07. Routes

Managing multiple pages

- Routes
- Tab
  - bottombar1.dart
  - default_tab_controller1.dart
- Navigator
  - navigator1.dart
  - navigator2.dart - Giving Arguments
- Route
  - route1.dart
  - route2.dart - Giving Arguments
  - Workflow

Routes

- Flutter supports `Routes` for managing multiple pages.
  - Tab
  - Navigator
  - Route

Tab

- Using the Tab is the simplest way to manage multiple pages.

bottombar1.dart

```
class Home1 extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: Text(

class Home2 extends StatelessWidget {
  ...
```

- We have two StatelessWidgets for each tab.

```
onTap: (i) {
  setState(() {tab = i;});
},
...
body: [Home1(), Home2()][tab],
```

- We use a variable `tab` to choose among multiple widgets.

```
bottomNavigationBar: BottomNavigationBar(
  currentIndex: tab,
  onTap: (i) { ... }, // user selection
  items: [ // display button
    BottomNavigationBarItem(...),
    BottomNavigationBarItem(...)
  ],
```

- We can use BottomNavigationBar and BottomNavigationBarItem widgets as buttons to choose the widget to display.

```
BottomNavigationBarItem(
  icon: Icon(Icons.home_outlined),
  label: 'Home1',
  backgroundColor: Colors.red),
```

- In this example, we display an icon and a label.

default_tab_controller1.dart

- For more detailed control, we can use DefaultTabController.
- It has three components to function as a tab controller.
  - `TabBar` to host Tab widgets.
  - `Tab` widgets.
  - `TabBarView` will host widgets that match the `Tab`.

```
return DefaultTabController(
  length: 3,
    child: Scaffold(
      appBar: AppBar(
        title: ...
        bottom: const TabBar( // 1
          tabs: <Widget>[
            Tab(...), ...     // 2
          ],),)
    // Body of DefaultTabController
      body: TabBarView(.      // 3
        children: <Widget>[
        ...
        ],),),),),
```

- This code snippet shows the

AppBar: TabBar/Tab

```
appBar: AppBar(
  title: const Text('Tab'),
  bottom: const TabBar(
    tabs: <Widget>[Tab(), Tab(), tab()]
```

- AppBar has two items:
  - Title
  - A TabBar to host three Tabs.

```
bottom: const TabBar(
  tabs: <Widget>[
    Tab(icon: Icon(Icons.tag_faces)),
    Tab(text: 'Menu2'),
    Tab(icon: Icon(Icons.info), text: 'Menu3'),
  ],
),
```

- The Tab can display an icon or text.

body: TabBarView

```
body: TabBarView(
  children: <Widget>[
    Home1(),
    Home2(),
    Container(color: Colors.red),
  ]
),
```

- When users click the Tab widget, the matching widget in the TabBarView is displayed.

Navigator

```
Navigator.of(context).pop();
```

Here's a rewritten version for clarity and conciseness:

- We previously explored how the `Navigator` stack operates in the context of Dialogs.
- In this section, calling `pop`

navigator1.dart

```
class PageA extends StatelessWidget {
  const PageA({super.key});
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Page A')),
      body: Column(
```

- The PageA widget is a simple Scaffold stateless widget.

- Navigator widget uses the context to switch between widgets.
  - We use Navigator.push to switch from the previous widget to the PageA widget.
  - When we return, we use Navigator.pop with the pushed context.

```
child: TextButton(
  child: Text('Click to Show Other Page'),
    onPressed: () {
      Navigator.push(context,
        MaterialPageRoute(
          builder: (c) {
            return PageA();
          }
  ...
```

- When users click the TextButton, the PageA() page is displayed.

```
Text('Page A'),
IconButton(
  onPressed: () {
    Navigator.pop(context);
  },
  icon: Icon(Icons.close)),
```

- In the PageA widget, when users click the `Icon.close` icon, it returns to the caller.

navigator2.dart - Giving Arguments

```
...
class PageA extends StatelessWidget {
  String info;
  PageA({Key? key,
        required this.info}) : super(key: key);
```

- We modify the PageA widget to receive arguments.

```
onPressed: () async {
  final result = await Navigator.push(
    context,
    MaterialPageRoute(
      builder: (c) {return PageA(info:'Page A');}
    )
  );
  print(result);
```

- We give an argument to the PageA widget.
- We wait for the return value from the widget, and print the result.

```dart
Navigator.pop(context, 10); // returns 10
...
// 10 is stored in the result
final result = await Navigator.push(
  context,
  MaterialPageRoute(...)
```

- The return value is passed with the 2nd argument of the `pop` method.

Route

- The Route class enables highly flexible but more complex page navigation in Flutter apps.
- MaterialApp and other root widgets support declarative routing by mapping route names to widget builders for easy navigation.

route1.dart

```
MaterialApp(
  initialRoute: '/',
  routes: {'/': (c) => MyPage(),
           '/page1': (c) => Page1(),
           '/page2': (c) => Page2()},
)
```

- We need to specify the path and its matching pages (widgets).

```
routes: {'/': (c) => MyPage(),
         '/page1': (c) => Page1(),
         '/page2': (c) => Page2()},
...
Navigator.pushNamed(context, '/page1');
```

- Navigator.pushNamed method is used to switch to other pages.

```
return Scaffold(
  body: Column(
    children: [
      TextButton(
        onPressed: () {
          Navigator.pushNamed(context, '/page1');
        },
        child: Text('Page1'),
      ),
```

- When users click the text button, the corresponding page opens.

```
return Scaffold(
  body: Column(
    children: [
      const Text('Page1'),
      IconButton(
        onPressed: () {
          Navigator.pop(context);
        },
        icon: const Icon(Icons.close)
      ),
```

- When users click the icon button, the caller widget is opened.

route2.dart - Giving Arguments

- To give arguments and return a value from the widget, we need to take more steps.
  - arguments
  - Page widgets with route name
  - pushNamed with arguments
  - setting the path in MaterialApp

arguments

```
class Arguments {
  final String message;
  final int value;

  Arguments(this.message, this.value);
}
```

- We need to make a class that stores all the input arguments.

page widgets with route name

```
class Page1 extends StatelessWidget {
  static String route = '/page1';
  Arguments arguments;

  Page1({required this.arguments,});
```

- Each page needs to have its route name.
- The argument is a member of the class.

```
 // arguments.value is the return value
onPressed: () {
  Navigator.pop(context, arguments.value);
},
```

- The return value is stored in the second argument of the `pop` method.

pushNamed with arguments

```
var res = await Navigator.pushNamed(
  context,
  Page1.route,
  arguments: Arguments('calling page1', 10),
);
```

- We can use the `pushNamed` method to pass (1) context, (2) widget route name, and (3) argument.

```
static Future<T?> pushNamed<T extends Object?>(
  BuildContext context,
  String routeName,
  {Object? arguments,})
```

- This is the signature of the pushNamed function.
- Because of this, we need to use named parameters only for the last one.

setting the path in MaterialApp

```
return MaterialApp(
  home: const MyPage(),
  onGenerateRoute: (settings) {
    ...
    if (settings.name == Page1.route) {...}
    else { ... }
```

- We must set up MaterialApp to route a path.
- The logic in the onGenerateRoute decides what page to switch to.

```dart
class RouteSettings {
  // The route's name, such as '/page1'
  final String? name;
  // The arguments passed to this route
  final Object? arguments;
  // (plus standard methods and constructors)
}
```

- The settings given to the onGenerateRoute property have a name and arguments field.

Workflow

- Using these mechanisms, when users press a button, Flutter can open the corresponding page in three steps.

1. Main Page (MyPage)
2. MaterialApp routing
3. Page1

1. Main Page (MyPage)

```
class MainPage extends StatelessWidget {
  static String route = '/page1';
  Widget build(BuildContext context) {
    TextButton(
      onPressed: () async {
        var res = await Navigator.pushNamed(
          context, Page1.route, arguments: Arguments('calling page1', 10),
        );
```

- In the page, when users press a TextButton, page information (Page1.route) and arguments are passed to the MaterialApp.

2. MaterialApp routing

```
onGenerateRoute: (settings) {
  if (settings.name == Page1.route) {
    return MaterialPageRoute(
      builder: (context) { return Page1(arguments: args,);
      },
    );
  }
```

- Using the name information, MaterialApp decides which Page to switch to (Page1) with the arguments information.

3. Page1

```
class Page1 extends StatelessWidget {
  ...
  Widget build(BuildContext context) {
    return Scaffold(
      body: Column(
        children: [
          Text(...)
          IconButton(
            onPressed: () {
              Navigator.pop(context, arguments.value);
            },
```

- Page1 has a Text and an IconButton to return to the caller.

```
var res = await Navigator.pushNamed(...);
```

- The return value is stored in the res variable.