EXPERIMENT NO: - 05

Name:- Himesh Pathai Class:- D15A Roll:No: - 34

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

Theory: -

Navigation, Routing, and Gesture Handling in Flutter

In Flutter, screens or pages are referred to as routes, and each route is essentially a widget. This concept is similar to Activities in Android. Navigating between pages defines an app's workflow, and the mechanism for handling this is known as routing.

Flutter provides a built-in routing system using MaterialPageRoute, along with the Navigator.push() and Navigator.pop() methods to move between routes.

Additionally, gestures allow apps to respond to user interactions like taps, swipes, and drags, making applications more dynamic and user-friendly.

Navigation and Routing in Flutter

1. Using the Navigator Widget

Flutter's Navigator widget manages a stack of routes, enabling seamless navigation between screens.

- Pushing a Route: Moves to a new screen using Navigator.push().
- Popping a Route: Returns to the previous screen using Navigator.pop().

Example:

```
ElevatedButton(
  onPressed: () {
    Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => SecondScreen()),
    );
  },
  child: Text('Go to Second Screen'),
);
```

---

2. Using Named Routes

For larger applications, named routes provide a cleaner and more structured way to manage navigation.

Step 1: Define Routes in MaterialApp

```
MaterialApp(
  initialRoute: '/',
  routes: {
    '/': (context) => HomeScreen(),
    '/second': (context) => SecondScreen(),
  },
);
```

Step 2: Navigate Using Navigator.pushNamed()

```
Navigator.pushNamed(context, '/second');
```

Handling Gestures in Flutter

Gestures enable user interaction through taps, swipes, pinches, and drags. Flutter provides various widgets and gesture detectors to manage these interactions effectively.

1. Tap Gestures

Taps are one of the most common interactions and can be handled using:

- GestureDetector
- InkWell
- ElevatedButton

Example (Tap Gesture using GestureDetector):

```
GestureDetector(
  onTap: () {
    print("Tapped!");
  },
  child: Container(
    padding: EdgeInsets.all(20),
    color: Colors.blue,
    child: Text('Tap Me'),
  ),
);
```

2. Long Press Gestures

Long-press interactions can be captured using the onLongPress callback in GestureDetector or InkWell.

```
InkWell(
  onLongPress: () {
    print("Long Pressed!");
  },
  child: Container(
    padding: EdgeInsets.all(20),
    color: Colors.red,
    child: Text('Long Press Me'),
  ),
);
```

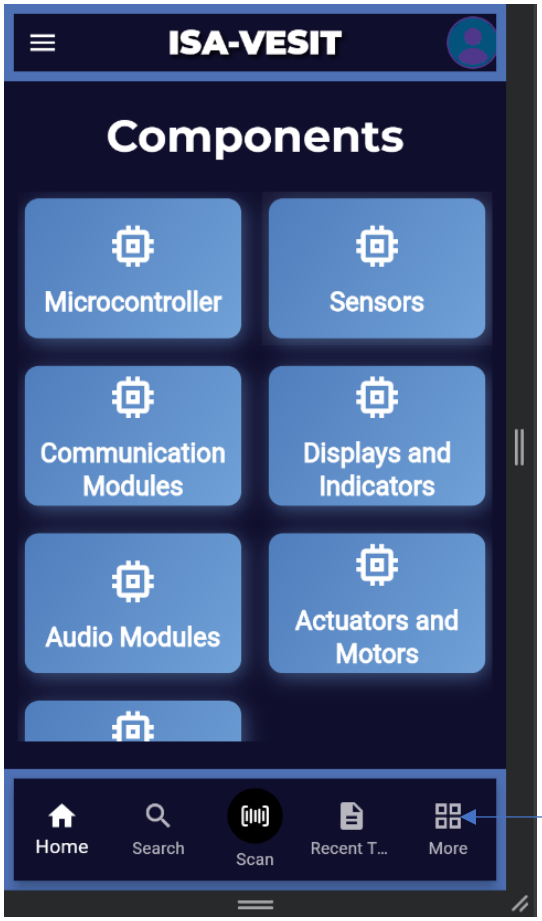
---

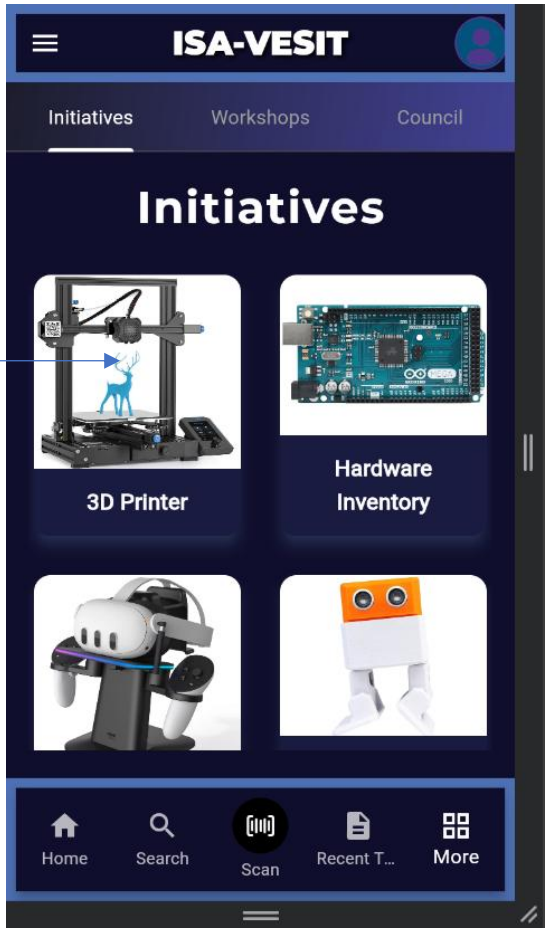
3. Swipe and Drag Gestures

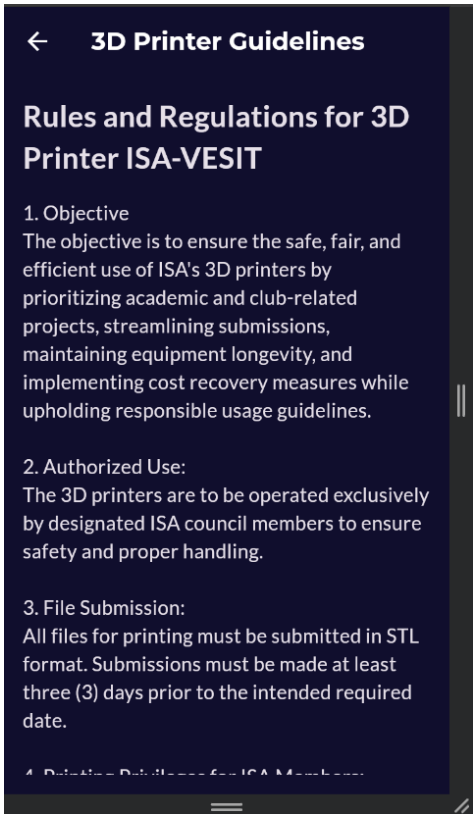
Flutter provides built-in methods like onHorizontalDragUpdate and onVerticalDragUpdate to detect swipe and drag actions.

Example (Swipe Detection):

```
GestureDetector(
  onHorizontalDragUpdate: (details) {
    if (details.primaryDelta! > 0) {
      print("Swiped Right!");
    } else {
      print("Swiped Left!");
    }
  },
  child: Container(
    padding: EdgeInsets.all(20),
    color: Colors.green,
    child: Text('Swipe Me'),
  ),
);
```


ISA-VESIT
Components
Microcontroller
Sensors
Communication Modules
Displays and Indicators
Audio Modules
Actuators and Motors
Home
Search
Scan
Recent T...
More


ISA-VESIT
Initiatives
Workshops
Council
Initiatives
3D Printer
Hardware Inventory
Home
Search
Scan
Recent T...
More


3D Printer Guidelines
Rules and Regulations for 3D Printer ISA-VESIT
1. Objective
The objective is to ensure the safe, fair, and efficient use of ISA's 3D printers by prioritizing academic and club-related projects, streamlining submissions, maintaining equipment longevity, and implementing cost recovery measures while upholding responsible usage guidelines.
2. Authorized Use:
The 3D printers are to be operated exclusively by designated ISA council members to ensure safety and proper handling.
3. File Submission:
All files for printing must be submitted in STL format. Submissions must be made at least three (3) days prior to the intended required date.