Lab Manual for CSM3114 - Framework-Based Mobile Application Development

Prepared by Mohamad Nor Hassan*

October 2023

*Universiti Malaysia Terengganu

The Flutter framework let the students to develop cross-platform mobile applications which can run on Android or iOS platforms.

This lab session will introduce to the student on the development of basic mobile applications focusing on the Flutter and Dart fundamentals that emphasise on core Flutter and dart syntax when developing the solutions.

The learning outcomes of this lab session are:

1. To use *Row* and *Column* widgets for developing the simple app.
2. To use *Container* widget for developing the simple app.
3. Using the arrow notation function.
4. Applying the *StatefulWidget* in mobile apps.

1 Basic concept on the common Flutter widgets used to develop mobile application

1.1 Widget Alignment

1. For the alignment, developer can control the alignment of row or column that aligns to its children using ***mainAxisAlignment*** and ***crossAxisAlignment*** properties.

2. For *Row* widget, the main axis runs horizontally and the cross axis runs vertically.

3. For *Column* widget, the main axis runs vertically and the cross axis runs horizontally .

4. Open your online DartPad.

5. Create the main app and enclosed the *MaterialApp*, *Scaffold*, *AppBar*, *Text* and *MyWidget* respectively.

```dart
/**
Author: MNor
Date   : 15 June 2023
**/
import 'package:flutter/material.dart';

void main() {
  //Run the app by calling runApp method and
  //passing in the MaterialApp widget...
  runApp(
    MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Rows & Columns'),
          centerTitle: true,
        ), //AppBar
        body: MyWidget(),
      ), //Scaffold
    ), //Material App
  ); //runApp
}

class MyWidget extends StatelessWidget {
```

6. Create a custom widget know as *MyWidget*. Follow there requirements:

   (a) Inside the *MyWidget*, return the *Column* widget.

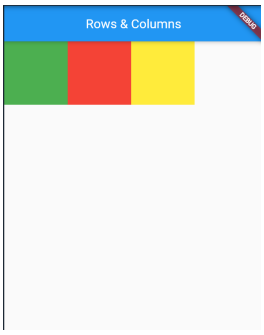
   (b) For the *Column* widget, create three (3) *Container* widget with the width and height properties as 100 respectively.

(c) Then, assign a color properties as red, green and yellow to each of the *Container* widget.

```
class MyWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Row(children: [
      Container(
        width: 100,
        height: 100,
        color: Colors.green,
      ), //Container
      Container(
        width: 100,
        height: 100,
        color: Colors.red,
      ), // Container
      Container(
        width: 100,
        height: 100,
        color: Colors.yellow,
      ),
    ]); // Row
  }
}
```

7. Refer to the snapshot of the source code to implement this task.

8. Save and run the code. You will get the following output.



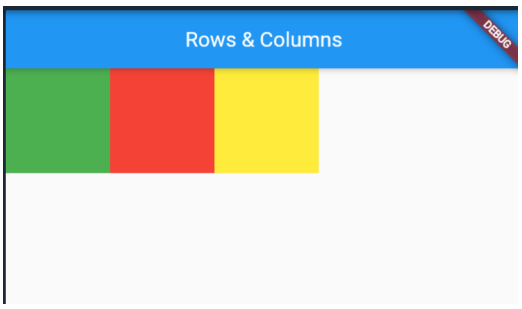
9. Modify *MyWidget* by replacing a *Row* widget with a *Column* widget. Re-run your code.

10. Attached the both source codes and the outputs for program in the report.

1.2 Define the space evenly the *Container* widget inside the screen

1. Based on the source code written for *MyWidget* in part 1.1, revert back *Column* widget to *Row*.

2. Try to shift the output to the left position. You will see the position of the *Column* widgets are static .



3. In order to prevent this issue, for *Row* widget, try to add *mainAxisAlignment* property to provide equal space among the *Container* widgets.

```
class MyWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      children: [
      Container(
        width: 100,
        height: 100,
        color: Colors.green,
      ), //Container
      Container(
```

4. Re-run the program and try to move left or right the output. You will notice that the *Container* widgets space will distributed equally.

5. Attached the portion of the source codes and the outputs for program in the report.

1.3 Use the *Center* widget to center the position of *Column* widgets

1. Inside the *MyWidget*, replace the *Row* widget to *Column* widget for the program written in part 1.2.

2. In order to re-position the *Column* widget to the center, you need to use *Center* widget.

3. Modify the code and add the *Center* widget.

4. Then, re-run your code.

5. Please attach the source in your report.

```
class MyWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child:  Column(
      mainAxisAlignment: MainAxisAlignment.spaceEvenly,
      children: [
      Container(
        width: 100,
        height: 100,
        color: Colors.green,
      ), //Container
      Container(
        width: 100,
        height: 100,
        color: Colors.red,
      ), // Container
      Container(
        width: 100,
        height: 100,
        color: Colors.yellow,
      ),
    ]) // Column
    ); // Container
  }
}
```

1.4 Exercise

1. Create a UI that having two (2) *Container* widgets and structure these widgets horizontally.

2. Add the label as 'Submit' for the 1st *Container* and 'Cancel' for the the 1st *Container*.

3. When user click to the 1st *Container*, display the message to Debug Console as *'Information successfully submitted'*.

4. When user click to the 2nd *Container*, display the message to Debug Console as *'Cancel submission'*.

5. Complete your coding and run the program.

6. Attach the source code and the output in the report.

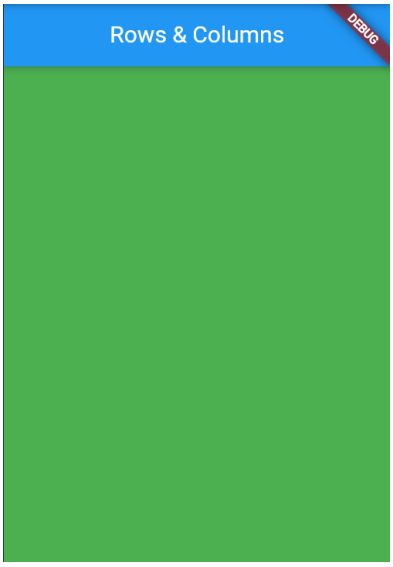
2 Using the Arrow Notation (⇒(); Function)

2.1 Single expression in function by using the arrow notation (⇒();) syntax

1. Based on the existing code in part 1.3, modify the code by leave it only one *Container* widget.

2. Remove the *Column* widget and both properties height and weight.

3. Try to run the code. You should get the following output.

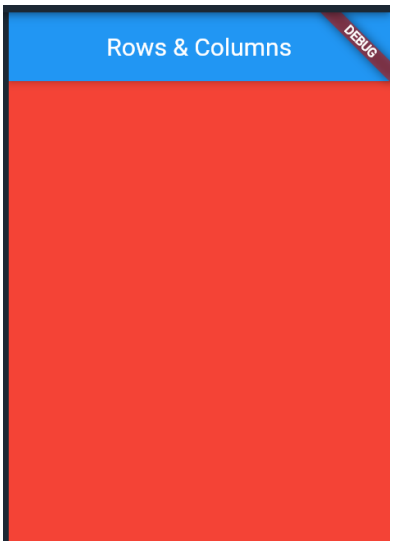


4. Then, remove the *Center* widget. You will get the similar output.

```
class MyWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return (
      Container(
                color: Colors.green,
      )
    ); //Container
  }
}
```

5. Change the method by using the arrow notation function for *main()* and *build()* methods by making modification for both methods and change the property *color* for *Container* widget to red.

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


void main() => runApp(MaterialApp(home:MyWidget()));


class MyWidget extends StatelessWidget {

  @override
Widget build(BuildContext context) => Scaffold(
      appBar: AppBar(
          title: Text('Rows & Columns'),
          centerTitle: true,
      ), //AppBar
     body: Container(color: Colors.red,));
}
```



6. Attached the portion of the source codes and the outputs for program in the report.

2.2 Exercise

1. Use the arrow function to develop *MyCustom* widget that consist the button and text with the default value as 'My text'.

2. When user click the button, the text will change to 'Mobile Framework'.

3. Finally, attach a print screen the snapshot of the output.

3 Implementing A StatefulWidget

3.1 Changing the background colour for widget

1. Open your Visual Studio IDE.

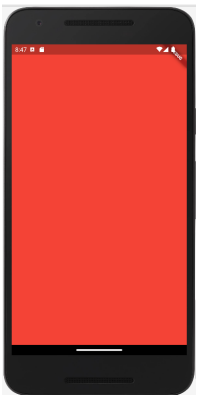
2. Create new project as *stateful_app* by running Flutter command.

3. Create main program calling *home* property as *MyApp()*.

4. Create class *MyApp()* by using *StatefulWidget*.

5. Complete the details on class *MyApp()*.

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

Run | Debug | Profile
void main() => runApp(
      MaterialApp(
        home: MyApp(),
      ), // MaterialApp
    );

class MyApp extends StatefulWidget {
  @override
  State<StatefulWidget> createState() => _MyApp();
}

class _MyApp extends State<MyApp> {
  //Assign initial variable for tracking purposes (not State)
  bool isButtonPressed = false;

  @override
  Widget build(BuildContext) {
    return Scaffold(
      body: Container(
        color: Colors.red,
      ), // Container
    ); // Scaffold
  }
}
```



6. Save and run a code.

7. You will get the following output.

8. Try to tap the emulator screen on the background red colour. The appearance still red (nothing changed).

9. In order to change the background red into yellow colour, try to use *GestureDetector* widget.

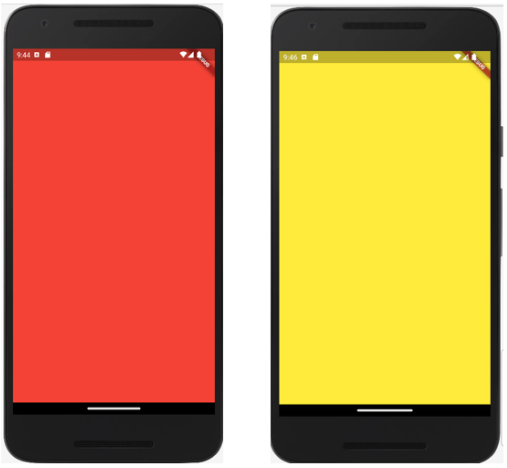
10. Start modify the *build()* method.

11. Modify the existing code by replacing *Container* widget with *GestureDetector* widget.

12. Construct the logic for *onTap* property by using the *setState()* function in order to change the appearance of background colour when user taps the background colour (widgets will rebuild..!!!).

```
  @override
  Widget build(BuildContext) {
    return Scaffold(
      body: GestureDetector(
        onTap: () {
          if (isButtonPressed) {
            setState(() {
              isButtonPressed = false;
            });
          } else {
            setState(() {
              isButtonPressed = true;
            });
          }
        },
        child: Container(color: getColor()),
      ), // GestureDetector
    ); // Scaffold
  }
  //Create function to return colour...
  Color getColor() {
    if (isButtonPressed) {
      return Colors.red;
    } else {
      return Colors.yellow;
    }
  }
}
```

13. Save and re-run your code. Try to simulate the background colour by tapping the emulator. [Note: You will see the value of *isButtonPressed* variables is changing whenever process of rebuilding the widgets occur when using *StatefulWidget*. It will reflect the changing on the background colour from red to yellow and via versa].



14. Attached the source code and the output in your report.

3.2 Implement Stateful Widget using *Button* widget

1. Save your source code in your previous work in section 3.1.

2. Delete existing code from section 3.1.

3. Create *main()* method with *StatefulWidget*.

```
b > main.dart > _CounterState > _increment
1   import 'package:flutter/material.dart';
2
    Run | Debug | Profile
3   void main() => runApp(
4         MaterialApp(
5           home: Scaffold(
6             body: Center(
7               child: Counter(),
8             ), // Center
9           ), // Scaffold
10        ), // MaterialApp
11      );
12
13  class Counter extends StatefulWidget {
14    //const Counter({super.key});
15
16    @override
17    State<Counter> createState() => _CounterState();
18  }
```

4. Create *Counter* class by inherits from *StatefulWidget*. Assign a *createState()* to the class.

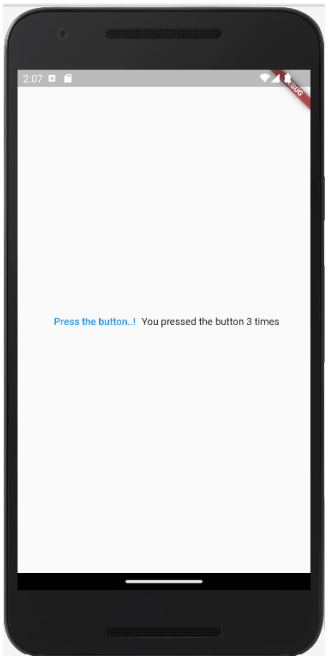
5. Create *_CounterState* by inherit from *State* class. Inside this class, add the logic for counter increment by defining *_inrement()* function or method.

6. Complete the remaining of code.

```
class _CounterState extends State<Counter> {
  // Define button..

  //Initialise the counter...
  int _count = 0;

  //Increment the counter....
  void _increment() {
    setState(() {
      _count++;
    });

    print('Debug count = $_count');
  }

  //Consruct Widget build() -> onPressed
  @override
  Widget build(BuildContext context) {
    return Row(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        TextButton(onPressed: _increment, child: Text('Press the button..!')),
        Text('You pressed the button $_count times'),
      ],
    ); // Row
  }
}
```

7. Save your code and run the program.

8. Verify by clicking the *Flat button*.



9. Attach the source code, the snapshot of UI in your report.

10. Review the value of *count* at Debug Console. Finally, enclosed also the Debug Console

```
PROBLEMS 5   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS                    Filter (e.g. text, !exclude)
Restarted application in 4,627ms.
I/flutter (25950): Debug count = 1
D/EGL_emulation(25950): app_time_stats: avg=11109.70ms min=11109.70ms max=11109.70ms count=1
D/EGL_emulation(25950): app_time_stats: avg=631.90ms min=37.86ms max=1225.94ms count=2
I/flutter (25950): Debug count = 2
I/flutter (25950): Debug count = 3
D/EGL_emulation(25950): app_time_stats: avg=61.77ms min=28.00ms max=347.72ms count=16
```

output in the report.

3.3 Applying *StatefulWidget* for Updating the State of Scaffold *bottomSheet* property via *TextField* widget

1. Save your source code in section 3.2 in Notepad for future reference.
2. Go to *main.dart* and remove all source code.
3. Construct the code based on the following requirements:
   (a) Create *main()* function by passing *MyApp* as a *StatelessWidget*.
   (b) Create *MyApp* class and return *MaterialApp* widget with *home* property assigned with *MyStatefulApp*.
   (c) Then, create *MyStatefulApp* as a *StatefulWidget*.
   (d) Create *_MyStatefulApp* class and construct the logic by initialise *_inputText* variable and construct the *build()* method.
   (e) Inside the *Scaffold* widget, include *AppBar*, *Center* and *TextField* widget.
   (f) In addition, inside the *TextField* widget, for *onChange* property, construct a CallBack function by passing the value obtain from *TextField* widget and invoke *setState()* function to dynamically change the value of *_inputText* variable.
4. Save your code.
5. Run your program.
6. Test the *TextField* by passing 2 or 3 text and verify the texts display at the *bottomSheet* of *Scaffold*.
7. Captured the output from virtual device and attach it in your report.

```dart
/*
 Author : MNor
 Date : 13 Sept 2023
*/
import 'package:flutter/material.dart';

Run | Debug | Profile
void main() => runApp(MyApp());

//Create Stateless widget....
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: MyStatefulApp(),
    ); // MaterialApp
  }
}

//Create Stateful widget..
class MyStatefulApp extends StatefulWidget {
  @override
  _MyStatefulApp createState() => _MyStatefulApp();
}
```

```dart
class _MyStatefulApp extends State<MyStatefulApp> {
  //initialise the variable..
  String _inputText = '';

  // For any rebuild widget.. for updating.. use setState()....
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('My Statefull App'),
        centerTitle: true,
      ), // AppBar
      body: Center(
        child: TextField(
          decoration: InputDecoration(hintText: "Enter text here...!"),
          onChanged: (value) {
            setState(() {
              _inputText = value;
            });
          },
        ), // TextField
      ), // Center
      bottomSheet: Container(
        alignment: Alignment.center,
        height: 50,
        child: Text('You text is : $_inputText'),
      ), // Container
    ); // Scaffold
  }
}
```


3:28
DEBUG
My Statefull App
Welcome to Mobile Development MNor..!
You text is : Welcome to Mobile Development MNor..!

3.4 Exercise - Applying *StatefulWidget*

1. Create one *TextField* in your UI.

2. Display 'Exceeded Credit Limit' if the value is > 500. Otherwise, display as 'Processing'. Display the bold message through text field.