MAD PWA Lab

EXPERIMENT NO.5

Name : Pranathi Narsupalli
Div : D15B Roll no. : 45

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

THEORY :

Flutter's navigation, routing, and gesture systems are critical components of building smooth, intuitive, and interactive mobile applications.

1. Navigation:

Navigation in Flutter refers to the process of moving between different screens or pages within an application. Flutter offers a `Navigator` class that manages a stack of `Route` objects. Each route represents a screen or page in the application. The `Navigator` facilitates pushing new routes onto the stack, popping existing routes off the stack, and managing transitions between routes.

2. Routing:

Routing in Flutter involves defining the routes available within an application and mapping them to specific widgets. Flutter uses a routing table to define the mapping between route names and corresponding widget builders. Developers can define routes using named routes or on-the-fly route creation using anonymous routes.

Here's a basic example of route definition in Flutter:

```
dart
MaterialApp(
  routes: {
    '/home': (context) => HomePage(),
    '/profile': (context) => ProfilePage(),
  },
  initialRoute: '/home',
)
```

3. Gestures:

Gestures in Flutter enable touch-based interaction with widgets. Flutter provides a rich set of gesture recognizers that detect various touch events such as taps, drags, swipes, pinches, etc. These recognizers allow developers to create custom touch-based interactions within their applications. Some commonly used gesture recognizers in Flutter include `GestureDetector`, `InkWell`, `GestureDetector`, and `Draggable`.

Here's a simple example of using GestureDetector to detect taps:

```
dart
  GestureDetector(
    onTap: () {
      print('Widget tapped');
    },
    child: Container(
      width: 200,
      height: 200,
      color: Colors.blue,
      child: Center(
        child: Text('Tap me'),
      ),
    ),
  )
```

4. Integration: Flutter's navigation, routing, and gesture systems are often integrated to create seamless user experiences. For example, developers can use gesture recognizers to detect swipes or taps on certain widgets and trigger navigation events accordingly. Similarly, navigation events can be used to push or pop routes within the application.

Here's an example of using a gesture to navigate to a new screen:

```
dart
  GestureDetector(
    onTap: () {
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => SecondScreen()),
      );
```

},
child: Text('Go to second screen'),
)

INPUT :

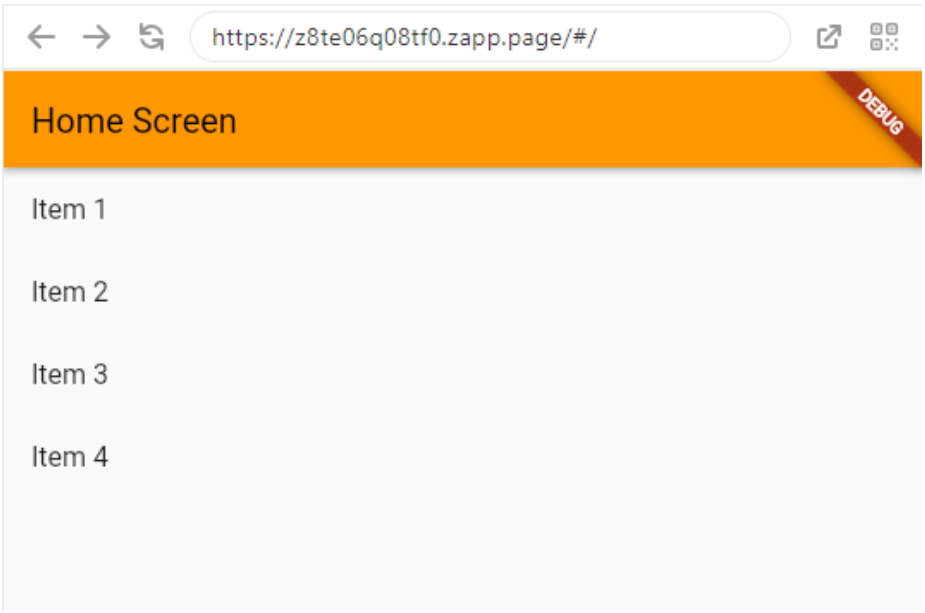
1. **Navigation and routing**

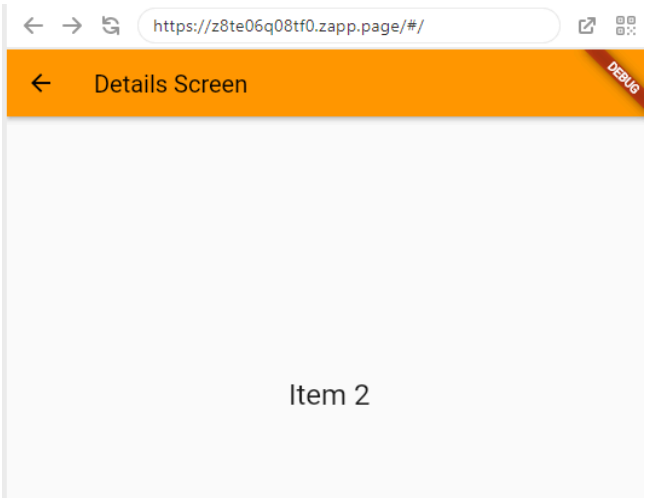
```
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Navigation Example',
      theme: ThemeData(
        primarySwatch: Colors.orange,
      ),
      initialRoute: '/',
      routes: {
        '/': (context) => HomeScreen(),
        '/details': (context) => DetailsScreen(),
      },
    );
  }
}

class HomeScreen extends StatelessWidget {
  final List<String> items = ['Item 1', 'Item 2', 'Item 3', 'Item 4'];

  @override
  Widget build(BuildContext context) {
    return Scaffold(
```

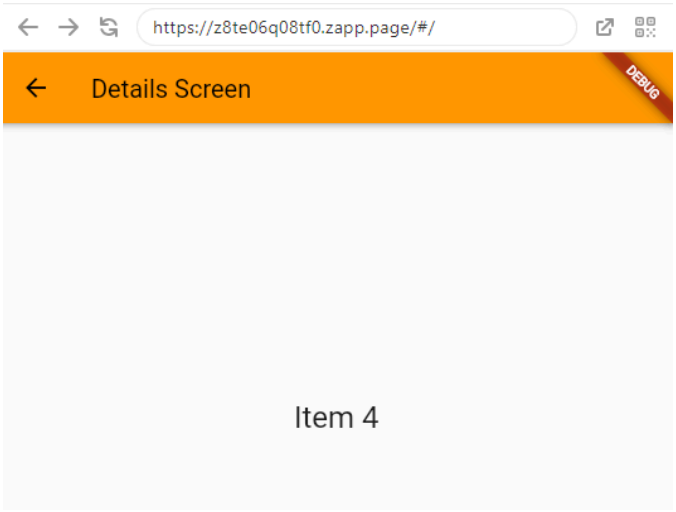
```
      appBar: AppBar(
        title: Text('Home Screen'),
      ),
      body: ListView.builder(
        itemCount: items.length,
        itemBuilder: (context, index) {
          return ListTile(
            title: Text(items[index]),
            onTap: () {
              // Navigate to details screen with data
              Navigator.pushNamed(
                context,
                '/details',
                arguments: items[index],
              );
            },
          );
        },
      ),
    );
  }
}

class DetailsScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    // Extract the data passed from the home screen
    final String item = ModalRoute.of(context)!.settings.arguments as String;

    return Scaffold(
      appBar: AppBar(
        title: Text('Details Screen'),
      ),
      body: Center(
        child: Hero(
          tag: 'hero-tag-$item', // Unique tag for Hero animation
```

```
        child: Material(
          child: InkWell(
            onTap: () {
              // Navigate back to the home screen
              Navigator.pop(context);
            },
            child: Container(
              padding: EdgeInsets.all(16.0),
              child: Text(
                item,
                style: TextStyle(fontSize: 24.0),
              ),
            ),
          ),
        ),
      ),
    ),
  );
   }
}
```

OUTPUT :







2. **Gestures in Flutter :**

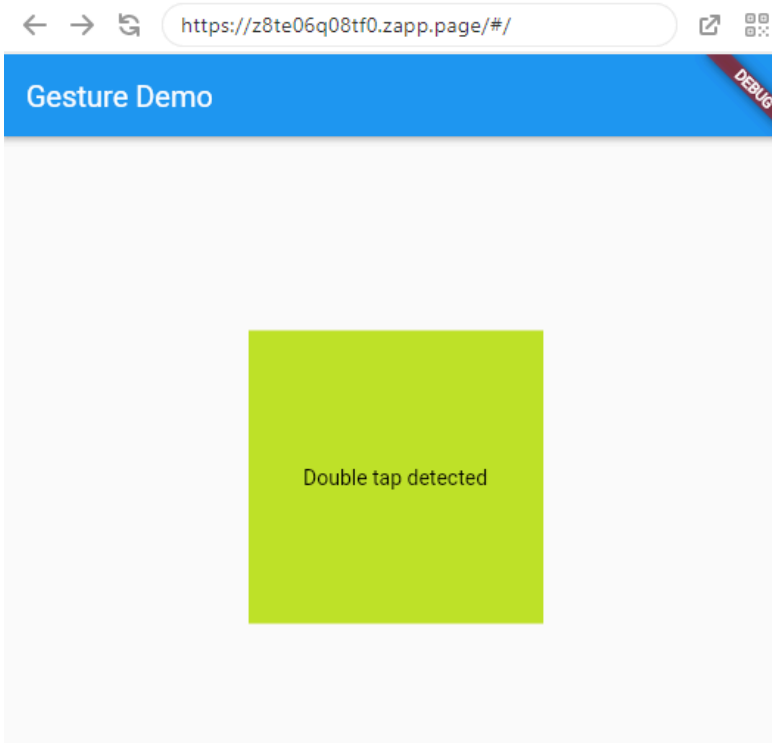
INPUT :

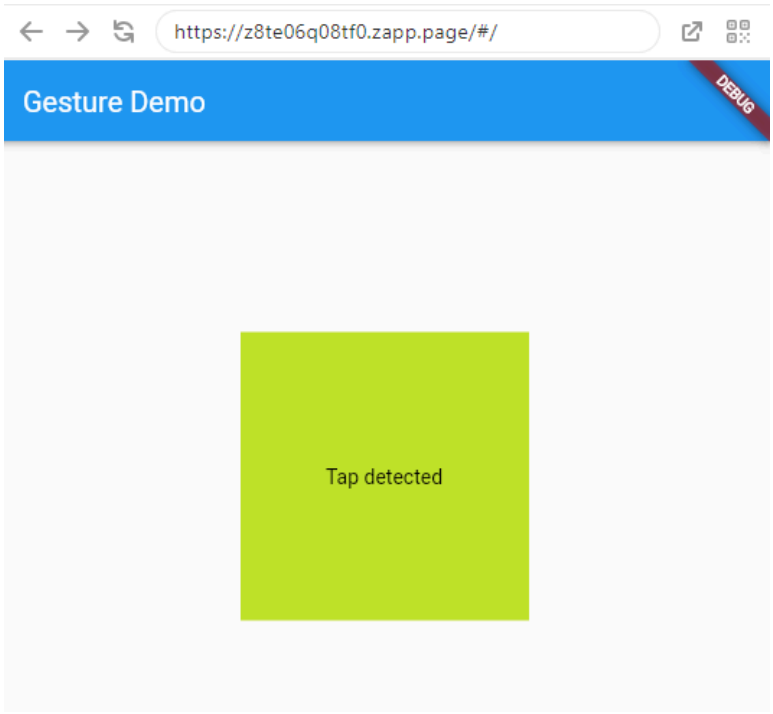
```
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
```

```
      title: 'Flutter Gestures Example',
      home: GestureDemo(),
    );
  }
}

class GestureDemo extends StatefulWidget {
  @override
  _GestureDemoState createState() => _GestureDemoState();
}

class _GestureDemoState extends State<GestureDemo> {
  String gestureResult = 'No gesture detected';

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Gesture Demo'),
      ),
      body: Center(
        child: GestureDetector(
          onTap: () {
            // Handle tap gesture
            setState(() {
              gestureResult = 'Tap detected';
            });
          },
          onDoubleTap: () {
            // Handle double tap gesture
            setState(() {
              gestureResult = 'Double tap detected';
            });
```

```
        },
        child: Container(
          width: 200,
          height: 200,
          color: Colors.blue,
          child: Center(
            child: Text(
              gestureResult,
              style: TextStyle(color: Colors.white),
            ),
          ),
        ),
      ),
    ),
  );
 }
}
```

OUTPUT :





CONCLUSION : In this experiment, we have gained insights into Flutter's routing, navigation, and gesture implementation. We learned to seamlessly navigate between screens using named routes, enhancing user experience. We implemented gesture detection, employing onTap and onDoubleTap callbacks to create interactive UIs. This hands-on experience deepened my understanding of Flutter's versatile capabilities in crafting dynamic and engaging mobile applications.