

GIỚI THIỆU



Huỳnh Tuấn Anh



1

NỘI DUNG

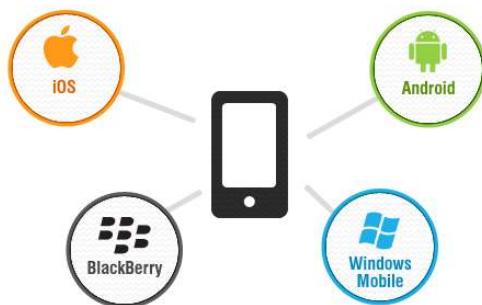
1. Các nền tảng di động
2. Lập trình đa nền tảng trên Flutter
3. Cài đặt môi trường phát triển ứng dụng trên Flutter

Huỳnh Tuấn Anh

2

Các platform của smart phone

- Android
- iOS
- Blackberry
- Windows Phone



Huỳnh Tuấn Anh

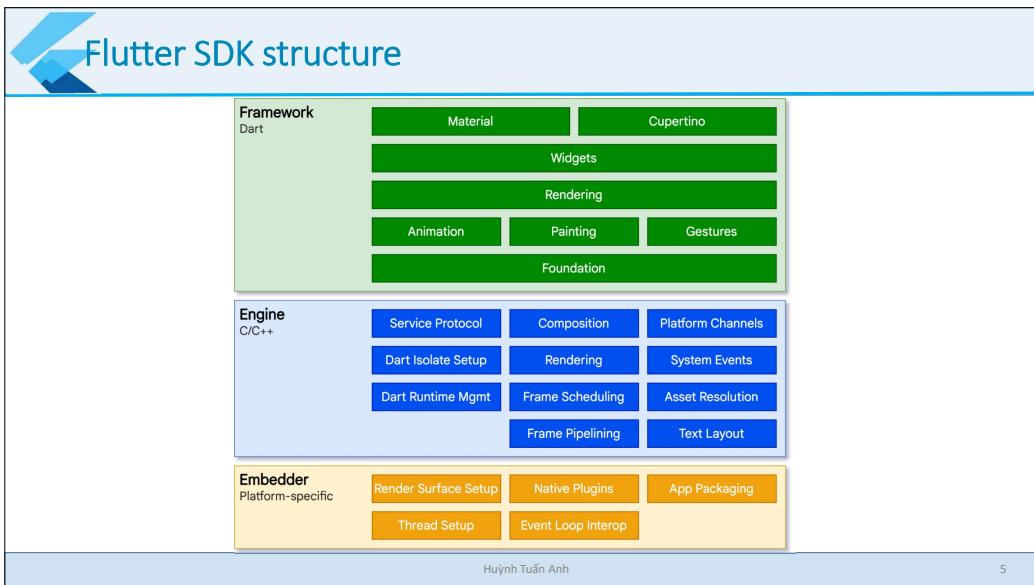
3

Native app và Hybrid app

- Native App
- Hybrid App
- Các SDK dùng để phát triển ứng dụng Native.
- So sánh Flutter và React Native.

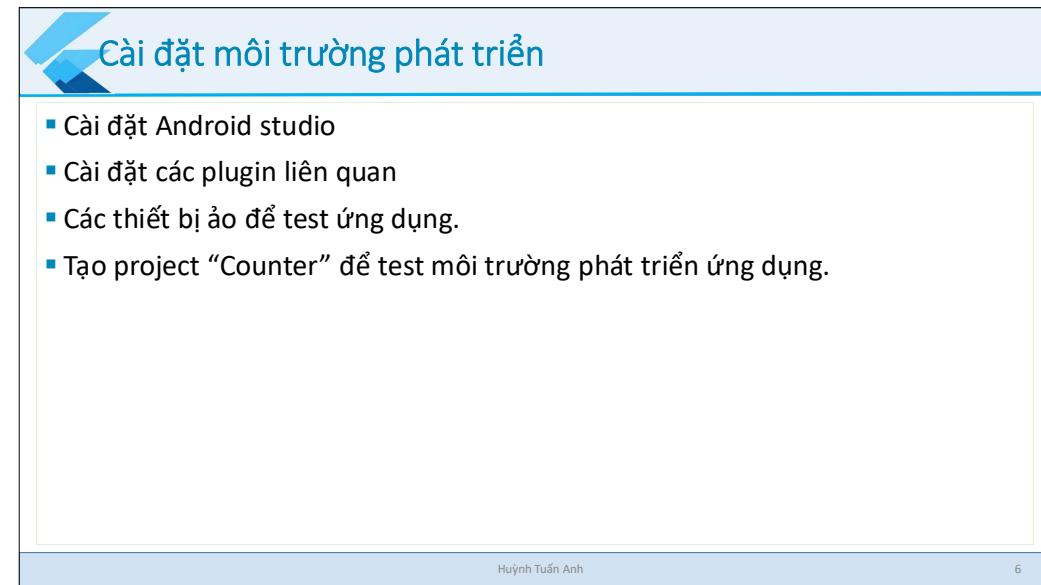
Huỳnh Tuấn Anh

4



Huỳnh Tuấn Anh

5



Huỳnh Tuấn Anh

6

CƠ BẢN VỀ DART



Huỳnh Tuấn Anh
Khoa CNTT - ĐHNT



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

1

Nội dung

- Chương trình Hello World
- Các khái niệm cơ bản trong ngôn ngữ Dart
- Luồng điều khiển
- Hàm
- Lập trình hướng đối tượng trong Dart
 - Class
 - Constructor
 - Inheritance
 - Factories và named constructor
 - Enum
- Async

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

2

Chương trình “Hello World”

```
void main() {  
    helloDart("Flutter");  
}  
  
void helloDart(String name) {  
    print("Hello $name");  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

3

Các thư viện trong Dart

- Dart SDK bao gồm nhiều thư viện. Thư viện được nạp mặc định vào chương trình là ‘dart : core’. Các thư viện khác khi sử dụng phải khai báo thông qua từ khóa *import*.
 - VD: import ‘dart : io’
- Một số thư viện thông dụng: *dart:html*, *dart:async*, *dart:math*...

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

4

Các khái niệm cơ bản

- Dart là một ngôn ngữ hướng đối tượng và hỗ trợ đơn thừa kế.
- Trong Dart, mọi thứ đều là **object**, mọi **object** đều là một thể hiện của một lớp. Mọi đối tượng đều kế thừa từ lớp **Object**. Các con số cũng là đối tượng chứ không phải thuộc các kiểu dữ liệu nguyên thủy.
- Dart là một ngôn ngữ định kiểu. VD: Không thể trả về một con số từ một hàm được khai báo kiểu trả về là **String**.
- Dart hỗ trợ top-level functions và top-level variables (hàm và biến được khai báo bên ngoài lớp), chúng được xem như là thành viên của thư viện

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

5

Kiểu trong Dart

- Chỉ định kiểu khi khai báo biến:

- VD: String name;
- int age = 30;

- Kiểu dữ liệu phức hợp:

- Khi sử dụng các cấu trúc dữ liệu như List hay Map, sử dụng <> để định nghĩa kiểu dữ liệu cho mỗi thành viên.

- VD:



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

6

Kiểu trả về của hàm

- Khi định nghĩa hàm, cần phải chỉ ra kiểu trả về của hàm
- VD:

```
int addNum(int x, int y)
{
    return x + y;
}

int addNum(int x, int y) => x+y;
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

7

Các kiểu dữ liệu cơ bản

- package dart:core
 - <https://api.dart.dev/stable/2.10.3/dart-core/dart-core-library.html>
- Kiểu số và kiểu Booleans

```
int meaningOfLife = 42;
double valueOfPi  = 3.141592;
bool visible     = true;
```

- Kiểu chuỗi
- Collections
- Date và Time
- URI

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

8

Kiểu dynamic

- Khi sử dụng từ khóa **dynamic** để khai báo biến, trình biên dịch sẽ chấp nhận mọi kiểu dữ liệu cho biến này.

- VD 1:

- dynamic number = "11"; // kiểu String
 - number = 11; // kiểu int
 - var number2 = "11"; // Kiểu String
 - number2 = 11; // Lỗi

- Kiểu dữ liệu **dynamic** tỏ ra hữu dụng khi sử dụng với dữ liệu Map khi mỗi thành viên của Map là một cặp key-value trong dữ liệu JSON.

- VD: Map<String, dynamic> json;

Comment

```
// Inline comments
/*
Blocks of comments. It's not convention to use
block comments in Dart.
*/
/// Documentation
///
/// This is what you should use to document
your classes.
```

Biến và phép gán

- Khi khai báo biến nhưng không gán giá trị, biến sẽ nhận giá trị **null** (**null** cũng là một object).
 - VD: String name; // name nhận giá trị null.
 - Tất cả các biến đều có thể được gán giá trị null

- Từ khóa **final**, **const**: Dùng để khai báo một biến có giá trị không thể thay đổi khi chạy chương trình. Các biến khi khai báo với final hay const đều phải khởi tạo giá trị lúc khai báo.

- VD1: final String name = "Smith";
 - VD2: const String name = "Smith";

Từ khóa final, const

- Khác nhau giữa final và const:

- biến được khai báo const phải được xác định giá trị khi compile.
 - biến được khai báo với từ khóa final chỉ được gán giá trị một lần
 - VD1: const String name = 'Nora \$lastname'; // Lỗi???
 - VD2: final String name = 'Nora \$lastname'; // Sử dụng được???

Các toán tử: Toán tử số học

OPERATOR	DESCRIPTION	SAMPLE CODE
+	Add	$7 + 3 = 10$
-	Subtract	$7 - 3 = 4$
*	Multiply	$7 * 3 = 21$
/	Divide	$7 / 3 = 2.33$

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

13

Toán tử so sánh

OPERATOR	DESCRIPTION	SAMPLE CODE
==	Equal	$7 == 3 = \text{false}$
!=	Not equal	$7 != 3 = \text{true}$
>	Greater than	$7 > 3 = \text{true}$
<	Less than	$7 < 3 = \text{false}$
>=	Greater than or equal to	$7 >= 3 = \text{true}$ $4 >= 4 = \text{true}$
<=	Less than or equal to	$7 <= 3 = \text{false}$ $4 <= 4 = \text{true}$

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

14

Toán tử kiểm tra kiểu

OPERATOR	DESCRIPTION	SAMPLE CODE
as	Typecast like import library prefixes.	<code>import 'travelpoints.dart' as travel;</code>
is	If the object contains the specified type, it evaluates to <code>true</code> .	<code>if (points is Places) = true</code>
is!	If the object contains the specified type, it evaluates to <code>false</code> (not usually used).	<code>if (points is! Places) = false</code>

as: còn được dùng để chuyển một object về đúng kiểu của nó.

VD: Person p = new Student("An", 20);
Student s = p as Student;

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

15

Toán tử logic

OPERATOR	DESCRIPTION	SAMPLE CODE
!	! is a logical 'not'. Returns the opposite value of the variable/expression.	<code>if (!(7 > 3)) = false</code>
&&	&& is a logical 'and'. Returns <code>true</code> if the values of the variable/expression are all <code>true</code> .	<code>if ((7 > 3) && (3 < 7)) = true</code> <code>if ((7 > 3) && (3 > 7)) = false</code>
	is a logical 'or'. Returns <code>true</code> if at least one value of the variable/expression is <code>true</code> .	<code>if ((7 > 3) (3 > 7)) = true</code> <code>if ((7 < 3) (3 > 7)) = false</code>

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

16

Toán tử điều kiện

OPERATOR	DESCRIPTION	SAMPLE CODE
condition ? value1 : value2	If the condition evaluates to true, it returns value1. If the condition evaluates to false, it returns value2. The value can also be obtained by calling methods.	(7 > 3) ? true : false = true (7 < 3) ? true : false = false

Cascade notation (..)

OPERATOR	DESCRIPTION	SAMPLE CODE
..	The cascade notation is represented by double dots (..) and allows you to make a sequence of operations on the same object.	Matrix4.identity() ..scale(1.0, 1.0) ..translate(30, 30);

Các toán tử

■ ~/: toán tử chia số nguyên.

■ as: từ khóa dùng để chuyển một object về đúng kiểu của nó.

- VD: Person p = new Student("An", 20);
- Student s = p as Student;

■ is và is!: kiểm tra kiểu của một đối tượng.

- VD: p is Person; p is! Person

■ ?: Kiểm tra null trước khi truy cập một thuộc tính của đối tượng:

```
this.userAge = user?.age;    ↪ if (user != null) {  
                           this.userAge = user.age;  
                         }
```

Nếu user = null thì userAge không được gán giá trị

Các toán tử

Toán tử ??:

- x ??= 5;
- Nếu x có giá trị null, gán 5 cho x
- Nếu x khác null, giữ nguyên giá trị của x

Luồng điều khiển

■ if, else

■ switch, case

■ Vòng lặp

- Standard for
- for-in
- forEach
- while
- do while

■ break và continue

Vòng lặp

```
for:   for(var i=0; i<10; i++) {  
    print(i);  
}  
  
for-in:  
List<String> pets = ["Tom", "Jerry", "Poo"];  
for(var pet in pets) {  
    print(pet);  
}  
  
forEach:  
List<String> pets = ["Tom", "Jerry", "Poo"];  
pets.forEach((pet) => print(pet));
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

21

functions

Cấu trúc của hàm trong Dart:

```
String makeGreeting(String name) { ← Function signature  
  return 'Hello, $name'; ← Return type  
}
```

Cú pháp ngắn gọn khi thân hàm chỉ một dòng:

```
String makeGreeting(String name) => 'Hello, $name';
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

23

Hàm - Function

- Cấu trúc của function
- Parameters
- Named parameters: Tùy chọn, không bắt buộc phải nhập khi sử dụng.
- Positional optional parameters: Tùy chọn không bắt buộc phải nhập khi sử dụng.
- Default parameter values: Định nghĩa giá trị mặc định cho tham số thông qua toán tử = khi định nghĩa hàm.
- Hàm ẩn danh (Anonymous functions)

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

22

Parameter

- Positional parameters

```
void debugger(String message, int lineNumber) {  
  // ...  
}  
  
debugger('A bug!', 55);
```

- Named Parameters

Định nghĩa hàm:

```
void debugger({String message, int lineNumber}) {  
  Gọi hàm:  
  debugger(message: 'A bug!', lineNumber: 44);
```

Named parameters



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

24

Parameters

- Positional optional parameters:

```
int addSomeNums(int x, int y, [int z]) {  
    int sum = x + y;  
    if (z != null) {  
        sum += z;  
    }  
    return sum;  
}
```

The third parameter is optional, so you don't have to pass in anything.

```
addSomeNums(5, 4) ←  
addSomeNums(5, 4, 3) ←
```

optional
parameter

You can pass in a third argument, since you've defined an optional parameter.

Anonymous functions

- Hầu hết các hàm đều có tên để có thể gọi để sử dụng sau này.
- Hàm ẩn danh không có tên, đôi khi được gọi là *lambda* or *closure*
- Có thể gán một hàm ẩn danh cho một biến. Do đó, cũng có thể thêm vào hoặc loại bỏ một hàm ẩn danh ra khỏi một tập hợp.
- Các định nghĩa hàm ẩn danh cũng giống như hàm có tên thông thường, tuy nhiên ta bỏ qua khai báo phần tên của hàm

```
([[Type] param1[, ...]]) {  
    codeBlock;  
};
```

Anonymous functions

- Ví dụ sau đây định nghĩa một hàm ẩn danh với một tham số không định kiểu, item. Hàm này chỉ đơn giản là nhận mỗi giá trị trong danh sách và in giá trị đó trên màn hình.

- Kiểu dữ liệu của tham số sẽ được suy ra từ kiểu dữ liệu của mỗi phần tử trong danh sách

```
void main() {  
    var list = ['apples', 'bananas', 'oranges'];  
    list.forEach((item) {  
        print('${list.indexOf(item)}: $item');  
    });  
}
```

Cú pháp rút gọn của hàm

- Được sử dụng khi hàm chỉ có một câu lệnh. Cú pháp rút gọn thường được dùng khi viết các hàm ẩn danh.
- Ví dụ:

```
int binhPhuong (int x) => x*x;
```

```
void main() {  
    var list = ['apples', 'bananas', 'oranges'];  
    list.forEach((item) =>  
        print('${list.indexOf(item)}: $item');  
    )
```

Lập trình hướng đối tượng trong Dart

- class
- constructor
- inheritance
- factories và named constructor
- enum

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

29

class

```
class Cat {  
    String name;  
    String color;  
}  
  
Cat nora = new Cat();  
nora.name = 'Nora';  
nora.color = 'Orange';  
  
Cat ruby = Cat();  
nora.name = 'Ruby';  
nora.color = 'Grey';
```

- ✓ Nếu không viết constructor cho class thì constructor mặc định không có tham số sẽ được tạo ra.
- ✓ Từ khóa `new` để tạo một thể hiện của class là tùy chọn

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

30

constructor

```
class Animal {  
    String name;  ← Declares properties of this  
    String type; class (they are null to start)  
  
    Animal(String name, String type) {  ← Default constructor  
        this.name = name;  
        this.type = type;  
    }  ← Passes in arguments to the constructor  
  
    class Animal {  
        String name, type;  
  
        Animal(this.name, this.type);  ← Automatically assigns arguments  
    }  ← to properties with the same name
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

31

factory và named constructor

- Named constructor: Cho phép tạo ra các constructor với các tên có ý nghĩa.

```
class Point {  
    num x, y;  
  
    Point(this.x, this.y);  
  
    // Named constructor  
    Point.origin() {  
        x = 0;  
        y = 0;  
    }  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

32

factory và named constructor

- factory: Được sử dụng khi muốn tạo constructor không chỉ để tạo ra một instance mới của class mà có thể tạo một instance từ cache, hoặc một subclass của nó.

```
class Customer{  
    String name, location;  
    int age;  
    Customer(this.name, this.age, this.location);  
    static final Customer origin = Customer("", 30, "");  
    factory Customer.create() => origin;  
    @override  
    String toString() {  
        return 'Customer{name: $name, location: $location, age: '$age}';  
    }  
}  
  
void main()  
{  
    Customer customer = Customer.create();  
    print(customer);  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

33

Inheritance: Single inheritance

```
class Spacecraft {  
    String name;  
    DateTime launchDate;  
    // Constructor, with syntactic sugar for assignment to members.  
    Spacecraft(this.name, this.launchDate) {  
        // Initialization code goes here.  
    }  
    // Named constructor that forwards to the default one.  
    Spacecraft.unlaunched(String name) : this(name, null);  
  
    int get launchYear =>  
        launchDate?.year; // read-only non-final property  
    void describe() {}...  
}  
  
class Orbiter extends Spacecraft {  
    double altitude;  
    Orbiter(String name, DateTime launchDate, this.altitude)  
        : super(name, launchDate);  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

34

Mixins – Thêm các feature vào một lớp

- Mixin là một cách sử dụng lại mã lệnh trong nhiều phân cấp lớp.
- Hai cách thực hiện mixin
 - Định nghĩa class mà không có constructor; hoặc
 - Thay thế từ khóa `class` bằng từ khóa `Mixin`
- Thành phần mixin được bổ sung vào một lớp bằng từ khóa `with`
- Ví dụ:

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

35

Mixin, Ví dụ:

```
 mixin Piloted { // hoặc class  
    int astronauts = 1;  
    void describeCrew() {  
        print('Number of astronauts: $astronauts');  
    }  
}  
  
class PilotedCraft extends Spacecraft with Piloted {  
    PilotedCraft(String name, DateTime launchDate) : super(name, launchDate);  
    // ...  
}
```

Lớp PilotedCraft bây giờ đã được bổ sung thêm trường `astronauts` và phương thức `describeCrew()`

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

36

Mixin – từ khóa on

- Hạn chế các kiểu (type) có thể sử dụng mixin,
- Muốn thành phần mixin có thể sử dụng một số các phương thức, thuộc tính mà nó không định nghĩa.
- Ví dụ:

```
class Person{  
    String firstName, lastName;  
    int age;  
    Person(this.firstName, this.lastName, this.age);  
  
    String getInfo(){  
        return "$firstName $lastName";  
    }  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

mixin - on

- Chỉ có các lớp con của Person mới có thể sử dụng mixin PersonWithAge

```
mixin PersonWithAge on Person{  
    String getFullInfo(){  
        return "Tên: ${getInfo()} \nTuổi: $age";  
    }  
}
```

- Lớp SinhVien thừa kế lớp Person và có các phương thức của lớp PersonWithAge

```
class Student extends Person with PersonWithAge{  
    String course;  
    SinhVien(fn, ln, age, this.course):super(fn, ln, age);  
    printInfoSV(){  
        print(getFullInfo());  
        print("Khóa học: $course");  
    }  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

38

interface

- Mỗi class trong Dart ngầm định nghĩa một **interface** chứa tất cả các thành viên của class đó và mọi interface mà nó implement.
 - Có thể implement một lớp bất kỳ
- Nếu muốn tạo class A hỗ trợ các API của class B mà không thừa kế các cài đặt phương thức đã có của class B, class A nên **implement interface B**.
- Có thể sử dụng **abstract class** để định nghĩa một **interface**.

```
class MockSpaceship implements Spacecraft {  
    @override  
    DateTime launchDate;  
    @override  
    String name;  
    @override  
    void describe() {  
        // TODO: implement describe  
    }  
    @override  
    // TODO: implement LaunchYear  
    int get launchYear => throw UnimplementedError();  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

Asynchrony support

- Dart cung cấp các thư viện hay các hàm trả về hai kiểu dữ liệu Stream và Future. Các hàm này sẽ return sau khi *thiết lập* một hoạt động có thể tiêu tốn nhiều thời gian (thường là các hoạt động I/O) mà không phải đợi cho đến khi hoạt động đó hoàn thành
- Khi bạn cần một kết quả Future đã hoàn thành, bạn có hai lựa chọn:
 - Sử dụng từ khóa `async` và `await`
 - Sử dụng Future API
- Một số trường hợp lập trình bất đồng bộ:
 - Lấy dữ liệu từ server.
 - Ghi vào database.
 - Đọc nội dung từ file.
 - Sử dụng các thư viện hỗ trợ lập trình bất đồng bộ.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

40

Định nghĩa hàm bất đồng bộ: `async/async*` và `await`

- Hai từ khóa `async/async*` và `await` cung cấp một cách khai báo để định nghĩa hàm bất đồng bộ.
 - Để định nghĩa một hàm bất đồng bộ ta thêm từ khóa `async` trước thân hàm.
 - Từ khóa `await` chỉ làm việc trong hàm `async`. Từ khóa `await` được sử dụng để lấy chờ lấy một kết quả xử lý bất đồng bộ trong thân hàm `async`.
 - Hàm được khai báo cùng với từ khóa `async` luôn trả về kiểu dữ liệu được *gói* bởi kiểu dữ liệu `Future` và sử dụng từ khóa `return` để trả về giá trị của hàm.
 - Hàm được khai báo cùng với từ khóa `async*` luôn trả về kiểu dữ liệu `Stream` và sử dụng từ khóa `yield` để trả về giá trị thay cho `return`.
 - Sử dụng từ khóa `await` để đợi một giá trị được phát ra bởi một `Stream`.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

41

return Future

- Sau khi thiết lập một hoạt động bất đồng bộ (được xác định bởi từ khóa `await`), hàm BĐB trả về một đối tượng `Future`.
- Future** là kết quả của hoạt động bất đồng bộ và có 2 trạng thái là **chưa hoàn thành** và **hoàn thành**:
 - Chưa hoàn thành:** Khi chúng ta gọi một hoạt động bất đồng bộ, nó sẽ trả về một `Future` chưa hoàn thành, đây là trạng thái của `Future` trước khi trả về kết quả.
 - Hoàn thành:** Khi hoạt động bất đồng bộ thực hiện xong thì `Future` sẽ ở trạng thái hoàn thành, `Future` có thể hoàn thành với một giá trị hoặc là một lỗi.
 - Future** hoàn thành với một giá trị thì nó có thể là `Future<T>` với giá trị có kiểu `T`, `Future<String>` với giá trị kiểu `String`, hoặc là một giá trị `void` với `Future<void>`.
 - Hoàn thành với một error: nếu một hoạt động bất đồng bộ được thực hiện bởi một hàm thất bại vì bất kỳ lý do gì thì `Future` hoàn thành với một error

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

42

Asynchronous functions

```
Future<String> fetchUserOrder() =>
  Future.delayed(
    Duration(seconds: 2),
    () => 'Large Late');
```

```
Future<String> createOrderMessage () async {
  var order =await fetchUserOrder();
  return 'Your order is $order';
}
```

```
void main()async{
  print('Fetching user order...');
  print(await createOrderMessage());
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

43

Asynchronous programming: streams

- Stream cung cấp một chuỗi dữ liệu không đồng bộ.
- Chuỗi dữ liệu bao gồm các sự kiện do người dùng tạo ra và dữ liệu đọc từ các tệp.
- Bạn có thể xử lý luồng bằng cách sử dụng `await` hoặc `listen()` từ Stream API.
- Stream cung cấp một cách để xử lý lỗi
- Có hai loại luồng: single subscription or broadcast.
- Lập trình bất đồng bộ trong Dart được đặc trưng bởi hai lớp `Future` và `Stream`

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

44

Stream, async, await

- Stream có thể được tạo theo nhiều cách, nhưng chúng có thể được sử dụng theo cùng một cách *asynchronous for loop* (vòng lặp for bất đồng bộ – thường được gọi là *await for*) lặp qua các sự kiện của một stream như vòng lặp for loop lặp qua một iterable.
- Từ khóa *await* chỉ làm việc trong hàm *async*
- **async**: Bạn có thể sử dụng từ khóa *async* trước thân hàm bất đồng bộ.
- **async function**: là một function được đánh dấu bởi từ khóa *async*.
- **await**: bạn có thể sử dụng từ khóa *await* để lấy kết quả từ một việc bất đồng bộ. Từ khóa *await* chỉ được sử dụng với hàm *async*.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

45

Nhận các sự kiện stream

```
Future<int> sumStream(Stream<int> stream) async {  
    var sum=0;  
    await for(var value in stream)  
        sum += value;  
    return sum;  
}  
  
Stream<int> countStream(int to) async*{  
    for(int i=0; i<=to; i++) {  
        yield i;  
    }  
}  
  
void main() async{  
    var stream = countStream(10);  
    var sum = await sumStream(stream);  
    print(sum);  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

46

Sử dụng Future API: await, then

```
const oneSecond = Duration(seconds: 1);  
Future<void> printWithDelay(String message) async {  
    await Future.delayed(oneSecond);  
    print(message);  
}
```



```
const oneSecond = Duration(seconds: 1);  
Future<void> printWithDelay(String message) {  
    return Future.delayed(oneSecond).then((_) {  
        print(message);  
    });  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

47

Sử dụng Future API: await, then

```
Future<int> sumStream(Stream<int> stream) async {  
    var sum=0;  
    await for(var value in stream)  
        sum += value;  
    return sum;  
}  
  
Stream<int> countStream(int to) async*{  
    for(int i=0; i<=to; i++) {  
        yield i;  
    }  
}  
  
void main() async{  
    var stream = countStream(10);  
    sumStream(stream)  
        .then((value) => print(value));  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

48

FLUTTER CƠ BẢN



Huỳnh Tuấn Anh
Khoa CNTT-ĐHNT



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT



Nội dung

1. Giới thiệu về Widget
2. Một số Widget thông dụng
3. Form Widget
4. Layouts trong Flutter
5. Constraints
6. Assets và Image
7. Navigation và Routing

Huỳnh Tuấn Anh - Khoa CNTT, DHNT

2

1. Giới thiệu về Widget



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT



Widget

- Mọi thứ trên giao diện đều là các **Widget**
- Mỗi **Widget** có thể được gắn với một trạng thái (state). Khi trạng thái của **widget** thay đổi, **widget** có thể được vẽ lại trên màn hình.
- Một widget có thể được bọc trong một widget cha để thừa hưởng các đặt trưng của widget cha.
 - VD: Bọc một **Text** bởi một **Container**
- Mỗi **Widget** thường có một thuộc tính **child** hay **children** và người dùng có thể gán một hay nhiều **Widget** cho thuộc tính này.



You have pushed the button this many times:

0

Decrement

Increment



Huỳnh Tuấn Anh - Khoa CNTT, DHNT

4

Widget – Hello World app!

```
void main() {
  runApp(
    Container(
      color: Colors.white,
      child: Center(
        child: Text(
          "Hello World!",
          textDirection: TextDirection.ltr,
          style: TextStyle(
            fontSize: 40,
            fontWeight: FontWeight.bold,
            color: Colors.black
          ),
        ),
      ),
    ),
  );
}
```

Run

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

5

Basic Widgets

- Text: Widget cho phép hiển thị chuỗi text có thể được định dạng trên màn hình ứng dụng.
- VD:

```
Text(
  'Hello World!',
  textDirection: TextDirection.ltr,
  style: TextStyle(
    color: Colors.blue,
    fontSize: 40
  ),
),
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

6

Basic Widgets

- Row, Column: Thường được sử dụng để tạo các layout một cách linh hoạt.
 - Row: Được sử dụng để tạo layout sắp xếp các Widget theo chiều ngang.
 - Column: Được sử dụng để tạo layout sắp xếp các Widget theo chiều dọc.
- Stack: Được sử dụng để xếp các Widget chồng lên nhau. Widget đưa vào Stack sau cùng được hiển thị trên các Widget đưa vào trước.
 - Positioned: Widget Được sử dụng để bọc một widget con của Stack để chỉ định vị trí của widget con đó trong stack.
- Container: Cho phép tạo một thành phần hiển thị hình chữ nhật. Thường được sử dụng để bọc các Widget khác để xác định không gian hiển thị hay "Decorate" cho các widget này.
 - BoxDecoration: Thành phần dùng để Decorate cho Widget

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

7

Container – Ví dụ

```
Center(
  child: Container(
    width: 300,
    height: 200,
    decoration: BoxDecoration(
      color: Colors.lightGreenAccent,
      border: Border.all(color: Colors.blue),
      borderRadius: BorderRadius.circular(20),
    ),
    child: Center(
      child: Text(
        'Hello, world!',
        textDirection: TextDirection.ltr,
      ),
    ),
  ),
),
```

Hello, world!

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

8

Statefull và Stateless Widget



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

9

Stateless Widget

- Widget không có trạng thái bên trong nào thay đổi trong thời gian tồn tại của nó.
- Widget này không quan tâm đến cấu hình và trạng thái mà nó hiển thị, nó có thể nhận dữ liệu hay cấu hình từ thành phần cha, nhưng tự nó không thể thay đổi dữ liệu và cấu hình của chính nó.
- Một Stateless Widget không thể tự phát sinh sự kiện để render lại chính mình.
- Không thay đổi khi người dùng tương tác với widget.
- Một số Stateless Widget: [Icon](#), [IconButton](#), [Text](#)
- Các Stateless Widget là lớp con của lớp [StatelessWidget](#)

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

10

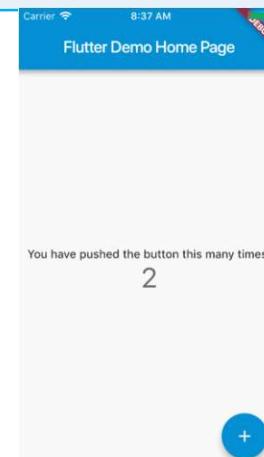
Stateful Widget

- Có trạng thái bên trong và có thể quản lý, tự thay đổi được trạng thái của chính mình.
- Có thể chứa nhiều StatefulWidget khác.
- Thay đổi khi người dùng tương tác với widget.
- Một số Stateful widget: [Checkbox](#), [Radio](#), [Slider](#), [InkWell](#), [Form](#), [TextField](#).
- Các Stateful widget là lớp con của lớp [StatefulWidget](#)
- Trạng thái của widget được lưu trữ trong đối tượng State tách biệt với StatefulWidget
 - Phương thức [setState\(\)](#): Báo cho framework vẽ lại widget với trạng thái mới đã được cập nhật.

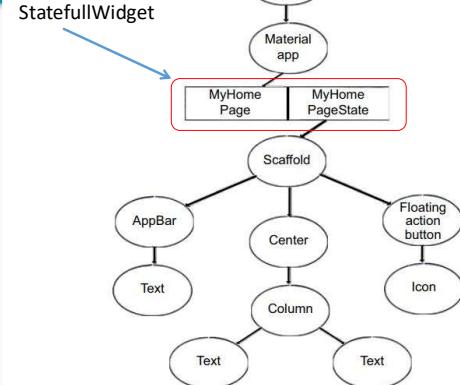
Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

11

Widget Tree



StatefulWidget



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

12

StatefulWidget

- Mỗi thể hiện của một StatefulWidget liên quan đến hai class

Overrides the superclass method createState

```
class MyHomePage extends StatefulWidget { ← Inherits from StatefulWidget
  @override
  _MyHomePageState createState() => _MyHomePageState(); ←
}
```

```
class _MyHomePageState extends State<MyHomePage> { ←
  @override
  Widget build(BuildContext context) {
    // ...
  }
}
```

StatefulWidget's required build method

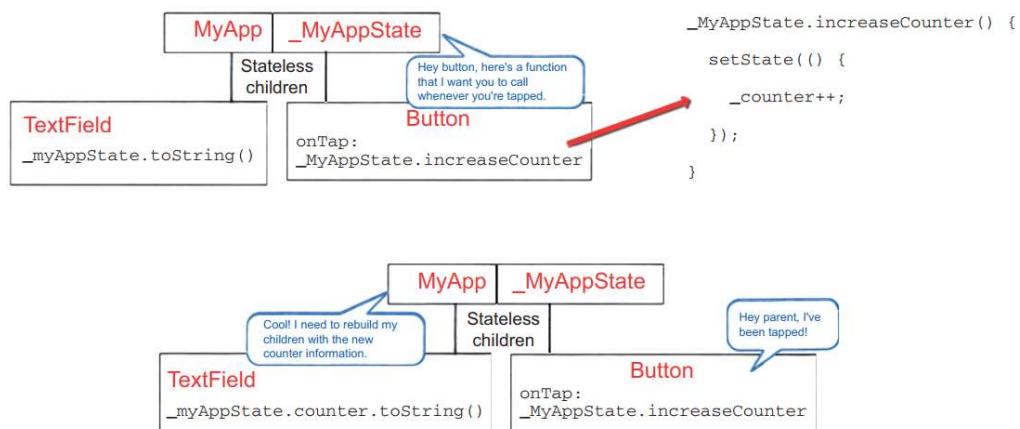
Every StatefulWidget must have a createState method that returns a State object.

Your state class inherits from the Flutter State object.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

13

State class



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

14

Phương thức initState

- Phương thức của lớp State dùng để khởi tạo trạng thái của widget khi đưa widget vào cây widget. Thông tin mà widget hiển thị lần đầu trên màn hình sẽ được khởi tạo trong phương thức này.

```
class FirstNameTextState extends State<FirstNameText> {
  String name;

  FirstNameTextState(this.name);

  @override
  initState() {
    super.initState();
    name = name.toUpperCase();
  }

  Widget build(BuildContext context) {
    return Text(name);
  }
}
```

The `State.initState` method is marked as `mustCallSuper` in the superclass. So, you must call the superclass implementation of `initState` in your overridden method.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

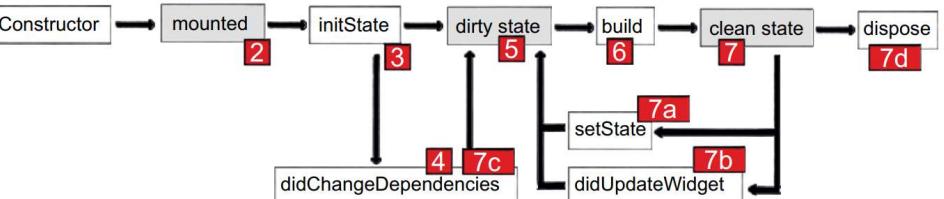
15

StatefulWidget Lifecycle

1 | StatefulWidget

Constructor → Widget.createState()

2 | State object



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

16

BuildContext

- Đối tượng được Flutter cung cấp để:
 - Theo dõi toàn bộ widget tree, chứa thông tin vị trí của widget trong widget tree.
 - Là tham số của phương thức build, tham chiếu đến vị trí của widget trong cây.
 - Mỗi widget có một buildContext riêng của nó.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

17

Widget Tree

Huỳnh Tuấn Anh - Khoa CNTT, DHNT

18

Widget Tree

- UI trong Flutter bao gồm nhiều Widget kết hợp với nhau hình thành nên một Widget Tree.
- Mỗi ứng dụng Flutter được biểu diễn bởi một Widget Tree trong đó mỗi node là một widget.
- Mỗi Widget có thuộc tính child chứa một widget con; hoặc thuộc tính children chứa một danh sách các widget con

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

19

Widget Tree

Carrier 8:37 AM

Flutter Demo Home Page

You have pushed the button this many times:

2

+

StatefulWidget

MyApp

Material app

MyHome Page

MyHome PageState

Scaffold

AppBar

Text

Center

Column

Floating action button

Icon

Text

Text

Huỳnh Tuấn Anh - Khoa CNTT, DHNT

20



Widget Key

- Cho phép Flutter phân biệt và quản lý các widget. Đặc biệt hữu ích trong các trường hợp các widget có cùng kiểu với nhau.
- Kiểm soát việc một widget thay thế một widget khác trong cây.
- Sử dụng Global key cho phép một widget di chuyển trong cây mà không mất đi trạng thái (state) của nó.
- Các dạng của widget key: ValueKey, ObjectKey, UniqueKey, GlobalKey, PageStorageKey

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

21



Widget key – Global Key

- Được sử dụng để quản lý trạng thái và di chuyển widget trong cây widget
- Ví dụ: Có thể sử dụng một CheckBox với một Global key trên nhiều trang. Global Key báo cho Flutter biết rằng các CheckBox trên các trang là một. Khi người dùng thay đổi trạng thái check của CheckBox ở một trang thì trạng thái của các CheckBox có cùng Key ở các trang khác cũng thay đổi theo.
- Tuy nhiên, việc sử dụng Global Key để quản lý state là không được khuyến nghị theo Flutter Team bởi vì nó ảnh hưởng đến hiệu năng của hệ thống.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

22



Wigget Key – Local Key

- Tất cả các khóa cục bộ đều giống nhau ở chỗ chúng được xác định theo bối cảnh xây dựng mà bạn đã tạo khóa.
 - ValueKey<T>
 - ObjectKey
 - UniqueKey
 - PageStorageKey

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

23

2. Một số widget thông dụng



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

24

Một số widget thông dụng

- 1. Container
- 2. Row, Column
- 3. Expanded
- 4. Stack
- 5. Image
- 6. Text, TextField
- 7. Icon
- 8. RaisedButton, DropdownButton
- 9. Scaffold

- 10. AppBar
- 11. SafeArea
- 12. SingleChildScrollView
- 13. Builder
- 14. StatefulBuilder
- 15. FutureBuilder<T>: Widget được xây dựng để sử dụng với các nguồn dữ liệu không đồng bộ.
- 16. StreamBuilder<T>: Widget được xây dựng để làm việc với dữ liệu Stream

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

25

RaisedButton

- RaisedButton: Nút bấm, thường xử lý một sự kiện nào đó
- Constructor:
 - onPressed: một VoidCallback để xử lý sự kiện khi người dùng chạm vào nút.
 - child: Widget, thường là một Text để hiển thị tên của nút bấm

```
Widget build(BuildContext context) {  
  return Container(  
    color: Colors.white,  
    child: Center(  
      child: RaisedButton(  
        child: const Text("Enable Button",  
          style: TextStyle(fontSize: 20),),  
        onPressed: () {},  
      ),  
    );  
}
```



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

26

Image

- Widget được sử dụng để hiển thị ảnh
- Hiển thị ảnh từ internet: Sử dụng Constructor: `Image.network(url)`
- Hiển thị ảnh từ thư mục asset: Sử dụng Constructor: `Image.asset(path_image)`
 - Cách thực hiện:
 - Tạo thư mục images trong thư mục gốc của project ứng dụng
 - Khai báo các image trong pubspec.yaml:

```
uses-material-design: true  
asset:  
  - images/pic1.jpg  
  - images/pic2.jpg
```
 - Ví dụ hiển thị pic1.jpg:
 - `Image.asset('images/pic1.jpg')`

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

27

Icon

- Widget hiển thị Icon
- Flutter cung cấp rất nhiều Icon đã được xây dựng sẵn.
- Ví dụ:

```
Row(  
  mainAxisAlignment: MainAxisAlignment.min,  
  children: [  
    Icon(Icons.star, color: Colors.green[500]),  
    Icon(Icons.star, color: Colors.green[500]),  
    Icon(Icons.star, color: Colors.green[500]),  
    Icon(Icons.star, color: Colors.black),  
    Icon(Icons.star, color: Colors.black),  
  ],  
)
```

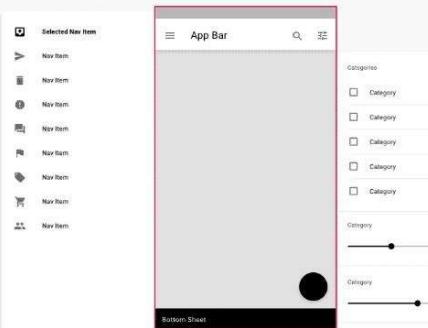


Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

28

Scaffold

- Cung cấp layout [Material Design](#) cơ bản, cho phép thêm các widget khác như [AppBar](#), [BottomAppBar](#), [FloatingActionButton](#), [Drawer](#), [SnackBar](#), [BottomSheet](#)...



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

29

Scaffold – Một số thuộc tính cơ bản

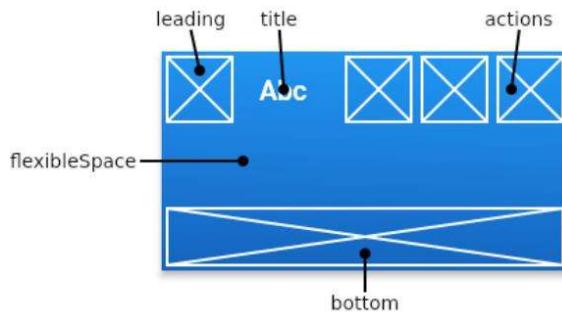
- body:** Widget: thể hiện giao diện chính của Scaffold.
- appBar:** [PreferredSizeWidget](#) Thường là một [AppBar](#) hay [TabBar](#) của một màn hình.
- drawer:** Widget Biểu diễn một Drawer ở phía bên trái.
- endDrawer:** Widget Biểu diễn một Drawer ở phía bên phải.
- bottomNavigationBar:** Widget biểu diễn thanh điều hướng ở dưới màn hình.
- floatingActionButton:** Widget biểu diễn một nút Floating button thường ở góc dưới bên phải của màn hình.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

30

AppBar

- Được sử dụng cho thuộc tính [appBar](#) của Scaffold



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

31

AppBar – Constructor

```
AppBar(  
    Key key,  
    Widget leading,  
    bool automaticallyImplyLeading: true,  
    Widget title,  
    List<Widget> actions,  
    Widget flexibleSpace,  
    PreferredSizeWidget bottom,  
    double elevation,  
    Color shadowColor,  
    ShapeBorder shape,  
    Color backgroundColor,  
)  
  
Brightness brightness,  
IconThemeData iconTheme,  
IconThemeData actionsIconTheme,  
TextTheme textTheme,  
bool primary: true,  
bool centerTitle,  
bool excludeHeaderSemantics: false,  
double titleSpacing: NavigationToolbar.kMiddleSpacing,  
double toolbarOpacity: 1.0,  
double bottomOpacity: 1.0,  
double toolbarHeight,  
double leadingWidth}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

32

AppBar

```
appBar: AppBar(
  title: Text('Hello Flutter!'),
  leading: Icon(Icons.menu),
  actions: <Widget>[
    IconButton(icon: Icon(Icons.share), onPressed: ()=>{}),
    IconButton(icon: Icon(Icons.call), onPressed: ()=>{}),
  ],
),
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 33

SingleChildScrollView

- Thêm khả năng cuộn ngang/dọc cho một widget

```
const SingleChildScrollView<
  {Key key,
  Axis scrollDirection: Axis.vertical,
  bool reverse: false,
  EdgeInsetsGeometry padding,
  bool primary,
  ScrollPhysics physics,
  ScrollController controller,
  Widget child,
  DragStartBehavior dragStartBehavior: DragStartBehavior.start,
  Clip clipBehavior: Clip.hardEdge,
  String restorationId}
()
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 34

SingleChildScrollView – ví dụ

```
body: SafeArea(
  child: SingleChildScrollView(
    child: Column(
      children: [
        Container(height: 300,
          color: Colors.yellow,),
        Container(height: 300,
          color: Colors.blue[300],),
        Container(height: 300,
          color: Colors.green[300],),
      ],
    ),
  ),
),
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 35

Expanded

- Widget giúp widget con của nó chia phần diện tích còn lại của Layout (Row hay Column)

```
body: Center(
  child: Row(
    children: [
      _getLoiChao("Hello",Colors.blue[100]),
      _getLoiChao("Chào",Colors.red[100], 2),
      _getLoiChao("Привет",Colors.green[100])
    ],
  ),
)

Widget _getLoiChao(String st,Color color, [int flex=1]) {
  return Expanded(flex:flex,
    child: Container(height: 100,color: color,
      child: Center(
        child: Text(st, style: TextStyle(fontSize: 20),),
      )));
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 36

TextField

- Widget cho phép user nhập chuỗi text từ bàn phím

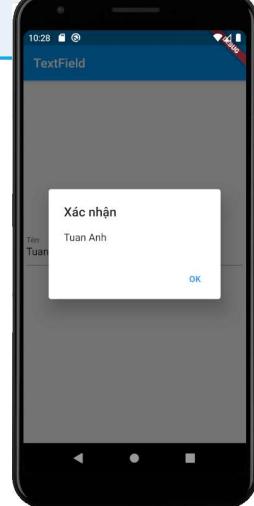
- Sử dụng TextEditingController để lấy/thiết lập nội dung cho TextField

```
body: Center(  
    child: Padding(  
        padding: const EdgeInsets.all(5.0),  
        child: Column(  
            mainAxisAlignment: MainAxisAlignment.center,  
            children: [  
                TextField(  
                    controller: nameEditingController,  
                    decoration: InputDecoration(  
                        hintText: "Nhập tên vào đây",  
                        labelText: "Tên:"  
                    ),  
                    RaisedButton(onPressed: () => _showAlertDialog(context),  
                        child: Text("Submit"),  
                    ),  
                ),  
            ],  
        ),  
    ),  
)
```



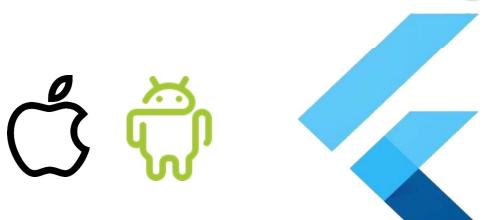
Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 37

```
void _showAlertDialog(BuildContext context){  
    AlertDialog dialog = AlertDialog(  
        title: Text("Xác nhận"),  
        content: Text(nameEditingController.text),  
        actions: [  
            FlatButton(onPressed: () => Navigator.pop(context),  
                child: Text("OK"))  
        ],  
    );  
    showDialog(context: context,  
        builder: (context) => dialog,  
    );  
}
```



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 38

3. Form Widget



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 39

Nội dung

- Form Widget
- GlobalKey<FormState>
- FormField Widget
- TextFormField
- DropdownButtonFormField

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 40

Form

- Là một container để nhóm các nhiều **FormField** cùng với nhau.
- Mỗi trường trong Form nên được bọc bởi **FormField** Widget với **Form** Widget là *ancestor* chung của tất cả các trường.
- Mỗi Form có một đối tượng **FormState** được khai báo thông qua **GlobalKey**
- Có thể gọi các phương thức trên **FormState** để lưu, đặt lại (reset) hoặc xác thực lại từng **FormField** con của Form.
- Để lấy **FormState**
 - Gọi **Form.of** với context có ancestor là Form; hoặc
 - Chuyển một **GlobalKey** cho thuộc tính **key** trong phương thức khởi tạo của **Form** và gọi **GlobalKey.currentState**.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

41

Form

- **key: GlobalKey<FormState>**
- **autovalidateMode: AutovalidateMode**
 - **AutovalidateMode.disabled** (default)
 - **AutovalidateMode.onUserInteraction**: Form field chỉ tự động validate sau khi nội dung thay đổi
 - **AutovalidateMode.always**: Có thể tự động validate mà không cần sự tương tác của user
- **onWillPop: WillPopCallback**
 - Cho phép Form phủ quyết nỗ lực của người dùng để loại bỏ ModalRoute có chứa Form.
 - **Return: Future<false>**: "form route" sẽ không được pop

```
const Form({  
  1.Key key,  
  2.@required Widget child,  
  3.WillPopCallback onWillPop,  
  4.VoidCallback onChanged,  
  5.AutovalidateMode autovalidateMode  
)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

42

GlobalKey<FormState>

- Đối tượng dùng để kiểm soát **Form** thông qua **FormState**.
- Các phương thức trên **FormState**
 - **validate()**: nếu **AutovalidateMode.disabled** sẽ không xảy ra một validation nào cho đến khi phương thức validate() được gọi
 - **save()**: Gọi các callback **onSaved** của các **FormField**
 - **reset()**

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

43

FormField Widget

- Form cung cấp **FormState** để làm việc với các trường đầu vào có liên quan với nhau. Để Form quản lý được các trường đầu vào này, chúng phải là các **FormField** Widget.
- **FormField** Widget được dùng để bọc các widget khác, giúp chúng có thể được quản lý bởi Form.
 - Ví dụ: Có thể sử dụng CheckBox trong Form nhưng nó phải được bọc trong **FormField**

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

44

TextField

- Là một **FormField** đặc biệt đã bọc sẵn một **TextField**
- **FormField** không cần thiết phải sử dụng với một **Form** làm ancestor. Trong trường hợp này cần phải chỉ định một **GlobalKey** cho **TextField** và sử dụng **GlobalKey.currentState** để lưu hoặc reset **FormField**

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

45

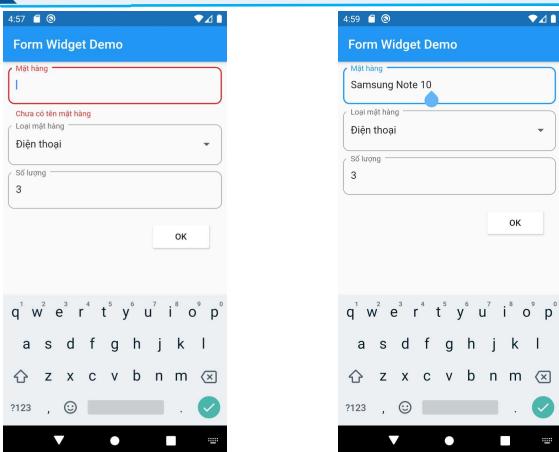
DropdownButtonFormField

- Là một widget tiện ích đã được bọc sẵn **DropdownButton** trong một **FormField**
- Các thuộc tính:
 - **@required List<DropdownMenuItem<T>> items,**
 - **@required this.onChanged,**
 - **FormFieldSetter<T> onSaved,**
 - **FormFieldValidator<T> validator,**

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

46

Ví dụ



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

47

App

```
class MyForm extends StatelessWidget {  
  GlobalKey<FormState> _formState = GlobalKey<FormState>();  
  MatHang mh = MatHang();  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(...), // AppBar  
      body: Form(...), // Form  
    ); // Scaffold  
  }  
  
  List<String> loaiMHs = ['Tivi', 'Điện thoại', 'Laptop'];  
  class MatHang{  
    String tenMH,loaiMH;  
    int soLuong;  
    MatHang({this.tenMH, this.loaiMH, this.soLuong});  
  }  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

48

```

  body: Form(
    key: _formState,
    autovalidateMode: AutovalidateMode.disabled,
    child: Padding(
      padding: const EdgeInsets.all(8.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.center,
        children: [
          TextFormField(...), // TextFormField
          SizedBox(height: 10.),
          DropdownButtonFormField<String>(...), // Input
          SizedBox(height: 10.),
          TextFormField(...), // TextFormField
          SizedBox(height: 20.),
          Row(
            children: [
              Expanded(child: SizedBox(), flex: 1.),
              RaisedButton(...), // RaisedButton
              SizedBox(width: 20.),
            ],
          ),
        ],
      ) // Row
    )
  )

```



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

49

TextField

```

TextField(
  decoration: InputDecoration(
    labelText: 'Mặt hàng',
    border: OutlineInputBorder(
      borderRadius: BorderRadius.all(Radius.circular(10))
    )
  ),
  onSaved: (newValue) {mh.tenMH=newValue;},
  validator: (value) => value.isEmpty ? "Chưa có tên mặt hàng" : null,
),

String _validateSoluong(String value) {
  int sl = value.isEmpty ? 0 : int.tryParse(value);
  return sl<=0 ? "Số lượng mặt hàng phải lớn hơn 0" : null;
}

```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

50

DropdownButtonFormField

```

DropdownButtonFormField<String>(
  items: loaiMHs.map((loaiMH) => DropdownMenuItem<String>(
    child: Text(loaiMH),
    value: loaiMH,
  ).toList(),
  onChanged: (value) {
    mh.loaiMH=value;
  },
  validator: (value) => value ==null? "Chưa chọn loại mặt hàng" : null,
  decoration: InputDecoration(
    labelText: 'Loại mặt hàng',
    border: OutlineInputBorder(
      borderRadius: BorderRadius.all(Radius.circular(10))
    )
  ),),

```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

51

RaisedButton

```

RaisedButton(
  child: Text('OK'),
  color: Colors.white,
  onPressed:() {
    if(_formState.currentState.validate())
    {
      _formState.currentState.save();
      _showAlertDialog(context);
    }
  },
)

```

```

void _showAlertDialog(BuildContext context){
  AlertDialog alertDialog = AlertDialog(
    title: Text('Xác nhận'),
    content: Text("Bạn nhập mặt hàng: ${mh.tenMH}\n"
      "Thuộc loại mặt hàng: ${mh.loaiMH}\n"
      "Với số lượng: ${mh.soluong}"),
    actions: [
      FlatButton(
        onPressed: () => Navigator.pop(context),
        child: Text('Ok'))
    ],
    showDialog(
      context: context,
      builder:(context) => alertDialog,
    );
}

```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

52

4. Layout trong Flutter



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

53

Layout trong Flutter

- Mọi thứ trong Flutter đều là Widget và các **Layout** trong Flutter cũng là các Widget.
- Widget rõ: Được nhìn thấy như: **Image, Icon, Text, Button..**
- Widget Layout: Không được nhìn thấy như: **Row, Column, Stack, Grid, Constraint...**

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

54

Ví dụ



CALL



ROUTE



SHARE

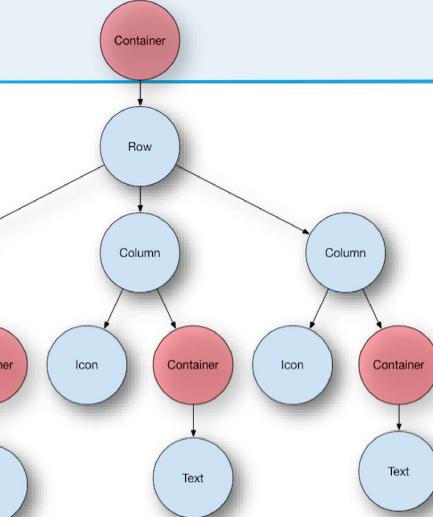
Có thể hình dung hình trên bao gồm một dòng có 3 cột. Mỗi cột gồm một Icon và một chuỗi Text



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

55

Ví dụ



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

56

Layout một widget

- Chọn một *layout widget* phù hợp với mong muốn hiển thị của một *Widget rõ*. Các ràng buộc hiển thị thường là:
 - *Cách thức các widget được sắp xếp*: Theo hàng, cột, lưới, danh sách. Các Widget thường được sử dụng là: *Row*, *Column*, *GridView*, *ListView*
 - *Các ràng buộc về hiển thị*: Căn lề, chừa lề, phân chia khoảng trống... Các Widget thường được sử dụng là: *Container*, *Padding*, *Center*, *Expanded*.
- Trong Flutter, *các thuộc tính về layout của widget cha sẽ truyền lại cho widget con*. Do đó, ta có thể sử dụng các Layout Widget để gói các Widget rõ để truyền các đặc tính về layout của widget cha cho widget con.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

57

Tạo một widget rõ

- Phương pháp: Gọi hàm khởi tạo của widget cần tạo:
- Tạo một Text widget:

```
Text('Hello World'),
```
- Tạo một Image widget:

```
Image.asset('images/lake.jpg',  
fit: BoxFit.cover,  
,)
```
- Tạo một Icon widget:

```
Icon(  
Icons.star,  
color: Colors.red[500],  
,)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

58

Đưa widget rõ vào layout widget

- Tất cả các layout widget đều có thuộc tính:
 - *child*: Có thể nhận một widget con. Ví dụ: Container, Center, Expanded... Hoặc
 - *children*: Có thể nhận một danh sách các widget con. Ví dụ: Row, Column, Stack...
- **Đưa widget rõ vào layout widget:** Truyền *widget rõ* cho thuộc tính *child* hoặc *children* của layout widget.
- **Ví dụ:** Để dòng Text hiển thị ở chính giữa một không gian cho trước nào đó, ta sử dụng widget Center để gói widget Text.

```
Center(  
child: Text('Hello World'),  
,)
```

```
Center(  
child: Text('Hello World'),  
,)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

59

Thêm layout widget vào một trang (page)

- Trong ứng dụng flutter, mỗi màn hình được xem là một trang (page).
- Bản thân ứng dụng Flutter cũng là một widget, hầu hết các widget đều có phương thức build. Việc tạo và trả về một widget trong ứng dụng sẽ hiển thị widget đó.
- **Material Apps:** Sử dụng *Scaffold* widget
 - Nên sử dụng các widget trong *Material Library* với *Material App*.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

60

Material Apps

```
class MyApp extends StatelessWidget {  
    @override  
    Widget build(BuildContext context) {  
        return MaterialApp(  
            title: 'Flutter layout demo',  
            home: Scaffold(  
                appBar: AppBar(  
                    title: Text('Flutter layout demo'),  
                ),  
                body: Center(  
                    child: Text('Hello World'),  
                ),  
            ),  
        );  
    }  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

None Material App

```
class MyApp extends StatelessWidget {  
    @override  
    Widget build(BuildContext context) {  
        return Container(  
            decoration: BoxDecoration(color: Colors.white),  
            child: Center(  
                child: Text(  
                    'Hello World',  
                    textDirection: TextDirection.ltr,  
                    style: TextStyle(  
                        fontSize: 32,  
                        color: Colors.black87,  
                    ),  
                ),  
            );  
    }  
}
```

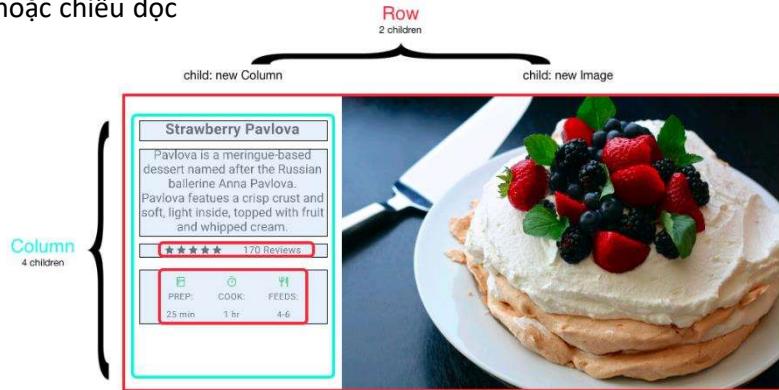
Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

61

62

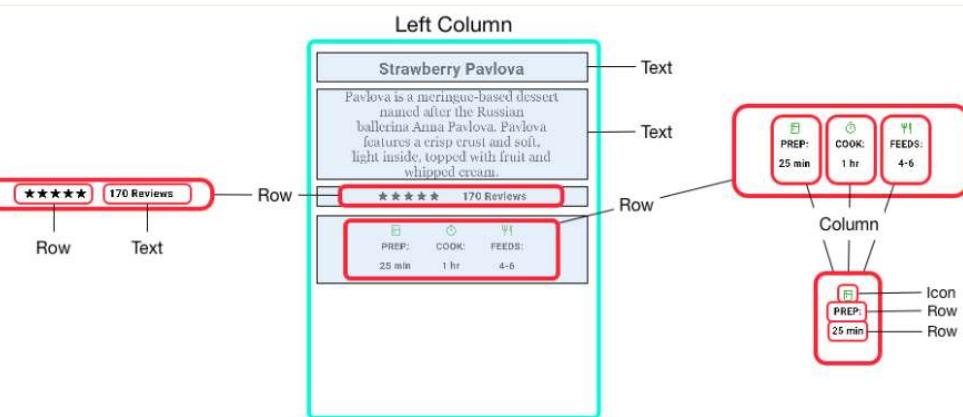
Sắp xếp các widget theo chiều dọc và theo chiều ngang

- Sử dụng **Row** và **Column** để thực hiện việc sắp xếp các widget theo chiều ngang hoặc chiều dọc



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

Sắp xếp các widget theo chiều dọc và theo chiều ngang



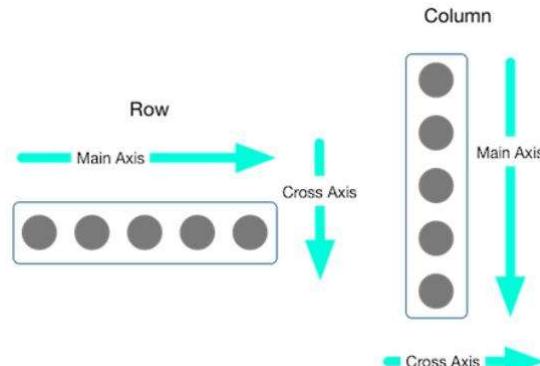
Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

63

64

Sắp các widget

- **mainAxisAlignment:** Cách các widget con được sắp theo trục chính
- **crossAxisAlignment:** Cách các widget được sắp theo trục chéo
- Các sắp các widget: **start, center, end, spaceEvenly, spaceBetween, spaceAround.**



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

65

Sắp các widget

```
Row(  
  mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
  children: [  
    Image.asset('images/pic1.jpg'),  
    Image.asset('images/pic2.jpg'),  
    Image.asset('images/pic3.jpg'),  
  ],  
)
```



App source: row_column

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

66

Kích thước các widget

- Khi một layout lớn hơn kích thước của thiết bị → lỗi hiển thị.
- **Expanded:** Widget cho phép widget con của nó được điều chỉnh kích thước để hiển thị vừa với kích thước của hàng hoặc cột.

```
Row(  
  crossAxisAlignment: CrossAxisAlignment.center,  
  children: [  
    Expanded(  
      child: Image.asset('images/pic1.jpg'),  
    ),  
    Expanded(  
      child: Image.asset('images/pic2.jpg'),  
    ),  
    Expanded(  
      child: Image.asset('images/pic3.jpg'),  
    ),  
  ],  
)
```



App source: sizing

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

67

Expanded Widget

- Sử dụng thuộc tính **flex** để xác định tỉ lệ phân chia không gian của layout

```
Row(  
  crossAxisAlignment: CrossAxisAlignment.center,  
  children: [  
    Expanded(  
      child: Image.asset('images/pic1.jpg'),  
    ),  
    Expanded(  
      flex: 2,  
      child: Image.asset('images/pic2.jpg'),  
    ),  
    Expanded(  
      child: Image.asset('images/pic3.jpg'),  
    ),  
  ],  
)
```



App source: sizing

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

68

Đóng gói các Widget

- Mặc định Row hoặc Column chiếm hết không gian có thể được theo trục main. Sử dụng thuộc tính `mainAxisSize` để đóng gói các widget con gần nhau hơn

```
Row(  
  mainAxisAlignment: MainAxisAlignment.min,  
  children: [  
    Icon(Icons.star, color: Colors.green[500]),  
    Icon(Icons.star, color: Colors.green[500]),  
    Icon(Icons.star, color: Colors.green[500]),  
    Icon(Icons.star, color: Colors.black),  
    Icon(Icons.star, color: Colors.black),  
  ],  
)
```



App source: pavlova

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

69

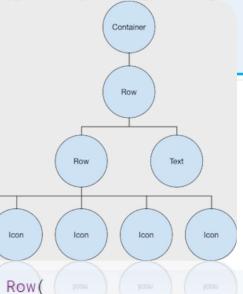
Row và Column lồng nhau

- Mọi thứ trong Flutter đều là Widget kế cả các Layout Widget. Do đó ta có thể sử dụng các Layout Widget như là các widget con.



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

70



```
var stars = Row(  
  mainAxisSize: MainAxisSize.min,  
  children: [  
    Icon(Icons.star, color: Colors.green[500]),  
    Icon(Icons.star, color: Colors.green[500]),  
    Icon(Icons.star, color: Colors.green[500]),  
    Icon(Icons.star, color: Colors.black),  
    Icon(Icons.star, color: Colors.black),  
  ],  
)
```

```
final ratings = Container(  
  padding: EdgeInsets.all(20),  
  child: Row(  
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,  
    children: [  
      stars,  
      Text(  
        '170 Reviews',  
        style: TextStyle(  
          color: Colors.black,  
          fontWeight: FontWeight.w800,  
          fontFamily: 'Roboto',  
          letterSpacing: 0.5,  
          fontSize: 20,  
        ),  
    ],  
  );
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

71

Các layout widget phổ biến

Standard widgets

- Container
- ListView
- GridView
- Stack

Material widgets

- Card
- ListTile

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

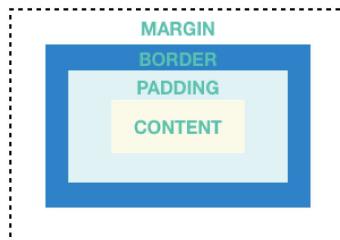
72

Container

- Sử dụng Container để tách cách các widget bằng các thuộc tính padding, border hay margin.

- Sử dụng Container khi cần:

- Thêm padding, lề, đường viền.
- Thay đổi màu nền hay hình ảnh
- Chứa một widget con đơn lẻ, và widget này có thể là Row, Column, hay thậm chí là gốc của cây widget



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

73

Container

```
Widget _buildImageColumn() => Container(  
  decoration: BoxDecoration(  
    color: Colors.black26,  
  ),  
  child: Column(  
    children: [  
      _buildImageRow(1),  
      _buildImageRow(3),  
    ],  
  ),  
)
```



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

74

Container

```
Widget _buildDecoratedImage(int imageIndex) => Expanded(  
  child: Container(  
    decoration: BoxDecoration(  
      border: Border.all(width: 10, color: Colors.black38),  
      borderRadius: const BorderRadius.all(const Radius.circular(8)),  
    ),  
    margin: const EdgeInsets.all(4),  
    child: Image.asset('images/pic$imageIndex.jpg'),  
  ),  
)  
  
Widget _buildImageRow(int imageIndex) => Row(  
  children: [  
    _buildDecoratedImage(imageIndex),  
    _buildDecoratedImage(imageIndex + 1),  
  ],  
)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

75

Gridview

- Sử dụng GridView để sắp xếp các widget dưới dạng *danh sách hai chiều*. GridView cung cấp hai danh sách được tạo sẵn hoặc bạn có thể tạo lưới tùy chỉnh của riêng mình. Khi GridView phát hiện thấy nội dung của nó quá dài để vừa với hộp kết xuất, nó sẽ tự động cuộn.

- Xây dựng một Gridview tùy chỉnh hoặc sử dụng một trong các grid được cung cấp sẵn:

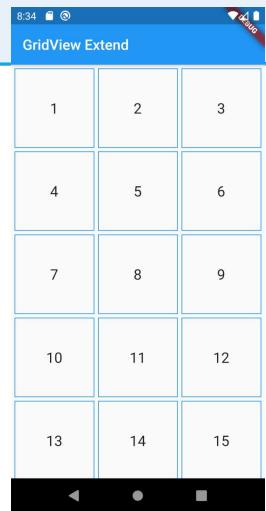
- GridView.count: Chỉ định số lượng cột trong grid.
- GridView.extent: Cho phép chỉ định độ rộng tối đa (theo pixel) của một ô lưới.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

76

Ví dụ: GridView.extent

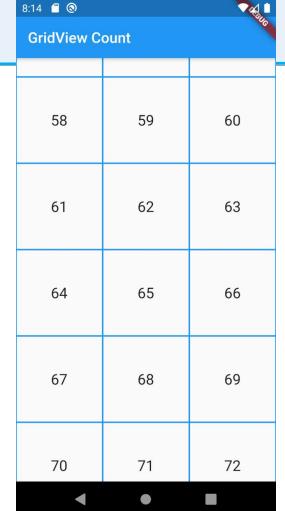
```
body: GridView.extent(  
  maxCrossAxisExtent: 150,  
  padding: EdgeInsets.all(5),  
  mainAxisSpacing: 5,  
  crossAxisSpacing: 5,  
  children: List.generate(100,  
    (index) => Container(  
      decoration: BoxDecoration(  
        border: Border.all(color: Colors.blue)  
      ),  
      child: Center(  
        child: Text("${index+1}",  
          style: TextStyle(fontSize: 20),),  
      ),  
    ),  
  ),  
)
```



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

Ví dụ: GridView.count

```
body: GridView.count(  
  crossAxisCount: 3,  
  children: List.generate(100,  
    (index) => Container(  
      decoration: BoxDecoration(  
        border: Border.all(color: Colors.blue)  
      ),  
      child: Center(  
        child: Text('${index+1}',  
          style: TextStyle(fontSize: 20),),  
      )),  
  ),
```



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

78

ListView

- Tương tự như Column, nhưng tự động cuộn khi các widget con vượt quá kích thước của render box.

▪ ListView:

- Là một Column đặc biệt để tổ chức một danh sách các box.
- Các widget con có thể được sắp theo chiều dọc hoặc chiều ngang.
- Tự động cuộn khi nội dung trong danh sách vượt quá kích thước hiển thị.
- Không cần phải cấu hình nhiều như Column, nhưng rất hỗ trợ cuộn rất dễ dàng

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

ListTile

- Là một row widget đặt biệt trong gói thư viện Material. ListTile gồm 3 phần:

- Trailing icon
- Title
- Subtitle



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

80

ListView

```
Widget build(BuildContext context) {
  return ListView(
    children: <Widget>[
      _tile('CineArts at the Empire', '85 W Portal Ave', Icons.theater),
      _tile('The Castro Theater', '429 Castro St', Icons.theaters),
      _tile('Alamo Drafthouse Cinema', '2550 Mission St', Icons.theat),
      _tile('Roxie Theater', '3117 16th St', Icons.theaters),
      _tile('United Artists Stonestown Twin', '501 Buckingham Way', Icons.theaters),
      _tile('AMC Metreon 16', '135 4th St #3000', Icons.theaters),
      Divider(),
      _tile('K\'s Kitchen', '757 Monterey Blvd', Icons.restaurant),
      _tile('Emmy\'s Restaurant', '1923 Ocean Ave', Icons.restaurant),
      _tile('Chaiya Thai Restaurant', '272 Claremont Blvd', Icons.resta),
      _tile('La Ciccia', '291 30th St', Icons.restaurant),
    ],
  );
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 81

ListTile

```
ListTile _tile(String title, String subtitle, IconData icon) => ListTile(
  title: Text(title,
    style: TextStyle(
      fontWeight: FontWeight.w500,
      fontSize: 20,
    )),
  subtitle: Text(subtitle),
  leading: Icon(
    icon,
    color: Colors.blue[500],
  ),
);
```

CineArts at the Empire
85 W Portal Ave

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 82

Stack

- Được sử dụng để xếp chồng các widget.

```
Widget _buildStack() => Stack(
  alignment: const Alignment(0.6, 0.6),
  children: [
    CircleAvatar(
      backgroundImage: AssetImage('images/pic.jpg'),
      radius: 100,
    ),
    Container(
      decoration: BoxDecoration(
        color: Colors.black45,
      ),
      child: Text(
        'Mia B',
        style: TextStyle(
          fontSize: 20,
          fontWeight: FontWeight.bold,
          color: Colors.white,
        ),
      ),
    ),
  ],
);
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 83

Card

- Một widget trong thư viện Material.
- Được sử dụng để hiển thị các thông tin ngắn gọn có liên quan với nhau được chứa bởi mọi các widget. Thông thường, Card được sử dụng với ListTile.

Không thể cuộn nội dung trên Card.

1625 Main Street
My City, CA 99984

(408) 555-1212

costa@example.com

Top 10 Australian beaches

Number 10
Whitethaven Beach
Whitsunday Island, Whitsunday Islands

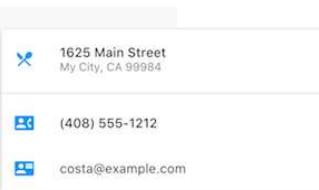
SHARE EXPLORE

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 84



Ví dụ

```
Widget _buildCard() =>
  SizedBox(
    height: 210,
    child: Card(
      child: Column(
        children: [ ListTile(
          title: Text('1625 Main Street',
            style: TextStyle(fontWeight: FontWeight.w500)),
          subtitle: Text('My City, CA 99984'),
          leading: Icon( Icons.restaurant_menu, color: Colors.blue[500],
        ),
      ),
      ...
    
```



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

85

5. Constraints

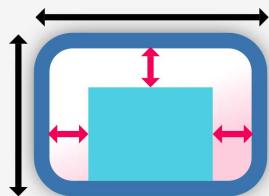


Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

86



Constraints go down.
Sizes go up.
Parent sets position.



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

87



Constraints

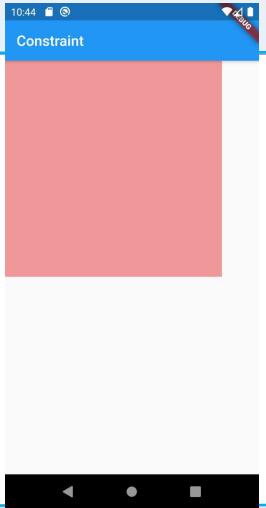
- Một widget nhận các ràng buộc riêng từ widget cha của nó. Ràng buộc chỉ là một tập hợp 4 giá trị double: **minimum and maximum width**, và **minimum và maximum height**.
- Widget cha duyệt qua danh sách con của nó và “nói” cho từng widget con constraint của chúng (có thể khác nhau đối với mỗi widget). Và sau đó “hỏi” kích thước mong muốn của từng widget con.
- Sau đó widget cha định vị các widget con của nó (theo trục x và theo trục y).
- Cuối cùng widget con “nói” cho widget cha về kích thước của nó (tất nhiên là thỏa mãn ràng buộc ban đầu)

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

88

Ví dụ 1

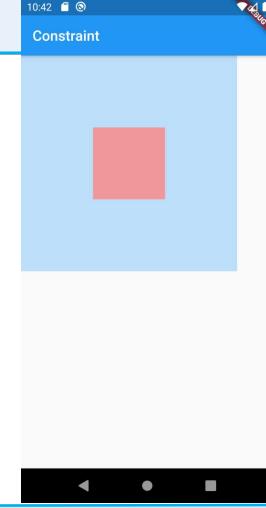
```
class MyConstraint extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text("Constraint"),  
      ),  
      body: Container(  
        width: 300, height: 300,  
        color: Colors.blue[100],  
        child: Container(  
          width: 100, height: 100,  
          color: Colors.red[100],  
        ),  
      ),  
    );  
  }  
}
```



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

Ví dụ 2

```
class MyConstraint extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text("Constraint"),  
      ),  
      body: Container(  
        width: 300, height: 300,  
        color: Colors.blue[100],  
        child: Center(  
          child: Container(  
            width: 100, height: 100,  
            color: Colors.red[200],  
          ),  
        ),  
      );  
    }  
  }  
}
```

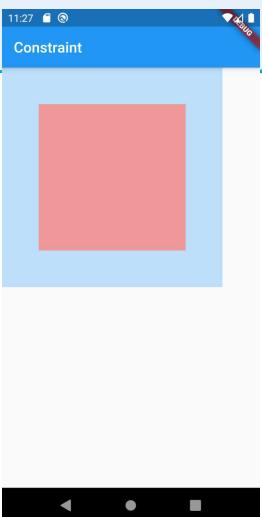


Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

90

Ví dụ 3

```
class MyConstraint2 extends StatelessWidget {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text("Constraint"),  
      ),  
      body: Container(  
        width: 300, height: 300,  
        color: Colors.blue[100],  
        child: Center(  
          child: Container(  
            constraints: BoxConstraints(  
              maxWidth: 200, maxHeight: 200,  
              minWidth: 100, minHeight: 100  
            ),  
            color: Colors.red[200],  
          ),  
        ),  
      ),  
    );  
  }  
}
```



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

91

Ràng buộc chặt chẽ và ràng buộc lỏng lẻo

▪ Ràng buộc chặt chẽ: Cung cấp một ràng buộc chính xác và duy nhất.

```
BoxConstraints.loose(Size size)  
: minWidth = 0.0,  
  maxWidth = size.width,  
  minHeight = 0.0,  
  maxHeight = size.height;
```

▪ Ràng buộc lỏng lẻo: Cung cấp một ràng buộc với các tham số ràng buộc trong một khoảng nào đó.

```
BoxConstraints.loose(Size size)  
: minWidth = 0.0,  
  maxWidth = size.width,  
  minHeight = 0.0,  
  maxHeight = size.height;
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

92

Expanded, Flexible widget

- **Flexible** widget: Cho phép widget con của nó có chiều rộng bằng hoặc nhỏ hơn chính bản thân của **Flexible**.
- **Expanded**: Bắt buộc widget con của nó có chiều rộng bằng với chính bản thân của **Expanded**
- Cả hai widget này đều *bỏ qua chiều rộng của widget con* khi tự xác định kích thước của mình.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

93

Expanded

```
body: Center(
  child: Row(
    children: [
      Expanded(child: Container(color: Colors.red[200],
        child: Text('This is a very long text that '
          'won\'t fit the line.', style: TextStyle(fontSize: 20),),
      )),
      Expanded(child: Container(color: Colors.green[200],
        child: Text('Goodbye!', style: TextStyle(fontSize: 20),),
      )));
    ],
  ),
),
```

This is a very long
text that won't fit
the line. Goodbye!

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

94

Flexible

```
body: Center(
  child: Row(
    children: [
      Flexible(child: Container(color: Colors.red[200],
        child: Text('This is a very long text that '
          'won\'t fit the line.', style: TextStyle(fontSize: 20),),
      )),
      Flexible(child: Container(color: Colors.green[200],
        child: Text('Goodbye!', style: TextStyle(fontSize: 20),),
      )));
    ],
  ),
),
```

8:29 8:29 8:29
Flexible
This is a very long
text that won't fit
the line. Goodbye!

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

95

6. Asset và Image



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

96

Asset

- Asset: Còn gọi là tài nguyên, là các tệp được đóng gói và triển khai cùng với ứng dụng và có thể được truy cập trong thời gian chạy.
- Các dạng Asset: Dữ liệu tĩnh
 - JSON File
 - Image, Icon
 - Các file cấu hình
- Mỗi Asset được định danh bởi một đường dẫn tường minh được khai báo ở tập tin [pubspec.yaml](#). Trật tự khai báo các asset không quan trọng.
- Các asset sẽ được build cùng với ứng dụng.

Khai báo Asset

- Sử dụng file pubspec.yaml, khai báo như sau:

```
flutter:  
  assets:  
    - assets/my_icon.png  
    - assets/background.png
```

Thư mục assets nằm trong thư mục gốc của project

- Để bao gồm tất cả các Asset trong một thư mục, khai báo tên thư mục với ký tự "/" đứng ở cuối

```
flutter:  
  assets:  
    - directory/  
    - directory/subdirectory/
```

Nạp (load) asset vào ứng dụng

- Có 2 phương thức chính trên asset bundle cho phép nạp string/text hoặc image/binary asset trên một logical key
 - `loadString(String path)`: Nạp string/text
 - `load(String path)`: Nạp image/binary

Nạp text asset

- Mỗi ứng dụng Flutter có một đối tượng rootBundle để truy cập asset bundle. Có thể nạp asset một cách trực tiếp bằng cách sử dụng đối tượng tĩnh toàn cục `rootBundle` từ: [package:flutter/services.dart](#).
- `rootBundle` thường được sử dụng bên ngoài một widget context để nạp trực tiếp asset

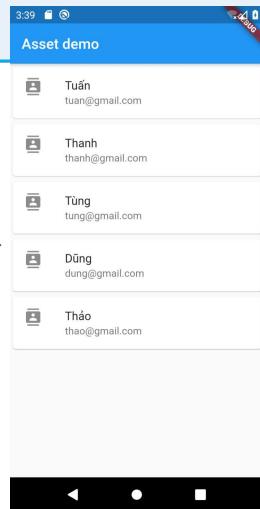
```
import 'dart:async' show Future;  
import 'package:flutter/services.dart' show rootBundle;  
  
Future<String> loadAsset() async {  
  return await rootBundle.loadString('assets/config.json');  
}
```



Ví dụ

- Nạp file Json từ asset

```
[  
  {  
    "name": "Tuấn",  
    "email": "tuan@gmail.com"  
  },  
  {  
    "name": "Thanh",  
    "email": "thanh@gmail.com"  
  },  
  {  
    "name": "Tùng",  
    "email": "tung@gmail.com"  
  },  
  {  
    "name": "Dũng",  
    "email": "dung@gmail.com"  
  },  
  {  
    "name": "Thảo",  
    "email": "thao@gmail.com"  
  }  
]
```



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

101



Code

```
import 'dart:convert';  
import 'package:flutter/services.dart' show rootBundle;  
  
class User {  
  final String name;  
  final String email;  
  User({this.name, this.email});  
  
User.fromJson(Map<String, dynamic> json):  
  name = json['name'],  
  email = json['email'];  
  
Map<String, dynamic> toJson()  
  {  
    return {  
      'name': name,  
      'email': email  
    };  
  }  
}
```

Hai phương thức cần có để serialization/deserialization

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

102



```
Future<List<User>> fetchUserFromJson(String path) async{  
  String jsonStr = await rootBundle.loadString(path);  
  List<User> users;  
  var list = json.decode(jsonStr) as List;  
  users = list.map((item)=> User.fromJson(item)).toList();  
  return users;  
}
```

Đọc text file

Chuyển chuỗi Json thành danh sách các đối tượng User

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

103



Page hiển thị: StatefulWidget

```
class _UserFromAssetPageState extends State<UserFromAssetPage> {  
  Future<List<User>> users;  
  @override  
  void initState() {  
    super.initState();  
    users = fetchUserFromJson('asset/jsons/users.json');  
  }  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(title: Text('Asset demo')),  
      body: Container(  
        child: FutureBuilder<List<User>>(  
          future: users,  
          builder: (context, snapshot) {  
            //...  
          },  
        ),  
      );  
  }  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

104

```

body: Container(
  child: FutureBuilder<List<User>>(
    future: users,
    builder: (context, snapshot) {
      if(snapshot.hasData) {
        return ListView(
          children: List.generate(snapshot.data.length,
            (index) => Card(
              child: ListTile(
                title: Text(snapshot.data[index].name),
                subtitle: Text(snapshot.data[index].email),
                leading: Icon(Icons.contacts),
              ),),);
      }
      else if(snapshot.hasError)
        return Text('Lỗi đọc dữ liệu');
      else
        return CircularProgressIndicator();
    },
  ),
),

```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

105

Nạp Images

- Flutter có thể nạp các image với độ phân giải phù hợp với thiết bị sử dụng
- Khai báo các asset với các ratio khác nhau trong tệp pubspec.yaml theo cấu trúc thư mục:

```

uses-material-design: true
asset:
  - icons/heart.png
  - icons/1.5x/heart.png
  - icons/2.0x/heart.png

```

▪ Nạp image:

```

Widget build(BuildContext context) {
  return Image(image: AssetImage('icons/heart.png'));
}

```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

106

7. Navigation và Routing



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

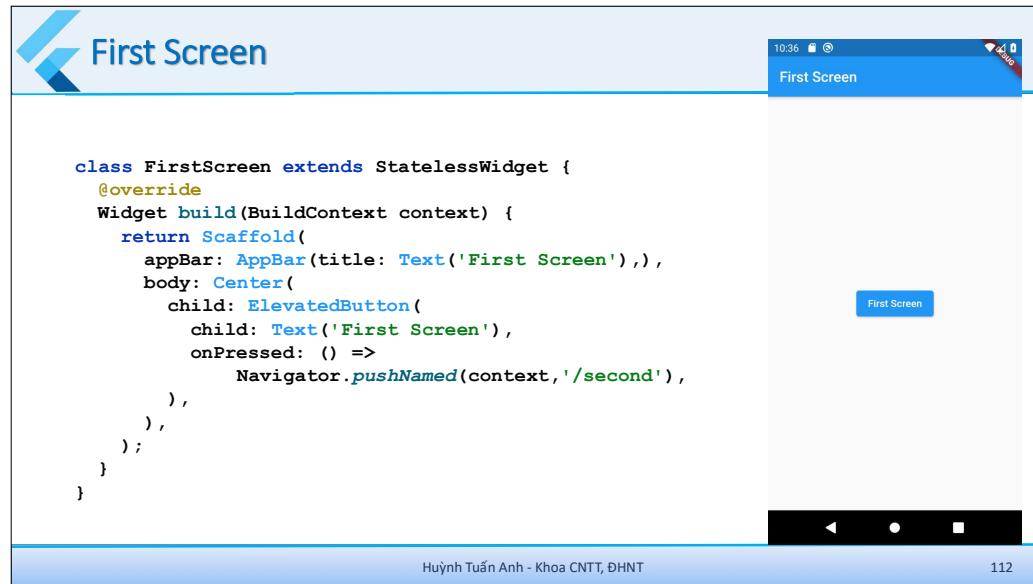
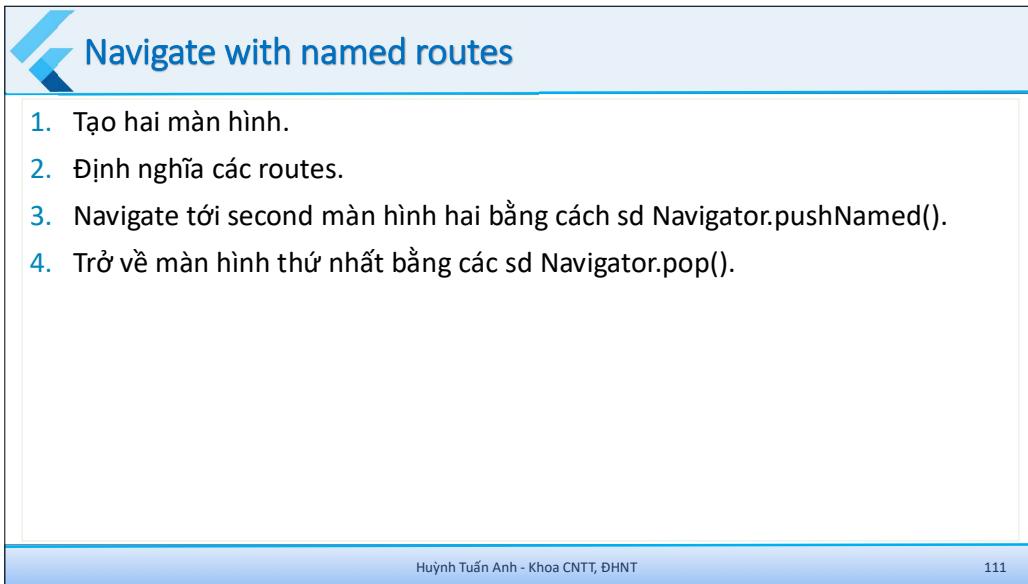
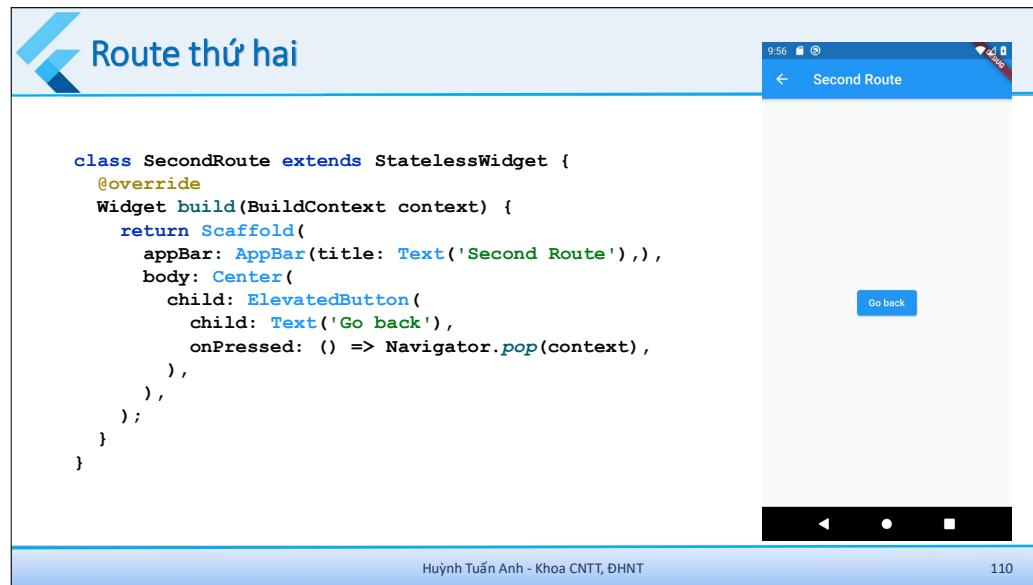
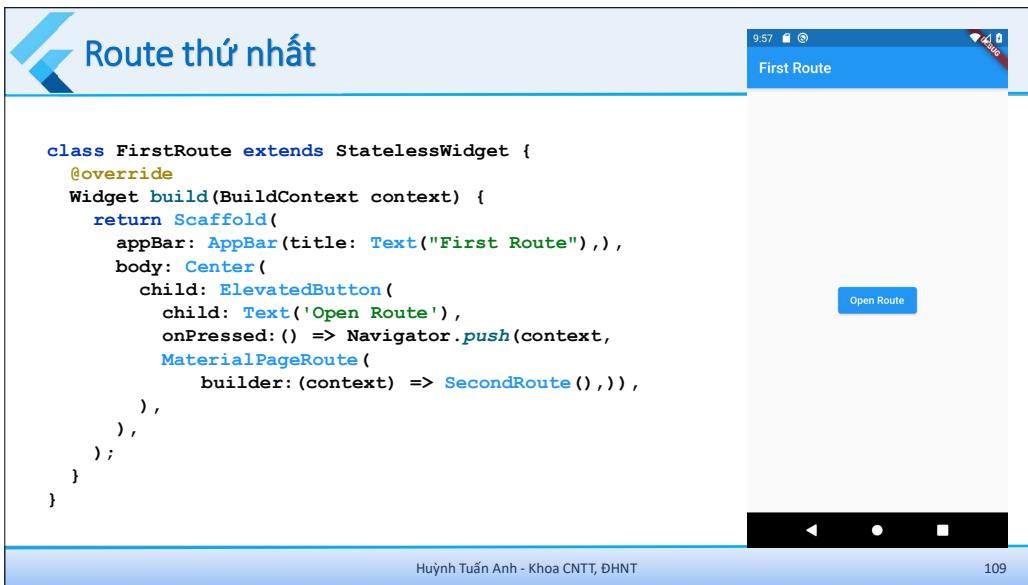
107

Navigate tới màn hình mới và quay ngược lại

- Tạo hai route
 - Trong Flutter thuật ngữ Route ám chỉ một màn hình (screen) hoặc trang (page)
- Điều hướng tới route thứ 2 sử dụng Navigator.push().
- Quay lại route thứ nhất bằng cách sử dụng Navigator.pop().

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

108



Second Screen

```

class SecondScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(title: Text('Second Screen')),
      body: Center(
        child: ElevatedButton(
          child: Text('Second Screen'),
          onPressed: () => Navigator.pop(context),
        ),
      ),
    );
  }
}

```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 113

Định nghĩa các route

```

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      initialRoute: '/',
      routes: {
        '/': (context) => FirstScreen(),
        '/second': (context) => SecondScreen()
      },
    );
  }
}

```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 114

Truyền tham số cho named route

- Định nghĩa các đối số cần truyền.
- Tạo một widget sử dụng các đối số: Trích xuất các đối số bằng ModalRoute.of()
- Đăng ký widget trong bảng route
- Điều hướng tới widget: sử dụng Navigator.pushNamed()

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 115

Ví dụ

11:07 11:09 ← Extract Argument Screen

Gởi tham số gồm title cho màn hình và nội dung của message

This message is extracted in the build method

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 116

Định nghĩa tham số cần truyền

- Định nghĩa lớp dữ liệu biểu diễn tham số cần truyền.
- Ví dụ:

```
class ScreenArguments{  
    String title;  
    String message;  
  
    ScreenArguments(this.title, this.message);  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

Tạo widget sử dụng các đối số được truyền vào route

- Để truy cập các đối số, sử dụng phương thức ModalRoute.of(). Phương thức này trả về route hiện tại với các tham số của nó.

```
class ExtractArgumentScreen extends StatelessWidget {  
    static const routeName = '/extractArguments';  
    @override  
    Widget build(BuildContext context) {  
        final ScreenArguments args =  
            ModalRoute.of(context).settings.arguments;  
        return Scaffold(  
            appBar: AppBar(  
                title: Text(args.title),  
            ),  
            body: Center(  
                child: Text(args.message,  
                    style: TextStyle(fontSize: 20),  
                ),  
            );  
    }  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

118

Đăng ký widget với bảng route

```
class MyApp extends StatelessWidget {  
    // This widget is the root of your application.  
    @override  
    Widget build(BuildContext context) {  
        return MaterialApp(  
            title: 'Flutter Demo',  
            theme: ThemeData(  
                primarySwatch: Colors.blue,  
                visualDensity: VisualDensity.adaptivePlatformDensity,  
            ),  
            initialRoute: '/',  
            routes: {  
                '/':(context) => ProviderArgumentScreen(),  
                ExtractArgumentScreen.routeName:(context)=>ExtractArgumentScreen(),  
            },  
        );  
    }  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

Điều hướng tới widget

- Cung cấp đối số cho route thông qua thuộc tính arguments

Đối số được
truyền vào route

```
class ProviderArgumentScreen extends StatelessWidget {  
    @override  
    Widget build(BuildContext context) {  
        return Scaffold(  
            appBar: AppBar(  
                title: Text("First Screen"),  
            ),  
            body: Center(  
                child: ElevatedButton(  
                    child: Text('Navigate to screen that Extract Argument'),  
                    onPressed:() => Navigator.pushNamed(context,  
                        ExtractArgumentScreen.routeName,  
                        arguments: ScreenArguments(  
                            'Extract Argument Screen',  
                            "This message is extracted in the build method" ),  
                    ),  
                ),  
            );  
    }  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

120



Dữ liệu trả về từ một màn hình

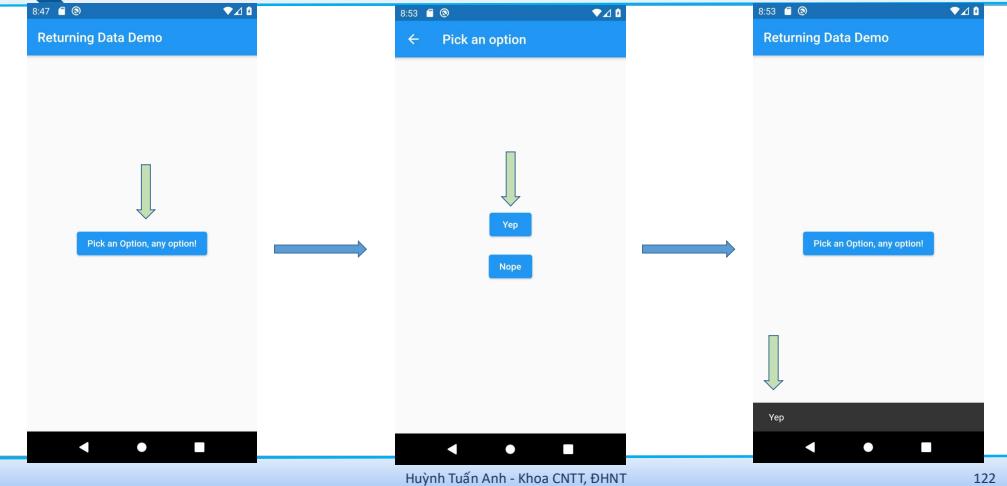
Sử dụng navigator.pop()

```
@optionalTypeArgs  
void pop <T extends Object>(  
    BuildContext context,  
    [T result]  
)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT



Ví dụ:



121



Returning Data Demo

Pick an Option, any option!

```
class SelectionButton extends StatelessWidget {  
    @override  
    Widget build(BuildContext context) {  
        return ElevatedButton(  
            onPressed: () => _navAndDispSelection(context),  
            child: Text('Pick an Option, any option!'));  
    }  
    Future<void> _navAndDispSelection(BuildContext context) async {  
        final result = await Navigator.push(context,  
            MaterialPageRoute(builder: (context)  
                => SelectionPage()));  
        Scaffold.of(context)  
            ..removeCurrentSnackBar()  
            ..showSnackBar(  
                SnackBar(content: Text('$result')));  
    }  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

123



Pick an option

Yep
Nope

```
body: Center(  
    child: Column(  
        mainAxisAlignment: MainAxisAlignment.center,  
        children: [  
            _getButtonOption(context, 'Yep'),  
            _getButtonOption(context, 'Nope')  
        ],  
    ),
```

```
Widget _getButtonOption(BuildContext context, String s) {  
    return Padding(  
        padding: EdgeInsets.all(8),  
        child: ElevatedButton(  
            child: Text(s),  
            onPressed: () => Navigator.pop(context, s),  
        ),  
    );  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

124

Gởi dữ liệu tới một màn hình mới

- Sử dụng Navigator.push
- Ví dụ

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 125

Định nghĩa tham số

```

class Todo{
    String title;
    String description;

    Todo(this.title, this.description);
}

List<Todo> getListTodo() {
    List<Todo> todos = List.generate(20, (index) => Todo(
        "Todo ${index +1}",
        "Đây là Todo thứ ${index +1}"
    ));
    return todos;
}

```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 126

Todo Screen

```

class TodoScreen extends StatelessWidget {
    final List<Todo> todos = getListTodo();
    @override
    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(title: Text('List todo')),
            body: ListView.builder(
                itemCount: todos.length,
                itemBuilder: (context, index) => Card(
                    child: ListTile(
                        title: Text(todos[index].title),
                        leading: Icon(Icons.work, color: Colors.blue,),
                        onTap: () => Navigator.push(context,
                            MaterialPageRoute(
                                builder: (context) => DetailScreen(todo: todos[index]),)),
                    ),
                ),
            );
    }
}

```

Dữ liệu cần gởi sang màn hình chi tiết

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 127

Detail Screen

```

class DetailScreen extends StatelessWidget {
    Todo todo; → Dữ liệu của màn hình 2
    DetailScreen({Key key, this.todo}): super(key: key);
    @override
    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(title: Text('Todo')),
            body: Column(mainAxisAlignment: MainAxisAlignment.center,
                children: [
                    Center(
                        child: Text(todo.title,
                            style: TextStyle(fontSize: 25, fontWeight: FontWeight.bold,
                                color: Colors.blue),),
                    Padding( padding: const EdgeInsets.all(8.0),
                        child: Text(todo.description, style: TextStyle(fontSize: 18,),),
                    )
                ],
            );
    }
}

```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT 128

State Management



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

1

Nội dung

- State management
- provider package
- BLOC package

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

2

Giới thiệu pub.dev

- Trang web cung cấp các pakage hỗ trợ cho việc phát triển ứng dụng Flutter.
- Url: pub.dev
- Một số package/plugin:
 - equitable: Hỗ trợ việc so sánh các đối tượng trong Dart bằng toán tử "=="
 - provider: Quản lý app state
 - BLoC: Quản lý app state
 - shared_preferences: Thư viện giúp đọc ghi các cặp key-value đơn giản.
 - flutter_local_notifications: platform đa nền tảng để hiển thị các thông báo local.
 - firebase_core: Thư viện cho phép kết nối nhiều ứng dụng với firebase

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

3

State Management



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

4

Flutter state

- Flutter mang tính chất khai báo. Điều này có nghĩa là Flutter xây dựng giao diện người dùng của mình để phản ánh trạng thái hiện tại của ứng dụng của bạn.
- Khi state thay đổi → Giao diện người dùng được xây dựng lại từ đầu

$$UI = f(\text{state})$$

The layout
on the screen

Your
build
methods

The application state

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

5

Trạng thái tạm thời

- Trạng thái tạm thời (Còn được gọi là trạng thái UI hay trạng thái cục bộ): Trạng thái có thể được chứa gọn gàng trong một widget. Ví dụ:
 - Trang hiện tại trong một PageView.
 - Tiến trình hiện tại trong một hoạt ảnh phức tạp.
 - Tab hiện tại được chọn trong BottomNavigationBar.
- Không cần phải sử dụng các kỹ thuật quản lý trạng thái đối với các loại trạng thái này. *Tất cả những gì cần thiết chỉ là một StatefulWidget.*
- Trạng thái tạm thời có thể được cài đặt bằng cách sử dụng **State** và phương thức **setState()**.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

6

Trạng thái tạm thời

- Ví dụ:
 - `_index` là một trạng thái tạm thời, chỉ tồn tại trong `MyHomepage`. Các thành phần khác của ứng dụng không cần phải truy cập đến biến `_index` này.
 - `_index` được khởi tạo lại giá trị 0 khi `MyHomePage` được gọi.

```
class MyHomepage extends StatefulWidget {  
  @override  
  _MyHomepageState createState() => _MyHomepageState();  
}  
  
class _MyHomepageState extends State<MyHomepage> {  
  int _index = 0;  
  
  @override  
  Widget build(BuildContext context) {  
    return BottomNavigationBar(  
      currentIndex: _index,  
      onTap: (newIndex) {  
        setState(() {  
          _index = newIndex;  
        });  
        // ... items ...  
      },  
    );  
  }  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

7

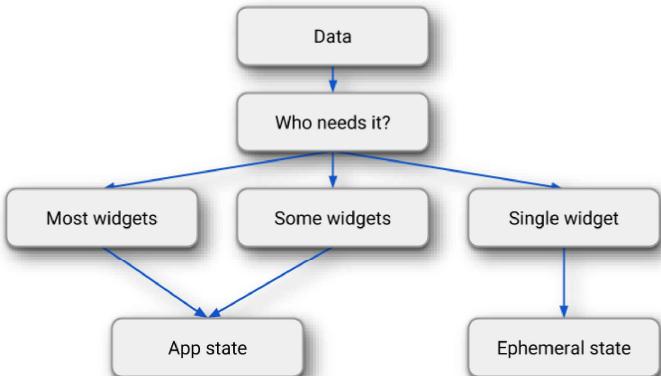
Trạng thái ứng dụng (App State)

- Là trạng thái không phải tạm thời và được chia sẻ giữa các thành phần khác nhau, các trang của ứng dụng và có thể được lưu giữ giữa các phiên sử dụng của người dùng.
- Ví dụ:
 - Sở thích của người dùng
 - Thông tin đăng nhập
 - Thông báo trong ứng dụng mạng xã hội
 - Giỏ hàng trong ứng dụng thương mại điện tử
 - Trạng thái đọc và chưa đọc của các bài báo trong một ứng dụng tin tức.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

8

Trạng thái tạm thời và trạng thái ứng dụng



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

Quản lý các trạng thái

- Trong Flutter, bạn nên giữ trạng thái trên các widget sử dụng nó vì:
 - Trong các Framework khai báo như Flutter, nếu muốn thay đổi giao diện thì phải xây dựng lại nó.
 - Khó thay đổi một widget từ bên ngoài bằng cách gọi một phương thức trên nó.** Nếu có thể làm được như vậy thì có khả năng sẽ xung đột với Flutter Framework thay vì tận dụng sự trợ giúp của Framework này.
 - Trong Flutter, **một Widget mới sẽ được tạo ra khi nội dung của nó thay đổi.** Do đó thay vì truyền trạng thái cần cập nhật cho đối số của một hàm cập nhật nào đó, **ta nên sử dụng hàm khởi tạo với đối số là trạng thái cần cập nhật.**

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

10

Ví dụ: Các trường hợp nên và không nên

```
// BAD: DO NOT DO THIS
void myTapHandler() {
    var cartWidget = somehowGetMyCartWidget();
    cartWidget.updateWith(item);
}

// BAD: DO NOT DO THIS
Widget build(BuildContext context) {
    return SomeWidget(
        // The initial state of the cart.
    );
}

void updateWith(Item item) {
    // Somehow you need to change the UI from here.
}
```

```
// GOOD
void myTapHandler(BuildContext context) {
    var cartModel = somehowGetMyCartModel(context);
    cartModel.add(item);
}

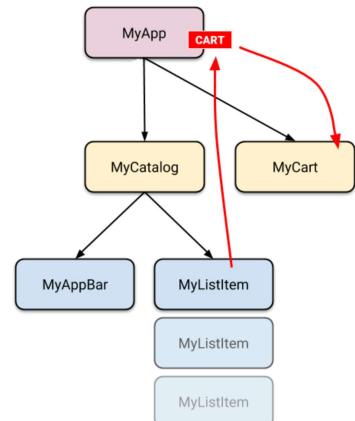
// GOOD
Widget build(BuildContext context) {
    var cartModel = somehowGetMyCartModel(context);
    return SomeWidget(
        // Just construct the UI once, using the current
        // ...
        // state of the cart.
    );
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

11

Một số cách tiếp cận

- Một số Framework để quản lý các trạng thái (<https://pub.dev/packages>):
 - Provider** (khuyên dùng)
 - setState**: Được sử dụng cho các trạng thái tạm thời của một widget cụ thể.
 - InheritedWidget & InheritedModel**: Cách tiếp cận ở mức thấp được sử dụng để giao tiếp giữa các widget tổ tiên và các widget con cháu trong cây.
 - BLoC / Rx**: Một họ các mẫu Stream/Observable



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

12

provider package



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

13

Giới thiệu chung về provider

- Một trình bao bọc quanh [InheritedWidget](#) để giúp chúng dễ sử dụng hơn và có thể tái sử dụng nhiều hơn.
- Đơn giản hóa việc phân bổ và xử lý tài nguyên.
- lazy-loading
- Khuôn mẫu để giảm việc tạo các lớp mới.
- Công cụ phát triển thân thiện.
- Một cách phổ biến để sử dụng các InheritedWidget
 - Ba khái niệm: ([ChangeNotifier](#) / [ChangeNotifierProvider](#) / [Consumer](#) hay [Selector](#))
- Tăng khả năng mở rộng cho các lớp có cơ chế lắng nghe phát triển theo độ phức tạp cấp số nhân

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

14

provider package

- Để sử dụng gói thư viện provider, cần phải khai báo nó trong tập tin pubspec.yaml.

```
dependencies:  
  provider: ^4.3.2+2
```

- Ba khái niệm trong provider:

- [ChangeNotifier](#): Cung cấp cách thức để gói trạng thái của ứng dụng.
- [ChangeNotifierProvider](#): Là một widget cung cấp một thể hiện của một [ChangeNotifier](#) cho các con của nó là các [Consumer](#).
- [Consumer](#): Sử dụng đối tượng [ChangeNotifier](#) do [ChangeNotifierProvider](#) cung cấp.
- [Selector](#): Tương đương với Consumer có thể lọc các cập nhật bằng cách chọn một số các giá trị giới hạn và *ngăn chặn rebuild* nếu chúng không thay đổi.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

15

ChangeNotifier

- Là một lớp đơn giản trong Flutter SDK cung cấp sự thông báo thay đổi cho thành phần listener của nó. Nói cách khác, nếu thứ gì đó là [ChangeNotifier](#), bạn có thể đăng ký để lắng nghe các thay đổi của nó (Observer Pattern).
- Trong provider, [ChangeNotifier](#) là một cách để đóng gói trạng thái ứng dụng của bạn. Đối với các ứng dụng phức tạp, sẽ có nhiều [ChangeNotifier](#) tương ứng với nhiều model.
- [notifyListeners\(\)](#): Gọi phương thức này khi mô hình của ứng dụng thay đổi và bạn muốn UI của ứng dụng cũng thay đổi theo.
- Cấu trúc thông thường của một lớp ChangeNotifier:
 - thuộc tính *state* (của ứng dụng)
 - Các phương thức thay đổi *state*, trong phương thức này gọi [notifyListeners\(\)](#) sau khi *state* bị thay đổi

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

16

ChangeNotifierProvider

- Là Widget cung cấp một instance của [ChangeNotifier](#) cho các con, là các [Consumer](#), của nó.
- Nơi đặt ChangeNotifierProvider trong widget tree: Phía trên widget cần truy cập đến nó. *Không nên đặt ChangeNotifierProvider quá cao trong widget tree.*
 - context của ChangeNotifierProvider phải "cao hơn" context của Consumer.

```
class CounterProviderApp extends StatelessWidget {  
    @override  
    Widget build(BuildContext context) {  
        return ChangeNotifierProvider(  
            create: (context) => Counter(),  
            child: MaterialApp(  
                title: "Provider Demo",  
                home: CounterPage(),),  
        );  
    }  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

MultiProvider

- Một Provider hợp nhất nhiều Provider thành một single linear widget tree. Nó được sử dụng để cải thiện khả năng đọc và giảm mã lệnh sẵn có của việc phải lồng nhiều lớp của các Provider.

```
void main() {  
    runApp(  
        MultiProvider(  
            providers: [  
                ChangeNotifierProvider(create: (context) => CartModel()),  
                Provider(create: (context) => SomeOtherClass()),  
            ],  
            child: MyApp(),  
        ),  
    );  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

18

Các dạng Provider

- Provider
- ChangeNotifierProvider
- ChangeNotifierProxyProvider: Một ChangeNotifierProvider build và đồng bộ một ChangeNotifier với các giá trị bên ngoài

```
ChangeNotifierProxyProvider<MyModel, MyChangeNotifier>(  
    create: (_) => MyChangeNotifier(),  
    update: (_, myModel, myNotifier) => myNotifier  
        ..update(myModel),  
    child: ...  
>;
```

- Nếu MyModel được cập nhật thì MyChangeNotifier sẽ được cập nhật một cách tự động.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

Consumer

- Sử dụng đối tượng [ChangeNotifier](#) do ChangeNotifierProvider cung cấp.
- Cần phải chỉ định kiểu của model cần truy cập.
- **builder:** là một hàm trong Consumer nhận 3 đối số và trả về một Widget (thường hiển thị trạng thái được gói trong ChangeNotifier).
 - context
 - Tham số thứ hai: là một thể hiện của ChangeNotifier
 - Tham số thứ ba, [child](#): là một subtree bên dưới Consumer không thay đổi khi model thay đổi.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

20

19

Selector

- Tương đương với Consumer có thể lọc các cập nhật bằng cách chọn một số các giá trị giới hạn và ngăn chặn rebuild nếu chúng không thay đổi.
- Selector sẽ nhận được một giá trị bằng cách sử dụng Provider.of, sau đó chuyển giá trị đó cho selector. Selector callback này sau đó có nhiệm vụ trả về một đối tượng chỉ chứa thông tin cần thiết để buider hoàn thành.
- Theo mặc định, Selector xác định xem builder có cần được gọi lại hay không bằng cách so sánh kết quả trước đó và kết quả mới của bộ chọn bằng cách sử dụng DeepCollectionEquality từ bộ package selection.
 - Việc này có thể được ghi đè bằng cách viết callback cho tham số shouldRebuild

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

21

Selector

- Để chọn nhiều giá trị mà không cần phải viết một lớp thực thi ==, giải pháp đơn giản nhất là sử dụng "Tuple" từ package tuple (<https://pub.dev/packages/tuple>):

```
Kiểu đối tượng          Kiểu giá trị được chọn trên đối tượng  
Selector<Foo, Tuple2<Bar, Baz>>(  
  selector: (_, foo) => Tuple2(foo.bar, foo.baz),  
  builder: (_, data, __) {  
    return Text('${data.item1} ${data.item2}');  
  }  
)  
Đối tượng               Dữ liệu được chọn trên đối tượng
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

22

Provider.of<T>(BuildContext context, bool listen: true)

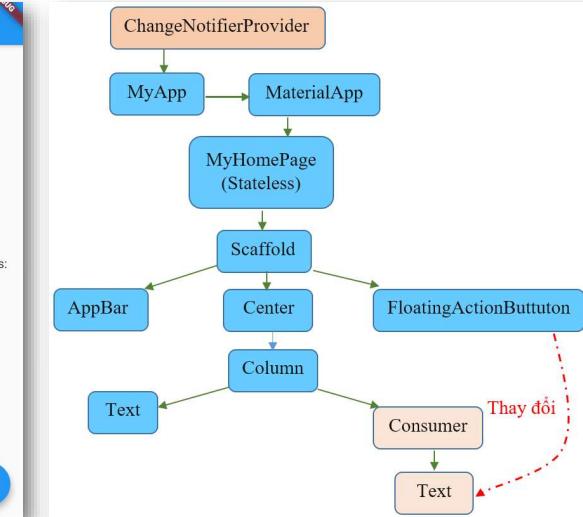
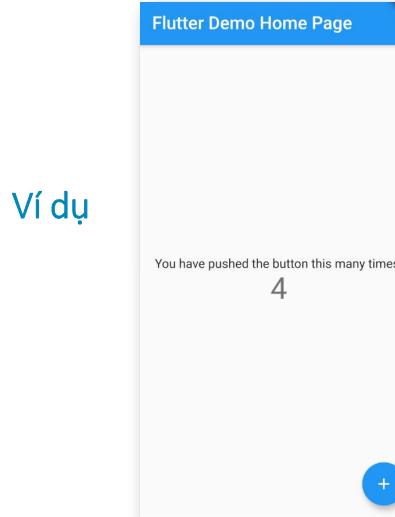
- Lấy Provide <T> gần nhất ở trên widget tree của nó và trả về giá trị của nó.
- listen: true: các thay đổi giá trị sau này sẽ kích hoạt State.build mới cho các widget và State.didChangeDependencies cho StatefulWidget.
- Được sử dụng khi không cần phải thay đổi UI khi dữ liệu trong mô hình thay đổi (listen:false), và người dùng vẫn cần truy cập dữ liệu. Ví dụ: nút ClearCart cho phép người dùng xóa mọi thứ ra khỏi giỏ hàng và không cần hiển thị nội dung của giỏ hàng, chỉ cần gọi phương thức clear () .
- Sử dụng Consumer trong trường hợp này có thể gây lãng phí bởi vì chúng ta sẽ yêu cầu framework rebuild lại những widget mà không cần rebuild.
- Trong trường hợp này, ta có thể sử dụng Provider.of

```
Provider.of<CartModel>(context, listen: false).removeAll();
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

23

Ví dụ



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

24

Ví dụ: Counter App viết theo Provider

- Sử dụng ChangeNotifier để gói trạng thái của ứng dụng

```
class Counter extends ChangeNotifier {
    int value = 0;
    int get value => _value;
    void increment() {
        _value++;
        notifyListeners();
    }
}
```

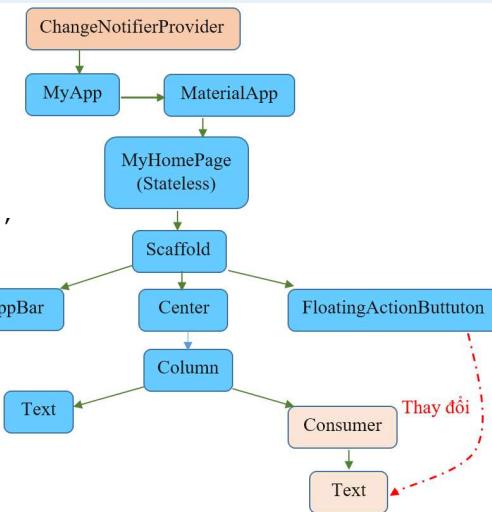
Trạng thái của ứng dụng

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

ChangeNotifierProvider

```
void main() {
    runApp(
        ChangeNotifierProvider(
            create: (context)=>Counter(),
            child: MyApp()));
}
```

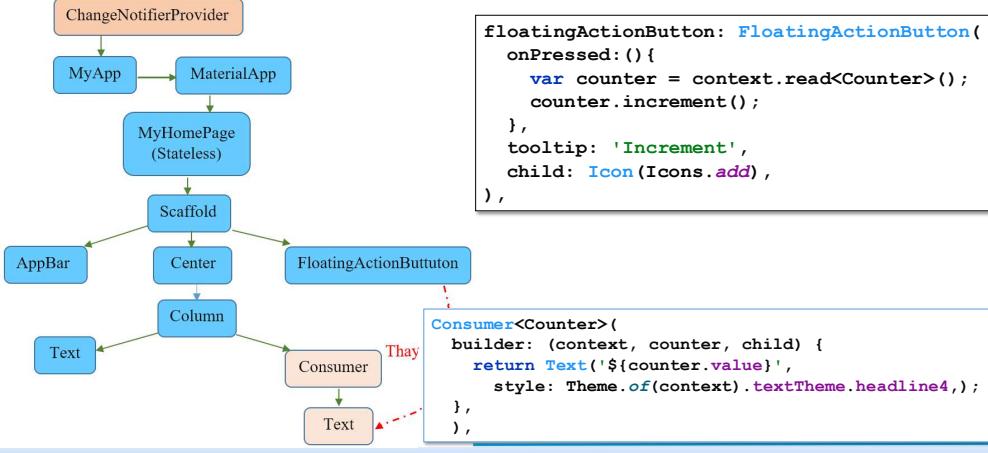
Cung cấp model cho tất cả các Widget trong ứng dụng



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

26

Consumer



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

Consumer

- child:** Tham số dùng để tối ưu hóa. Tham số này dùng để "chứa" các widget không cần phải rebuild khi state của ứng dụng thay đổi

```
Consumer<Counter>(
    builder: (context, counter, child) => Column(
        children: [
            Text('${counter.value}'),
            style: Theme.of(context).textTheme.headline4,
            child,
        ],
        child: Text("This is not rebuild widget"),
),
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

28

Selector

```
Selector<Counter, int>(
    selector: (context, data)=> data.value,
    shouldRebuild: (pre, next)=>next<=10 ,
    builder: (context,counter,child)=>Column(
        children: [
            Text(
                '$counter',
                style: Theme.of(context).textTheme.headline4
            ),
            child,
        ],
        child: Text("This is not rebuild widget"),
    )
)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

ChangeNotifierProvider

- Nếu cần cung cấp nhiều hơn một provider, sử dụng MultiProvider

```
void main() {
    runApp(
        MultiProvider(
            providers: [
                ChangeNotifierProvider(create: (context)=>Counter() ,),
                //Other Providers
            ],
            child: MyApp())));
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

30

Các extension trên BuildContext của Provider

WatchContext: phương thức: T watch<T>()

- Nhận một giá trị từ một provider tổ tiên gần nhất thuộc loại [T] và đã đăng ký với provider.
- Gọi phương thức này tương đương với gọi pt Provider.of<T>(context).
- Phương thức này chỉ được truy cập bên trong StatelessWidget.build và State.build. Nếu truy cập bên ngoài phạm vi các phương thức build này, sử dụng Provider.of để thay thế và sẽ không có các hạn chế này.
- Nếu giá trị nhận được thay đổi (phương thức notifyListeners() được gọi) thì widget sẽ được rebuild
- Không được gọi bên trong các phương thức build nếu các giá trị chỉ được sử dụng cho các events

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

Các extension trên BuildContext của Provider

SelectContext: Phương thức: R select<T, R>(R selector(T value))

- Xem (watch) một giá trị kiểu T từ một provider và chỉ lắng nghe một phần (dạng R) các thay đổi.
- select phải được sử dụng bên trong phương thức build của một widget, Nó sẽ không hoạt động trong các pt vòng đời khác, bao gồm State.didChangeDependencies.
- Khi sử dụng select, thay vì xem toàn bộ đối tượng, trình lắng nghe sẽ chỉ rebuild nếu giá trị được trả về bởi selector thay đổi.
- Khi một provider phát bản cập nhật, nó sẽ gọi đồng bộ tất cả các selector. Sau đó, nếu chúng trả về một giá trị khác với giá trị được trả về trước đó, thì phần phụ thuộc sẽ được đánh dấu là cần rebuild.
- Các phương thức watch(); Provider.of<T>(context) không thể chỉ lắng nghe một phần đối tượng đã đăng ký với Provider.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

32

31

Các extension trên BuildContext của Provider

- SelectContext: Phương thức: R select<T, R>(R selector(T value)) tt...
 - Để xác định được một đối tượng thay đổi giá trị, các phương thức == và phương thức hashCode được sử dụng của đối tượng đó được sử dụng.
 - Ta cần override hai phương thức này trong một số trường hợp.
 - Giải pháp khác: Sử dụng gói: equatable trên pub.dev. Khi đó ta chỉ cần định nghĩa lớp extends lớp Equatable

```
class Credentials extends Equatable {  
  const Credentials({this.username, this.password});  
  final String username;  
  final String password;  
  
  @override  
  List<Object> get props => [username, password];  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

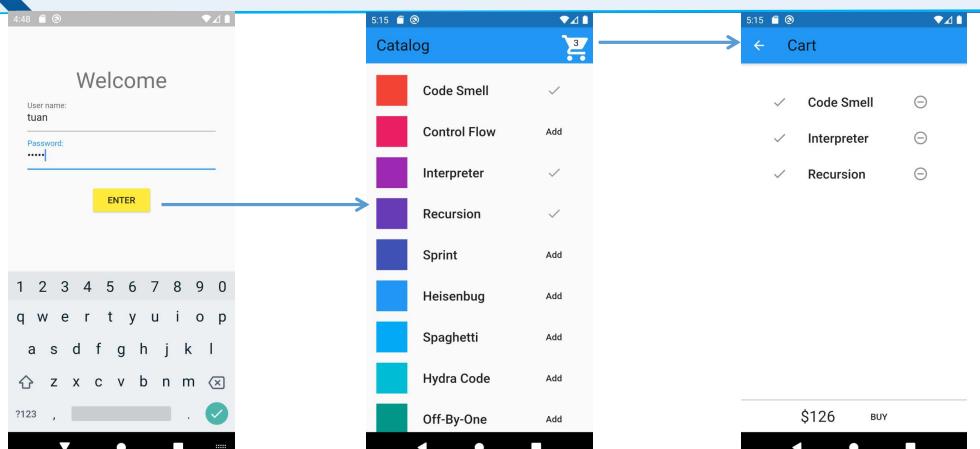
Các extension trên BuildContext của Provider

- ReadContext: Phương thức T read<T>()
 - Nhận một giá trị từ provider tổ tiên gần nhất thuộc loại [T].
 - Không làm cho widget rebuild và không được gọi bên trong các phương thức StatelessWidget.build/State.build.
 - Có thể gọi bên ngoài những phương thức này.
 - Nếu không phù hợp với điều kiện sử dụng thì có thể sử dụng Provider.of(context, listen: false) để thay thế và cho kết quả tương tự nhưng không có những hạn chế trên.
 - KHÔNG gọi read bên trong build nếu giá trị chỉ được sử dụng cho các sự kiện.
 - Có thể gọi read bên trong các event handler (ví dụ như các sự kiện onPressed)
 - KHÔNG sử dụng read để tạo widget có giá trị không bao giờ thay đổi. Thay vào đó, hãy sử dụng select để lấy ra giá trị không thay đổi có đối tượng được cung cấp bởi provider.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

34

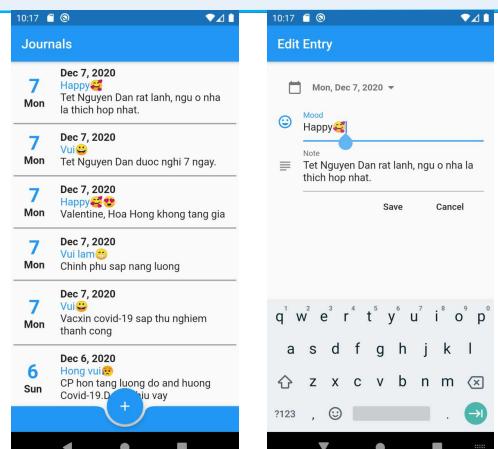
Bài tập



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

Bài tập

- Yêu cầu:
 - Dữ liệu lưu thành file .json trong thư mục của ứng dụng
 - Version 1: Sử dụng setState để quản lý trạng thái.
 - Version 2: Sử dụng Provider để quản lý trạng thái.
 - Do đọc, ghi dữ liệu sử dụng các phương thức async nên các provider được sử dụng là: MultiProvider, FutureProvider, ChangeNotifierProxyProvider



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

36

BLoC (Business Logic Component)



<https://bloclibrary.dev/>

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

37



▪ Một thư viện quản lý trạng thái có thể dự đoán giúp thực hiện mẫu thiết kế BLoC.

▪ Các package

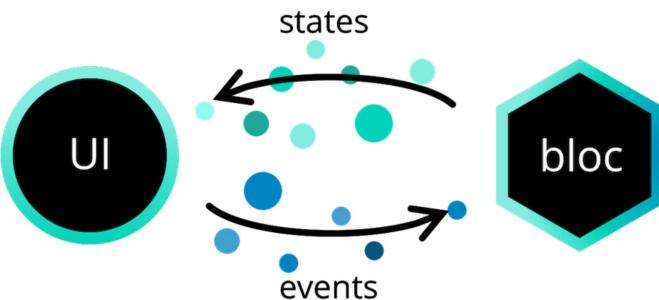
- *bloc*: Thư viện BLoC cơ bản
- *bloc_test*: Thư viện kiểm thử
- *flutter_bloc*: Các Flutter Widget mạnh mẽ được xây dựng để làm việc với bloc nhằm xây dựng nhanh các reactive mobile app.
- *angular_bloc*
- *hydrated_bloc*: Một mở rộng cho thư viện quản lý trạng thái bloc, tự động lưu giữ và khôi phục trạng thái bloc
- *replay_bloc*: Một mở rộng của thư viện quản lý trạng thái bloc, hỗ trợ thêm cho việc undo và redo.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

38



- Mục tiêu của thư viện này là giúp dễ dàng tách phần trình bày (presentation) khỏi logic nghiệp vụ (application logic), tạo điều kiện thuận lợi cho việc kiểm tra và tái sử dụng mã lệnh



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

39



▪ Đăng ký thư viện trong file pubspec.yaml

- Các khái niệm cơ bản:
 - Streams
 - Cubits
 - Bloc

dependencies:
bloc: ^6.0.0
flutter_bloc: ^6.0.0

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

40

Streams

- Stream là một chuỗi dữ liệu bất đồng bộ (**asynchronous data**).
 - Có thể hình dung một đường ống với nước chảy bên trong. Đường ống là Stream và nước chính là dữ liệu bất đồng bộ.
- Chúng ta có thể tạo một Stream bằng cách viết một hàm async*:

```
Stream<int> countStream(int max) async*{
    for(int i=0; i<max; i++)
        yield i;
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

Sử dụng Stream

```
Future<int> sumStream(Stream<int> stream) async{
    int sum = 0;
    await for(int value in stream)
        sum += value;
    return sum;
}
```

```
void main() async {
    // Tạo một Stream là chuỗi các số từ 0 - 10
    Stream<int> stream = countStream(10);
    // Tính tổng các số trong Stream
    int sum = await sumStream(stream);
    print(sum);
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

42

Cubit

- Cubit là một Stream đặc biệt được sử dụng làm cơ sở cho lớp Bloc.
- Cubit có thể đưa ra các function có thể được gọi để kích hoạt các thay đổi trạng thái.
- State là đầu ra của Cubit và biểu diễn một phần trạng thái của ứng dụng. Các thành phần của UI được thông báo và vẽ lại phần của chính chúng dựa trên trạng thái hiện tại



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

Cubit

- Tạo một cubit bằng cách mở rộng lớp Cubit và chỉ định kiểu của state.
- Ví dụ: Tạo một lớp cubit có tên CounterCubit và có kiểu của state là int.

```
class CounterCubit extends Cubit<int> {
    CounterCubit(int initialState) : super(initialState);
}
```

- Thay đổi state: Mỗi cubit có khả năng tạo ra một state mới thông qua `emit`:

```
class CounterCubit extends Cubit<int> {
    CounterCubit() : super(0);

    void increment() => emit(state + 1);
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

44

Sử dụng Cubit

- Cách sử dụng cơ bản

```
void main() {  
    final cubit = CounterCubit();  
    print(cubit.state); // 0  
    cubit.increment();  
    print(cubit.state); // 1  
    cubit.close();  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

45

- Cách sử dụng Stream

```
Future<void> main() async {  
    final cubit = CounterCubit();  
    final subscription = cubit.listenable.listen(print); // 1  
    cubit.increment();  
    await Future.delayed(Duration.zero);  
    await subscription.cancel();  
    await cubit.close();  
}
```

Hàm đăng ký để được gọi mỗi khi
trạng thái của cubit thay đổi

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

45

Observing a Cubit

- Khi một cubit emit một state mới, một đối tượng **Change** xuất hiện. Chúng ta có thể theo dõi (observe) tất cả các sự thay đổi của một Cubit bằng cách ghi đè phương thức **onChange** của nó.

Chú ý: Đối tượng Change xuất hiện chỉ trước khi state của Cubit được cập nhật. Đối tượng Change chưa currentState và nextState .

```
class CounterCubit extends Cubit<int> {  
    CounterCubit() : super(0);  
  
    void increment() => emit(state + 1);  
  
    @override  
    void onChange(Change<int> change) {  
        print(change);  
        super.onChange(change);  
    }  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

46

Observing a Cubit

```
class CounterCubit extends Cubit<int> {  
    CounterCubit() : super(0);  
  
    void increment() => emit(state + 1);  
  
    @override  
    void onChange(Change<int> change) {  
        print(change);  
        super.onChange(change);  
    }  
}
```

```
void main() {  
    CounterCubit()  
        ..increment()  
        ..close();  
}
```

Change { currentState: 0, nextState: 1 }

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

47

BlocObserver

- Nếu chúng ta muốn truy cập nhiều đối tượng **Change** ở cùng một chỗ, **BlocObserver** có thể giúp ta thực hiện công việc này.
- Trong ứng dụng, thường có nhiều Cubit quản lý các phần khác nhau của state ứng dụng.
- Nếu cần phải xử lý một công việc để hồi đáp tất cả các **Change**, chúng ta chỉ đơn giản là tạo một lớp mở rộng lớp **BlocObserver**.
 - Tất cả những gì cần thiết phải làm là ghi đè phương thức **onChange**.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

48

BlocObserver

```
class SimpleBlocObserver extends BlocObserver {  
    @override  
    void onChange(Cubit cubit, Change change) {  
        print('${cubit.runtimeType} $change');  
        super.onChange(cubit, change);  
    }  
}  
  
void main() {  
    Bloc.observer = SimpleBlocObserver();  
    CounterCubit()  
        ..increment()  
        ..close();  
}  
  
Change { currentState: 0, nextState: 1 }  
CounterCubit Change { currentState: 0, nextState: 1 }
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

49

Error Handling

- Mỗi cubit có một phương thức onError để cho biết rằng có một lỗi xuất hiện.

- onError trong mỗi Cubit chỉ xử lý lỗi cục bộ cho mỗi cubit đó.

```
@override  
void onError(Object error, StackTrace stackTrace) {  
    print('$error, $stackTrace');  
    super.onError(error, stackTrace);  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

50

Error Handling

- Để xử lý lỗi ở mức toàn cục, ta có thể ghi đè phương thức onError trong BlocObserver.

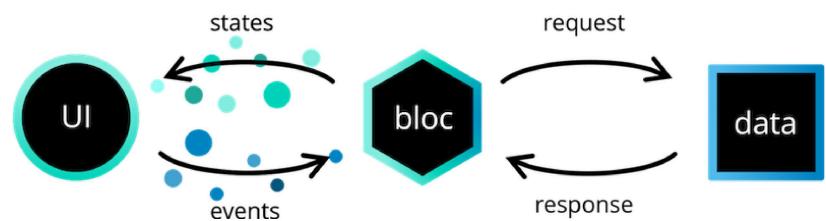
```
class SimpleBlocObserver extends BlocObserver {  
    @override  
    void onChange(Cubit cubit, Change change) {  
        print('${cubit.runtimeType} $change');  
        super.onChange(cubit, change);  
    }  
  
    @override  
    void onError(Cubit cubit, Object error, StackTrace stackTrace) {  
        print('${cubit.runtimeType} $error $stackTrace');  
        super.onError(cubit, error, stackTrace);  
    }  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

51

Bloc

- Bloc là một dạng đặc biệt của Cubit, nó chuyển đổi các sự kiện (incoming events) đến thành các trạng thái (outgoing states).



Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

52

Tạo Bloc

- Thay vì ghi đè phương thức `onChange` như trong Cubit.
Trong Bloc, chúng ta cần ghi đè phương thức `mapEventToState`

```
enum CounterEvent { increment }

class CounterBloc extends Bloc<CounterEvent, int> {
    CounterBloc() : super(0);

    @override
    Stream<int> mapEventToState(CounterEvent event) async* {
        switch (event) {
            case CounterEvent.increment:
                yield state + 1;
                break;
        }
    }
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

53

Sử dụng Bloc

Sử dụng cơ bản

```
Future<void> main() async {
    final bloc = CounterBloc();
    print(bloc.state); // 0
    bloc.add(CounterEvent.increment);
    await Future.delayed(Duration.zero);
    print(bloc.state); // 1
    await bloc.close();
}
```

Sử dụng Stream

```
Future<void> main() async {
    final bloc = CounterBloc();
    final subscription = bloc.listen(print);
    bloc.add(CounterEvent.increment);
    await Future.delayed(Duration.zero);
    await subscription.cancel();
    await bloc.close();
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

54

Observing a Bloc

- Bởi vì tất cả Bloc đều mở rộng từ Cubit (một Bloc là một Cubit), chúng ta có thể theo dõi tất cả các sự thay đổi state cho một Bloc bằng cách sử dụng `onChange`.

```
void main() {
    CounterBloc()
        ..add(CounterEvent.increment)
        ..close();
}
```

Change { currentState: 0, nextState: 1 }

```
enum CounterEvent { increment }

class CounterBloc extends Bloc<CounterEvent, int> {
    CounterBloc() : super(0);

    @override
    Stream<int> mapEventToState(CounterEvent event) async* {
        switch (event) {
            case CounterEvent.increment:
                yield state + 1;
                break;
        }
    }
}

@override
void onChange(Change<int> change) {
    print(change);
    super.onChange(change);
}
```

Huỳnh Tuâ

Transition

- Một nhân tố khác biệt chính giữa Bloc và Cubit là Bloc hoạt động theo hướng sự kiện, nên chúng ta cũng có thể nắm bắt thông tin về những gì đã kích hoạt sự thay đổi state.
 - Chúng ta thực hiện bằng cách ghi đè phương thức `onChange`
- Một thay đổi từ một state sang một state mới được gọi là transition, một transition bao gồm `state hiện tại`, `sự kiện`, và `state kế tiếp`.
- `onTransition` được gọi trước `onChange` và chứa sự kiện kích hoạt sự thay đổi từ `currentState` sang `nextState`.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

56

Transition

- Ghi đè phương thức onTransition

```
@override  
void onTransition(Transition<CounterEvent, int> transition) {  
    print(transition);  
    super.onTransition(transition);  
}  
  
void main() {  
    CounterBloc()  
        ..add(CounterEvent.increment)  
        ..close();  
}
```

```
Transition { currentState: 0, event: CounterEvent.increment, nextState: 1 }  
Change { currentState: 0, nextState: 1 }
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

BlocObserver

- Chúng ta có thể ghi đè onTransition trong một BlocObserver tùy chỉnh để theo dõi tất cả transition xuất hiện từ một nơi duy nhất.

```
class SimpleBlocObserver extends BlocObserver {  
    @override  
    void onChange(Cubit cubit, Change change) {  
        print('${cubit.runtimeType} $change');  
        super.onChange(cubit, change);  
    }  
    @override  
    void onTransition(Bloc bloc, Transition transition) {  
        print('${bloc.runtimeType} $transition');  
        super.onTransition(bloc, transition);  
    }  
}
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

58

BlocObserver

```
void main() {  
    Bloc.observer = SimpleBlocObserver();  
    CounterBloc()  
        ..add(CounterEvent.increment)  
        ..close();  
}
```

```
Transition { currentState: 0, event: CounterEvent.increment, nextState: 1 }  
CounterBloc Transition { currentState: 0, event: CounterEvent.increment, nextState: 1 }  
Change { currentState: 0, nextState: 1 }  
CounterBloc Change { currentState: 0, nextState: 1 }
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

So sánh Cubit và Bloc

- Sử dụng đơn giản: Khi tạo Cubit chỉ cần định nghĩa state và các hàm để thay đổi state.
- Không biết được sự kiện kích hoạt thay đổi state.

- Phức tạp hơn: Khi tạo Bloc, phải định nghĩa trạng thái, sự kiện và cài đặt phương thức mapEventToState.
- Biết được chuỗi các thay đổi state, sự kiện kích hoạt các chuỗi đó.

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

60

package:flutter_bloc.

- Bloc Widgets
 - BlocBuilder
 - BlocProvider
 - MultiBlocProvider
 - BlocListener
 - MultiBlocListener
 - BlocConsumer
 - RepositoryProvider
 - MultiRepositoryProvider

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

61

BlocBuilder

- Là một widget đòi hỏi một Bloc và một hàm builder. BlocBuilder xử lý việc xây dựng widget để đáp ứng một state mới.
- Nếu tham số cubit bị bỏ qua, BlocBuilder sẽ tự động thực hiện tra cứu bằng BlocProvider và BuildContext hiện tại.

```
BlocBuilder<BlocA, BlocAState>(  
    cubit: blocA, // provide the local cubit instance  
    builder: (context, state) {  
        // return widget here based on BlocA's state  
    }  
)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

62

BlocBuilder

- buildWhen: Kiểm soát khi nào hàm builder được gọi
 - Nếu trả về true: Hàm builder được gọi cùng với state và widget được vẽ lại.
 - Nếu trả về false: Hàm builder không được gọi với state và widget không được vẽ lại

```
BlocBuilder<BlocA, BlocAState>(  
    buildWhen: (previousState, state) {  
        // return true/false to determine whether or not  
        // to rebuild the widget with state  
    },  
    builder: (context, state) {  
        // return widget here based on BlocA's state  
    }  
)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

63

BlocProvider

- Là một Flutter widget cung cấp một bloc cho các con của nó thông qua `BlocProvider.of<T>(context)`. Nó được sử dụng như là một dependency injection (DI) widget vì vậy một thể hiện của một bloc có thể được cung cấp đến nhiều widget trong một cây con.
- Trong hầu hết các trường hợp, BlocProvider được sử dụng để tạo các bloc mới để cung cấp cho phần còn lại của cây con. Trong những trường hợp này, BlocProvider cũng tự động chịu trách nhiệm đóng bloc đã tạo.

```
BlocProvider(  
    create: (BuildContext context) => BlocA(),  
    child: ChildA(),  
)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

64

BlocProvider

- Trong một số trường hợp, BlocProvider có thể được sử dụng để cung cấp một bloc hiện có cho một phần của cây widget. Trường hợp này cũng được sử dụng phổ biến khi cần cung cấp một cubit hiện có cho một route mới. Trong trường hợp này, BlocProvider không được tự động đóng vì nó không tạo ra bloc.
- Context của BlocProvider phải cao hơn context của BlocBuilder

```
BlocProvider.value(  
    value: BlocProvider.of<BlocA>(context),  
    child: ScreenA(),  
)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

65

BlocProvider

```
BlocProvider.value(  
    value: BlocProvider.of<BlocA>(context),  
    child: ScreenA(),  
)
```

- Trong ScreenA (hoặc childA), chúng ta có thể truy cập BlocA như sau:

```
// with extensions  
context.bloc<BlocA>();  
  
// without extensions  
BlocProvider.of<BlocA>(context)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

66

MultiBlocProvider

- Là một Flutter widget hợp nhất nhiều BlocProvider widget thành một. MultiBlocProvider cải thiện khả năng đọc và loại bỏ sự cần thiết phải lồng nhau nhiều BlocProvider. Bằng cách sử dụng MultiBlocProvider chúng ta có thể thay:

```
BlocProvider<BlocA>(  
    create: (BuildContext context) => BlocA(),  
    child: BlocProvider<BlocB>(  
        create: (BuildContext context) => BlocB(),  
        child: BlocProvider<BlocC>(  
            create: (BuildContext context) => BlocC(),  
            child: ChildA(),  
        )  
    )  
)
```



```
MultiBlocProvider(  
    providers: [  
        BlocProvider<BlocA>(  
            create: (BuildContext context) => BlocA(),  
        ),  
        BlocProvider<BlocB>(  
            create: (BuildContext context) => BlocB(),  
        ),  
        BlocProvider<BlocC>(  
            create: (BuildContext context) => BlocC(),  
        ),  
        child: ChildA(),  
    ]  
)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

67

BlocListener

- Là một Flutter widget nhận một **BlocWidgetListener** và một **Bloc** tùy chọn và gọi **listener** này để phản ứng sự thay đổi trong **bloc**. Nó nên được sử dụng cho các chức năng cần thực hiện một lần cho mỗi sự thay đổi state như: navigation, hiển thị Snackbar, hiển thị Dialog...
- listener** chỉ được gọi một lần mỗi khi thay đổi state (không bao gồm state khởi tạo ban đầu) không giống như builder trong BlocBuilder và nó là một hàm có kiểu trả về là void.
- Nếu tham số cubit bị bỏ qua, BlocListener sẽ tự động thực hiện tra cứu bằng BlocProvider và BuildContext hiện tại

```
BlocListener<BlocA, BlocAState>(  
    listener: (context, state) {  
        // do stuff here based on BlocA's state  
    },  
    child: Container(),  
)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

68

BlocListener

- Chỉ chỉ định **bloc** nếu bạn muốn cung cấp một **bloc** không thể truy cập được thông qua BlocProvider và BuildContext hiện tại

```
BlocListener<BlocA, BlocAState>(  
  cubit: blocA,  
  listener: (context, state) {  
    // do stuff here based on BlocA's state  
  },  
  child: Container()  
)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

69

BlocListener

- Tương tự như BlocBuilder, sử dụng buildWhen để kiểm soát việc được gọi của listener

```
BlocListener<BlocA, BlocAState>(  
  listenWhen: (previousState, state) {  
    // return true/false to determine whether or not  
    // to call listener with state  
  },  
  listener: (context, state) {  
    // do stuff here based on BlocA's state  
  },  
  child: Container(),  
)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

70

MultiBlocListener

- Là một Flutter widget hợp nhất nhiều BlocListener widget thành một. MultiBlocListener cải thiện khả năng đọc và loại bỏ sự cần thiết phải lồng nhau nhiều BlocListener.

```
BlocListener<BlocA, BlocAState>(  
  listener: (context, state) {},  
  child: BlocListener<BlocB, BlocBState>(  
    listener: (context, state) {},  
    child: BlocListener<BlocC, BlocCState>(  
      listener: (context, state) {},  
      child: ChildA(),  
    ),  
  ),  
)
```



```
MultiBlocListener(  
  listeners: [  
    BlocListener<BlocA, BlocAState>(  
      listener: (context, state) {},  
    ),  
    BlocListener<BlocB, BlocBState>(  
      listener: (context, state) {},  
    ),  
    BlocListener<BlocC, BlocCState>(  
      listener: (context, state) {},  
    ),  
  ],  
  child: ChildA(),  
)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

71

BlocConsumer

- BlocConsumer cung cấp một builder và listener để phản ứng với các state mới. BlocConsumer tương tự như BlocListener và BlocBuilder lồng vào nhau nhưng làm giảm số lượng boilerplate cần thiết. BlocConsumer nên được sử dụng khi cần xây dựng lại giao diện người dùng và đồng thời thực hiện các phản ứng khác đối với các thay đổi state trong cubit.

```
BlocConsumer<BlocA, BlocAState>(  
  listener: (context, state) {  
    // do stuff here based on BlocA's state  
  },  
  builder: (context, state) {  
    // return widget here based on BlocA's state  
  }  
)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

72

BlocConsumer

```
BlocConsumer<BlocA, BlocAState>(
  listenWhen: (previous, current) {
    // return true/false to determine whether or not
    // to invoke listener with state
  },
  listener: (context, state) {
    // do stuff here based on BlocA's state
  },
  buildWhen: (previous, current) {
    // return true/false to determine whether or not
    // to rebuild the widget with state
  },
  builder: (context, state) {
    // return widget here based on BlocA's state
  }
)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

RepositoryProvider

- Là một widget Flutter cung cấp một kho lưu trữ cho con của nó thông qua RepositoryProvider.of <T> (context). Nó được sử dụng như một dependency injection (DI) widget để một thể hiện của kho lưu trữ có thể được cung cấp cho nhiều widget trong một cây con. BlocProvider nên được sử dụng để cung cấp các Bloc trong khi RepositoryProvider chỉ nên được sử dụng cho các kho lưu trữ.

```
RepositoryProvider(
  create: (context) => RepositoryA(),
  child: ChildA(),
);
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

74

RepositoryProvider

- Từ ChildA, chúng ta có thể truy xuất tới thể hiện của kho lưu trữ:

```
// with extensions
context.repository<RepositoryA>();

// without extensions
RepositoryProvider.of<RepositoryA>(context)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

MultiRepositoryProvider

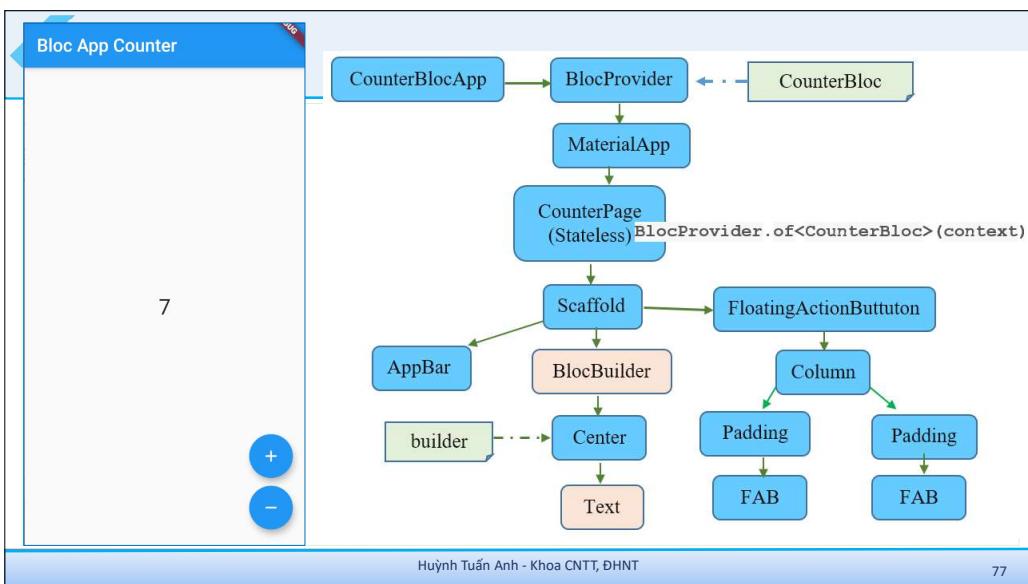
```
RepositoryProvider<RepositoryA>(
  create: (context) => RepositoryA(),
  child: RepositoryProvider<RepositoryB>(
    create: (context) => RepositoryB(),
    child: RepositoryProvider<RepositoryC>(
      create: (context) => RepositoryC(),
      child: ChildA(),
    )
  )
)
```

```
MultiRepositoryProvider(
  providers: [
    RepositoryProvider<RepositoryA>(
      create: (context) => RepositoryA(),
    ),
    RepositoryProvider<RepositoryB>(
      create: (context) => RepositoryB(),
    ),
    RepositoryProvider<RepositoryC>(
      create: (context) => RepositoryC(),
    ),
  ],
  child: ChildA(),
)
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

76

75



Code

```

enum CounterEvents {
    increment, decrement
}

class CounterBloc extends Bloc<CounterEvents, int> {

    CounterBloc(int state):super(state);

    @override
    Stream<int> mapEventToState(CounterEvents event) async* {
        switch(event) {
            case CounterEvents.increment:
                yield state+1;
                break;
            case CounterEvents.decrement:
                yield state -1;
                break;
        }
    }
}
  
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

78

main()

```

void main() {
    runApp(MyApp());
}

class MyApp extends StatelessWidget {
    @override
    Widget build(BuildContext context) {
        return MaterialApp(
            title: 'Flutter Demo',
            theme: ThemeData(
                primarySwatch: Colors.blue,
                visualDensity: VisualDensity.adaptivePlatformDensity,
            ),
            home: BlocProvider(
                create: (context) => CounterBloc(0),
                child: CounterPage(),
            ),
        );
    }
}
  
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

79

CounterPage

```

Widget build(BuildContext context) {
    // ignore: close_sinks
    final CounterBloc _counterBloc = BlocProvider.of<CounterBloc>(context);
    return Scaffold(
        appBar: AppBar(
            title: Text('Bloc App Counter'),
        ),
        body: BlocBuilder<CounterBloc, int>(
            builder: (context, count){
                return Center(
                    child: Text(
                        '$count',
                        style: TextStyle(fontSize: 24),
                    ),
                );
            },
        ),
    );
}
  
```

Huỳnh Tuấn Anh - Khoa CNTT, ĐHNT

80

CounterPage: FloatingActionButton

```
floatingActionButton: Column(  
  mainAxisAlignment: MainAxisAlignment.end,  
  mainAxisSize: MainAxisSize.end,  
  children: [  
    Padding(  
      padding: EdgeInsets.symmetric(vertical: 5.0),  
      child: FloatingActionButton(  
        child: Icon(Icons.add),  
        onPressed: () {  
          _counterBloc.add(CounterEvents.increment);  
        },  
      ),  
    ),  
    Padding(...)  
  ],
```

Data and Backend



Huỳnh Tuấn Anh - ĐHNT

1

Shared Preferences



Huỳnh Tuấn Anh - ĐHNT

3



Nội dung

- Shared Preferences
- Read and Write Files
- SQLite
- JSON and Serialization
- Firebase

Huỳnh Tuấn Anh - ĐHNT

2



Shared preferences plugin

- Plugin đa nền tảng hỗ trợ lưu trữ dữ liệu đơn giản: NSUserDefaults trên iOS và macOS, SharedPreferences trên Android.
- Dữ liệu được ghi vào đĩa không đồng bộ: Không đảm bảo dữ liệu sẽ được ghi sau khi thao tác ghi được trả về, vì vậy nên cẩn thận khi sử dụng plugin này để ghi các dữ liệu quan trọng của ứng dụng.
- Sử dụng:
 - Khai báo dependencies pubspec.yaml
 - import package:

dependencies:
shared_preferences:

```
import 'package:shared_preferences/shared_preferences.dart';
```

Huỳnh Tuấn Anh - ĐHNT

4

Ví dụ: Counter App: class _MyHomePageState

```
@override  
void initState() {  
    _getSavedCounter();  
}  
  
void _getSavedCounter () async{  
    SharedPreferences sharedpreferences = await SharedPreferences.getInstance();  
    _counter = (sharedpreferences.getInt('counter') ?? 0);  
    setState(() {  
        _counter++;  
    });  
    SharedPreferences sharedpreferences = await SharedPreferences.getInstance();  
    await sharedpreferences.setInt('counter', _counter);  
}
```

Huỳnh Tuấn Anh - DHNT

Methods

- **Static:**
 - getInstance() → Future<SharedPreferences>
- **clear() → Future<bool>:** Xóa User Preferences và trả về giá trị true khi hoàn tất.
- **containsKey(String key) → bool**
- **get(String key) → dynamic**
- **getBool(String key) → bool**
- **getDouble(String key) → double**
- **getInt(String key) → int**
- **getKeys() → Set<String>:** Trả về tất cả các key được lưu trữ

Huỳnh Tuấn Anh - DHNT

6

Methods

- **getString(String key) → String**
- **getStringList(String key) → List<String>**
- **reload() → Future<void>:** Nạp giá trị sau cùng từ nền tảng lưu trữ
- **remove(String key) → Future<bool>**
- **setBool(String key, bool value) → Future<bool>**
- **setDouble(String key, double value) → Future<bool>**
- **setInt(String key, int value) → Future<bool>**
- **setString(String key, String value) → Future<bool>**
- **setStringList(String key, List<String> value) → Future<bool>**

Huỳnh Tuấn Anh - DHNT

7

Read and Write Files



Huỳnh Tuấn Anh - DHNT

8

Recipe

1. Xác định chính đường dẫn cục bộ.
2. Tạo một tham chiếu đến vị trí tệp.
3. Ghi dữ liệu vào tệp.
4. Đọc dữ liệu từ tệp.

Huỳnh Tuấn Anh - DHNT

9

Xác định đường dẫn cục bộ

- Sử dụng `path_provider` (`pub.dev`): Hỗ trợ quyền truy cập vào hai vị trí hệ thống tệp:

- *Temporary directory*: Thư mục tạm thời (catch) mà hệ thống có thể xóa bất kỳ lúc nào.

```
Future<String> get _tempDirPath async{
  final temp = await getTemporaryDirectory();
  return temp.path;
}
```

- *Documents directory*: Một thư mục dành riêng cho app mà chỉ nó mới có thể truy cập. Hệ thống chỉ có thể xóa thư mục này nếu app bị xóa.

```
Future<String> get _docDirPath async{
  final directory = await getApplicationDocumentsDirectory();
  return directory.path;
}
```

Huỳnh Tuấn Anh - DHNT

10

Tạo một tham chiếu đến vị trí tệp

- Sử dụng lớp `File` trong thư viện `dart:io`.

```
Future<File> getLocalFile(String fileName) async{
  String docPath = await _docDirPath;
  return File('$docPath/$fileName');
}
```

Huỳnh Tuấn Anh - DHNT

11

Ghi dữ liệu vào file

- Tạo tham chiếu đến đối tượng `file`

- Ghi dữ liệu vào file, sử dụng các phương thức:

- `Future<File> writeAsString(String contents, { FileMode mode: FileMode.write, Encoding encoding: utf8, bool flush: false })`;

- `FileMode`: write, append

- `void writeAsStringSync(String contents, { FileMode mode: FileMode.write, Encoding encoding: utf8, bool flush: false })`;

- file tự động đóng khi ghi xong dữ liệu

```
Future<File> writeCounter(int counter) async{
  final File file = await getLocalFile('counter.txt');
  // Write the file.
  return file.writeAsString('$counter');
}
```

Huỳnh Tuấn Anh - DHNT

12

Đọc dữ liệu từ file

- Sử dụng các phương thức của lớp File:

- Future<String> readAsString ({Encoding encoding: utf8}): Đọc toàn bộ nội dung của file
- Future<List<String>> readAsLines ({Encoding encoding: utf8}): Đọc toàn bộ nội dung file dưới dạng các dòng văn bản.
- String readAsStringSync({Encoding encoding: utf8}): Đọc toàn bộ nội dung của file một cách đồng bộ.
- List<String> readAsLinesSync({Encoding encoding: utf8}): Đọc toàn bộ nội dung file dưới dạng các dòng văn bản một cách đồng bộ.

Huỳnh Tuấn Anh - ĐHNT

13

Đọc dữ liệu từ file

- Ví dụ:

```
Future<int> readCounter(String fileName) async {
  try {
    final file = await getLocalFile(fileName);
    String counter = await file.readAsString();
    return int.parse(counter);
  } catch(e) {
    return 0;
  }
}
```

Huỳnh Tuấn Anh - ĐHNT

14

JSON and Serialization



Huỳnh Tuấn Anh - ĐHNT

15

Serializing JSON manually using dart:convert

- Sử dụng gói thư viện dart:convert:

- import 'dart:convert'
- Phương thức Map<String , dynamic> jsonDecode(String str): Chuyển một chuỗi thành một đối tượng Map (gồm các cặp Key-Value).
- Phương thức jsonEncode(): Nhận một đối tượng và chuyển đổi đối tượng đó thành chuỗi Json

```
void main() {
  var jsonString ='{
    "name": "Tuan",
    "email": "tuan@gmail.com"
  }';
  Map<String , dynamic> user = jsonDecode(jsonString);
  print("Chào ${user['name']}");
}
```

Huỳnh Tuấn Anh - ĐHNT

16

Serializing JSON inside model classes

- Cài đặt class có các phương thức sau:
 - Named constructor dùng để khởi tạo một thể hiện của lớp từ cấu trúc Map. Ví dụ: User.fromJson()
 - Phương thức toJson(): chuyển một thể hiện của lớp thành một Map.

```
class User {  
    final String name;  
    final String email;  
  
    User(this.name, this.email);  
  
    User.fromJson(Map<String, dynamic> json)  
        : name = json['name'],  
          email = json['email'];  
  
    Map<String, dynamic> toJson() =>  
    {  
        'name': name,  
        'email': email,  
    };  
}
```

Huỳnh Tuấn Anh - DHNT

17

Serializing JSON inside model classes

```
void main() {  
    var jsonString ='{'  
        '"name":"Tuan",'  
        '"email":"tuan@gmail.com"}';  
    Map<String, dynamic> userMap = jsonDecode(jsonString);  
    User user = User.fromJson(userMap);  
    print("Chào ${user.name}");  
}
```

- Để chuyển một User thành một chuỗi Json, không cần phải gọi phương thức toJson(), phương thức jsonEncode() đã làm công việc này:

- String json = jsonEncode(user);

Huỳnh Tuấn Anh - DHNT

18

JSON Array to List Object

▪ B1: Serializing JSON inside model classes

- Ví dụ: Định nghĩa lớp Photo với 2 phương thức:
 - Constructor Photo.fromJson(Map<String, dynamic> json)
 - Phương thức toJson() trả về một Map<String, dynamic>

▪ B2: Sử dụng phương thức json.decode để trả về một List các object

- Ví dụ: jsonString là một chuỗi biểu diễn một mảng Array các photo

```
List<Photo> photos;  
photos = (json.decode(jsonString) as List).map((item) =>  
    Photo.fromJson(item)).toList();
```

Huỳnh Tuấn Anh - DHNT

19

Ví dụ:

- Tại địa chỉ: <https://jsonplaceholder.typicode.com/photos> chứa một Json Array

```
[  
    {  
        "albumId": 1,  
        "id": 1,  
        "title": "accusamus beatae ad facilis cum similique qui sunt",  
        "url": "https://via.placeholder.com/600/92c952",  
        "thumbnailUrl": "https://via.placeholder.com/150/92c952"  
    },  
    {  
        "albumId": 1,  
        "id": 2,  
        "title": "reprehenderit est deserunt velit ipsam",  
        "url": "https://via.placeholder.com/600/771796",  
        "thumbnailUrl": "https://via.placeholder.com/150/771796"  
    },
```

Huỳnh Tuấn Anh - DHNT

20

Ví dụ:

Đọc chuỗi Json Array này, phân tích và hiển thị trên màn hình của ứng dụng:

Các thư viện sử dụng:

import 'package:http/http.dart' as http;



Huỳnh Tuấn Anh - DHNT

class Photo

```
class Photo{
    final int albumId;
    final int id;
    final String title;
    final String url;
    final String thumbnailUrl;

    Photo({this.albumId, this.id, this.title, this.url, this.thumbnailUrl});

    factory Photo.fromJson(Map<String, dynamic> json){
        return Photo(
            albumId : json['albumId'] as int,
            id : json['id'] as int,
            title : json['title'] as String,
            url : json['url'] as String,
            thumbnailUrl : json['thumbnailUrl'] as String);
    }
}
```

Phương thức cần thiết để Decode một chuỗi Json thành một đối tượng Photo

Huỳnh Tuấn Anh - DHNT

22

Chuyển Json Array thành List<Photo>

```
Future<List<Photo>> fetchPhotos() async{
    final response = await http.get('https://jsonplaceholder.typicode.com/photos');
    if(response.statusCode==200)
    {
        List<Photo> photos;
        var list = json.decode(response.body) as List;
        photos = list.map((item) => Photo.fromJson(item)).toList();
        return photos;
    }
    else{
        print("Không tải được Album");
        throw Exception("Khong tai duoc Album");
    }
}
```

Chuyển chuỗi Json Array thành danh sách các đối tượng Photo

Huỳnh Tuấn Anh - DHNT

23

PhotoPage: StatefulWidget

```
class _PhotosPageState extends State<PhotosPage> {
    Future<List<Photo>> photos;
    @override
    void initState() {
        super.initState();
        photos = fetchPhotos();
    }
    @override
    Widget build(BuildContext context) {
        return Scaffold(
            appBar: AppBar(...), // AppBar
            body: FutureBuilder<List<Photo>>(...), // FutureBuilder
        ); // Scaffold
    }
}
```

Huỳnh Tuấn Anh - DHNT

24



▪ FutureBuilder: Widget làm việc với dữ liệu không đồng bộ

- future: Đối tượng dữ liệu không đồng bộ (Async)
- builder: Phương thức trả về một Widget hiển thị dữ liệu không đồng bộ

```
body: FutureBuilder<List<Photo>>(
  future: photos,
  builder: (context, snapshot) {
    if(snapshot.hasError) {
      print("Lỗi xảy ra");
      return Text("Lỗi xảy ra");
    }
    return snapshot.hasData
      ? Photolist(photos: snapshot.data,)
      : Center(child: CircularProgressIndicator(),);
  },
),
```

Huỳnh Tuấn Anh - ĐHNT



PhotoList

```
class PhotoList extends StatelessWidget {
  List<Photo> photos;
  PhotoList({Key key, this.photos}):super(key:key);
  @override
  Widget build(BuildContext context) {
    return GridView.extent(
      maxCrossAxisExtent: 200, padding: EdgeInsets.all((5)),
      mainAxisSpacing: 5, crossAxisSpacing: 5,
      children: List.generate(photos.length, (index) => Container(
        decoration: BoxDecoration(
          border: Border.all(color: Colors.blue),
        ),
        child: Image.network(photos[index].thumbnailUrl),
      )));
  }
}
```

Huỳnh Tuấn Anh - ĐHNT

26

25

SQLite



Huỳnh Tuấn Anh - ĐHNT



SQLite

▪ Plugin hỗ trợ SQLite trong flutter

- sqflite: <https://pub.dev/packages/sqflite>

- Cài đặt:

```
dependencies:
  ...
  sqflite: ^1.3.0
```

- Import: import 'package:sqflite/sqflite.dart';

- SV tìm hiểu thêm về sqflite online

Huỳnh Tuấn Anh - ĐHNT

28

27

SQLite database

- Cơ sở dữ liệu quan hệ rút gọn thường dùng trong các hệ thống mobile
- Cơ sở dữ liệu SQLite trong mobile là một tệp trong hệ thống tệp được xác định bằng một đường dẫn.
 - Phương thức: getDatabasesPath() trả về đường dẫn thư mục cơ sở dữ liệu mặc định trên Android và thư mục tài liệu trên iOS
- Các bước làm việc với SQLite database
 - Thiết kế mô hình dữ liệu
 - Mở cơ sở dữ liệu
 - Thực hiện câu truy vấn tạo bảng
 - Sử dụng cơ sở dữ liệu (thực hiện truy vấn CRUD)

Huỳnh Tuấn Anh - DHNT

29

Open Database

```
// Get a Location using getDatabasesPath
var databasesPath = await getDatabasesPath();
String path = databasesPath + '/demo.db';
// Delete the database
// await deleteDatabase(path);
// open the database
Database database = await openDatabase(path, version: 1,
onCreate: (Database db, int version) async {
    // When creating the db, create the table
    await db.execute(
        'CREATE TABLE Test (id INTEGER PRIMARY KEY, name TEXT, value INTEGER, num REAL)');
    db.execute(
        'CREATE TABLE Users (id INTEGER PRIMARY KEY, name TEXT, phone TEXT, email TEXT)');
});
// database.close();
```

Nếu: Database chưa tồn tại Oncreate sẽ được gọi.

Huỳnh Tuấn Anh - DHNT

30

Chèn dữ liệu

```
await database.transaction((txn) async {
    int id1 = await txn.rawInsert(
        'INSERT INTO Test(name, value, num) VALUES("some name", 1234, 456.789)');
    print('inserted1: $id1');
    int id2 = await txn.rawInsert(
        'INSERT INTO Test(name, value, num) VALUES(?, ?, ?)',
        ['another name', 12345678, 3.1416]);
    print('inserted2: $id2');
});
```

Phương thức rawInsert trả về id của bản ghi cuối cùng được chèn vào bảng

Huỳnh Tuấn Anh - DHNT

31

Ví dụ:

```
Future<int> insert(User user) async {
    int id = await database.rawInsert(
        'INSERT INTO Users(name, phone, email) '
        'VALUES(${user.name},${user.phone},${user.email})');
    return id;
}
```

```
Future<int> insert(User user) async {
    int id = await database.transaction(
        (txn) async {
            int id = await txn.rawInsert(
                'INSERT INTO Users(name, phone, email) '
                'VALUES(${user.name},${user.phone},${user.email})');
            return id;
        });
    return id;
}
```

Huỳnh Tuấn Anh - DHNT

32

Update

```
// Update some record
int count = await database.rawUpdate(
    'UPDATE Test SET name = ?, value = ? WHERE name = ?',
    ['updated name', '9876', 'some name']);
print('updated: $count');
```

```
Future<int> update(User newUser, String oldName) async {
    int count = await database.rawUpdate(
        'UPDATE SET name = ?, phone = ?, email = ? WHERE name = ?',
        [newUser.name, newUser.phone, newUser.email, oldName]
    );
    return count;
}
```

Huỳnh Tuấn Anh - ĐHNT

33

Query

```
List<Map> list = await database.rawQuery('SELECT * FROM Test');
```

Kết quả:

```
[
    {'name': 'updated name', 'id': 1, 'value': 9876, 'num': 456.789},
    {'name': 'another name', 'id': 2, 'value': 12345678, 'num': 3.1416}
];
```

Trả về kết quả là danh sách User:

```
Future<List<User>> getUsers() async{
    List<Map> list = await database.rawQuery("SELECT * FROM Users");
    return list.map((userJson) => User.fromJson(userJson)).toList();
}
```

Huỳnh Tuấn Anh - ĐHNT

34

Firebase



Huỳnh Tuấn Anh - ĐHNT

35

Firebase

- Firebase là nền tảng phát triển ứng dụng Backend-as-a-Service (BaaS) cung cấp các dịch vụ backend được lưu trữ trên máy chủ như cơ sở dữ liệu thời gian thực, lưu trữ đám mây, xác thực, báo cáo sự cố, máy học, cấu hình từ xa và lưu trữ cho các tệp tĩnh của ứng dụng mà không cần phải duy trì máy chủ riêng cho ứng dụng.

- Thư viện sử dụng: firebase_core:

- https://pub.dev/packages/firebase_core
- Là một Flutter plugin để sử dụng Firebase Core API, cho phép kết nối với nhiều ứng dụng Firebase.
- FlutterFire: Các plugin do Google phát triển để làm việc với Firebase

Huỳnh Tuấn Anh - ĐHNT

36

Làm việc với Firebase

- Tạo dự án trên Firebase console.
- Tạo dự án Flutter trong Android Studio.
- Kết nối dự án Flutter với dự án trên Firebase (sinh viên tự tìm hiểu)
 - Android: <https://firebase.flutter.dev/docs/installation/android>
 - iOS: <https://firebase.flutter.dev/docs/installation/ios>
- Cloud Firestore: Cơ sở dữ liệu NoSQL trên Firebase
- cloud_firestore: API làm việc với csdl Cloud Firestore do Google cung cấp (pub.dev)

Huỳnh Tuấn Anh - DHNT

37

Thư viện cloud_store

- https://pub.dev/documentation/cloud_firestore/latest/cloud_firestore/cloud_firestore-library.html
- CollectionReference
 - Một đối tượng CollectionReference có thể được sử dụng để thêm các documents, lấy các DocumentReference và truy vấn các documents
- DocumentReference
 - DocumentReference tham chiếu đến một document location trong cơ sở dữ liệu FirebaseFirestore và có thể được sử dụng để write, read hoặc listen tại location đó.
 - Tài liệu tại vị trí được tham chiếu có thể tồn tại hoặc không. Một DocumentReference cũng có thể được sử dụng để tạo một CollectionReference cho một bộ sưu tập con.

Huỳnh Tuấn Anh - DHNT

38

Thư viện cloud_store

- QuerySnapshot:
 - Chứa kết quả câu truy vấn. Nó có thể chứa không hay nhiều đối tượng DocumentSnapshot.
- DocumentSnapshot
 - Một DocumentSnapshot chứa dữ liệu đọc từ một document trong CSDL FirebaseFirestore.
 - Dữ liệu có thể được trích xuất từ DocumentSnapshot bằng phương thức `data()`, trả về một Map<String, dynamic>
- QueryDocumentSnapshot: Lớp mở rộng từ DocumentSnapshot
 - Một QueryDocumentSnapshot chứa dữ liệu được đọc từ tài liệu trong cơ sở dữ liệu FirebaseFirestore của bạn như một phần của truy vấn.

Huỳnh Tuấn Anh - DHNT

39

Thư viện cloud_store

- FieldValue: Giá trị Sentinel (lính canh) có thể được sử dụng khi ghi các document field với phương thức `set()` hoặc `update()`.
- Các static method:
 - `delete() → FieldValue`
 - `increment(num value) → FieldValue`
 - `serverTimestamp() → FieldValue`
 - `arrayRemove(List elements) → FieldValue`
 - `arrayUnion(List elements) → FieldValue`

Huỳnh Tuấn Anh - DHNT

40

Sử dụng Cloud Firestore

- import:
 - `import 'package:cloud_firestore/cloud_firestore.dart';`
- Khởi tạo FlutterFire trước khi sử dụng:
 - `await Firebase.initializeApp();`
- Tạo một FireStore instance:
 - `FirebaseFirestore firestore = FirebaseFirestore.instance;`
 - Mặc định getter này sẽ trả về một default Firebase App (khi cài đặt FlutterFire trên hệ thống của bạn). Nếu muốn sử dụng một Firestore với một Firebase App thứ hai, sử dụng phương thức `instanceFor` :
 - `FirebaseApp secondaryApp = Firebase.app('SecondaryApp');`
 - `FirebaseFirestore firestore = FirebaseFirestore.instanceFor(app: secondaryApp);`

Huỳnh Tuấn Anh - DHNT

41

Sử dụng Cloud Firestore: Collections & Documents

- Để làm việc với một Collection (gần giống với bảng trong CSDL quan hệ), sử dụng một đối tượng `CollectionReference`:
 - `CollectionReference users = FirebaseFirestore.instance.collection('users');`
- Thêm một Document vào một Collection: Sử dụng phương thức `add` của đối tượng `CollectionReference`

```
Future<void> addUser() {  
  return users.add({  
    'full_name': "Minh Thanh",  
    'company': "ABC bakery",  
    'age': 40  
  }).then((value) => print("User Added"))  
    .catchError((error) => print("Failed to add user: $error"));  
}
```

Huỳnh Tuấn Anh - DHNT

42

Writing Data

- Thêm một Document mới vào một Collection: Sử dụng phương thức `add` với tham số là một `Map<String, dynamic>` trên đối tượng `CollectionReference`
 - Phương thức `add` trả về một đối tượng `Future<DocumentReference>`
 - **Document được thêm vào có ID được Firestore sinh ra một cách tự động**

```
Future<DocumentReference> addDocument() async{  
  CollectionReference users = FirebaseFirestore.instance.  
    collection('users');  
  
  var doc= await users.add({  
    'full_name': 'Minh Thanh',  
    'company': 'ABC Bakery',  
    'age': 40  
  });  
  return doc;  
}
```

Huỳnh Tuấn Anh - DHNT

43

Writing Data

- Thêm một Document có Id do người dùng chỉ định: Sử dụng phương thức `set` của đối tượng `DocumentReference` với tham số là một `Map<String, dynamic>`
 - Nếu Document có Id đã tồn tại, document này sẽ bị ghi đè

```
Future<void> setDocument(String id) async{  
  CollectionReference users = FirebaseFirestore.instance.  
    collection('users');  
  var doc= await users.doc(id).set({  
    'full_name': 'Minh Thanh',  
    'company': 'ABC Bakery',  
    'age': 40  
  });  
  return doc;  
}
```

Huỳnh Tuấn Anh - DHNT

44

Writting Data

- Update Document: Sử dụng phương thức `update` của đối tượng `DocumentReference` với đối số là một Map<string, dynamic>

```
Future<void> updateDocument(String id) async{
    var asset_image = await rootBundle.load('assets/images/sample.jpg');
    var avatar = asset_image.buffer.asUint8List();
    CollectionReference users = FirebaseFirestore.instance.
        collection('users');
    var doc= await users.doc(id).update({
        'phone': "12345678",
        'info.address.location': GeoPoint(53.483959, -2.244644),
        'info.avatar': Blob(avatar)
    });
}
```

Huỳnh Tuấn Anh - DHNT

45

Field values

- string: Up to 1,048,487 bytes (1 MiB - 89 bytes). Only the first 1,500 bytes of the UTF-8 representation are considered by queries.
- number: integer (64-bit signed), float (64-bit double precision)
- Boolean: true, false
- map: {key:value...}. Được sắp xếp theo key. VD: {a: "foo", b: "bar", c: "qux"}.
- array: mảng không chứa mảng khác. VD: [1, 2, 3, 1]
- null
- geopoint: By latitude, then longitude

Huỳnh Tuấn Anh - DHNT

46

Field values

- reference: By path elements (collection, document ID, collection, document ID...)
 - VD: `projects/[PROJECT_ID]/databases/[DATABASE_ID]/documents/[DOCUMENT_PATH]`.
- Byte: Up to 1,048,487 bytes (1 MiB - 89 bytes). Only the first 1,500 bytes are considered by queries.
- Date and time: When stored in Cloud Firestore, precise only to microseconds; any additional precision is rounded down.

Huỳnh Tuấn Anh - DHNT

47

Delete document

1. Lấy đối tượng `DocumentReference`, `docRef`, của Document cần xóa
2. Gọi phương thức `delete` trên đối tượng `docRef`

```
Future<bool> deleteUser(String id) async{
    CollectionReference users = FirebaseFirestore.instance.
        collection('users');

    try {
        await users.doc(id).delete();
        return true;
    } catch(e) {
        return false;
    }
}
```

Huỳnh Tuấn Anh - DHNT

48

Delete field

- Gọi phương thức delete trên lớp FieldValue

```
Future<bool> deleteField() async{
    CollectionReference users = FirebaseFirestore.instance.
        collection('users');

    try{
        await users.doc('Id123').update({
            'phone':FieldValue.delete(),
        });
        return true;
    }catch(e){
        return false;
    }
}
```

Huỳnh Tuấn Anh - DHNT

Transactions

▪ Transaction là một cách để đảm bảo rằng hoạt động ghi chỉ xảy ra bằng cách sử dụng dữ liệu mới nhất có sẵn trên máy chủ. Các giao dịch không bao giờ áp dụng ghi một phần và hoạt động ghi thực hiện khi giao dịch kết thúc thành công.

- Khi sử dụng các transaction, hãy lưu ý rằng:

- Thao tác đọc phải đến trước khi thao tác ghi
- Giao dịch sẽ không thành công khi client offline, họ không thể sử dụng dữ liệu trong bộ nhớ cache.

- Bạn không nên trực tiếp sửa đổi trạng thái ứng dụng bên trong giao dịch, vì trình xử lý có thể thực hiện nhiều lần. Thay vào đó, bạn nên trả về một giá trị ở cuối trình xử lý, cập nhật trạng thái ứng dụng khi giao dịch đã hoàn tất.

Huỳnh Tuấn Anh - DHNT

50

Transaction

```
void transaction(){
    // Tạo một tham chiếu đến document mà transaction sẽ sử dụng.
    DocumentReference documentReference = FirebaseFirestore.instance
        .collection('users').doc('id123');
    FirebaseFirestore.instance.runTransaction((transaction) async{
        DocumentSnapshot snapshot = await transaction.get(documentReference);
        if (!snapshot.exists) {
            throw Exception("User does not exist!");
        }
        int newFollowerCount = snapshot.data()['followers'] + 1;
        transaction.update(documentReference, {'followers': newFollowerCount});
        // Return giá trị khi giao dịch hoàn tất
        return newFollowerCount;
    }).then((value) => print("Follower count updated to $value"))
        .catchError((error) => print("Failed to update user followers: $error"));
}
```

Huỳnh Tuấn Anh - DHNT

51

Batch write

▪ Firestore cho phép bạn thực hiện nhiều hoạt động ghi dưới dạng một lô (batch) duy nhất có thể chứa các kết hợp bất kỳ của các hoạt động: *set*, *update*, *delete*.

- Thực hiện:

- Tạo một thể hiện của WriteBatch thông qua phương thức batch()
- Thực hiện các hoạt động trên batch.
- Gọi commit() trên batch khi các hoạt động trên batch sẵn sàng. Phương thức commit đảm bảo các hoạt động ghi trong batch được xem là một đơn vị nguyên tử duy nhất. Ngoài ra commit còn ngăn không cho bất kỳ các hoạt động nào ở tương lai được thêm vào.

Huỳnh Tuấn Anh - DHNT

52

Batch Write

```
Future<void> batchDelete() async{
    CollectionReference users = FirebaseFirestore.
        instance.collection('users');
    WriteBatch batch = FirebaseFirestore.instance.batch();
    QuerySnapshot querySnapshot = await users.get();
    querySnapshot.docs.forEach((doc) {
        batch.delete(doc.reference);
    });
    return batch.commit();
}
```

Huỳnh Tuấn Anh - DHNT

53

Truy vấn dữ liệu: Document & Query Snapshots

- Khi thực hiện một truy vấn, Firestore trả về một QuerySnapshot hoặc một DocumentSnapshot.
- **QuerySnapshot:** QuerySnapshot được trả về từ truy vấn một Collection và cho phép bạn kiểm tra Collection, chẳng hạn như có bao nhiêu Document tồn tại bên trong nó, cấp quyền truy cập vào các Document trong Document, xem bất kỳ thay đổi nào kể từ query cuối cùng và nhiều hơn nữa.
- **DocumentSnapshot:** được trả về từ một query hoặc bằng cách truy cập trực tiếp vào Document. Ngay cả khi không có Document nào tồn tại trong cơ sở dữ liệu, một snapshot sẽ luôn được trả về.

Huỳnh Tuấn Anh - DHNT

54

QuerySnapshot

- Truy vấn Collection user và trả về đối tượng Future<QuerySnapshot>, duyệt qua các Document trong thuộc tính `docs` của đối tượng này

```
void querySnapshot() async{
    var snapshot = await FirebaseFirestore.
        instance.collection('users').get();
    snapshot.docs.forEach((doc)=>
        print(doc['first_name']));
}
```

Huỳnh Tuấn Anh - DHNT

55

Querying

- Filtering: Để lọc ra các Documents trong một Collection thỏa mãn một điều kiện nào đó, ta sử dụng phương thức where:

```
void filtering() async{
    var snapshot = await FirebaseFirestore.instance.
        collection('users').where('age', isGreaterThanOrEqualTo: 20).
        orderBy('age').limit(5).get();
    snapshot.docs.forEach((doc)=>
        print(doc['last_name']));
}
```

Huỳnh Tuấn Anh - DHNT

56

Ví dụ: Truy vấn và hiển thị một document

```
Widget build(BuildContext context) {
  CollectionReference users = FirebaseFirestore.instance.collection('users');
  return FutureBuilder<DocumentSnapshot>(
    future: users.doc('doc123').get(),
    builder: (context, snapshot) {
      if (snapshot.hasError) {
        return Text("Something went wrong");
      }
      if (snapshot.connectionState == ConnectionState.done) {
        Map<String, dynamic> data = snapshot.data.data();
        return Text("Full Name: ${data['full_name']} ${data['last_name']}");
      }
      return Text("loading");
    },
  );
}
```

Huỳnh Tuấn Anh - DHNT

57

DocumentSnapshot

- Truy vấn một Document trong Collection users và trả về đối tượng Future<DocumentSnapshot>

```
void documentSnapshot(String docId) async{
  var snapshot = await FirebaseFirestore.instance
    .collection('users').doc(docId).get();
  if(snapshot.exists) {
    Map<String, dynamic> doc = snapshot.data();
    print(doc['last_name']);
  }
}
```

- Phương thức `get(dynamic field)` có thể trả về các field lồng bên trong DocumentSnapshot:

```
dynamic last_name = snapshot.get('last_name');
```

Huỳnh Tuấn Anh - DHNT

58

Collections & Documents: Read data

- One-time Read: Để đọc một collection hay một document một lần: Gọi phương thức `Query.get` hay `DocumentReference.get`
 - Sử dụng `FutureBuilder Widget` để hỗ trợ quản lý trạng thái của request trên UI.
- Realtime changes: Sử dụng `Stream` để kết nối giữa client và `FirebaseFirestore`
 - FlutterFire cung cấp sự hỗ trợ xử lý các thay đổi trong thời gian thực đối với Collections và Documents. Một sự kiện mới được cung cấp theo yêu cầu ban đầu và mọi thay đổi tiếp theo đối với Collections /Documents mỗi khi xảy ra sự thay đổi (thêm, xóa, sửa) sẽ được tự động cập nhật trên các widget trong `StreamBuilder` (sử dụng Stream này).
 - Sử dụng phương thức `snapshots()` để lấy data Stream

```
Stream<QuerySnapshot> collectionStream = FirebaseFirestore.instance.
  collection('users').snapshots();
Stream<DocumentSnapshot> documentStream = FirebaseFirestore.instance.
  collection('users').doc('ABC123').snapshots();
```

Huỳnh Tuấn Anh - DHNT

59

Stream<QuerySnapshot>

- Sử dụng `Stream<QuerySnapshot>` kết hợp với `StreamBuilder Widget` để bind kết quả một truy vấn với các Widget được xây dựng trong `StreamBuilder`.
 - Khi dữ liệu liên quan đến `QuerySnapshot` thay đổi, các Widget sẽ tự động được rebuild để hiển thị dữ liệu mới.

▪ Ví dụ:

- `FirebaseFirestore.instance.collection("users").snapshots()`: Trả về một `Stream<QuerySnapshot>`, mỗi `QuerySnapshot` chứa toàn bộ các Document của Collection users.
- `FirebaseFirestore.instance.collection("users").where("age", isGreater Than: 30).snapshots() --> Stream<QuerySnapshot>;`

Huỳnh Tuấn Anh - DHNT

60

Stream<DocumentSnapshot>

- Tương tự như Stream<QuerySnapshot> nhưng chỉ làm việc trên mỗi Document.
- Kết hợp với StreamBuilder để bind một Document với các widget nhằm tự động rebuild để hiển thị dữ liệu khi dữ liệu thay đổi.

Ví dụ:

```
▪ FirebaseFirestore.instance.collection("journals").doc("abc").snapshots(); -->  
Stream<DocumentSnapshot>
```

Huỳnh Tuấn Anh - DHNT

Ví dụ

```
Widget build(BuildContext context) {  
    CollectionReference users = FirebaseFirestore.instance.collection('users');  
    return StreamBuilder<QuerySnapshot>(  
        stream: users.snapshots(),  
        builder: (context, snapshot) {  
            if (snapshot.hasError) {return Text('Something went wrong');}  
            if (snapshot.connectionState == ConnectionState.waiting){  
                return Text("Loading");  
            }  
            return new ListView(  
                children: snapshot.data.docs.map((DocumentSnapshot document){  
                    return new ListTile(  
                        title: new Text(document.data()['full_name']),  
                        subtitle: new Text(document.data()['company']),  
                    );  
                }).toList(),  
            );  
        },  
    );  
}
```

Huỳnh Tuấn Anh - DHNT

62

listening metadata change

- Mặc định, listeners không cập nhật nếu có sự thay đổi chỉ ảnh hưởng đến siêu dữ liệu. Nếu bạn muốn nhận các sự kiện khi siêu dữ liệu của document hay query thay đổi, bạn có thể sử dụng thuộc tính includeMetadataChanges cho phương thức snapshots:

```
FirebaseFirestore.instance  
.collection('users')  
.snapshots(includeMetadataChanges: true);
```

Huỳnh Tuấn Anh - DHNT

Access Data Offline

- Firestore cung cấp khả năng ngoại tuyến ra bên ngoài. Khi đọc và ghi dữ liệu, Firestore sử dụng *cơ sở dữ liệu cục bộ tự động đồng bộ hóa với máy chủ*. Chức năng Cloud Firestore vẫn tiếp tục khi người dùng ngoại tuyến và tự động xử lý việc di chuyển dữ liệu (migration) khi họ lấy lại kết nối.
- Chức năng này được bật theo mặc định, tuy nhiên nó có thể bị tắt nếu cần. Các cài đặt phải được đặt trước khi thực hiện bất kỳ tương tác nào với Firestore:

```
//Web  
await FirebaseFirestore.instance.enablePersistence();  
// All other platforms.  
FirebaseFirestore.instance.settings =  
    Settings(persistenceEnabled: false);
```

Huỳnh Tuấn Anh - DHNT

64

Access Data Offline

- Nếu bạn muốn xóa mọi dữ liệu duy trì (persisted data), bạn có thể gọi phương thức clearPersistence () .

```
await FirebaseFirestore.instance.clearPersistence();
```

- Gọi các phương thức để cập nhật setting hoặc xóa persistence phải được thực hiện trước khi sử dụng Firestore. Nếu được gọi sau đó, chúng sẽ có hiệu lực đối với yêu cầu tiếp theo của Firestore (ví dụ: khởi động lại ứng dụng).

Huỳnh Tuấn Anh - DHNT

65

Configure Cache Size

- Khi tính năng persistence được bật, Firestore lưu trữ mọi tài liệu để truy cập ngoại tuyến. Sau khi vượt quá kích thước bộ nhớ cache, Firestore sẽ cố gắng xóa dữ liệu cũ hơn, không sử dụng. Bạn có thể định cấu hình các kích thước bộ nhớ cache khác nhau hoặc vô hiệu hóa quá trình xóa:

```
FirebaseFirestore.instance.settings =  
    Settings(cacheSizeBytes: Settings.CACHE_SIZE_UNLIMITED);
```

- Disable and Enable Network Access

```
await FirebaseFirestore.instance.disableNetwork();
```

```
await FirebaseFirestore.instance.enableNetwork();
```

Huỳnh Tuấn Anh - DHNT

66

Tham khảo

- Firebase – Flutter:

- <https://flutter.dev/docs/development/data-and-backend/firebase>

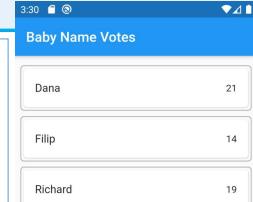
Huỳnh Tuấn Anh - DHNT

67

Ví dụ

- Thiết kế ứng dụng Baby Name Votes

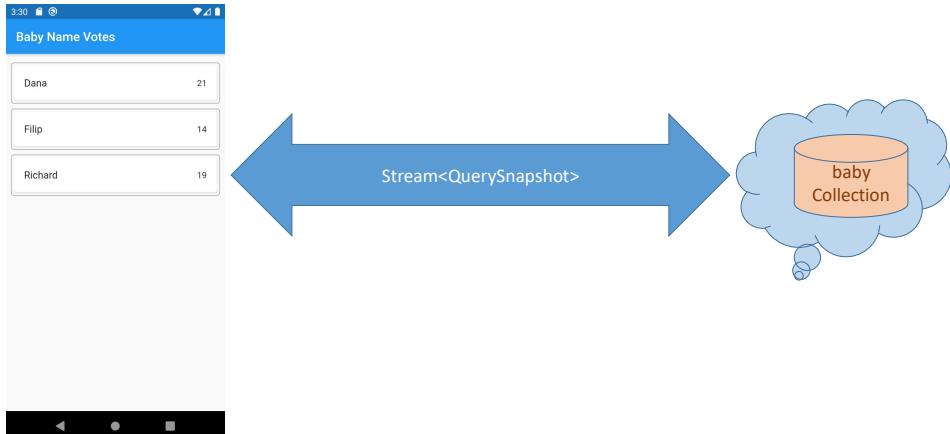
- Dữ liệu được lưu trữ trên Firebase Firestore
- Khi người sử dụng click vào tên trong danh sách số lượt vote cho tên đó sẽ tự động tăng lên một và dữ liệu sẽ lưu vào Firestore



Huỳnh Tuấn Anh - DHNT

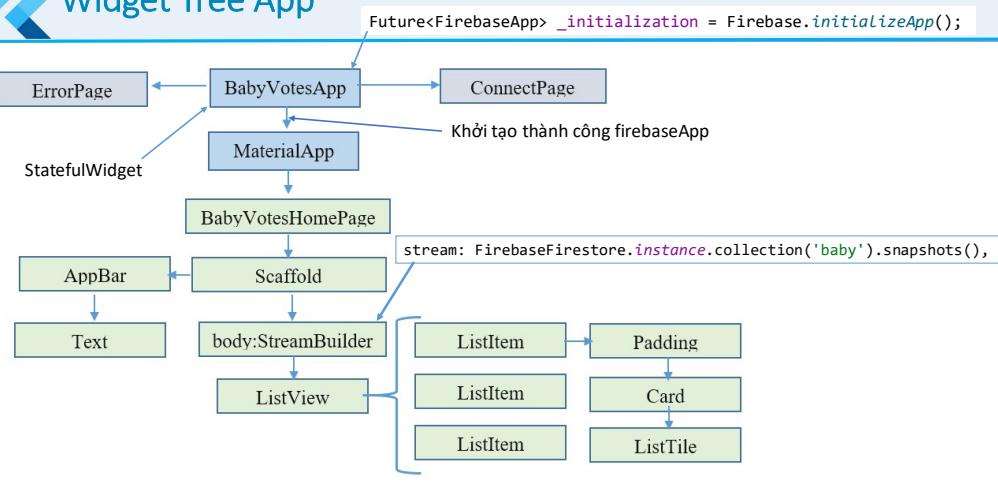
68

Streaming data



Huỳnh Tuấn Anh - DHNT

Widget Tree App



Huỳnh Tuấn Anh - DHNT

70

Record class

```
class Record{
    final String name;
    final int votes;
    final DocumentReference reference;
    Record.fromMap(Map<String, dynamic> map, {this.reference})
        : assert(map['name']!=null),
        assert(map['votes']!=null),
        name=map['name'],
        votes=map['votes'];

    Record.fromSnapshot(DocumentSnapshot snapshot)
        : this.fromMap(snapshot.data(), reference:snapshot.reference);

    @override
    String toString() => "Record<$name:$votes>";
}
```

Tham chiếu đến một Document trong Collection trên CSDL Firestore

Huỳnh Tuấn Anh - DHNT

BabyVotesHomePage

- Là một StatefulWidget

```
Widget _buildBody(BuildContext context) {
    //Firebase.initializeApp();
    return StreamBuilder<QuerySnapshot>(
        stream: FirebaseFirestore.instance.collection('baby').snapshots(),
        builder: (context, snapshot) {
            if(snapshot.hasError)
                return _buildError(context);
            else
                if(snapshot.hasData)
                    return _buildList(context, snapshot.data.docs);
            return Center(child: CircularProgressIndicator());
        },
    );
}
```

Scaffold

AppBar

body: _buildBody

Stream<QuerySnapshot>

List<QueryDocumentSnapshot>

Huỳnh Tuấn Anh - DHNT

72

```

Widget _buildList(BuildContext context, List<DocumentSnapshot> docs) {
    return ListView(
        padding: EdgeInsets.only(top: 10),
        children: List.generate(docs.length,
            (index) => _buildListItem(context, docs[index])),
    );
}

```

↓
DocumentSnapshot

Huỳnh Tuấn Anh - DHNT

73

ListItem

```

buildListItem(BuildContext context, DocumentSnapshot data) {
    final Record record = Record.fromSnapshot(data);
    return Padding(
        key: ValueKey(record.name),
        padding: const EdgeInsets.symmetric(horizontal: 8.0, vertical: 4.0),
        child: Container(
            decoration: BoxDecoration(
                border: Border.all(color: Colors.grey),
                borderRadius: BorderRadius.circular(5.0),
            ),
            child: Card(
                child: ListTile(
                    title: Text(record.name),
                    trailing: Text(record.votes.toString()),
                    onTap: () => record.reference.update({'votes':FieldValue.increment(1)}),
                ),
            ),
        );
}

```

Richard

19

Huỳnh Tuấn Anh - DHNT

74

Firebase.initializeApp()

```

class BabyVotesPage extends StatelessWidget {
    final Future<FirebaseApp> _initialization = Firebase.initializeApp();
    @override
    Widget build(BuildContext context) {
        return FutureBuilder(
            future: _initialization,
            builder:(context, snapshot){
                if(snapshot.connectionState==ConnectionState.done)
                    return BabyVotesHomePage();
                else
                    if(snapshot.hasError)
                        return ErrorPage('Lỗi kết nối!');
                    return ConnectPage("Đang kết nối");
            },
        );
    }
}

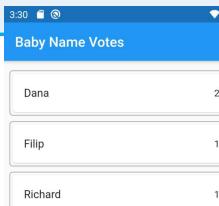
```

Huỳnh Tuấn Anh - DHNT

75

Bài tập

- Thiết kế ứng dụng Baby Name Votes sử dụng Firebase và Provider



Huỳnh Tuấn Anh - DHNT

76

Bài tập

■ Yêu cầu: Thiết kế ứng dụng với version 3:

- Dữ liệu được lưu trữ trên firebase firestore.
- Sử dụng Provider.

