Experiment No:02

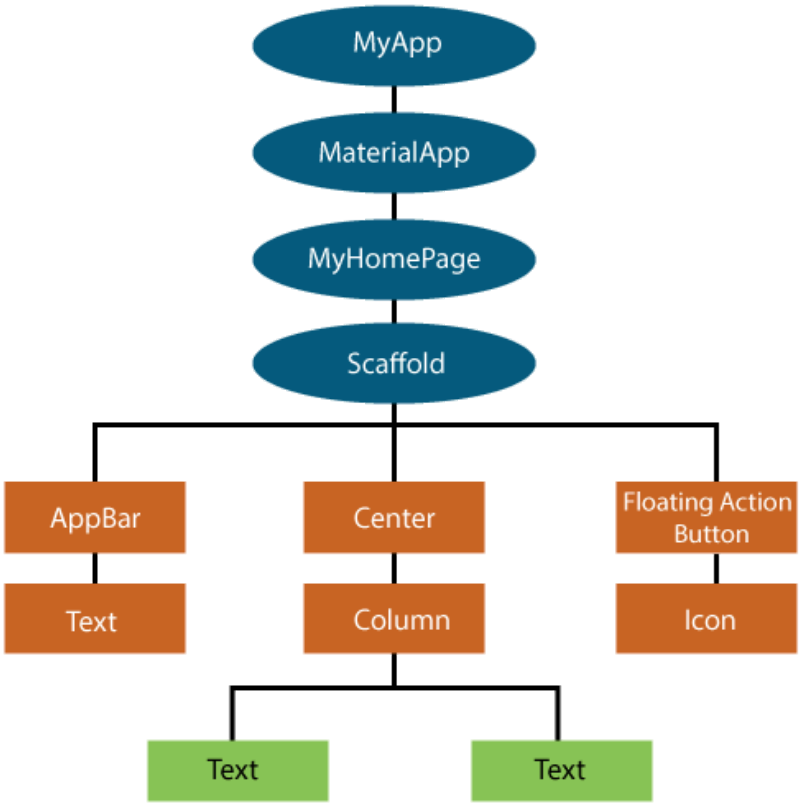
Aim: To design Flutter UI by including common widgets.

Theory:

Whenever you are going to code for building anything in Flutter, it will be inside a widget. The central purpose is to build the app out of widgets. It describes how your app view should look like with their current configuration and state. When you made any alteration in the code, the widget rebuilds its description by calculating the difference of previous and current widget to determine the minimal changes for rendering in UI of the app.

Widgets are nested with each other to build the app. It means the root of your app is itself a widget, and all the way down is a widget also. For example, a widget can display something, can define design, can handle interaction, etc.

The below image is a simple visual representation of the widget tree.



Types of Widget

We can split the Flutter widget into two categories:

1. Visible (Output and Input)
2. Invisible (Layout and Control)

Common Widgets in Flutter

Flutter provides a wide range of widgets that developers can use to build rich and interactive user interfaces. These widgets serve various purposes, from displaying text and images to handling user input and managing layouts.

In the below Flutter code, the following widgets are used:

1. MaterialApp: Represents the root widget of the application and configures the overall theme and home page.

2. Scaffold: Provides a basic layout structure for the home page, including an app bar and body content area.

3. AppBar: Displays a toolbar at the top of the screen with a title.

4. Center: Centers its child widget both vertically and horizontally within its container.

5. Column: Arranges its children widgets vertically.

6. Text: Displays text on the screen with customizable styles.

7. SizedBox: Provides a box with a specified size, used for adding space between widgets.

8. TextField: Allows users to input text.

9. ElevatedButton: Represents a button with a raised appearance.

10. TextButton: Represents a button with text but no background color or elevation.

11. IconButton: Represents a button with an icon.

12. Icon: Displays an icon image.

13. Image.asset: Displays an image loaded from the assets folder of the project.

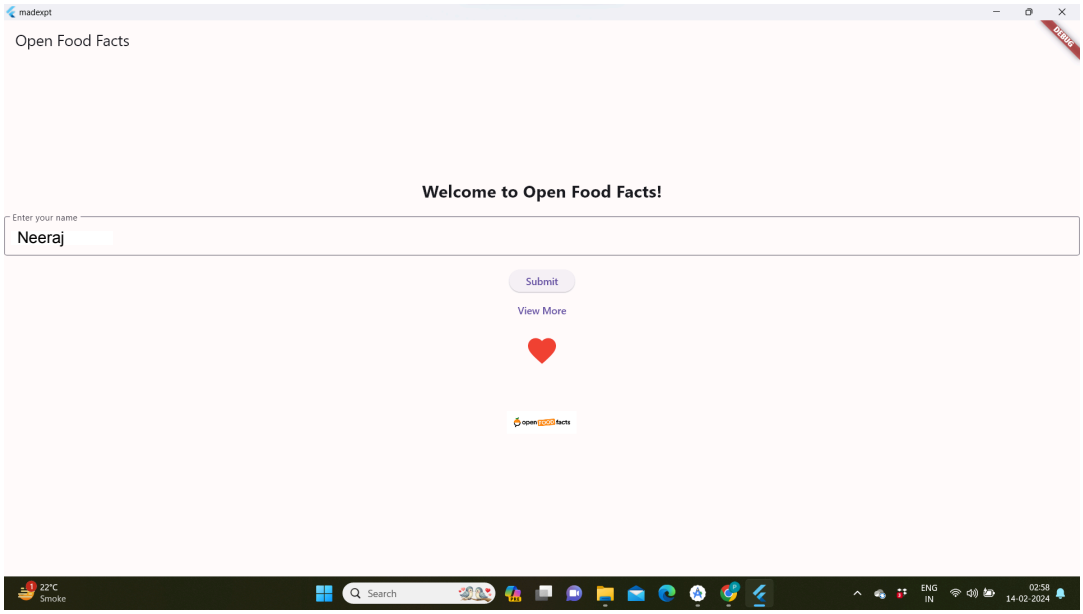
These widgets are used to create a basic UI with text, input fields, buttons, and images.

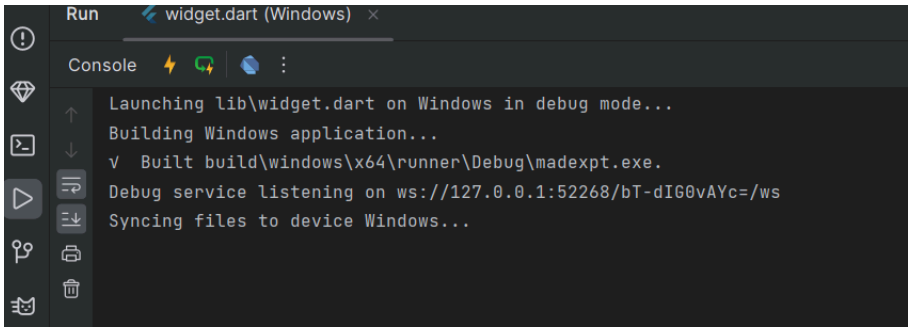
Code:

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

void main() {
  runApp(OpenFoodFactsApp());
}

class OpenFoodFactsApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Open Food Facts',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Open Food Facts'),
          backgroundColor: Colors.purple.shade100,
        ),
        body: Center(
          child: SingleChildScrollView(
            padding: EdgeInsets.all(20.0),
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: <Widget>[
                Text(
                  'Welcome to Open Food Facts!',
                  style: TextStyle(
                    fontSize: 20.0,
                    fontWeight: FontWeight.bold,
                  ),
                  textAlign: TextAlign.center,
                ),
                SizedBox(height: 20.0),
                TextField(
                  decoration: InputDecoration(
                    labelText: 'Enter your name',
                    border: OutlineInputBorder(),
                  ),
                ),
                SizedBox(height: 20.0),
                ElevatedButton(
                  onPressed: () {
                    // Submit logic
                  },
                  style: ElevatedButton.styleFrom(
```

```
                primary: Colors.purple.shade100,
              ),
              child: Text('Submit'),
            ),
            SizedBox(height: 10.0),
            TextButton(
              onPressed: () {
                // View more logic
              },
              child: Text(
                'View More',
                style: TextStyle(
                  color: Colors.deepPurple,
                ),
              ),
            ),
            SizedBox(height: 20.0),
            Icon(
              Icons.favorite,
              color: Colors.red,
              size: 40.0,
            ),
            SizedBox(height: 20.0),
            Image.asset(
              'images/openfoodfactslogo.png', // make sure the image is present in your assets
              height: 40.0,
            ),
          ],
        ),
      ),
    ),
  );
 }
}
```

Output:





Conclusion:

In conclusion, designing Flutter UIs with common widgets provides a robust foundation for creating beautiful, functional, and responsive applications. By leveraging the versatility and flexibility of these widgets, developers can efficiently build UIs that deliver an exceptional user experience on both iOS and Android platforms.