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

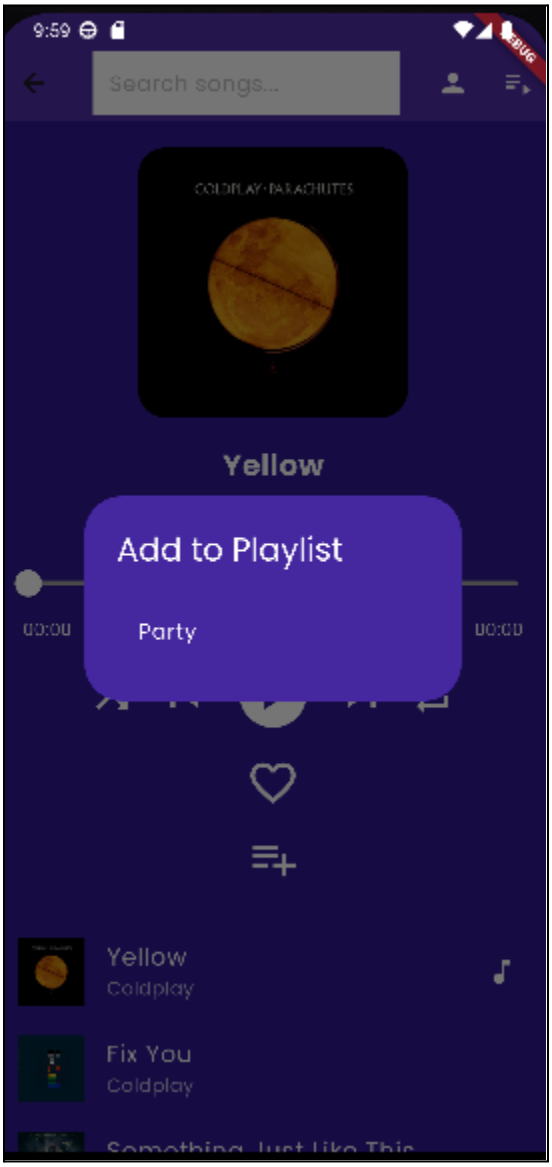
Gestures are user interactions like taps, swipes, long presses, etc., that your app can respond to.

Flutter provides the **GestureDetector** widget to handle different gestures.

Common gesture callbacks include:

- onTap – Detects a single tap.
- onDoubleTap – Detects a double-tap.
- onLongPress – Triggered when a user presses and holds.
- onPanUpdate – Detects dragging movements (like swipes).

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.