Aim: To apply navigation, routing and gestures in Flutter App

Theory:

Navigation in Flutter is the mechanism of transitioning between different screens or views (also called routes). Flutter follows a stack-based navigation model where screens are pushed and popped from the stack.

Purpose of Navigation:

- Allows movement between various UI components (screens)
- Helps build multi-screen mobile applications.
- Maintains user flow and logical screen transitions.

Basic Syntax:

- Navigate to a new screen:

Navigator.push(context, MaterialPageRoute(builder: (context) => NextScreen()));

- Navigate using a named route: Navigator.pushNamed(context, '/form');
- Go back to the previous screen: Navigator.pop(context);

Routing in Flutter:

Routing refers to the configuration of named paths and their corresponding screen widgets in an application. Flutter provides both named and unnamed routing options.

Importance of Routing:

- Organizes navigation paths in a centralized way.
- Makes large applications easier to manage and scale.
- Enables structured access to screens via route names.

Basic Syntax:

- Define routes in MaterialApp: MaterialApp(

initialRoute: '/', routes: {

'/': (context) => HomeScreen(), '/form': (context) => FormScreen(),

},

)

- Use route name to navigate: Navigator.pushNamed(context, '/form');

Gestures in Flutter:

Gestures refer to touch-based interactions such as taps, swipes, drags, and long presses. Flutter provides the GestureDetector widget to recognize and handle these interactions.

Purpose of Using Gestures:

- Makes the app more interactive.
- Captures and responds to user input.
- Enhances user experience by adding custom actions to touch events.

Basic Syntax:

- Detect a tap: GestureDetector(onTap: () {

// Code to execute on tap

},

child: Widget(), // Child widget to wrap

);

Other gestures include:

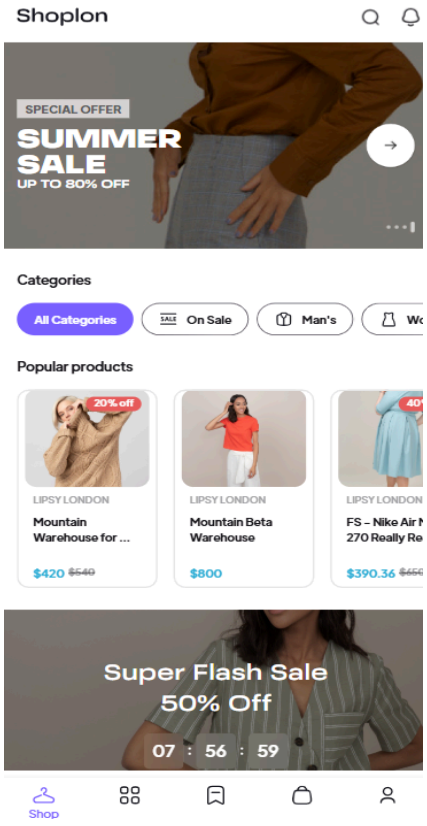
- onDoubleTap
- onLongPress

- onHorizontalDragUpdate, etc.

Code: https://github.com/Kingmaker-2/MPL-APP.git

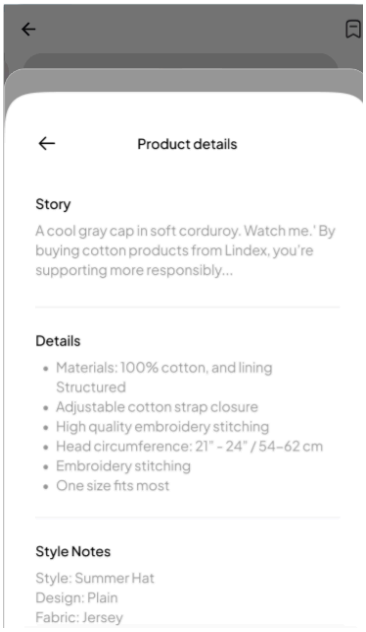
Output:

Page 1 (Home Page)

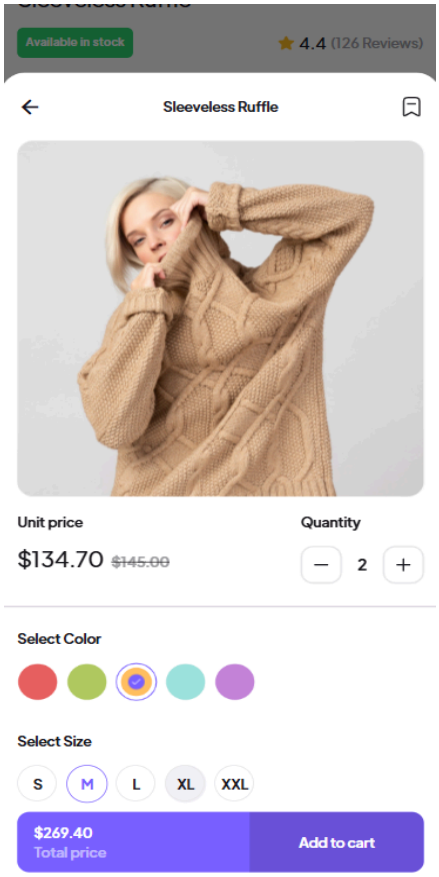


Page 2 Product Page (User clicked on one Product):

Product Details:



Add to Cart:



Conclusion: By applying navigation, routing, and gesture handling in a Flutter application, we have learned how to create smooth transitions between screens and build interactive user experiences. Understanding how to use Navigator, manage routes, and respond to gestures is essential for developing dynamic, multi-screen applications. These features play a key role in improving app usability and enabling intuitive user interactions.