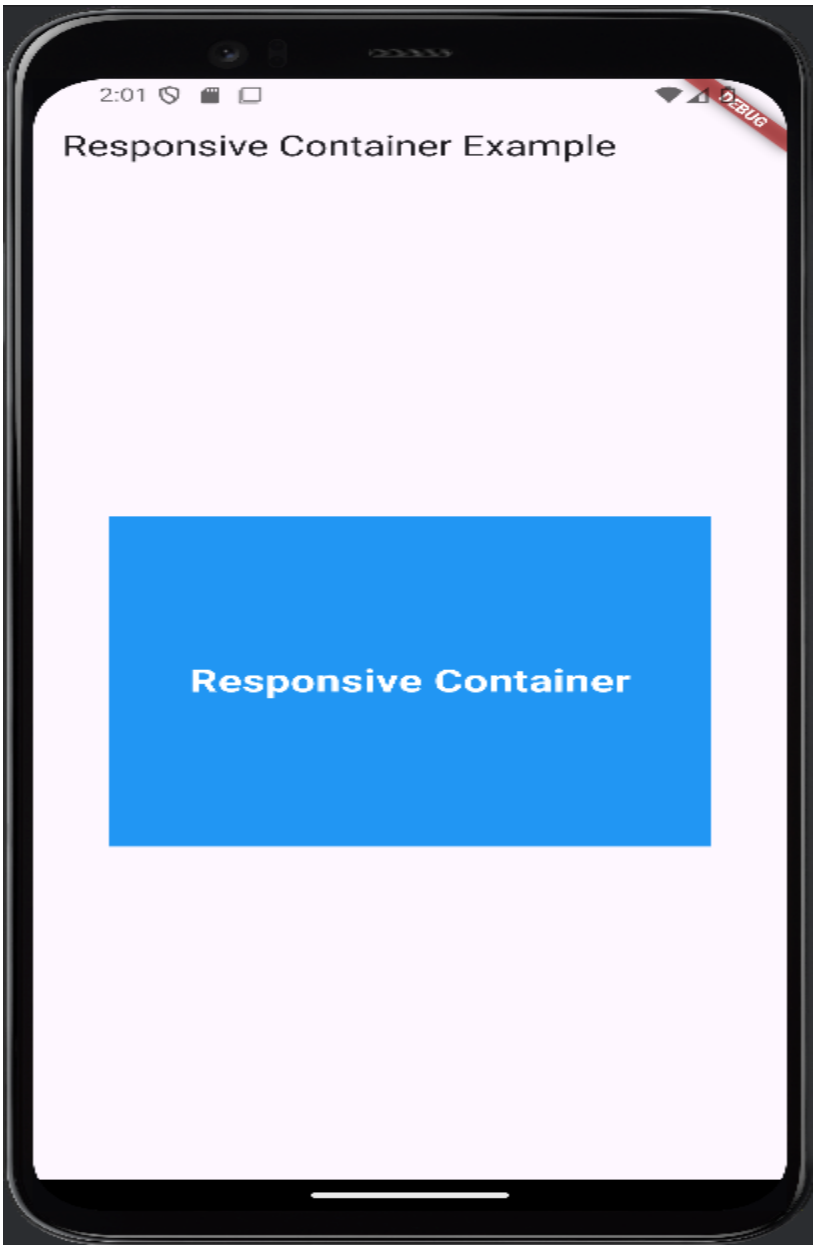

1:55
DEBUG
Responsive UI Example
Header
Menu Item 1
Menu Item 2
Menu Item 3
Main Content
Footer

3 b) Implement media queries and breakpoints for responsiveness.

```
import 'package:flutter/material.dart';
void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     home: ResponsiveContainer(),
   );
 }
}
class ResponsiveContainer extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   // Get the screen size using MediaQuery
   final screenSize = MediaQuery.of(context).size;
   final screenWidth = screenSize.width;
   final screenHeight = screenSize.height;
   // Determine container size based on screen width
   double containerWidth;
   double containerHeight;
   if (screenWidth < 600) {
```

```
  // Small screen size (e.g., mobile phones)
  containerWidth = screenWidth * 0.8; // 80% of screen width
  containerHeight = screenHeight * 0.3; // 30% of screen height
 } else {
  // Larger screen size (e.g., tablets or desktops)
  containerWidth = screenWidth * 0.5; // 50% of screen width
  containerHeight = screenHeight * 0.4; // 40% of screen height
 }
 return Scaffold(
  appBar: AppBar(title: Text('Responsive Container Example')),
  body: Center(
   child: Container(
    width: containerWidth,
    height: containerHeight,
    color: Colors.blue,
    child: Center(
     child: Text(
      'Responsive Container',
      style: TextStyle(
       color: Colors.white,
       fontSize: 24,
       fontWeight: FontWeight.bold,
      ),),),),),);}}
```

OUTPUT


2:01
DEBUG
Responsive Container Example
Responsive Container

VIVA QUESTIONS

S.No	Question	CO	Blooms Taxonomy
1	What is "responsive UI design"?	CO3	Remember
2	Name one common Flutter widget that helps in creating flexible layouts.	CO3	Remember
3	What are "breakpoints" in responsive design?	CO3	Remember
4	Explain *why* responsive UI design is important for modern applications.	CO3	Understand
5	Describe how Flex widgets (like Row and Column) contribute to responsiveness.	CO3	Understand
6	How does the Layout Builder widget differ from Media Query in providing layout information?	CO3	Understand
7	Demonstrate how to display different content (e.g., an image vs. just text) based on the current screen orientation.	CO3	Apply
8	Implement a layout where a sidebar is visible on large screens but hidden on small screens.	CO3	Apply
9	Discuss how the Aspect Ratio widget contributes to maintaining the visual integrity of content (like images or videos) across different screen sizes.	CO3	Analyze
10	Examine a scenario where a Wrap widget might be a more suitable choice for responsiveness than a Row or Column.	CO3	Analyze
11	Assess the effectiveness of using only Expanded and Flexible for responsiveness without considering other layout widgets or techniques.	CO3	Evaluate
12	Given a specific design requirement, justify your choice of using Media Query or Layout Builder to implement responsiveness.	CO3	Evaluate
13	What is a "media query" in the context of Flutter?	CO3	Remember
14	How do you access Media Query Data in Flutter?	CO3	Remember
15	What does Media Query.of(context).size. width return?	CO3	Remember
16	Explain how Media Query allows your Flutter application to adapt its UI based on device characteristics.	CO3	Understand
17	Describe the difference between Media Query. of(context).size. height and Media Query. of(context).size. shortest Side. When would you use each?	CO3	Understand
18	Discuss the implications of constantly re-calculating	CO3	Analyze

	MediaQuery values within deeply nested widgets. Are there performance considerations?		
19	Compare and contrast the approach of using MediaQuery directly with a dedicated responsive package (e.g., responsive_framework). What are the pros and cons of each?	CO3	Analyze
20	Show how you would change the padding of a Container based on the device's pixel density.	CO3	Apply

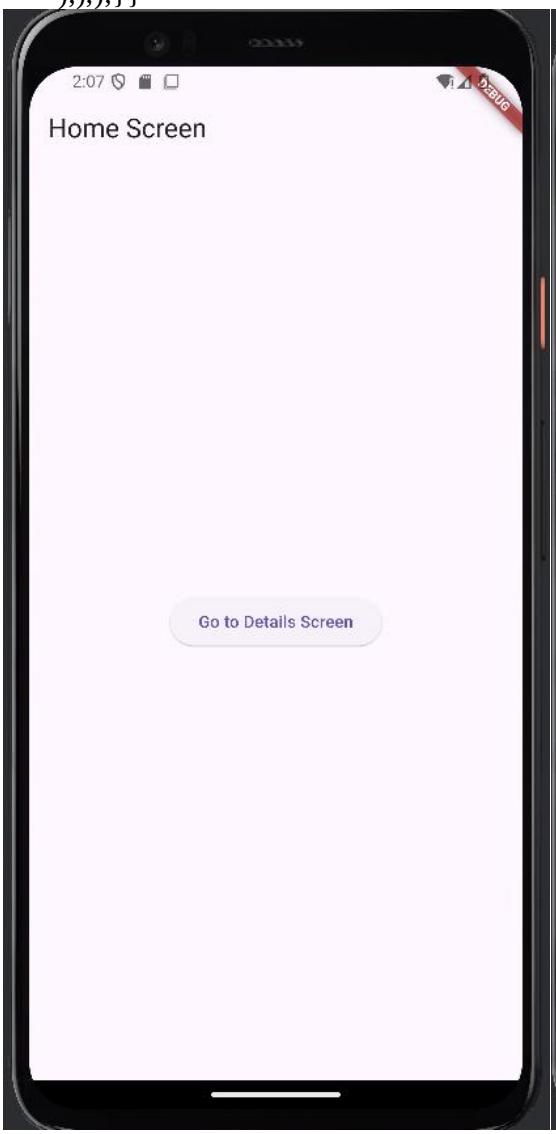
Note :Each experiment should contain Minimum 20 Viva Questions

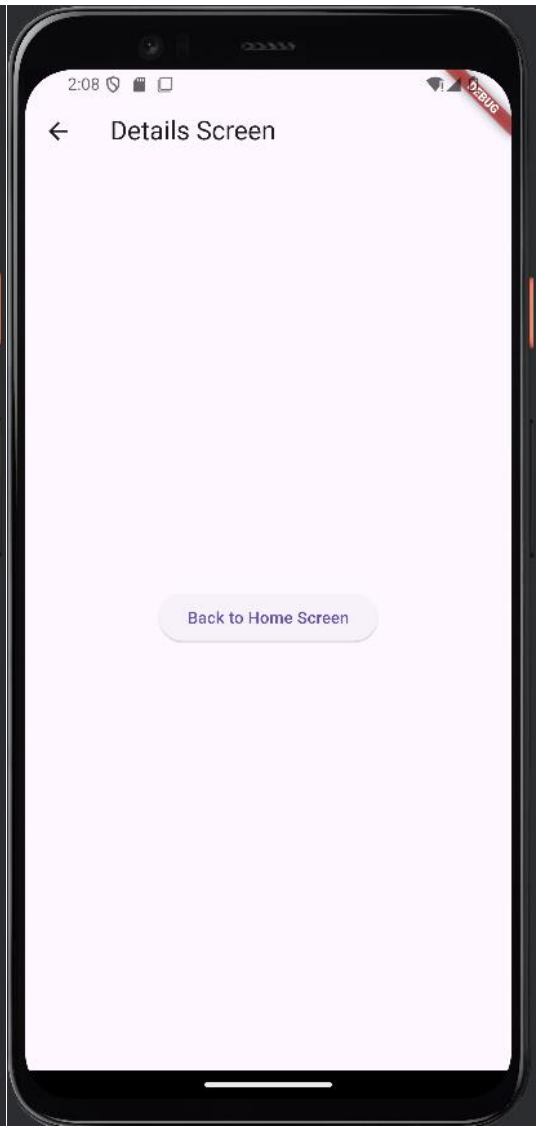
Experiment 4:

AIM: a) Set up navigation between different screens using Navigator.

```
import 'package:flutter/material.dart';
void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Navigation Example',
     theme: ThemeData(
       primarySwatch: Colors.blue,
     ),
     home: HomeScreen(),
   );
 }
}

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

class DetailsScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
```

```
return Scaffold(
  appBar: AppBar(
    title: Text('Details Screen'),
  ),
  body: Center(
    child: ElevatedButton(
      onPressed: () {
        Navigator.pop(context);
      },
      child: Text('Back to Home Screen'),
    ),),);}}
```

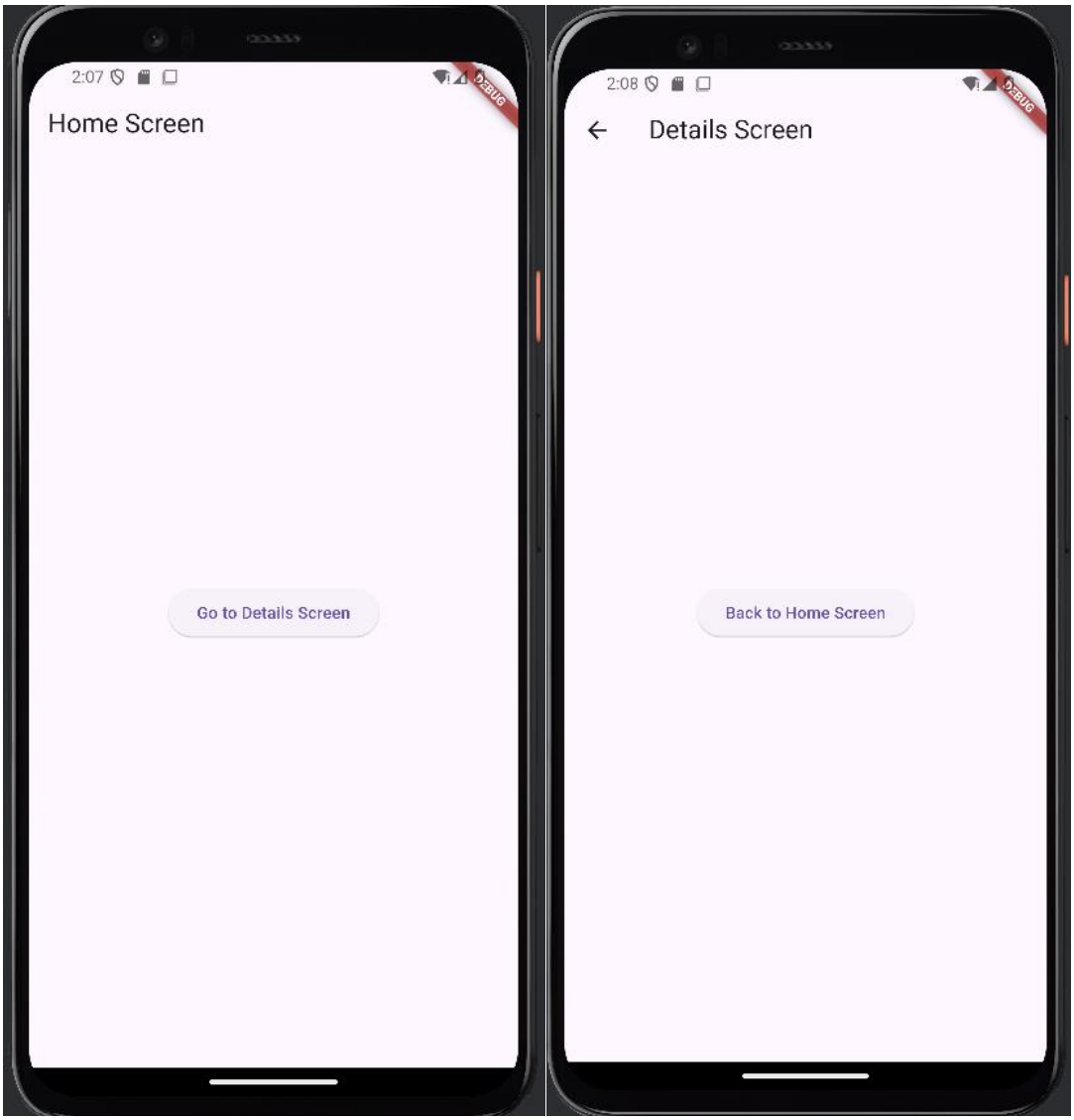




4 b) AIM:Implement navigation with named routes.

```
import 'package:flutter/material.dart';
void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Named Routes Example',
     theme: ThemeData(
       primarySwatch: Colors.blue,
     ),
     initialRoute: '/',
     routes: {
       '/': (context) => HomeScreen(),
       '/details': (context) => DetailsScreen(),
     },
   );
 }
}

class HomeScreen extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(
       title: Text('Home Screen'),
     ),
     body: Center(
       child: ElevatedButton(
         onPressed: () {
           Navigator.pushNamed(context, '/details');
         },
         child: Text('Go to Details Screen'),
       ),
     ),
   );
 }
}

class DetailsScreen extends StatelessWidget {
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Details Screen'),
    ),
    body: Center(
      child: ElevatedButton(
        onPressed: () {
          Navigator.pop(context);
        },
        child: Text('Back to Home Screen'),),),);}}
```



VIVA QUESTIONS

S.No	Question	CO	Blooms Taxonomy
1	Which method is used to remove the current screen from the navigation stack?	CO4	Remember
2	What is a "route" in Flutter navigation?	CO4	Remember
3	What is the difference between push() and pop()?	CO4	Remember
4	Explain the concept of a "navigation stack" in Flutter. How does push() and pop() interact with it?	CO4	Understand
5	Describe the lifecycle events a screen goes through when it's pushed onto or popped from the navigation stack.	CO4	Understand
6	How does Navigator.of(context) obtain the correct Navigator instance?	CO4	Understand
7	Write a simple Flutter code snippet that navigates from ScreenA to ScreenB when a button is pressed.	CO4	Apply
8	Demonstrate how to return data from ScreenB back to ScreenA when ScreenB is popped.	CO4	Apply
9	Implement a scenario where clicking a "Back" button customizes the Navigator.pop() behavior to also show a dialog.	CO4	Apply
10	Analyze the difference between Navigator.push() and Navigator.pushReplacement(). When would you choose one over the other?	CO4	Analyze
11	Discuss the potential issues that could arise from not calling Navigator.pop() when a screen is no longer needed (e.g., memory leaks, unexpected back button behavior).	CO4	Analyze
12	Examine a scenario where you might need to pop multiple screens from the stack simultaneously. How would you achieve this?	CO4	Analyze
13	Analyze the impact of navigation on the widget tree and the rebuild process of widgets.	CO4	Analyze
14	What is a "named route" in Flutter?	CO4	Remember
15	Which property of MaterialApp is used to define named routes?	CO4	Remember
16	Explain the advantages of using named routes over direct MaterialPageRoute construction.	CO4	Understand
17	Describe how onGenerateRoute callback works in conjunction with named routes. When would you use it?	CO4	Understand
18	How does the arguments property of	CO4	Understand

	Navigator.pushNamed() facilitate passing data?		
19	Analyze the benefits of using named routes for large-scale applications with many screens.	CO4	Analyze
20	Discuss the trade-offs between directly using routes map in Material App and using on Generate Route for named route management.	CO4	Analyze

Note :Each experiment should contain Minimum 20 Viva Questions

Experiment 5:

5a) AIM:Learn about stateful and stateless widgets.

Stateless Widgets StatelessWidget is used when the part of the UI you are building does not change. Stateless widgets are immutable, meaning their properties cannot change once they are created. They are ideal for static content that doesn't depend on any dynamic data.

StatelessWidget is used when the part of the UI you are building does not change. Stateless widgets are immutable, meaning their properties cannot change once they are created. They are ideal for static content that doesn't depend on any dynamic data.

```
class Abc extends StatelessWidget {

 const Abc({super.key});

 @override

 Widget build(BuildContext context) {

   return const Placeholder();

 }

}
```

Stateful Widgets

StatefulWidget is used when the part of the UI you are building can change dynamically. Stateful widgets are mutable and can maintain state that changes over time. This is useful for interactive elements where the appearance or behavior of the widget depends on user input or other factors.

A StatefulWidget consists of two classes:

1. The StatefulWidget class: This is immutable and creates an instance of the State class.
2. The State class: This is where the mutable state is maintained and updated.

```
class Abc extends StatefulWidget {

 const Abc({super.key});
```

```
 @override
 State<Abc> createState() => _AbcState();
}

class _AbcState extends State<Abc> {
 @override
 Widget build(BuildContext context) {
   return const Placeholder();
 }
}
```

For better understanding of StatefulWidget, take a look at this below example

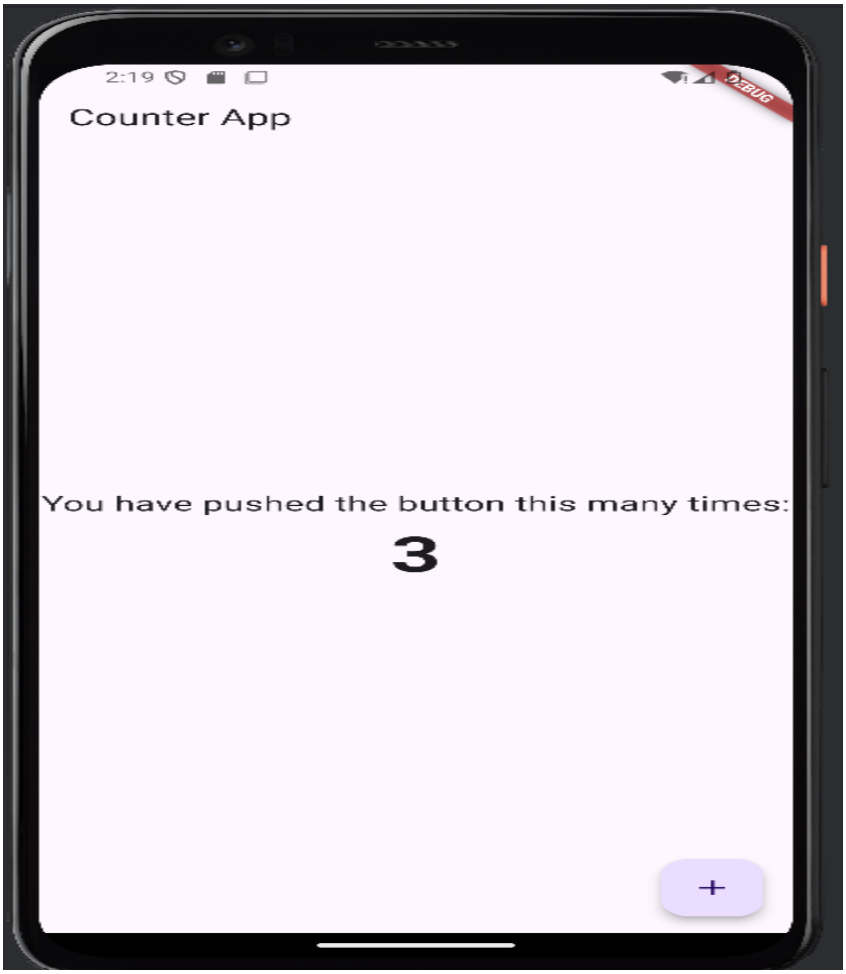
```
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
```

```
    title: 'StatefulWidget Example',
    theme: ThemeData(
      primarySwatch: Colors.blue,
    ),
    home: CounterScreen(),
  );
 }
}

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

class _CounterScreenState extends State<CounterScreen> {
 int _counter = 0; // State variable to keep track of the counter value

 void _incrementCounter() {
  setState(() {
    _counter++; // Update the state
  });
 }

 @override
```

```
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Counter App'),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Text(
            'You have pushed the button this many times:',
            style: TextStyle(fontSize: 20),
          ),
          Text(
            '$_counter', // Display the current counter value
            style: TextStyle(fontSize: 48, fontWeight: FontWeight.bold),
          ),
        ],
      ),
    ),
    floatingActionButton: FloatingActionButton(
      onPressed: _incrementCounter, // Increment counter when button is pressed
      tooltip: 'Increment',
      child: Icon(Icons.add),
```

),); }}

OUTPUT


2:19
DEBUG
Counter App
You have pushed the button this many times:
3
+

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

State Management Techniques:

- **Local State Management: Managing state within a single widget or a small part of the widget tree.**

```
class CounterScreen extends StatefulWidget {
 @override
 _CounterScreenState createState() => _CounterScreenState();
}
class _CounterScreenState extends State<CounterScreen> {
 int _counter = 0;
 void _incrementCounter() {
   setState(() {
     _counter++;
   });
 }
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     appBar: AppBar(title: Text('Counter')),
     body: Center(
       child: Column(
```

```
mainAxisAlignment: MainAxisAlignment.center,
children: <Widget>[
  Text('Count: $_counter'),
  ElevatedButton(onPressed: _incrementCounter, child: Text('Increment')),],),),);}}
```

- Global State Management: Managing state that needs to be accessed and updated across multiple widgets or the entire application.

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

// Define the CounterModel with ChangeNotifier
class CounterModel with ChangeNotifier {
 int _counter = 0;

 int get counter => _counter;

 void increment() {
  _counter++;
  notifyListeners(); // Notify listeners to rebuild
 }
```

```
}

void main() {
 runApp(
   ChangeNotifierProvider(
     create: (context) => CounterModel(),
     child: MyApp(),
   ),
 );
}

class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Provider Example',
     home: CounterScreen(),
   );
 }
}

class CounterScreen extends StatelessWidget {
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: Text('Provider Example')),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          // Use Consumer to listen to changes in CounterModel
          Consumer<CounterModel>(
            builder: (context, counterModel, child) {
              return Text(
                'Count: ${counterModel.counter}',
                style: TextStyle(fontSize: 48, fontWeight: FontWeight.bold),
              );
            },
          ),
          SizedBox(height: 20),
          ElevatedButton(
            onPressed: () {
              // Update the state using Provider
              Provider.of<CounterModel>(context, listen: false).increment();
```

```
          },
          child: Text('Increment'),
        ),
      ],
    ),
  ),
);
}
}
```

To add the provider package to your Flutter project, follow these steps:

1. Add provider to pubspec.yaml

1. Open your Flutter project in your preferred IDE or text editor.
2. Locate the pubspec.yaml file in the root directory of your project.

Under the dependencies section, add provider along with the version number. As of the last update, the latest version is 6.0.0, but you should check pub.dev for the latest version.
yaml
Copy code

```
Dependencies:
  flutter:
    sdk: flutter
```

```
# The following adds the Cupertino Icons font to your application.
# Use with the CupertinoIcons class for iOS style icons.
cupertino_icons: ^1.0.6
provider: ^6.0.0  #Add This
```

3. Save the pubspec.yaml file.

2. Install the Package

To install the new package, run the following command in the terminal

flutter pub get

This command will fetch the provider package and add it to your project.

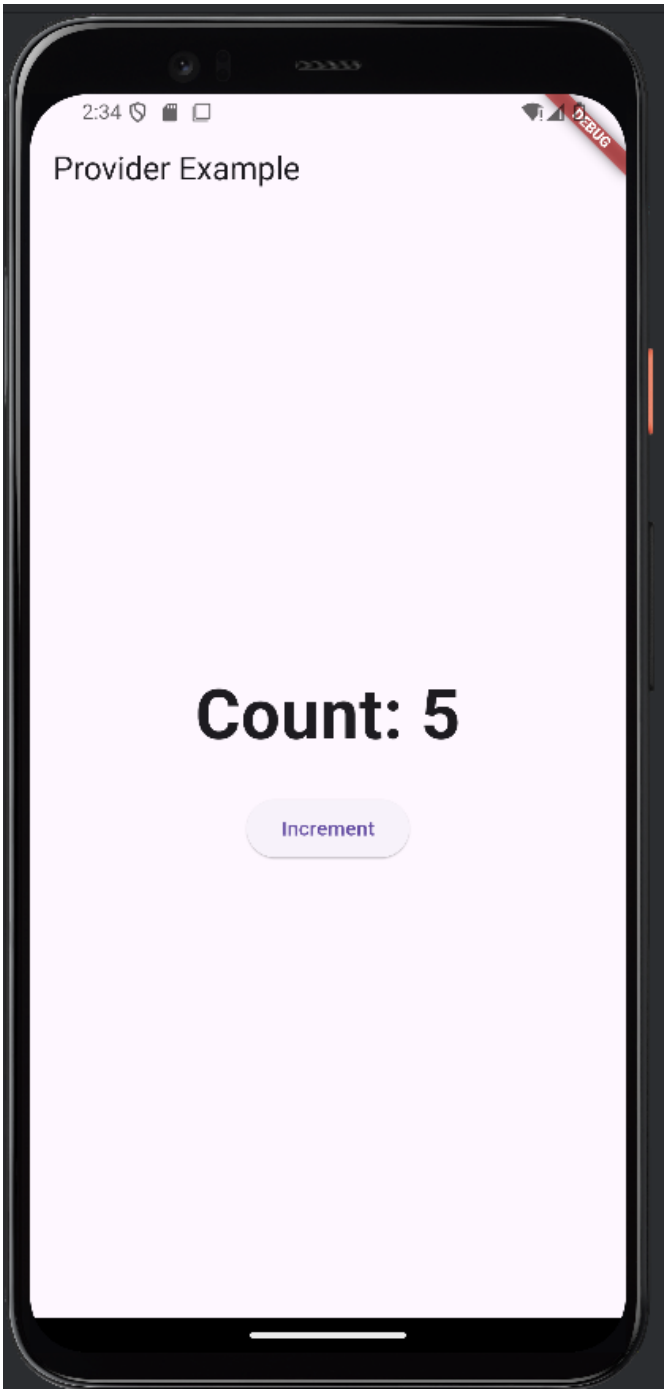
3. Import the Provider Package

In your Dart files where you need to use Provider, import it at the top of the file:

dart

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

Output for State Management using Provider:



VIVA QUESTIONS

S.No	Question	CO	Blooms Taxonomy
1	What is the fundamental difference between a Stateless Widget and a Stateful Widget?	CO5	Remember
2	Which widget type is immutable?	CO5	Remember
3	What is the primary purpose of the build method in a widget?	CO5	Remember
4	Can a Stateless Widget change its appearance over time?	CO5	Remember
5	Explain why a Stateless Widget is suitable for displaying static content. Provide an example.	CO5	Understand
6	Describe the lifecycle of a Stateful Widget. When is create State() called?	CO5	Understand
7	Write a simple Flutter code snippet for a Stateless Widget that displays a fixed text.	CO5	Apply
8	Create a Stateful Widget that has a counter that increments when a button is pressed.	CO5	Apply
9	Demonstrate how to pass data from a parent Stateless Widget to a child Stateless Widget.	CO5	Apply
10	Analyze a given Flutter UI design. Identify which parts should ideally be Stateless Widgets and which should be Stateful Widgets, justifying your choices.	CO5	Analyze
11	Compare and contrast the performance implications of rebuilding a Stateless Widget versus a Stateful Widget.	CO5	Analyze
12	Discuss the problems that might arise if you accidentally try to manage mutable state within a Stateless Widget.	CO5	Analyze
13	Evaluate the trade-offs of breaking down a large Stateful Widget into smaller, more manageable Stateless Widgets.	CO5	Evaluate
14	Assess the readability and maintainability of code that uses deeply nested Stateful Widgets without proper state management.	CO5	Evaluate
15	What is setState() primarily used for in Flutter?	CO5	Remember
16	What package needs to be added to pubspec.yaml to use Provider?	CO5	Remember
17	How do you make a Change Notifier available to widgets using Provider?	CO5	Remember
18	Implement a simple data model using Change Notifier to hold a user's name.	CO5	Apply

19	Show how to provide this Change Notifier to the widget tree using Change Notifier Provider.	CO5	Apply
20	Demonstrate how to update the user's name from a widget, using Provider.of<MyModel>(context, listen: false).	CO5	Apply

Note :Each experiment should contain Minimum 20 Viva Questions

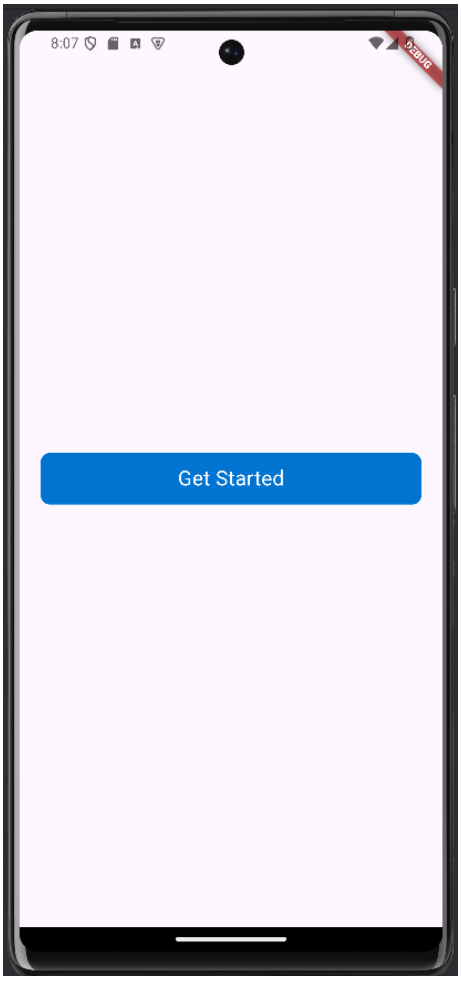
Experiment 6:

6. a)AIM: Create custom widgets for specific UI elements.

```
import 'package:flutter/material.dart';
void main() {
 runApp(Abc());
}
class Abc extends StatelessWidget {
 const Abc({super.key});
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     home: Def(),
   );
 }
}
class Def extends StatelessWidget {
 const Def({super.key});
 @override
 Widget build(BuildContext context) {
   return Scaffold(
     body: Column(
           mainAxisAlignment: MainAxisAlignment.center,
           crossAxisAlignment: CrossAxisAlignment.center,
       children: [
```

```
GestureDetector(
  onTap: () {},
  child: Center(
    child: Container(
      decoration: BoxDecoration(
        color: Color(0xff0174d2),
        borderRadius: BorderRadius.circular(9),
      ),
      height: 50,
      width: MediaQuery.of(context).size.width * 0.9,
      child: Center(
        child: Text(
          "Get Started",
          style: TextStyle(color: Colors.white, fontSize: 20),
        ),),),),),],),);}}
```

OUTPUT



6 b) Apply styling using themes and custom styles.

```
import 'package:flutter/material.dart';
void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Flutter Demo',
     theme: ThemeData.dark(),
     home: MyHomePage(),
   );
 }
}
class MyHomePage extends StatefulWidget {
 @override
 _MyHomePageState createState() => _MyHomePageState();
}

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

 void _incrementCounter() {
```

```
 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(
          'You have pushed the button this many times:',
        ),
        Text(
          '$_counter',
          style: Theme.of(context).textTheme.headlineMedium,),],),),
  floatingActionButton: FloatingActionButton(
    onPressed: _incrementCounter,
    tooltip: 'Increment',
    child: Icon(Icons.add),     ),);}}
```

OUTPUT


8:05
DEBUG
Counter App
You have pushed the button this many times:
4
+

VIVA QUESTIONS

S.No	Question	CO	Blooms Taxonomy
1	What is a "custom widget" in Flutter?	CO6	Remember
2	Which method must every widget (custom or built-in) implement?	CO6	Remember
3	What is the purpose of the child or children property in many widgets?	CO6	Remember
4	Explain *why* you would create a custom widget instead of just composing existing widgets directly in your main build method.	CO6	Understand
5	Describe the process of making a custom widget reusable across different parts of your application.	CO6	Understand
6	How does parameterizing a custom widget (passing arguments to its constructor) make it more flexible?	CO6	Understand
7	Explain the concept of "widget composition" and how it's fundamental to creating custom widgets.	CO6	Understand
8	Write a simple Flutter code snippet for a custom Stateless Widget that takes a String title and displays it.	CO6	Apply
9	Create a custom Stateful Widget that represents a toggle button, managing its is On state internally.	CO6	Apply
10	Demonstrate how to add a Gesture Detector to a custom widget to make it tappable.	CO6	Apply
11	Show how you would encapsulate a Container with specific padding, margin, and decoration into a reusable custom widget.	CO6	Apply
12	Analyze a complex UI design. Break it down into potential custom widget candidates, explaining your rationale for each.	CO6	Analyze
13	Compare and contrast the benefits of a highly generalized custom widget (many parameters) versus several specific custom widgets (fewer parameters).	CO6	Analyze
14	Discuss the challenges of managing state within a custom Stateful Widget that interacts with a parent widget's state.	CO6	Analyze
15	Examine a scenario where a custom widget might need to expose a callback function to its parent. How would you implement this?	CO6	Analyze
16	Evaluate the trade-offs between creating a custom widget from scratch versus using an existing package that provides similar functionality.	CO6	Evaluate
17	Assess the reusability of a given custom widget. What	CO6	Evaluate

	improvements could be made to enhance its reusability?		
18	What is a "Theme" in Flutter?	CO6	Remember
19	Which widget is typically used at the root of a Flutter application to define a theme?	CO6	Remember
20	Describe the concept of "theme inheritance" in Flutter. How does it work?	CO6	Understand

Note :Each experiment should contain Minimum 20 Viva Questions

Experiment 7:

AIM: a) Design a form with various input fields.

b) Implement form validation and error handling.

```
import 'package:flutter/material.dart';
void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     home: MyForm(),);
}}
class MyForm extends StatefulWidget {
 @override
 _MyFormState createState() => _MyFormState();
}
class _MyFormState extends State<MyForm> {
 final _formKey = GlobalKey<FormState>();
 final _nameController = TextEditingController();
 final _emailController = TextEditingController();
 final _passwordController = TextEditingController();
 @override
 Widget build(BuildContext context) {
```

```
return Scaffold(
  appBar: AppBar(
    title: Text('Form Example'),
  ),
  body: Padding(
    padding: const EdgeInsets.all(16.0),
    child: Form(
      key: _formKey,
      child: Column(
        children: <Widget>[
          TextFormField(
            controller: _nameController,
            decoration: InputDecoration(labelText: 'Name'),
            validator: (value) {
              if (value == null || value.isEmpty) {
                return 'Please enter your name';
              }
              return null;
            },
          ),
          TextFormField(
            controller: _emailController,
            decoration: InputDecoration(labelText: 'Email'),
            keyboardType: TextInputType.emailAddress,
```

```
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter your email';
    } else if (!RegExp(r'^[^@]+@[^@]+\.[^@]+').hasMatch(value)) {
      return 'Please enter a valid email address';
    }
    return null;
  },
),
TextFormField(
  controller: _passwordController,
  decoration: InputDecoration(labelText: 'Password'),
  obscureText: true,
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter your password';
    } else if (value.length < 6) {
      return 'Password must be at least 6 characters long';
    }
    return null;
  },
),
SizedBox(height: 20),
ElevatedButton(
```

```
        onPressed: () {
          if (_formKey.currentState?.validate() ?? false) {
            // If the form is valid, display a snackbar or perform other actions


          }
        },
        child: Text('Submit'),
      ),],),),),);}}
```

OUTPUT

VIVA QUESTIONS

S.No	Question	CO	Blooms Taxonomy
1	What Flutter widget is commonly used to create a text input field?	CO4	Remember
2	How do you get the current value from a TextField?	CO4	Remember
3	What is a Checkbox widget used for?	CO4	Remember
4	Name a widget used to select an item from a dropdown list.	CO4	Remember
5	Which widget allows for multiple-line text input?	CO4	Remember
6	Explain the difference between TextField and TextFormField. When would you choose one over the other for a form?	CO4	Understand
7	Describe how to obtain the value of a Radio button that has been selected.	CO4	Understand
8	Write a Flutter code snippet that creates a TextField for a user's email address.	CO4	Apply
9	Implement a simple form with a TextField for a name and a Checkbox for "Agree to Terms".	CO4	Apply
10	Demonstrate how to pre-fill a TextField with an initial value.	CO4	Apply
11	Show how to create a DropdownButton with a few predefined options (e.g., "Male", "Female", "Other").	CO4	Apply
12	Create a Slider widget that allows a user to select a rating from 1 to 5.	CO4	Apply
13	Compare and contrast the user experience of using a Date Picker vs. manually entering a date into a TextField. When is each appropriate?	CO4	Analyze
14	Discuss the challenges of designing forms for different screen sizes and orientations. How do you ensure usability?	CO4	Analyze
15	Examine the role of Focus Node in form design and its impact on user interaction (e.g., moving between fields).	CO4	Analyze
16	Analyze how a Form widget (the parent) manages the state and validation of its child TextFormFields.	CO4	Analyze
17	Explain the difference between "client-side" and "server-side" validation. Which one is primarily covered by Flutter's built-in validation?	CO4	Understand
18	Describe how Form State.validate() works. When should it typically be called?	CO4	Understand
19	Write a validator function for a TextFormField that	CO4	Apply

	checks if the input is empty.		
20	Implement a validation check for an email field to ensure it contains "@" and ".".	CO4	Apply

1. **Note :Each experiment should contain Minimum 20 Viva Questions**

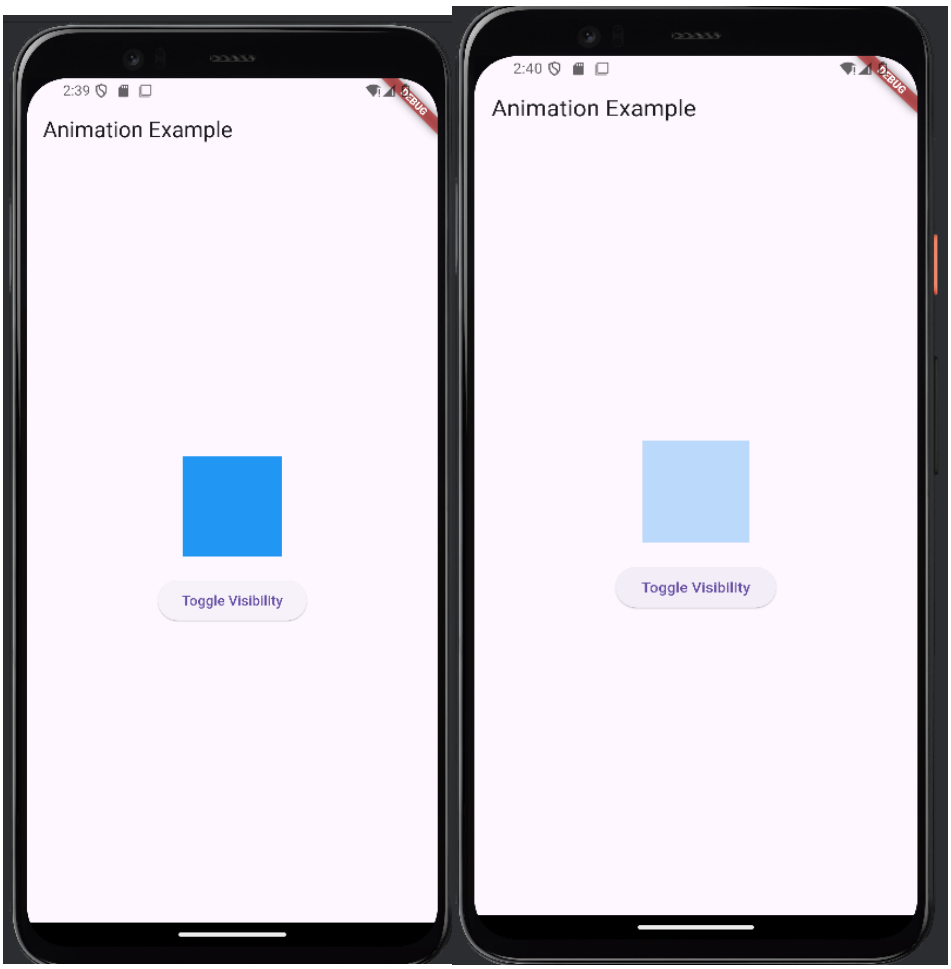
Experiment 8:

8. a) AIM: Add animations to UI elements using Flutter's animation framework.

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

void main() {

 runApp(MyApp());

}

class MyApp extends StatelessWidget {

 @override

 Widget build(BuildContext context) {

   return MaterialApp(

     title: 'Animation Example',

     theme: ThemeData(

       primarySwatch: Colors.blue,

     ),

     home: AnimationScreen(),

   );

 }

}

class AnimationScreen extends StatefulWidget {

 @override

 _AnimationScreenState createState() => _AnimationScreenState();

}

class _AnimationScreenState extends State<AnimationScreen> {

 bool _visible = true;

 void _toggleVisibility() {

   setState(() {
```

```
    _visible = !_visible;
  });
}
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(title: Text('Animation Example')),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          AnimatedOpacity(
            opacity: _visible ? 1.0 : 0.0,
            duration: Duration(seconds: 1),
            child: Container(
              width: 100,
              height: 100,
              color: Colors.blue,
            ),
          ),
          SizedBox(height: 20),
          ElevatedButton(
            onPressed: _toggleVisibility,
            child: Text('Toggle Visibility'),
          ),],),),);}}
```

Output: Fading Animation



8. b) AIM: Experiment with Different Types of Animations

Here are examples of different types of animations you can experiment with: fade, slide, and scale.

width: 100,

height: 100,

color: Colors.blue, import 'package:flutter/material.dart';

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


class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Animation Examples',
     theme: ThemeData(
       primarySwatch: Colors.blue,
     ),
     home: AnimationExamples(),
   );
 }
}
class AnimationExamples extends StatefulWidget {
```

```
 @override
 _AnimationExamplesState createState() => _AnimationExamplesState();
}


class _AnimationExamplesState extends State<AnimationExamples> with
SingleTickerProviderStateMixin {
 late AnimationController _controller;
 late Animation<double> _fadeAnimation;
 late Animation<Offset> _slideAnimation;
 late Animation<double> _scaleAnimation;
 @override
 void initState() {
  super.initState();
  _controller = AnimationController(
   duration: Duration(seconds: 2),
   vsync: this,
  )..repeat(reverse: true);
  _fadeAnimation = Tween<double>(begin: 0.0, end: 1.0).animate(_controller);
  _slideAnimation = Tween<Offset>(begin: Offset(1.0, 0.0), end:
Offset.zero).animate(_controller);
  _scaleAnimation = Tween<double>(begin: 0.5, end: 1.0).animate(_controller);
 }
 @override
 Widget build(BuildContext context) {
  return Scaffold(
```

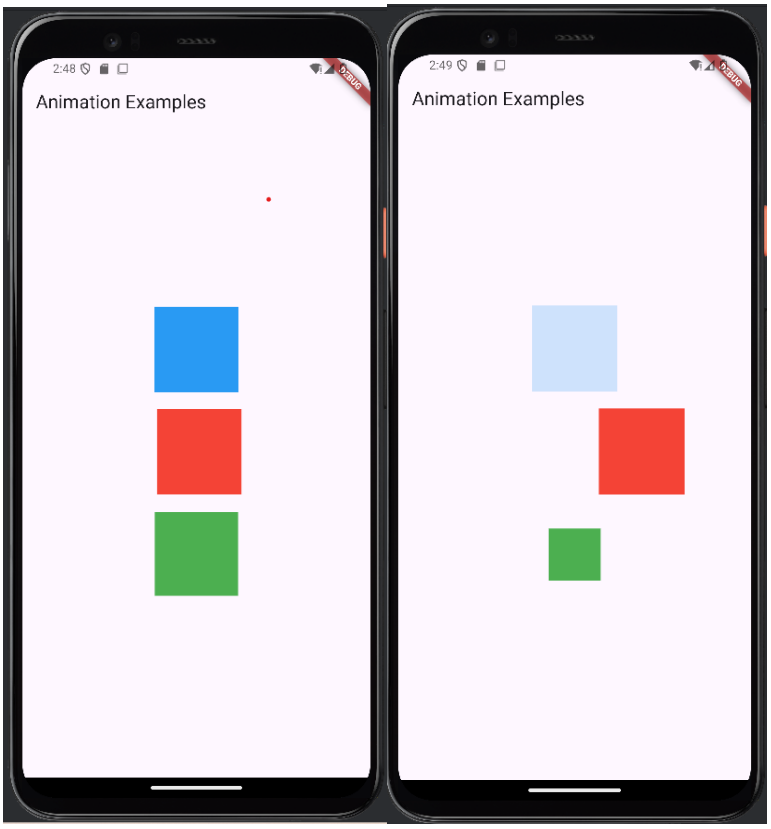
```
appBar: AppBar(title: Text('Animation Examples')),
body: Center(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: <Widget>[
      FadeTransition(
        opacity: _fadeAnimation,
        child: Container(


        ),
      ),
      SizedBox(height: 20),
      SlideTransition(
        position: _slideAnimation,
        child: Container(
          width: 100,
          height: 100,
          color: Colors.red,
        ),
      ),
      SizedBox(height: 20),
      ScaleTransition(
        scale: _scaleAnimation,
        child: Container(
```

```
            width: 100,
            height: 100,
            color: Colors.green,),),],),),);}
@override
void dispose() {
  _controller.dispose();
  super.dispose();
 }
}
```

Output :

Fade Animation
Slide Animation

Scale Animation


2:48
Animation Examples
DEBUG
2:49
Animation Examples
DEBUG

VIVA QUESTIONS

S.No	Question	CO	Blooms Taxonomy
1	What are the two core classes in Flutter's animation framework?	CO4	Remember
2	What is the purpose of an AnimationController?	CO4	Remember
3	Which widget is often used as a wrapper to listen to animation changes and rebuild?	CO4	Remember
4	What does the duration property of an AnimationController define?	CO4	Remember
5	What is the role of a TickerProvider in animations?	CO4	Remember
6	Explain the relationship between an AnimationController, an Animation (e.g., CurvedAnimation), and an AnimatedWidget.	CO4	Understand
7	Describe how setState() is used in conjunction with AnimationController to trigger UI updates during an animation.	CO4	Understand
8	Write a simple Flutter code snippet to create an Animation Controller that lasts for 1 second.	CO4	Apply
9	Implement a basic fading animation on a Text widget using an Animation Controller and an Animated Builder.	CO4	Apply
10	Demonstrate how to make an animation repeat indefinitely.	CO4	Apply
11	Show how to apply a Curved Animation (e.g., Curves. easeIn) to control the acceleration/deceleration of an animation.	CO4	Apply
12	Create a simple animation that scales a widget up and down when a button is pressed.	CO4	Apply
13	Discuss the challenges of coordinating multiple independent animations to create a complex animated sequence.	CO4	Analyze
14	Examine a scenario where an animation might cause layout issues (e.g., a widget growing beyond its parent's bounds). How can you prevent this?	CO4	Analyze
15	Analyze the role of vsync in Animation Controller and explain why it's necessary.	CO4	Analyze
16	What type of animation changes a widget's transparency?	CO4	Remember
17	Which animation type changes a widget's position?	CO4	Remember
18	What is a "transition widget" in Flutter animations?	CO4	Remember
19	Name an Animated Widget that handles fading implicitly.	CO4	Remember
20	Describe how Slide Transition changes a widget's position from one offset to another.	CO4	Understand

Note :Each experiment should contain Minimum 20 Viva Questions

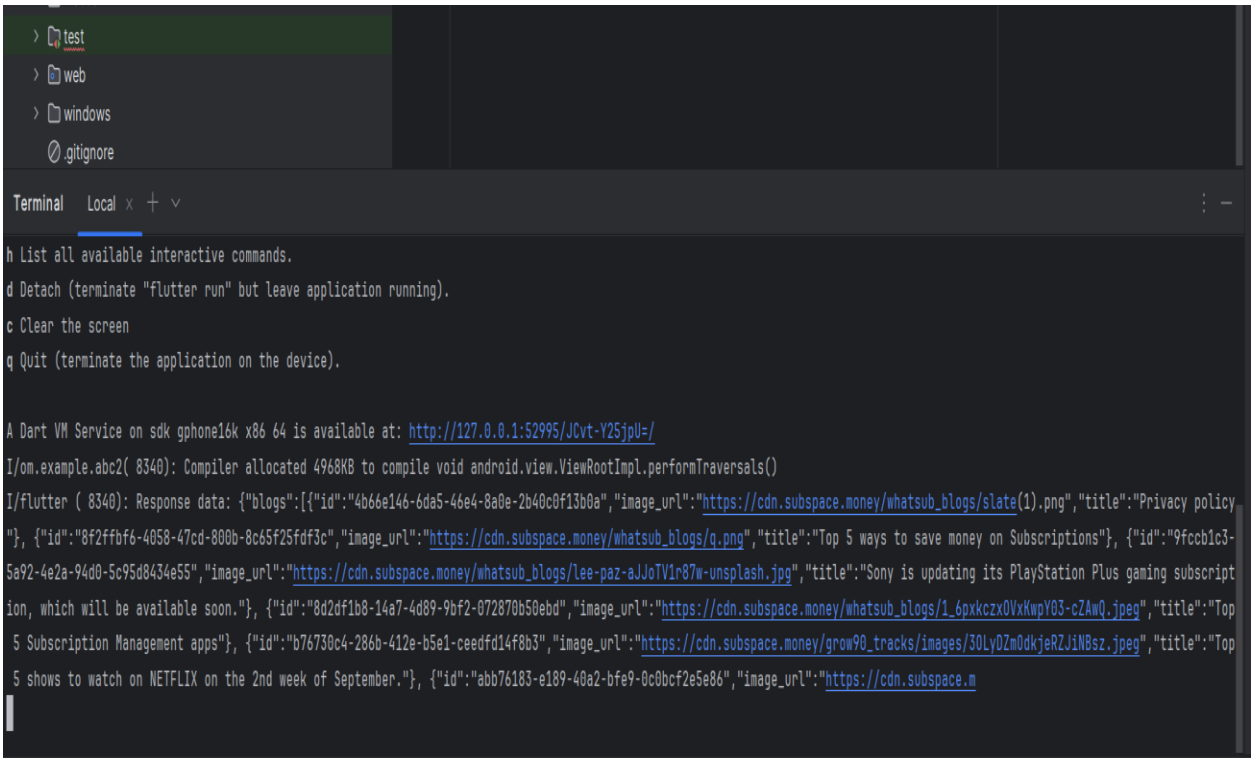
Experiment 9:

9. a) AIM: Fetch data from a REST API.

Note: You have to get the "http" package from "pub.dev" website.

```
import 'package:http/http.dart' as http;

void fetchBlogs() async {
 final String url = 'https://intent-kit-16.hasura.app/api/rest/blogs';
 final String adminSecret =
'32qR4KmXOIpsGPQKMqEJHGJS27G5s7HdSKO3gdtQd2kv5e852SiYwWNfxkZOBuQ6';
 try {
   final response = await http.get(Uri.parse(url), headers: {
     'x-hasura-admin-secret': adminSecret,
   });
   if (response.statusCode == 200) {
     // Request successful, handle the response data here
     print('Response data: ${response.body}');
   } else {
     // Request failed
     print('Request failed with status code: ${response.statusCode}');
     print('Response data: ${response.body}');
   }
 } catch (e) {
         // Handle any errors that occurred during the request
         print('Error: $e');  }
}
```

```
void main() {
 fetchBlogs();
}
```



9 b) AIM: Display the fetched data in a meaningful way in the UI.

```
import 'package:flutter/material.dart';
import 'dart:convert';
import 'package:http/http.dart' as http;
void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Blog App',
     theme: ThemeData(
       primarySwatch: Colors.blue,
     ),
     home: BlogListScreen(),
   );
 }
}

class BlogListScreen extends StatefulWidget {
 @override
 _BlogListScreenState createState() => _BlogListScreenState();
}
```

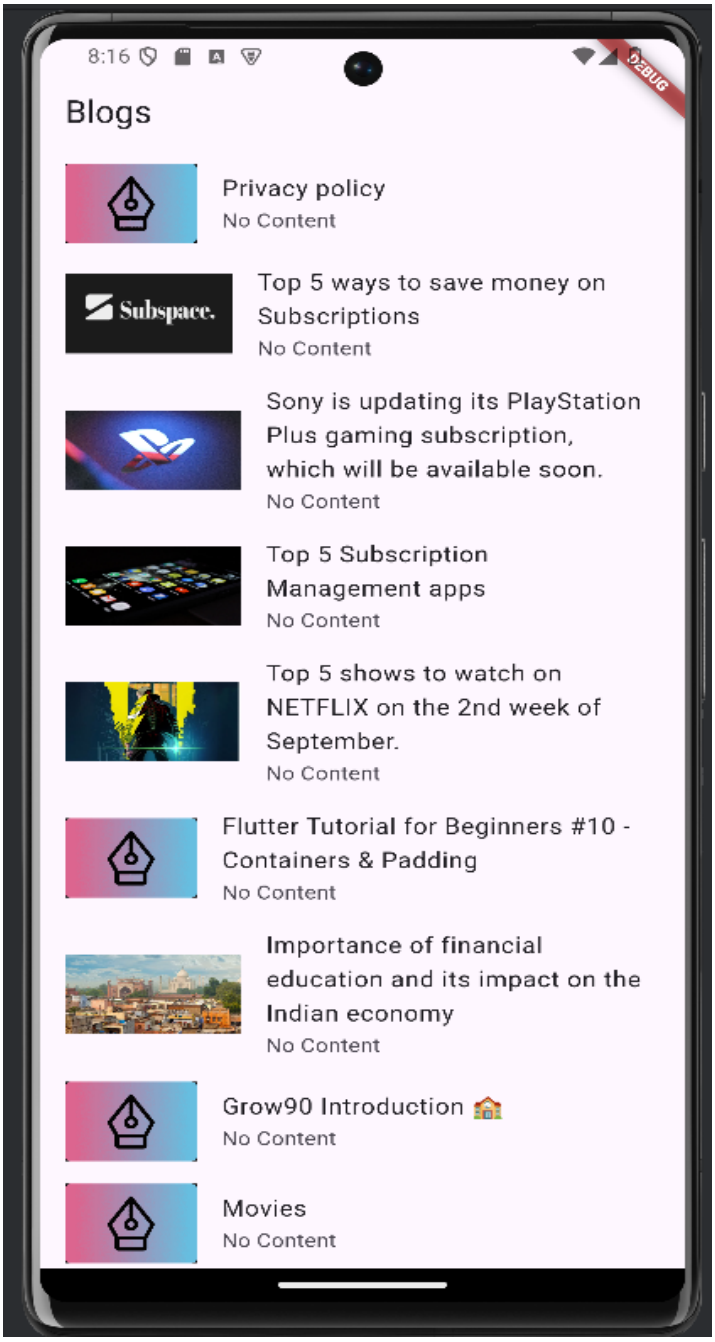
```
class _BlogListScreenState extends State<BlogListScreen> {
 late Future<List<Blog>> futureBlogs;
 @override
 void initState() {
  super.initState();
  futureBlogs = fetchBlogs();
 }
 @override
 Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
     title: Text('Blogs'),
    ),
    body: FutureBuilder<List<Blog>>(
     future: futureBlogs,
     builder: (context, snapshot) {
      if (snapshot.connectionState == ConnectionState.waiting) {
       return Center(child: CircularProgressIndicator());
      } else if (snapshot.hasError) {
       return Center(child: Text('Error: ${snapshot.error}'));
      } else if (!snapshot.hasData || snapshot.data!.isEmpty) {
       return Center(child: Text('No blogs available.'));
      } else {
       final blogs = snapshot.data!;
```

```
        return ListView.builder(
          itemCount: blogs.length,
          itemBuilder: (context, index) {
            final blog = blogs[index];
            return ListTile(
              leading: blog.imageUrl != null
                  ? Image.network(blog.imageUrl!)
                  : null,
              title: Text(blog.title),
              subtitle: Text(blog.content),
            );
          },
        );
      }
    },
  ),
 );
}
}

Future<List<Blog>> fetchBlogs() async {
 final String url = 'https://intent-kit-16.hasura.app/api/rest/blogs';
 final String adminSecret =
    '32qR4KmXOIpsGPQKMqEJHGJS27G5s7HdSKO3gdtQd2kv5e852SiYwWNfxkZOBuQ6';
```

```
try {
  final response = await http.get(Uri.parse(url), headers: {
    'x-hasura-admin-secret': adminSecret,
  });

  if (response.statusCode == 200) {
    final Map<String, dynamic> data = json.decode(response.body);
    final List<dynamic> blogList =
    data['blogs']; // Adjust this line based on your actual JSON structure
    return blogList.map((json) => Blog.fromJson(json)).toList();
  } else {
    throw Exception('Failed to load blogs');
  }
 } catch (e) {
  throw Exception('Error: $e');
 }
}

class Blog {
 final String title;
 final String content;
 final String? imageUrl;
 Blog({required this.title, required this.content, this.imageUrl});
```

```
factory Blog.fromJson(Map<String, dynamic> json) {
  return Blog(
    title: json['title'] ?? 'No Title',
    content: json['content'] ?? 'No Content',
    imageUrl:
    json['image_url'], // Adjust the key based on your JSON structure
  );
 }
}
```

OUTPUT:



Note : https://subspace.money/blog/whatsub-single-docs-internship-onboarding-docs-problem-statement-flutter-blog-explorer

VIVA QUESTIONS

S.No	Question	CO	Blooms Taxonomy
1	Which HTTP method is typically used to retrieve data from an API?	CO5	Remember
2	What does "REST API" stand for?	CO5	Remember
3	Explain the concept of "asynchronous programming" in Dart/Flutter and why it's crucial for fetching data.	CO5	Understand
4	Describe the role of Future and async/await in making API calls.	CO5	Understand
5	How do you parse a JSON string into a Dart object?	CO5	Understand
6	Write a simple Flutter code snippet using the http package to fetch data from a public API (e.g., JSONPlaceholder).	CO5	Apply
7	Demonstrate how to handle potential network errors (e.g., no internet connection) when making an API call.	CO5	Apply
8	Show how to parse a simple JSON response containing a list of items into a Dart List<Map<String, dynamic>>.	CO5	Apply
9	Implement a loading indicator (e.g., CircularProgressIndicator) that appears while data is being fetched and disappears when data is loaded.	CO5	Apply
10	Analyze a given API response structure (JSON) and explain how you would design your Dart model classes to efficiently parse it.	CO5	Analyze
11	Discuss the advantages and disadvantages of manually parsing JSON vs. using code generation packages (e.g., json_serializable).	CO5	Analyze
12	Examine a scenario where an API might return different data structures for success and error responses. How would you handle this in your data fetching logic?	CO5	Analyze
13	Analyze the performance implications of making too many simultaneous API calls or fetching excessively large datasets.	CO5	Analyze
14	Evaluate the robustness of an API fetching implementation in terms of error handling (e.g., timeout, server errors, invalid JSON).	CO5	Evaluate
15	Assess the security implications of fetching sensitive data from an API without proper authentication or encryption.	CO5	Evaluate
16	Given a choice between a REST API and a GraphQL API for a specific data fetching task, which would you	CO5	Evaluate

	choose and why?		
17	Explain how FutureBuilder helps in managing the different states (loading, data, error) of an asynchronous operation when building the UI.	CO5	Understand
18	Describe the difference between ListView and ListView.builder for displaying lists of fetched data. When would you use each?	CO5	Understand
19	Implement an error message display in your UI when FutureBuilder indicates snapshot.hasError.	CO5	Apply
20	Demonstrate how to display a CircularProgressIndicator while data is loading using FutureBuilder.	CO5	Apply

Experiment 10:

10. a)AIM: Write unit tests for UI components.

main.dart

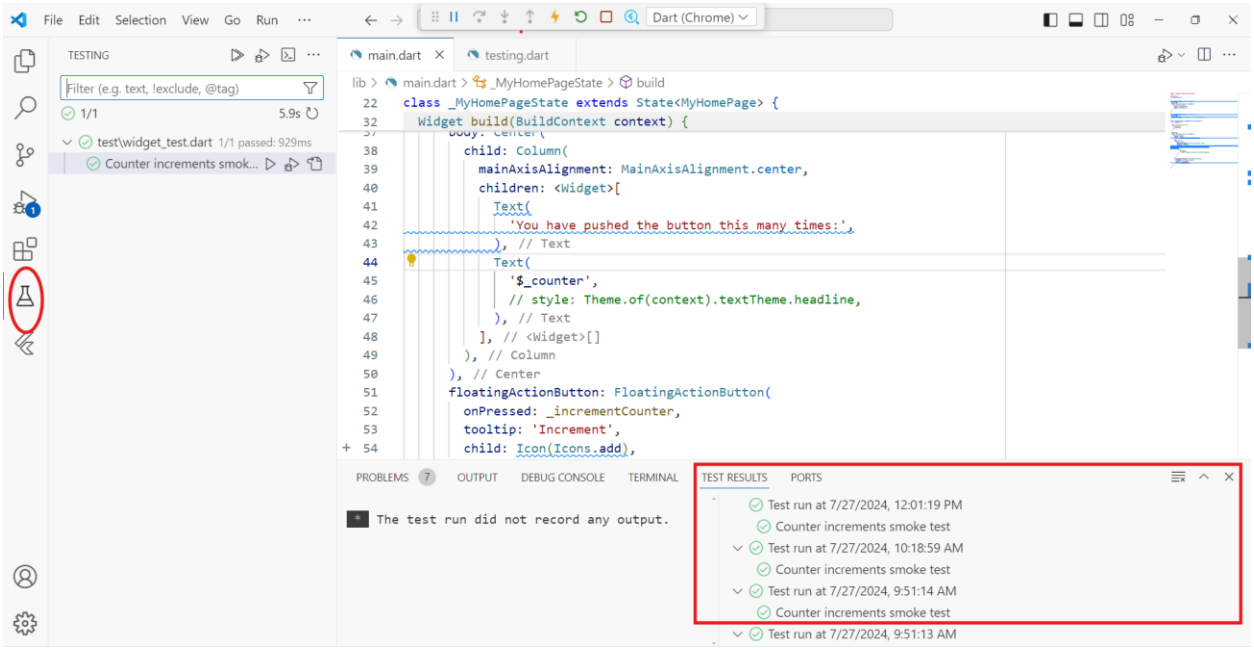
```
import 'package:flutter/material.dart';
void main() {
 runApp(MyApp());
}
class MyApp extends StatelessWidget {
 @override
 Widget build(BuildContext context) {
   return MaterialApp(
     title: 'Flutter Demo',
     home: MyHomePage(),
   );
 }
}

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

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

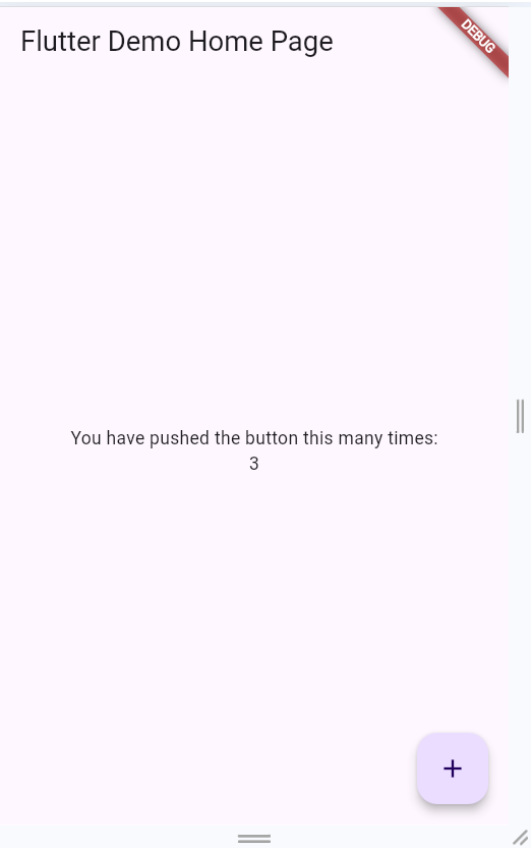
```
void _incrementCounter() {
  setState(() {
    _counter++;
  });
}
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('Flutter Demo Home Page'),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          Text(
            'You have pushed the button this many times:',
          ),
          Text(
            '$_counter',
            // style: Theme.of(context).textTheme.headline,
          ),
        ],),),
```

```
    floatingActionButton: FloatingActionButton(
      onPressed: _incrementCounter,
      tooltip: 'Increment',
      child: Icon(Icons.add),
    ),);}}
```

testing.dart

```
import 'package:flutter/material.dart';
import 'package:flutter_test/flutter_test.dart';
import 'package:testing/main.dart';
void main() {
 testWidgets('Counter increments smoke test', (WidgetTester tester) async {
   // Build our app and trigger a frame.
   await tester.pumpWidget(MyApp());
   // Verify that our counter starts at 0.
   expect(find.text('0'), findsOneWidget);
   expect(find.text('1'), findsNothing);
   // Tap the '+' icon and trigger a frame.
   await tester.tap(find.byIcon(Icons.add));
   await tester.pump();
   // Verify that our counter has incremented.
   expect(find.text('0'), findsNothing);
   expect(find.text('1'), findsOneWidget);
 });
}
```

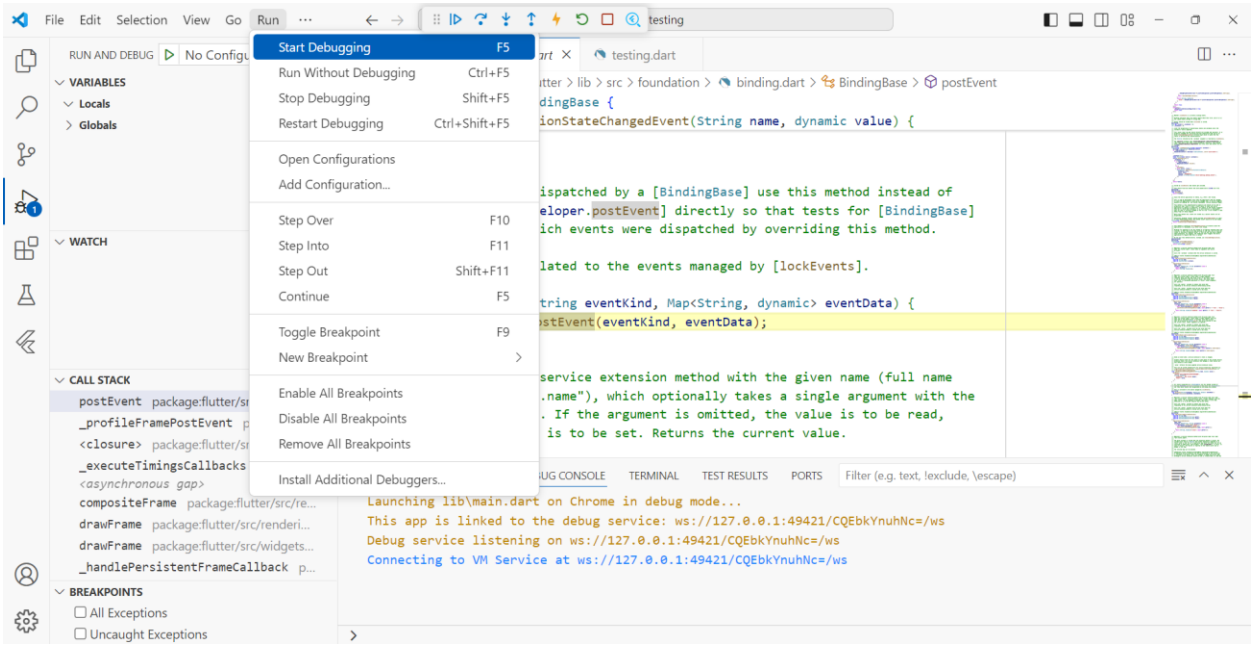

File Edit Selection View Go Run
Dart (Chrome)
TESTING
Filter (e.g. text, !exclude, @tag)
1/1
5.9s
test\widget_test.dart 1/1 passed: 929ms
Counter increments smok...
main.dart
testing.dart
lib > main.dart > _MyHomePageState > build
22 class _MyHomePageState extends State<MyHomePage> {
32 Widget build(BuildContext context) {
38 child: Column(
39 mainAxisAlignment: MainAxisAlignment.center,
40 children: <Widget>[
41 Text(
42 'You have pushed the button this many times:',
43), // Text
44 Text(
45 '$_counter',
46 // style: Theme.of(context).textTheme.headline,
47), // Text
48], // <Widget>[]
49), // Column
50), // Center
51 floatingActionButton: FloatingActionButton(
52 onPressed: _incrementCounter,
53 tooltip: 'Increment',
54 child: Icon(Icons.add),
PROBLEMS 7
OUTPUT
DEBUG CONSOLE
TERMINAL
TEST RESULTS
PORTS
The test run did not record any output.
Test run at 7/27/2024, 12:01:19 PM
Counter increments smoke test
Test run at 7/27/2024, 10:18:59 AM
Counter increments smoke test
Test run at 7/27/2024, 9:51:14 AM
Counter increments smoke test
Test run at 7/27/2024, 9:51:13 AM


Flutter Demo Home Page
DEBUG
You have pushed the button this many times:
3
+

10 b) Use Flutter's debugging tools to identify and fix issues.

Monitor the debug console for any error messages or logs. Flutter provides detailed error messages that often include stack traces and helpful tips.

It is used for checking the flow of the Application by passing through classes, and their functions.



VIVA QUESTIONS

S.No	Question	CO	Blooms Taxonomy
1	What is the difference between a "unit test" and a "widget test" in Flutter?	CO5	Remember
2	Which function is used to define a single test case in Flutter?	CO5	Remember
3	Describe the concept of "test isolation" and why it's important for unit tests.	CO5	Understand
4	How does the flutter_test package allow you to simulate widget interactions for testing purposes?	CO5	Understand
5	Write a simple unit test for a Dart class that performs a calculation, ensuring it returns the correct result.	CO5	Apply
6	Create a simple widget test for a StatelessWidget to verify that it displays a specific text.	CO5	Apply
7	Demonstrate how to simulate a tap gesture on a Button widget in a widget test.	CO5	Apply
8	Show how to verify that a TextField has a certain initial value in a widget test.	CO5	Apply
9	Implement a test that checks if a StatefulWidget updates its text correctly after a button press.	CO5	Apply
10	Analyze a given Flutter UI component. Discuss which parts could be unit tested (pure logic) and which would require widget tests (UI interaction/rendering).	CO5	Analyze
11	Compare and contrast the trade-offs between test coverage percentage and the quality/meaningfulness of tests.	CO5	Analyze
12	Discuss how mocking dependencies (e.g., API services, state management objects) is crucial for writing effective unit and widget tests.	CO5	Analyze
13	Examine a scenario where a widget test might be insufficient to catch a bug (e.g., a bug involving complex asynchronous operations or integration with native code).	CO5	Analyze
14	Design a testing strategy for a new Flutter feature that involves data fetching, state management, and UI rendering.	CO5	Create
15	Create a custom "test utility" function that simplifies common widget testing scenarios (e.g., finding a widget by a specific key and asserting its text).	CO5	Create
16	Name one tab or section within Flutter DevTools.	CO5	Remember
17	What is "Hot Reload" used for?	CO5	Remember

18	Demonstrate how to set a breakpoint in your Flutter code using your IDE (e.g., VS Code or Android Studio).	CO5	Apply
19	Show how to step through code line by line using the debugger controls (e.g., "Step Over," "Step Into").	CO5	Apply
20	Use the Widget Inspector in DevTools to identify the padding applied to a specific Text widget.	CO5	Apply

Note :Each experiment should contain Minimum 20 Viva Questions