Name: Kanish Jadhwani Class: D15B Roll No.: 22

Experiment-5

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

Theory :

Navigation and routing are fundamental concepts in mobile applications, enabling users to transition between different pages or screens. In Flutter, screens and pages are referred to as routes, each being represented by a widget.

In Flutter, routes are akin to Activities in Android and ViewControllers in iOS. They facilitate the movement between pages, defining the workflow of the application.

Navigating between pages is handled through routing, with Flutter providing tools like MaterialPageRoute, Navigator.push(), and Navigator.pop() for basic navigation operations.

Pushing a Screen: Use Navigator.push to navigate to a new screen.
Example:

```
Navigator.push(context, MaterialPageRoute(builder: (context) => DetailsScreen()));
```

Popping a Screen: Use Navigator.pop to go back to the previous screen.
Example:

```
Navigator.pop(context);
```

Routing: Routing:

In the context of Flutter, involves defining the paths or routes that lead to different screens in your app. It allows you to organize and structure the flow of your application. Flutter supports both named routes and unnamed (or default) routes.

Navigation:

Navigation in Flutter involves moving between different screens or pages within the app. Flutter offers the Navigator widget to manage the navigation stack and execute common navigation tasks.

Routing:

Routing in Flutter entails defining paths or routes leading to different screens, enabling the organization and structuring of the application flow. Flutter supports both named and unnamed (default) routes.

Named Routes:

Named routes are identified by unique string identifiers, offering a more structured and maintainable way to navigate between screens. You can define named routes within the MaterialApp widget using the routes property.

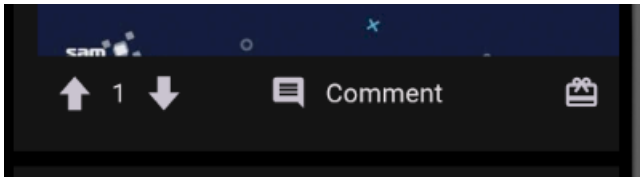
Code :

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

final themeNotifierProvider = StateNotifierProvider<ThemeNotifier,
ThemeData>((ref) {
  return ThemeNotifier();
});

class Pallete {
  // Colors
  static const blackColor = Color.fromRGBO(1, 1, 1, 1); // primary color
  static const greyColor = Color.fromRGBO(26, 39, 45, 1); // secondary color
  static const drawerColor = Color.fromRGBO(18, 18, 18, 1);
  static const whiteColor = Colors.white;
  static var redColor = Colors.red.shade500;
  static var blueColor = Colors.blue.shade300;

  // Themes
  static var darkModeAppTheme = ThemeData.dark().copyWith(
    scaffoldBackgroundColor: blackColor,
    cardColor: greyColor,
    appBarTheme: const AppBarTheme(
      backgroundColor: drawerColor,
      iconTheme: IconThemeData(
        color: whiteColor,
      ),
    ),
    drawerTheme: const DrawerThemeData(
      backgroundColor: drawerColor,
    ),
    primaryColor: redColor,
    backgroundColor: drawerColor, // will be used as alternative background
color
  );
```

```
  static var lightModeAppTheme = ThemeData.light().copyWith(
    scaffoldBackgroundColor: whiteColor,
    cardColor: greyColor,
    appBarTheme: const AppBarTheme(
      backgroundColor: whiteColor,
      elevation: 0,
      iconTheme: IconThemeData(
        color: blackColor,
      ),
    ),
    drawerTheme: const DrawerThemeData(
      backgroundColor: whiteColor,
    ),
    primaryColor: redColor,
    backgroundColor: whiteColor,
  );
}

class ThemeNotifier extends StateNotifier<ThemeData> {
  ThemeMode _mode;
  ThemeNotifier({ThemeMode mode = ThemeMode.dark})
      : _mode = mode,
        super(
        Pallete.darkModeAppTheme,
      ) {
    getTheme();
  }

  ThemeMode get mode => _mode;

  void getTheme() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    final theme = prefs.getString('theme');

    if (theme == 'light') {
      _mode = ThemeMode.light;
      state = Pallete.lightModeAppTheme;
    } else {
      _mode = ThemeMode.dark;
      state = Pallete.darkModeAppTheme;
    }
  }

  void toggleTheme() async {
```

```
    SharedPreferences prefs = await SharedPreferences.getInstance();

    if (_mode == ThemeMode.dark) {
      _mode = ThemeMode.light;
      state = Pallete.lightModeAppTheme;
      prefs.setString('theme', 'light');
    } else {
      _mode = ThemeMode.dark;
      state = Pallete.darkModeAppTheme;
      prefs.setString('theme', 'dark');
    }
  }
}
```

Output :-



Conclusion : Hence, we have successfully implemented the required functionalities for our app.