**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

**🙡--☼--🙣**



**TÀI LIỆU SEMINAR**

**BLoC**

Môn: Phát triển ứng dụng cho thiết bị di động nâng cao

Lớp: 19\_3

Thành viên: 19120061 – Ngô Trọng Đức

19120383 – Huỳnh Tấn Thọ

19120426 – Phan Đặng Diễm Uyên

Giảng viên: Phạm Hoàng Hải

**Thành phố Hồ Chí Minh, tháng 11 năm 2022**

Mục lục

[1. Khái niệm 3](#_Toc120275095)

[1.1. BLoC là gì? 3](#_Toc120275096)

[1.2. Các khái niệm liên quan trong BLoC 3](#_Toc120275097)

[1.2.1. Stream 3](#_Toc120275098)

[1.2.2. Event 4](#_Toc120275099)

[1.2.3. State 4](#_Toc120275100)

[1.2.4. Cubit 4](#_Toc120275101)

[1.3. Kiến trúc ứng dụng khi sử dụng BLoC 5](#_Toc120275102)

[2. Tại sao, khi nào dùng BLoC 6](#_Toc120275103)

[2.1. Tại sao nên sử dụng BLoC? 6](#_Toc120275104)

[2.2. BLoC và các cách quản lý trạng thái khác trong Flutter 7](#_Toc120275105)

[2.3. Khi nào nên sử dụng BLoC? 7](#_Toc120275106)

[3. Ứng dụng của BLoC trong Flutter 7](#_Toc120275107)

[3.1. Tính hiệu quả 7](#_Toc120275108)

[3.1.1. Tách giao diện ra khỏi business logic 7](#_Toc120275109)

[3.1.2. Quản lý State của ứng dụng hiệu quả hơn 8](#_Toc120275110)

[3.1.3. Tăng khả năng scale của ứng dụng 9](#_Toc120275111)

[3.1.4. Dễ kiểm thử 9](#_Toc120275112)

[3.2. Mức độ phổ biến 9](#_Toc120275113)

[3.3. Ưu, nhược điểm của việc sử dụng BLoC 10](#_Toc120275114)

[3.3.1. Ưu điểm 10](#_Toc120275115)

[3.3.2. Nhược điểm 10](#_Toc120275116)

[4. Các thư viện về BLoC mà Flutter hỗ trợ 10](#_Toc120275117)

[4.1. Bloc 10](#_Toc120275118)

[4.1.1. Giới thiệu chung 10](#_Toc120275119)

[4.1.2. Cubit 10](#_Toc120275120)

[4.1.3. Bloc 11](#_Toc120275121)

[4.1.4. Tài liệu đọc thêm 13](#_Toc120275122)

[4.2. flutter\_bloc 13](#_Toc120275123)

[4.2.1. Giới thiệu chung 13](#_Toc120275124)

[4.2.2. BlocBuilder 13](#_Toc120275125)

[4.2.3. BlocProvider 14](#_Toc120275126)

[4.2.4. BlocListener 14](#_Toc120275127)

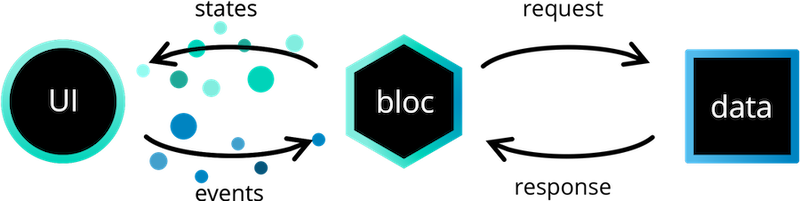
[4.2.5. Tài liệu đọc thêm 14](#_Toc120275128)

[5. Tài liệu tham khảo 15](#_Toc120275129)

# Khái niệm

## BLoC là gì?

* BLoC là mẫu thiết kế được Google công bố vào năm 2018.
* BLoC đại diện cho **B**usiness **Lo**gic **C**omponent.
* BLoC giúp phân tách tầng logic và tầng giao diện, từ đó giúp việc coding trở nên nhanh chóng, dễ kiểm thử và tái sử dụng.
* BLoC được thiết kế trên 3 giá trị chính:
  + Đơn giản: Dễ hiểu và dễ sử dụng
  + Mạnh mẽ: Tạo ra các ứng dụng phức tạp bằng cách kết hợp các component nhỏ
  + Có thể kiểm thử: Dễ dàng kiểm thử các thành phần của ứng dụng
* BLoC nhận đầu vào là một sự kiện, xử lý logic, và kết quả đầu ra là một trạng thái.



## Các khái niệm liên quan trong BLoC

### Stream

* Stream là luồng dữ liệu bất đồng bộ.
* Hình dung đơn giản, một ống nước có nước chảy qua, ống nước chính là Stream, và nước chính là dữ liệu bất đồng bộ.
* Sử dụng StreamController để thao tác với Stream.
* Khi cần đẩy dữ liệu vào Stream, sử dụng thuộc tính sink.
* Publish dữ liệu ra bên ngoài, sử dụng thuộc tính Stream.
* Để transform, điều chỉnh hoặc các thao tác xoá, cập nhật, sử dụng StreamTransformer.
* Có 2 loại Stream
  + Single subscription stream:
    - Chỉ lắng nghe 1 lần
    - Khi thực hiện lắng nghe lại, toàn bộ dữ liệu thực hiện ở ban đầu sẽ bị bỏ lỡ, phần còn lại trở nên vô nghĩa.
    - Khởi tạo:

|  |
| --- |
| StreamController counterController = new StreamController<int>(); |

* + Broadcast stream:
    - Cho phép nhiều chỗ có thể lắng nghe sự kiện thay đổi của stream cùng một lúc.
    - Sau khi huỷ đăng ký, vẫn có thể lắng nghe lại sau đó.
    - Tuy nhiên, cần lưu ý huỷ các lắng nghe không hoạt động nhằm tránh lãng phí bộ nhớ (memory leak).
    - Khởi tạo:

|  |
| --- |
| StreamController counterController = new StreamController<int>.broadcast(); |

### Event

* Event là những tương tác đến từ nhiều phía, từ phía giao diện, các tương tác của người dùng, thay đổi kết nối mạng, …
* Event sẽ thông báo cho BLoC biết cần thực hiện gì.

|  |
| --- |
| enum CounterEvent { increment, decrement } |

### State

* State đại diện cho các thông tin sẽ được xử lý bởi phía giao diện. Các widget sẽ dựa vào state để thay đổi trạng thái của chính nó.

|  |
| --- |
| part of 'counter\_bloc.dart';  class CounterState {  int counterValue;  CounterState({  required this.counterValue,  });  } |

### Cubit

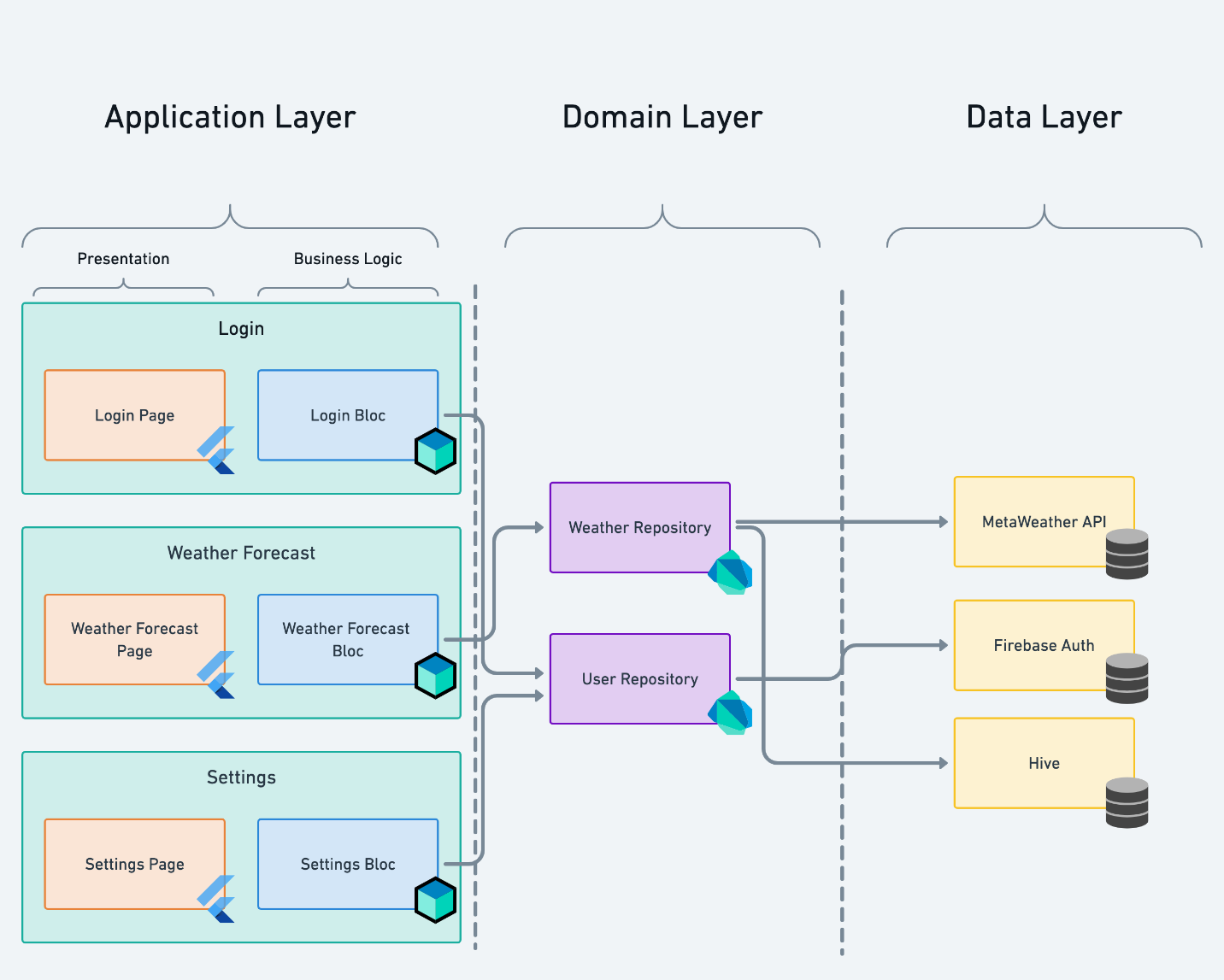
* Cubit là phiên bản đơn giản của BLoC.
* Trong Cubit, dữ liệu đầu vào là các function, và chúng không được xem là Stream.
* Chỉ có 1 Stream duy nhất chính là các State.



|  |  |  |
| --- | --- | --- |
|  | **Cubit** | **BLoC** |
| Dữ liệu đầu vào | Functions | Events |
| Boilerplate code | Không | Có |
| Có thể truy vết sự kiện | Không | Có |

## Kiến trúc ứng dụng khi sử dụng BLoC

* Khi sử dụng BLoC, ứng dụng được phân thành 3 lớp:
  + Presentation: hiển thị các widget dựa theo state được trả về từ tầng Business Logic.
  + Business Logic: xử lý các dữ liệu đầu vào từ tầng Presentation, kết nối với Repository để truy xuất dữ liệu cần thiết, và trả về state cho tầng Presentation.
  + Data
    - Repository: quản lý các data provider.
    - Data Provider: chịu trách nhiệm thực hiện các thao tác CRUD dữ liệu.



# Tại sao, khi nào dùng BLoC

## Tại sao nên sử dụng BLoC?

* Giúp tách biệt tầng logic và tầng giao diện, phân chia ứng dụng thành các tầng khác nhau 🡪 Code có tổ chức, dễ bảo trì và kiểm thử hơn.
* BLoC giúp đáp ứng các nhu cầu cần thiết của lập trình viên:
  + Nắm được trạng thái của ứng dụng tại một thời điểm bất kỳ.
  + Dễ dàng kiểm thử với mọi trường hợp nhằm đảm bảo ứng dụng đáp ứng và phản hồi đúng yêu cầu.
  + Theo dõi từng tương tác của người dùng trên ứng dụng → đưa ra quyết định dựa trên dữ liệu.
  + Làm việc hiệu quả nhất có thể và tái sử dụng các component trong và ngoài ứng dụng.
  + Các lập trình viên có thể cùng làm việc trên 1 code base duy nhất dựa trên các mẫu thiết kế và các quy tắc.
  + Phát triển các ứng dụng nhanh và reactive.
* BLoC chỉ thực hiện rebuilt lại các widget thực sự cần thiết khi có thay đổi về trạng thái → hiệu năng tốt hơn.
* Dễ dàng thay đổi trạng thái.

## BLoC và các cách quản lý trạng thái khác trong Flutter

|  |  |
| --- | --- |
| **Cách quản lý**  **trạng thái** | **Phù hợp với** |
| setState | * Các ứng dụng nhỏ và đơn giản. * Người mới làm quen với Flutter. |
| Provider | * Người mới làm quen với Flutter. * Lập trình viên không quen thuộc với Stream. |
| BLoC | * Lập trình viên quen thuộc với Stream. * Ứng dụng có cấu trúc phức tạp, quy mô từ trung bình trở lên. |
| Redux | * Các dự án lớn. * Có global app state như xác thực người dùng, ngôn ngữ, đơn vị tiền tệ, … |
| GetIt |  |
| MobX | * Các ứng dụng nhỏ và đơn giản. |
| GetX | * Các ứng dụng có yêu cầu sử dụng ít bộ nhớ. |

## Khi nào nên sử dụng BLoC?

* Khi muốn tách biệt tầng logic với tầng UI.
* Việc quản lý trạng thái trở nên phức tạp.
* Yêu cầu ứng dụng có thể mở rộng (scalable) và bảo trì (maintainable).
* Các widget cần tương tác với tầng dữ liệu.
* Xây dựng ứng dụng có hiệu năng tốt hơn và giảm tình trạng lạm dụng bộ nhớ.

# Ứng dụng của BLoC trong Flutter

## Tính hiệu quả

### Tách giao diện ra khỏi business logic

* Một trở ngại khi lập trình di động sử dụng Flutter là Flutter không dùng một ngôn ngữ trung gian nào dành cho việc dựng giao diện.
* Không như Android dùng XML cho giao diện, các lập trình viên Flutter phải định nghĩa giao diện người dùng ngay bên trong code.
* Do đó, business logic (quy trình nghiệp vụ ứng dụng) có thể bị xen lẫn vào trong các đoạn code phần giao diện, làm mã nguồn ứng dụng trở nên khó quản lý và bảo trì.
* Vì vậy việc tách giao diện ra khỏi business logic là rất cần thiết. Với BloC, chúng ta có thể làm được việc này một cách rất hiệu quả
  + Sử dụng Stream để quản lý và lan truyền các thay đổi về State cho các thành phần bên trong ứng dụng
  + Giúp giao diện và business logic trở nên độc lập với nhau, dễ quản lý và bảo trì mã nguồn, hơn vì lập trình viên sẽ không cần quan tâm giao diện có những gì, cũng như lo sợ mình làm sẽ ảnh hưởng đến phần của người khác

### Quản lý State của ứng dụng hiệu quả hơn

* BLoC hỗ trợ quản lý state của ứng dụng một cách hiệu quả hơn
  + BLoC nhận vào các Event và trả về các State.
  + Các widget chỉ có nhiệm vụ lắng nghe và rebuild khi có sự thay đổi về State phát ra từ BLoC.
  + Tự bản thân các widget sẽ không làm thay đổi state của ứng dụng.
  + Ví dụ:
    - Ta có danh sách các món hàng, mỗi món hàng có một nút Yêu thích
    - Khi nhấn nút, thực hiện thêm món hàng vào Danh sách yêu thích và đổi màu nút đó
    - Hành động nhấn nút là một event, BLoC nhận event và thực hiện các thao tác nghiệp vụ tương ứng (đọc ghi database…) rồi trả về state.
    - Lúc này widget mới rebuild để đổi màu nút. Bản thân nó sẽ không tự đổi màu mà phải dựa trên state lắng nghe được từ BLoC
* BloC không phải cách duy nhất để quản lý State của ứng dụng. Flutter cung cấp sẵn cho ta một số cách. Tuy nhiên chúng đều tồn tại các nhược điểm khác nhau
* setState
  + Đây là cách thông dụng nhất, dễ tiếp cận và làm quen, phù hợp với những ứng dụng rất nhỏ
  + Tuy nhiên, khi dùng setState, toàn bộ widget tree đều được rebuild dù cho chỉ có duy nhất một widget thay đổi state
  + Ngoài ra, dùng setState không giúp ích trong việc tách business logic ra khỏi giao diện

|  |
| --- |
| setState(() { \_value = newValue }); |

* InheritedWidget
  + Giải quyết được một vấn đề của setState đó là không rebuild toàn bộ widget tree khi có sự thay đổi về state
  + Tuy nhiên, các State được khai báo là final, do đó chúng không thể thay đổi về sau.
  + Ngoài ra, InheritedWidget không có các phương thức dispose để giải phóng tài nguyên, có thể gây rò rỉ bộ nhớ.

|  |
| --- |
| class FrogColor extends InheritedWidget {  const FrogColor({  super.key,  required this.color,  required super.child,  });  **final Color color;**  static FrogColor of(BuildContext context) {  final FrogColor? result = context.dependOnInheritedWidgetOfExactType<FrogColor>();  assert(result != null, 'No FrogColor found in context');  return result!;  }  @override  bool updateShouldNotify(FrogColor old) => color != old.color;  } |

### Tăng khả năng scale của ứng dụng

* Các thành phần trên giao diện không liên lạc và trao đổi thông tin trực tiếp với nhau, mà chúng chỉ có thể liên lạc trực tiếp với BLoC
* Do đó, sẽ rất dễ dàng trong việc thêm các tính năng mới cho ứng dụng, vẫn đảm bảo được tính dễ quản lý và bảo trì mà không cần viết lại code, tái cấu trúc dự án, hay làm giảm performance

### Dễ kiểm thử

* Việc sử dụng BLoC giúp giảm chi phí, thời gian, công sức khi thực hiện kiểm thử. Ta có thể dễ dàng dự đoán và kiểm tra được các state của ứng dụng, nhờ giao diện và business logic đã tách ra.
* Khi có sự thay đổi về state, BLoC sẽ thông báo đến các widget để thực hiện việc rebuild giao diện. Vì vậy, trong trường hợp cần kiểm thử một chức năng hoạt động đúng hay không, ta có thể dễ dàng kiểm tra từ BLoC mà không cần kiểm tra từng widget riêng biệt.

## Mức độ phổ biến

* Trừ những ứng dụng cực kỳ đơn giản, chỉ có từ 3 đến 5 màn hình trở xuống. Còn lại đa phần các ứng dụng đều áp dụng BLoC
* Do BLoC là một design pattern, nên nó không bị ràng buộc về ngôn ngữ, framework.
* BLoC không chỉ dùng được riêng cho Flutter mà còn cả AngularDart và React

## Ưu, nhược điểm của việc sử dụng BLoC

### Ưu điểm

* Dễ dàng tách giao diện ra khỏi business logic
* Performance tốt
  + Giao diện độc lập với business logic
  + Hạn chế việc rebuild toàn bộ widget tree khi có sự thay đổi về state
* Dễ kiểm thử, tái sử dụng code
* Tăng khả năng scale của ứng dụng, dễ dàng thêm các tính năng mới

### Nhược điểm

* Chỉ có hiệu quả rõ rệt khi dùng trong các ứng dụng lớn vừa đủ
* Phụ thuộc rất nhiều vào Stream, do đó code bị boilerplate

# Các thư viện về BLoC mà Flutter hỗ trợ

## Bloc

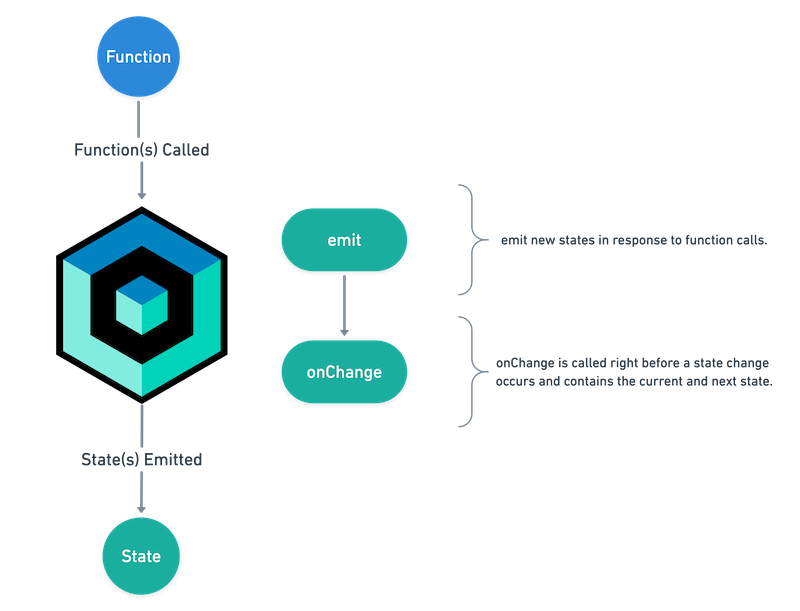
### Giới thiệu chung

* Là một package dành cho Dart hỗ trợ cho việc cài đặt BloC
* Giúp đơn giản hóa việc cài đặt BLoC, giúp lập trình viên có thể tập trung nhiều hơn cho phần business logic

### Cubit

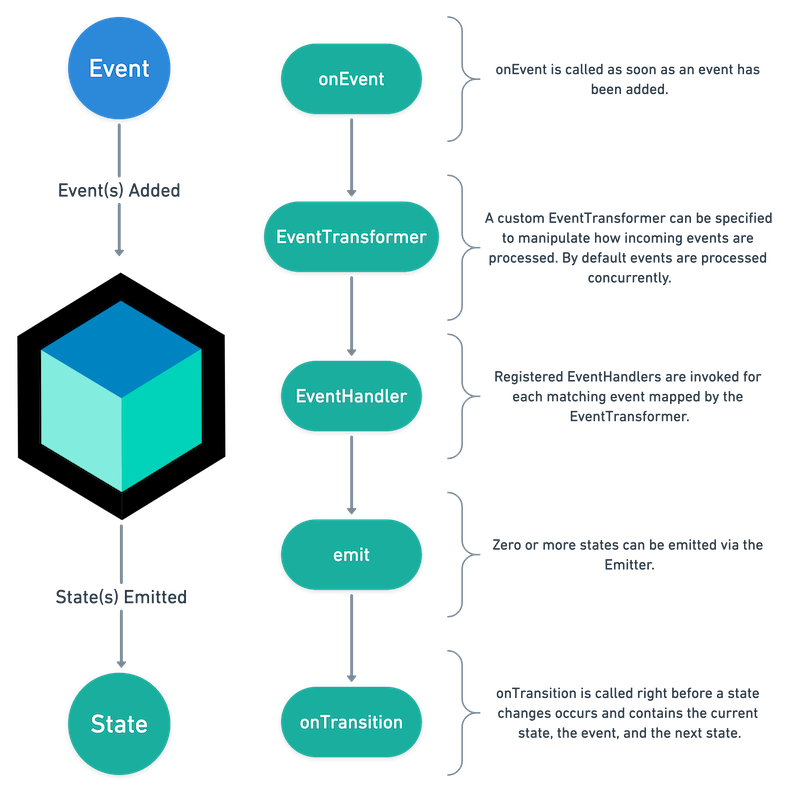
* Là một class kế thừa từ BlocBase, và có thể được kế thừa bởi bất kỳ loại state nào.
* Các thức hoạt động
  + Yêu cầu initial state (state ban đầu), là state trước khi phương thức emit được gọi
  + State hiện tại có thể được truy cập bởi getter, và được cập nhật trong một hàm gọi phương thức emit.
  + Phương thức emit trả về state mới khi hàm tương ứng được gọi.
  + Hàm onChange
    - Được gọi ngay trước khi state thay đổi, chứa state hiện tại về state kế tiếp
    - Dùng để quan sát thay đổi cho một cubit
  + Hàm onError
    - Dùng để quan sát lỗi xảy ra trong một Cubit (nếu có)
  + Hai hàm onChange và onError đều có thể override

|  |
| --- |
| class CounterCubit extends Cubit<int> {  CounterCubit() : super(0);  void increment() => emit(state + 1);  @override  void onChange(Change<int> change) {  super.onChange(change);  print(change);  }  @override  void onError(Object error, StackTrace stackTrace) {  print('$error, $stackTrace');  super.onError(error, stackTrace);  }  } |



### Bloc

* Bloc cũng kế thừa từ BlocBase, nghĩa là nó sẽ có các phương thức tương tự Cubit, ví dụ: onChange, onError…
* Tuy nhiên, không như Cubit – dựa vào việc gọi hàm, Bloc nhận vào các events và chuyển đổi chúng thành các state mới.
* Các thức hoạt động
  + onEvent được gọi khi có event truyền vào Bloc
  + Các events được xử lý đồng thời bởi EventTransformer
  + EventHandler được gọi để emit các state (nếu có) cho mỗi event tương ứng
  + Cuối cùng, onTransition được gọi ngay trước khi state được cập nhật, trong đó chứa: state hiện tại, event, state kế tiếp



|  |
| --- |
| abstract class CounterEvent {}  class CounterIncrementPressed extends CounterEvent {}  class CounterBloc extends Bloc<CounterEvent, int> {  CounterBloc() : super(0) {  on<CounterIncrementPressed>((event, emit) => emit(state + 1));  }  @override  void onEvent(CounterEvent event) {  super.onEvent(event);  print(event);  }  @override  void onChange(Change<int> change) {  super.onChange(change);  print(change);  }  @override  void onTransition(Transition<CounterEvent, int> transition) {  super.onTransition(transition);  print(transition);  }  @override  void onError(Object error, StackTrace stackTrace) {  print('$error, $stackTrace');  super.onError(error, stackTrace);  }  } |

### Tài liệu đọc thêm

* Đọc thêm về bloc tại: <https://pub.dev/packages/bloc>

## flutter\_bloc

### Giới thiệu chung

* Là một package hỗ trợ cho việc tích hợp Bloc và Cubit vào trong Flutter
* Đơn giản hóa việc cài đặt BLoC, giúp lập trình viên có thể tập trung nhiều hơn vào phần business logic

### BlocBuilder

* Tương tự StreamBuilder, có tham số là bloc và builder nhận vào một hàm.
* Hàm builder thực hiện việc build các widget với state tương ứng và có thể được gọi nhiều lần.
* Nếu tham số bloc bị bỏ qua, BlocBuilder tự động tìm kiếm dựa trên BlocProvider và BuildContext.
* Tham số buildWhen dùng để kiểm tra xem có nên rebuild hay không, dựa vào state hiện tại và state trước đó

|  |
| --- |
| BlocBuilder<BlocA, BlocAState>(  bloc: BlocA,  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  }  ) |

### BlocProvider

* Cung cấp bloc cho các widget con của nó thông qua BlocProvider.of<BlocA>(context)

|  |
| --- |
| BlocProvider(  create: (BuildContext context) => BlocA(),  child: Container(…),  ); |

### BlocListener

* Tương tự BlocBuilder
* Thực hiện lắng nghe các thay đổi từ Bloc và gọi hàm listenter tương ứng.
* Hàm listener chỉ được gọi một lần khi state thay đổi, và nó không trả về widget. Do đó, thường được dùng để gọi các thao tác như Navigator, hiện dialog,…

|  |
| --- |
| BlocListener<BlocA, BlocAState>(  bloc: BlocA  listener: (BuildContext context) {  // YOUR CODE HERE  },  child: Container(…),  ); |

### Tài liệu đọc thêm

* Đọc thêm về bloc tại: <https://pub.dev/packages/flutter_bloc>

# Tài liệu tham khảo

|  |  |
| --- | --- |
| [1] | “10 Flutter State Management library, that you must know about it,” [Trực tuyến]. Available: https://shirsh94.medium.com/10-flutter-state-management-library-that-you-must-know-about-it-1e472c99255e. |
| [2] | “Asynchronous programming Streams | Dart,” [Trực tuyến]. Available: https://dart.dev/tutorials/language/streams. |
| [3] | “bloc | Dart Package,” [Trực tuyến]. Available: https://pub.dev/packages/bloc. |
| [4] | “Bloc State Management Library,” [Trực tuyến]. Available: https://bloclibrary.dev/#/. |
| [5] | “Flutter State Management: What to Choose- Provider, BLoC, or Redux?,” [Trực tuyến]. Available: https://medium.com/dhiwise/flutter-state-management-what-to-choose-provider-bloc-or-redux-214160adbae4. |
| [6] | “List of state management approaches | Flutter,” [Trực tuyến]. Available: https://docs.flutter.dev/development/data-and-backend/state-mgmt/options#general-overview. |
| [7] | “State management in Flutter using the BLoC design pattern,” [Trực tuyến]. Available: https://blog.logrocket.com/state-management-flutter-bloc-pattern/. |
| [8] | “Stream trong Flutter,” [Trực tuyến]. Available: https://viblo.asia/p/stream-trong-flutter-E375zQrRlGW. |
| [9] | A. Syal, “BLoC pattern in Flutter | FlutterDevs,” 3 May 2019. [Trực tuyến]. Available: https://medium.flutterdevs.com/bloc-pattern-in-flutter-part-1-flutterdevs-128f90059f5c. |
| [10] | P. Darji, “State management in Flutter using the BLoC design pattern,” 19 May 2021. [Trực tuyến]. Available: https://blog.logrocket.com/state-management-flutter-bloc-pattern/. |
| [11] | S. Cherednichenko, “How to Implement the BLoC Architecture in Flutter: Benefits and Best Practices,” 19 May 2022. [Trực tuyến]. Available: https://www.mobindustry.net/blog/how-to-implement-the-bloc-architecture-in-flutter-benefits-and-best-practices/. |
| [12] | “Getx Vs Bloc Which Is Faster With Code Examples,” September 2022. [Trực tuyến]. Available: https://www.folkstalk.com/2022/09/getx-vs-bloc-which-is-faster-with-code-examples.html. |
| [13] | “Top 5 Benefits of Using BLoC Pattern,” 25 February 2022. [Trực tuyến]. Available: https://www.mitrais.com/news-updates/top-5-benefits-of-using-bloc-pattern/. |
| [14] | K. Makadiya, “Flutter State Management: What to Choose- Provider, BLoC, or Redux?,” 11 May 2022. [Trực tuyến]. Available: https://medium.com/dhiwise/flutter-state-management-what-to-choose-provider-bloc-or-redux-214160adbae4. |
| [15] | Flutter, “InheritedWidget class,” 23 November 2022. [Trực tuyến]. Available: https://api.flutter.dev/flutter/widgets/InheritedWidget-class.html. |
| [16] | bloclibrary.dev, “bloc | Dart Package,” 6 August 2022. [Trực tuyến]. Available: https://pub.dev/packages/bloc. |
| [17] | bloclibrary.dev, “flutter\_bloc | Flutter Package,” 14 August 2022. [Trực tuyến]. Available: https://pub.dev/packages/flutter\_bloc. |