**Name:Ajinkya Sunil Patil  
D15B/42  
Lab-mpl**

### **Experiment 5**

### **Aim:Applying Navigation, Routing, and Gestures in a Flutter App**

**Introduction:**  
Flutter is a powerful framework for building cross-platform mobile applications that can run on Android, iOS, and the web. One of the key features that contribute to Flutter’s ease of use is its built-in support for navigation, routing, and gestures. These three components are essential for creating user-friendly and interactive apps, providing a seamless experience for users as they move between screens and interact with various elements.

In this writeup, we will explore the theoretical aspects of navigation, routing, and gestures in Flutter, covering their concepts, importance, and how they are typically implemented in Flutter apps.

**Theory:**

### **1. Navigation in Flutter**

**Navigation** refers to the process of transitioning between different screens or views (called widgets in Flutter) within an app. Most mobile apps consist of multiple screens such as a home screen, settings, profile, and so on. Users move between these screens by navigating from one to another.

In Flutter, navigation is typically managed using the **Navigator** class. The Navigator is a widget that manages a stack of pages, or routes, and provides methods for navigating between them. When a user moves from one page to another, the current page is pushed onto the stack, and when the user goes back, the page is popped from the stack.

Flutter uses a stack-based navigation system, where the stack follows the last-in, first-out (LIFO) principle. The stack's structure allows for easy transitions between screens and keeps track of the user’s navigation history. Common methods used in navigation are:

* **push**: Adds a new route to the stack.
* **pop**: Removes the current route from the stack and returns to the previous one.
* **pushReplacement**: Replaces the current route with a new one, removing the existing one.

#### **Importance of Navigation**

Navigation is essential in any app with multiple screens, as it allows the user to move seamlessly between them. Proper navigation ensures:

* A smooth user experience.
* Consistency in app flow and screen transitions.
* Efficient resource management, as only necessary screens are loaded into memory.

Effective navigation in an app also contributes to user retention and satisfaction, as it avoids confusion and makes the app easier to use.

### **2. Routing in Flutter**

**Routing** is the process of defining and controlling how the app responds to different navigation requests. A route represents a screen or a page in Flutter, and routing determines which route to load when navigating through the app. Flutter’s routing system is flexible and can be implemented in various ways, depending on the app's complexity.

Flutter supports two primary types of routing:

* **Imperative Routing**: This is the default approach where routes are defined dynamically as needed. Developers call Navigator methods to push and pop routes, explicitly controlling the navigation.
* **Declarative Routing**: This approach involves defining routes and navigation paths at the start. It’s useful in more complex apps where there is a need for predefined and predictable routing.

#### **Types of Routing**

* **Basic Routing**: In this simple form of routing, the developer manually defines routes using the Navigator’s push and pop methods. Basic routing is ideal for small apps with fewer screens, as it is relatively straightforward to implement.
* **Named Routing**: In more extensive apps, it is often beneficial to use named routes, which allow developers to define routes as named identifiers. Named routing helps improve code organization and makes navigation between screens more readable and easier to maintain.
* **On-Generate Route**: Flutter provides a callback called onGenerateRoute, which is used to handle unknown routes or to perform dynamic route generation based on user actions. This approach is often used when there are complex navigation patterns or when routes need to be constructed dynamically.
* **Nested Navigation**: Flutter supports nested navigation, which allows developers to create independent navigators within different sections of an app. This is helpful in scenarios where different parts of the app require independent navigation flows (e.g., navigating within tabs).

#### **Importance of Routing**

Routing is crucial because it determines how users navigate within an app. Proper routing provides:

* Clear paths for users to move through the app's content.
* Improved code organization, especially in larger apps with complex navigation flows.
* Flexibility to handle various navigation scenarios, including deep linking and nested routes.

### **3. Gestures in Flutter**

**Gestures** are actions that users perform on a mobile device’s touchscreen, such as tapping, swiping, or dragging. Flutter provides a robust gesture detection system that allows developers to capture user interactions and respond to them accordingly.

Flutter’s gesture system is based on widgets called **GestureDetectors**, which capture various gestures and provide callback functions that trigger when a gesture is detected. Some of the common gestures include:

* **Tap**: A quick touch on the screen.
* **Double-tap**: Two rapid touches on the screen.
* **Long-press**: A prolonged touch on the screen.
* **Swipe**: A quick drag across the screen, typically in a specific direction (left, right, up, or down).
* **Drag**: Moving the finger across the screen without lifting it.

#### **GestureDetector Widget**

The GestureDetector widget is the primary tool for capturing gestures in Flutter. It allows developers to define callback functions for specific gestures. For example, a button tap can be captured with the onTap callback, while a swipe action can be detected with onHorizontalDrag.

#### **Importance of Gestures**

Gestures play a vital role in modern mobile apps, as they:

* Provide an intuitive and natural way for users to interact with the app.
* Enhance the user experience by making the app feel more responsive and fluid.
* Allow developers to create dynamic interfaces with interactions such as scrolling, zooming, and dragging.

Gestures also form the backbone of many mobile interfaces, such as swipe-to-delete, pull-to-refresh, and pinch-to-zoom. Implementing gestures correctly ensures that users can navigate the app efficiently and enjoy a smooth experience.

### **4. Combining Navigation, Routing, and Gestures**

In a typical Flutter app, navigation, routing, and gestures work together to provide a cohesive and intuitive user experience. For instance:

* Navigation allows users to move between different screens.
* Routing defines how the app responds to navigation requests.
* Gestures enable users to interact with the app's elements, such as tapping buttons to navigate or swiping to trigger actions.

By combining these three elements, developers can create sophisticated and user-friendly apps that offer smooth transitions, clear navigation paths, and interactive gestures. For example, users can swipe through different pages in a gallery app, tap a button to navigate to the details of a photo, and use named routing to easily return to the gallery from any screen.

### **Conclusion**

Navigation, routing, and gestures are fundamental concepts in Flutter app development. Together, they create a seamless, intuitive, and interactive experience for users. Proper implementation of these components allows developers to build apps that are not only functional but also engaging and easy to use.

In Flutter, the navigation system, including the Navigator and routes, manages transitions between different screens. Routing ensures that users are guided through the app in a logical and efficient way, while gestures enhance interactivity by allowing users to perform actions through touch-based inputs. When these three elements are integrated into an app, they significantly contribute to its overall usability and design.

**Code:**  
import 'package:flutter/material.dart';

import 'package:font\_awesome\_flutter/font\_awesome\_flutter.dart';

import 'loan\_emi\_calculator.dart';

import 'time\_unit\_calculator.dart';

import 'currency\_converter.dart';

import 'unit\_converter.dart'; // ✅ Import Unit Converter Screen

class AllCalculatorsScreen extends StatelessWidget {

const AllCalculatorsScreen({super.key});

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: Text("All Calculators"),

backgroundColor: Colors.grey[900],

),

backgroundColor: Colors.black,

body: GridView.count(

padding: EdgeInsets.all(12),

crossAxisCount: 2,

crossAxisSpacing: 12,

mainAxisSpacing: 12,

children: [

\_buildCalculatorButton(context, "Calculator",

FontAwesomeIcons.calculator, Colors.orange),

\_buildCalculatorButton(context, "Time Converter & Calculations",

FontAwesomeIcons.clock, Colors.green),

\_buildCalculatorButton(context, "Currency Converter",

FontAwesomeIcons.moneyBill, Colors.blue),

\_buildCalculatorButton(context, "Unit Converter",

FontAwesomeIcons.ruler, Colors.purple), // ✅ FIXED

\_buildCalculatorButton(context, "Loan EMI Calculator",

FontAwesomeIcons.creditCard, Colors.red),

\_buildCalculatorButton(context, "Date Calculations",

FontAwesomeIcons.calendar, Colors.teal),

],

),

);

}

Widget \_buildCalculatorButton(

BuildContext context, String label, IconData icon, Color color) {

return GestureDetector(

// 🔹 Gesture detection for button tap

onTap: () {

print("$label button tapped!"); // 🔍 Debugging Print

// 🔹 Navigation logic for each calculator

if (label == "Unit Converter") {

Navigator.push(context,

MaterialPageRoute(builder: (context) => UnitConverterScreen())); // ✅ Navigate to Unit Converter Screen

} else if (label == "Loan EMI Calculator") {

Navigator.push(context,

MaterialPageRoute(builder: (context) => LoanEmiCalculator())); // ✅ Navigate to Loan EMI Calculator

} else if (label == "Time Converter & Calculations") {

Navigator.push(context,

MaterialPageRoute(builder: (context) => TimeUnitCalculator())); // ✅ Navigate to Time Unit Calculator

} else if (label == "Currency Converter") {

Navigator.push(

context,

MaterialPageRoute(

builder: (context) => CurrencyConverterScreen())); // ✅ Navigate to Currency Converter

} else {

ScaffoldMessenger.of(context).showSnackBar(SnackBar(

content: Text("$label is in working state"),

duration: Duration(seconds: 2),

));

}

},

child: Container(

decoration: BoxDecoration(

color: color,

borderRadius: BorderRadius.circular(12),

),

padding: EdgeInsets.all(16),

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

FaIcon(icon, size: 40, color: Colors.white),

SizedBox(height: 10),

Text(label,

textAlign: TextAlign.center,

style: TextStyle(color: Colors.white, fontSize: 16)),

],

),

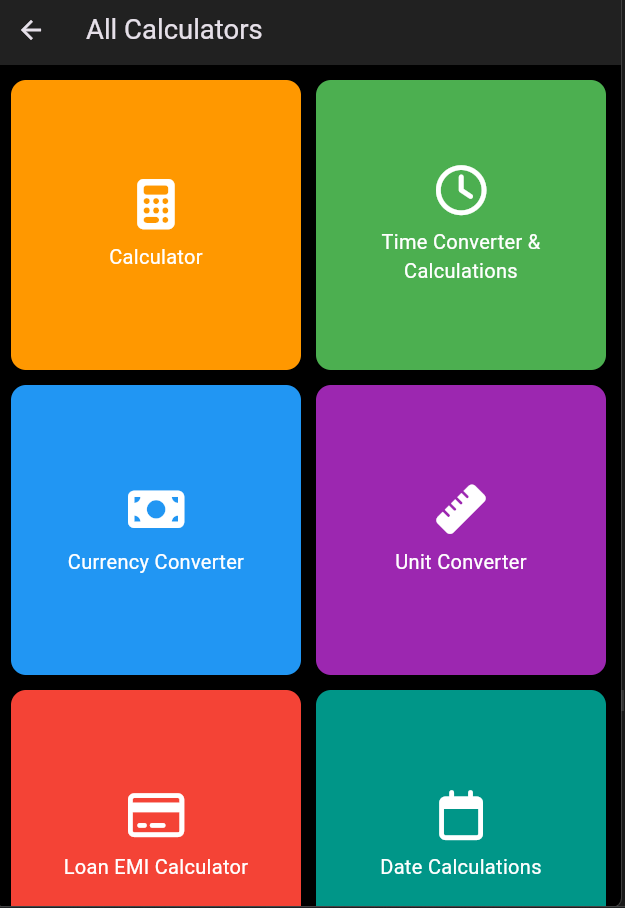
),

);

}

}  
**Highlighted Sections:**

* **Gesture Detection**: The GestureDetector is used to capture taps on the buttons.
* **Navigation**: The app navigates to the respective screen (Unit Converter, Loan EMI Calculator, etc.) based on the button tapped.

**OUTPUT:  
**