EXPERIMENT NO. 5
MAD and PWA Lab

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

Theory:

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, my app can have a screen that contains a login button. When the user taps on that button, immediately it will display the home screen of the app.

Flutter has an imperative routing mechanism, the Navigator widget, and a more idiomatic declarative routing mechanism (which is similar to build methods as used with widgets), the Router widget. The two systems can be used together (indeed, the declarative system is built using the imperative system). Typically, small applications are served well by just using the Navigator API, via the MaterialApp constructor's MaterialApp.routes property.

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.

1. First, you need to create two routes.
2. Then, navigate to one route from another route by using the Navigator.push() method.
3. Finally, navigate to the first route by using the Navigator.pop() method.

Create two routes

Here, we are going to create two routes for navigation. In both routes, we have created only a single button. When we tap the button on the Login Screen, it will navigate to the Home Screen. Again, when we tap the button on the Home Screen, it will return to the Login Screen.

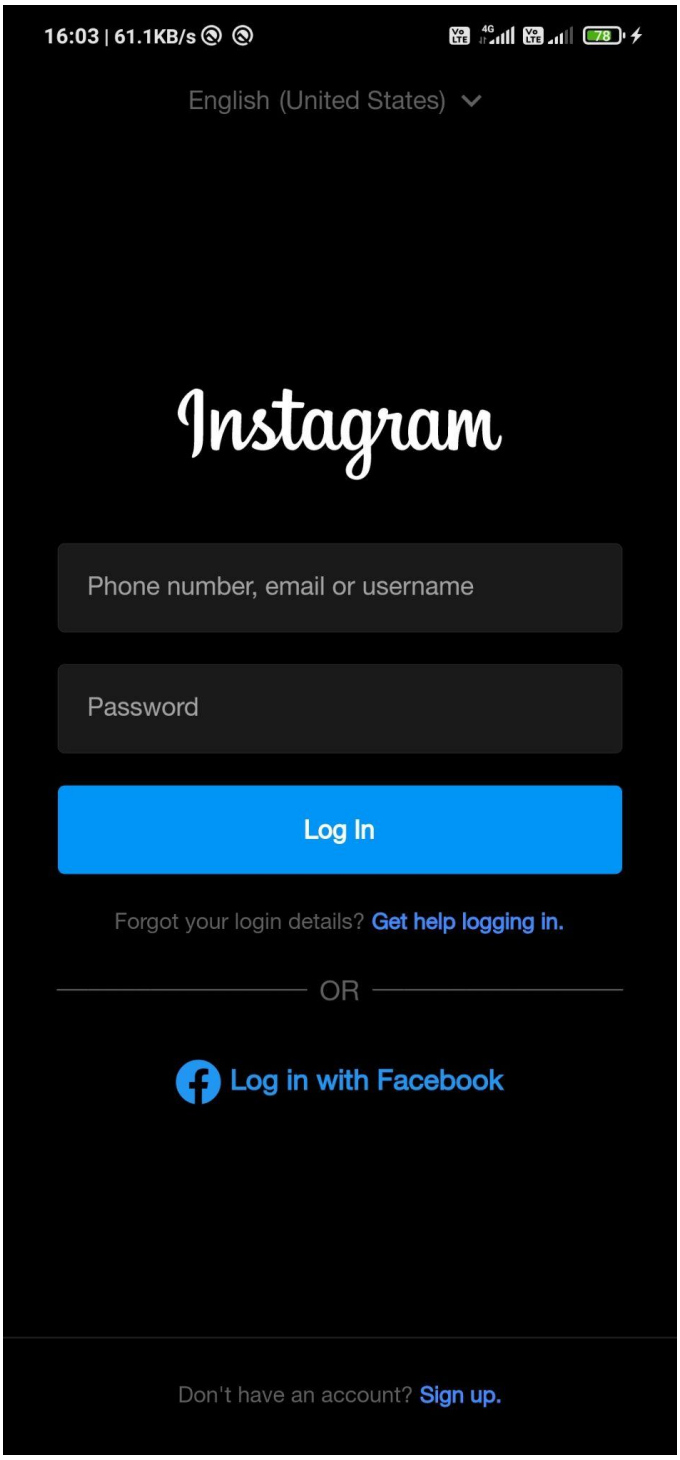
The **Navigator.push()** method is used to navigate/switch to a new route/page/screen. Here, the push() method adds a page/route on the stack and then manages it by using the Navigator. Again we use the MaterialPageRoute class that allows transition between the routes using a platform-specific animation.

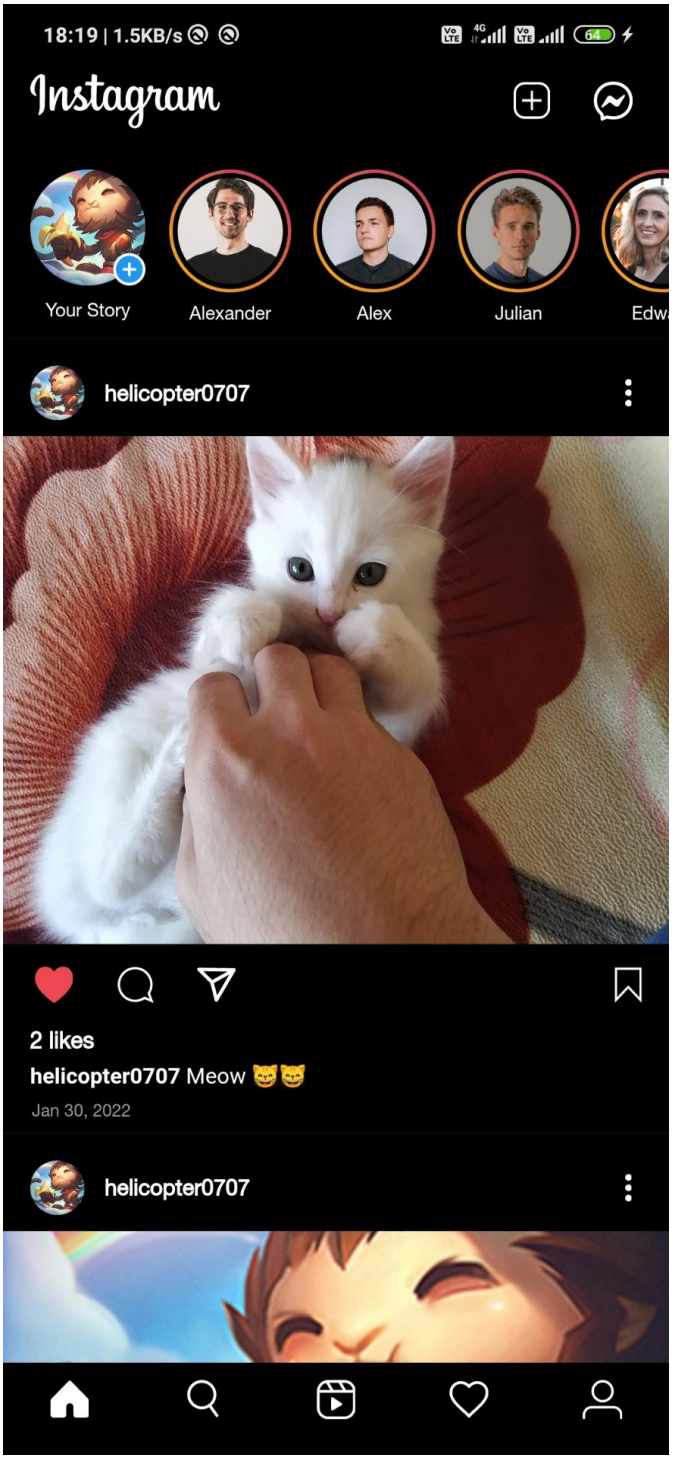
```
onPressed: () {
      Navigator.push(
            context,
            MaterialPageRoute(
                  builder: (context) => HomeScreen()),
      );
},
```

Now, we need to use the **Navigator.pop()** method to close the second route and return to the first route. The pop() method allows us to remove the current route from the stack, which is managed by the Navigator. To implement a return to the original route, we need to update the onPressed() callback method in the HomeScreen route widget.

```
onPressed: () {
      Navigator.pop(
            context,
            MaterialPageRoute(
                  builder: (context) => LoginScreen()),
      );
},
```

So, on clicking the **Log In** button in the Login Screen, we redirect to the Home Screen. And after clicking on the **Sign Out** button in the Home Screen, we go back to the Login Screen.


16:03 | 61.1KB/s
English (United States)
Instagram
Phone number, email or username
Password
Log In
Forgot your login details? Get help logging in.
OR
Log in with Facebook
Don't have an account? Sign up.


18:19 | 1.5KB/s
Instagram
Your Story
Alexander
Alex
Julian
helicopter0707
2 likes
helicopter0707 Meow
Jan 30, 2022
helicopter0707

Navigation with Named Routes

We can work with named routes by using the **Navigator.pushNamed()** function. This function takes two required arguments (build context and string) and one optional argument. Also, we know about the MaterialPageRoute, which is responsible for page transition. If we do not use this, then it is difficult to change the page.

The following steps are necessary, which demonstrate how to use named routes.

1. **First, we need to create two screens**

Here, we are going to create two routes for navigation. In both routes, we have created only a single button. When we tap the button on the Login Screen, it will navigate to the Home Screen. Again, when we tap the button on the Home Screen, it will return to the Login Screen.

2. **Define the routes**

In this step, we have to define the routes. The MaterialApp constructor is responsible for defining the initial route and other routes themselves. Here the initial route tells the start of the page and routes property defines the available named routes and widgets.

```
MaterialApp(
        debugShowCheckedModeBanner: false,
        title: 'Instagram Clone',
        theme: ThemeData(
          fontFamily: 'Helvetica',
          brightness: Brightness.dark,
          scaffoldBackgroundColor: mobileBackgroundColor,
        ),
        initialRoute: '/',
        routes: {
          // When navigating to the "/" route, build the Login Screen widget.
          '/': (context) => LoginScreen(),
          // When navigating to the "/home" route, build the Home Screen widget.
          '/home': (context) => LoginScreen(),
        },
),
```

3. **Navigate to the second screen using the Navigator.pushNamed() function**

In this step, we need to call the Navigator.pushNamed() method for navigation. For this, we need to update an onPressed() callback in the build method of Login Screen like below code snippets.

```
onPressed: () {
      Navigator.pushNamed(
            context,
            '/home',
      );
},
```

4. **Use a Navigator.pop() function to return to the first screen**

It is the final step, where we will use the Navigator.pop() method to return to the Login Screen from Home Screen.

```
onPressed: () {
      Navigator.pop(
            context,
      );
},
```

The output is the same as above.

Gesture Detector

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. Some of the examples of gestures are:

1. When the mobile screen is locked, you slide your finger across the screen to unlock it
2. Tapping a button on your mobile screen
3. Tapping and holding an app icon on a touch-based device to drag it across screens

Flutter provides a widget that gives excellent support for all types of gestures by using the **GestureDetector** widget. The GestureWidget is non-visual widgets, which is primarily used for detecting the user's gesture. The basic idea of the gesture detector is a stateless widget that contains parameters in its constructor for different touch events.

The GestureDetector widget decides which gesture is going to recognize based on which of its callbacks are non-null. Let us learn how we can use these gestures in our application with a simple onTap() event and determine how the GestureDetector processes this. I have used the GestureDetector widget for Login Button and Sign Up button and many more.

1. Login Button

```
Container(
      padding: const EdgeInsets.symmetric(
            horizontal: 32,
      ),
      child: GestureDetector(
            onTap: loginUser,
            child: Container(
            child: _isLoading
            ? const Center(
                  child: CircularProgressIndicator(
                        color: primaryColor,
                  ),
            )
            : const Text(
                  'Log In',
                  style: TextStyle(
                        fontSize: 16.0,
                        fontWeight: FontWeight.bold,
                  ),
            ),
            width: double.infinity,
            alignment: Alignment.center,
            padding: const EdgeInsets.symmetric(
                  vertical: 17,
            ),
            decoration: const ShapeDecoration(
                  shape: RoundedRectangleBorder(
                        borderRadius: BorderRadius.all(
                              Radius.circular(4),
                        ),
                  ),
                  color: blueColor),
```

```
            ),
      ),
),
```

2. Asking for sign up text button

```
GestureDetector(
      onTap: () {
            Navigator.push(
                  context,
                  MaterialPageRoute(
                        builder: (context) => PhoneOrEmail()),
                  );
            },
      child: Container(
            child: const Text(
                  "Sign up.",
                  style: TextStyle(
                        fontWeight: FontWeight.bold,
                        color: Colors.blueAccent,
                  ),
            ),
            padding: const EdgeInsets.symmetric(
                  vertical: 8,
            ),
      ),
),
```

Conclusion: Hence, we understood how to apply navigation, routing and gestures in the Login Screen, Sign Up screen and Home Screen of my Flutter Application.