Experiment 5

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

Theory:

Navigation:

Navigation is the process of moving between different screens or views within your app. It's crucial for providing users with a seamless way to explore your app's features and content. In Flutter, navigation is managed using the Navigator class, which maintains a stack of routes representing the app's navigation history.

- Implicit navigation: This involves directly pushing new routes onto the navigation stack using Navigator.push() and popping routes off the stack using Navigator.pop(). It's suitable for simple navigation flows.
- Named routes: With named routes, you define a route table in your app's main.dart file, mapping route names to corresponding widgets. This approach makes navigation more declarative and easier to manage, especially in larger apps with multiple screens. You can then navigate to named routes using Navigator.pushNamed() and Navigator.popAndPushNamed().
- Nested navigation: Sometimes, you may need to manage navigation independently within different parts of your app, such as tab bars, bottom navigation bars, or drawers. Flutter supports nested navigation by using multiple instances of the Navigator widget. Each nested navigator maintains its own navigation stack, allowing you to handle navigation within a specific context without affecting the rest of the app.

- Routing:
  Routing involves defining the routes available in your app and handling navigation requests to those routes. Key concepts include:
  - Route management: You define a route table in your app's main.dart file, which maps route names to corresponding widget builders or constructors. This table serves as a central registry for all the screens in your app.
  - Route transitions: Flutter provides various route transition animations out-of-the-box, such as slide transitions, fade

transitions, and scale transitions. You can customize these animations or create your own by subclassing the PageRoute class and implementing custom transition effects.
    - Route parameters: Routes can accept parameters or data that influence the behavior or content of the destination screen. You can pass parameters to named routes using the arguments parameter of Navigator.pushNamed(), allowing you to customize the destination screen based on the context of the navigation request.
- Gestures:
  Gestures enable users to interact with your app through touch input, making the user experience more intuitive and engaging. Key concepts include:
    - Gesture recognizers: Flutter provides a rich set of gesture recognizers that you can attach to widgets to detect and handle various types of user interactions. For example, GestureDetector allows you to detect taps, double taps, long presses, drags, and more.
    - Gesture callbacks: Gesture recognizers trigger callbacks when specific gestures are detected. For example, you can use the onTap, onDoubleTap, onLongPress, onHorizontalDragUpdate, and onVerticalDragUpdate callbacks to respond to user input and implement interactive behaviors.
    - Gesture widgets: Flutter provides specialized gesture recognizer widgets like InkWell, Draggable, Dismissible, and GestureDetector. These widgets encapsulate common gesture handling patterns and make it easier to add interactivity to your app's UI components.

Code:

Make a folder screens in the lib folder. Under screens folder make a home.dart,feed.dart,post_screen.dart,favorite_screen.dart,post_screen.dart,sear ch.dart,profile_screen.dart

Main.dart

```
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/material.dart';
import 'package:thread_clone_flutter/firebase_options.dart';
import 'package:thread_clone_flutter/screens/home.dart';
```

```
import 'screens/login.dart';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp(options: DefaultFirebaseOptions.currentPlatform);
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(),
      home: StreamBuilder<User?>(
          stream: FirebaseAuth.instance.authStateChanges(),
          builder: (context, snapshot) {
            if (snapshot.hasData && snapshot.data != null) {
              return const Home();
            } else {
              return const LoginScreen();
            }
          }),
    );
  }
}
```

Home.dart

```
import 'package:flutter/material.dart';
import 'package:sliding_up_panel/sliding_up_panel.dart';
import 'package:thread_clone_flutter/screens/post_screen.dart';
import 'package:thread_clone_flutter/screens/profile_screen.dart';

import 'favorite_screen.dart';
import 'feed.dart';
import 'search.dart';

class Home extends StatefulWidget {
  const Home({super.key});

  @override
```

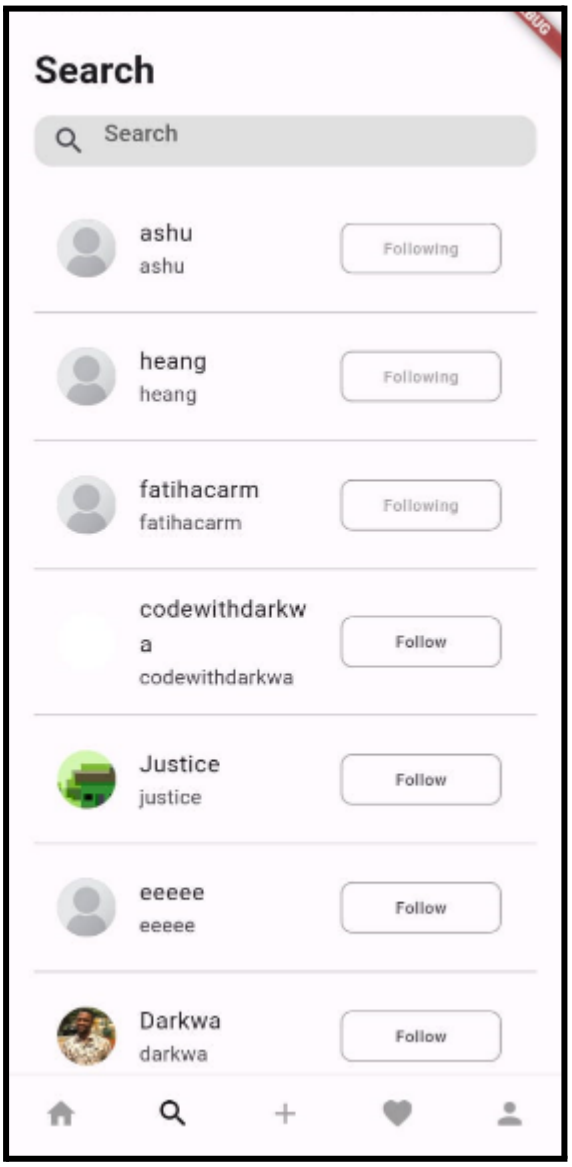
```
  State<Home> createState() => _HomeState();
}

class _HomeState extends State<Home> {
  int selectedIndex = 0;

  List<Widget> pages = [];
  PanelController panelController = PanelController();

  @override
  void initState() {
    pages = [
      const FeedScreen(),
      const SearchScreen(),
      PostScreen(panelController: panelController),
      const FavoriteScreen(),
      const ProfileScreen(),
    ];
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SlidingUpPanel(
        controller: panelController,
        minHeight: 0,
        maxHeight: MediaQuery.of(context).size.height * 0.8,
        borderRadius: const BorderRadius.only(
          topLeft: Radius.circular(25),
          topRight: Radius.circular(25),
        ),
        panelBuilder: (ScrollController sc) {
          return PostScreen(panelController: panelController);
        },
        body: pages[selectedIndex],
      ),
      bottomNavigationBar: BottomNavigationBar(
        currentIndex: selectedIndex,
        selectedItemColor: Colors.black,
        unselectedItemColor: Colors.grey,
        showSelectedLabels: false,
        showUnselectedLabels: false,
        onTap: (index) {
          if (index == 2) {
            panelController.isPanelOpen
                ? panelController.close()
                : panelController.open();
```

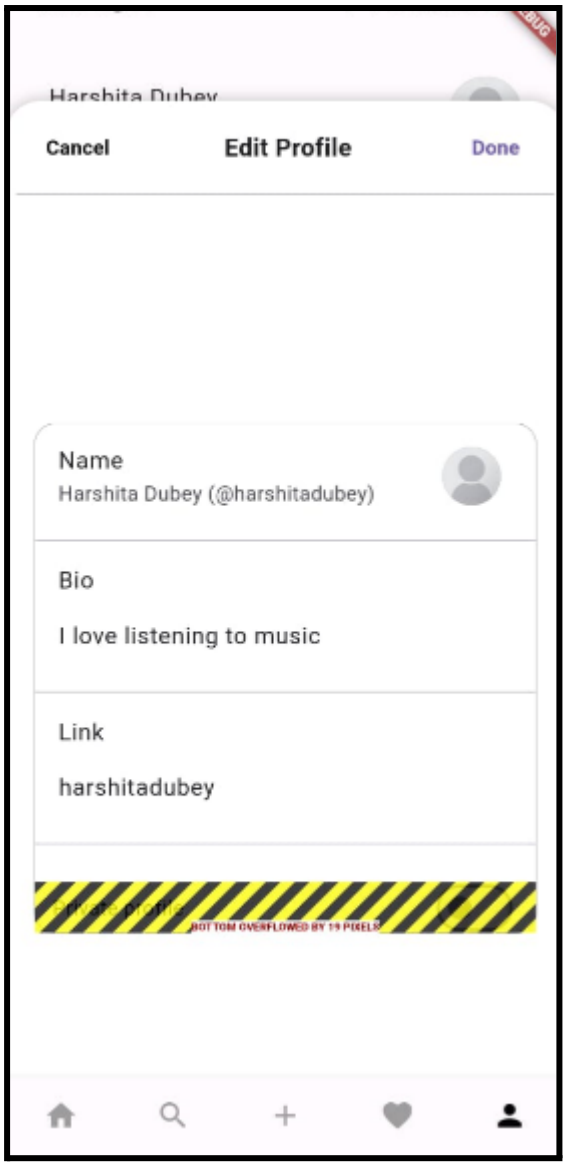
```
        } else {
          panelController.close();
          setState(() {
            selectedIndex = index;
          });
        }
      },
      type: BottomNavigationBarType.fixed,
      items: const [
        BottomNavigationBarItem(icon: Icon(Icons.home), label: 'Feed'),
        BottomNavigationBarItem(icon: Icon(Icons.search), label: 'search'),
        BottomNavigationBarItem(icon: Icon(Icons.add), label: 'post'),
        BottomNavigationBarItem(
            icon: Icon(Icons.favorite), label: 'favorite'),
        BottomNavigationBarItem(icon: Icon(Icons.person), label: 'profile'),
      ],
    ),
  );
 }
}
```

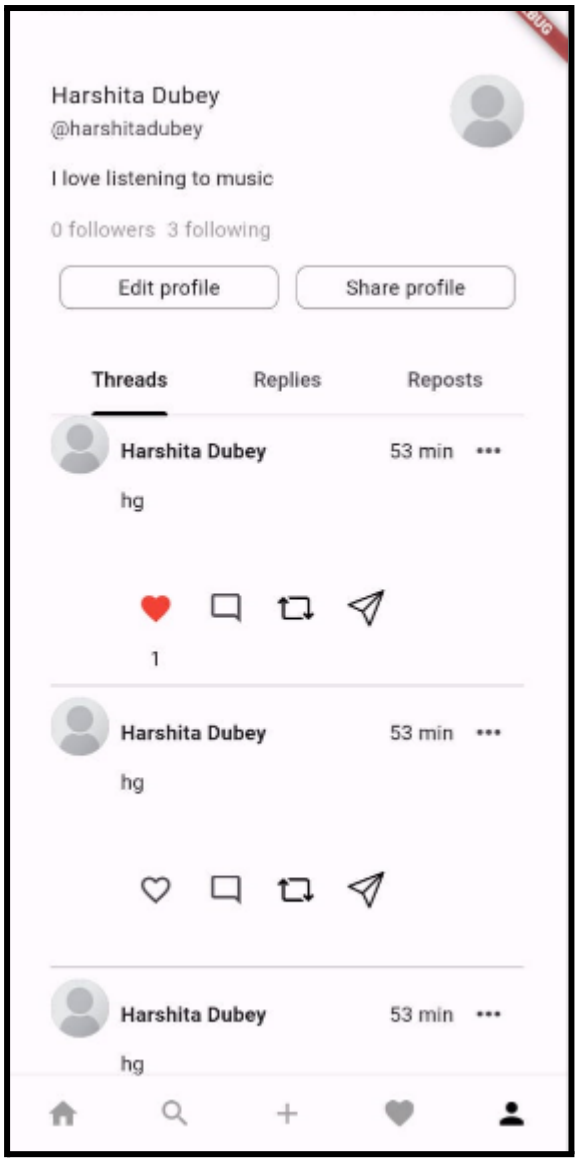
feed.dart

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

class FeedScreen extends StatefulWidget {
const FeedScreen({super.key});

@override

State createState() => _FeedScreenState();

}
class _FeedScreenState extends State {

@override

Widget build(BuildContext context) {
 return const Scaffold( body: Center( child: Text("Feed Screen"),
),
);
```

```
 }
}
```


Search
Search
ashu
ashu
Following
heang
heang
Following
fatihacarm
fatihacarm
Following
codewithdarkwa
codewithdarkwa
Follow
Justice
justice
Follow
eeeee
eeeee
Follow
Darkwa
darkwa
Follow


Harshita Dubey
Cancel
Edit Profile
Done
Name
Harshita Dubey (@harshitadubey)
Bio
I love listening to music
Link
harshitadubey



CONCLUSION

In conclusion navigation, routing, and gestures are fundamental components of building a user-friendly and interactive Flutter app. By mastering these concepts, I have created seamless and engaging user experience that facilitates effortless navigation, smooth transitions between screens, and intuitive touch interactions.