Lab Manual for CSM3114 - Framework-Based Mobile Application Development

Prepared by Mohamad Nor Hassan*

October 2024

*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 solely focusing on the Flutter and Dart fundamentals that emphasise on core Flutter and dart syntax when developing the solutions.

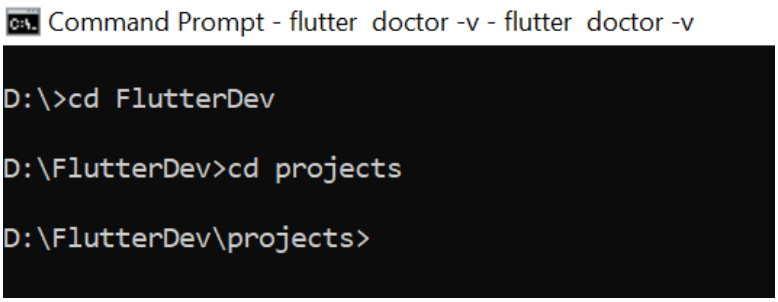
The learning outcomes of this lab session are:

1. Exploring the core Flutter and Dart syntax.
2. Write a Flutter and Dart syntax.
3. Applying the Flutter widgets when build the mobile apps.
4. Combining the widgets when develop the mobile app.

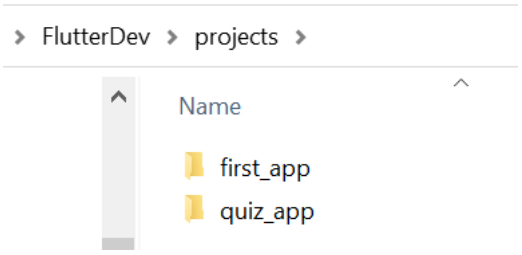
1 Creating a simple quiz application

1.1 Creating a New Application

1. Go to the command prompt.
2. Go to your Flutter workspace. For example, *D:\FlutterDev\projects*.



3. Create a new Flutter project and rename it as *quiz_app* [Note: Use command *flutter create quiz_app*].
4. Go to your Windows explorer and check quiz_app's folder created.



1.2 Adding a Layout Widget for Quiz app

1. Open the *main.dart* file via Visual Studio Code.
2. The next tasks are to fulfill the following requirements:
   (a) Adding *Scaffold* widget to the UI.
   (b) Inside the *Scaffold* widget, add the *AppBar* widget with properties *title* as 'My Quiz App'.
   (c) Add quiz questions and followed by the three (3) button that representing the answers. [Note: In order to group the question and answers in one single UI, you must create the *Column* widget - group a widgets together].

(d) Inside the *Column* widget, add *Text* widget and three (3) *ElevatedButton* widgets.

(e) The sample of coding is shown below.

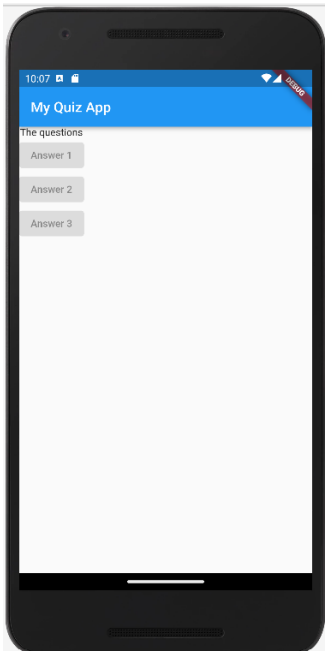
lib > main.dart > ...

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

Run | Debug | Profile
void main() {
  runApp(const MyApp());
}

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

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('My Quiz App'),
        ), // AppBar
        body: const Column(
          children: [
            Text('The questions'),
            ElevatedButton(onPressed: null, child: Text('Answer 1')),
            ElevatedButton(onPressed: null, child: Text('Answer 2')),
            ElevatedButton(onPressed: null, child: Text('Answer 3')),
          ],
        ), // Column
      ), // Scaffold
    ); // MaterialApp
  }
}
```



3. Open your Android Studio and Android Virtual Device (AVD).

4. Start up the Android emulator.

5. Go to Visual Studio Code, go to Run and click Run Without Debugging.

6. Your will get the above output.

1.3 Exercise: Replace the *ElevatedButton* widget with *OutlineButton* widget

1. Based on coding part 1.2, modify the source code by replacing the first existing button with *OutlinedButton* widget.

2. Attached your source code and the output.

3. Once you attach the source code, change it to *ElevatedButton* widget again.

2 The Flutter Official Documentation

2.1 Finding the Flutter Official Documentation

1. Go to Flutter official website.
2. Go to the *Docs*.
3. Show the steps by steps how you want to find the details implementation of *ElevatedButton* widget.
4. You are required to attach a print screen the snapshot of the *ElevatedButton* widget information.

2.2 Exploring *Text* Widget

1. Go to Flutter official documentation.
2. Modify the existing source code you wrote in part 1.2 by changing the *Text* widget style that display 'The questions' as bold.
3. Attached the source code.
4. Finally, attach a print screen the snapshot of the output.

3 Passing Callback function

3.1 Creating function and attach to button

1. Expand the codes written in part 1.2 by creating new function known as *void answerQuestion()*.

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

  void answerQuestion() {
    print('Answer chosen..!');
  }

  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: const Text('My Quiz App'),
        ), // AppBar
        body:  Column(
```

2. Then, attach the function to the three (3) *ElevatedButton* widget.

```
        body:  Column(
          children: [
            Text('The questions'),
            ElevatedButton(onPressed: answerQuestion, child: Text('Answer 1')),
            ElevatedButton(onPressed: answerQuestion, child: Text('Answer 2')),
            ElevatedButton(onPressed: answerQuestion, child: Text('Answer 3')),
          ],
        ), // Column
      ), // Scaffold
    ); // MaterialApp
  }
}
```

3. Run the apps via Visual Studio code.

4. Verify the output from the debug console and attach the output in your lab report.

3.2 Using anonymous function

1. Modify the code from part 3.1, by assigning the anonymous function to second (2nd) and third (3rd) *ElevatedButton* widget.

2. Run your app. Pressed second (2nd) button. Captured the output from Debug Console.

3. Subsequently, pressed first (1st) button. Captured the output from Debug Console.

4. The following code shows the implementation of anonymous function.

```
22          body: Column(
23            children: [
24              Text('The questions'),
25              ElevatedButton(onPressed: answerQuestion, child: Text('Answer 1')),
26              ElevatedButton(
27                  onPressed: () => print('Answer 2 chosen...!'),
28                  child: Text('Answer 2')), // ElevatedButton
29              ElevatedButton(
30                  onPressed: () => print('Answer 3 chosen...!'),
31                  child: Text('Answer 3')), // ElevatedButton
32            ],
33          ), // Column
34        ), // Scaffold
35      ); // MaterialApp
36    }
37  }
```

3.3 Updating Widget Data using Stateful Widget

1. From the *main.dart* in part 3.1, rename the stateless widget to stateful widget.

2. Create a second class known as *MyAppState* by inherit the *State*'s class. Implement the details logic for the class.

3. Inside the function *void answerQuestion()*, add the method or function known as *setState()*.

4. Define a list of questions.

5. Assign a *answerQuestion* function to each of the *ElevatedButton* widget.

```
main.dart > MyAppState
1   import 'package:flutter/material.dart';
2
    Run | Debug | Profile
3   void main() {
4     runApp(const MyApp());
5   }
6
7   class MyApp extends StatefulWidget {
8     @override
9     const MyApp({super.key});
10    State<StatefulWidget> createState() {
11      // TODO: implement createState
12      return MyAppState();
13      throw UnimplementedError();
14    }
15  }
```

6. Run the apps. Test the app by clicking to button Answer 1 and Answer 2. Explain the findings

```
17  class MyAppState extends State<MyApp> {
18    var questionIndex = 0;
19
20    void answerQuestion() {
21      //print('Answer chosen!');
22      setState(() {
23        questionIndex = questionIndex + 1;
24      });
25
26      print(questionIndex);
27    }
28
29    // This widget is the root of your application.
30    @override
31    Widget build(BuildContext context) {
32      //Define a lists
33      var questions = [
34        'What \'s your favourite colour?',
35        'What \'s your favourite animal?',
36      ];
37
```

```
38      return MaterialApp(
39        home: Scaffold(
40          appBar: AppBar(
41            title: const Text('My Quiz App'),
42          ), // AppBar
43          body: Column(
44            children: [
45              Text(
46                questions[questionIndex],
47                //questions.elementAt(0),
48              ), // or questions[0]; // Text
49              ElevatedButton(onPressed: answerQuestion, child: Text('Answer 1')),
50              ElevatedButton(onPressed: answerQuestion, child: Text('Answer 2')),
51              ElevatedButton(onPressed: answerQuestion, child: Text('Answer 3')),
52            ],
53          ), // Column
54        ), // Scaffold
55      ); // MaterialApp
56    }
57  }
```

3.4 Exercise

1. Develop the simple UI for application used to choose two(2) types of discount.