UIT UNIVERSITY

CSC-318 – Mobile Application Development

LAB 5: Route and Navigation II

Name : _____

Roll No : _____

Section : _____

Semester : _____

FALL 2024

COMPUTER SCIENCE DEPARTMENT

LAB-5 | Routing and Navigation -II

Return data from a screen

In some cases, you might want to return data from a new screen. For example, say you push a new screen that presents two options to a user. When the user taps an option, you want to inform the first screen of the user's selection so that it can act on that information.

You can do this with the Navigator.pop() method using the following steps:

Define the home screen

1. Add a button that launches the selection screen
2. Show the selection screen with two buttons
3. When a button is tapped, close the selection screen
4. Show a snackbar on the home screen with the selection

1. Define the home screen

#

The home screen displays a button. When tapped, it launches the selection screen.

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Returning Data Demo'),
      ),
      // Create the SelectionButton widget in the next step.
      body: const Center(
        child: SelectionButton(),
      ),
    );
  }
}
```

2. Add a button that launches the selection screen

#

Now, create the SelectionButton, which does the following:

Launches the SelectionScreen when it's tapped.
Waits for the SelectionScreen to return a result.

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

```
  @override
  State<SelectionButton> createState() => _SelectionButtonState();
}

class _SelectionButtonState extends State<SelectionButton> {
  @override
  Widget build(BuildContext context) {
    return ElevatedButton(
      onPressed: () {
        _navigateAndDisplaySelection(context);
      },
      child: const Text('Pick an option, any option!'),
    );
  }

  Future<void> _navigateAndDisplaySelection(BuildContext context) async {
    // Navigator.push returns a Future that completes after calling
    // Navigator.pop on the Selection Screen.
    final result = await Navigator.push(
      context,
      // Create the SelectionScreen in the next step.
      MaterialPageRoute(builder: (context) => const SelectionScreen()),
    );
  }
}
```

3. Show the selection screen with two buttons

#

Now, build a selection screen that contains two buttons. When a user taps a button, that app closes the selection screen and lets the home screen know which button was tapped.

This step defines the UI. The next step adds code to return data.

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Pick an option'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Padding(
```

```
            padding: const EdgeInsets.all(8),
            child: ElevatedButton(
              onPressed: () {
                // Pop here with "Yep"...
              },
              child: const Text('Yep!'),
            ),
          ),
          Padding(
            padding: const EdgeInsets.all(8),
            child: ElevatedButton(
              onPressed: () {
                // Pop here with "Nope"...
              },
              child: const Text('Nope.'),
            ),
          )
        ],
      ),
    ),
  );
 }
}
```

4. When a button is tapped, close the selection screen

#

Now, update the onPressed() callback for both of the buttons. To return data to the first screen, use the Navigator.pop() method, which accepts an optional second argument called result. Any result is returned to the Future in the SelectionButton.

Yep button

#

```
ElevatedButton(
  onPressed: () {
    // Close the screen and return "Yep!" as the result.
    Navigator.pop(context, 'Yep!');
  },
  child: const Text('Yep!'),
)
```

Nope button

#

```
ElevatedButton(
  onPressed: () {
    // Close the screen and return "Nope." as the result.
    Navigator.pop(context, 'Nope.');
  },
  child: const Text('Nope.'),
```

)

5. Show a snackbar on the home screen with the selection
#
Now that you're launching a selection screen and awaiting the result, you'll want to do something with the information that's returned.

In this case, show a snackbar displaying the result by using the _navigateAndDisplaySelection() method in SelectionButton:

```
// A method that launches the SelectionScreen and awaits the result from
// Navigator.pop.
Future<void> _navigateAndDisplaySelection(BuildContext context) async {
  // Navigator.push returns a Future that completes after calling
  // Navigator.pop on the Selection Screen.
  final result = await Navigator.push(
    context,
    MaterialPageRoute(builder: (context) => const SelectionScreen()),
  );

  // When a BuildContext is used from a StatefulWidget, the mounted property
  // must be checked after an asynchronous gap.
  if (!context.mounted) return;

  // After the Selection Screen returns a result, hide any previous snackbars
  // and show the new result.
  ScaffoldMessenger.of(context)
    ..removeCurrentSnackBar()
    ..showSnackBar(SnackBar(content: Text('$result')));
}
```

Interactive example
#
```
import 'package:flutter/material.dart';

void main() {
  runApp(
    const MaterialApp(
      title: 'Returning Data',
      home: HomeScreen(),
    ),
  );
}

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

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

```
    return Scaffold(
      appBar: AppBar(
        title: const Text('Returning Data Demo'),
      ),
      body: const Center(
        child: SelectionButton(),
      ),
    );
  }
}

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

  @override
  State<SelectionButton> createState() => _SelectionButtonState();
}

class _SelectionButtonState extends State<SelectionButton> {
  @override
  Widget build(BuildContext context) {
    return ElevatedButton(
      onPressed: () {
        _navigateAndDisplaySelection(context);
      },
      child: const Text('Pick an option, any option!'),
    );
  }

  // A method that launches the SelectionScreen and awaits the result from
  // Navigator.pop.
  Future<void> _navigateAndDisplaySelection(BuildContext context) async {
    // Navigator.push returns a Future that completes after calling
    // Navigator.pop on the Selection Screen.
    final result = await Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => const SelectionScreen()),
    );

    // When a BuildContext is used from a StatefulWidget, the mounted property
    // must be checked after an asynchronous gap.
    if (!context.mounted) return;

    // After the Selection Screen returns a result, hide any previous snackbars
    // and show the new result.
    ScaffoldMessenger.of(context)
      ..removeCurrentSnackBar()
      ..showSnackBar(SnackBar(content: Text('$result')));
```

```
  }
}

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Pick an option'),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Padding(
              padding: const EdgeInsets.all(8),
              child: ElevatedButton(
                onPressed: () {
                  // Close the screen and return "Yep!" as the result.
                  Navigator.pop(context, 'Yep!');
                },
                child: const Text('Yep!'),
              ),
            ),
            Padding(
              padding: const EdgeInsets.all(8),
              child: ElevatedButton(
                onPressed: () {
                  // Close the screen and return "Nope." as the result.
                  Navigator.pop(context, 'Nope.');
                },
                child: const Text('Nope.'),
              ),
            )
          ],
        ),
      ),
    );
  }
}
```

Tasks:

Task 1: Simple To-Do List Application

Create a to-do list application where the user can add tasks. Implement a screen to add a new task with two options: "Add" and "Cancel." When "Add" is pressed, return the new task to the main screen and display it in a list. If "Cancel" is pressed, return to the main screen without adding a task.

Task 2: Color Picker Application
Develop a color picker application where the user can select a color from a selection screen. The selection screen should have multiple color buttons (e.g., Red, Green, Blue). When a color is selected, return the chosen color name to the home screen and display it as the background color of the main screen. Also, show a snackbar with the selected color name.

Task 3: Quiz Application
Build a simple quiz application where the user can answer multiple-choice questions. On the question screen, present one question with multiple answer options (e.g., A, B, C, D). When the user selects an answer, return the selected option to the main screen and display it in a snackbar. Also, show the correct answer for reference.

Task: Create a Meal Selection Application
Objective: Develop a complete application that allows users to select a meal option from a list of available meals and display the selection on the home screen.

Requirements:
Home Screen:

Create a home screen with a title "Meal Selection App."
Include a button labeled "Select a Meal" that navigates to the meal selection screen.
Meal Selection Screen:

Design a meal selection screen with a title "Choose Your Meal."
Present three meal options as buttons:
"Pizza"
"Burger"
"Sushi"
When a button is pressed, return the selected meal name to the home screen using Navigator.pop().
Ensure that the screen has a back button to return without making a selection.

Displaying the Selection:

On the home screen, after returning from the meal selection screen, display a snackbar showing the selected meal.
If no meal is selected, display a snackbar saying "No meal selected."

UI Enhancements:

Use appropriate padding, margins, and styling for buttons to enhance the user experience.
Consider using different colors or icons for each meal option to make the selection visually appealing.

Functionality Testing:

Ensure that the navigation works smoothly between the two screens.
Test the application to confirm that the selected meal appears correctly in the snackbar.

Bonus:
Add an additional feature that allows users to long-press on a meal button to show a brief description of that meal in a tooltip or modal.

Instructions for Students:

- Implement the application according to the requirements.
- Test the navigation and data return functionality.
- Enhance the UI as desired.
- Optional: Add any extra features you think would improve the app.