Là framework ngôn ngữ dảt build tầng frame work

# Privider

Google khuyến cáo sử dụng

Bloc

Mapeventtostate input là event

Output state update

State update thông qua event

Stream function trả về giá trị nhiều lần theo thứ tự (yield) có chỉ một lần gọi

Bloc là generic class với hai thành phần là events và int

Luôn cho state ban đầu không để null

Khi nào nên dùng app có nhiều state thay đổi liên tục, dùng kiến trúc bloc dưới dạng event state

Dạng Login signup có nhiều state về authentication…

Ưu điểm:

- app chạy mượt khi chuyển state

- dễ mở rộng, debug, quản lý

Nhược điểm:

- khó hiểu cho người mới học

- cần làm các công đoạn tỉ mỉ

Khi đã làm quen nó sẽ là một công cụ mạnh triển khai các ứng dụng vừa và lớn

Không phải có bloc là tất cả các màn hình đều dùng bloc, một số màn hình

Bloc và flutter\_bloc\_builder luôn đi cùng với nhau

Blocbuilder là một widget yêu cầu một Bloc và một builder function.

Bloclistener

Blocconsumer, blocprovider

# 2. Học qua các bài demo

## 2.1. flutter counter

Sử dung Bloc library

Key topics

- quan sát state thay đổi với BlocObserver

- BlocProvider, flutter widget cung cấp 1 bloc cho its children

- BlocBuilder, flutter widget xử lý building widget in response to new states

- Sử dụng Cubit thay vì (instead of) Bloc

- Thêm events với context.read

Take a look at… hãy xem…

**BlocObserver** giúp quan sát mọi state changes trong app

### Tạo BlocObserver

Điều đầu tiên chúng ta sẽ tạo BlocObserver, BlocObserver sẽ giúp chúng ta quan sát tất cả state changes trong ứng dụng.

**class CounterObserver extends BlocObserver {**

**@override**

**void onChange(BlocBase bloc, Change change) {**

**super.onChange(bloc, change);**

**print('${bloc.runtimeType} $change');**

**}**

**}**

**//main.dart**

**void main() {**

**BlocOverrides.runZoned(**

**() => runApp(const CounterApp()),**

**blocObserver: CounterObserver(),**

**);**

**}**

Chỉ Text widget wrap trong BlocBuilder

child: BlocBuilder<CounterCubit, int>(  
 builder: (context, state) {  
 return Text(  
 '$state ',  
 style: textTheme.headline2,  
 );  
 },  
),

vì đó là widget duy nhất cần được rebuild để đáp ứng với những thay đổi state trong CounterCubit. Tránh wrap các widget không cần thiết để khỏi phải rebuild thừa khi state thay đổi.

tổng kết:

Chúng tôi đã tách lớp trình bày (presentation layer) khỏi lớp logic nghiệp vụ. CounterView không biết điều gì sẽ xảy ra khi người dùng nhấn một button; nó chỉ thông báo cho CounterCubit.

CounterCubit không biết điều gì đang xảy ra với trạng thái (giá trị bộ đếm); nó chỉ đơn giản là phát ra các trạng thái mới để đáp ứng với các phương thức đang được gọi.

## 2.2. timer

**how to build a timer application using the bloc library**

mục tiêu

Ngăn chặn việc rebuild không cần thiết với Equatable.

Tìm hiểu cách sử dụng StreamSubscription trong một Bloc.

Ngăn chặn rebuild không cần thiết với buildWhen.

## 2.3. Infinite list

Bây giờ chúng ta đã có model Post object, hãy bắt đầu làm việc trên Thành phần Business Logic Component (bloc).

Post events

Tóm lại, PostBloc sẽ nhận PostEvents và chuyển đổi chúng thành PostStates. Chúng tôi đã xác định tất cả các PostEvents (PostFetched), vì vậy cần xác định thêm PostState.

Post States

## 2.6. todos

Ứng dụng của chúng tôi bao gồm ba lớp chính:

* data layer
* domain layer
* feature layer
  + presentation/UI (widgets)
  + business logic (blocs/cubits)

**Data Layer**

Lớp này là lớp thấp nhất và chịu trách nhiệm truy xuất dữ liệu thô từ các nguồn bên ngoài như databases, API, v.v.

Ở bài này, data layer bao gồm các packages todos\_api và local\_storage\_todos\_api.

**Domain Layer**

Lớp này kết hợp (combines) một hoặc nhiều data providers và áp dụng " business rules" cho dữ liệu.

Mỗi thành phần trong lớp này được gọi là kho lưu trữ (repository) và mỗi repository quản lý một domain duy nhất.

Các packages trong lớp repository thường chỉ tương tác với data layer. Trong ví dụ này, repository layer bao gồm gói todos\_repository.

**Feature Layer**

Trong mỗi feature, state của feature cùng với bất kỳ logic nghiệp vụ nào đều được quản lý bởi các blocs.

Blocs không tương tác hoặc tương tác với nhiều repositories. Blocs phản ứng với các events và emit (phát ra) các state để kích hoạt các thay đổi về UI.

Các widget trong mỗi feature thường chỉ phụ thuộc vào bloc tương ứng và hiển thị UI dựa vào current state.

UI có thể thông báo cho bloc về input của người dùng thông qua các events. Trong ví dụ này, ứng dụng sẽ bao gồm các tính năng home, todos\_overview, stats và edit\_todos.

## 2.4. Login

Key topics

* [**BlocProvider**](https://bloclibrary.dev/#/flutterbloccoreconcepts?id=blocprovider), Flutter widget cung cấp một bloc children của nó.
* BlocBuilder, Flutter widget xử lý việc xây dựng widget con theo các state mới.

2.5. weather

Key topics

**BlocConsumer:**

- Hiển thị trình tạo (builder) và trình lắng nghe (listener) để phản ứng với các states mới.

- Tương tự như BlocListener và BlocBuilder lồng vào nhau nhưng làm giảm số lượng boilerplate cần thiết.

- Chỉ nên được sử dụng khi cần rebuild UI và thực hiện các phản ứng khác đối với các thay đổi states trong bloc.

- Lấy BlocWidgetBuilder và BlocWidgetListener bắt buộc và một bloc tùy chọn, BlocBuilderCondition và BlocListenerCondition.