**Expt.No.:** 1-i **Date:**

**AIM:** Install Flutter and Dart SDK.

**DESCRIPTION:**

Flutter is an open-source UI software development toolkit created by Google for building natively compiled applications for mobile (iOS, Android), web, desktop, and embedded devices from a single codebase. Flutter allows developers to write code once and deploy on multiple platforms, ensuring consistent UI and performance.

* **Framework Layer (Dart):**
  + Built with Dart language, offering a rich set of pre-built widgets.
  + Handles UI rendering, user input, gestures, animations, and state management.
  + Enables high-level, platform-agnostic UI development.
* **Flutter Engine (C++):**
  + Handles rendering using the Skia graphics library.
  + Manages animation, layout, and accesses platform-specific APIs.
  + Bridges the Dart framework with native platform capabilities (camera, GPS, etc.).
* **Embedder Layer:**
  + Platform-specific components that interface between the Flutter engine and underlying OS.
  + Ensures smooth integration on Android, iOS, web, Windows, macOS, Linux.
* **Operations:**
  + Flutter compiles Dart code ahead-of-time (AOT) into native code for efficient execution.
  + Provides a reactive framework for UI updates; changes in state trigger rebuilding of widgets.
  + Supports hot reload to instantly reflect code changes during development, enhancing productivity.

**PROCEDURE:**

## **Flutter Installation Steps (Windows Example)**

1. Download Flutter SDK:
   * Visit the official Flutter install page at <https://docs.flutter.dev/get-started/install/windows>.
   * Download the latest Flutter SDK zip file.
2. Extract the SDK:
   * Extract the downloaded zip file to a preferred location (e.g., C:\flutter).
3. Set Environment Variable:
   * Add the Flutter bin directory (e.g., C:\flutter\bin) to your system PATH environment variable.
4. Run Flutter Doctor:
   * Open a new command prompt window and run:

flutter doctor

* + This command checks your environment and shows any missing dependencies.

1. Install Android Studio and Plugins:
   * Install the latest Android Studio.
   * From Android Studio, install Flutter and Dart plugins via File > Settings > Plugins.
   * Accept the prompt to install the Dart plugin when installing Flutter plugin.
2. Set up Android Emulator or Device:
   * Start an Android emulator or connect a physical Android device with USB debugging enabled.
3. Verify Installation:
   * Run flutter doctor again to confirm everything is set up.
   * You can create a new Flutter project via flutter create my\_app and test it with flutter run.

**Expt.No.:** 1-ii **Date:**

**AIM:** Write a simple Dart program to understand the language basics

**DESCRIPTION:**

The provided Dart program demonstrates basic programming concepts such as variable declaration, control flow (if-else, switch-case, loops), functions, exception handling, object-oriented programming with classes and methods, and enums. It calculates the Fibonacci sequence recursively, performs math operations using Dart's math library, handles exceptions using try-catch blocks, and uses an enum to demonstrate conditional logic.

The program highlights Dart's strong typing, dynamic typing, and type inference, showcasing a well-rounded introduction to the language's syntax and features.

Let me know if a more detailed or section-wise explanation is needed.

**PROGRAM:**

import 'dart:math';

void main() {

int a = 10;

double b = 20.2;

num c = 10;

c = 20.2;

var d = 'a';

bool e = true;

dynamic f = 'a';

f = 10;

if (a > 10)

print("a is greater than 10");

else

print("a is less than or equal to 10");

switch (a) {

case 10:

print("ten");

case 20:

print("twenty");

}

int i = 0;

while (i <= 5) {

print("hello");

i++;

}

for (int j = 0; j < 5; j++) print("World");

int k = fibonacci(5);

print("5th fibonacci term $k");

var degrees = 30;

var radians = degrees \* (pi / 180);

var sinOf30degrees = sin(radians);

print('sin(30) value is $sinOf30degrees');

Point X = Point(10, 20);

Point Y = Point(20, 30);

print('Difference between point X and Y is ${X.difference(Y)}');

try {

dontCallMe();

} on Exception catch (e) {

print('Caught the exception - $e');

}

Color col = Color.RED;

switch (col) {

case Color.RED:

print('Red');

case Color.GREEN:

print('Green');

case Color.BLUE:

print('Blue');

}

}

int fibonacci(int n) {

if (n == 0 || n == 1) return n;

return fibonacci(n - 1) + fibonacci(n - 2);

}

class Point {

int x;

int y;

Point(this.x, this.y);

double difference(Point other) {

return sqrt(pow((x - other.x), 2) + pow((y - other.y), 2));

}

}

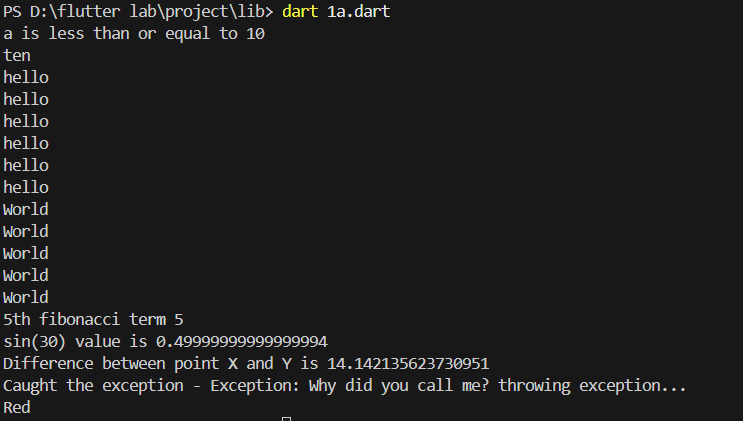
void dontCallMe() {

throw Exception('Why did you call me? throwing exception...');

}

enum Color { RED, GREEN, BLUE }

**OUTPUT:**



**Expt.No.:** 2-i **Date:**

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

**DESCRIPTION:**

1. First create a new flutter project using the following command

flutter create lab\_2a

cd lab\_2a

1. You will see the standard folder structure and basic files created for you.

Replace the lib\main.dart with the one from this lib folder.

1. You can use flutter build to build the project  
   Use the below command for android project.

flutter build apk

1. Use the below command for web project

flutter build web

1. You can use flutter run command to launch the project

flutter run

1. Emulator - launch the emulator and run the application in the emulator. Note - Applicable for VS Code, as a pre-requisite you need to first create a virtual device through android virtual device manager.

ctrl + shift + p

Flutter: Launch Emulator

choose the device you created using AVD.

**PROGRAM:**

import 'package:flutter/material.dart';

void main() {

runApp(

MaterialApp(

home: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

const Text('Hello, Flutter!'),

const Text(

'Styled Text',

style: TextStyle(

color: Colors.deepPurple,

fontSize: 24.0,

fontWeight: FontWeight.bold,

fontStyle: FontStyle.italic,

),

),

Image.network(

'https://www.gstatic.com/flutter-onestack-prototype/genui/example\_1.jpg',

width: 150,

height: 150,

fit: BoxFit.cover,

),

Container(

margin: const EdgeInsets.all(40.0),

padding: const EdgeInsets.all(40.0),

color: Colors.lightBlue[100],

child: const Text('Padded & Margined container'),

),

],

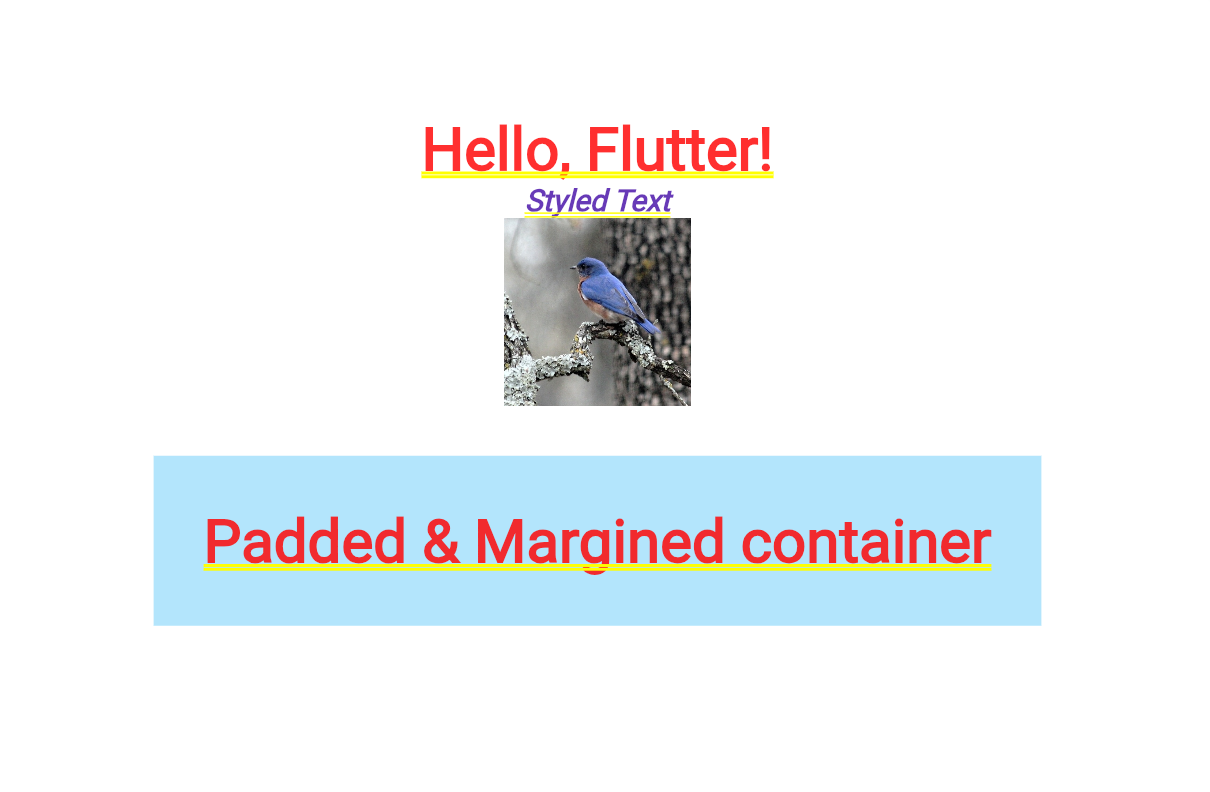
),

),

);

}

**OUTPUT:**



**Expt.No.:** 2-ii **Date:**

**AIM:** Implement different layout structures using Row, Column, and Stack widgets.

**DESCRIPTION:**

1. First create a new flutter project using the following command

flutter create lab\_2b

cd lab\_2b

1. You will see the standard folder structure and basic files created for you.

Replace the content of lib\main.dart with the one from this lib folder one at a time.

Note - This is an optional step you can directly use flutter run command. 3. You can use flutter build to build the project

Use the below command for android project.

flutter build apk

Use the below command for web project

flutter build web

1. You can use flutter run command

flutter run

1. Emulator - launch the emulator and run the application in the emulator. Note - Applicable for VS Code, as a pre-requisite you need to first create a virtual device through android virtual device manager.

ctrl + shift + p

Flutter: Launch Emulator

choose the device you created using AVD.

**PROGRAM:**

import 'package:flutter/material.dart';

void main() {

runApp(const MyApp());

}

class MyApp extends StatelessWidget {

const MyApp(); // Good idea!

@override

Widget build(BuildContext context) {

return MaterialApp(

home: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Text("Hello"),

Row(

mainAxisAlignment: MainAxisAlignment.center,

children: [Text("World"), SizedBox(width: 100), Text("!!")],

),

],

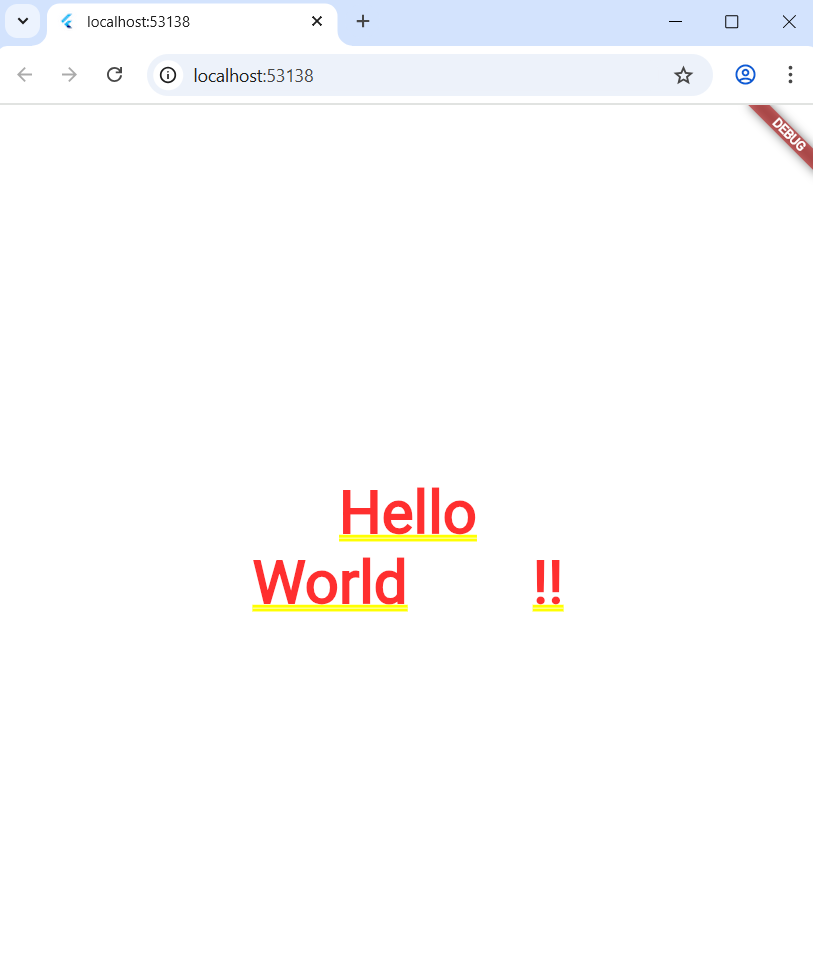
),

);

}

}

**OUTPUT:**



**Expt.No.:** 3-i **Date:**

**AIM:** Design a responsive UI that adapts to different screen size.

**DESCRIPTION:**

1. First create a new flutter project using the following command

flutter create lab\_3a

cd lab\_3a

1. You will see the standard folder structure and basic files created for you.

Replace the lib\main.dart with the one from this lib folder.

1. You can use flutter build to build the project  
   Use the below command for android project.

flutter build apk

1. Use the below command for web project

flutter build web

1. You can use flutter run command to launch the project

flutter run

1. Emulator - launch the emulator and run the application in the emulator. Note - Applicable for VS Code, as a pre-requisite you need to first create a virtual device through android virtual device manager.

ctrl + shift + p

Flutter: Launch Emulator

choose the device you created using AVD.

**PROGRAM:**

import 'package:flutter/material.dart';

void main() {

runApp(const MyApp());

}

class MyApp extends StatelessWidget {

const MyApp({super.key});

@override

Widget build(BuildContext context) {

return MaterialApp(home: const ItemListPage());

}

}

class ItemListPage extends StatelessWidget {

const ItemListPage({super.key});

@override

Widget build(BuildContext context) {

return Scaffold(

body: LayoutBuilder(

builder: (context, constraints) {

if (constraints.maxWidth < 768) {

return ListData();

}

return GridData();

},

),

);

}

}

class ListData extends StatelessWidget {

const ListData({super.key});

@override

Widget build(BuildContext context) {

return ListView.builder(

itemCount: 50,

itemBuilder: (context, id) {

return ListTile(

leading: const Icon(Icons.add\_box),

title: Text("Item $id"),

);

},

);

}

}

class GridData extends StatelessWidget {

const GridData({super.key});

@override

Widget build(BuildContext context) {

return GridView.count(

crossAxisCount: 2,

children: List.generate(50, (index) {

return Center(

child: ListTile(

leading: const Icon(Icons.add\_box),

title: Text("Item $index"),

),

);

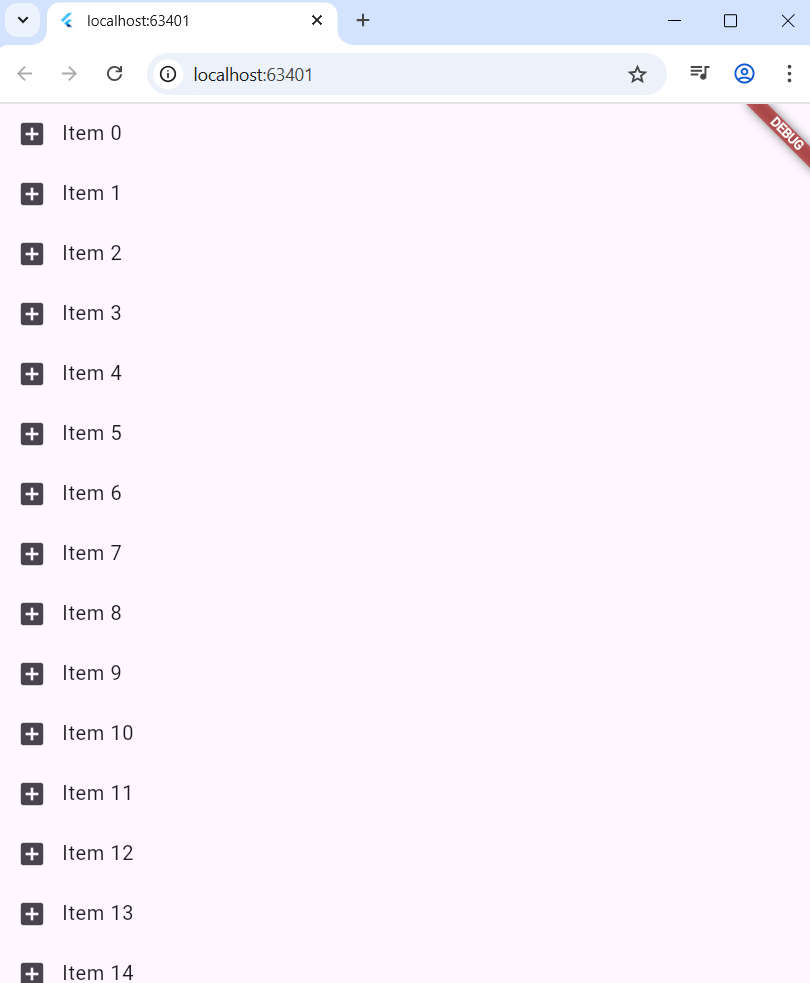
}),

);

}

}

OUTPUT:



**Expt.No.:** 3-ii **Date:**

**AIM:** Implement media queries and breakpoints for responsiveness.

**DESCRIPTION:**

1. First create a new flutter project using the following command

flutter create lab\_3b

cd lab\_3b

1. You will see the standard folder structure and basic files created for you.

Replace the lib\main.dart with the one from this lib folder.

1. You can use flutter build to build the project  
   Use the below command for android project.

flutter build apk

1. Use the below command for web project

flutter build web

1. You can use flutter run command to launch the project

flutter run

1. Emulator - launch the emulator and run the application in the emulator. Note - Applicable for VS Code, as a pre-requisite you need to first create a virtual device through android virtual device manager.

ctrl + shift + p

Flutter: Launch Emulator

choose the device you created using AVD.

**PROGRAM:**

import 'package:flutter/material.dart';

void main() {

runApp(const MyApp());

}

class MyApp extends StatelessWidget {

const MyApp({super.key});

@override

Widget build(BuildContext context) {

double width = MediaQuery.of(context).size.width;

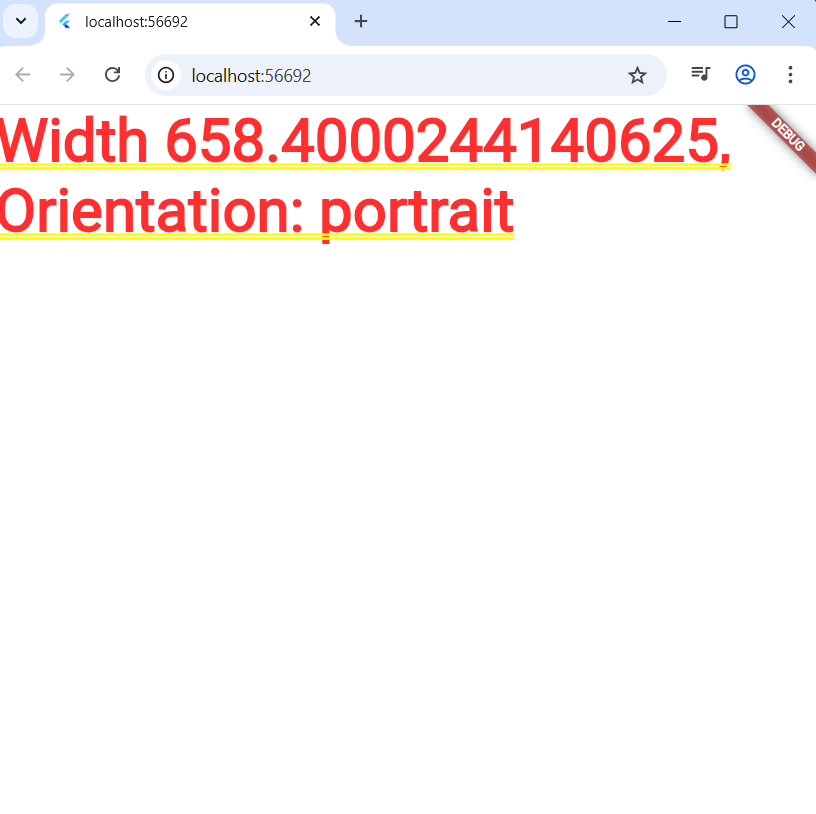
String orientation = MediaQuery.of(context).orientation.name;

return MaterialApp(home: Text('Width $width, Orientation: $orientation'));

}

}

**OUTPUT:**



**Expt.No.:** 4-i **Date:**

**AIM:** Set up navigation between different screens using Navigator.

**DESCRIPTION:**

### **1.What is Screen Navigation?**

In Flutter, navigation refers to moving from one screen (or page) to another. Screens are usually represented by Widgets like Scaffold.

Flutter uses a navigation stack:

* The first screen is pushed onto the stack when the app starts.
* Every time you navigate forward, a new screen is pushed on top.
* Going back pops the top screen off, returning you to the previous one.

### **2. How It Works in this Example?**

It has two screens:

* FirstScreen (with counter & button to go forward)
* SecondScreen (with a button to go back)

*A. Moving from First Screen to Second Screen*

Code from the FirstScreen:

ElevatedButton(

onPressed: () {

Navigator.push<void>(

context,

MaterialPageRoute<void>(

builder: (BuildContext context) => const SecondScreen(),

),

);

},

child: const Text('Go to Second Screen'),

**3. The Role of context**

* The context is tied to a widget’s position in the widget tree.
* Navigator uses it to find the current navigation stack and perform push/pop operations.

### **4. Key Takeaways from Your Example**

* *Navigator.push* → Go to a new screen.
* *Navigator.pop* → Return to the previous screen.
* *MaterialPageRoute* → is a common way to create navigation transitions in Material apps.
* *Navigation in Flutter operates like a stack data structure: Last In, First Out (LIFO).*

**PROGRAM:**

import 'package:flutter/material.dart';

void main() {

runApp(MaterialApp(title: 'Navigation Demo App', home: FirstPage()));

}

class FirstPage extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('Hello World')),

body: Center(

child: Column(

children: [

ElevatedButton(

onPressed: () {

Navigator.push(

context,

MaterialPageRoute(builder: (context) => SecondPage()),

);

},

child: Text('Next Page'),

),

SizedBox(height: 16),

ElevatedButton(

onPressed: () {

ScaffoldMessenger.of(

context,

).showSnackBar(SnackBar(content: Text("Hello, World!")));

},

child: Text("Msg"),

),

],

),

),

);

}

}

class SecondPage extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('Second Page')),

body: Center(

child: Column(

children: [

Text('This is the second page'),

SizedBox(height: 16),

ElevatedButton(

onPressed: () {

if (Navigator.canPop(context)) {

Navigator.pop(context);

} else {

ScaffoldMessenger.of(context).showSnackBar(

SnackBar(content: Text("No back history item")),

);

}

},

child: Text('Go Back'),

),

],

),

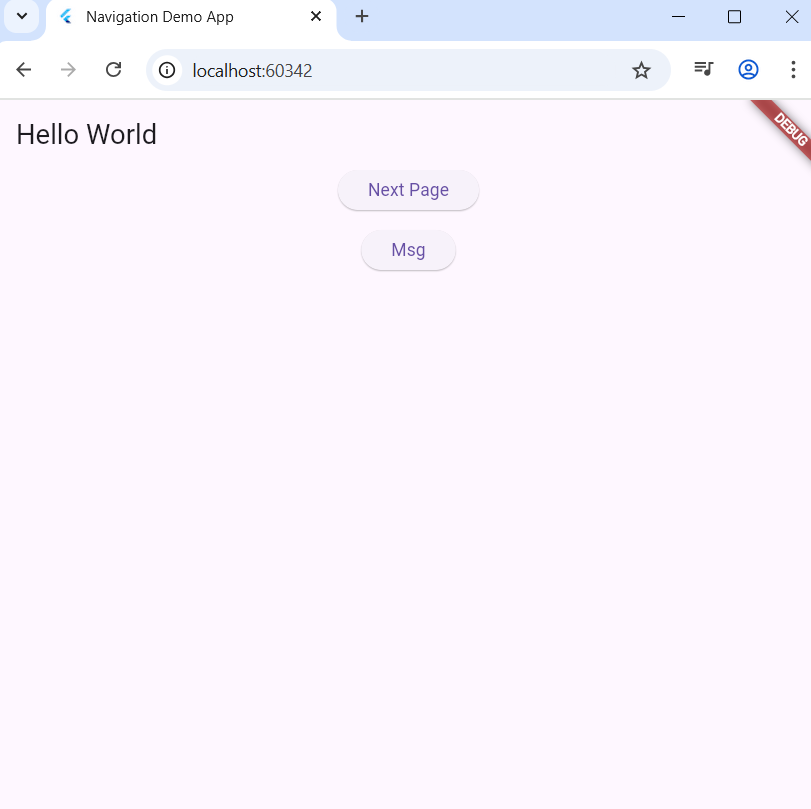
),

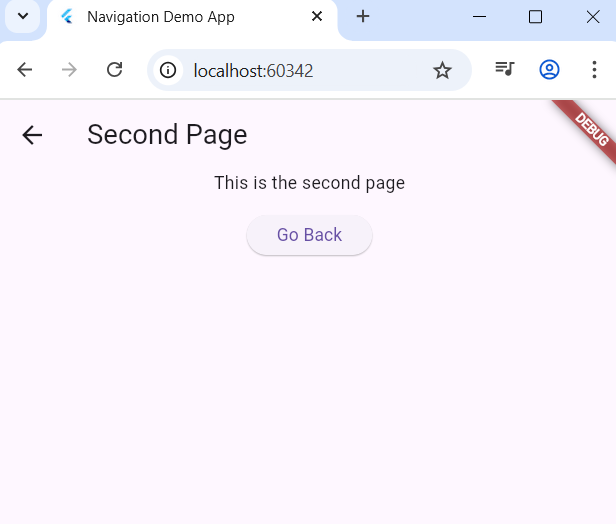
);

}

}

**OUTPUT:**





**Expt.No.:** 4-ii **Date:**

**AIM:** Implement navigation with named routes.

**DESCRIPTION:**

This Flutter program demonstrates basic navigation between two screens (pages) using named routes and the Navigator API.

* The MaterialApp widget sets the app’s title, initial route ("/"), and a map of route names to widget builders (routes), defining two pages: FirstPage and SecondPage.
* FirstPage shows a Scaffold with an AppBar and two buttons:
  + The Next Page button navigates to SecondPage using Navigator.pushNamed.
  + The Msg button displays a SnackBar with a "Hello, World!" message using ScaffoldMessenger.
* SecondPage displays text and a Go Back button:
  + The button attempts to navigate back using Navigator.pop if possible.
  + If there is no back history, it shows a SnackBar indicating no back item exists.

This code illustrates Flutter’s declarative UI style, navigation stack management with Navigator, and user feedback with SnackBars. The program makes use of StatelessWidget for immutable page widgets and builds UI using widget trees composed of Scaffold, AppBar, Center, Column, and buttons.

**PROGRAM:**

import 'package:flutter/material.dart';

// Lab experiment for Flutter navigation

void main() {

runApp(

MaterialApp(

title: 'Navigation Demo App',

initialRoute: "/",

routes: {

"/": (context) => FirstPage(),

"/second": (context) => SecondPage(),

},

),

);

}

class FirstPage extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('Hello World')),

body: Center(

child: Column(

children: [

ElevatedButton(

onPressed: () {

Navigator.pushNamed(context, "/second");

},

child: Text('Next Page'),

),

SizedBox(height: 16),

ElevatedButton(

onPressed: () {

ScaffoldMessenger.of(

context,

).showSnackBar(SnackBar(content: Text("Hello, World!")));

},

child: Text("Msg"),

),

],

),

),

);

}

}

class SecondPage extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('Second Page')),

body: Center(

child: Column(

children: [

Text('This is the second page'),

SizedBox(height: 16),

ElevatedButton(

onPressed: () {

if (Navigator.canPop(context)) {

Navigator.pop(context);

} else {

ScaffoldMessenger.of(context).showSnackBar(

SnackBar(content: Text("No back history item")),

);

}

},

child: Text('Go Back'),

),

],

),

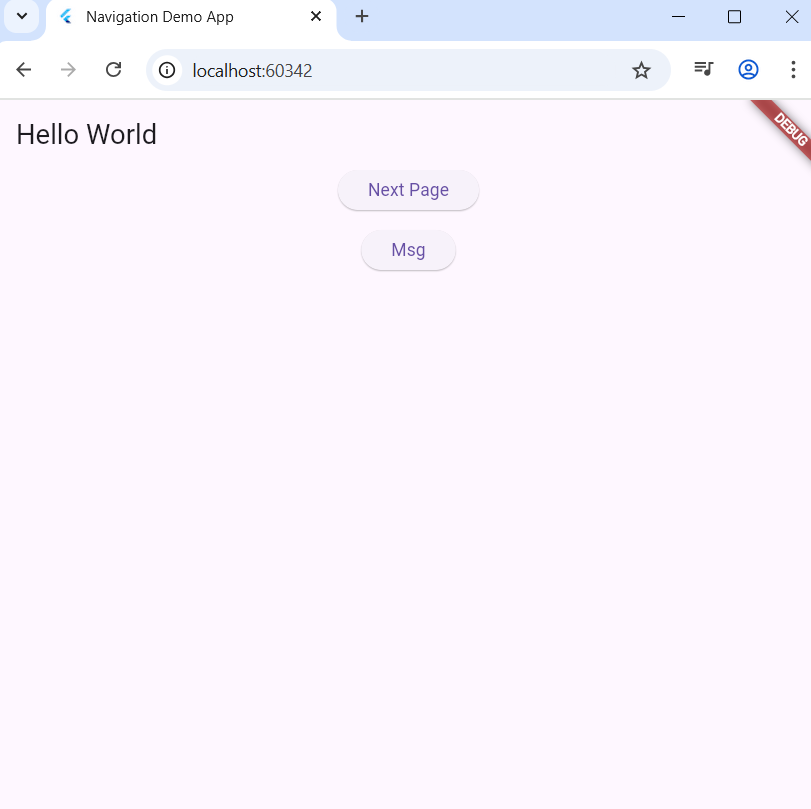
),

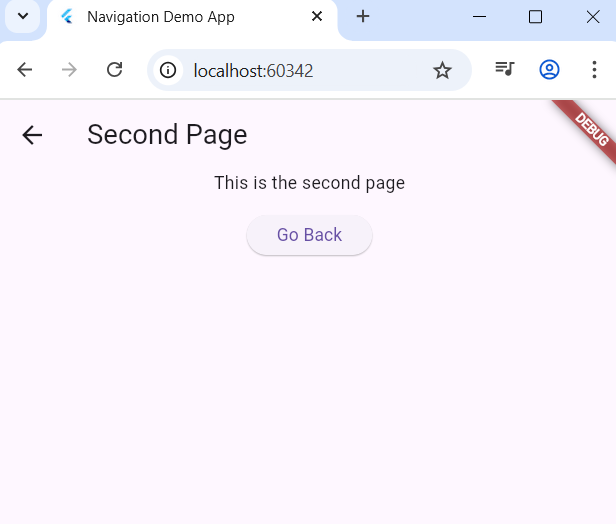
);

}

}

**OUTPUT:**





**Expt.No.:** 5-i **Date:**

**AIM:** Learn about stateful and stateless widgets.

**DESCRIPTION:**

Stateful widget. Use this kind of widget when you need to create a piece of UI that is going to change over the time. In this case the UI is going to dynamically change due to external events such as the received response of an HTTP request or the callback triggered by a button tap.

1. Counter is the widget itself and thus it’s inserted in the widget tree; \_CounterState is the mutable state of the Counter widget. When Flutter rebuilds the widget tree to refresh the UI, the build(...) method of State is called.
2. Subclasses of State gain access to the setState(...) method which rebuilds the widget (it’s like a refreshing tool).

class Counter extends StatefulWidget {

const Counter({super.key});

@override

State<StatefulWidget> createState() => \_CounterState();

}

class \_CounterState extends State<Counter> {

int \_counter = 0;

@override

Widget build(BuildContext context) {

return Row(

mainAxisSize: MainAxisSize.min,

mainAxisAlignment: MainAxisAlignment.center,

children: [

Text('$\_counter'),

SizedBox(width: 40),

IconButton(

onPressed: () {

// With out setState the widget is not rebuilt and you won't see the change in the value.

setState(() => \_counter++);

},

icon: Icon(Icons.add),

),

],

);

}

}

**PROGRAM:**

import 'package:flutter/material.dart';

void main() {

runApp(const MyApp());

}

class MyApp extends StatelessWidget {

const MyApp({super.key});

@override

Widget build(BuildContext context) {

return MaterialApp(home: const MyInfo());

}

}

class MyInfo extends StatelessWidget {

// Notice the constant constructor

const MyInfo();

@override

Widget build(BuildContext context) {

return Column(

mainAxisSize: MainAxisSize.min,

mainAxisAlignment: MainAxisAlignment.center,

children: [

Row(

children: const [

Icon(Icons.person),

SizedBox(width: 40),

Text("ABC"),

],

),

Row(

children: const [

Icon(Icons.email),

SizedBox(width: 40),

Text("ABC@xyz.com"),

],

),

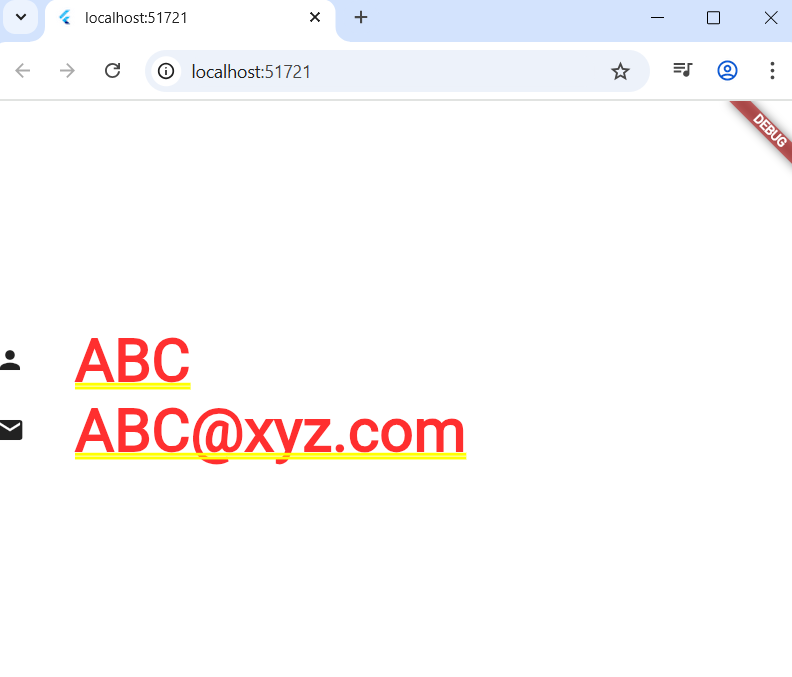
],

);

}

}

**OUTPUT:**



**Expt.No.:** 5-ii **Date:**

**AIM:** Implement state management using set State and Provider.

**DESCRIPTION:**

Instead of using setState we are using the provider way, setState will be hard to maintain in cases where change need to be propogated to multiple widgets in the hierarchy.

Provider uses Flutter’s ChangeNotifier to create a class that encapsulates the state and, when something changes, the interested widgets are notified and rebuilt. As the name suggests, it’s a notifier that alerts listeners about changes.

Here we need a model object to maintain the required state and ChangeNotifier is a mixin which provides notifyListeners method which signals the interested widgets that something had changed and rebuild is required to reflect the change.

class CounterModel with ChangeNotifier {

int \_counter = 0;

void increment() {

\_counter++;

notifyListeners();

}

void decrement() {

\_counter--;

notifyListeners();

}

int get currentCount => \_counter;

}

Now setup the ChangeNotifierProvider at a point where it will be available all the child widgets in the hierarchy through Provider.

class MyApp extends StatelessWidget {

const MyApp({super.key});

@override

Widget build(BuildContext context) {

return MaterialApp(

// Through this we make CounterModel available

// to all the child widgets of DemoPage.

home: ChangeNotifierProvider(

create: (context) => CounterModel(),

child: DemoPage(),

),

);}}

class DemoPage extends StatelessWidget

@override

Widget build(BuildContext context) {

// Fetch counter model object through provider

final counter = Provider.of<CounterModel>(context); }

}

**PROGRAM:**

import 'package:flutter/material.dart';

import 'package:provider/provider.dart';

void main() {

runApp(const MyApp());

}

class CounterModel with ChangeNotifier {

int \_counter = 0;

void increment() {

\_counter++;

notifyListeners();

}

void decrement() {

\_counter--;

notifyListeners();

}

int get currentCount => \_counter;

}

class MyApp extends StatelessWidget {

const MyApp({super.key});

// This widget is the root of your application.

@override

Widget build(BuildContext context) {

return MaterialApp(

home: ChangeNotifierProvider(

create: (context) => CounterModel(),

child: DemoPage(),

),

);

}

}

class DemoPage extends StatelessWidget {

const DemoPage({super.key});

@override

Widget build(BuildContext context) {

final counter = Provider.of<CounterModel>(context);

return Row(

mainAxisAlignment: MainAxisAlignment.center,

children: [

ElevatedButton(

child: const Text(

"+1",

style: TextStyle(color: Colors.green, fontSize: 25),

),

onPressed: () => counter.increment(),

),

SizedBox(width: 40),

Text("${counter.currentCount}", style: const TextStyle(fontSize: 30)),

SizedBox(width: 40),

ElevatedButton(

child: const Text(

"-1",

style: TextStyle(color: Colors.red, fontSize: 25),

),

onPressed: () => counter.decrement(),

),

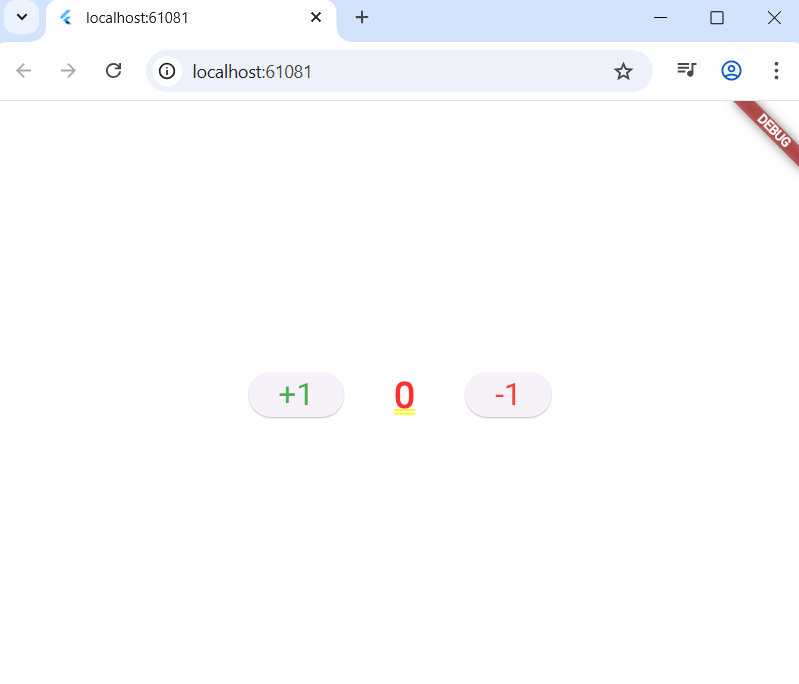
],

);

}

}

**OUTPUT:**



**Expt.No.:** 6-i **Date:**

**AIM:** Create custom widgets for specific UI elements.

**DESCRIPTION:**

The Product widget is a custom Flutter widget that displays a product information. This widget demonstrates how to create reusable UI components in Flutter by:

* Defining custom parameters (name, icon, price)
* Style the components as needed.
* Add custom behaviour. Once done, this component can be used as a widget in itself and can be composed as a child for other widgets.

class Product extends StatelessWidget {

final String name;

final IconData icon;

final double price;

const Product({

required this.name,

required this.icon,

required this.price,

super.key,

});

@override

Widget build(BuildContext context) {

return Column(children:[Icon(icon), Text(name), Text("$price")]);

}

}

Demo usage -

Column(

children: [

Product(name: "Computer", icon: Icons.computer, price: 20000),

Product(name: "Keyboard", icon: Icons.keyboard, price: 2000),

],

**PROGRAM:**

import 'package:flutter/material.dart';

void main() {

runApp(MaterialApp(home: LaptopProductsPage()));

}

class Product extends StatelessWidget {

final String name;

final IconData icon;

final double price;

final VoidCallback onAddToCart;

const Product({

required this.name,

required this.icon,

required this.price,

required this.onAddToCart,

super.key,

});

@override

Widget build(BuildContext context) {

return Card(

elevation: 3,

shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(12)),

child: Padding(

padding: const EdgeInsets.all(12.0),

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Icon(icon, size: 48, color: Colors.blue),

const SizedBox(height: 10),

Text(

name,

textAlign: TextAlign.center,

style: const TextStyle(fontSize: 14, fontWeight: FontWeight.bold),

),

const SizedBox(height: 6),

Text(

"₹${price.toStringAsFixed(2)}",

style: TextStyle(fontSize: 13, color: Colors.grey[700]),

),

const SizedBox(height: 8),

ElevatedButton(

onPressed: onAddToCart,

child: const Text("Add to Cart"),

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

// --- Grid Display Page ---

class LaptopProductsPage extends StatelessWidget {

LaptopProductsPage({super.key});

final List<Map<String, dynamic>> laptops = [

{'name': 'HP Pavilion 15', 'icon': Icons.laptop, 'price': 67990.0},

{'name': 'Dell Inspiron 14', 'icon': Icons.laptop\_mac, 'price': 58999.0},

{'name': 'Samsung Galaxy Book3', 'icon': Icons.devices, 'price': 71999.0},

];

void \_showSnackBar(BuildContext context, String name) {

ScaffoldMessenger.of(context).showSnackBar(

SnackBar(

content: Text('$name added to cart!'),

duration: const Duration(seconds: 1),

), ); }

@override

Widget build(BuildContext context) {

int columns = (MediaQuery.of(context).size.width / 448).floor();

if (columns < 1) columns = 1;

return Scaffold(

appBar: AppBar(title: const Text("Laptops on Amazon.in")),

body: GridView.builder(

padding: const EdgeInsets.all(8),

gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(

crossAxisCount: columns,

crossAxisSpacing: 8,

mainAxisSpacing: 8

),

itemCount: laptops.length,

itemBuilder: (context, index) {

final laptop = laptops[index];

return Product(

name: laptop['name'],

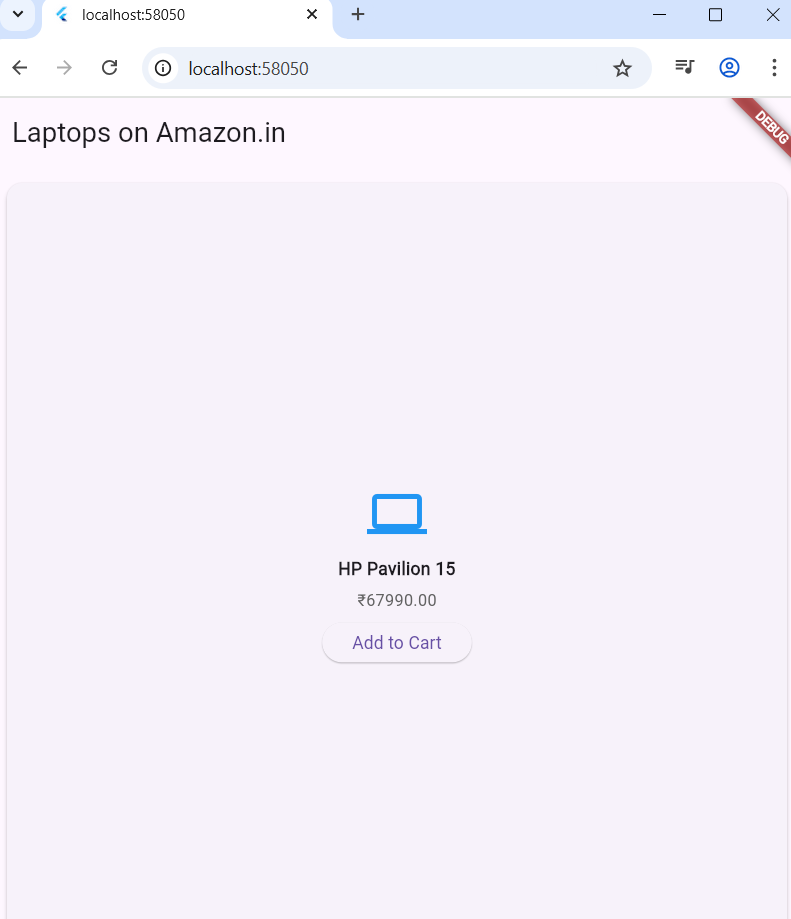
icon: laptop['icon'],

price: laptop['price'],

onAddToCart: () => \_showSnackBar(context, laptop['name']),

);},),);}}

**OUTPUT:**



**Expt.No.:** 6-ii **Date:**

**AIM:** Apply styling using themes and custom styles.

**DESCRIPTION:**

How the Theme Flows Through the Tree

1. MaterialApp with theme: ThemeData(...)
   * This theme automatically propagates to all widgets in the widget tree below.
   * Widgets can access the global theme using Theme.of(context).
2. Global Theme Styles Used
   * Colors:
     + primaryColor → AppBar background
     + colorScheme.secondary → FAB background, OutlinedButton border
   * Text styles:
     + headlineLarge, bodyLarge → Several Text widgets
   * ElevatedButtonTheme:
     + All ElevatedButtons get consistent padding, rounded corners
3. Custom / Local Styles
   * Some styles are set inline (e.g., deepPurple TextStyle in customStyle)
   * These override the global theme only for that widget
4. Theme.of(context)
   * Looks up the nearest Theme in the widget tree (from MaterialApp’s ThemeData)
   * Allows widgets to remain consistent with app-wide styling
   * Example:

[Theme.of](http://theme.of)(context).colorScheme.secondary

Theme.of(context).textTheme.headlineLarge

**PROGRAM:**

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Theming Demo',

debugShowCheckedModeBanner: false,

theme: ThemeData(

// Define the default brightness and colors.

brightness: Brightness.light,

primaryColor: Colors.teal,

colorScheme: ColorScheme.fromSwatch().copyWith(

secondary: Colors.orange, // Accent color

),

// Define the default font family.

fontFamily: 'Roboto',

// Define the default `TextTheme`.

textTheme: TextTheme(

headlineLarge: TextStyle(

fontSize: 32.0,

fontWeight: FontWeight.bold,

color: Colors.teal,

),

bodyLarge: TextStyle(fontSize: 18.0, color: Colors.black87),

labelLarge: TextStyle(fontSize: 16.0, color: Colors.white),

), // Global elevated button theme

elevatedButtonTheme: ElevatedButtonThemeData(

style: ElevatedButton.styleFrom(

padding: EdgeInsets.symmetric(horizontal: 24, vertical: 12),

shape: RoundedRectangleBorder(

borderRadius: BorderRadius.circular(12),

),

),

),

),

home: HomeScreen(),

);

}

}

class HomeScreen extends StatelessWidget {

@override

Widget build(BuildContext context) {

// Custom text style not in the theme

final TextStyle customStyle = TextStyle(

fontSize: 20,

fontWeight: FontWeight.w600,

letterSpacing: 1.2,

color: Colors.deepPurple,

);

return Scaffold(

appBar: AppBar(title: Text('Flutter Theme Demo')),

body: Padding(

padding: const EdgeInsets.all(16.0),

child: Column(

crossAxisAlignment: CrossAxisAlignment.start,

children: [

Text(

'Styled with Theme',

style: Theme.of(context).textTheme.headlineLarge,

),

SizedBox(height: 10),

Text(

'Using the bodyLarge style from the app theme.',

style: Theme.of(context).textTheme.bodyLarge,

),

SizedBox(height: 20),

Text('Using a custom inline style.', style: customStyle),

SizedBox(height: 30),

ElevatedButton(

onPressed: () {},

child: Text('Themed Elevated Button'),

),

SizedBox(height: 15),

OutlinedButton(

style: OutlinedButton.styleFrom(

side: BorderSide(

color: Theme.of(context).colorScheme.secondary,

width: 2,

),

foregroundColor: Theme.of(context).colorScheme.secondary,

),

onPressed: () {},

child: Text('Custom Styled Outlined Button'),

),], ),),

floatingActionButton: FloatingActionButton(

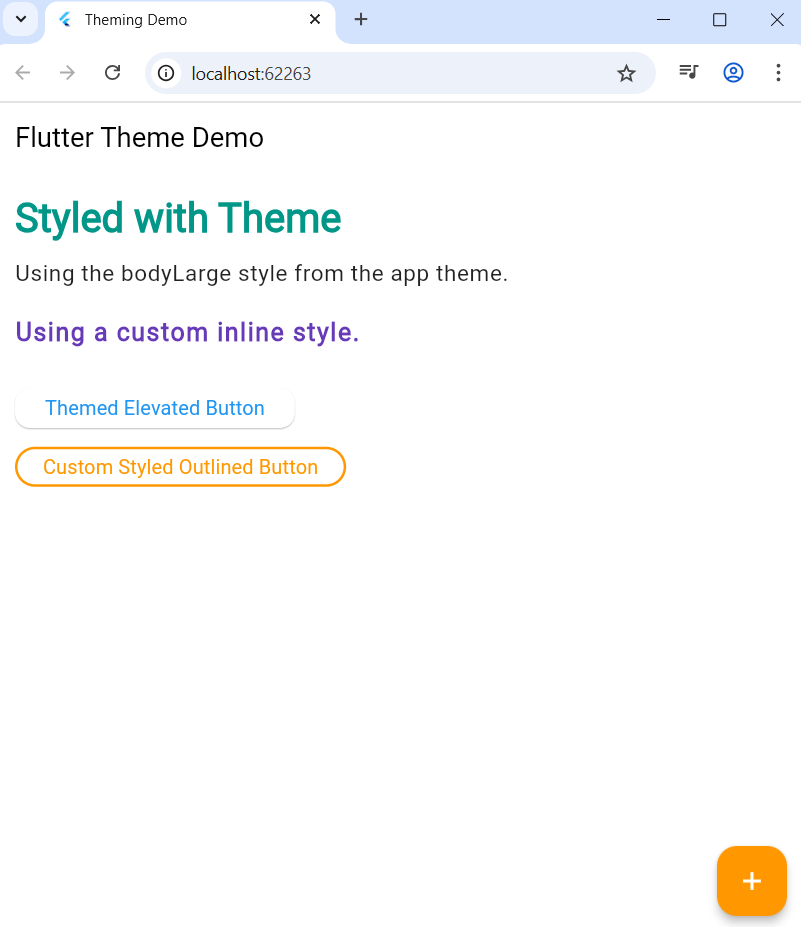
onPressed: () {},

child: Icon(Icons.add),

backgroundColor: Theme.of(context).colorScheme.secondary,

),);}}

**OUTPUT:**



**Expt.No.:** 7 **Date:**

**AIM:** Design a form with various input fields and implement form validation and error handling.

**DESCRIPTION:**

Code Breakdown – Simple Flutter Form

1. Form Creation
   * Use the Form widget to group input fields (TextFormField, DropdownButtonFormField, etc.).

Form(

key: \_formKey,

child: Column(

children: [

// Name Field

TextFormField(

controller: nameController,

decoration: InputDecoration(labelText: 'Name'),

validator: (value) =>

value!.isEmpty ? 'Please enter your name' : null,

)]));

* + Assign a GlobalKey to the form for accessing validation and saving methods.
    - It provides a unique handle to the Form widget.
    - It allows direct access to the form’s state anywhere in your widget tree.
    - Enables use of methods like validate(), save(), and reset() on the associated form.
    - Usage:
      * Create a single instance:

final \_formKey = GlobalKey<FormState>();

* + - * Pass it to the Form widget:

Form(key: \_formKey, ...)

* + - * Call

\_formKey.currentState!.validate()

* + - to run all field validators.

1. TextEditingController
   * Purpose:
     + Controls and retrieves the current value of a text field.
   * How it works:
     + Provides access to the text being edited in TextField or TextFormField.
     + Can set text programmatically and listen for changes.
   * Usage:
     + Declare and initialize:

final nameController = TextEditingController();

* + - Attach to a field:

TextFormField(controller: nameController, ...)

* + - Retrieve or set value:

nameController.text

* + - Dispose of it when no longer used to free resources.

1. Validation
   * Attach a validator function to each field (via validator: parameter).

TextFormField( validator: (value) =>

value!.isEmpty ? 'Please enter your name' : null,)

* + The validator checks the entered value and returns an error message if invalid (or null if valid).
  + Trigger validation using

formKey.currentState!.validate();

1. this will run all field validators.
2. Submission
   * Wrap the submit logic in a button (e.g. ElevatedButton).
   * On submit, call validate()—if valid, proceed (display a success message, send data, etc.).
   * If not valid, error messages will appear below the respective invalid fields.
3. ScaffoldMessenger.of(context).showSnackBar(....)
   * Displays the message at the bottom.

**PROGRAM:**

import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(debugShowCheckedModeBanner: false, home: SimpleForm());

}

}

class SimpleForm extends StatefulWidget {

@override

\_SimpleFormState createState() => \_SimpleFormState();

}

class \_SimpleFormState extends State<SimpleForm> {

final \_formKey = GlobalKey<FormState>();

final TextEditingController nameController = TextEditingController();

final TextEditingController emailController = TextEditingController();

final TextEditingController passwordController = TextEditingController();

@override

void dispose() {

nameController.dispose();

emailController.dispose();

passwordController.dispose();

super.dispose();

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text("Simple Form")),

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: (value) =>

value!.isEmpty ? 'Please enter your name' : null,

),

SizedBox(height: 10),

// Email Field

TextFormField(

controller: emailController,

decoration: InputDecoration(labelText: 'Email'),

keyboardType: TextInputType.emailAddress,

validator: (value) {

if (value!.isEmpty) return 'Please enter your email';

if (!value.contains('@')) return 'Enter a valid email';

return null;

},

),

SizedBox(height: 10),

// Password Field

TextFormField(

controller: passwordController,

decoration: InputDecoration(labelText: 'Password'),

obscureText: true,

validator: (value) => value!.length < 6

? 'Password must be at least 6 chars'

: null,

),

SizedBox(height: 20),

// Submit Button

ElevatedButton(

onPressed: () {

if (\_formKey.currentState!.validate()) {

String name = nameController.text;

ScaffoldMessenger.of(context).showSnackBar(

SnackBar(

content: Text(

'Hello $name !! Form submitted successfully!',

),

),

);

}

},

child: Text('Submit'),

),

],

),

),

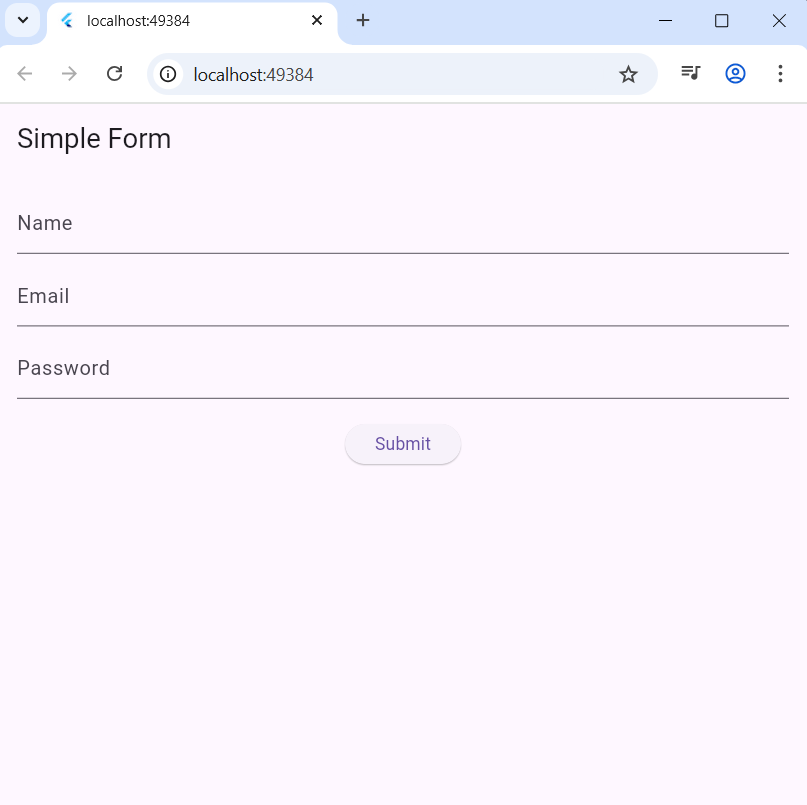
),

);

}

}

OUTPUT:



**Expt.No.: 8-i Date:**

**AIM:** Add animations to UI elements using Flutter’s animation framework.

**DESCRIPTION:**

ImplicitlyAnimatedWidgets (and their subclasses) automatically animate changes in their properties whenever they change. These widgets take an Animation as an argument to power the animation. This gives the developer full control over the animation at the cost of requiring you to manually manage the underlying AnimationController.

Examples such as

1. AnimatedContainer
2. AnimatedDefaultTextStyle
3. AnimatedPadding etc.

### **AnimatedContainer**

Flutter provides the AnimatedContainer widget. Like the Container widget, AnimatedContainer allows you to define the width, height, background colors, and more. However, when the AnimatedContainer is rebuilt with new properties, it automatically animates between the old and new values. In Flutter, these types of animations are known as "implicit animations."

AnimatedContainer(

width: \_boxSize,

height: \_boxSize,

decoration: BoxDecoration(

color: \_boxColor,

borderRadius: \_borderRadius,

),

duration: Duration(seconds: 1),

curve: Curves.easeInOut,

)

Note: Changes to the variables *\_boxSize*, *\_boxColor* and *\_borderRadius* will animate the change for the duration of 1 second. Instead if you use plain Container you will not see the animation.

### **AnimatedDefaultTextStyle**

Similar to AnimatedContainer, AnimatedDefaultTextStyle widget will animate the changes to the TextStyle.

AnimatedDefaultTextStyle(

duration: const Duration(milliseconds: 600),

curve: Curves.linear,

style: TextStyle(

fontSize: \_fontSize,

color: \_fontColor,

fontWeight: FontWeight.bold,

),

child: const Text("Flutter Rocks!"),

)

**PROGRAM:**

import 'dart:math';

import 'package:flutter/material.dart';

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

class AnimationDemoApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(home: AnimationDemo());

}

}

class AnimationDemo extends StatefulWidget {

@override

\_AnimationDemoState createState() => \_AnimationDemoState();

}

class \_AnimationDemoState extends State<AnimationDemo> {

double \_fontSize = 100;

Color \_fontColor = Colors.blue;

double \_boxSize = 100;

Color \_boxColor = Colors.blue;

BorderRadiusGeometry \_borderRadius = BorderRadiusGeometry.circular(25);

final random = Random();

void \_animateBox() {

setState(() {

\_fontSize = 20 + random.nextInt(80).toDouble();

\_boxSize = random.nextInt(400).toDouble();

\_boxColor = Color.fromRGBO(

random.nextInt(256),

random.nextInt(256),

random.nextInt(256),

random.nextDouble(),

);

\_fontColor = \_boxColor;

\_borderRadius = BorderRadiusGeometry.circular(

random.nextInt(25).toDouble(),

);

});

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text("Simple Animation Example")),

body: Center(

child: Column(

children: [

AnimatedContainer(

width: \_boxSize,

height: \_boxSize,

decoration: BoxDecoration(

color: \_boxColor,

borderRadius: \_borderRadius,

),

duration: Duration(seconds: 1),

curve: Curves.easeInOut,

),

SizedBox(height: 40),

AnimatedDefaultTextStyle(

duration: const Duration(milliseconds: 600),

curve: Curves.linear,

style: TextStyle(

fontSize: \_fontSize,

color: \_fontColor,

fontWeight: FontWeight.bold,

),

child: const Text("Flutter Rocks!"),

),

],

),

),

floatingActionButton: FloatingActionButton(

onPressed: \_animateBox,

child: Icon(Icons.play\_arrow),

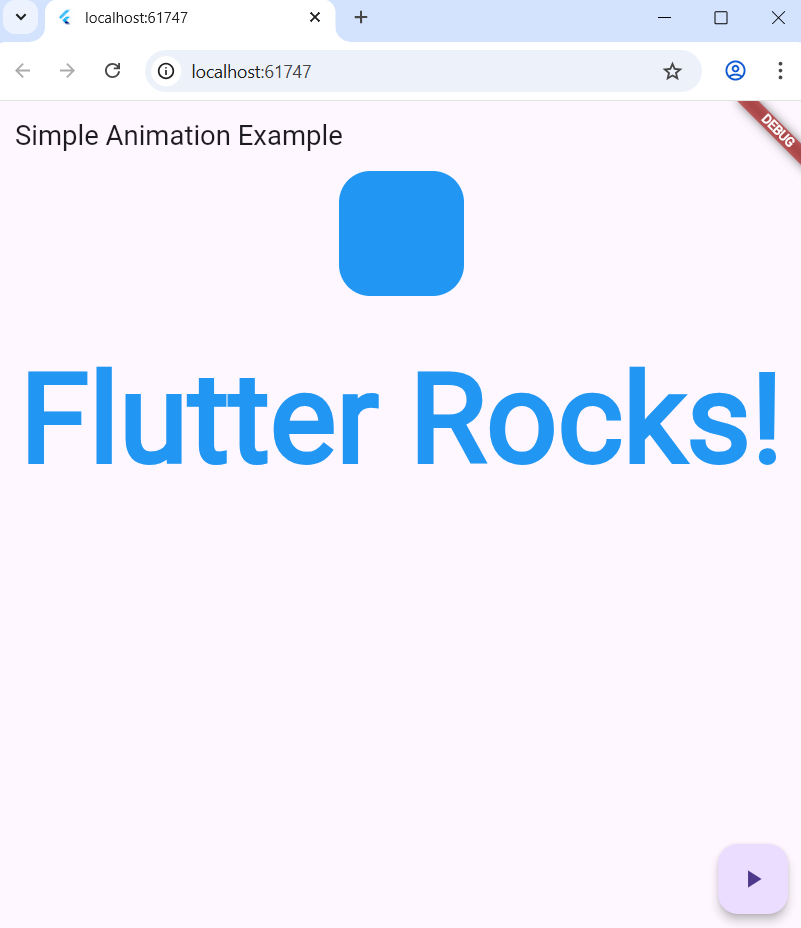
),

);

}

}

**OUTPUT:**



**Expt.No.:** 8-ii **Date:**

**AIM:** Experiment with different types of animations (fade, slide, etc.).

**DESCRIPTION:**

### **AnimationController**

* It is a special animation object that controls the animation.
* Manages the animation's duration, playback, and state (forward, reverse, stopped).
* Provides a vsync parameter to optimize resource usage by syncing with the screen refresh rate, preventing offscreen animations from consuming battery.
* In the example, \_slideController drives the slide animation's timing and progress over 500 milliseconds.
* Methods like .forward() and .reverse() start and reverse the animation respectively.

// Slide Animation Setup

\_slideController = AnimationController(

duration: const Duration(milliseconds: 500),

vsync: this,

);

### **2. Tween**

* Stands for "Tweening," which means generating intermediate frames between two values to create smooth animations.
* Defines the start (begin) and end (end) values for an animation property.
* The generic type T could be something like double, Color, or, as in this case, Offset.
* In the example, Tween animates a widget's position from an offset of (-1.0, 0.0) (off-screen to the left) to Offset.zero (original position).

### **3. CurvedAnimation**

* Wraps another animation (usually an AnimationController) to apply non-linear motion curves like easing, bounce, or acceleration.
* Controls the rate of change of the animation to make it look more natural.
* In the example, Curves.easeInOut is used for smooth, gradual acceleration and deceleration of the slide.

\_slideAnimation =

Tween<Offset>(begin: Offset(-1.0, 0.0), end: Offset.zero)

.animate(

CurvedAnimation(parent: \_slideController, curve: Curves.easeInOut),

);

### **4. Animation**

* Represents the animated value at a given frame.
* The Tween combined with AnimationController produces this animated output.
* Used as the input to SlideTransition to move the widget along the specified offset.

### **5. SlideTransition**

* A widget that transitions its child by changing its position.
* It takes an Animation as its position argument and moves the child widget accordingly.
* Here, it uses the \_slideAnimation to slide the green box horizontally.

SlideTransition(

position: \_slideAnimation,

child: Container(....)

)

### **6. SingleTickerProviderStateMixin**

* A mixin on the State class providing a single ticker (frame callback) for animation controllers.
* Ensures the AnimationController stays in sync with the device's screen refresh rate.
* Essential for efficient animation.

class \_AnimationDemoScreenState extends State<AnimationDemoScreen>

with SingleTickerProviderStateMixin {

…

}

**PROGRAM:**

import 'package:flutter/material.dart';

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

class AnimationDemoApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(home: AnimationDemoScreen());

}

}

class AnimationDemoScreen extends StatefulWidget {

@override

\_AnimationDemoScreenState createState() => \_AnimationDemoScreenState();

}

class \_AnimationDemoScreenState extends State<AnimationDemoScreen>

with SingleTickerProviderStateMixin {

bool \_visible = true;

late AnimationController \_slideController;

late Animation<Offset> \_slideAnimation;

@override

void initState() {

super.initState();

// Slide Animation Setup

\_slideController = AnimationController(

duration: const Duration(milliseconds: 500),

vsync: this,

);

\_slideAnimation = Tween<Offset>(begin: Offset(-2.0, 0.0), end: Offset.zero)

.animate(

CurvedAnimation(parent: \_slideController, curve: Curves.easeInOut),

);

// Start slide animation initially

\_slideController.forward();

}

@override

void dispose() {

\_slideController.dispose();

super.dispose();

}

void \_toggleFade() {

setState(() {

\_visible = !\_visible;

});

}

void \_toggleSlide() {

if (\_slideController.status == AnimationStatus.completed) {

\_slideController.reverse();

} else {

\_slideController.forward();

}

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('Flutter Animation Demo')),

body: Padding(

padding: const EdgeInsets.all(20.0),

child: Column(

children: [

Text('Fade Animation:', style: TextStyle(fontSize: 20)),

SizedBox(height: 10),

AnimatedOpacity(

opacity: \_visible ? 1.0 : 0.0,

duration: Duration(seconds: 1),

child: Container(

width: 150,

height: 150,

color: Colors.blue,

alignment: Alignment.center,

child: Text(

'Fade Me',

style: TextStyle(color: Colors.white, fontSize: 18),

),),),

ElevatedButton(

onPressed: \_toggleFade,

child: Text(\_visible ? 'Fade Out' : 'Fade In'),

),

Divider(height: 40),

Text('Slide Animation:', style: TextStyle(fontSize: 20)),

SizedBox(height: 10),

SlideTransition(

position: \_slideAnimation,

child: Container(

width: 150,

height: 150,

color: Colors.green,

alignment: Alignment.center,

child: Text(

'Slide Me',

style: TextStyle(color: Colors.white, fontSize: 18),

),),),

ElevatedButton(

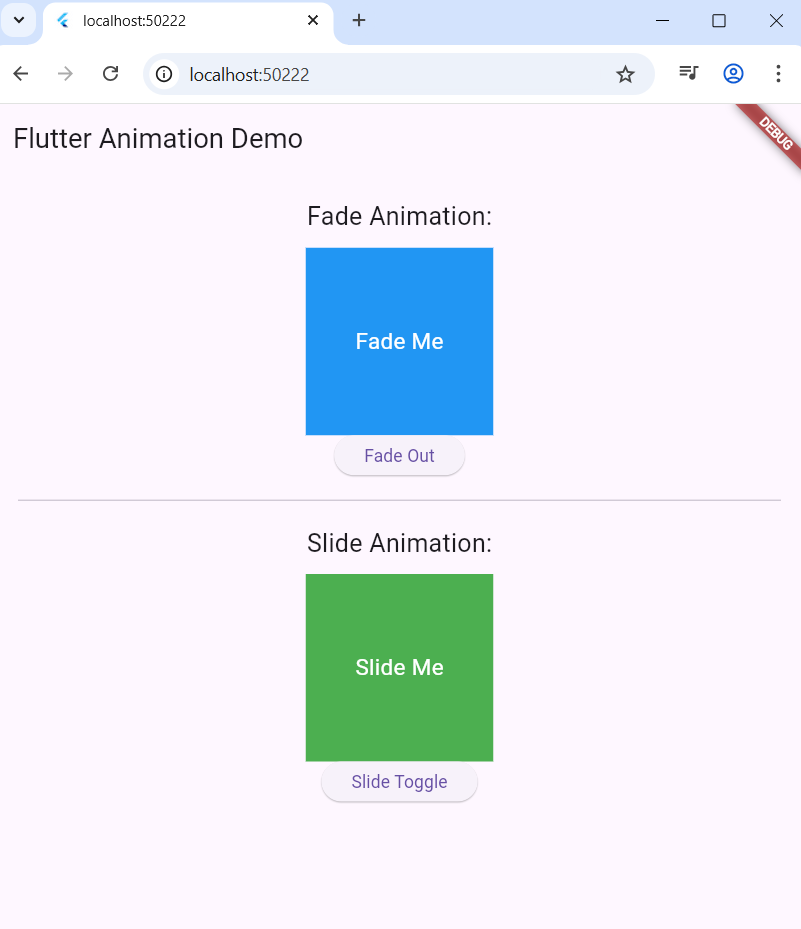
onPressed: \_toggleSlide,

child: Text('Slide Toggle'),

),],),),);}

}

**OUTPUT:**



**Expt.No.:** 9 **Date:**

**AIM:** Fetch data from a REST API and Display the fetched data in a meaningful way in the UI.

**DESCRIPTION:**

The example uses the http package in Flutter to make an HTTP GET request to a public REST API (<https://jsonplaceholder.typicode.com/posts>). This involves:

* Importing the http package.
* Writing an asynchronous function that performs an HTTP GET request using http.get().
* Parsing the JSON response into Dart objects.
* Handling the response status to ensure correct data fetching or error handling.

### **Import http**

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

### **Asynchronous fetch using http.get()**

Future<List<Post>> fetchPosts() async {

// fetch the response from http endpoint and wait for the response.

final response = await http.get(Uri.parse('https://jsonplaceholder.typicode.com/posts'));

// if successful

if (response.statusCode == 200) {

// Decode JSON response body

List jsonResponse = json.decode(response.body);

// For each JSON object from the response list => map it to Post object

// and collect the result as a list of Post objects.

return jsonResponse.map((post) => Post.fromJson(post)).toList();

} else {

// Throw an exception in case of error

throw Exception('Failed to load posts');

}

}

* http.get(Uri.parse(url)) sends a GET request to the specified URL.
* Response returned contains statusCode and body.
* If statusCode is 200, the response body is decoded from JSON.
* The data is converted to a list of Post objects for easier use in the app.
* Errors are handled by throwing exceptions for failed requests.

### **FutureBuilder**

* The Future<List<Post>> returned by fetchPosts is used inside a FutureBuilder to update the UI based on request state (loading, error, or loaded data).
* Wait for and display API fetched data asynchronously.

FutureBuilder<List<Post>>(

future: fetchPosts(), // <=== fetchPosts() will make the API call

builder: (context, snapshot) {

if (snapshot.connectionState == ConnectionState.waiting) {

return CircularProgressIndicator();

} else if (snapshot.hasError) {

return Text('Error: ${snapshot.error}');

} else if (snapshot.hasData) {

return ListView(...); // <=== present the data in the UI

} else {

return Text('No data found');

}

},

)

### **ListView.builder**

* Creates a scrollable, linear array of widgets that are built lazily.
* Displays the list of posts dynamically.

ListView.builder(

itemCount: posts.length,

itemBuilder: (context, index) { // <=== for each item/object

return ListTile(

title: Text(posts[index].title),

subtitle: Text(posts[index].body),

);

},

)

**PROGRAM:**

import 'package:flutter/material.dart';

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

import 'dart:convert';

// Define a Post model

class Post {

final int id;

final String title;

final String body;

Post({required this.id, required this.title, required this.body});

factory Post.fromJson(Map<String, dynamic> json) {

return Post(id: json['id'], title: json['title'], body: json['body']);

}

}

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(title: 'Flutter REST API Demo', home: PostsScreen());

}

}

class PostsScreen extends StatefulWidget {

@override

\_PostsScreenState createState() => \_PostsScreenState();

}

class \_PostsScreenState extends State<PostsScreen> {

late Future<List<Post>> futurePosts;

@override

void initState() {

super.initState();

futurePosts = fetchPosts();

}

// Fetch posts from the API

Future<List<Post>> fetchPosts() async {

final response = await http.get(

Uri.parse('https://jsonplaceholder.typicode.com/posts'),

);

if (response.statusCode == 200) {

List jsonResponse = json.decode(response.body);

return jsonResponse.map((post) => Post.fromJson(post)).toList();

} else {

throw Exception('Failed to load posts');

}

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('Posts from REST API')),

body: Center(

child: FutureBuilder<List<Post>>(

future: futurePosts,

builder: (context, snapshot) {

if (snapshot.connectionState == ConnectionState.waiting) {

return CircularProgressIndicator();

} else if (snapshot.hasError) {

return Text('${snapshot.error}');

} else if (snapshot.hasData) {

List<Post> posts = snapshot.data!;

return ListView.builder(

itemCount: posts.length,

itemBuilder: (context, index) {

return Card(

margin: EdgeInsets.symmetric(vertical: 20, horizontal: 26),

child: ListTile(

leading: Text(posts[index].id.toString()),

title: Text(posts[index].title),

subtitle: Text(posts[index].body),

),

);

},

);

} else {

return Text('No data found');

}

},

),

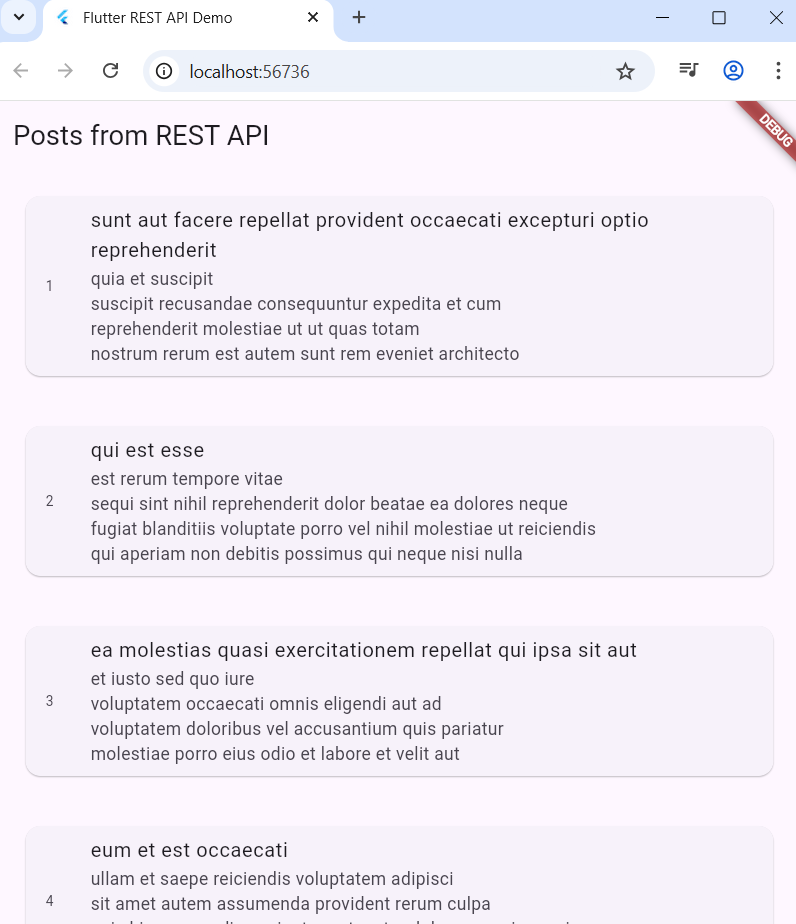
),

);

}

}

OUTPUT:



**Expt.No.:** 10-i **Date:**

**AIM:** Write unit tests for UI components.

**DESCTIPTION:**

### **testWidgets**

Purpose: Defines a widget test and sets up the testing environment.

How it works: It runs the provided callback (test body) inside a simulated Flutter test environment. This is necessary for any code that interacts with widgets, as it handles setup and teardown automatically and provides a fresh WidgetTester instance for each test.

Usage in Example:

testWidgets('Counter initializes and increments', (WidgetTester tester) async { ... })

This method takes a test description (for reporting) and a test callback with a WidgetTester parameter.

### **2. tester.pumpWidget**

Purpose: Renders the widget in a simulated test environment so you can interact with or inspect the UI.

How it works: Builds the widget tree and renders it, much like mounting a widget in the live app.

Usage in Example:

await tester.pumpWidget(MaterialApp(home: CounterWidget(initialValue: 2)));

Required before you can query, interact with, or test any widgets.

### **3. Finders — find.text, find.byType**

Purpose: Locates widgets in the rendered tree for inspection or interaction.

How it works: Uses various strategies (find.text, find.byType, etc.) to find widgets based on text, type, key, or instance.

Usage in Example:

expect(find.text('2'), findsOneWidget);

await tester.tap(find.byType(ElevatedButton));

expect(find.text('3'), findsOneWidget);

### **4. tester.tap**

Purpose: Simulates a user clicking/tapping on a widget.

How it works: Finds the widget and generates a tap event as if a user interacted with it.

Usage in Example:

await tester.tap(find.byType(ElevatedButton));

Often followed by tester.pump() to process changes.

### **5. tester.pump**

Purpose: Processes any pending UI frames and rebuilds the widget tree.

How it works: Ensures that state changes (like setting new count) are reflected in the UI after an interaction.

Usage in Example:

await tester.pump();

This is crucial after interactions, so the updated UI is available for inspection.

### **6. expect**

Purpose: Asserts that certain conditions are true, such as the presence, count, or absence of widgets.

How it works: Uses matchers (like findsOneWidget, findsNothing, etc.) to check if your expectations about the widget tree and its state are met.

Usage in Example:

expect(find.text('2'), findsOneWidget);

expect(find.text('3'), findsOneWidget);

**PROGRAM:**

import 'package:flutter/material.dart';

void main() {

runApp(MaterialApp(home: CounterWidget()));

}

class CounterWidget extends StatefulWidget {

final int initialValue;

CounterWidget({this.initialValue = 0});

@override

\_CounterWidgetState createState() => \_CounterWidgetState();

}

class \_CounterWidgetState extends State<CounterWidget> {

int count = 0;

@override

void initState() {

super.initState();

count = widget.initialValue;

}

void \_increment() {

setState(() {

count++;

});}

@override

Widget build(BuildContext context) {

return Column(

children: [

Text('$count'),

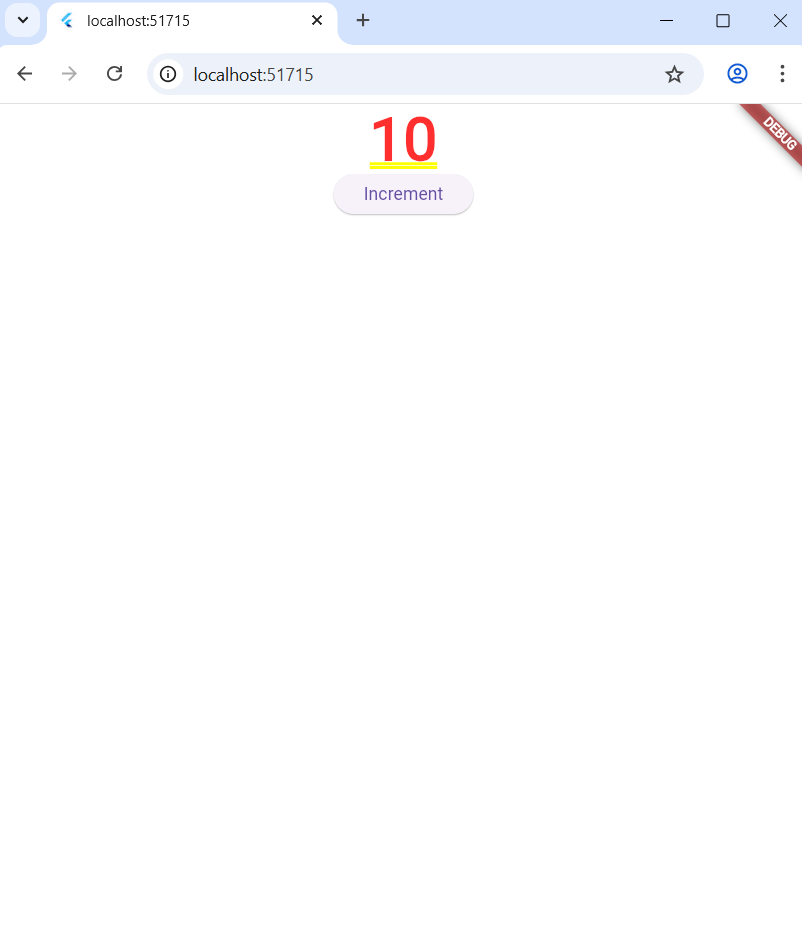
ElevatedButton(onPressed: \_increment, child: Text('Increment')),

],);

}

}

OUTPUT:



**Expt.No.: 10-ii Date:**

**AIM:** Use Flutter’s debugging tools to identify and fix issues.

**DESCRIPTION:**

Flutter provides a comprehensive set of debugging tools to help developers identify, diagnose, and fix issues efficiently. Here are the key Flutter debugging tools and their functionalities:

### **1.Flutter DevTools:**

Key features and benefits of running in debug mode:

#### **Hot Reload:**

This allows you to instantly see the effects of code changes without restarting the application, significantly speeding up development.

#### **Error Checking:**

The application performs more extensive error checking and provides detailed error messages, aiding in bug identification.

#### **Step-by-step Debugging:**

You can set breakpoints and step through your code line by line, inspecting variable values and understanding the execution flow. This is crucial for pinpointing the source of issues.

#### **Performance Monitoring (with DevTools):**

While debug mode performance is not representative of release mode, it enables the use of Flutter DevTools for detailed performance analysis, including CPU usage, memory allocation, and UI rendering.

#### **Widget Inspector:**

This tool, accessible through DevTools, allows you to visualize the widget tree and inspect the properties of individual widgets, assisting in UI layout and debugging.

**PROGRAM:**

import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {

const MyApp({super.key});

@override

Widget build(BuildContext context) {

return MaterialApp(

title: 'Flutter Demo',

debugShowCheckedModeBanner: false,

theme: ThemeData(colorSchemeSeed: Colors.blue),

home: const MyHomePage(title: 'Flutter Demo Home Page'),

);

}

}

class MyHomePage extends StatefulWidget {

final String title;

const MyHomePage({super.key, required this.title});

@override

State<MyHomePage> 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(widget.title)),

body: Center(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

const Text('You have pushed the button this many times:'),

Text(

'$\_counter',

style: Theme.of(context).textTheme.headlineMedium,

),

],

),

),

floatingActionButton: FloatingActionButton(

onPressed: \_incrementCounter,

tooltip: 'Increment',

child: const Icon(Icons.add),

),

);

}

}

OUTPUT:

