ĐẠI HỌC BÁCH KHOA - ĐÀ NẴNG
KHOA ĐIỆN TỬ - VIỄN THÔNG

BLOC PATTERN - BUSINESS LOGIC COMPONENT
MÔN: LẬP TRÌNH ĐA NỀN TẢNG

Giáo viên hướng dẫn: TS. Nguyễn Duy Nhật Viễn
Nhóm: Phạm Hùng Cường
Trương Văn An
Đinh Công Huy

Đà Nẵng 2025

Nội dung trình bày

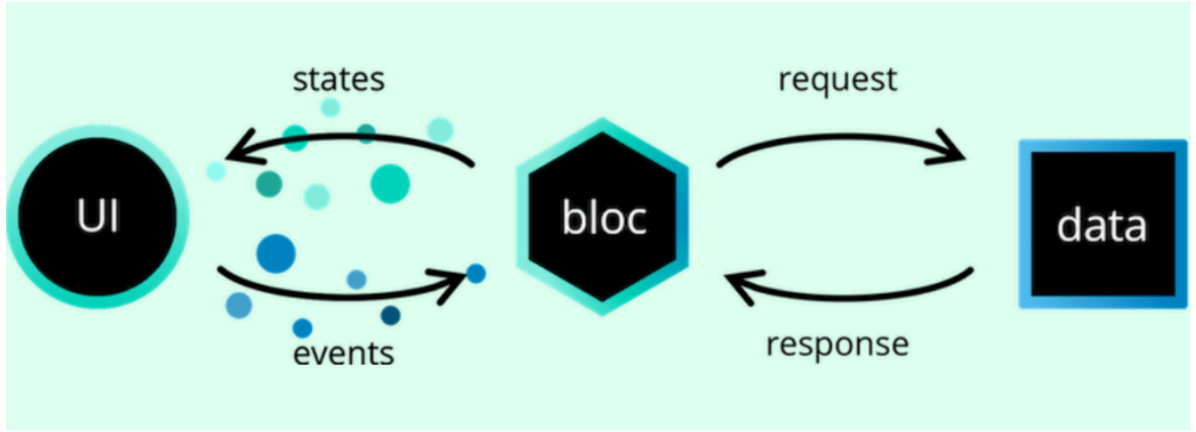
1. Tạo BLoC class với events và states
2. BlocBuilder và BlocListener widgets
3. Event handling và state transitions
4. Testing BLoC với bloc_test

Phân công công việc

TT	HỌ VÀ TÊN	NỘI DUNG	TỶ LỆ ĐÓNG GÓP
1	Đinh Công Huy	Làm phần 3, phần 4	33%
2	Trương Văn An	Làm phần 2, phần 4	33%
3	Phạm Hùng Cường	Làm phần 1, phần 4	33%

Giới thiệu BLoC Pattern

1. Giới thiệu BLoC Pattern



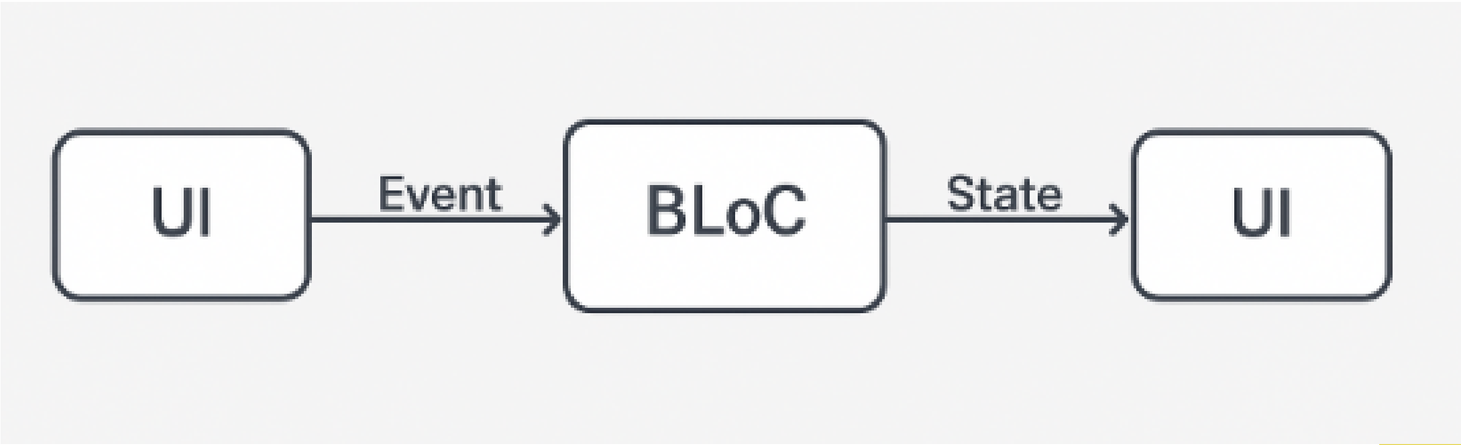
- Kiến trúc được sử dụng trong Flutter nhằm tách biệt phần xử lý logic (business logic) khỏi giao diện (UI).

- Giúp mã nguồn dễ mở rộng, dễ kiểm thử và tái sử dụng.

Giới thiệu BLoC Pattern

1. Nguyên lý hoạt động:

- UI gửi sự kiện (Event) đến BLoC.
- BLoC xử lý sự kiện, cập nhật trạng thái (State) mới.
- UI lắng nghe và hiển thị lại giao diện dựa trên State hiện tại.



PHẦN 1: Tạo BLoC class với events và states

Ví dụ: Bộ đếm (Counter App)

- 1. Khởi tạo thư viện

-Cài gói flutter_bloc trong pubspec.yaml:

+Bloc, BlocBuilder, BlocListener >giúp việc quản lý state và event trở nên đơn giản hơn.

```
dependencies:
  flutter:
    sdk: flutter
  flutter_bloc: ^9.1.1
  equatable: ^2.0.7
```

PHẦN 1: Tạo BLoC class với events và states

Ví dụ: Bộ đếm (Counter App)

- 2. Tạo thư mục cấu trúc BLoC

🟦 Cấu trúc này giúp chia tách logic (Bloc) với giao diện (UI) rõ ràng

```
lib/
  └──── counter/
        ├──── counter_bloc.dart
// Chứa class BLoC chính
        ├──── counter_event.dart
// Định nghĩa các sự kiện
(Event)
        └──── counter_state.dart
// Định nghĩa các trạng thái
(State)
```

PHẦN 1: Tạo BLoC class với events và states

Ví dụ: Bộ đếm (Counter App)

- 3. Định nghĩa Event

counter_event.dart

Mỗi Event đại diện cho một hành động từ người dùng

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

abstract class CounterEvent extends Equatable {
  const CounterEvent();

  @override
  List<Object> get props => [];
}

// Sự kiện tăng giá trị
class IncrementEvent extends CounterEvent {}

// Sự kiện giảm giá trị
class DecrementEvent extends CounterEvent {}
```

PHẦN 1: Tạo BLoC class với events và states

Ví dụ: Bộ đếm (Counter App)

- 4. Định nghĩa State

counter_state.dart

-State biểu diễn tình trạng hiện tại của ứng dụng mà UI sẽ hiển thị.

-Ví dụ, ứng dụng đếm số có một biến trạng thái duy nhất là giá trị đếm (counterValue).

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

class CounterState extends Equatable {
  final int counterValue;

  const CounterState(this.counterValue);

  @override
  List<Object> get props => [counterValue];
}
```

PHẦN 1: Tạo BLoC class với events và states

Ví dụ: Bộ đếm (Counter App)

- 5. Tạo BLoC class

📄 counter_bloc.dart

```dart
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:equatable/equatable.dart';
import 'counter_event.dart';
import 'counter_state.dart';

// Lớp CounterBloc kế thừa từ Bloc<CounterEvent, CounterState>
class CounterBloc extends Bloc<CounterEvent, CounterState> {
  // Constructor khởi tạo state ban đầu = 0
  CounterBloc(): super(CounterState(0)) {
    // Khi nhận được IncrementEvent -> tăng giá trị
    on<IncrementEvent>((event, emit) {
      emit(CounterState(state.counterValue + 1));
    });

    // Khi nhận được DecrementEvent -> giảm giá trị
    on<DecrementEvent>((event, emit) {
      emit(CounterState(state.counterValue - 1));
    });
  }
}
```

👉 Kết quả :Một BLoC class quản lý logic tăng/giảm giá trị, với events và states tách biệt rõ ràng.

PHẦN 2: BlocBuilder và BlocListenter widgets

1. BlocBuilder - Khái niệm

BlocBuilder là gì?

- Widget tự động **rebuild UI** khi state thay đổi
- Chỉ rebuild phần UI được bọc

Cú pháp 1: BlocBuilder cơ bản

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

PHẦN 2: BlocBuilder và BlocListenter widgets

1. BlocBuilder - Khái niệm

Cú pháp 2: BlocBuilder với tham số bloc

```
BlocBuilder<BlocA, BlocAState>(
  bloc: blocA, // provide the local bloc instance
  builder: (context, state) {
    // return widget here based on BlocA's state
  },
);
```

PHẦN 2: BlocBuilder và BlocListenter widgets

1. BlocBuilder - Khái niệm

Cú pháp 3: BlocBuilder với điều kiện buildWhen

```
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
  },
);
```

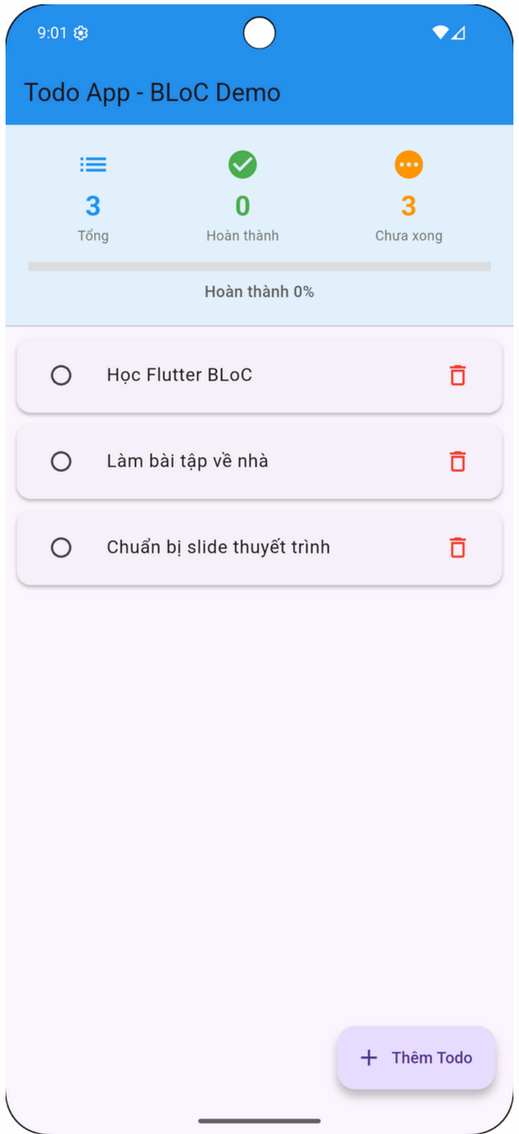
PHẦN 2: BlocBuilder và BlocListenter widgets

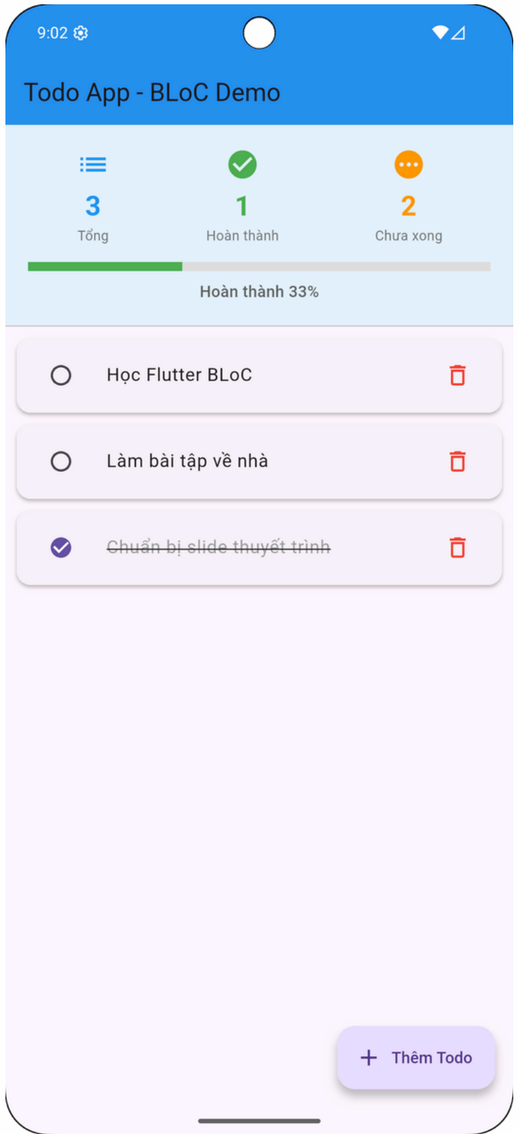
1. BlocBuilder - Demo Thống kê

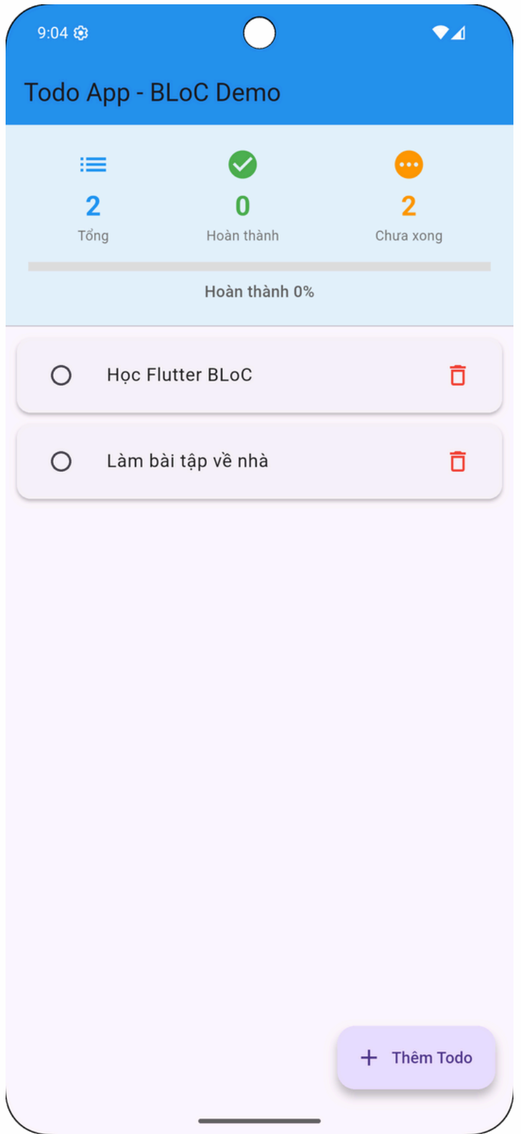
```
BlocBuilder<TodoBloc, TodoState>(
  builder: (context, state) {
    if (state is TodoLoaded) {
      return Row(
        children: [
          Text('Tổng: ${state.totalCount}'),
          Text('Hoàn thành: ${state.completedCount}'),
          Text('Chưa xong: ${state.totalCount - state.completedCount}'),
        ],
      ); Row
    }
    return SizedBox.shrink();
  },
); BlocBuilder
```

PHẦN 2: BlocBuilder và BlocListenter widgets

1. BlocBuilder - Demo Thống kê







PHẦN 2: BlocBuilder và BlocListenter widgets

1. BlocBuilder - buildWhen

Tối ưu rebuild:

Lợi ích:

- Giảm số lần rebuild không cần thiết
- Tối ưu performance

```dart
BlocBuilder<TodoBloc, TodoState>(
  buildWhen: (previous, current) {
    if (previous is TodoLoaded && current is TodoLoaded) {
      return previous.totalCount != current.totalCount ||
          previous.completedCount != current.completedCount;
    }
    return true;
  },
  builder: (context, state) {
    return _StatisticsCard();
  },
);
```

BlocBuilder

PHẦN 2: BlocBuilder và BlocListenter widgets

2. BlocListener - Khái niệm

BlocListener là gì?

- Widget xử lý **hành động phụ** khi state thay đổi
- **KHÔNG rebuild UI**
- Dùng cho: Navigation, Dialog, SnackBar,...

Cú pháp 1: BlocListener cơ bản

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

PHẦN 2: BlocBuilder và BlocListenter widgets

2. BlocListener - Khái niệm

Cú pháp 2: BlocListener với tham số bloc

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

PHẦN 2: BlocBuilder và BlocListenter widgets

2. BlocListener - Khái niệm

Cú pháp 3: BlocListener với điều kiện listenWhen

```
BlocListener<BlocA, BlocAState>(
  listenWhen: (previousState, state) {
    // return true/false to determine whether or not
    // to call listener with state
  },
  listener: (context, state) {
    // do stuff here based on BlocA's state
  },
  child: const SizedBox(),
);
```

PHẦN 2: BlocBuilder và BlocListenter widgets

2. BlocListener - Khái niệm

Cú pháp mở rộng: MultiBlocListener

```
BlocListener<BlocA, BlocAState>(
  listener: (context, state) {},
  child: BlocListener<BlocB, BlocBState>(
    listener: (context, state) {},
    child: BlocListener<BlocC, BlocCState>(
      listener: (context, state) {},
      child: ChildA(),
    ),
  ),
);
```

```
MultiBlocListener(
  listeners: [
    BlocListener<BlocA, BlocAState>(
      listener: (context, state) {},
    ),
    BlocListener<BlocB, BlocBState>(
      listener: (context, state) {},
    ),
    BlocListener<BlocC, BlocCState>(
      listener: (context, state) {},
    ),
  ],
  child: ChildA(),
);
```

PHẦN 2: BlocBuilder và BlocListenter widgets

2. BlocListener - Demo Todo App

```
BlocListener<TodoBloc, TodoState>(
  listener: (context, state) {
    if (state is TodoAdded) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('✅ Đã thêm todo')),
      );
    }

    if (state is TodoDeleted) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('🗑️ Đã xóa: ${state.deletedTitle}'),
          action: SnackBarAction(label: 'HOÀN TÁC', onPressed: () {}),
        ), SnackBar
      );
    }

    if (state is TodoError) {
      showDialog(
        context: context,
        builder: (_) => AlertDialog(
          title: Text('❌ Lỗi'),
          content: Text(state.message),
        ), AlertDialog
      );
    }
  },
  child: TodoListView(),
), BlocListener
```

PHẦN 2: BlocBuilder và BlocListenter widgets

3. BlocConsumer - Kết hợp cả 2

```
BlocConsumer<TodoBloc, TodoState>(
  listener: (context, state) {
    if (state is TodoDeleted) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('Đã xóa')),
      );
    }
  },

  builder: (context, state) {
    if (state is TodoLoading) {
      return CircularProgressIndicator();
    }
    if (state is TodoLoaded) {
      return ListView.builder(
        itemCount: state.todos.length,
        itemBuilder: (context, index) => TodoItem(state.todos[index]),
      ); ListView.builder
    }
    return Container();
  },
), BlocConsumer
```

Khi nào dùng?

VỪA rebuild UI
VỪA side effects

PHẦN 2: BlocBuilder và BlocListenter widgets

4. So sánh 3 widgets

Widget	Rebuild UI	Side Effects	Khi nào dùng
BlocBuilder	✅	❌	Chỉ cần hiển thị data
BlocListener	❌	✅	Dialog, SnackBar
BlocConsumer	✅	✅	Cả 2

PHẦN 2: BlocBuilder và BlocListenter widgets

5. Best Practices

NÊN:

- Dùng **buildWhen** để tối ưu
- Dùng **listenWhen** để kiểm soát
- Tách UI nhỏ thành BlocBuilder riêng
- Dùng BlocConsumer khi cần cả 2

KHÔNG NÊN:

- Rebuild toàn bộ app bằng 1 BlocBuilder
- Dùng Listener cho việc update UI
- Quên dispose BLoC

PHẦN 3: Event handling và state transitions

3.1 Event Handling

- Khái niệm: Là quá trình BLoC tiếp nhận event từ UI và bắt đầu thực thi logic nghiệp vụ.
- Event: Là các class đại diện cho ý định hoặc hành động của người dùng.
- Cách gửi event:

```
ElevatedButton(
  onPressed: () {
    context.read<CounterBloc>().add(CounterIncremented());
  },
  child: const Text('Tăng'),
```

PHẦN 3: Event handling và state transitions

3.2.State Transitions

- Khái niệm: Là quá trình BLoC phát ra một State mới sau khi đã xử lý Event thành công.
- Cơ chế emit():Hàm emit(NewState()) là lệnh cuối cùng trong Event Handler để thông báo cho thế giới bên ngoài (UI) rằng trạng thái đã thay đổi.
- BLoC có vai trò đảm bảo logic nghiệp vụ diễn ra giữa State cũ → State mới.

PHẦN 3: Event handling và state transitions

3.2.State Transitions

- Ví dụ (BLoC Logic):

```dart
class CounterBloc extends Bloc<CounterEvent, int> {
  CounterBloc() : super(0) {
    on<CounterIncremented>((event, emit) {
      final newState = state + 1;
      emit(newState);
    });
  }
}
```

PHẦN 3: Event handling và state transitions

3.3.Phản ứng của UI

- **BlocBuilder:**
  - Phản ứng: Xây dựng lại widget khi state thay đổi.
  - Sử dụng: Hiển thị dữ liệu, thay đổi màu sắc, ẩn/hiện element.

```dart
BlocBuilder<CounterBloc, int>(
  builder: (context, count) {
    return Text('Giá trị hiện tại: $count');
  },
)
```

PHẦN 3: Event handling và state transitions

3.3.Phản ứng của UI

- **BlocListener:**
  - Phản ứng: Thực hiện các hành động không cần build lại UI.
  - Sử dụng: Hiển thị Snackbar, Dialog, điều hướng (Navigation).

```
BlocListener<AuthBloc, AuthState>(
  listener: (context, state) {
    if (state is AuthSuccess) {
      Navigator.of(context).pushReplacement(...);
    }
  },
  child:
)
```

PHẦN 3: Event handling và state transitions

3.3.Phản ứng của UI

- Ví dụ:

```dart
import 'package:flutter/material.dart';
import 'package:flutter_bloc/flutter_bloc.dart';
import 'package:equatable/equatable.dart';

void main() {
  runApp(const MyApp());
}

abstract class CounterEvent extends Equatable {
  const CounterEvent();

  @override
  List<Object> get props => [];
}

class CounterIncremented extends CounterEvent {
  const CounterIncremented();
}
class CounterBloc extends Bloc<CounterEvent, int> {
  CounterBloc() : super(0) {
    on<CounterIncremented>((event, emit) {

      final newState = state + 1;
      emit(newState);
    });
  }
}
```

```dart

class MyApp extends StatelessWidget {
  const MyApp({super.key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'BLoC Demo: Event & State Transition',
      theme: ThemeData(primarySwatch: Colors.blue),
      home: BlocProvider(
        create: (context) => CounterBloc(),
        child: const CounterScreen(),
      ),
    );
  }
}

class CounterScreen extends StatelessWidget {
  const CounterScreen({super.key});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: const Text('BLoC Counter App')),
      body: BlocListener<CounterBloc, int>(
        listener: (context, count) {
```

PHẦN 3: Event handling và state transitions

3.3.Phản ứng của UI

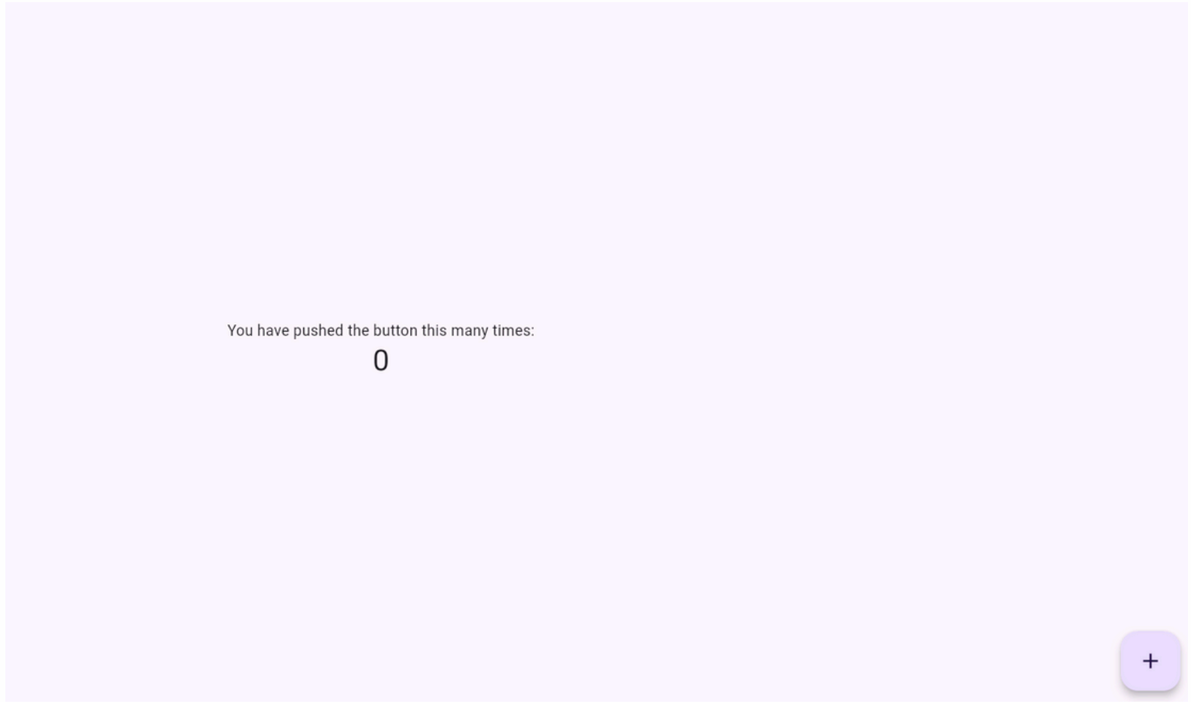
- Ví dụ:

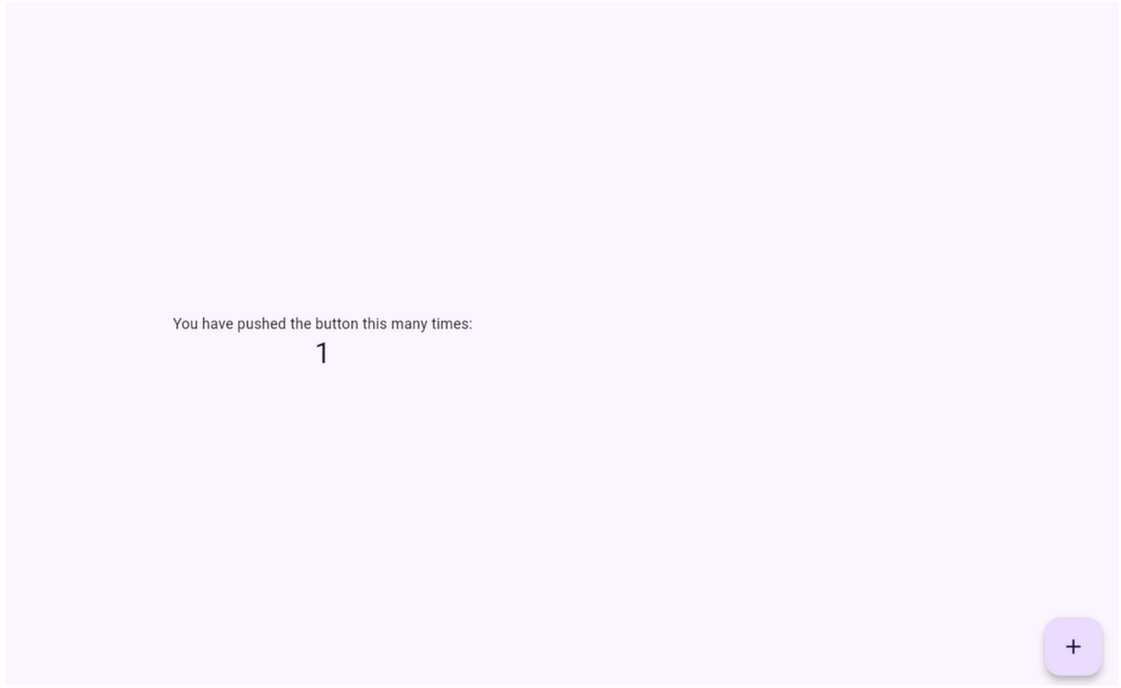
```
55          if (count == 10) {
56            ScaffoldMessenger.of(context).showSnackBar(
57              const SnackBar(
58                content: Text('Đã đạt mốc 10! State Transition thành công.'),
59                duration: Duration(seconds: 2),
60              ),
61            );
62          }
63        },
64        child: Center(
65          child: Column(
66            mainAxisAlignment: MainAxisAlignment.center,
67            children: <Widget>[
68              const Text(
69                'Giá trị hiện tại của biến đếm:',
70                style: TextStyle(fontSize: 18),
71              ),
72
73              BlocBuilder<CounterBloc, int>(
74                builder: (context, count) {
75                  return Text(
76                    '$count',
77                    style: Theme.of(context).textTheme.headlineLarge,
78                  );
79                },
80              ),
81            ],
82          ),
83        ),
84      ),
85
86      floatingActionButton: FloatingActionButton(
87        onPressed: () {
88          context.read<CounterBloc>().add(const CounterIncremented());
89        },
90        tooltip: 'Increment',
91        child: const Icon(Icons.add),
92      ),
93    );
94  }
95 }
```

PHẦN 3: Event handling và state transitions

3.3.Phản ứng của UI

- **Kết quả:**





PHẦN 4: Testing BLoC với bloc_test

1. Tại sao cần test BLoC?

BLoC = Business Logic = Phần quan trọng nhất

Ưu điểm:

- Tách biệt hoàn toàn khỏi UI
- Chỉ cần test: Event → State
- Chạy nhanh, không cần emulator

Setup:

```yaml
dev_dependencies:
  flutter_test:
    sdk: flutter
  bloc_test: ^10.0.0
  mocktail: ^1.0.4
```

PHẦN 4: Testing BLoC với bloc_test

2. blocTest - Cú pháp cơ bản

```
blocTest<TodoBloc, TodoState>(
  'mô tả test',
  build: () => TodoBloc(),
  act: (bloc) => bloc.add(LoadTodos()),
  expect: () => [
    TodoLoading(),
    TodoLoaded([]),
  ],
);
```

3 thành phần chính:

- **build:** Tạo BLoC
- **act:** Add event
- **expect:** Kiểm tra states

PHẦN 4: Testing BLoC với bloc_test

3. Test Initial State

```dart
void main() {
  group('TodoBloc', () {
    late TodoBloc todoBloc;

    setUp(() {
      todoBloc = TodoBloc();
    });

    tearDown(() {
      todoBloc.close();
    });

    test('initial state is TodoInitial', () {
      expect(todoBloc.state, isA<TodoInitial>());
    });
  });
}
```

Kết quả

```
PS C:\Users\Ancod\Downloads\bloc_demo> flutter test test/bloc/todo_bloc_test.dart
Resolving dependencies...
  _fe_analyzer_shared 85.0.0 (92.0.0 available)
  analyzer 7.7.1 (9.0.0 available)
  characters 1.4.0 (1.4.1 available)
  flutter_lints 5.0.0 (6.0.0 available)
  lints 5.1.1 (6.0.0 available)
  material_color_utilities 0.11.1 (0.13.0 available)
  meta 1.16.0 (1.17.0 available)
  test 1.26.2 (1.26.3 available)
  test_api 0.7.6 (0.7.7 available)
  test_core 0.6.11 (0.6.12 available)
Got dependencies!
10 packages have newer versions incompatible with dependency constraints.
Try `flutter pub outdated` for more information.
00:01 +1: All tests passed!
```

PHẦN 4: Testing BLoC với bloc_test

4. Test LoadTodos

```
blocTest<TodoBloc, TodoState>(
  'emits [TodoLoading, TodoLoaded] when LoadTodos is added',
  build: () => TodoBloc(),
  act: (bloc) => bloc.add(LoadTodos()),
  expect: () => [
    isA<TodoLoading>(),
    isA<TodoLoaded>()
        .having((s) => s.todos.length, 'length', 3),
  ],
  wait: const Duration(seconds: 2),
);
```

PHẦN 4: Testing BLoC với bloc_test

5. Test AddTodo

```
blocTest<TodoBloc, TodoState>(
  'emits [TodoAdded] with new todo',
  build: () => TodoBloc(),
  seed: () => TodoLoaded([
    const Todo(id: '1', title: 'Existing'),
  ]), TodoLoaded
  act: (bloc) => bloc.add(AddTodo('New todo')),
  expect: () => [
    isA<TodoAdded>()
        .having((s) => s.todos.length, 'length', 2)
        .having((s) => s.todos.last.title, 'title', 'New todo'),
  ],
);
```

PHẦN 4: Testing BLoC với bloc_test

6. Test DeleteTodo

```
blocTest<TodoBloc, TodoState>(
  'emits [TodoDeleted] when todo is deleted',
  build: () => TodoBloc(),
  seed: () => TodoLoaded([
    const Todo(id: '1', title: 'To delete'),
    const Todo(id: '2', title: 'Keep'),
  ]),  TodoLoaded
  act: (bloc) => bloc.add(DeleteTodo('1')),
  expect: () => [
    isA<TodoDeleted>()
        .having((s) => s.todos.length, 'length', 1)
        .having((s) => s.deletedTitle, 'deleted', 'To delete'),
  ],
);
```

PHẦN 4: Testing BLoC với bloc_test

7. Test ToggleTodo

```
blocTest<TodoBloc, TodoState>(
  'toggles todo completion status',
  build: () => TodoBloc(),
  seed: () => TodoLoaded([
    const Todo(id: '1', title: 'Test', isCompleted: false),
  ]), TodoLoaded
  act: (bloc) => bloc.add(ToggleTodo('1')),
  expect: () => [
    isA<TodoLoaded>()
        .having((s) => s.todos[0].isCompleted, 'completed', true),
  ],
);
```

Test cả 2 chiều: false → true, true → false

PHẦN 4: Testing BLoC với bloc_test

8. Test nhiều Events

```
blocTest<TodoBloc, TodoState>(
  'handles multiple events in sequence',
  build: () => TodoBloc(),
  seed: () => TodoLoaded([]),
  act: (bloc) {
    bloc.add(AddTodo('First'));
    bloc.add(AddTodo('Second'));
    bloc.add(DeleteTodo('1'));
  },
  expect: () => [
    isA<TodoAdded>().having((s) => s.todos.length, 'l', 1),
    isA<TodoAdded>().having((s) => s.todos.length, 'l', 2),
    isA<TodoDeleted>().having((s) => s.todos.length, 'l', 1),
  ],
);
```

PHẦN 4: Testing BLoC với bloc_test

9. Best Practices

NÊN:

- Test tất cả events
- Đặt tên rõ ràng
- Dùng **seed()** cho state ban đầu
- Test cả success và error cases

KHÔNG NÊN:

- Test private methods
- Test implementation details
- Bỏ qua edge cases

Mục tiêu:

- 80%+ coverage
- Tất cả events đều có test

bloc_test = Test BLoC dễ dàng

Lợi ích:

- Phát hiện bug sớm
- Tự tin refactor
- Code quality cao

TÀI LIỆU THAM KHẢO

[1]. Flutter BLoC Documentation: https://bloclibrary.dev

[2]. Flutter BLoC Package: https://pub.dev/packages/flutter_bloc

[3]. bloc_test Package: https://pub.dev/packages/bloc_test

[4]. Equatable Package: https://pub.dev/packages/equatable

[5]. Google I/O 2018 - Build reactive mobile apps with Flutter (BLoC Pattern Introduction)

Cảm ơn thầy và các bạn đã lắng nghe!