Reactive Programming
BLoC
(Business Logic Component)

ST6002CEM-Mobile Application Development

Reactive Programming

"It is programming with **Asynchronous** Data Streams"

OR

"Do something (=React) when an **Event** is triggered"

BLoC

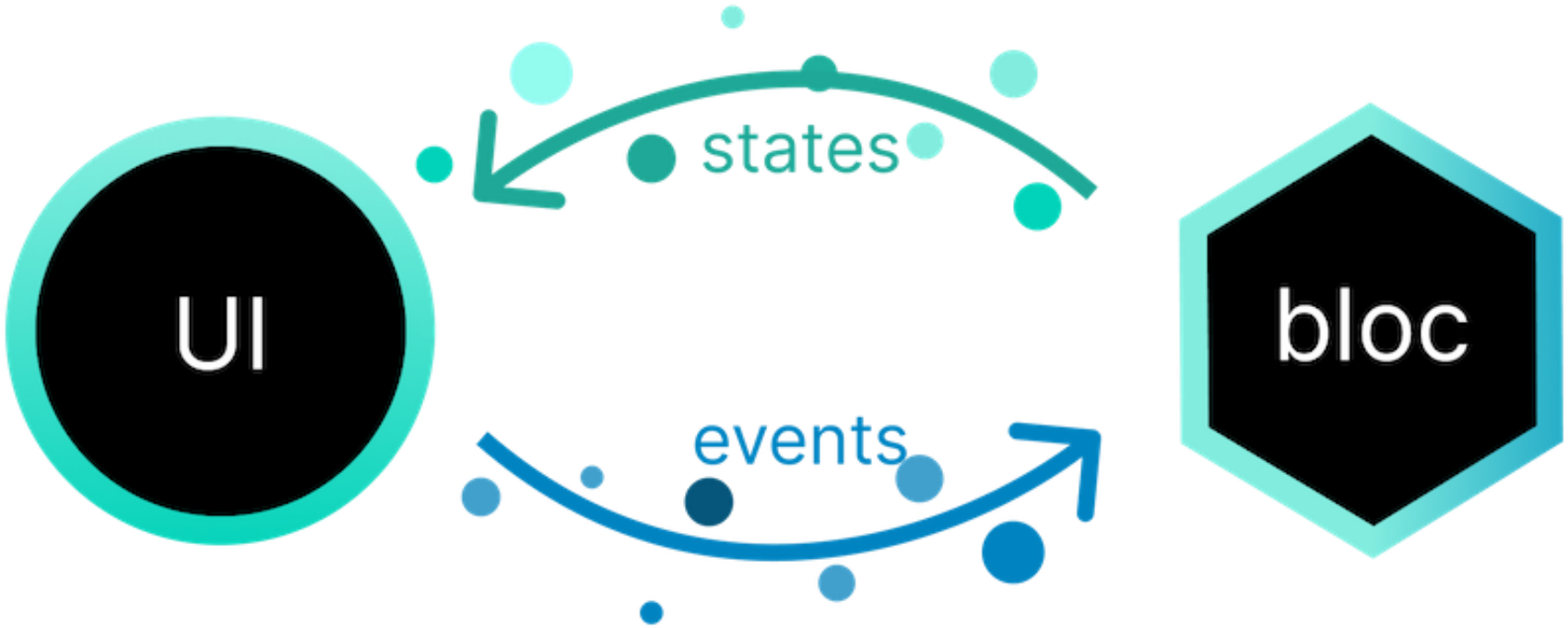
- **BLoC** stands for **Business Logic Component**.
- It is an <u>architectural pattern</u> designed to separate business logic from UI components.
- Bloc was designed with three core values in mind:
  1. Simple: Easy to understand & can be used by developers with varying skill levels.
  2. Powerful: Help make amazing, complex applications by composing them of smaller components.
  3. Testable: Easily test every aspect of an application so that we can iterate with confidence.

Why use BLoC?

- **Separation of Concerns:** Decouples UI from business logic.
- **Reusability:** Reusable components across the app.
- **Testability:** Simplifies unit testing.
- **Scalability:** Ideal for larger applications with complex logic.

How BLoC works?

- **Events:** User interactions trigger events.
- **BLoC:** Handles the business logic and processes events.
- **States:** Outputs new states based on events.
- **UI:** Listens to state changes to update the interface.


states
UI
bloc
events

BLoC and Cubit

- State management patterns used in Flutter.
- They are part of flutter_bloc package
- Help separate business logic from the UI, making your codebase more maintainable, reusable and testable

BLoC

- BLoC (Business Logic Component) is a design pattern where:
  1. **Events**: Represent user interactions or external triggers (e.g., button clicks, API calls).
  2. **States**: Represent the condition of the app (e.g., loading, success, error).
  3. **Business Logic**: Resides in the BLoC class, where events are mapped to states.
- The BLoC listens to events and emits states using streams.
- Key Features:
  - Separates the UI from business logic.
  - Uses streams (Stream and StreamSink) to handle state changes.
  - Ideal for complex state management with multiple inputs and outputs.

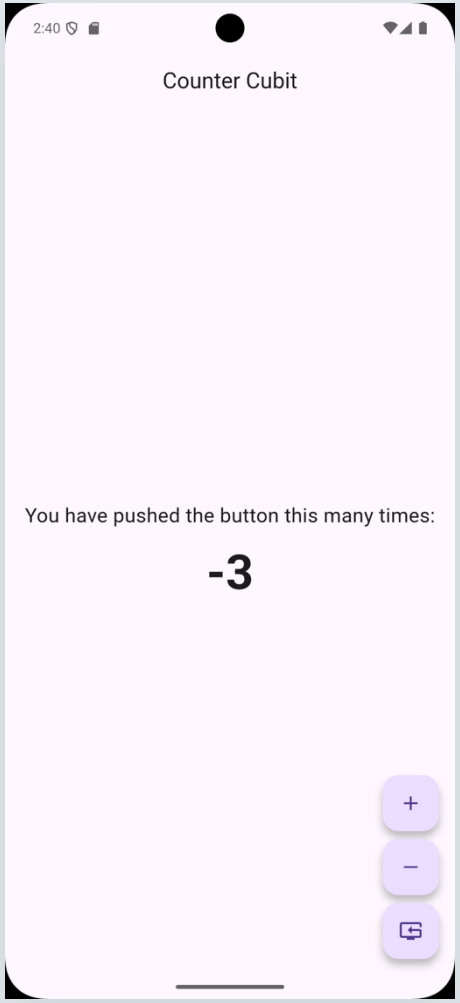
Cubit

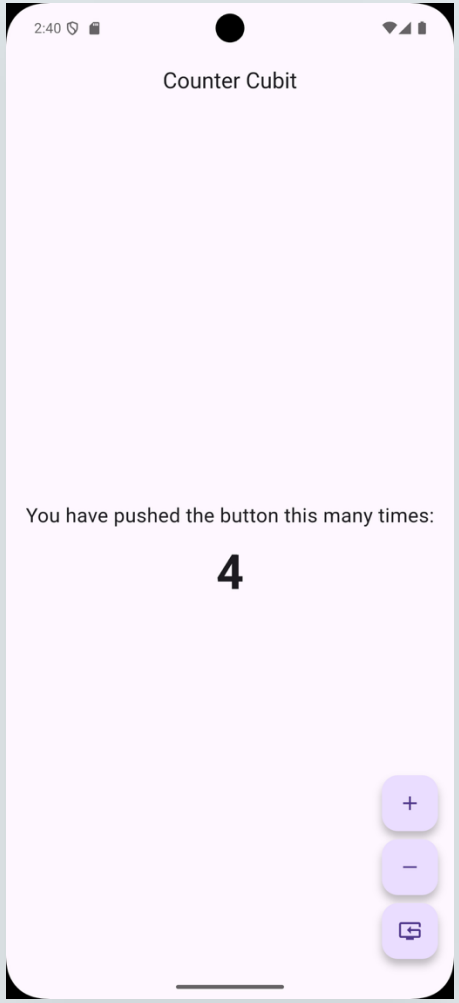
- Cubit is a simplified version of BLoC, designed for managing state without the need for events. Instead of mapping events to states, you directly emit new states.
- Key Features:
  - Easier to use and less boilerplate than BLoC.
  - No need for events; state transitions happen through methods.
  - Suitable for simpler state management scenarios.

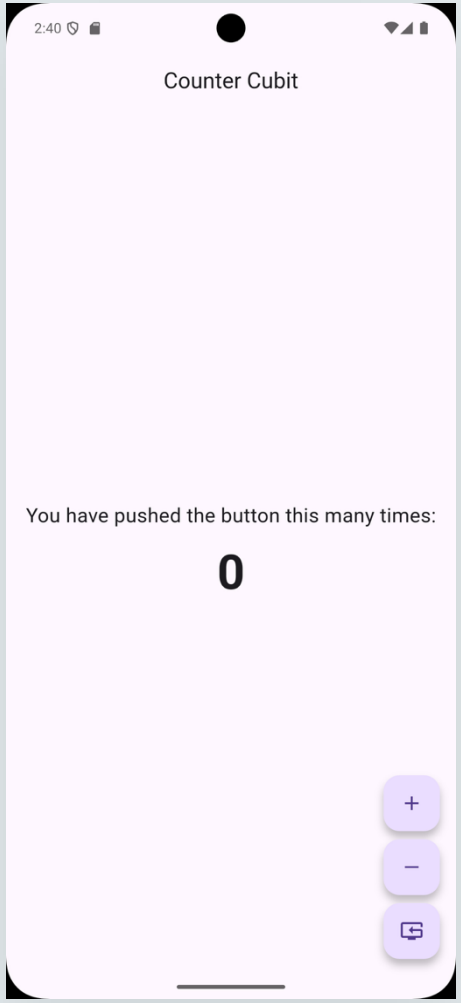
Difference between BLoC and Cubit

Feature	BLoC	Cubit
Complexity	Higher (require events and states)	Lower (only require states)
Flexibility	Suitable for complex, large-scale apps	Suitable for simpler use cases
Boilerplate	More (events and states)	Less (only states)
Streams	Uses Stream for both input and output	Only uses Stream for output
Ease of Use	Require more setup	Easier and Quicker to implement

Example


2:40
Counter Cubit
You have pushed the button this many times:
-3


2:40
Counter Cubit
You have pushed the button this many times:
4


2:40
Counter Cubit
You have pushed the button this many times:
0

Download starter project from GitHub

- https://github.com/kiranrana8973/bloc_starter.git

1. Add package “flutter_bloc”

flutter_bloc 9.1.1

Published 25 days ago • bloclibrary.dev Dart 3 compatible

SDK FLUTTER | PLATFORM ANDROID IOS LINUX MACOS WEB WINDOWS

7.7K

Readme Changelog Example Installing Versions Scores

Use this package as a library

Depend on it

Run this command:

With Flutter:

```
$ flutter pub add flutter_bloc
```

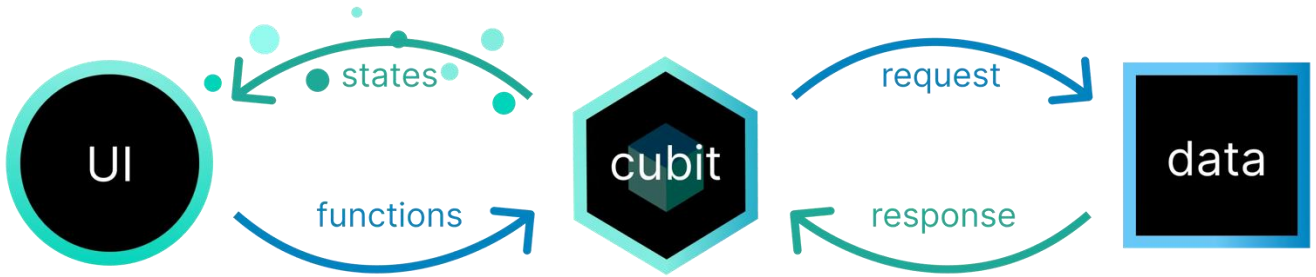
This will add a line like this to your package's pubspec.yaml (and run an implicit `flutter pub get`):

```
dependencies:
  flutter_bloc: ^9.1.1
```

Alternatively, your editor might support `flutter pub get`. Check the docs for your editor to learn more.

Cubit

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



Creating a Cubit

```
class CounterCubit extends Cubit<int> {
  CounterCubit() : super(0);
}
```

OR

```
class CounterCubit extends Cubit<int> {
  CounterCubit(int initialState) : super(initialState);
}
```

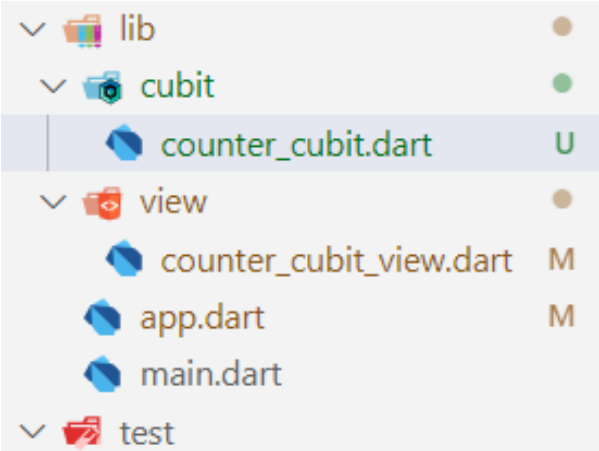
```
final cubitA = CounterCubit(0); // state starts at 0
final cubitB = CounterCubit(10); // state starts at 10
```

Cubit State Changes

```
class CounterCubit extends Cubit<int> {
  CounterCubit() : super(0);


  void increment() => emit(state + 1);
}
```

2. Create a cubit named “counter_cubit”




```dart
import 'package:flutter_bloc/flutter_bloc.dart';

class CounterCubit extends Cubit<int> {
  CounterCubit() : super(0);

  void increment() {
    emit(state + 1);
  }

  void decrement() {
    emit(state - 1);
  }

  void reset() {
    emit(0);
  }
}
```

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.

3.
CounterCubitView

```
19        const Text(
20          'You have pushed the button this many times:',
21          style: TextStyle(
22            fontSize: 20,
23          ), // TextStyle
24        ), // Text
25        const SizedBox(height: 8),
26        BlocBuilder<CounterCubit, int>(
27          builder: (context, state) {
28            return Text(
29              '$state',
30              style: const TextStyle(
31                fontSize: 48,
32                fontWeight: FontWeight.bold,
33              ), // TextStyle
34            ); // Text
35          },
36        ), // BlocBuilder
```

BlocProvider

- BlocProvider is a Flutter widget which provides a bloc to its children via BlocProvider.of<T>(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.

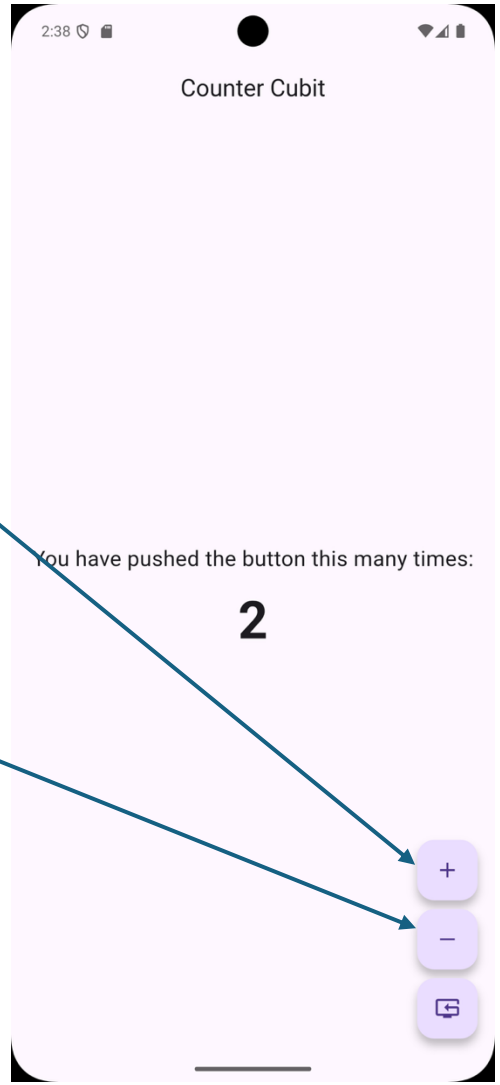
4. app.dart

```dart
class App extends StatelessWidget {
  const App({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'Flutter BLoC',
      home: BlocProvider(
        create: (context) => CounterCubit(),
        child: CounterCubitView(),
      ), // BlocProvider
    ); // MaterialApp
  }
}
```

5. Calling cubit function from View

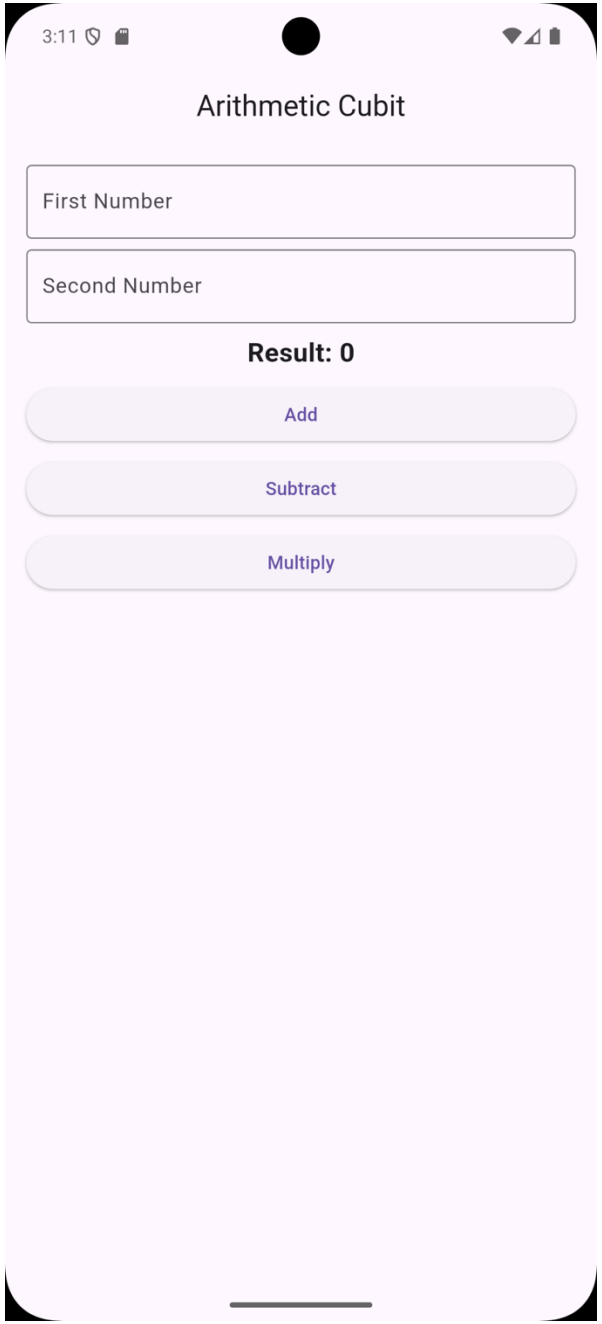
```
FloatingActionButton(
  onPressed: () {
    context.read<CounterCubit>().increment();
  },
  tooltip: 'Increment',
  child: const Icon(Icons.add),
), // FloatingActionButton
FloatingActionButton(
  onPressed: () {
    context.read<CounterCubit>().decrement();
  },
  tooltip: 'Decrement',
  child: const Icon(Icons.remove),
), // FloatingActionButton
```



6. Now add logic to the decrement function so that it can go down to -5, but it should stop after that

```
void decrement() {
  if (state == -5) {
    return;
  }
  emit(state - 1);
}
```

Example 2


3:11
Arithmetic Cubit
First Number
Second Number
Result: 0
Add
Subtract
Multiply

1. Arithmetic_Cubit_View

```
5   class ArithmeticCubitView extends StatelessWidget {
6     ArithmeticCubitView({super.key});
7
8     final _firstNumberController = TextEditingController();
9     final _secondNumberController = TextEditingController();
10
11    final _globalKey = GlobalKey<FormState>();
12
13    @override
14    Widget build(BuildContext context) {
```

2. Create a cubit

```
class ArithmeticCubit extends Cubit<int> {
  ArithmeticCubit() : super(0);

  void add(int firstNumber, int secondNumber) {
    emit(firstNumber + secondNumber);
  }

  void subtract(int firstNumber, int secondNumber) {
    emit(firstNumber - secondNumber);
  }

  void multiply(int firstNumber, int secondNumber) {
    emit(firstNumber * secondNumber);
  }
}
```

3. BlocBuilder in UI

```
57          Align(
58            alignment: Alignment.center,
59            child: BlocBuilder<ArithmeticCubit, int>(
60              builder: (context, state) {
61                return Text(
62                  "Result: $state",
63                  style: TextStyle(
64                    fontSize: 20,
65                    fontWeight: FontWeight.bold,
66                  ), // TextStyle
67                ); // Text
68              },
69            ), // BlocBuilder
70          ), // Align
```

3. Calling cubit on Button click

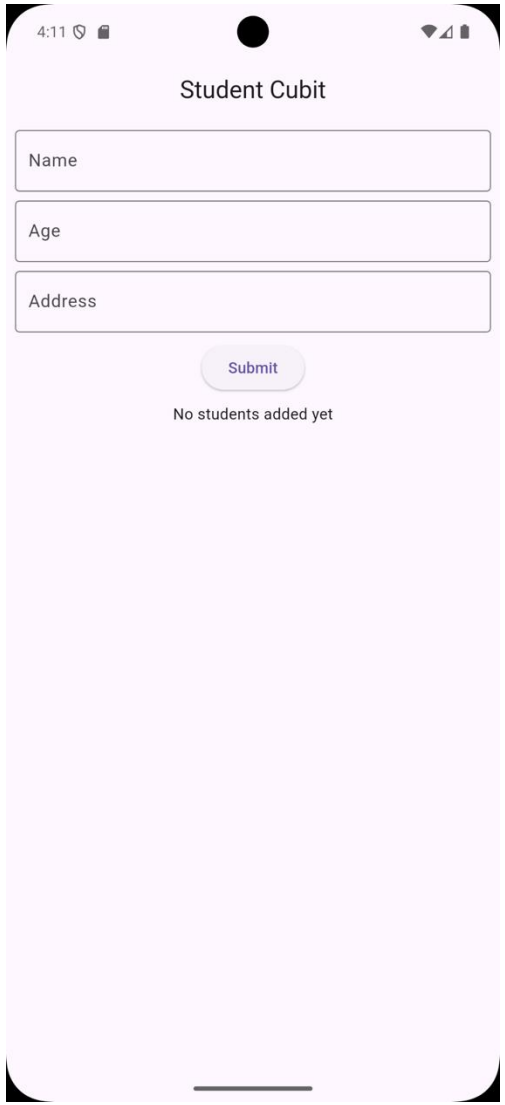
```
ElevatedButton(
  onPressed: () {
    if (_globalKey.currentState!.validate()) {
      final firstNumber = int.parse(_firstNumberController.text);
      final secondNumber =
          int.parse(_secondNumberController.text);
      context
          .read<ArithmeticCubit>()
          .add(firstNumber, secondNumber);
    }
  },
  child: const Text('Add'),
), // ElevatedButton
```

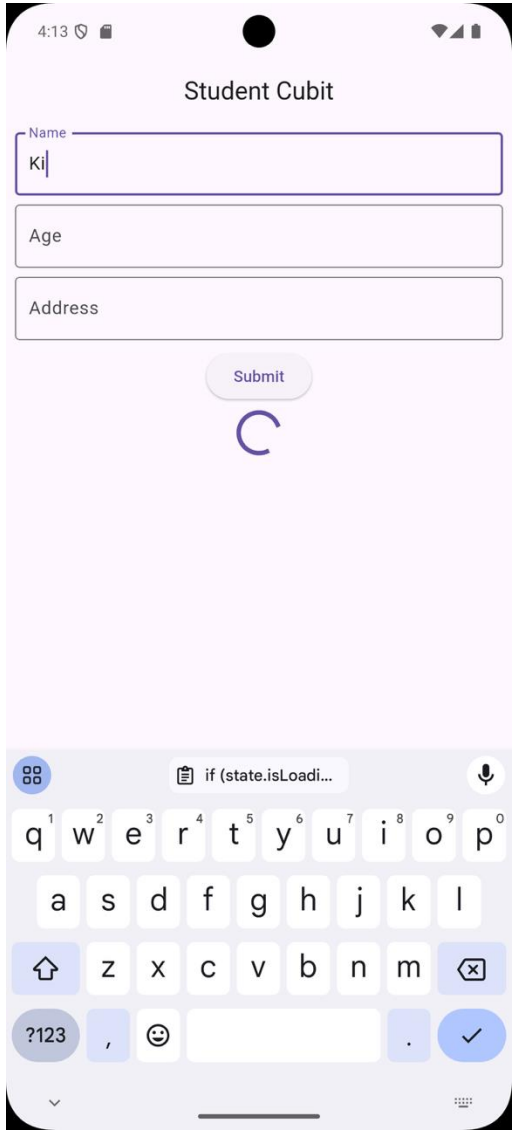
4. BlocProvider in app.dart

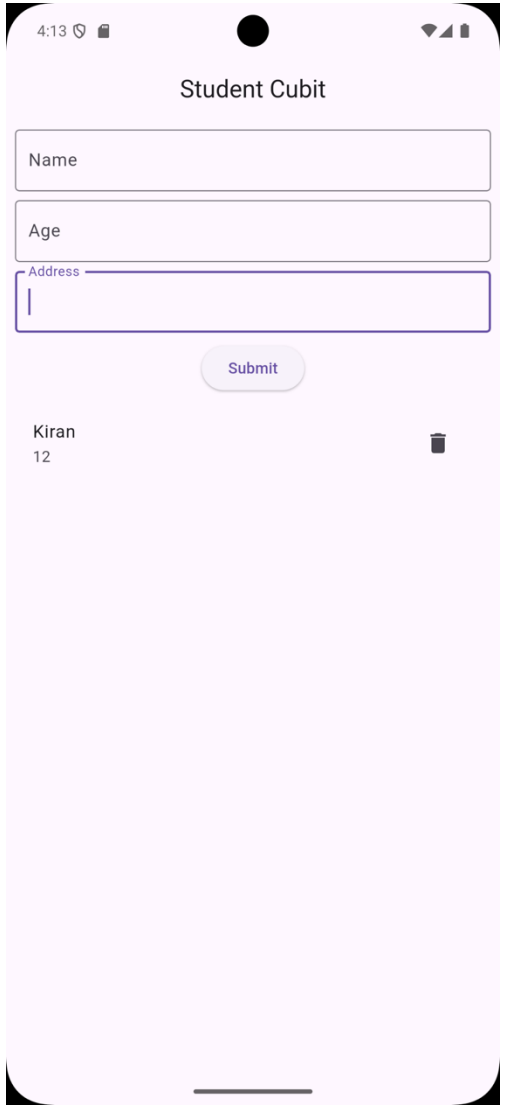
```
9    @override
10   Widget build(BuildContext context) {
11     return MaterialApp(
12       debugShowCheckedModeBanner: false,
13       title: 'Flutter BLoC',
14       home: BlocProvider(
15         create: (context) => ArithmeticCubit(),
16         child: ArithmeticCubitView(),
17       ), // BlocProvider
18     ); // MaterialApp
19   }
20 }
```

Let’s create cubit for complex type

1. Student_cubit_view







1. Student_cubit_view

```
7   class StudentCubitView extends StatelessWidget {
8     StudentCubitView({super.key});
9
10    final _nameController = TextEditingController();
11    final _ageController = TextEditingController();
12    final _addressController = TextEditingController();
13
14    final _formKey2 = GlobalKey<FormState>();
15
16    @override
```

Add Equatable package

equatable 2.0.7

Published 26 days ago • fluttercommunity.dev Dart 3 compatible • Latest: 2.0.7 / Prerelease: 3.0.0-dev.1

SDK DART FLUTTER PLATFORM ANDROID IOS LINUX MACOS WEB WINDOWS 3.2K

Readme Changelog Example Installing Versions Scores

Use this package as a library

Depend on it

Run this command:

With Dart:

```
$ dart pub add equatable
```

With Flutter:

```
$ flutter pub add equatable
```

This will add a line like this to your package's pubspec.yaml (and run an implicit `dart pub get`):

```
dependencies:
  equatable: ^2.0.7
```

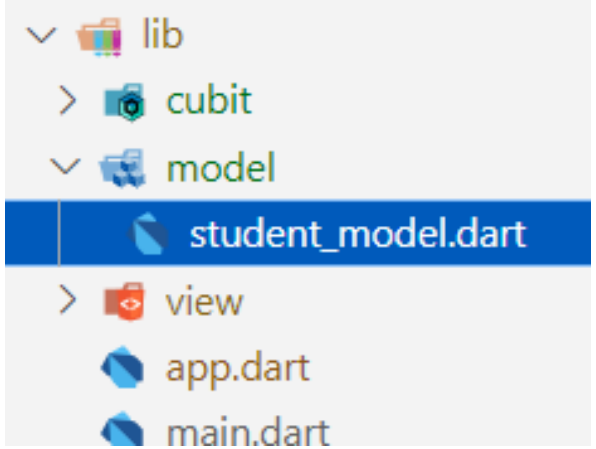
Alternatively, your editor might support `dart pub get` or `flutter pub get`. Check the docs for your editor to learn more.

Import it

Now in your Dart code, you can use:

```
import 'package:equatable/equatable.dart';
```

2. Model




```dart
import 'package:equatable/equatable.dart';

class StudentModel extends Equatable {
  final String name;
  final int age;
  final String address;

  const StudentModel({
    required this.name,
    required this.age,
    required this.address,
  });

  @override
  List<Object?> get props => [
        name,
        age,
        address,
      ];
}
```

3. State

```
3   class StudentState {
4     final List<StudentModel> lstStudents;
5     final bool isLoading;
6
7     const StudentState({
8       required this.lstStudents,
9       required this.isLoading,
10    });
11
12    factory StudentState.initial() {
13      return StudentState(
14        lstStudents: [],
15        isLoading: false,
16      );
17    }
```

```
19    StudentState copyWith({
20      List<StudentModel>? lstStudents,
21      bool? isLoading,
22    }) {
23      return StudentState(
24        lstStudents: lstStudents ?? this.lstStudents,
25        isLoading: isLoading ?? this.isLoading,
26      );
27    }
28  }
```

4. Cubit

```
5   class StudentCubit extends Cubit<StudentState> {
6     StudentCubit() : super(StudentState.initial());
7
8     void addStudent(StudentModel student) {
9       emit(state.copyWith(isLoading: true));
10      //wait for 1 second, so that we can see the loading state
11      Future.delayed(Duration(seconds: 1), () {
12        emit(
13          state.copyWith(
14            lstStudents: state.lstStudents..add(student),
15            isLoading: false,
16          ),
17        );
18      }); // Future.delayed
19    }
```

4. Cubit

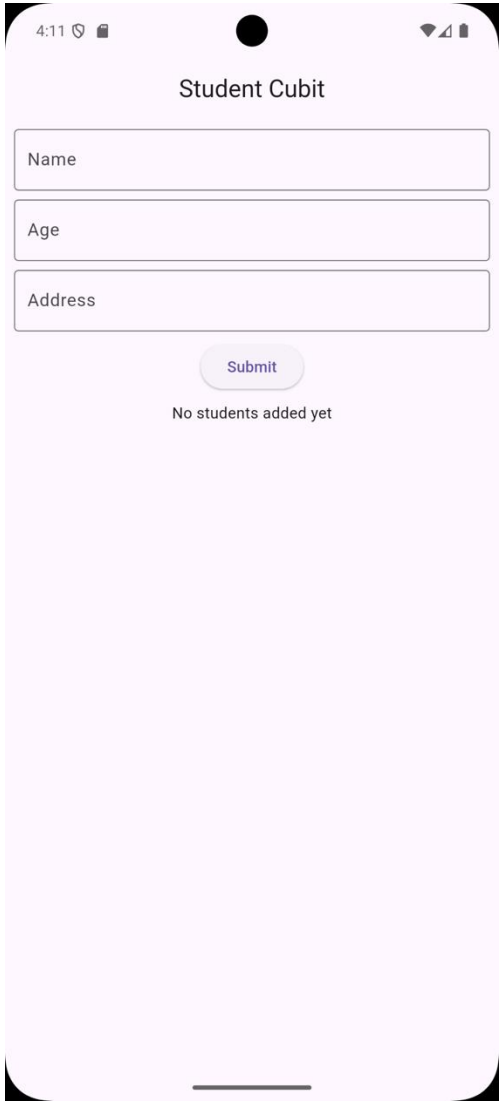
```
21    void deleteStudent(int index) {
22      emit(state.copyWith(isLoading: true));
23      //wait for 1 second, so that we can see the loading state
24      Future.delayed(Duration(seconds: 1), () {
25        emit(
26          state.copyWith(
27            lstStudents: state.lstStudents..removeAt(index),
28            isLoading: false,
29          ),
30        );
31      }); // Future.delayed
32    }
33  }
```

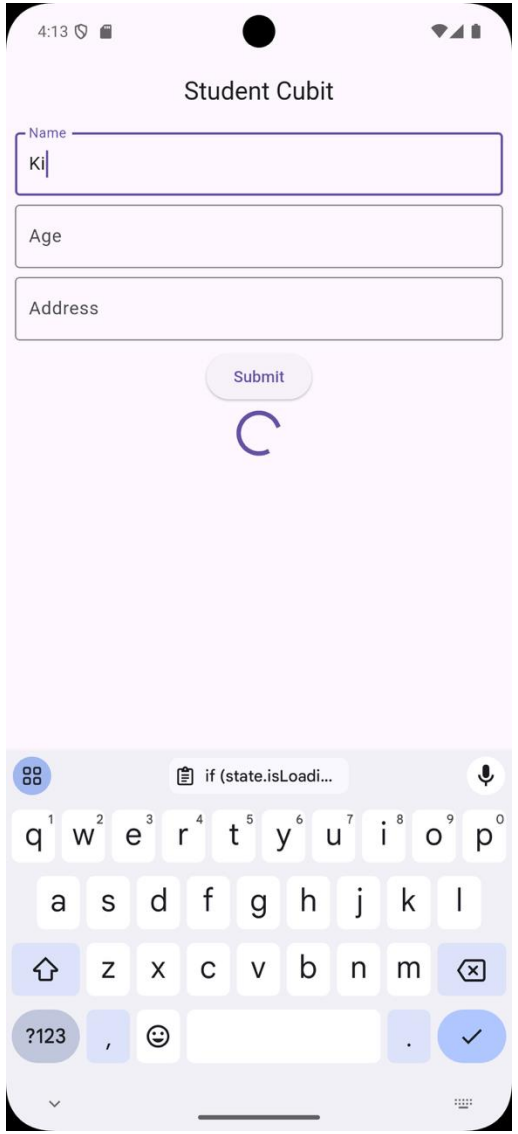
5. BlocBuilder

```
ElevatedButton(
  onPressed: () {
    if (_formKey2.currentState!.validate()) {
      StudentModel student = StudentModel(
        name: _nameController.text,
        age: int.parse(_ageController.text),
        address: _addressController.text,
      ); // StudentModel
      context.read<StudentCubit>().addStudent(student);
      _nameController.clear();
      _ageController.clear();
      _addressController.clear();
    }
  },
  child: const Text('Submit'),
), // ElevatedButton
```

5. BlocBuilder

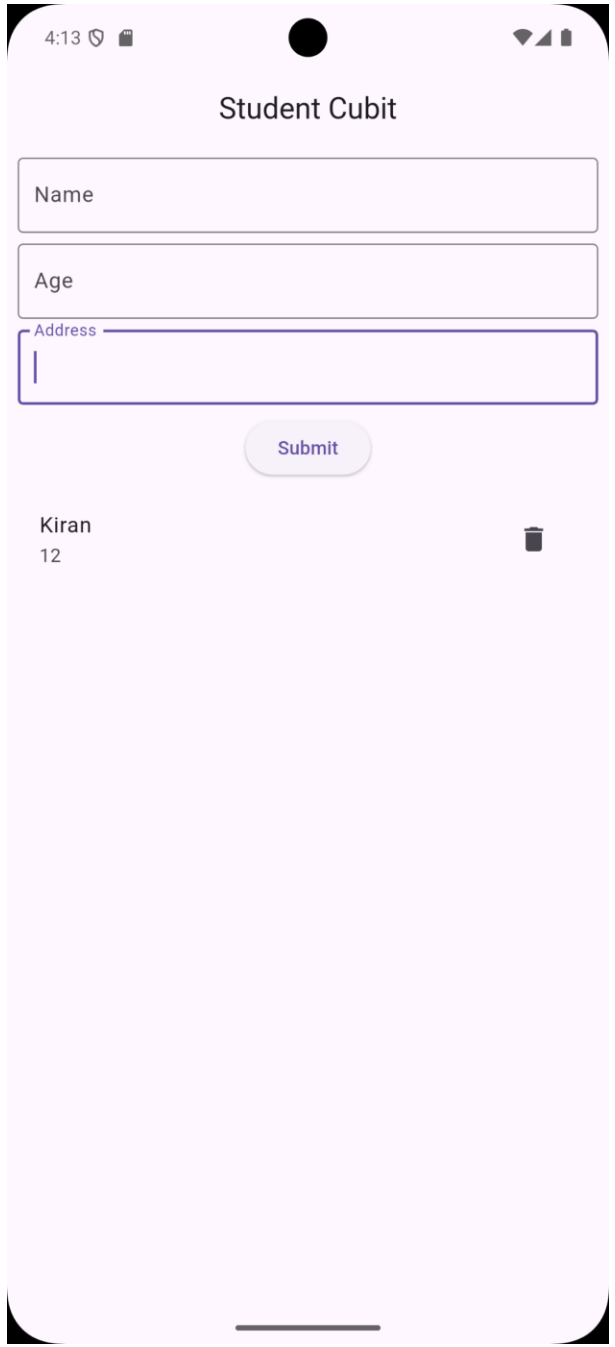
```
BlocBuilder<StudentCubit, StudentState>(
  builder: (context, state) {
    if (state.isLoading) {
      return const CircularProgressIndicator();
    } else if (state.lstStudents.isEmpty) {
      return const Text('No students added yet');
    } else {
      return ListView.builder(
```





5. BlocBuilder

```
} else {
  return ListView.builder(
    shrinkWrap: true,
    itemCount: state.lstStudents.length,
    itemBuilder: (context, index) {
      return ListTile(
        title: Text(state.lstStudents[index].name),
        subtitle:
            Text(state.lstStudents[index].age.toString()),
        trailing: IconButton(
          icon: const Icon(Icons.delete),
          onPressed: () {
            context.read<StudentCubit>().deleteStudent(index);
          },
        ), // IconButton
      ); // ListTile
    },
  ); // ListView.builder
}
```



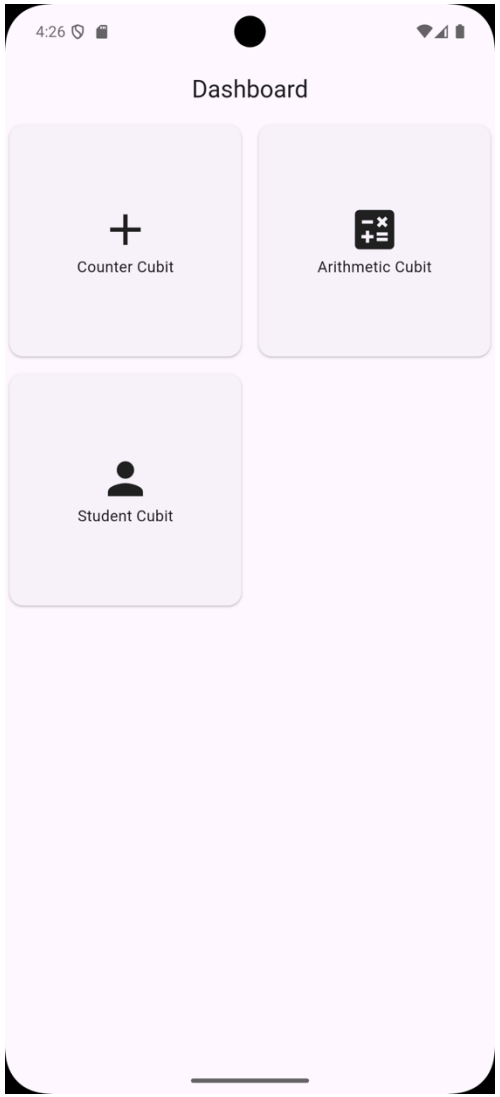
5. BlocBuilder

```
} else {
  return Expanded(
    child: CustomScrollView(
      slivers: [
        SliverList(
          delegate: SliverChildBuilderDelegate(
            childCount: state.lstStudents.length,
            (context, index) {
              final student = state.lstStudents[index];
              return ListTile(
                title: Text(student.name),
                subtitle: Text(
                    'Age: ${student.age}, Address: ${student.address}'),
                leading: CircleAvatar(
                  child: Text(student.name[0]),
                ), // CircleAvatar
                trailing: IconButton(
                  icon: const Icon(Icons.delete),
                  onPressed: () {
                    context
                        .read<StudentCubit>()
                        .deleteStudent(student);
                  },
                ), // IconButton
              ); // ListTile
            },
          ), // SliverChildBuilderDelegate
        ), // SliverList
      ],
    ), // CustomScrollView
  ); // Expanded
}
```

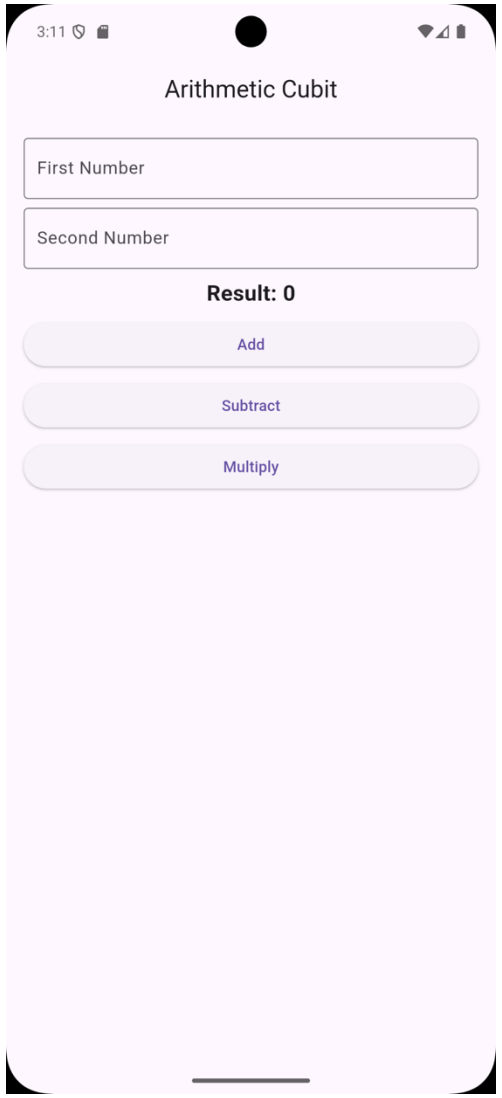
6. BlocProvider

```
home: BlocProvider(
  create: (context) => StudentCubit(),
  child: StudentCubitView(),
), // BlocProvider
```

Example 3 – Navigation using Dashboard



Always navigate using cubit



1. Cubit

```
7   class DashboardCubit extends Cubit<void> {
8     DashboardCubit() : super(null);
9
10    void openCounterView(BuildContext context) {
11      Navigator.push(
12        context,
13        MaterialPageRoute(
14          builder: (_) => CounterCubitView(),
15        ), // MaterialPageRoute
16      );
17    }
18
19    void openArithmeticView(BuildContext context) {
20      Navigator.push(
21        context,
22        MaterialPageRoute(
23          builder: (_) => ArithmeticCubitView(),
24        ), // MaterialPageRoute
25      );
26    }
```

Problem

- We need to pass Bloc when opening a view.

Solution

```
10  class DashboardCubit extends Cubit<void> {
11    DashboardCubit(
12      this._counterCubit,
13      this._arithmeticCubit,
14      this._studentCubit,
15    ) : super(null);
16
17    final CounterCubit _counterCubit;
18    final ArithmeticCubit _arithmeticCubit;
19    final StudentCubit _studentCubit;
```

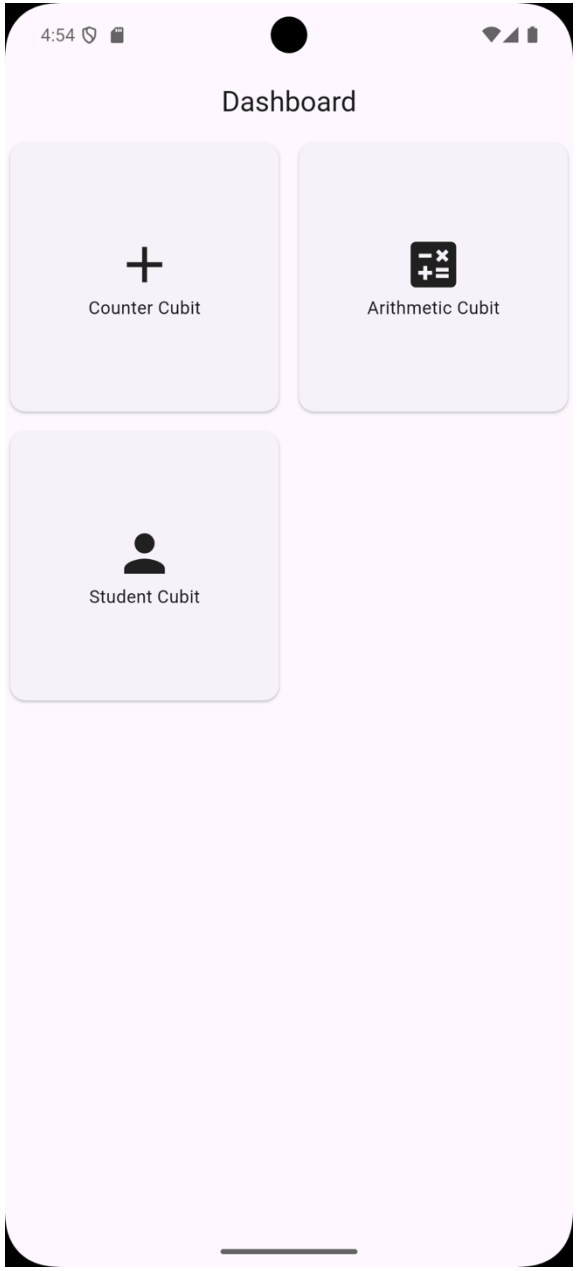
```
21  void openCounterView(BuildContext context) {
22    Navigator.push(
23      context,
24      MaterialPageRoute(
25        builder: (_) => BlocProvider.value(
26          value: _counterCubit,
27          child: CounterCubitView(),
28        ), // BlocProvider.value
29      ), // MaterialPageRoute
30    );
31  }
```

Solution

```
33    void openArithmeticView(BuildContext context) {
34      Navigator.push(
35        context,
36        MaterialPageRoute(
37          builder: (_) => BlocProvider.value(
38            value: _arithmeticCubit,
39            child: ArithmeticCubitView(),
40          ), // BlocProvider.value
41        ), // MaterialPageRoute
42      );
43    }
44
45    void openStudenView(BuildContext context) {
46      Navigator.push(
47        context,
48        MaterialPageRoute(
49          builder: (_) => BlocProvider.value(
50            value: _studentCubit,
51            child: StudentCubitView(),
52          ), // BlocProvider.value
53        ), // MaterialPageRoute
54      );
55    }
56  }
```

2. UI

```
15        body: GridView(
16          gridDelegate: const SliverGridDelegateWithFixedCrossAxisCount(
17            crossAxisCount: 2,
18            crossAxisSpacing: 8,
19            mainAxisSpacing: 8,
20          ), // SliverGridDelegateWithFixedCrossAxisCount
21          children: <Widget>[
22            Card(
23              child: InkWell(
24                onTap: () {
25                  context.read<DashboardCubit>().openCounterView(context);
26                },
27                child: Column(
28                  mainAxisAlignment: MainAxisAlignment.center,
29                  children: const <Widget>[
30                    Icon(Icons.add, size: 48),
31                    Text('Counter Cubit'),
32                  ], // <Widget>[]
33                ), // Column
34              ), // InkWell
35            ), // Card
```



3. MultiBlocProvider

```
13    Widget build(BuildContext context) {
14      return MultiBlocProvider(
15        providers: [
16          BlocProvider(create: (context) => CounterCubit()),
17          BlocProvider(create: (context) => ArithmeticCubit()),
18          BlocProvider(create: (context) => StudentCubit()),
19          BlocProvider(
20            create: (context) => DashboardCubit(
21              context.read<CounterCubit>(),
22              context.read<ArithmeticCubit>(),
23              context.read<StudentCubit>(),
24            ), // DashboardCubit
25          ), // BlocProvider
26        ],
27        child: MaterialApp(
28          debugShowCheckedModeBanner: false,
29          title: 'Flutter BLoC',
30          home: DashboardView(),
31        ), // MaterialApp
32      ); // MultiBlocProvider
33    }
34  }
```