**TRƯỜNG ĐẠI HỌC TÀI NGUYÊN VÀ MÔI TRƯỜNG TP.HCM**

**KHOA: HỆ THỐNG THÔNG TIN VÀ VIỄN THÁM**

****

**BÁO CÁO ĐỒ ÁN CUỐI KÌ MÔN**

**CÔNG NGHỆ LẬP TRÌNH ĐA NỀN TẢNG**

**CHO ỨNG DỤNG DI ĐỘNG**

**TỔNG QUAN VỀ NAVIGATOR, ROUTER. LẬP TRÌNH BẤT ĐỒNG BỘ VỚI FUTURE, ASYNC, AWAIT.**

|  |  |  |  |
| --- | --- | --- | --- |
| **GVHD:** | Th.S Nguyễn Thanh Truyền |  |  |
| **SVTH:** |  | **Mssv:** |  |
|  | Lê Hữu Nghĩa |  | 0850070032 |
|  | Huỳnh Hoàng Huy |  | 0850070018 |
|  | Huỳnh Thị Thanh Yến |  | 0850070071 |
|  | Nguyễn Anh Tú |  | 0850070063 |
|  | Phạm Hùng Quốc Thiện |  | 0850070052 |
|  | Nguyễn Hồ Thiện |  | 0850070051 |
| **Lớp:** | 08\_ĐH\_TTMT |  |  |
| **Khóa:** | 2019-2023 |  |  |

***TP. Hồ Chí Minh, tháng 08 năm 2023***

**TRƯỜNG ĐẠI HỌC TÀI NGUYÊN VÀ MÔI TRƯỜNG TP.HCM**

**KHOA: HỆ THỐNG THÔNG TIN VÀ VIỄN THÁM**

****

**BÁO CÁO ĐỒ ÁN CUỐI KÌ MÔN**

**CÔNG NGHỆ LẬP TRÌNH ĐA NỀN TẢNG**

**CHO ỨNG DỤNG DI ĐỘNG**

**TỔNG QUAN VỀ NAVIGATOR, ROUTER. LẬP TRÌNH BẤT ĐỒNG BỘ VỚI FUTURE, ASYNC, AWAIT.**

|  |  |  |  |
| --- | --- | --- | --- |
| **GVHD:** | Th.S Nguyễn Thanh Truyền |  |  |
| **SVTH:** |  | **Mssv:** |  |
|  | Lê Hữu Nghĩa |  | 0850070032 |
|  | Huỳnh Hoàng Huy |  | 0850070018 |
|  | Huỳnh Thị Thanh Yến |  | 0850070071 |
|  | Nguyễn Anh Tú |  | 0850070063 |
|  | Phạm Hùng Quốc Thiện |  | 0850070052 |
|  | Nguyễn Hồ Thiện |  | 0850070051 |
| **Lớp:** | 08\_ĐH\_TTMT |  |  |
| **Khóa:** | 2019-2023 |  |  |

***TP. Hồ Chí Minh, tháng 08 năm 2023***

**LỜI MỞ ĐẦU**

Trong thế giới số hóa ngày nay, ứng dụng di động đã trở thành một phần không thể thiếu của cuộc sống và là công cụ hỗ trợ hàng đầu cho việc truy cập thông tin, giải trí, giao tiếp và nhiều hoạt động khác. Để đáp ứng nhu cầu đa dạng của người dùng, việc xây dựng một ứng dụng di động không chỉ đòi hỏi sự tinh tế về giao diện, mà còn cần phải tập trung vào cách thức hiển thị và quản lý nội dung một cách hiệu quả.

Trong bối cảnh này, việc sử dụng Navigator và Router trong việc phát triển ứng dụng di động trở thành một yếu tố quyết định để tạo ra trải nghiệm người dùng tốt nhất. Navigator là thành phần quản lý các màn hình và tạo sự liên kết giữa chúng, trong khi Router đảm nhiệm quản lý quá trình điều hướng người dùng giữa các màn hình khác nhau. Sự kết hợp giữa Navigator và Router không chỉ giúp ứng dụng dễ dàng chuyển đổi giữa các màn hình, mà còn giúp điều hướng người dùng đến đúng nội dung mà họ mong muốn.

Một ứng dụng di động thành công không chỉ là về giao diện đẹp mắt mà còn là về trải nghiệm người dùng thông qua sự tương tác trực quan và thuận tiện. Bằng cách sử dụng Navigator và Router, chúng ta có thể tạo ra các luồng điều hướng rõ ràng và linh hoạt, đồng thời duy trì trạng thái của ứng dụng một cách hiệu quả. Điều này giúp người dùng dễ dàng tiếp cận thông tin, chức năng và tính năng mà họ cần mà không gặp rắc rối trong việc điều hướng.

Trong bài báo cáo này, nhóm em sẽ đi sâu vào việc tìm hiểu cách Navigator và Router tương tác với nhau để tạo ra sự liên kết mượt mà giữa các màn hình, cách chúng điều hướng người dùng và cách quản lý trạng thái của ứng dụng. Chúng ta cũng sẽ khám phá các phương pháp tích hợp Navigator và Router vào quy trình phát triển ứng dụng, từ việc xác định cấu trúc màn hình đến việc xử lý sự kiện và dữ liệu.

**LỜI CÁM ƠN**

Thay mặt cả lớp nhóm chúng em xin gửi lời cảm ơn chân thành và sâu sắc nhất đến thầy ThS.Nguyễn Thanh Truyền đã hỗ trợ và đồng hành cùng nhóm chúng em trong quá trình hoàn thành báo cáo này. Nhờ sự hổ trợ nhiệt tình của thầy mà chúng em đã hiểu biết hơn về môn học và ứng dụng môn học này vào thực tế cụ thể là ở bài báo cáo.

Tiếp đến nhóm chúng mình xin cảm ơn các bạn. Nhờ sự giúp đỡ nhiệt tình của các bạn mà nhóm chúng mình đã vượt qua được một số khó khăn trong quá trình làm bài báo cáo này.

Cuối cùng, nhóm xin bày tỏ lòng biết ơn chân thành, sự tri ân đến thầy và các bạn bè. Sự hỗ trợ và đồng hành của thầy và các bạn đã góp phần quan trọng vào thành công của báo cáo này. Chúng em hy vọng rằng chúng ta có thể tiếp tục hợp tác và xây dựng những thành công tiếp theo trong tương lai. Tuy đã rất cố gắn nhưng trong quá trình báo cáo không thể tránh khỏi có những sai sót nhỏ nhóm rất mong nhận được sự góp ý chân thành đến từ thầy và các bạn để nhóm có thể sửa chửa, cải thiện tốt hơn ở bài báo cáo này và những bài báo cáo tiếp theo.

**NHẬN XÉT**

**(Của giảng viên bộ môn)**

TP.HCM, ngày … tháng … năm 2021

Giảng viên bộ môn

**LỊCH LÀM VIỆC**

(Của Sinh viên)

|  |  |  |
| --- | --- | --- |
| Tuần | Nội dung  công việc được giao | Tự nhận xét về mức độ hoàn thành |
| **1**  Từ ngày 10/07  đến ngày 16/07 | Tìm hiểu lý thuyết và thảo luận phân chia công việc | Hoàn thành |
| **2**  Từ ngày 17/07  đến ngày 23/07 | Tìm hiểu lý thuyết và thảo luận phân chia công việc | Hoàn thành |
| **3**  Từ ngày 24/07  đến ngày 30/08 | Lập trình bất đồng bộ với future, async, await, thực hiện các ví dụ minh họa. | Đang Triển khai |
| **4**  Từ ngày 31/07  đến ngày 06/08 | Tìm hiểu phương pháp truyền dữ liệu giữa hai màn hình | Hoàn thành |
| **5**  Từ ngày 07/08  đến ngày 13/08 | Kiểm thử và bổ sung một số phần | Hoàn thành |
| **6**  Từ ngày 02/01  đến ngày 08/01 | Tổng kết và nhận xét | Hoàn thành |

TP. Hồ Chí Minh, ngày 05 tháng 08 năm 2023

**Sinh Viên**

**LỊCH LÀM VIỆC**

|  |  |
| --- | --- |
| **Tuần 2 ( 17/7 – 23/7)** | |
| **Thành viên** | Tìm hiểu lý thuyết và phân công công việc, đề xuất ý tưởng |
| Nghĩa |
| Yến |
| Huy |
| Tú |
| Hồ Thiện |
| Quốc Thiện |

|  |  |
| --- | --- |
| **Tuần 3/4 ( 23/7 – 30/7)** | |
| Thành Viên | Công việc |
| Lê Hữu Nghĩa | Design màn hình ứng dụng, them các sản phẩm.  Tìm hiểu về cách di chuyển dữ liệu giữa các màn hình. |
| Nguyễn Anh Tú | Lập trình bất đồng bộ với future, async, await, thực hiện các ví dụ minh họa.  Tìm hiểu về cách di chuyển dữ liệu giữa các màn hình. |
| Phạm Hùng Quốc Thiện | Tổng hợp và viết báo cáo tổng quan về đề tài, bao gồm các nội dung đã được tìm hiểu và phân tích. |
| Huỳnh Thị Thanh Yến | Tổng hợp và viết báo cáo tổng quan về đề tài, bao gồm các nội dung đã được tìm hiểu và phân tích. |
| Huỳnh Hoàng Huy | Tổng hợp và viết báo cáo tổng quan về đề tài, bao gồm các nội dung đã được tìm hiểu và phân tích. |
| Nguyễn Hồ Thiện | Không liên lạc được |

**MỤC LỤC**

[CHƯƠNG 1: TỔNG QUAN 1](#_Toc142924882)

[**1.1. Giới thiệu công việc được giao 1**](#_Toc142924883)

[**1.2. Phạm vi đề tài 3**](#_Toc142924884)

[**2.1. Navigator và Router. 4**](#_Toc142924885)

[**2.1.1. Khái niệm Navigator và Router trong Flutter. 4**](#_Toc142924886)

[**2.1.2. Chức năng Navigator và Router trong Flutter 6**](#_Toc142924887)

[**2.1.3. MaterialPageRoute 8**](#_Toc142924888)

[**2.2. Truyền dữ liệu giữa các màn hình. 8**](#_Toc142924889)

[**2.3. Lập trình bất đồng bộ 8**](#_Toc142924890)

[**2.3.1. Khái niệm Future 10**](#_Toc142924891)

[**2.3.2. Cách triển khai lập trình bất đồng bộ với Future, async, await 10**](#_Toc142924892)

[CHƯƠNG 3: CÀI ĐẶT THỬ NGHIỆM 11](#_Toc142924893)

[**3.1. Phương pháp nghiên cứu 11**](#_Toc142924894)

[**3.2. Phương pháp thực hiện 11**](#_Toc142924895)

[**3.2.1. Cài đặt ứng dụng 11**](#_Toc142924896)

[**3.2.2. Phân tích và thiết kế hệ thống 12**](#_Toc142924897)

[CHƯƠNG 4 : KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN 18](#_Toc142924898)

[**4.1. Kết luận 18**](#_Toc142924899)

[**4.2. Hướng phát triển 19**](#_Toc142924900)

[DANH MỤC TÀI LIỆU THAM KHẢO 21](#_Toc142924901)

**DANH MỤC BẢNG, SƠ ĐỒ, HÌNH**

[**Hinh 1: Ảnh trực quan về việc bất đồng bộ 9**](#_Toc142924905)

[**Hinh 2: Cấu trúc thư mục 12**](#_Toc142924906)

[**Hinh 3: food.dart 13**](#_Toc142924907)

[**Hinh 4: food\_detail\_screen.dart 14**](#_Toc142924908)

[**Hinh 5: food\_list\_screen.dart 15**](#_Toc142924909)

[**Hinh 6: main.dart 16**](#_Toc142924910)

[**Hinh 7: Giao diện app 17**](#_Toc142924911)

# **CHƯƠNG 1: TỔNG QUAN**

* 1. **Giới thiệu công việc được giao**

Nội dung công việc được nhóm bàn bạc thống nhất được triển khai như sau:

* **Nghiên cứu về Navigator và Router:**
* **Tìm hiểu cơ bản:** Hiểu về khái niệm và chức năng cơ bản của Navigator và Router trong phát triển ứng dụng di động.
* **Đọc tài liệu:** Tìm và đọc tài liệu chính thức về cách sử dụng Navigator và Router trong framework bạn đang làm việc.
* **Xem ví dụ:** Tìm và xem ví dụ thực tế về việc sử dụng Navigator và Router để thấy cách chúng được áp dụng trong thực tế.
* **Tạo dự án thử nghiệm:** Tạo dự án đơn giản để thử nghiệm cách hoạt động và tương tác giữa Navigator và Router.
* **Đọc mã nguồn mẫu:** Tìm mã nguồn mẫu hoặc dự án mã nguồn mở sử dụng Navigator và Router để hiểu cách tích hợp chúng trong dự án.
* **Thực hành:** Tích hợp Navigator và Router vào dự án của bạn và thực hành để hiểu rõ hơn về cách chúng hoạt động.
* Giải quyết vấn đề: Đối mặt với vấn đề, sử dụng tài liệu và kiến thức đã học để tìm giải pháp.
* **Cập nhật kiến thức:** Theo dõi tài liệu mới và thông tin cập nhật để duy trì kiến thức về Navigator và Router.
* **Thiết kế giao diện người dùng:**
* **Xác định yêu cầu:** Hiểu rõ chức năng và nội dung cần thiết trong ứng dụng, cùng với sự tương tác giữa các màn hình.
* **Phác thảo khung giao diện:** Vẽ bản phác thảo sơ bộ về cấu trúc tổng thể của ứng dụng, liên kết giữa các màn hình qua Navigator và Router.
* **Lập kế hoạch bố cục:** Xác định vị trí và kích thước của các thành phần giao diện, đảm bảo bố cục hài hòa và trực quan.
* **Thiết kế màn hình con:** Tạo giao diện cho từng màn hình con, tạo điểm nhấn cho từng màn hình và sắp xếp các thành phần giao diện.
* **Chọn màu sắc và hình ảnh:** Lựa chọn bảng màu phù hợp và hình ảnh thích hợp để tạo sự hấp dẫn và truyền tải thông tin.
* **Tạo giao diện đơn giản:** Thiết kế giao diện theo phong cách đơn giản, dễ hiểu và dễ sử dụng, đảm bảo tương tác mượt mà.
* **Sắp xếp thành phần:** Sắp xếp các thành phần giao diện một cách hợp lý, tạo cân đối và gọn gàng.
* **Thử nghiệm giao diện:** Kiểm tra giao diện trên các thiết bị khác nhau để đảm bảo hiển thị đúng và dễ sử dụng trên mọi nền tảng.
* **Kiểm tra cuối cùng:** Tiến hành kiểm tra cuối cùng để đảm bảo không có lỗi giao diện và tất cả tính năng hoạt động đúng.
* **Triển khai:** Đưa ứng dụng vào quá trình triển khai và chạy thử nghiệm trên các thiết bị di động.
* **Triển khai Navigator và Router:**
* **Thêm thư viện Navigator và Router:** Sử dụng công cụ quản lý gói hoặc trình quản lý thư viện để thêm thư viện Navigator và Router vào dự án của bạn.
* **Xác định cấu trúc màn hình:** Xác định các màn hình chính và tương tác giữa chúng. Điều này giúp bạn xác định cách Navigator và Router sẽ được áp dụng.
* **Tạo các màn hình:** Tạo các màn hình riêng lẻ cho từng phần của ứng dụng, chẳng hạn như trang chủ, chi tiết, danh sách, v.v.
* **Khởi tạo Navigator và Router:** Trong thành phần chính của ứng dụng, khởi tạo một Navigator và cấu hình Router để quản lý việc chuyển đổi giữa các màn hình.
* **Xác định tuyến đường (Routes):** Xác định các tuyến đường cho mỗi màn hình, đảm bảo rằng Router biết cách điều hướng người dùng khi họ thực hiện các tương tác.
* **Liên kết với tương tác người dùng:** Liên kết các tương tác người dùng, chẳng hạn như nút nhấn, liên kết, để kích hoạt các tuyến đường và thay đổi màn hình thông qua Navigator.
* **Quản lý trạng thái:** Đảm bảo trạng thái của ứng dụng được duy trì khi chuyển đổi giữa các màn hình, thông qua việc lưu trạng thái hoặc dữ liệu cần thiết.
* **Thử nghiệm và sửa lỗi:** Thử nghiệm chuyển đổi giữa các màn hình, xác minh tính đúng đắn và xử lý lỗi nếu có.
* **Tối ưu hóa hiệu suất:** Đảm bảo việc sử dụng Navigator và Router không ảnh hưởng đến hiệu suất ứng dụng, tối ưu hóa quá trình điều hướng.
* **Triển khai và theo dõi:** Đưa ứng dụng vào quá trình triển khai và theo dõi hoạt động của Navigator và Router trong sản phẩm thực tế.
* **Tài liệu hóa:**
* Tạo tài liệu hướng dẫn sử dụng cho Navigator và Router trong ứng dụng
* Viết báo cáo và chuẩn bị tài liệu trình bày về quá trình viết App và minh họa nội dung Navigator và Router
  1. **Phạm vi đề tài**

Trong quá trình thực hiện nội dung và phát triển ứng dụng minh họa nội dung Navigator, Router phạm vi đề tài được hình thành ở phạm vi:

* **Giới thiệu về Navigator và Router:** Trình bày khái niệm và chức năng cơ bản của Navigator và Router trong phát triển ứng dụng di động. Giải thích vì sao chúng quan trọng và cần thiết để tạo trải nghiệm người dùng tốt hơn.
* **Nghiên cứu về Navigator và Router:** Tìm hiểu sâu hơn về cách Navigator và Router hoạt động trong các framework phát triển ứng dụng di động phổ biến như React Native, Flutter, hoặc Vue Native. Khám phá cách chúng tương tác với nhau và quản lý chuyển đổi giữa các màn hình.
* **Thiết kế giao diện người dùng:** Đề cập đến quá trình thiết kế giao diện cho ứng dụng, đảm bảo sự trực quan và tương tác thông qua Navigator và Router. Bao gồm cách xây dựng bố cục, chọn màu sắc, hình ảnh và các thành phần giao diện.
* **Triển khai Navigator và Router:** Mô tả cụ thể về cách triển khai Navigator và Router trong ứng dụng thực tế. Bao gồm các bước từ thêm thư viện, khởi tạo, xác định tuyến đường, tương tác người dùng và quản lý trạng thái.
* **Thử nghiệm và tối ưu hóa:** Tiến hành thử nghiệm chuyển đổi giữa các màn hình để đảm bảo tính đúng đắn và tương tác mượt mà. Điều chỉnh và tối ưu hiệu suất sử dụng Navigator và Router.

Tổng kết lại quá trình nghiên cứu, thiết kế và triển khai Navigator và Router trong ứng dụng di động. Trình bày những kết quả và hiểu biết đã thu được từ việc thực hiện dự án này. Đề tài sẽ tập trung vào việc xây dựng ứng dụng di động minh họa về cách sử dụng Navigator và Router để quản lý tương tác và chuyển đổi giữa các màn hình một cách hiệu quả và tạo ra trải nghiệm người dùng tốt hơn.

**CHƯƠNG 2: CƠ SỞ LÝ LUẬN**

## **2.1. Navigator và Router.**

Navigator và Router là hai khái niệm quan trọng trong phát triển ứng dụng di động, giúp quản lý và điều hướng giữa các màn hình một cách hiệu quả. Chúng đóng vai trò quan trọng trong việc tạo ra trải nghiệm người dùng mượt mà và thuận tiện.

Navigator là một thành phần cho phép quản lý các màn hình và chuyển đổi giữa chúng trong ứng dụng. Nó tạo ra cấu trúc gần giống như "staple" trong tập hình, giúp bạn dễ dàng chuyển đổi giữa các màn hình bằng cách quản lý lịch sử và tương tác của người dùng. Với Navigator, bạn có thể thực hiện các thao tác như chuyển đổi màn hình, thêm, xóa và thay đổi trạng thái của chúng.

Router là một thành phần quan trọng trong việc quản lý điều hướng của ứng dụng. Nó xác định cách ứng dụng đáp ứng với các tương tác từ người dùng và chuyển hướng họ đến các màn hình tương ứng. Router định nghĩa các tuyến đường (Routes) cho từng màn hình và xác định cách chúng tương tác với nhau, giúp định hình toàn bộ quá trình điều hướng trong ứng dụng.

Để tìm hiểu rỏ hơn về Navigator và Router, ta có thể xem xét các tính năng và cách thức hoạt động của chúng chi tiết hơn như sau:

### **2.1.1. Khái niệm Navigator và Router trong Flutter.**

**a. Khái niệm Navigator:**

Navigator trong Flutter là một lớp quản lý và điều hướng giữa các màn hình trong ứng dụng di động. Được tích hợp sẵn trong framework Flutter, Navigator cho phép phát triển viên tạo ra cấu trúc phân lớp của ứng dụng, xây dựng luồng điều hướng của người dùng và tạo ra trải nghiệm tương tác mượt mà.

* **Ngăn xếp Màn hình:**

Navigator hoạt động bằng cách quản lý một "ngăn xếp màn hình", giống như ngăn xếp đĩa trong ngăn kéo. Mỗi khi bạn mở một màn hình mới, nó được thêm vào đỉnh ngăn xếp. Khi bạn quay lại hoặc đóng một màn hình, nó sẽ bị loại bỏ khỏi đỉnh ngăn xếp. Điều này cho phép người dùng chuyển đổi giữa các màn hình và quay lại một cách trôi chảy.

* **Phương thức chính:**

Navigator cung cấp nhiều phương thức quan trọng để quản lý và điều hướng màn hình:

* push: Mở một màn hình mới và thêm vào ngăn xếp.
* pop: Đóng màn hình hiện tại và quay lại màn hình trước đó.
* pushReplacement: Mở màn hình mới và loại bỏ màn hình hiện tại khỏi ngăn xếp.
* popUntil: Đóng các màn hình cho đến khi đạt được màn hình được chỉ định.

Và nhiều phương thức khác để kiểm soát điều hướng.

* **Sự quản lý trạng thái:**

Navigator cũng giúp duy trì trạng thái của ứng dụng khi chuyển đổi giữa các màn hình. Khi một màn hình bị đóng, trạng thái của nó vẫn được lưu giữ trong ngăn xếp. Điều này giúp người dùng không bị mất dữ liệu hoặc trạng thái khi chuyển đổi màn hình.

* **Chuyển đổi giữa các màn hình:**

Navigator cho phép bạn chuyển đổi giữa các màn hình bằng các thao tác như nhấn nút, kéo và thả, hoặc thậm chí tự động theo thời gian. Điều này giúp bạn tạo ra sự tương tác và chuyển đổi linh hoạt giữa các phần của ứng dụng.

Tóm lại, Navigator trong Flutter là một công cụ quan trọng giúp quản lý và điều hướng giữa các màn hình trong ứng dụng di động. Nó tạo ra cấu trúc luồng điều hướng, duy trì trạng thái và giúp tạo nên trải nghiệm người dùng tương tác và mượt mà.

**b. Khái niệm Router:**

Router trong Flutter là một phần quan trọng của việc quản lý điều hướng và tạo ra luồng điều hướng trong ứng dụng. Nhiệm vụ chính của Router là xác định cách ứng dụng phản hồi với các tương tác từ người dùng và chuyển hướng họ đến các màn hình tương ứng. Nó cung cấp một cơ chế linh hoạt để định nghĩa và quản lý các tuyến đường (Routes), cho phép bạn dễ dàng tạo trải nghiệm điều hướng tùy chỉnh cho ứng dụng của mình.

* **Các khái niệm quan trọng:**
* **Routes (Tuyến đường):** Là các định danh cho các màn hình cụ thể trong ứng dụng. Mỗi tuyến đường đại diện cho một màn hình và chứa thông tin về cách hiển thị và tương tác với nó.
* **Route Table (Bảng tuyến đường):** Là nơi bạn xác định các tuyến đường trong ứng dụng. Bảng tuyến đường liệt kê các tuyến đường và xác định cách chúng tương tác với các màn hình tương ứng.
* **Navigator:** Trong mô hình điều hướng thống nhất (Unified Navigation Model), Router là một phần của Navigator. Navigator sẽ sử dụng thông tin từ Router để quản lý và điều hướng các màn hình.
* **Sử dụng Router:**
* **Khởi tạo Router:** Bắt đầu bằng việc khởi tạo một Router. Bạn có thể sử dụng MaterialPageRoute để thiết lập một Router cho toàn bộ ứng dụng.
* **Xác định các tuyến đường:** Trong bảng tuyến đường, bạn định nghĩa các tuyến đường bằng cách ánh xạ tên đến màn hình tương ứng. Điều này cho phép bạn quyết định cách hiển thị và tương tác với từng màn hình.
* **Thực hiện điều hướng:** Sử dụng các phương thức của Router để thực hiện điều hướng người dùng đến các tuyến đường cụ thể, thay đổi màn hình và trải nghiệm.

Router trong Flutter giúp tạo ra một cấu trúc điều hướng tùy chỉnh cho ứng dụng của bạn. Bằng cách sử dụng các tuyến đường và bảng tuyến đường, bạn có thể quản lý cách người dùng tương tác với các phần khác nhau của ứng dụng và tạo ra trải nghiệm điều hướng đáp ứng và linh hoạt.

### **2.1.2. Chức năng Navigator và Router trong Flutter**

* **Chức năng Navigator:**

Navigator trong Flutter có chức năng quản lý và điều hướng giữa các màn hình trong ứng dụng. Nó cung cấp một loạt các phương thức và tính năng để tạo, chuyển đổi và quản lý các màn hình khác nhau, giúp tạo ra trải nghiệm người dùng mượt mà và hiệu quả. Dưới đây là những chức năng chính của Navigator trong Flutter:

* **Mở Màn hình Mới (push):** Bằng cách sử dụng phương thức Navigator.push, bạn có thể mở một màn hình mới và thêm vào ngăn xếp. Người dùng có thể chuyển đổi giữa các màn hình bằng cách nhấn nút hoặc tương tác với ứng dụng.
* **Đóng Màn hình Hiện Tại (pop):** Phương thức Navigator.pop cho phép bạn đóng màn hình hiện tại và quay lại màn hình trước đó. Điều này giúp tạo sự trôi chảy trong quá trình điều hướng.
* **Mở Màn hình Mới và Loại Bỏ Màn hình Hiện Tại (pushReplacement):** Để thay thế màn hình hiện tại bằng màn hình mới, bạn có thể sử dụng Navigator.pushReplacement. Màn hình mới sẽ thêm vào ngăn xếp và loại bỏ màn hình hiện tại.
* **Đóng Các Màn hình cho Đến Màn hình Xác Định (popUntil):** Bằng cách sử dụng Navigator.popUntil, bạn có thể đóng các màn hình trong ngăn xếp cho đến khi đạt được màn hình được xác định.
* **Điều Hướng Đến Màn hình theo Tên (pushNamed):** Sử dụng Navigator.pushNamed để chuyển đến một màn hình dựa trên tên tuyến đường được định nghĩa trong Router.
* **Truyền Dữ Liệu giữa Các Màn hình (arguments):** Khi mở một màn hình mới, bạn có thể truyền dữ liệu thông qua thuộc tính arguments, giúp truyền thông tin từ màn hình này sang màn hình khác.
* **Xử Lý Khi Màn hình Đóng (popUntil và addPostFrameCallback):** Bạn có thể sử dụng popUntil để thực hiện các thao tác khi màn hình đóng, ví dụ như cập nhật trạng thái của màn hình trước đó. Hơn nữa, addPostFrameCallback cho phép bạn thực hiện các thao tác sau khi giao diện người dùng được xây dựng lại sau khi màn hình đóng.
* **Quản Lý Lịch Sử Màn hình (pop và popUntil):** Navigator duy trì lịch sử các màn hình đã được mở, cho phép người dùng quay lại các màn hình trước đó một cách trực quan.

Navigator trong Flutter giúp xây dựng luồng điều hướng tương tác và quản lý các màn hình trong ứng dụng. Tích hợp các phương thức và chức năng linh hoạt, nó là công cụ mạnh mẽ giúp tạo nên trải nghiệm người dùng tốt hơn và mở ra nhiều cơ hội thiết kế giao diện đa dạng.

* **Chức năng Router:**

Router trong Flutter có chức năng quản lý và xác định cách điều hướng người dùng giữa các màn hình và tạo ra một cấu trúc điều hướng trong ứng dụng. Dưới đây là những chức năng chính của Router trong Flutter:

* **Xác Định Các Tuyến Đường (Routes):** Router cho phép bạn xác định các tuyến đường (Routes) trong ứng dụng. Mỗi tuyến đường đại diện cho một màn hình cụ thể và chứa thông tin về cách hiển thị và tương tác với màn hình đó.
* **Tạo Luồng Điều Hướng:** Bằng cách định nghĩa các tuyến đường, Router giúp bạn tạo ra một luồng điều hướng cho người dùng. Điều này cho phép người dùng chuyển đổi giữa các màn hình và trải nghiệm ứng dụng một cách liền mạch.
* **Truyền Dữ Liệu giữa Các Màn Hình:** Router cho phép bạn truyền dữ liệu giữa các màn hình thông qua thuộc tính arguments, giúp bạn chuyển dữ liệu từ một màn hình sang màn hình khác.
* **Quản Lý Màn Hình Hiện Tại:** Router giúp quản lý trạng thái màn hình hiện tại và xác định cách màn hình mới sẽ hiển thị khi người dùng thực hiện tương tác.
* **Tùy Chỉnh Chuyển Đổi Màn Hình:** Bạn có thể tùy chỉnh cách chuyển đổi giữa các màn hình bằng cách sử dụng các phương thức và thuộc tính được cung cấp bởi Router.
* **Xử Lý Sự Kiện Khi Màn Hình Đóng:** Router cho phép bạn thực hiện các thao tác sau khi người dùng thoát khỏi màn hình, giúp bạn cập nhật dữ liệu hoặc thực hiện các xử lý cuối cùng.

Router trong Flutter giúp bạn tạo nên một hệ thống điều hướng linh hoạt và quản lý màn hình một cách hiệu quả. Bằng cách sử dụng các tuyến đường và tính năng điều hướng tùy chỉnh, bạn có thể xây dựng trải nghiệm người dùng tốt hơn và quản lý sự tương tác giữa các màn hình trong ứng dụng của mình.

### **2.1.3. MaterialPageRoute**

MaterialPageRoute trong Flutter là một lớp dùng để xác định cách hiển thị và tạo một màn hình mới khi bạn chuyển đổi đến một tuyến đường cụ thể trong ứng dụng của bạn. Nó liên quan chặt chẽ đến nguyên tắc thiết kế của Material Design, giúp tạo ra trải nghiệm chuyển đổi mượt mà và thân thiện với người dùng.

## **2.2. Truyền dữ liệu giữa các màn hình.**

Trong Flutter, việc truyền dữ liệu giữa các màn hình có thể thực hiện thông qua thuộc tính arguments của RouteSettings. Đây là một cách mạnh mẽ để chia sẻ thông tin từ một màn hình gốc sang một màn hình mới khi bạn chuyển đổi giữa chúng.

Lưu ý rằng dữ liệu truyền đi và nhận về có thể là bất kỳ loại dữ liệu nào, không chỉ là chuỗi (string).

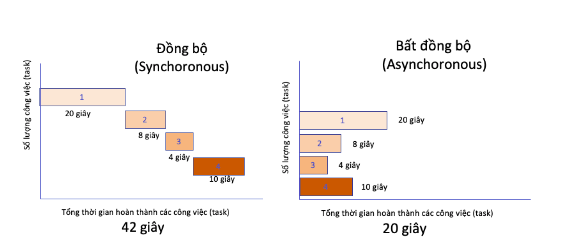
## **2.3. Lập trình bất đồng bộ**

Lập trình bất đồng bộ là một khái niệm quan trọng trong lĩnh vực lập trình, đặc biệt là trong các ứng dụng đa luồng, đa tiến trình hoặc trong việc giao tiếp với các tác vụ mà không làm treo chương trình chính. Lập trình bất đồng bộ cho phép ứng dụng tiếp tục thực hiện các công việc khác trong khi một tác vụ đang chờ hoàn thành, giúp tối ưu hóa thời gian và tài nguyên.

Trong Flutter, lập trình bất đồng bộ chủ yếu được thực hiện bằng cách sử dụng Future, async, và await. Dưới đây là một giải thích chi tiết về các khái niệm này:

* **Future:** Future đại diện cho một giá trị hoặc tác vụ mà sẽ hoàn thành trong tương lai. Điều quan trọng là Future cho phép bạn thực hiện các tác vụ mà không cần chờ đợi chúng hoàn thành ngay lập tức.
* **Async:** Khai báo một hàm async cho biết rằng hàm đó sẽ chứa các tác vụ bất đồng bộ. Khi bạn gọi một tác vụ bất đồng bộ bên trong hàm này, bạn có thể sử dụng await để chờ kết quả của tác vụ đó mà không treo chương trình.
* **Await:** Khi bạn sử dụng await bên trong hàm async, chương trình sẽ tạm dừng thực hiện cho đến khi tác vụ bất đồng bộ hoàn thành. Việc này giúp tránh treo chương trình và cho phép bạn tiếp tục thực hiện các công việc khác trong thời gian chờ.

Khi giao diện người dùng bị đóng băng, người dùng sẽ không thể tương tác với ứng dụng và cảm thấy rất khó chịu. Điều này có thể dẫn đến trải nghiệm người dùng kém và khiến người dùng không muốn sử dụng ứng dụng của bạn nữa.



Hinh 1: Ảnh trực quan về việc bất đồng bộ

Các hoạt động không đồng bộ giúp cho chương trình của bạn hoàn thành công việc trong khi chờ hoạt động khác kết thúc. Một số ví dụ như là:

* Lấy data từ network.
* Ghi, update dữ liệu vào database.
* Đọc dữ liệu từ một file.

Các khái niệm quan trọng trong lập trình bất đồng bộ của Flutter :

* Future: Future là một đối tượng đại diện cho một giá trị hoặc một lỗi sẽ được trả về trong tương lai. Khi bạn muốn thực hiện một tác vụ bất đồng bộ, bạn có thể trả về một Future, đồng thời đưa các tác vụ đó vào trong Future để thực hiện.
* async: async là một từ khóa dùng để đánh dấu một hàm là bất đồng bộ. Khi sử dụng async, bạn có thể sử dụng await để chờ đợi cho một Future hoàn thành trước khi tiếp tục thực hiện các tác vụ tiếp theo trong hàm đó.
* await: await là một từ khóa dùng để đợi cho một Future hoàn thành. Khi bạn sử dụng await để đợi cho một Future, hàm đó sẽ bị tạm dừng cho đến khi Future hoàn thành. Khi Future hoàn thành, giá trị trả về hoặc lỗi sẽ được trả về và hàm sẽ tiếp tục thực hiện các tác vụ tiếp theo.

Khi muốn thực hiện một tác vụ bất đồng bộ như tải dữ liệu từ mạng, bạn có thể sử dụng Future để trả về kết quả tải dữ liệu đó. Để thực hiện tác vụ này trong một hàm bất đồng bộ, bạn có thể đánh dấu hàm đó bằng từ khóa async và sử dụng await để đợi cho Future hoàn thành.

### **2.3.1. Khái niệm Future**

Future là kết quả trả về của hoạt động bất đồng bộ. Future có hai trạng thái là : hoàn thành và chưa hoàn thành.

**Future hoàn thành** : Khi hoạt động bất đồng bộ thực hiện xong, lúc đó future ở trạng thái hoàn thành và trả về một giá trị hoặc một lỗi.

**Future chưa hoàn thành**: Khi hoạt động đồng bộ được gọi nó sẽ trả về một Future chưa hoàn thành.

### **2.3.2. Cách triển khai lập trình bất đồng bộ với Future, async, await**

Dựa vào ví dụ dưới đây bạn có thể hiểu được cách dùng : Future , await, async :  
+ Future : dùng cho kiểu dữ liệu trả về.  
+ async : được đặt trước khối chứa await, nó đánh dấu là bất đồng bộ.  
+ await : được đặt bên trong async và đặt ở trước các phương thức thực hiện việc load dữ liệu, hay ghi, update vào database.

# **CHƯƠNG 3: CÀI ĐẶT THỬ NGHIỆM**

## **3.1. Phương pháp nghiên cứu**

* + Tìm hiểu về Navigator và Router trong Flutter:

Đọc tài liệu chính thức của Flutter về Navigator và Router để hiểu cách chúng hoạt động và cung cấp những tính năng nào.

Tìm hiểu về cách Navigator quản lý các tệp tin, điều hướng giữa các màn hình và quản lý ngăn xếp của các màn hình.

* Xác định yêu cầu chức năng của ứng dụng:

Xác định các chức năng cần thiết trong ứng dụng liên quan đến Navigator, Router.

Đặt một danh sách các yêu cầu chức năng cơ bản mà ứng dụng cần thực hiện.

* Xây dựng ứng dụng theo từng bước:

Thêm các tương tác và chức năng yêu cầu vào từng màn hình.

Kiểm tra và xác nhận rằng điều hướng và tương tác hoạt động như mong đợi.

* Kiểm tra và sửa lỗi:

Kiểm tra ứng dụng để đảm bảo rằng Navigator và Router hoạt động chính xác và đáp ứng đúng yêu cầu chức năng.

Xác định và sửa lỗi nếu có …

## **3.2. Phương pháp thực hiện**

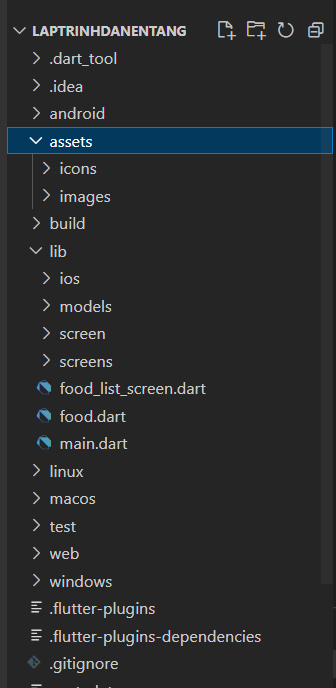
### **3.2.1. Cài đặt ứng dụng**

Để thực hiện cài đặt hoàn chỉnh cho việc phát triển đề tài chúng ta cần cài đặt :

* Flutter SDK: Flutter là một framework phát triển ứng dụng di động đa nền tảng. Bạn cần cài đặt Flutter SDK để phát triển ứng dụng Android. Bạn có thể tải SDK từ trang web chính thức của Flutter và cài đặt theo hướng dẫn tương ứng với hệ điều hành của bạn.
* Android Studio: Android Studio là môi trường phát triển tích hợp (IDE) cho Android. Nó cung cấp các công cụ và môi trường để phát triển ứng dụng Android. Bạn có thể tải Android Studio từ trang web chính thức của Android và cài đặt theo hướng dẫn cụ thể.
* Dart và Flutter Plugin: Sau khi cài đặt Android Studio, bạn cần cài đặt Dart và Flutter plugin để hỗ trợ viết mã Dart và phát triển ứng dụng Flutter. Để cài đặt plugin, hãy mở Android Studio, vào trình đơn "File" > "Settings" (hoặc "Preferences" trên macOS), chọn "Plugins", tìm kiếm "Dart" và "Flutter" và cài đặt plugin cho cả hai.
* Android Emulator hoặc Thiết bị Android: Để chạy và kiểm tra ứng dụng Android, bạn cần một thiết bị ảo (emulator) hoặc thiết bị Android thật. Trong Android Studio, bạn có thể tạo và quản lý các thiết bị ảo thông qua Android Virtual Device (AVD) Manager hoặc kết nối một thiết bị Android thật thông qua USB.
* Cài đặt môi trường Flutter: Để sử dụng Flutter từ Command Line/Command Prompt/Terminal, bạn cần cấu hình biến môi trường để có thể truy cập vào Flutter SDK từ bất kỳ vị trí nào. Hướng dẫn cấu hình biến môi trường được cung cấp trong tài liệu Flutter.

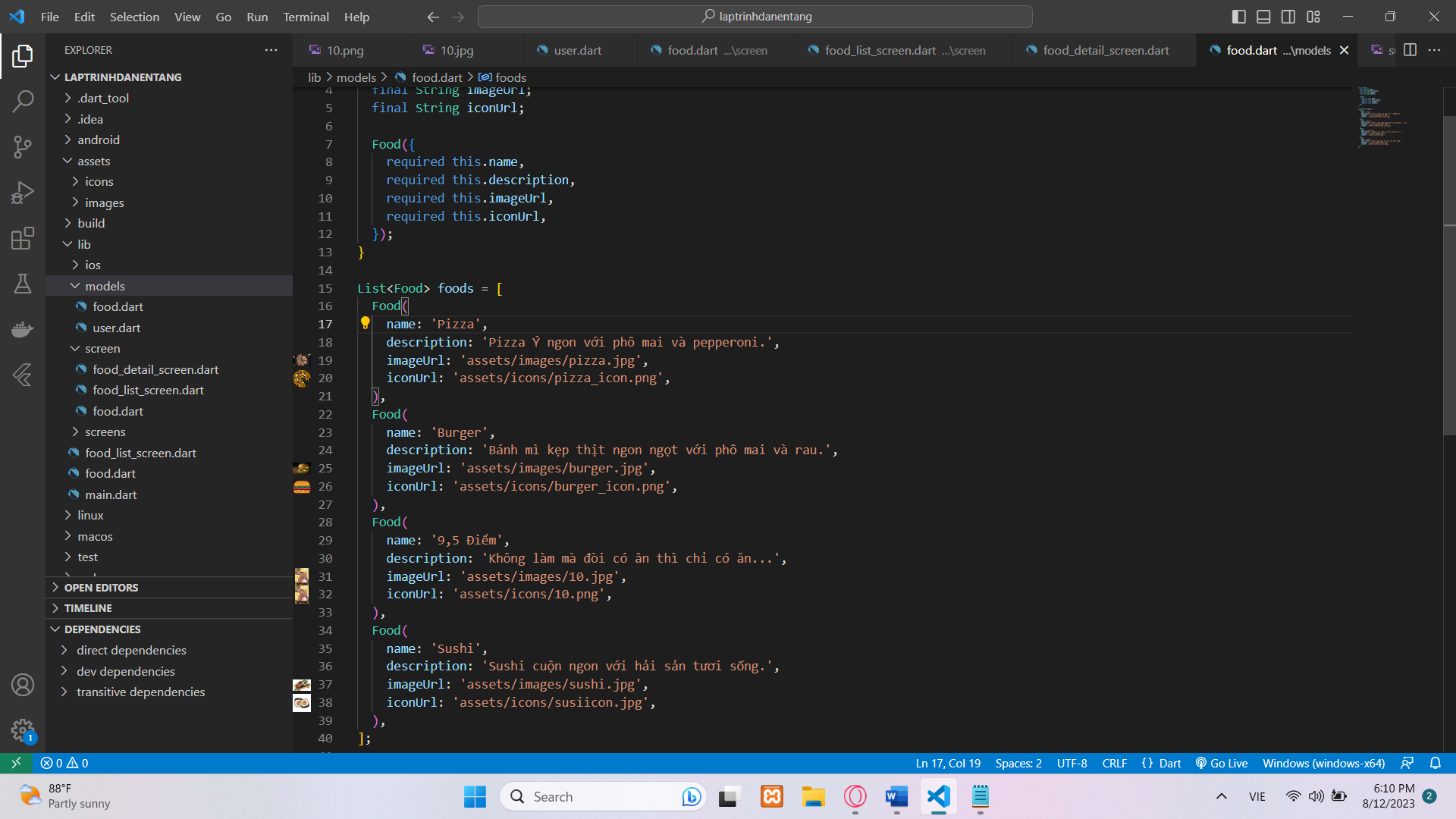
### **3.2.2. Phân tích và thiết kế hệ thống**

Hệ thống thiết kế dựa theo cấu trúc mặc định của các file thiết lập sẵn và theo sự phát triển của nhóm.



Hinh 2: Cấu trúc thư mục

Cấu trúc thư mục con để dễ dàng tìm kiếm, chỉnh sửa và quản lý các màn hình của ứng dụng một cách hiệu quả.



Hinh 3: food.dart

Đây là một đoạn mã định nghĩa lớp Food và danh sách các món ăn (foods). Trong đoạn mã này:

Food là một lớp đại diện cho thông tin về một món ăn.

name: Tên món ăn.

description: Mô tả về món ăn.

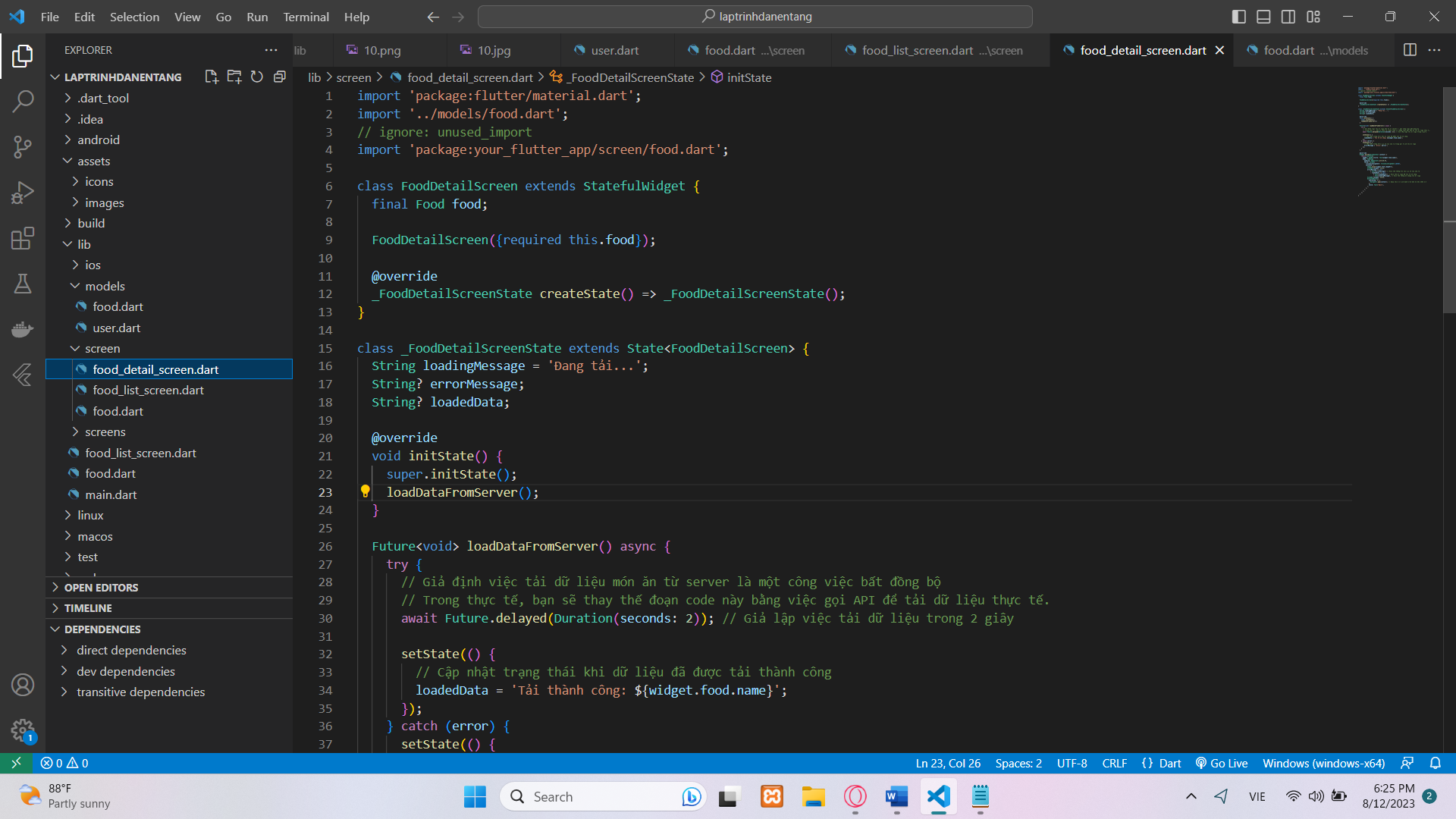
imageUrl: Đường dẫn tới hình ảnh món ăn.

iconUrl: Đường dẫn tới biểu tượng (icon) của món ăn.

List<Food> foods là một danh sách các đối tượng Food, đại diện cho danh sách các món ăn. Mỗi đối tượng Food được khởi tạo với các thông tin như tên, mô tả, đường dẫn hình ảnh và đường dẫn biểu tượng.

Ví dụ, danh sách món ăn đã được định nghĩa bao gồm Pizza, Burger, một món có tên "9,5 Điểm" và Sushi, mỗi món ăn có các thông tin tương ứng.

Có thể sử dụng danh sách này để hiển thị thông tin về các món ăn trong ứng dụng của mình, chẳng hạn như hiển thị danh sách món ăn trên màn hình chính hoặc trong các màn hình khác.



Hinh 4: food\_detail\_screen.dart

Đoạn mã này đề xuất trang chi tiết món ăn trong ứng dụng, nơi có thể hiển thị thông tin chi tiết về một món ăn cụ thể.

Import và Class Definition:

Đoạn đầu của mã là phần khai báo import, nơi bạn import các thư viện và tệp cần thiết cho trang.

FoodDetailScreen là một StatefulWidget, có một thuộc tính là Food để truyền dữ liệu món ăn vào trang.

initState():

Phương thức initState() được gọi khi trang được tạo ra. Trong trường hợp này, nó được sử dụng để gọi phương thức loadDataFromServer() để tải dữ liệu từ máy chủ.

loadDataFromServer():

Phương thức này là nơi bạn thực hiện công việc tải dữ liệu từ máy chủ. Trong ví dụ này, chúng ta giả lập việc tải dữ liệu bằng cách sử dụng Future.delayed để đợi 2 giây trước khi cập nhật dữ liệu đã tải thành công hoặc thông báo lỗi.

Nếu việc tải thành công, setState được sử dụng để cập nhật loadedData với thông báo tải thành công.

build(BuildContext context):

Phương thức này xây dựng giao diện của trang.

Một Scaffold chứa một AppBar với tiêu đề là tên của món ăn.

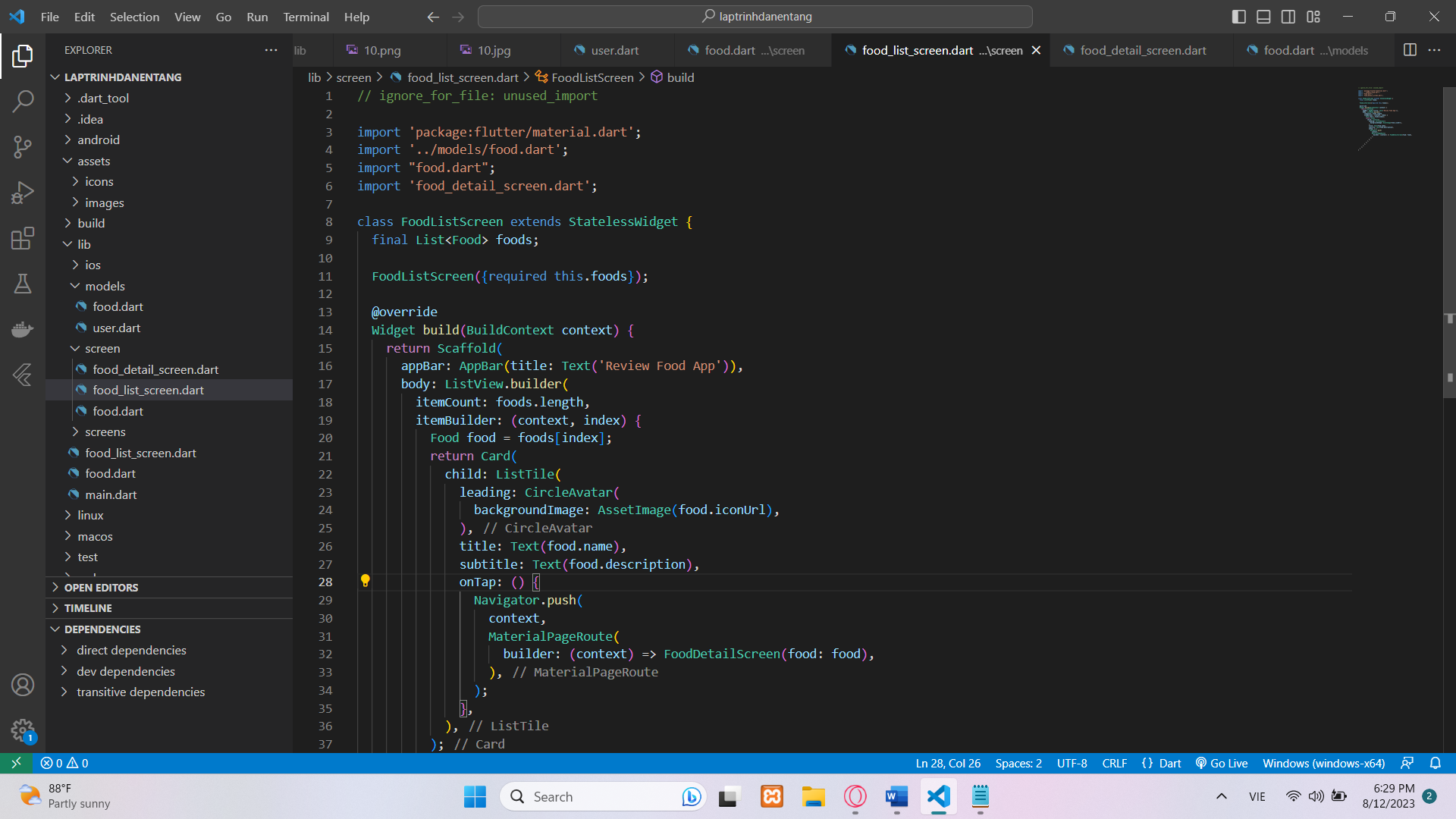
Phần thân của trang chứa một cột với các thành phần sau:

Image.asset: Hiển thị hình ảnh của món ăn bằng cách sử dụng đường dẫn hình ảnh được cung cấp từ đối tượng widget.food.

Thông báo: Hiển thị một trong ba thông báo: thông báo lỗi nếu có errorMessage, thông báo tải thành công nếu có loadedData, hoặc thông báo đang tải dữ liệu nếu không có dữ liệu nào.

Nút "Back": Nút này cho phép quay lại màn hình danh sách món ăn khi người dùng nhấn.

Thông qua mã nguồn trên, có thể thấy cách sử dụng Future, async, và await để thực hiện việc tải dữ liệu bất đồng bộ từ máy chủ. Nó cho phép cập nhật giao diện người dùng một cách thích hợp dựa trên kết quả của việc tải dữ liệu.



Hinh 5: food\_list\_screen.dart

Đoạn mã này định nghĩa một FoodListScreen là một widget không có trạng thái (stateless) để hiển thị danh sách các món ăn. Hãy xem xét từng phần của mã:

final List<Food> foods; thuộc tính foods là danh sách các món ăn được truyền vào khi khởi tạo widget FoodListScreen.

Phần build(BuildContext context) chứa một Scaffold để hiển thị giao diện. Scaffold chứa AppBar và một ListView.builder để hiển thị danh sách các món ăn.

ListView.builder được sử dụng để hiển thị danh sách món ăn. Mỗi món ăn được hiển thị trong một Card bên trong ListTile.

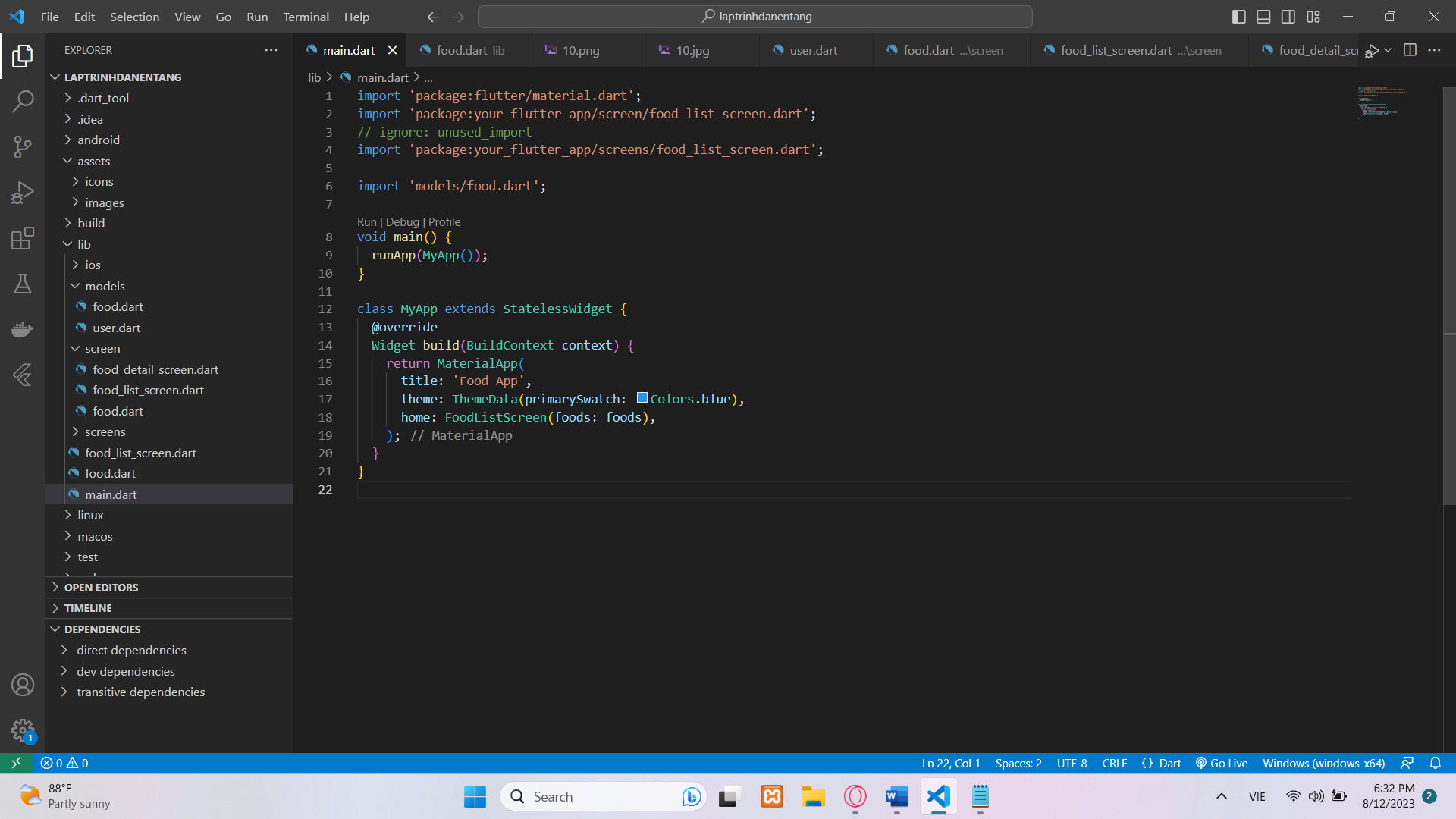
Trong phần itemBuilder, mỗi món ăn được lấy từ danh sách foods theo chỉ mục index.

Trong ListTile, bạn sử dụng CircleAvatar để hiển thị biểu tượng (icon) của món ăn bằng cách sử dụng AssetImage với đường dẫn từ thuộc tính iconUrl.

Tiêu đề và mô tả của món ăn được hiển thị bằng Text với dữ liệu tương ứng từ name và description.

Khi người dùng nhấn vào một món ăn trong danh sách, hàm onTap sẽ được gọi. Trong đó, sử dụng Navigator để chuyển hướng tới màn hình FoodDetailScreen và truyền đối tượng food đang được chọn.

Như vậy, đoạn mã này sẽ hiển thị danh sách các món ăn trong giao diện và cho phép người dùng nhấn vào một món ăn để xem chi tiết. Khi người dùng nhấn vào một món ăn, màn hình FoodDetailScreen sẽ được hiển thị và hiển thị chi tiết về món ăn đó.



Hinh 6: main.dart

void main(): Đây là hàm main() của ứng dụng. Đây là điểm khởi đầu của ứng dụng Flutter.

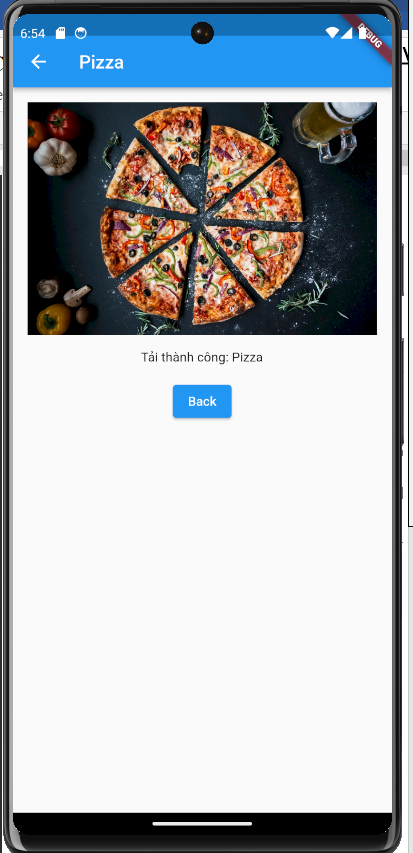
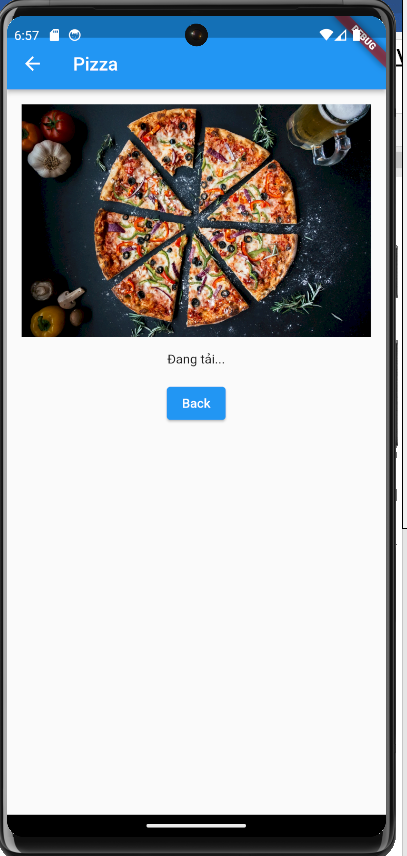
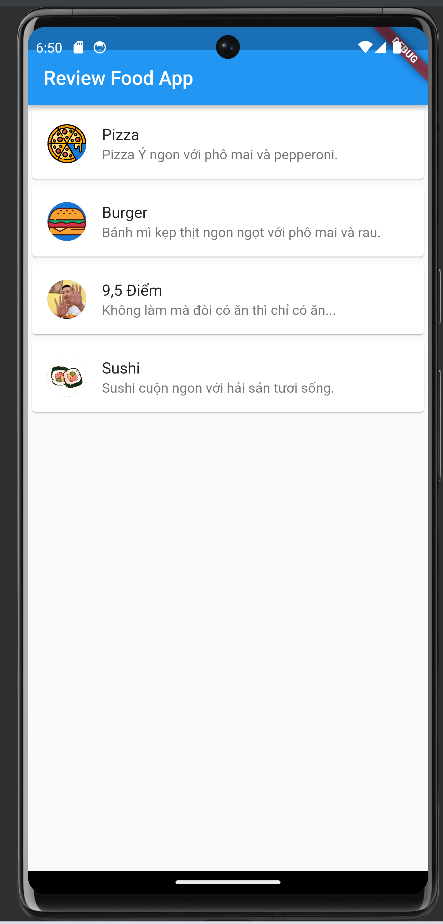
runApp(MyApp());: Trong hàm main(), gọi runApp() để chạy ứng dụng. Ở đây, chạy widget MyApp (được định nghĩa ở phần tiếp theo) làm màn hình chính của ứng dụng.

class MyApp extends StatelessWidget: Đây là lớp MyApp mở rộng từ StatelessWidget. Lớp này là widget chính của ứng dụng.

Trong phương thức build(BuildContext context), trả về một MaterialApp để làm giao diện chính của ứng dụng. MaterialApp chứa thông tin về chủ đề (theme) của ứng dụng và cài đặt màn hình chính là FoodListScreen (danh sách món ăn).

Đoạn mã này khởi chạy ứng dụng của bạn bằng cách tạo một MyApp (widget chính) và cài đặt màn hình chính là FoodListScreen để hiển thị danh.

**Kết Quả:**



Hinh 7: Giao diện app

# **CHƯƠNG 4 : KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN**

## **4.1. Kết luận**

Tiến bộ liên tục trong lĩnh vực công nghệ đang thúc đẩy sự phát triển của ứng dụng di động và khung làm việc Flutter của Google đã tạo ra một cách tiếp cận đầy thú vị trong việc phát triển các ứng dụng đa nền tảng. Trong quá trình nghiên cứu và khám phá về khả năng của Flutter, chúng ta đã tìm hiểu về hai khái niệm quan trọng: Navigator và Router, cùng với khả năng lập trình bất đồng bộ.

Navigator, như một bộ điều hướng linh hoạt, đóng vai trò quản lý tuyến đường và màn hình trong ứng dụng. Nó cho phép chúng ta dễ dàng chuyển đổi giữa các giao diện người dùng mà không cần phải lo lắng về việc quản lý trạng thái và hiển thị. Thông qua việc thực hiện các tuyến đường, Navigator giúp chia nhỏ ứng dụng thành các thành phần nhỏ hơn, dễ quản lý hơn và giúp tách biệt logic của từng màn hình.

Router, một khái niệm liên quan, cung cấp cách xác định cách hiển thị màn hình khi chuyển đổi giữa các tuyến đường. Điều này cho phép chúng ta xác định cách một màn hình cụ thể sẽ được tạo ra và hiển thị. Chúng ta có thể tùy chỉnh các hiệu ứng chuyển đổi, quản lý trạng thái và chia sẻ dữ liệu qua Router, tạo ra trải nghiệm người dùng mượt mà và linh hoạt.

Lập trình bất đồng bộ cũng đã nêu lên khả năng thay đổi cách chúng ta tiếp cận các tác vụ chậm và tốn thời gian. Với sự hỗ trợ của Future, async, và await, chúng ta có khả năng thực hiện các tác vụ không đồng bộ mà không làm ảnh hưởng đến hiệu suất và trải nghiệm người dùng. Điều này đặc biệt quan trọng trong việc tải dữ liệu, xử lý yêu cầu mạng và các tác vụ tương tự.

Áp dụng những kiến thức trên vào đề tài “Viết app minh họa nội dung Navigator, Router” đã tạo được cho nhóm chúng em cơ hội trải nghiệm thực tế những gì đã được học

Thông qua việc xây dựng ứng dụng minh họa, chúng em đã hiểu được:

* Navigator trong Flutter là một công cụ mạnh để quản lý và điều hướng giữa các màn hình trong ứng dụng. Router giúp xác định cách hiển thị và tạo các tuyến đường giữa các màn hình, giúp xây dựng cấu trúc điều hướng linh hoạt và rõ ràng.
* Sử dụng thuộc tính arguments của RouteSettings, chúng ta có thể truyền dữ liệu từ một màn hình gốc sang một màn hình mới. Điều này giúp tạo giao tiếp giữa các màn hình và chia sẻ thông tin quan trọng.
* Sử dụng MaterialPageRoute, chúng ta có thể tạo hiệu ứng chuyển đổi tương thích với nguyên tắc thiết kế Material Design. Hiệu ứng này giúp ứng dụng có trải nghiệm chuyển đổi mượt mà và thú vị.
* Sử dụng Future, async, và await, chúng ta có thể thực hiện các tác vụ bất đồng bộ mà không làm treo chương trình. Điều này cải thiện hiệu suất ứng dụng và trải nghiệm người dùng.
* Sử dụng Navigator và Router, chúng ta có thể phân chia logic ứng dụng thành các phần nhỏ và dễ quản lý, mỗi phần đại diện cho một tuyến đường và màn hình cụ thể.
* Router cho phép chúng ta quản lý trạng thái của các màn hình và tùy chỉnh cách màn hình mới sẽ hiển thị, giúp duy trì sự liên kết giữa các màn hình và dữ liệu.

Tóm lại, việc hiểu và sử dụng Navigator và Router trong Flutter giúp bạn xây dựng ứng dụng có cấu trúc tốt, trải nghiệm người dùng tốt hơn và quản lý các tác vụ một cách hiệu quả. Bằng cách tận dụng các khái niệm này, chúng ta có thể tạo ra các ứng dụng đa dạng và hấp dẫn trên nền tảng Flutter.

## **4.2. Hướng phát triển**

Hướng phát triển tương lai của đề tài "Viết App Minh Họa Nội Dung Navigator và Router trong Flutter" tiềm ẩn nhiều cơ hội và khả năng cải tiến để tối ưu hóa trải nghiệm người dùng và phát triển ứng dụng. Dưới đây là một số hướng phát triển đáng xem xét:

Tối Ưu Hiệu Năng và Tốc Độ: Việc tối ưu hiệu năng và tốc độ của Navigator và Router là một phần quan trọng. Cân nhắc sử dụng các thư viện và kỹ thuật tối ưu hóa để đảm bảo rằng chuyển đổi giữa các màn hình diễn ra một cách mượt mà và nhanh chóng.

Chia Sẻ Dữ Liệu Đa Dạng Hơn: Mở rộng khả năng truyền dữ liệu giữa các màn hình để hỗ trợ nhiều loại dữ liệu khác nhau, bao gồm các đối tượng phức tạp, danh sách và hình ảnh. Điều này đảm bảo rằng ứng dụng có thể chia sẻ thông tin linh hoạt giữa các màn hình một cách dễ dàng.

Kết Hợp Với Các Kỹ Thuật Đồ Họa Khác: Kết hợp việc sử dụng Navigator và Router với các kỹ thuật đồ họa khác như animations và transitions có thể tạo ra trải nghiệm người dùng thú vị hơn. Hiệu ứng chuyển đổi và thay đổi giao diện người dùng sẽ đem lại sự hấp dẫn và tạo điểm nhấn cho ứng dụng.

# **DANH MỤC TÀI LIỆU THAM KHẢO**

<https://docs.flutter.dev/>

<https://docs.flutter.dev/codelabs>

<https://github.com/flutter/samples>

<https://stackoverflow.com/questions/tagged/flutter>

<https://www.youtube.com/results?search_query=flutter+tutorials>