Name:Kshitij Hundre Experiment-5 D15C/18

**AIM**: - To apply navigation, routing, and gestures in Flutter App

## Theory:

Flutter is a powerful framework for building cross-platform mobile applications, and it provides efficient mechanisms for:

### 🔁 1. Navigation & Routing

Navigation allows moving between different screens (also called routes or pages) in a Flutter app. Flutter provides multiple methods to implement navigation:

**a) Basic Navigation (Navigator.push and Navigator.pop)**

- Navigator.push(context, MaterialPageRoute(builder: (_) => SecondPage()));
  Pushes a new route onto the stack.

- Navigator.pop(context);
  Pops the top-most route from the stack and returns to the previous screen.

**b) Named Routing**

- Define routes in MaterialApp's routes property:

**c) Navigation Stack**

Flutter uses a **stack-based** navigation model where each new screen is "pushed" onto a stack and can be "popped" to return to the previous screen.

### ✋ 2. Gestures in Flutter

Gestures are used to detect user interaction like taps, swipes, drags, etc.

Flutter uses the GestureDetector widget to handle gestures:

| Gesture Type | Widget/Callback Used |
|---|---|
| Tap | onTap |
| Double Tap | onDoubleTap |
| Long Press | onLongPress |
| Vertical Drag | onVerticalDragUpdate |
| Horizontal Drag | onHorizontalDragUpdate |

GestureDetector is a powerful tool for creating interactive UIs and responding to user inputs like swipe-to-dismiss, tap-to-select, or drag-to-move elements.

**InkWell vs GestureDetector:**

- GestureDetector: Pure logic-based gesture detection.
- InkWell: Similar, but adds **ripple/touch feedback** when tapped. Ideal for buttons.

## Combining Navigation & Gestures

A common real-world example is:

- User **taps a button** → navigates to another screen.
- User **swipes** to dismiss a card or perform an action. These interactions improve the **UX (User Experience)** by making apps feel smooth and intuitive.

## Code:

### Example: Gesture + Navigation Together

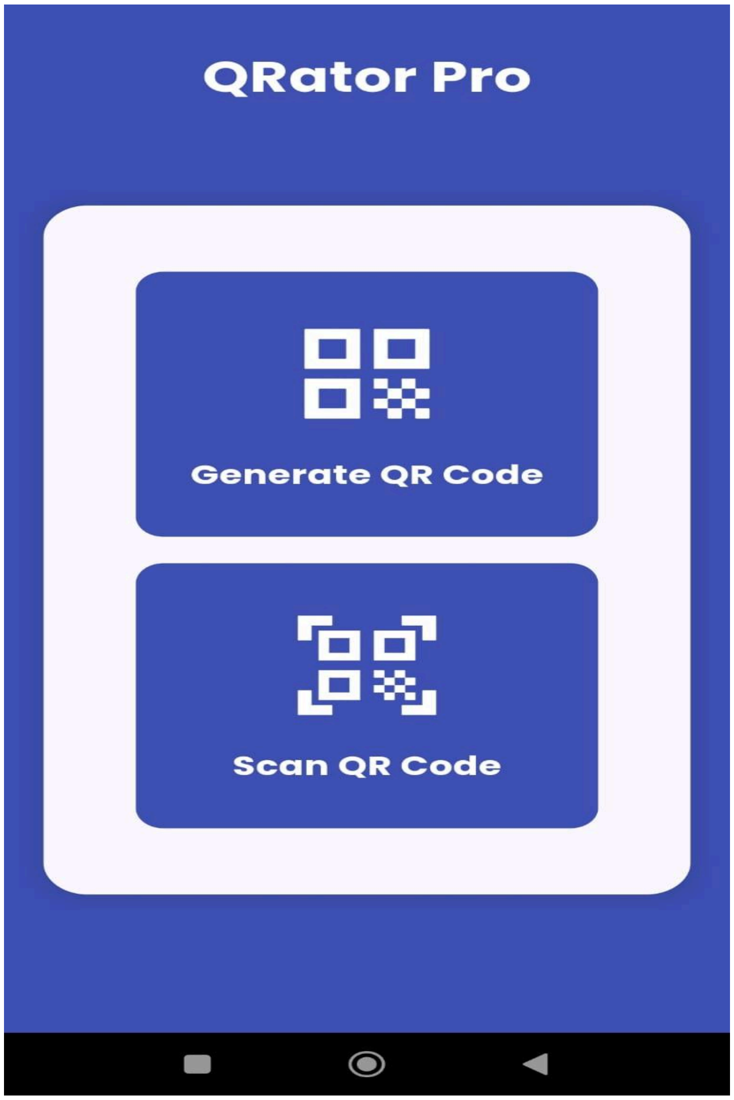
```
GestureDetector(
  onTap: () {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => QRGeneratorScreen()),
    );
  },
  child: Card(
    child: Padding(
      padding: EdgeInsets.all(16),
      child: Text("Tap to Generate QR"),
    ),
  ),
),
```

#### Navigate to Scanner from Your Home / Generator Screen

```
ElevatedButton.icon(
  icon: Icon(Icons.qr_code_scanner),
  label: Text("Scan QR"),
  onPressed: () async {
    final result = await Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => QRScanScreen()),
    );

    if (result != null) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Scanned: $result')),
      );
    }
```

## Output:



## Conclusion:

In this experiment, we successfully learned and implemented the concepts of **navigation**, **routing**, and **gesture detection** in Flutter. We used the Navigator class to move between screens, understood the difference between **basic and named routing**, and applied GestureDetector to handle various user interactions. This enhances the overall user experience by making the app more dynamic and interactive.