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

Theory:

➔ **Navigation and Routing**

Navigation and routing are some of the core concepts of all mobile applications, which allows the user to move between different pages. We know that every mobile application contains several screens for displaying different types of information. For example, an app can have a screen that contains various products. When the user taps on that product, immediately it will display detailed information about that product.

In Flutter, the screens and pages are known as routes, and these routes are just a widget. In Android, a route is similar to an Activity, whereas, in iOS, it is equivalent to a ViewController

In any mobile app, navigating to different pages defines the workflow of the application, and the way to handle the navigation is known as routing. Flutter provides a basic routing class MaterialPageRoute and two methods

Navigator.push() and Navigator.pop() that shows how to navigate between two routes. The following steps are required to start navigation in your application.

➔ **Navigation:**

Navigation in Flutter refers to the ability to move between different screens or pages within an app. Flutter provides a Navigator widget to manage the navigation stack and perform common navigation operations.

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.

- Named Routes:

Named routes are routes identified by a unique string identifier. They provide a more organized and maintainable way to navigate between screens. You can define named routes in the MaterialApp widget using the routes property.

Example,

```
MaterialApp( routes: {
'/': (context) => HomeScreen(),
'/details': (context) => DetailsScreen(),
},
)
```

- Unnamed (Default) Routes:

-Unnamed routes refer to the default route that is used when the app starts.

-You can set the initialRoute property in MaterialApp to specify the initial route.

Example,

```
MaterialApp(
initialRoute: '/', routes: {
'/': (context) => HomeScreen(),
'/details': (context) => DetailsScreen(),
},
)
```

Gestures in Flutter:

Gestures in Flutter refer to user interactions with the screen, such as taps, swipes, pinches, etc. 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. Flutter provides various widgets for gesture detection:

GestureDetector:

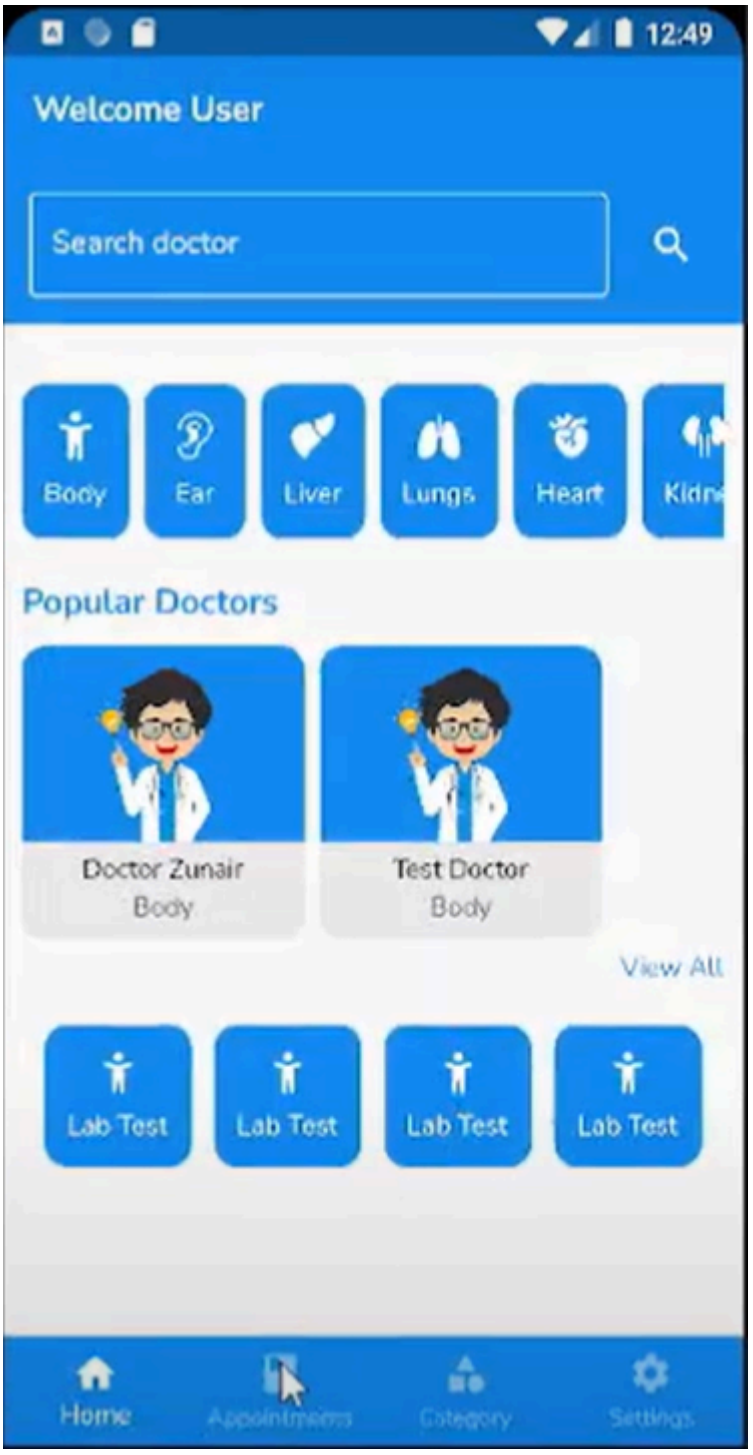
The GestureDetector widget is a versatile widget that can detect various gestures. It wraps its child and provides callbacks for different types of gestures.

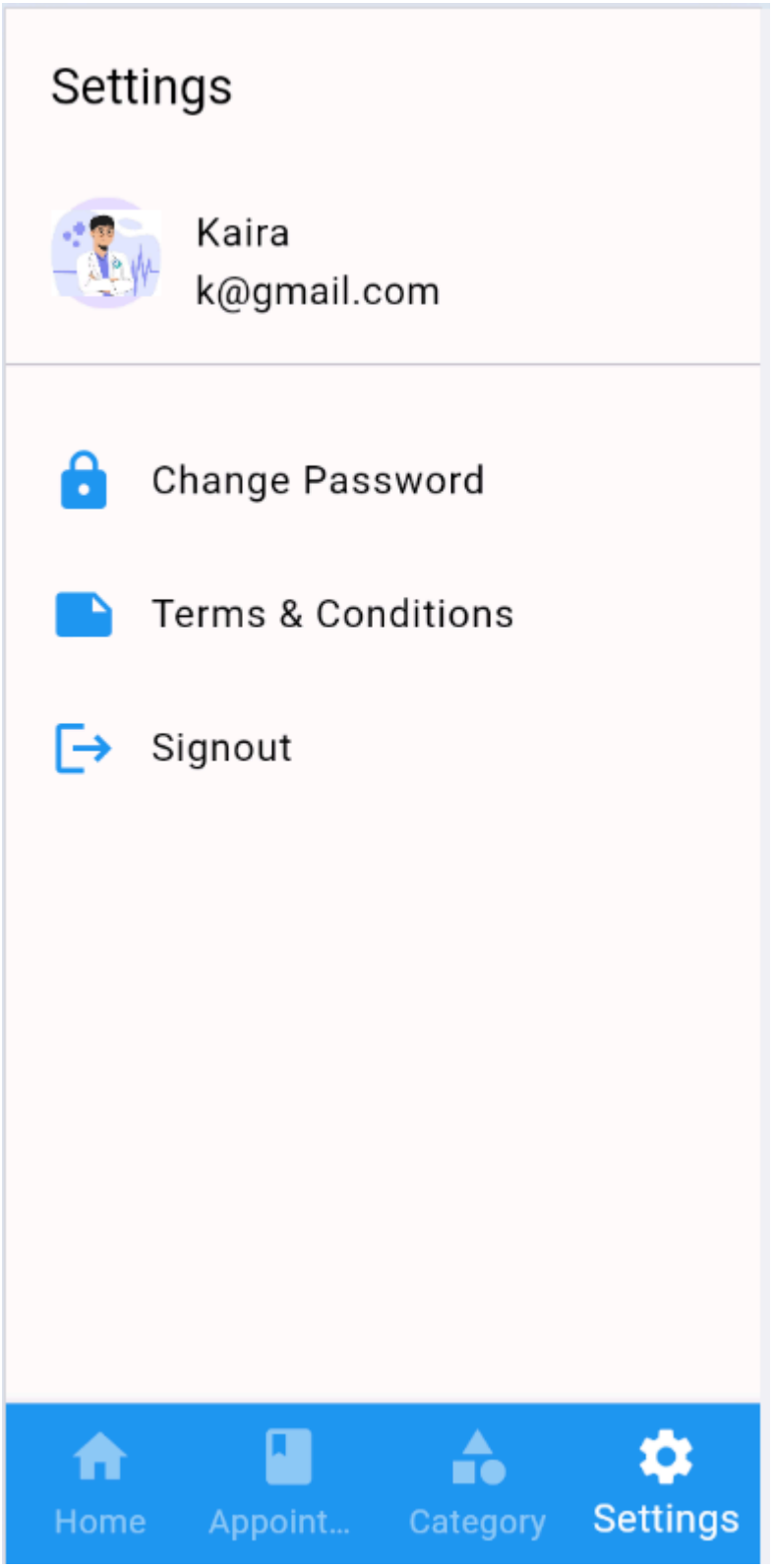
Example,

```
GestureDetector( onTap: () {
// Handle tap gesture
},
onDoubleTap: () {
// Handle double tap gesture
},
// Other gesture callbacks...
child: YourWidget(),
)
```

Home.dart :

```dart
import 'package:doctorapp/consts/consts.dart';
import 'package:doctorapp/views/appointment_view/appointment_view.dart';
import 'package:doctorapp/views/category_view/category_view.dart';
import 'package:doctorapp/views/home_view/home_view.dart';
import 'package:doctorapp/views/setting_view/settings_view.dart';
import 'package:cloud_firestore/cloud_firestore.dart';

import '../doctor_view/doctor_view.dart';
import '../login_view/login_view.dart';

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

     @override
 State<Home> createState() => _HomeState();
}

class _HomeState extends State<Home>{

 int selectedIndex = 0;
 List screenList = [
   const HomeView(),
   const AppointmentView(),
   const CategoryView(),
   // const DoctorView(),
   const SettingsView(),
 ];
 @override
 Widget build(BuildContext context){
 return Scaffold(
    body: screenList.elementAt(selectedIndex),
     bottomNavigationBar: BottomNavigationBar(
       unselectedItemColor: Colors.white.withOpacity(0.5),
       selectedItemColor: AppColors.whiteColor,
       selectedLabelStyle: const TextStyle(
         color: AppColors.whiteColor,
       ),
       selectedIconTheme: const IconThemeData(
         color: AppColors.whiteColor,
       ),
       backgroundColor: AppColors.blueColor,
       type: BottomNavigationBarType.fixed,
       currentIndex: selectedIndex,
       onTap: (value){
         setState(() {
           selectedIndex = value;
         });
       },
       items: const [
       BottomNavigationBarItem(icon: Icon(Icons.home), label: "Home" ),
       BottomNavigationBarItem(icon: Icon(Icons.book), label: "Appointment" ),
       BottomNavigationBarItem(icon: Icon(Icons.category), label: "Category" ),
       BottomNavigationBarItem(icon: Icon(Icons.settings), label: "Settings" ),
       ],
      ),
     );
   }
  }
```


12:49
Welcome User
Search doctor
Body
Ear
Liver
Lungs
Heart
Popular Doctors
Doctor Zunair
Body
Test Doctor
Body
View All
Lab Test
Lab Test
Lab Test
Lab Test
Home
Appointments
Category
Settings


Settings
Kaira
k@gmail.com
Change Password
Terms & Conditions
Signout
Home
Appoint...
Category
Settings

Conclusion : Through this experiment, we've learned a lot about how to use Flutter for moving between screens smoothly, setting up routes, and using gestures for interaction. We got really good at making the app flow nicely between different sections using specific names for each part. This made the app easier to use and more enjoyable for people. Overall, this hands-on experience helped me understand how flexible Flutter can be for making really cool mobile apps.