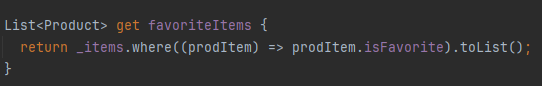
**FLUTTER DOCUMENT**

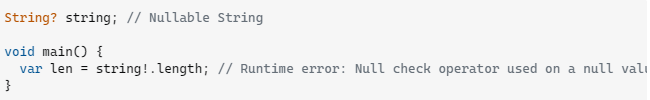
**- PHAM TRONG HUNG -**

**1.Kiểu dữ liệu trong Dart / Flutter**- Late: từ khóa dùng để khai báo biến sẽ được gán giá trị sau khi nó được tạo  
- Phải được khởi tạo trước khi sử dụng ko sẽ sinh ra lỗi LateInitializationError  
  
- const: từ khía dùng để khai báo biến có giá trị không thể đổi sau khi khởi tạo  
- Phải khai báo biến khởi tạo với Const =>   
- Const không được dùng cho các biến có giá trị lúc compile  
VD: const name; (sai) / final name; (đúng)

- final: từ khóa dùng để khai báo biến có giá trị không đổi sau khi khởi tạo

- Trong Flutter thì dấu \_ khai báo cho 1 method là private

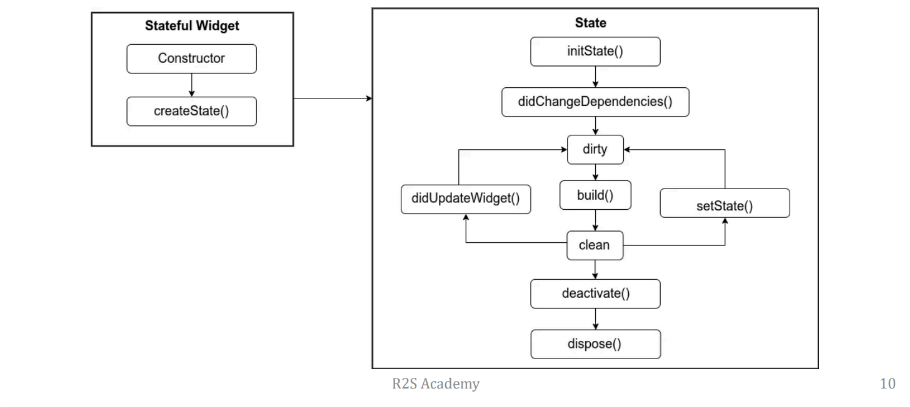
- Từ khóa get trong Flutter có nghĩa là Getter  
(Lấy danh sách item yêu thích)  


- ? & ! operator: Toán tử check null của riêng Flutter   
- ? thông báo cho trình biên dịch biết rằng giá trị này có thể null  
- ! giá trị được gán dấu ? khi muốn gọi thì thêm dấu ! để ép giá trị từ null sang không null  
- Nếu giá trị null thì thường sẽ báo lỗi : NULL CHECK USED ON NULL VALUE   
- Fix bằng cách chuyển dấu ? thành ?? ‘<default\_value>’;  
VD:   


Fix:



**2.Life Cycle của StateFulwidget trong Flutter**



**3.FutureBuilder**

- Là một widget được sử dụng để xây dựng giao diện dựa trên trạng thái của một Future.

- Cho phép hiển thị giao diện chờ (CircularProgressIndicator,…) trong khi dữ liệu được tải và

hiển thị kết quả Future hoàn thành

- Hữu ích với các tác vụ bất đồng bộ như truy vấn dữ liệu từ API hoặc database

- Khi bắt đầu Future chưa có giá trị (ở trạng thái pending)

- Sau khi hoàn thành, Future sẽ:

+ Trả về kết quả nếu tác vụ thành công

+ Trả về lỗi nếu không thành công

Cấu trúc của một FutureBuilder:

|  |
| --- |
| FutureBuilder<T>(  future: myFuture, // Đây là tương lai mà bạn muốn theo dõi, có thể là 1 list data fetch từ API  builder: (context, snapshot) {  if (snapshot.connectionState == ConnectionState.waiting) {  return CircularProgressIndicator(); // Hiển thị Loading  } else if (snapshot.hasError) {  return Text("Error: ${snapshot.error}"); // Hiển thị lỗi  } else {  return Text("Data: ${snapshot.data}"); // Hiển thị dữ liệu }  }, ) |

**4.ConnectionState**

- Là một enum trong Flutter được sử dụng để biểu thị trạng thái của một kết nối hoặc một tiến

trình đồng bộ như Future hoặc Stream

- Khi đang sử dụng FutureBuilder hoặc StreamBuilder => connectionState giúp bạn biết được

Trạng thái hiện tại của quá trình BĐB để cập nhật giao diện

- **connectionState** có 4 trạng thái chính:

+ **ConnectionState.none**: Không có kết nối nào được khởi tạo. Thường là trạng thái ban đầu

trước khi Future hoặc Stream bắt đầu chạy.

+ **ConnectionState.waiting**: Đang chờ kết nối hoàn thành, thường xuất hiện khi Future đang

thực thi hoặc dữ liệu từ Stream đang chờ tải về.

+ **ConnectionState.active**: Được sử dụng cho Stream, khi dữ liệu đang liên tục được cung

cấp.Nó ít được sử dụng với Future vì Future chỉ hoàn thành một lần.

+ **ConnectionState.done**: Kết nối đã hoàn thành. Trong trường hợp của Future, điều này có

nghĩa là Future đã trả về dữ liệu hoặc lỗi.

|  |
| --- |
| if(snapshot.connectionState == ConnectionState.waiting) {  return const Loader();  } |

**5.ListView.builder**

- itemCount : Dựa theo chiều dài của Arrays/Lists mà chạy ra đủ builder

- itemBuilder: (context, index) { }

- Được bọc trong 1 widget cha có kích thước được xác định, vd: SizedBox(height:..) hoặc là Expand

- Nếu bị báo lỗi “Can hit test render….” thì là lỗi do thiếu widget cha bọc

- scrollDirection để điều chỉnh hướng của ListView - vd: Axis.horizontal (ngang) & Axis.vertical (dọc)

**6.BoxShadow cho Container**

- Sử dụng bên trong BoxDecoration

- boxshadow chứa một mảng BoxShadow chứa các thuộc tính để dễ dàng chỉnh sửa

- Ví dụ code:

child: Container(

margin: *const* EdgeInsets.all(15),

decoration: BoxDecoration(

borderRadius: BorderRadius.circular(10),

border: Border.all(color: CupertinoColors.inactiveGray),

boxShadow: [BoxShadow(

color: Colors.black.withOpacity(0.5), *// Shadow color*

offset: *const* Offset(0,4), *// Shadow position*

blurRadius: 6, *// Shadow blur*

spreadRadius: 2, *// Shadow spread*

)]

),

**7.ButtonStyle**

**padding: MaterialStateProperty.(....)**

- Dùng để định nghĩa padding bên trong button

Code:

ButtonStyle(

padding: MaterialStateProperty.all(*const* EdgeInsets.all(10)),

backgroundColor: MaterialStateProperty.all(

selectedType == type ? Colors.grey[300] : Colors.transparent,

),

),

**ValueChanged - Kiểu dữ liệu - Hàm callback**

**ValueChanged<String> onCategorySelected**

- Là một hàm callback được định nghĩa sẵn trong SDK, cụ thể là ValueChanged<T>

- Nhận vào 1 giá trị kiểu T và không trả về giá trị nào cả

=> typedef ValueChanged<T> = void Function(T value);

- Tham số đầu vào là String, Int, Double,...

Khi sử dụng ValueChanged<String>, nó tương đương với một hàm có chữ ký như sau:

= void Function(String value);

- Ví dụ thực tế:

|  |
| --- |
| // Định nghĩa một callback nhận vào một chuỗi và không trả về gì ValueChanged<String> onCategorySelected = (String category) {  print('Category selected: $category'); };  // Gọi hàm callback onCategorySelected('Running'); |

**8.Container with onPress() - Muốn nhấn vào Container**

- Bọc 1 widget như GestureDetector hoặc InkWell

- Cả 2 widget này đều cung cấp các sự kiện nhấn (tap) và các sự kiện tương tác khác

**Kích thước theo màn hình**

|  |
| --- |
| Size size = MediaQuery.of(context).size; double screenWidth = size.width; // Chiều rộng của màn hình double screenHeight = size.height; // Chiều cao của màn hình |

**9.Tạo animation và animation khi chuyển trang**

- Sử dụng thư viện flutter\_animated và page\_transition

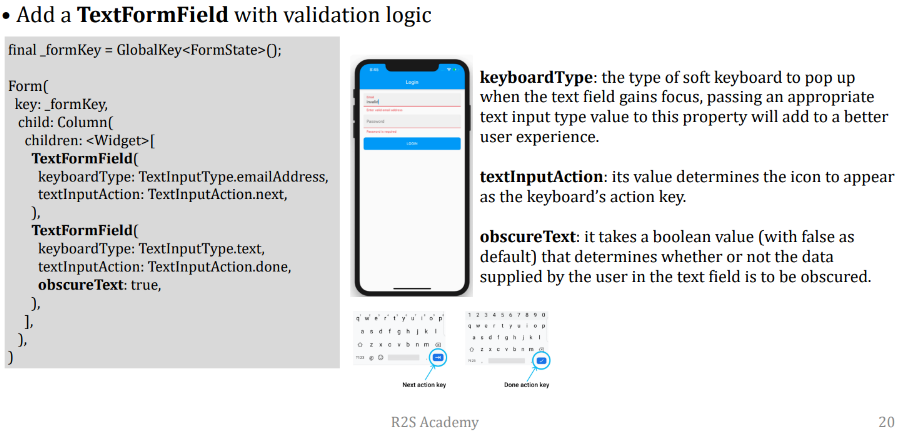
- Lên gg lấy và cập nhật tùy phiên bản

**10.Future<List<data\_type>>**

- Sử dụng là một quản lý bất đồng bộ   
- Future là quản lý BĐB  
- Late khai báo 1 giá trị sẽ được gán sau khi khởi tạo

**11.TextFormField**

- Sử dụng như 1 text field

  
- keyboardType để khai báo kiểu dữ liệu của bàn phím

- obscureText: <bool\_value> cho phép coi dữ liệu trong TextForm

- Có thể sử dụng label, icon, inline, hint\_text, error\_text

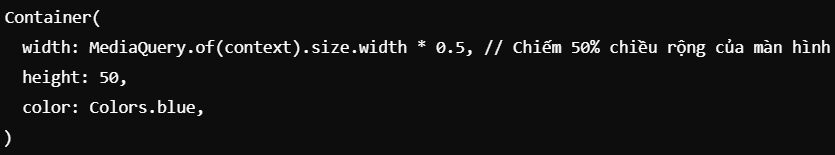
- Dùng decoration: InputDecoration để CSS cho TextForm, có thể validator bằng validator: (value) {return RegExp}

**12.Width & Height**

- Sử dụng số pixel cụ thể, hoặc sử dụng double.infinity hoặc MediaQuery.of(context).size.width

- Sử dụng double.infinity sẽ lấy hết kích thước của widget cha

- Sử dụng MediaQuery.of(context).size.width/height để lấy được kích thước chiều rộng cũng như là độ cao của màn hình để dễ dàng chỉnh sửa =)) nói thẳng ra là responsive



**13.Kết nối CDSL sử SQLite**

- Sử dụng SQLite để xử lý dữ liệu cục bộ cần lưu trữ trên máy, thường dùng cho ứng dụng điện thoại

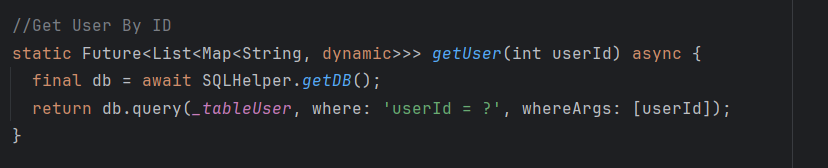
- MySQL thì sử dụng để lưu trữ số lượng lớn dữ liệu và xử lý 1 số lượng lớn người dùng cho ứng dụng

- Tạo bảng trực trong SQL\_Helper.dart

CreateUser.dart



GetUserByID

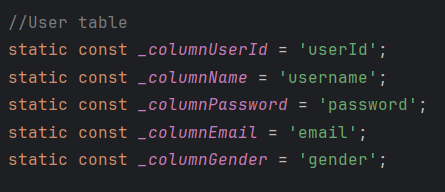
  
- whereArgs gán giá trị cho dấu ? trong mệnh đề where

db.query(<tên bảng>) mặc định là câu lệnh select nếu không có điều kiện

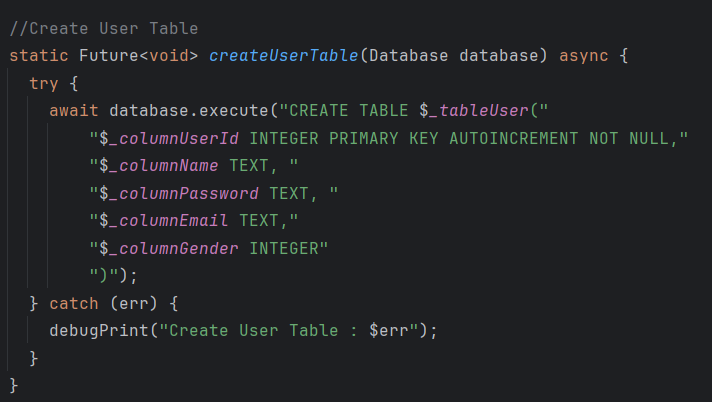
\* Quy trình khai báo và tạo bảng  
Bước 1: Khai báo tên database



Bước 2: Khai báo tên các cột dữ liệu trong bảng



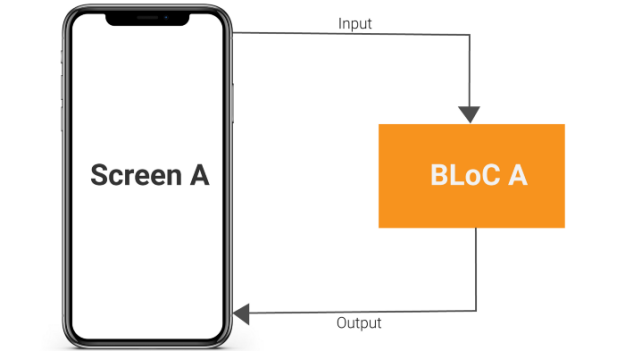
Bước 3: Câu lệnh query tạo bảng



Bước 4: gọi lệnh khởi tạo onCreate để tạo bảng



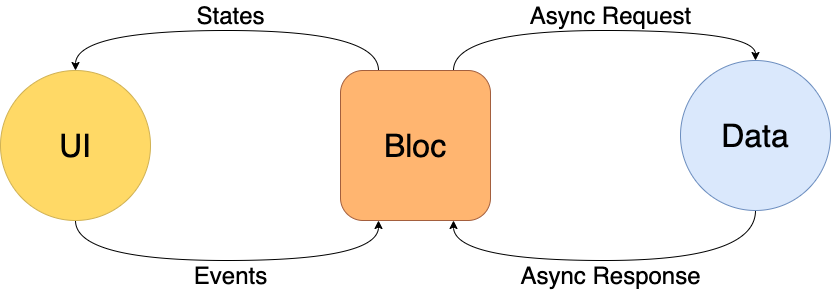
**14.Kết nối CDSL sử MySQL**

**State Management with BLOC**- Sử dụng Bloc Pattern để tạo nên 1 project thống nhất và dễ bảo trì  
- Tách các xử lý nghiệp vụ ra khỏi cái widget UI  


- Input được truyền vào bloc từ bloc trả state cho UI để render UI  


- Input là 1 EVENT và output là 1 STATE

- Quy trình cơ bản 1 project có sử dụng Bloc State Management



- Chúng ta sẽ có 4 thành phần chính là UI, State, Event, Bloc

-

**Stream Controller**

- StreamController đơn giản là 1 controller bên trong có stream

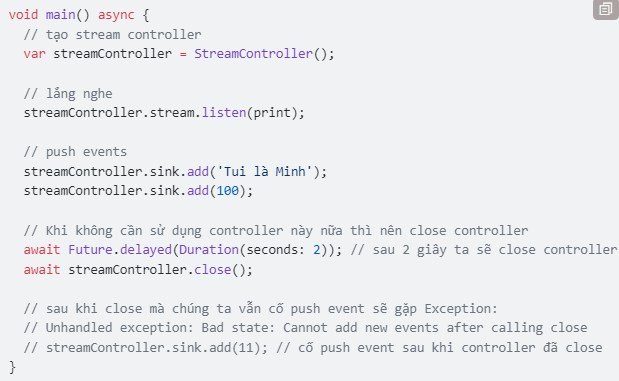
- Giúp chúng ta quản lý stream đó rõ ràng

- Stream Controller có 2 đối tượng là sink và stream

+ sink giúp push event đến stream

+ stream giúp nhận event từ sink

- Để tạo ra nhiều thằng lắng nghe stream ta cũng có thể sử dụng BroadcastController



- Sau khi đã đóng controller thì ta không thể push event từ sink vào stream nữa

Function của Bloc State Management

EventHandling: Lưu trữ hành động của user (event). Bloc nhận và tạo ra quá trình convert map event thành state

State Emission: Dựa trên những events được nhận thì Bloc phát hành ra state mới sau đó thì update render UI dựa trên state

BlocProvider: Là 1 widget dùng để lắng nghe sự thay đổi state và build lại UI khi state mới phát hành

Bloc Listener: Là 1 widget dùng lắng nghe sự thay đổi state trong sự phản hồi của state mà k render UI

**BlocProvider:** Sử dụng được chỉ cho 1 thằng cha và trong thằng cha có con

**MultiBlocProvider**: Sử dụng cho nhiều thằng cha nhiều context

- Yield dùng để phát ra trạng thái, nói đúng hơn thì là phát ra 1 emit. Từ đó giao diện người dùng (UI) lắng nghe các thay đổi từ BLOC và tự cập UI

- on<Event> thay cho method mapEventToState

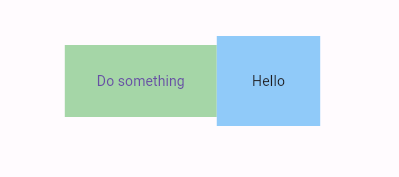
- Khi sử dụng context.read<BLOC>.add(event) => sẽ được vào stream của event và xử lí tại BLOC

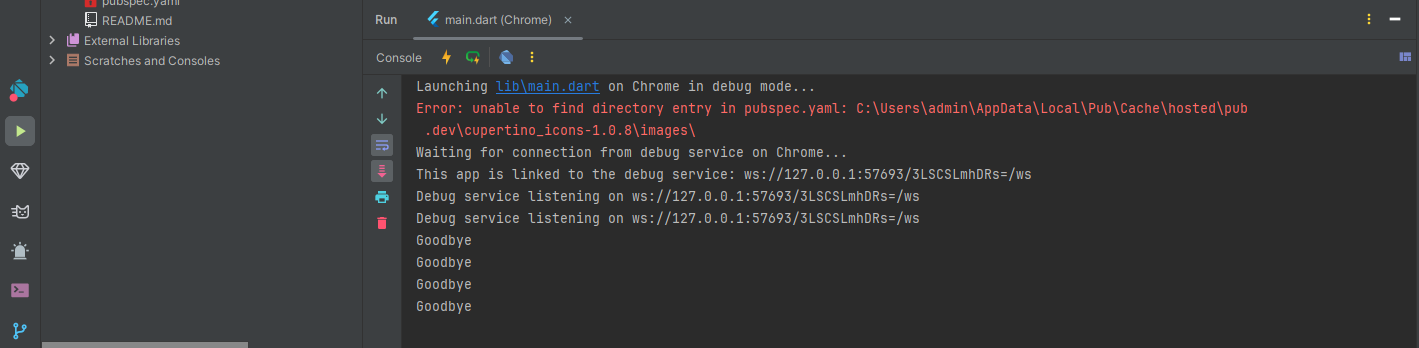
- Cubit thì cũng khá là đơn giản

**Sử dụng Provider để Quản Lí Trạng Thái**

- Provider widget không thể dùng để lắng nghe được các trạng thái nếu như sử dụng Provider để bọc cha của các widget

- ChangeNotifierProvide mới dùng để lắng nghe sự thay đổi trong data model





- Khi nhấn vào button<Do\_something> thì trong console sẽ in ra giá trị là “goodbye”

- Nhưng giao diện vẫn chữ ‘Hello’ vì Provider<widget> không thể lắng nghe những giá trị thay đổi

=> Sử dụng ChangeNotifierProvider để lắng nghe những giá trị thay đổi mà render UI

Điều này cung cấp cho bạn quyền truy cập vào notifyListeners() và bất kỳ lúc nào bạn gọi notifyListeners() thì ChangeNotifierProvider sẽ được thông báo và tất cả các widget bên trong Consumers sẽ được rebuild lại.

- Kết thúc của một xử lý logic thì phải có notifyListeners() để rebuild lại UI

- Nếu như widget không cần thiết để cập nhật thì có thể sử dụng Provider.of rồi đưa listen về false

