02. Scaffold

The framework of a GUI application

- Scaffold
  - Why Flutter Scaffold?
  - scaffold1.dart
  - AppBar1.dart
  - bottomnavigatorbar1.dart
  - BottomNavigationBar2.dart
  - FloatingActionButton1.dart
  - SingleChildScrollView.dart

Scaffold

- Imagine you're building a house.
- We don't build a house from scratch; we need a `framework`:
  - **Foundation** (floor)
  - **Frame** (walls)
  - **Roof**

Why Flutter Scaffold?

- We need a similar `framework` when building an app.
  - Think of Scaffold as the House Framework
- It gives your mobile app screen's basic structure.

- Scaffold gives you these components ready to use:
  - **AppBar** (top bar with title)
  - **Body** (main content area)
  - **Floating Action Button** (the round "+" button)
  - **Bottom Navigation Bar** (buttons at bottom)
  - and more

Without Scaffold

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

void main() => runApp(MaterialApp(
  home: Text('Hello Students'),
));
```

- Result: Just plain text floating in space

With Scaffold

```
import 'package:flutter/material.dart';
void main() => runApp(MaterialApp(
  home: Scaffold(
    appBar: AppBar(title: Text('My App')),
    body: Text('Hello Students'),
  ),
));
```

- Result: Well-structured app with title bar and organized content

Downside of using Scaffold

- However, scaffolding is not a solution for every Flutter app design.
- It may add unnecessary complexity when the app does not need the skeleton.

scaffold1.dart

```
void main() => runApp(MaterialApp(home:MyApp()));
class MyApp extends StatelessWidget {
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Title'),),
      body: Text('Hello'),
      floatingActionButton: FloatingActionButton(
        onPressed: () {}, child: Icon(Icons.add),
      ),
    );
  }
}
```

- We can use the Widget/build().

Basic Scaffold Template

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        ...
      ),
    );
  }
}
```

```
appBar: AppBar(title: Text('App Title'),),
body: Center(child: Text('Your content goes here'),),
floatingActionButton: FloatingActionButton(...)
bottomNavigationBar: BottomNavigationBar(...)
```

- Add appBar (for title)
- Add body (for content)
- Add floatingActionButton (for input)
- Add buttomNavigationBar (for input/output)

AppBar1.dart

```
class MyHomePage extends StatelessWidget {
  const MyHomePage({required this.title});
  final String title;
  ...
```

- The title is used for the Text Widget.
- If it is not provided, Dart will raise a compiler error (required).

```
appBar: AppBar(
  title: const Text(this.title), // set from the constructor
  actions: [
    IconButton(
      icon: const Icon(Icons.info),
      onPressed: () {} // action when the icon is clicked
    ),
  ],
),
```

- AppBar has a title and an action item (IconButton).
- The () {} is a `lambda expression` that is executed when the icon button is pressed.

Constructor in the Widget

```
class MyApp extends StatelessWidget { // Dart
  const MyApp({super.key});
  ...
```

- `MyApp({super.key})` is a special Dart syntax for constructor.
- It receives the key in the constructor, and sets the parent's (super's) key.

bottomnavigatorbar1.dart

```
BottomNavigationBar(
  items:[
    BottomNavigationBarItem(), BottomNavigationBarItem(),
  ]
```

- BottomNavigationBar can have multiple BottomNavigationBarItems.
- We use a list `[...]` to store multiple components.

```
bottomNavigationBar: BottomNavigationBar(
  items: const [
    BottomNavigationBarItem(
      icon: Icon(Icons.home),
      label: 'Home',
    ),
    BottomNavigationBarItem(
      icon: Icon(Icons.settings),
      label: 'Settings',
    ),
  ],
),
```

BottomNavigationBar2.dart

```
Widget build(BuildContext context) {
  return MaterialApp(
    debugShowCheckedModeBanner: false, // <- remove banner
    home: BottomNavigationPage(),
  );
}
```

- In this example, we remove the "debug" banner by adding an option.
- For other configuration options, we can use the Dart document.

index, setState, and updateIndex

```
int index = 0;

void updateIndex(int index) {
  // Notify Dart UI to update the screen
  setState(() {this.index = index;});
}
```

- In the State<T> class, we add an index variable to track what item is selected.

```
void updateIndex(int index) {
  setState(() {this.index = index;});
}
```

- When the index is updated, we should use the `setState()` to redraw widgets by calling all the `build()` functions.
- In this example, we use the `updateIndex()` function.

```
bottomNavigationBar: BottomNavigationBar(
  currentIndex: index,
  onTap: (index) {updateIndex(index);},
```

- When one of the buttons is tapped, it calls the updateIndex() to update the index.
  - Then, `setState()` function is run.
  - All the widgets are redrawn.

```
bottomNavigationBar: BottomNavigationBar(
    currentIndex: index,
    onTap: (index) {updateIndex(index);},
    items: const [
      BottomNavigationBarItem(
        icon: Icon(Icons.home), label: 'Home'),
      BottomNavigationBarItem(
        icon: Icon(Icons.person), label: 'Profile'),
      BottomNavigationBarItem(
        icon: Icon(Icons.notifications), label: 'Notificatio',
      ),
    ],
  ),
);
```

- When users click one of the items, the index is updated and calls the `updateIndex`.

FloatingActionButton1.dart

```
Scaffold(
  appBar: AppBar(title: Text(widget.title),),
  body: Text('$_counter'),
  floatingActionButton:
    FloatingActionButton(
      onPressed: _incrementCounter,
      child: Icon(Icons.add),
    )
)
```

- The FAB (Floating Action Button) has a + icon.

```
void _incrementCounter() {
  setState(() {_counter++;});
}
```

- When the icon is pressed, the `_incrementCounter` is called.
- In the function, _counter variable is increased by one in the `setState()` function.
- Flutter widgets are redrawn with the updated _counter.

Decorating Widgets

```
body: Center(
  child: Text(
    '$_counter',
    style: TextStyle(fontSize: 50),
  ),
),
```

- We use `Center` to move the location and `TextStyle` to set the size of the text.

VSCode for Flutter programming

- Do we have to memorize the parameters such as `child` and `style` ?
  - When we use VSCode with the Flutter extension, it automatically suggests the parameters; so, in general, we don't have to.

List in the Widget

```
Container(
  child: Column(          // single widget
    children: [ ... ]  // multiple widgets
```

- When the widget needs multiple widgets, the parameter is `children` to specify that it requires a list of widgets.

SingleChildScrollView.dart

```
return Scaffold(
  body: SingleChildScrollView(
    child: Center(
      child: Text(
        '''
Welcome to Flutter!
Counter Value: $_counter ...
```

- We can use the `SingleChildScrollView` widget for scrolling.