Bloc State Management

- Bloc is a state management library for Dart and Flutter that provides an easy and intuitive way to manage application state. It uses the concept of reactive programming to handle events and actions in a predictable and scalable manner.
- In Bloc, the state of an application is represented by a stream of data that can be consumed by widgets or other parts of the application. The state can be changed by dispatching events, which are then processed by the bloc and used to generate new states.
- One of the key benefits of using Bloc is its separation of concerns. By separating business logic from presentation, Bloc allows developers to create more modular and testable code. Additionally, the use of streams and reactive programming makes it easier to handle asynchronous operations and prevent common problems like race conditions and invalid state.
- Overall, Bloc is a powerful tool for managing application state in Dart and Flutter projects, and it's worth exploring if you're looking for a more organized and scalable approach to state management.

Code Explanation

- I created a dart file **states.dart** that contains our states, and create another dart file **cubit.dart** that contains the cubit.
- I created a class **CounterCubit** that extends **Cubit** to write our logic inside it.

- **Cubit** class needs a generic type, so we can only identify it with one, and that's where we define our states class, but at the same time we have more than one state and we can't pass them all to the **Cubit**, therefore we define the **CounterStates** class as an **abstract** class, then we define other classes and make each one of them **extends** it.
- In the following line, we define the starting point of the states which is **CounterInitialState()**.

```
CounterCubit() : super(CounterInitialState());
```

- I defined a method **getObject()** to help me use an object from my cubit where I want anywhere in the whole project.
- I used the widget **BlocProvider()** to create our cubit **CounterCubit()**.
- I used the widget **BlocConsumer()** to identify the **Cubit** and the **states** it works on. It takes two generic types, the first is the cubit (**CounterCubit**) we work on and rebuild when we call some state, and the second is the states (**CounterStates**) it listens to.
- **listener** attribute takes two parameters, the first is a **context** and the second is **state**, this state is the state which we call to make it listen to it.
- **builder** attribute also takes two parameters, the first is, **context** and the second is **state**. It rebuilds the **UI** according to the **state** it has, which comes from **listener** attribute.
- I used **emit()** function to change the current state to this state.
- I used **MyBlocObserver** to help me observing the changing among the states.