Báo Cáo Khóa học Futter

The Complete 2020 Flutter Development Bootcamp with Dart

Created by Dr. Angela Yu

Học viên: Trần Phú Nguyện

Mục lục

[I. Dart 3](#_Toc38035808)

[1. Basic: 4](#_Toc38035809)

[2. Variable: 4](#_Toc38035810)

[3. Function: 4](#_Toc38035811)

[4. Access Modifier: 5](#_Toc38035812)

[5. Generic type: 5](#_Toc38035813)

[6. String: 5](#_Toc38035814)

[7. Collection 5](#_Toc38035815)

[8. OOP 5](#_Toc38035816)

[ Class: 6](#_Toc38035817)

[ Tính chất cơ bản: 6](#_Toc38035818)

[ 4 tính chất trụ cột: 6](#_Toc38035819)

[ Interface 6](#_Toc38035820)

[ Mixins 6](#_Toc38035821)

[ Lớp Object 6](#_Toc38035822)

[ Toán tử cascade. 6](#_Toc38035823)

[9. Asynchoronous: 7](#_Toc38035824)

[10. Stream: 7](#_Toc38035825)

[11. Exception: 7](#_Toc38035826)

[12. Null 7](#_Toc38035827)

[13. Import 7](#_Toc38035828)

[14. Random: 7](#_Toc38035829)

[16. Composition vs Inheriatance: 7](#_Toc38035830)

[17. Các Nguyên Lý ( Principle) khi lập trình: 7](#_Toc38035831)

[18. Khác 7](#_Toc38035832)

[II. Flutter: 7](#_Toc38035833)

[1. Basic 8](#_Toc38035834)

[2. Widget tree 8](#_Toc38035835)

[3. Vì Sao cần đến 3 tree (Widget, Element, Render): 9](#_Toc38035836)

[4. Statefull và Stateless Widgets: 9](#_Toc38035837)

[5. StatefulWidget Lifecycle 10](#_Toc38035838)

[6. Build method: 11](#_Toc38035839)

[7. Build Context: 12](#_Toc38035840)

[8. Routes: 12](#_Toc38035841)

[9. HTTP 12](#_Toc38035842)

[10. Animation 12](#_Toc38035843)

[11. Hot Reload và Hot Restart: 13](#_Toc38035844)

[12. Cách custom font: 13](#_Toc38035845)

[13. Theme 13](#_Toc38035846)

[14. State Management: 14](#_Toc38035847)

[15. Khác 14](#_Toc38035848)

[III. Widgets: 15](#_Toc38035849)

[1. Tính chất chung: 15](#_Toc38035850)

[2. MaterialApp: 15](#_Toc38035851)

[3. Scaffold: 15](#_Toc38035852)

[4. Text 15](#_Toc38035853)

[5. Container (standart) 15](#_Toc38035854)

[6. Column/ Row(standart) 16](#_Toc38035855)

[7. Expanded: 16](#_Toc38035856)

[8. Flexible: 16](#_Toc38035857)

[9. Icon 16](#_Toc38035858)

[10. Image: 16](#_Toc38035859)

[11. SizedBox: 16](#_Toc38035860)

[12. Safe Area: 16](#_Toc38035861)

[13. Visibility: 16](#_Toc38035862)

[14. GestureDetector: 16](#_Toc38035863)

[15. FloatingActionButton: 17](#_Toc38035864)

[16. RawMaterialButton: 17](#_Toc38035865)

[17. TextField( Material Design) 17](#_Toc38035866)

[18. ListView 17](#_Toc38035867)

[19. BottomSheet: 17](#_Toc38035868)

[20. Slider: 17](#_Toc38035869)

[21. DropdownButton <> CupertinoPicker 17](#_Toc38035870)

[22. CupertinoPicker 17](#_Toc38035871)

[23. StreamBuilder 17](#_Toc38035872)

[24. Material: 17](#_Toc38035873)

[25. Tips 18](#_Toc38035874)

[IV. Tips 18](#_Toc38035875)

[1. Dealing with distractions(Sự xao lảng): 18](#_Toc38035876)

[2. Để ngăn chặn sự thay đổi thuộc tính từ bên ngoài (Cách làm giống kotlin) 18](#_Toc38035877)

[V. Các website hữu ích và app tham khảo. 18](#_Toc38035878)

1. Dart
   1. Basic:
      * Everything is objects, kể cả function và null.
   2. Variable:
      * Là ngôn ngữ **static typing, x**ác định kiểu dữ liệu tại thời điểm **compile-time** ( Khác với **dynamic typing** (vd: JavaScript) xác định kiểu dữ liệu tại thời **runtime**.
      * So sánh với dynamic typing.
        1. Ngắn gọn: dynamic typing ngắn gọn hơn do không cần khai báo kiểu.
        2. Tính đúng đắn: trong static typing lỗi về kiểu (type errors) có thể được phát hiện tại thời điểm compile vì vậy chương trình sẽ an toàn và ít lỗi hơn.
        3. Hiệu năng: static typing sẽ nhanh hơn vì không mất thêm chi phí để kiểm tra kiểu lúc runtime.

* Nếu muốn là dynamic ta có thể dùng từ khóa *dynamic.*
* Nếu khi khai báo *var* không có kiểu dữ liệu phía sau có nghĩa nó là *dynamic* hay kiểu trả về hoặc parameter của function không định nghĩa kiểu dữ liệu thì nó cũng là *dynamic.*
* final
  + Được khởi tạo 1 lần duy nhất (run time).
  + Vd: final time = DateTime.now(). Working
* const
  + Được khởi tạo lúc compile.
  + Vd: const time = DateTime.now(). Not working
* static
  + Dùng khi sử dụng biến const thông qua tên class mà không cần khởi tạo class (class-wide) (Khởi tạo ở thời điểm compile time).
  + Tại sao cần static khi set property của class là const: Vì nếu không có static thì biến của class sẽ được khởi tạo ở run-time (khi mà tạo ra instance của class) mà const phải được khởi tạo ở compile time nên cần static để được khởi tạo mà không cần instance của class (ở compile time).
* Các property của class chỉ có thể là final mà k thể là const vì class được khởi tạo ở run time.
* Có thể tự suy ra (infer) kiểu dữ liệu từ giá trị nhận được.
* Khi khai báo ta nên khai báo kiểu dữ liệu cho dữ liệu.
* Hổ trợ top-level variable: Sử dụng ở bất kỳ đâu mà không cần thông qua class.
* Int và double đều là subtypes của num.
  1. Function:
     + => (fat arrow): được sử dụng khi thân hàm chỉ có 1 biểu thức (expression).
     + Hỗ trợ top-level function (vd *main()*): Có thể sử dụng ở bất kỳ đâu mà không cần thông qua tên class hay instance của class.
     + Có thể tạo 1 hàm bên trong 1 hàm khác (nested function).
     + *main* function: điểm bắt đầu khi chạy ứng dụng.
     + Optional named parameters:
       1. function({type variable}).
       2. Được dùng khi có nhiều parameter.
       3. @required: thuộc tính phải cung cấp. Chỉ có trong Flutter không có trong Dart.
     + Optional positional parameters:
       1. Function([type variable]).
       2. Được dùng khi có thể không cần truyền giá trị cho parameter đó.
     + Default parameter values.
  2. Access Modifier:
     + Không hổ trợ public, protected, private.
     + Nếu identifier (variable, function) bắt đầu với dấu gạch dưới ‘\_’ thì nó private trong library của nó. Mỗi file .dart được coi là 1 library.
  3. Generic type:
     + Có thể xây dựng các hàm, lớp mà không làm việc trên 1 kiểu dữ liệu cụ thể.VD: Class<T,E> function<T>(T t).
  4. String:
     + Nên dùng dấu ngoặc đơn.
     + Interpolation: ‘text${variable}text’.
     + Để tạo multi-line:’’’ text /n text ‘’’
     + Dùng “|”(backslash) nếu muốn dùng ký tự đặc biệt.
  5. Collection
     + Tính chất chung:
       1. Là lớp Generic.
       2. Là một Iterable.
     + List
       1. Chiều dài cố định (Fixed):
          1. List<T> list = List(length);
       2. Chiều dài có thể thay đổi:
          1. List<T> list = [];
       3. *reversed*: Trả về một đối tượng Iterable chứa các phần tử theo thức tự ngược lại.
  6. OOP
     + Class:
       1. Là một Blueprint.
       2. Bao gồm: Properties/fields và Method.
     + Tính chất cơ bản:
       1. single inheritance pattern ( chỉ cho phép extends 1 class).
     + 4 tính chất trụ cột:
       1. Abstraction
       2. Encapsulation
          1. Đóng gói dữ liệu giúp chương trình an toàn hơn. Lớp bên ngoài không thể tùy tiện thay đổi dữ liệu bên trong lớp.
       3. Inheritance
       4. Polymorphism
     + Interface
       1. Dart mặc định tất cả các lớp điều là interface nếu lớp đó được triển khai bởi lớp khác bằng từ khóa implements. Khi đó class triển khải phải override toàn bộ thuộc tính, phương thức của lớp được coi là interface.
     + Mixins
       1. Là một class nhưng không được sử dụng để tạo ra đổi tượng ( giống abstract class/ interface giống abstract hơi khi có thể implement các method trong đó, trong khi đó thì interface chỉ định nghĩa).
       2. Chứa các thuộc tính và phương thức để gộp vào một lớp khác.
       3. Là cách implement đa kế thừa.(thông qua composition)
       4. Làm cho chương trình dễ maintain hơn (chia sẻ capability với nhau mà không cần kế thừa).
       5. Nên sử dụng thay cho kế thừa.
     + Lớp Object
       1. là lớp cở của Dart mọi thứ đều mở rộng từ lớp này, kể cả null, function.
       2. hashCode: mã hash của object.
       3. toString(): trả về chuổi mô tả object.
       4. ==: so sánh theo hashCode 2 đổi tượng.
     + Toán tử cascade.
       1. Khi cần có một chuỗi tác vụ trên đối tượng (gọi phương thức, thiết lập thuộc tính) thay vì phải viết đầy đủ đối tượng thì chỉ cần viết nó một lần, các tương tác tiếp theo thay thế bằng “..”
       2. Ví dụ:

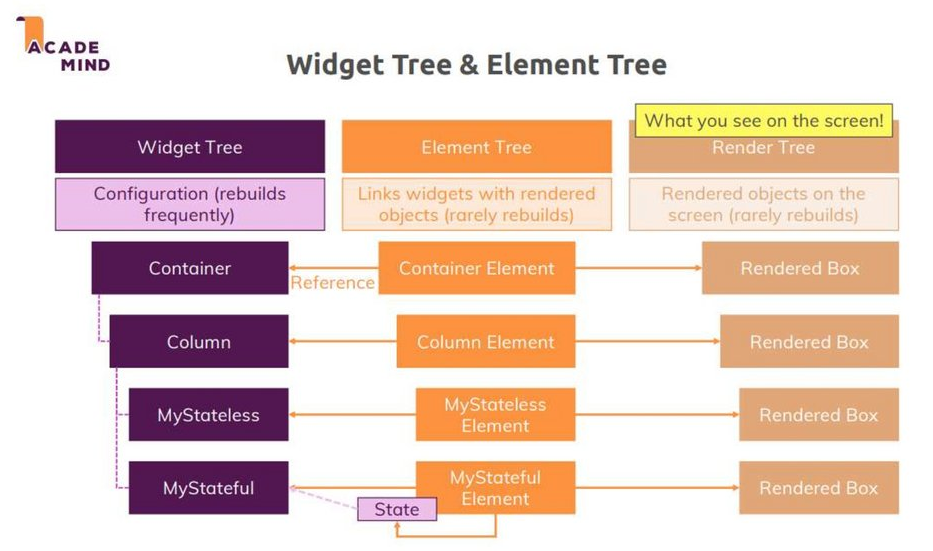
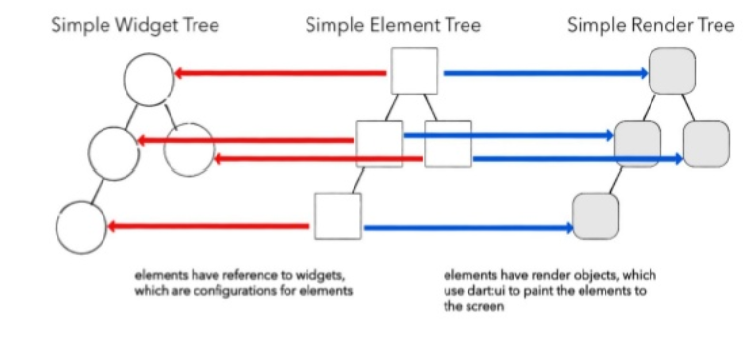
var table = new Table(1);

table

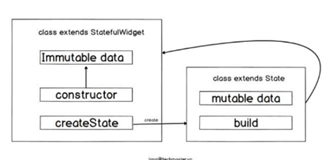
..calulateTotal() //Thay cho table.calulateTotal();

..length=100 //Thay cho table.length=100;

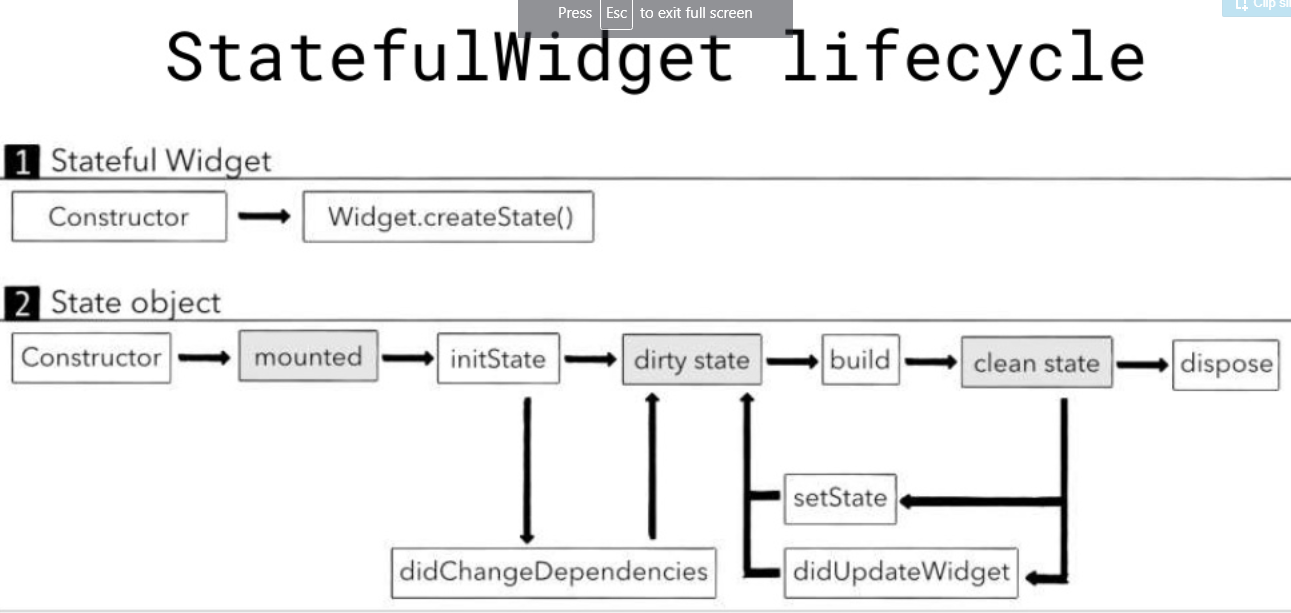
* 1. Asynchoronous:
     + Future (giống Promise bên JavaScript):
     + Async & await.
  2. Stream:
     + Một luồng dữ liệu bất đồng bộ.
  3. Exception:
     + Try
     + Catch
     + Throw: on ExceptionClass (chỉ lắng nghe những exeption từ class exception đó).
  4. Null
     + ?? : Null aware operator
     + someVarialbe ?? defaultValue.
  5. Import
     + *show* + class ( chỉ hiển thị class này trong library).
     + *hide* + class (ẩn class này trong library).
     + *as* + name (Khi muốn truy cập library phải dùng thông qua tiền tố name).
  6. Random:
     + Random().nextInt(max): trả ra giá trị từ 0 tới max – 1.
  7. Imperative vs Reactive:
     + Reactive:
       1. Prop Drilling: Chúng ta phải đi qua cả tree để đi đến level thấp nhất.
  8. Composition vs Inheriatance:
     + Flutter chủ yếu sử dụng composition (Widget) (Android sử dụng inheritance các widgets chủ yếu được inheritance từ Text).
     + Nên sử dụng composition hơn inheritance.
  9. Các Nguyên Lý ( Principle) khi lập trình:
     + DRY: Don’t Repeat Yourself.
  10. Khác
      + Dùng == để so sánh 2 object.

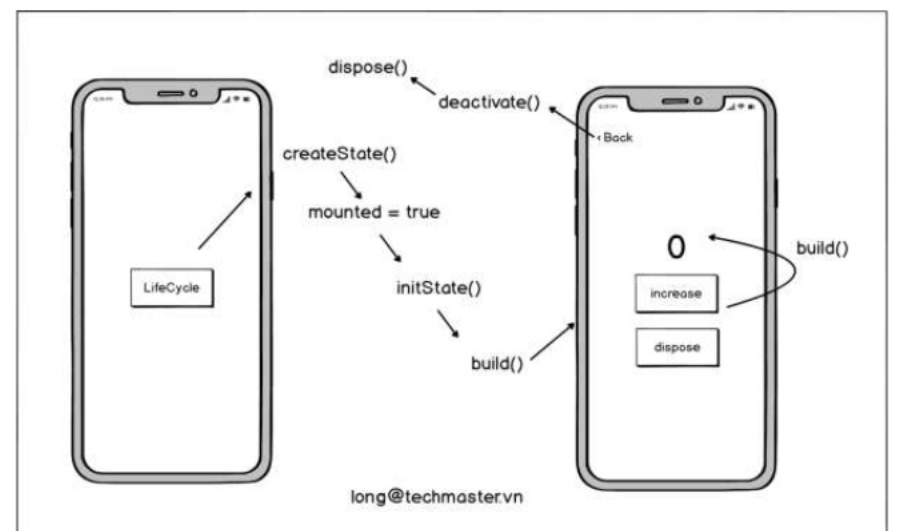
1. Flutter:
   1. Basic
      * Everything is widgets.
   2. Widget tree
      * Là cấu trúc đại diện cho toàn bộ ứng dụng( giống DOM).
      * Các bước chạy chương trình.
      * 
        1. Gọi hàm runApp(), Flutter build một Widget tree chứa các **widget**.
        2. Các **Element** được tạo bởi các widget. Sauk hi widget được build, framework gọi hàm Widget.createElement(this).
        3. Element gọi phương thức createRenderObject() để tạo ra các **RenderObject**, các **RenderObject** này cấu thành Render Tree.
        4. Mỗi một **Element** tham chiếu đến một **Widget** và một **RenderObject**

* Lưu ý: StateObject thực sự được quản lý bởi Element, không phải widget. Flutter render dựa trên Elements và State Object, chứ nó không liên quan gì đến widget.
  1. Vì Sao cần đến 3 tree (Widget, Element, Render):
     + **Element** không giữ cấu hình widget, nhưng có thể nhìn cấu hình chi tiết thông qua việc tham chiếu đến widget tương ứng.
     + **RenderObject** chưa toàn bộ logic để **rendering** widget, nên nó khá nặng để khởi tạo.
     + Khi widget trong tree thay đổi, Flutter sử dụng Elements để so sánh với widget mới với RenderObject đã tồn tại, nếu type của widget không đổi, Flutter sẽ không tạo lại các RenderObject nặng nề đó, thay vào đó chỉ cập nhật mutable configuration.
     + Widget rất nhẹ để khởi tạo, nên nó được dùng để mô tả state hiện tại.
  2. Statefull và Stateless Widgets:
     + Stateless widgets:
       1. Không có state.
       2. Không chấp nhận sự thay đổi bên trong nó.
       3. Khi widget cha thay đổi nó sẽ thụ động thay đổi theo.
     + Stateful widgets:
       1. Tạo ra UI mà state có thể thay đổi.
       2. Mối quan hệ giữa state và statefulWidget



* + - 1. Được sử dụng khi cần có sự thay đổi giao diện linh hoạt.
      2. Bản thân statefulWidget không chứa dữ liệu mutable mà State mới là nơi chưa dữ liệu mutable.
      3. State:
         1. 
         2. Trạng thái: Đại diện cho dữ liệu thay đổi trong vòng đời ứng ưng.
         3. Khi dữ liệu của stateful widget thay đổi, UI vẽ lại widget để phản ánh state mới.
         4. Phần giao diện chỉ đại diện cho state tại 1 thời điểm, thay đổi state -> thay đổi giao diện.
         5. Muốn truy cập tới Widget liên kết với state ta dùng thuộc tính *widget*.
         6. Có Global state được truy cập toàn app.
  1. StatefulWidget Lifecycle





* + - createState(): Khi xây dựng StatefulWidget, ngay lập tức gọi createdState().
    - initState(): Để khởi tạo state, chỉ được gọi một lần khi widget được tạo.
    - didChangeDependencies:
      1. Có thể được gọi một lần sau initState, cũng có thể được gọi lại trong Lifecycle.
      2. Tại thời điểm này, State được coi là “dirty”, đó là cách Flutter theo dõi những widget nào cần được rebuild, nó tự đánh dấu là “dirty”.
    - build():
      1. Xây dựng đầy đủ state object.
      2. Sau build(), state được đánh dấu là “clean” -> lifecycle hoàn thành một “single track”.
    - setState():
      1. Đánh dấu state ở trạng thái “dirty”, thực hiện re-run lại phương thức build().
      2. Một widget cha ở trên có thể yêu cầu xây dựng lại widget đang thao tác. Nếu vị trí sẽ được rebuild cùng loại và key widget, thì sẽ gọi didUpdateWidget(old widget). State cũng bị đánh dấu là “dirty”
      3. Khi không có hàm này biến vẫn sẽ được thay đổi nhưng không hiển thị lên màng hình.
      4. Nếu setState() cha được gọi thì sẽ vẽ lại tất cả các con.
    - Disposed()
      1. Xóa state object ra khỏi tree, xóa tài nguyên, widget sẽ không bao giờ được gọi lại.
  1. Build method:
     + Trả về một widget object chứa thông tin cấu hình để flutter render.
     + Được gọi khi:
       1. StatelessWidget
          1. Lần đầu tiên được thêm vào tree.
          2. Khi widget cha thay đổi.
          3. Khi các giá trị từ nguồn khác thay đổi. (Ví dụ 1 widget kế thừa sẽ phụ thuộc vào các thay đổi từ nơi nó kết thừa). (!! Tìm hiểu sau).
       2. StatefulWidget
          1. Sau khi gọi initState.
          2. Sau khi gọi didUpdateWidget ( Được gọi nếu widget cha thay đổi và phải xây dựng lại widget).
  2. Build Context:
     + Mọi widget trong Flutter được tạo từ build() method và method này đều lấy BuildContext làm đối số(argument).
     + Context tham chiếu đến vị trí của Widget trong cấu trúc tree.
     + Một Widget chỉ thuộc về một widget.
     + Mối widget có phương thức build() và context riêng.
     + BuildContext là cha của widget được trả về.
  3. Routes:
     + Navigator:
       1. Push:
          1. dùng MaterialPageRoute.
          2. Kiểu trả về là Future<T>: T là kiểu data khi màng hình được push trả về(khi pop(context,data)).
       2. PushNamed:
          1. Dùng Named route:

Thuộc tính routes trong MaterialApp.

‘/’: home screen.

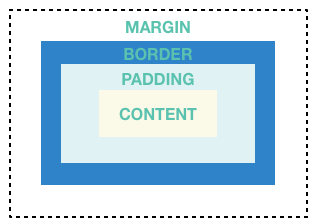
Nên dùng initalRoute thay cho home.

* + - 1. Pop:
         1. Trở về màng hình trước đó.
         2. Có thể gửi dữ liệu về màng hình trước đó.
  1. HTTP
     + Get(Future<Response>) response body.
     + Codes cheat sheet:
       1. 1\*\* Hold on
       2. 2\*\* Here you go
       3. 3\*\* Go away
       4. 4\*\* You fucked up
       5. 5\*\* I fucked up
     + Nên dùng “*as* http”( Specifying a library prefix: thêm tiền tố (sau *as*) khi dùng thư viện.) khi import http package.
  2. Animation
     + Bao gồm:
       1. A **ticker**.
       2. Animation Controller.
          1. Vsync (SingleTickerProviderStateMixin): Cung cấp **ticker** cho animation controller.
          2. forward: forward animation.
          3. reverse: đảo ngược animation.
          4. addListener: Lắng nghe value của animation thay đổi.
          5. duration: thời gian của animation (60 tick/second).
       3. Animation Value.
     + Class Animation:
       1. addStatusListener:
          1. completed: kết thúc forward animation.
          2. Dismissed: kết thúc reversed animation.
     + Các loại animation
       1. CurvedAnimation
       2. ColorTween.
       3. Animated\_text\_kit (package).
     + Class Hero Animation
       1. Thực hiện animation giữa hai màng hình.
       2. Bao gồm:
          1. 2 HeroWidgets ở 2 màn hình.
          2. Tên của tag chung giữa 2 màn hình.
     + Nên dispose animation để tránh memory leak.
     + Nên xác định kiểu của animation(generic Animation<T>) để tránh nhầm lẫn, lỗi khi tương tác với value.
  3. Hot Reload và Hot Restart:
     + Hot Reload
       1. Chỉ hoạt động trên Stateless hoặc statefull widgets.
       2. Chỉ gọi lại phương thức *build* của widgets.
       3. Sử dụng: Ctrl + S.
       4. Không mất dữ liệu (state) của app. (ví dụ: form, text, …).
       5. Giúp giảm thời gian phát triển ứng dụng.
     + Hot Restart
       1. Reset app (reset data (state) của app).
  4. Cách custom font:
     + 1. Import the font files.
       2. Định nghĩa font trong pubspec.
       3. Set font mặc định.
       4. Sử dụng font trong widget.
  5. Theme
     + Chỉnh sửa theme chung của app thông qua ThemeData
     + Cũng là một Widget.
  6. State Management:
     + Các cách để quản lý state:
       1. Bloc, MobX, Redux, Provider, InheritedWidget Scoped Model, Lifting State Up, MVC, …
       2. Lựa chọn cách quản lý tùy theo Requirement Based.
     + Provider
       1. Dựa trên InheritedWidget.
       2. Phát data:
          1. Ở parent của widget cần nhận.
          2. Data không thay đổi: Provider<T>(builder: return data);
          3. Data thay đổi:

Data phải kế thừa thông qua (extends hoặc with) ChangeNotifier (thuộc về Flutter) (nên import của foundation vì foundation có trong material nếu dùng material thì sẽ import những thứ không cần thiết).

ChangeNotifierProvider<T>(builder: return data).

* + - 1. Nhận data:
         1. Provider.of<T>(context, listener: bool): data ( Khi muốn thay đổi data ta set listener = false).
         2. Cusomer<T>(builder(context, T t, child))(Widget).
    - InheritedWidget:
      1. Cho phép truyền data xuống mà không phải thông qua những level trung gian.
  1. Khác
     + Nên thêm dấu phẩy sau mỗi dấu đóng ngoặc đơn để dễ format code.
     + YAML (Ain’t Markup Language): Cẩn thận khi chỉnh sửa, đặc biệt là sử dụng dấu cách.
     + Assets: Thay vì để đường dẫn chính xác cả tên file ta có thể chỉ cần dùng tên folder.
     + Folder Icon:
       1. Android: app/src/main/res/…
       2. IOS: Runner/Assets.xcassets/Appicon.appiconset/…

1. Widgets:
   1. Tính chất chung:
      * Size của widget con sẽ phụ thuộc vào widget cha.
      * Key class:
        1. Để định danh (identifier) widgets, element, …
        2. Dùng để định danh các item trong list.
      * Muốn set size max ta dùng: size = double.infinity
   2. MaterialApp:
      * Dùng phong cách thiết kế Material Design của Google.
      * Cấu hình top-level Navigator.
   3. Scaffold:
      * Implement các thiết kế cơ bản của Material Design.
      * Cung cấp API để hiển thị Drawer, Snack Bar, Bottom Sheet.
      * Chỉnh sửa theme.
      * Thuộc tính:
        1. *appBar*: Hiển thị appBar của app.
        2. *body*: The primary content của scaffold.
   4. Text
      * Mặc định align là góc trái trên cùng.
   5. Container (standart)
      * 
      * Nếu dùng decoration- (ví dụ: shape là circle) thì sẽ dùng min(height,width) của child.
      * Không có child: as big as possible.
      * Có child: điều chỉnh để vừa với child nếu widget cha không dùng constrain.
      * Khi dùng alignment hoặc dùng constraint expand: sẽ mở rộng hết cở so với max size của widget cha.
      * Vì container phước tạp nên tốn chi phí tính toán nếu chỉ dùng padding ta có thể dùng Widget Padding để giảm chi phí.
      * Trang trí bằng decoration: BoxDecoration(color: , shape, …).
      * Nếu có thuộc tính decoration mà muốn thêm màu ta phải định nghĩa màu ở trong decoration và xóa thuộc tính color của Container.
   6. Column/ Row(standart)
      * Mở rộng hết cở theo chiều dọc/ngang để đủ khoảng trống cho widget con. Nếu gặp giới hạn sẽ bị overflowed.
      * mainAxisAlignment: Sắp xếp children theo trục chính.
      * crossAxisAlignment:
        1. Sắp xếp children theo trục phụ.
        2. stretch: childer sẽ max size trục phụ.
        3. Để sử dụng crossAxisAlignment.end ta thêm Container với width max.
        4. baseLine: Các dòng chữ bằng nhau dù cho fontSize chênh lệch. (Phải có textBaseLine).
   7. Expanded:
      * Làm cho child của Row, Column, Flex chiếm khoảng trống nhiều nhất có thể ở trục chính.
      * Nếu nhiều child được expanded thì khoảng trống mỗi child chiếm được phụ thuộc vào flex.
   8. Flexible:
      * Điều khiển cách mà child của Row, Column, Flex hiển thị.
      * Không giống Expanded, Flexible không yêu cầu child lấp đầy available space.
      * flex ( không gian mà mỗi child chiếm.
   9. Icon
      * Là 1 vector.
      * Linh hoạt (flexible) hơn khi sử dụng Image.
        1. Có thể thay color, size mà không bị pixelated(thấy từng pixel riêng lẻ của hình khi zoom hình chất lượng thấp).
   10. Image:
       * AssetImage: là Image Provider khác với Image.asset là widget.
   11. SizedBox:
       * Thêm khoảng trống giữa 2 widget.
   12. Safe Area:
       * Đặt giao diện trong safe area ( không bao gồm notch(phần trên cùng của IOS và status bar của Android), interactive area(phần cuối cùng của IOS)).
   13. Visibility:
       * set visibility của widgets.
   14. GestureDetector:
       * Thao tác với những cử chỉ (gesture) của người dùng.
   15. FloatingActionButton:
       * Thường dùng trong Android.
       * Chỉ nên có 1 FloatActionButton trong mỗi màng hình.
   16. RawMaterialButton:
       * Dùng để custom button.
       * Các button chỉ hiển thị elevation khi enable (onPressed != null).
   17. TextField( Material Design)
       * autofocus: set autofocus.
       * keyboardType: set type.
       * obscureText = true (che thông tin password).
       * decoration(InputDecoration):
         1. lineThrough(~~Gạch ngang chữ~~).
         2. filled = bool: nếu true thì bên trong sẽ có màu của fillColor.
         3. Icon: set icon.
   18. ListView
       * Reverse = true: Đảo ngược thứ tự hiển thị. Khi kết hợp với reversed của list ta sẽ thực hiện được chức năng giống như nhắn tin.
       * Builder:
         1. Sử dụng khi có số lượng lớn children.
         2. Chỉ khởi tạo những item mà thực sự hiển thị.
   19. BottomSheet:
       * Muốn tạo border cho bottomSheet ta dùng 2 container chồng lên nhau, (Lớp 1: trùng màu nền khi hiện lên, lớp 2: màu cần hiển thị.)
   20. Slider:
       * Custom bằng SliderTheme(widget).
   21. DropdownButton <> CupertinoPicker
       * Thường dùng bên Android.
   22. CupertinoPicker
       * Thường được dùng bên IOS.
   23. StreamBuilder
       * Khi dùng nên check data bằng phương thức hasData (AsyncSnapshot).
   24. Material:
       * Là thành phần cơ bản của Material Design.
       * Dùng để custom widget.
       * RadiusBorder
       * Elevation.
       * Ink Effects
   25. Tips
       * Đổi với các widget nhỏ chỉ cần trong 1 widget cha mà có lặp lại cần tách ra để tránh trùng lặp code ta nên dùng builder Widget( dùng function để trả về widget).
       * Nên extract style và decoration. Nếu có 1 vài thuộc tính cần được thay đổi ta dùng *copywith* (giống với themeData).
2. Tips
   1. Dealing with distractions(Sự xao lảng):
      * Tìm khoảng thời gian rảnh và không gian yên tĩnh.
      * Đặt chế độ máy bay và đặt điện thoại ở nơi không nhìn thấy.
   2. Để ngăn chặn sự thay đổi thuộc tính từ bên ngoài (Cách làm giống kotlin)
      * List<Type> \_list;
      * UnmodifiableListView<Type> get list { return UnmodifiableListView(\_list) }
3. Các website hữu ích và app tham khảo.
   1. Appicon.co
   2. Icons8.com
   3. Canva.com
   4. Vecteezy.com
   5. Dribbble.com
   6. Layout Cheat Sheet: <https://medium.com/flutter-community/flutter-layout-cheat-sheet-5363348d037e>
   7. Tool ColorZilla
   8. App History of EveryThing.