Experiment No: 2 a) **Date:**

Aim:

To explore various Flutter widgets such as Text, Image, Container etc.,.

Objective:

In this lab experiment, we will explore various Flutter widgets such as Text, Image, and Container. Follow the steps below to set up a basic Flutter application and customize it according to our needs.

Procedure:

1. Create a new Flutter project by running the following command in your terminal:
   ```
   flutter create my_flutter_app
   ```
   The command creates a Flutter project directory called my_flutter_app that contains a simple demo app that uses Material Components.
2. Change to the Flutter project directory.
   ```
   cd my_flutter_app
   ```
3. Open the lib/main.dart file in your Flutter project.
4. Replace the existing code with the following code snippet:

```
void main() {
runApp(const MainApp()); // Run the MainApp widget as the root of the application
}

class MainApp extends StatelessWidget {
const MainApp({super.key}); // Constructor for the MainApp widget

@override
Widget build(BuildContext context) {
   return MaterialApp( // Create a MaterialApp widget
   home: Scaffold(
      body: Center(
      child: Column(
         mainAxisAlignment: MainAxisAlignment.center, // Center the column vertically
         children: [
         const Text( // Display a text widget with the given text
            'Welcome to Flutter!',
            style: TextStyle(fontSize: 24),
         ),
         const SizedBox(height: 16),
         Image.asset( // Display an image from the assets folder
            'assets/images/flutter_logo.png',
            width: 200, // Set the width, height of the image
            height: 200,
         ),
         const SizedBox(height: 16),
         Container( // Create a container widget
            width: 200,
            height: 50,
            color: Colors.blue, // Set the background color of the container
            child: const Center(
            child: Text( // Display a text widget
               'Start',
               style: TextStyle(color: Colors.white), // Set the text color
            ),
            ),
         ),
         ],
      ),
      ),
```

```
    ),
    );
}
}
```

5. Get the image flutter_logo.png from wikipedia and save it in the assets/images/ directory of your Flutter project or replace the image path 'assets/images/flutter_logo.png' with the actual path to your image file and then save the file.
6. Open the pubspec.yaml file in your Flutter project and add the following lines after flutter: to include the image asset in your project:

```
flutter:
  uses-material-design: true
  assets: <-- Add this line
    - assets/images/flutter_logo.png <-- Add this line
```

   Save the file.
7. Run your Flutter project using the following command:

```
flutter run
```

   Select the appropriate device to run the app.
   - Enter r to hot reload the app and see the changes you made to the code.
   - Enter q to quit the app.

Output:

Experiment No: 2 b) **Date:**

Aim:

To implement different layout structures using Row, Column, and Stack widgets.

Code Snippet (main.dart):

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

void main() {
  runApp(const MainApp());
}

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

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: Scaffold(
        body: LayoutApp(),
      ),
    );
  }
}

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

  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        const Text('I\'m in a Coloum and Centered. The below is a row.'),
        const SizedBox(height: 20),
        Row(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: [
            for (var color in [Colors.red, Colors.green, Colors.blue])
              Container(
                width: 100,
                height: 100,
                color: color,
              ),
          ],
        ),
        const SizedBox(height: 20),
        Stack(
          alignment: Alignment.center,
          children: [
            Container(
              width: 300,
```

```
            height: 200,
            color: Colors.yellow,
          ),
          const Text(
            'Stacked on Yellow Box',
            style: TextStyle(fontSize: 24, fontWeight: FontWeight.bold),
          ),
        ],
      ),
    ],
  );
 }
}
```

Output:

Experiment No: 3 a) **Date:**

Aim:

To design a responsive UI that adapts to different screen sizes.

Objective:

The objective of this experiment is to create a Flutter project that demonstrates the concept of responsive UI design. The UI should adjust its layout and appearance based on the screen size of the device.

Code Snippet (main.dart):

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

void main() {
runApp(const MainApp());
}

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

@override
Widget build(BuildContext context) {
   return const MaterialApp(
   title: 'Responsive UI',
   home: ResponsiveHomePage(),
   );
}
}

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

@override
Widget build(BuildContext context) {
   return Scaffold(
   appBar: AppBar(
      title: const Text('Responsive UI'),
   ),
   body: LayoutBuilder(
      builder: (BuildContext context, BoxConstraints constraints) {
      if (constraints.maxWidth >= 600) {
         // For Wide Screen (Tablet/Desktop)
         return _buildWideContainers();
      } else {
         // For Narrow Screen (Phone)
         return _buildNarrowContainers();
      }
      },
   ),
   );
}

Widget _buildWideContainers() {
   return Container(
   color: Colors.blue,
   child: const Center(
      child: Text(
      'You are using a Wide screen!',
      style: TextStyle(
```

```
            fontSize: 26, color: Colors.white, fontWeight: FontWeight.bold),
        ),
    ),
    );
}

Widget _buildNarrowContainers() {
    return Container(
    color: Colors.green,
    child: const Center(
        child: Text(
        'You are using a Narrow screen!',
        style: TextStyle(fontSize: 16, color: Colors.white),
        ),
    ),
    );
}
}
```

Output:

Experiment No: 3 b) **Date:**

Aim:

To implement media queries and breakpoints for responsiveness.

Objective:

The objective of this experiment is to understand how to use media queries and breakpoints to create a responsive layout in Flutter.

Code Snippet (main.dart):

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

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

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

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: ResponsiveHomePage(),
    );
  }
}

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

  static const colorCodes = (
    body: Color(0xFFF8E287), // Sweet Corn approx.
    navigation: Color(0XFFC5ECCE), // Padua approx.
    pane: Color(0XFFEEE2BC), // Chamois approx
  );
  static const _style = TextStyle(fontSize: 20, fontWeight: FontWeight.bold);
  static const body = Center(child: Text('Body', style: _style));
  static const navigation = Center(child: Text('Navigation', style: _style));
  static const panes = Center(child: Text('Pane', style: _style));

  @override
  Widget build(BuildContext context) {
    final screenWidth = MediaQuery.of(context).size.width;
    return Scaffold(
      appBar: AppBar(
        title: () {
          // immediately-invoked function expression (IIFE)
          if (screenWidth < 600) {
            return const Text('Responsive UI - Phone');
          } else if (screenWidth < 840) {
            return const Text('Responsive UI - Tablet');
          } else if (screenWidth < 1200) {
            return const Text('Responsive UI - Landscape');
          } else {
            return const Text('Responsive UI - Large Desktop');
          }
        }(),
      ),
```

```
    body: () {
      // immediately-invoked function expression (IIFE)
      if (screenWidth < 600) {    // <-- breakpoint
        return buildCompactScreen();
      } else if (screenWidth < 840) {// <-- breakpoint
        return buildMediumScreen();
      } else if (screenWidth < 1200) {// <-- breakpoint
        return buildExpandedScreen();
      } else {                      // <-- breakpoint
        return buildLargeScreen();
      }
    }(),
  );
}

Widget buildCompactScreen() {
  return Column(
    children: [
      Expanded(
        child: Container(color: colorCodes.body, child: body),
      ),
      Container(height: 80, color: colorCodes.navigation, child: navigation),
    ],
  );
}

Widget buildMediumScreen() {
  return Row(
    children: [
      Container(width: 80, color: colorCodes.navigation),
      Expanded(
        child: Container(color: colorCodes.body, child: body),
      ),
    ],
  );
}

Widget buildExpandedScreen() {
  return Row(
    children: [
      Container(width: 80, color: colorCodes.navigation),
      Container(width: 360, color: colorCodes.body, child: body),
      Expanded(
        child: Container(color: colorCodes.pane, child: panes),
      ),
    ],
  );
}

Widget buildLargeScreen() {
  return Row(
    children: [
      Container(color: colorCodes.navigation, width: 360, child: navigation),
      Container(width: 360, color: colorCodes.body, child: body),
      Expanded(
        child: Container(color: colorCodes.pane, child: panes),
      ),
    ],
  );
}}
```

Output:

Experiment No: 4 a) **Date:**

Aim:

To set up navigation between different screens using Navigator

Objective:

To learn how to navigate between screens in a Flutter app using the Navigator class.

Code Snippet (main.dart):

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

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

classMainApp extends StatelessWidget {
constMainApp({super.key});

  @override
  Widget build(BuildContext context) {
returnconstMaterialApp(
home: HomeScreen(),
    );
  }
}

classHomeScreen extends StatelessWidget {
constHomeScreen({super.key});

  @override
  Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(title: const Text('Home')),
body: Center(
child: ElevatedButton(
onPressed: () {
Navigator.push(
context,
MaterialPageRoute(builder: (context) =>constSecondScreen()),
          );
        },
child: const Text('Go to Second Screen'),
      ),
    ),
   );
  }
}

classSecondScreen extends StatelessWidget {
constSecondScreen({super.key});

  @override
  Widget build(BuildContext context) {
return Scaffold(
appBar: AppBar(title: const Text('Second Screen')),
body: Center(
child: ElevatedButton(
onPressed: () {
```

```
Navigator.pop(context);
          },
child: const Text('Go Back'),
        ),
      ),
    );
  }
}
```

Output:

Experiment No: 4 b) **Date:**

Aim:

To implement navigation with named routes.

Objective:

To understand how to navigate between different screens in a Flutter app using named routes.

Code Snippet (main.dart):

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

void main() {
  runApp(const MainApp());
}

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

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      initialRoute: '/',
      routes: {
        '/': (context) => const HomeScreen(),
        '/second': (context) => const SecondScreen(),
      },
    );
  }
}

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Home')),
      body: Center(
        child: ElevatedButton(
          onPressed: () {
            Navigator.pushNamed(context, '/second');
          },
          child: const Text('Go to Second Screen'),
        ),
      ),
    );
  }
}

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Second Screen')),
      body: Center(
        child: ElevatedButton(
```

```
          onPressed: () {
            Navigator.pop(context);
          },
          child: const Text('Go Back'),
        ),
      ),
    );
  }
}
```

Output:

Experiment No: 5 a) **Date:**

Aim:

To learn about stateful and stateless widgets.

Description:

Stateless Widgets

Stateless widgets are immutable, meaning their properties can't change once they're built. They are typically used for widgets that don't require any mutable state (i.e., they don't change over time or in response to user interactions). Examples of stateless widgets include Text, Icon, and Container.

Code Snippet (main.dart):

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

void main() => runApp(const MaterialApp(home: MyStatelessWidget()));

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Stateless Widget')),
      body: const Center(
        child: Text('Hello, I am a stateless widget!'),
      ),
    );
  }
}
```

Output:

Stateful Widgets

Stateful widgets are dynamic and can change their state during their lifetime. They are used when the widget needs to update based on user interactions or other factors. Examples of stateful widgets include Checkbox, Radio, Slider, and TextField.

Code Snippet (main.dart):

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

void main() => runApp(const MaterialApp(home: MyStatefulWidget()));

class MyStatefulWidget extends StatefulWidget {
  const MyStatefulWidget({super.key});

  @override
  State<MyStatefulWidget> createState() => _MyStatefulWidgetState();
}

class _MyStatefulWidgetState extends State<MyStatefulWidget> {
  int _counter = 0;

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

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('Stateful Widget')),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            const Text('You have pushed the button this many times:'),
            Text('$_counter', style: Theme.of(context).textTheme.displayLarge),
          ],
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: const Icon(Icons.add),
      ),
    );
  }
}
```

Output:

Experiment No: 5 b) **Date:**

Aim:

To implement state management using set State and Provider.

Objective:

To understand how to manage the state of a Flutter app using the setState method and the Provider package.

Procedure :

1. Create a new Flutter project by running the following command in your terminal:
   ```
   flutter create my_flutter_app
   ```
   The command creates a Flutter project directory called my_flutter_app that contains a simple demo app that uses Material Components.
2. Change to the Flutter project directory.
   ```
   cd my_flutter_app
   ```
3. Create a new Dart file called counter_model.dart in the lib directory of your Flutter project.
   ```
   touch lib/counter_model.dart
   ```
4. Open the pubspec.yaml file in your Flutter project and add the following lines after flutter: to include the image asset in your project:
   ```
   dependencies:
   flutter:
      sdk: flutter
   provider: ^6.0.0 <-- Add this line
   ```
   Save the file.
5. Add the following code snippet to the counter_model.dart file:
   ```
   import 'package:flutter/foundation.dart';

   // CounterModel class that extends ChangeNotifier
   class CounterModel extends ChangeNotifier {
     int _counter = 0;

     // Getter for the counter value
     int get counter => _counter;

     // Method to increment the counter and notify listeners
     void increment() {
       _counter++;
       notifyListeners();
     }
   }
   ```
   Save the file.
6. Open the lib/main.dart file in your Flutter project.
7. Replace the existing code with the following code snippet:
   ```
   // Import necessary packages
   import 'package:flutter/material.dart';
   import 'package:provider/provider.dart';
   import 'counter_model.dart';

   // Main function
   void main() {
     runApp(
       // Wrap the app with ChangeNotifierProvider to provide the CounterModel
       ChangeNotifierProvider(
         create: (context) => CounterModel(),
         child: const MyApp(),
       ),
     );
   }
   ```

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

  @override
  Widget build(BuildContext context) {
    return const MaterialApp(
      home: CounterPage(),
    );
  }
}

// CounterPage widget
class CounterPage extends StatelessWidget {
  const CounterPage({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text('Counter App'),
      ),
      body: Center(
        child: Consumer<CounterModel>(
          builder: (context, counterModel, child) {
            return Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                const Text('You have pushed the button this many times:'),
                Text(
                  '${counterModel.counter}',
                  style: Theme.of(context).textTheme.displayLarge,
                ),
              ],
            );
          },
        ),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: () {
          // Call the increment method of CounterModel using Provider
          Provider.of<CounterModel>(context, listen: false).increment();
        },
        tooltip: 'Increment',
        child: const Icon(Icons.add),
      ),
    );
  }
}
```

8. Save the file.
9. Run your Flutter project using the following command:

```
flutter run
```

   Select the appropriate device to run the app.
10. During the app execution, you can use the following commands:
    - Enter r to hot reload the app and see the changes you made to the code.
    - Enter q to quit the app.

Output: