Stateful and Stateless Widgets

Widgets in Flutter

- Widgets are the building blocks of any Flutter app.
- Everything in Flutter is a widget: UI elements, structure, layout, etc.
- Widgets describe what their view should look like given their current configuration and state.

StatelessWidget

- `StatelessWidget` is a widget that represents part of the user interface, **but** do not have any changeable state.
- It represents parts of the UI that are **static** once created.
- Ideal for displaying static content.

StatefulWidget

- `StatefulWidget` has a **mutable state** that can change based on user input, animations, or events.
- Can be updated **dynamically** (e.g., forms, interactive UI elements).
- Requires a State object to manage changes.
- Built using `StatefulWidget` and a `State` class.

StatefulWidget

```
class MyStatefulWidget extends StatefulWidget {
 @override
 _MyStatefulWidgetState  createState() => _MyStatefulWidgetState();
}
class _MyStatefulWidgetState extends State<MyStatefulWidget> {
 int counter = 0;
 void incrementCounter() {
   setState(() {
     counter++;
   }); }
 @override
 Widget build(BuildContext context) {
   return Column(
     children: [
       Text('Counter: $counter'),
       ElevatedButton(
         onPressed: incrementCounter,
         child: Text('Increment'),
),],); }}
```

State Management in Stateful Widgets

- `setState()`: Manages the state, it triggers a rebuild of the widget tree.
- `initState()`: Called once when the widget is first inserted into the widget tree.

Stateful vs. Stateless Widgets

- **`StatelessWidget`**: Used when the widget's UI is static and does not depend on any internal state (e.g., displaying a title,
static lists).

- **`StatefulWidget`**: Used when the widget needs to update dynamically based on user interaction, animations, or API calls (e.g., form fields, counters).

Stateful vs. Stateless Widgets

- **`StatelessWidget`** are more performant because they don’t require state management.
- **`StatefulWidget`**: Tries to manage the state properly by using `setState` efficiently to maintain performance.
- **Best Practice**: Keep widgets as stateless as possible and only use stateful widgets when necessary.

ElevatedButton

It displays a child widget & triggers an action when pressed.

```
ElevatedButton(
        onPressed: () {
          // Define action
        },
        child: Text('Click here),
      )
```

- `onPressed`: The action or function that gets called when the button is tapped.
- `child`: widget displayed on the button (Text, Icon, etc.).

ElevatedButton Style

```
ElevatedButton(
        onPressed: () {},
        style: ElevatedButton.styleFrom(
          backgroundColor: Colors.blue,
          textStyle: TextStyle(fontSize: 20),
        ),
        child: Text('Customized Button'),)
```

- Refer to ElevatedButton documentation for more styling.

ElevatedButton States (Active vs. Disabled)

- An ElevatedButton can be **disabled** by setting `onPressed` to `null`.
-

```
ElevatedButton(
        onPressed: null, // Disabled button
        child: Text('Disabled'),
      )
```

GestureDetector

- `GestureDetector` is a widget that detects various gestures form users to interact with the app such as taps, swipes, and long presses, etc.
- Essential for creating responsive and user-friendly mobile interfaces.

GestureDetector

- Syntax:

```
GestureDetector(
        onTap: () {
          print("Widget tapped!");
        },
        child: Text('Tap Here!'),
      )
```