Lab Manual for CSM3114 - Framework-Based Mobile Application Development

Prepared by Mohamad Nor Hassan*

October 2024

*Universiti Malaysia Terengganu

The Flutter framework let the students to develop cross-platform mobile applications which can run on Android or iOS platforms.

This lab session will introduce to the student on the development of basic mobile applications focusing on the Flutter and Dart fundamentals that emphasise on core Flutter and dart syntax when developing the solutions.

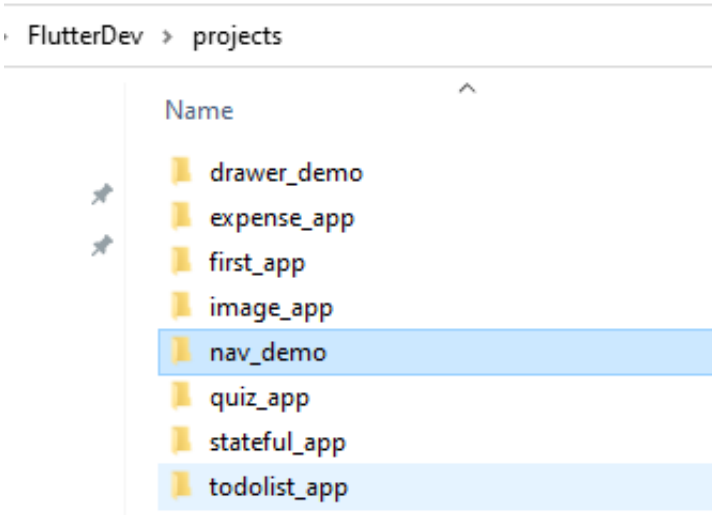
The learning outcomes of this lab session are:

1. Implementing the concept of navigation across the screens in the mobile apps.
2. Using *Drawer* to perform navigation process in the mobile app.
3. Passing data to the next screen.

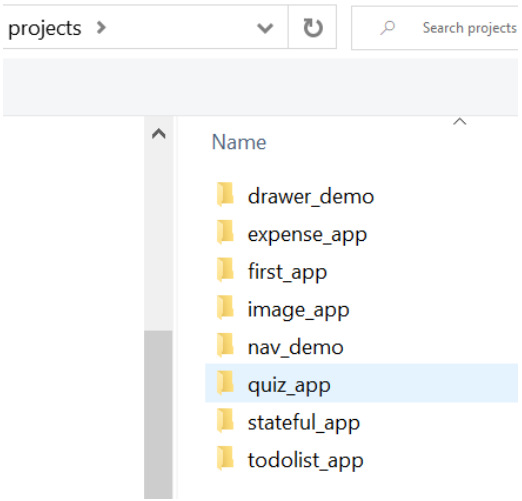
1 Navigation of Screens

1.1 Creating a Flutter project

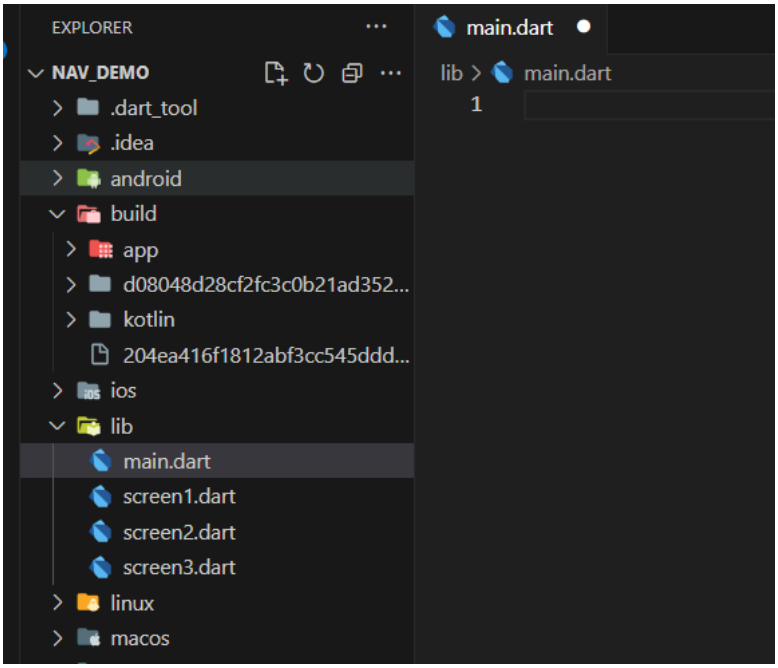
1. Open your command prompt.
2. Go to your current Flutter's project folder. [**Note**: Your command to navigate to *FlutterDev*'s folder might be different if you use DOS command prompt..!]



3. Creating the Flutter's project known as *nav_demo*.
4. Open your *FlutterDev*'s folder. You need to ensure the *nav_demo* created.
5. Go to VS Code IDE. Open *nav_demo*'s project folder.

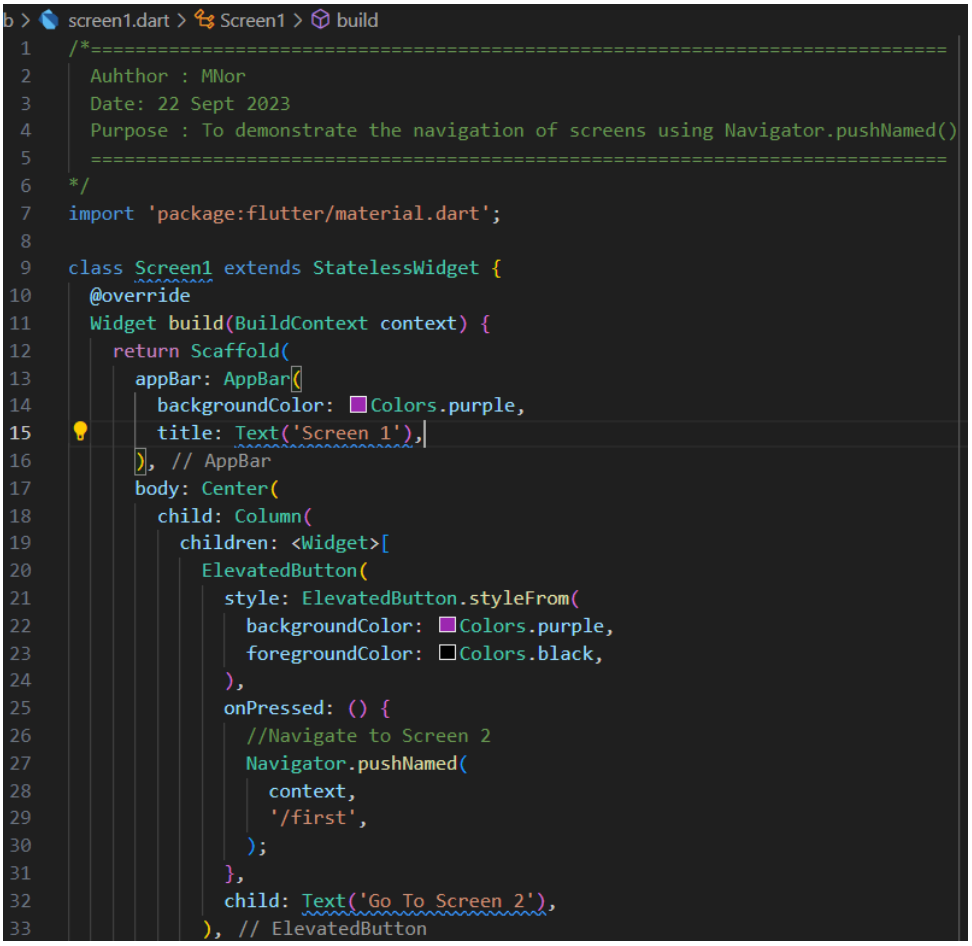


6. Go to *main.dart* file. Delete the existing source code.

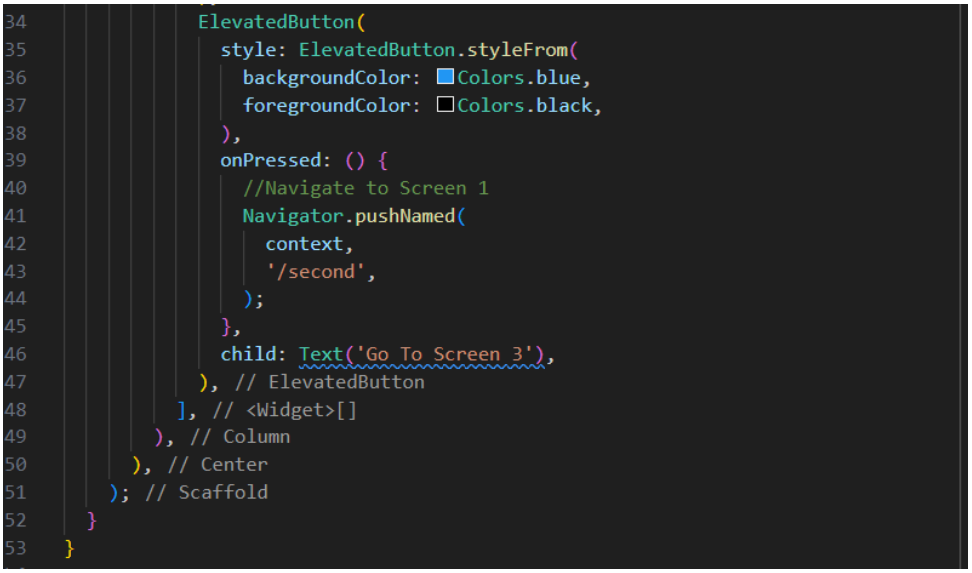


1.2 Creating the UI for screen 1

1. Go to *Lib* directory.
2. Create a file known as *screen1.dart* as a Stateless widget.
3. The construction of UI based on the following requirements:
   (a) Return the *Widget build(BuildContext context)* as a *Scaffold* widget.
   (b) Add the *AppBar* widget with the title as *'Screen 1'*.
   (c) For the *body* property of *Scaffold*, attach the *Center* widget.
   (d) Then, for *Center* widget, add *Column* widget.
   (e) Finally, attach two *ElevatedButton* that used to navigate to screen 2 and screen 3 respectively.
4. For the first *ElevatedButton*, implement the logic for the current UI to navigate to second screen through the *onPressed* property using *Navigator.pushNamed(context, '/second',);*.
5. For the second *ElevatedButton*, implement the logic for the current UI to navigate to second screen through the *onPressed* property using *Navigator.pushNamed(context, '/third',);*.
6. Rename the both *ElevatedButtons* as *'Go to Screen 2'* and *'Go to Screen 3'* respectively.



```dart
/*==========================================================================
  Auhthor : MNor
  Date: 22 Sept 2023
  Purpose : To demonstrate the navigation of screens using Navigator.pushNamed()
  ==========================================================================
*/
import 'package:flutter/material.dart';

class Screen1 extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.purple,
        title: Text('Screen 1'),
      ), // AppBar
      body: Center(
        child: Column(
          children: <Widget>[
            ElevatedButton(
              style: ElevatedButton.styleFrom(
                backgroundColor: Colors.purple,
                foregroundColor: Colors.black,
              ),
              onPressed: () {
                //Navigate to Screen 2
                Navigator.pushNamed(
                  context,
                  '/first',
                );
              },
              child: Text('Go To Screen 2'),
            ), // ElevatedButton
```



```dart
            ElevatedButton(
              style: ElevatedButton.styleFrom(
                backgroundColor: Colors.blue,
                foregroundColor: Colors.black,
              ),
              onPressed: () {
                //Navigate to Screen 1
                Navigator.pushNamed(
                  context,
                  '/second',
                );
              },
              child: Text('Go To Screen 3'),
            ), // ElevatedButton
          ], // <Widget>[]
        ), // Column
      ), // Center
    ); // Scaffold
  }
}
```

1.3 Creating the UI for screen 2

1. Go to *Lib* directory.

2. Create a file known as *screen2.dart* as a Stateless widget.

3. The construction of UI based on the following requirements:

   (a) Return the *Widget build(BuildContext context)* as a *Scaffold* widget.

   (b) Add the *AppBar* widget with the title as *'Screen 2'*.

   (c) For the *body* property of *Scaffold*, attach the *Center* widget.

   (d) Then, for *Center* widget, add *Column* widget.

   (e) Finally, attach one *ElevatedButton* that used to navigate to screen 3.

lib > screen2.dart > ...

```dart
/*============================================================================
  Auhthor : MNor
  Date: 22 Sept 2023
  Purpose : To demonstrate the navigation of screens using Navigator.pushNamed()
            and Navigator.push()
  ============================================================================
*/
import 'package:flutter/material.dart';
import 'screen3.dart';

class Screen2 extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.red,
        title: Text('Screen 2'),
      ), // AppBar
      body: Center(
        child: ElevatedButton(
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.red,
            foregroundColor: Colors.black,
          ),
          onPressed: () {
            //Navigate to Screen 3
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) {
                return Screen3();
              }), // MaterialPageRoute
            );
          },
          child: Text('Go To Screen 3'),
        ), // ElevatedButton
      ), // Center
    ); // Scaffold
  }
}
```

4. For the *ElevatedButton*, implement the logic for the current UI to navigate to second screen through the *onPressed* property using *Navigator.push();* with respective parameters.

5. Rename the *ElevatedButtons* as *'Go to Screen 3'*.

6. Save your source code.

1.4 Creating the UI for screen 3

1. Go to *Lib* directory.

2. Create a file known as *screen2.dart* as a Stateless widget.

```dart
/*================================================================================
  Auhthor : MNor
  Date: 22 Sept 2023
  Purpose : To demonstrate the navigation of screens using Navigator.pushNamed()
  ================================================================================
*/
import 'package:flutter/material.dart';

class Screen3 extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.blue,
        title: Text('Screen 3'),
      ), // AppBar
      body: Center(
        child: ElevatedButton(
          style: ElevatedButton.styleFrom(
            backgroundColor: Colors.blue,
            foregroundColor: Colors.black,
          ),
          onPressed: () {
            //Navigate to Screen 1
            Navigator.pushNamed(context, '/');
          },
          child: Text('Go Back To Screen 1'),
        ), // ElevatedButton
      ), // Center
    ); // Scaffold
  }
}
```

3. The construction of UI based on the following requirements:

   (a) Return the *Widget build(BuildContext context)* as a *Scaffold* widget.

   (b) Add the *AppBar* widget with the title as *'Screen 3'*.

   (c) For the *body* property of *Scaffold*, attach the *Center* widget.

(d) Then, for *Center* widget, add *Column* widget.

(e) Finally, attach one *ElevatedButton* that used to navigate to screen 1.

4. For the *ElevatedButton*, implement the logic for the current UI to navigate to second screen through the *onPressed* property using *Navigator.pushNamed(context, '/');*.

5. Rename the *ElevatedButtons* as *'Go Back To Screen 1'*.

6. Save your source code.

1.5 Updating the main program

1. Go to *main.dart* in your *Lib* directory.

2. Delete the code if there is the existing code there.

3. Perform importing on the related files used in this program.

4. Write a coding to run main application.

5. Create a class known as *MyApp* as a Stateless widget.

```
lib > main.dart > ...
1  /*============================================================================
2    Auhthor : MNor
3    Date: 22 Sept 2023
4    Purpose : To the initialRoute and routes for navigation of screens.
5    ============================================================================
6  */
7  import 'package:flutter/material.dart';
8  import 'screen1.dart';
9  import 'screen2.dart';
10 import 'screen3.dart';
11
   Run | Debug | Profile
12 void main() => runApp(MyApp());
13
14 class MyApp extends StatelessWidget {
15   @override
16   Widget build(BuildContext context) {
17     return MaterialApp(
18       initialRoute: '/',
19       routes: {
20         '/': (context) => Screen1(),
21         '/first': (context) => Screen2(),
22         '/second': (context) => Screen3(),
23       },
24     ); // MaterialApp
25   }
26 }
```

6. The construction of UI based on the following requirements:

   (a) Return the *Widget build(BuildContext context)* as a *MaterialApp* widget.

   (b) Define the property known as *initialRoute* and assign as a root directory ('/')

   (c) Define the property known as *routes*.

   (d) Assign the *routes* property with respective navigation of screen 1, 2 and 3. [**Note**: This is the part to define the router].

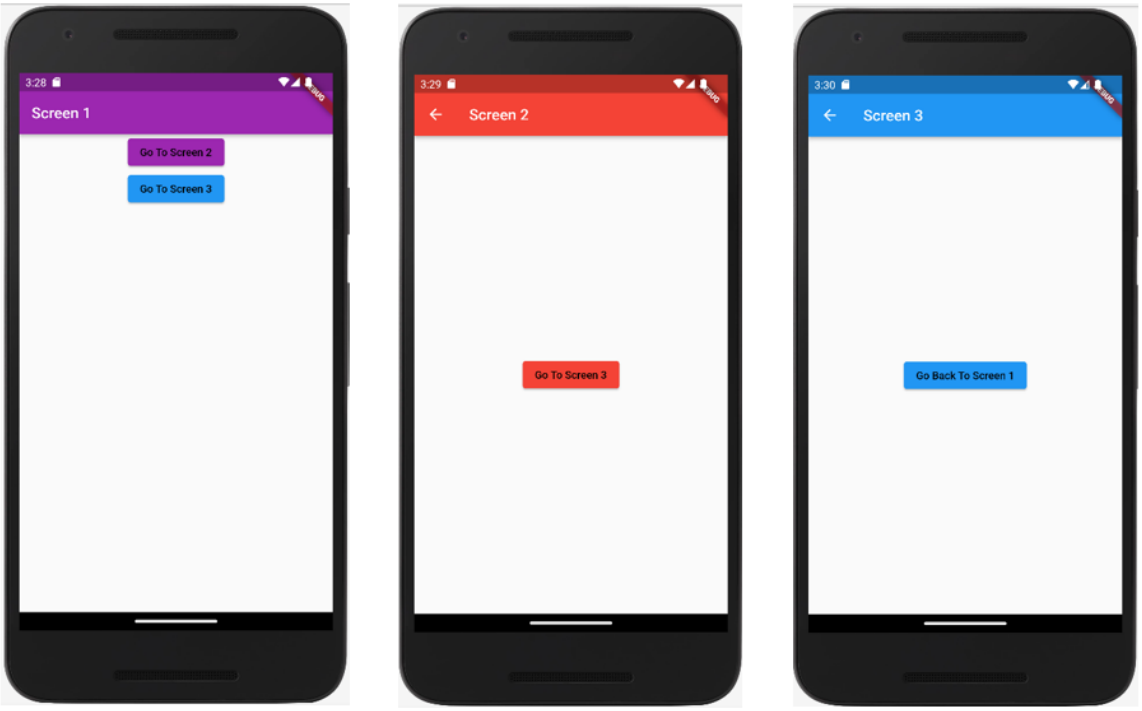
7. Review your code.

8. Save your source code.

9. Run your Android emulator.

10. Run your *main.dart*.

11. You should get the following output.



1.6 Exercise

1. Create 3 screens for data entry on e-Voting UI for the ABC Sdn Bhd.

2. The first screen used to display the options 1) Employee Details and 2) Vote the Candidate (Note: Implement using ElevatedButton].

3. The second screen used to display a menu for employee details which composed of staff number and department.

4. The third screen used to perform vote that display a candidate A, candidate B and candidate C in terms of radio button.

5. Screen 2 can navigate to first screen and third screen.

6. Screen 3 can navigate to to first screen and second screen.

7. Complete your coding.

8. Attach the source code and your output in the lab report.

9. Explain in details the differences between using *Navigator.pushNamed()* and *Navigator.push(()*.

2 Screens Navigation

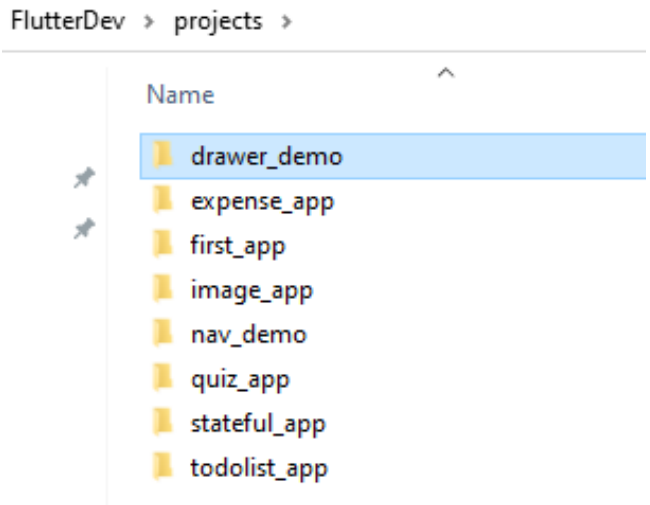
2.1 Implementing *Drawer* widget with the *ListView* and *ListTile* widget

.

1. Open your command prompt.
2. Go to your current Flutter's project folder. [**Note**: Your command to navigate to *FlutterDev*'s folder might be different if you use DOS command prompt..!]

```
MINGW64:/D/FlutterDev/projects

Owner@MNor-HPElite-G3 MINGW64 /D
$ pwd
/D

Owner@MNor-HPElite-G3 MINGW64 /D
$ cd FlutterDev/projects

Owner@MNor-HPElite-G3 MINGW64 /D/FlutterDev/projects
$ create flutter drawer_demo
```

3. Creating the Flutter's project known as *drawer_demo*.
4. Open your *FlutterDev*'s folder. You need to ensure the *drawer_demo* created.
5. Go to VS Code IDE. Open *drawer_demo*'s project folder.



6. Go to *main.dart* file. Delete the existing source code.

7. The requirements to create the UI as below:

   (a) Implement *MaterialApp* widget inside the *runApp()* method.

```dart
/*=============================================================================
  Auhthor       : MNor
  Prorgram Name : dart.main
  Date          : 22 Sept 2023
  Purpose       : To demonstrate the use fo Drawer for navigation of screens.
  ============================================================================
*/

import 'package:flutter/material.dart';

//void main() => runApp(MyApp());

Run | Debug | Profile
void main() {
  runApp(
    MaterialApp(home: MyApp()), // use MaterialApp
  );
}
```

   (b) Return the *Widget build(BuildContext context)* as a *Scaffold* widget.

   (c) Add the *AppBar* widget with the title as *'Demo using Drawer'*.

   (d) For the *body* property of *Scaffold*, attach the *Center* widget with the title as 'My Main Screen'.

   (e) For the *drawer* property of *Scaffold*, define the *Drawer* widget.

   (f) Inside the *Drawer* widget, define the *ListView* widget.

   (g) In the *ListView* widget, we can defined *DrawerHeader* widget and two *ListTile* widgets.

   (h) In order to get the proper size of *DrawerHeader* widget, wrap this widget inside the *SizedBox* widget and set the *height* property as 60.

   (i) For first *ListTile* widget, define the title as *'Product'*.

   (j) For secodn *ListTile* widget, define the title as *'Stock'*.

   (k) To close the the *ListView* widget, when user click either *'Product'* or *'Stock'*, we need to implement *Navigator.pop(context)* at the *onTap*'s property.

8. Before you continue, try to go to Flutter SDK API documentation and do a reading for *Drawer* widget. [**Note:** Refer to this URL https://api.flutter.dev/flutter/material/Drawer-class.html]

9. Proceed the coding for *Widget build(BuildContext context)* method.

10. Please ensure your code must start with the *MaterialApp* widget.

```
19  class MyApp extends StatelessWidget {
20    const MyApp({super.key});
21
22    Widget build(BuildContext context) {
23      return Scaffold(
24        appBar: AppBar(
25          title: Text('Demo using Drawer'),
26          centerTitle: true,
27        ), // AppBar
28        body: Center(
29          child: Text('My Main Screen'),
30        ), // Center
```

```
31        drawer: Drawer(
32          child: ListView(
33            children: [
34              const SizedBox(
35                height: 60.0,
36                child: const DrawerHeader(
37                  decoration: BoxDecoration(color: Colors.purple),
38                  child: Text('Drawer Header'),
39                ), // DrawerHeader
40              ), // SizedBox
41              ListTile(
42                title: const Text('Products'),
43                onTap: () {
44                  Navigator.pop(context);
45                },
46              ), // ListTile
47              ListTile(
48                title: const Text('Stock'),
49                onTap: () {
50                  Navigator.pop(context);
51                },
52              ), // ListTile
53            ],
54          ), // ListView
55        ), // Drawer
56      ); // Scaffold
57    }
58  }
```

11. Complete your coding.

12. Open your Android virtual device.

13. Save your code.

14. Finally, run your *main.dart* program.

15. You should perform a testing by tapping the *ListTile* represent *'Product'*.

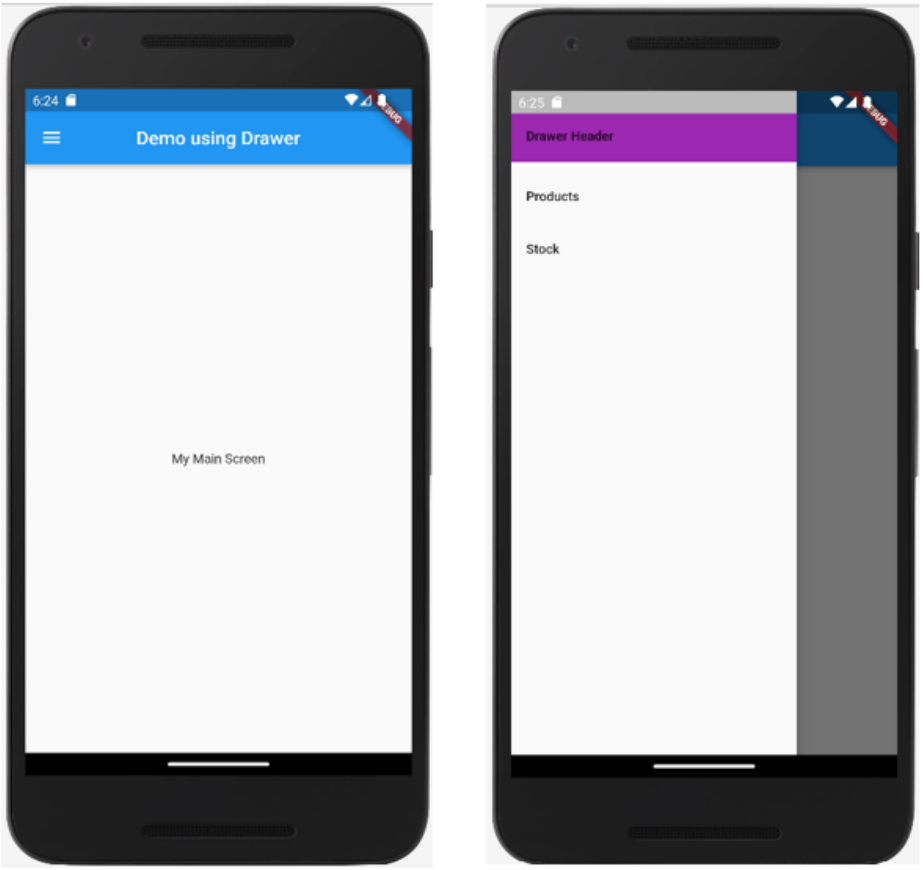
16. Repeat the process by tapping the *ListTile* representing *'Stock'*.

17. You need to ensure your *Navigator.pop()* is work properly.

18. You should get the following output.

19. Attach your code and the output in your lab report.

20. In this hand on session, the *ListTile* return to the previous screen via a calling og *Navigator.pop()*.



2.2 Navigate to the next screen by invoking a *Navigator.push()* method inside the *ListTile* widget

1. Open your existing source code written in part 2.1.

2. In order to navigate to the next screen, we need to replace an existing code *Navigator.pop()* with the *Navigator.push()* method.

3. Implement this modification for both *ListTile* widgets representing *'Product'* and *'Stock'*.

4. Re-route the next screen as *ProductScreen()* and *StockScreen()*.

```
41              ListTile(
42                  title: const Text('Products'),
43                  onTap: () {
44                    //  Navigator.pop(context);
45                    Navigator.push(
46                        context,
47                        MaterialPageRoute(
48                          builder: (context) => ProductScreen(),
49                        )); // MaterialPageRoute
50                  }), // ListTile
51              ListTile(
52                title: const Text('Stock'),
53                onTap: () {
54                  //  Navigator.pop(context);
55                  Navigator.push(
56                        context,
57                        MaterialPageRoute(
58                          builder: (context) => StockScreen(),
59                        )); // MaterialPageRoute
60                },
61              ), // ListTile
```

5. Write the coding for for both *ListTile* widgets. [**Note:** In your code, you will get the syntax error for *ProductScreen()* and *StockScreen()* since the details UI for these widgets are not be implemented yet.]

6. Next, create a *ProductScreen()* and *StockScreen()* widgets as a *StatelessWidget*.

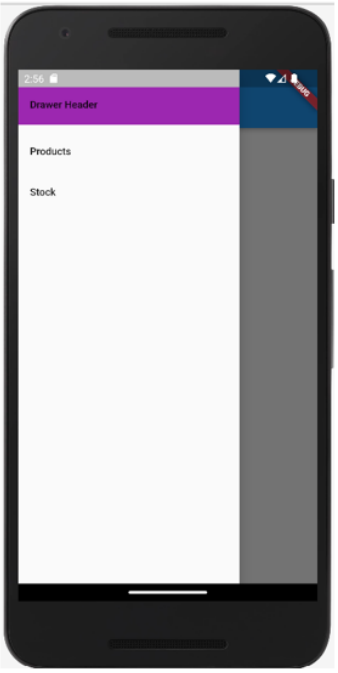
7. Inside the *ProductScreen()* widget, attach the *Scaffold* widget. Subsequently, include both *AppBar* with the title as *'Product Screen'* and *Center* widgets with title as *'Overview of Product Screen'*.

8. Repeat the same logic for *StockScreen()* widget and attach title as *'Stock Screen'* and *'Overview of Stock Screen'* respectively.

```dart
class ProductScreen extends StatelessWidget {
  const ProductScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Product Screen'),
        centerTitle: true,
      ), // AppBar
      body: const Center(
        child: Text('Overview of Product Screen'),
      ), // Center
    ); // Scaffold
  }
}

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Stock Screen'),
        centerTitle: true,
      ), // AppBar
      body: const Center(
        child: Text('Overview of Stock Screen'),
      ), // Center
    ); // Scaffold
  }
}
```

9. Complete the remaining of your code in the *main.dart*.
10. Once finishing writing the code, review your code to ensure it is free from any errors.
11. Save you file.
12. Open your Android emulator.
13. Run your program.
14. Test your output by tap to *Drawer*. Then tap to Products. You will see the screen will go to *Product* screen.
15. Repeat the same process by tapping to Stock. You will see the screen will go to *Stock* screen.
16. You should get the following outputs.
17. Attached your source code and the outputs in your lab report.


2:56
Demo using Drawer
My Main Screen


2:56
Drawer Header
Products
Stock


2:58
Product Screen
Overview of Product Screen


2:59
Stock Screen
Overview of Stock Screen

2.3 Exercise

1. Based on the coding in part 2.2, the two new widgets *ProductScreen()* and *Stock()* are created inside the *main.dart*.

2. Modify the existing code and create these two widgets in a separate dart file.

3. Finally, call these widgets in the *main.dart*.

4. Attach your source code and your outputs.

3 Passing Data to the Next Screen

3.1 Create New Project

1. Open your command prompt.

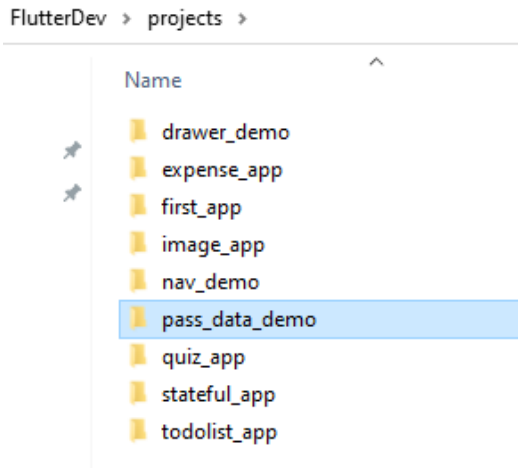
2. Go to your current Flutter's project folder. [**Note**: Your command to navigate to *FlutterDev*'s folder might be different if you use DOS command prompt..!]

3. Creating the Flutter's project known as *pass_data_demo*.

```
Owner@MNor-HPElite-G3 MINGW64 /D/FlutterDev/projects
$ flutter create pass_data_demo
Creating project pass_data_demo...
Resolving dependencies in pass_data_demo...
Got dependencies in pass_data_demo.
Wrote 129 files.

All done!
You can find general documentation for Flutter at: https://docs.flutter.dev/
Detailed API documentation is available at: https://api.flutter.dev/
If you prefer video documentation, consider: https://www.youtube.com/c/flutterde
v

In order to run your application, type:

  $ cd pass_data_demo
  $ flutter run

Your application code is in pass_data_demo\lib\main.dart.
```

4. Open your *FlutterDev*'s folder. You need to ensure the *pass_data_demo* created.

5. Go to VS Code IDE. Open *pass_data_demo*'s project folder.



6. Go to *main.dart* file. Delete the existing source code.

3.2 Define a Class *Product*

1. Create a file known as *product.dart* inside the *Lib*'s folder.

2. Define the attributes and constructor for *Product* class.

lib > product.dart > ...

```dart
/*==========================================================================
  Auhthor       : MNor
  Prorgram Name : product.dart
  Date          : 23 Sept 2023
  Purpose       : Create Product class.
  ==========================================================================
*/
class Product {
  final String code;
  final String description;

  const Product(this.code, this.description);
}
```

3. Save your coding.

3.3 Create a *ProductDetailsScreen* widget

1. Create a file known as *prod_detail_screen.dart* inside the *Lib*'s folder.

ib > prod_detail_screen.dart > ...

```dart
/*==========================================================================
  Auhthor       : MNor
  Prorgram Name : prod_detail_screen.dart
  Date          : 23 Sept 2023
  Purpose       : A user defined widget to display the product details.
  ==========================================================================
*/
import 'package:flutter/material.dart';
import 'product.dart';

class ProductDetailScreen extends StatelessWidget {
  final Product products;
  const ProductDetailScreen({Key? key, required this.products})
      : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(products.code),
      ), // AppBar
      body: Center(
        child: Text(products.description),
      ), // Center
    ); // Scaffold
  }
}
```

2. Define the constructor as *ProductDetailsScreen* and passing a list of product as a mandatory parameter. [**Note:** Key? : key indicate the parameter is mandatory]

3. Create a *build* method and return a *Scaffold* to display the product code at the *AppBar* widget and product description inside the *Center* widget.

4. Save your file once complete writing a code.

3.4 Create a *ProductScreen* widget via *main.dart*

1. Go to the *main.dart* file inside the *Lib*'s folder.

2. Define the related import files and define the details implementation of *void main()*.

main.dart > ...

```dart
/*================================================================================
  Auhthor       : MNor
  Prorgram Name : main.dart
  Date          : 23 Sept 2023
  Purpose       : To demonstrate passing the data to the next screen.
  ================================================================================
*/
import 'package:flutter/material.dart';
import 'product.dart';
import 'prod_detail_screen.dart';

List<Product> productList = [
  Product('A1001', 'Panasonic Aircond HP 1.5'),
  Product('A2023', 'Daikin Aircond HP 2.0'),
  Product('A1058', 'Panasonic Vacuum Cleaner'),
];

Run | Debug | Profile
void main() {
  runApp(
    MaterialApp(
        debugShowCheckedModeBanner: false,
        title: 'Inventory App',
        home: ProductScreen(products: productList)), // MaterialApp
  );
}
```

3. Create *ProductScreen* widget as a *StatelessWidget*.

4. Inside *ProductScreen* widget, define the variable for a list of *Product*.

5. Define the constructor for *ProductScreen* by passing product list as a parameter.

6. Create a *build* method return a *Scaffold* to display a list of product code at the *ListView.builder* and *ListTile* widget.

7. Inside the *onTap* property implement the logic to navigate to *Product DetailScreen* when user taps to specific product code.

8. Complete your coding.

```dart
class ProductScreen extends StatelessWidget {
  final List<Product> products;
  const ProductScreen({Key? key, required this.products}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('List of Products'),
        centerTitle: true,
      ), // AppBar
      body: ListView.builder(
          itemCount: products.length,
          itemBuilder: (context, index) {
            return ListTile(
              title: Text(products[index].code),
              onTap: () {
                Navigator.push(
                    context,
                    MaterialPageRoute(
                      builder: (context) =>
                          ProductDetailScreen(products: products[index]),
                    )); // MaterialPageRoute
              },
            ); // ListTile
          }), // ListView.builder
    ); // Scaffold
  }
}
```

9. Review your code and save it.

10. Open your Android emulator.

11. Finally, run your program.

12. Your will get the main screen representing a list of product code.

13. Try to tap to specific product code.

14. Verify whether your action will navigate to a *Product DetailScreen* based on the present product code your tap via the main screen.

15. Finally, tap the arrow icon at the left screen to return to a *ProductScreen* UI.

16. You should produce the following output.

17. Attach your source code and the output inside your lab report.

18. In your report explain what are the important points you learned from this hand on session.