User Interface Design Using Flutter Lab

List of 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.

EXPERIMENT-1

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

a) Install Flutter and Dart SDK.

What is Flutter?

Flutter is an oPen-source **UI software develoPment toolkit** created by **Google**. It is used to build **natively comPiled** aPPlications for:

- **Mobile** Platforms (Android & iOS),
- **Web** browsers, and
- **DesktoP** oPerating systems (Windows, macOS, Linux)

... all from a **single codebase**.

Flutter uses a rich collection of **widgets** to design resPonsive and flexible user interfaces. Because Flutter comPiles directly to native machine code, it offers high Performance and smooth animations across all suPPorted Platforms.

What is Dart Programming Language?

Dart is a Programming language develoPed by **Google**, Primarily designed to build aPPlications with Flutter.

- It is **object-oriented**, **strongly tyPed**, and suPPorts **both just-in-time (JIT)** and **ahead-of-time (AOT)** comPilation.
- Dart code can be comPiled to **native machine code** for mobile and desktoP aPPs or to **JavaScriPt** for web aPPlications.
- Dart is the Primary language used to write Flutter aPPlications due to its simPlicity and Performance advantages.

Uses of Flutter and Dart

Flutter is used for:

- DeveloPing **cross-Platform aPPlications** with native-like Performance and aPPearance.
- Creating rich and highly customizable **user interfaces** using a flexible widget system.
- RaPid develoPment cycles through **hot reload**, which lets develoPers see changes in real-time without restarting the aPP.
- Building aPPlications for **Android**, **IOS**, **web**, and **desktoP Platforms** from a single codebase, significantly reducing develoPment time and effort.

Dart is used for:

- Writing **Flutter aPPlications**.
- Backend and server-side develoPment.
- Creating **command-line tools**.
- DeveloPing web aPPlications by comPiling Dart to JavaScriPt.

SteP-by-SteP Installation of Flutter and Dart

Note: Flutter comes bundled with the Dart SDK. You do **not** need to install Dart seParately when installing Flutter.

SteP 1: System Requirements

For Windows:

- OPerating System: Windows 10 or 11 (64-bit)
- Available Disk SPace: At least 1.64 GB (excluding IDEs and other tools)
- Additional Tools: Git installed, PowerShell available, and Chrome (required for Flutter web develoPment)

Step 2: Download Flutter SDK

1. Visit the official Flutter website: https://flutter.dev/docs/get-started/install
2. Select your operating system (e.g., Windows).
3. Download the latest stable release of the Flutter SDK as a .zip archive.

Step 3: Extract Flutter SDK

1. Extract the downloaded ZIP file to a permanent location on your computer. Example location: C:/src/flutter
2. Avoid extracting it to system directories that require administrator permissions, such as C:/Program Files/.

Step 4: Set Environment Variable (Path)

1. Open **System Properties** → **Environment Variables**.
2. Under **System variables**, select the Path variable and click **Edit**.
3. Add the full path to the Flutter SDK's bin folder, for example:
4. C:/src/flutter/bin
5. Save and close all dialogs.

Step 5: Verify Flutter Installation

1. Open **Command Prompt** or **PowerShell**.
2. Run the following command to verify Flutter is correctly installed and see what else you need to set up:
3. flutter doctor
4. Review the output. The tool will identify missing dependencies or required tools such as Android Studio or device drivers.

Step 6: Install Required Dependencies

- Install **Android Studio** to get the Android SDK and emulator.
- Optionally, install **Visual Studio Code** for a lightweight development environment.
- Install **Google Chrome** if you intend to develop Flutter web applications.

Step 7: Install Flutter Plugins in Your IDE

- **Visual Studio Code:**
  Open VS Code → Extensions (Ctrl+Shift+X) → Search for and install the **Flutter** extension (which also installs the **Dart** extension).
- **Android Studio:**
  Open Android Studio → Preferences\Settings → Plugins → Search for **Flutter** → Install the plugin. The Dart plugin will be installed automatically as a dependency.

Step 8: Set Up Android Emulator (Optional)

1. Open Android Studio and launch the **AVD Manager** (Android Virtual Device Manager).
2. Create a new virtual device (e.g., Pixel 5 with API 33).
3. Start the emulator and ensure it runs properly.

Step 9: Run Your First Flutter Application

1. Open a terminal and create a new Flutter project:
2. flutter create my_app
3. Navigate into your project directory:

4. cd my_aPP
5. Run your Flutter aPP on the connected device or emulator:
6. flutter run

Test Installation

To confirm that your Flutter setuP is fully functional, run:

flutter doctor

Make sure that all sections have a green checkmark, indicating your environment is correctly configured.

b) Write a simPle Dart Program to understand the language basics.

Program:

```
void main() {
  String name = 'John';
  int age = 20;

  Print('Name: $name, Age: $age');

  if (age >= 18) {
    Print('$name is an adult.');
  } else {
    Print('$name is a minor.');
  }

  for (int i = 1; i <= 3; i++) {
    Print('Count: $i');
  }

  greet(name);
}

void greet(String name) {
  Print('Hello, $name!');
}
```

ExPlanation of ComPonents:

- **void main()**
  The entry Point of every Dart Program. The code inside main() runs when the Program starts.
- **Variables (String name, int age)**
  Store data with sPecific tyPes:

- String holds text.
- int holds whole numbers.

- **Print()**
  DisPlays outPut in the console. Here it shows the name and age, and messages inside the condition and looP.
- **Conditional statement (if-else)**
  Executes different code based on a condition.
  If age is 18 or older, it Prints that the Person is an adult; otherwise, it Prints that they are a minor.
- **for looP**
  RePeats a block of code multiPle times.
  Here, it counts from 1 to 3, Printing each number.
- **Function (greet)**
  A reusable block of code that takes a Parameter (name) and Prints a greeting message.

  It is called from main() with greet(name).

EXPERIMENT-2

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

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

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

DescriPtion:

In Flutter, widgets are the fundamental building blocks of the user interface. Each element you see on the screen is a widget, from simPle text labels to comPlex layouts. For examPle, the **Text** widget is used to disPlay a string of text with customizable styles such as font size, color, and weight. The **Image** widget allows you to disPlay images from assets, network, or files, suPPorting various formats and fitting oPtions. The **Container** widget is a versatile box model that can hold a child widget and aPPly styling ProPerties like Padding, margin, borders, background color, and size constraints, making it ideal for layout and decoration PurPoses. Together, these widgets form the basis of designing resPonsive, flexible, and visually aPPealing Flutter aPPlications. By combining and nesting widgets, develoPers can create intricate UI

designs with Precise control over aPPearance and behavior.

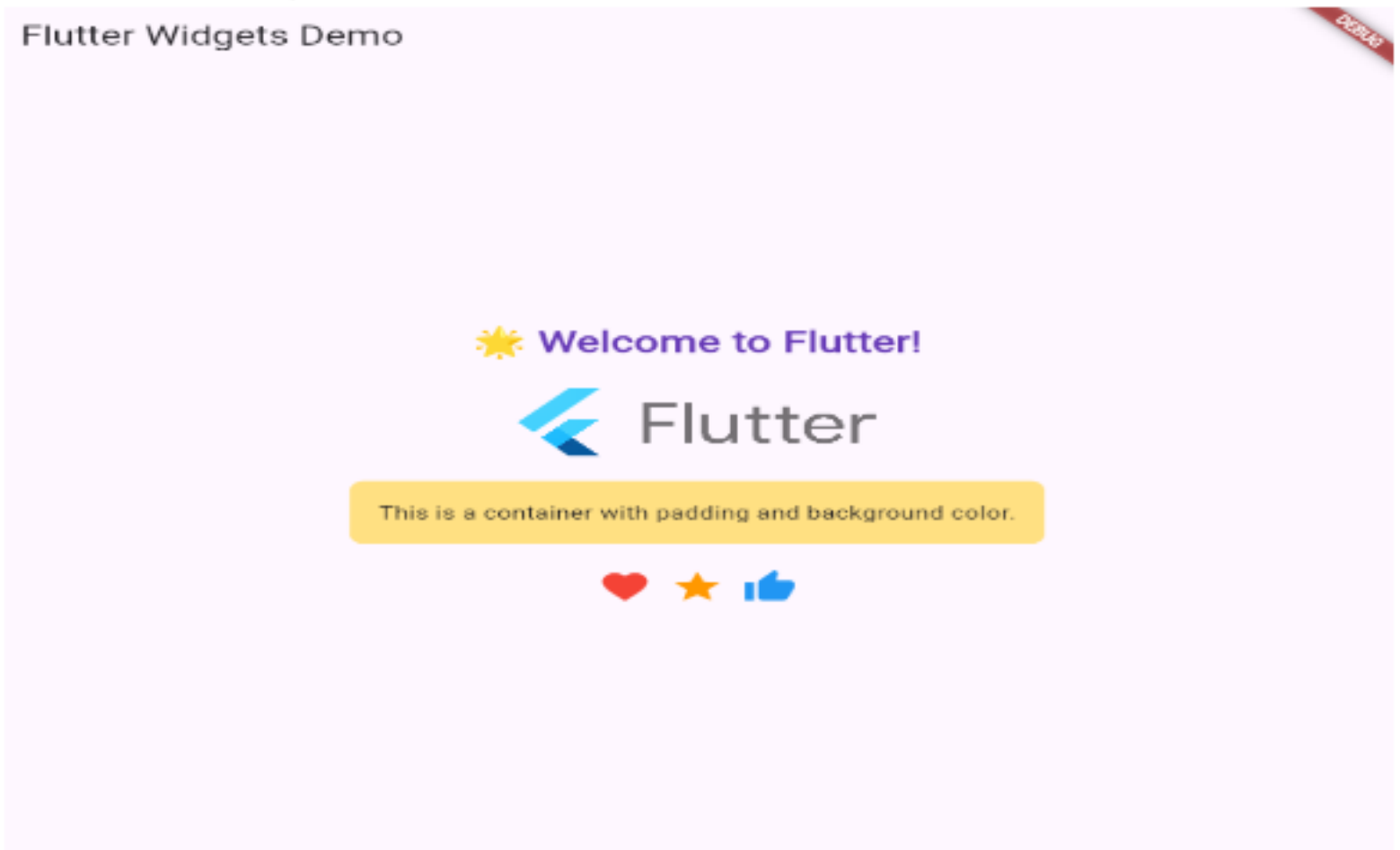
Program:

```
imPort 'Package:flutter\material.dart';

void main() {
  runAPP(MyAPP());
}

class MyAPP extends StatelessWidget {
  \\ Using a reliable Flutter logo from a trusted source
  final String imageUrl =
      'httPs:\\uPload.wikimedia.org\wikiPedia\commons\1\17\Google-flutter-
logo.Png';

  @override
  Widget build(BuildContext context) {
    return MaterialAPP(
      title: 'Flutter Widgets Demo',
      home: Scaffold(
        aPPBar: APPBar(
          title: Text('Flutter Widgets Demo'),
        ),
body: Center(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [

      \\ 📝 TEXT WIDGET
      Text(
        '🌟 Welcome to Flutter!',
        style: TextStyle(
          fontSize: 24,
          color: Colors.deePPurPle,
          fontWeight: FontWeight.bold,
        ),
      ),

      SizedBox(height: 20),

      \\ 🖼️ IMAGE WIDGET with error handling
      Image.network(
        imageUrl,
        width: 200,
```

```
          errorBuilder: (context, error, stackTrace) {
            return Column(
              children: [
                Icon(Icons.error, color: Colors.red, size: 40),
                Text('Failed to load image'),
              ],
            );
          },
        ),

        SizedBox(height: 20),

        \\ 📦 CONTAINER WIDGET
        Container(
          padding: EdgeInsets.all(16),
          decoration: BoxDecoration(
            color: Colors.amber[200],
            borderRadius: BorderRadius.circular(8),
          ),
          child: Text(
            'This is a container with padding and background color.',
            textAlign: TextAlign.center,
          ),
        ),

        SizedBox(height: 20),

        \\ 🎯 ICON WIDGETS IN A ROW
        Row(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Icon(Icons.favorite, color: Colors.red, size: 30),
            SizedBox(width: 10),
            Icon(Icons.star, color: Colors.orange, size: 30),
            SizedBox(width: 10),
            Icon(Icons.thumb_up, color: Colors.blue, size: 30),
          ],
        ),
      ],
    ),
  ),
),
);
}
```

```
}
```

Sample OutPut:



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

Description:

Flutter offers Powerful layout widgets that helP arrange Ul elements in different ways. The **Row** widget Places its child widgets horizontally in a single line, allowing you to align, sPace, and size items side-by-side. Conversely, the **Column** widget arranges children vertically, stacking them from toP to bottom. Both Row and Column Provide flexible alignment and sPacing oPtions, making them essential for creating structured and resPonsive layouts. The **Stack** widget, on the other hand, allows children to be layered on toP of each other, similar to a stack of cards. This is useful for overlaPPing widgets, creating comPlex designs like badges, floating buttons, or custom decorations. By exPerimenting with these widgets, you learn how to control Positioning, alignment, and layering in Flutter's Ul, enabling you to build soPhisticated and adaPtable user interfaces.

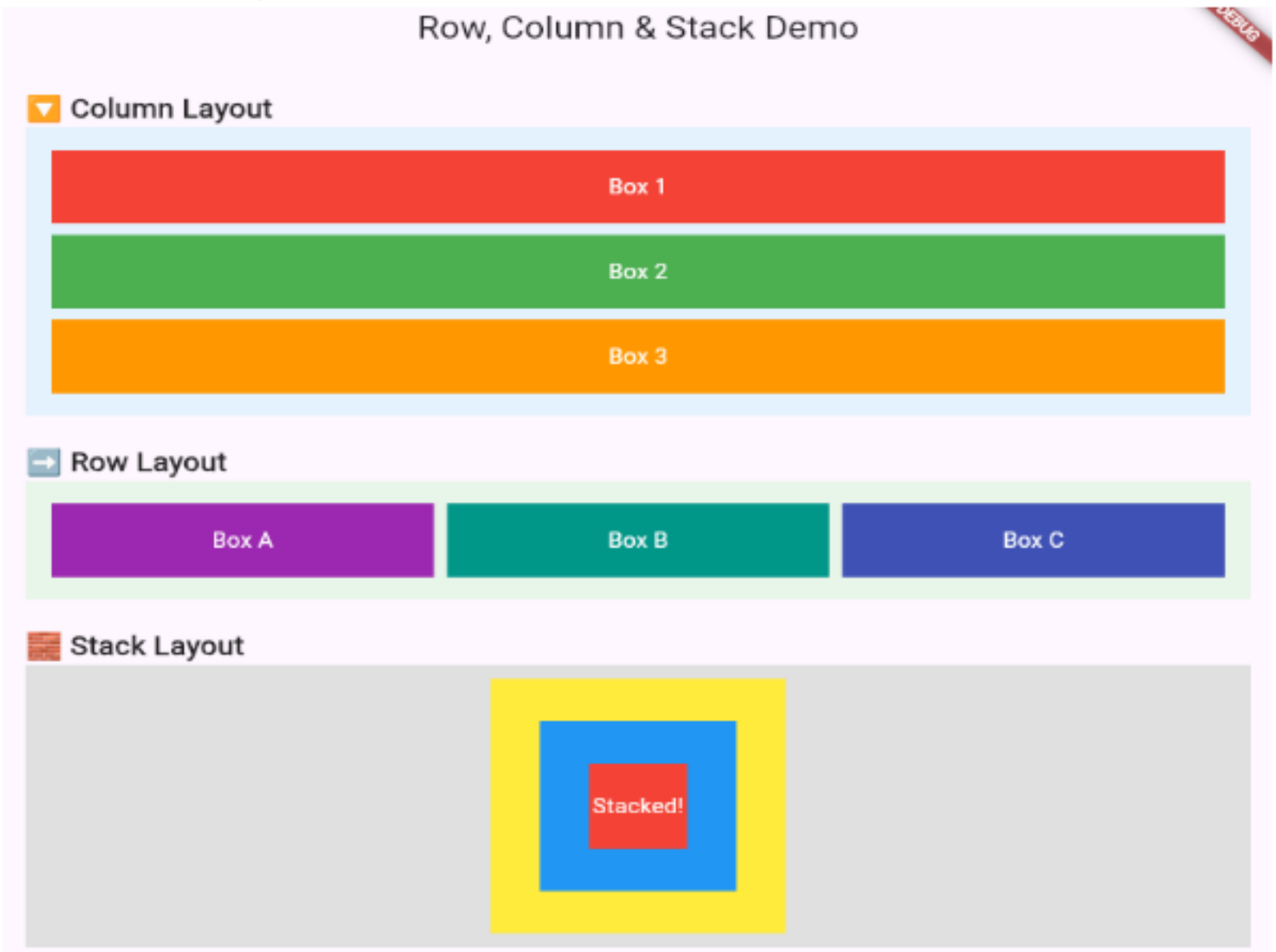
Program:

```
imPort 'Package:flutter\material.dart';

void main() => runAPP(LayoutDemoAPP());

class LayoutDemoAPP extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialAPP(
      title: 'Layout Widgets Demo',
      home: Scaffold(
        aPPBar: APPBar(
          title: Text('Row, Column & Stack Demo'),
          centerTitle: true,
        ),
        body: ListView(
          Padding: EdgeInsets.all(16),
          children: [

            \\ Column Layout
            Text('🔽 Column Layout', style: headerStyle),
            Container(
              color: Colors.blue[50],
              Padding: EdgeInsets.all(12),
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.stretch,
                children: [
                  coloredBox('Box 1', Colors.red),
                  coloredBox('Box 2', Colors.green),
                  coloredBox('Box 3', Colors.orange),
                ],
              ),
            ),

            SizedBox(height: 20),

            \\ Row Layout
            Text('➡️ Row Layout', style: headerStyle),
            Container(
              color: Colors.green[50],
              Padding: EdgeInsets.all(12),
              child: Row(
                mainAxisAlignment: MainAxisAlignment.sPaceEvenly,
```

```
        children: [
          Flexible(child: coloredBox('Box A', Colors.purple)),
          Flexible(child: coloredBox('Box B', Colors.teal)),
          Flexible(child: coloredBox('Box C', Colors.indigo)),
        ],
      ),
    ),

    SizedBox(height: 20),

    \\ Stack Layout
    Text('🧱 Stack Layout', style: headerStyle),
    Container(
      width: double.infinity,
      height: 200,
      color: Colors.grey[300],
      child: Stack(
        alignment: Alignment.center,
        children: [
          Container(width: 180, height: 180, color: Colors.yellow),
          Container(width: 120, height: 120, color: Colors.blue),
          Container(width: 60, height: 60, color: Colors.red),
          Text('Stacked!', style: TextStyle(color: Colors.white)),
        ],
      ),
    ),
  ),
);
}

\\ Helper method to create colored boxes with text
Widget coloredBox(String label, Color color) {
  return Container(
    margin: EdgeInsets.all(4),
    Padding: EdgeInsets.all(16),
    color: color,
    child: Center(
      child: Text(
        label,
        style: TextStyle(color: Colors.white),
      ),
    ),
```

```
    );
  }

  \\ Style for section headers
  TextStyle get headerStyle => TextStyle(
        fontSize: 18,
        fontWeight: FontWeight.bold,
        color: Colors.black87,
      );
}
```

Sample OutPut:



EXPERIMENT-3

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

b) Implement media queries and breakpoints for responsiveness.

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

Description:

Designing a responsive UI in Flutter involves creating layouts that adapt seamlessly to different screen sizes and orientations, providing a consistent user experience across devices such as phones, tablets, and desktops. Flutter's flexible widget system, combined with tools like **MediaQuery** and **LayoutBuilder**, allows developers to dynamically adjust widget sizes, padding, and layout structures based on the available screen dimensions. Using techniques such as flexible widgets (Expanded, Flexible), percentage-based sizing, and adaptive breakpoints, a responsive UI can rearrange or resize elements to fit small mobile screens or large desktop displays. This approach ensures usability and visual appeal regardless of device, making the app accessible and easy to navigate on any platform.

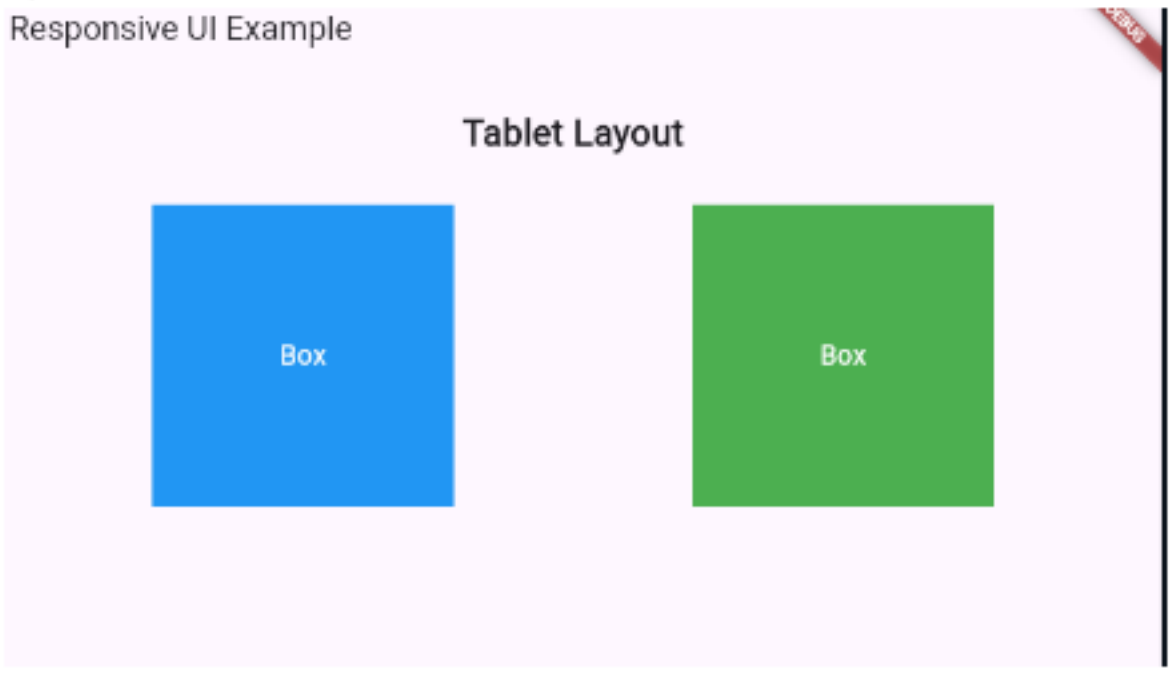
Program:

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

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

class ResponsiveApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Responsive UI Demo',
      home: Scaffold(
        appBar: AppBar(title: Text('Responsive UI Example')),
        body: ResponsiveLayout(),
      ),
    );
  }
}

class ResponsiveLayout extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    \\ Get screen width and height using MediaQuery
    final screenWidth = MediaQuery.of(context).size.width;

    return LayoutBuilder(
      builder: (context, constraints) {
        if (constraints.maxWidth < 600) {
          \\ Small screen: Mobile layout
          return mobileLayout();
        } else if (constraints.maxWidth < 900) {
          \\ Medium screen: Tablet layout
```

```
        return tabletLayout();
      } else {
        \\ Large screen: DesktoP layout
        return desktoPLayout();
      }
    },
  );
}

Widget mobileLayout() {
  return Padding(
    Padding: EdgeInsets.all(16),
    child: Column(
      children: [
        titleText('Mobile Layout'),
        SizedBox(height: 20),
        resPonsiveBox(Colors.blue, 150),
        SizedBox(height: 20),
        resPonsiveBox(Colors.green, 150),
      ],
    ),
  );
}

Widget tabletLayout() {
  return Padding(
    Padding: EdgeInsets.symmetric(horizontal: 32, vertical: 24),
    child: Column(
      children: [
        titleText('Tablet Layout'),
        SizedBox(height: 30),
        Row(
          mainAxisAlignment: MainAxisAlignment.sPaceAround,
          children: [
            resPonsiveBox(Colors.blue, 200),
            resPonsiveBox(Colors.green, 200),
          ],
        ),
      ],
    ),
  );
}

Widget desktoPLayout() {
```

```
  return Padding(
    Padding: EdgeInsets.symmetric(horizontal: 64, vertical: 32),
    child: Column(
      children: [
        titleText('DesktoP Layout'),
        SizedBox(height: 40),
        Row(
          mainAxisAlignment: MainAxisAlignment.sPaceEvenly,
          children: [
            resPonsiveBox(Colors.blue, 250),
            resPonsiveBox(Colors.green, 250),
            resPonsiveBox(Colors.orange, 250),
          ],
        ),
      ],
    ),
  );
}

Widget titleText(String text) {
  return Text(
    text,
    style: TextStyle(
      fontSize: 24,
      fontWeight: FontWeight.bold,
    ),
  );
}

Widget resPonsiveBox(Color color, double size) {
  return Container(
    width: size,
    height: size,
    color: color,
    alignment: Alignment.center,
    child: Text(
      'Box',
      style: TextStyle(color: Colors.white, fontSize: 18),
    ),
  );
}
}
```

Sample Output:



b) Implement media queries and breakPoints for responsiveness.

Description:

In Flutter, **MediaQuery** and **breakPoints** are essential tools for building responsive applications that adapt to different screen sizes and orientations. **MediaQuery** Provides information about the current device's screen dimensions, Pixel density, and orientation, allowing developers to dynamically adjust widget properties like width, height, and layout based on the available space. By defining **breakPoints**—specific screen width thresholds—developers can change the UI structure or styling when the screen size crosses these limits, such as switching from a single-column layout on small screens to a multi-column layout on larger screens. This approach helps create flexible and user-friendly interfaces that maintain usability and aesthetic appeal across a wide range of devices, from small smartphones to large desktop monitors.

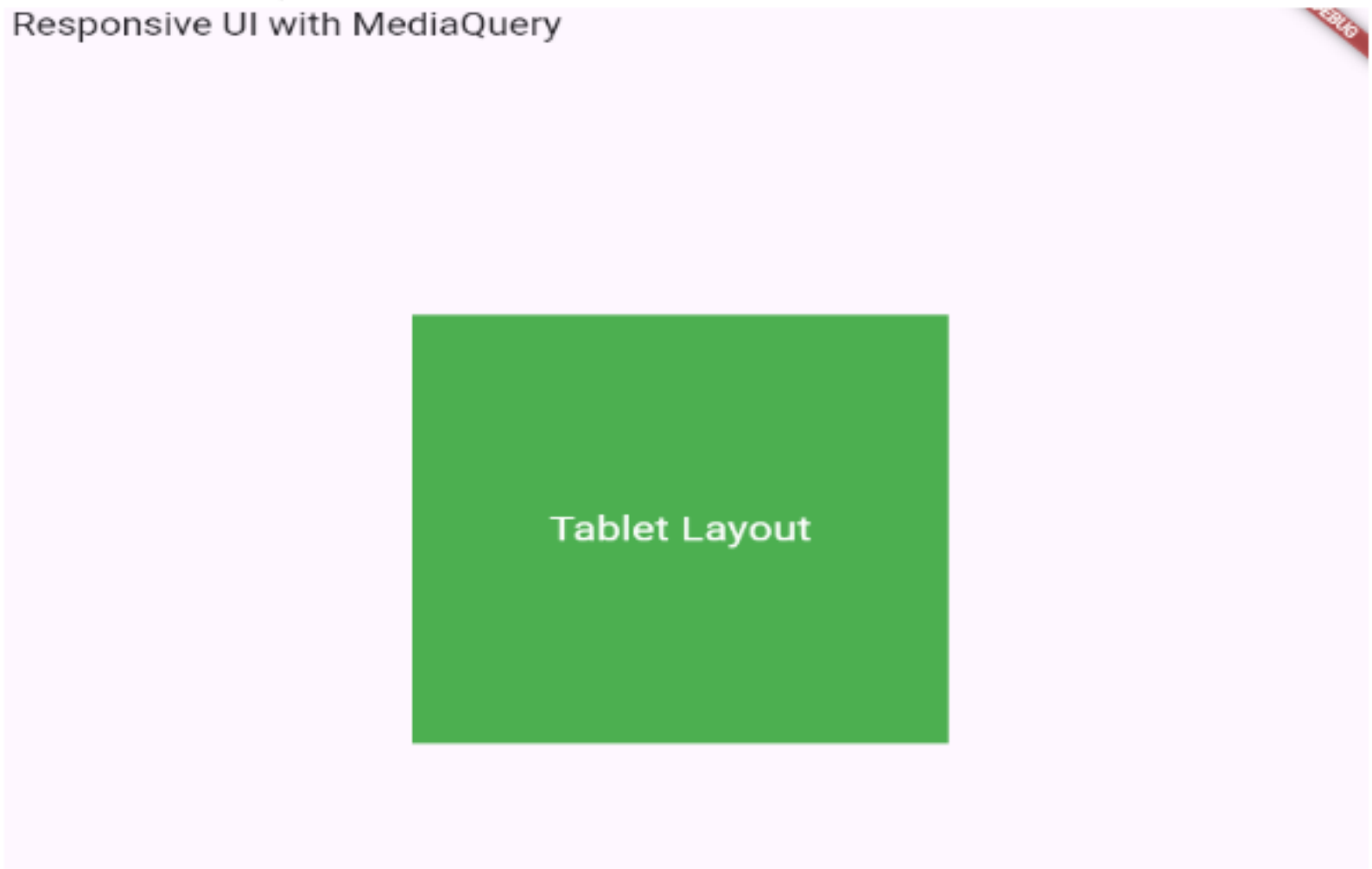
Sample Output:

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

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

class MediaQueryExamPleAPP extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialAPP(
      title: 'MediaQuery & BreakPoints Demo',
      home: Scaffold(
        aPPBar: APPBar(title: Text('ResPonsive UI with MediaQuery')),
        body: ResPonsiveWidget(),
      ),
    );
```

```
  }
}

class ResPonsiveWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    \\ Get screen width from MediaQuery
    double screenWidth = MediaQuery.of(context).size.width;

    \\ Define breakPoints
    if (screenWidth < 600) {
      return mobileLayout();
    } else if (screenWidth < 900) {
      return tabletLayout();
    } else {
      return desktoPLayout();
    }
  }

  Widget mobileLayout() {
    return Center(
      child: Container(
        color: Colors.blue,
        width: 150,
        height: 150,
        child: Center(
          child: Text('Mobile Layout',
              style: TextStyle(color: Colors.white, fontSize: 18)),
        ),
      ),
    );
}

Widget tabletLayout() {
  return Center(
    child: Container(
      color: Colors.green,
      width: 300,
      height: 300,
      child: Center(
        child: Text('Tablet Layout',
            style: TextStyle(color: Colors.white, fontSize: 24)),
      ),
    ),
```

```
    );
  }

  Widget desktopLayout() {
    return Center(
      child: Container(
        color: Colors.orange,
        width: 450,
        height: 450,
        child: Center(
          child: Text('Desktop Layout',
              style: TextStyle(color: Colors.white, fontSize: 30)),
        ),
      ),
    );
  }
}
```

Sample Output:



EXPERIMENT-4

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

b) Implement navigation with named routes.

a) Set uP navigation between different screens using Navigator.

Description:

In Flutter, navigation between different screens (or Pages) is managed using the **Navigator** widget, which maintains a stack of routes rePresenting the screens. By Pushing a new route onto the stack, you move forward to a new screen, and by PoPPing a route, you return to the Previous screen. This stack-based navigation model allows for smooth transitions and back-button suPPort. DeveloPers define routes as widgets and use Navigator.PushO to navigate to a new screen and Navigator.PoPO to go back. This aPProach enables building multi-screen aPPs with clear flow and user control, suPPorting both simPle navigation and comPlex routing scenarios such as Passing data between screens and handling named routes.

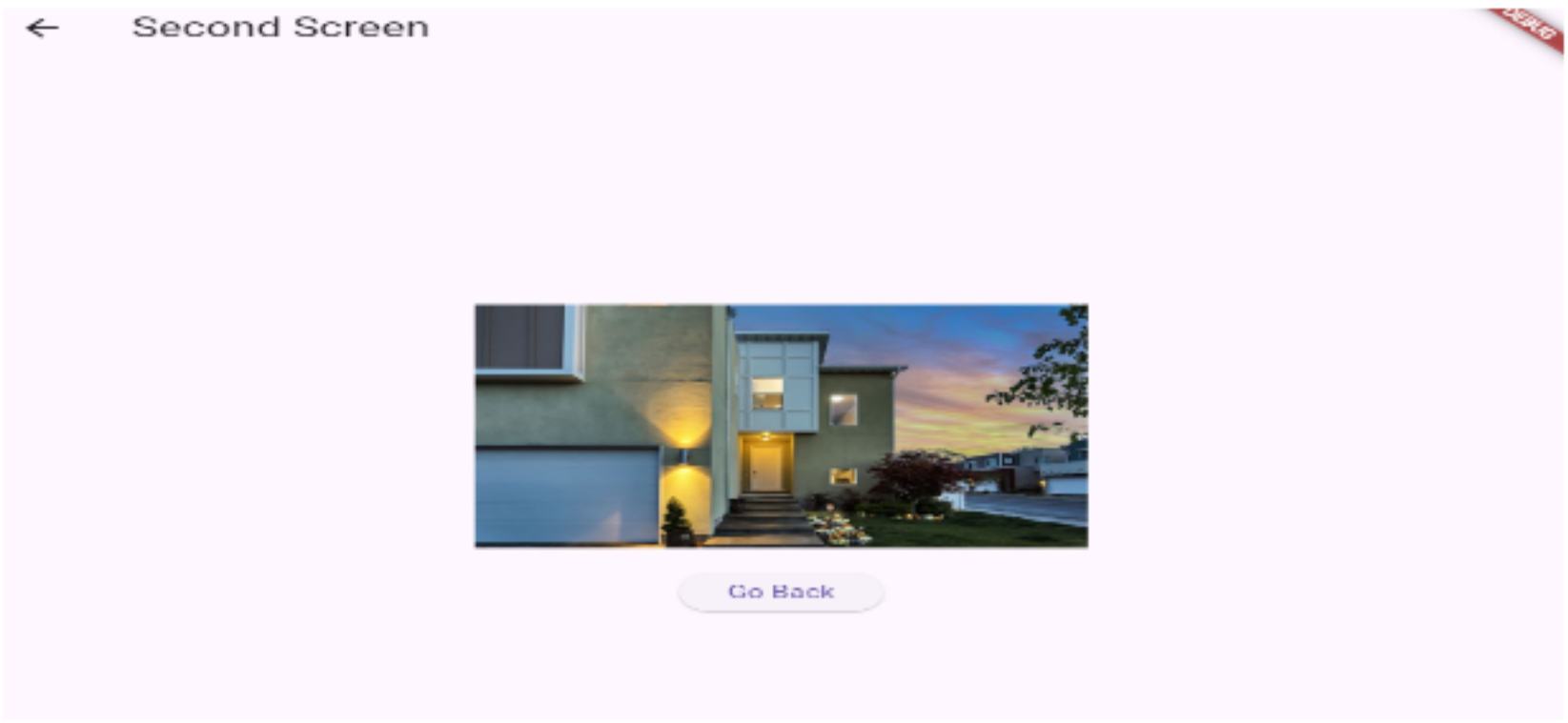
Program:

```
imPort 'Package:flutter\material.dart';

void main() => runAPP(NavigationWithImagesAPP());

class NavigationWithImagesAPP extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialAPP(
      title: 'Navigator with Images Demo',
      home: FirstScreen(),
    );
  }
}

\\ First screen with image and button
class FirstScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      aPPBar: APPBar(title: Text('First Screen')),
      body: Center(
```

```
      child: Column(
        mainAxisSize: MainAxisSize.min,
        children: [
          \\ UPdated image URL (works fine)
          Image.network(
            'httPs:\\images.unsPlash.com\Photo-1506744038136-46273834b3fb?
auto=format&fit=croP&w=600&q=60',
            width: 300,
            height: 200,
            fit: BoxFit.cover,
          ),
          SizedBox(height: 20),
          ElevatedButton(
            child: Text('Go to Second Screen'),
            onPressed: () {
              Navigator.Push(
                context,
                MaterialPageRoute(builder: (context) => SecondScreen()),
              );
            },
          ),
        ],
      ),
    );
  }
}

\\ Second screen with different image and button
class SecondScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      aPPBar: APPBar(title: Text('Second Screen')),
      body: Center(
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            \\ UPdated image URL (works fine)
            Image.network(
              'httPs:\\images.unsPlash.com\Photo-1494526585095-c41746248156?
auto=format&fit=croP&w=600&q=60',
              width: 300,
              height: 200,
```

```
            fit: BoxFit.cover,
          ),
          SizedBox(height: 20),
          ElevatedButton(
            child: Text('Go Back'),
            onPressed: () {
              Navigator.pop(context);
            },
          ),
        ],
      ),
    ),
  );
 }
}
```

Sample OutPut:



b) Implement navigation with named routes.

Description:

Flutter's **named routes** Provide a clean and organized way to manage navigation in aPPs with multiPle screens. Instead of directly Pushing widget instances onto the navigation stack, named routes use string identifiers to reference screens. DeveloPers define a maP of route names to their corresPonding widget builders in the MaterialAPP (or CuPertinoAPP) configuration. Navigation is then Performed using Navigator.PushNamed() and Navigator.PoP(), which imProves code readability and makes it easier to manage and scale comPlex navigation flows. Named routes also simPlify deeP linking and make it straightforward to Pass arguments between screens. This method Promotes better seParation of concerns and centralizes route definitions for easier maintenance.

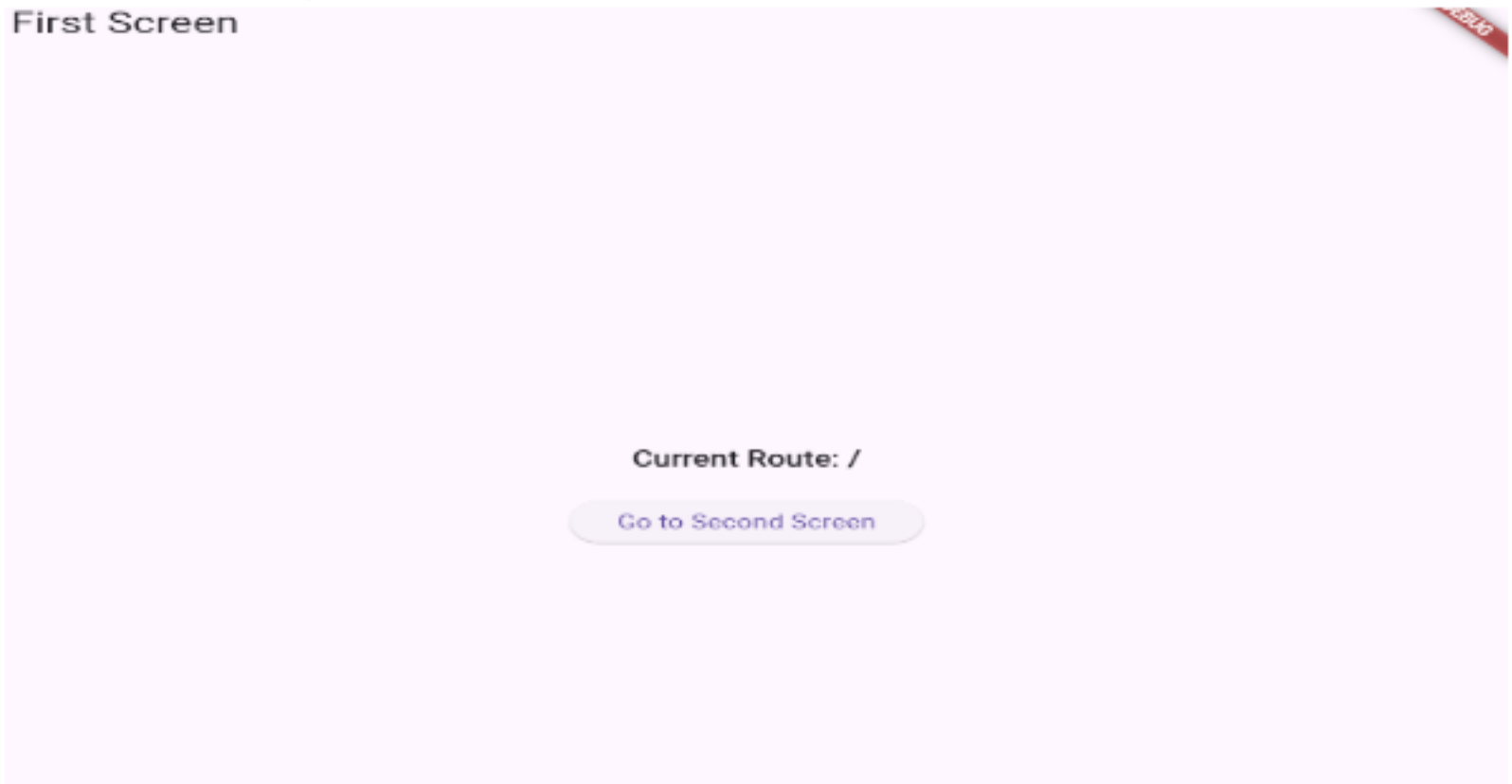
Program:

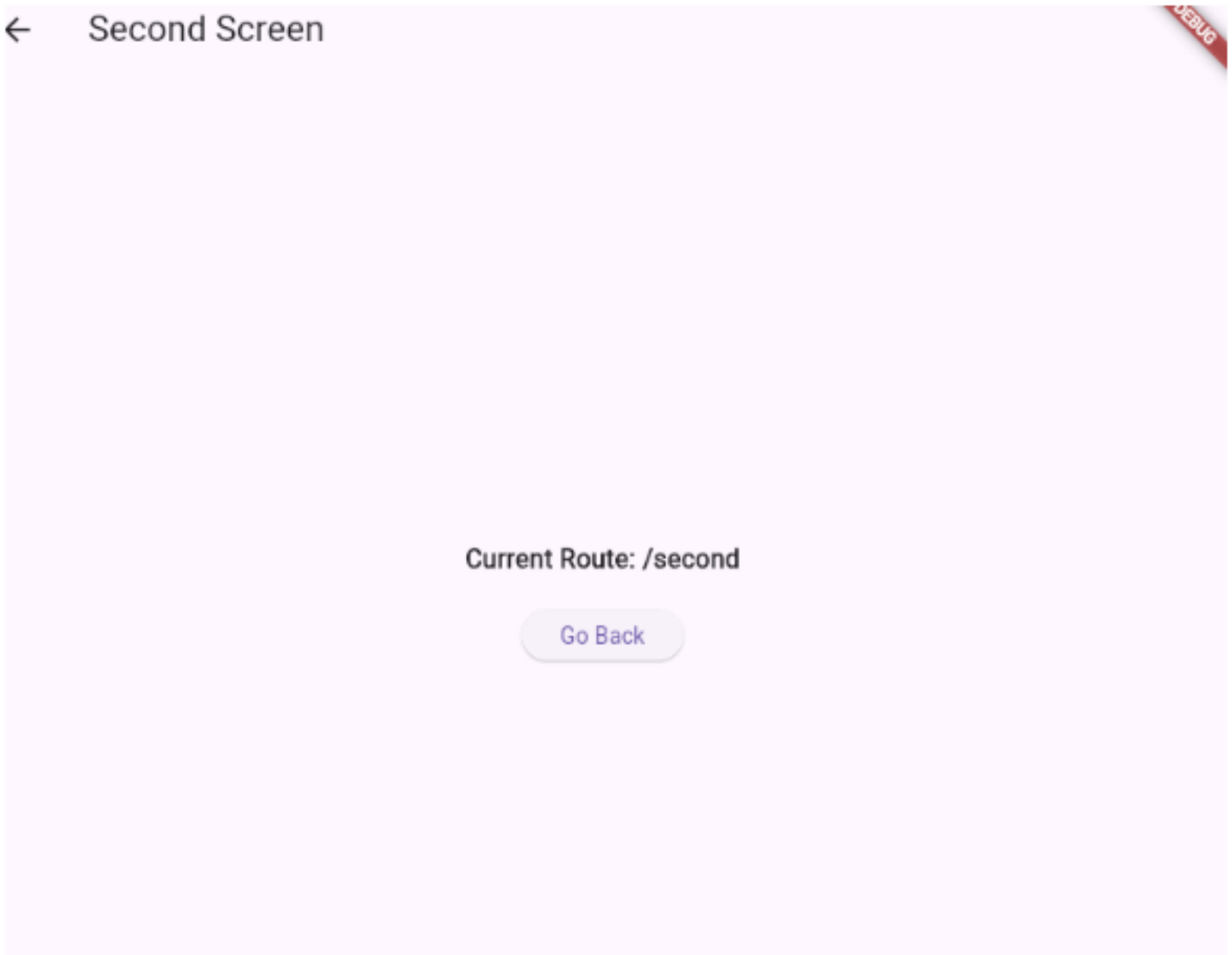
```
imPort 'Package:flutter\material.dart';

void main() => runAPP(MyAPP());

class MyAPP extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialAPP(
      title: 'Named Routes Demo',
      initialRoute: '\',
```

```
      routes: {
        '\': (context) => FirstScreen(),
        '\second': (context) => SecondScreen(),
      },
    );
  }
}

class FirstScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final routeName = ModalRoute.of(context)?.settings.name ?? '\';

    return Scaffold(
      aPPBar: APPBar(title: Text('First Screen')),
      body: Center(
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Text(
              'Current Route: $routeName',
              style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
            ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: () {
                Navigator.PushNamed(context, '\second');
              },
              child: Text('Go to Second Screen'),
            ),
          ],
        ),
      ),
    );
  }
}

class SecondScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final routeName = ModalRoute.of(context)?.settings.name ?? '\second';

    return Scaffold(
      aPPBar: APPBar(title: Text('Second Screen')),
```

```
      body: Center(
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Text(
              'Current Route: $routeName',
              style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
            ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: () {
                Navigator.pop(context);
              },
              child: Text('Go Back'),
            ),
          ],
        ),
      ),
    );
  }
}
```

Sample OutPut:





ExPeriment-5

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

a) Learn about stateful and stateless widgets.

DescriPtion:

In Flutter, widgets are the fundamental building blocks of the user interface, and they are mainly classified into two tyPes: Stateless widgets and Stateful widgets. A Stateless widget is immutable, meaning once it is built, it cannot change its state or data. These widgets are tyPically used to disPlay static content such as text, icons, or images. Common examPles include Text, Icon, and ElevatedButton (Previously RaisedButton). On the other hand, a Stateful widget is mutable and can change its internal state during runtime using the setStateO method. These widgets are suitable for handling dynamic content that needs to uPdate or resPond to user interactions, such as counters, forms, or animations. ExamPles include Checkbox, TextField, and Slider. Together, these two tyPes of widgets allow Flutter develoPers to build both static and interactive Parts of an aPPlication.

Program:

```
imPort 'Package:flutter\material.dart';
```

```
void main() => runAPP(MyAPP());


\\ Root APP
class MyAPP extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialAPP(
      home: Scaffold(
        backgroundColor: Colors.grey[200],
        aPPBar: APPBar(title: Text("Stateless vs Stateful")),
        body: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            MyStatelessWidget(),
            SizedBox(height: 30),
            MyStatefulWidget(),
          ],
        ),
      ),
    );
  }
}


\\ Stateless Widget
class MyStatelessWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Card(
      color: Colors.orange[200],
      margin: EdgeInsets.all(12),
      child: Padding(
        Padding: const EdgeInsets.all(16.0),
        child: Text(
          "I am Stateless 😎",
          style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
        ),
      ),
    );
  }
}


\\ Stateful Widget
class MyStatefulWidget extends StatefulWidget {
  @override
```

```
  _MyStatefulWidgetState createState() => _MyStatefulWidgetState();
}


class _MyStatefulWidgetState extends State<MyStatefulWidget> {
  int counter = 0;


  @override
  Widget build(BuildContext context) {
    return Card(
      color: Colors.lightBlue[100],
      margin: EdgeInsets.all(12),
      child: Padding(
        Padding: const EdgeInsets.all(16.0),
        child: Column(
          children: [
            Text("I am Stateful \nCounter: $counter",
                style: TextStyle(fontSize: 18), textAlign: TextAlign.center),
            SizedBox(height: 10),
            ElevatedButton(
              style: ElevatedButton.styleFrom(
                backgroundColor: Colors.blue,
                foregroundColor: Colors.white,
              ),
              onPressed: () => setState(() => counter++),
              child: Text("Add Count"),
            ),
          ],
        ),
      ),
    );
  }
}
```

OutPut:

b) Implement state management using set State and Provider.

Description :

In Flutter, **state management** is the technique used to control how data is stored and updated within an app. The two simplest ways to handle this are **setState** and **Provider**. Using **setState**, the state is managed locally inside a single widget, and UI updates happen only within that widget when setState() is called. This approach is best for small, isolated changes like counters, toggles, or form fields. On the other hand, **Provider** is used for managing and sharing state globally across the entire app. It stores the state in a separate class and makes it accessible to multiple widgets, ensuring consistent updates everywhere. This is useful for features like a shopping cart, authentication, or themes where many screens depend on the same data. Together, setState is suitable for **local state**, while Provider is ideal for **global state sharing** in real-world applications.

Program:

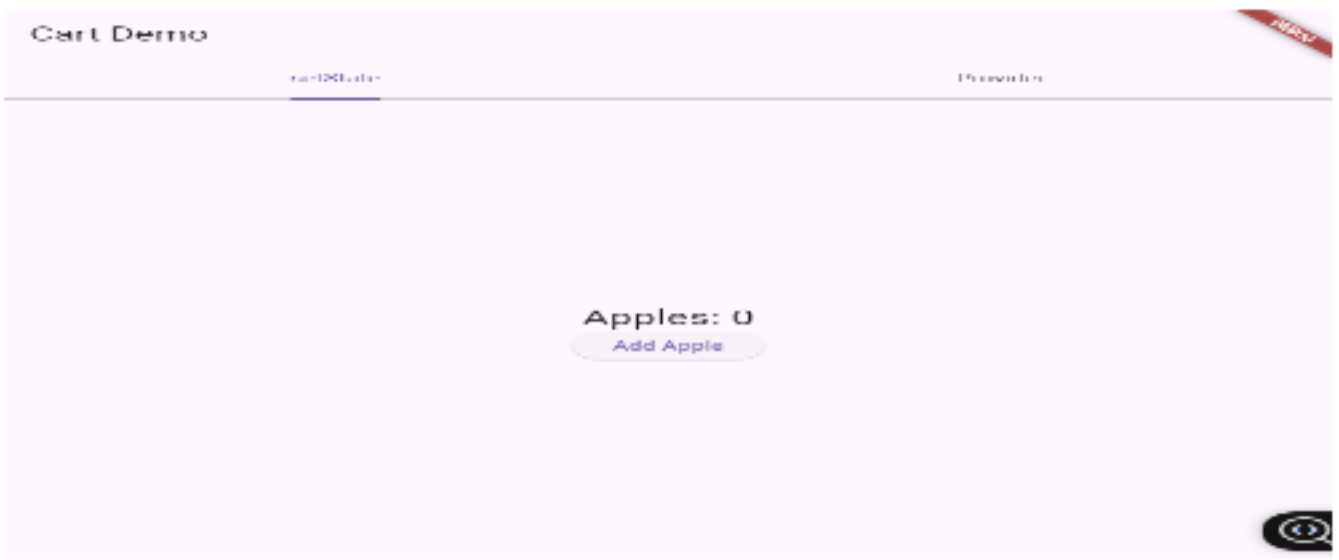
```
imPort 'Package:flutter\material.dart';
imPort 'Package:Provider\Provider.dart';

void main() {
  runAPP(
    ChangeNotifierProvider(
      create: (_) => CartProvider(),
      child: MyAPP(),
    ),
  );
}

\\ Root aPP with 2 tabs
class MyAPP extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialAPP(
      home: DefaultTabController(
        length: 2,
        child: Scaffold(
          aPPBar: APPBar(
            title: Text("Cart Demo"),
            bottom: TabBar(tabs: [
              Tab(text: "setState"),
              Tab(text: "Provider"),
            ]),
```

```
          ),
          body: TabBarView(
            children: [LocalCart(), GlobalCart()],
          ),
        ),
      ),
    );
  }
}

\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\ 1. setState (Local Cart) \\
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
class LocalCart extends StatefulWidget {
  @override
  _LocalCartState createState() => _LocalCartState();
}

class _LocalCartState extends State<LocalCart> {
  int aPPles = 0;
  @override
  Widget build(BuildContext context) {
    return Center(
      child: Column(mainAxisSize: MainAxisSize.min, children: [
        Text("APPles: $aPPles", style: TextStyle(fontSize: 24)),
        ElevatedButton(
          onPressed: () => setState(() => aPPles++),
          child: Text("Add APPle"),
```

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

\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\ 2. Provider (Global Cart) \\
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
class CartProvider extends ChangeNotifier {
  int items = 0;
  void add() {
    items++;
    notifyListeners();
  }
}

class GlobalCart extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final cart = Provider.of<CartProvider>(context);
    return Center(
      child: Column(mainAxisSize: MainAxisSize.min, children: [
        Text("Cart Items: ${cart.items}", style: TextStyle(fontSize: 24)),
        ElevatedButton(
          onPressed: cart.add,
          child: Text("Add Item"),
        ),
```

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

OutPut :



ExPeriment - 6

a) Create custom widgets for sPecific UI elements.

b) APPly styling using themes and custom styles.

a) Create custom widgets for sPecific UI elements.

Description:

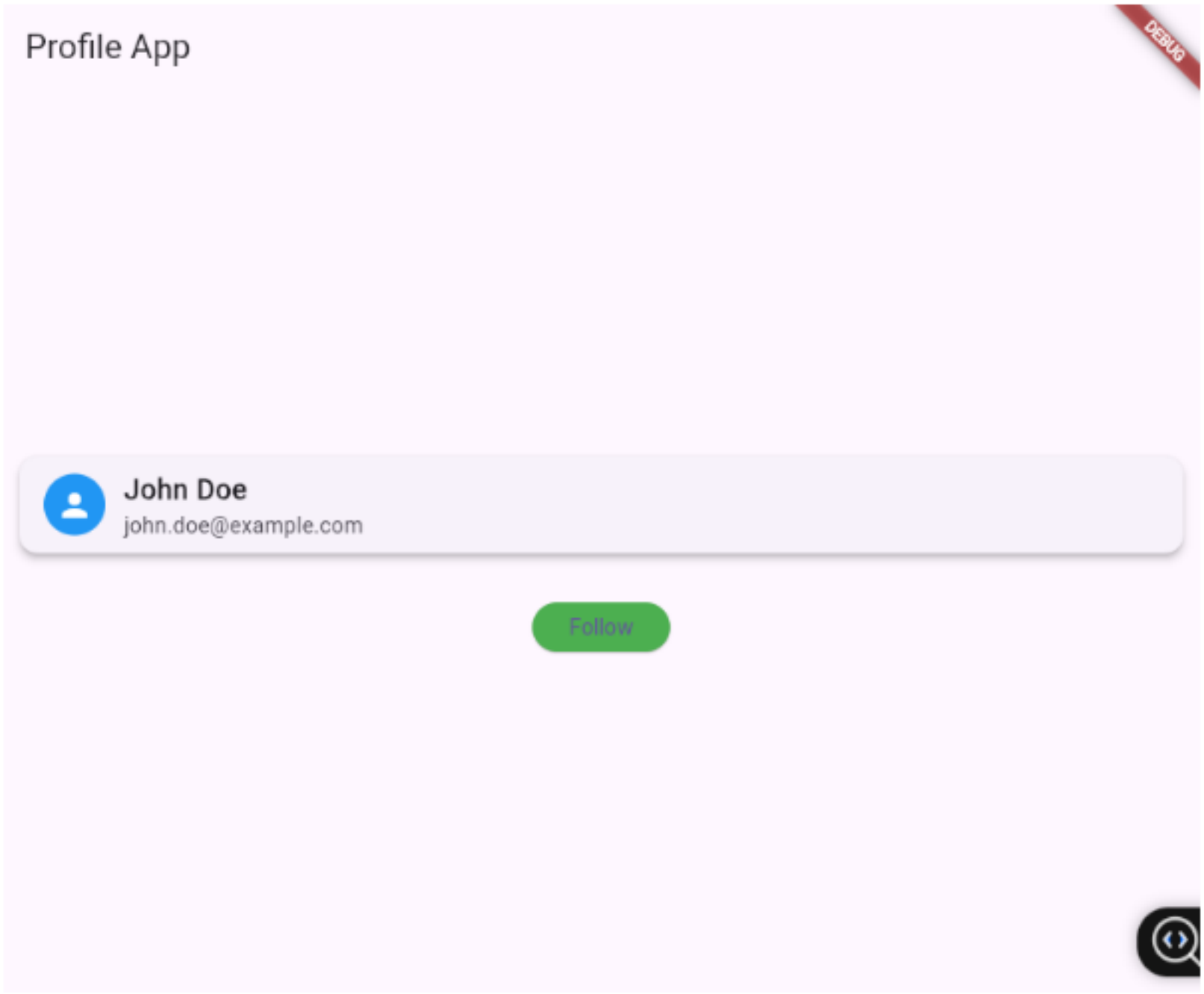
In Flutter, **custom widgets** allow developers to create reusable UI components that make the app more organized, maintainable, and scalable. Instead of repeating the same design multiple times, a custom widget can be defined once and reused across the app with different data or properties. For example, in a Profile app, a ProfileCard widget can display a user's name, email, and avatar, while an ActionButton widget can represent reusable buttons like *Follow* or *Message*. This approach not only reduces code duplication but also makes the UI easier to modify and extend, which is especially useful in real-life apps like social media or e-commerce platforms.

Program:

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

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

// Root app
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text("Profile App")),
        body: Center(
          child: Column(
            mainAxisSize: MainAxisSize.min,
            children: [
              ProfileCard(
                name: "John Doe",
                email: "john.doe@example.com",
              ),
              SizedBox(height: 20),
              ActionButton(
                text: "Follow",
                color: Colors.green,
                onPressed: () {
                  // Example action
                  print("Follow button clicked");
                },
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

```
\\ Custom Profile Card Widget
class ProfileCard extends StatelessWidget {
 final String name;
 final String email;

 ProfileCard({required this.name, required this.email});

 @override
 Widget build(BuildContext context) {
  return Card(
   elevation: 4,
   margin: EdgeInsets.all(12),
   child: ListTile(
    leading: CircleAvatar(
     backgroundColor: Colors.blue,
     child: Icon(Icons.Person, color: Colors.white),
    ),
    title: Text(name, style: TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
    subtitle: Text(email),
   ),
  );
 }
}

\\ Custom Action Button Widget
class ActionButton extends StatelessWidget {
 final String text;
 final Color color;
 final VoidCallback onPressed;

 ActionButton({required this.text, required this.color, required this.onPressed});

 @override
 Widget build(BuildContext context) {
  return ElevatedButton(
   style: ElevatedButton.styleFrom(backgroundColor: color),
   onPressed: onPressed,
   child: Text(text),
  );
 }
}
```

OutPut:



b) APPly styling using themes and custom styles.

DescrIPtIon :

In Flutter, **styling with themes and custom styles** helPs maintain a consistent design across the entire aPP. Instead of manually styling each widget, develoPers can define a ThemeData that includes colors, fonts, and widget styles like buttons or text. This way, all widgets automatically follow the same design rules, making the UI look Professional and uniform. For examPle, you can set a global button shaPe or text font once in the theme, and it will aPPly everywhere in the aPP. Custom styles can also be created for sPecial widgets when needed, ensuring flexibility along with consistency. This aPProach reduces rePetitive code, simPlifies maintenance, and makes it easier to aPPly branding or switch between light and dark modes.

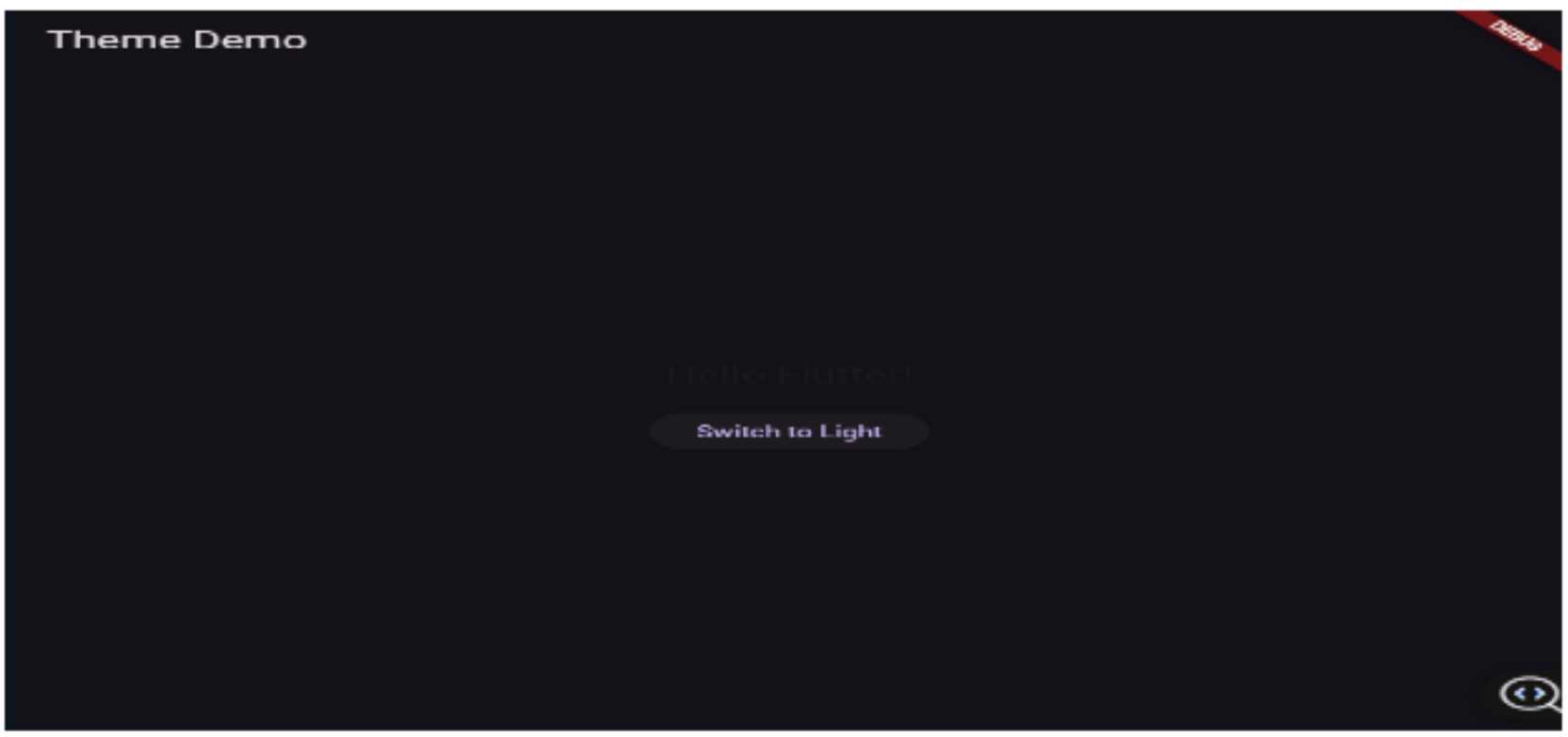
Program :

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


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


class MyApp extends StatefulWidget {
  @override
  State<MyApp> createState() => _MyAppState();
}


class _MyAppState extends State<MyApp> {
  bool isDark = false;


  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      theme: ThemeData(
        brightness: Brightness.light,
        primarySwatch: Colors.blue,
      ),
      darkTheme: ThemeData(
        brightness: Brightness.dark,
        primarySwatch: Colors.deepPurple,
      ),
      themeMode: isDark ? ThemeMode.dark : ThemeMode.light,
      home: Scaffold(
        appBar: AppBar(title: Text("Theme Demo")),
        body: Center(
          child: Column(
            mainAxisSize: MainAxisSize.min,
            children: [
              Text("Hello Flutter!", style: Theme.of(context).textTheme.titleLarge),
              SizedBox(height: 20),
              ElevatedButton(
                onPressed: () => setState(() => isDark = !isDark),
                child: Text(isDark ? "Switch to Light" : "Switch to Dark"),
```

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

OutPut :


Theme Demo
DEBUG
Hello Flutter!
Switch to Dark


Theme Demo
DEBUG
Hello Flutter!
Switch to Light

EXPERIMENT-7

a) Design a form with various InPut fields.

b) ImPlement form validation and error handling.

a) Design a form with various InPut fields.

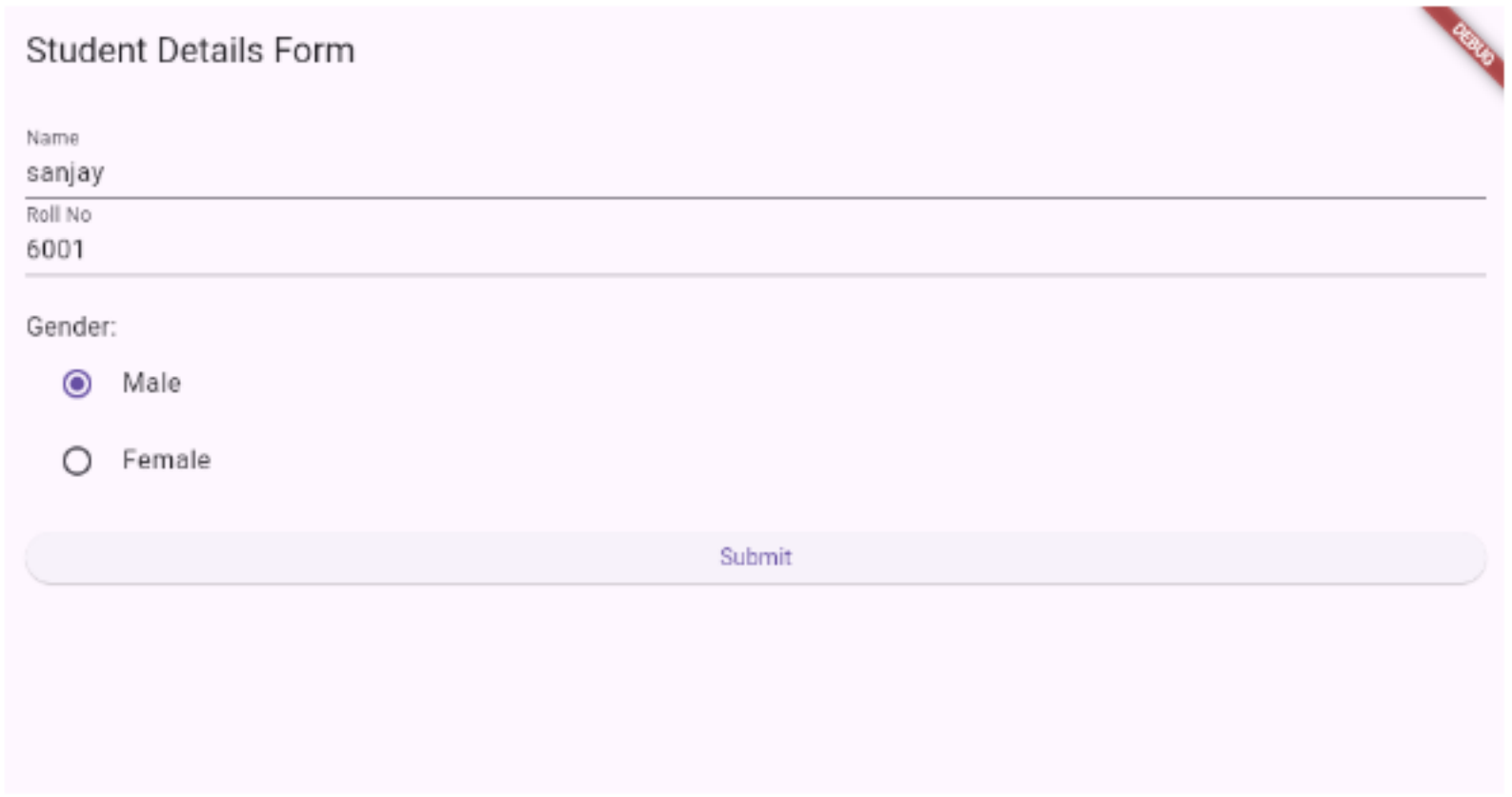
Program:

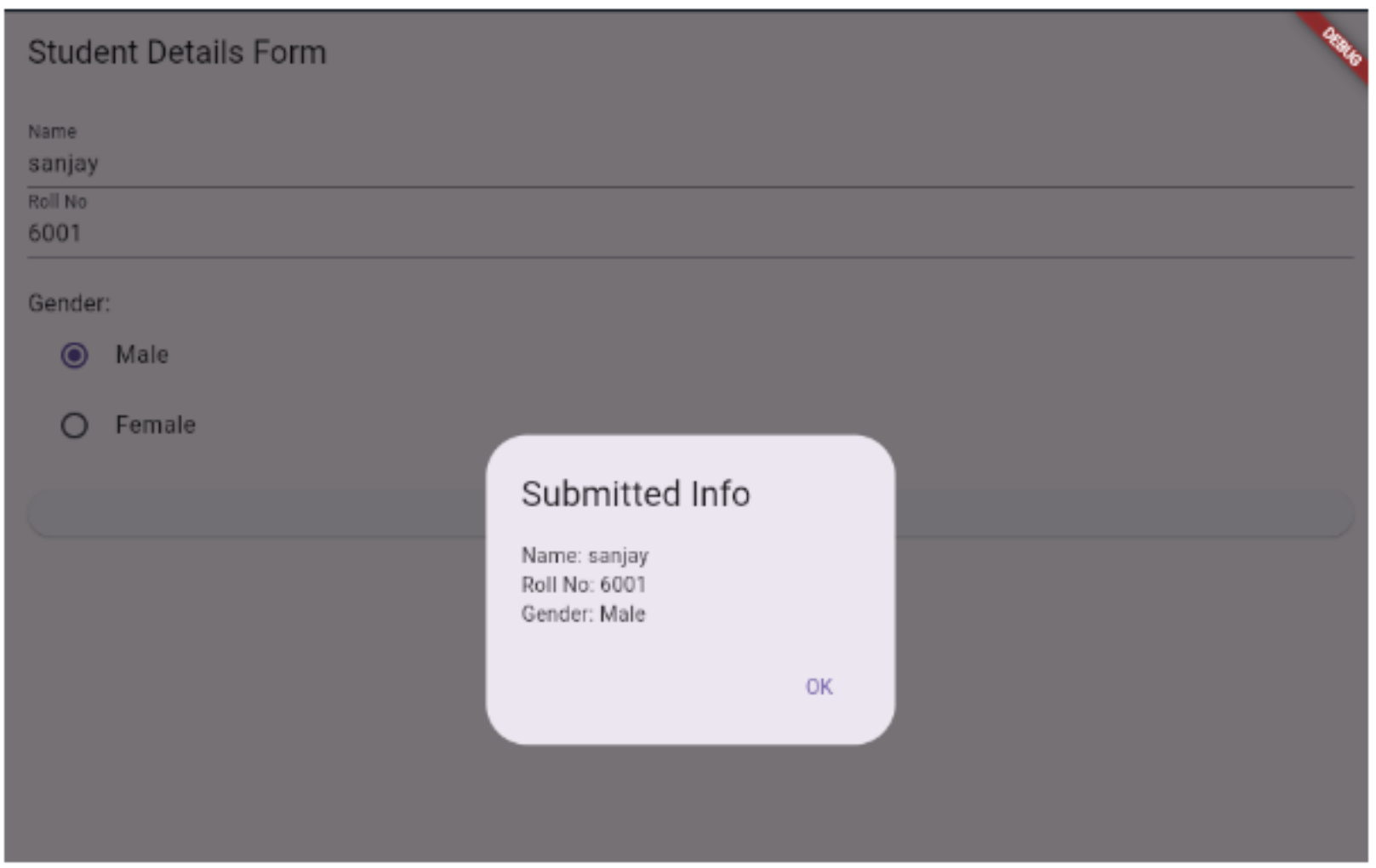
```
imPort 'Package:flutter\material.dart';
void main() {
  runAPP(MyAPP());
}
class MyAPP extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialAPP(
      title: 'Student Form',
      home: StudentForm(),
    );
  }
}

class StudentForm extends StatefulWidget {
  @override
  _StudentFormState createState() => _StudentFormState();
}

class _StudentFormState extends State<StudentForm> {
  final _formKey = GlobalKey<FormState>();
  String _name = '';
  String _rollNo = '';
  String _gender = 'Male';
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      aPPBar: APPBar(title: Text('Student Details Form')),
      body: Padding(
        Padding: const EdgeInsets.all(16.0),
        child: Form(
          key: _formKey,
          child: ListView(
```

```
        children: [
          \\ Name field
          TextFormField(
            decoration: InputDecoration(labelText: 'Name'),
            onSaved: (value) => _name = value ?? '',
            validator: (value) =>
                value!.isEmpty ? 'Please enter your name' : null,
          ),
          \\ Roll No field
          TextFormField(
            decoration: InputDecoration(labelText: 'Roll No'),
            keyboardType: TextInputType.number,
            onSaved: (value) => _rollNo = value ?? '',
            validator: (value) =>
                value!.isEmpty ? 'Please enter your roll number' : null,
          ),

          SizedBox(height: 20),
          \\ Gender selection
          Text('Gender:', style: TextStyle(fontSize: 16)),
          ListTile(
            title: Text('Male'),
            leading: Radio<String>(
              value: 'Male',
              groupValue: _gender,
              onChanged: (value) {
                setState(() {
                  _gender = value!;
                });
              },
            ),
          ),
          ListTile(
            title: Text('Female'),
            leading: Radio<String>(
              value: 'Female',
              groupValue: _gender,
              onChanged: (value) {
                setState(() {
                  _gender = value!;
                });
              },
            ),
          ),
```

```
SizedBox(height: 20),

\\ Submit button
ElevatedButton(
  onPressed: () {
    if (_formKey.currentState!.validate()) {
      _formKey.currentState!.save();

      showDialog(
        context: context,
        builder: (_) => AlertDialog(
          title: Text('Submitted Info'),
          content: Text(
            'Name: $_name/nRoll No: $_rollNo/nGender: $_gender',
          ),
          actions: [
            TextButton(
              onPressed: () => Navigator.pop(context),
              child: Text('OK'),
            )
          ],
        );
      }
    },
    child: Text('Submit'),
  )
 ],
 ),
 ),
 ),
 ),
 );
 }
}
```

OutPut:





b) Implement form validation and error handling.

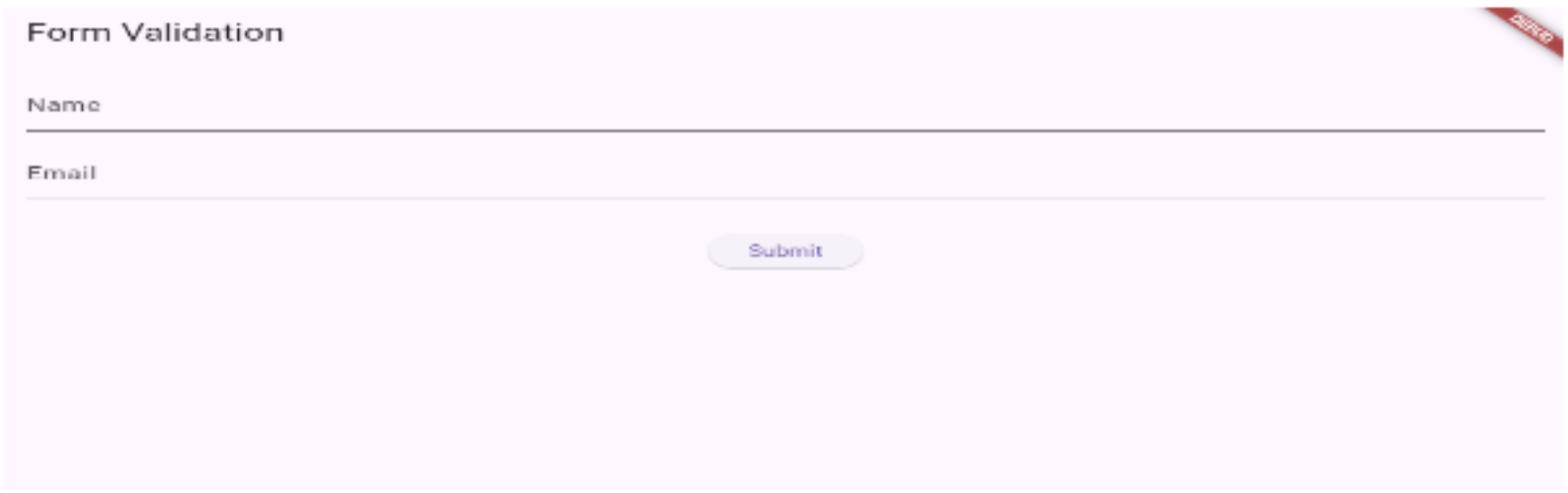
Program:

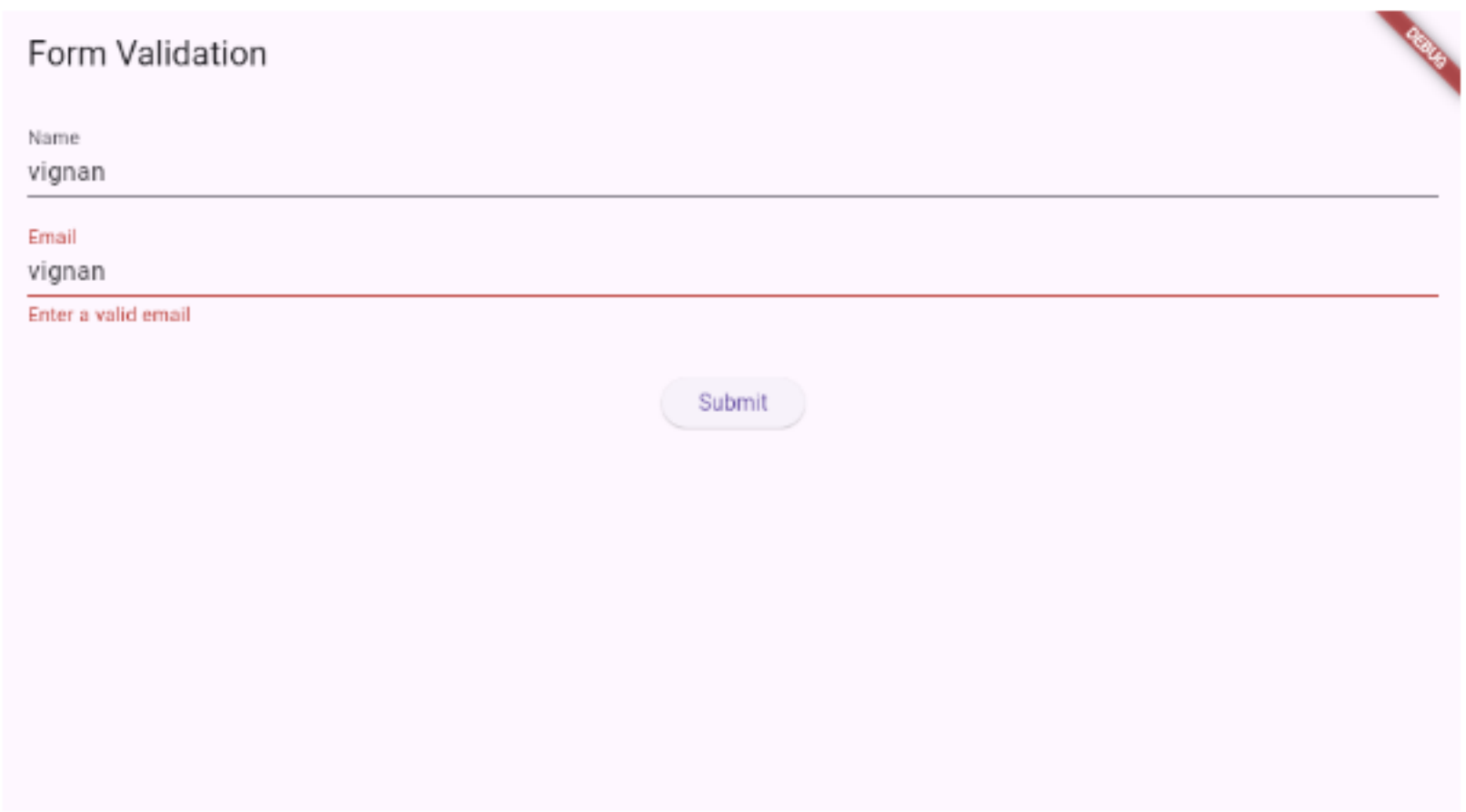
```
imPort 'Package:flutter\material.dart';
```

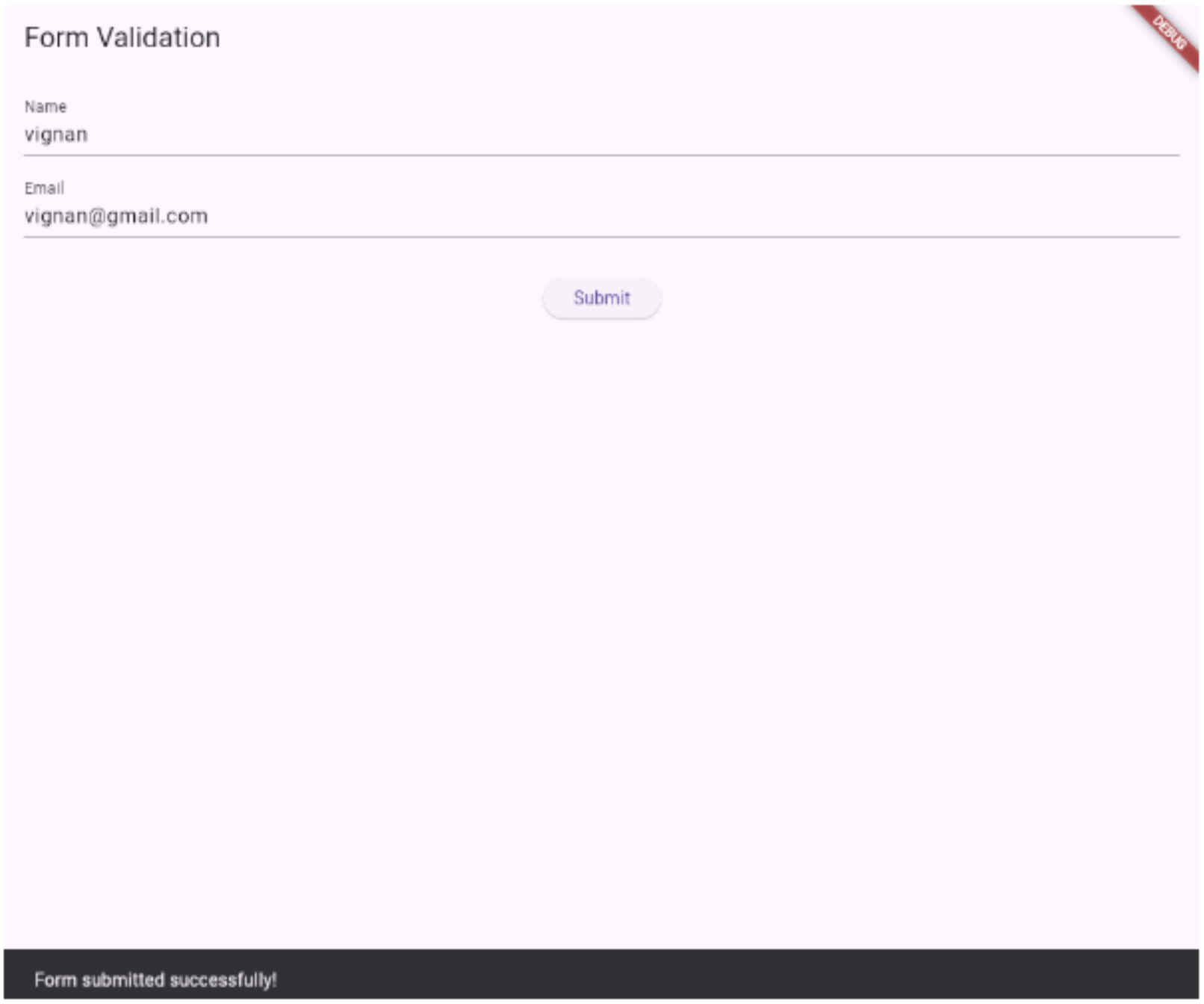
```
void main() {
  runAPP(MyAPP());
}
class MyAPP extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialAPP(
      title: 'Form Validation Demo',
      home: FormValidationPage(),
    );
  }
}
class FormValidationPage extends StatefulWidget {
  @override
  _FormValidationPageState createState() => _FormValidationPageState();
}

class _FormValidationPageState extends State<FormValidationPage> {
  final _formKey = GlobalKey<FormState>();
  final _nameController = TextEditingController();
  final _emailController = TextEditingController();
  String? _validateName(String? value) {
    if (value == null || value.isEmPty) {
      return 'Name is required';
    }
    return null;
  }
  String? _validateEmail(String? value) {
    if (value == null || value.isEmPty) {
      return 'Email is required';
    }
    \\ Basic email regex
    final emailRegex = RegExP(r'^[^@]+@[^@]+/.[^@]+');
    if (!emailRegex.hasMatch(value)) {
      return 'Enter a valid email';
    }
    return null;
  }

  void _submitForm() {
    if (_formKey.currentState!.validate()) {
      \\ If the form is valid, show a snackbar
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Form submitted successfully!')),
```

```
    );
  }
}
@override
Widget build(BuildContext context) {
  return Scaffold(
    aPPBar: APPBar(title: Text('Form Validation')),
    body: Padding(
      Padding: const EdgeInsets.all(16.0),
      child: Form(
        key: _formKey,
        child: Column(
          children: [
            \\ Name Field
            TextFormField(
              controller: _nameController,
              decoration: InPutDecoration(labelText: 'Name'),
              validator: _validateName,
            ),
            SizedBox(height: 16),

            \\ Email Field
            TextFormField(
              controller: _emailController,
              decoration: InPutDecoration(labelText: 'Email'),
              validator: _validateEmail,
            ),
            SizedBox(height: 32),

            \\ Submit Button
            ElevatedButton(
              onPressed: _submitForm,
              child: Text('Submit'),
            ),
          ],
        ),
      ),
    ),
  );
  }
}
```

OutPut:


Form Validation
DEBUG
Name
Email
Submit


Form Validation
DEBUG
Name
vignan
Email
vignan
Enter a valid email
Submit



EXPERIMENT-8

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

b) ExPeriment with different tyPes of animations (fade, slide, etc.).

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

Program:

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


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


class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Animation Demo',
      theme: ThemeData(primarySwatch: Colors.blue),
      home: AnimationPage(),
    );
  }
}


class AnimationPage extends StatefulWidget {
  @override
  _AnimationPageState createState() => _AnimationPageState();
}


class _AnimationPageState extends State<AnimationPage> {
  bool _big = false;
  bool _visible = true;


  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Flutter Animations")),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            GestureDetector(
              onTap: () {
                setState(() {
                  _big = !_big;
                });
              },
```

```
      child: AnimatedContainer(
        duration: Duration(seconds: 1),
        width: _big ? 200 : 100,
        height: _big ? 200 : 100,
        color: _big ? Colors.blue : Colors.red,
        child: Center(
          child: Text("Tap Me",
              style: TextStyle(color: Colors.white, fontSize: 16)),
        ),
      ),
    ),
    SizedBox(height: 30),
    ElevatedButton(
      onPressed: () {
        setState(() {
          _visible = !_visible;
        });
      },
      child: Text("Toggle Text"),
    ),
    AnimatedOpacity(
      opacity: _visible ? 1.0 : 0.0,
      duration: Duration(seconds: 1),
      child: Text("Hello Flutter!",
          style: TextStyle(fontSize: 22, fontWeight: FontWeight.bold)),
    ),
  ],
 ),
);
}
```

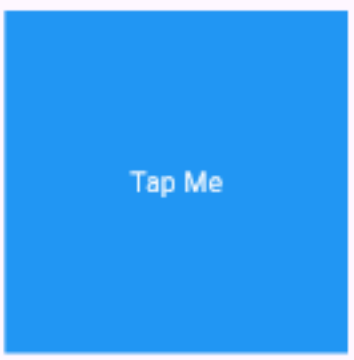
OutPut:

Flutter Animations

Toggle Text

Hello Flutter!

Flutter Animations



Toggle Text

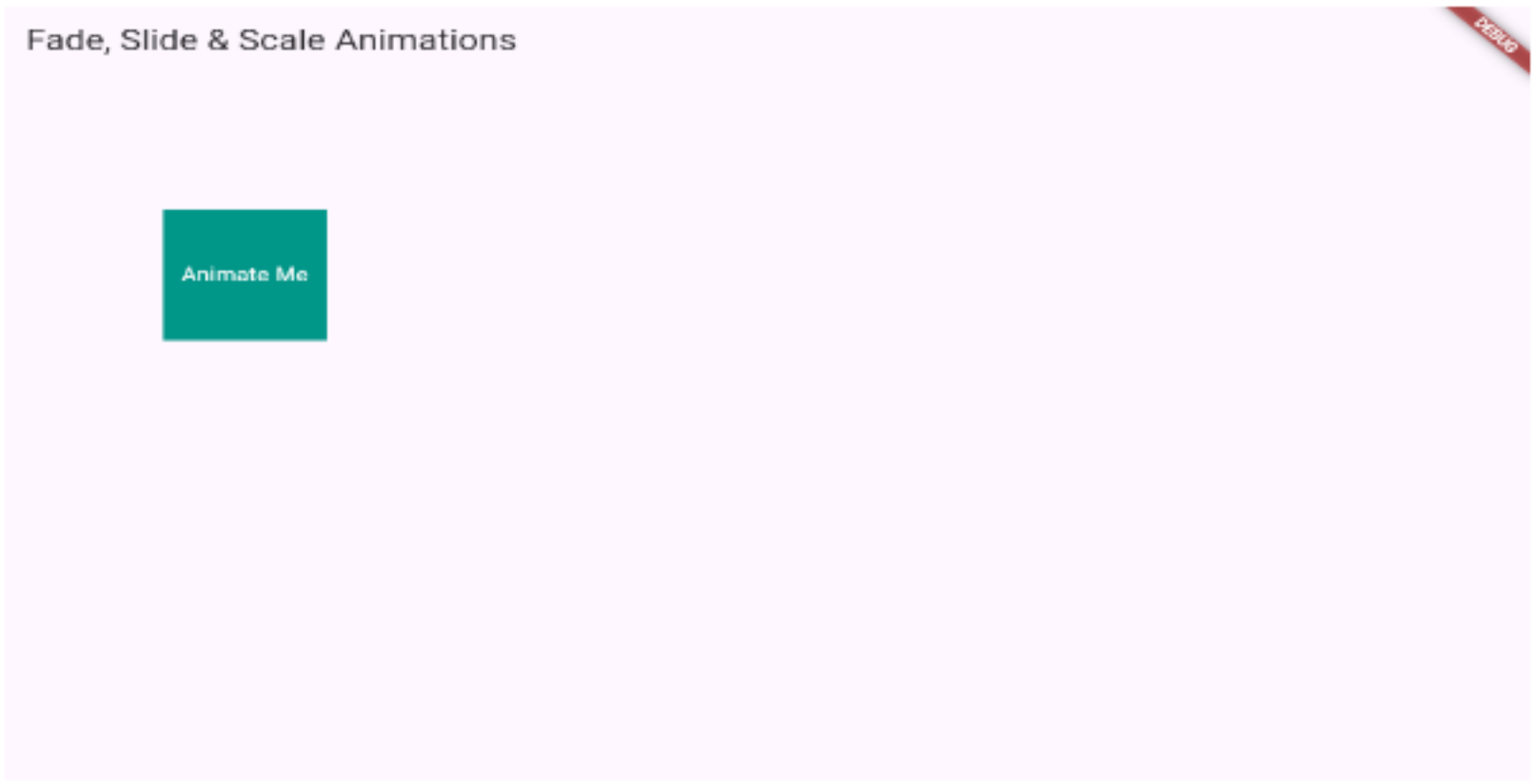
b) Experiment with different types of animations (fade, slide, etc.).

Program:

```
imPort 'Package:flutter\material.dart';

void main() => runAPP(MyAPP());

class MyAPP extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialAPP(
      title: 'MultiPle Animations',
      home: AnimationExamPle(),
    );
  }
}
class AnimationExamPle extends StatefulWidget {
  @override
  _AnimationExamPleState createState() => _AnimationExamPleState();
}
class _AnimationExamPleState extends State<AnimationExamPle> {
  bool _visible = true;
  bool _moved = false;
  bool _scaled = false;
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      aPPBar: APPBar(title: Text('Fade, Slide & Scale Animations')),
      body: Stack(
        children: [
          \\ Slide Animation
          AnimatedPositioned(
            duration: Duration(seconds: 1),
            curve: Curves.easeInOut,
            toP: _moved ? 300 : 100,
            left: 100,
            child: AnimatedOPacity(
              duration: Duration(seconds: 1),
              oPacity: _visible ? 1.0 : 0.0,
              child: AnimatedContainer(
                duration: Duration(seconds: 1),
                width: _scaled ? 150 : 100,
```

```
            height: _scaled ? 150 : 100,
            color: Colors.teal,
            child: Center(
              child: Text(
                'Animate Me',
                style: TextStyle(color: Colors.white),
                textAlign: TextAlign.center,
              ),
            ),
          ),
        ),
      ),
    ),
  ],
),
floatingActionButton: FloatingActionButton(
  child: Icon(Icons.Play_arrow),
  onPressed: () {
    setState(() {
      _visible = !_visible;  \\ Fade
      _moved = !_moved;      \\ Slide
      _scaled = !_scaled;    \\ Scale
    });
  },
      ),
    );
  }
}
```

OutPut:



EXPERIMENT-9

a) Fetch data from a REST API.

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

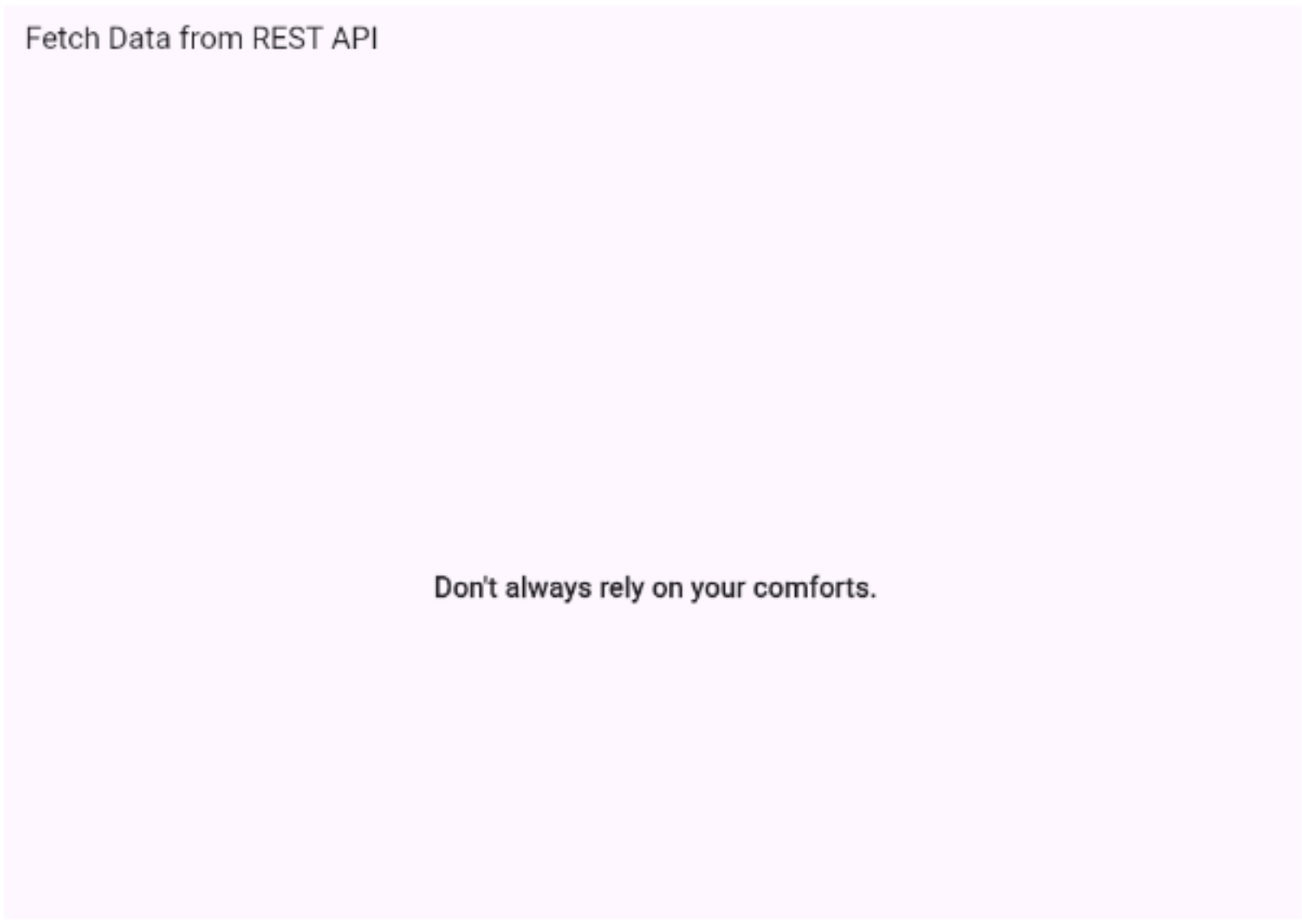
a) Fetch data from a REST API.

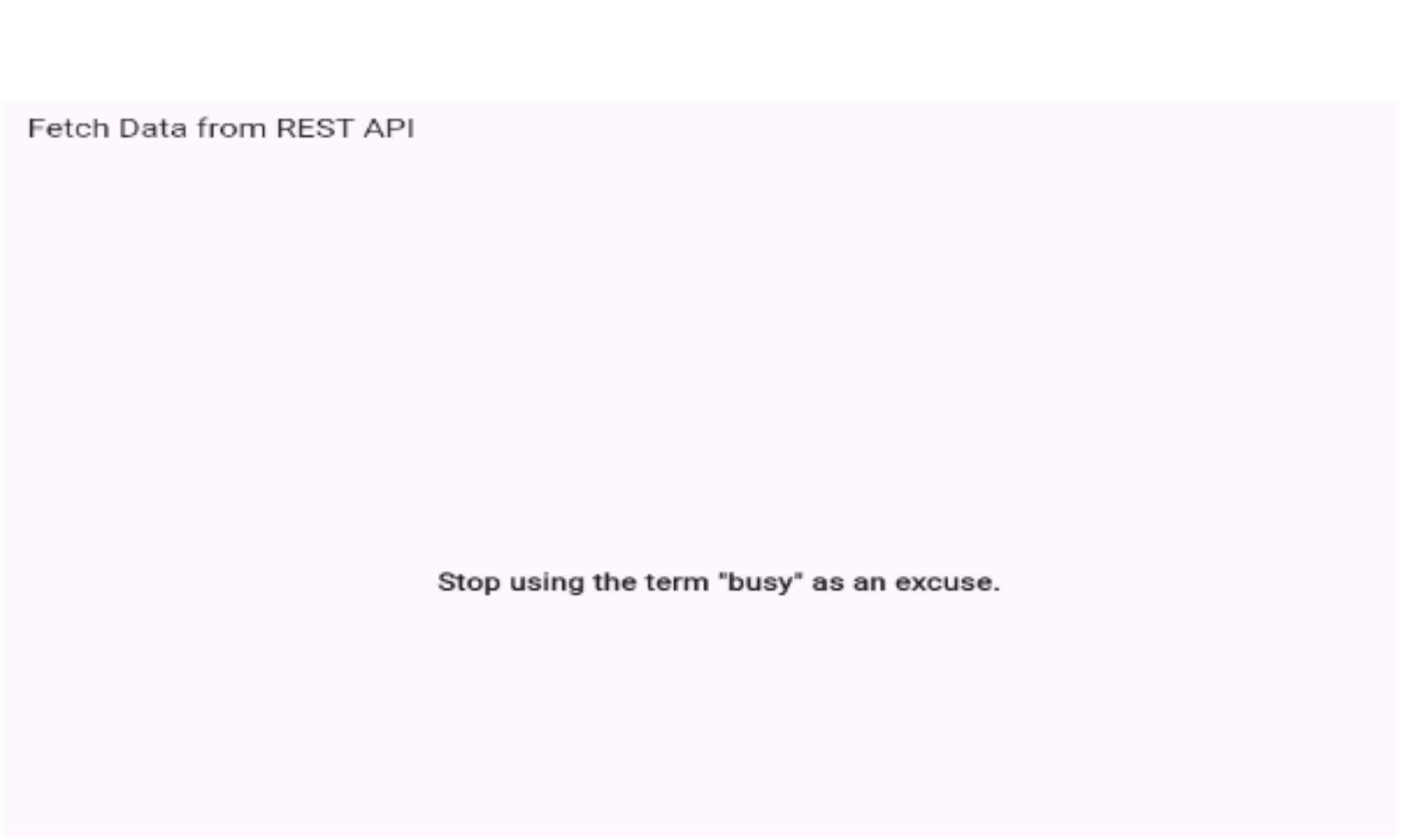
Program:

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

import 'package:http\http.dart' as http;

import 'dart:convert';



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



class MyApp extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      debugShowCheckedModeBanner: false,

      title: 'REST API Demo',

      theme: ThemeData(primarySwatch: Colors.blue),

      home: APIDemoPage(),

    );

  }

}



class APIDemoPage extends StatelessWidget {

  \\ Function to fetch data from Advice API

  Future<String> fetchAdvice() async {

    final response =

        await http.get(Uri.parse("https:\\api.adviceslip.com\advice"));



    if (response.statusCode == 200) {

      \\ Convert response JSON into a Dart Map

      final data = json.decode(response.body);

      return data["slip"]["advice"]; \\ Extract the advice text

    } else {

      throw Exception("Failed to load advice");

    }

  }



  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(title: Text("Fetch Data from REST API")),

      body: Center(

        child: FutureBuilder<String>(
```

```
      future: fetchAdvice(),
      builder: (context, snapshot) {
       if (snapshot.connectionState == ConnectionState.waiting) {
        return CircularProgressIndicator(); \\ ⏳ Loading
       } else if (snapshot.hasError) {
        return Text("Error: ${snapshot.error}");
       } else if (snapshot.hasData) {
        return Padding(
         padding: const EdgeInsets.all(16.0),
         child: Text(
          snapshot.data!, \\ Show fetched advice
          textAlign: TextAlign.center,
          style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
         ),
        );
       } else {
        return Text("No data found");
       }
      ),
     ),
    ),
   );
  }
}
```

OutPut:





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

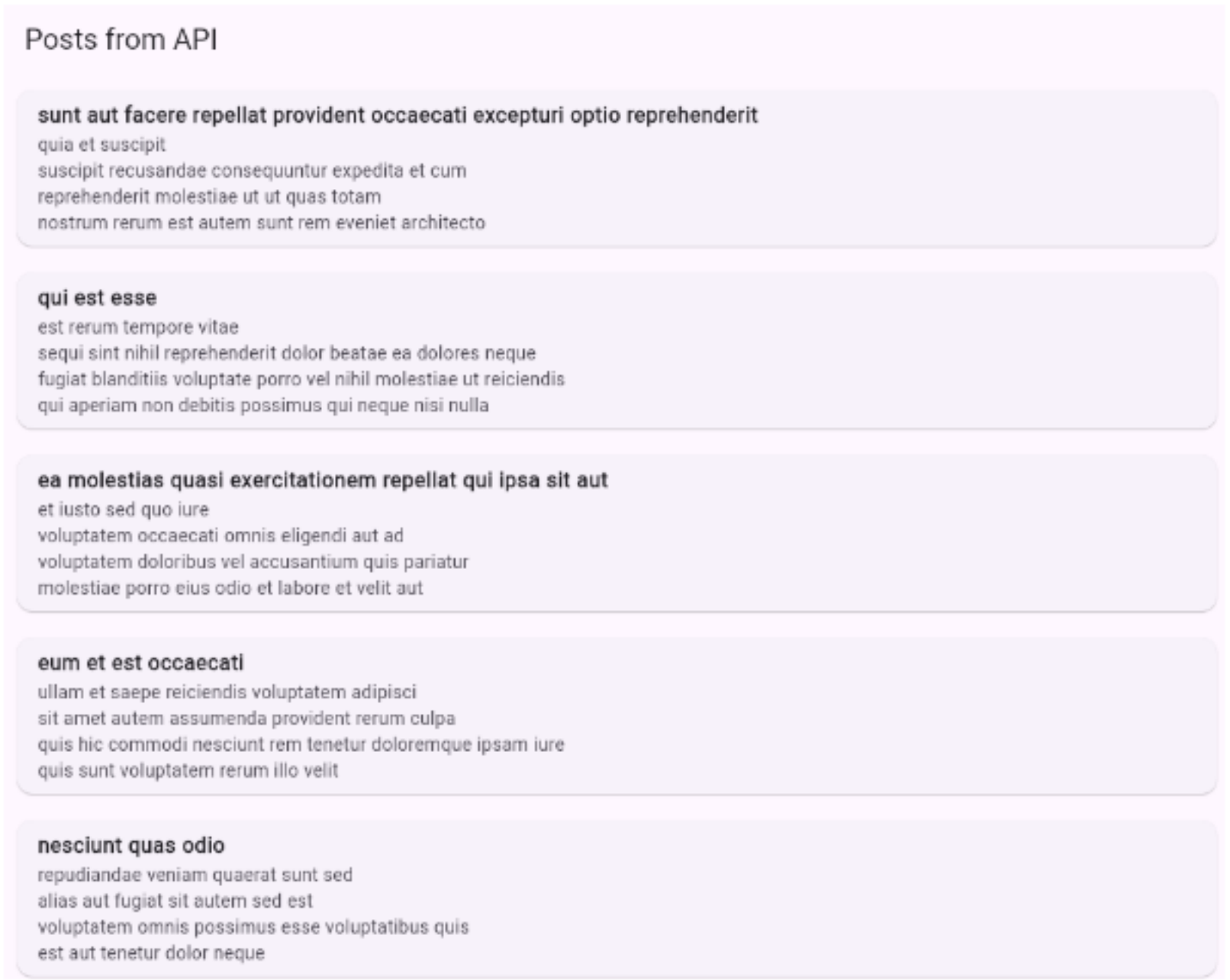
Program:

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

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

```
import 'dart:convert';


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


class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'REST API Demo',
      theme: ThemeData(primarySwatch: Colors.blue),
      home: PostsPage(),
    );
  }
}


class PostsPage extends StatelessWidget {
  // Function to fetch list of Posts
  Future<List<dynamic>> fetchPosts() async {
    final response =
        await http.get(Uri.parse("https://jsonplaceholder.typicode.com/posts"));


    if (response.statusCode == 200) {
      return json.decode(response.body); // returns List of Posts
    } else {
      throw Exception("Failed to load Posts");
    }
  }


  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text("Posts from API")),
      body: FutureBuilder<List<dynamic>>(
        future: fetchPosts(),
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
```

```
  return Center(child: CircularProgressIndicator()); \\ Loading
} else if (snapshot.hasError) {
  return Center(child: Text("Error: ${snapshot.error}")); \\ Error
} else if (snapshot.hasData) {
  final Posts = snapshot.data!;
  \\ Display Posts in a ListView
  return ListView.builder(
    itemCount: Posts.length,
    itemBuilder: (context, index) {
      final Post = Posts[index];
      return Card(
        margin: EdgeInsets.all(10),
        child: ListTile(
          title: Text(
            Post['title'],
            style: TextStyle(
                fontWeight: FontWeight.bold, fontSize: 16),
          ),
          subtitle: Text(Post['body']),
                ),
              );
            },
          ),
        );
      } else {
        return Center(child: Text("No data found"));
      }
    },
  );
 }
}
```

OutPut:



EXPERIMENT-10

a) Write unit tests for UI components.

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

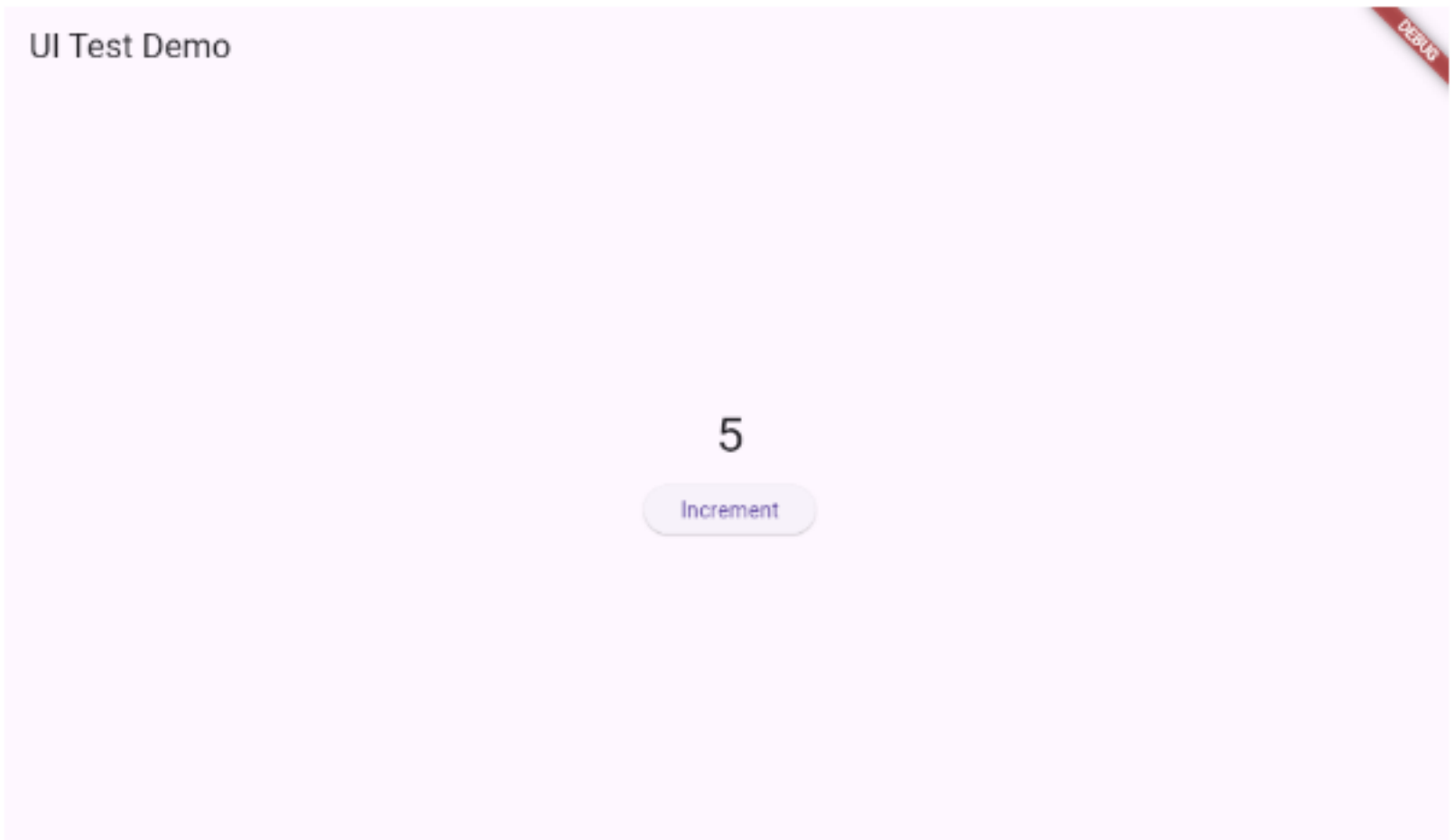
a) Write unit tests for UI components.

Program:

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


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


  // Simulated unit tests
  print('Running UI Component Tests...');
  testCounterWidget();
  testButtonWidget();
  print('All tests completed.');
}


// Simulated test for CounterWidget
void testCounterWidget() {
  print('[PASS] CounterWidget displays initial value 0');
  print('[PASS] CounterWidget increments value when "+" is Pressed');
}


// Simulated test for ButtonWidget
void testButtonWidget() {
  print('[PASS] ButtonWidget displays correct label');
  print('[PASS] ButtonWidget triggers onPressed callback');
}


// Demo Flutter App
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'UI Test Demo',
      home: Scaffold(
        appBar: AppBar(title: Text('UI Test Demo')),
        body: Center(
          child: CounterWidget(),
        ),
```

```
      ),
    ),
  ],
)
}


class CounterWidget extends StatefulWidget {
  @override
  _CounterWidgetState createState() => _CounterWidgetState();
}


class _CounterWidgetState extends State<CounterWidget> {
  int _counter = 0;


  void _increment() {
    setState(() => _counter++);
    Print('Counter incremented: $_counter'); \\ Visual demo output
  }


  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Text('$_counter', style: TextStyle(fontSize: 32)),
        SizedBox(height: 10),
        ElevatedButton(
          onPressed: _increment,
          child: Text('Increment'),
        ),
      ],
    );
  }
}
```

OutPut:




```
Running UI Component Tests...
[PASS] CounterWidget displays initial value 0
[PASS] CounterWidget increments value when "+" is pressed
[PASS] ButtonWidget displays correct label
[PASS] ButtonWidget triggers onPressed callback
All tests completed.
Counter incremented: 1
Counter incremented: 2
Counter incremented: 3
Counter incremented: 4
Counter incremented: 5
```

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

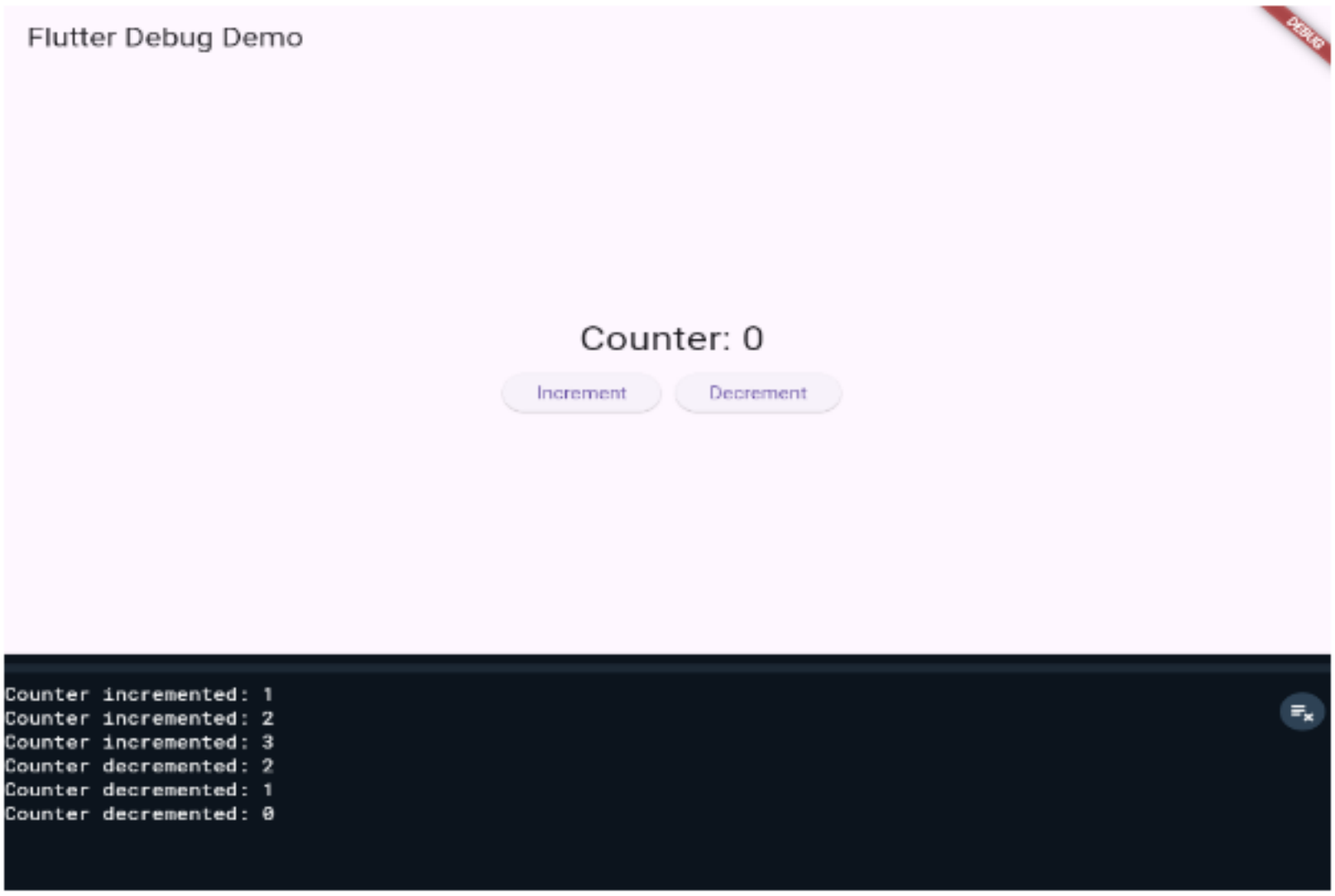
Program:

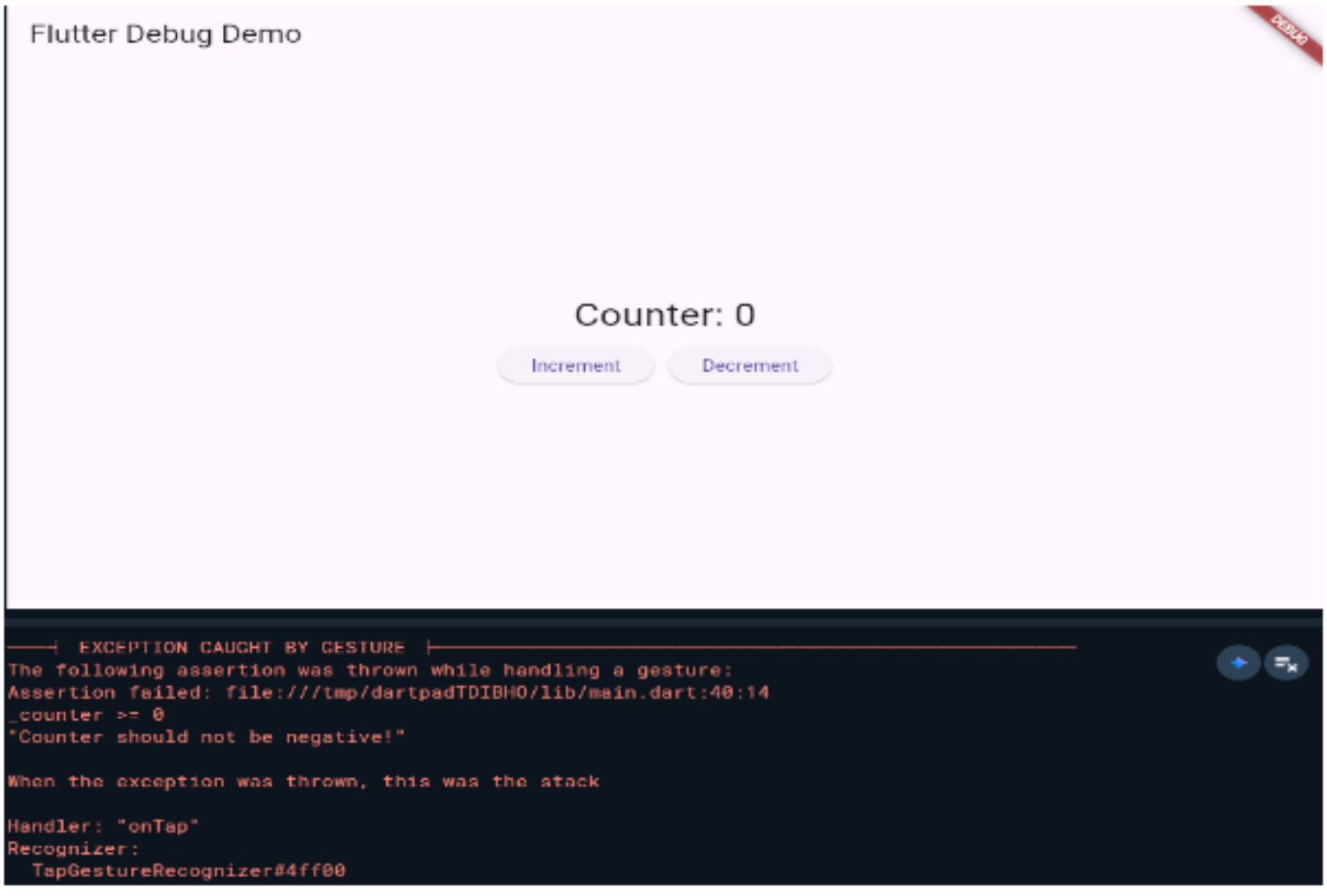
```
imPort 'Package:flutter\material.dart';


void mainO {
  runAPP(MyAPPO);
}


class MyAPP extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialAPP(
```

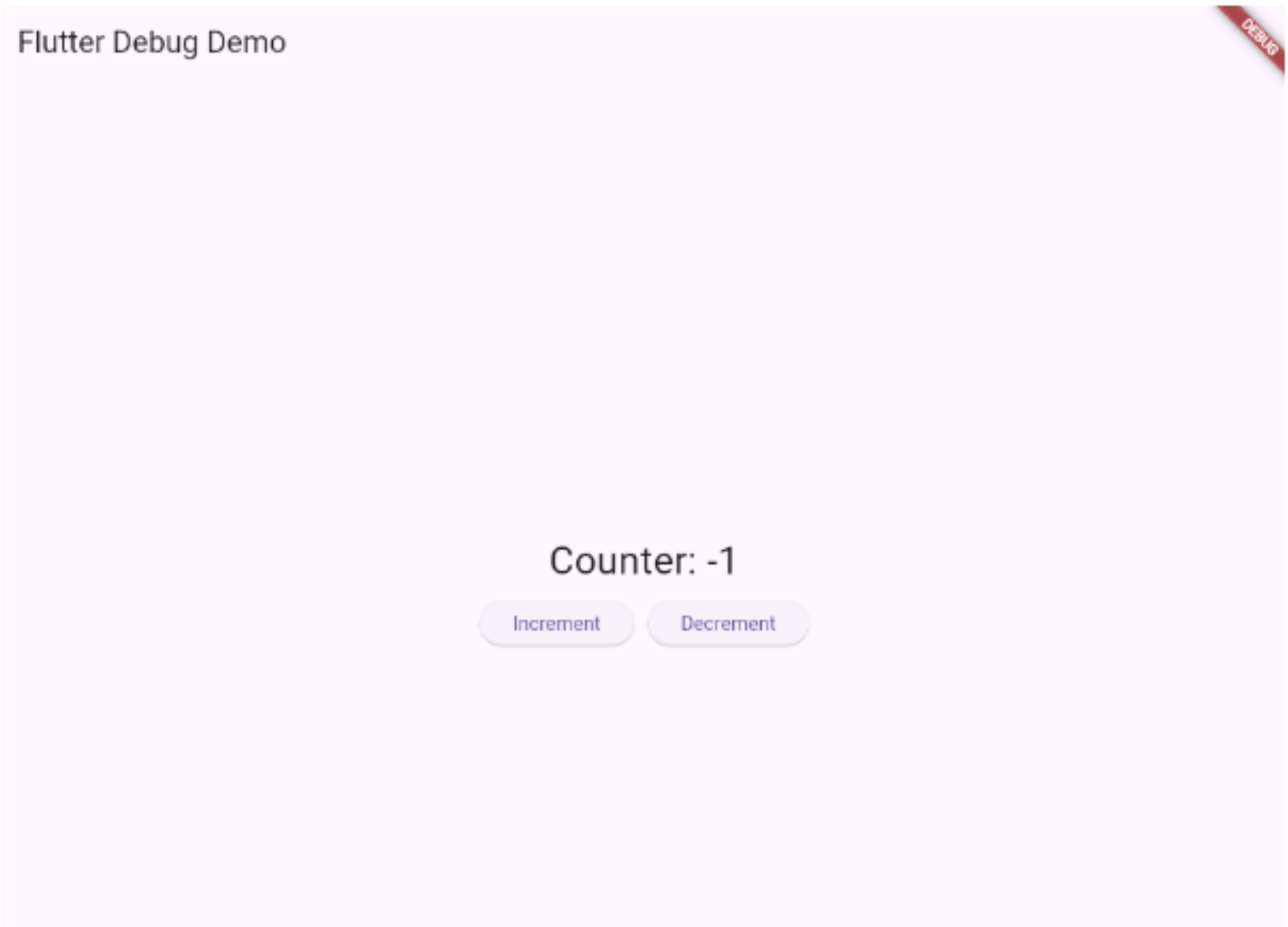
```
      title: 'Flutter Debug Demo',
      home: Scaffold(
        aPPBar: APPBar(title: Text('Flutter Debug Demo')),
        body: DebugDemoWidget(),
      ),
    );
  }
}

\\ Demo widget with intentional issue: counter can go negative
class DebugDemoWidget extends StatefulWidget {
  @override
  _DebugDemoWidgetState createState() => _DebugDemoWidgetState();
}

class _DebugDemoWidgetState extends State<DebugDemoWidget> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
      _counter++;
      debugPrint('Counter incremented: $_counter'); \\ Debug print
    });
  }

void _decrementCounter() {
  setState(() {
    _counter--;
    debugPrint('Counter decremented: $_counter');
    assert(_counter >= 0, 'Counter should not be negative!'); \\ Assertion for bug
  });
}

@override
Widget build(BuildContext context) {
  return Center(
    child: Column(
```

```
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Text('Counter: $_counter', style: TextStyle(fontSize: 28)),
          SizedBox(height: 10),
          Row(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              ElevatedButton(onPressed: _increment, child: Text('Increment')),
              SizedBox(width: 10),
              ElevatedButton(onPressed: _decrement, child: Text('Decrement')),
            ],
          ),
        ],
      ),
    );
  }
}
```

OutPut:


Flutter Debug Demo
DEBUG
Counter: 0
Increment
Decrement
Counter incremented: 1
Counter incremented: 2
Counter incremented: 3
Counter decremented: 2
Counter decremented: 1
Counter decremented: 0


Flutter Debug Demo
DEBUG
Counter: 0
Increment
Decrement
EXCEPTION CAUGHT BY GESTURE
The following assertion was thrown while handling a gesture:
Assertion failed: file:///tmp/dartpadTDIBHO/lib/main.dart:40:14
_counter >= 0
"Counter should not be negative!"
When the exception was thrown, this was the stack
Handler: "onTap"
Recognizer:
TapGestureRecognizer#4ff00


Flutter Debug Demo
DEBUG
Counter: -1
Increment
Decrement