Unit 4

Introduction to Widget Tree

In Flutter, a **widget tree** is the hierarchical structure of all the widgets that make up your app's UI. Every Flutter app consists of a series of widgets nested inside other widgets, and this nesting forms a tree-like structure, known as the widget tree. The root of the widget tree is usually the `MaterialApp` or `CupertinoApp` widget, and every widget inside it is a branch of this tree.

Key Points:

- **Everything is a Widget**: In Flutter, everything you see on the screen, such as text, buttons, layouts, and even the entire app itself, is a widget.
- **Parent-Child Relationships**: In the widget tree, each widget can have one or more child widgets. For example, a `Column` widget can contain multiple `Text` or `Button` widgets as its children.
- **Nested Structure**: The widget tree is formed by nesting widgets inside each other. This means each widget can act as a parent to other widgets, creating a hierarchical structure.

Widget Tree Structure Example

Here is a simple example of a widget tree for a basic app:

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Widget Tree Example'),
```

```
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: <Widget>[
              Text('Hello, World!'),
              ElevatedButton(
                onPressed: () {},
                child: Text('Click Me'),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Breakdown of the Widget Tree:

1. `MaterialApp`: The root of the widget tree, which defines the app.
2. `Scaffold`: A widget that provides the structure of the visual interface (e.g., app bar, body).
3. `AppBar`: A widget that sits inside the `Scaffold` and represents the app's header.
4. `Center`: A layout widget that centers its child widget on the screen.
5. `Column`: A layout widget that arranges its children in a vertical list.
6. `Text`: A widget that displays a string of text.
7. `ElevatedButton`: A button widget that can be clicked.

In this example, the widget tree would look something like this:

```
MaterialApp
|
└── Scaffold
    |
    ├── AppBar
```

```
    │   └── Text ('Widget Tree Example')
    │
    └── Center
        │
        └── Column
            ├── Text ('Hello, World!')
            └── ElevatedButton
                └── Text ('Click Me')
```

Types of Widgets in the Tree:

- **Leaf Widgets**: These are the widgets at the ends of the branches in the tree, meaning they do not have any child widgets. For example, `Text` and `ElevatedButton` are leaf widgets.
- **Container Widgets**: These are widgets that contain other widgets. For example, `Column`, `Center`, and `Scaffold` are container widgets.

Importance of the Widget Tree:

1. **Rendering**: Flutter uses the widget tree to render the UI on the screen. When a widget is added, modified, or removed, Flutter rebuilds the widget tree to reflect those changes.
2. **UI Organization**: The widget tree helps organize the structure and layout of an app. It defines how widgets relate to each other and how they are positioned.
3. **State Management**: State changes (in `StatefulWidgets`) trigger the widget tree to rebuild parts of the tree where the state has changed, making the UI dynamic and responsive.

Widget Trees and Performance:

- **Deep Widget Trees**: A very deep widget tree can sometimes lead to performance issues if not properly managed. Flutter's optimizations, such as the use of `const` widgets and efficient tree rebuilding, help mitigate these issues.
- **Rebuilding the Tree**: When the app state changes, only the parts of the widget tree that are affected are rebuilt. This makes Flutter efficient in managing updates to the UI.

In summary, the **widget tree** is the fundamental concept behind how Flutter organizes and renders its UI. Understanding and managing the widget tree is crucial to building efficient and well-structured Flutter apps.

Material Design Vs. Cupertino

Here's a comparison of the `MaterialApp` and `CupertinoApp` widgets in Flutter, detailing their features, characteristics, and usage. Both serve as the entry point for apps but are tailored for different design paradigies.

Feature	MaterialApp	CupertinoApp
Purpose	To provide Material Design widgets and themes.	To provide Cupertino (iOS) style widgets and themes.
Main Class	`MaterialApp`	`CupertinoApp`
Default Style	Material Design (Android)	iOS-style design
Navigation	Uses `Navigator` and `MaterialPageRoute` for routing.	Uses `CupertinoPageRoute` for navigation.
App Bar	`AppBar` provides a standard app bar with actions.	`CupertinoNavigationBar` for an iOS-style navigation bar.
Buttons	`ElevatedButton`, `TextButton`, etc.	`CupertinoButton`, which mimics iOS button styles.
Themes	Customizable themes via `ThemeData`.	Customizable themes via `CupertinoThemeData`.
Dialogs	`showDialog`, `AlertDialog` for material alerts.	`CupertinoAlertDialog` for iOS-styled alerts.
Widgets	Extensive set of widgets like `Card`, `FloatingActionButton`, etc.	Limited set of widgets focusing on iOS styles, like `CupertinoSwitch`, `CupertinoPicker`.
Animation Style	Material animations with various transitions.	iOS-style animations and transitions.
Scroll Behavior	Uses `MaterialScrollBehavior` for material-style scrolling.	Uses `CupertinoScrollBehavior` for iOS-style scrolling.
Platform Awareness	Primarily designed for Android but can be used on iOS.	Primarily designed for iOS but can be used on Android.

Common Widgets in MaterialApp and CupertinoApp

Here's a comparison of the `MaterialApp` and `CupertinoApp` widgets in Flutter, detailing their features, characteristics, and usage. Both serve as the entry point for apps but are tailored for different design paradigies.

Here's a list of some common widgets, along with brief descriptions of their purpose:

Widget	Description
Container	A box model that can contain a single widget, apply padding, margins, and decoration.
Row	A widget that arranges its children in a horizontal line.
Column	A widget that arranges its children in a vertical line.
Stack	A widget that overlays its children on top of each other.
Text	Displays a string of text with various styles and formats.
Image	Displays an image from various sources (asset, network, file).
Icon	Displays a material or Cupertino icon.
Button	The base class for buttons; can be used with `ElevatedButton` or `CupertinoButton`.
Switch	A toggle switch that can be on or off, providing a way for users to change a setting.
Checkbox	A box that can be checked or unchecked to indicate a boolean value.
TextField	A field for entering text input.
ListView	A scrollable list of widgets that can be displayed vertically.
AppBar	A material app bar (`AppBar`) or an iOS-style navigation bar (`CupertinoNavigationBar`).
Card	A material design card that can display content and actions.
Divider	A horizontal line that can be used to separate content.
SnackBar	A lightweight message with an optional action, used for temporary notifications.
Dialog	Can be created as a `AlertDialog` in Material or `CupertinoAlertDialog` in Cupertino.
Progress Indicator	A widget that shows the progress of a task (circular or linear).

Tooltip	A message that appears when the user long-presses a widget.

TextField Widget

The `TextField` widget is used for receiving text input from the user. It allows for various configurations, such as hint text, label text, and input validation.

Key Features:

- **Input Types**: Supports various input types (e.g., text, email, password).
- **Decoration**: You can customize the appearance with styles, borders, and hints.
- **Controller**: A `TextEditingController` can be used to retrieve the input text.

Example of TextField

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('TextField Example')),
        body: Padding(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            children: [
              TextField(
                decoration: InputDecoration(
                  border: OutlineInputBorder(),
                  labelText: 'Enter your text',
                  hintText: 'Type something...',
                ),
```

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

FloatingActionButton Widget

The `FloatingActionButton` (FAB) is a circular button that floats above the content of an application. It's commonly used for the primary action in an app, such as adding new items.

Key Features:

- **Icon**: You can place an icon inside the button to indicate its function.
- **Location**: The FAB typically floats in the bottom corner of the screen.
- **Customization**: You can customize the background color, size, and shape.

Example of FloatingActionButton

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('FloatingActionButton Ex
ample')),
        body: Center(
```

```
          child: Text('Press the button below!'),
        ),
        floatingActionButton: FloatingActionButton(
          onPressed: () {
            // Action when the button is pressed
            print('FAB Pressed!');
          },
          child: Icon(Icons.add),
          tooltip: 'Add Item',
        ),
      ),
    );
  }
}
```

In Flutter, `RaisedButton` and `FlatButton` are widgets that provide different styles of buttons. However, it's important to note that as of Flutter 2.0, these buttons have been deprecated in favor of `ElevatedButton` and `TextButton`, respectively. Here's a brief overview and examples of both:

RaisedButton (Now ElevatedButton)

`RaisedButton` is used to create a button that appears to "float" above the content. It typically has a shadow and is used for the primary actions in an application.

Example of ElevatedButton

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Raised Button Exampl
```

```
e')),
        body: Center(
          child: ElevatedButton(
            onPressed: () {
              print('RaisedButton Pressed!');
            },
            child: Text('Click Me'),
          ),
        ),
      ),
    );
  }
}
```

FlatButton (Now TextButton)

`FlatButton` is a button without elevation or background color. It is generally used for actions that are not the primary focus, such as links or secondary actions.

Example of TextButton

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Flat Button Example')),
        body: Center(
          child: TextButton(
            onPressed: () {
              print('FlatButton Pressed!');
            },
            child: Text('Click Me'),
```

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

Summary

- **ElevatedButton** (formerly `RaisedButton`): Used for primary actions with an elevated appearance.
- **TextButton** (formerly `FlatButton`): A simple, flat button for secondary actions.

Note

It's recommended to use `ElevatedButton` and `TextButton` moving forward as they provide a more consistent and modern approach to button design in Flutter.

IconButton

The `IconButton` widget is a clickable button that displays an icon. It's often used for actions that require minimal space, such as a settings or favorite button.

Key Features:

- **Icon**: Displays a single icon.
- **OnPressed**: A callback that is triggered when the button is pressed.
- **Customization**: You can customize its size, color, and other visual aspects.

Example of IconButton

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

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

class MyApp extends StatelessWidget {
```

```
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('IconButton Example')),
        body: Center(
          child: IconButton(
            icon: Icon(Icons.favorite),
            color: Colors.red,
            iconSize: 50,
            onPressed: () {
              print('IconButton Pressed!');
            },
          ),
        ),
      ),
    );
  }
}
```

PopupMenuButton

The `PopupMenuButton` widget is used to display a menu when the user taps on it. It's commonly used for presenting a list of options or actions.

Key Features:

- **Item Builder**: A builder function that creates the menu items.
- **OnSelected**: A callback that is triggered when an item is selected.
- **Customizable**: You can customize the appearance of the menu and its items.

Example of PopupMenuButton

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

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

```
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('PopupMenuButton')),
        body: Center(
          child: PopupMenuButton<String>(
            onSelected: (value) => print('Selected: $valu
e'),
            itemBuilder: (_) => [
              PopupMenuItem(value: 'Option 1', child: Text
('Option 1')),
              PopupMenuItem(value: 'Option 2', child: Text
('Option 2')),
              PopupMenuItem(value: 'Option 3', child: Text
('Option 3')),
            ],
            child: Icon(Icons.more_vert),
          ),
        ),
      ),
    );
  }
}
```

Summary

- `IconButton`: A button that displays an icon, often used for actions with minimal space.
- `PopupMenuButton`: A button that displays a menu of options when tapped, useful for presenting a list of actions.

These widgets are essential for creating interactive and user-friendly interfaces in Flutter applications!

ButtonBar

The `ButtonBar` widget is a horizontal arrangement of buttons, often used to group related actions together. It helps in organizing the layout of buttons, providing spacing and alignment options.

Key Features:

- **Alignment**: You can align the buttons to the start, center, or end.
- **Spacing**: Provides spacing between buttons automatically.
- **Button Type**: Typically contains `TextButton`, `ElevatedButton`, or `FlatButton` (now `TextButton` and `ElevatedButton`).

Example of ButtonBar

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('ButtonBar Example')),
        body: Center(
          child: ButtonBar(
            alignment: MainAxisAlignment.center,
            children: [
              ElevatedButton(
                onPressed: () {
                  print('First Button Pressed!');
                },
                child: Text('Button 1'),
              ),
              ElevatedButton(
                onPressed: () {
```

```
                  print('Second Button Pressed!');
                },
                child: Text('Button 2'),
              ),
            ],
          ),
        ),
      ),
    );
  }
}
```

Image

The `Image` widget is used to display images in your Flutter application. It can load images from various sources, such as asset files, network URLs, or local files.

Key Features:

- **Image Source**: Supports `AssetImage`, `NetworkImage`, and `FileImage`.
- **Fit**: You can specify how the image should be fitted within its container (e.g., `BoxFit.cover`, `BoxFit.contain`).
- **Width & Height**: You can set dimensions to control the image size.

Example of Image

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Image Example')),
```

```
        body: Center(
          child: Image.network(
            '<https://via.placeholder.com/150>', // Sample
image URL
            width: 150,
            height: 150,
            fit: BoxFit.cover,
          ),
        ),
      ),
    );
  }
}
```

Summary

- `ButtonBar`: A widget that arranges buttons horizontally, helping to organize related actions.
- `Image`: A versatile widget for displaying images from various sources, with options for fitting and sizing.

FLUTTER ICON

In Flutter, the `Icon` widget is used to display icons on the screen. Icons are typically used for visual representation of actions, features, or content, enhancing the user interface and user experience.

Key Features of Icon Widget

- **Icon Data**: The icon is created using the `Icons` class, which contains a vast collection of material icons.
- **Size and Color**: You can customize the size and color of the icon.
- **Semantic Label**: Useful for accessibility; you can provide a description for screen readers.

Small Icon Example

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

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

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(title: Text('Small Icon Example')),
        body: Center(
          child: Icon(
            Icons.star,        // Using the star icon from
the Icons class
            color: Colors.yellow, // Setting the color to y
ellow
            size: 30,           // Setting a small size for
the icon
          ),
        ),
      ),
    );
  }
}
```

Explanation

- `Icons.star`: The icon being used is a star from the built-in `Icons` class.
- **Color**: The color is set to yellow.
- **Size**: The size is set to `30`, making it a small icon.
- **Center**: The icon is placed in the center of the screen using a `Center` widget.