Name: Obafemi Oluwatodunni Ademilogo
Matric Number: 22/10MSS013
Degree Program: BSc. Software Engineering

November 17, 2024

Assignment: SWE413

Question: Write extensively on Stateless and Stateful Widgets in Flutter

In Flutter, widgets are the fundamental building blocks of user interfaces. They are responsible for defining the structure, style, and behavior of UI elements. To create dynamic and interactive user experiences, Flutter provides two primary types of widgets: **Stateful** and **Stateless**.

1. Stateless Widgets

A Stateless Widget is one that does not have any mutable state that it needs to track. The only area of focus of a stateless widget is the information displayed and the user interface. They deal with situations that are independent of the user's input. A Stateless Widget does not tell the framework when to remove it or rebuild it, it gets a command from the framework itself. They create user interfaces that do not change dynamically upon updation in the nearby values. We use a stateless widget when we create an application that does not require redrawing a widget again and again.
The following are properties of a Stateless Widget:

- **Immutable:** Once created, their properties cannot change. This makes them ideal for static UI elements that don't require dynamic updates.
- **Lightweight and Efficient:** They have a lower memory footprint and are faster to render, making them suitable for performance-critical scenarios.
- **Simple to Use:** Their simplicity reduces the complexity of managing state and lifecycle methods.

Common Use Cases for Stateless Widgets:

- **Static UI Elements:** Text, images, icons, and simple layouts like rows and columns that don't change over time.
- **Reusable Components:** Creating reusable UI components that don't require internal state management.
- **Performance Optimization:** Using stateless widgets for parts of the UI that don't need to be updated frequently can improve performance.

Below is an example of the structure of a Stateless Widget:

```
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```

2. Stateful Widgets

A Stateful Widget has its own mutable state that it needs to track. It is modified according to the user's input. A Stateful Widget looks after two things primarily, the changed state based on its previous state and an updated view of the user interface. A track of the previous state value has to be looked at because there is a need to self-rebuild the widget to show the new changes made to your application. A Stateful Widget triggers a build method for creating its children widgets and the subclass of the state holds the related data. It is often used in cases where redrawing of a widget is needed.
The following are the properties of a Stateful Widget:

- **Mutable:** Their properties can change, triggering UI updates. This enables dynamic and interactive UI experiences.
- **Complex:** They require careful management of state and lifecycle methods to ensure correct behavior and avoid performance issues.
- **Powerful:** They allow for complex UI interactions, animations, and data-driven updates.

Common Use Cases for Stateful Widgets:

- **User Interactions:** Handling user input, such as button clicks, text input, and gestures.
- **Data-Driven UI:** Displaying data that changes over time, like fetching data from APIs or user preferences.
- **Animations and Transitions:** Creating smooth and visually appealing animations and transitions.
- **Forms:** Building complex forms with validation and error handling.

Below is an example of the structure of a Stateful Widget:

```
class MyApp extends StatefulWidget {
  const MyApp({Key? key}) : super(key: key);

  @override
  State<MyApp> createState() => _MyAppState();
}

class _MyAppState extends State<MyApp> {
  @override
  Widget build(BuildContext context) {
    return Container();
  }
}
```