Bloc

What is Bloc and How it works?

- In Flutter, a **Bloc** (short for Business Logic Component) is a design pattern that helps manage state in your application in a predictable and efficient way. It's commonly used to separate the presentation layer from the business logic layer. Here's a brief overview of how Bloc works in Flutter:

1- State Management:

- Bloc helps manage the state of your application. Instead of managing state directly in your widgets, you delegate this responsibility to the Bloc.

2- Events and States:

- Bloc operates based on events and states. An event is a signal that something has happened (e.g., a button click), and a state represents the condition of your application at a specific point in time.

3- Bloc Class:

- Typically, you create a class that extends the Bloc or Cubit class from the bloc package. This class will handle incoming events, update the state, and notify the UI.

```
class MyBloc extends Bloc<MyEvent, MyState> {
  // Logic for handling events and updating state
}
```

4- Events and States Classes:

- You'll define classes for events and states. Events are triggered by user actions or other inputs, and states represent the different conditions your UI can be in.

```
// Define events
abstract class MyEvent {}

// Define states
abstract class MyState {}
```

5- State Changes

- Bloc requires us to register event handlers via the on API, An event handler is responsible for converting any incoming events into zero or more outgoing states.

```
sealed class CounterEvent {}

final class CounterIncrementPressed extends CounterEvent {}

class CounterBloc extends Bloc<CounterEvent, int> {
  CounterBloc() : super(0) {
    on<CounterIncrementPressed>((event, emit) {
      // handle incoming `CounterIncrementPressed` event
    });
  }
}
```

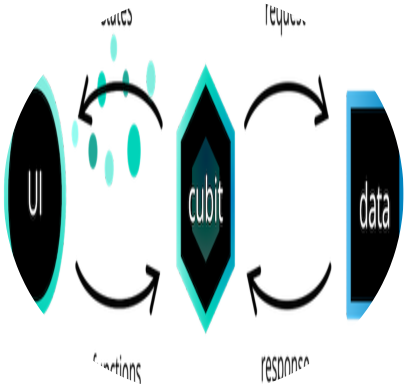
6-UI Integration

- Connect your Bloc to your UI using widgets like **BlocBuilder** or **BlocListener**. These widgets rebuild parts of your UI in response to changes in the Bloc's state.

```
BlocBuilder<MyBloc, MyState>(
  builder: (context, state) {
    // Rebuild UI based on the current state
    return YourWidget(state.data);
  },
)
```

Cubit

- A Cubit is a class which extends BlocBase and can be extended to manage any type of state.



- A Cubit can expose functions which can be invoked to trigger state changes. - States are the output of a Cubit and represent a part of your application's state. UI components can be notified of states and redraw portions of themselves based on the current state.

BlocBuilder

- BlocBuilder is a Flutter widget which requires a Bloc and a builder function. BlocBuilder handles building the widget in response to new states. BlocBuilder is very similar to StreamBuilder but has a more simple API to reduce the amount of boilerplate code needed. The builder function will potentially be called many times and should be a pure function that returns a widget in response to the state.

```
BlocBuilder<BlocA, BlocAState>(
  builder: (context, state) {
    // return widget here based on BlocA's state
  }
)
```

- For fine-grained control over when the builder function is called an optional buildWhen can be provided. buildWhen takes the previous bloc state and current bloc state and returns a boolean. If buildWhen returns true, builder will be called with state and the widget will rebuild. If buildWhen returns false, builder will not be called with state and no rebuild will occur.

```
BlocBuilder<BlocA, BlocAState>(
  buildWhen: (previousState, state) {
    // return true/false to determine whether or not
    // to rebuild the widget with state
  },
  builder: (context, state) {
    // return widget here based on BlocA's state
  }
)
```

BlocSelector

- BlocSelector is a Flutter widget which is analogous to BlocBuilder but allows developers to filter updates by selecting a new value based on the current bloc state. Unnecessary builds are prevented if the selected value does not change. The selected value must be immutable in order for BlocSelector to accurately determine whether builder should be called again.
- If the bloc parameter is omitted, BlocSelector will automatically perform a lookup using BlocProvider and the current BuildContext.

```
BlocSelector<BlocA, BlocAState, SelectedState>(
  selector: (state) {
    // return selected state based on the provided state.
  },
  builder: (context, state) {
    // return widget here based on the selected state.
  },
)
```

BlocProvider

- BlocProvider is a Flutter widget which provides a bloc to its children via BlocProvider.of(context). It is used as a dependency injection (DI) widget so that a single instance of a bloc can be provided to multiple widgets within a subtree.
- In most cases, BlocProvider should be used to create new blocs which will be made available to the rest of the subtree. In this case, since BlocProvider is responsible for creating the bloc, it will automatically handle closing the bloc.

```
BlocProvider(
  create: (BuildContext context) => BlocA(),
  child: ChildA(),
);
```

- By default, BlocProvider will create the bloc lazily, meaning create will get executed when the bloc is looked up via BlocProvider.of(context).
- To override this behavior and force create to be run immediately, lazy can be set to false.

```
BlocProvider(
  lazy: false,
  create: (BuildContext context) => BlocA(),
  child: ChildA(),
);
```

MultiBlocProvider.

- MultiBlocProvider is a Flutter widget that merges multiple BlocProvider widgets into one. MultiBlocProvider improves the readability and eliminates the need to nest multiple BlocProviders. By using MultiBlocProvider we can go from:

```
BlocProvider<BlocA>(
  create: (BuildContext context) => BlocA(),
  child: BlocProvider<BlocB>(
    create: (BuildContext context) => BlocB(),
    child: BlocProvider<BlocC>(
      create: (BuildContext context) => BlocC(),
      child: ChildA(),
    )
  )
)
```

-to

```
MultiBlocProvider(
  providers: [
    BlocProvider<BlocA>(
      create: (BuildContext context) => BlocA(),
    ),
    BlocProvider<BlocB>(
      create: (BuildContext context) => BlocB(),
    ),
    BlocProvider<BlocC>(
      create: (BuildContext context) => BlocC(),
    ),
  ],
  child: ChildA(),
)
```

BlocListener.

- BlocListener is a Flutter widget which takes a BlocWidgetListener and an optional Bloc and invokes the listener in response to state changes in the bloc. It should be used for functionality that needs to occur once per state change such as navigation, showing a SnackBar, showing a Dialog, etc...
- listener is only called once for each state change (NOT including the initial state) unlike builder in BlocBuilder and is a void function.
- If the bloc parameter is omitted, BlocListener will automatically perform a lookup using BlocProvider and the current BuildContext.

```
BlocListener<BlocA, BlocAState>(
  listener: (context, state) {
    // do stuff here based on BlocA's state
  },
  child: Container(),
)
```