Navigation with Named Routes:

We know how to navigate to a new screen by creating a new route and manage it by using the Navigator. The Navigator maintains the stack-based history of routes. If there is a need to navigate to the same screen in many parts of the app, this approach is not beneficial because it results in code duplication. The solution to this problem can be removed by defining the named routes and can use the named routes for navigation.

We can work with named routes by using the Navigator.pushNamed() function. This function takes two required arguments (build context and string) and one optional argument. Also, we know about the MaterialPageRoute, which is responsible for page transition. If we do not use this, then it is difficult to change the page.

The following steps are necessary, which demonstrate how to use named routes.

Step 1: First, we need to create two screens.

Step 2: Define the routes.

Step 3: Navigate to the second screen using the Navigator.pushNamed() function.

Step 4: Use a Navigator.pop() function to return to the first screen.

Flutter Gestures:

Gestures are an interesting feature in Flutter that allows us to interact with the mobile app (or any touch-based device). Generally, gestures define any physical action or movement of a user in the intention of specific control of the mobile device. Some of the examples of gestures are:

- When the mobile screen is locked, you slide your finger across the screen to unlock it.
- Tapping a button on your mobile screen, and
- Tapping and holding an app icon on a touch-based device to drag it across screens.

We use all these gestures in everyday life to interact with your phone or touch-based device.

Flutter divides the gesture system into two different layers, which are given below:

1. Pointers
2. Gestures

1. Pointers:

Pointers are the first layer that represents the raw data about user interaction. It has events, which describe the location and movement of pointers such as touches, mice, and style across the screens. Flutter does not provide any mechanism to cancel or stop the pointer-events from being dispatched further. Flutter provides a Listener widget to listen to the pointer-events directly from the widgets layer. The pointer-events are categories into mainly four types:

PointerDownEvents: It allows the pointer to contact the screen at a particular location.

PointerMoveEvents: It allows the pointer to move from one location to another location on the screen.

PointerUpEvents: It allows the pointer to stop contacting the screen.

PointerCancelEvents: This event is sent when the pointer interaction is canceled.

2. Gestures:

It is the second layer that represents **semantic actions** such as tap, drag, and scale, which are recognized from multiple individual pointer events. It is also able to dispatch multiple events corresponding to gesture lifecycle like drag start, drag update, and drag end. Some of the popularly used gesture are listed below:

Tap: It means touching the surface of the screen from the fingertip for a short time and then releasing them. This gesture contains the following events:

- onTapDown
- onTapUp
- onTap
- onTapCancel

Double Tap: It is similar to a Tap gesture, but you need to tapping twice in a short time. This gesture contains the following events:

- onDoubleTap

Drag: It allows us to touch the surface of the screen with a fingertip and move it from one location to another location and then releasing them. Flutter categories the drag into two types:

1. **Horizontal Drag:** This gesture allows the pointer to move in a horizontal direction. It contains the following events:
    - onHorizontalDragStart
    - onHorizontalDragUpdate
    - onHorizontalDragEnd
2. **Vertical Drag:** This gesture allows the pointer to move in a vertical direction. It contains the following events:
    - onVerticalDragStart
    - onVerticalDragStart
    - onVerticalDragStart

Long Press: It means touching the surface of the screen at a particular location for a long time. This gesture contains the following events:

- onLongPress

Pan: It means touching the surface of the screen with a fingertip, which can move in any direction without releasing the fingertip. This gesture contains the following events:

- onPanStart
- onPanUpdate
- onPanEnd

Pinch: It means pinching (move one's finger and thumb or bring them together on a touchscreen) the surface of the screen using two fingers to zoom into or out of a screen.

Gesture Detector

Flutter provides a widget that gives excellent support for all types of gestures by using the **GestureDetector** widget. The GestureWidget is non-visual widgets, which is primarily used for detecting the user's gesture. The basic idea of the gesture detector is a **stateless** widget that contains parameters in its constructor for different touch events.

In some situations, there might be multiple gesture detectors at a particular location on the screen, and then the framework disambiguates which gesture should be called. The GestureDetector widget decides which gesture is going to recognize based on which of its callbacks are non-null.

11. Installation Steps / Performance Steps and Results –

Q1	Use Flutter Navigation and Routing in flutter App development.

Source code:

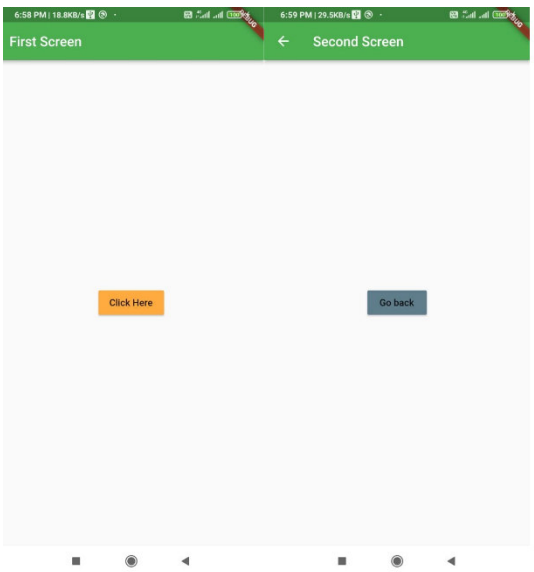
```
import 'package:flutter/material.dart';
//Example for explaining Flutter Navigation and Routing
void main() {
  runApp(MaterialApp(
    title: 'Flutter Navigation',
    theme: ThemeData(
      // This is the theme of your application.
      primarySwatch: Colors.green,
    ),
    home: FirstRoute(),
  ));
}

class FirstRoute extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('First Screen'),
      ),
      body: Center(
        child: RaisedButton(
          child: Text('Click Here'),
```

```
          color: Colors.orangeAccent,
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => SecondRoute()),
            );
          },
        ),
      ),
    );
  }
}

class SecondRoute extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text("Second Screen"),
      ),
      body: Center(
        child: RaisedButton(
          color: Colors.blueGrey,
          onPressed: () {
            Navigator.pop(context);
          },
          child: Text('Go back'),
        ),
      ),
    );
  }
}
```

Output:



Q2	Use flutter image.asset to add image in flutter App development.

Source code:

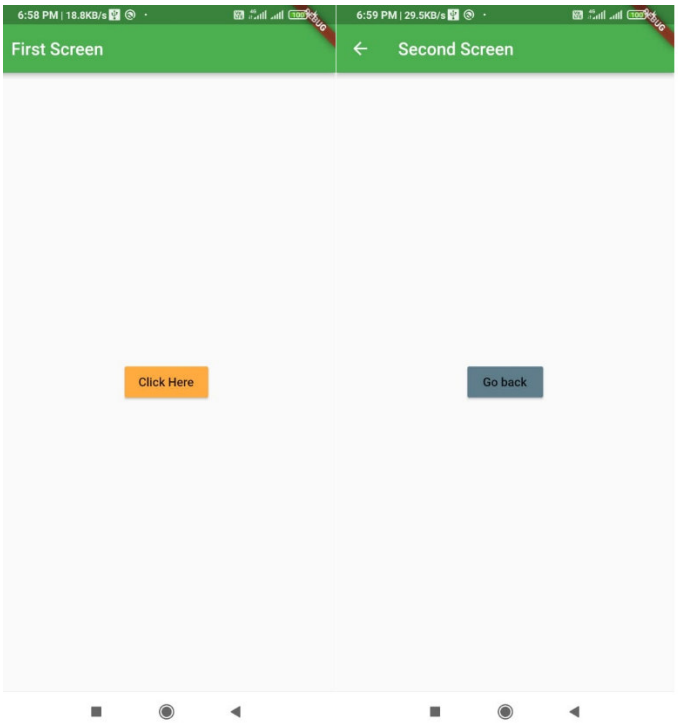
```
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
// This widget is the root
// of your application

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Insert Image Demo'),
        ),
        body: Center(
          child: Column(
            children: <Widget>[
              Image.asset('assets/images/famtitlab.jpg'),
            ],
          ),
        ),
      ),
    );
  }
}
```

Output:



Q3	Use flutter to implement Gesture in App development.

Source code:

```
import 'package:flutter/material.dart';
//This example explains implementation of Gesture on view of App
void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo Application',
      theme: ThemeData(
        primarySwatch: Colors.green,
      ),
      home: MyHomePage(),
    );
  }
}

class MyHomePage extends StatefulWidget {
  @override
  MyHomePageState createState() => new MyHomePageState();
}

class MyHomePageState extends State<MyHomePage> {
  @override
  Widget build(BuildContext context) {
    return new Scaffold(
      appBar: new AppBar(
        title: new Text('Gestures Example'),
        centerTitle: true,
      ),
      body: new Center(
          child: GestureDetector(
              onTap: () {
                print('Box Clicked');
              },
              child: Container(
                height: 60.0,
                width: 120.0,
                padding: EdgeInsets.all(10.0),
                decoration: BoxDecoration(
                  color: Colors.blueGrey,
                  borderRadius: BorderRadius.circular(15.0),
                ),
                child: Center(child: Text('Click Me')),
              ))),
```

```
    );
  }
}
```

Output: