

TRƯỜNG ĐẠI HỌC TRÀ VINH  
KHOA KỸ THUẬT VÀ CÔNG NGHỆ



ISO 9001:2015

NGUYỄN NHẤT SANG

Nghiên cứu nền tảng Headless  
Ecommerce Saleor, ngôn ngữ GraphQL  
và xây dựng trang thương mại điện tử

ĐỒ ÁN TỐT NGHIỆP  
NGÀNH CÔNG NGHỆ THÔNG TIN

TRÀ VINH, NĂM 2024

TRƯỜNG ĐẠI HỌC TRÀ VINH  
KHOA KỸ THUẬT VÀ CÔNG NGHỆ

**NGHIÊN CỨU NỀN TẢNG HEADLESS  
ECOMMERCE SALEOR, NGÔN NGỮ GRAPHQL  
VÀ XÂY DỰNG TRANG THƯƠNG MẠI ĐIỆN TỬ**

**ĐỒ ÁN TỐT NGHIỆP  
NGÀNH CÔNG NGHỆ THÔNG TIN**

Sinh viên: Nguyễn Nhất Sang

Lớp: DA20TTB

MSSV: 110120151

GVHD: TS. Nguyễn Bảo Ân

TRÀ VINH, NĂM 2024

## LỜI MỞ ĐẦU

Trong bối cảnh nền kinh tế toàn cầu ngày càng phát triển, việc ứng dụng các công nghệ mới vào quản lý và kinh doanh trở thành yếu tố then chốt giúp doanh nghiệp nâng cao hiệu quả hoạt động và năng lực cạnh tranh. Xuất phát từ nhu cầu thực tiễn này, tôi đã chọn nghiên cứu đề tài “Nghiên cứu nền tảng Headless Ecommerce Saleor, ngôn ngữ GraphQL và xây dựng trang thương mại điện tử”.

Đồ án tốt nghiệp này nhằm mục đích nghiên cứu và phát triển một ứng dụng quản lý cửa hàng thú cưng, sử dụng công nghệ Flutter cho giao diện người dùng và Saleor cho hệ thống backend. Ứng dụng này không chỉ tối ưu hóa quy trình quản lý sản phẩm, đơn hàng và khách hàng mà còn nâng cao trải nghiệm người dùng.

Đối tượng nghiên cứu bao gồm nền tảng Headless Ecommerce Saleor, ngôn ngữ GraphQL và framework Flutter. Phạm vi nghiên cứu tập trung vào việc triển khai Saleor và GraphQL để xây dựng backend cho hệ thống thương mại điện tử, và sử dụng Flutter để phát triển frontend. Phương pháp nghiên cứu bao gồm thu thập thông tin từ các tài liệu học thuật, bài báo khoa học và tài liệu hướng dẫn của Saleor, GraphQL và Flutter; cài đặt, cấu hình Saleor, triển khai API sử dụng GraphQL và xây dựng frontend với Flutter.

Cấu trúc của khóa luận được chia thành các chương như sau:

- ❖ Chương 1: Đặt vấn đề
- ❖ Chương 2: Cơ sở lý thuyết
- ❖ Chương 3: Hiện thực hóa nghiên cứu
- ❖ Chương 4: Kết quả nghiên cứu
- ❖ Chương 5: Kết luận và hướng phát triển

Tôi xin chân thành cảm ơn sự hướng dẫn tận tình của thầy Nguyễn Bảo Ân cùng sự giúp đỡ quý báu từ bạn bè và gia đình, đã tạo điều kiện cho tôi hoàn thành đồ án tốt nghiệp.

## LỜI CẢM ƠN

Trước hết, tôi xin gửi lời cảm ơn chân thành đến thầy Nguyễn Bảo Ân, giảng viên Khoa Kỹ thuật & Công nghệ, người đã tận tình hướng dẫn tôi trong suốt quá trình thực hiện đồ án tốt nghiệp.

Tôi xin bày tỏ lòng biết ơn sâu sắc đến toàn thể các Thầy (Cô) trong Trường Đại học Trà Vinh nói chung và các Thầy (Cô) trong Khoa Kỹ thuật & Công nghệ nói riêng, đã truyền đạt cho tôi kiến thức đầy đủ về các môn học đại cương cũng như chuyên ngành, giúp tôi có được nền tảng lý thuyết vững chắc và luôn tạo điều kiện thuận lợi cho tôi trong suốt quá trình học tập.

Cuối cùng, trong quá trình tìm hiểu và thực hiện đề tài đồ án tốt nghiệp, tôi vẫn còn nhiều hạn chế và gặp không ít khó khăn. Vì vậy, tôi rất mong nhận được những ý kiến đóng góp từ quý Thầy (Cô) để có thể hoàn thiện hơn.

Trà Vinh, ngày ..... tháng ... năm 2024

Sinh viên thực hiện

Nguyễn Nhất Sang

**NHẬN XÉT**  
**(Của giảng viên hướng dẫn trong đồ án, khoá luận của sinh viên)**

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

---

**Giảng viên hướng dẫn**  
(ký và ghi rõ họ tên)

BẢN NHẬN XÉT ĐỒ ÁN, KHÓA LUẬN TỐT NGHIỆP  
(Của giảng viên hướng dẫn)

Họ và tên sinh viên: ..... MSSV: .....

Ngành: ..... Khóa: .....

Tên đê tài: .....

Họ và tên Giáo viên hướng dẫn: .....

Chức danh: ..... Học vị: .....

**NHẬN XÉT**

1. Nội dung đê tài:

.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....

2. Ưu điểm:

.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....

3. Khuyết điểm:

.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....

4. Điểm mới đề tài:

.....  
.....  
.....  
.....  
.....

5. Giá trị thực trên đề tài:

.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....

7. Đề nghị sửa chữa bổ sung:

.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....  
.....

8. Đánh giá:

.....  
.....  
.....  
.....  
.....

Trà Vinh, ngày ..... tháng ..... năm 2024  
Giảng viên hướng dẫn  
(Ký & ghi rõ họ tên)

## MỤC LỤC

<b>CHƯƠNG 1. ĐẶT VẤN ĐỀ.....</b>	<b>1</b>
1.1. Lý do chọn đề tài.....	1
1.2. Mục tiêu .....	1
1.3. Nội dung cấu trúc báo cáo.....	1
1.4. Đối tượng và phạm vi nghiên cứu.....	1
1.5. Phương pháp nghiên cứu.....	2
<b>CHƯƠNG 2. CƠ SỞ LÝ THUYẾT.....</b>	<b>3</b>
2.1. Saleor .....	3
2.1.1. Giới thiệu Saleor .....	3
2.1.2. Kiến trúc của Saleor .....	3
2.1.3. Các tính năng của Saleor .....	4
2.1.4. Xác thực người dùng trong Saleor .....	7
2.2. GraphQL .....	8
2.2.1. Giới thiệu GraphQL .....	8
2.2.2. Các thành phần của GraphQL .....	9
2.2.3. So sánh GraphQL với REST API .....	12
2.3. Flutter .....	14
2.3.1. Giới thiệu chung Flutter .....	14
2.3.2. Kiến trúc của Flutter.....	15
2.3.3. Ngôn ngữ lập trình Dart .....	17
2.3.4. Quản lý trạng thái ứng dụng.....	19
<b>CHƯƠNG 3. HIỆN THỰC HÓA NGHIÊN CỨU.....</b>	<b>21</b>
3.1. Quy trình thực hiện .....	21
3.2. Cài đặt môi trường .....	21
3.2.1. Saleor backend .....	21
3.2.2. Flutter .....	26
3.3. Xây dựng giao diện .....	26
3.4. Thiết kế API .....	26
3.4.1. API đăng ký .....	26
3.4.2. API đăng nhập.....	26
3.4.3. API đăng xuất.....	27
3.4.4. API lấy danh sách category.....	27
3.4.5. API lấy danh sách sản phẩm theo category.....	27
3.4.6. API tạo phiếu thanh toán.....	27
3.4.7. API thêm sản phẩm vào giỏ hàng .....	27

3.4.8. API lấy thông tin người dùng đăng nhập .....	27
3.4.9. API lấy thông tin giỏ hàng .....	27
3.4.10. API cập nhật thông tin tài khoản.....	28
3.4.11. API thêm địa chỉ .....	28
3.4.12. API cập nhật địa chỉ giao hàng và địa chỉ thanh toán .....	28
3.5. Tích hợp Flutter với Saleor qua GraphQL .....	28
3.5.1. Cấu hình GraphQL Client .....	29
3.5.2. Sử dụng GraphQL Client .....	29
3.6. Triển khai và bảo trì .....	29
<b>CHƯƠNG 4. KẾT QUẢ NGHIÊN CỨU .....</b>	<b>31</b>
4.1. Lưu token và sử dụng token của người dùng .....	31
4.1.1. Đăng nhập và nhận token.....	31
4.1.2. Lưu token vào SharedPreferences.....	31
4.1.3. Sử dụng token để xác thực yêu cầu tiếp theo.....	32
4.1.4. Làm mới token (Refresh Token).....	32
4.1.5. Đăng xuất .....	32
4.2. Giao diện trên web .....	33
4.3. Giao diện trên Android.....	34
4.4. Giao diện trên IOS .....	35
4.4.1. Đăng nhập .....	35
4.4.2. Đăng ký .....	36
4.4.3. Trang chủ .....	37
4.4.4. Chi tiết sản phẩm.....	38
4.4.5. Giỏ hàng .....	39
4.4.6. Lịch sử đơn hàng.....	40
4.4.7. Thông tin cá nhân.....	41
4.5. Chức năng .....	42
4.5.1. Đăng nhập .....	42
4.5.2. Đăng ký .....	43
4.5.3. Trang chủ .....	45
4.5.4. Chi tiết sản phẩm.....	47
4.5.5. Giỏ hàng .....	48
4.5.6. Lịch sử đơn hàng.....	49
4.5.7. Thông tin cá nhân.....	50
<b>CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN.....</b>	<b>52</b>
5.1. Kết luận .....	52

5.2. Hướng phát triển .....	52
<b>DANH MỤC TÀI LIỆU THAM KHẢO.....</b>	<b>53</b>

## **PHỤ LỤC**

PHỤ LỤC 1. API đăng ký .....	54
PHỤ LỤC 2. API đăng nhập.....	56
PHỤ LỤC 3. API đăng xuất.....	58
PHỤ LỤC 4. API lấy danh sách category.....	60
PHỤ LỤC 5. API lấy danh sách sản phẩm theo category.....	62
PHỤ LỤC 6. API tạo phiếu thanh toán.....	64
PHỤ LỤC 7. API thêm sản phẩm vào giỏ hàng .....	66
PHỤ LỤC 8. API lấy thông tin người dùng đăng nhập .....	68
PHỤ LỤC 9. API lấy thông tin giỏ hàng .....	71
PHỤ LỤC 10. API cập nhật thông tin tài khoản.....	73
PHỤ LỤC 11. API thêm địa chỉ .....	75
PHỤ LỤC 12. API cập nhật địa chỉ giao hàng và địa chỉ thanh toán .....	77

## **DANH MỤC CÁC BẢNG, SƠ ĐỒ, HÌNH**

Hình 2. 1. Kiến trúc Saleor.....	4
Hình 2. 2. GraphQL .....	8
Hình 2. 3. Ví dụ Schema .....	10
Hình 2. 4. Ví dụ Resolver .....	10
Hình 2. 5. Ví dụ Query.....	11
Hình 2. 6. Ví dụ Mutation .....	11
Hình 2. 7. Ví dụ Subscription .....	12
Hình 2. 8. Truy vấn dữ liệu của GraphQL và REST .....	12
Hình 2. 9. Flutter .....	14
Hình 2. 10. Kiến trúc của Flutter.....	15
Hình 2. 11. Dart.....	17
Hình 2. 12. Cấu trúc thư mục dự án Flutter .....	17
Hình 3. 1. Quy trình thực hiện nghiên cứu.....	21
Hình 3. 2. Đăng ký Saleor Cloud .....	22
Hình 3. 3. Tạo dự án mới .....	22
Hình 3. 4. Cấu hình dự án .....	23
Hình 3. 5. Saleor API URL .....	23
Hình 3. 6. Saleor dashboard .....	24
Hình 3. 7. Saleor Playground .....	24
Hình 3. 8. Thêm dependencies .....	28
Hình 3. 9. Cấu hình GraphQL Client .....	29
Hình 3. 10. Sử dụng GraphQL Client .....	29
Hình 4. 1. Đăng nhập và nhận token .....	31
Hình 4. 2. Lưu token vào bộ nhớ cục bộ .....	31
Hình 4. 3. Sử dụng token để xác thực các yêu cầu tiếp theo.....	32
Hình 4. 4. Làm mới token .....	32
Hình 4. 5. Xóa token .....	32
Hình 4. 6. Giao diện trên web .....	33
Hình 4. 7. Giao diện trên Android.....	34
Hình 4. 8. Giao diện đăng nhập.....	35
Hình 4. 9. Giao diện đăng ký .....	36
Hình 4. 10. Giao diện trang chủ .....	37
Hình 4. 11. Giao diện chi tiết sản phẩm .....	38
Hình 4. 12. Giao diện giỏ hàng .....	39
Hình 4. 13. Giao diện lịch sử đơn hàng .....	40
Hình 4. 14. Giao diện thông tin cá nhân .....	41
Hình 4. 15. Chức năng đăng nhập.....	42
Hình 4. 16. Chức năng đăng ký.....	43
Hình 4. 17. Chức năng trang chủ .....	45
Hình 4. 18. Chức năng chi tiết sản phẩm .....	47
Hình 4. 19. Chức năng giỏ hàng .....	48
Hình 4. 20. Chức năng lịch sử đơn hàng.....	49
Hình 4. 21. Chức năng thông tin cá nhân.....	50

## **DANH MỤC TỪ VIẾT TẮT**

Từ viết tắt	Ý nghĩa
AOT	Ahead-of-Time
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifier
IOS	iPhone Operating System
JIT	Just-in-Time
JSON	JavaScript Object Notation
JWT	JSON Web Token
OOP	Object-Oriented Programming
REST	Representational State Transfer
SDK	Software Development Kit
SDL	Schema Definition Language
UI	User Interface
URL	Uniform Resource Locator
VM	Virtual Machine

# CHƯƠNG 1. ĐẶT VẤN ĐỀ

## 1.1. Lý do chọn đề tài

Trong bối cảnh phát triển nhanh chóng của thương mại điện tử, việc xây dựng các nền tảng bán hàng trực tuyến hiệu quả, linh hoạt và dễ dàng mở rộng trở nên vô cùng quan trọng. Headless Ecommerce là một xu hướng mới, tách biệt giữa backend và frontend, mang lại nhiều lợi ích như tính linh hoạt, dễ dàng tùy chỉnh giao diện người dùng và cải thiện hiệu suất. Saleor là một trong những nền tảng Headless Ecommerce tiên tiến, kết hợp với ngôn ngữ GraphQL để cung cấp API mạnh mẽ và linh hoạt. Việc nghiên cứu và ứng dụng Saleor cùng GraphQL trong việc xây dựng trang thương mại điện tử không chỉ giúp nâng cao kiến thức chuyên môn mà còn đáp ứng yêu cầu thực tế của thị trường. Flutter, một framework phát triển frontend mạnh mẽ, giúp hiện thực hóa các lợi ích của headless eCommerce bằng cách tạo ra giao diện người dùng mượt mà và hiệu quả.

## 1.2. Mục tiêu

Sử dụng backend sẵn có của Headless Ecommerce Saleor, thiết kế API bằng GraphQL và kết nối với giao diện ứng dụng được xây dựng bằng Flutter phát triển ứng dụng thương mại điện tử bán các sản phẩm cho thú cưng.

## 1.3. Nội dung cấu trúc báo cáo

Khóa luận này bao gồm các nội dung chính sau:

Chương 1: Tổng quan về headless eCommerce và tầm quan trọng của nó trong lĩnh vực thương mại điện tử.

Chương 2: Giới thiệu và phân tích nền tảng Saleor cùng ngôn ngữ GraphQL, Flutter

Chương 3: Quy trình và phương pháp xây dựng trang thương mại điện tử sử dụng Saleor, GraphQL và Flutter.

Chương 4: Thực nghiệm và đánh giá kết quả xây dựng trang thương mại điện tử.

Chương 5: Kết luận và đề xuất hướng phát triển.

## 1.4. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu: Nền tảng headless eCommerce Saleor, ngôn ngữ GraphQL và framework Flutter.

Phạm vi nghiên cứu: Tập trung vào việc triển khai Saleor và GraphQL để xây dựng backend cho hệ thống thương mại điện tử, sử dụng Flutter để phát triển frontend.

## **1.5. Phương pháp nghiên cứu**

Phương pháp thu thập tài liệu: Tìm hiểu thông tin từ các tài liệu học thuật, bài báo khoa học, và tài liệu hướng dẫn của Saleor, GraphQL và Flutter.

Phương pháp thực nghiệm: Tiến hành cài đặt và cấu hình Saleor, triển khai API sử dụng GraphQL và xây dựng frontend với Flutter.

## CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

### 2.1. Saleor

#### 2.1.1. Giới thiệu Saleor



Hình 2. Saleor

Saleor là một nền tảng e-commerce mã nguồn mở được thiết kế dưới dạng headless. Điều này có nghĩa là Saleor tách biệt hoàn toàn phần backend và frontend, cho phép các nhà phát triển có thể xây dựng giao diện người dùng riêng biệt mà vẫn sử dụng hệ thống quản lý dữ liệu mạnh mẽ của Saleor. Được xây dựng trên Django và Python, Saleor tận dụng các công nghệ tiên tiến để cung cấp một hệ thống quản lý thương mại điện tử linh hoạt và dễ mở rộng[1].

Headless là kiến trúc phần mềm tách biệt phần giao diện người dùng (frontend) khỏi phần xử lý logic và dữ liệu (backend). Trong một hệ thống Headless, API là trung tâm kết nối mọi thành phần trong hệ thống, backend cung cấp các API để truy cập và quản lý dữ liệu, trong khi frontend có thể được xây dựng bằng bất kỳ công nghệ nào và kết nối với backend thông qua các API này.

Thương mại điện tử headless đang trở thành xu hướng chủ đạo trong lĩnh vực bán lẻ trực tuyến. Sự phát triển của các công nghệ mới như Jamstack, serverless computing, và microservices càng thúc đẩy sự phổ biến của kiến trúc này. Dự đoán trong tương lai, headless commerce sẽ tiếp tục được hoàn thiện và ứng dụng rộng rãi hơn, đáp ứng nhu cầu ngày càng cao của người tiêu dùng và doanh nghiệp.

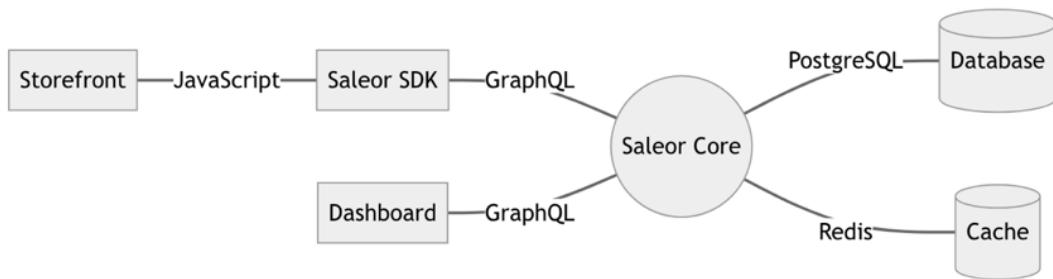
#### 2.1.2. Kiến trúc của Saleor

**Microservices:** Saleor chia nhỏ các chức năng thành các microservices độc lập, giúp hệ thống dễ dàng mở rộng và bảo trì. Mỗi microservice chịu trách nhiệm cho một phần cụ thể của hệ thống, ví dụ như dịch vụ thanh toán, dịch vụ vận chuyển, dịch vụ email.

**GraphQL API:** GraphQL là một ngôn ngữ truy vấn cho API, được sử dụng để tương tác với dữ liệu của Saleor. GraphQL cung cấp một cách linh hoạt để truy xuất và cập nhật dữ liệu, cho phép client chỉ yêu cầu những dữ liệu cần thiết, tránh tình trạng over-fetching hoặc under-fetching.

### ❖ Các thành phần chính

Saleor áp dụng kiến trúc phân tán, chia thành ba thành phần chính tương tác với nhau thông qua GraphQL API trên nền HTTPS[1]:



Hình 2. 1. Kiến trúc Saleor

**Saleor Core:** Đóng vai trò làm máy chủ backend của Saleor cung cấp GraphQL API, chịu trách nhiệm xử lý logic nghiệp vụ, quản lý dữ liệu. Core được viết bằng Python và sử dụng PostgreSQL làm cơ sở dữ liệu và một phần thông tin được lưu vào bộ nhớ đệm Redis để tăng tốc độ truy xuất.

**Saleor Dashboard:** Thành phần cung cấp giao diện quản trị cho phép nhân viên vận hành cửa hàng. Được xây dựng dưới dạng ứng dụng React chạy trên trình duyệt, Saleor Dashboard tương tác với Saleor Core thông qua GraphQL API. Dashboard được thiết kế là một trang web tĩnh, không chứa mã back-end.

**Saleor Storefront:** Đây là một ví dụ storefront được Saleor cung cấp, triển khai bằng React với Next.js. Nhà phát triển có thể tùy chỉnh mã nguồn của storefront này hoặc xây dựng storefront riêng dựa trên Saleor SDK.

#### 2.1.3. Các tính năng của Saleor

##### 2.1.3.1. Quản lý danh mục sản phẩm

Cấu trúc phân cấp: Cho phép tạo các danh mục sản phẩm theo cấp bậc, tổ chức sản phẩm một cách logic và dễ tìm kiếm. Ví dụ: Danh mục chính > Danh mục con > Sản phẩm.

Biến thể sản phẩm:

- Kích thước: Quản lý các kích cỡ khác nhau của một sản phẩm (S, M, L, XL).
- Màu sắc: Quản lý các màu sắc khác nhau của một sản phẩm.
- Chất liệu: Quản lý các chất liệu khác nhau của một sản phẩm.
- Kết hợp biến thể: Tạo các biến thể kết hợp (ví dụ: Áo thun size M màu đỏ).

Thuộc tính tùy chỉnh: Tạo các thuộc tính tùy chỉnh để mô tả chi tiết sản phẩm, ví dụ: thương hiệu, xuất xứ, bảo hành.

Quản lý kho: Theo dõi số lượng tồn kho của từng biến thể sản phẩm, cảnh báo khi hàng tồn kho thấp.

Tìm kiếm và lọc: Cho phép khách hàng tìm kiếm sản phẩm dựa trên các tiêu chí như tên sản phẩm, danh mục, giá cả, thuộc tính.

### **2.1.3.2. Quản lý đơn hàng**

Vòng đời đơn hàng: Theo dõi các giai đoạn của đơn hàng từ khi đặt hàng, xác nhận đơn hàng, xử lý thanh toán, đóng gói, giao hàng, hoàn thành, hủy đơn.

Xử lý thanh toán: Tích hợp với nhiều cổng thanh toán khác nhau, hỗ trợ thanh toán trực tuyến, thanh toán khi giao hàng, thanh toán trả góp.

Quản lý vận chuyển: Tính toán phí vận chuyển dựa trên nhiều yếu tố, tạo nhãn vận chuyển, theo dõi đơn hàng đang vận chuyển.

Hoàn trả và đổi trả: Xử lý các yêu cầu hoàn trả, đổi trả hàng hóa một cách dễ dàng.

In hóa đơn: Tự động tạo và in hóa đơn cho khách hàng.

### **2.1.3.3. Quản lý khách hàng**

Hồ sơ khách hàng: Lưu trữ thông tin chi tiết về khách hàng như tên, địa chỉ, số điện thoại, lịch sử mua hàng.

Phân nhóm khách hàng: Phân nhóm khách hàng dựa trên hành vi mua hàng, nhân khẩu học để thực hiện các chiến dịch marketing phù hợp.

Chương trình khách hàng thân thiết: Tích điểm, cấp bậc thành viên và cung cấp các ưu đãi đặc biệt.

Danh sách mong muốn: Cho phép khách hàng lưu trữ các sản phẩm yêu thích để mua sau.

Đánh giá sản phẩm: Thu thập đánh giá của khách hàng về sản phẩm để cải thiện chất lượng sản phẩm và dịch vụ.

#### **2.1.3.4. Thanh toán và vận chuyển**

Tích hợp đa dạng cổng thanh toán: Hỗ trợ các cổng thanh toán phổ biến như Stripe, PayPal, và các cổng thanh toán địa phương.

Thanh toán an toàn: Đảm bảo thông tin thanh toán của khách hàng được bảo mật.

Tích hợp với nhiều dịch vụ vận chuyển: Kết nối với các hãng vận chuyển để tạo nhãn vận chuyển và theo dõi đơn hàng.

Tính toán phí vận chuyển: Tính toán phí vận chuyển dựa trên nhiều yếu tố như trọng lượng, kích thước, địa chỉ giao hàng, khoảng cách.

Theo dõi đơn hàng: Theo dõi đơn hàng đang vận chuyển và cung cấp thông tin cho khách hàng.

#### **2.1.3.5. Tiếp thị**

Email marketing: Tạo các chiến dịch email marketing tự động, gửi email chào mừng, email khuyến mãi, email theo dõi đơn hàng.

Mã giảm giá: Tạo các mã giảm giá và áp dụng cho các sản phẩm, đơn hàng.

Chương trình khuyến mãi: Tổ chức các chương trình khuyến mãi như mua 1 tặng 1, giảm giá theo số lượng.

Phân tích dữ liệu: Theo dõi hiệu quả của các chiến dịch marketing, hành vi của khách hàng để tối ưu hóa các chiến dịch tiếp theo.

#### **2.1.3.6. Tùy biến cao**

Mã nguồn mở: Cho phép tùy chỉnh mọi thành phần của cửa hàng, từ giao diện đến chức năng.

API GraphQL: Tương tác với dữ liệu của cửa hàng một cách linh hoạt.

Plugins: Mở rộng chức năng của cửa hàng bằng các plugin.

Themes: Tùy chỉnh giao diện của cửa hàng bằng các theme.

## **2.1.4. Xác thực người dùng trong Saleor**

### **2.1.4.1. Khái niệm**

Xác thực người dùng là quá trình xác minh danh tính của người dùng khi họ cố gắng truy cập vào hệ thống. Trong Saleor, xác thực có thể được thực hiện thông qua nhiều phương thức khác nhau, bao gồm đăng nhập bằng email và mật khẩu, đăng ký tài khoản mới và tích hợp với các nhà cung cấp dịch vụ xác thực bên ngoài.

### **2.1.4.2. Quy trình**

**Đăng ký tài khoản:** Người dùng cung cấp thông tin cần thiết như tên, email, và mật khẩu để tạo tài khoản.

**Đăng nhập:** Người dùng nhập email và mật khẩu đã đăng ký. Hệ thống sẽ kiểm tra tính hợp lệ của thông tin.

**Cung cấp mã xác thực:** Nếu xác thực thành công, hệ thống sẽ tạo ra một mã xác thực (thường là JWT - JSON Web Token) để sử dụng trong các yêu cầu tiếp theo. Mã này sẽ được gửi về phía client và được lưu trữ an toàn.

**Quản lý phiên làm việc:** Mỗi khi người dùng thực hiện một yêu cầu đến API, mã xác thực sẽ được gửi kèm theo trong header của yêu cầu để xác minh quyền truy cập.

### **2.1.4.3. Token JWT**

**Mã hóa thông tin:** JWT chứa thông tin mã hóa về người dùng, bao gồm ID người dùng, thời gian hết hạn và các quyền hạn (roles) của họ.

**Bảo mật:** Token JWT có thể được xác thực một cách an toàn trên server mà không cần truy cập vào cơ sở dữ liệu mỗi lần.

**Thời gian tồn tại:** JWT thường có thời gian tồn tại nhất định (expiration), sau đó người dùng sẽ cần phải xác thực lại.

### **2.1.4.4. Quản lý quyền truy cập**

Saleor sử dụng roles (vai trò) và quyền (permissions) để xác định quyền truy cập của người dùng vào các tài nguyên trong hệ thống. Mỗi người dùng có thể có một hoặc nhiều roles, mỗi roles sẽ được gán một tập hợp các quyền cụ thể.

### **2.1.4.5. Xác thực với GraphQL**

**Đăng nhập và đăng ký qua GraphQL:** Saleor cung cấp các mutation để thực hiện

việc đăng ký và đăng nhập người dùng thông qua GraphQL API.

Gửi token trong yêu cầu: Sau khi đăng nhập thành công, token JWT sẽ được gửi cùng với các yêu cầu GraphQL để xác thực.

#### **2.1.4.6. Bảo mật thông tin**

Bảo vệ mật khẩu: Mật khẩu người dùng sẽ được mã hóa và không lưu trữ dưới dạng văn bản thuần.

HTTPS: Tất cả các yêu cầu cần được thực hiện qua HTTPS để bảo mật thông tin.

#### **2.1.4.7. Xử lý lỗi xác thực**

Thông báo lỗi rõ ràng: Hệ thống cần cung cấp thông báo lỗi rõ ràng nếu có sự cố trong quá trình đăng nhập hoặc đăng ký, chẳng hạn như mật khẩu sai hoặc email không tồn tại.

## **2.2. GraphQL**

### **2.2.1. Giới thiệu GraphQL**



Hình 2. 2. GraphQL

GraphQL là một ngôn ngữ truy vấn dữ liệu mã nguồn mở, được phát triển bởi Facebook và công bố lần đầu vào năm 2015. GraphQL cho phép client chỉ định chính xác dữ liệu cần thiết, loại bỏ vấn đề dư thừa hoặc thiếu hụt dữ liệu thường gặp trong API RESTful[2].

**Đặc điểm nổi bật của GraphQL:**

**Truy vấn có chọn lọc:** Client có toàn quyền kiểm soát dữ liệu được trả về, yêu cầu chính xác những trường thông tin cần thiết, tối ưu hóa bằng thông và hiệu suất.

**Tích hợp dữ liệu:** GraphQL cho phép kết hợp dữ liệu từ nhiều nguồn khác nhau trong một yêu cầu duy nhất, đơn giản hóa việc truy cập và xử lý thông tin.

Kiểu dữ liệu mạnh mẽ: GraphQL sử dụng một hệ thống kiểu (type system) chặt chẽ để định nghĩa dữ liệu, đảm bảo tính toàn vẹn và dự đoán được của dữ liệu trao đổi giữa client và server.

Khả năng tự xem xét (Introspection): GraphQL cung cấp cơ chế introspection, cho phép client tự động khám phá cấu trúc và kiểu dữ liệu của API, hỗ trợ việc tạo ra các công cụ và ứng dụng client hiệu quả.

### 2.2.2. Các thành phần của GraphQL

#### 2.2.2.1. Schema

Schema là nền tảng của GraphQL, mô tả tất cả các dữ liệu có sẵn thông qua API.

Sử dụng ngôn ngữ định nghĩa riêng, thường được gọi là Schema Definition Language (SDL)[3], với cú pháp đơn giản, dễ đọc.

Schema định nghĩa các kiểu dữ liệu (types) cho dữ liệu, bao gồm:

- Scalar Types: Int, String, Boolean, Float, ID.
- Object Types: đại diện cho các đối tượng với các trường dữ liệu.
- Enum Types: danh sách các giá trị cố định.
- List Types: danh sách các phần tử cùng kiểu.
- Các kiểu kết hợp khác.

Ví dụ: Định nghĩa kiểu dữ liệu User, Post.

User là một object types, id, name, email là các trường của kiểu User.

Post là một object types, id, title, content, author là các trường của kiểu Post.



```
1 type User {
2   id: ID!
3   name: String!
4   email: String!
5   posts: [Post!]!
6 }
7
8 type Post {
9   id: ID!
10  title: String!
11  content: String!
12  author: User!
13 }
```

Hình 2. 3. Ví dụ Schema

### 2.2.2.2. Resolver

Kết nối Schema với dữ liệu: Resolver là các hàm chịu trách nhiệm lấy dữ liệu thực tế từ các nguồn dữ liệu (database, API khác,...) và trả về cho GraphQL server theo đúng định nghĩa trong Schema.

Logic nghiệp vụ: Resolver cũng là nơi xử lý logic nghiệp vụ, xác thực, ủy quyền,... trước khi trả về dữ liệu.

Ví dụ (sử dụng JavaScript):

```
1 const resolvers = {
2   Query: {
3     user: (parent, args, context, info) => {
4       // Lấy thông tin user từ database dựa trên args.id
5     },
6   },
7   Mutation: {
8     createPost: (parent, args, context, info) => {
9       // Tạo post mới trong database với args.title và args.content
10    },
11  },
12};
```

Hình 2. 4. Ví dụ Resolver

### 2.2.2.3. Query

Query là cách client yêu cầu dữ liệu từ server.

Cú pháp: Sử dụng cú pháp giống JSON, chỉ định các trường dữ liệu cần lấy.

Ví dụ: Lấy thông tin người dùng (user) có ID “123”.



```
1 query {
2   user(id: 123) {
3     name
4     email
5     posts {
6       title
7     }
8   }
9 }
```

Hình 2. 5. Ví dụ Query

#### 2.2.2.4. Mutation

Mutation dùng để thực hiện các thao tác thay đổi dữ liệu trên server, bao gồm tạo mới, cập nhật và xóa dữ liệu.

Cú pháp: Tương tự như query, nhưng sử dụng từ khóa mutation.

Ví dụ: Tạo một bài viết (Post) mới.



```
1 mutation {
2   createPost(title: "Bài viết mới", content: "Nội dung bài viết") {
3     id
4     title
5   }
6 }
```

Hình 2. 6. Ví dụ Mutation

#### 2.2.2.5. Subscription

Một số ứng dụng có thể yêu cầu kết nối thời gian thực đến máy chủ để nhận thông tin ngay lập tức về một số sự kiện. Điều này có thể được thực hiện bằng cách sử dụng subscriptions trong GraphQL.

Khi một sự kiện được subscriptions bởi một client, một kết nối ổn định được mở tới máy chủ. Bất cứ khi nào sự kiện đó xảy ra, dữ liệu sẽ được đẩy từ máy chủ đến client. Điều này có nghĩa là mặc dù được viết cùng cú pháp như queries và mutations, subscriptions đại diện cho một luồng dữ liệu thay vì một chu kỳ request-response[2].

Sử dụng cho ứng dụng real-time: chat, hệ thống thông báo, bảng tin trực tuyến.

Cú pháp: từ khóa bắt đầu subscription, liệt kê các trường muốn nhận thông báo khi có sự thay đổi.

Ví dụ: Subscription gửi thông báo cho client mỗi khi có một tin nhắn mới được thêm vào phòng chat có ID là "123".

```

● ● ●
1 // Định nghĩa Schema
2 type Message {
3   id: ID!
4   text: String
5   sender: User
6 }
7
8 type Subscription {
9   newMessage(roomId: ID!): Message
10 }
11
12 // Subscription
13 subscription {
14   newMessage(roomId: "123") {
15     id
16     text
17     sender {
18       id
19       name
20     }
21   }
22 }

```

Hình 2. 7. Ví dụ Subscription

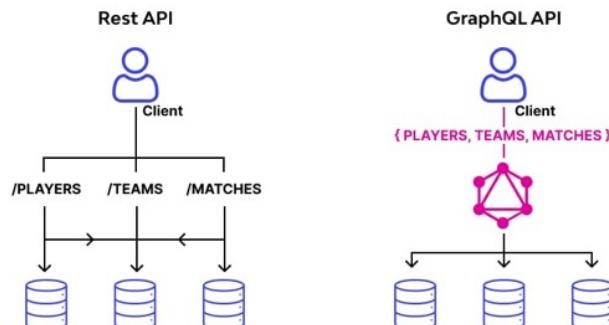
### 2.2.2.6. Runtime

Thực thi GraphQL: Runtime là môi trường thực thi GraphQL, nhận query từ client, xác thực, xử lý và trả về kết quả.

Các thư viện: Có nhiều thư viện runtime cho các ngôn ngữ lập trình khác nhau, ví dụ Apollo Server cho Node.js, GraphQL Java cho Java.

### 2.2.3. So sánh GraphQL với REST API

#### 2.2.3.1. Kiến trúc và truy vấn dữ liệu



Hình 2. 8. Truy vấn dữ liệu của GraphQL và REST

GraphQL: Sử dụng một endpoint duy nhất. Client yêu cầu chính xác dữ liệu cần thiết, tránh over-fetching (nhận quá nhiều dữ liệu không cần thiết) và under-fetching (không nhận đủ dữ liệu, cần phải gửi thêm request)[4].

REST: Sử dụng nhiều endpoint, mỗi endpoint đại diện cho một tài nguyên. Client nhận dữ liệu cố định từ server, có thể dẫn đến over-fetching hoặc under-fetching.

#### **2.2.3.2. Schema và hệ thống kiểu dữ liệu**

GraphQL: Sử dụng schema để định nghĩa cấu trúc dữ liệu và validate truy vấn.

REST: Không có schema chính thức, dữ liệu có thể không nhất quán.

#### **2.2.3.3. Caching**

GraphQL: Phức tạp, cần sử dụng các kỹ thuật như persisted queries.

REST: Đơn giản hơn, tận dụng bộ nhớ đệm HTTP.

#### **2.2.3.4. Versioning**

GraphQL: Không cần versioning (phiên bản) rõ ràng, schema thay đổi linh hoạt.

REST: Thường sử dụng versioning trong URL.

#### **2.2.3.5. Hiệu năng**

GraphQL: Nhanh hơn trong các trường hợp truy vấn phức tạp.

REST: Hiệu quả hơn trong các trường hợp đơn giản.

#### **2.2.3.6. Độ phức tạp**

GraphQL: Phức tạp hơn để học và triển khai ban đầu.

REST: Đơn giản hơn, dễ tiếp cận hơn cho nhà phát triển.

## 2.3. Flutter

### 2.3.1. Giới thiệu chung Flutter



Hình 2. 9. Flutter

Flutter là một framework mã nguồn mở do Google phát triển. Nó cho phép nhà phát triển xây dựng ứng dụng đẹp mắt, nhanh chóng và hiệu quả bằng cách sử dụng một mã nguồn duy nhất. Hỗ trợ phát triển ứng dụng đa nền tảng như Mobile, Web, Desktop, Embedded[5]. Theo định nghĩa của Google trên trang chính thức của Flutter, được mô tả như sau:

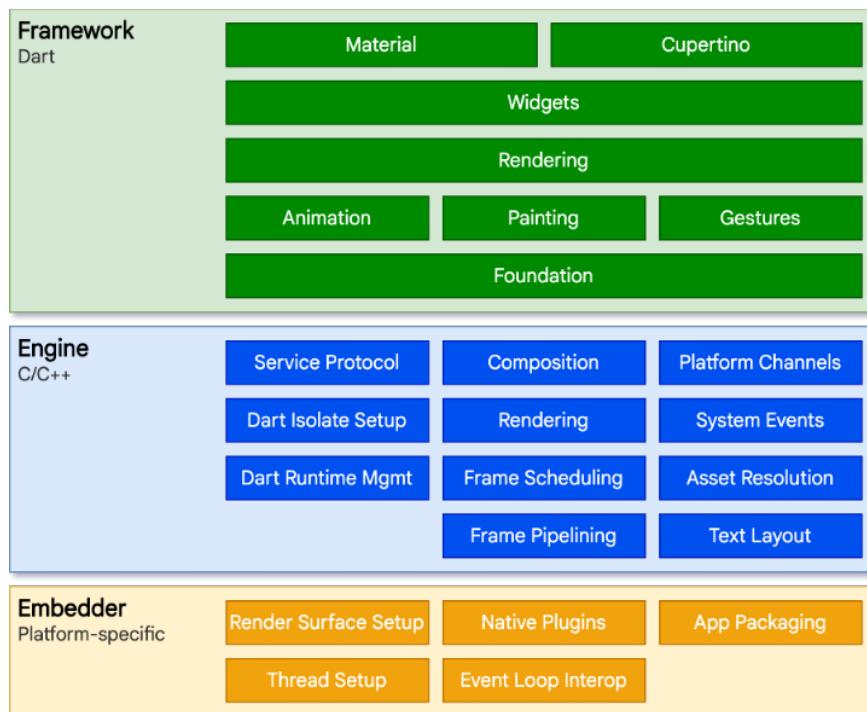
“Flutter is an open-source framework by Google for building beautiful, natively compiled, multi-platform applications from a single codebase.” – Google, flutter.dev.

Flutter không chỉ đơn thuần là framework hay thư viện, mà là một bộ công cụ phát triển phần mềm (SDK) hoàn chỉnh. Sử dụng ngôn ngữ lập trình Dart, Flutter cho phép tạo ra các ứng dụng di động trên cả iOS và Android mà không cần viết mã riêng cho từng nền tảng. Dart được đánh giá cao về hiệu năng, tính ổn định và dễ tiếp cận, góp phần tạo nên sức mạnh cho Flutter.

Điểm sáng của Flutter nằm ở khả năng tùy biến giao diện người dùng thông qua việc kết hợp các widget một cách linh hoạt. Hơn nữa, Flutter cung cấp một hệ sinh thái phong phú với nhiều công cụ và thư viện hỗ trợ, giúp quá trình phát triển diễn ra thuận lợi. Đặc biệt, tính năng Hot Reload cho phép nhà phát triển quan sát thay đổi trên giao diện ngay lập tức mà không cần khởi động lại ứng dụng, tối ưu hóa đáng kể quy trình làm việc.

Với những ưu điểm nổi bật, Flutter là một lựa chọn tối ưu cho việc phát triển ứng dụng di động chất lượng cao, đa nền tảng, tiết kiệm thời gian và chi phí.

### 2.3.2. Kiến trúc của Flutter



Hình 2. 10. Kiến trúc của Flutter

**Ngôn ngữ Dart:** Flutter sử dụng Dart, một ngôn ngữ lập trình do Google phát triển. Dart hỗ trợ cả lập trình hướng đối tượng và hướng thủ tục, đem lại sự linh hoạt và hiệu quả khi phát triển ứng dụng. Những đặc điểm nổi bật của Dart bao gồm hiệu suất cao, cú pháp rõ ràng, và khả năng biên dịch cả trước thời gian (AOT) và ngay khi chạy (JIT), giúp các ứng dụng Flutter vận hành mượt mà và nhanh chóng.

**Flutter Engine:** Được xem như trái tim của Flutter, Flutter Engine được viết bằng C++. Engine này đảm nhiệm việc render giao diện người dùng, xử lý các đầu vào và sự kiện. Nó kết nối trực tiếp với cảm biến và các phần cứng khác của thiết bị, đồng thời là cầu nối giữa mã Dart và nền tảng gốc của thiết bị.

**Foundation Library:** Thư viện này cung cấp các API cơ bản và cần thiết để xây dựng ứng dụng, bao gồm các lớp và hàm để quản lý và tương tác với các thành phần giao diện người dùng.

**Widgets:** Thành phần giao diện người dùng cơ bản trong Flutter, widgets có vai trò như các khối xây dựng UI. Có hai loại chính là StatelessWidget (không thay đổi) và

StatefulWidget (có thể thay đổi trạng thái). Flutter cung cấp một bộ sưu tập lớn các widgets dựng sẵn như Text, Image, Row, Column, Container và nhiều widget khác. Bộ sưu tập này bao gồm các widgets cho cả Material Design của Google và Cupertino của Apple iOS.

**Rendering:** Hệ thống render chịu trách nhiệm vẽ các widget lên màn hình. Sử dụng cây render (render tree) để quản lý và cập nhật các thay đổi trong giao diện người dùng.

**Animation và Motion:** Tạo các hiệu ứng chuyển động, tăng tính tương tác và trải nghiệm người dùng. Các lớp như AnimationController, Tween, cùng với các widget như AnimatedBuilder, AnimatedContainer, hỗ trợ việc tạo và điều khiển các hiệu ứng này.

**Gestures:** Xử lý các thao tác cảm ứng như chạm, kéo, thả, phóng to, tạo ra các trải nghiệm tương tác tự nhiên.

**Cupertino (iOS-styled widgets):** Cung cấp các widget theo phong cách thiết kế của iOS, đảm bảo tính nhất quán và trải nghiệm người dùng tương tự như trên nền tảng iOS gốc.

**Material Design:** Được thiết kế theo tiêu chuẩn Material Design của Google, cung cấp các widget phong phú và nhất quán.

**Interactivity and Navigation:** Hỗ trợ các thao tác tương tác của người dùng và điều hướng giữa các màn hình trong ứng dụng. Các lớp như Navigator, Route, Hero giúp quản lý điều hướng và tạo hiệu ứng chuyển cảnh mượt mà.

**State Management:** Quản lý trạng thái của ứng dụng, đảm bảo rằng giao diện người dùng luôn được cập nhật. Các giải pháp phổ biến như Provider, Bloc, Redux cung cấp các cách tiếp cận khác nhau để quản lý trạng thái.

**Platform Integration:** Tương tác với các chức năng gốc của hệ điều hành (Android/iOS), cho phép sử dụng các tính năng phần cứng và phần mềm đặc thù của từng nền tảng. Các plugin như camera, shared\_preferences, url\_launcher hỗ trợ việc tích hợp các tính năng này vào ứng dụng Flutter.

**Dart Packages:** Hỗ trợ việc sử dụng các gói thư viện từ pub.dev, mở rộng tính năng và khả năng của ứng dụng.

### 2.3.3. Ngôn ngữ lập trình Dart

#### 2.3.3.1. Ngôn ngữ Dart



Hình 2. 11. Dart

Dart, được phát triển bởi Google, là một ngôn ngữ lập trình mã nguồn mở. Dart được thiết kế với cấu trúc hướng đối tượng và cú pháp dựa trên ngôn ngữ C. Dart hỗ trợ đầy đủ các đặc điểm của lập trình hướng đối tượng, cho phép lập trình viên sáng tạo không giới hạn trong việc phát triển ứng dụng[6].

Dart cũng nổi tiếng là ngôn ngữ lập trình chính của Flutter, một framework phát triển ứng dụng di động nhanh chóng và linh hoạt của Google. Sự kết hợp này tạo ra một công cụ mạnh cho việc tạo ra các ứng dụng đa nền tảng với hiệu suất cao và giao diện người dùng mượt mà.

#### 2.3.3.2. Cấu trúc thư mục dự án Flutter

```
1  .dart_tool/
2  .github/
3  android/
4  assets/
5  ios/
6  lib/
7  test/
8  flutter-plugins/
9  flutter-plugins-dependencies/
10 .gitignore
11 .metadata
12 analysis_options.yaml
13 pubspec.lock
14 pubspec.yaml
15 README.md
```

Hình 2. 12. Cấu trúc thư mục dự án Flutter

Thư mục **.dart\_tool**: Bên trong gồm 2 thành phần chính

- flutter\_build: Lưu trữ các bộ nhớ đệm về bản build của Flutter.
- package\_config.json: Nơi chỉ định cụ thể vị trí và phiên bản của những package, dependency hay thư viện mà dự án sử dụng.

Thư mục **android**: Là nơi chứa các cài đặt, tài nguyên cần thiết để chạy ứng dụng Flutter trên hệ điều hành Android.

Thư mục **assets**: Lưu trữ các hình ảnh, phông chữ,... được sử dụng trong toàn bộ dự án.

Thư mục **ios**: Giống như thư mục android, là nơi chứa các cài đặt, tài nguyên cần thiết để chạy ứng dụng Flutter trên hệ điều hành IOS

Thư mục **lib**:

- Là thư mục quan trọng nhất của dự án, nơi thực hiện xây dựng dự án.
- Mặc định thư mục lib chứa tệp main.dart. Khi chạy dự án tệp main.dart được chạy đầu tiên.

Thư mục **test**: Chứa code kiểm thử phần mềm.

Tệp **pubspec.yaml**: Là nơi chứa cấu hình cụ thể cho ứng dụng như tên, phiên bản, môi trường, ...

### 2.3.3.3. Điểm nổi bật của Dart

**Hướng đối tượng:** Dart là ngôn ngữ lập trình hướng đối tượng, cho phép mô hình hóa các đối tượng thực tế thành các lớp (class) và đối tượng (object), có các thuộc tính và phương thức. Dart hỗ trợ các khái niệm cơ bản trong lập trình OOP như trừu tượng hóa, đóng gói, kế thừa và đa hình.

**Bất đồng bộ:** Dart hỗ trợ lập trình không đồng bộ thông qua các Future và Stream. Điều này cho phép chạy đồng thời nhiều task một cách độc lập, tăng hiệu suất xử lý.

**Các thư viện tích hợp:** Dart tích hợp sẵn nhiều thư viện như dart: core, dart: async, dart: math,... giúp phát triển ứng dụng nhanh. Người lập trình dễ dàng sử dụng lại mã thay vì viết lại.

**Hỗ trợ đa nền tảng:** Nhờ Dart VM và Dart compile to native, Dart có thể chạy trên

nhiều nền tảng như Windows, Linux, MacOS, cũng như các mobile platform.

### 2.3.4. Quản lý trạng thái ứng dụng

#### 2.3.4.1. Giới thiệu

Mỗi phần tử giao diện động trong Flutter phải có trạng thái tương ứng, và chỉ có thể thay đổi phần tử này bằng cách thay đổi trạng thái và thông báo cho framework rằng có thể đã xảy ra thay đổi.

Giao diện người dùng thường chứa nhiều hơn một phần tử động. Một màn hình có thể có hàng chục hoặc thậm chí hàng trăm widget động, trạng thái của chúng có thể thay đổi dựa trên trạng thái của các widget khác. Ví dụ, nếu chúng ta có một công tắc và một nhãn văn bản hiển thị “On” hoặc “Off” dựa trên việc công tắc được bật hay tắt, chúng ta có thể nói rằng trạng thái của nhãn phụ thuộc vào trạng thái của công tắc.

Trong trường hợp của công tắc đã mô tả, việc theo dõi trạng thái khá dễ dàng với các cơ chế tích hợp của Flutter. Tuy nhiên, khi ứng dụng trở nên phức tạp hơn và thêm các chức năng như gọi API của hệ điều hành, truy cập lưu trữ cục bộ, yêu cầu HTTP,... việc theo dõi trạng thái của tất cả các widget và cập nhật trạng thái một cách chính xác và hiệu quả trở nên khó khăn hơn rất nhiều, trong khi vẫn giữ cho mã nguồn dễ đọc, dễ kiểm tra và duy trì. Đây là lúc các nhà phát triển nhận ra rằng cần có các phương pháp quản lý trạng thái tiên tiến hơn[7].

#### 2.3.4.2. Các thư viện quản lý trạng thái ứng dụng

Provider: Phương pháp quản lý chính trong đề tài này. Một phương pháp dễ sử dụng và tích hợp tốt với framework.

setState: Phương pháp cấp thấp để quản lý trạng thái ngắn hạn của widget, thích hợp cho các trường hợp đơn giản và tạm thời.

InheritedWidget & InheritedModel: Phương pháp cấp thấp dùng để truyền dữ liệu và trạng thái từ tổ tiên đến các con cháu trong cây widget, thường được sử dụng làm nền tảng cho các phương pháp quản lý trạng thái phức tạp hơn.

Redux: Một phương pháp quản lý trạng thái được đưa từ phát triển web vào Flutter, sử dụng một container trạng thái tập trung và nguyên tắc lập trình chức năng.

BLoC / Rx: Một nhóm các mẫu thiết kế dựa trên stream và observable, cho phép

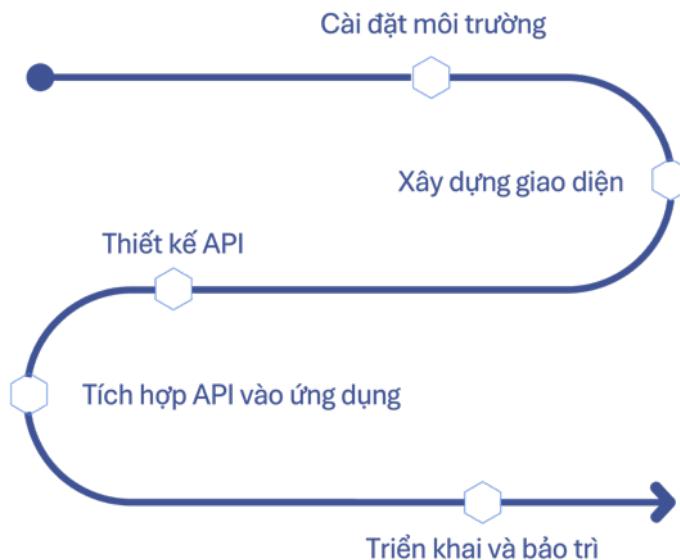
quản lý trạng thái một cách có cấu trúc và có thể mở rộng.

MobX: Một thư viện quản lý trạng thái phổ biến dựa trên các nguyên tắc observables và reactions, giúp tự động cập nhật giao diện người dùng khi trạng thái thay đổi.

GetX: Một giải pháp quản lý trạng thái đơn giản và phản ứng nhanh, tích hợp nhiều tính năng như điều hướng, quản lý phụ thuộc và quản lý trạng thái.

## CHƯƠNG 3. HIỆN THỰC HÓA NGHIÊN CỨU

### 3.1. Quy trình thực hiện



Hình 3. 1. Quy trình thực hiện nghiên cứu

Cài đặt môi trường: Môi trường phát triển cần được cài đặt gồm các công cụ và phần mềm như Flutter SDK, Saleor backend.

Xây dựng giao diện: Sử dụng Flutter để xây dựng các màn hình của ứng dụng.

Thiết kế API: Sử dụng GraphQL API do Saleor cung cấp thiết kế API xác thực người dùng, lấy và cập nhật dữ liệu liên quan đến sản phẩm, đơn hàng.

Tích hợp API vào ứng dụng: Sử dụng thư viện graphql\_flutter để kết nối giao diện người dùng với backend.

Triển khai và bảo trì: Đảm bảo ứng dụng hoạt động ổn định và được cập nhật liên tục.

### 3.2. Cài đặt môi trường

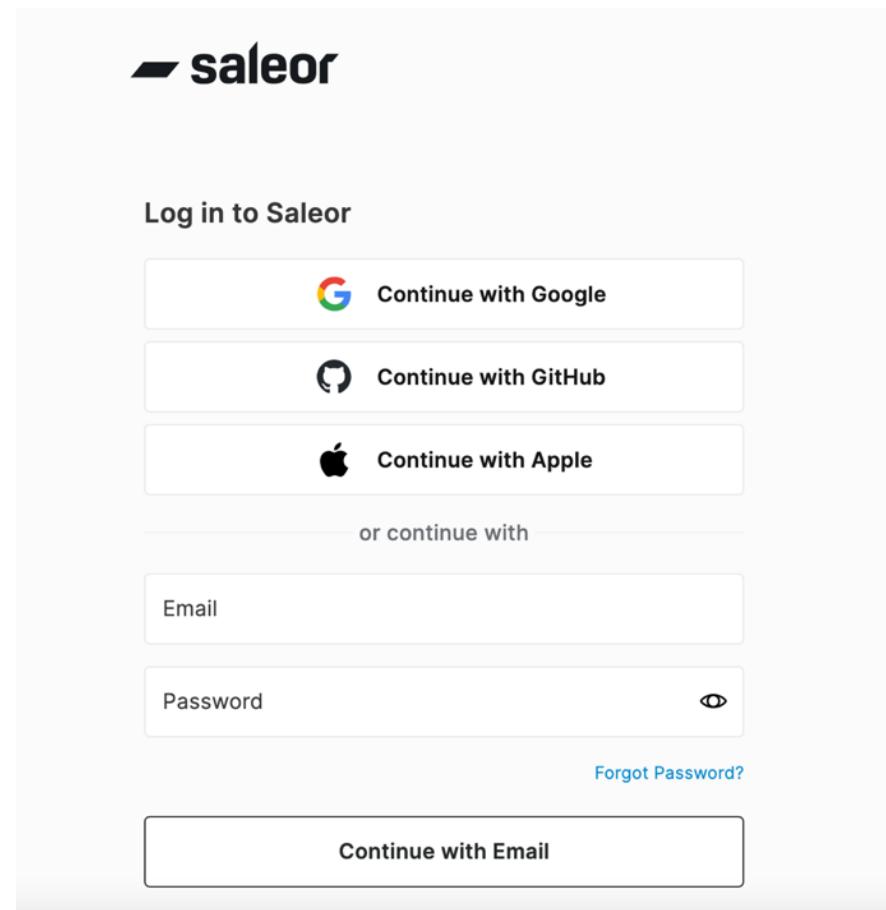
Sử dụng môi trường backend của Saleor cung cấp để thực hiện nghiên cứu.

#### 3.2.1. Saleor backend

##### 3.2.1.1. Cài đặt Saleor trên Saleor Cloud

###### ❖ Đăng ký tài khoản Saleor Cloud

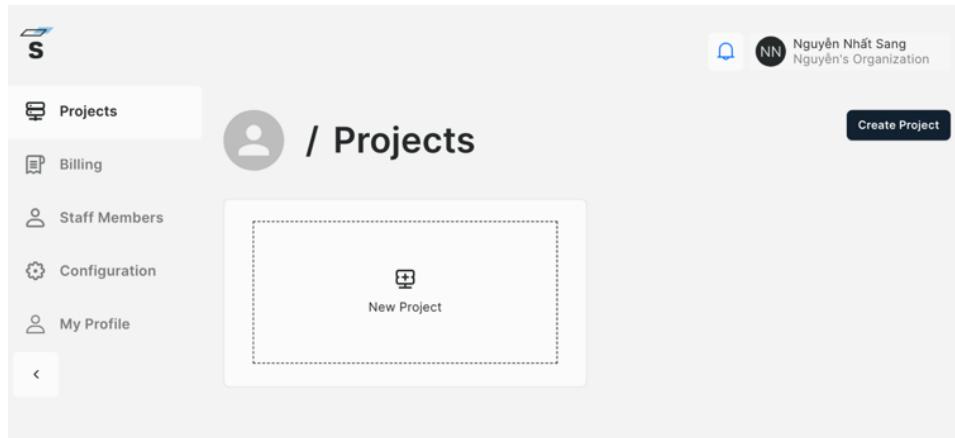
Truy cập <https://cloud.saleor.io/> và đăng ký tài khoản.



Hình 3. 2. Đăng ký Saleor Cloud

#### ❖ Tạo dự án mới

Sau khi đăng nhập, vào mục "Projects" và chọn "Create Project". Điền thông tin cần thiết cho Project và chọn "Create".



Hình 3. 3. Tạo dự án mới

#### ❖ Cấu hình dự án

Chọn dự án vừa tạo, vào phần "Environment" và chọn "Create Sandbox" và điền thông tin cần thiết.

**Create a Sandbox Environment**

**Details**  
Provide basic information about your environment, including the environment name and domain. Select Saleor version.

<b>Environment Details</b>
Environment Name *
Environment Domain * .eu.saleor.cloud
<b>Saleor Details</b>
Saleor Version

**API Restriction**  
Set up API domain authentication.

<input type="checkbox"/> Restrict public access A password will be required to access the API in this domain.
--

**Database**  
Select a database type for the Saleor Dashboard or opt for a blank database.

<input checked="" type="radio"/> Sample database
<input type="radio"/> Blank database
<input type="radio"/> Use Snapshot

Hình 3. 4. Câu hình dự án

Sau khi “Environment” đã được tạo và khởi chạy, Saleor Cloud cung cấp một Saleor API URL để truy cập Saleor Dashboard và Saleor API.

Truy cập các URL này để kiểm tra và sử dụng ứng dụng.

**Environment Details**

SALEOR API URL  
demo-example.eu.saleor.cloud/graphql/

SERVER REGION    SALEOR VERSION  
Ireland            3.20.37

**Usage**              Manage Plan

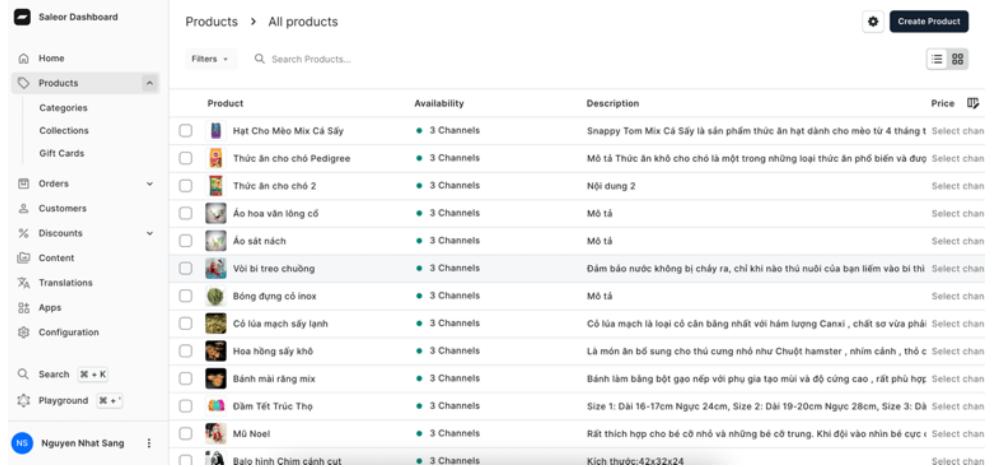
**demo Environment**  
No usage data collected yet

**Store Dashboard**  
Product Information Management (PIM),  
Order Management System (OMS),  
Integrations (Apps & Webhooks).  
[Go to Dashboard](#)

**GraphQL API**  
Using the Saleor GraphQL API allows you to query and modify all of your shop's data flexibly and efficiently.  
[Go to Playground](#)

Hình 3. 5. Saleor API URL

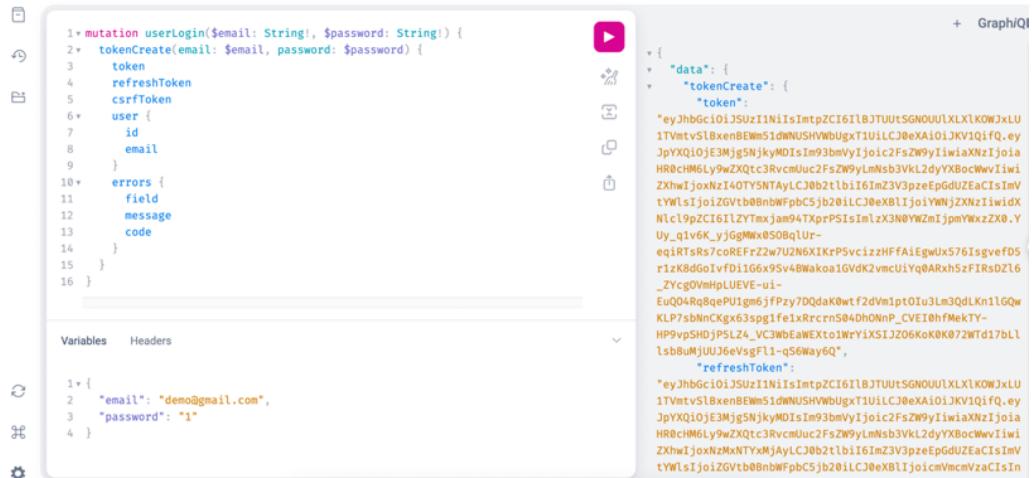
## ❖ Dashboard



Hình 3. 6. Saleor dashboard

Saleor Dashboard là giao diện quản lý trực quan cửa hàng dành cho người quản trị.

## ❖ Playground



Hình 3. 7. Saleor Playground

Saleor Playground là một môi trường tương tác để thử nghiệm và thực thi các truy vấn GraphQL. Cung cấp các tính năng sau:

Thực thi truy vấn GraphQL: Cho phép viết và thực thi các truy vấn và mutation GraphQL để lấy dữ liệu hoặc cập nhật dữ liệu trên máy chủ Saleor.

Tìm hiểu API: Giúp khám phá các khả năng của API Saleor bằng cách duyệt qua các loại truy vấn, mutation, và các đối tượng dữ liệu có sẵn.

Kiểm tra và gỡ lỗi: Hỗ trợ kiểm tra và gỡ lỗi các truy vấn GraphQL trước khi tích hợp vào ứng dụng, giúp đảm bảo truy vấn hoạt động đúng cách và trả về dữ liệu mong muốn.

Tài liệu API động: Cung cấp tài liệu chi tiết về các schema GraphQL của Saleor, giúp bạn hiểu rõ cấu trúc và các trường dữ liệu có thể truy vấn.

### 3.2.1.2. Cài đặt Saleor ở máy cục bộ

#### ❖ Cài đặt Docker

Truy cập <https://docs.docker.com/engine/install/> và cài đặt Docker theo hướng dẫn.

#### ❖ Clone Repository Saleor

Mở terminal và thực hiện các lệnh sau để clone repository Saleor:

```
git clone https://github.com/saleor/saleor-platform.git
```

```
cd saleor-platform
```

```
docker compose build
```

#### ❖ Áp dụng các thay đổi

Thực hiện lệnh cập nhật lược đồ cơ sở dữ liệu lên phiên bản mới nhất.

```
docker compose run --rm api python3 manage.py migrate
```

#### ❖ Thêm dữ liệu mẫu

Thêm dữ liệu mẫu vào cơ sở dữ liệu (nếu cần).

```
docker compose run --rm api python3 manage.py populatedb
```

Sau đó, làm theo hướng dẫn để thêm tên người dùng và mật khẩu quản trị viên.

#### ❖ Chạy Saleor sử dụng Docker Compose

Trong thư mục gốc saleor – platform của Saleor, chạy lệnh sau để khởi động tất cả các container cần thiết cho Saleor.

```
docker-compose up
```

Lệnh này sẽ tải xuống và thiết lập tất cả các images cần thiết và khởi chạy các container.

#### ❖ Kiểm tra hoạt động của Saleor

Sau khi các container đã được khởi động, có thể truy cập Saleor Dashboard bằng cách mở trình duyệt và đi tới địa chỉ: <http://localhost:9000/dashboard/>

API GraphQL của Saleor có thể được truy cập tại: <http://localhost:8000/graphql/>

### **3.2.2. Flutter**

Truy cập <https://docs.flutter.dev/get-started/install> và cài đặt theo hướng dẫn.

Sau khi cài đặt xong mở terminal thực hiện lệnh `flutter doctor` để kiểm tra cài đặt Flutter. Lệnh này sẽ kiểm tra các thành phần cần thiết cho việc phát triển ứng dụng Flutter, đồng thời chỉ ra những phần nào đang hoạt động tốt và những phần nào cần được cấu hình hoặc cài đặt bổ sung.

### **3.3. Xây dựng giao diện**

Màn hình đăng nhập: gồm các ô nhập thông tin tài khoản, mật khẩu và nút đăng nhập.

Màn hình đăng ký: gồm các ô nhập email, số điện thoại, mật khẩu và nút đăng ký.

Màn hình trang chủ: gồm thanh tìm kiếm, bộ lọc và danh sách sản phẩm

Màn hình chi tiết sản phẩm: gồm tên, hình ảnh, mô tả, giá của sản phẩm và nút thêm vào giỏ hàng.

Màn hình giỏ hàng: gồm danh sách sản phẩm đã thêm vào giỏ hàng, tổng tiền của đơn hàng và nút thanh toán.

Màn hình thông tin cá nhân: gồm tên, email, số điện thoại và địa chỉ của khách hàng.

### **3.4. Thiết kế API**

#### **3.4.1. API đăng ký**

Hàm register thực hiện quá trình đăng ký người dùng mới trên hệ thống. Nó gửi một yêu cầu mutation GraphQL đến máy chủ để tạo một tài khoản người dùng mới với các thông tin được cung cấp. Hàm này bao gồm các thông tin cần thiết như email, mật khẩu, tên và họ của người dùng, cũng như URL chuyển hướng sau khi đăng ký thành công.

#### **3.4.2. API đăng nhập**

Hàm loginUser thực hiện đăng nhập người dùng bằng cách gửi email và mật khẩu lên server qua GraphQL mutation. Hàm trả về token xác thực và thông tin người dùng, hoặc lỗi nếu đăng nhập thất bại. Công dụng chính của hàm là xác thực người dùng và lấy thông tin cần thiết để sử dụng trong ứng dụng.

### **3.4.3. API đăng xuất**

Hàm logout thực hiện đăng xuất người dùng thông qua mutation GraphQL. Nó gửi yêu cầu logout đến server với pluginId được truyền vào và nhận lại phản hồi từ server. Nếu có lỗi xảy ra, các lỗi sẽ được hiển thị; nếu không, hàm sẽ xóa dữ liệu đăng nhập. Công dụng chính của hàm là đăng xuất người dùng khỏi hệ thống và xóa thông tin đăng nhập đã lưu.

### **3.4.4. API lấy danh sách category**

Hàm fetchCategories dùng để lấy danh sách category để hỗ trợ cho hàm lấy danh sách sản phẩm theo category ở mục 3.2.5.

### **3.4.5. API lấy danh sách sản phẩm theo category**

Như đã đề cập phía trên, sau khi thực hiện xong mục 3.2.4 chúng ta đã có danh sách category. Hàm fetchProductsForUser gửi một truy vấn GraphQL để lấy danh sách sản phẩm thuộc một danh mục cụ thể cho người dùng, dựa trên categoryId, channel và các bộ lọc tùy chọn.

### **3.4.6. API tạo phiếu thanh toán**

Hàm createCheckout gửi một truy vấn GraphQL để tạo một phiên thanh toán (checkout) mới với thông tin người dùng, danh sách sản phẩm và kênh (channel) cụ thể.

### **3.4.7. API thêm sản phẩm vào giỏ hàng**

Hàm addToCart gửi một truy vấn GraphQL để thêm sản phẩm vào giỏ hàng (cart) của người dùng dựa trên phiên thanh toán (checkout) hiện tại. Nếu không có thì tạo checkout ở mục 3.2.6.

### **3.4.8. API lấy thông tin người dùng đăng nhập**

Hàm này gửi một truy vấn để lấy thông tin cá nhân của người dùng từ máy chủ, bao gồm các thuộc tính như ID, email, tên, trạng thái hoạt động, địa chỉ giao hàng và hóa đơn mặc định, các địa chỉ đã lưu, hình đại diện, thông tin giỏ hàng, và danh sách đơn hàng của người dùng.

### **3.4.9. API lấy thông tin giỏ hàng**

Hàm này gửi một truy vấn GraphQL để lấy thông tin chi tiết về giỏ hàng, bao gồm

ID của giỏ hàng, các mặt hàng trong giỏ, số lượng, thông tin về biến thể sản phẩm, tên sản phẩm, hình ảnh thu nhỏ và giá tổng của giỏ hàng.

#### 3.4.10. API cập nhật thông tin tài khoản

Hàm này gửi một truy vấn GraphQL để cập nhật thông tin tài khoản người dùng với các trường firstName và lastName. Nếu cập nhật thành công, hàm sẽ trả về thông tin người dùng mới.

#### 3.4.11. API thêm địa chỉ

Hàm này gửi một truy vấn GraphQL để tạo một địa chỉ mới cho tài khoản người dùng với các thông tin như tên, địa chỉ, thành phố và mã bưu chính. Sau khi thực hiện thành công, hàm trả về thông tin địa chỉ vừa được tạo.

#### 3.4.12. API cập nhật địa chỉ giao hàng và địa chỉ thanh toán

Hàm này thực hiện hai truy vấn GraphQL để cập nhật địa chỉ giao hàng và địa chỉ thanh toán cho một checkout cụ thể dựa trên checkoutId và thông tin địa chỉ được cung cấp. Nếu cả hai truy vấn đều thành công, hàm sẽ trả về thông tin địa chỉ thanh toán.

### 3.5. Tích hợp Flutter với Saleor qua GraphQL

#### ❖ Thêm dependencies



```
1  dependencies:  
2    flutter:  
3      sdk: flutter  
4    graphql_flutter: ^5.0.0
```

Hình 3.8. Thêm dependencies

Trong dự án Flutter mở file pubspec.yaml và thêm các dependencies cần thiết.

### 3.5.1. Cấu hình GraphQL Client

```
● ○ ●  
1 import 'package:graphql_flutter/graphql_flutter.dart';  
2  
3 class GraphQLConfig {  
4   static HttpLink httpLink = HttpLink("https://your-saleor-api-url/graphql/");  
5  
6   static GraphQLClient clientToQuery() {  
7     return GraphQLClient(  
8       cache: GraphQLCache(store: HiveStore()),  
9       link: httpLink,  
10    );  
11  }  
12 }  
13  
14
```

Hình 3. 9. Cấu hình GraphQL Client

GraphQL Client được sử dụng để thực hiện các yêu cầu đối với API GraphQL.

Thay "https://your-saleor-api-url/graphql/" bằng URL thực tế API Saleor cung cấp.

### 3.5.2. Sử dụng GraphQL Client

```
● ○ ●  
1 import 'package:flutter/material.dart';  
2 import 'package:graphql_flutter/graphql_flutter.dart';  
3 import 'graphql_config.dart';  
4  
5 void main() async {  
6   await initHiveForFlutter();  
7   runApp(MyApp());  
8 }  
9  
10 class MyApp extends StatelessWidget {  
11   @override  
12   Widget build(BuildContext context) {  
13     return GraphQLProvider(  
14       client: ValueNotifier(GraphQLConfig.clientToQuery()),  
15       child: MaterialApp(  
16         title: 'Flutter Saleor App',  
17         home: MyHomePage(),  
18       ),  
19     );  
20   }  
21 }
```

Hình 3. 10. Sử dụng GraphQL Client

Sử dụng GraphQLProvider cung cấp GraphQL client cho tất cả các widget con cần tương tác với API.

## 3.6. Triển khai và bảo trì

Sử dụng Saleor Cloud để triển khai Saleor backend. Đóng gói ứng dụng và phát

hành lên các chợ ứng dụng. Theo dõi và sửa lỗi, cập nhật tính năng mới và đảm bảo hiệu suất ứng dụng được tối ưu.

## CHƯƠNG 4. KẾT QUẢ NGHIÊN CỨU

### 4.1. Lưu token và sử dụng token của người dùng

#### 4.1.1. Đăng nhập và nhận token

Khi người dùng đăng nhập thành công, hệ thống sẽ trả về một JWT (JSON Web Token) và một refresh token. Token này được dùng để xác thực người dùng trong các yêu cầu tiếp theo.

Quy trình:

- Người dùng nhập thông tin đăng nhập (email và mật khẩu).
- Ứng dụng gửi yêu cầu đăng nhập tới server Saleor.
- Server Saleor xác thực thông tin đăng nhập.
- Nếu thành công, server trả về token và refresh token



```
1 Future<Map<String, dynamic>> loginUser(String email, String password) async {
2   const String loginMutation = ''
3   mutation TokenCreate(\$email: String!, \$password: String!) {
4     tokenCreate(email: \$email, password: \$password) {
5       token
6       refreshToken
7       csrfToken
8     }
9   }
10 }
```

Hình 4. 1. Đăng nhập và nhận token

#### 4.1.2. Lưu token vào SharedPreferences

Sau khi nhận được token, ứng dụng lưu trữ chúng vào bộ nhớ cục bộ (SharedPreferences).



```
1 void _login() async {
2   final response = await _authService.loginUser(email, password);
3   if (response != null && response['token'] != null) {
4     final SharedPreferences sharedpreferences = await SharedPreferences.getInstance();
5     await sharedpreferences.setString('token', response['token']);
6     await sharedpreferences.setString('refreshToken', response['refreshToken']);
7     // Điều hướng tới trang chủ
8   } else {
9     // Hiển thị thông báo lỗi
10 }
11 }
```

Hình 4. 2. Lưu token vào bộ nhớ cục bộ

#### 4.1.3. Sử dụng token để xác thực yêu cầu tiếp theo

Trong các yêu cầu tiếp theo tới máy chủ, ứng dụng đính kèm token trong header để xác thực người dùng.

Quy trình:

- Lấy token từ SharedPreferences.
- Đính kèm token vào header của yêu cầu.
- Gửi yêu cầu tới server.



```
1 // Tạo AuthLink để thêm token xác thực vào mỗi request
2 final AuthLink authLink = AuthLink(
3     getToken: () async => token != null ? 'Bearer $token' : '',
4 );
```

Hình 4. 3. Sử dụng token để xác thực các yêu cầu tiếp theo

#### 4.1.4. Làm mới token (Refresh Token)

Khi JWT hết hạn, ứng dụng có thể sử dụng refresh token để lấy một JWT mới mà không cần người dùng phải đăng nhập lại.



```
1 await sharedPreferences.setString(
2     AppConstants.keyRefreshToken, response['refreshToken']);
```

Hình 4. 4. Làm mới token

#### 4.1.5. Đăng xuất

Người dùng đăng xuất, ứng dụng sẽ xóa token và refresh token khỏi bộ nhớ cục bộ.



```
1 final prefs = await SharedPreferences.getInstance();
2 await prefs.clear();
```

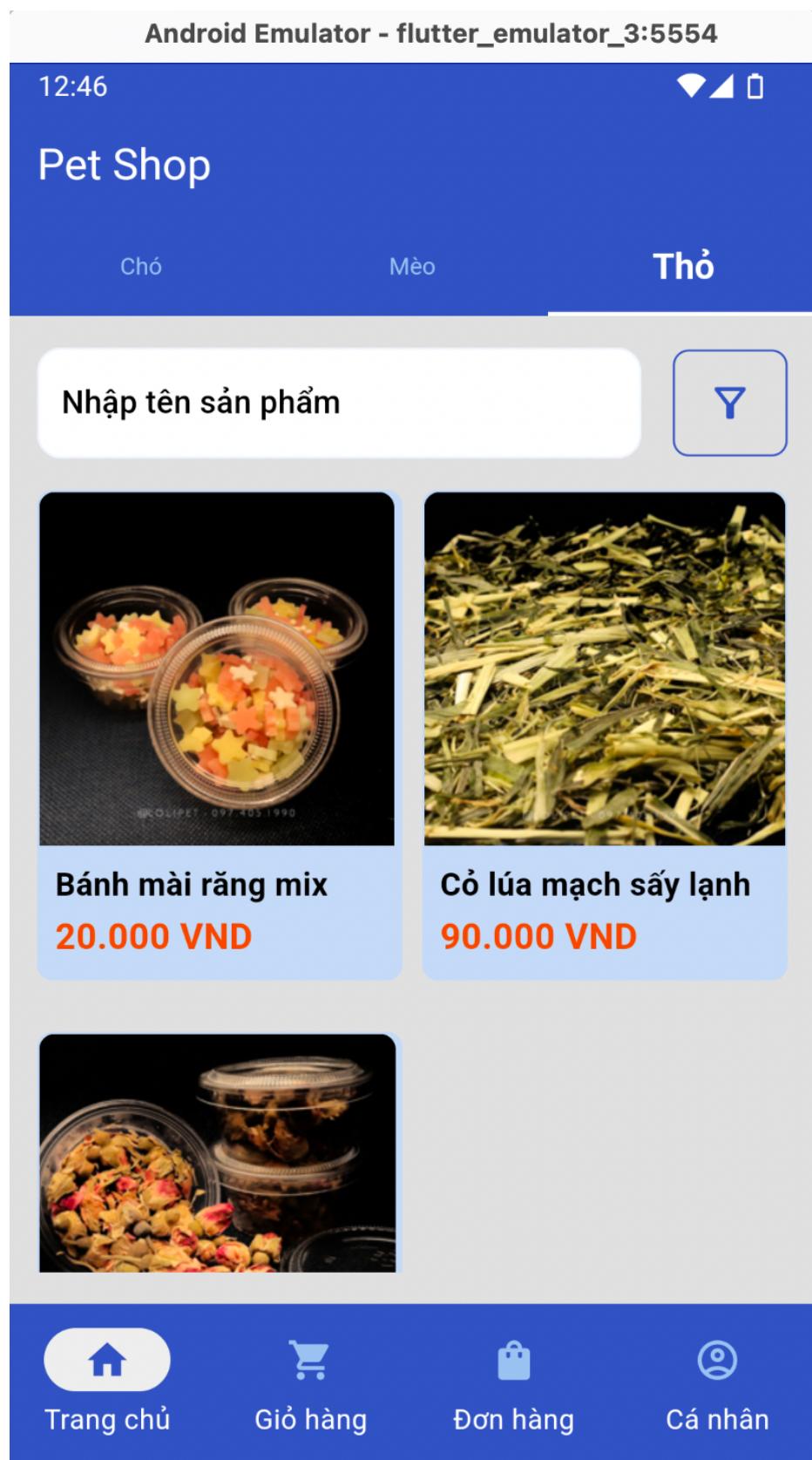
Hình 4. 5. Xóa token

## 4.2. Giao diện trên web



Hình 4. 6.Giao diện trên web

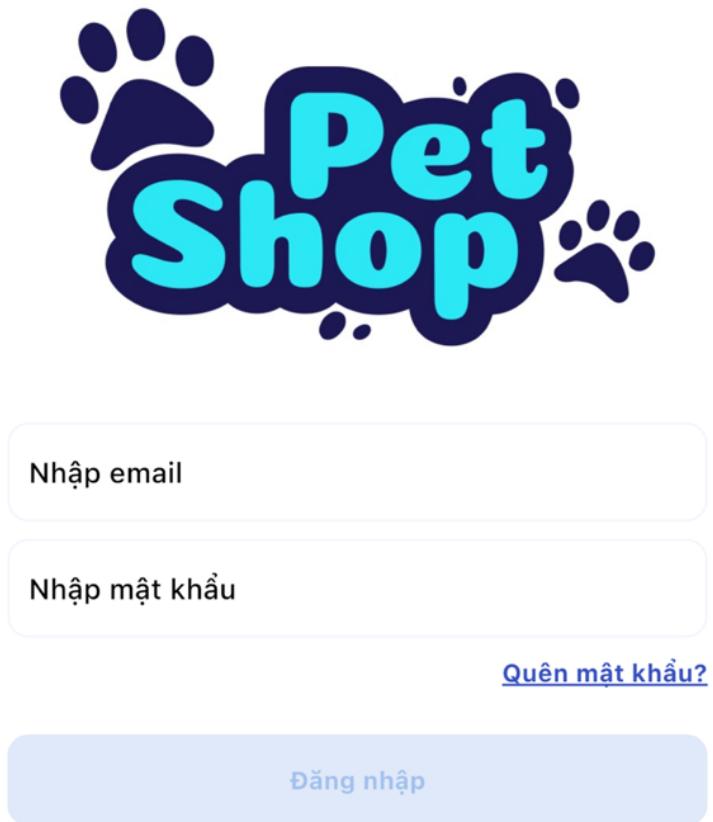
#### 4.3. Giao diện trên Android



Hình 4. 7. Giao diện trên Android

## 4.4. Giao diện trên IOS

### 4.4.1. Đăng nhập



Hình 4. 8. Giao diện đăng nhập

Widget chính đã dùng để xây dựng giao diện đăng nhập:

LoadingOverlay: Được sử dụng để hiển thị màn hình loading khi đang xử lý.

SafeArea: Bảo vệ nội dung hiển thị khỏi các vùng không an toàn như notch (một phần của màn hình được cắt đi để tạo chỗ cho camera trước và các cảm biến khác), status bar (thường nằm phía trên cùng của màn hình), etc.

InputBase: Widget tùy chỉnh để nhập dữ liệu. Được sử dụng cho email và mật khẩu.

ButtonCustomContent: Widget nút tùy chỉnh với nội dung tùy chỉnh, dùng cho nút "Quên mật khẩu?" và "Đăng ký".

ButtonBase: Widget nút dùng cho nút "Đăng nhập".

Column: Bố trí các widget con theo chiều dọc.

Padding: Thêm khoảng cách xung quanh nội dung con.

Center: Căn giữa nội dung con theo cả trục x và y.

Image.asset: Hiển thị hình ảnh từ thư mục assets

Text: Hiển thị văn bản tinh.

#### 4.4.2. Đăng ký

The image shows a mobile application interface for sign-up. At the top is a logo consisting of the words "Pet Shop" in a stylized font, where the letters are partially filled with blue and purple colors, and two dark blue paw prints are integrated into the design. Below the logo are four input fields with rounded corners, each containing a placeholder text: "Nhập tên" (Enter name), "Nhập email" (Enter email), "Nhập mật khẩu" (Enter password), and "Nhập lại mật khẩu" (Enter password again). Below these input fields is a large, light blue rectangular button with the text "Đăng ký" (Sign up) in white. At the bottom of the screen, there is a link "Đã có tài khoản? [Đăng nhập](#)" (Already have an account? [Log in](#)) in a smaller font.

Hình 4. 9. Giao diện đăng ký

Widget chính đã dùng để xây dựng giao diện đăng ký:

Sử dụng các widget tương tự giao diện đăng nhập: LoadingOverlay, Scaffold,

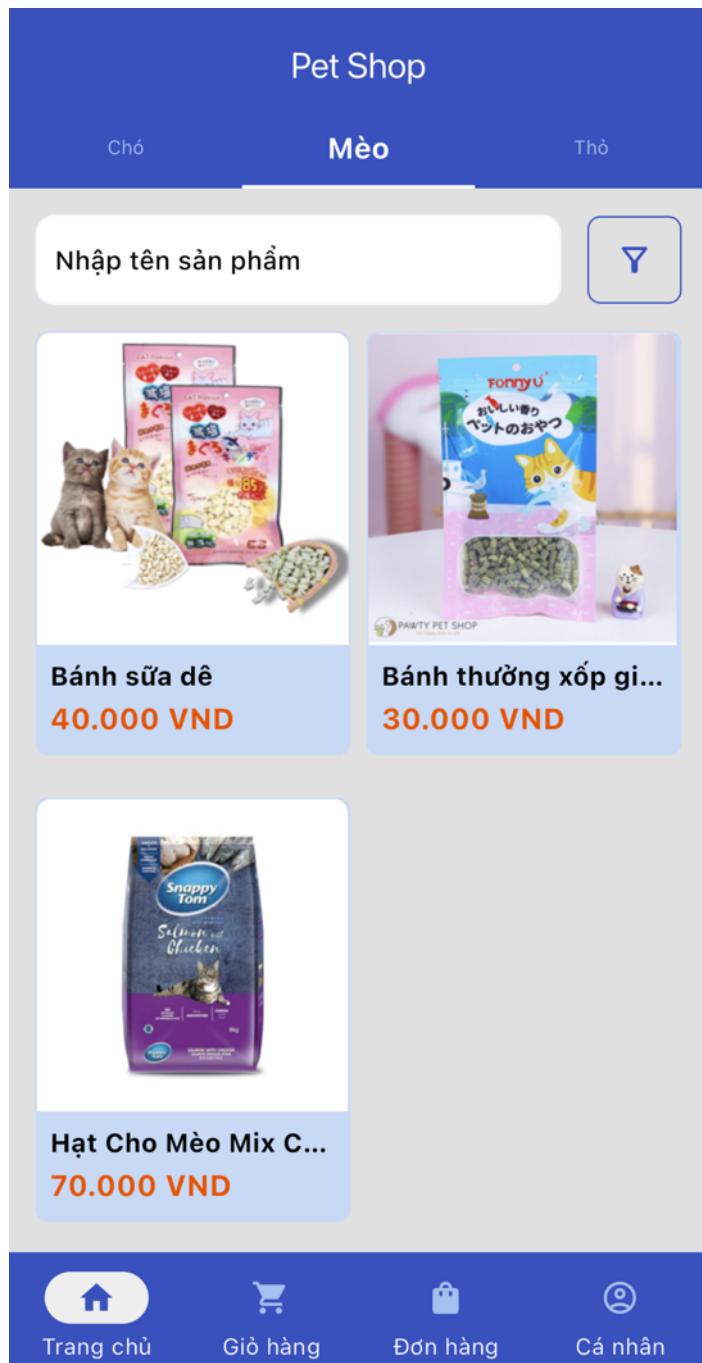
SafeArea, Padding, Center, Column, Text, etc.

InputBase: Widget tùy chỉnh để nhập dữ liệu. Được sử dụng cho tên, email, số điện thoại, mật khẩu và nhập lại mật khẩu.

ButtonBase: Widget nút với nội dung, dùng cho nút "Đăng ký".

ButtonCustomContent: Widget nút tùy chỉnh với nội dung tùy chỉnh, dùng cho nút "Đăng nhập".

#### 4.4.3. Trang chủ



Hình 4. 10. Giao diện trang chủ

Widget chính đã dùng để xây dựng giao diện trang chủ:

DefaultTabController: Widget để quản lý trạng thái của TabBar và TabBarView.

TabBar: Điều hướng các tab con, bao gồm ba tab: "Chó", "Mèo", "Thỏ".

TabBarView: Hiển thị nội dung tương ứng với mỗi tab.

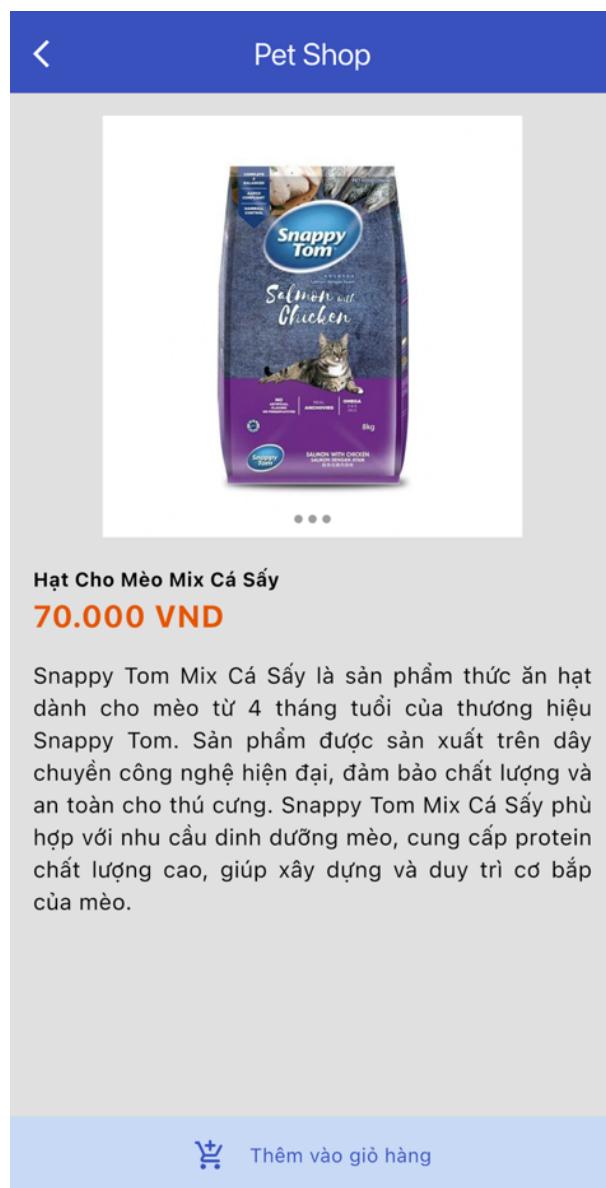
NestedTabBar: Widget tùy chỉnh, để hiển thị các sản phẩm trong từng danh mục.

InputBase: Widget tùy chỉnh để nhận văn bản tìm kiếm sản phẩm từ người dùng.

ButtonCustomContent: Widget tùy chỉnh để tạo nút lọc sản phẩm.

GridView.builder: Hiển thị danh sách sản phẩm dưới dạng lưới.

#### 4.4.4. Chi tiết sản phẩm



Hình 4. 11. Giao diện chi tiết sản phẩm

Widget chính đã dùng để xây dựng giao diện chi tiết sản phẩm:

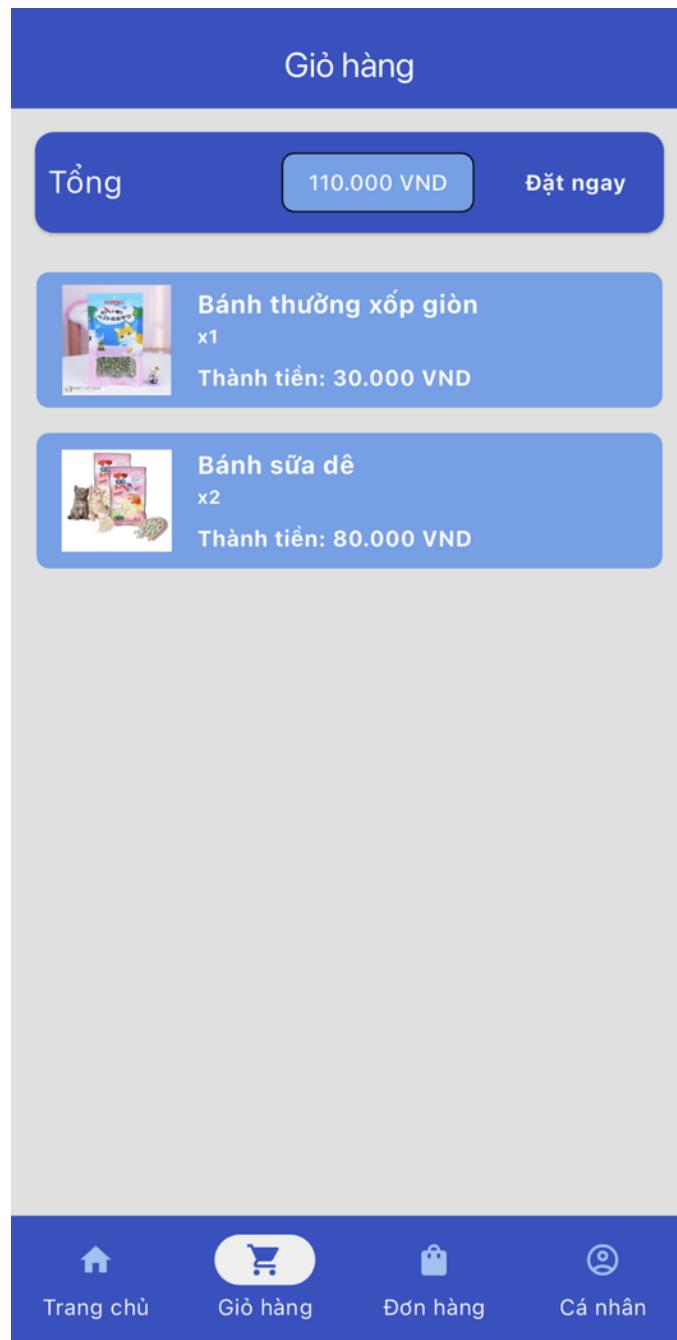
LoadingOverlay: Bọc toàn bộ màn hình để hiển thị overlay khi đang tải dữ liệu.

SingleChildScrollView: Cho phép cuộn nội dung màn hình khi nội dung mô tả của sản phẩm vượt quá chiều cao màn hình.

ImageSlideshow: Hiển thị trình chiếu chiều hình ảnh sản phẩm.

ButtonCustomContent: Nút tùy chỉnh cho hành động thêm vào giỏ hàng.

#### 4.4.5. Giỏ hàng



Hình 4. 12.Giao diện giỏ hàng

Widget chính đã dùng để xây dựng giao diện giỏ hàng:

ListView.builder: Tạo danh sách các sản phẩm trong giỏ hàng.

ButtonCustomContent: Widget tùy chỉnh nút bấm, sử dụng cho thao tác đặt hàng.

Card: Tạo một thẻ chứa thông tin tổng số tiền và nút đặt hàng.

#### 4.4.6. Lịch sử đơn hàng

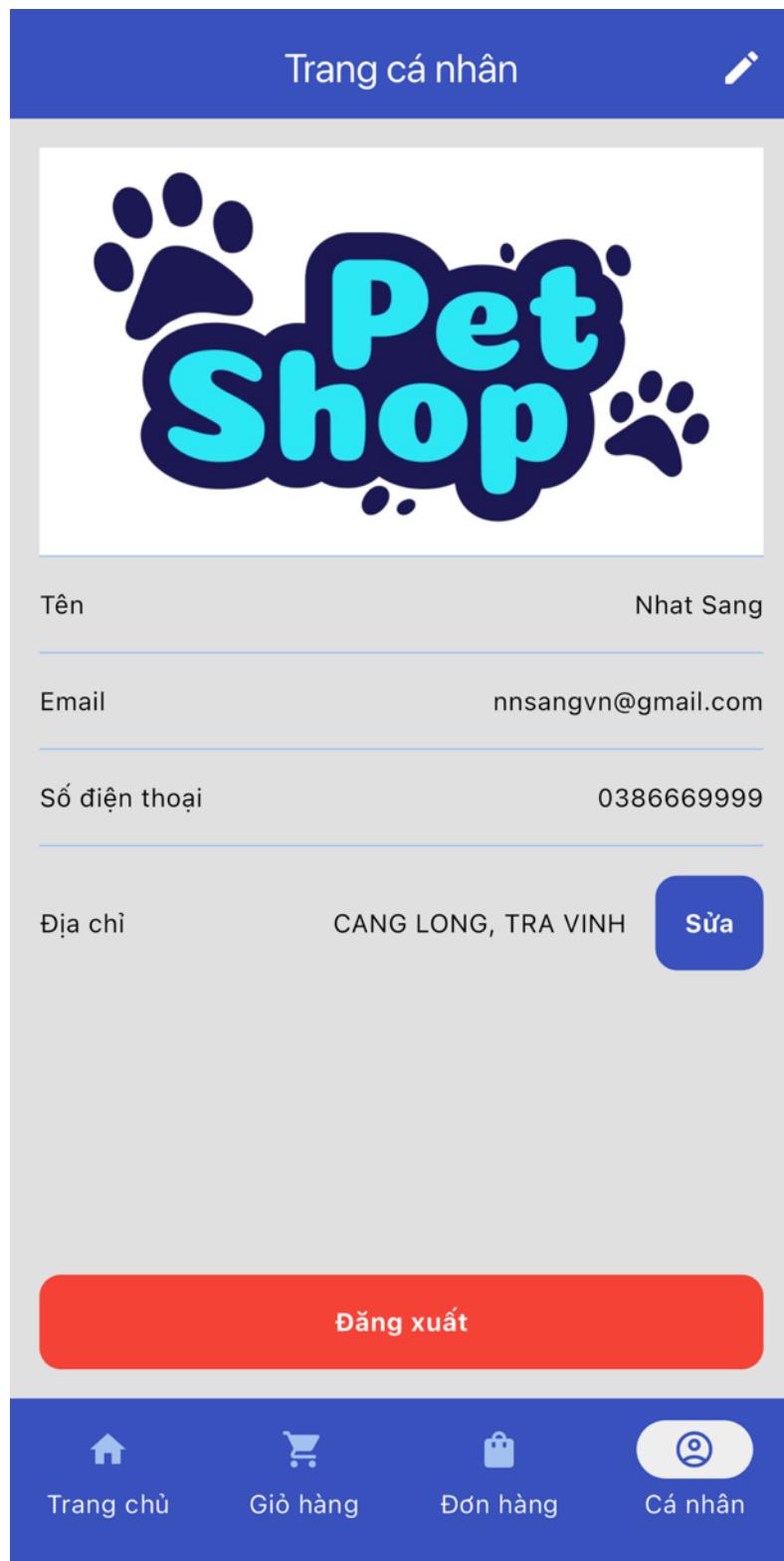


Hình 4. 13. Giao diện lịch sử đơn hàng

Widget chính đã dùng để xây dựng giao diện lịch sử đơn hàng:

ListView.builder: Tạo danh sách cuộn các phần tử đơn hàng.

#### 4.4.7. Thông tin cá nhân



Hình 4. 14. Giao diện thông tin cá nhân

Widget chính đã dùng để xây dựng giao diện thông tin cá nhân:

Icon: Hiển thị biểu tượng dùng để chỉnh sửa thông tin cá nhân.

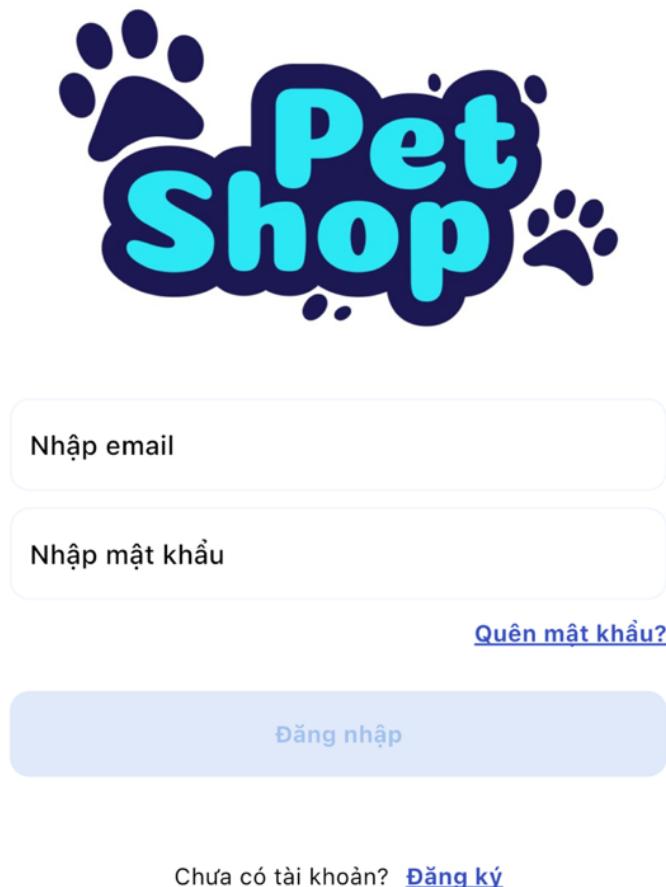
ButtonBase: Widget tùy chỉnh để tạo nút bấm sửa địa chỉ, nút đăng xuất.

InputBorder: Widget tùy chỉnh để tạo trường nhập liệu thông tin thay đổi.

showBarModalBottomSheet: Hiển thị bottom sheet khi thêm hoặc sửa địa chỉ.

## 4.5. Chức năng

### 4.5.1. Đăng nhập



Hình 4. 15. Chức năng đăng nhập

Nhập email và mật khẩu: Người dùng nhập email và mật khẩu vào các InputBase widget.

Nhân nút "Đăng nhập": Nếu email và mật khẩu không rỗng, nút "Đăng nhập" được

kích hoạt và gọi hàm `_login`.

Hiển thị loading: Khi `_login` được gọi, `loadingService.showLoading()` hiển thị overlay loading.

Gọi API đăng nhập: `AuthService.loginUser` được gọi với email và mật khẩu để thực hiện yêu cầu đăng nhập.

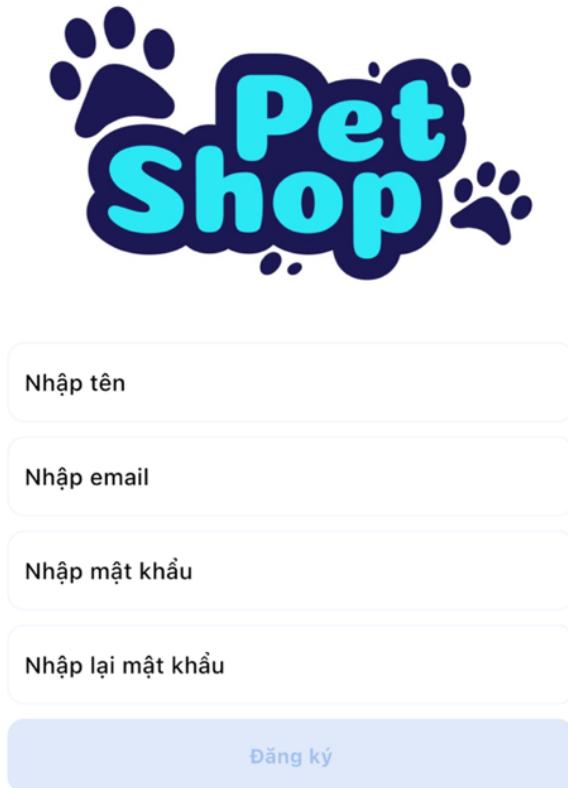
Xử lý phản hồi: Nếu phản hồi từ API chứa token, lưu token và refreshToken vào `SharedPreferences` (thư viện lưu trữ dữ liệu cục bộ của Flutter).

Điều hướng đến `MyHomePage` nếu đăng nhập thành công.

Nếu đăng nhập thất bại, hiển thị thông báo lỗi bằng `Utils().showToast`.

Ẩn loading: Dù đăng nhập thành công hay thất bại, `loadingService.hideLoading` được gọi để ẩn overlay loading.

#### 4.5.2. Đăng ký



Hình 4. 16. Chức năng đăng ký

Nhập thông tin người dùng: Người dùng nhập tên, email, số điện thoại, mật khẩu và nhập lại mật khẩu vào các InputBase widget.

Nhấn nút "Đăng ký": Nếu tất cả các trường đều không rỗng, nút "Đăng ký" được kích hoạt và gọi hàm register.

Kiểm tra tính hợp lệ: Hàm isValidateEmail và validatePhone được gọi để kiểm tra tính hợp lệ của email và số điện thoại.

Nếu email hoặc số điện thoại không hợp lệ, hiển thị thông báo lỗi và kết thúc hàm.

Hiển thị loading: Khi register được gọi, loadingService.showLoading() hiển thị overlay loading.

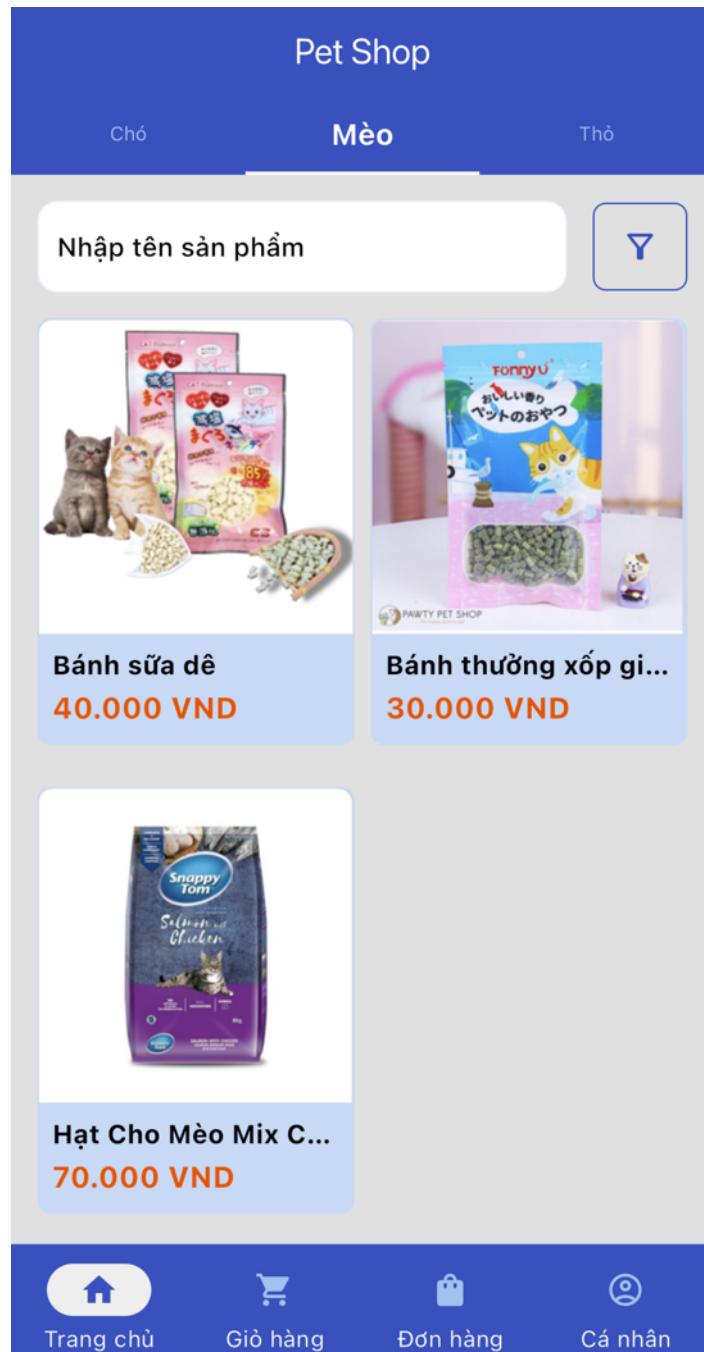
Gọi API đăng ký: AuthService.register được gọi với email, mật khẩu, tên và số điện thoại để thực hiện yêu cầu đăng ký.

Xử lý phản hồi: Nếu phản hồi từ API chứa thông tin thành công, hiển thị thông báo thành công và điều hướng người dùng trở về trang trước đó.

Nếu đăng ký thất bại, hiển thị thông báo lỗi bằng Utils().showToast.

Ẩn loading: Dù đăng ký thành công hay thất bại, loadingService.hideLoading được gọi để ẩn overlay loading.

#### 4.5.3. Trang chủ



Hình 4. 17. Chức năng trang chủ

Khởi tạo màn hình: Khi màn hình được khởi tạo, sử dụng DefaultTabController để quản lý trạng thái của TabBar và TabBarView.

Lấy danh sách danh mục: handleGetCategories() được gọi để lấy danh sách danh mục sản phẩm từ API của AuthService.

Danh mục sản phẩm gồm: dogCategories, catCategories, rabbitCategories.

Hiển thị danh mục và sản phẩm: Mỗi tab trong TabBar tương ứng với một loại danh

mục (chó, mèo, thỏ).

Khi người dùng chọn tab, TabBarView hiển thị nội dung tương ứng với tab đó thông qua widget NestedTabBar.

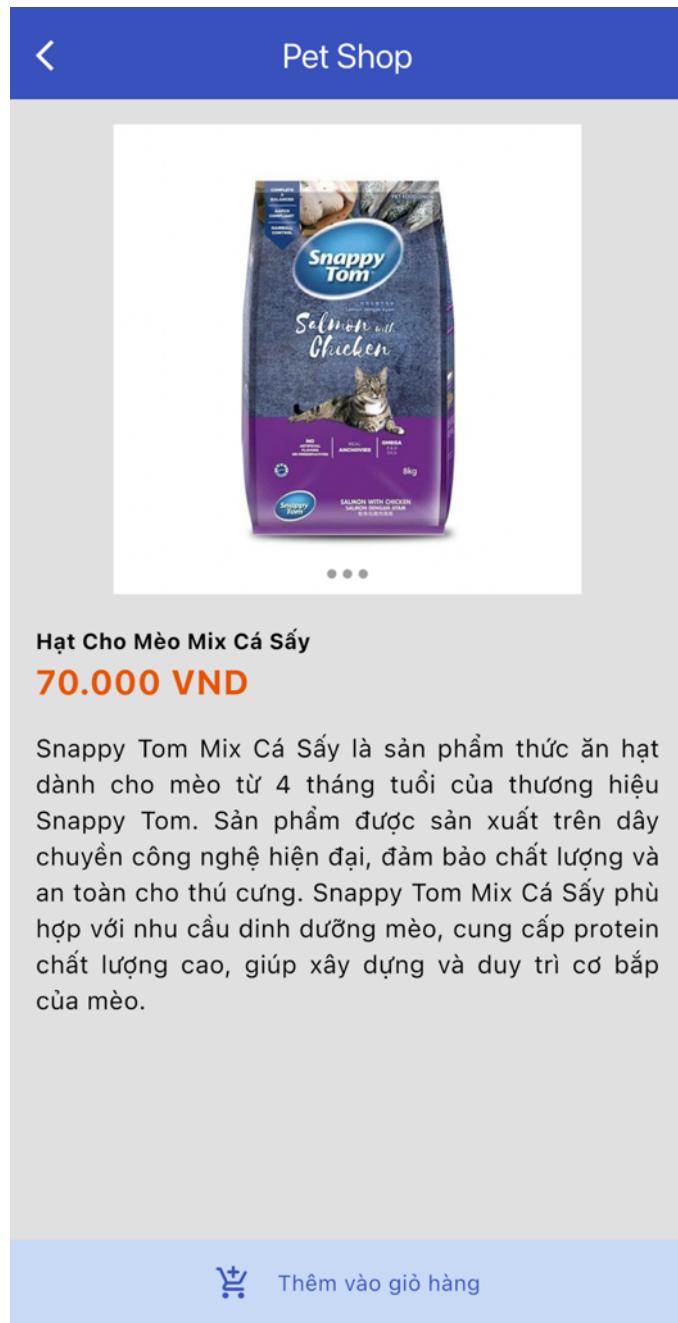
Trong NestedTabBar, hàm handleGetProduct() được gọi để lấy danh sách sản phẩm theo danh mục đã chọn từ API của AuthService.

Cập nhật danh sách sản phẩm: Khi danh mục được thay đổi, handleGetProduct() được gọi lại để cập nhật danh sách sản phẩm.

Danh sách sản phẩm được hiển thị dưới dạng lưới sử dụng GridView.builder.

Tìm kiếm và lọc sản phẩm: Người dùng có thể tìm kiếm sản phẩm bằng cách nhập từ khóa vào InputBase. Nút lọc (ButtonCustomContent) cho phép người dùng chọn danh mục sản phẩm từ loại.

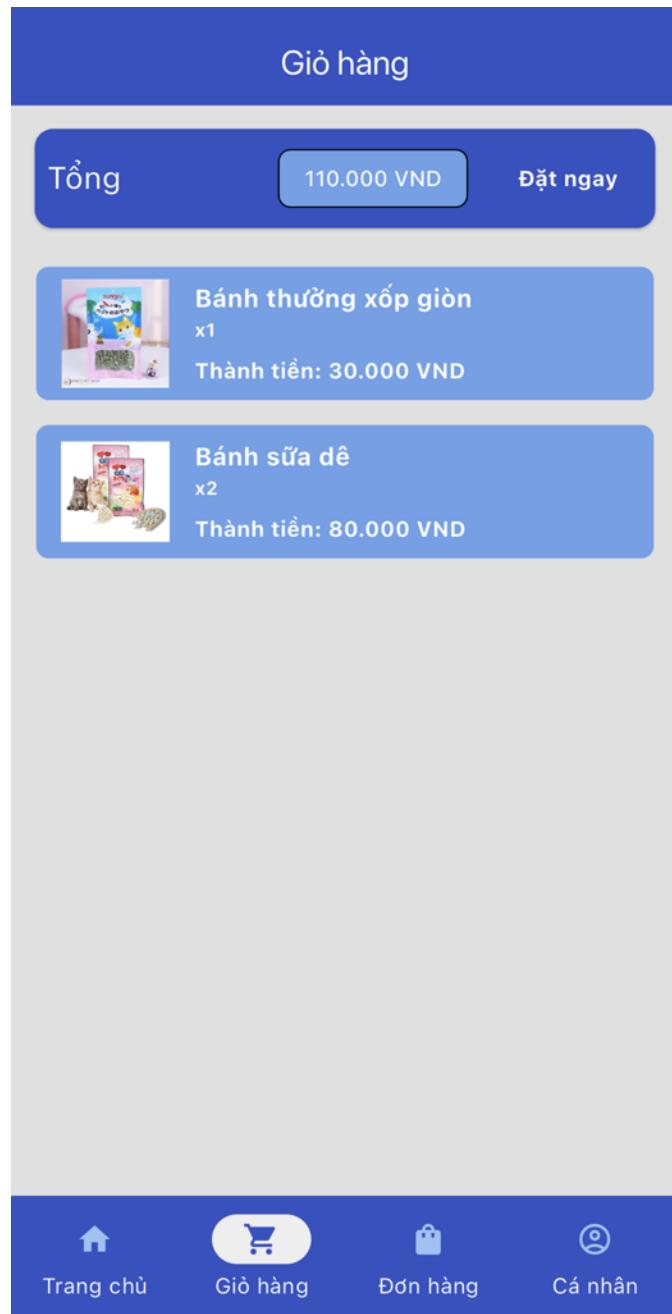
#### 4.5.4. Chi tiết sản phẩm



Hình 4. 18.Chức năng chi tiết sản phẩm

Khi người dùng nhấn vào nút "Thêm vào giỏ hàng", hàm handleAddToCart() được gọi. Nếu không có ID giỏ hàng hiện tại, hàm handleCreateCheckOutId() sẽ được gọi để tạo mới. Sau khi thêm sản phẩm vào giỏ thành công, hiển thị thông báo kết quả cho người dùng.

#### 4.5.5. Giỏ hàng



Hình 4. 19. Chức năng giỏ hàng

Xử lý thêm sản phẩm vào giỏ hàng:

Khi màn hình giỏ hàng được khởi tạo, hàm initState được gọi và thực hiện handleGetUserInfo sau 1 giây.

Hàm handleGetUserInfo gọi fetchUserInfo từ AuthService để lấy thông tin người dùng và cập nhật trạng thái.

Sau khi có thông tin người dùng, handleGetItem được gọi để lấy thông tin các sản phẩm trong giỏ hàng dựa trên ID checkout của người dùng.

**Hiển thị giỏ hàng:** Giao diện giỏ hàng được xây dựng trong hàm build, bao gồm AppBar, Column, và ListView.builder để hiển thị danh sách sản phẩm.

**Đặt hàng:** Khi người dùng nhấn nút đặt hàng, hàm handleOrder được gọi để cập nhật địa chỉ giao hàng và tạo đơn hàng mới. Nếu thành công, giỏ hàng sẽ được làm trống và hiển thị thông báo thành công.

#### 4.5.6. Lịch sử đơn hàng



Hình 4. 20. Chức năng lịch sử đơn hàng

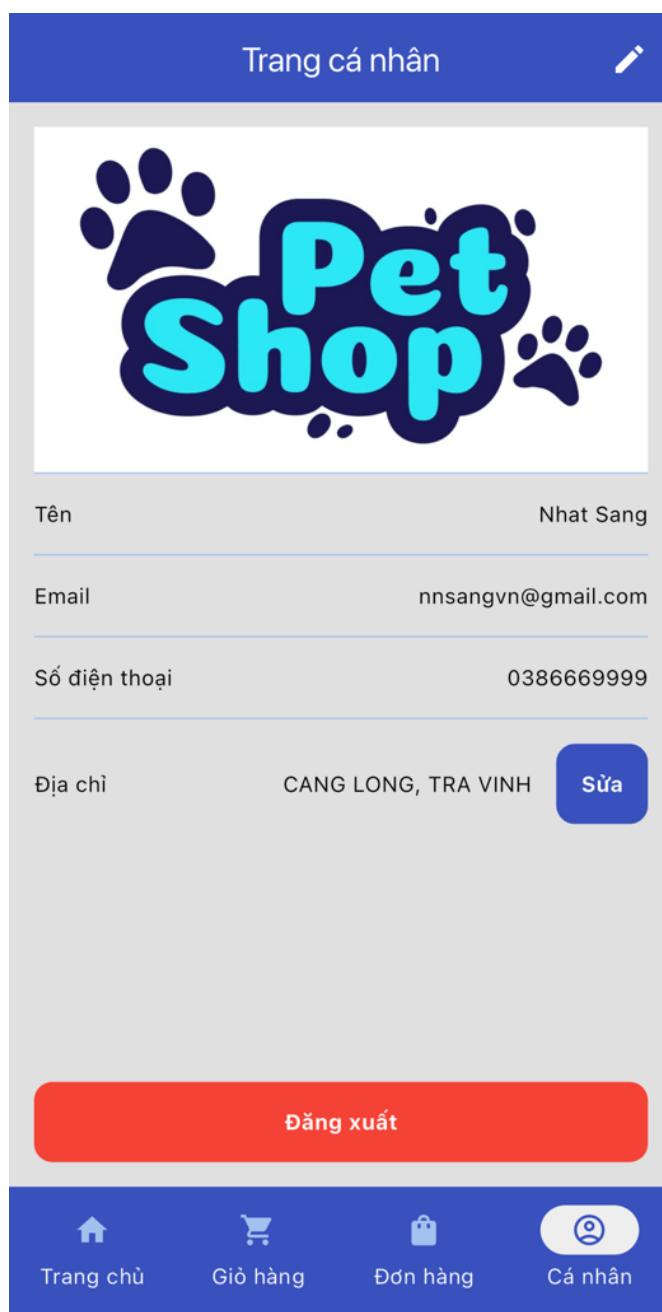
Xử lý danh sách đơn hàng:

Gọi handleGetUserInfo() để lấy thông tin người dùng.

Hiển thị overlay loading trong khi dữ liệu đang được tải. Khi lấy được thông tin, cập nhật trạng thái với thông tin người dùng.

Hiển thị danh sách đơn hàng bằng ListView.builder. Mỗi đơn hàng được hiển thị trong một Container với chi tiết từng dòng sản phẩm. Tính tổng tiền cho mỗi đơn hàng và hiển thị dưới mỗi danh sách sản phẩm.

#### 4.5.7. Thông tin cá nhân



Hình 4. 21. Chức năng thông tin cá nhân

Khi người dùng mở trang cá nhân: Hàm handleGetUserInfo được thực thi để lấy thông tin người dùng từ server và cập nhật trạng thái.

Khi người dùng nhấn nút chỉnh sửa thông tin cá nhân: Hiển thị form chỉnh sửa.

Các trường nhập liệu được cập nhật với thông tin hiện tại của người dùng.

Khi người dùng nhập thông tin mới và nhấn nút lưu: Hàm handleChangeInfo được gọi. Hàm validatePhone kiểm tra định dạng số điện thoại. Nếu hợp lệ, hàm updateAccount từ AuthService được gọi để cập nhật thông tin người dùng trên server. Trang thái được cập nhật với thông tin mới và form chỉnh sửa được ẩn đi.

Khi người dùng thêm hoặc sửa địa chỉ: Hàm handleShowModalAddAddress được gọi để hiển thị bottom sheet chứa các trường nhập liệu cho địa chỉ.

Khi nhấn nút lưu, hàm handleCreateAddress được gọi để tạo mới hoặc cập nhật địa chỉ trên server và cập nhật trang thái.

Khi người dùng nhấn nút đăng xuất: Hàm \_logout được gọi. Gọi hàm logout từ AuthService, hiển thị loading trong quá trình đăng xuất. Chuyển hướng người dùng về trang đăng nhập.

## **CHƯƠNG 5. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN**

### **5.1. Kết luận**

Đồ án tốt nghiệp này đã nghiên cứu và phát triển một ứng dụng quản lý cửa hàng thú cưng, sử dụng công nghệ Flutter cho giao diện người dùng và Saleor cho hệ thống backend. Kết quả cho thấy ứng dụng đã tối ưu hóa quy trình quản lý sản phẩm, đơn hàng và khách hàng. Những đóng góp mới của nghiên cứu bao gồm việc kết hợp thành công các công nghệ tiên tiến như Flutter và Saleor trong một ứng dụng thực tiễn, cung cấp một giải linh hoạt cho việc quản lý cửa hàng bán sản phẩm cho thú cưng. Đồ án còn hạn chế chưa tích hợp thanh toán trực tuyến.

Quá trình phát triển gặp phải một số khó khăn khi tích hợp Saleor. Mặc dù Saleor cung cấp hướng dẫn chi tiết, nhưng với quá nhiều chức năng, người mới gặp khó khăn trong việc tiếp cận và sử dụng. Việc làm quen với nền tảng thương mại điện tử phức tạp này đòi hỏi thời gian và công sức.

### **5.2. Hướng phát triển**

Phát triển chức năng mã giảm giá để khuyến khích người tiêu dùng mua sắm, cho phép nhập mã giảm giá trong quá trình thanh toán nhằm nâng cao trải nghiệm và tăng doanh thu cho cửa hàng. Hệ thống này có thể bao gồm các chương trình khuyến mãi theo mùa hoặc cho sản phẩm mới, tạo động lực cho hành vi mua sắm.

Hệ thống tích hợp các công cụ hỗ trợ như chat trực tuyến với nhân viên, hệ thống đánh giá và phản hồi sản phẩm, cùng bảng điều khiển quản lý tài khoản cá nhân. Điều này giúp người dùng dễ dàng quản lý thông tin, theo dõi lịch sử mua hàng và tương tác với cộng đồng. Ngoài ra, người dùng cũng nhận thông báo về sản phẩm mới, khuyến mãi đặc biệt và cập nhật từ cửa hàng.

## **DANH MỤC TÀI LIỆU THAM KHẢO**

- [1] “Saleor Documentation | Saleor Commerce Documentation”. Truy cập: 14 Tháng Mười 2024. [Online]. Available at: <https://docs.saleor.io/>
- [2] A. Ritsilä, “GraphQL: The API Design Revolution”. Truy cập: 14 Tháng Mười 2024. [Online]. Available at: <http://www.theseus.fi/handle/10024/141989>
- [3] “Introduction to GraphQL | GraphQL”. Truy cập: 14 Tháng Mười 2024. [Online]. Available at: <https://graphql.org/learn/>
- [4] T. Andersson và H. Reinholdsson, *REST API vs GraphQL: A literature and experimental study*. 2021. Truy cập: 14 Tháng Mười 2024. [Online]. Available at: <https://urn.kb.se/resolve?urn=urn:nbn:se:hkr:diva-22063>
- [5] “Flutter documentation”. Truy cập: 14 Tháng Mười 2024. [Online]. Available at: <https://docs.flutter.dev>
- [6] “Dart documentation”. Truy cập: 14 Tháng Mười 2024. [Online]. Available at: <https://dart.dev/guides/>
- [7] D. Slepnev, “State Management Approaches in Flutter”.

## PHỤ LỤC 1. API đăng ký

```
● ○ ●
1 Future<Map<String, dynamic>> register({
2   required String email,
3   required String password,
4   required String firstName,
5   required String lastName,
6   required String redirectUrl,
7 } ) async {
8   const String registerMutation = """
9 mutation accountRegister(\$email: String!, \$password: String!, \$firstName: String!, \$lastName: String!, \$redirectUrl: String!) {
10   accountRegister(input: { email: \$email, password: \$password, firstName: \$firstName, lastName: \$lastName, redirectUrl: \$redirectUrl }) {
11     accountErrors {
12       field
13       message
14     }
15     user {
16       id
17       email
18       firstName
19       lastName
20       isActive
21       isStaff
22       dateJoined
23       lastLogin
24       languageCode
25       defaultShippingAddress {
26         streetAddress1
27         city
28         postalCode
29       }
30       defaultBillingAddress {
31         streetAddress1
32         city
33         postalCode
34       }
35       addresses {
36         streetAddress1
37         city
38         postalCode
39       }
40       avatar {
41         url
42       }
43     }
44   }
45 }
46 """
47
48 final MutationOptions options = MutationOptions(
49   document: gql(registerMutation),
50   variables: {
51     'email': email,
52     'password': password,
53     'firstName': firstName,
54     'lastName': lastName,
55     'redirectUrl': redirectUrl,
56     'channelSlug': 'default'
57   },
58 );
59
60 final QueryResult result = await client.value.mutate(options);
61
62 if (result.hasException) {
63   print(result.exception.toString());
64   return null;
65 }
66
67 return result.data;
68 }
```

*API đăng ký*

### ❖ Mô tả

Hàm register thực hiện quá trình đăng ký người dùng mới trên hệ thống. Nó gửi một yêu cầu mutation GraphQL đến máy chủ để tạo một tài khoản người dùng mới với các

thông tin được cung cấp. Hàm này bao gồm các thông tin cần thiết như email, mật khẩu, tên và họ của người dùng, cũng như URL chuyển hướng sau khi đăng ký thành công.

#### ❖ Tham số

- email: Địa chỉ email của người dùng (bắt buộc).
- password: Mật khẩu mà người dùng muốn sử dụng (bắt buộc).
- firstName: Tên riêng của người dùng (bắt buộc).
- lastName: Họ của người dùng (bắt buộc).
- password: URL mà người dùng sẽ được chuyển hướng đến sau khi hoàn tất quá trình đăng ký (bắt buộc).

#### ❖ Quá trình thực hiện

Định nghĩa Mutation:

Một mutation GraphQL được định nghĩa dưới dạng chuỗi, được gọi là accountRegister. Mutation này yêu cầu các thông tin đầu vào là email, mật khẩu, tên, họ và URL chuyển hướng.

Kết quả trả về từ mutation sẽ bao gồm thông tin người dùng mới và các lỗi có thể xảy ra trong quá trình đăng ký.

Cấu hình Options:

Tạo một đối tượng MutationOptions để cấu hình yêu cầu mutation, bao gồm tài liệu GraphQL và biến đầu vào

Gửi yêu cầu:

Kiểm tra xem yêu cầu có gặp lỗi không. Nếu có, in ra thông báo lỗi và trả về null.

Nếu không có lỗi, trả về dữ liệu của người dùng đã đăng ký mới.

#### ❖ Công dụng

Hàm này rất hữu ích trong việc quản lý quá trình đăng ký người dùng trong ứng dụng, giúp tự động hóa việc tạo tài khoản và xử lý thông tin người dùng một cách hiệu quả.

## PHỤ LỤC 2. API đăng nhập

```
● ● ●  
1 Future<Map<String, dynamic>?> loginUser(String email, String password) async {  
2   const String loginMutation = ''  
3   mutation TokenCreate(\$email: String!, \$password: String!) {  
4     tokenCreate(email: \$email, password: \$password) {  
5       token  
6       refreshToken  
7       csrfToken  
8       user {  
9         id  
10        email  
11        firstName  
12        lastName  
13        checkoutIds  
14        defaultBillingAddress {  
15          firstName  
16          lastName  
17          companyName  
18          streetAddress1  
19          streetAddress2  
20          city  
21          cityArea  
22          postalCode  
23          country {  
24            code  
25            country  
26          }  
27          countryArea  
28          phone  
29          isDefaultBillingAddress  
30          isDefaultShippingAddress  
31        }  
32      }  
33      errors {  
34        field  
35        message  
36        code  
37      }  
38    }  
39  }  
40  '';  
41  
42  final MutationOptions options = MutationOptions(  
43    document: gql(loginMutation),  
44    variables: <String, dynamic>{  
45      'email': email,  
46      'password': password,  
47    },  
48  );  
49  
50  final QueryResult result = await client.value.mutate(options);  
51  
52  if (result.hasException) {  
53    print(result.exception.toString());  
54    return null;  
55  }  
56  
57  return result.data?['tokenCreate'];  
58}
```

*API đăng nhập*

## ❖ Mô tả

Hàm loginUser thực hiện quá trình đăng nhập người dùng vào hệ thống. Nó gửi một yêu cầu mutation GraphQL để xác thực thông tin đăng nhập bằng email và mật khẩu. Khi đăng nhập thành công, hệ thống sẽ trả về các token và thông tin chi tiết của người dùng, bao gồm địa chỉ thanh toán và giao hàng mặc định.

## ❖ Tham số

- email: Địa chỉ email của người dùng (bắt buộc).
- password: Mật khẩu mà người dùng muốn sử dụng (bắt buộc).

## ❖ Quá trình thực hiện

Định nghĩa Mutation:

Một mutation GraphQL được định nghĩa dưới dạng chuỗi có tên tokenCreate. Mutation này yêu cầu email và mật khẩu đầu vào để tạo một phiên đăng nhập cho người dùng.

Kết quả trả về bao gồm token xác thực, refresh token, csrf token, và thông tin người dùng (ví dụ: id, email, tên, họ, các địa chỉ mặc định).

Cấu hình Options:

Tạo một đối tượng MutationOptions để cấu hình yêu cầu mutation, bao gồm tài liệu GraphQL và các biến đầu vào như email và mật khẩu.

## ❖ Công dụng

Hàm này được sử dụng để xử lý quá trình đăng nhập người dùng trong ứng dụng, xác thực và quản lý phiên làm việc của họ thông qua token và các thông tin bảo mật khác.

## PHỤ LỤC 3. API đăng xuất



```
1 Future<void> logout(String pluginId) async {
2     const String logoutMutation = '''
3     mutation externalLogout(\$input: JSONString!, \$pluginId: String!) {
4         externalLogout(input: \$input, pluginId: \$pluginId) {
5             accountErrors {
6                 field
7                 message
8             }
9         }
10    }
11 ''';
12
13     final String inputJson = '{}';
14
15     final MutationOptions options = MutationOptions(
16         document: gql(logoutMutation),
17         variables: {
18             'input': inputJson,
19             'pluginId': pluginId,
20         },
21     );
22
23     final QueryResult result = await client.value.mutate(options);
24
25     if (result.hasException) {
26         print('Logout error: ${result.exception.toString()}');
27     } else {
28         final List<dynamic> accountErrors =
29             result.data?['externalLogout']['accountErrors'];
30
31         if (accountErrors.isNotEmpty) {
32             List<String> messages = [];
33             // Nếu có lỗi, in ra lỗi
34             accountErrors.forEach((error) {
35                 messages.add(error?['message']);
36                 Utils().showToast(messages.toString(), ToastType.failed);
37             });
38         } else {
39             final prefs = await SharedPreferences.getInstance();
40             await prefs.clear();
41         }
42     }
43 }
```

### ❖ Mô tả

Hàm logout thực hiện quá trình đăng xuất người dùng khỏi hệ thống thông qua một plugin bên ngoài. Nó gửi một yêu cầu mutation GraphQL để thông báo cho hệ thống rằng người dùng đã đăng xuất. Hàm cũng xử lý các lỗi tiềm ẩn và xóa dữ liệu của người dùng khỏi bộ nhớ cục bộ (SharedPreferences) khi đăng xuất thành công.

### ❖ Tham số

pluginId: ID của plugin được sử dụng để xử lý việc đăng xuất (bắt buộc).

### ❖ Quá trình thực hiện

Định nghĩa Mutation:

Một mutation GraphQL được định nghĩa dưới dạng chuỗi, tên là externalLogout. Mutation này yêu cầu hai đầu vào: một chuỗi JSON (trong trường hợp này là rỗng) và pluginId để xác định plugin xử lý đăng xuất.

Kết quả trả về từ mutation bao gồm các lỗi tài khoản có thể xảy ra trong quá trình đăng xuất.

Cấu hình Options:

Tạo một đối tượng MutationOptions để cấu hình yêu cầu mutation, bao gồm tài liệu GraphQL và các biến đầu vào.

### ❖ Công dụng

Hàm này giúp xử lý việc đăng xuất người dùng một cách hiệu quả, đặc biệt khi cần tích hợp với các plugin bên ngoài. Nó đảm bảo người dùng được đăng xuất khỏi hệ thống và các dữ liệu liên quan được xóa khỏi bộ nhớ cục bộ.

## PHỤ LỤC 4. API lấy danh sách category



```
1 Future<List<dynamic>?> fetchCategories() async {
2     const String getCategoriesQuery = '''
3     query {
4         categories(first: 100) {
5             edges {
6                 node {
7                     id
8                     name
9                     slug
10                }
11            }
12        }
13    }
14    '''
15
16    final QueryOptions options = QueryOptions(
17        document: gql(getCategoriesQuery),
18    );
19
20    final QueryResult result = await client.value.query(options);
21
22    if (result.hasException) {
23        print(result.exception.toString());
24        return null;
25    }
26
27    return result.data?['categories']['edges']
28        .map((edge) => edge['node'])
29        .toList();
30 }
```

*API lấy danh sách category*

### ❖ Mô tả

Hàm fetchCategories thực hiện việc truy vấn danh sách các danh mục (categories) từ hệ thống. Nó gửi một yêu cầu query GraphQL để lấy thông tin về các danh mục sản phẩm có sẵn, bao gồm id, name và slug của mỗi danh mục. Hàm này trả về danh sách các danh mục nếu quá trình truy vấn thành công.

### ❖ Tham số

Không có tham số đầu vào.

### ❖ Quá trình thực hiện

Định nghĩa Mutation:

Một truy vấn GraphQL được định nghĩa dưới dạng chuỗi, yêu cầu lấy thông tin của các danh mục với tối đa 100 danh mục đầu tiên. Mỗi danh mục bao gồm id, name và slug.

Cấu hình Options:

Tạo một đối tượng QueryOptions để cấu hình yêu cầu query, bao gồm tài liệu GraphQL.

### ❖ Công dụng

Hàm này được sử dụng để lấy danh sách các danh mục sản phẩm trong hệ thống, rất hữu ích trong việc hiển thị các danh mục cho người dùng khi cần duyệt qua sản phẩm theo danh mục.

## PHỤ LỤC 5. API lấy danh sách sản phẩm theo category

```
● ● ●

1 Future<List<ProductModel>> fetchProductsForUser(
2     String categoryId, String channel,
3     {Map<String, dynamic>? filter}) async {
4     const String getProductsForUserQuery = '''
5 query GetProductsForUser($categoryId: ID!, $first: Int, $channel: String!, $filter: ProductFilterInput) {
6     category(id: $categoryId) {
7         products(first: $first, channel: $channel, filter: $filter) {
8             edges {
9                 node {
10                     id
11                     name
12                     description
13                     thumbnail {
14                         url
15                     }
16                     media {
17                         url
18                     }
19                     pricing {
20                         priceRange {
21                             start {
22                                 net {
23                                     amount
24                                     currency
25                                 }
26                             }
27                         }
28                     }
29                     variants {
30                         id
31                         name
32                         sku
33                     }
34                 }
35             }
36         }
37     }
38 }
39 ''';
40
41     final QueryOptions options = QueryOptions(
42         document: gql(getProductsForUserQuery),
43         variables: {
44             'categoryId': categoryId,
45             'first': 100,
46             'channel': channel,
47             'filter': filter,
48         },
49     );
50
51     final QueryResult result = await client.value.query(options);
52
53     if (result.hasException) {
54         print('Error fetching products for user: ${result.exception.toString()}`);
55         return [];
56     }
57
58     return result.data?['category']['products']['edges']
59         .map<ProductModel>((edge) => ProductModel.fromMap(edge['node']))
60         .toList();
61 }
```

*API lấy danh sách sản phẩm theo category*

## ❖ Mô tả

Hàm fetchProductsForUser thực hiện việc truy vấn danh sách các sản phẩm thuộc một danh mục cụ thể. Nó gửi yêu cầu query GraphQL để lấy thông tin về sản phẩm, bao gồm các thông tin như id, name, description, thumbnail, media, pricing, và các biến thể (variants) của sản phẩm. Hàm này trả về danh sách sản phẩm nếu quá trình truy vấn thành công.

## ❖ Tham số

- categoryId: ID của danh mục cần lấy sản phẩm (bắt buộc).
- channel: Kênh hiển thị sản phẩm (ví dụ: "default") (bắt buộc).
- filter: Một đối tượng tùy chọn để lọc các sản phẩm theo các điều kiện khác nhau.

## ❖ Quá trình thực hiện

Định nghĩa Mutation:

Một truy vấn GraphQL được định nghĩa dưới dạng chuỗi, yêu cầu lấy các sản phẩm thuộc danh mục được chỉ định với id của danh mục, kênh và điều kiện lọc tùy chọn.

Thông tin sản phẩm bao gồm: id, name, description, hình ảnh thu nhỏ (thumbnail), danh sách hình ảnh (media), thông tin giá (pricing), và các biến thể của sản phẩm.

Cấu hình Options:

Tạo một đối tượng QueryOptions để cấu hình yêu cầu query, bao gồm tài liệu GraphQL và các biến đầu vào (categoryId, first, channel, và filter).

## ❖ Công dụng

Hàm này rất hữu ích khi cần lấy danh sách các sản phẩm theo danh mục trong hệ thống. Nó giúp hiển thị sản phẩm cho người dùng theo danh mục đã chọn và hỗ trợ lọc các sản phẩm theo nhu cầu.

## PHỤ LỤC 6. API tạo phiếu thanh toán



```
1 Future<Map<String, dynamic>?> createCheckout(
2     String email, List<Map<String, dynamic>> lines, String channel) async {
3     const String createCheckoutMutation = ''
4     mutation CreateCheckout(\$email: String!, \$lines: [CheckoutLineInput!]!, \$channel: String!) {
5         checkoutCreate(
6             input: {
7                 email: \$email
8                 lines: \$lines
9                 channel: \$channel
10            }
11        ) {
12            checkout {
13                id
14                token
15                email
16                lines {
17                    id
18                    quantity
19                    variant {
20                        id
21                        name
22                    }
23                }
24            }
25            errors {
26                field
27                message
28            }
29        }
30    }
31    '';
32
33    final MutationOptions options = MutationOptions(
34        document: gql(createCheckoutMutation),
35        variables: {
36            'email': email,
37            'lines': lines, // Danh sách sản phẩm với variantId và quantity
38            'channel': channel,
39        },
40    );
41
42    final QueryResult result = await client.value.mutate(options);
43
44    if (result.hasException) {
45        return null;
46    }
47
48    return result.data?['checkoutCreate'];
49 }
```

*API tạo phiếu thanh toán*

## ❖ Mô tả

Hàm createCheckout thực hiện việc tạo một checkout (giỏ hàng) mới cho người dùng trên hệ thống. Nó gửi một mutation GraphQL để tạo checkout với thông tin email người dùng, danh sách sản phẩm muốn mua, và kênh bán hàng được chỉ định. Kết quả trả về bao gồm thông tin của checkout đã tạo, như id, token, email, và các dòng sản phẩm (lines) trong checkout.

## ❖ Tham số

- email: Địa chỉ email của người dùng (bắt buộc).
- lines: Danh sách sản phẩm được thêm vào giỏ hàng (bắt buộc). Mỗi sản phẩm bao gồm variantId và số lượng (quantity) ...
- channel: Kênh bán hàng (ví dụ: "default") mà checkout sẽ được tạo (bắt buộc).

## ❖ Quá trình thực hiện

Định nghĩa Mutation:

Một mutation GraphQL được định nghĩa dưới dạng chuỗi, yêu cầu tạo checkout với thông tin đầu vào là email, danh sách sản phẩm và kênh.

Thông tin trả về bao gồm id, token, email của checkout cùng với các dòng sản phẩm, bao gồm id, quantity, và biến thể sản phẩm (variant).

Cấu hình Options:

Tạo một đối tượng MutationOptions để cấu hình mutation, bao gồm tài liệu GraphQL và biến đầu vào (email, lines, và channel).

## ❖ Công dụng

Hàm này giúp tự động tạo checkout cho người dùng, quản lý danh sách sản phẩm muốn mua và kết nối với kênh bán hàng phù hợp. Đây là bước đầu tiên trong quá trình đặt hàng và giúp tạo ra trải nghiệm mua sắm mượt mà cho người dùng.

## PHỤ LỤC 7. API thêm sản phẩm vào giỏ hàng



```
1 Future<bool> addToCart(
2     String checkoutId, String variantId, int quantity) async {
3     const String addToCartMutation = '''
4     mutation CheckoutLinesAdd(\$checkoutId: ID!, \$lines: [CheckoutLineInput!]!) {
5         checkoutLinesAdd(checkoutId: \$checkoutId, lines: \$lines) {
6             checkout {
7                 id
8                 lines {
9                     quantity
10                    variant {
11                        id
12                        name
13                    }
14                }
15            }
16            errors {
17                field
18                message
19            }
20        }
21    }
22    ''';
23
24    final MutationOptions options = MutationOptions(
25        document: gql(addToCartMutation),
26        variables: {
27            'checkoutId': checkoutId,
28            'lines': [
29                {
30                    'variantId': variantId,
31                    'quantity': quantity,
32                },
33            ],
34        },
35    );
36
37    final QueryResult result = await client.value.mutate(options);
38
39    if (result.hasException) {
40        print(result.exception.toString());
41        return false;
42    } else {
43        return true;
44    }
45}
```

*API thêm sản phẩm vào giỏ hàng*

## ❖ Mô tả

Hàm addToCart thêm một sản phẩm vào giỏ hàng (checkout) đã được tạo trước đó. Nó thực hiện mutation GraphQL để thêm dòng sản phẩm với biến thể sản phẩm (variant) và số lượng cụ thể vào checkout hiện tại. Kết quả trả về bao gồm thông tin về các dòng sản phẩm hiện có trong checkout và các lỗi nếu có.

## ❖ Tham số

- checkoutId: ID của giỏ hàng (checkout) mà sản phẩm sẽ được thêm vào (bắt buộc).
- variantId: ID của biến thể sản phẩm muốn thêm vào giỏ hàng (bắt buộc).
- quantity: Số lượng của sản phẩm muốn thêm vào giỏ hàng (bắt buộc).

## ❖ Quá trình thực hiện

Định nghĩa Mutation:

Một mutation GraphQL được định nghĩa dưới dạng chuỗi, yêu cầu thêm dòng sản phẩm vào giỏ hàng dựa trên checkoutId, danh sách dòng sản phẩm bao gồm variantId và quantity.

Kết quả trả về bao gồm thông tin về giỏ hàng (checkout) với danh sách các dòng sản phẩm đã thêm, cùng với các lỗi nếu có.

Cấu hình Options:

Tạo một đối tượng MutationOptions để cấu hình mutation, bao gồm tài liệu GraphQL và biến đầu vào (checkoutId, danh sách lines chứa variantId và quantity).

## ❖ Công dụng

Hàm này giúp tự động tạo checkout cho người dùng, quản lý danh sách sản phẩm muốn mua và kết nối với kênh bán hàng phù hợp. Đây là bước đầu tiên trong quá trình đặt hàng và giúp tạo ra trải nghiệm mua sắm mượt mà cho người dùng.

## PHỤ LỤC 8. API lấy thông tin người dùng đăng nhập



```
1 Future<Map<String, dynamic>?> fetchUserInfo() async {
2     const String getUserQuery = ''
3     query {
4         me {
5             id
6             email
7             firstName
8             lastName
9             isActive
10            isStaff
11            dateJoined
12            lastLogin
13            languageCode
14            defaultShippingAddress {
15                streetAddress1
16                city
17                postalCode
18            }
19            defaultBillingAddress {
20                streetAddress1
21                city
22                postalCode
23            }
24            addresses {
25                id
26                firstName
27                lastName
28                streetAddress1
29                city
30                postalCode
31            }
32            avatar {
33                url
34            }
35            checkout {
36                id
37                totalPrice {
38                    gross {
39                        amount
40                        currency
41                    }
42                }
43                lines {
44                    quantity
45                    variant {
46                        id
47                        name
48                        product {
49                            name
50                            thumbnail {
51                                url
52                            }
53                        }
54                    }
55                }
56            }
}
```

●

●

●

```
1 orders(first: 100) {
2     edges {
3         node {
4             id
5             number
6             created
7             status
8             total {
9                 gross {
10                    amount
11                    currency
12                }
13            }
14            lines {
15                quantity
16            variant {
17                id
18                name
19                product {
20                    name
21                    thumbnail {
22                        url
23                    }
24                    pricing {
25                        priceRange {
26                            start {
27                                net {
28                                    currency
29                                    amount
30                                }
31                            }
32                        }
33                    }
34                }
35            }
36        }
37    }
38}
39}
40}
41}
42''';
```

```
43
44 final QueryOptions options = QueryOptions(
45     document: gql(getUserQuery),
46 );
47
48 final QueryResult result = await client.value.query(options);
49 notifyListeners();
50 if (result.hasException) {
51     print(result.exception.toString());
52     return null;
53 }
54
55 return result.data?['me'];
56 }
```

*API lấy thông tin người dùng đăng nhập*

## ❖ Mô tả

Hàm fetchUserInfo thực hiện truy vấn GraphQL để lấy thông tin chi tiết của người dùng hiện tại, bao gồm thông tin cá nhân, địa chỉ mặc định, danh sách các đơn hàng, giỏ hàng hiện tại và các thông tin liên quan. Kết quả trả về bao gồm một loạt thông tin chi tiết về người dùng, từ thông tin tài khoản đến lịch sử đơn hàng và giỏ hàng.

## ❖ Tham số

Hàm này không nhận tham số đầu vào. Nó chỉ gửi một truy vấn GraphQL đến máy chủ để lấy thông tin của người dùng đang đăng nhập.

## ❖ Quá trình thực hiện

Định nghĩa Mutation:

Một truy vấn GraphQL được định nghĩa dưới dạng chuỗi, yêu cầu lấy thông tin chi tiết về người dùng hiện tại.

Truy vấn trả về các thông tin về người dùng bao gồm: email, tên, trạng thái kích hoạt, ngày tham gia, thông tin địa chỉ mặc định, danh sách địa chỉ khác, giỏ hàng hiện tại (checkout), và lịch sử đơn hàng (orders).

Cấu hình Options:

Tạo đối tượng QueryOptions để cấu hình truy vấn, bao gồm tài liệu GraphQL và chính sách lấy dữ liệu (luôn lấy từ mạng - FetchPolicy.networkOnly).

## ❖ Công dụng

Hàm này rất hữu ích trong các ứng dụng yêu cầu lấy thông tin người dùng để hiển thị thông tin tài khoản, lịch sử mua hàng, hoặc giỏ hàng hiện tại. Nó giúp quản lý thông tin người dùng một cách hiệu quả và toàn diện.

## PHỤ LỤC 9. API lấy thông tin giỏ hàng

```
● ● ●
1 Future<CheckoutResponse?> fetchCheckoutItems(String checkoutId) async {
2   const String getCheckoutQuery = """
3   query GetCheckout($checkoutId: ID!) {
4     checkout(id: $checkoutId) {
5       id
6       lines {
7         quantity
8         variant {
9           id
10          name
11          product {
12            name
13            thumbnail {
14              url
15            }
16            pricing {
17              priceRange {
18                start {
19                  net {
20                    currency
21                    amount
22                  }
23                }
24              }
25            }
26          }
27        }
28      }
29      totalPrice {
30        gross {
31          amount
32          currency
33        }
34      }
35    }
36  }
37  """;
38
39  final QueryOptions options = QueryOptions(
40    document: gql(getCheckoutQuery),
41    variables: {
42      'checkoutId': checkoutId,
43    },
44  );
45
46  try {
47    final QueryResult result = await client.value.query(options);
48
49    if (result.hasException) {
50      print('Error fetching checkout items: ${result.exception.toString()}');
51      return null;
52    }
53
54    final checkoutData = result.data?['checkout'];
55    if (checkoutData != null) {
56      return CheckoutResponse.fromMap({'checkout': checkoutData});
57    } else {
58      print('No checkout found for ID: $checkoutId');
59      return null;
60    }
61  } catch (e) {
62    print('An unexpected error occurred: $e');
63    return null;
64  }
65}
```

*API lấy thông tin giỏ hàng*

### ❖ Mô tả

Hàm fetchCheckoutItems thực hiện truy vấn GraphQL để lấy thông tin chi tiết về các mặt hàng trong giỏ hàng (checkout) của người dùng dựa trên checkoutId. Hàm này sẽ trả về các thông tin về số lượng mặt hàng, sản phẩm, giá tổng, và các thông tin khác liên quan đến checkout.

### ❖ Tham số

checkoutId: ID của giỏ hàng cần lấy thông tin (bắt buộc).

### ❖ Quá trình thực hiện

Định nghĩa Mutation:

Một truy vấn GraphQL được định nghĩa dưới dạng chuỗi, yêu cầu thông tin về giỏ hàng từ checkoutId.

Truy vấn trả về các thông tin như ID của giỏ hàng, danh sách các mặt hàng (lines), mỗi mặt hàng bao gồm số lượng, thông tin biến thể (variant), sản phẩm liên quan (name, thumbnail, và pricing), và giá tổng của giỏ hàng.

Cấu hình Options:

Tạo một đối tượng QueryOptions để cấu hình truy vấn, bao gồm tài liệu GraphQL và biến đầu vào checkoutId.

### ❖ Công dụng

Hàm này rất hữu ích trong việc lấy thông tin về giỏ hàng của người dùng, cho phép hiển thị chi tiết các sản phẩm mà người dùng đã thêm vào giỏ, cùng với các thông tin cần thiết để thanh toán hoặc chỉnh sửa giỏ hàng.

## PHỤ LỤC 10. API cập nhật thông tin tài khoản



```
1 Future<Map<String, dynamic>> updateAccount({  
2     required String firstName,  
3     required String lastName,  
4 } ) async {  
5     const String updateAccountMutation = '''  
6     mutation accountUpdate(\$firstName: String!, \$lastName: String!) {  
7         accountUpdate(input: { firstName: \$firstName, lastName: \$lastName }) {  
8             user {  
9                 id  
10                email  
11                firstName  
12                lastName  
13            }  
14            accountErrors {  
15                field  
16                message  
17                code  
18            }  
19        }  
20    }  
21    ''';  
22  
23    final MutationOptions options = MutationOptions(  
24        document: gql(updateAccountMutation),  
25        variables: {  
26            'firstName': firstName,  
27            'lastName': lastName, // Truyền lastName vào biến  
28        },  
29    );  
30  
31    final QueryResult result = await client.value.mutate(options);  
32  
33    if (result.hasException) {  
34        print('Update Account error: ${result.exception.toString()}');  
35        return null;  
36    }  
37  
38    return result.data?['accountUpdate']?['user'];  
39}
```

*API cập nhật thông tin tài khoản*

### ❖ Mô tả

Hàm updateAccount thực hiện cập nhật thông tin tài khoản người dùng, bao gồm tên và họ, trên hệ thống thông qua một yêu cầu mutation GraphQL. Hàm này gửi các thông

tin mới đến máy chủ để cập nhật và nhận lại thông tin người dùng đã được cập nhật.

### ❖ Tham số

- `firstName`: Tên mới của người dùng (bắt buộc).
- `lastName`: Họ mới của người dùng (bắt buộc).

### ❖ Quá trình thực hiện

Định nghĩa Mutation:

Một mutation GraphQL được định nghĩa dưới dạng chuỗi, gọi là `accountUpdate`. Mutation này yêu cầu hai thông tin đầu vào là `firstName` và `lastName`.

Kết quả trả về từ mutation sẽ bao gồm thông tin người dùng đã cập nhật và các lỗi có thể xảy ra trong quá trình cập nhật.

Cấu hình Options:

Tạo một đối tượng `MutationOptions` để cấu hình yêu cầu mutation, bao gồm tài liệu GraphQL và biến đầu vào.

### ❖ Công dụng

Hàm này rất hữu ích trong việc quản lý thông tin tài khoản người dùng trong ứng dụng, giúp người dùng dễ dàng cập nhật tên và họ của họ mà không cần phải đăng xuất hoặc tạo tài khoản mới.

## PHỤ LỤC 11. API thêm địa chỉ

```
● ● ●

1 Future<dynamic> createAddress({
2   required String firstName,
3   required String streetAddress1,
4   required String city,
5   required String phone,
6 } ) async {
7   const String mutation = '''
8     mutation AddAddress(\$input: AddressInput!) {
9       accountAddressCreate(input: \$input) {
10         address {
11           id
12           firstName
13           lastName
14           streetAddress1
15           city
16           postalCode
17         }
18         errors {
19           field
20           message
21         }
22       }
23     }
24   ''';
25
26   final MutationOptions options = MutationOptions(
27     document: gql(mutation),
28     variables: {
29       'input': {
30         'firstName': firstName,
31         'lastName': '',
32         'streetAddress1': streetAddress1,
33         'city': city,
34         'postalCode': '90213',
35         'country': 'US',
36         'countryArea': 'CA',
37       },
38     },
39   );
40
41   final QueryResult result = await client.value.mutate(options);
42
43   if (result.hasException) {
44     return null;
45   } else {
46     final address = result.data?['accountAddressCreate']['address'];
47     return address;
48   }
49 }
```

*API thêm địa chỉ*

## ❖ Mô tả

Hàm createAddress thực hiện việc tạo một địa chỉ mới cho tài khoản người dùng trong hệ thống. Hàm này gửi một yêu cầu mutation GraphQL để thêm địa chỉ với các thông tin cần thiết như tên, địa chỉ, thành phố và mã bưu điện.

## ❖ Tham số

- firstName: Tên của người dùng (bắt buộc).
- streetAddress1: Địa chỉ đường phố đầu tiên (bắt buộc).
- city: Tên thành phố (bắt buộc).
- phone: Số điện thoại của người dùng (bắt buộc).

## ❖ Quá trình thực hiện

Định nghĩa Mutation:

Một mutation GraphQL được định nghĩa dưới dạng chuỗi, gọi là AddAddress. Mutation này yêu cầu một đối tượng đầu vào có tên là input, chứa các thông tin địa chỉ.

Kết quả trả về từ mutation sẽ bao gồm thông tin địa chỉ đã được tạo và các lỗi có thể xảy ra trong quá trình tạo địa chỉ.

Cấu hình Options:

Tạo một đối tượng MutationOptions để cấu hình yêu cầu mutation, bao gồm tài liệu GraphQL và biến đầu vào chứa thông tin địa chỉ, bao gồm firstName, lastName, streetAddress1, city, postalCode, country, và countryArea.

## ❖ Công dụng

Hàm này rất hữu ích cho việc quản lý địa chỉ của người dùng trong ứng dụng, cho phép người dùng dễ dàng thêm địa chỉ mới mà không gặp phải khó khăn nào. Điều này có thể hỗ trợ quá trình thanh toán hoặc giao hàng trong các ứng dụng thương mại điện tử.

## PHỤ LỤC 12. API cập nhật địa chỉ giao hàng và địa chỉ thanh toán

```
● ● ●
1 Future<dynamic> updateCheckoutAddresses(
2     String checkoutId, Map<String, dynamic> address) async {
3     // Mutation cho cập nhật địa chỉ giao hàng
4     const String updateShippingAddressMutation = '';
5     mutation UpdateCheckoutShippingAddress($checkoutId: ID!, $shippingAddress: AddressInput!) {
6         checkoutShippingAddressUpdate(checkoutId: $checkoutId, shippingAddress: $shippingAddress) {
7             checkout {
8                 id
9                 shippingAddress {
10                     firstName
11                     lastName
12                     streetAddress1
13                     city
14                     postalCode
15                     country {
16                         code
17                     }
18                 }
19             }
20             errors {
21                 field
22                     message
23             }
24         }
25     }
26 };
27
28 // Mutation cho cập nhật địa chỉ thanh toán
29 const String updateBillingAddressMutation = '';
30 mutation UpdateCheckoutBillingAddress($checkoutId: ID!, $billingAddress: AddressInput!) {
31     checkoutBillingAddressUpdate(checkoutId: $checkoutId, billingAddress: $billingAddress) {
32         checkout {
33             id
34             billingAddress {
35                 firstName
36                 lastName
37                 streetAddress1
38                 city
39                 postalCode
40                 country {
41                     code
42                 }
43             }
44         }
45         errors {
46             field
47                 message
48         }
49     }
50 }
51 ...
52
53 // Cập nhật địa chỉ giao hàng
54 final MutationOptions shippingOptions = MutationOptions(
55     document: gql(updateShippingAddressMutation),
56     variables: {
57         'checkoutId': checkoutId,
58         'shippingAddress': address,
59     },
60 );
61
62 final QueryResult shippingResult =
63     await client.value.mutate(shippingOptions);
64
65 if (shippingResult.hasException) {
66     return null;
67 } else {
68     // Cập nhật địa chỉ thanh toán
69     final MutationOptions billingOptions = MutationOptions(
70         document: gql(updateBillingAddressMutation),
71         variables: {
72             'checkoutId': checkoutId,
73             'billingAddress': address,
74         },
75     );
76
77     final QueryResult billingResult =
78         await client.value.mutate(billingOptions);
79
80     if (billingResult.hasException) {
81         return null;
82     } else {
83         final billingAddress = billingResult;
84         return billingAddress;
85     }
86 }
87 }
```

*API cập nhật địa chỉ giao hàng và địa chỉ thanh toán*

## ❖ Mô tả

Hàm updateCheckoutAddresses thực hiện việc cập nhật địa chỉ giao hàng và địa chỉ thanh toán trong quy trình thanh toán. Hàm này gửi hai yêu cầu mutation GraphQL để cập nhật thông tin địa chỉ cho một checkout cụ thể.

## ❖ Tham số

- checkoutId: ID của checkout mà bạn muốn cập nhật địa chỉ (bắt buộc).
- address: Một đối tượng chứa thông tin địa chỉ (bao gồm tên, họ, địa chỉ, thành phố, mã bưu điện, và mã quốc gia) cần được cập nhật (bắt buộc).

## ❖ Quá trình thực hiện

Định nghĩa Mutation:

Cập nhật địa chỉ giao hàng: UpdateCheckoutShippingAddress nhận ID của checkout và địa chỉ giao hàng mới.

Cập nhật địa chỉ thanh toán: UpdateCheckoutBillingAddress nhận ID của checkout và địa chỉ thanh toán mới.

Cấu hình Options:

Cập nhật địa chỉ giao hàng: Tạo một đối tượng MutationOptions để cấu hình yêu cầu mutation, bao gồm tài liệu GraphQL và biến đầu vào chứa thông tin checkoutId và shippingAddress.

Cập nhật địa chỉ thanh toán: Tạo một đối tượng MutationOptions để cấu hình yêu cầu mutation, bao gồm tài liệu GraphQL và biến đầu vào chứa thông tin checkoutId và billingAddress.

## ❖ Công dụng

Hàm này rất quan trọng trong quy trình thanh toán của ứng dụng, cho phép người dùng cập nhật địa chỉ giao hàng và địa chỉ thanh toán của họ một cách linh hoạt. Điều này giúp cải thiện trải nghiệm người dùng và giảm thiểu lỗi trong quá trình xử lý đơn hàng.