**FLUTTER**

# **Install**

## **Installation the latest stable release of the Flutter SDK**

* Download Flutter SDK mới nhất từ homepage flutter
* Extract the zip file and place the contained flutter in the desired installation location for the Flutter SDK (for example, C:\src\flutter).

## **Update your path**

If you wish to run Flutter commands in the regular Windows console, take these steps to add Flutter to the PATH environment variable:

* From the Start search bar, enter ‘env’ and select **Edit environment variables for your account**.
* Under **User variables** check if there is an entry called **Path**:
  + If the entry exists, append the full path to flutter\bin using ; as a separator from existing values.
  + If the entry doesn’t exist, create a new user variable named Path with the full path to flutter\bin as its value.

## **Tại CMD or PowerShell**

Run command: flutter doctor để check các thành phần cần thiết, nếu thành phần nào báo X or ! thì cần xử lý thêm (install, update, run command…) cho đến khi nào tất check xanh là ok.

## **For Windows desktop development, you need the following in addition to the Flutter SDK:**

* [Visual Studio 2022](https://visualstudio.microsoft.com/downloads/) or [Visual Studio Build Tools 2022](https://visualstudio.microsoft.com/downloads/#build-tools-for-visual-studio-2022) When installing Visual Studio or only the Build Tools, you need the “Desktop development with C++” workload installed for building windows, including all of its default components.

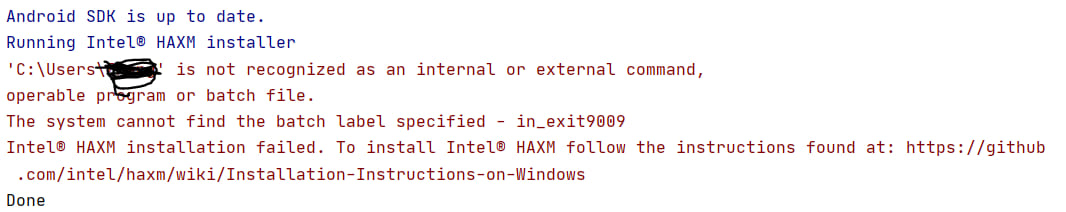
## **Agree to Android Licenses**

Before you can use Flutter, you must agree to the licenses of the Android SDK platform. This step should be done after you have installed the tools listed above.

$ flutter doctor --android-licenses

## **Install Android Studio**

* Sau khi install xong, vào SDK Manager – Tab SDK Tools
  + Cài Android SDK Command-line Tools
  + Cài Google Play Service
  + Intel x86 Emulator Acceletator (HAXM Installation)



Lỗi như hình trên – có thể do tên folder (user name) trên pc vi phạm quy tắc ko no space, no specials charactor

Flutter app can run on Android and IOS emulators with Android Studio, but ONLY on Apple machines (not Windows). So if you have Android Studio installed on Windows machine you won't have IOS emulator.

This info isn't obvious in Android Studio documentation, but you can find more details in Flutter installation guide:

macOS install (Flutter, Android Studio, emulators for Android and IOS)

Windows install (Flutter, Android Studio, emulator for Android)

So in your case the best way is to develop flutter apps on your Mac.

Setup the iOS simulator???

https://www.makeuseof.com/windows-10-install-ios-emulator/

* Tạo Virtual Device
  + Add Skin Samsung
    - Tải skin tại <https://developer.samsung.com/galaxy-emulator-skin>
    - Unzip skin file vào folder chỉ định
    - Trong màn hình Decive Manager click Create Device, New Hardwave Profile, Nhập các giá trị Screen Size và Resolution y như mô tả của skin từ <https://developer.samsung.com/galaxy-emulator-skin/galaxy-s.html> (tùy version device).

Kéo xuống dưới cùng có phần Default Skin, click … browser tới folder chứa skin

* Open AVD: Open Android Studio (khi đã xóa hết app, ko quản lý app nào), click More Action, chọn Virtual Device Manager, chọn máy ảo, click run máy ảo. Lúc này có thể close Android Studio, chỉ để lại AVD

## **Setup AVD không install Android Studio**

## **Kết nối Physical Device chưa được**

## **Smartface IDE - Develop iOS and Android apps on Windows, macOS or Linux. No more OS or hardware lock-in.**

**Install**

https://smartface.io/

**Using**

https://docs.smartface.io/smartface-getting-started/create-a-new-project

## **Install VS Code (if using VS Code)**

VS Code is a lightweight editor with complete Flutter app execution and debug support.

* Install the Flutter and Dart plugins

## **Code demo trên web**

<https://dart.dev/#try-dart>

https://dartpad.dev/?

# **VS Code**

Hot Reload – Run without Debugging – Ctrl + F5

## **Xử Lý Lỗi**

* Run flutter doctor –vv hoặc các lệnh flutter từ cmd báo Error: Unable to find git in your PATH.

Chạy cmd: git config --global --add safe.directory '\*'

## **Flutter Project Structure**

**Android – Nơi chứa code Android – viết Android Native code**

**Ios – Nơi chứa code Ios – viết Ios Native code**

**Lib – Nơi chứa toàn bộ code flutter (file .dart)**

**Test – Nơi viết test code**

**Pubspec.yaml**

Mỗi dự án Flutter bao gồm file pubspec.yaml. Nó nằm ở đầu cây dự án và chứa siêu dữ liệu về dự án mà công cụ Dart và Flutter cần biết. Pubspec được viết bằng YAML , mà con người có thể đọc được, nhưng hãy lưu ý rằng khoảng trắng (tab v dấu cách) mới quan trọng .

Tệp pubspec chỉ định các phụ thuộc mà dự án yêu cầu, chẳng hạn như các gói cụ thể (và phiên bản của chúng), phông chữ hoặc tệp hình ảnh. Nó cũng chỉ định các yêu cầu khác, chẳng hạn như sự phụ thuộc vào gói dành cho nhà phát triển (như gói thử nghiệm hoặc mô phỏng) hoặc các ràng buộc cụ thể đối với phiên bản SDK Flutter.

**Pubspec.lock**

Lần đầu tiên bạn xây dựng dự án của mình, nó sẽ tạo file pubspec.lock chứa phiên bản cụ thể của các gói được bao gồm. Điều này đảm bảo rằng bạn sẽ nhận được cùng một phiên bản vào lần tiếp theo dự án được xây dựng.

**analysis\_options.yaml**

**YAML is sensitive to whitespace.** Don’t use tabs in a YAML file, and use 2 spaces to denote each level of indentation.

Phân tích tĩnh cho phép tìm ra các vấn đề trước khi thực thi một dòng mã duy nhất. Đó là một công cụ mạnh mẽ được sử dụng để ngăn chặn lỗi và đảm bảo mã tuân thủ các hướng dẫn về kiểu dáng.Với sự trợ giúp của bộ phân tích, bạn có thể tìm ra những lỗi đơn giản như viết sai chính tả. Ví dụ, có thể đã có một dấu chấm phẩy không cần thiết bị đưa vào câu lệnh if một cách tình cờ

Rules được gán trong file này

# **DART**

<https://dart.dev/tools/dart-devtools#1-start-the-target-app>

<https://dart.dev/tools/linter-rules/prefer_interpolation_to_compose_strings>

## Package trong Dart

Theo định nghĩa, một package trong Dart là một bộ sưu tập các tập tin Dart liên quan và được tổ chức theo một cấu trúc cụ thể¹. Một package có thể chứa các hàm, lớp, biến, kiểu dữ liệu, widget, plugin, hoặc bất kỳ đoạn code Dart nào có thể được tái sử dụng và chia sẻ. Một package cũng có thể phụ thuộc vào các package khác để sử dụng các chức năng của chúng.

Một package trong Dart có thể được phân loại thành ba loại chính²:

- Package Dart: là những package có thể được sử dụng ở cả môi trường web lẫn thiết bị di động. Ví dụ như [english\_words] là một package chứa khoảng 5000 từ tiếng Anh và có các chức năng cơ bản như danh từ, âm tiết, v.v.

- Package Flutter: là những package phụ thuộc vào framework Flutter và chỉ có thể được sử dụng trong môi trường thiết bị di động. Ví dụ như [fluro] là một custom router cho Flutter.

- Plugin Flutter: là những package phụ thuộc vào framework Flutter cũng như nền tảng cơ bản (Android SDK hoặc iOS SDK). Ví dụ như [camera] là một plugin dùng để tương tác với thiết bị camera.

Để sử dụng một package trong Dart, bạn cần phải thực hiện các bước sau³:

- Tìm kiếm và chọn một package phù hợp với nhu cầu của bạn trên [pub.dev](^4^), một máy chủ trực tuyến để lưu trữ và xuất bản các package Dart.

- Nhập tên và phiên bản của package vào file pubspec.yaml của dự án của bạn, trong phần dependencies. Ví dụ:

dependencies:

english\_words: ^3.1.5

- Chạy lệnh `flutter pub get` hoặc `dart pub get` để cài đặt package vào dự án của bạn.

- Nhập file Dart chính của package vào file code của bạn bằng lệnh `import`. Ví dụ:

import 'package:english\_words/english\_words.dart';

- Sử dụng các hàm, lớp, biến, hoặc widget của package theo ý muốn.

# **FLUTTER**

## Widget

Flutter là một framework phát triển ứng dụng di động và web sử dụng ngôn ngữ lập trình Dart. Trong Flutter, giao diện người dùng được xây dựng bằng cách kết hợp các widget. Widget là các thành phần cơ bản trong Flutter, và chúng được sắp xếp lại để tạo thành giao diện người dùng. Dưới đây là một số widget quan trọng trong Flutter:

* StatelessWidget: Đây là một widget không thay đổi trạng thái sau khi được xây dựng ban đầu. Nó được sử dụng cho các thành phần tĩnh của giao diện người dùng.
* StatefulWidget: Đây là một widget có thể thay đổi trạng thái sau khi được xây dựng. Nó được sử dụng cho các thành phần động của giao diện người dùng.
* Container: Widget này tạo một hình chữ nhật có thể chứa các widget con và được sử dụng để tạo khoảng trống, định dạng và cài đặt kiểu cho các widget con.
* Text: Widget này hiển thị văn bản trên giao diện người dùng. Bạn có thể định dạng văn bản bằng cách sử dụng thuộc tính style.
* Image: Widget này hiển thị hình ảnh từ các nguồn như asset hoặc mạng.
* Column và Row: Đây là các widget chứa các widget con theo chiều dọc (Column) hoặc chiều ngang (Row). Chúng giúp bạn sắp xếp các widget con một cách linh hoạt.
* ListView: Widget này cho phép bạn tạo danh sách cuộn được tạo bằng cách xếp các widget con dọc theo trục chính.
* AppBar: Widget này tạo một thanh đầu của ứng dụng, thường chứa tiêu đề, các biểu tượng thao tác và nút điều hướng.
* FlatButton, RaisedButton, IconButton: Các widget này tạo các nút tương tác, cho phép người dùng thực hiện các hành động khi nhấp vào.
* TextField: Widget này cho phép người dùng nhập dữ liệu văn bản.
* GestureDetector: Widget này cho phép bạn bắt các sự kiện tương tác như chạm, vuốt, nhấn giữ và nhiều thứ khác.
* Scaffold: Widget này cung cấp cấu trúc cơ bản cho một màn hình, bao gồm AppBar, body và các thành phần khác.
* InkWell: Widget này tạo ra một vùng có hiệu ứng mực khi người dùng chạm vào, giúp tạo ra giao diện tương tác thú vị.

Đây chỉ là một số ví dụ cơ bản về các widget trong Flutter. Flutter cung cấp một loạt các widget khác nhau để bạn có thể xây dựng các giao diện người dùng phong phú và linh hoạt.

## StatelessWidget & StatefulWidget

Trong Flutter, tất cả các widget đều thuộc một trong hai loại: StatelessWidget hoặc StatefulWidget.

**StatelessWidget:** Đây là loại widget mà trạng thái của nó không thay đổi sau khi được xây dựng ban đầu. Điều này có nghĩa là nếu bạn muốn thay đổi giao diện của widget này, bạn phải xây dựng một phiên bản mới của nó với dữ liệu mới. StatelessWidget thường được sử dụng cho các thành phần giao diện người dùng tĩnh và không thay đổi, như văn bản hiển thị, hình ảnh, biểu đồ, vv.

Hiểu nôm na là Widget tĩnh và nó không thể tự thay đổi được những gì mà nó hiển thị sau khi đã được Render xong.

Widget này cần 1 hàm Widget build(BuildContext context) để render dữ liệu lên màn hình. Hàm build chỉ được gọi 1 lần khi ứng dụng đang hoạt động, như vậy thì dữ liệu chỉ được render 1 và không thay đổi suốt quá trình sử dụng ứng dụng. Dữ liệu hiển thị bạn có thể hard code hoặc truyền thông qua hàm Constructors của class và dữ liệu này sẽ không thay đổi suốt quá trình hiển thị trên màn hình.

*Vậy đặt ra vấn đề Tôi muốn thay đổi text đang hiển thị trên màn hình thì làm thế nào?*

Tuy StatelessWidget không thể tự thay đổi được chính nó, nhưng khi Widget cha thay đổi thì StatelessWidget sẽ được khởi tạo lại.

Example:

class MyWidget extends StatelessWidget {

  final bool load\_state;

  const MyWidget(this.load\_state);

  @override

  Widget build(BuildContext context) { // render data output

    return load\_state

        ? Text('loading state $load\_state')

        : const CircularProgressIndicator();

  }

}

**StatefulWidget:** Đây là loại widget có trạng thái có thể thay đổi sau khi được xây dựng ban đầu. StatefulWidget được sử dụng cho các thành phần giao diện người dùng có thể tương tác và thay đổi dựa trên tương tác người dùng, thời gian hoặc dữ liệu thay đổi.

Khác với StatelessWidget thì StatefulWidget là 1 Widget động và nó có thể thay đổi những gì đang hiển thị bằng cách thay đổi State của chính nó.

Widget này cần hàm State<StatefulWidget> createState() để cung cấp State cho StatefulWidget.

Class \_MyHomePageState sẽ overrides phương thức Widget build(BuildContext context) hàm này trả về Widget. Đây là nơi bạn định nghĩa UI mà class hiển thị. StatefulWidget quản lý trạng thái UI thông qua State, khi State thay đổi thì StatefulWidget sẽ render lại UI mà nó đang hiển thị.

Vậy câu hỏi đặt ra là Làm thế nào tôi có thể thay đổi được State của Statefull hay đơn giản là update dữ liệu đang hiển thị trên màn hình?

Câu trả lời này khá đơn giản. StatefulWidget cung cấp phương thức setState() để bạn có thể thay đổi State của class. Hiểu đơn giản là khi bạn muốn Update UI của StatefulWidget thì cần gọi phương thức setState() để thông báo cho StatefullWidget là tôi muốn bạn UpdateUI. Tất nhiên bạn có thể gọi setState() nhiều lần mỗi khi cần thay đổi UI trong vòng đời của ứng dụng.

Và đặc biệt, nên sử dụng StatefulWidget trong Widget con của ListView để các item của chúng có thể tự động cập nhật trạng thái mà không cần cả ListView thay đổi trạng thái.

Nhìn như này mình đoán sẽ nhiều bạn đặt câu hỏi Thế tại sao không dùng StatefulWidget vừa có thể hiển thị vừa có thể thay đổi dc UI việc gì phải dùng StatelessWidget. Thì đúng là như thế, nhưng các bạn lưu ý do StatelessWidget không có State nên việc render UI của nó nhẹ hơn và nhanh hơn rất nhiều. Các bạn cần tìm hiểu rõ về State để lựa chọn sử dụng giữa StatelessWidget và StatefulWidget để có thể dễ dàng trong việc viết code và tối ưu hiệu năng.

Example: 01

// MyWidgetFul extends StatefulWidget call method createState() return kieu State<StatefulWidget> cua MyWidgetFul\_state

class MyWidgetFul extends StatefulWidget {

  final bool loading; // Fields in a Widget subclass are always marked "final".

  const MyWidgetFul(this.loading);

  @override

  State<StatefulWidget> createState() {

    return MyWidgetFul\_state();

  }

}

// MyWidgetFul\_state lai extends nguoc ve State<MyWidgetFul>, call method build – render data output

class MyWidgetFul\_state extends State<MyWidgetFul> {

  late bool \_local\_load\_state;

  @override

  void initState() {

    // khoi tao gia tri ban dau, run truoc build function

    \_local\_load\_state = widget.loading;

  }

  @override

  Widget build(BuildContext context) { // This method is rerun every time setState method is called

    return \_local\_load\_state

        ? Text('StatefulWidget')

        : CircularProgressIndicator();

  }

}

Example: 02

class MyHomePage extends StatefulWidget {

  const MyHomePage({super.key, required this.title});

  final String title; // Fields in a Widget subclass are always marked "final".

  @override

  State<MyHomePage> createState() => \_MyHomePageState();

}

class \_MyHomePageState extends State<MyHomePage> {

  int \_counter = 0;

  void \_incrementCounter() {

    setState(() { // setState thông báo cho Flutter rằng có một thay đổi đã xảy ra trong State này, dẫn đến run lại phương thức build ở dưới để cập nhật hiển thị dựa trên các giá trị đã được cập nhật. Nếu chúng ta thay đổi giá trị của \_counter mà không gọi setState(), thì phương thức build sẽ ko run lại, do đó không có gì thay đổi trên giao diện hiển thị.

      \_counter++;

    });

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        backgroundColor: Theme.of(context).colorScheme.inversePrimary,

        title: Text(widget.title),

      ),

      body: Center(

        child: Column(

          children: <Widget>[

            const Text(

              'You have pushed the button this many times:',

            ),

            Text(

              '$\_counter',

              style: Theme.of(context).textTheme.headlineMedium,

            ),

          ],

        ),

      ),

      floatingActionButton: FloatingActionButton(

        onPressed: \_incrementCounter,

        tooltip: 'Increment',

        child: const Icon(Icons.add),

      ), // This trailing comma makes auto-formatting nicer for build methods.

    );

  }

## [Text](https://api.flutter.dev/flutter/widgets/Text-class.html) widget

A run of text with a single style.

The Text widget displays a string of text with single style. The string might break across multiple lines or might all be displayed on the same line depending on the layout constraints.

### Một số properties

Widget build(BuildContext context) {

    return const Text(

      'Tiện ích Văn bản hiển thị một chuỗi văn bản với một kiểu duy nhất. Chuỗi có thể ngắt trên nhiều dòng hoặc tất cả có thể được hiển thị trên cùng một dòng tùy thuộc vào các ràng buộc về bố cục.',

      textAlign: TextAlign.left,

      maxLines: 2, // show tối đa bao nhiêu dòng

      overflow: TextOverflow.ellipsis, // kiểu báo hiệu khi nội dung text vượt quá khung container

      textScaleFactor: 2.0, // thay đổi kích thước font theo tỉ lệ so với fontsize gốc

      style: TextStyle(fontWeight: FontWeight.bold), // thay đổi text style

    );

  }

Widget build(BuildContext context) {

// Using the Text.rich constructor, the Text widget can display a paragraph with differently styled TextSpans. The sample that follows displays "Hello beautiful world" with different styles for each word.

    return const Text.rich(

      style: TextStyle(fontSize: 20 \* 1.5),

      TextSpan(

        text: 'Hello', // default text style

        children: <TextSpan>[ // mỗi TextSpan áp dụng style riêng

          TextSpan(

              text: ' beautiful ',

              style: TextStyle(fontStyle: FontStyle.italic)),

          TextSpan(

              text: 'world', style: TextStyle(fontWeight: FontWeight.bold)),

        ],

      ),

    );

  }

### Selection

Text is not selectable by default. To make a Text selectable, one can wrap a subtree with a SelectionArea widget. To exclude a part of a subtree under SelectionArea from selection, once can also wrap that part of the subtree with SelectionContainer.disabled.

 Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: const Text('SelectionContainer.disabled Sample')),

        body: const Center(

          child: SelectionArea( // SelectionArea – cho phép selection text

            child: Column(

              mainAxisAlignment: MainAxisAlignment.center,

              children: <Widget>[

                Text('Selectable text'),

                SelectionContainer.disabled(child: Text('Non-selectable text')), // SelectionContainer.disabled – ko cho phép selection object này

                Text('Selectable text'),

              ],

            ),

          ),

        ),

      ),

    );

  }

### style: TextStyle

 style: TextStyle(

fontFamily: 'Roboto' // nếu mặc định Flutter ko hổ trợ thì phải setup font theo hướng dẫn bên dưới

fontSize: 50,

          fontWeight: FontWeight.bold, // từ w100(normal) đến w900(bold)

          wordSpacing: 0.5, // space giữa các word

          letterSpacing: 1, // space giữa các letter

          decoration: TextDecoration.lineThrough, // gạch ngang dòng text

          decorationColor: Colors.red, // màu lằn kẻ

          fontStyle: FontStyle.italic, // italic or normal

// một số cách set color

decorationColor: Colors.red, // Colors.ten\_color

          color: Colors.greenAccent, // Colors.ten\_color\_biến\_thể

          backgroundColor: Color.fromARGB(255, 243, 103, 145), // nhập mã ARGB

          color: Color(0xFF3366FF)), // nhập mã ARGB theo hexa (0xAARRGGBB)

// Set độ trong suốt text - Using the RichText constructor, the Text widget can display a paragraph with differently styled TextSpans.

  Widget build(BuildContext context) {

    return RichText(

      text: TextSpan(

        style: const TextStyle(fontSize: 20),

        children: <TextSpan>[

          TextSpan(

            text: "You don't have the votes.\n",

            style: TextStyle(color: Colors.black.withOpacity(0.6)),

          ),

          TextSpan(

            text: "You don't have the votes!\n",

            style: TextStyle(color: Colors.black.withOpacity(0.8)),

          ),

          TextSpan(

            text:

                "You're gonna need congressional approval and you don't have the votes!\n",

            style: TextStyle(color: Colors.black.withOpacity(1.0)),

          ),

        ],

      ),

    );

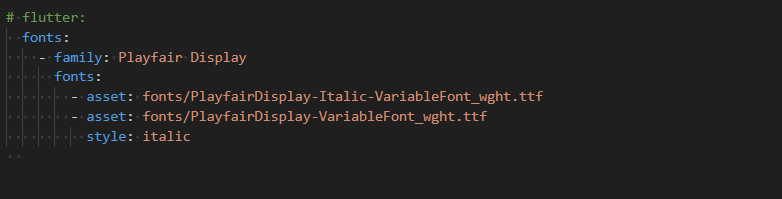
  }

### Định nghĩa fontSize trong MartialApp qua TextTheme

* Định nghĩa trong MartialApp qua TextTheme
* void main() {
* runApp(MaterialApp(
* theme: ThemeData(
* //fontFamily: 'Playfair Display',
* textTheme: const TextTheme(
* bodyLarge: TextStyle(fontSize: 18.0),
* bodyMedium: TextStyle(fontSize: 26.0),
* bodySmall: TextStyle(fontSize: 26.0),
* // ...  // and so on for every text style
* ),
* Áp dụng fontSize cho object
* TextSpan(
* text: "You don't have the votes!\n",
* style: TextStyle(
* color: Colors.black.withOpacity(0.8),
* fontSize: Theme.of(context).textTheme.bodyLarge?.fontSize), // gọi lại định nghĩa ở trên MaterialApp
* ),

### Use a custom font

* Import the font files: tạo folder fonts trong project
* Declare the font in the pubspec.yaml – chú ý: khoảng cách thụt dòng phải chính xác



* Set a font as the default trong MaterialApp – family chính là fontFamily



* Use a font in a specific widget.

### Kiểm tra properties TextStyle với các method getParagraphStyle() và getTextStyle()

void printText\_Para\_Style({required BuildContext context}) {

  TextStyle textStyle = DefaultTextStyle.of(context).style;

  List<String> listA = textStyle.getParagraphStyle().toString().split(',');

  List<String> listB = textStyle.getTextStyle().toString().split(',');

  print('getParagraphStyle:\n');

  for (var style in listA) {

    // Kiểm tra các thuộc tính của style để xem nó có chứa từ 'unspecified' hay không

    if (!style.contains('unspecified')) {

      print(style.toString());

    }

  }

  print('----------------');

  print('getTextStyle:\n');

  for (var style in listB) {

    // Kiểm tra các thuộc tính của style để xem nó có chứa từ 'unspecified' hay không

    if (!style.contains('unspecified')) {

      print(style.toString());

    }

  }

}

## Title của MaterialApp khác với Title của AppBar thế nào

Trong Flutter, title của MaterialApp và title của AppBar có mục đích và vị trí sử dụng khác nhau:

* Title của MaterialApp:

Thuộc tính title của MaterialApp đặt tiêu đề của toàn bộ ứng dụng. Điều này thường được sử dụng để xác định tiêu đề của ứng dụng trên thanh tiêu đề của hệ điều hành hoặc trong danh sách các ứng dụng đang chạy.

Được đặt trong phần khởi tạo của MaterialApp.

* Title của AppBar

Thuộc tính title của AppBar đặt tiêu đề cho thanh tiêu đề nằm trong AppBar. Thường được sử dụng để hiển thị một tiêu đề đặc biệt cho từng màn hình hoặc trạng thái của ứng dụng.

Được đặt trong phần body của AppBar.

Dưới đây là một ví dụ để thấy sự khác biệt giữa hai thuộc tính này:

void main() {

  runApp(MaterialApp(

    title: 'My Awesome App', // Chưa rõ sẽ show lên trong trường hợp nào?

    home: Scaffold(

      appBar: AppBar(

        title: Text('Screen Title'), // Tiêu đề của ứng dụng

      ),

      body: TextWidget(),

    ),

  ));

}

## Reference

https://docs.flutter.dev/ui/widgets

https://docs.flutter.dev/cookbook