**Flutter – Splitting App into Widgets**

One of the key features of Flutter is its ability to easily split an app into smaller, more manageable widgets.

When building an app in Flutter, it's important to think about the structure of the app and how the different widgets interact. As the app grows in complexity, it can become more difficult to manage the code and make changes. One way to combat this is by splitting the app into smaller widgets.

There are several reasons to split an app into widgets in Flutter:

* Large Widgets: Whenever we have a large widget with many nested widgets and other smaller widgets inside of it, it's advisable to split the large widgets into smaller widgets that can be managed easily.
* Repeating Widgets: Sometimes we have a situation where we create some custom widget (e.g. custom Button) which we might use at many places, then it is a good practice to split the piece of code into a separate widget. This helps us clean up extra lines of code and manage the changes to the widget in one place.
* Complex Widget: There are situations when the widget might not be too large, but because of the implementation it becomes complex so to implement the complex parts separately for better understanding we might split the codes into different widgets.
* Improve Performance: Splitting widgets can also improve the performance of the app by reducing the rebuilds of the widgets.

Methods of splitting widgets in Flutter:

Create separate widgets for each component and then using them together in the main widget.

Extract the common functionality of a widget into a separate widget and then using it in multiple places.

StatelessWidget and StatefulWidget: Use 'StatelessWidget' and 'StatefulWidget' to split widgets. StatelessWidget are widgets that will not change over time, while StatefulWidget are widgets that can change over time. This can help to keep the code more organized and efficient.

When splitting widgets, it's important to consider the state management of the widgets, the lifecycle of the widgets and how they interact with each other. It's also important to have a clear understanding of the structure of the app and how the different widgets interact before making any changes.

Overall, splitting widgets in Flutter can be a powerful technique for making an app more manageable, but it's important to do it thoughtfully and with the right approach. It requires a good understanding of the app and its structure, as well as the principles of good software design.

In conclusion, splitting app into widgets is a powerful technique in Flutter, it can make the code more manageable, and improve the performance of the app. Careful consideration of the state management, the lifecycle of the widgets and the interaction between them is crucial while splitting widgets. With the right approach, it can make the development process more efficient and the final product more user-friendly.

Let see with an Example :

Here is an example of a Basic Calculator app that takes two numbers as input from the user and produces an output as the user clicks on any of the addition, subtraction, etc. buttons.

main.dart

import 'package:flutter/material.dart';

import 'home\_page.dart';

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(

title: 'Tutorials Point',

theme: ThemeData(

primarySwatch: Colors.blue,

),

home: const HomePage(),

);

}

}

The above lines show the main screen of the app, Where we call HomePage

home\_page.dart

import 'package:flutter/material.dart';

import 'custom\_button.dart';

import 'custom\_textfield.dart';

class HomePage extends StatefulWidget {

const HomePage({Key? key}) : super(key: key);

@override

State<HomePage> createState() => \_HomePageState();

}

class \_HomePageState extends State<HomePage> {

late double num1, num2, output = 0;

final String add = '+';

final String sub = '-';

final String mul = '×';

final String div = '÷';

String? selectedOperator;

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

title: const Text('Tutorials Point'),

),

body: Container(

padding: const EdgeInsets.all(16),

child: Column(

children: [

const Text(

'Calculator',

style: TextStyle(fontSize: 24),

),

const SizedBox(height: 16),

Row(

children: [

CustomTextField(

hint: '1st Number',

onChanged: (value) {

num1 = double.parse(value ?? '0');

},

),

const SizedBox(width: 8),

CustomTextField(

hint: '2nd Number',

onChanged: (value) {

num2 = double.parse(value ?? '0');

},

),

],

),

const SizedBox(height: 8),

Row(

mainAxisAlignment: MainAxisAlignment.spaceEvenly,

children: [

CustomButton(

selectedOperator: selectedOperator,

symbol: add,

onTap: () {

setState(() {

selectedOperator = add;

output = (num1 + num2);

});

},

),

CustomButton(

selectedOperator: selectedOperator,

symbol: sub,

onTap: () {

setState(() {

selectedOperator = sub;

output = (num1 - num2);

});

},

),

CustomButton(

selectedOperator: selectedOperator,

symbol: mul,

onTap: () {

setState(() {

selectedOperator = mul;

output = (num1 \* num2);

});

},

),

CustomButton(

selectedOperator: selectedOperator,

symbol: div,

onTap: () {

setState(

() {

selectedOperator = div;

if (num2 == 0) {

output = -1;

} else {

output = (num1 / num2).toDouble();

}

},

);

},

),

],

),

const SizedBox(height: 10),

const Text(

'Output',

style: TextStyle(fontSize: 18),

),

const SizedBox(height: 5),

Container(

padding: const EdgeInsets.all(8),

width: double.infinity,

height: 50,

decoration: BoxDecoration(

color: Colors.black12,

borderRadius: BorderRadius.circular(16),

),

child: Center(

child: Text(

output.toStringAsFixed(2),

style: const TextStyle(fontSize: 24),

),

),

)

],

),

),

);

}

}

HomePage Widget comprises several in-built widgets and several custom widgets (made from in-built widgets). These widgets are below.

custom\_textfield.dart

// This is a Custom TextField Widget

import 'package:flutter/material.dart';

class CustomTextField extends StatelessWidget {

const CustomTextField({

Key? key,

required this.onChanged,

required this.hint,

}) : super(key: key);

final Function(String?) onChanged;

final String hint;

@override

Widget build(BuildContext context) {

return Expanded(

child: TextField(

onChanged: onChanged,

style: const TextStyle(fontSize: 18),

decoration: InputDecoration(

contentPadding: const EdgeInsets.symmetric(horizontal: 16),

filled: true,

fillColor: Colors.black12,

border: OutlineInputBorder(

borderSide: BorderSide.none,

borderRadius: BorderRadius.circular(18),

),

hintText: hint,

helperText: 'Keep it Short',

),

),

);

}

}

The First Custom Widget is the CustomTextField which is extracted using the above methods and some fields are added in the constructor for better usage and less code.

custom\_button.dart

//This is a Custom Button Widget.

import 'package:flutter/material.dart';

class CustomButton extends StatelessWidget {

const CustomButton({

Key? key,

required this.selectedOperator,

required this.onTap,

required this.symbol,

}) : super(key: key);

final String? selectedOperator;

final String symbol;

final Function() onTap;

@override

Widget build(BuildContext context) {

return GestureDetector(

onTap: onTap,

child: Container(

height: 64,

width: 64,

decoration: BoxDecoration(

borderRadius: const BorderRadius.all(Radius.circular(12)),

border: Border.all(color: Colors.blueGrey),

color: selectedOperator == symbol ? Colors.blueGrey : Colors.white,

),

child: Center(

child: Text(

symbol,

style: TextStyle(

fontSize: 32,

fontWeight: FontWeight.bold,

color:

selectedOperator == symbol ? Colors.white : Colors.blueGrey,

),

),

),

),

);

}

}

The Second Custom Widget Used here is CustomButton which is made using the Gesture Detector.

Output:

