Experiment 02 : To design Flutter UI by including common widgets.

Aim: To design Flutter UI by including common widgets.

Theory:

What are Flutter Widgets?

Flutter Widgets are essentially the building blocks of Flutter applications. They are the UI elements that are used to construct the user interface of the app. Widgets can be simple, like a button or a text field, or they can be complex, like a page layout or an animation.

Properties of Widgets:

Immutable: Widgets in Flutter are immutable. Once created, their properties cannot be changed. Instead, when a property needs to be updated, a new widget with the updated property is created.
Composable: Widgets in Flutter are composable, meaning they can be nested and combined to create complex UIs. This allows for easy reuse of UI components and building blocks.
Declarative: Flutter uses a declarative approach to UI development, meaning that UIs are described in terms of their desired state rather than in terms of the steps needed to achieve that state. This makes it easier to reason about and maintain UI code.
Stateful and Stateless: Widgets in Flutter can be either stateful or stateless. Stateless widgets are immutable and do not have any internal state, while stateful widgets can change their appearance or behavior in response to user interactions or other events.

Common Categories of Widgets:

Layout Widgets: These widgets are used to arrange other widgets on the screen, such as Row, Column, Stack, etc.
Material Design Widgets: Flutter provides a rich set of widgets that implement the material design guidelines, such as AppBar, FloatingActionButton, Card, etc.
Text and Styling Widgets: Widgets like Text, RichText, and TextStyle are used to display text on the screen with different styles and formatting.
Input Widgets: Flutter provides widgets for user input, such as TextField, Checkbox, Radio, Slider, etc.
Scrolling Widgets: Widgets like ListView, GridView, SingleChildScrollView, etc., are used to create scrollable views.
Animation Widgets: Flutter provides widgets for creating animations, such as AnimatedContainer, AnimatedOpacity, etc.

Widget Tree:

In Flutter, widgets are organized in a hierarchical tree structure called the widget tree. At the root of the tree is the MaterialApp or CupertinoApp widget, depending on whether you're using material design or Cupertino (iOS-style) design. Each widget in the tree has a parent-child relationship with other widgets. When Flutter renders the UI, it traverses this tree and builds the UI accordingly.

Some basic Flutter widgets are:

1. Container: A rectangular box that can be decorated with a background, border, or shadow.
2. Text: Displays a string of text with a single style.
3. Row: Arranges its children in a horizontal array.
4. Column: Arranges its children in a vertical array.
5. Stack: Allows widgets to be stacked on top of each other.
6. Image: Displays an image.
7. ListView: A scrollable list of widgets arranged sequentially.
8. GridView: A scrollable grid of widgets arranged in rows and columns.
9. AppBar: A material design app bar.
10. Scaffold: Provides a framework for implementing the basic material design visual layout structure.
11. TextField: A widget that allows users to enter and edit text.
12. RaisedButton and FlatButton: Buttons with different styles for user interaction.
13. AlertDialog and BottomSheet: Dialog and modal bottom sheet widgets for showing alerts or additional information.
14. Drawer: A panel that slides in from the side of the screen.
15. Card: A material design card.
16. Icon: A material design icon.
17. Switch and Checkbox: Widgets for toggling boolean values.
18. ProgressIndicator: Displays a visual indication of progress.

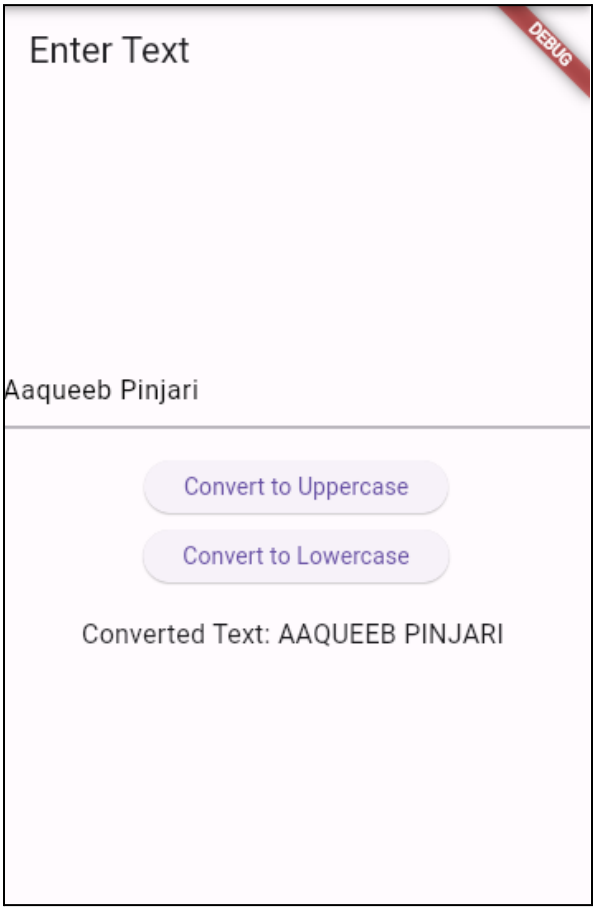
Code:

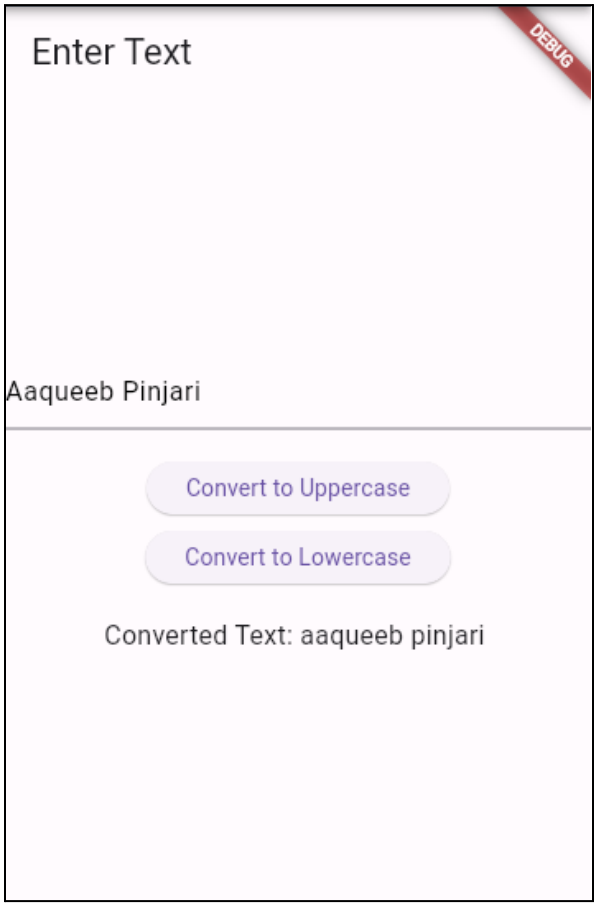
```
import 'package:flutter/material.dart';

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Text Display App',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        backgroundColor: Colors.yellow[100], // Change background color
        textTheme: TextTheme(
          bodyText1: TextStyle(
            fontFamily: 'Roboto', // Change font family
            fontSize: 16,
            color: Colors.black, // Change text color
```

```
          ),
        ),
      ),
      home: TextDisplayScreen(),
    );
  }
}

class TextDisplayScreen extends StatefulWidget {
  @override
  _TextDisplayScreenState createState() => _TextDisplayScreenState();
}

class _TextDisplayScreenState extends State<TextDisplayScreen> {
  String _enteredText = '';
  String _convertedText = '';

  void _updateEnteredText(String newText) {
    setState(() {
      _enteredText = newText;
    });
  }

  void _convertToUpperCase() {
    setState(() {
      _convertedText = _enteredText.toUpperCase();
    });
  }

  void _convertToLowerCase() {
    setState(() {
      _convertedText = _enteredText.toLowerCase();
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Enter Text'),
      ),
      body: Center(
```

```
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            TextField(
              decoration: InputDecoration(
                hintText: 'Enter your text here',
              ),
              onChanged: (text) {
                _updateEnteredText(text);
              },
            ),
            SizedBox(height: 20),
            ElevatedButton(
              onPressed: _convertToUpperCase,
              child: Text('Convert to Uppercase'),
            ),
            SizedBox(height: 10),
            ElevatedButton(
              onPressed: _convertToLowerCase,
              child: Text('Convert to Lowercase'),
            ),
            SizedBox(height: 20),
            Text(
              'Converted Text: $_convertedText',
              style: TextStyle(fontSize: 16),
            ),
          ],
        ),
      ),
    );
  }
}
```

Output:





Conclusion:

Understanding widgets and how they work together is fundamental to building Flutter applications efficiently and effectively. As you become more familiar with Flutter, you'll discover a wide range of widgets and learn how to use them to create beautiful and functional user interfaces.