EXPERIMENT NO. 5

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

Theory:

Navigation: In Flutter, navigation involves moving between different screens or "routes" within an app. This can be accomplished using the Navigator widget, which manages a stack of routes. When navigating to a new screen, you push a new route onto the stack, and when navigating back, you pop the current route off the stack. Flutter provides various navigation methods like Navigator.push() to push a new route onto the stack, Navigator.pop() to remove the current route from the stack, and Navigator.pushNamed() to navigate to a named route.

Routing: Routing in Flutter refers to the process of defining and managing named routes for different screens or pages in the app. Named routes provide a way to navigate to a specific screen using a unique identifier rather than directly referencing the widget class. This improves code readability, organization, and maintenance, especially in larger apps. Routes are typically defined in the MaterialApp widget using the routes parameter, where each route is associated with a corresponding widget.

Gestures: Gestures in Flutter enable users to interact with the app through touch-based actions like taps, swipes, pinches, and long-presses. Flutter provides a comprehensive set of gesture detection widgets such as GestureDetector, InkWell, and LongPressGestureDetector, which allow developers to detect and handle various user gestures. For example, you can use GestureDetector to detect taps, drags, and other gestures on any widget, and then respond to those gestures by invoking specific actions or navigation events.

Process:

1. Define Routes: Begin by defining routes for each screen in your app. This involves creating a
mapping between route names and corresponding widget classes.
2. Navigate Between Screens: Use navigation methods to move between screens based on user interactions, such as button clicks or gestures. You can push new routes onto the stack,
pop routes off the stack, or replace routes as needed.

3. Pass Data Between Screens: Utilize route parameters to pass data between screens when
navigating. This allows screens to communicate with each other and update their UI accordingly.
4. Handle Navigation Events: Implement logic to handle navigation events and respond to user
actions appropriately. This may involve showing loading indicators, validating inputs, or performing other tasks before navigating to a new screen.
5. Handle Navigation Back: Consider how users will navigate back to previous screens using
the system back button or gestures. Ensure that your app handles navigation back correctly and
maintains a consistent user experience.

Code:

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

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

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter UI Example',
      theme: ThemeData(
        primarySwatch: Colors.purple,
      ),
      home: const MyHomePage(),
    );
  }
}
```

```
class MyHomePage extends StatefulWidget {
  const MyHomePage({Key? key}) : super(key: key);

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  List<String> tasks = [];
  late String newTask; // Variable to hold the value entered in the
TextFormField

  @override
  void initState() {
    super.initState();
    newTask = ''; // Initialize newTask to empty string
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Today\'s Tasks'),
      ),
      body: Container(
        decoration: const BoxDecoration(
          image: DecorationImage(
            image: AssetImage('assets/bgimg.png'), // Add your image path
            fit: BoxFit.cover,
          ),
        ),
        child: SingleChildScrollView(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.stretch,
            children: <Widget>[
              const SizedBox(height: 20),
              for (int i = 0; i < tasks.length; i++)
                TaskTile(
                  title: tasks[i],
```

```
                onDelete: () {
                  setState(() {
                    tasks.removeAt(i);
                  });
                },
              ),
            const SizedBox(height: 40),
            const Text(
              'Add New Task',
              style: TextStyle(
                fontSize: 20,
                fontWeight: FontWeight.bold,
              ),
            ),
            const SizedBox(height: 20),
            TextFormField(
              decoration: const InputDecoration(
                filled: true,
                fillColor: Colors.white,
                labelText: 'Enter task name',
                labelStyle: TextStyle(fontSize: 20),
                border: OutlineInputBorder(),
              ),
              onChanged: (value) {
                setState(() {
                  newTask = value;
                });
              },
              onFieldSubmitted: (value) {
                setState(() {
                  tasks.add(value);
                });
                // Clear the TextFormField after adding task
                newTask = '';
              },
            ),
            const SizedBox(height: 20),
            ElevatedButton(
              onPressed: () {
                setState(() {
```

```
                    if (newTask.isNotEmpty) {
                      tasks.add(newTask);
                      // Clear the TextFormField after adding task
                      newTask = '';
                    }
                  });
                },
                child: const Text('Add Task'),
              ),
            ],
          ),
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          Navigator.push(
            context,
            MaterialPageRoute(builder: (context) => SettingsScreen()),
          );
        },
        child: const Icon(Icons.settings),
      ),
    );
  }
}

class TaskTile extends StatelessWidget {
  final String title;
  final VoidCallback onDelete;

  const TaskTile({
    Key? key,
    required this.title,
    required this.onDelete,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onLongPress: () {
```

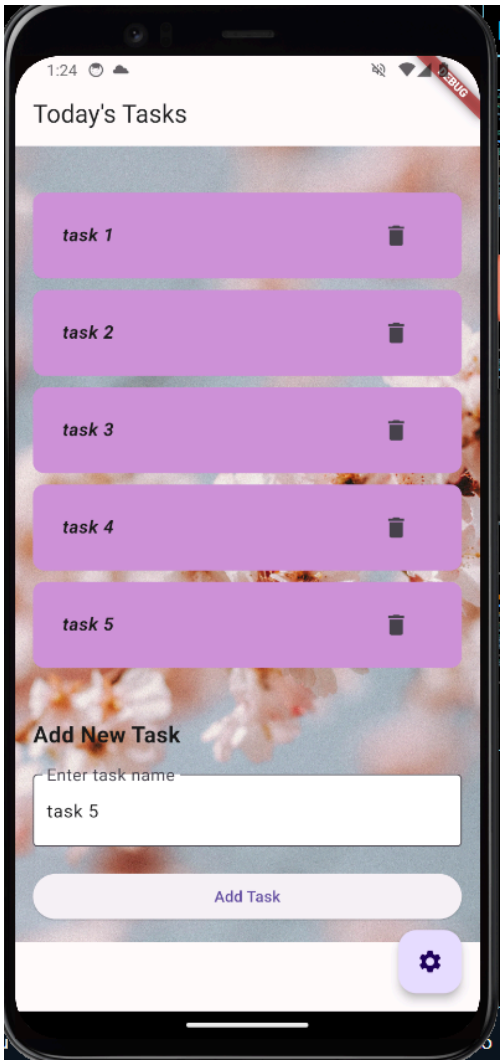
```
        showDialog(
          context: context,
          builder: (_) => AlertDialog(
            title: const Text('Delete Task'),
            content: Text('Are you sure you want to delete "$title"?'),
            actions: <Widget>[
              TextButton(
                onPressed: () {
                  Navigator.pop(context);
                },
                child: const Text('Cancel'),
              ),
              TextButton(
                onPressed: onDelete,
                child: const Text('Delete'),
              ),
            ],
          ),
        );
      },
      child: Container(
        margin: const EdgeInsets.symmetric(vertical: 5.0),
        padding: const EdgeInsets.all(10.0),
        decoration: BoxDecoration(
          color: Colors.purple.shade200,
          borderRadius: BorderRadius.circular(10.0),
        ),
        child: ListTile(
          title: Text(
            title,
            style: const TextStyle(
              fontFamily: 'Montserrat',
              fontSize: 16,
              fontWeight: FontWeight.bold,
              fontStyle: FontStyle.italic,
            ),
          ),
          trailing: IconButton(
            icon: const Icon(Icons.delete),
            onPressed: () {
```

```
              showDialog(
                context: context,
                builder: (_) => AlertDialog(
                  title: const Text('Delete Task'),
                  content: Text('Are you sure you want to delete
"$title"?'),
                  actions: <Widget>[
                    TextButton(
                      onPressed: () {
                        Navigator.pop(context);
                      },
                      child: const Text('Cancel'),
                    ),
                    TextButton(
                      onPressed: onDelete,
                      child: const Text('Delete'),
                    ),
                  ],
                ),
              );
            },
          ),
        ),
      ),
    );
  }
}


class SettingsScreen extends StatefulWidget {
  const SettingsScreen({Key? key}) : super(key: key);

  @override
  _SettingsPageState createState() => _SettingsPageState();
}

class _SettingsPageState extends State<SettingsScreen> {
  bool enableGestures = true;

  @override
```

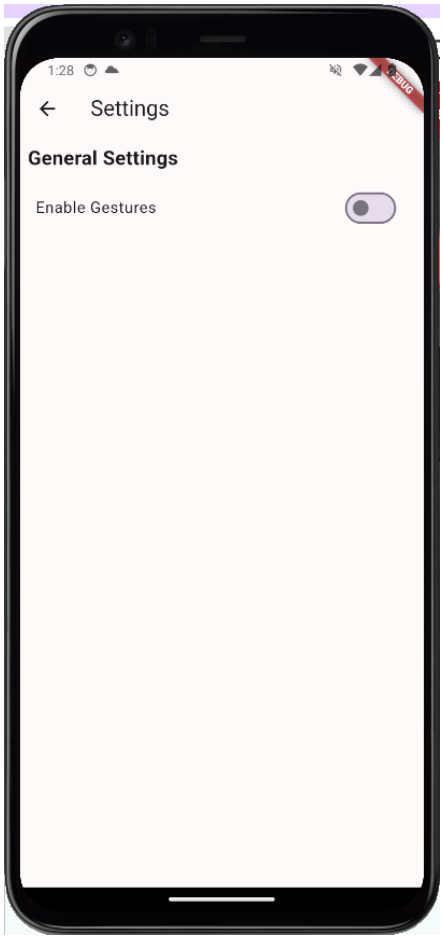
```
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Settings'),
      ),
      body: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          const Padding(
            padding: EdgeInsets.all(8.0),
            child: Text(
              'General Settings',
              style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
            ),
          ),
          ListTile(
            title: const Text('Enable Gestures'),
            trailing: Switch(
              value: enableGestures,
              onChanged: (value) {
                setState(() {
                  enableGestures = value;
                });
              },
            ),
          ),
        ],
      ),
    );
  }
}
```

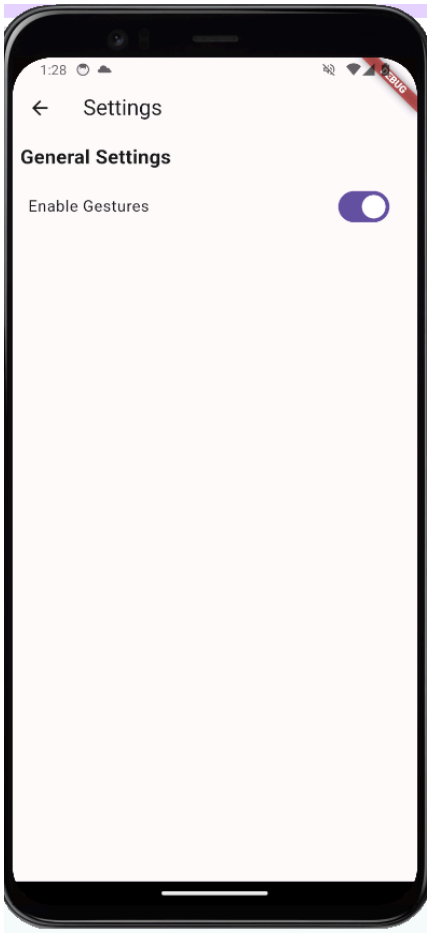
Output:

Home Screen with settings button



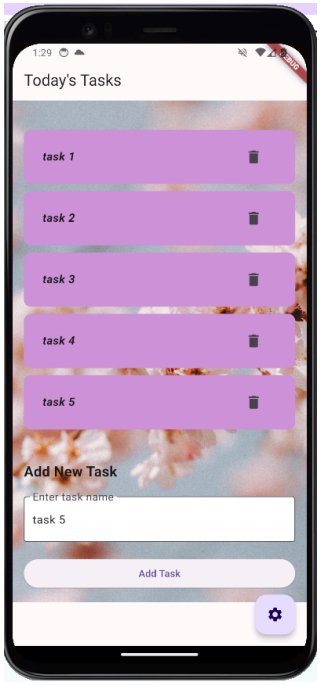
On pressing Settings button, it navigates us to a new settings page as follows



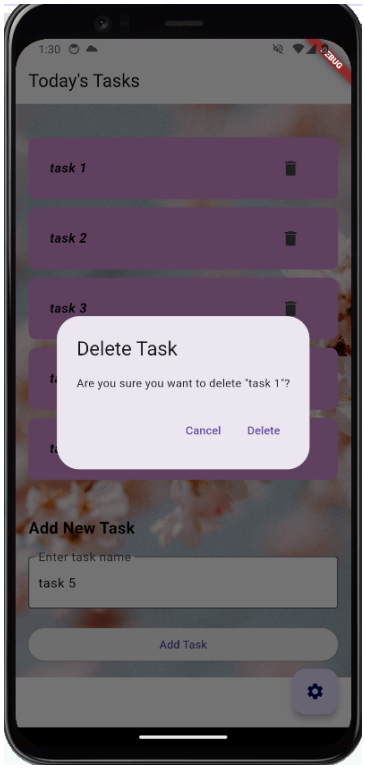


On toggling the gestures button, it enables gestures

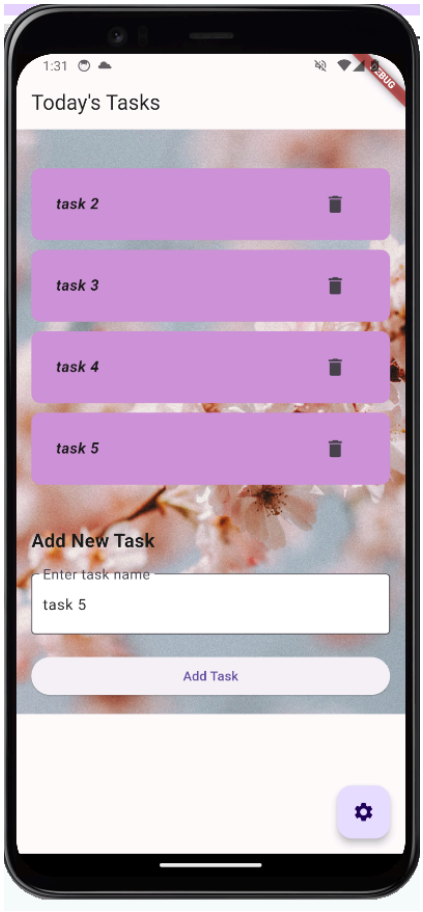
We can navigate to the home page on pressing back button.



On press and holding any one task it gives a confirmation dialog of if we want to delete a task



On pressing delete, it deletes the task.



Conclusion: Hence we have successfully applied navigation, routing and gestures on our flutter app.In summary, incorporating navigation, routing, and gestures in a Flutter app ensures smooth user interaction and intuitive navigation between screens. By defining routes, detecting gestures, and integrating confirmation dialogs, we create a seamless user experience, enhancing app usability and functionality.