**Chapter 4**

In Chapter 4, I learned how to differentiate between **StatelessWidget** and **StatefulWidget**, which are the backbone of Flutter's UI design. Stateless widgets, such as Text or Icon, are immutable and do not change during runtime. I learned how to create a custom stateless widget by overriding the build method to return a widget tree. On the other hand, stateful widgets allow dynamic updates through their State object, which I explored using examples like toggling a button’s appearance. This chapter also emphasized the importance of the setState() method for updating the UI when the state changes. Understanding this distinction helped me grasp how to manage static and dynamic content effectively.

I also learned about lifecycle methods in stateful widgets, which control how widgets behave during creation, updates, and destruction. The initState() method initializes data when the widget is created, while the build() method reconstructs the widget tree whenever the state changes. I explored the dispose() method to clean up resources, such as controllers, when a widget is removed from the tree. These lifecycle methods demonstrated how to manage stateful widgets efficiently and avoid memory leaks. Additionally, the chapter introduced concepts like local and global state management, preparing me for more advanced scenarios.

Finally, I learned how to implement interactivity using stateful widgets. By creating features like counters and toggling switches, I practiced capturing user input and reflecting it in the UI. I also learned to handle user interactions using widgets like GestureDetector and InkWell to add custom touch behavior. The chapter highlighted Flutter’s flexibility in handling complex interactive UIs by combining stateless and stateful widgets. By the end of Chapter 4, I felt confident in creating dynamic, interactive apps and managing the widget lifecycle effectively.