Đề tài: Future và Async/Await trong Flutter

Thành viên : Lê Xuân Nam
Nguyễn Nam Phương

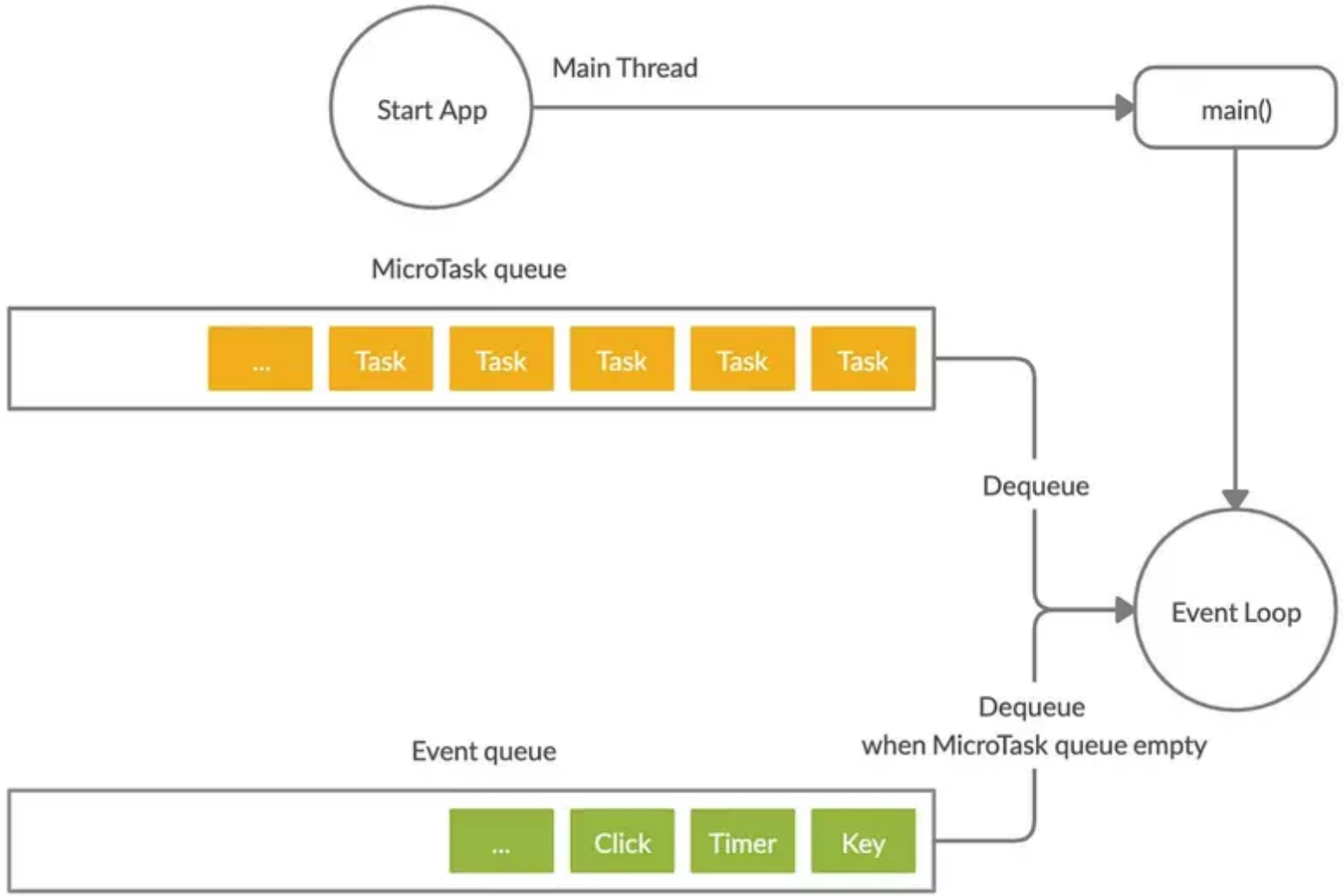
Mục lục

1.Future

- Là 1 đối tượng đại diện cho kết quả của hoạt động bất đồng bộ sẽ có trong tương lai

Ví dụ: Khi đặt hàng, cửa hàng sẽ hứa giao hàng sau 3 phút, sau 3 phút thì nhận được món hàng

- Có 2 trạng thái chính:
  + Chưa hoàn thành: Future vừa được khởi tạo, hoạt động đang được thực hiện
  + Hoàn thành: Trả về giá trị hoặc lỗi


Start App
Main Thread
main()
MicroTask queue
...
Task
Task
Task
Task
Task
Dequeue
Event Loop
Dequeue
when MicroTask queue empty
Event queue
...
Click
Timer
Key

```dart
import 'dart:async';

Future<String> tenHocSinh() {
  return Future.value("Nguyen Van A");
}

Run | Debug | Profile
void main() {
  tenHocSinh().then((name) {
    print(name);
  });
  print("tuoi: 16");
}
```

```
PS C:\Users\KimAnh\
tuoi: 16
Nguyen Van A
```

2.Async/await

- Async là một từ khóa trong Dart dùng để đánh dấu một hàm là bất đồng bộ

- Hàm async luôn trả về một Future, ngay cả khi không khai báo rõ ràng

- Ví dụ:

```
Future<String> tenHocSinh() async{
  return "Nguyen Van A";
}
```

- Await là từ khóa dùng bên trong hàm async để tạm dừng thực thi hàm cho đến khi một Future hoàn thành, lấy giá trị từ Future đó.

- Ví dụ:

```dart
import 'dart:async';

Future<String> tenHocSinh() async{
  return "Nguyen Van A";
}

Run | Debug | Profile
void main() async{
  String name = await tenHocSinh();
  print(name);
  print("tuoi: 16");
}
```

```
Nguyen Van A
tuoi: 16
```

3.FutureBuilder

- Là widget giúp xây dựng UI dựa trên kết quả của 1 Future
- Tự động cập nhật giao diện khi Future thay đổi trạng thái
- Giúp xử lý các trạng thái UI (loading, error, empty, content) một cách dễ dàng

```dart
import 'package:flutter/material.dart';
import 'dart:async';

Future<String> tenHocSinh() async {
  await Future.delayed(Duration(seconds: 2));
  return 'Nguyễn Văn A';
}

Run | Debug | Profile
void main() {
  runApp(
    MaterialApp(
      home: Scaffold(
        body: FutureBuilder<String>(
          future: tenHocSinh(),
          builder: (context, snapshot) {
            if (snapshot.connectionState == ConnectionState.waiting) {
              return const CircularProgressIndicator();
            } else if (snapshot.hasError) {
              return Text('Lỗi: ${snapshot.error}');
            } else if (snapshot.hasData) {
              return Text('Tên: ${snapshot.data}');
            }
            return const Text('Không có dữ liệu');
          },
        ), // FutureBuilder
      ), // Scaffold
    ), // MaterialApp
  );
}
```

4.Error Handling với try–catch

- `try-catch` là **cấu trúc xử lý lỗi (error handling)** trong Dart/Flutter
- Dùng để **phát hiện và xử lý lỗi** xảy ra trong lúc chạy chương trình, đặc biệt là trong **hàm bất đồng bộ (async/Future)**.
- Nếu không có `try-catch`, khi Future bị lỗi (ví dụ mất mạng, API sai, đọc file lỗi) → **app có thể bị crash**.
- Cách khai báo:

```
try {
  // đoạn code có thể gây lỗi
} catch (e) {
  // xử lý khi có lỗi xảy ra
}
```

```
try {
  await Data();
} catch (e) {
  print("Lỗi: $e");
}
```

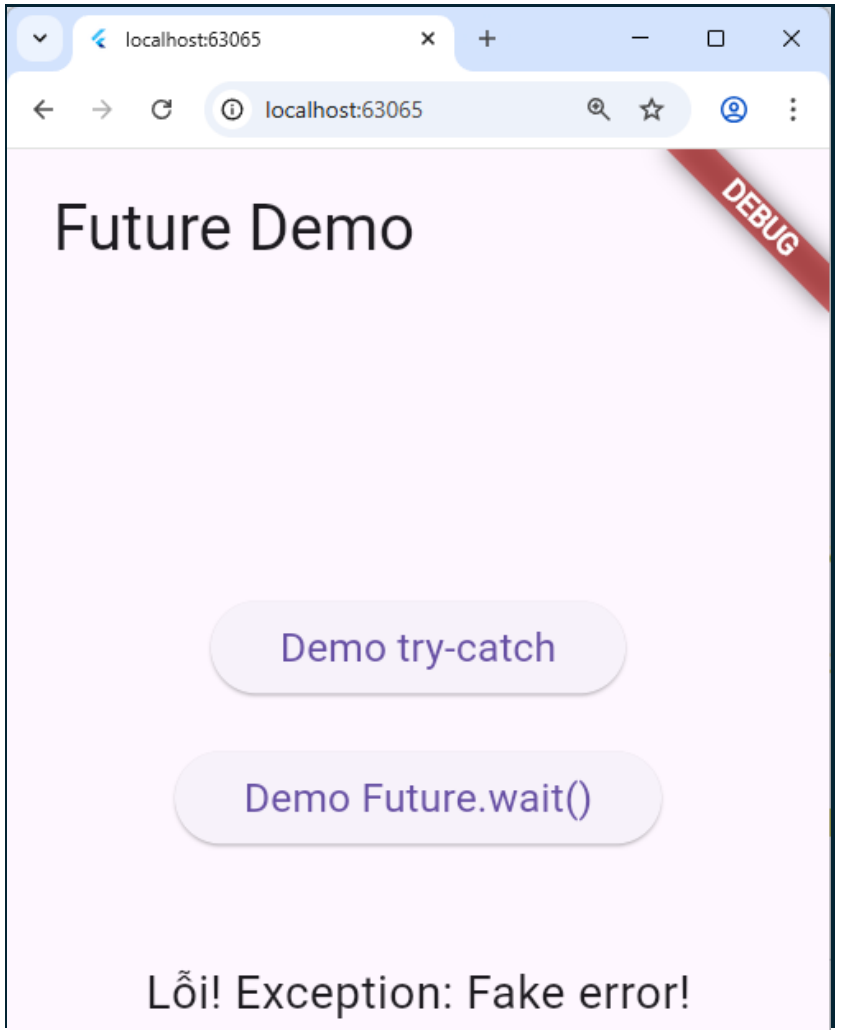
4.Error Handling với try–catch

Ví dụ: Hàm errorD() ném một lỗi Exception để giả lập lỗi

```
Future<String> errorD() async {
  await Future.delayed(const Duration(seconds: 1));
  throw Exception("Fake error!"); // demo lỗi
}
```

- Sử dụng try-catch gọi hàm errorD() và bắt lỗi nếu có, hiển thị kết quả bằng setState().

```
try {
  await errorD();
  setState(() => message = "Thành công!");
} catch (e) {
  setState(() => message = "Lỗi!");
}
```



5.Kết hợp nhiều Future với Future.wait()

- Future.wait() là một hàm trong Dart cho phép chạy nhiều Future cùng lúc
- Nó chờ tất cả Future hoàn thành rồi mới trả kết quả.
- Cách khai báo: `await Future.wait([future1, future2, future3]);`

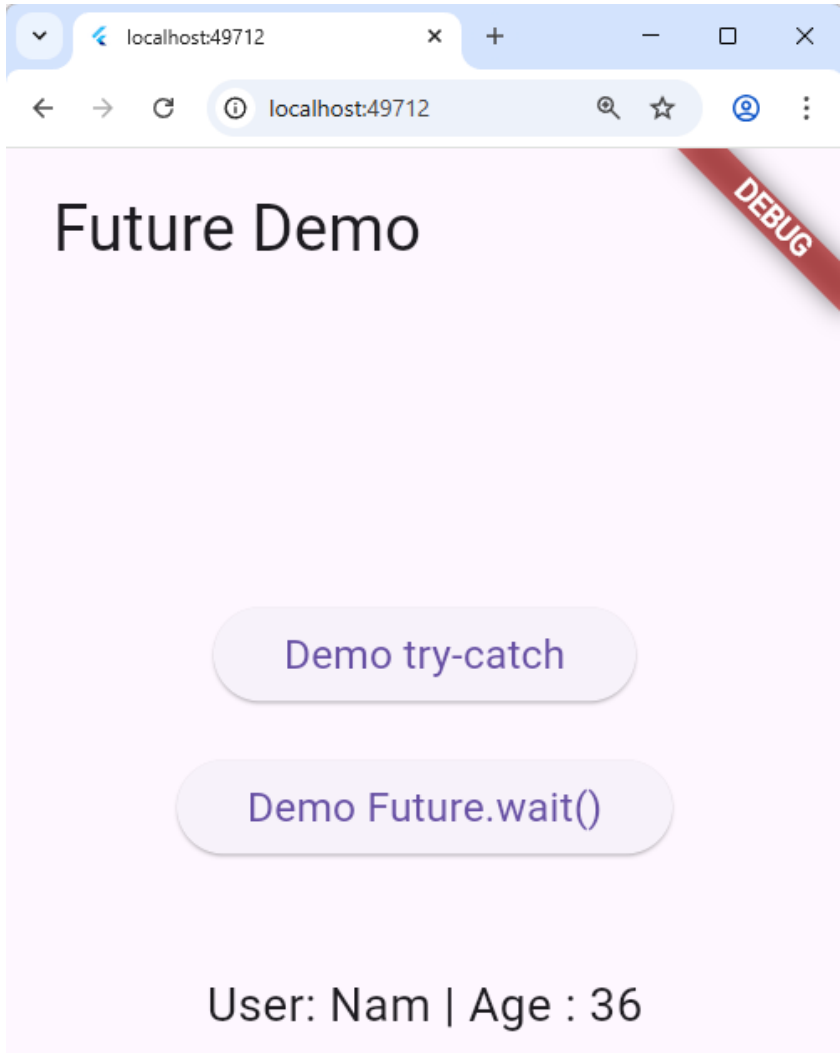
5.Kết hợp nhiều Future với Future.wait()

Ví dụ: Tạo 2 hàm getUser() và getAge() để mô phỏng việc tải dữ liệu

```
Future<String> getUser() async {
  await Future.delayed(const Duration(seconds: 1));
  return "User: Nam";
}
Future<String> getAge() async {
  await Future.delayed(const Duration(seconds: 2));
  return "Age : 36";
}
```

Sau đó Future.wait() sẽ chạy 2 Future cùng một lúc và trả kết quả khi cả 2 hoàn thành

```
var results = await Future.wait([getUser(), getAge()]);
setState(() => message = results.join(" | "));
```

Tài Liệu Tham Khảo

[1] Flutter Documentation – "Asynchronous programming: futures, async, await," dart.dev.
Available : https://dart.dev/libraries/async/async-await

[2] Flutter Documentation – "Error handling," dart.dev.
Available: https://dart.dev/language/error-handling

[3] Danish Sharma – "Efficiently Handle Concurrent Tasks in Flutter with Future.wait," Medium, 2023.
Available: https://medium.com/@danishsharma13255/efficiently-handle-concurrent-tasks-in-flutter-with-future-wait-7221076fcff0

[4] Jhon Munoz – "How to Use Future.wait in Dart for Better Futures Handling," Medium, 2022.
Available: https://medium.com/@jhonmunozromero/how-to-use-future-wait-in-dart-for-better-futures-handling-0e180e65ef2

[5] TS.Nguyễn Duy Nhật Viễn – "Tài liệu hướng dẫn Flutter (Slide bộ môn)," Trường Đại học Bách Khoa Đại Học Đà Nẵng, 2025.
Available: https://drive.google.com/file/d/1PMQm5DOBg1ZM6Z6BWeE5G6vBzOg6-4Ib/view

Phân công công việc

STT	HỌ VÀ TÊN	NHIỆM VỤ	KHỐI LƯỢNG
1	Lê Xuân Nam	Error handling với try-catch Combine multiple Futures với Future.wait()	50%
2	Nguyễn Nam Phương	Future, async/await concepts Demo FutureBuilder widget	50%