Báo Cáo Khóa học Futter

The Complete 2020 Flutter Development Bootcamp with Dart

Created by Dr. Angela Yu

Mục lục

[I. Dart 3](#_Toc38013138)

[1. Basic: 3](#_Toc38013139)

[2. Variable: 3](#_Toc38013140)

[3. Function: 3](#_Toc38013141)

[4. Access Modifier: 4](#_Toc38013142)

[5. Generic type: 4](#_Toc38013143)

[6. String: 4](#_Toc38013144)

[7. Collection 4](#_Toc38013145)

[8. OOP 4](#_Toc38013146)

[9. Random: 5](#_Toc38013147)

[10. Composition vs Inheriatance: 5](#_Toc38013148)

[11. Các Nguyên Lý ( Principle) khi lập trình: 5](#_Toc38013149)

[12. Khác 5](#_Toc38013150)

[II. Flutter: 5](#_Toc38013151)

[1. Basic 5](#_Toc38013152)

[2. Widget tree 5](#_Toc38013153)

[3. Vì Sao cần đến 3 tree (Widget, Element, Render): 5](#_Toc38013154)

[4. Statefull và Stateless Widgets: 5](#_Toc38013155)

[5. StatefulWidget Lifecycle 5](#_Toc38013156)

[6. *build* method: 5](#_Toc38013157)

[7. Build Context: 5](#_Toc38013158)

[8. Routes: 5](#_Toc38013159)

[9. Hot Reload và Hot Restart: 6](#_Toc38013160)

[10. Cách custom font: 6](#_Toc38013161)

[11. Theme 6](#_Toc38013162)

[12. Khác 6](#_Toc38013163)

[III. Widgets: 6](#_Toc38013164)

[1. Tính chất chung: 6](#_Toc38013165)

[2. MaterialApp: 6](#_Toc38013166)

[3. Scaffold: 6](#_Toc38013167)

[4. Text 6](#_Toc38013168)

[5. Container (standart) 6](#_Toc38013169)

[6. Column/ Row(standart) 7](#_Toc38013170)

[7. Expanded: 7](#_Toc38013171)

[8. Icon 7](#_Toc38013172)

[9. Image: 7](#_Toc38013173)

[10. SizedBox: 8](#_Toc38013174)

[11. Safe Area: 8](#_Toc38013175)

[12. Visibility: 8](#_Toc38013176)

[13. GestureDetector: 8](#_Toc38013177)

[14. FloatingActionButton: 8](#_Toc38013178)

[15. RawMaterialButton: 8](#_Toc38013179)

[16. Trick 8](#_Toc38013180)

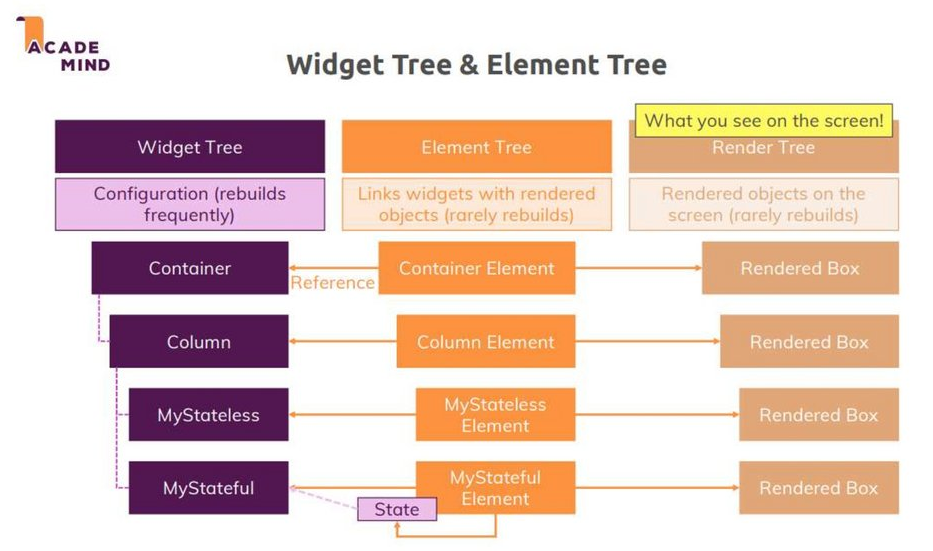
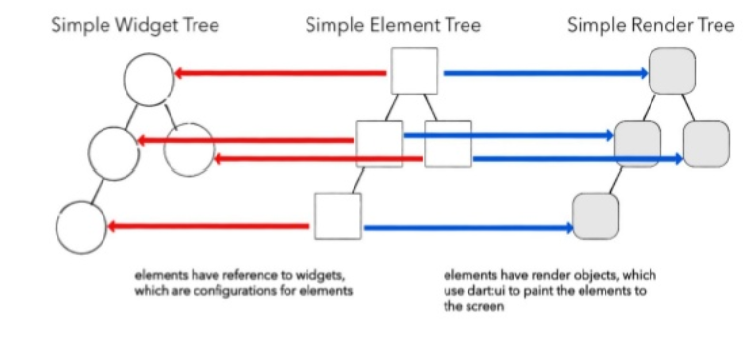
[IV. Tips 8](#_Toc38013181)

[1. Dealing with distractions(Sự xao lảng): 8](#_Toc38013182)

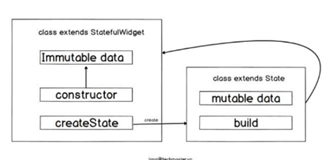
[V. Các website hữu ích và app tham khảo. 8](#_Toc38013183)

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.

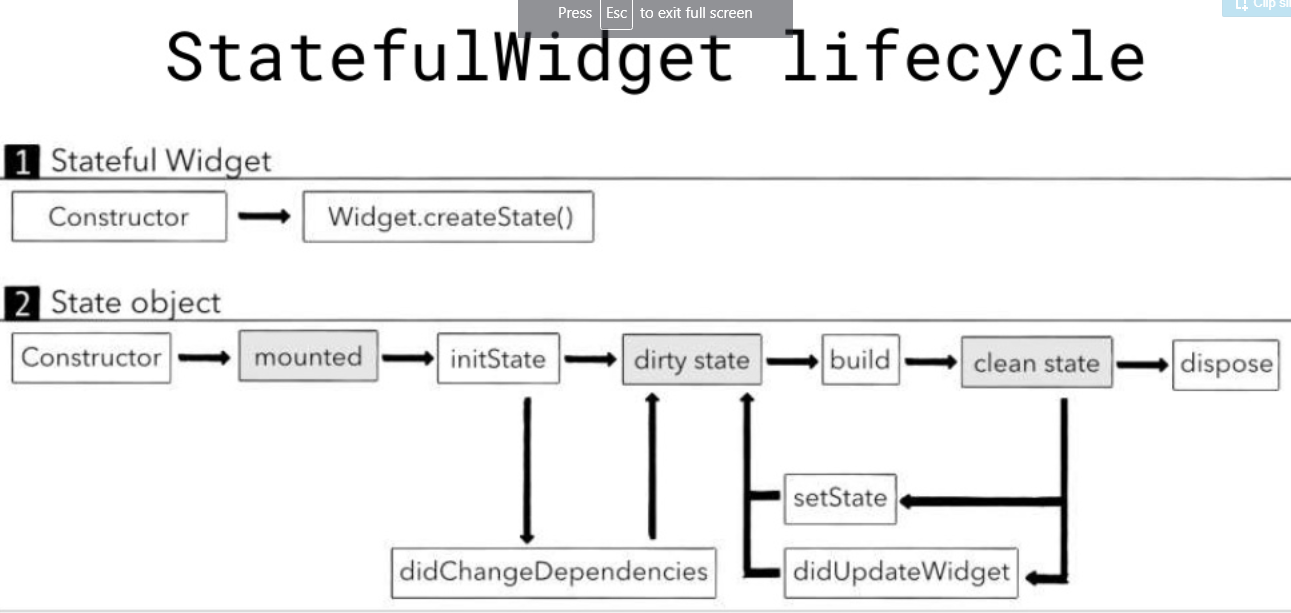
* Nếu muốn là dynamic ta có thể dùng từ khóa *dynamic.*
* Nếu khi khai báo *var* không có kiểu dữ liệu phía sau có nghĩa nó là *dynamic* hay kiểu trả về hoặc parameter của function không định nghĩa kiểu dữ liệu thì nó cũng là *dynamic.*
* final
  + Được khởi tạo 1 lần duy nhất (run time).
  + Vd: final time = DateTime.now(). Working
* const
  + Được khởi tạo lúc compile.
  + Vd: const time = DateTime.now(). Not working
* 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.
     + Set
       1. Đảm bảo mỗi phần tử chỉ xuất hiện 1 lần.
  6. OOP
     + Class:
       1. Là một Blueprint.
       2. Bao gồm: Properties/fields và Method.
     + Tính chất:
       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
  7. Random:
     + Random().nextInt(max): trả ra giá trị từ 0 tới max – 1.
  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.

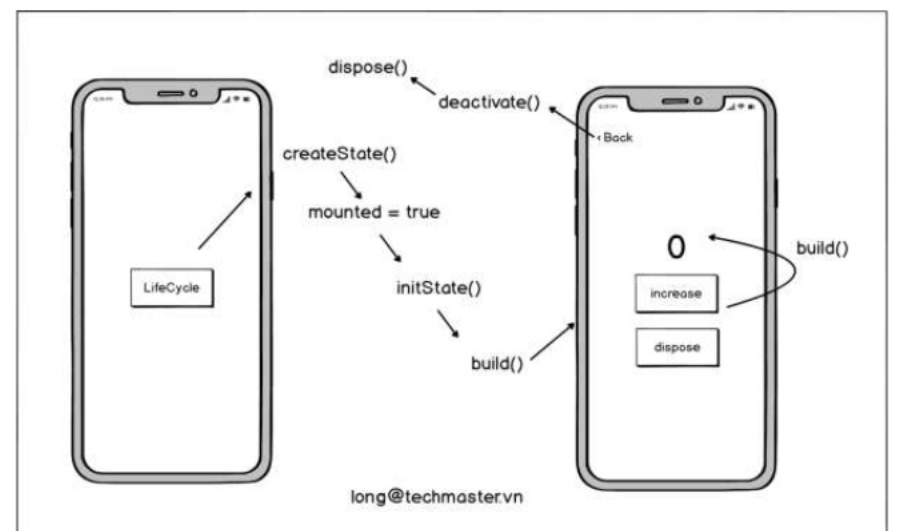
1. Flutter:
   1. Basic
      * Everything is widgets.
   2. Widget tree
      * Là cấu trúc đại diện cho toàn bộ ứng dụng( giống DOM).
      * Các bước chạy chương trình.
      * 
        1. Gọi hàm runApp(), Flutter build một Widget tree chứa các **widget**.
        2. Các **Element** được tạo bởi 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. 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.
      4. Tạo ra UI mà state có thể thay đổi.
  1. StatefulWidget Lifecycle





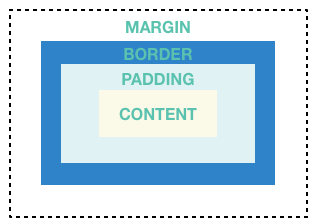
* + - createState(): Khi xây dựng StatefulWidget, ngay lập tức gọi createdState().
    - initState(): Để khởi tạo state, chỉ được gọi một lần khi widget được tạo.
    - didChangeDependencies:
      1. Có thể được gọi một lần sau initState, cũng có thể được gọi lại trong Lifecycle.
      2. Tại thời điểm 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
       3. Sau khi gọi initState.
       4. 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: dùng MaterialPageRoute.
       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. 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).
  2. 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.
  3. Theme
     + Chỉnh sửa theme chung của app thông qua ThemeData
     + Cũng là một Widget.
  4. 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. 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).
   9. Image:
      * AssetImage: là Image Provider khác với Image.asset là widget.
   10. SizedBox:
       * Thêm khoảng trống giữa 2 widget.
   11. 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)).
   12. Visibility:
       * set visibility của widgets.
   13. GestureDetector:
       * Thao tác với những cử chỉ (gesture) của người dùng.
   14. FloatingActionButton:
       * Thường dùng trong Android.
       * Chỉ nên có 1 FloatActionButton trong mỗi màng hình.
   15. RawMaterialButton:
       * Dùng để custom button.
       * Các button chỉ hiển thị elevation khi enable (onPressed != null).
   16. Slider:
       * Custom bằng SliderTheme(widget).
   17. Trick
       * 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).
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.
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.