**State Manager:**

1. **Stream:**
   1. **Stream la gi:**

- Stream là một chuỗi các events bất đồng bộ.

- Event có thể là một data, cũng có thể là một error, hoặc một trạng thái done.

- Emit: Hành động phát ra Event của Stream

- Step in stream:

+ create stream

+ subscrible stream: để có thể nhận

+ StreamSubscription: Hàm listen ở trên sẽ trả về một đối tượng StreamSubscription. StreamSubscription giúp ta điều khiển stream. Ta có thể pause stream, resume stream và hủy stream.

* 1. **Stream Operators:**

**- Function return Future:**

+ elementAt: get được phần tử tại index bất kỳ trong Stream

+ length : get được số lượng phần tử có trong stream.

+ firstWhere : get được phần tử đầu tiên thỏa mãn một điều kiện được chúng ta cho trước.

+ join : nối tất cả element thành 1 String duy nhất, giữa các element được ngăn cách bởi 1 ký tự separator do chúng ta truyền vào.

+ await for : tính tổng các phần tử trong Stream một cách bất đồng bộ. Vì là tính bất đồng bộ nên nó sẽ tính nhanh hơn dùng for thông thường rất nhiều.

**- Function return Stream:**

transform (biến đổi) một Stream sang một Stream

+ map : truyền vào 1 hàm, hàm này sẽ giúp ta biến đổi từng element của Source Stream.

+ where : lọc ra những phần tử trong Source Stream thỏa mãn điều kiện cho trước.

+ takeWhile : emit ra các phần tử CÓ thỏa mãn điều kiện cho trước cho đến khi có một phần tử KHÔNG thỏa mãn điều kiện thì nó dừng Stream.

+ skipWhile : bỏ qua, không emit ra các phần tử CÓ thỏa mãn điều kiện cho trước cho đến khi có một phần tử KHÔNG thỏa mãn điều kiện được emit ra. skipWhile một khi đã emit được 1 phần tử đầu tiên rồi thì nó sẽ không skip bất cứ phần tử nào nữa.

+ distinct : skip (bỏ qua) các phần tử mà chúng equal (bằng) với phần tử đã được emit trước đó (the previous element). distinct còn cho phép chúng ta có thể định nghĩa lại thế nào là bằng nhau.

* 1. **Handle Error:**

- function handleError dùng để bắt lỗi trong Stream.

- handleError hơn onError: cung cấp cho ta 1 optional param là hàm test, ở hàm test ta có thể check xem cái lỗi đó có đáng để ta catch hay không, nếu không đáng thì ta có thể cho ném lỗi đó luôn không cần catch (tương tự rethrow lỗi).

- Return true: catch error

- return false: ko catch error

- default là return true.

* 1. **Stream controller:**

- là 1 controller bên trong có 1 stream và controller này giúp ta điều khiển stream đó dễ dàng.

- StreamController()

- đây là một con đường khác giúp ta tạo ra 1 Stream và thoải mái emit events và nhận events bất cứ thời điểm nào chúng ta muốn.

- StreamController chứa 2 đối tượng: sink(giúp chúng ta push events đến stream) và stream (giúp chúng ta nhận events từ sink)

* 1. **Broadcast StreamController:**

- Với Broadcast StreamController chúng ta có thể tạo ra bao nhiêu thằng lắng nghe Stream đó cũng được.

- StreamController.broadcast()

* 1. **Rxdart:**

- là một package cung cấp thêm một số class và function để tăng cường sức mạnh cho Stream.

**- Những class Stream của RxDart:**

+ MergeStream :

có 2 stream gọi là Stream 1 và Stream 2 chạy song song nhau và 2 stream đó được merge lại thành 1 MergeStream, các event trong MergeStream được emit theo thứ tự đúng y hệt dòng thời gian của Stream 1 và Stream 2. Có thể merge nhiều stream với nhau.

+ ZipStream :

stream1 stream2 được merge lại với nhau theo cách:

Phần tử thứ 1 của Stream 1 sẽ kết hợp với phần tử thứ 1 bên Stream 2

Phần tử thứ 2 của Stream 1 sẽ kết hợp với phần tử thứ 2 bên Stream 2

...

+ TimerStream :

Truyền vào 1 value, và 1 duration. Sau 1 khoảng thời gian duration đó Stream mới emit value được truyền vào.

+ RangeStream :

Truyền vào 2 biến: startInclusive và endInclusive để tạo thành 1 range. RangeStream sẽ emit một chuỗi các số nguyên liên tiếp trong range đó.

+ RetryStream :

Truyền vào 1 stream. Khi stream được truyền vào gặp lỗi RetryStream sẽ cho retry lại nó. Chúng ta có thể truyền thêm 1 biến count chính là số lần được phép retry. Nếu không truyền biến count thì nó sẽ retry mãi mãi.

**- extension function của RxDart:**

+ debounceTime :

truyền vào 1 Duration, nếu Source Stream emit ra 1 event mà sau Duration nó không emit tiếp 1 event khác thì Output Stream mới emit event đó ra

cực kỳ hữu ích để làm tính năng live search cho app (real time search) bởi vì app sẽ tránh spam server do call API quá nhiều.

+ onErrorResumeNext :

Hàm này truyền vào 1 Stream gọi là recoveryStream đi ha. Khi Source Stream gặp lỗi thay vì nó emit cái event lỗi đó thì nó lại vào emit các phần tử trong recoveryStream.

+ interval :

truyền vào 1 duration. Cứ sau khoảng thời gian duration đó thì nó mới emit ra 1 event.

+ concatWith :

Merge 2 stream lại thành 1 ConcatStream và emit tất cả event trong lần lượt từng stream một. Sau khi emit xong tất cả event của Stream 1, thì ConcatStream vẫn chờ x giây (trong hình) mới emit phần tử đầu tiên của Stream 2 chứ không emit ngay lập tức.

+ distinctUnique :

dùng để emit các phần tử không trùng nhau trong Source Stream.

**- Subjects:**

Subjects bản chất là StreamController.

+ BehaviorSubject :

là một Broadcast StreamController.

mỗi khi có một listener mới subscribe thì nó lập tức emit phần tử mới nhất vừa được add vào controller.

+ ReplaySubject :

là một Broadcast StreamController.

mỗi khi có một listener mới subscribe thì nó lập tức emit tất cả các phần tử đã được add vào controller

+ PublishSubject :

giống một Broadcast StreamController thông thường. Có nghĩa là lúc subscription thứ 2 được tạo ra nó sẽ không chôm chĩa bất cứ data nào trước đó của controller mà nó chờ có data mới được add vào controller nó mới nhận.

**- CompositeSubscription :**

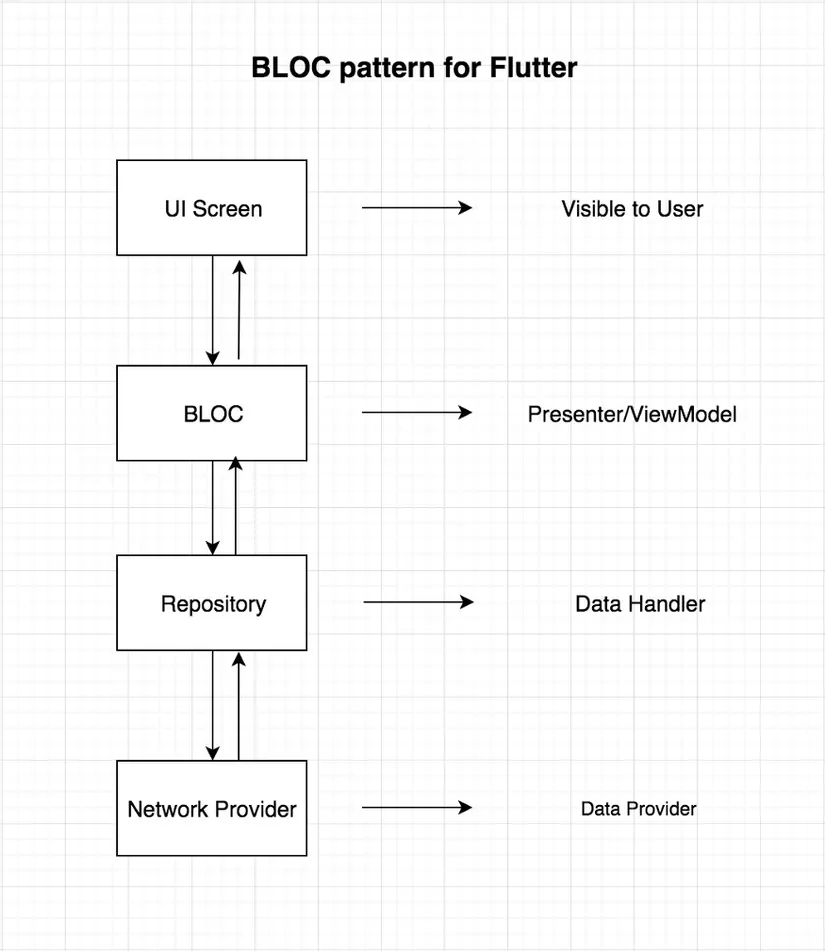
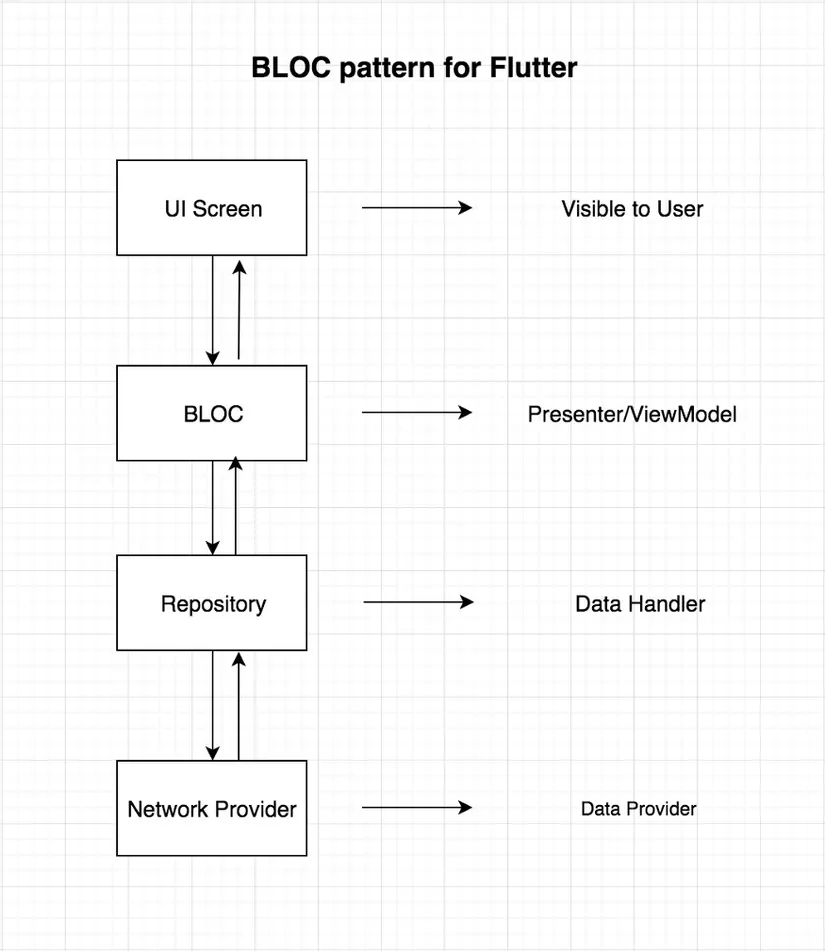
+ giống như một cái vùng chứa tất cả StreamSubscription, để sau này chúng ta tiện cancel tất cả StreamSubscription cùng một lúc.

<https://www.dbestech.com/tutorials/flutter-bloc-pattern-examples>

1. **BloC:**

Data provider

Data handle



Get data

Get data

event

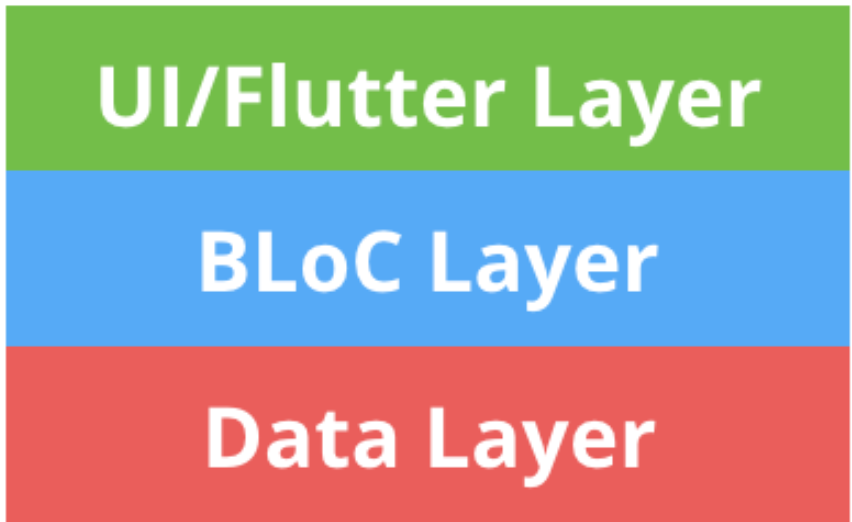
Network provider

Repository

bloc

UI screen

state



- Bussiness Logic component: tang chuyen xu ly logic cua app

-block patent: dữ liệu được chuyển từ BLOC đến UI hoặc ngược lại từ UI xuống BLOC duoi dạng ****STREAMS.****

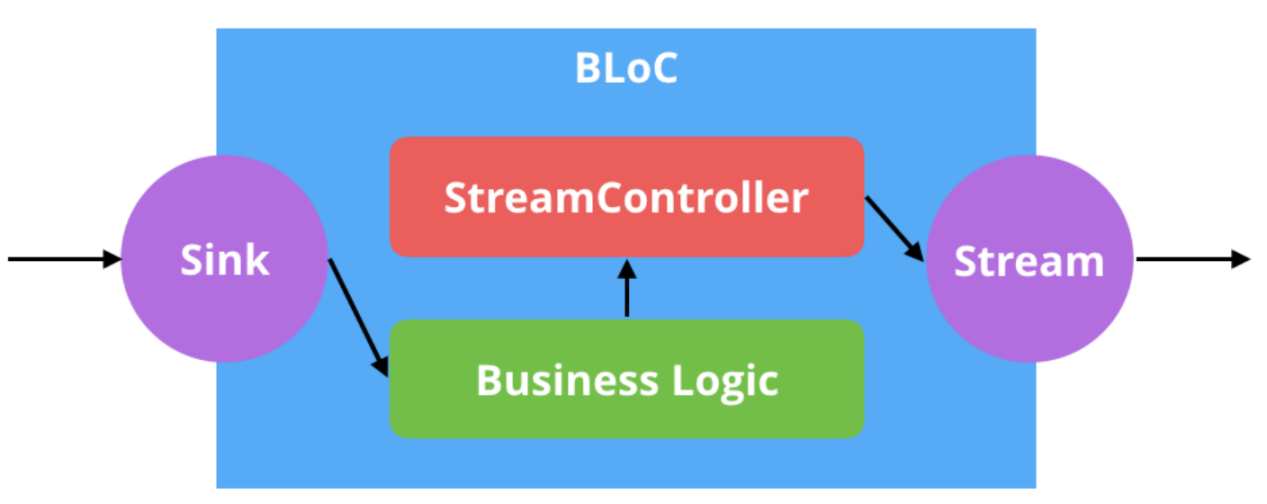
- Structure of bloc: bloc + event + state

- điểm mạnh của stream: có thể control đc flow data

- điểm yêú: xử lý streams khó hơn với listenable.

- Data layer: managing data models, connecting to back-end. It doesn’t contain UI

- BloC Layer: receives input events, processes business logic using the data layer and responds with output events to the UI layers or other listeners.



- là 1 lib theo bloc architecture, có các util widget là các base bloc class, giúp cho việc implement bloc trở nên dễ dàng hơn

- 3 impotant define: blocks, events, and states.

- States: is common, using for other screens.

- *equatable*: is lib to comparing 2 objects

1. *bloc*: contain function MapEventToStates: there is a event, how state will be changed from old state to new state.

+ create Bloc: create a class, using mapEventToState() to map event with change of state respectively.

*2. events:* will receive event and convert to states

+ create events: create a abstract class BlBlocCounterEvent > create event classes extend BlBlocCounterEvent.

*3. states:* contain states of app initial, success, failure:

Initial state: tell layer load a loading indicator

Success state: tell layer it has content.

Failure state:

+ create state: create a class BlBlocCounterState > init with states.

-using **context.read<TimerBloc>()** to access the **TimerBloc**

1. BlocProvider:

- BlocProvider will create a BLoC which extends Bloc.

- BlocProvider is usually called from your view. So it's also the entry point before you use Bloc.

- Dùng BlocProvider để update state lên UI

1. BlocBuilder:

- builds a widget when state is updated.

- You should wrap widget using BlocBuilder which update the widget based on state.

- Block Builder is called from your view.

1. mapEventToState:

- If you use Bloc, everything is translated as event from user side. User interaction in the view is translated as event to Bloc. Bloc receives this events and map them to states using mapEventToState.

- mapEventToState should be implemented within the class which extends Bloc.

1. BlocProvider.of<T>:

- This method allows widgets or your UIto access the Bloc instances.

- You will use Bloc instances to show on the UI or make changes to the values or properties or fields of Bloc.

- with this you can access the instances of Bloc.

1. **Provider:**

- dung Listenable để lắng nghe sự thay đổi của c Widget trên .

- Điểm mạnh: dễ học, thân thiện với người mới.

- Điểm yếu:

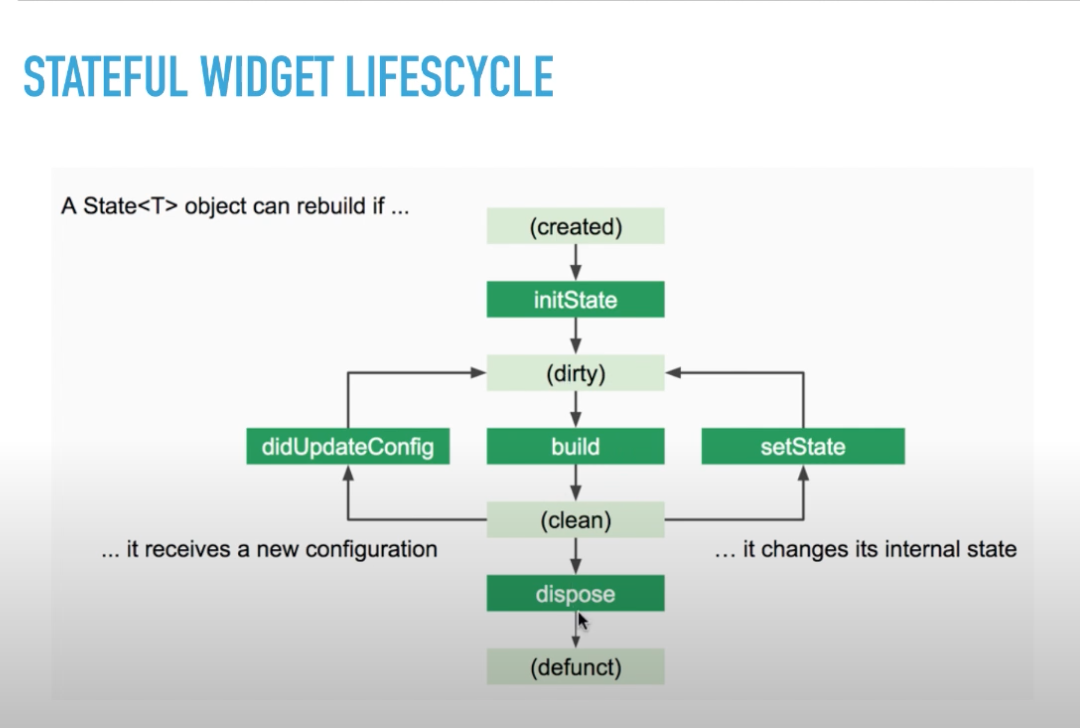
+ ChangeNotifier :sử dụng ObserverList, add/remove listener là add/remove callback vào ObserverList

+ notifyListeners(): duyệt 1 vòng for trong ObserverList và invoke nó lên: ChangeNotifier tối ưu với 1 số lượng Listener nhỏ. Add/remove Listener là O(N), dispatching notifications là O(N2).

=> không hiệu quả với project lớn.

1. **Mobx:**

**I. StateFullWidget:**



**II. Material:**

- is one design style which is published by Google.

- is special widget, contain main other widgets: button, scafold,...

- defining all about: button, calendar, text, animations, ....

**III. Scarfold:**

- is special widget, it implement basic Material Widgets: appBar, Floatingbutton, menuSide, bottomBar

**IV. Notes:**

*- Text widget:*

-handle action click on Text(): wrap in *GestureDetector()*

- TextField():

-Get state of TextField(): *TextEditingController()*

- Error : Connection closed before full header was received :

+cause:

1. problem with the URL: using Uri.parse() instead of Uri.http() when creating the URL. Uri.parse(): allows you to specify the full URL, including the scheme (https), host name (<DATABASE\_NAME>.firebaseio.com), and path (/products.json).

2. the security rules in Firebase do not allow the connection.

3. the URL is not a valid one.

4. failure in the server.

- StreamBuilder: là một widget trong Flutter, dùng để đọc dữ liệu từ Stream và xây dựng phần logic cho UI, lắng nghe sự thay đổi của Stream và làm mới lại UI. gồm 2 phần chính:

+ stream: là một stream asynchronous data.

+ builder: xây dựng logic cho UI.