Lab session 3: Multi-Screens Design using Flutter

Task-2 Mutli-screen application in Flutter using the routing and tabular navigation

Building a multiscreen app is a common requirement in mobile application development. By dividing the app's functionality into different screens, we can create a more organized and intuitive user experience. Flutter provides powerful tools and libraries to simplify the implementation of multiscreen navigation, including routing and tabular navigation.

Using routing allows us to define a navigation hierarchy and easily switch between screens using named routes. Tabular navigation, on the other hand, offers a convenient way to present multiple screens within the same view, typically using a tab bar or a bottom navigation bar.

Throughout this documentation, we will cover the step-by-step process of creating screens, setting up routing, and implementing tabular navigation in a Flutter app.

Prerequisite

For the purpose of this task, we will be using a provided Dart file that contains the basic structure and code for implementing a multiscreen app in Flutter. It is composed by:

- Homescreen.dart
- Main.dart
- Profile.dart
- Setting_screen.dart
- Tabular_navigation.dart

Project Setup

1. **Create a new Flutter project:**

   Open a new terminal or command prompt and run the following command in the desired directory to create a new Flutter project:

```
flutter create multiscreen_apk
```

This command will create a new directory named `multiscreen_app` with the basic structure of a Flutter project.

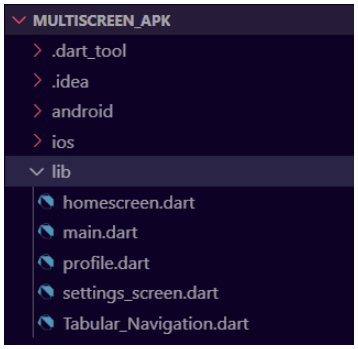
2. **Move the provided Dart file**:

   Locate the provided Dart file for implementing the multiscreen app.

   Copy or move these files into the lib folder of the newly created Flutter project. The lib folder is where your app's main code resides.

3. **Replace the main.dart file**:

   By default, Flutter creates a main.dart file in the lib folder. We will replace the contents of this file with the provided main.dart file that we moved in the previous step. This Dart file should contain the necessary code structure and boilerplate for implementing the multiscreen app.



Creating Tabular Navigation File

The tabular navigation file is responsible for implementing tab-based navigation between different screens within your app. This allows users to switch between screens by tapping on the corresponding tabs.

Code Overview

The tabular navigation file consists of a TabularNavigationApp widget that serves as the root widget for the app. It uses the TabBar and TabBarView widgets from the Flutter framework to handle the tab navigation.

```dart
import 'package:flutter/material.dart';
import 'homescreen.dart';
import 'profile.dart';
import 'settings_screen.dart';

Run | Debug | Profile
void main() {
  runApp(const MyApp());
}

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

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: TabularNavigationApp(),
    ); // MaterialApp
  }
}

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

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

class _TabularNavigationAppState extends State<TabularNavigationApp>
    with SingleTickerProviderStateMixin {
  final List<Widget> _tabs = [
    const Tab(text: 'Home'),
    const Tab(text: 'Profile'),
    const Tab(text: 'Settings'),
  ];
```

```dart
final List<Widget> _tabViews = [
  const HomeScreen(),
  const ProfileScreen(),
  const SettingsScreen(),
];

late TabController _tabController;

@override
void initState() {
  super.initState();
  _tabController = TabController(length: _tabs.length, vsync: this);
}

@override
void dispose() {
  _tabController.dispose();
  super.dispose();
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('Tabular Navigation App'),
      bottom: TabBar(
        controller: _tabController,
        tabs: _tabs,
        onTap: (index) {
          switch (index) {
            case 0:
              Navigator.pushNamed(context, '/home');
              break;
            case 1:
              Navigator.pushNamed(context, '/profile');
              break;
            case 2:
              Navigator.pushNamed(context, '/settings');
              break;
          }
        },
      ), // TabBar
    ), // AppBar
    body: TabBarView(
      controller: _tabController,
      children: _tabViews,
    ), // TabBarView
  ); // Scaffold
}
}
```

Customizing Tabs and Tab Content

To customize the tabs and their associated content, we can modify the _tabs and _tabViews lists within the _TabularNavigationAppState class.

_tabs: This list contains the Tab widgets that represent each tab in the tab bar. You can customize the text or appearance of each tab by modifying the text property of each Tab widget.

_tabViews: This list contains the widgets that correspond to the content of each tab. You can replace the HomeScreen(), ProfileScreen(), and SettingsScreen() widgets with your own screen widgets.

Make sure that the number of tabs and tab views match. Each tab should have a corresponding widget in the tab views list.

Modifying the Main File for Routing and Tabular Navigation

The main file of your Flutter app serves as the entry point and is responsible for initializing the app and defining the routes for navigation. In this section, we will modify the main file to integrate routing and tabular navigation.

Step 1: Import Dependencies

Make sure to import the necessary dependencies at the beginning of your main file:

lib > main.dart > ...

```dart
import 'package:flutter/material.dart';
import 'homescreen.dart';
import 'profile.dart';
import 'settings_screen.dart';
import 'tabular_navigation.dart';
```

Step 2: Define Routes

In the main file, define the routes using the MaterialApp widget. Each route corresponds to a specific screen in the app.

lib > main.dart > MyApp > build

```dart

class MyApp extends StatelessWidget {
  const MyApp({super.key});
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Multi-Screen App',
      initialRoute: '/',
      routes: {
        '/': (context) => TabularNavigationApp(),
        '/home': (context) => const HomeScreen(),
        '/profile': (context) => const ProfileScreen(),
        '/settings': (context) => const SettingsScreen(),
      },
    ); // MaterialApp
  }
}
```

The root route '/' is associated with the TabularNavigationApp widget, which provides the tabular navigation UI. The other routes, such as '/home', '/profile', and '/settings', are associated with the corresponding screen widgets.

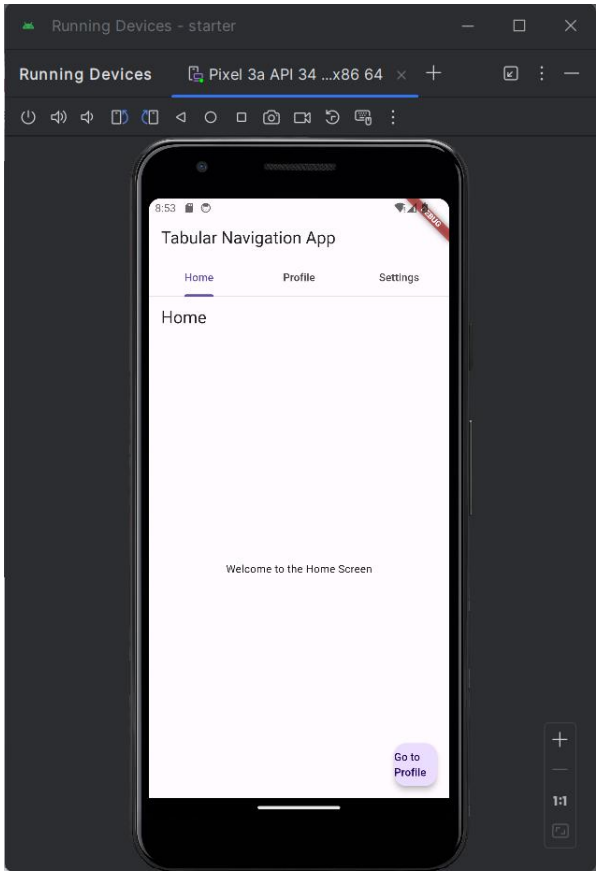
Step 3: Implement Navigation in TabularNavigationApp

In the TabularNavigationApp widget, update the TabBar widget to handle navigation to the desired screens when a tab is tapped.

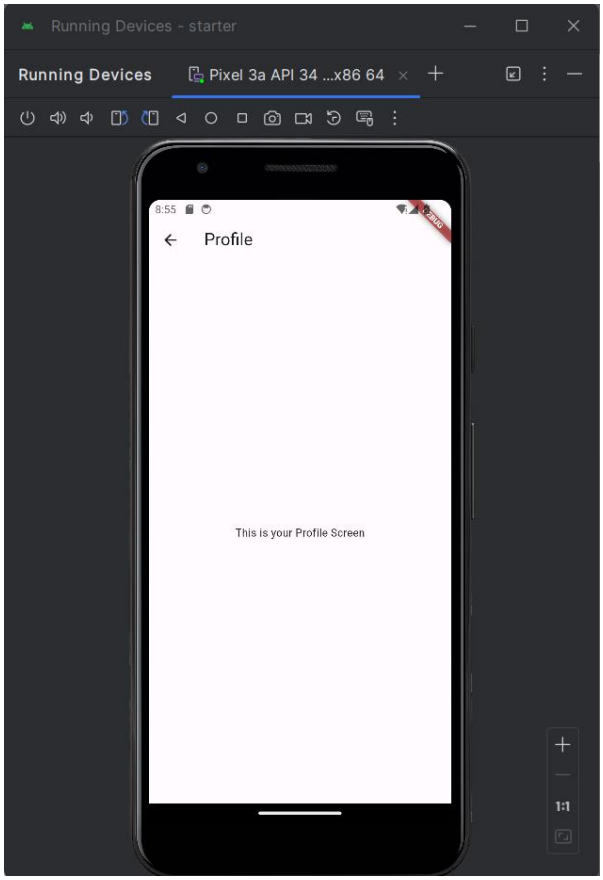
By using Navigator.pushNamed, we can navigate to the desired screen by specifying the route name. In this case, we navigate to '/home', '/profile', or '/settings' based on the selected tab index.

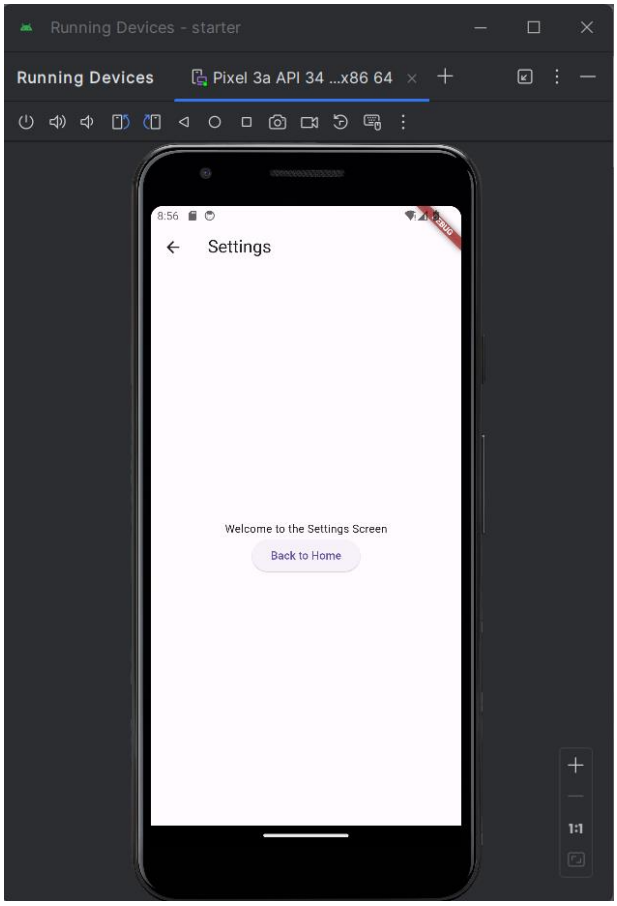
Testings

The Home screen is the first screen that appers after launching the app.



We can see the tabular navigation on the top of the application taking us successfully to each screen.





Conclusion

In this documentation, we learned how to modify the main file of a Flutter app to incorporate routing and tabular navigation. By defining routes and utilizing the TabBar widget, we created a seamless navigation experience for users, allowing them to switch between screens effortlessly.

Armed with this knowledge, you are now equipped to create engaging and user-friendly Flutter apps with tabular navigation.