**FLUTTER**

# **RoadMap**

# **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.

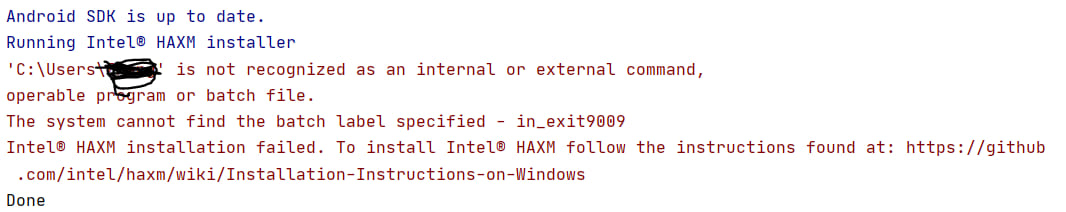
## **Agree to Android Licenses**

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 – ko khả thi**

You do not need Android Studio to create or run a virtual device. Just use sdkmanager and avdmanager from the Android SDK tools.

* ProcedureTo Install the JDK Software and Set Environment Variables JAVA\_HOME on a Windows System: <https://docs.oracle.com/cd/E19182-01/820-7851/inst_cli_jdk_javahome_t/index.html>

avdmanager create avd --name AndroidDevice01 --package "system-images;android-21;default;x86\_64"

Nơi lưu AVD: Đường dẫn mặc định: **C:\Users\<tên người dùng>\.android\avd**

## Run AVD Emulator without Android Studio

* Show list AVD trên hệ thống:

Run

cd ~/Android/Sdk/tools/bin && ./avdmanager list avd

OR

cd ~/Android/Sdk/tools && ./emulator -list-avds

Demo: C:\Users\Thinkpad T480s\./avdmanager list avd AppData\Local\Android\Sdk\tools>emulator -list-avds

PS C:\Users\Thinkpad T480s\AppData\Local\Android\Sdk\emulator> .\emulator -list-avds // cú pháp trên PS khác: phải thêm '.\'

* Copy name of the device you want to run and then: cd ~/Android/Sdk/tools && ./emulator -avd NAME\_OF\_YOUR\_DEVICE, nếu AVD nào báo lỗi 'PANIC: Missing emulator engine program for 'x86' CPU.' thì vào path ~/Android/Sdk/emulator

Demo: C:\Users\Thinkpad T480s\AppData\Local\Android\Sdk\emulator>.\emulator.exe -avd Pixel\_5\_API\_30

## **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

## Shortcut

* Show Widget Tree (gồm các function, class nhưng ko bao gồm các properties): Ctrl + Shift + P 🡪 View: Show Flutter

Với các widget có icon  ta có thể click chọn, right click để thực hiện 1 số thao tác trên nó

* Tạo StatelessWidget: type 'stl' or 'stless'
* Tạo StatefulWidget: type 'stf' or 'stful'

# **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.
* MaterialApp là widget kế thừa các kiến trúc theo chuẩn design marterial (xem như html tag trong trang web)
* Scaffold: Widget này cung cấp cấu trúc cơ bản cho một màn hình, bao gồm AppBar, body và các thành phần khác. Scaffold là widget bao chứa toàn bộ code chúng ta vào trong nó (phủ đầy giao diện chúng ta, như thẻ body của html vậy thôi)
* AppBar: Widget này tạo một thanh đầu của ứng dụng, thường chứa tiêu đề, các biểu tượng thao tác và nút điều hướng.
* 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.
* 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.
* 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.
* GestureDetector: Widget này cho phép bạn bắt các sự kiện tương tác như chạm, vuốt, nhấn giữ và nhiều thứ khác.

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.

    );

  }

## Truyền tham số cho widget

1. Khai báo variable làm tham số, định nghĩa hàm constructor với tham số

class PaddingAndMargin extends StatelessWidget {

// khai báo variable làm tham số, định nghĩa hàm constructor với tham số

  double pixel = 0;

  PaddingAndMargin(this.pixel);

  @override

  Widget build(BuildContext context) {

    return Card(

      margin: EdgeInsets.all(20.0),

      color: Colors.amber,

      child: Padding(

padding: EdgeInsets.symmetric(horizontal: pixel), // sử dụng tham số được truyền vào

child: Text('Flutter'),

      ),

    );

  }

}

//-----------------------------

1. Trong hàm main(): gọi hàm constructor PaddingAndMargin đã định nghĩa ở bước 1

 body: PaddingAndMargin(20),

## Quy tắc cơ bản khi dùng widget

* Chia nhỏ các tác vụ vào widget
* Tận dụng tối đa các widget build-in như Container, Text, Row, Column,…
* Tránh lồng nhiều lớp widget
* Dùng const khi có thể cho các object
* Áp dụng Material Design và Cupertino
* Tránh logic phức tạp trong build:

Hạn chế việc thực hiện các phép tính phức tạp, truy vấn cơ sở dữ liệu, gọi API mạng hay các xử lý lớn khác trong hàm build của widget. Hàm build được gọi lại nhiều lần trong quá trình render giao diện và có thể gây lag nếu chứa các thao tác tốn nhiều thời gian.

Việc tính toán phức tạp trong hàm build có thể làm chậm quá trình render giao diện, làm cho ứng dụng trở nên không mượt và không đáp ứng tốt, dẫn đến trải nghiệm người dùng kém.

Thay vì thực hiện logic phức tạp trong build, bạn nên thực hiện những việc như tính toán, truy vấn dữ liệu, gọi API mạng, và các xử lý phức tạp khác trước khi đến phần xây dựng giao diện. Kết quả của các xử lý này nên được lưu trong trạng thái của widget hoặc các biến tạm thời, và sau đó bạn chỉ cần sử dụng các giá trị đã tính toán trong hàm build.

Ví dụ, nếu bạn cần lấy dữ liệu từ cơ sở dữ liệu, hãy thực hiện việc này trong phương thức initState của StatefulWidget và lưu kết quả trong trạng thái. Sau đó, trong hàm build, bạn chỉ cần sử dụng dữ liệu đã lấy từ trạng thái mà không cần phải thực hiện truy vấn cơ sở dữ liệu trong mỗi lần render.

* Thuộc tính child: nên đặt sau cùng trong list thuộc tính

class MyWidget extends StatefulWidget {

  @override

  \_MyWidgetState createState() => \_MyWidgetState();

}

class \_MyWidgetState extends State<MyWidget> {

  List<Data> fetchedData = []; // lưu data lấy về từ method initState()

  @override

  void initState() {

    super.initState();

    fetchDataFromDatabase();

  }

  void fetchDataFromDatabase() async {

    // Logic lấy dữ liệu từ cơ sở dữ liệu

    // và lưu vào fetchedData

  }

  @override

  Widget build(BuildContext context) {

    return ListView.builder(

      itemCount: fetchedData.length,

      itemBuilder: (context, index) {

        // Sử dụng fetchedData[index] để hiển thị dữ liệu

      },

    );

  }

}

………………

## [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 TextSpan 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.

// TextSpan widget có tính chất đệ quy và nó có thể bao gồm các TextSpan khác

    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 widget qua TextTheme

* Định nghĩa trong MartialApp qua TextTheme
* void main() {
* runApp(MaterialApp( // MaterialApp là widget build-in kế thừa các kiến trúc theo chuẩn design marterial
* 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
* ),

### Dùng TextTheme thiết lập TextStyle cho cả ứng dụng

 Widget build(BuildContext context) {

    return MaterialApp(

      theme: ThemeData(

        textTheme: const TextTheme(

          bodyMedium: TextStyle( // văn bản thông thường dùn thuộc tính bodyMedium

              fontFamily: 'Roboto',

              fontSize: 30,

              color: Colors.green,

              fontWeight: FontWeight.bold), // Font chữ mặc định

        ),

      ),

      home: HomeScreen(),

    );

  }

### DefaultTextStyle

DefaultTextStyle trong Flutter được sử dụng để thiết lập một TextStyle mặc định cho một nhóm widget con trong cây giao diện. Khi bạn bọc một hoặc nhiều widget con bên trong DefaultTextStyle, các widget đó sẽ thừa hưởng các thuộc tính của TextStyle được thiết lập trong DefaultTextStyle, trừ khi bạn ghi đè thuộc tính trong widget con.

DefaultTextStyle có thể hữu ích khi bạn muốn áp dụng cùng một TextStyle cho nhiều widget con, mà không cần phải đặt lại TextStyle cho từng widget. Thay vì chỉnh sửa từng widget riêng lẻ, bạn chỉ cần thiết lập DefaultTextStyle bao ngoài chúng.

 body: const Center(

          child: DefaultTextStyle( // áp dụng TextStyle cho các object trong child: Column phía dưới

            style: TextStyle(

              fontSize: 30,

              fontWeight: FontWeight.w500,

              color: Color.fromARGB(255, 183, 12, 225),

            ),

            child: Column(

              mainAxisAlignment: MainAxisAlignment.center,

              children: [

                Text('Hello'),

                Text('World'),

                Text('Flutter'),

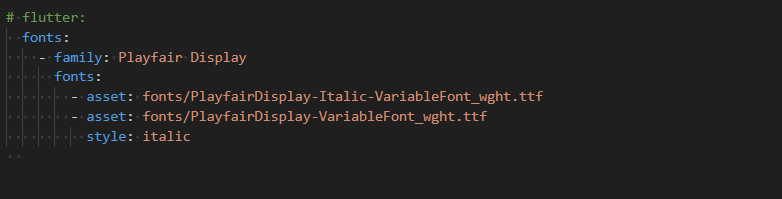
              ],

            ),

          ),

### 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());

    }

  }

}

### Padding & Margin

 Widget build(BuildContext context) {

    return Card(

      margin: EdgeInsets.all(20.0),

      margin: EdgeInsets.fromLTRB(50, 50, 30, 50),

      margin: EdgeInsets.symmetric(horizontal: 50),

      color: Colors.amber,

      child: Padding(

        padding: EdgeInsets.all(10.0), // set cùng value cho 4 cạnh

        padding: EdgeInsets.fromLTRB(10, 10, 20,20), // set value cho từng cạnh: Left - Top - Right - Bottom

        padding: EdgeInsets.only(left: 30), // set value cho các cạnh chỉ định

        padding: EdgeInsets.only(left: 30, bottom: 30),

        padding: EdgeInsets.symmetric(horizontal: pixel), // set value 2 cạnh đối xứng (symmetric)

        padding: EdgeInsets.symmetric(horizontal: 10, vertical: 40),

        // child: Text('Flutter self-taught'),

        child: Text('Flutter'),

      ),

    );

  }

}

Một số ngữ cảnh sử dụng padding

1. Trong widget cha để căn lề nội dung của các widget con:

Container(

            padding: EdgeInsets.all(16.0), // Khoảng cách xung quanh nội dung

            child: Text('Hello, world!'),

          )),

 ListView(

          padding: EdgeInsets.symmetric(vertical: 16.0), // Lề ở cả hai phía dọc

          children: [

            ListTile(title: Text('Item 1')),

            ListTile(title: Text('Item 2')),

            // ...

          ],

        )

1. Trong widget Card để tạo lề cho các chi tiết bên trong:

 Card(

          margin: EdgeInsets.all(16.0), // Khoảng cách xung quanh thẻ

          child: Column(

            children: [

              ListTile(title: Text('Title')),

              Padding(

                padding: EdgeInsets.all(16.0), // Lề cho nội dung bên trong

                child: Text('Content'),

              ),

            ],

          ),

        )

## Image

A widget that displays an image.

Several constructors are provided for the various ways that an image can be specified:

Image.asset, for obtaining an image from an AssetBundle using a key.

* Tạo folder chứa image (example /assets tại root folder)
* Khai báo trong file pubspec.yaml
* assets:
* - assets/ # Khai báo path tổng quát cho all file
* - assets/image3.jpg # Khai báo riêng cho từng file
* - assets/image2.jpg
* - assets/image1.png
* Truy cập
* children: [
* Image.asset('assets/image1.png', width: 150, height: 150),
* Image.asset('assets/image2.jpg', width: 150, height: 150),
* Image.asset('assets/image3.jpg', width: 150, height: 150),
* Image.asset('assets/fish.png', width: 150, height: 150),
* ],

Image.network, for obtaining an image from a URL.

 Widget build(BuildContext context) {

    return Row(

      mainAxisAlignment: MainAxisAlignment.spaceEvenly,

      children: [

        const Image(

            width: 150,

            height: 150,

            image: NetworkImage(// Fetches the given URL from the network

                'https://flutter.github.io/assets-for-api-docs/assets/widgets/owl.jpg')),

        Image.network(

            // Image.network for a shorthand of an Image widget backed by NetworkImage.

            'https://flutter.github.io/assets-for-api-docs/assets/widgets/owl-2.jpg',

            width: 150,

            height: 150),

      ],

    );

  }

}

Image.file, for obtaining an image from a File.

import 'dart:io'; // tham chiếu cho object File

import 'package:flutter/material.dart';

class MyFileImageWidget extends StatelessWidget {

  final File imageFile; // phải import 'dart:io'

  MyFileImageWidget(this.imageFile);

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(

          title: Text('File Image Example'),

        ),

        body: Center(

          child: Image.file(imageFile),

        ),

      ),

    );

  }

}

void main() {

  runApp(MyFileImageWidget(File('/path/to/your/image.jpg')));

}

Image.memory, for obtaining an image from a Uint8List.

The following image formats are supported: JPEG, PNG, GIF, Animated GIF, WebP, Animated WebP, BMP, and WBMP. Additional formats may be supported by the underlying platform. Flutter will attempt to call platform API to decode unrecognized formats, and if the platform API supports decoding the image Flutter will be able to render it.

To automatically perform pixel-density-aware asset resolution, specify the image using an AssetImage and make sure that a MaterialApp, WidgetsApp, or MediaQuery widget exists above the Image widget in the widget tree.

The image is painted using paintImage, which describes the meanings of the various fields on this class in more detail.

### Một số thuộc tính của widget Image

* Mức ưu tiên các properties set kích thước ảnh: có cảnh hưởng lẫn nhau nhưng ko được lớn hơn kích thước set với width/height
* Scale: ko ưu tiên bằng width/height. Thay đổi kích thước ảnh, default = 1, scale càng lớn ảnh càng nhỏ và ngược lại
* cacheWidth/cacheHeight: cung cấp kích thước ước tính của hình ảnh. Các giá trị này có thể giúp Flutter tối ưu hóa việc đọc và hiển thị hình ảnh từ cache, đặc biệt khi kích thước hiển thị không cần phải giống với kích thước gốc của hình ảnh.
* semanticLabel property: A Semantic description of the image. Used to provide a description of the image to TalkBack on Android, and VoiceOver on iOS.

TalkBack là một tính năng trợ năng tích hợp sẵn trên Android, được thiết kế để giúp đọc nội dung trên màn hình cho người dùng. Cụ thể, những mục, nội dung mà người dùng chạm vào sẽ được đọc thành tiếng khi tính năng này được bật.

TalkBack sẽ rất hữu ích với những người khiếm thị hoặc trong những tình huống mà người dùng không thể nhìn vào màn hình, nhưng vẫn muốn nắm bắt được nội dung đang hiển thị trên đó. Do vậy, nếu bạn hoàn toàn có thể nhìn và đọc nội dung trên màn hình thiết bị hoàn toàn bình thường, có thể tắt tính năng này đi chỉ với một vài thao tác đơn giản.

* opacity: độ trong suốt của ảnh – value = 1 is 100%, set < 1.

Trường hợp animation tĩnh - phải truyền bằng An animation that is always stopped at a given value như sau:

opacity: const AlwaysStoppedAnimation(0.7),

* color & colorBlendMode: trộn màu cho ảnh: color - màu muốn trộn với ảnh, colorBlendMode – kiểu trộn

Example

 child: Column(

            mainAxisAlignment: MainAxisAlignment.center,

            children: [

              Image.asset(

                'assets/bear.jpg',

                width: 200,

                height: 200,

                color: Colors.yellow[500], // set màu trộn

                colorBlendMode: BlendMode.color, // set kiểu trộn

              ),

              SizedBox(height: 10),

              Image.asset(

                'assets/bear.jpg',

                width: 200,

                height: 200,

                color: Colors.yellow[500],

                colorBlendMode: BlendMode.colorBurn,

              ),

              SizedBox(height: 10),

              Image.asset(

                'assets/bear.jpg',

                width: 200,

                height: 200,

                color: Colors.yellow[500],

                colorBlendMode: BlendMode.difference,

              ),

            ],

          ),

* fit: Thuộc tính fit của widget Image trong Flutter có nhiều giá trị khác nhau để điều chỉnh cách hình ảnh được hiển thị trong không gian hiển thị. Dưới đây là mô tả về tất cả các giá trị của thuộc tính fit

(Để test các fit value phải đặt image trong 1 widget cha như Container, set border trước)

1. **BoxFit.fill:** Hình ảnh sẽ được kéo dãn để che đậy toàn bộ không gian hiển thị mà có thể làm biến dạng hình ảnh nếu tỷ lệ khung hình không khớp.
2. **BoxFit.contain:** Hình ảnh sẽ được co lại hoặc kéo dãn sao cho nó phù hợp vừa vặn trong không gian hiển thị mà vẫn giữ nguyên tỷ lệ khung hình ban đầu.
3. **BoxFit.cover:** Hình ảnh sẽ được co lại hoặc kéo dãn sao cho nó che đậy toàn bộ không gian hiển thị mà vẫn giữ nguyên tỷ lệ khung hình ban đầu. Một phần của hình ảnh có thể bị cắt bỏ để vừa vặn.
4. **BoxFit.fitWidth:** Hình ảnh sẽ được kéo dãn chiều ngang để vừa vặn với chiều rộng của không gian hiển thị, giữ nguyên tỷ lệ khung hình.
5. **BoxFit.fitHeight:** Hình ảnh sẽ được kéo dãn chiều cao để vừa vặn với chiều cao của không gian hiển thị, giữ nguyên tỷ lệ khung hình.
6. **BoxFit.none:** Hình ảnh sẽ được hiển thị với kích thước thực tế của nó mà không điều chỉnh.
7. **BoxFit.scaleDown:** Hình ảnh sẽ được hiển thị như thường, nhưng nó sẽ co lại (nếu cần) để không vượt qua kích thước gốc của hình ảnh.

Mỗi giá trị của **fit** mang ý nghĩa và ứng dụng khác nhau tùy thuộc vào thiết kế giao diện cụ thể của bạn và cách bạn muốn hiển thị hình ảnh.

* - Top of Form
* alignment: căn lề cho image theo container (Để test các alignment value phải đặt image trong 1 widget cha như Container, set border trước)

Example

  children: [

              Container(

                width: 300,

                height: 200,

                decoration: BoxDecoration(

                  border: Border.all(color: Colors.blue, width: 2.0),

                ),

                child: Image.asset(

                  'assets/bear.jpg',

                  // width: 150,

                  // height: 150,

                  scale:

                      4.0, // size gốc ảnh > size container: set width/height thì ảnh tự bung ra full size container --> phải set qua scale

                  // fit: BoxFit.scaleDown,

                  alignment: Alignment.bottomCenter,

                ),

              ),

              SizedBox(height: 10),

              Container(

                width: 300,

                height: 200,

                decoration: BoxDecoration(

                  border: Border.all(color: Colors.blue, width: 2.0),

                ),

                child: Image.asset(

                  'assets/bear.jpg',

                  scale: 4,

                  alignment: Alignment.topRight,

                ),

              ),

              SizedBox(height: 10),

              Container(

                width: 300,

                height: 200,

                decoration: BoxDecoration(

                  border: Border.all(color: Colors.blue, width: 2.0),

                ),

                child: Image.asset(

                  'assets/bear.jpg',

                  scale: 4,

                  alignment: Alignment.centerRight,

                ),

              ),

* repeate: show lặp ảnh trong container

## Icon widget

Hiển thị hình ảnh các icon cơ bản trong *IconData* class.

Example

class MyIconWidget2 extends StatelessWidget {

  const MyIconWidget2({super.key});

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(

          title: const Text('Icon Widget Demo'),

        ),

        body: Column(

          children: [

            const Row(children: [

//Hiển thị Icon đơn giản

              SizedBox(

                height: 50,

                child: Text('Icon đơn giản'),

              ),

              Icon(

                Icons.star,

                color: Colors.yellow,

                size: 60.0,

              ),

            ]),

//Icon trong Button

            Row(children: [

              const SizedBox(

                height: 20,

                child: Text('Icon trong Button'),

              ),

              ElevatedButton.icon(

                onPressed: () {},

                icon: Icon(Icons.add),

                label: Text('Add'),

              ),

            ]),

// Icon cùng dòng với text

            const Row(children: [

              SizedBox(

                height: 20,

                child: Text(

                  'Icon cùng dòng với text',

                ),

              ),

              Row(

                children: [

                  Icon(Icons.email, color: Colors.blue),

                  SizedBox(width: 5),

                  Text('nmson2504@gmail.com'),

                ],

              ),

            ]),

            //

            /\*

            Đặt ListTile trực tiếp vào Row sẽ gây lỗi:

            Exception caught by scheduler library ═════════════════════════════════

Updated layout information required for RenderParagraph#b0f3f NEEDS-LAYOUT NEEDS-PAINT to calculate semantics.

'package:flutter/src/rendering/object.dart':

object.dart:1

Failed assertion: line 3457 pos 12: '!\_needsLayout'

           const Row(

              children: [

                Text('Icon trong ListTile'),

                ListTile(

                  leading: Icon(Icons.account\_circle),

                  title: Text('John Doe'),

                  subtitle: Text('Software Engineer'),

                  trailing: Icon(Icons.arrow\_forward),

                ),

              ],

            ), \*/

 // Icon trong ListTitle

            const Row(

              children: [

                Expanded(

                  child: Column(

                    children: [

                      Text('Icon trong ListTile'),

                      ListTile(

                        leading: Icon(Icons.account\_circle),

                        title: Text('John Doe'),

                        subtitle: Text('Software Engineer'),

                        trailing: Icon(Icons.arrow\_forward),

                      ),

                    ],

                  ),

                ),

              ],

            ),

// Icon trong nền 1 object

            Row(children: [

              const Text('Icon trong nền 1 object'),

              // Icon trong nền 1 object

              Container(

                width: 100,

                height: 100,

                decoration: const BoxDecoration(

                  color: Colors.blue,

                  shape: BoxShape.circle,

                ),

                child: const Icon(

                  Icons.ac\_unit,

                  color: Colors.white,

                  size: 20.0,

                ),

              ),

            ]),

// Icon cho 1 hành động

            Row(children: [

              const Text('Icon cho 1 hành động'),

              // Icon cho 1 hành động

              GestureDetector(

                onTap: () {

                  // Hành động khi người dùng chạm vào icon

                },

                child: const Icon(Icons.touch\_app),

              )

            ])

          ],

        ),

      ),

    );

  }

}

## Button

Trong Flutter, có một số loại button khác nhau mà bạn có thể sử dụng để tạo các tương tác người dùng khác nhau. Dưới đây là một số loại button phổ biến trong Flutter:

1. **ElevatedButton**: Đây là một loại button trung tính, có hiệu ứng độ nổi bật để làm nổi bật trước mặt. Nó thường được sử dụng cho các hành động quan trọng.
2. **TextButton**: Đây là một loại button văn bản đơn giản, thường được sử dụng cho các tương tác ít quan trọng hơn.
3. **OutlinedButton**: Loại button này có viền xung quanh và không có màu nền. Nó thường được sử dụng khi bạn muốn tạo ra các hành động như "Cancel" hoặc "Dismiss".
4. **IconButton**: Đây là một loại button có biểu tượng bên trong. Nó thường được sử dụng để thực hiện các hành động cụ thể.
5. **FloatingActionButton**: Đây là một button hình tròn nổi bật thường được đặt ở góc dưới bên phải màn hình. Nó thường được sử dụng cho các hành động chính trong ứng dụng.

Dưới đây là 1 số example code cùng với các properties của button

Example 1

//

Widget build(BuildContext context) {

    return Scaffold(body: Center(

      child: FloatingActionButton(onPressed: () {

        showDialog(

            context: context,

            builder: (ctxt) => new  AlertDialog(

                  title: Text("Text Dialog"),

                ));

      }),

    ));

  }

//

class MyAppImpl extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return Scaffold(

        appBar: AppBar(title: const Text('Alert Dialog Example')),

        body: Center(

          child: TextButton(

            onPressed: () {

              showDialog(

                  context: context,

                  builder: (ctxt) => new AlertDialog(

                        title: Text("Text Dialog"),

                      ));

            },

            child: Text('Text Button'),

          ),

        ));

  }

}

Example 2

class MyAppImpl3 extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return Scaffold(

        appBar: AppBar(title: const Text('Test Properties Example 2')),

        body: Center(

          child: ElevatedButton.icon(

            style: ElevatedButton.styleFrom(

              backgroundColor: // màu nền button

                  Color.fromARGB(255, 250, 253, 66), // cũ là primary

              foregroundColor: // màu chữ button

                  Color.fromARGB(163, 14, 246, 2), // cũ là onPrimary

              side: const BorderSide( // viền button

                // viền button

                color: Colors.blue,

                width: 2.0,

              ),

              shape: const RoundedRectangleBorder( // tạo định dạng bo góc cho button

                borderRadius: BorderRadius.horizontal(

                  left: Radius.circular(20), // Bán kính cho góc trái

                  right: Radius.circular(30), // Bán kính cho góc phải),

                ),

              ),

            ),

            onHover: (value) { // rê chuột lên

              print('onHover: .......');

            },

            onLongPress: () { // click giữ button

              // ấn giữ button

              print('onLongPress:.........');

            },

            onPressed: () { // click

              print('onPressed - ElevatedButton');

              // Lấy chủ đề hiện tại

            },

// child: Text('Elevated Button'), // khi dùng button.icon thì thuộc tính child: sẽ bị thay thế bằng 2 thuộc tính icon: và label:

            label: const Text('Elevated Button Icon'),

            icon: const Icon(

              Icons.abc,

              size: 30,

              color: Colors.redAccent,

            ),

          ),

        ));

  }

}

Example 3

class MyAppImpl1 extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return Scaffold(

        appBar: AppBar(title: const Text('Test Properties Example')),

        body: Center(

          child: ElevatedButton(

            style: ElevatedButton.styleFrom(

              backgroundColor:

                  Color.fromARGB(255, 250, 253, 66), // cũ là primary

              foregroundColor: Color.fromARGB(163, 14, 246, 2),

              disabledBackgroundColor: Color.fromARGB(255, 223, 234, 238), // để show hiệu ứng này thì phải disable button bằng lệnh: onPressed: null,

              disabledForegroundColor: const Color.fromARGB(221, 42, 29, 29),

              //  minimumSize: Size(110, 20),

              // maximumSize: Size(160, 20)

              padding: EdgeInsets.all(30),

              elevation: 10, // độ nổi (elevation) 3D so với nền

              shadowColor:

                  Colors.blue, // phải set elevation > 0 mới có tác dụng

              side: BorderSide(

                // viền button

                color: Colors.blue,

                width: 2.0,

              ),

              shape: RoundedRectangleBorder(

                borderRadius: BorderRadius.circular(

                    20), // Đặt hình dáng thành hình tròn với bán kính 20

              ),

            ),

            // onPressed: null, // disable button

            child: Text('Elevated Button'),

          ),

        ));

  }

}

Xử lý lỗi "FlutterError (No MaterialLocalizations found…" click button (run app rồi vẫn chưa báo lỗi):

// Khi widget return ko có MaterialApp và void main() gọi …

class MyButtonAppError1 extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

        home: Scaffold(

            appBar: AppBar(title: const Text('Alert Dialog Example')),

            body: Center(

              child: FloatingActionButton(onPressed: () {

                // Hiển thị hộp thoại khi button được nhấn

                showDialog(

                    context: context,

                    builder: (ctxt) => new AlertDialog(

                          title: const Text("Text Dialog"),

                        ));

              }),

              // child: Text('Hiển thị Hộp thoại'),

            )));

  }

}

// file main.dart – void main() gọi:

void main() {

  runApp(MyButtonAppError1());

}

// Click button sẽ báo lỗi: "FlutterError (No MaterialLocalizations found…"

// Giải pháp: tách MartialApp ra nằm ở widget cha (widget gọi widget chứa các thành phần con của MaterialApp)

// return MaterialApp ở widget cha

class MyButtonApp1 extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(home: MyAppImpl());

  }

}

class MyAppImpl extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return Scaffold(

        appBar: AppBar(title: const Text('Alert Dialog Example')),

        body: Center(

          child: TextButton(

            onPressed: () {

              showDialog(

                  context: context,

                  builder: (ctxt) => new AlertDialog(

                        title: Text("Text Dialog"),

                      ));

            },

            child: Text('Text Button'),

          ),

        ));

  }

}

// void main() gọi widget cha:  runApp(MyButtonApp1());

// Hoặc: gọi trực tiếp widget con như sau: runApp(MaterialApp(home: MyAppImpl()));

## Tạo lưới trên màn hình bằng GridView

class MyGrid extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return MaterialApp(

      home: MyGridPage(),

    );

  }

}

class MyGridPage extends StatelessWidget {

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: Text('Grid Overlay Example'),

      ),

      body: Stack(

        children: [

          // Your main content goes here

          Center(

            child: Text('Main Content'),

          ),

          // Overlay grid

          GridView.builder(

            gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(

              crossAxisCount: 10, // Số cột trong lưới

            ),

            itemBuilder: (context, index) {

              return Container(

                decoration: BoxDecoration(

                  border: Border.all(color: Colors.grey.shade300),

                ),

              );

            },

          ),

        ],

      ),

    );

  }

## Container

Container trong Flutter được sử dụng để tạo một khung chứa (container) để bao bọc và tổ chức các widget con bên trong. Nó cung cấp nhiều thuộc tính cho phép bạn tùy chỉnh kích thước, màu sắc, viền, độ trong suốt và nhiều tính năng khác. Mục đích chính của việc sử dụng widget Container là để thiết lập và tùy chỉnh hình dáng và vị trí của các phần tử trong giao diện ứng dụng của bạn.

* Height, Width:

Default ko set width, height và ko có object con: container bao phủ full screen.

Nếu chỉ có object con mà ko set width, height: container tự động co kéo cho vừa object con

Nếu set width, height: container giữ cứng kích thước này.

Widget build(BuildContext context) {

    return MaterialApp(

        home: Scaffold(

      appBar: AppBar(title: Text('Flutter Container')),

      body: Container(

        color: Colors.amberAccent,

        width: 400,

        height: 400,

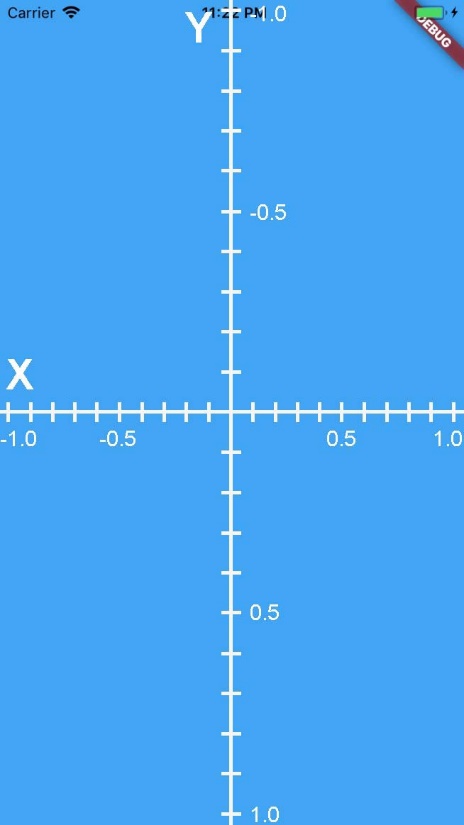
        child: Text('This is container'),

      ),

    ));

  }

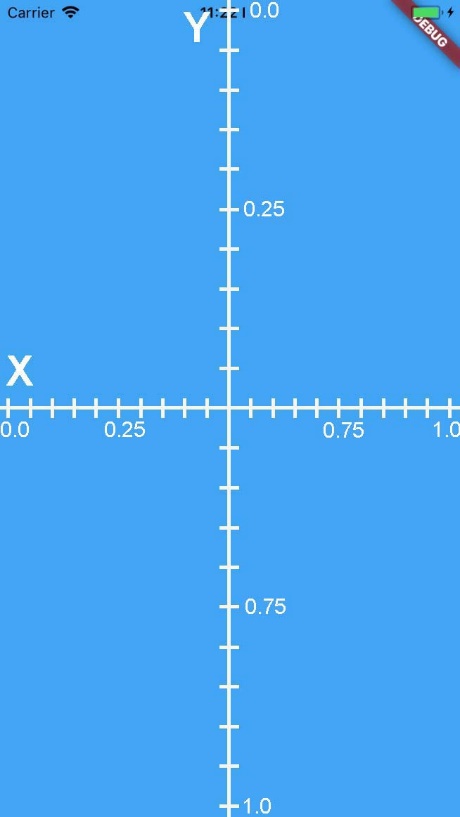
* alignment: set vị trí object con so với container theo các value constant or hệ (x,y) như ảnh dưới: gốc toạ độ (0,0) nằm giữa màn hình



  alignment: Alignment.centerLeft, // alignment object con so với khung container = constant

  alignment: Alignment(-1, -1), // alignment object con so với khung container = (x,y)

set vị trí object con so với container theo hệ FractionalOffset(x,y): x chạy từ left to right (0 -> 1), y chạy từ top to bottom (0 ->1)



  alignment: FractionalOffset(0.25, 0.25),

* padding & margin:

padding: EdgeInsets.all(20), // padding object con với khung container. nếu có gán alignment thì padding += alignment

margin: EdgeInsets.all(20), // margin của container này với object cha chứa nó

* child: chứa nội dung trong container
* decoration: các thuộc tính định dạng container nâng cao.
* decoration: BoxDecoration(
* color: const Color.fromARGB(255, 43, 221, 90).withOpacity(
* 0.3), // ko thể set color cùng lúc với color trước đó của container (chỉ có thể chọn 1 trong 2)
* border: Border.all(
* width: 2,
* color: Colors.red,
* ),
* borderRadius: BorderRadius.circular(20), // or borderRadius: const BorderRadius.all(Radius.circular(20)),
* shape: BoxShape.circle,
* /\*
* Lưu ý khi sử dụng BoxShape.circle và borderRadius cùng lúc sẽ báo lỗi
* Exception has occurred.
* \_AssertionError
* ('package:flutter/src/painting/box\_decoration.dart': Failed assertion: line 128 pos 12: 'shape != BoxShape.circle || borderRadius == null': is not true.)
* --> bỏ set 1 trong 2
* \*/
* boxShadow: const [
* BoxShadow(
* color: Color.fromARGB(255, 109, 239, 22), // Màu của bóng
* offset: Offset(3, 3), // Vị trí tương đối của bóng (ngang, dọc)
* blurRadius: 20, // Độ mờ của bóng
* spreadRadius: 0, // Độ lan tỏa của bóng
* ),
* ],
* backgroundBlendMode: BlendMode.exclusion, // Chế độ blend – trộn màu
* image: const DecorationImage( // set ảnh nền
* image: AssetImage('assets/image3.jpg'), // Đường dẫn hình ảnh
* fit: BoxFit
* .cover, // Cách hình ảnh được hiển thị (cover, contain, ...)
* ),

// gradient: hiệu ứng loang màu. Có 3 loại: SweepGradient, RadialGradient và LinearGradient

gradient: SweepGradient(

* colors: [Colors.red, Colors.blue],
* center: Alignment.center, // Điểm tâm gradient
* startAngle: 0, // Góc bắt đầu gradient (độ)
* endAngle: 3.14, // Góc kết thúc gradient (độ)
* ),
* // or
* gradient: RadialGradient(
* colors: [Colors.yellow, Colors.orange],
* center: Alignment.center, // Điểm tâm gradient
* radius: 0.8, // Bán kính gradient
* ),
* // or
* gradient: LinearGradient(
* colors: [
* Colors.blue,
* Colors.purple
* ], // Danh sách màu tạo gradient
* begin: Alignment.topLeft, // Điểm bắt đầu gradient
* end: Alignment.bottomRight, // Điểm kết thúc gradient
* ),
* ),

## SizedBox

SizedBox là một hộp trong suốt, khác với Container bạn không thể thiết lập kiểu dáng cho nó (chẳng hạn mầu nền, margin, padding,...). Nếu bạn chỉ định một kích thước cụ thể cho SizedBox, kích thước đó cũng sẽ được áp dụng cho widget con của nó. Ngược lại nếu chiều rộng của SizedBox không được chỉ định hoặc null thì widget con của sẽ có chiều rộng theo thiết lập riêng hoặc bằng 0 (nếu không được thiết lập). Với chiều cao cũng có hành vi tương tự.

Tất cả các tham số tham gia vào việc tạo ra một SizedBox như width, height, size, child đều là các tuỳ chọn (Không bắt buộc).

return MaterialApp(

      home: Scaffold(

          appBar: AppBar(title: const Text('Flutter SizedBox')),

          body: SizedBox(

            width: 300,

            height: double

                .maxFinite, // double.infinity - giá trị vô cực, full hết màn hình

            /\*

            cực âm: double.negativeInfinity, double.minPositive

            cực dương: double.infinity, double.maxFinite

             \*/

            child: Container(

              color: Colors.blue,

              child: const Center(

                child: Text('SizedBox Example'),

              ),

            ),

          )),

    );

  }

Một ứng dụng thường được sử dụng là chèn SizedBox vào làm khoảng cách giữa các object khác

  Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

        appBar: AppBar(title: const Text('Flutter SizedBox Space')),

        body: const Row(children: [

          Text(

            'text 001',

            style: TextStyle(backgroundColor: Colors.deepOrange),

          ),

          SizedBox(

            width: 10,

          ),

          Text(

            'text 011',

            style: TextStyle(backgroundColor: Colors.deepOrange),

          ),

          SizedBox(

            width: 10,

          ),

          Text(

            'text 101',

            style: TextStyle(backgroundColor: Colors.deepOrange),

          ),

          SizedBox(

            width: 10,

          ),

          Text(

            'text 111',

            style: TextStyle(backgroundColor: Colors.deepOrange),

          ),

        ]),

      ),

    );

  }

Constructor SizedBox.shrink được sử dụng để tạo ra một SizedBox với kích thước nhỏ nhất theo chỉ định từ widget cha của nó.

 Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

          appBar: AppBar(title: const Text('Flutter SizedBox 2')),

          body: ConstrainedBox(

              constraints: const BoxConstraints(

                // Min: 80x20 - ghi đè lên các thuộc tính tương đương của object con

                // \*Nếu ko set values cặp thuộc tính minWidth/minHeight thì object con sẽ ko show ra

                minWidth: 200.0,

                minHeight: 200.0,

              ),

              child: SizedBox.shrink(

                child: ElevatedButton(

                    child: const Text('Button'),

                    onPressed: () {},

                    style: ElevatedButton.styleFrom(

                        backgroundColor:

                            Color.fromARGB(255, 250, 253, 66), // cũ là primary

                        foregroundColor: Color.fromARGB(163, 14, 246, 2),

                        fixedSize: const Size(300, 300), // ko tác dụng

                        minimumSize: Size(200, 200))), // ko tác dụng

              ))),

    );

  }

}

Constructor SizedBox.fromSize được sử dụng để tạo một SIzedBox với kích thước được chỉ định thông qua tham số tuỳ chọn - size.

 home: Scaffold(

            appBar: AppBar(title: const Text('Flutter SizedBox.fromSize')),

            body: SizedBox.fromSize(

              size: const Size(200, 200),

              child: ElevatedButton(

                  onPressed: () {},

                  style: ElevatedButton.styleFrom(

                    backgroundColor: const Color.fromARGB(255, 250, 253, 66), // cũ là primary

                  ),

                  child: const Text('Button SizedBox.fromSize')),

            )));

## ConstrainedBox

ConstrainedBox là một widget trong Flutter được sử dụng để giới hạn kích thước của widget con bên trong nó. Bạn có thể sử dụng ConstrainedBox để đặt giới hạn về chiều rộng, chiều cao hoặc cả hai cho widget con.

ghi đè lên các thuộc tính tương đương của object con

Widget build(BuildContext context) {

    return MaterialApp(

      home: Scaffold(

          appBar: AppBar(title: const Text('Flutter SizedBox 2')),

          body: ConstrainedBox(

            constraints: const BoxConstraints(

              minWidth: 100, // Chiều rộng tối thiểu

              maxWidth: 200, // Chiều rộng tối đa

              minHeight: 50, // Chiều cao tối thiểu

              maxHeight: 150, // Chiều cao tối đa

            ),

            child: Container(

              color: Colors.blue,

              child: const Center(

                child: Text('ConstrainedBox Example'),

              ),

            ),

          )),

    );

  }

\*Nếu ko set values cặp thuộc tính minWidth/minHeight thì object con sẽ ko show ra

## Các loại hộp thoại

## 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(),

    ),

  ));

}

## Widget có khả năng chứa nhiều widget con

Một số widget coa khả năng chứa nhiều widget con như Column, Row, Stack, **ListView**,TextSpan,…

Example 1

child: Column(

            mainAxisAlignment: MainAxisAlignment.center,

            children: [

              PaddingAndMargin\_01(),

              PaddingAndMargin\_02(),

            ],

          ),

## Thuộc tính **children**

Thuộc tính children là một thuộc tính phổ biến trong các widget trong Flutter, đặc biệt là các widget nhóm như Column, Row, ListView, Stack và nhiều widget khác. Thuộc tính này cho phép bạn cung cấp một danh sách các widget con mà bạn muốn hiển thị hoặc xử lý trong một widget cha.

Cú pháp cơ bản của thuộc tính children là:

List<Widget> children = [

// Danh sách các widget con

Widget1(),

Widget2(),

// ...

];

Một số ví dụ triển khai

Column(

  children: [

    Text('Widget 1'),

    Text('Widget 2'),

    Text('Widget 3'),

  ],

)

//--------------------

Row(

  children: [

    Icon(Icons.star),

    Icon(Icons.star),

    Icon(Icons.star),

  ],

)

//--------------------

ListView(

  children: [

    ListTile(title: Text('Item 1')),

    ListTile(title: Text('Item 2')),

    ListTile(title: Text('Item 3')),

  ],

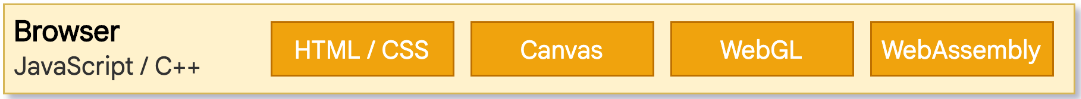
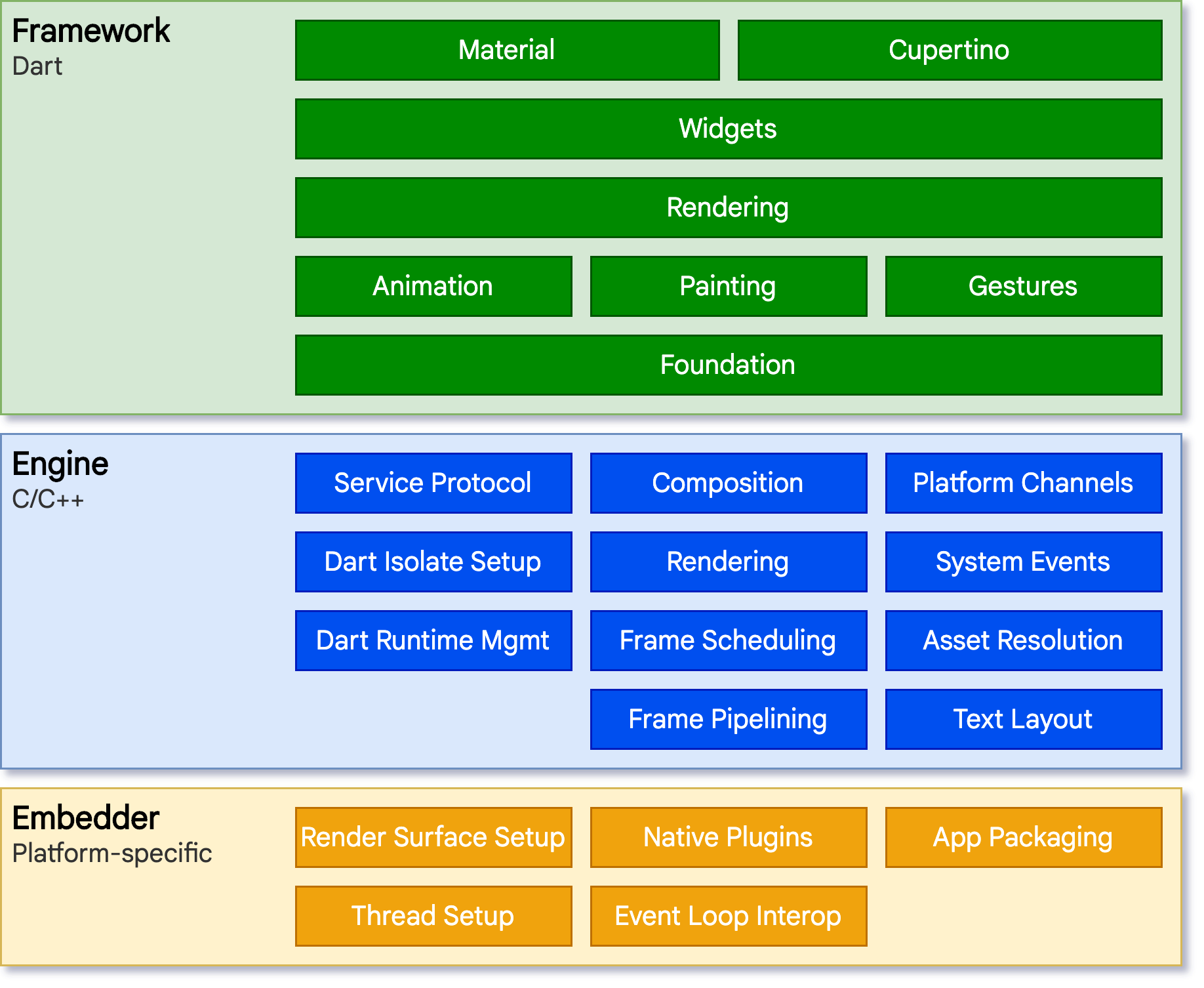
)

## Gestures

Flutter widget hỗ trợ tương tác thông qua một widget đặc biệt gọi là GestureDetector. GestureDetector không hiển thị trên giao diện nhưng có khả năng nắm bắt các thao tác của người dùng như nhấp, kéo, vuốt, chạm.... Phần lớn widget gốc của Flutter hỗ trợ tương tác giao diện thông qua GestureDetector.

## Layers

Layers are an important concept of the Flutter framework, which are grouped into multiple categories in terms of complexity and arranged in the top-down approach. The topmost layer is the UI of the application, which is specific to the Android and iOS platforms. The second topmost layer contains all the Flutter native widgets. The next layer is the rendering layer, which renders everything in the Flutter app. Then, the layers go down to Gestures, foundation library, engine, and finally, core platform-specific code. The following diagram specifies the layers in Flutter app development.



## Reference

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

<https://docs.flutter.dev/cookbook>

<https://fonts.google.com> – fonts, icons và hướng dẫn sử dụng

## Set default size window app

Mở file windows\runner\main.cpp trong project flutter, edit values như bên dưới

Win32Window::Point origin(1000, 10); // set value vị trí góc trên-trái (lề trái, lề trên)

Win32Window::Size size(600, 800); // set value size window (width, height)