TRƯỜNG ĐẠI HỌC BÁCH KHOA

KHOA ĐIỆN TỬ VIỄN THÔNG

BÁO CÁO

LẬP TRÌNH ĐA NỀN TẢNG

CHỦ ĐỀ: Future và Async/Await trong Flutter

Sinh viên thực hiện:

01. Lê Thị Kim Yến Lớp: 22KTMT1  MSSV:  106220243

02. Dương Thị Kim Ngân Lớp: 22KTMT2  MSSV:  106220263

Người hướng dẫn:

TS. Nguyễn Duy Nhật Viễn

Đà Nẵng, 20XX.

BẢNG PHÂN CÔNG CÔNG VIỆC TRONG NHÓM

|  |  |  |  |
| --- | --- | --- | --- |
| STT | HỌ VÀ TÊN | NHIỆM VỤ | KHỐI LƯỢNG |
| 01 | LÊ THỊ KIM YẾN | Tìm hiểu nội dung, làm word báo cáo | 50% |
| 02 | DƯƠNG THỊ KIM NGÂN | Tìm hiểu nội dung, làm slide thuyết trình | 50% |

Link code github: [duongthikimngan/Future\_AsyncAwait](https://github.com/duongthikimngan/Future_AsyncAwait)

Link json: [myjson.online/app/records](https://myjson.online/app/records)

Mục lục

[I. NỘI DUNG: 4](#_Toc211718425)

[1. Future: 4](#_Toc211718426)

[2. Async/ Await: 5](#_Toc211718427)

[3. Error handling với try-catch: 5](#_Toc211718428)

[4. Demo FutureBuilder widget: 6](#_Toc211718429)

[5. Combine multiple Futures với Future.wait(): 11](#_Toc211718430)

[TÀI LIỆU THAM KHẢO 13](#_Toc211718431)

1. NỘI DUNG:
2. Future:
3. Khái niệm:

* Future là một đối tượng đại diện cho một kết quả tiềm năng hoặc một lỗi sẽ có trong tương lai.

1. Trạng thái future:

* Chưa hoàn thành ( Uncompleted ): Đây là trạng thái ban đầu khi bạn bắt đầu một tác vụ bất đồng bộ. Nó đang chờ tác vụ đó hoàn tất.
* Hoàn thành (Completed): Khi tác vụ kết thúc, Future sẽ hoàn thành và mang một trong hai kết quả:
* Với một giá trị: Tác vụ đã thành công và trả về một giá trị (ví dụ: dữ liệu người dùng từ API).
* Với một lỗi: Tác vụ đã thất bại và trả về một lỗi (ví dụ: không có kết nối mạng).

1. Lợi ích khi dùng Future:

* Future là nền tảng của lập trình bất đồng bộ trong Flutter. Nó giúp ứng dụng không bị "đơ" hoặc "lag" khi đang thực hiện các tác vụ tốn thời gian. Thay vì chặn giao diện người dùng (UI) để chờ đợi, ứng dụng có thể tiếp tục chạy mượt mà và xử lý các tương tác khác của người dùng trong khi tác vụ nền đang được thực hiện.

1. Cách tạo một future:

* Trong các tác vụ bất đồng bộ biến đã có sẵn thư viện trả về Future nên phần lớn sẽ không cần tự tạo Future. Nhưng vẫn có thể tạo Future thủ công bằng các constructor sau:
* **Future()** mặc định: Nhận vào một hàm , trả về một future
* **Future.value()**: Khi đã có sẵn giá trị, hữu ích khi bạn có dữ liệu trong bộ nhớ đệm (cache).
* **Future.error()**: Tương tự Future.value nhưng cho trường hợp trả về lỗi.
* **Future.delayed()**: Giống Future mặc định nhưng chờ một khoảng thời gian trước khi thực thi hàm và hoàn thành Future.

1. Cách sử dụng Future:

* Để lấy giá trị dùng **.then()**
* Nhận vào một callback
* Khi Future hoàn thành với giá trị , callback sẽ được thực thi
* Xử lí lỗi với **.catchError()** : nếu future hoàn thành với lỗi, .then() không chạy được
* Tương tự .then() , nhưng dành cho lỗi
* Trả về một Future mới
* Có thể truyền vào một hàm kiểm tra để lọc lỗi trước khi xử lí
* Có thể viết nhiều .catchError() để xử lý các loại lỗi khác nhau.
* Phương thức **whenComplete()** : cho phép thực thi đoạn mã khi Future hoàn thành , dù thành công hay thất bại. Tương tự như khối finally trong try-catch-finally.

1. Async/ Await:
2. Từ khóa **Async**:

* Chức năng:
* Đánh dấu hàm bất đồng bộ
* Cho phép sử dụng từ khóa await trước mỗi Future mà hàm cần chờ đợi
* Đảm bảo rằng hàm sẽ trả về một Future. Ngay cả khi return một string.
* Ví dụ cú pháp:

Future<DuLieuHocSinh> LayDuLieuHocSinh() async{

Final List<String> danhSachID = await \_taiDanhSachLop();

Final String id = danhSachID[0];

Final String data = await \_taiThongTinTuID(id);

Return DuLieuHocSinh(data);

1. Từ khóa **Await** :

* Chia nhỏ các hàm thành các điểm chờ
* Được đặt trước một biểu thức trả về Future
* Chỉ được sử dụng bên trong một hàm async
* Chức năng:
* Chỉ tạm dừng việc thực thi của hàm hiện tại cho đến khi Future đó hoàn thành mà không đóng băng toàn bộ ứng dụng.
* Khi Future hoàn thành:
* Nếu Future hoàn thành thì await trả về giá trị
* Nếu Future thất bại với lỗi thì await sẽ dừng hàm và ném (throw) lỗi đó ra ngoài.

1. Error handling với try-catch:

* Một hàm có async
* Try-catch có thể bắt lỗi bất đồng bộ giống như lỗi đồng bộ
* Có thể dùng on và catch để xử lý lỗi cụ thể
* Finally vẫn chạy sau cùng
* Ví dụ:

Future<ThongTinHocSinh> layDuLieuHocSinh() async {

try {

final List<String> danhSachID = await \_taiDanhSachLop();

final String id = danhSachID[0];

final String data = await \_taiThongTinChiTiet(id);

return ThongTinHocSinh(data);

} on HttpException catch (err) {

print('Lỗi mạng: $err');

return ThongTinHocSinh.empty();

} finally {

print('Đã hoàn tất!'); }

}

1. Demo FutureBuilder widget:

* FutureBuilder là widget giúp xây dựng UI dựa trên kết quả của một Future.
* Tự động cập nhật giao diện khi Future thay đổi trạng thái: waiting, hasError, hasData.
* Giúp xử lý các trạng thái UI (loading, error, empty, content) một cách dễ dàng.
* Code demo trên máy ảo:

import 'package:flutter/material.dart';

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

import 'dart:convert';

void main() {

  runApp(const MyApp());

}

class MyApp extends StatelessWidget {

  const MyApp({super.key});

  @override

  Widget build(BuildContext context) {

    return const MaterialApp(

      debugShowCheckedModeBanner: false,

      home: FutureBuilderDemo(),

    );

  }

}

class FutureBuilderDemo extends StatelessWidget {

  const FutureBuilderDemo({super.key});

  Future<String> fetchData() async {

await Future.delayed(const Duration(seconds: 2));

    final url = Uri.parse(

      'https://api.myjson.online/v1/records/d534b8f3-20e3-4b61-8a8b-b0dcba97ecbf',

    );

    final response = await http.get(url);

    if (response.statusCode == 200) {

      final data = json.decode(response.body);

      if (data['record'] == null ||

          (data['record'] is Map && data['record'].isEmpty)) {

        return '';

      }

      return const JsonEncoder.withIndent('  ').convert(data['record']);

    } else {

      throw Exception('Lỗi server: ${response.statusCode}');

    }

  }

  @override

  Widget build(BuildContext context) {

    return Scaffold(

      appBar: AppBar(

        title: const Text('DEMO FutureBuilder',

          style: TextStyle(color: Colors.blue),

        ),

      ),

      body: Center(

        child: FutureBuilder<String>(

          future: fetchData(),

          builder: (context, snapshot) {

            if (snapshot.connectionState == ConnectionState.waiting) {

              return const CircularProgressIndicator();

            }

            else if (snapshot.hasError) {

              return Text(

                'Lỗi: ${snapshot.error}',

                style: const TextStyle(color: Colors.red),

                textAlign: TextAlign.center,

              );

            }

            else if (snapshot.hasData && snapshot.data!.isNotEmpty) {

              return SingleChildScrollView(

                padding: const EdgeInsets.all(16),

                child: Text(

                  snapshot.data!,

                  style: const TextStyle(fontSize: 16),

                ),

              );

            }

            else {

              return const Text(

                'Không có dữ liệu',

                style: TextStyle(fontSize: 20, color: Colors.purple),

              );

            }

          },

        ),

      ),

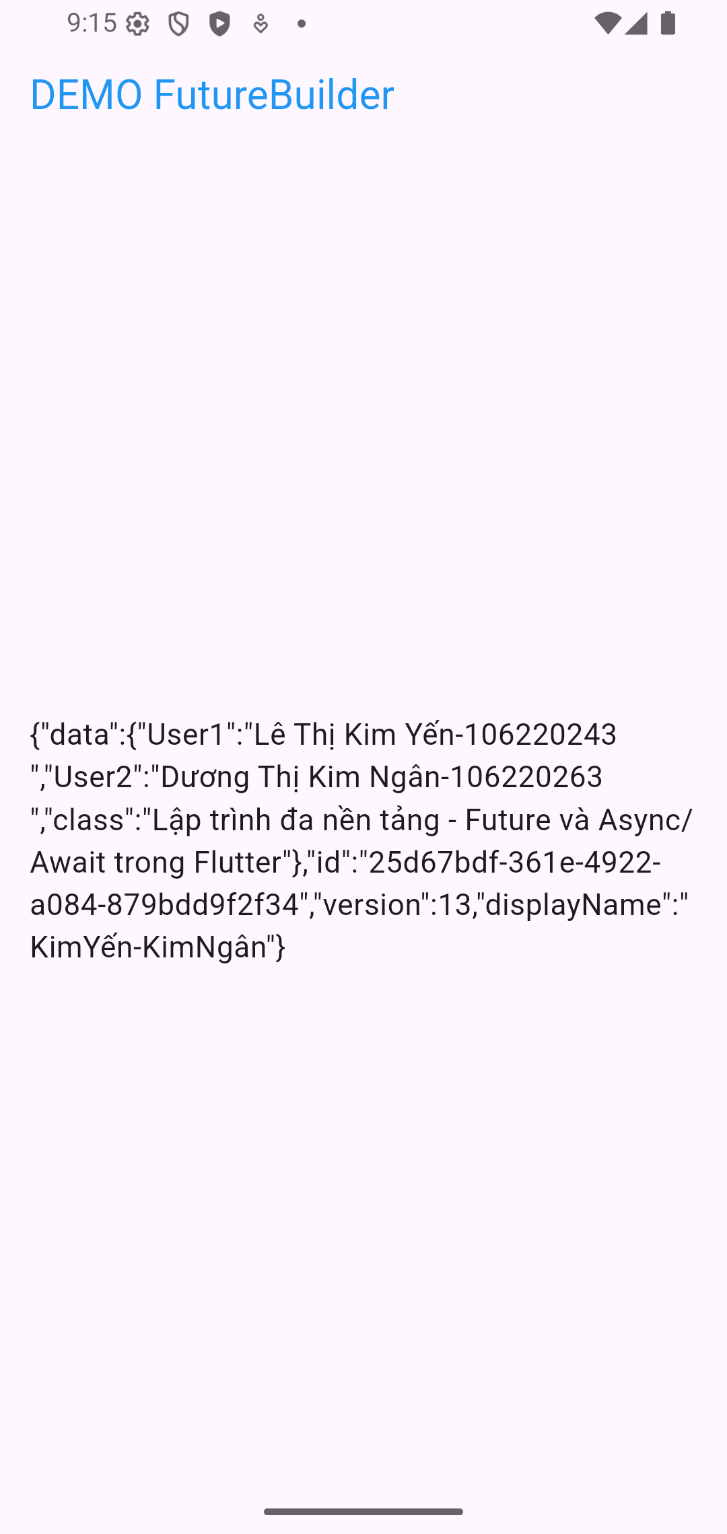
    );

  }

}

* Kết quả trên máy ảo:

TH1: truy cập đúng URL API



TH2: truy cập sai URL API



TH3: Record trong API rỗng



1. Combine multiple Futures với Future.wait():
2. Future.wait():

* **Future.wait()** là một công cụ cho phép bạn chạy nhiều Future cùng một lúc (song song) và chỉ nhận được thông báo khi tất cả chúng đều hoàn thành.

1. Cách hoạt động:

* Khi tất cả Future trong Future.wait() thành công:
* Code :
* import 'dart:async';
* Future<String> taiFileA() async {
* await Future.delayed(Duration(seconds: 3));
* return "Nội dung File A";
* }
* Future<String> taiFileB() async {
* await Future.delayed(Duration(seconds: 1));
* return "Nội dung File B";
* }
* Future<void> main() async {
* print("Bắt đầu tải nhiều file...");
* Stopwatch watch = Stopwatch()..start();
* List<String> ketQua = await Future.wait([
* taiFileA(),
* taiFileB(),
* ]);
* watch.stop();
* print(ketQua[0]);
* print(ketQua[1]);
* print("Tổng thời gian: ${watch.elapsedMilliseconds / 1000} giây");
* }
* Kết quả :

A black background with white text

AI-generated content may be incorrect.

* Khi có 1 Future trong đó bị lỗi:
* Code :
* import 'dart:async';
* Future<String> taiThanhCong() async {
* await Future.delayed(Duration(seconds: 5));
* return "Thành công";
* }
* Future<String> taiThatBai() async {
* await Future.delayed(Duration(seconds: 2));
* throw Exception('Mất kết nối mạng!');
* }
* Future<void> main() async {
* print("Bắt đầu chạy...");
* try {
* await Future.wait([
* taiThanhCong(),
* taiThatBai(),
* ]);
* } catch (e) {
* print("ĐÃ XẢY RA LỖI: $e");
* }
* Kết quả:

A black background with white text

AI-generated content may be incorrect.

# TÀI LIỆU THAM KHẢO

[1]. https://dart.dev/libraries/async/futures-error-handling

[2]. https://dart.dev/codelabs/async-await

[3]. https://api.flutter.dev/flutter/widgets/FutureBuilder-class.html