**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**KHOA ĐIỆN TỬ VIỄN THÔNG**

**BÁO CÁO**

**LẬP TRÌNH ĐA NỀN TẢNG**

**Chủ đề 11: Future và Async/Await trong Flutter**

**Sinh viên thực hiện:**

**01. Đinh Hoàng Thuận Lớp: 22KTMT2**  **MSSV:**  106220271

**02. Trần Kiêm Quang Minh**  **Lớp: 22KTMT1**  **MSSV:**  106220225

**Người hướng dẫn:**

**TS. Nguyễn Duy Nhật Viễn**

**Đà Nẵng, 2025.**

**THUYẾT MINH**

**BÁO CÁO**

**LẬP TRÌNH ĐA NỀN TẢNG**

**Chủ đề 11: Future và Async/Await trong Flutter**

BẢNG PHÂN CÔNG CÔNG VIỆC TRONG NHÓM

|  |  |  |  |
| --- | --- | --- | --- |
| STT | HỌ VÀ TÊN | NHIỆM VỤ | KHỐI LƯỢNG |
| 01 | Đinh Hoàng Thuận | Tìm kiếm, tìm hiểu các nội dung chủ để 11, code nội dung : Error handling với try-catch    Combine multiple Futures với Future.wait() | 50% |
| 02 | Trần Kiêm Quang Minh | Tìm kiếm, tìm hiểu các nội dung chủ để 11, và code nội dung: Giải thích Future, async/await concepts    - Demo FutureBuilder widget | 50% |

 Link code github:

Contents

[**1.** **GIỚI THIỆU VỀ FUTURE VÀ ASYNC/AWAIT TRONG FLUTTER** 4](#_Toc212476816)

[**1.1. Future** 4](#_Toc212476817)

[**1.2.** **Async và Await** 5](#_Toc212476818)

[**1.3.** **Demo FutureBuilder Widget** 5](#_Toc212476819)

[**1.4. Erro Handling với try cat-catch** 8](#_Toc212476820)

[**1.5. Combine multiple Futures với Future.wait()** 10](#_Toc212476821)

[**2.** **DEMO WEATHER APP** 10](#_Toc212476822)

[**3.** **KẾT LUẬN** 11](#_Toc212476823)

[**4.** **TÀI LIỆU THAM KHẢO** 12](#_Toc212476824)

**NỘI DUNG**

# 

# **GIỚI THIỆU VỀ** **FUTURE VÀ ASYNC/AWAIT TRONG FLUTTER**

Trong quá trình phát triển ứng dụng Flutter, việc xử lý **dữ liệu bất đồng bộ** là rất quan trọng, đặc biệt khi ứng dụng cần **gọi API, đọc dữ liệu cảm biến hoặc truy cập Internet.** Nếu không sử dụng cơ chế bất đồng bộ, giao diện có thể bị treo khi chờ kết quả trả về.

Để khắc phục điều đó, Flutter cung cấp **Future** và cú pháp **async/await** giúp lập trình viên xử lý dữ liệu bất đồng bộ **một cách đơn giản và hiệu quả.** Future đại diện cho một kết quả sẽ có trong tương lai, còn async/await giúp viết code bất đồng bộ dễ đọc hơn mà vẫn giữ luồng chương trình mạch lạc.

Khi kết hợp với **FutureBuilder,** ứng dụng có thể hiển thị dữ liệu tải về, báo lỗi hoặc trạng thái đang chờ theo thời gian thực mà không làm ảnh hưởng đến giao diện người dùng.

## **1.1. Future**

**Đặc điểm:** Future là đối tượng đại diện cho một giá trị hoặc lỗi sẽ có trong tương lai.Được dùng cho các tác vụ cần thời gian thực hiện như gọi API, đọc file hay truy vấn dữ liệu. Future có ba trạng thái: chưa hoàn thành (uncompleted)*,* hoàn thành với giá trị(Completed with value)*, và* hoàn thành với lỗi (Completed with error)*.*

**Mô tả hoạt động:** Khi một hàm trả về Future, chương trình không chờ kết quả mà tiếp tục thực thi. Sau khi tác vụ hoàn tất, kết quả hoặc lỗi được xử lý thông qua .then(), .catchError() hoặc await. Nhờ vậy ứng dụng vẫn phản hồi tốt trong khi tải dữ liệu.

## **1.2.** **Async và Await**

**Đặc điểm:** Async được thêm vào trước hàm để khai báo hàm bất đồng bộ, còn await giúp tạm dừng hàm cho đến khi Future hoàn tất.

**Mô tả hoạt động:** Khi gặp lệnh await, chương trình tạm dừng tại đó cho đến khi Future trả về kết quả, nhưng giao diện ứng dụng vẫn tiếp tục hoạt động bình thường. Nhờ đó, lập trình viên có thể viết code xử lý dữ liệu mạng, file hoặc API như code đồng bộ, giúp chương trình dễ hiểu và bảo trì hơn.

Code

Future<Map<String, dynamic>> fetchCurrentWeather(String city) async {  
 final Uri url = Uri.*parse*(  
 "https://api.openweathermap.org/data/2.5/weather?q=$city&appid=$weatherApiKey&units=metric&lang=vi");  
 final http.Response response = await http.get(url);  
 if (response.statusCode == 200) {  
 final Map<String, dynamic> data = jsonDecode(response.body) as Map<String, dynamic>;  
 return data;  
 } else {  
 throw Exception(" Không tìm thấy thành phố hoặc API lỗi!");  
 }  
}  
Future<Map<String, dynamic>> fetchAirQuality(double lat, double lon) async {  
 final Uri url = Uri.*parse*(  
 "https://api.openweathermap.org/data/2.5/air\_pollution?lat=$lat&lon=$lon&appid=$weatherApiKey");  
 final http.Response response = await http.get(url);  
 if (response.statusCode == 200) {  
 final Map<String, dynamic> data =  
 jsonDecode(response.body) as Map<String, dynamic>;  
 return data;  
 } else {  
 throw Exception(" Không lấy được chất lượng không khí!");  
 }  
}

Code phần future và async/await

## **1.3.** **Demo FutureBuilder Widget**

**Đặc điểm:** FutureBuilder là widget dùng để liên kết giữa Future (dữ liệu bất đồng bộ) và UI (giao diện ).

FutureBuilder có 3 trạng thái: waiting: Future đang chạy → hiển thị vòng tròn loading. hasError: Future gặp lỗi → hiển thị thông báo lỗi. hasData: Future trả kết quả → hiển thị dữ liệu thật lên UI.

FutureBuilder tự động lắng nghe trạng thái của Future và cập nhật lại giao diện khi dữ liệu thay đổi. Giúp xử lý các trạng thái UI (loading, error, empty, content) một cách dễ dàng.

Code:

FutureBuilder<(Map<String, dynamic>, Map<String, dynamic>)>(  
 future: futureWeather,  
 builder: (context, snapshot) {  
 if (snapshot.connectionState ==  
 ConnectionState.waiting) {  
 return const Center(  
 child: CircularProgressIndicator());  
 }  
  
 if (snapshot.hasError) {  
 final String errorMessage =  
 snapshot.error.toString();  
 return Center(  
 child: Column(  
 mainAxisSize: MainAxisSize.min,  
 children: [  
 Text(errorMessage,  
 textAlign: TextAlign.center,  
 style:  
 const TextStyle(color: Colors.*red*)),  
 const SizedBox(height: 12),  
 ElevatedButton(  
 onPressed: searchCity,  
 child: const Text("Thử lại"),  
 )  
 ],  
 ),  
 );  
 }  
  
 if (snapshot.hasData) {  
 final Map<String, dynamic> weather =  
 snapshot.data!.$1;  
 final Map<String, dynamic> air =  
 snapshot.data!.$2;  
  
 return WeatherCard(  
 weather: weather,  
 air: air,  
 refresh: searchCity,  
 );  
 }  
  
 return const SizedBox();  
 },  
),

Ảnh có chứa văn bản, ảnh chụp màn hình, Điện thoại di động, Thiết bị liên lạc

Nội dung do AI tạo ra có thể không chính xác.Ảnh có chứa văn bản, ảnh chụp màn hình, Điện thoại di động, Thiết bị di động

Nội dung do AI tạo ra có thể không chính xác.Kết quả hiển thị: Ảnh có chứa Điện thoại di động, đồ phụ tùng, Thiết bị liên lạc, Thiết bị liên lạc di động

Nội dung do AI tạo ra có thể không chính xác.

3 trạng thái

## **1.4. Erro Handling với try cat-catch**

**Đặc điểm:** Cho phép chương trình bắt và xử lý các ngoại lệ xảy ra trong quá trình thực thi bất đồng bộ như mất mạng, lỗi API hoặc hết thời gian chờ, giúp ứng dụng không bị dừng đột ngột.

Một số ngoại lệ phổ biến:

**FormatException**: Lỗi định dạng (ví dụ: parse string thành số)  
**RangeError**: Truy cập index ngoài phạm vi của List  
**ArgumentError**: Tham số không hợp lệ  
**StateError**: Trạng thái đối tượng không hợp lệ  
**FileSystemException**: Lỗi thao tác với file/thư mục  
**SocketException**: Lỗi kết nối mạng  
**TimeoutException**: Hết thời gian chờ

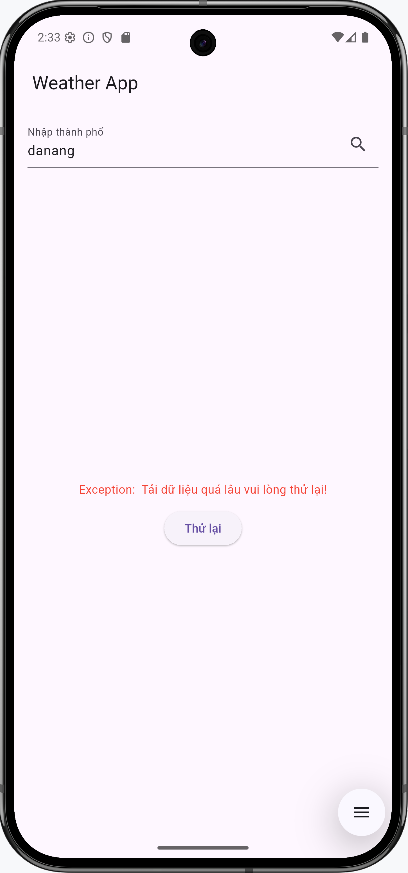
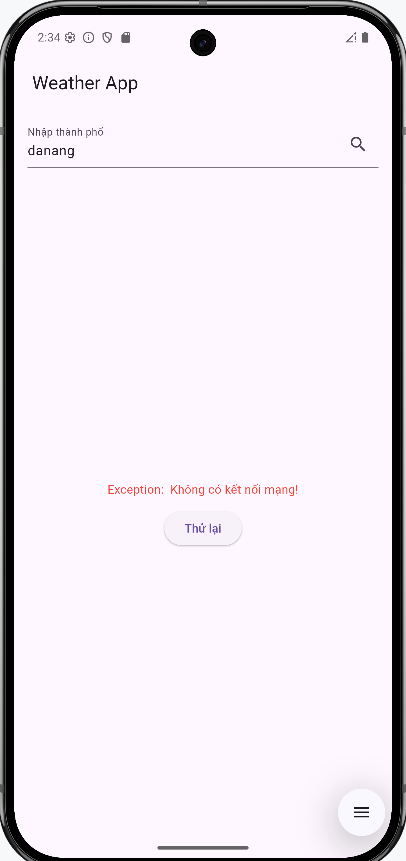
**Mô tả hoạt động:** Khối try chứa đoạn mã có thể gây lỗi; nếu lỗi xảy ra, catch hoặc các khối on tương ứng (SocketException, TimeoutException) sẽ xử lý và hiển thị thông báo thân thiện cho người dùng.

Code:

Future<(Map<String, dynamic>, Map<String, dynamic>)> loadWeatherData(  
 String city) async {  
 try {  
 final Map<String, dynamic> weather =  
 await fetchCurrentWeather(city).timeout(  
 const Duration(seconds: 5),  
 );  
 final double lat = weather["coord"]["lat"] as double;  
 final double lon = weather["coord"]["lon"] as double;  
 final results = await Future.*wait*([  
 fetchCurrentWeather(city),  
 fetchAirQuality(lat, lon),  
 ]);  
 return (results[0], results[1]);  
 } on SocketException {  
 throw Exception(" Không có kết nối mạng!");  
 } on TimeoutException {  
 throw Exception(" Tải dữ liệu quá lâu vui lòng thử lại!");  
 } catch (e) {  
 throw Exception(" Đã xảy ra lỗi không xác định: $e");  
 }  
}

Kết quả hiển thị:

Ảnh có chứa văn bản, ảnh chụp màn hình, Điện thoại di động, Thiết bị liên lạc

Nội dung do AI tạo ra có thể không chính xác.

## **1.5. Combine multiple Futures với Future.wait()**

**Đặc điểm:** Cho phép chạy song song nhiều Future để rút ngắn thời gian chờ và xử lý đồng thời nhiều tác vụ bất đồng bộ.

**Mô tả hoạt động:** Future.wait() nhận vào danh sách các Future, đợi tất cả hoàn tất rồi trả về danh sách kết quả, giúp ứng dụng tối ưu hiệu năng khi cần lấy nhiều nguồn dữ liệu cùng lúc.

Code:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Nội dung do AI tạo ra có thể không chính xác.

1. **DEMO WEATHER APP**

Ứng dụng Weather App là một ứng dụng di động được xây dựng bằng Flutter nhằm hiển thị thông tin thời tiết theo thời gian thực dựa trên dữ liệu lấy từ API OpenWeatherMap.

Người dùng có thể nhập tên thành phố để xem các thông số như nhiệt độ, độ ẩm, lượng mưa, tốc độ gió và chỉ số chất lượng không khí (AQI).

Ứng dụng áp dụng lập trình bất đồng bộ với Future, async/await, FutureBuilder và xử lý lỗi bằng trycatch giúp đảm bảo trải nghiệm mượt mà và không bị gián đoạn.

Ảnh có chứa văn bản, ảnh chụp màn hình, Điện thoại di động, Thiết bị di động

Nội dung do AI tạo ra có thể không chính xác.

# **KẾT LUẬN**

Qua quá trình tìm hiểu và thực hành, ta thấy rằng **Future** và **async/await** đóng vai trò cốt lõi trong việc xử lý các tác vụ **bất đồng bộ (asynchronous)** của Flutter. Nhờ đó, ứng dụng có thể thực hiện các công việc tốn thời gian như **gọi API, đọc/ghi dữ liệu mạng,** hay **tính toán phức tạp** mà **không làm đứng hoặc giật giao diện người dùng.**

Cơ chế **Future** giúp lập trình viên quản lý kết quả của các thao tác chưa hoàn thành, trong khi cặp từ khóa **async/await** mang đến cú pháp dễ đọc, rõ ràng hơn so với việc sử dụng nhiều callback lồng nhau.

Widget **FutureBuilder** lại đóng vai trò trung gian giữa **dữ liệu bất đồng bộ và giao diện,** tự động cập nhật trạng thái UI tương ứng với từng giai đoạn của Future (đang tải, có lỗi, hoặc hoàn tất).

Bên cạnh đó, việc sử dụng **try–catch** trong các hàm bất đồng bộ giúp ứng dụng **xử lý và thông báo lỗi một cách an toàn,** tránh tình trạng crash do lỗi mạng, API hoặc timeout.  
Đặc biệt, **Future.wait()** cho phép **thực thi nhiều Future song song,** giúp giảm đáng kể thời gian tải dữ liệu, tăng hiệu năng và trải nghiệm người dùng.

Tổng hợp lại, việc nắm vững các kỹ thuật trên không chỉ giúp người lập trình Flutter viết mã hiệu quả hơn mà còn là nền tảng quan trọng cho các ứng dụng thực tế như **ứng dụng thời tiết, hệ thống IoT, quản lý dữ liệu từ Firebase, hoặc giao tiếp RESTful API.**  
Các kiến thức này là bước đệm cần thiết để phát triển những ứng dụng Flutter **tối ưu, linh hoạt và giàu tương tác trong tương lai.**

# **TÀI LIỆU THAM KHẢO**

[1] Nguyễn Duy Nhật Viễn, *“Lập trình đa nền tảng ”* tài liệu bài giảng PDF, Khoa Điện tử Viễn thông, Trường Đại học Bách khoa – Đại học Đà Nẵng, 2025.  
Truy cập tại:[Tại đây](https://drive.google.com/file/d/1PMQm5DOBg1ZM6Z6BWeE5G6vBzOg6-4Ib/view?usp=sharing)

[2] Flutter Team, *“Future<T> class”*  
Truy cập: [Tại đây](https://api.flutter.dev/flutter/dart-async/Future-class.html)

[3] Flutter Team, *“FutureBuilder<T> class”*  
Truy cập: [Tại đây](https://api.flutter.dev/flutter/widgets/FutureBuilder-class.html)

[4] Dart., *“Error handling”*  
Truy cập: [Tại đây](https://api.flutter.dev/flutter/widgets/FutureBuilder-class.html)

[5] OpenWeatherMap API, *“How to make an API call,”*

Truy cập : [Tại đây](https://openweathermap.org/current)

***Chú ý:***

Phải tuân thủ format của tài liệu tham khảo.

Nếu có tham khảo các thuyết minh, phải trích dẫn và thể hiện cụ thể trong các tài liệu tham khảo. Code của khóa trước phải upload đầy đủ code cũ lẫn code mới, trong thuyết minh phải chỉ ra được cái mới của code nhóm thực hiện. Nếu phát hiện sao chép mà không trích dẫn, không có trong mục tài liệu tham khảo thì toàn nhóm sẽ bị điểm 0.