**BÁO CÁO CUỐI KỲ**

**Họ và tên:** Nguyễn Kỳ Anh  
**MSSV:** 20225793  
 **GR1**  
**Giáo viên hướng dẫn:** Nguyễn Đức Toàn

# Cài đặt Flutter cấu hình JDK trên VSCode

* Hệ điều hành: Windows 10 (64- bit)
* Flutter SDK: phiên bản 3.29.3
* Dart SDK: phiên bản 3.7.2
* DevTools: Phiên bản 2.24.3
* Trình biên tập mã : Visual Studio Code

# Làm quen với ngôn ngữ Dart

## Một số kiểu dữ liệu

|  |  |  |
| --- | --- | --- |
| **Kiểu dữ liệu** | **Mô tả** | **Ví dụ** |
| int | Đại diện cho số nguyên | int a = 10; |
| double | Đại diện cho số thực | double pi = 3.14; |
| bool | Đại diện cho giá trị luận lý đúng/sai | bool isValid = true; |
| String | Chuỗi ký tự | String name = "Ngọc"; |
| var | Kiểu suy diễn, tự nhận kiểu từ giá trị ban đầu | var city = "Hà Nội"; |
| dynamic | Có thể thay đổi kiểu dữ liệu trong quá trình chạy | dynamic value = 5; value = "Five"; |

## Các kiểu toán tử trong Dart.

### Toán tử số học

int a = 10, b = 3;

print(a + b); // 13

print(a % b); // 1

### Toán tử quan hệ

print(a > b); // true

print(a == b); // false

### Toán tử logic

bool isOpen = true;

bool isFull = false;

print(isOpen && isFull); // false

### Toán tử gán

int x = 5;

x += 3; // x = 8

### Toán tử điều kiện

int score = 85;

String result = (score >= 50) ? "Đậu" : "Rớt";

print(result);

## Câu lệnh điều kiện và vòng lặp

### Câu lệnh điều kiện

int age = 20;

if (age >= 18) {

print("Người lớn");

} else {

print("Trẻ em");

}

### Vòng lặp for, while,do-while

// For loop

for (int i = 1; i <= 3; i++) {

print("Lần $i");

}

// While loop

int j = 0;

while (j < 3) {

print("While $j");

j++;

}

// Do-while loop

int k = 0;

do {

print("Do-while $k");

k++;

} while (k < 2);

## Hàm cơ bản trong Dart

### Hàm có kiểu trả về

int tinhTong(int a, int b) {

return a + b;

}

### Hàm không có kiểu trả về.

void chaoNguoiDung(String ten) {

print("Xin chào, $ten!");

}

## Enum và Iterable

### Enum

enum TrangThai { choDuyet, daDuyet, daHuy }

TrangThai tt = TrangThai.daDuyet;

print(tt); // TrangThai.daDuyet

### Iterable

List<String> dsTen = ['An', 'Bình', 'Chi'];

for (var ten in dsTen) {

print(ten);

}

## Cấu trúc dữ liệu List, Map, Set

### List

List<int> numbers = [1, 2, 3];

numbers.add(4);

print(numbers); // [1, 2, 3, 4]

### Map

Map<String, int> tuoi = {

"An": 25,

"Bình": 30

};

print(tuoi["An"]); // 25

### Set

Set<int> tapHop = {1, 2, 2, 3};

print(tapHop); // {1, 2, 3}

## Truyền tham số cho hàm

### Tham số vị trí

void hienThi(String ten, int tuoi) {

print("$ten - $tuoi tuổi");

}

### Tham số đặt tên

void show({required String name, int age = 18}) {

print("Tên: $name - Tuổi: $age");

}

## Class trong Dart

class Person {

String name;

int age;

Person(this.name, this.age);

void introduce() {

print("Tôi tên là $name, $age tuổi.");

}

}

## Getter và Setter

class Car {

String \_model;

Car(this.\_model)

String get model => \_model;

set model(String value) => \_model = value;

}

## Initializer List

class Point {

final int x;

final int y;

Point(int a, int b) : x = a, y = b;

}

## Named Constructors

class Animal {

String name;

Animal(this.name);

Animal.fromJson(Map<String, String> json) : name = json['name'] ?? 'Unknown';

}

var a = Animal.fromJson({'name': 'Chó'});

## Hàng đợi – Queue

* Với hàng đợi Queue thì chỉ tương tác với cấu trúc này ở đầu hàng đợi và cuối hàng đợi (thêm / bớt).

|  |  |
| --- | --- |
| add, addLast | Thêm phần tử vào cuối hàng add(ele); hoặc addLast(ele) |
| addFirst | Thêm phần tử vào đầu hàng addFirst(ele); |
| removeFirst() | Xóa phần tử đầu tiên |
| removeLast() | Xóa phần tử cuối cùng |
| remove() | Xóa 1 phần tử remove(ele) |

**VD:**

var q = new Queue();

q.add('A');

q.add('B');

q.addFirst('A0');

q.addLast('B0');

print(q);

q.removeFirst();

q.removeLast();

q.remove('B');

## Function

Các cách khai báo hàm

* Hàm có tên: Khai báo với tên rõ ràng, có thể trả về giá trị (ví dụ: int add(int a, int b) { return a + b; }) hoặc không (void sayHello(String name) { print('Hello, $name!'); }).
* Hàm mũi tên: Ngắn gọn cho biểu thức đơn (int add(int a, int b) => a + b;).
* Hàm ẩn danh: Không tên, dùng trong forEach hoặc map (ví dụ: list.forEach((item) => print(item \* 2));).
* Hàm với tham số tùy chọn: Sử dụng [] (positional) hoặc {} (named) (ví dụ: String greet(String name, [String greeting = 'Hello']) => '$greeting, $name!';).
* Hàm gán cho biến: Gán hàm cho biến để sử dụng như đối tượng (var add = (int a, int b) => a + b;).
* Hàm với kiểu dynamic: Trả về nhiều kiểu dữ liệu (dynamic getValue(bool isNumber) => isNumber ? 42 : 'Hello';).

## Tính kế thừa

* Trong một phần mềm hay chương trình, được cấu tạo bởi nhiều lớp khác nhau cùng các thành phần khác.
* Mối quan hệ giữa các lớp, có mối quan hệ kế thừa, gồm lớp cha ( super class )và các lớp con(sub class), Các lớp con đó lại có thể là lớp cha của các lớp khác.

A diagram of a diagram

AI-generated content may be incorrect.

* Mục đích của kế thừa là tái sử dụng.  
  Lớp con có thể sở hữu các thuộc tính và phương thức public của lớp cha nhưng không được sở hữu các thuộc tính hay phương thức private và các hàm constructor.
* Biểu diễn kế thừa trong Dart cũng tương tự như trong Java : dùng extends

ví dụ:

class Car extends XeBonBanh{...}

class Truck extends XeBonBanh{...}

## Abstract (Lớp trừu tượng)

* Lớp trừu tượng trong ngôn ngữ Dart là lớp có hành vi chưa được xác định rõ.
* Ghi đè phương thức : Overriding – là trườnh hợp khi lớp con muốn sửa phương thức của lớp cha. Đặc biệt nếu lớp cha có những phương thức trừu tượng thì việc ghi đè là bắt buộc.
* Từ khoá super trong kế thừa: Dùng để truy cập đến các biến, hàm ( public) và cả constructor của lớp cha.

VD:

// Lớp cha

class LopCha {

  void sendMessage(){

  print ("Message từ lớp cha");

  }

// Lớp con

import 'package:dart\_language/lop\_cha.dart';

class LopCon extends LopCha{

  @override

  void sendMessage() {

    // TODO: implement sendMessage

    super.sendMessage();

    print ("Message từ lớp con");

  }

}

// Sử dụng trong main.dart

import 'package:dart\_language/lop\_con.dart';

void main() {

  var lopCon = LopCon();

  lopCon.sendMessage();

}

* Kết quả

****

## Tính đa hình.

- Tính đa hình trong ngôn ngữ Dart cũng có ý nghĩa giống như trong các ngôn ngữ khác. Cùng biểu diễn một hành vi nhưng từng lớp có cách biểu diễn khác nhau.

VD:

// Khai báo về tính năng của Xe

abstract class Xe {

  void chuyenCho(); // Chuyên chở

}

// Khai báo của Xe tải

import 'package:dart\_language/XeCo.dart';

class XeTai extends Xe{

  @override

  void chuyenCho() {

    print (" Chỉ chở hàng");

  }

}

// Khai báo về Xe khách

import 'package:dart\_language/XeCo.dart';

class XeKhach extends Xe{

  @override

  void chuyenCho() {

    print (" Chỉ chở người");

  }

}

## Mixin progaramming

- Mixin là một cách định nghĩa mã có thể được sử dụng lại trong nhiều hệ thống phân cấp lớp. Chúng được thiết kế để cung cấp các triển khai thành viên hàng loạt.

- Để sử dụng mixin, hãy sử dụng with từ khóa theo sau là một hoặc nhiều tên mixin. Ví dụ sau đây cho thấy hai lớp sử dụng (hoặc, là các lớp con của) mixin:

class Musician extends Performer with Musical {

// ···

}

class Maestro extends Person with Musical, Aggressive, Demented {

Maestro(String maestroName) {

name = maestroName;

canConduct = true;

}

}

- Để định nghĩa một mixin, hãy sử dụng mixinkhai báo. Trong trường hợp hiếm hoi khi bạn cần định nghĩa cả mixin *và* class, bạn có thể sử dụng [mixin class khai báo](https://dart.dev/language/mixins#class-mixin-or-mixin-class) .

Mixin và các lớp mixin không thể có extends mệnh đề và không được khai báo bất kỳ hàm tạo nào.

Ví dụ:

mixin Musical {

bool canPlayPiano = false;

bool canCompose = false;

bool canConduct = false;

void entertainMe() {

if (canPlayPiano) {

print('Playing piano');

} else if (canConduct) {

print('Waving hands');

} else {

print('Humming to self');

}

}

}

* + Chỉ định các thành viên mà mixin có thể gọi đến chính nó

Đôi khi một mixin phụ thuộc vào khả năng gọi một phương thức hoặc truy cập các trường, nhưng không thể tự xác định các thành viên đó (vì mixin không thể sử dụng tham số xây dựng để khởi tạo các trường của riêng chúng).

- Các phần sau đây đề cập đến các chiến lược khác nhau để đảm bảo bất kỳ lớp con nào của mixin đều xác định bất kỳ thành viên nào mà hành vi của mixin phụ thuộc vào.

Xác định các thành viên trừu tượng trong mixin

- Việc khai báo một phương thức trừu tượng trong mixin sẽ buộc bất kỳ kiểu nào sử dụng mixin phải xác định phương thức trừu tượng mà hành vi của nó phụ thuộc vào.

VD:

mixin Musician {

void playInstrument(String instrumentName); // Abstract method

void playPiano() {

playInstrument('Piano');

}

void playFlute() {

playInstrument('Flute');

}

}

class Virtuoso with Musician {

@override

void playInstrument(String instrumentName) { // Subclass must define.

print('Plays the $instrumentName beautifully');

}

}

* + Triển khai một giao diện

Tương tự như việc khai báo tóm tắt mixin, việc đặt một implementsmệnh đề vào mixin trong khi không thực sự triển khai giao diện cũng sẽ đảm bảo mọi phụ thuộc thành viên đều được xác định cho mixin.

VD:

abstract interface class Tuner {

void tuneInstrument();

}

mixin Guitarist implements Tuner {

void playSong() {

tuneInstrument();

print('Strums guitar majestically.');

}

}

class PunkRocker with Guitarist {

@override

void tuneInstrument() {

print("Don't bother, being out of tune is punk rock.");

}

}

* + Sử dụng onmệnh đề để khai báo một siêu lớp

Mệnh đề này ontồn tại để xác định loại mà supercác cuộc gọi được giải quyết. Vì vậy, bạn chỉ nên sử dụng nó nếu bạn cần có một supercuộc gọi bên trong một mixin.

Mệnh đề này onbuộc bất kỳ lớp nào sử dụng mixin cũng phải là lớp con của loại trong onmệnh đề. Nếu mixin phụ thuộc vào các thành viên trong siêu lớp, điều này đảm bảo các thành viên đó có sẵn khi mixin được sử dụng.

VD:

class Musician {

musicianMethod() {

print('Playing music!');

}

}

mixin MusicalPerformer on Musician {

performerMethod() {

print('Performing music!');

super.musicianMethod();

}

}

class SingerDancer extends Musician with MusicalPerformer { }

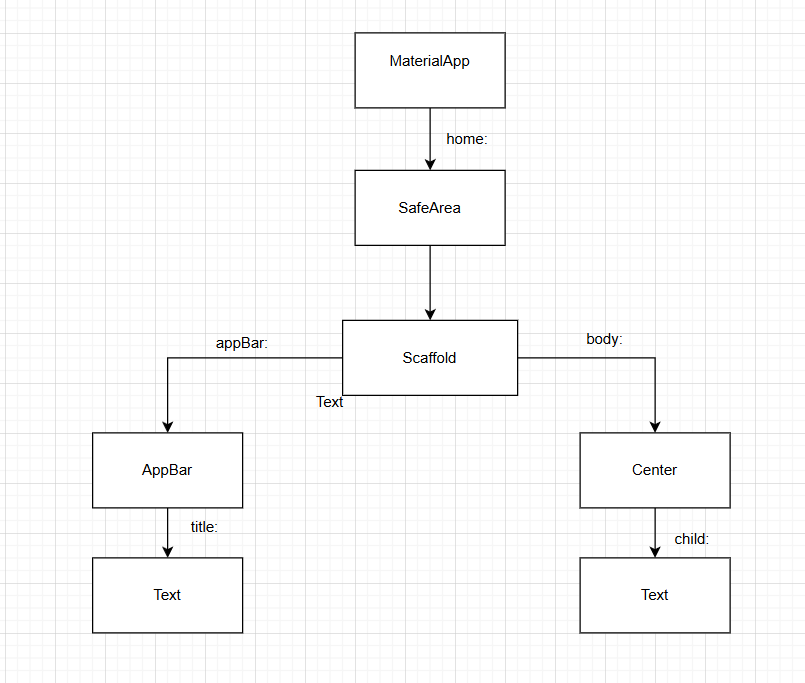
main() {

SingerDancer().performerMethod();

}

# Widgets

Widget tree:



Chúng ta có thể tạo một widget trong Flutter như sau:

Class ImageWidget extends StatelessWidget {

// Class-Stuff

}

VD: Hello Everyone

import 'package:flutter/material.dart';

class HomePage extends StatelessWidget {

// Constructor của HomePage, nhận vào một tiêu đề (title)

HomePage({Key? key, this.title}) : super(key: key);

// Đây là widget màn hình chính của ứng dụng

final String title;

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(

// Hiển thị tiêu đề của trang

title: Text(this.title),

),

body: Center(

// Hiển thị văn bản "Hello Everyone" ở giữa màn hình

child: Text('Hello Everyone'),

),

);

}

}

- WIDGET TYPES: Widget trong Flutter có thể được chia thành hai loại:

* Hiển thị (Visible): Gồm các widget đầu ra (Output) và đầu vào (Input).
* Ẩn (Invisible): Gồm các widget bố cục (Layout) và điều khiển (Control).

## StatelessWidget

* StatelessWidget là một loại widget không có trạng thái trong Flutter. Nó có đặc điểm:
* Không thay đổi trong suốt vòng đời – Dữ liệu bên trong StatelessWidget là bất biến.
* Chỉ được xây dựng một lần – Khi widget được tạo, nó không thể cập nhật lại giao diện trừ khi được khởi tạo lại từ đầu.
* Phù hợp cho giao diện tĩnh – Các phần của UI không cần thay đổi, chẳng hạn như văn bản, hình ảnh, biểu tượng.

Khi nào sử dụng StatelessWidget?

* Hiển thị nội dung tĩnh như văn bản, hình ảnh, nút bấm mà không cần cập nhật lại.
* Không cần lắng nghe sự kiện thay đổi từ người dùng.
* Khi muốn tối ưu hiệu suất vì StatelessWidget không yêu cầu quản lý trạng thái.

Vd:

import 'package:flutter/material.dart';

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: HomePage(),

);

}

}

class HomePage extends StatelessWidget {

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('StatelessWidget Example')),

body: Center(

child: Text('Hello, this is a StatelessWidget!'),

),

);

}

}

void main() {

runApp(MyApp());

}

## StatefulWidget

* Khái niệm: StatefulWidget là một loại widget có trạng thái trong Flutter. Nó có khả năng thay đổi dữ liệu trong quá trình chạy ứng dụng và cập nhật lại giao diện khi trạng thái thay đổi.
* Đặc điểm của StatefulWidget:

+ Có thể thay đổi trạng thái (state) – Dữ liệu bên trong có thể thay đổi sau khi widget được tạo.  
+ Cập nhật lại giao diện (rebuild) – Khi trạng thái thay đổi, widget sẽ tự động cập nhật lại.

+ Tạo ra hai lớp:

* Lớp chính (StatefulWidget) – Xác định widget.
* Lớp trạng thái (State) – Chứa dữ liệu thay đổi và logic cập nhật UI.

- Khi nào sử dụng StatefulWidget?

* Khi giao diện cần thay đổi theo dữ liệu hoặc sự kiện.
* Khi có tương tác người dùng (ví dụ: nút bấm, nhập dữ liệu).
* Khi muốn cập nhật trạng thái mà không cần khởi tạo lại widget từ đầu.

VD:

import 'package:flutter/material.dart';

void main() {

runApp(MyApp());

}

class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

home: CounterScreen(), // Màn hình chính

);

}

}

// Lớp StatefulWidget

class CounterScreen extends StatefulWidget {

@override

\_CounterScreenState createState() => \_CounterScreenState();

}

// Lớp State chứa logic

class \_CounterScreenState extends State<CounterScreen> {

int \_counter = 0; // Biến trạng thái

// Hàm tăng giá trị bộ đếm

void \_incrementCounter() {

setState(() {

\_counter++; // Cập nhật trạng thái

});

}

@override

Widget build(BuildContext context) {

return Scaffold(

appBar: AppBar(title: Text('StatefulWidget Example')),

body: Center(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Text(

'Số lần nhấn:',

style: TextStyle(fontSize: 20),

),

Text(

'$\_counter', // Hiển thị giá trị bộ đếm

style: TextStyle(fontSize: 40, fontWeight: FontWeight.bold),

),

],

),

),

floatingActionButton: FloatingActionButton(

onPressed: \_incrementCounter, // Gọi hàm tăng số

child: Icon(Icons.add),

),

);

}

}

## Text

* Data; Chuỗi văn bản hiển thị.
* Style: Định dạng văn bản như kích thước, màu sắc, độ đậm, kiểu chữ.

VD:

Text(

'Đây là văn bản có định dạng!',

style: TextStyle(

fontSize: 24, // Cỡ chữ

fontWeight: FontWeight.bold, // Độ đậm (bold)

fontStyle: FontStyle.italic, // Kiểu chữ nghiêng

letterSpacing: 2.0, // Khoảng cách giữa các chữ cái

wordSpacing: 5.0, // Khoảng cách giữa các từ

color: Colors.blue, // Màu chữ

decoration: TextDecoration.underline, // Gạch chân

decorationColor: Colors.red, // Màu của gạch chân

decorationThickness: 2.0, // Độ dày của gạch chân

),

)

* TextAlign: Căn chỉnh văn bản( left, center , right, justify).
* TextAlign.left – Căn trái (mặc định).
* TextAlign.right – Căn phải.
* TextAlign.center – Căn giữa.
* TextAlign.justify – Căn đều hai bên.

VD:

Text(

'Chào mừng bạn đến với Flutter!',

textAlign: TextAlign.center, // Căn giữa văn bản

)

* **textDirection**: Hướng văn bản (LTR - trái sang phải, RTL - phải sang trái).
* TextDirection.ltr – Từ trái sang phải (mặc định, dùng cho tiếng Anh, Việt Nam…).
* TextDirection.rtl – Từ phải sang trái (dùng cho tiếng Ả Rập, Do Thái…).

VD:

Text(

'مرحبا بك في Flutter!',

textDirection: TextDirection.rtl, // Văn bản từ phải sang trái (dùng cho tiếng Ả Rập, Hebrew)

)

* softWrap: Cho phép hoặc không cho phép xuống dòng tự động.

Giá trị có thể sử dụng:

* true – Văn bản sẽ tự động xuống dòng khi chạm mép màn hình.
* false – Văn bản sẽ không xuống dòng, có thể bị cắt hoặc tràn khỏi màn hình.

VD:

Text(

'Flutter là một framework mạnh mẽ giúp lập trình viên phát triển ứng dụng đa nền tảng.',

softWrap: false, // Không tự động xuống dòng

)

* overflow: Xử lý khi văn bản quá dài (có thể cắt bỏ hoặc hiển thị dấu ...).

Các giá trị của TextOverflow:

* TextOverflow.clip – Cắt bỏ phần dư mà không có dấu hiệu gì.
* TextOverflow.fade – Làm mờ dần phần cuối của văn bản.
* TextOverflow.ellipsis – Hiển thị dấu ... khi văn bản bị cắt.

Vd:

Text(

'Flutter là một framework tuyệt vời để phát triển ứng dụng di động và web.',

maxLines: 1, // Giới hạn số dòng hiển thị

overflow: TextOverflow.ellipsis, // Hiển thị "..." khi văn bản quá dài

)

* maxLines: Giới hạn số dòng hiển thị.

Giá trị mặc định: null – Không giới hạn số dòng.

Vd:

Text(

'Dart là ngôn ngữ lập trình mạnh mẽ giúp phát triển ứng dụng với Flutter.',

maxLines: 2, // Chỉ hiển thị tối đa 2 dòng

)

* textScaler – Điều chỉnh tỷ lệ văn bản.

Các cách sử dụng textScaler:

* TextScaler.linear(2.0) – Tăng gấp đôi kích thước chữ.
* TextScaler.noScaling – Không thay đổi kích thước chữ.

Vd:

Text(

'Đây là văn bản có kích thước điều chỉnh!',

textScaler: TextScaler.linear(1.5), // Phóng to văn bản 1.5 lần

)

* semanticsLabel – Gán mô tả cho công cụ đọc màn hình

Vd:

Text(

'📌 Đây là nội dung quan trọng!',

semanticsLabel: 'Đây là nội dung quan trọng không chứa emoji', // Trợ năng

)

* TextDirection: Được sử dụng để xác định khi nào Text sẽ không vừa với không gian sẵn.

+ TextDirection.ltr – Hướng của văn bản sẽ từ trái sang phải

+ TextDirection.rtl – Hướng của văn bản sẽ từ phải sang trái

Vd:

Text(

‘Hello Work’,

textDirection: TextDirection.ltr,

)

* Overflow : Được sử dụng để xác định khi nào Text sẽ không vừa với không gian sẵn có. Khi hết không gian nó sẽ có “…” ở cuối đoạn Text.

Các giá trị của TextOverflow:

+ TextOverflow.clip - Cắt văn bản tại ranh giới mà không hiển thị dấu hiệu cắt.

+ TextOverflow.fade - Làm mờ dần văn bản tại ranh giới.

+ TextOverflow.ellipsis - Hiển thị dấu ... khi văn bản bị cắt.

+ TextOverflow.visible - Văn bản có thể tràn ra ngoài giới hạn của widget.

VD:

Text(

'Đây là một đoạn văn bản dài có thể bị cắt...',

overflow: TextOverflow.ellipsis, // Hiển thị dấu "..." khi bị cắt

)

* TextScaleFactor: được sử dụng để xác định tỷ lệ của Text được hiển thị bởi widget Text. Giả sử đã chỉ định hệ số tỷ lệ Text là 1,5, thì Text của chúng ta sẽ lớn hơn 50 phần trăm so với kích thước phông chữ được chỉ định.

Vd:

Text(

‘Hello Work’,

TextScaleFactor: 1.5,

)

* SoftWrap: cho phép xuống dòng nếu không đủ không gian của Text nếu nó trả về true. Còn nếu trả về false thì không cho phép.

VD:

Text(

'Đây là một đoạn văn bản rất dài và có thể cần xuống dòng nếu không đủ không gian hiển thị.',

softWrap: true, // Cho phép tự động xuống dòng

)

* MaxLines: được sử dụng để giới hạn số dòng của Text.

Vd:

Text(

‘Hello Work’,

maxlines: 3,// giới hạn số dòng tối đa của Text là 3

)

* TextWidthBasis: được sử dụng để xác định cách tính toán chiều rộng của văn bản, đặc biệt hữu ích khi làm việc với văn bản có khoảng trắng hoặc ký tự rộng.

VD:

Text(

'Hello Flutter!',

textWidthBasis: TextWidthBasis.longestLine, // Cơ sở tính toán dựa trên dòng dài nhất

)

* TextHeightBehavior: giúp kiểm soát cách chiều cao của văn bản được tính toán và áp dụng trong layout. Nó ảnh hưởng đến cách văn bản căn chỉnh theo chiều dọc trong một dòng hoặc trong nhiều dòng.
* Các thuộc tính của TextHightBehavior:

+ applyHeightToFirstAscent(bool) - Nếu true, chiều cao (height) sẽ ảnh hưởng đến phần trên của chữ (ascent).

+ applyHeightToLastDescent (bool) - Nếu true, chiều cao sẽ ảnh hưởng đến phần dưới của chữ (descent).

+ leadingDistribution (TextLeadingDistribution) - Kiểm soát cách phân bố khoảng cách giữa các dòng.

VD:

TextHeightBehavior({

bool applyHeightToFirstAscent = true,

bool applyHeightToLastDescent = true,

TextLeadingDistribution leadingDistribution = TextLeadingDistribution.proportional,

})

## RichText

* RichText sử dụng TextSpan để xác định các đoạn văn bản và phong cách riêng biệt cho từng đoạn.
* Các thành phần chính:
* text: Nội dung văn bản
* style: Kiểu chữ (màu sắc, font, độ đậm, nghiêng, v.v.)
* children: Danh sách các TextSpan con

VD:

RichText(

text: TextSpan(

text: 'Xin chào, ',

style: TextStyle(fontSize: 24, color: Colors.black),

children: <TextSpan>[

TextSpan(

text: 'Flutter',

style: TextStyle(fontWeight: FontWeight.bold, color: Colors.blue),

),

TextSpan(

text: ' là tuyệt vời!',

style: TextStyle(fontStyle: FontStyle.italic, color: Colors.green),

),

],

),

)

* lưu ý khi dùng RichText:

+ Khi cần hiển thị nhiều phong cách khác nhau trong một đoạn văn bản.

+ Khi cần định dạng văn bản một cách linh hoạt.

+ Khi cần hiển thị văn bản với nhiều hiệu ứng như **bài viết**, **trang tin tức**, hoặc **mô tả sản phẩm**.

## Padding and Margin

* Padding là khoảng cách bên trong giữa nội dung của widget và viền của chính nó.

Vd:

Padding(

padding: EdgeInsets.all(10.0),

child: RichText(

text: TextSpan(

text: 'Hello, ',

style: TextStyle(fontSize: 24, color: Colors.black),

children: <TextSpan>[

TextSpan(

text: 'Flutter!',

style: TextStyle(fontWeight: FontWeight.bold, color: Colors.blue),

),

],

),

),

)

* Trong Flutter, Margin được thiết lập thông qua EdgeInsets trong Container. Một số cách phổ biến để khai báo Margin bao gồm:
* EdgeInsets.all(value): Thiết lập khoảng cách đều ở tất cả các cạnh.
* EdgeInsets.symmetric(horizontal: x, vertical: y): Thiết lập khoảng cách theo chiều ngang và dọc.
* EdgeInsets.only(left: x, right: y, top: z, bottom: w): Thiết lập khoảng cách riêng cho từng cạnh.

VD:

Container(

margin: EdgeInsets.all(16.0),

child: Text(

'Xin chào, Flutter!',

style: TextStyle(fontSize: 24, color: Colors.black),

),

)

* Lưu ý khi dung Margin:
* Khi muốn tạo khoảng cách giữa các widget để cải thiện bố cục.
* Khi cần căn chỉnh một widget theo ý muốn trong giao diện người dùng.
* Khi cần kiểm soát không gian xung quanh một thành phần cụ thể

## TextButton, ElevatedButton và OutlineButton

* TextButton: là loại nút tối giản, chỉ hiển thị văn bản mà không có viền hoặc nền.

Lưu ý khi sử dụng:

* Khi cần một nút đơn giản, không làm nổi bật so với giao diện xung quanh.
* Khi muốn tạo liên kết hoặc hành động nhẹ nhàng mà không thu hút quá nhiều sự chú ý.

Vd:

TextButton(

onPressed: () {

print('TextButton được nhấn');

},

child: Text('Nhấn vào đây'),

)

* ElevatedButton: là loại nút có nền và hiệu ứng nâng, giúp tạo sự nổi bật rõ ràng trên giao diện.

Lưu ý khi sử dụng:

* Khi cần một nút quan trọng, thu hút sự chú ý của người dùng.
* Khi muốn tạo sự nổi bật cho một hành động chính trong giao diện.

Vd:

ElevatedButton(

onPressed: () {

print('ElevatedButton được nhấn');

},

child: Text('Nhấn vào đây'),

)

* OutlineButton: là nút có viền bao quanh, giúp làm nổi bật hơn so với TextButton, nhưng vẫn giữ phong cách nhẹ nhàng.

Lưu ý khi sử dụng:

* Khi cần một nút có độ nổi bật trung bình giữa TextButton và ElevatedButton.
* Khi muốn tạo một phong cách nút với viền nhưng không làm nổi bật quá mức.

VD:

OutlinedButton(

onPressed: () {

print('OutlinedButton được nhấn');

},

child: Text('Nhấn vào đây'),

)

|  |  |  |  |
| --- | --- | --- | --- |
| **Thuộc tính** | **TextButton** | **OutlinedButton** | **ElevatedButton** |
| Viền | Không có | Có viền | Không có |
| Nền | Không có | Không có | Có nền |
| Hiệu ứng nâng | Không có | Không có | Có |
| Độ nổi bật | Thấp | Trung bình | Cao |

## Container

* Container là một Widget đơn giản, giống như một hộp (box) bao quanh một Widget con (child). Cho phép tùy chỉnh giao diện và bố cục của Widget con thông qua các thuộc tính.
* Kích thước (width, height)
* Màu nền (color)
* Viền (border)
* Đệm (padding) và lề(margin)
* Căn chỉnh (alignment)
* Hình dàng (shape) và hiệu ứng (làm bóng – boxShadow)

VD:

Container(

width: 100.0, // Chiều rộng

height: 100.0, // Chiều cao

color: Colors.blue, // Màu nền

padding: EdgeInsets.all(16.0), // Đệm bên trong

margin: EdgeInsets.all(16.0), // Lề bên ngoài

alignment: Alignment.center, // Căn chỉnh widget con

child: Text('Hello, Flutter!'), // Widget con

)

## SizedBox

* SizedBox là một Widget dung để:
* Đặt khích thước cụ thể (chiều rông width và chiều cao height) cho Widget con (child).
* Tạo khoảng cách (spacer) giữa các Widget mà không cần nội dung.

VD:

SizedBox(

width: 100.0, // Chiều rộng

height: 50.0, // Chiều cao

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

)

## Row

* Row được sử dụng để sắp xếp các Widget theo chiều ngang.

VD:

Row(

mainAxisAlignment: ..., // canh chỉnh theo chiều ngang (main axis)

crossAxisAlignment: ..., // canh chỉnh theo chiều dọc (cross axis)

children: [

Widget1(),

Widget2(),

...

],

)

* **Các thuộc tính quan trọng của Row**
* mainAxisAlignment - Căn chỉnh theo chiều ngang.

VD:

Row (

MainAxisAlignment.start // Canh trái (mặc định)

MainAxisAlignment.center // Canh giữa

MainAxisAlignment.end // Canh phải

MainAxisAlignment.spaceBetween // Khoảng cách đều, không có khoảng ở đầu/cuối

MainAxisAlignment.spaceAround // Khoảng cách đều, có khoảng đầu/cuối

MainAxisAlignment.spaceEvenly // Khoảng cách đều nhau giữa tất cả các phần tử

Children: [],

)

* crossAxíAlignment – căn chỉnh theo chiều dọc

VD:

Row(

CrossAxisAlignment.start // Canh trên

CrossAxisAlignment.center // Canh giữa (theo chiều dọc)

CrossAxisAlignment.end // Canh dưới

Children: []

)

* mainAxisSize – Kích thước hàng theo chiều ngang.

VD:

Row(

MainAxisSize.max // Chiếm hết chiều ngang

MainAxisSize.min // Vừa đủ để chứa các widget con

Children: [],

)

## Column

* Column dung để sắp xếp các Widget theo chiều dọc.

VD:

Column(

mainAxisAlignment: ..., // canh chỉnh theo chiều dọc (main axis)

crossAxisAlignment: ..., // canh chỉnh theo chiều ngang (cross axis)

children: [

Widget1(),

Widget2(),

...

],

)

* **Các thuộc tính của Column**
* mainAxisAlignment - Căn chỉnh theo chiều ngang.

VD:

Column(

MainAxisAlignment.start // Canh trái (mặc định)

MainAxisAlignment.center // Canh giữa

MainAxisAlignment.end // Canh phải

MainAxisAlignment.spaceBetween // Khoảng cách đều, không có khoảng ở đầu/cuối

MainAxisAlignment.spaceAround // Khoảng cách đều, có khoảng đầu/cuối

MainAxisAlignment.spaceEvenly // Khoảng cách đều nhau giữa tất cả các phần tử

Children: [],

)

* crossAxíAlignment – căn chỉnh theo chiều dọc

VD:

Column(

CrossAxisAlignment.start // Canh trên

CrossAxisAlignment.center // Canh giữa (theo chiều dọc)

CrossAxisAlignment.end // Canh dưới

Children: []

)

* mainAxisSize – Kích thước hàng theo chiều ngang.

VD:

Column (

MainAxisSize.max // Chiếm hết chiều ngang

MainAxisSize.min // Vừa đủ để chứa các widget con

Children: [],

)

## Expanded

* Expended là một Widget thuộc nhóm layout widgets trong Flutter, được sử dụng để kiểm soát các widget con phân bổ không gian trong bố cục cha (Row, Column , hoặc Flex).
* Nó chỉ hoạt động trong các widget bố cục Flex (như Row, Column), vù chúng có khái niệm main axis sau khi các widget khác không sử dụng Expanded) đã được phân bổ kích thước cố định

VD:

Column(

children: [

Expanded(

child: Container(color: Colors.blue),

),

Expanded(

child: Container(color: Colors.red),

),

],

)

* **Cách hoạt động.**
* Expanded làm cho wedget con mở rộng để lấp đầy không gian trống trong Row hoặc Column.
* Nếu có nhiều Expanded trong cùng một bố cục, không gian trống sẽ được chia đều hoặc chia theo tỷ lệ dựa trên thuộc tính flex.
* **Thuộc tính chính.**
* child: Widget con mà Expanded sẽ mở rộng. Đây là widget sẽ lấp đầy không gian trống.
* flex: Một số nguyên (mặc định là 1) xác định tỷ lệ phân chia không gian trống giữa các Expanded trong cùng một bố cục.

Vd: nếu có hai Expanded với flex:1 và flex:2, thì widget thứ 2 sẽ gấp đôi không gian so với widget thứ nhất.

VD:

Row(

children: [

Expanded(

flex: 1,

child: Container(color: Colors.blue),

),

Expanded(

flex: 2,

child: Container(color: Colors.red),

),

],

)

## Stack

* Stack là một widget bố cục cho phép xếp chồng các widget con theo thứ tự (Z – index).
* Các widget con trong Stack có thể được đặt ở vị trí tương đối hoặc tuyệt đối trong không gian của Stack.
* Stack thường được sử dụng khi bạn cần các widget chồng lên nhau, ví dụ:

 Văn bản hiển thị trên hình ảnh.

 Các nút hoặc biểu tượng nổi trên giao diện.

 Hiệu ứng lớp phủ hoặc giao diện tùy chỉnh.

VD:

Stack(

alignment: Alignment.center,

children: [

Container(

width: 200,

height: 200,

color: Colors.blue,

),

Container(

width: 100,

height: 100,

color: Colors.red,

),

],

)

## AppBar

* Vị trí: AppBar thường nằm ở đầu của một Scaffold widget, đóng vai trò là thanh điều hướng chính của màn hình.
* Chức năng:
* Hiển thị tiêu đề (title) của ứng dụng hoặc màn hình.
* Cung cấp các nút hành động (action buttons) như tìm kiếm, thêm, menu ,…
* Hỗ trợ nút dẫn ngược ( leading icon/button), thường là nút quay lại hoặc menu hamburger.
* Tùy chỉnh giao diện như màu sắc, chiều cao hoặc đổ bóng
* Các thuộc tính cơ bản của AppBar:
* Title: Widget hiển thị tiêu đề chính thường là dạng Text()
* Leading: Widget hiển thị ở bên trái, thường là các nút (menu, back). Nếu không được chỉ định Flutter sẽ tự động thêm nút quay lại nếu có thể.
* Actions: Danh sách các Widget (thường là các IconButton) ở bên phải AppBar.
* backgroundColor: Màu nền của AppBar.
* Elevation: độ cao của đổ bóng(shadow).
* automaticallylmplyLeading: nếu true (mặc định), tự động thêm nút quay lại khi cần.
* flexibleSpace: Cho phép thêm nội dung tùy chỉnh phía sau AppBar.
* Bottom: Widget hiển thị dưới AppBar, thường dung cho TabBar hoặc thanh tìm kiếm.

VD:

AppBar(

title: Text('Tiêu đề'),

leading: IconButton(

icon: Icon(Icons.menu),

onPressed: () {},

),

actions: [

IconButton(

icon: Icon(Icons.search),

onPressed: () {},

),

IconButton(

icon: Icon(Icons.more\_vert),

onPressed: () {},

),

],

backgroundColor: Colors.blue,

elevation: 4.0,

)

## InkWell

* Chức năng:
* Cung cấp hiệu ứng gợn sóng( ripple effect) khi người dung chạm, giữ hoặc kéo.
* Cho phép xử lý các sự kiện cảm ứng như nhấn (tap), nhân đúp (double tap), nhấn lâu (long press),…
* Thường được sử dụng để làm cho một Widget trở nên “Nhấn được” với phản hồi trực quan.
* Vị trí: InkWell thường được bao bọc một Material Widget đó có khả năng tương tác.
* Yêu cầu: InkWell cần được một Material Widget để hiện thị hiệu ứng gợn sóng đúng cách, vì hiệu ứng này dựa trên Material Design.
* Các thuộc tính của InkWell:
* onTap: Hàm gọi khi người dung nhấn vào InkWell
* onDoubleTap: Hàm gọi khi người dung ấn đúp.
* onLongPress: Hàm gọi khi người dung ấn và giữ lâu.
* Child: Widget con mà InkWell bao bọc, xác định khu vực có thể ấn.
* splashColor: Màu nền của hiệu ứng gợn sóng.
* highlightColor: Màu nền khi khu vực được nhấn hoặc giữ.
* Radius: Bán kính tối đa của hiệu ứng gợn sóng.
* customBorder: Tùy chỉnh dạng của vùng hiệu ứng(Vd: hình tròn, hình chữ nhật,..).
* onHover: Hàm gọi khi con trỏ chuột di chuyển qua vùng InkWell.

VD:

InkWell(

onTap: () {

print('Đã nhấn vào nút!');

},

onLongPress: () {

print('Đã nhấn lâu!');

},

splashColor: Colors.blueAccent,

highlightColor: Colors.blue.withOpacity(0.2),

borderRadius: BorderRadius.circular(8.0),

child: Container(

padding: EdgeInsets.all(16.0),

decoration: BoxDecoration(

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

borderRadius: BorderRadius.circular(8.0),

),

child: Text(

'Nhấn vào tôi',

style: TextStyle(fontSize: 18),

),

),

## GestureDetetor

* Chức năng:
* Phát hiện các cử chỉ của người dung trên một khu vực cụ thể của giao diện.
* Cho phép xử lý nhiều loại sự kiện như nhấn, kéo, phóng to/ thu nhỏ, hoặc di chuyển.
* Không cung cấp hiệu ứng trực quan (như gợn sóng) mặc định, nhưng có thể kết hợp với các Widget khác để tạo hiệu ứng.
* Vị trí: GestureDetector bao bọc một Widget con (child) để làm cho Widget đó có khả năng phản hồi các tứ chi.
* Tính linh hoạt: GestureDetector hỗ trợ nhiều loại cử chỉ hơn so với các Widget khác như InkWell, phù hợp cho các tương tác phức tạp.
* Các thuộc tính của GestureDetector:
* onTap: Gọi khi người dùng nhấn và thả ngón tay.
* onDoubleTap: Gọi khi người dùng nhấn đúp.
* onLongPress: Gọi khi người dùng nhấn và giữ lâu.
* onPanStart/onPanUpdate/onPanEnd: Xử lý cử chỉ kéo (drag) với thông tin về vị trí và tốc độ.
* onScaleStart/onScaleUpdate/onScaleEnd: Xử lý cử chỉ phóng to/thu nhỏ (pinch/zoom).
* onHorizontalDragStart/Update/End: Phát hiện kéo ngang.
* onVerticalDragStart/Update/End: Phát hiện kéo dọc.
* child: Widget con mà GestureDetector bao bọc, xác định khu vực tương tác.
* behavior: Xác định cách GestureDetector xử lý các sự kiện (ví dụ: HitTestBehavior.opaque để mở rộng vùng nhận diện)

VD:

GestureDetector(

onTap: () {

print('Đã nhấn!');

},

onDoubleTap: () {

print('Đã nhấn đúp!');

},

onLongPress: () {

print('Đã nhấn lâu!');

},

onPanUpdate: (details) {

print('Đang kéo: ${details.localPosition}');

},

child: Container(

width: 200,

height: 200,

color: Colors.teal,

child: Center(

child: Text(

'Tương tác với tôi',

style: TextStyle(color: Colors.white, fontSize: 18),

),

),

),

),

## TextField

* Chức năng:
* Cung cấp một khu vực nhập văn bản có thể chỉnh sửa
* Hỗ trợ các tính năng như gợi ý, kiểm tra đầu vào, và xử lý sự kiện khi nhập.
* Tùy chỉnh giao diện như viền, nhãn, biểu tượng và màu sắc.
* Vị trí: TextField thường được gọi trong Scaffold hoặc các Widget bố cục như Column, Row hoặc Form.
* Tương tác: TextField tự động hiển thị bàn phím khi được nhấn và hỗ trợ các kiểu bàn phím khác nhau ( văn bản, số, email,…).
* Các thuộc tính chính của TextField:
* Controller: Một TextEditingController để kiểm soát nội dung văn bản, lấy giá trị hoặc đặt giá trị ban đầu.
* Decoration: Tùy chỉnh giao diện của TextField thông qua InputDecoration, bao gồm:
* lableText: Nhãn hiển thị.
* hintText: Văn bản gợi ý khi TextField chống.
* border: Kiểu viền (VD: OutlineInputBorder, UnderlineInputBorder).
* prefixIcon/suffixIcon: Biểu tượng Icon ở đầu hoặc ở cuối TextField.
* onChanged: Hàm gọi mỗi khi nội dung TextField thay đổi.
* onSumitted: Hàm gọi khi người dung nhấn nút “Done” hoặc “Enter” trên bàn phím.
* keyboardType: Xác định loại bàn phím ze (VD:TextInputType.number,TextInputType.emailAddress).
* obscureText: Nếu true, ẩn văn bản( dung cho mật khẩu).
* maxLines: Số dòng tối đa (mặc định là 1, đặt null để hỗ trợ nhiều dòng).
* maxLength: giới hạn số ký tự tối đa.
* Enabled: Nếu false, TextField không thể chỉnh sửa.
* inputFormatters: Danh sách các bộ lọc để định dạng hoặc giới hạn đầu vào(VD: chỉ cho phép số).
* VD:

Padding(

padding: EdgeInsets.all(16.0),

child: Column(

children: [

TextField(

controller: \_controller,

decoration: InputDecoration(

labelText: 'Nhập tên của bạn',

hintText: 'Ví dụ: Nguyễn Văn A',

border: OutlineInputBorder(),

prefixIcon: Icon(Icons.person),

),

keyboardType: TextInputType.name,

onChanged: (value) {

print('Tên: $value');

},

onSubmitted: (value) {

print('Đã gửi: $value');

},

),

SizedBox(height: 20),

TextField(

decoration: InputDecoration(

labelText: 'Mật khẩu',

border: OutlineInputBorder(),

suffixIcon: Icon(Icons.visibility),

),

obscureText: true, // Ẩn văn bản (mật khẩu)

),

SizedBox(height: 20),

ElevatedButton(

onPressed: () {

print('Tên đã nhập: ${\_controller.text}');

},

child: Text('Lấy dữ liệu'),

),

],

),

),

## Form

* Khái niệm:
* Form là một container widget bao bọc các widget nhập liệu (như TextFormField).
* Nó yêu cầu một key (thường là GlobalKey<FormState>) để quản lý trạng thái của form, bao gồm việc xác thực và lưu dữ liệu.
* Các thành phần chính
* TextFormFiled: Widget phổ biến nhất trong Form, dung để nhập văn bản. Nó hỗ trợ xác thực qua thuộc tính validator.
* Validator: Hàm kiểm tra tính hợp lệ của dữ liệu nhập. Nếu dữ liệu không hợp lệ, nó trả về một chuỗi lỗi để hiển thị cho người dùng.
* onSaved: Hàm được gọi để lưu giá trị của trường khi gọi FormState.save().

VD:

class LoginForm extends StatelessWidget {

final \_formKey = GlobalKey<FormState>();

@override

Widget build(BuildContext context) {

return Form(

key: \_formKey,

child: Column(

children: [

TextFormField(

decoration: InputDecoration(labelText: 'Email'),

validator: (value) {

if (value == null || !value.contains('@')) {

return 'Email không hợp lệ';

}

return null;

},

),

TextFormField(

decoration: InputDecoration(labelText: 'Mật khẩu'),

obscureText: true,

validator: (value) {

if (value == null || value.length < 6) {

return 'Mật khẩu phải dài hơn 6 ký tự';

}

return null;

},

),

ElevatedButton(

onPressed: () {

if (\_formKey.currentState!.validate()) {

ScaffoldMessenger.of(context).showSnackBar(

SnackBar(content: Text('Đăng nhập thành công')),

);

}

},

child: Text('Đăng nhập'),

),

],

),

);

}

}

## TextFormField

* Chức năng:
* Cho phép người dung nhập và chỉnh sử văn bản.
* Tích hợp các tính năng như xác thực dữ liệu, hiển thị thông báo lỗi, và lưu giá trị nhập liệu.
* Thường được sử dụng trong Form để thu thập thông tin như email, mật khẩu, v.v.
* Các thuộc tính quang trọng:
* Decoration: Tùy chỉnh giao diện của trường nhập liệu, bao gồm:
* labelText: nhãn hiển thị
* hintText: Gợi ý văn bản.
* icon,prefixIcon, suffixIcon: Thêm biểu tượng.
* boder: Tùy chỉnh viền (vd: OutlineInputBoder).
* Validator: hàm kiểm tra tính hợp lệ của giữ liệu. Trả về một chuỗi nếu dữ liệu không hợp lệ, hoặc null nếu hợp lệ.
* onSaved: Hàm được gọi khi FormState.save() chạy, dung để lưu giá trị nhập.
* controller: một TextEditingController để theo dõi và điều kiển nội dung văn bản.
* keyboardType: xác định loại bán phím (vd: TextInputType.emailAddress,TextInputType.number).
* obscureText: ẩn văn bản khi nhập (dung cho mật khẩu).
* onChanged: Hàm gọi mỗi khi nội dung trường thay đổi.

VD:

TextFormField(

decoration: InputDecoration(

labelText: 'Mật khẩu',

border: OutlineInputBorder(),

),

obscureText: true,

validator: (value) {

if (value == null || value.length < 6) {

return 'Mật khẩu phải dài hơn 6 ký tự';

}

return null;

},

onSaved: (value) {

// Lưu mật khẩu vào biến hoặc gửi đến server

},

)

## DropDownList

* Chức năng:
* Hiển thị một dánh sách thả xuống (dropdown) chứa các mục mà người dùng có thể chọn.
* Khi người dung chọn một mục, giá trị của DropdownButton sẽ được cập nhật và có thể được xử lý thông qua hàm onChanged.
* Các thuộc tính quan trọng:
* Value: giá trị hiện tại của DropDownButton (phải nằm trong danh sách items).
* Items: Danh sách các DropdownMenuItem đại diện cho các tùy chọn. Mỗi DropdownMenuItem cần có value(giá trị thực) và child (nội dung hiển thị).
* onChanged: Hàm được gọi khi người dung chọn một giá trị mới. Thường dùng để cập nhật trạng thái.
* Hint: Widget hiển thị khi chưa có giá trị nào được chọn.
* dropdownColor: màu nền của danh sách thả xuống.
* style: Tùy chỉnh kiểu chữ của các mục.
* isExpanded: nếu true, DropdownButton sẽ chiếm toàn bộ chiều rộng khả dụng.

VD:

class MyDropdown extends StatefulWidget {

@override

\_MyDropdownState createState() => \_MyDropdownState();

}

class \_MyDropdownState extends State<MyDropdown> {

String? selectedCategory;

@override

Widget build(BuildContext context) {

return DropdownButton<String>(

value: selectedCategory,

isExpanded: true,

hint: Text('Chọn danh mục'),

items: <String>['Điện tử', 'Thời trang', 'Gia dụng']

.map((String value) {

return DropdownMenuItem<String>(

value: value,

child: Text(value),

);

}).toList(),

onChanged: (String? newValue) {

setState(() {

selectedCategory = newValue;

});

},

);

}

}

## RadioListTitle

* Chức năng:
* RadioListTitle là một Widget trong flutter, kết hợp một radio button với tiêu đề, phụ đề, và các thành phần giao diện khác để tạo ra một mục chọn trong danh sách các lựa chọn.
* Các thuộc tính quan trọng:
* Value: giá trị đại diện cho RadioListTitle, dung để xác định lựa chọn.
* groupValue: giá trị hiện tại của nhóm radio
* onChanged: Hàm callback được gọi khi người dung chọn RadioListTitle, cập nhật groupValue.
* Title: Tiêu đề chính thức của mục, thường là một widget Text.
* Secondary: Widget hiển thị bên cạnh (vd: biểu tượng).
* activeColor: Màu của radio button khi được chọn( mặc định là màu xanh).
* controlAffinity: Quy định vị trí của radio button (bên phải hoặc trái tiêu đề).

Vd:

return MaterialApp(

home: Scaffold(

appBar: AppBar(title: Text('Ví dụ RadioListTile')),

body: Column(

children: [

RadioListTile(

title: Text('Lựa chọn 1'),

value: 'option1',

groupValue: \_selectedOption,

onChanged: (value) {

setState(() {

\_selectedOption = value.toString();

});

},

),

RadioListTile(

title: Text('Lựa chọn 2'),

value: 'option2',

groupValue: \_selectedOption,

onChanged: (value) {

setState(() {

\_selectedOption = value.toString();

});

},

),

],

),

),

);

}

}

## CheckBox

* Chức năng:
* CheckBox: Là một Widget trong flutter dung để tạo một ô tích cho phép người dung chọn hoặc không.
* CheckBox cho phép chọn nhiều tùy chọn độc lập.
* Các thuộc tính quan trọng:
* Value(bắt buộc): Gía trị Boolean (true hoặc false) xác định trạng thái của checkbox (chọn hoặc không).
* onChanged (bắt buộc): Hàm callback được gọi khi người dung thay đổi trạng thái của checkbox, nhận giá trị boolean mới.
* activeColor: Màu của checkbox khi được chọn
* checkColor: màu của dấu kiểm trong checkbox.
* Trisate: nếu true, checkbox có thể có trạng thái thứ ba (null) ngoài true và false.
* Side: Tùy chỉnh viền của checkbox(vd: màu sắc, độ dày).

VD:

return MaterialApp(

home: Scaffold(

appBar: AppBar(title: Text('Ví dụ Checkbox')),

body: Column(

children: [

CheckboxListTile(

title: Text('Tùy chọn 1'),

value: \_isChecked1,

onChanged: (bool? value) {

setState(() {

\_isChecked1 = value ?? false;

});

},

activeColor: Colors.blue,

checkColor: Colors.white,

),

CheckboxListTile(

title: Text('Tùy chọn 2'),

value: \_isChecked2,

onChanged: (bool? value) {

setState(() {

\_isChecked2 = value ?? false;

});

},

activeColor: Colors.blue,

checkColor: Colors.white,

),

],

),

),

);

}

}

## DatePicker

* Chức năng:
* DatePicker là một widget trong flutter cho phép người dung chọn một ngày từ giao diện lịch (calendar) hoặc nhập trực tiếp.
* Trong Flutter, DatePicker thường được hiển thị thông qua hàm showDatePicker, tạo một hộp thoại chọn ngày.
* Các thuộc tính quan trọng:
* Context( bắt buộc): BuildContext để hiển thị hộp thoại.
* initalDate: Ngày mặc định được chọn khi DatePicker mở (thường là ngày hiện tại).
* DateTime.now(): ngày hôm nay.
* firstDate: Ngày sớm nhất mà người dung có thể chọn
* lastDate: Ngày muộn nhất mà người dung có thê chọn.
* currentDate: Ngày hiện tại dùng để làm nổi bật lên lịch.
* initialDatePickerMode: Chế độ hiển thị ban đầu (day hoặc year).
* onDateChanged; Hàm callback khi người dùng thay đổi ngày.
* onConfirm: Hàm callback khi người dung xác nhận ngày.
* Locate: định dạng ngôn ngữ và khu vực (VD: Locale(‘vi’,’VN’) cho tiếng Việt).
* Builder: Tùy chỉnh giao diện của DatePicker.

VD:

class MyApp extends StatefulWidget {

@override

\_MyAppState createState() => \_MyAppState();

}

class \_MyAppState extends State<MyApp> {

DateTime? \_selectedDate;

Future<void> \_selectDate(BuildContext context) async {

final DateTime? picked = await showDatePicker(

context: context,

initialDate: DateTime.now(),

firstDate: DateTime(2000),

lastDate: DateTime(2030),

locale: Locale('vi', 'VN'),

);

if (picked != null && picked != \_selectedDate) {

setState(() {

\_selectedDate = picked;

});

}

}

@override

Widget build(BuildContext context) {

return MaterialApp(

home: Scaffold(

appBar: AppBar(title: Text('Ví dụ DatePicker')),

body: Center(

child: Column(

mainAxisAlignment: MainAxisAlignment.center,

children: [

Text(

\_selectedDate == null

? 'Chưa chọn ngày'

: 'Ngày đã chọn: ${\_selectedDate!.toString().substring(0, 10)}',

),

ElevatedButton(

onPressed: () => \_selectDate(context),

child: Text('Chọn ngày'),

),

],

),

),

),

);

}

}

# Project

## UI đăng nhập