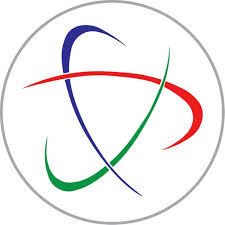
**TRƯỜNG ĐẠI HỌC BÁCH KHOA**

**KHOA ĐIỆN TỬ VIỄN THÔNG**



**BÁO CÁO**

**LẬP TRÌNH ĐA NỀN TẢNG**

**Future và Async/Await trong Flutter**

**Sinh viên thực hiện:**

**01. Trần Anh Toàn Lớp: 22KTMT1**  **MSSV:**  106220237

**02. Lê Thanh Trà** **Lớp: 22KTMT2**  **MSSV:**  106220273

**Người hướng dẫn:**

**TS. Nguyễn Duy Nhật Viễn**

**Đà Nẵng, 2025.**

**THUYẾT MINH**

**BÁO CÁO**

**LẬP TRÌNH ĐA NỀN TẢNG**

**Flutter Architecture và Widget Tree**

BẢNG PHÂN CÔNG CÔNG VIỆC TRONG NHÓM

|  |  |  |  |
| --- | --- | --- | --- |
| STT | HỌ VÀ TÊN | NHIỆM VỤ | KHỐI LƯỢNG |
| 01 | Trần Anh Toàn | Tìm hiểu và soạn nội dung phần Future, async/await, và code app demo FutureBuilder widget | 50% |
| 02 | Lê Thanh Trà | Tìm hiểu và soạn nội dung phần error handling với try-catch, và code app demo combine multiple Futures với Future.wait() | 50% |

 Link code github: <https://github.com/thanhtra3105/USV-Cross-platform>

**Mục lục**

[**1.** **Future, async/await concepts:** 4](#_Toc212387752)

[**1.1.** **Future:** 4](#_Toc212387753)

[**1.2.** **async/await:** 4](#_Toc212387754)

[**2.** **Demo FutureBuilder widget:** 4](#_Toc212387755)

[**2.1.** **FutureBuilder:** 4](#_Toc212387756)

[**2.2.** **Demo** 4](#_Toc212387757)

[**3.** **Error handling với try-catch:** 9](#_Toc212387758)

[**3.1.** **Lý thuyết:** 9](#_Toc212387759)

[**3.2.** **Demo:** 9](#_Toc212387760)

[**4.** **Combine multiple Futures với Future.wait():** 9](#_Toc212387761)

[**4.1.** **Lý thuyết:** 9](#_Toc212387762)

[**4.2.** **Demo:** 9](#_Toc212387763)

[**5.** **Tổng kết:** 14](#_Toc212387764)

**TÀI LIỆU THAM KHẢO**

1. **Future, async/await concepts:**
   1. **Future:**

* Future là một đối tượng trong Dart đại diện cho một giá trị sẽ có trong tương lai, thường được tạo ra bởi các tác vụ bất đồng bộ (như đọc file, gọi API, truy vấn cơ sở dữ liệu, ...).
* Khi gọi một hàm trả về Future, chương trình không dừng lại để chờ kết quả mà tiếp tục chạy các lệnh khác, giúp ứng dụng không bị treo hoặc giao diện.
  1. **async/await:**
* async là từ khóa dùng để đánh dấu một hàm là bất đồng bộ, hàm nãy sẽ trả về một Future
* await là từ khóa dùng để đợi kết quả của một Future. Khi sử dụng await, mã trong phạm vi hàm async sẽ tạm dừng cho đến khi Future hoàn thành, trả quyền điều khiển lại cho hệ thống để điều khiển các tác vụ khác
* async/await là cú pháp giúp viết code bất đồng bộ dễ hiểu hơn, gần giống với code tuần tự.

1. **Demo FutureBuilder widget:**
   1. **FutureBuilder:**

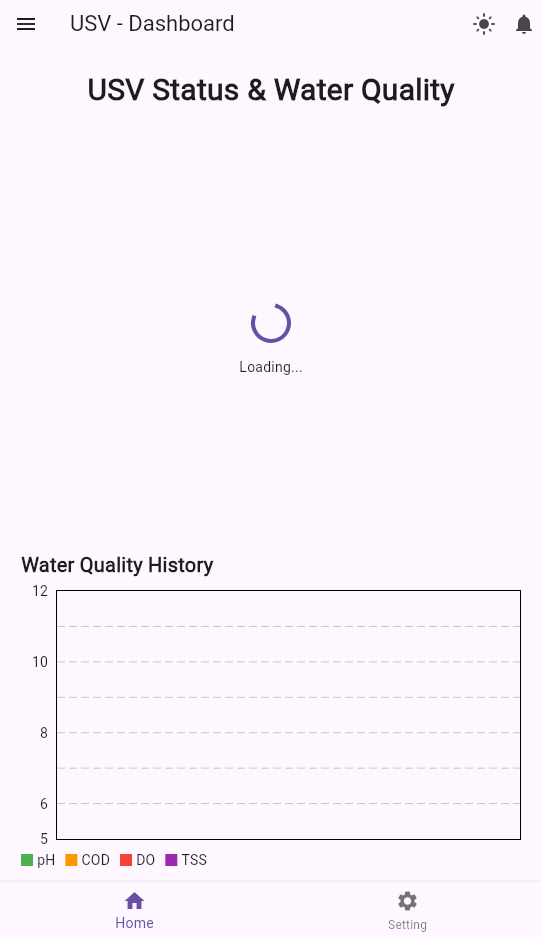
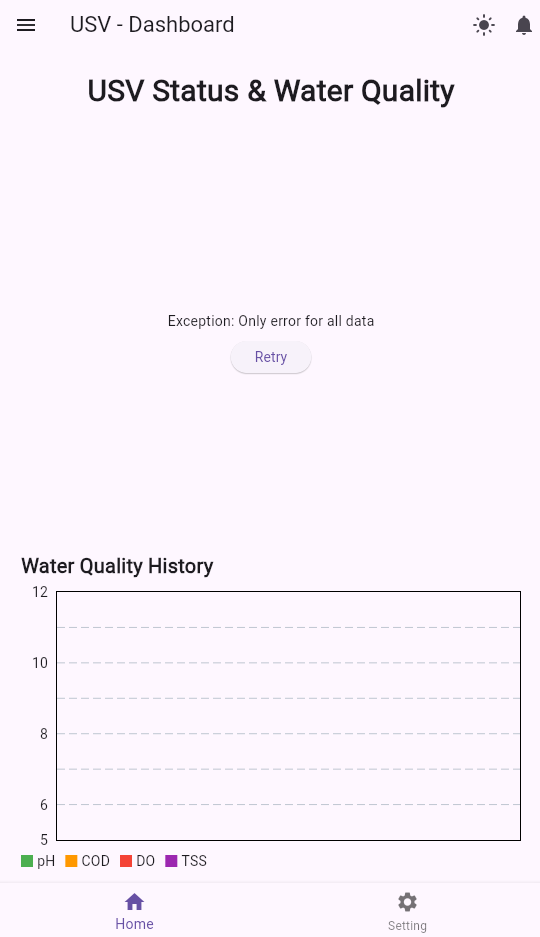
* FutureBuilder là một widget đặc biệt của Flutter giúp kết hợp dữ liệu bất đồng bộ (Future) với UI.
* Nó tự động rebuild UI khi trạng thái của Future thay đổi: chờ, hoàn thành, lỗi.
* Giúp xử lý các trạng thái UI (loading, error, empty, content) một cách dễ dàng.
* FutureBuilder cung cấp một đối tượng AsyncSnapshot (thường được gọi là snapshot). Bao gồm trạng thái kết nối với snapshot.connectionState, snapshot.hasError, snapshot.hasData,…
  1. **Demo**



*Hình 1: Tạo Future*



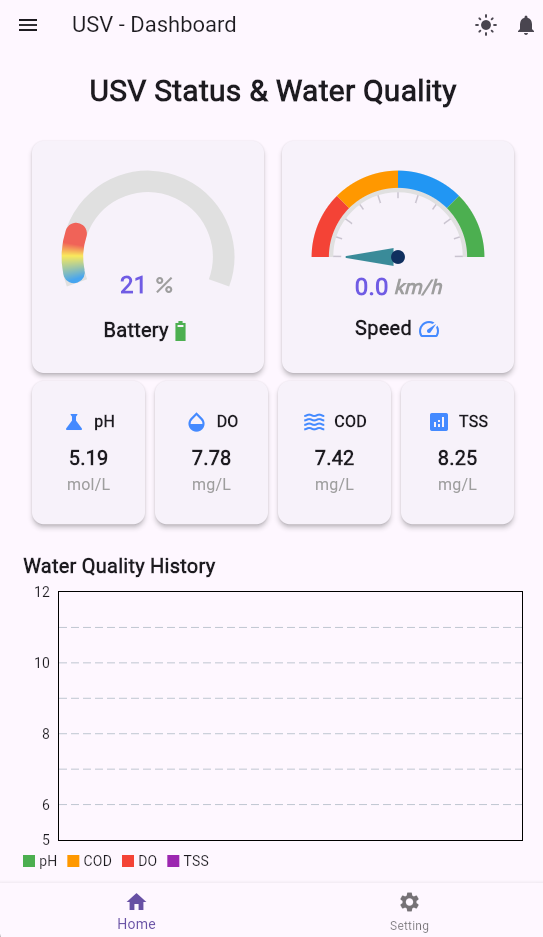
*Hình 2: Tạo FutureBuilder trạng thái waiting và hasError*



*Hình 3: Trạng thái waiting và hasError*



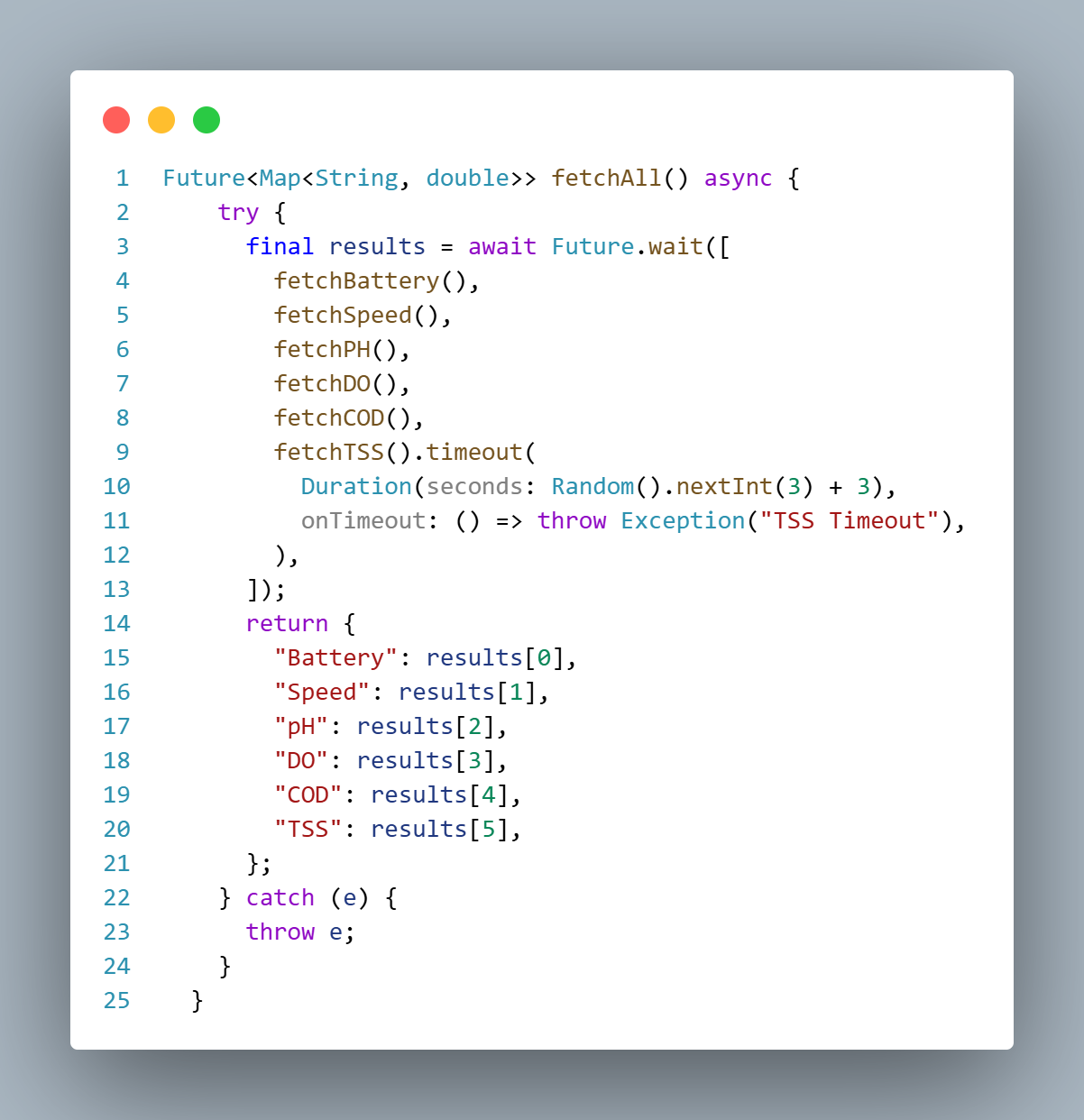
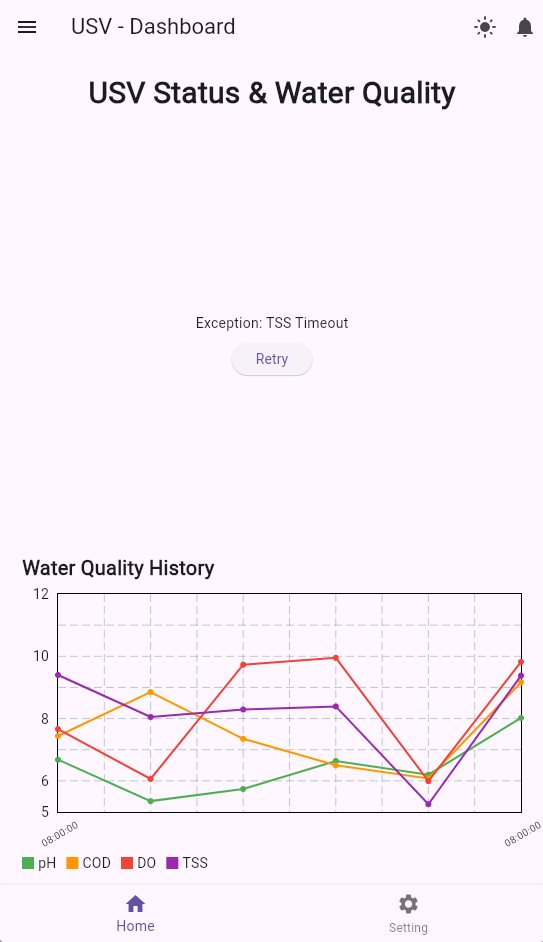
*Hình 4: Tạo FutureBuilder trạng thái hasData*



*Hình 5. Trạng thái hasData*

1. **Error handling với try-catch:**
   1. **Lý thuyết:**

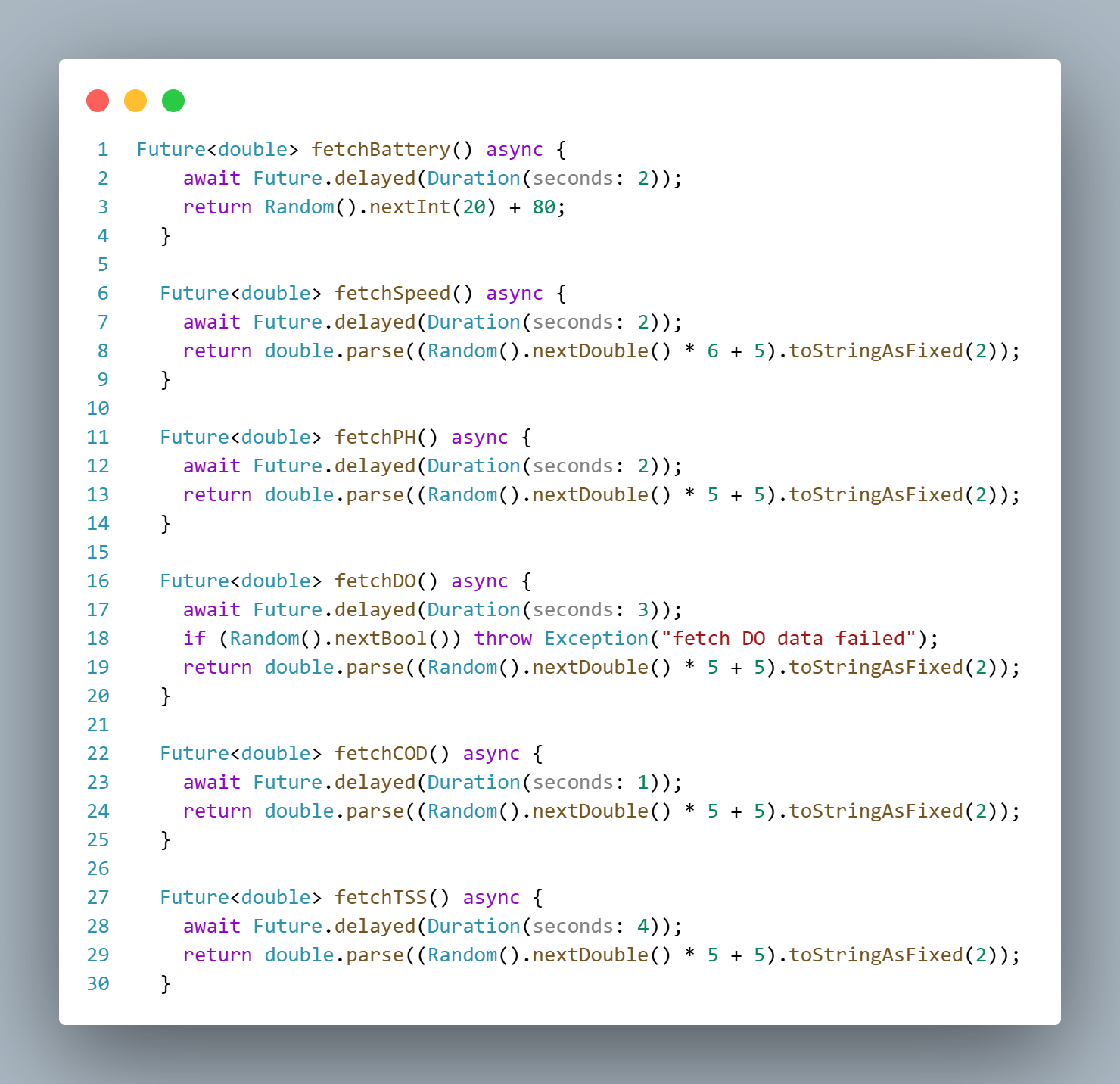
* Dart cung cấp try-catch-finally để xử lý lỗi một cách an toàn.
* Khi làm việc với Future hoặc async function, có thể xảy ra lỗi (như mất mạng, dữ liệu sai định dạng, timeout,…).
* Sử dụng try để thực hiện hàm, catch để bắt lỗi và finally để giải phóng bộ nhớ tránh memory leak.
  1. **Demo:**



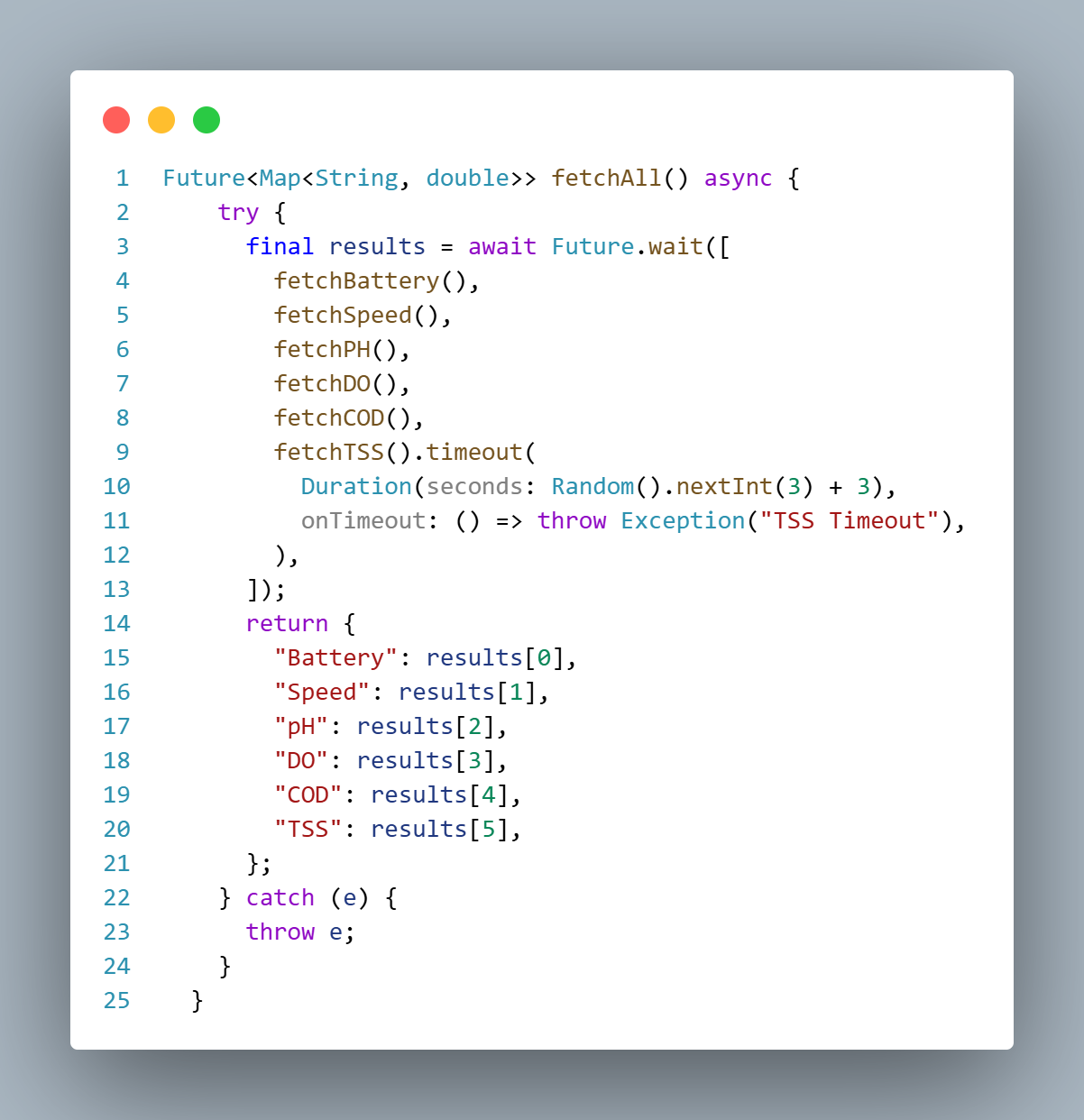
*Hình 6: Error handling với try-catch và kết quả*

1. **Combine multiple Futures với Future.wait():**
   1. **Lý thuyết:**

* Future.wait() là một phương thức cho phép chạy song song nhiều Future và chỉ trả về khi tất cả hoàn thành.
* Giúp tiết kiệm thời gian so với việc chờ từng Future một.
* Nhược điểm: Nếu một Future trong danh sách thất bại, toàn bộ Future.wait() sẽ thất bại.
  1. **Demo:**



*Hình 7: Khởi tạo các Future riêng để fetch từng dữ liệu khác nhau*



*Hình 8: Khởi tạo Future để combine các dữ liệu riêng lẻ ở trên*

* Để có thể cập nhật nhiều dữ liệu thì nhóm fetch tất cả dữ liệu lần đầu trong initState(), sau đó sử dụng Timer.periodic() để fetch data mỗi 3s để cập nhật lên dashboard.

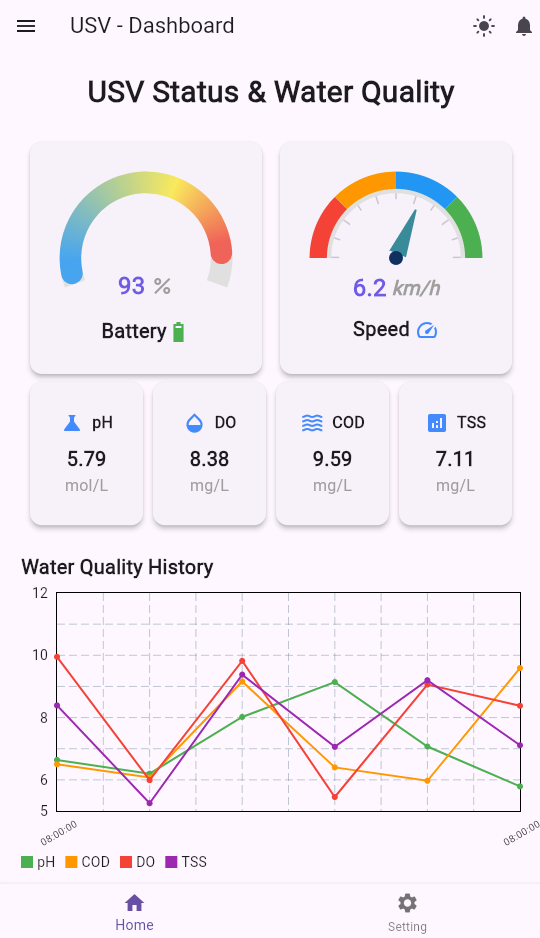
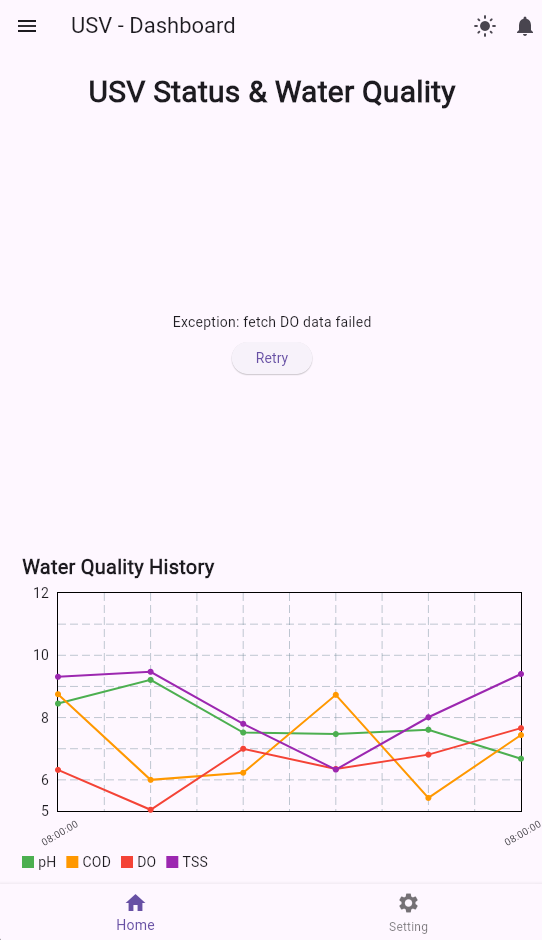


*Hình 9: Hàm initState để khởi tạo việc fecth dữ liệu và vẽ biểu đồ*

A screenshot of a computer program

AI-generated content may be incorrect.

*Hình 10: Đặt futureData vào trong FutureBuilder để cập nhật dữ liệu*



*Hình 11: Kết quả của combine multiple Futures dùng Future.wait() khi hasError và hasData*

1. **Tổng kết:**

* Future, async/await là công cụ của lập trình bất đồng bộ nâng cao trải nghiệm của người dùng, tránh bị treo hoặc đơ giao diện.
* FutureBuilder là một widget quan trọng trong việc kết hợp dữ liệu bất đồng bộ và UI.
* Error handling với try-catch là một cách để xử lý lỗi an toàn giúp tiện lợi hơn trong quá trình phát triển.
* Sử dụng combine multiple Futures với Future.wait() giúp chạy song song được nhiều Future và tiết kiệm thời gian hơn.

**TÀI LIỆU THAM KHẢO**

[1]. TS. Nguyễn Duy Nhật Viễn, “Slide bài giảng môn Lập trình Đa Nền Tảng”

[2]. Dart Dev, <https://dart.dev/libraries/async/async-await>, 2025

[3]. Flutter Docs, “<http://docs.flutter.dev/cookbook/networking/fetch-data>”, 2025

[4]. Dart Dev “<https://dart.dev/language/async>”, 2025

[5]. Flutter Mapp, “<https://www.youtube.com/watch?v=3kaGC_DrUnw>”, 2025