

**SEB-312 Mobile Application Development**

**LAB # 7**

**LAB Title**

|  |
| --- |
| Implementing a simple Drawer Navigation, TabBar Navigation and navigation between screens. Building a login page and Form validation using TextFormField. |

**Assessment of CLO: 03, PLO: 05**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name:** |  | | |
| **Roll No.** |  | | |
| **Semester** |  | **Session** |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S. No.** | **Perf. Level**  **Criteria** | **Excellent**  **(2.5)** | **Good**  **(2)** | **Satisfactory**  **(1.5)** | **Needs Improvement**  **(0 ~ 1)** | **Marks Obtained** |
| **1** | Project Execution & Implementation | Fully functional, optimized, and well-structured. | Minor errors, mostly functional. | Some errors, requires guidance. | Major errors, non-functional, or not Performed. |  |
| **2** | Results & Debugging  Or Troubleshooting | Accurate results with effective debugging  Or Troubleshooting. | Mostly correct, some debugging Or Troubleshooting needed. | Partial results, minimal debugging  Or Troubleshooting. | Incorrect results, no debugging Or Troubleshooting, or not attempted. |  |
| **3** | Problem-Solving & Adaptability  (VIVA) | Creative approach, efficiently solves challenges. | Adapts well, minor struggles. | Some adaptability, needs guidance. | Lacks innovation or no innovation, unable to solve problems. |  |
| **4** | Report Quality & Documentation | Clear, structured, with detailed visuals. | Mostly clear, minor gaps. | Some clarity issues, missing details. | Poorly structured, lacks clarity, or not submitted. |  |
| **Total Marks Obtained Out of 10** | | | | | |  |

**Experiment evaluated by**

|  |  |  |  |
| --- | --- | --- | --- |
| **Instructor’s Name** | **Sidra Khatoon** | | |
| **Date** |  | **Signature** |  |

**Objective**

The objective of lab is to implementing a simple Drawer Navigation, TabBar Navigation and navigation between screens. Also building a login page and Form validation using TextFormField.

**Instructions**

You have to perform the following tasks yourselves. Raise your hand if you face any difficulty in understanding and solving these tasks. **Plagiarism** is an abhorrent practice and you should not engage in it.

**Navigation Drawer:**

The mobile apps that use Material Design have two primary options for navigation. These navigations are Tabs and Drawers. A drawer is an alternative option for tabs because sometimes the mobile apps do not have sufficient space to support tabs.

A drawer is an invisible side screen. It is a sliding left menu that generally contains important links in the application and occupies half of the screen when displayed.

The navigation drawer in Flutter allows users to navigate to different pages of your app. The navigation drawer is added using the [Drawer](https://youtu.be/WRj86iHihgY) widget. It can be opened via swipe gesture or by clicking on the menu icon in the app bar.

**When to use a navigation drawer**

The navigation drawer can be used as an alternate option to the TabBar widget. It is recommended to use a navigation drawer when you have at least five pages to navigate. If your app has several pages, providing the navigation inside the TabBar makes for a less intuitive user experience.

**How to add a basic navigation drawer in Flutter**

To add a basic navigation drawer in Flutter, you must first use MaterialApp in your project. Then, the Drawer widget can be added to the Scaffold widget.

Here are the step-by-step instructions:

1. Make sure you are using the MaterialApp
2. Inside the Scaffold, add the Drawer property and assign the Drawer widget
3. Inside the Drawer widget, add the ListView as a child widget
4. Inside the ListView, add the DrawerHeader widget. This will create a material design drawer header
5. Inside the DrawerHeader, add a Text widget with some text
6. Below the DrawerHeader, add a ListTile widget with an icon and title that represents a single page
7. Similarly add other ListTile for other pages

We add drawer navigation code in first screen class after app bar and before body.

drawer: Drawer(

      child: ListView(

        // Important: Remove any padding from the ListView.

        padding: EdgeInsets.zero,

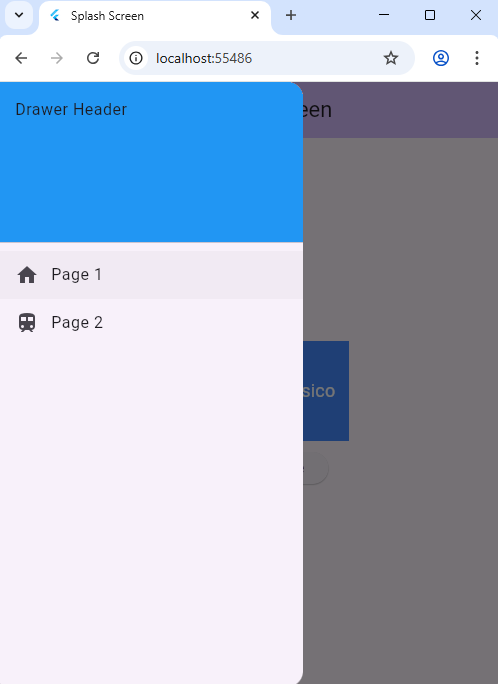
        children: [

          const DrawerHeader(

            decoration: BoxDecoration(

              color: Colors.blue,

            ),

            child: Text('Drawer Header'),

          ),

          ListTile(

            leading: Icon(

              Icons.home,

            ),

            title: const Text('Page 1'),

            onTap: () {

              Navigator.pop(context);

            },

          ),

          ListTile(

            leading: Icon(

              Icons.train,

            ),

            title: const Text('Page 2'),

            onTap: () {

              Navigator.pop(context);

            },

          ),

        ],

      ),

    ),

**Displaying user details in navigation drawer header**

The basic example above shows the drawer header with simple text, but in a real-world scenario, you may want to display the current user information inside the drawer header. You can do that with the help of a ready-made widget called UserAccountsDrawerHeader.

The UserAccountsDrawerHeader is used to display all user-related information, such as profile picture, username, and email. You can also open the user details page when users tap on the user information.

To display user details in the navigation drawer:

1. In the example code above, just replace the DrawerHeader widget with the UserAccountsDrawerHeader
2. Add the accountName parameter and set the username
3. Add the accountEmail parameter and set the user email
4. Add the currentAccountPicture parameter and set the logged user’s profile picture

We can replace our previous code of drawer navigation with this

drawer: Drawer(

        child: ListView(

          padding: const EdgeInsets.all(0),

          children: [

             UserAccountsDrawerHeader(

                accountName: Text("Sidra",style: TextStyle(fontSize: 18),),

                accountEmail: Text("sidra@gmail.com", style: TextStyle(fontSize: 18),),

                currentAccountPictureSize: Size.square(50),

                currentAccountPicture: CircleAvatar(

                  child: Text("S",style: TextStyle(fontSize: 30,color: Colors.black),),

                ),

                ),

              ListTile(

                 leading: Icon(

                 Icons.home,

                ),

                  title: const Text('Page 1'),

                onTap: () {

                Navigator.pop(context);

                 },

              ),

              ListTile(

                leading: Icon(

                  Icons.train,

                ),

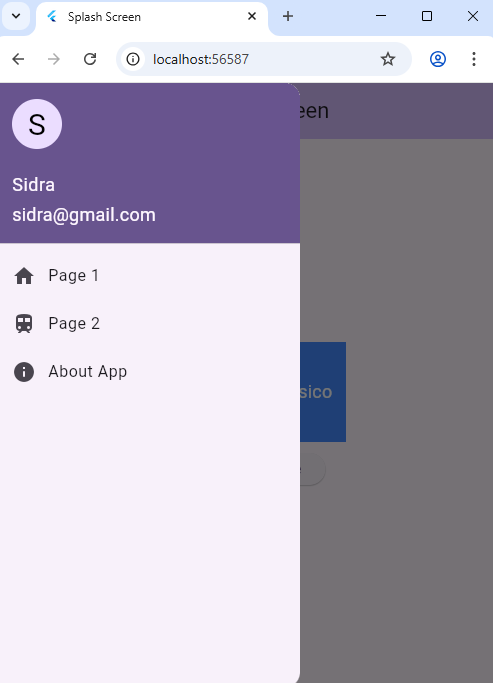
                title: const Text('Page 2'),

                onTap: () {

                  Navigator.pop(context);

                },

              ),

              ListTile(

                leading: Icon(

                  Icons.info\_rounded,

                ),

                title: const Text('About App'),

                onTap: () {

                  Navigator.pop(context);

                },

              ),

          ],

        ),

      ),

**Tab Navigation**

The tabs are mainly used for mobile navigation. Tab are placed on bottom and Top as per designer requirement. Working with tabs is a common pattern in Android and iOS apps that follow the Material Design guidelines. Flutter provides a convenient way to create a tab layout. To add tabs to the app, we need to create a TabBar and TabBarView and attach them with the TabController. The controller will sync both so that we can have the behavior which we need.

To implement TabBar in your Flutter app, complete the following steps:

1. Wrap the Scaffold widget inside the DefaultTabController in code first screen which we previously updated. This should be used for most simple use cases.
2. Place the TabBar widget as the bottom property of AppBar
3. Provide TabBarView in the body of the AppBar. TabBarView is like PageView, which is used mostly with TabBar because it shows the widget based on the currently selected tab

class FirstScreen extends StatelessWidget {

  const FirstScreen({super.key});

  @override

  Widget build(BuildContext context) {

    return DefaultTabController(

      length: 2,

      child: Scaffold(

        appBar: AppBar(

          backgroundColor: Theme.of(context).colorScheme.inversePrimary,

          bottom: const TabBar(

            tabs: [

              Tab(icon: Icon(Icons.home)),

              Tab(icon: Icon(Icons.home\_max\_rounded)),

            ],

          ),

          title: const Center(child: Text("FirstScreen")),

        ),

        drawer: Drawer(

          child: ListView(

            padding: const EdgeInsets.all(0),

            children: [

              const UserAccountsDrawerHeader(

                accountName: Text("Sidra", style: TextStyle(fontSize: 18)),

                accountEmail: Text("sidra@gmail.com", style: TextStyle(fontSize: 18)),

                currentAccountPictureSize: Size.square(50),

                currentAccountPicture: CircleAvatar(

                  child: Text("S", style: TextStyle(fontSize: 30, color: Colors.black)),

                ),

              ),

              ListTile(

                leading: const Icon(Icons.home),

                title: const Text('Page 1'),

                onTap: () {

                  Navigator.pop(context);

                },

              ),

              ListTile(

                leading: const Icon(Icons.train),

                title: const Text('Page 2'),

                onTap: () {

                  Navigator.pop(context);

                },

              ),

              ListTile(

                leading: const Icon(Icons.info\_rounded),

                title: const Text('About App'),

                onTap: () {

                  Navigator.pop(context);

                },

              ),

            ],

          ),

        ),

        body: const TabBarView(

          children: [

            ScreenOne(),

            ScreenTwo(),

          ],

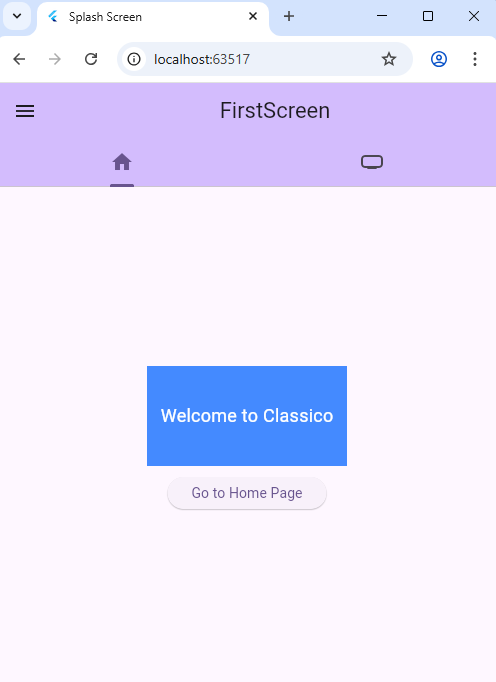
        ),

      ),

    );

  }

}



Now we can create two new screens, ScreenOne and ScreenTwo, which will be navigated from the TabBar.

class ScreenOne extends StatelessWidget {

  const ScreenOne({super.key});

  @override

  Widget build(BuildContext context) {

    return Center(

      child: Column(

        mainAxisAlignment: MainAxisAlignment.center,

        children: [

          Container(

            height: 100,

            width: 200,

            color: Colors.blueAccent,

            child: const Center(

              child: Padding(

                padding: EdgeInsets.all(8.0),

                child: Text(

                  "Welcome to Classico",

                  style: TextStyle(

                      fontSize: 18,

                      fontWeight: FontWeight.w400,

                      color: Colors.white),

                ),

              ),

            ),

          ),

          const SizedBox(height: 11),

          ElevatedButton(

              onPressed: () {

                Navigator.push(

                  context,

                  MaterialPageRoute(builder: (context) => ScreenTwo()),

                );

              },

              child: const Text("Go to Home Page"))

        ],

      ),

    );

  }

}

class  ScreenTwo extends StatelessWidget {

  const  ScreenTwo({super.key});

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      body: Container(

        color: Colors.cyan,

        child: Center(

            child: Container(

          height: 100,

          width: 200,

          color: Colors.blue,

          child: Center(

            child: Text(

              'Welcome to My home Page',

              style: TextStyle(

                  fontSize: 12,

                  fontWeight: FontWeight.bold,

                  color: Colors.white),

            ),

          ),

        )),

      ),

    );

  }

}

**Login Page:**

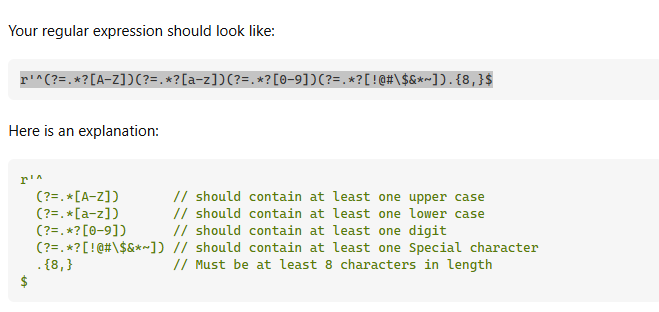
It’s a good practice to authenticate users before giving access to websites, mobile apps, computer applications to prevent unauthorized access to personal information. In this lab we learn how to build a login page user interface. We have used form widget of flutter to create login page. We have used the TextFormField widget, for user input as username and password. Elevated Button widget, to show action.

**Form Widget:**

The Form widget is container for form field in flutter. It helps in managing form state, validation and submission. Its act as a parent widget for form filed and provides methods and properties to handle form-related operations. In the provided code, the form widget is used as the parent of from fields.

**Textfield Validation: Form widget GlobalKey<FormState>**

The key property of the form widget is used to assign a unique identifier to the forms. It is necessary to associate the form with a GlobalKey<FormState> so that we can access and manipulate the form’s including form validation and submission.



**Implementation of Login Page:**

**Step 1:** Create flutter new project with basic counter application. Remove the counter related code and comments. Now MyHomePage code look like this.

class MyHomePage extends StatelessWidget {

  const MyHomePage({super.key});

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: Text("Home Page"),

        centerTitle: true,

        backgroundColor: Colors.cyan,

      ),

      body: Center(

        child: Container(

          height: 200,

          width: 200,

          color: Colors.cyan,

          child: Center(child: Text("Welcome to Home Screen")),

        ),

      ),

    );

  }

}

**Step** **2:** We create new class for named it LoginPage extend with stateful widget. In MyApp class change the code in home widget with LoginScreen(). So our first screen is login page.

class LoginScreen extends StatefulWidget{

  const LoginScreen({super.key});

  @override

  State<LoginScreen> createState() => \_LoginScreenState();

}

class \_LoginScreenState extends State<LoginScreen> {

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        backgroundColor: Theme.of(context).colorScheme.inversePrimary,

        title: Text("Login Screen"),

        centerTitle: true,

      ),

      body: Column(),

    );

  }

}

**Step 3:** Now we build UI for login Screen which consist on 2 TextFormField and 1 button. So, we need column widget to align all widget vertically.

class LoginScreen extends StatefulWidget{

  const LoginScreen({super.key});

  @override

  State<LoginScreen> createState() => \_LoginScreenState();

}

class \_LoginScreenState extends State<LoginScreen> {

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        backgroundColor: Theme.of(context).colorScheme.inversePrimary,

        title: Text("Login Screen"),

        centerTitle: true,

      ),

      body: Column(

        mainAxisAlignment: MainAxisAlignment.center,

        crossAxisAlignment: CrossAxisAlignment.center,

        children: [

          Padding(

            padding: const EdgeInsets.all(8.0),

            child: TextFormField(

              decoration: InputDecoration(

                labelText: "Email",

                hintText: "Enter Email",

                prefixIcon: Icon(Icons.email),

                border: OutlineInputBorder()

              ),

            ),

          ),

          SizedBox(

            height: 20,

          ),

          Padding(

            padding: const EdgeInsets.all(8.0),

            child: TextFormField(

              decoration: InputDecoration(

                labelText: "Password",

                hintText: "Enter Password",

                prefixIcon: Icon(Icons.lock),

                border: OutlineInputBorder()

              ),

            ),

          ),

          SizedBox(

            height: 20,

          ),

          ElevatedButton(

            onPressed: (){},

            child: Text("Login"))

        ],

      ),

    );

  }

}

**Step 4:** Now we wrap the column widget with form widget. Form widget is used to validate the email and password.

 final GlobalKey<FormState> \_formkey = GlobalKey<FormState>();

Add above code in LoginScreenState class to create variable for formkey. Code after add **form** widget in LoginScreenState.

class \_LoginScreenState extends State<LoginScreen> {

  final GlobalKey<FormState> \_formkey = GlobalKey<FormState>();

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        backgroundColor: Theme.of(context).colorScheme.inversePrimary,

        title: Text("Login Screen"),

        centerTitle: true,

      ),

      body: Form(

        key: \_formkey,

        child: Column(

          mainAxisAlignment: MainAxisAlignment.center,

          crossAxisAlignment: CrossAxisAlignment.center,

          children: [

            Padding(

              padding: const EdgeInsets.all(8.0),

              child: TextField(

                decoration: InputDecoration(

                  labelText: "Email",

                  hintText: "Enter Email",

                  prefixIcon: Icon(Icons.email),

                  border: OutlineInputBorder()

                ),

              ),

            ),

            SizedBox(

              height: 20,

            ),

            Padding(

              padding: const EdgeInsets.all(8.0),

              child: TextField(

                decoration: InputDecoration(

                  labelText: "Password",

                  hintText: "Enter Password",

                  prefixIcon: Icon(Icons.lock),

                  border: OutlineInputBorder()

                ),

              ),

            ),

            SizedBox(

              height: 20,

            ),

            ElevatedButton(

              onPressed: (){},

              child: Text("Login"))

          ],

        ),

      ),

    );

  }

}

**Step 5:** Create two texteditingcontroller to input text from textfield. Add this code in LoginScreenState class.

final emailC = TextEditingController();

final passC = TextEditingController();

Now add controller into the both textfield.

 TextField(

                controller: emailC,

                decoration: InputDecoration(

                  labelText: "Email",

                  hintText: "Enter Email",

                  prefixIcon: Icon(Icons.email),

                  border: OutlineInputBorder()

                ),

              ),

TextField(

                controller: passC,

                decoration: InputDecoration(

                  labelText: "Password",

                  hintText: "Enter Password",

                  prefixIcon: Icon(Icons.lock),

                  border: OutlineInputBorder()

                ),

              ),

**Step 6:** We create two methods for validating email and password

 String? \_validateEmail(value) {

    if (value!.isEmpty) {

      return 'Enter email';

    }

    RegExp emailRegExp = RegExp(

        r"^[a-zA-Z0-9.a-zA-Z0-9.!#$%&'\*+-/=?^\_`{|}~]+@[a-zA-Z0-9]+\.[a-zA-Z]+");

    if (!emailRegExp.hasMatch(value)) {

      return 'Enter valid email';

    }

    return null;

  }

  String? \_validatepassword(value) {

    if (value!.isEmpty) {

      return 'Enter email';

    }

    RegExp passwordRegExp =

        RegExp(r'^(?=.\*?[A-Z])(?=.\*?[a-z])(?=.\*?[0-9])(?=.\*?[!@#\$&\*~]).{8,}$');

    if (!passwordRegExp.hasMatch(value)) {

      return 'Enter valid password';

    }

    return null;

  }

**Step 7:** Now, we call these methods into TextFormField widget to validate the inputs.

TextFormField(

                controller: emailC,

                autovalidateMode: AutovalidateMode.onUserInteraction,

                decoration: InputDecoration(

                  labelText: "Email",

                  hintText: "Enter Email",

                  prefixIcon: Icon(Icons.email),

                  border: OutlineInputBorder()

                ),

                validator: \_validateEmail

              ),

TextFormField(

                controller: passC,

                 autovalidateMode: AutovalidateMode.onUserInteraction,

                decoration: InputDecoration(

                  labelText: "Password",

                  hintText: "Enter Password",

                  prefixIcon: Icon(Icons.lock),

                  border: OutlineInputBorder()

                ),

                validator: \_validatepassword

              ),

**Step 8:** Now are working on Tap function of elevated button. \_submitform() is method which navigate to next screen. We call this method in on pressed function of button.

void \_submitform() {

    if (\_formkey.currentState!.validate()) {

      Navigator.pushReplacement(

          context, MaterialPageRoute(builder: (context) => MyHomePage()));

    }

  }

ElevatedButton(

              onPressed: (){

                \_submitform;

              },

              child: Text("Login"))

Step 9: We know that in every app password is hidden. So we hide password in our code.

TextFormField(

                    controller: passC,

                    autovalidateMode: AutovalidateMode.onUserInteraction,

                    obscureText: true,

                    obscuringCharacter: "\*",

                    decoration: InputDecoration(

                      labelText: "Password",

                      hintText: "Enter Password",

                      prefixIcon: Icon(Icons.lock),

                      border: OutlineInputBorder()

                    ),

                    validator: \_validatepassword

                  ),

**Assessment 1: Drawer Navigation in Flutter**

**Objective:**  
Create a Flutter application that uses a **Drawer** to navigate between multiple screens. This task tests your understanding of UI layout, navigation using Navigator, and proper widget structuring in Flutter.

**Instructions:**

1. **Main Screen (HomeScreen)**:
   * Use a Scaffold with an AppBar and a Drawer.
   * The main content should show a welcome message (e.g., "Welcome to Home Screen").
2. **Drawer Menu Items**: Add the following items inside the Drawer:
   * **Home** → Navigates to HomeScreen
   * **Screen One** → Navigates to ScreenOne
   * **Screen Two** → Navigates to ScreenTwo
   * **About App** → Navigates to AboutScreen
3. **Each screen should**:
   * Be a separate StatelessWidget or StatefulWidget.
   * Use Scaffold with an AppBar showing the screen name.
   * Display a Text widget in the center (e.g., "This is Screen One").

**Requirements:**

* Use Navigator.push or Navigator.pushReplacement for screen transitions.
* Add UserAccountsDrawerHeader at the top of the Drawer with dummy name and email.
* Each drawer item should close the drawer before navigating to a new screen.

**Assessment 2: SignUp Screen or Registration Screen**

Create a Signup or registration page with different Textfield. Form consists on Name, Email, phone number and password. On register button you will navigate to the homepage. Apply validation on all fields by using regex expression.