UIT UNIVERSITY

CSC-318 – Mobile Application Development

LAB 3: Navigation and Routing in Flutter App – I

Name : _____

Roll No : _____

Section : _____

Semester : _____

FALL 2024

COMPUTER SCIENCE DEPARTMENT

LAB-3 | Navigation and Routing in Flutter App – I

i. **Navigate to a new screen and back**
ii. **Send data to a new screen**

---

1. Navigate to a new screen and back:

Most apps contain several screens for displaying different types of information. For example, an app might have a screen that displays products. When the user taps the image of a product, a new screen displays details about the product.
In Flutter, screens and pages are called routes. In Android, a route is equivalent to an Activity. In iOS, a route is equivalent to ViewController. In Flutter, a route is just a widget.

2. Navigator class

A widget that manages a set of child widgets with a stack discipline.
Many apps have a navigator near the top of their widget hierarchy in order to display their logical history using an Overlay with the most recently visited pages visually on top of the older pages. Using this pattern lets the navigator visually transition from one page to another by moving the widgets around in the overlay. Similarly, the navigator can be used to show a dialog by positioning the dialog widget above the current page.

3. Steps to navigate between two routes:

i. Create two routes.
ii. Navigate to the second route using Navigator.push().
iii. Return to the first route using Navigator.pop().

3.1 Create two routes

First, create two routes to work with. Since this is a basic example, each route contains only a single button. Tapping the button on the first route navigates to the second route. Tapping the button on the second route returns to the first route.

First, set up the visual structure:

class FirstRoute:

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('First Route'),
      ),
      body: Center(
        child: ElevatedButton(
          child: const Text('Open route'),
```

```
          onPressed: () {
            // Navigate to second route when tapped.
          },
        ),
      ),
    );
  }
}
```

class SecondRoute:

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Second Route'),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            // Navigate back to first route when tapped.
          },
          child: const Text('Go back!'),
        ),
      ),
    );
  }
}
```

3.2 Navigate to the second route using Navigator.push()

To switch to a new route, use the Navigator.push() method. The push() method adds a Route to the stack of routes managed by the Navigator. Where does the Route come from? You can create your own, or use a MaterialPageRoute, which is useful because it transitions to the new route using a platform-specific animation.
In the build() method of the FirstRoute widget, update the onPressed() callback:

```
// Within the `FirstRoute` widget:
onPressed: () {
  Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => const SecondRoute()),
  );
   }
```

3.3 Return to the first route using Navigator.pop()

How do you close the second route and return to the first? By using the Navigator.pop() method. The pop() method removes the current Route from the stack of routes managed by the Navigator.
To implement a return to the original route, update the onPressed() callback in the SecondRoute widget:

```
// Within the SecondRoute widget
onPressed: () {
  Navigator.pop(context);
}
```

4. Interactive example-I

```
import 'package:flutter/material.dart';

void main() {
  runApp(const MaterialApp(
    title: 'Navigation Basics',
    home: FirstRoute(),
  ));
}
class FirstRoute extends StatelessWidget {
  const FirstRoute({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('First Route'),
      ),
      body: Center(
        child: ElevatedButton(
          child: const Text('Open route'),
          onPressed: () {
            Navigator.push(
              context,
              MaterialPageRoute(builder: (context) => const SecondRoute()),
            );
          },
        ),
      ),
    );
  }
}

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

```
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Second Route'),
      ),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.pop(context);
          },
          child: const Text('Go back!'),
        ),
      ),
    );
  }
}
```

5. Send data to a new screen

Often, you not only want to navigate to a new screen, but also pass data to the screen as well. For example, you might want to pass information about the item that's been tapped.
Remember: Screens are just widgets. In this example, create a list of todos. When a todo is tapped, navigate to a new screen (widget) that displays information about the todo. Consider the following steps:

1. Define a todo class.
2. Create a List of todos.
3. Display a list of todos.
4. Create a Todo screen to display the list
5. Create a detail screen that can display information about a todo.
6. Navigate and pass data to the detail screen.

5.1 Define a todo class

First, you need a simple way to represent todos. For this example, create a class that contains two pieces of data: the title and description.

```
class Todo {
  final String title;
  final String description;

  const Todo(this.title, this.description);
   }
```

5.2 Create a list of todos

Second, display a list of todos. In this example, generate 20 todos and show them using a ListView.

Generate the list of todos:

```
final todos = List.generate(
  20,
  (i) => Todo(
    'Todo $i',
    'A description of what needs to be done for Todo $i',
  ),
);
```

5.3 Display the list of todos using a ListView

```
ListView.builder(
  itemCount: todos.length,
  itemBuilder: (context, index) {
    return ListTile(
      title: Text(todos[index].title),
    );
  },
)
```

5.4 Create a Todo screen to display the list

For this, we create a StatelessWidget. We call it TodosScreen. Since the contents of this page won't change during runtime, we'll have to require the list of todos within the scope of this widget.
We pass in our ListView.builder as body of the widget we're returning to build(). This'll render the list on to the screen for you to get going!

```
class TodosScreen extends StatelessWidget {
  // Requiring the list of todos.
  const TodosScreen({super.key, required this.todos});

  final List<Todo> todos;

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Todos'),
      ),
      //passing in the ListView.builder
      body: ListView.builder(
        itemCount: todos.length,
        itemBuilder: (context, index) {
          return ListTile(
            title: Text(todos[index].title),
          );
```

```
        },
      ),
    );
  }
}
```

5.5 Create a detail screen to display information about a todo

Now, create the second screen. The title of the screen contains the title of the todo, and the body of the screen shows the description. Since the detail screen is a normal StatelessWidget, require the user to enter a Todo in the UI. Then, build the UI using the given todo.

```
class DetailScreen extends StatelessWidget {
  // In the constructor, require a Todo.
  const DetailScreen({super.key, required this.todo});

  // Declare a field that holds the Todo.
  final Todo todo;

  @override
  Widget build(BuildContext context) {
    // Use the Todo to create the UI.
    return Scaffold(
      appBar: AppBar(
        title: Text(todo.title),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Text(todo.description),
      ),
    );
  }
}
```

5.6 Navigate and pass data to the detail screen

With a DetailScreen in place, you're ready to perform the Navigation. In this example, navigate to the DetailScreen when a user taps a todo in the list. Pass the todo to the DetailScreen.
To capture the user's tap in the TodosScreen, write an onTap() callback for the ListTile widget. Within the onTap() callback, use the Navigator.push() method.

```
body: ListView.builder(
  itemCount: todos.length,
  itemBuilder: (context, index) {
    return ListTile(
      title: Text(todos[index].title),
      // When a user taps the ListTile, navigate to the DetailScreen.
      // Notice that you're not only creating a DetailScreen, you're
      // also passing the current todo through to it.
```

```
      onTap: () {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => DetailScreen(todo: todos[index]),
          ),
        );
      },
    );
  },
),
```

6. Interactive example-II

```
import 'package:flutter/material.dart';

class Todo {
  final String title;
  final String description;

  const Todo(this.title, this.description);
}

void main() {
  runApp(
    MaterialApp(
      title: 'Passing Data',
      home: TodosScreen(
        todos: List.generate(
          20,
          (i) => Todo(
            'Todo $i',
            'A description of what needs to be done for Todo $i',
          ),
        ),
      ),
    ),
  );
}

class TodosScreen extends StatelessWidget {
  const TodosScreen({super.key, required this.todos});

  final List<Todo> todos;

  @override
  Widget build(BuildContext context) {
```

```
    return Scaffold(
      appBar: AppBar(
        title: const Text('Todos'),
      ),
      body: ListView.builder(
        itemCount: todos.length,
        itemBuilder: (context, index) {
          return ListTile(
            title: Text(todos[index].title),
            // When a user taps the ListTile, navigate to the DetailScreen.
            // Notice that you're not only creating a DetailScreen, you're
            // also passing the current todo through to it.
            onTap: () {
              Navigator.push(
                context,
                MaterialPageRoute(
                  builder: (context) => DetailScreen(todo: todos[index]),
                ),
              );
            },
          );
        },
      ),
    );
  }
}

class DetailScreen extends StatelessWidget {
  // In the constructor, require a Todo.
  const DetailScreen({super.key, required this.todo});

  // Declare a field that holds the Todo.
  final Todo todo;

  @override
  Widget build(BuildContext context) {
    // Use the Todo to create the UI.
    return Scaffold(
      appBar: AppBar(
        title: Text(todo.title),
      ),
      body: Padding(
        padding: const EdgeInsets.all(16),
        child: Text(todo.description),
      ),
    );
  }
}
```

Exercises:

1- Develop a simple flutter app with multiple screens. The home screen should include buttons to navigate to the 'About,' 'Contact,' and 'Settings' screens.

2- Design a contact list screen that displays names. It should navigate to a detail screen with additional information; passing the contact's details, when a user taps a contact.

3- Create a flutter app that includes a user input screen where users can fill out a form with fields for their name, email address, and a message or feedback. After submitting the form, the app should navigate to a second screen that showcases the entered information.