# SMART SHOPPING ASSISTANT USING MOBILE APPLICATION DEVELOPMENT

By

Lok Wai Loon

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION TECHNOLOGY (HONOURS)

COMMUNICATIONS AND NETWORKING

Faculty of Information and Communication Technology
(Kampar Campus)

JUNE 2025

# ACKNOWLEDGEMENTS

I would like to express my sincere and heartfelt gratitude to my supervisor, Dr. Farina Saffa binti Mohamad Samsamnun, for her invaluable guidance, support, and encouragement throughout the course of this project. Her insightful advice, constructive feedback, and continuous motivation have been instrumental in the successful completion of my Final Year Project.

I would also like to extend my deepest appreciation to my family, especially my parents, for their unconditional love, unwavering support, and encouragement throughout my academic journey. Their belief in me has been a constant source of strength and inspiration.

Lastly, I would like to thank all my friends and peers who have supported and motivated me during this period. Your encouragement and assistance have been greatly appreciated. Thank you all.

# COPYRIGHT STATEMENT



This Final Year Project proposal is submitted in partial fulfillment of the requirements for the degree of Bachelor of Information Technology (Honours) Communications and Networking at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project proposal represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project proposal may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

# ABSTRACT

In the contemporary retail landscape, mobile applications have become indispensable tools for grocery shoppers. However, many existing platforms fail to offer a truly integrated and intelligent experience, lacking advanced features for personalized health-conscious decision-making, effortless list management, and practical in-store support. This project introduces the Smart Shopping Assistant, a cross-platform mobile application designed to address these limitations by creating a seamless, secure, and highly personalized shopping journey. The application is developed using the Flutter framework for broad device compatibility and is powered by Supabase for robust and secure backend services, including user authentication, data management, and PostGIS-based geospatial queries. At its core, the assistant leverages a sophisticated AI engine, orchestrated through Dify and utilizing the Gemini API, to provide users with personalized nutritional analysis based on their specific health profiles. A key innovation is the on-device integration of Optical Character Recognition (OCR) via Google's ML Kit, which empowers users to instantly scan and digitize product nutrition tables for real-time evaluation. Furthermore, the inclusion of the Google Maps Service provides comprehensive location-based features, including nearby store discovery, address selection with autocompletion, and route visualization. Developed under an Agile methodology, the project prioritizes a user-centric design that blends convenience with valuable insights. By unifying an intelligent AI assistant, real-time nutritional data analysis via OCR, a full-featured e-commerce system with order management, and practical location-based services within a single platform, the Smart Shopping Assistant aims to empower consumers to make more informed, efficient, and health-aware purchasing decisions. This work contributes to the evolution of mobile commerce by delivering an integrated solution that enhances user trust, convenience, and dietary consciousness in everyday shopping.

Area of Study (Maximum 2): Mobile Application Development, Artificial Intelligence

Keywords (Maximum 5): Smart Shopping Assistant, Artificial Intelligence, Secure Authentication, Nutritional Analysis, Location-Based Service (LBS)

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1

# Project Background

In this chapter, we discuss the background and rationale for our research, highlight our contributions to the field, and provide an overview of the thesis structure.

## 1.1 Introduction

The rapid evolution of mobile technology has significantly transformed the retail landscape, particularly in how consumers engage with digital platforms for daily shopping activities. Mobile shopping applications have become essential tools, enhancing user convenience and enabling efficient browsing and purchasing experiences [1].

However, many mobile applications still face significant challenges. Many Android applications contain well-known vulnerabilities, posing serious security concerns for the entire Android application landscape [2]. These issues expose users to risks such as data breaches and unauthorized access, threatening both personal information and the credibility of mobile platforms. Mobile application security requires awareness among developers, app market administrators, and users to ensure that devices and data are protected from emerging threats and vulnerabilities [3].

This project proposes the development of a smart shopping assistant mobile application that integrates security, usability, and intelligent support. It seeks to address existing weaknesses by combining a secure, trust-centric architecture with an AI-powered nutrition analysis engine and integrated Location-Based Services (LBS) within a unified, user-friendly platform.

## 1.2 Problem Statement

While there are many mobile shopping apps available, most need help to effectively address the core challenges that they encounter in their daily lives. These challenges include:

### 1.2.1 Security and Privacy Concerns

As mobile shopping becomes increasingly prevalent, users are entrusting applications with sensitive personal and financial information. However, many existing shopping apps lack robust security measures, such as multi-factor authentication and secure data encryption, leaving users vulnerable to unauthorized access and data breaches [4]. This lack of security not only compromises user trust but also poses significant risks to personal data integrity.

### 1.2.2 Absence of Intelligent Shopping Assistance

While basic browsing and checkout are common, many apps do not provide AI-driven assistance that adapts to individual preferences, health profiles, and budgets. The lack of personalized recommendations, healthier or cheaper alternatives, and real-time, context-aware guidance limits users' ability to make informed purchasing decisions efficiently [5][6].

### 1.2.3 Poor User Experience and Inefficient Design

Many contemporary mobile grocery applications, despite their functional capabilities, are undermined by a poor user experience (UX) that creates significant friction and inefficiency. The core purpose of these applications is to simplify and expedite the chore of grocery shopping; however, users are often confronted with unintuitive interfaces, convoluted navigation, and a fragmented process flow. This results in a frustrating and time-consuming experience that fails to meet the modern consumer's expectation for speed and convenience. [7]

## 1.3 Motivation

The widespread adoption of mobile technology has revolutionized the retail sector, making mobile applications essential for daily grocery shopping. While these apps offer convenience, a significant gap remains between their current functionalities and the evolving needs of modern consumers, particularly in the areas of personalized health guidance, data security, and seamless user experience. This project is motivated by the opportunity to bridge this gap.

A primary motivation is the growing concern over the security of sensitive personal information. As users share more data with applications, including highly personal health profiles required for tailored advice, the need for robust data protection becomes paramount. The motivation is to create a secure platform where users feel safe entrusting their data, which is a prerequisite for building the trust needed for a personalized health and wellness tool.

The core motivation for this project stems from the increasing consumer demand for healthier lifestyles. Shoppers often struggle to make informed dietary decisions in a fast-paced retail environment. Manually reading and interpreting complex nutrition labels for every product is impractical, especially for individuals with specific health conditions like allergies or diabetes. This project is driven by the goal of empowering these users with an intelligent tool. By integrating Artificial Intelligence (AI) with Optical Character Recognition (OCR), the application can provide instant, personalized nutritional feedback, transforming the complex task of healthy shopping into an effortless and insightful experience.

Finally, this project is motivated by the need to reduce the friction and fragmentation common in the mobile shopping trips. Many applications offer a disjointed experience for discovering stores, managing delivery addresses, and completing a purchase. There is a clear need for a more integrated solution. By incorporating Location-Based Services (LBS), this application is designed to create a streamlined workflow—from locating a nearby store to a seamless checkout—thereby enhancing overall user convenience and satisfaction.

In summary, this project is to create a holistic mobile shopping application that is not only secure and convenient but also serves as a proactive partner in the user's health and wellness journey.

## 1.4 Project Objectives

The primary objective of this project is to design, develop, and evaluate an intelligent mobile shopping assistant that enhances the grocery shopping experience by integrating personalized, health-conscious decision-making tools and location-based convenience features, all within a secure and trustworthy environment. The specific objectives are as follows:

### 1.4.1 To establish a security and privacy-centric architecture for handling sensitive user data.

This objective prioritizes the protection of user information, which includes personal health profiles, order history, and saved addresses. Building on Supabase authentication and authorization, the system will implement multi-factor authentication (MFA), row-level security (RLS), encryption in transit (TLS) and at rest, least-privilege access, audit logging, and privacy-by-design data minimization to ensure robust authentication flows and compliant data stewardship, thereby strengthening user trust in the platform.

### 1.4.2 To create an artificial intelligence–driven decision support component that leverages OCR-parsed nutrition data and user context to deliver personalized recommendations during shopping.

The application will employ on-device optical character recognition (OCR) to digitize nutrition labels and integrate the extracted information with the user's health profile, preferences, and budget. An AI engine will provide real-time, personalized outputs including nutritional analyses, health-suitability scores, allergen alerts, suggestions for healthier or more economical substitutes, and individualized product recommendations with context-aware guidance—thereby empowering users to make informed choices at the point of purchase.

### 1.4.3 To design and implement a user-friendly interface with integrated Location-Based Services (LBS) to streamline the shopping process.

This objective focuses on creating an intuitive and seamless user experience. The application will incorporate advanced LBS to automatically detect the user's location, suggest nearby stores, and provide a map-based interface for selecting delivery addresses, thereby reducing navigation depth and interaction steps and enhancing overall usability and convenience for the shopper.

By achieving these objectives, the project will deliver a holistic mobile application that is not only intelligent and user-friendly but also fundamentally secure, addressing key consumer needs in the modern digital retail environment.

## 1.5 Project Scope and Direction

The scope of this project is focused on the design, development, and implementation of a smart shopping assistant mobile application built with the Flutter framework. The application enhances the grocery shopping experience by integrating three core pillars: a secure architecture for user data, an intelligent AI-powered engine for health-conscious decision-making, and location-based services for enhanced convenience. The project encompasses the entire development lifecycle, from backend setup and API integration to the creation of a polished, user-facing mobile interface.

A foundational component within the project's scope is the secure handling of sensitive user information. This goes beyond a standard login system by establishing a trust-centric architecture to protect a user's entire data footprint, including their confidential health profile, saved delivery addresses, and complete order history. The scope includes implementing robust user registration and authentication workflows leveraging Supabase's built-in security features. All data transactions will be managed through secure practices to ensure that the highly personal information required for the app's intelligent features is protected against unauthorized access, thereby building essential user trust.

The central and most innovative feature is the development of an AI-powered Nutrition Assistant. The scope for this module is comprehensive, beginning with the integration of an on-device Optical Character Recognition (OCR) module using Google's ML Kit to capture text from product nutrition labels accurately. It also includes creating a detailed, user-managed health profile for inputting dietary restrictions and health conditions. The core of this feature is the development of an AI service, orchestrated via Dify.ai, that processes this combined data. The service will deliver real-time, structured feedback, including nutritional breakdowns, health suitability scores, allergen alerts, and personalized dietary recommendations, moving beyond simple data display to offer actionable insights.

To ensure a seamless and modern user experience, the scope also encompasses the integration of comprehensive Location-Based Services (LBS). This involves

utilizing the device's GPS to identify the user's current location for context-aware suggestions. The project will implement advanced geospatial queries via Supabase PostGIS to efficiently find and display nearby grocery stores. Furthermore, it includes building an intuitive and interactive address management system, complete with a Google Maps interface, allowing users to effortlessly save, select, and visualize delivery locations, which significantly streamlines the checkout process and enhances overall usability.

The direction of this project is to deliver a practical and innovative mobile application that addresses the specific, real-world needs of modern, health-conscious consumers. By focusing the scope on the tightly integrated triad of security, AI-driven nutritional intelligence, and location-based convenience, the project aims to create a holistic and cohesive tool. The final outcome will be a functional prototype that empowers users to shop smarter, healthier, and more efficiently, setting it apart from conventional e-commerce platforms.

## 1.6 Contributions

This project contributes to the field of mobile application development by introducing an integrated solution that addresses critical gaps in the current smart shopping landscape. The primary contributions are centered on three core areas: the novel application of AI for health-conscious decision-making, the enhancement of user convenience through deep integration of location-based services, and the establishment of a secure architecture for handling sensitive personal data.

One of the most significant contributions is the development of a novel system for real-time, personalized nutritional analysis. Unlike conventional shopping apps, this project pioneers the use of on-device Optical Character Recognition (OCR) to instantly capture data from physical product labels. This data is then processed by an AI engine in the context of a user's specific health profile. This approach makes a substantial contribution by transforming the abstract concept of "healthy eating" into a practical, interactive, and immediate tool, empowering users with dietary conditions to make safe and informed decisions at the point of purchase.

Another major contribution is the enhancement of the mobile shopping experience through the seamless integration of Location-Based Services (LBS). The project moves beyond basic product listings by incorporating a sophisticated location-aware framework. This includes features such as discovering nearby stores, providing a rich map-based interface for managing delivery addresses, and visualizing routes. This deep integration contributes a more streamlined, context-aware, and frictionless user journey, directly addressing common usability pain points in existing e-commerce applications.

Finally, the project contributes a blueprint for a secure and trust-centric mobile health and e-commerce platform. By leveraging a modern backend service like Supabase, it implements a robust architecture for securing sensitive information, particularly the user's personal health data. This focus on security provides the necessary foundation of trust for users to engage with the application's advanced

personalization features, demonstrating a model where functionality and data privacy are co-equal priorities.

In conclusion, this project delivers an intelligent and user-centric mobile shopping platform that makes tangible contributions in the domains of applied AI, user experience design, and secure application architecture. It provides a strong foundation for future mobile solutions that aim to act as genuine assistants in a user's daily life and wellness journey.

# CHAPTER 2

# Literature Review

## 2.1 Review of Operating System

Android and iOS are the leading operating systems powering smartphones and tablets today. Both platforms provide unique features and specifications tailored to different user needs.

### 2.1.1 Android

Figure 2.1 Android Logo

Android, a Linux-based operating system, was initially developed by Android Inc. and later acquired by Google in 2005. While the core of Android, known as the Android Open-Source Project (AOSP), is open-sourced, certain components like Google Mobile Services (GMS) and the Google Play Store remain proprietary and under Google's control. This combination of open-source flexibility and proprietary services has made Android the most widely used smartphone platform globally. It powers devices from major manufacturers such as Samsung, Oppo, Vivo, Huawei, Xiaomi, and Honor, catering to a diverse range of users worldwide.

Initially, Android app development relied heavily on the Java programming language. However, since 2019, Google has endorsed Kotlin as a fully interoperable

alternative, which has become the preferred language for Android development. In addition to Kotlin, Android now supports a variety of programming languages, including JavaScript, Dart, and more. This flexibility allows developers to choose from a range of tools and frameworks, such as React Native for web-based applications and Flutter for cross-platform development, enabling them to create versatile and high-quality apps.

With approximately 3.9 billion users worldwide, Android has solidified its position as the leading mobile operating system. Its open-source nature, combined with flexible design guidelines and extensive customization options, provides developers with unparalleled freedom and adaptability. This has fostered a vibrant ecosystem of apps and innovations, making Android the operating system of choice for both developers and users. Furthermore, the streamlined process of publishing apps on the Google Play Store, Android's largest app marketplace, ensures rapid deployment and widespread accessibility.

Android's dominance in the mobile industry is a testament to its versatility and user-centric approach. By offering a balance of open-source innovation and proprietary services, Android continues to evolve, meeting the needs of a global audience. Its support for multiple programming languages and frameworks, coupled with its vast user base, ensures that it remains at the forefront of mobile technology, driving innovation and accessibility for years to come.

### 2.1.2 iOS


Figure 2.2 iOS Logo

iOS is a proprietary mobile operating system developed by Apple Inc. exclusively for its hardware, including the iPhone, iPad, and iPod Touch. First released in 2007 alongside the original iPhone, iOS revolutionized the smartphone industry with its intuitive touch-based interface and seamless integration with Apple's ecosystem. Unlike Android, iOS is a closed-source system, meaning its code is not publicly accessible, and it is tightly controlled by Apple. This approach ensures a high level of security, stability, and uniformity across all Apple devices.

iOS app development primarily relies on programming languages such as Swift and Objective-C. Swift, introduced by Apple in 2014, has become the preferred language due to its modern syntax, performance, and safety features. Developers use Apple's integrated development environment (IDE), Xcode, to create apps for iOS. The platform also supports cross-platform development frameworks like Flutter and React Native, enabling developers to build apps that work on both iOS and Android. However, iOS development is known for its strict adherence to Apple's design guidelines and rigorous app review process.

With over 1.5 billion active devices worldwide, iOS holds a significant share of the global mobile operating system market, particularly in regions like North America and Europe. Its closed ecosystem ensures consistent user experience, high-quality apps, and regular updates, which are automatically delivered to supported devices. The App Store, Apple's official marketplace, is home to millions of apps and is known for its stringent quality control, ensuring a secure and reliable experience for users. Publishing apps on the App Store requires adherence to Apple's guidelines, which can be more restrictive compared to Android but ensures a high standard of quality.

iOS is renowned for its seamless integration with other Apple products and services, such as macOS, watchOS, iCloud, and the Apple Watch. This ecosystem provides users with cohesive experience across devices, enhancing productivity and convenience. While iOS offers less customization compared to Android, its focus on privacy, security, and user experience has earned it a loyal customer base. As Apple continues to innovate with features like augmented reality (AR), machine learning, and enhanced privacy controls, iOS remains a leading platform in the mobile industry.

### 2.1.3 Android and iOS Comparison

Table 2.1 Comparative Analysis of Android and iOS Platforms

| Aspect | Android | iOS |
|---|---|---|
| **Developer** | Developed by Android Inc. (acquired by Google in 2005). | Developed by Apple Inc. |
| **Initial Release** | September 2008 (HTC Dream). | June 2007 (iPhone OS 1, with the first iPhone). |
| **Source Model** | Open source (AOSP), but Google services are proprietary. | Closed source (proprietary). |
| **Customization** | Highly customizable for users and manufacturers. | Limited customization; tightly controlled by Apple. |
| **App Development** | Supports Java, Kotlin, JavaScript, Dart, and more. | Primarily Swift and Objective-C. |
| **Development Tools** | Android Studio, with support for frameworks like Flutter and React Native. | Xcode, with support for Flutter and React Native. |
| **User Base** | ~3.9 billion active devices worldwide. | ~1.5 billion active devices worldwide. |
| **Device Compatibility** | Runs on devices from multiple manufacturers (Samsung, Xiaomi, Oppo, etc.). | Exclusively runs on Apple devices (iPhone, iPad, iPod Touch). |
| **Security** | Open-source nature can lead to vulnerabilities; relieve on manufacturers. | Tightly controlled ecosystem; known for strong security and privacy. |
| **Ecosystem Integration** | Limited integration with non-Google devices. | Seamless integration with Apple ecosystem (macOS, watchOS, iCloud). |
| **Price Range** | Devices available at all price points (budget to premium). | Primarily premium devices; higher price range. |
| **User Experience** | Flexible but can vary across devices. | Consistent and polished across all devices. |

Android and iOS are the two dominant mobile operating systems, each with distinct characteristics and strengths. Android, developed by Google, is an open-source platform based on the Linux kernel. It powers a wide range of devices from various manufacturers, such as Samsung, Xiaomi, and Oppo, offering extensive customization options for both users and developers. In contrast, iOS, developed by Apple, is a closed-source operating system exclusively designed for Apple devices like the iPhone, iPad, and iPod Touch. It is known for its seamless integration with the Apple ecosystem and consistent user experience.

In terms of app development, Android supports multiple programming languages, including Java, Kotlin, JavaScript, and Dart, providing developers with flexibility and a variety of frameworks like Flutter and React Native. iOS, on the other hand, primarily uses Swift and Objective-C, with development tools like Xcode. While Android's Google Play Store has a more open app submission process, iOS's App Store enforces strict guidelines, ensuring high-quality apps but with a more rigorous review process.

Android dominates the global market, particularly in regions like Asia, Africa, and South America, with around 3.9 billion active devices. It caters to a wide range of price points, from budget to premium devices. iOS, with approximately 1.5 billion active devices, has a strong presence in North America, Europe, and high-income markets, focusing on premium devices. Android's open-source nature allows for greater customization and support for custom ROMs, while iOS prioritizes security, privacy, and a controlled ecosystem.

Ultimately, the choice between Android and iOS depends on user preferences. Android appeals to those seeking flexibility, customization, and affordability, while iOS is ideal for users who value a polished, secure, and integrated ecosystem. Both platforms continue to evolve, driving innovation in the mobile industry.

## 2.2 Review of Framework

Flutter is an open-source UI software development kit (SDK) created by Google, designed to build cross-platform applications for mobile, web, and desktop from a single codebase. It allows developers to create high-performance, visually appealing apps that work seamlessly on multiple platforms, including Android, iOS, Windows, macOS, Linux, and web browsers. Flutter uses the Dart programming language, which is optimized for fast performance and ease of learning. One of its standout features is its widget-based architecture, where everything in the UI is a widget, enabling highly customizable and flexible designs. Additionally, Flutter's hot reload feature allows developers to see real-time changes in the app's UI without restarting the app, significantly speeding up the development and debugging process.

Flutter is known for its high performance, as apps are compiled into native machine code, ensuring smooth animations and fast execution. It provides a rich set of pre-designed widgets and tools, making it easy to create beautiful, natively compiled applications. Its cross-platform capabilities make it a versatile choice for developers, as they can maintain a single codebase for multiple platforms, reducing development time and costs. Flutter also has a growing community and ecosystem, offering a wide range of packages and plugins to extend functionality, such as APIs, databases, and third-party integrations. This makes it an ideal choice for startups and businesses looking to build MVPs (Minimum Viable Products) quickly and cost-effectively.

However, Flutter does have some limitations. Apps built with Flutter tend to have a larger file size compared to native apps, which may be a concern for some developers. Additionally, while Flutter's ecosystem is expanding, it may lack some platform-specific libraries available in native development. Developers also need to learn Dart, which may present a learning curve for those accustomed to other programming languages like Java or Swift. Despite these challenges, Flutter's advantages, such as faster development, consistent UI across platforms, and cost-effectiveness, make it a popular choice for building cross-platform applications.

Flutter is widely used for developing mobile apps for Android and iOS, web applications, and desktop applications. It is particularly well-suited for apps with complex, custom UIs, such as those requiring advanced animations and transitions. Some popular apps built with Flutter include Google Ads, Alibaba, BMW, eBay, and Reflectly. Overall, Flutter's focus on performance, flexibility, and ease of use has made it a powerful tool for developers and businesses looking to create high-quality, cross-platform applications efficiently.

## 2.3 Previous works

### 2.3.1 Hargapedia [8]

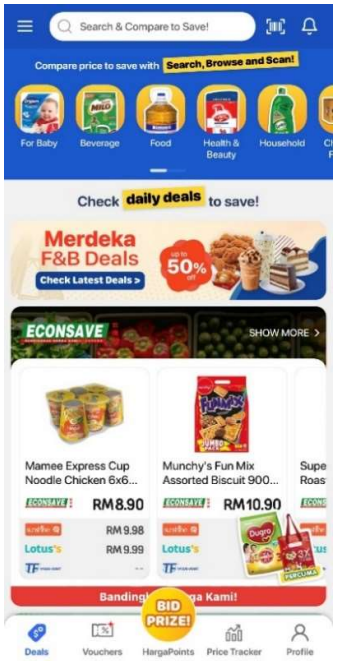



Figure 2.3 Hargapedia Deals and Search Page

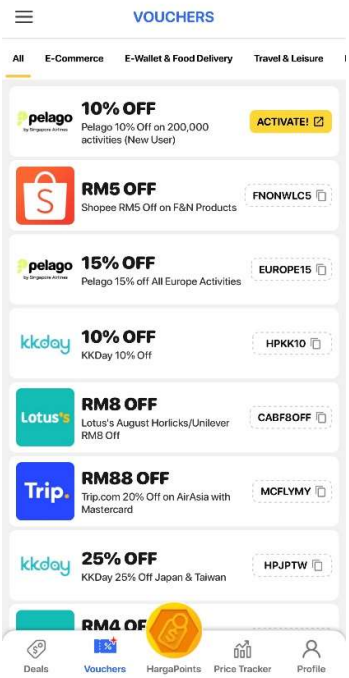



Figure 2.4 Hargapedia Voucher Page

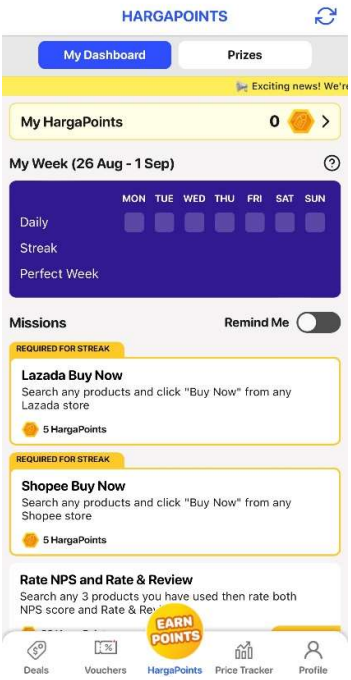



Figure 2.5 Hargapedia Harga Points Page

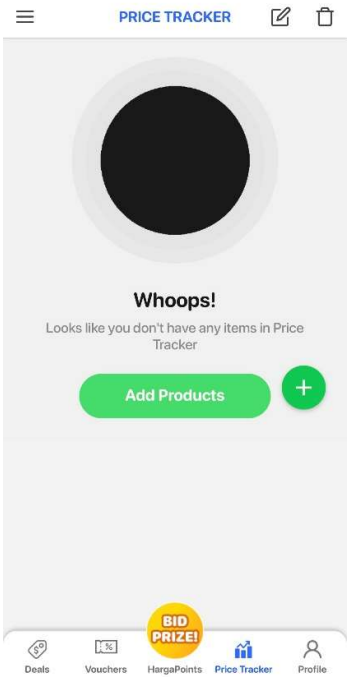



Figure 2.6 Hargapedia Price Tracker Page

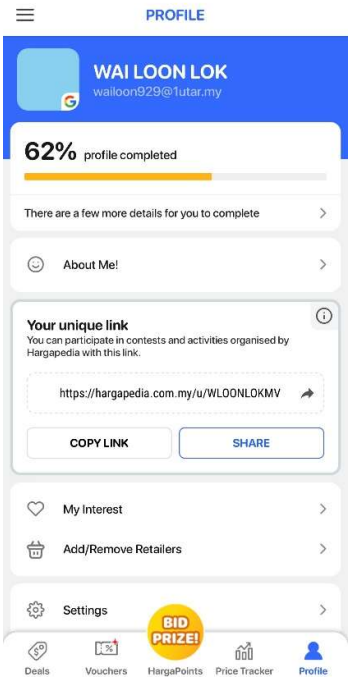



Figure 2.7 Hargapedia Profile Page

Hargapedia, which Ontrack Digital Sdn Bhd developed, is an application that helps users compare prices to locate the top deals on groceries, health, and beauty products in Malaysia. The app's comprehensive coverage and user-friendly interface have propelled it to a 3.9-star rating in the market. Hargapedia monitors and assesses prices for more than 40,000 products from 42 well-known retailers in Malaysia, encompassing physical stores and over 800 online shops. Users can make well-informed purchasing choices by comparing prices across various options. The application guarantees accuracy by updating prices daily, ensuring users receive the most up-to-date and trustworthy information, ultimately establishing a reputation for dependability.

Hargapedia is particularly appealing to cost-conscious consumers, and it includes daily discounts, special offers, and exclusive promotion codes in various categories. These features significantly enhance the app's attractiveness, offering users multiple opportunities to save money. The 'Price Tracker' feature also allows users to set alerts for specific products, notifying them when prices drop to a desired level. This proactive price monitoring enhances Hargapedia's utility as a shopping aid, enabling users to make intelligent purchases quickly and affordably.

Despite its strengths, Hargapedia faces some challenges that cap its overall usefulness. One notable limitation is the prevalence of ads within the app, which can lead to unintentional clicks and redirect users to websites, potentially disrupting the shopping experience. Moreover, the app specializes in a limited number of categories, focusing primarily on groceries and health and beauty products. While this focus makes Hargapedia highly effective in these areas, it does limit the app's utility for users interested in a broader range of products. The average user rating suggests room for improvement in functionality and product range, with feedback often urging enhancements to improve the user experience.

To overcome these constraints, Hargapedia could expand its offerings into additional categories such as electronics, home goods, and clothing, catering to a broader audience and enhancing its appeal. Addressing the app's speed issues and responding to feedback on user interface concerns could lead to a global UX solution that significantly improves the application experience for today's complex market needs.

### 2.3.2 **Foodpanda** [9]

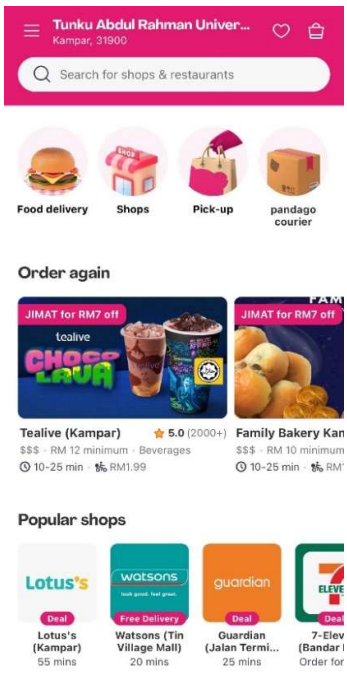



Figure 2.8 Foodpanda Main Page

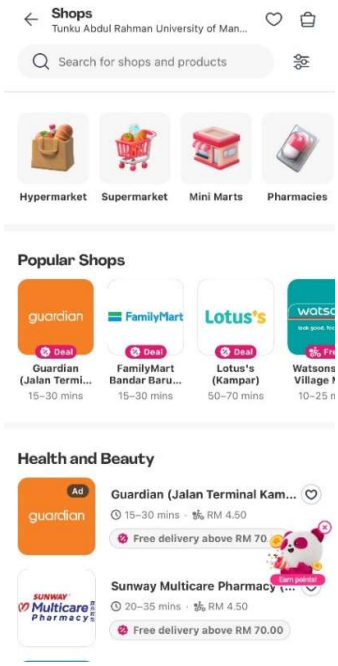



Figure 2.9 Foodpanda Shops Page


Figure 2.10 Foodpanda Search Product Page

Foodpanda is an app for delivering goods and grocery shopping that Delivery Hero SE developed. Foodpanda can be downloaded from both the Android Play Store and the Apple App Store, with millions of downloads and a 4.2-star rating, making it a user favorite for its attractive UI/UX design that simplifies finding desired food items. The ratings demonstrated robust user engagement and satisfaction, showcasing the app's effective positioning for convenient shopping.

Currently, the Foodpanda app has over 3,000 products from nearby supermarkets and specialty stores in its grocery section. The store has a wide range of products, from fresh produce to fruits and snacks and household essentials. This app, featuring various grocery shops, allows customers to add items with no limitations efficiently and is supported across Lotus's, Giant, Aeon, or other local stores for purchasing groceries from different categories.

Depending upon the user's location, faster delivery services that can reach the user within an hour or on the same day make grocery shopping even more convenient for Foodpanda subscribers. The app offers updates on order status in real-time, making shopping easy and affirmative. To entice others to save a small quantity of money, Foodpanda offers regular promos and discounts, bundle deals, and its "PandaPro" subscription service for things like free delivery or special pricing.

Furthermore, Foodpanda boasts an elegant user interface capable of switching between product categories and searching for specific items from various stores with prices side by side. It provides users with accurate and up-to-date information to help them make careful purchase decisions. Convenient features include saving favorite items and easy reorder options for frequent shoppers. Foodpanda also benefits from various payment methods, including credit and debit cards, digital wallets, and cash on delivery to match user needs.

Nevertheless, there are still aspects in which the grocery delivery service offered by Foodpanda could be enhanced. Foodpanda has a restriction where customers can only compare prices of products at nearby stores, which might not guarantee the best deals. Moreover, Foodpanda might be more expensive than physical shops because of service fees on the platform. This pricing tactic could lead to customers spending more than they usually would at the store. Despite these challenges, Foodpanda's grocery store marks a substantial improvement in online grocery shopping, providing a diverse selection at competitive rates and quick and convenient delivery, making it a self-sustaining choice for shopping.

### 2.3.3 **Instacart** [10]

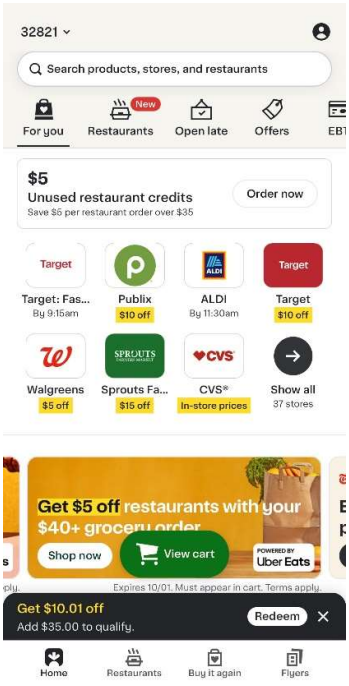



Figure 2.11 Instacart Home Page

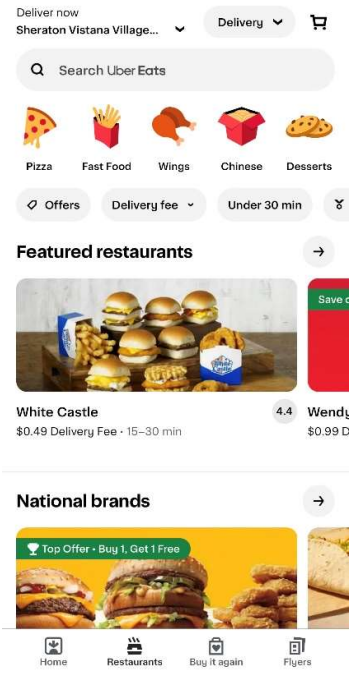



Figure 2.12 Instacart Restaurants Page

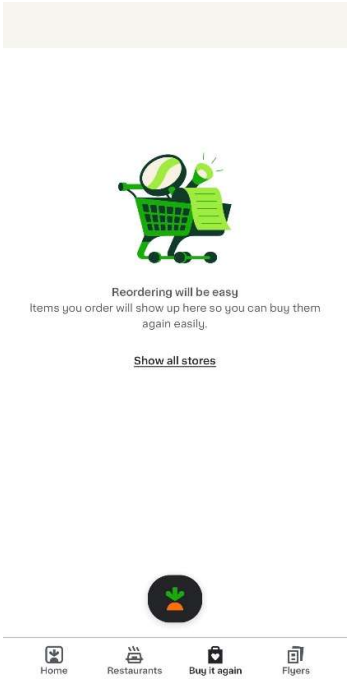



Figure 2.13 Instacart "Buy It Again" Page


Figure 2.14 Instacart Flyers Page

Maplebear Inc. developed Instacart, a widely used grocery delivery and pick-up service in the United States and Canada. Grocery shopping becomes easy and handy with this ultra-modern online grocer, which offers its users a wide range of services. Having received millions of downloads and boasting a 4.8 out of 5 stars rating on the iOS App Store, as well as from the Google Play Store, Instacart has established itself as an overall great way to go around grocery shopping due to its offer of seamless experience with ease, giving everything that customers need in one place. The service's intense customer satisfaction and endearment reflect well in its superior ratings.

Instacart has a broad range of partner stores, including big national chains Costco, Safeway, Kroger, and Publix, as well as numerous local supermarkets. Clients can purchase products from various merchants simultaneously in a single cart and through a single checkout. This implies that clients can typically find all they need in a single place, without the necessity to use multiple shopping websites.

The application allows users to browse products, use filters, and compare prices from various retailers. Instacart offers customized and suggestions based on buying history, helping customers discover new items that suit their preferences and enhance their shopping experience. Another critical aspect that draws people to Instacart is its flexibility in delivery choices. Customers can choose same-day delivery or schedule deliveries when they anticipate being home, with specific orders arriving in under an hour. The flexibility of time slots is beneficial for urgent and scheduled deliveries such as groceries, and the app allows live order tracking and direct communication with personal shoppers. This functionality lets users change their orders or include notes, reducing order errors and improving customer satisfaction.

Even though Instacart has many benefits, there are constraints that may affect user satisfaction. The higher total cost is one of the biggest drawbacks. Delivery charges and service fees can make the small orders placed via Instacart expensive in relation to shopping in the store, discouraging thrifty users. Additionally, with more features in the app, user-friendliness suffers. The interface can be confusing, and users often have to resort to trial and error in order to discover the functions they need. Some features even lead users to rapidly switch between different pages, which adds to confusion.

One obvious way for Instacart to address these challenges is by increasing its price transparency and disclosing any extra fees or markups up front. It would have helped manage user expectations and potential upset about costs. Better inventory management and working closely with store inventory systems can help prevent out-of-stock products and reduce unwanted substitutes. Providing incentives for the most consistent users, like loyalty discounts or reduced fees, also encourages continued use of the app and reduces some variance in overall costs.

## 2.4 Comparison between previous works and proposed works

Table 2.2 Comparison between previous works and proposed works

| Criteria | Hargapedia | Instacart | Foodpanda | Proposed Work |
|---|---|---|---|---|
| **Design of UI and UX** | Unattractive design, cluttered with ads. | Complex interface can be confusing for new users. | Attractive and modern design. | Simple, user-friendly, and visually appealing design. |
| **Secure Sign-Up Verification** | No verification required. | Basic verification without email confirmation. | Verification required with email notice. | Verification required with email confirmation notice via Supabase. |
| **AI-Powered Personalization** | Not available. | Basic for helpdesk prefix question only. | Not available. | Advanced (AI nutritional analysis based on user's personal health profile). |
| **Optical Character Recognition (OCR)** | Not available. | Not available. | Not available. | Yes, for instantly scanning and analyzing product nutrition labels. |
| **Location-Based Services (LBS)** | Limited to showing store locations for price comparison. | Yes, core feature for delivery logistics. | Yes, core feature for delivery logistics. | Yes, integrated for nearby store discovery and interactive address management. |
| **Key Differentiating Feature** | Multi-store price comparison and alerts. | Delivery from a wide range of national and local retailers. | Fast delivery and integrated food/grocery service. | Personalized, health-conscious shopping decisions via AI and OCR. |

The subsequent sections of this document undertake a systematic evaluation of three prominent grocery shopping and delivery platforms: **Hargapedia, Instacart, and Foodpanda**. This analysis moves beyond a review of standard industry functionalities to assess each platform against a set of specific criteria critical to the modern, health-conscious consumer. These criteria include user experience, security architecture, and, most importantly, the integration of intelligent tools for personalized nutritional decision-making.

By scrutinizing the strengths and weaknesses of these existing solutions, the primary purpose of this evaluation is to identify the functional gaps and opportunities for innovation within the current market landscape. These comparative insights are vital as they directly inform the design and justify the development of the proposed system. The findings from this analysis are encapsulated in Table 2.2, which juxtaposes the capabilities of each platform against the innovative features of the proposed work.

# CHAPTER 3

# Proposed Method/Approach

## 3.1 Software Development Life Cycle


Figure 3.1 The 7 Phases of the Software Development Life Cycle (SDLC)

SDLC stands for Software Development Life Cycle. It is a structured process used by software development teams to design, develop, test, and deploy high-quality software. The SDLC provides a framework for planning, creating, and maintaining software systems, ensuring that the final product meets user requirements and is delivered on time and within budget. The process is divided into distinct phases, each with specific goals and deliverables, to ensure systematic and efficient software development.

**Planning:** This is the initial phase that forms the foundation of the entire project. It's where the objectives, scope, and feasibility of the project are well defined. Planning activities such as cost planning, scheduling, and resource planning are done to form a clear roadmap. The output of this phase is the required documents like the project plan and the Software Requirements Specification (SRS), which serve as guideline documents during development.

**Requirements Analysis:** Following planning, this phase is focused on understanding the software needs in detail. It involves gathering and recording detailed requirements from stakeholders like users and business representatives. This task makes sure that the software will be capable of performing the required function by understanding user requirements, business requirements, and system capabilities. The principal output of this phase is a complete requirements document that contains all the specifications.

**Design:** Once the requirements have been defined, the design stage transforms them into a blueprint for the software. This is achieved by designing the system architecture, user interface, and database schema. The aim is to develop a comprehensive design, specification that will guide the development team on how to develop the software, such that the resulting product meets the defined requirements.

**Implementation (Coding):** This is the phase where the actual software development takes place. The programmers code based on the design specifications created in the earlier phase. This phase entails programming, building, and integrating various software components to create the functional software application.

**Testing:** After the software is developed, it undergoes rigorous testing to identify and fix any defects or bugs. This phase employs a number of testing methods, including unit testing, integration testing, and system testing, to verify that the software is in accordance with the specified requirements and functions correctly. The final goal is to deliver a stable and quality product.

**Deployment:** Once the software is completely tested and verified, it is made available to the end-users. Tasks involve installation, configuration, and user training to ensure a seamless handover and efficient implementation within the production environment. This procedure makes the software accessible and operational for its intended users.

**Maintenance:** The final stage is an ongoing process involving providing constant support, fixing any bugs that arise, and releasing updates and enhancements to the software. This stage is necessary to keep the software in a working, secure, and up-to-date condition, in accordance with evolving user requirements and technological developments over time.

## 3.2 Methodology Chosen - Scrum - Agile methodology

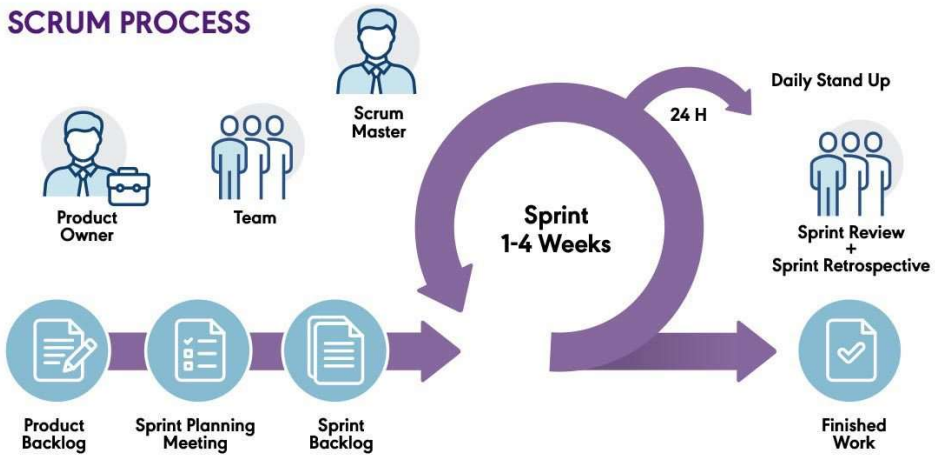



Figure 3.2 - Scrum Framework

Scrum is a principal application of agile frameworks mainly used in software development. It is more of a systematic way of pursuing product development, and its path is a cycle of iterations, teamwork, and flexibility. It also makes it possible to deliver value to customers in smaller bites. Since the entire project is divided into several small tasks, it is easier to manage and complete. The Scrum framework for developing a car rental mobile application means following several steps that correlate with the general Scrum development approach based on iterations and customer feedback. This leads to efficient project management with steady checks and balances to detect if there is lagging. Below are the stages for implementing Scrum in the development of a car rental mobile application:

**1. Product Backlog Phase:**

The first step is to list all the enhancements, bugs, and fixes that should be included in the Smart Shopping Assistant. It offers features such as a powerful search for the best prices on similar products, descriptions of the product's characteristics, and basic features of budgeting functions. With this input from potential users, stakeholders, and other forms of market research, one can determine which features must be developed and in what order due to their potential impact or lack of physical possibility of implementation. This prioritized list will be the basis and guide for development.

**2. Sprint Planning Phase:**

In the sprint planning, the team decides which features in the Product Backlog should be implemented in the following sprint. The most emphasis is placed on features that offer superior value propositions to the user and can be developed quickly. Other team members, including the development team, the product owner, and the Scrum master, will provide a forecast needed to perform a particular task, for instance, configuring the backend services for real-time price updates or designing the user interface of the shopping list feature.

**3. Daily Scrum Meeting Phase:**

It is important to maintain the pace of work and be able to respond to them before the problem grows to be unmanageable. This entails that in brief daily meetings, every team member explains what he or she has accomplished since the last meeting, what the member plans to do next, and if any activities could hinder his/ her work. It is common for these meetings to be crucial for the coordination of the team efforts and to guarantee that they report all possible issues that may have occurred in the sprint so that it gets back on track.

**4. Sprint Development Phase:**

This phase involves managing the activities and deliverables in line with the sprint backlog. This involves design and test processes intended to arrive at a potentially shippable product increment by the end of a sprint. For instance, developers would develop the price comparison aspect, while designers would be appointed to focus on the website's aesthetics and usability.

**5. Sprint Review Phase:**

This phase brings the end of each sprint and enables the team to showcase to the stakeholders the delivered work and have feedback. This is the time to show new features, discuss obstacles, and whether the sprint objectives were met. This feedback can then be used to reform the product, such as changing the user interface after receiving user feedback or changing the backend to increase efficiency.

**6. Product Backlog Refinement Phase:**

This phase concerns reviewing and reprioritizing the Product Backlog. At the end of the sprint, the product backlog is adjusted or modified depending on the user feedback from the sprint review and or new information arising from an ongoing market analysis. Some of the activities may be further decomposed, others may be moved up or down, or more or new activities may be included to fit changes in project boundaries or user requirements.

## 3.3 Software and Hardware

### 3.3.1 Hardware

Below are the hardware specifications for a laptop and a mobile device. The laptop is utilized to code the system, while the final testing will be conducted on the mobile device.

Table 3.1 Specifications of Laptop

| Description | Specifications |
|---|---|
| Model | HP EliteBook x360 1040 G8 |
| Processor | Intel Core i7-1165G7 |
| Operating System | Windows 10 Pro (Version 22H2) |
| Graphic | Intel® Iris® $X^e$ Graphics |
| Memory | 16GB LPDDR4 |
| Storage | 512TB SSD |

Table 3.2 Specifications of Mobile Emulator

| Description | Specifications |
|---|---|
| Model | Pixel 2 |
| Operating System | Android 13 (Tiramisu) |
| Memory | 2GB RAM |
| Storage | 32GB |

### 3.3.2 Software

To develop a Smart Shopping Assistant mobile application using Figma, Flutter, and Firebase, the entire process from design to deployment can be streamlined efficiently.

### 3.3.3 **Figma** [11]

In the initial design phase, Figma is utilized to architect the application's user interface (UI) and user experience (UX). This collaborative design tool is employed to create wireframes, high-fidelity mockups, and interactive prototypes. This process establishes the visual blueprint for the application, ensuring that the final product has an intuitive, user-friendly, and aesthetically pleasing interface that simplifies complex features like the AI nutrition chat and the location-based store finder.

### 3.3.4 **Flutter** [12]

The application is developed using Flutter, Google's cross-platform UI toolkit. This allows for the creation of a single codebase in the Dart programming language that compiles natively for both Android and iOS platforms, ensuring a consistent user experience across devices. Flutter's extensive library of customizable widgets is used to faithfully implement the Figma designs and build a responsive and high-performance interface for all application features, from product lists to the interactive map.

### 3.3.5 **Supabase** [13]

Supabase serves as the primary backend-as-a-service (BaaS) platform for the project, replacing traditional server-side development. Its role is critical, as it handles all user registration and login processes, providing a secure and reliable authentication system. Furthermore, its powerful PostgreSQL database is used for persisting all application data, including user profiles, health information, product inventory, and order histories. A key feature of Supabase utilized in this project is its support for PostGIS extensions, which enables powerful and efficient geospatial queries to find nearby stores based on user coordinates.

### 3.3.6 **Dify.ai** [14]

Dify.ai is the AI application development platform used to orchestrate the intelligent features of the nutrition assistant. It provides the crucial middleware that connects the mobile application to the underlying Large Language Model (LLM), Gemini. Dify is used to design, manage, and deploy the AI agent's logic, including the prompts that guide the analysis of OCR-extracted text and user health profiles, ensuring a structured and reliable response.

### 3.3.7 **Google ML Kit** [15] **& Google Maps Platform** [16]

Google ML Kit is used specifically for its on-device Optical Character Recognition (OCR) capabilities, which enable the application to quickly and accurately extract text from images of product nutrition labels directly on the user's device. Complementing this, a suite of APIs from the Google Maps Platform is used to provide rich geographical features, including converting addresses to coordinates (geocoding), address autocompletion (Places API), and planning routes.

By integrating this comprehensive set of software, the Smart Shopping Assistant is developed as a robust, scalable, and intelligent application that effectively bridges the gap between design, functionality, and advanced AI-powered features.

## 3.4 Project Milestone

| Project Task | FYP 1 | | | | | | | | | | | | | | FYP 2 | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 |
| **Planning** | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Form a project title | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Research | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **Analysis** | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Identify project background | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Identify problem statement and motivation | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Literature Review | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Compare the Current practice | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Identify Project Scope and Objectives | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Identify Contributions and Project Organization | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **Design** | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Identify the methodologies and user fucntion | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| illusulate system design diagrams | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Identify implementation issue and challenges | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **Implementation** | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Develop the system and create a prototype | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Prototype improvement | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| System Testing | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| System Deployment | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Documentation | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Figure 3.3 Gantt Chart of Project Milestones for Smart Shopping Application

The project timeline, illustrated in the Gantt chart, outlines the workflow for a two-phase development cycle comprising FYP1 and FYP2, with each phase broken down into weekly intervals. The chart segments the project into five main categories: Planning, Analysis, Design, Implementation, and System Delivery, each containing specific tasks allocated over distinct time periods.

- **Planning**: Early in FYP1, formation a project title and initial research. These foundational tasks are crucial for setting the direction and scope of the work ahead.

- **Analysis**: A substantial portion of FYP1 is to analytical tasks such as identifying the project background, determining problem statement and motivation, conducting a literature review, and comparing current practices.

- **Design**: Midway through FYP1, the focus shifts to system design. This entails determination of methodologies and user functions, creation of holistic system design diagrams, and fixing possible implementation problems. These are the pre-requisites for a smooth development stage.

- **Implementation**: Implementation embraces both FYP1 and FYP2. It begins at the later end of FYP1 with a prototype system design and intensifies in FYP2, where the prototype gets improved. Implementation phase allows system development through iterative improvement and iteration based on the feedback from the testing.

- **System Delivery**: In FYP2, once significant development milestones are achieved, the system is tested intensively to confirm functionality and performance. After validation, it goes on to system deployment. Concurrently, documentation work continues, with all areas of the system being fully documented from design decisions to ultimate implementation for future use or user guidance.

This structured timeline demonstrates a methodical approach, where FYP1 is primarily focused on groundwork which is comprising planning, analysis, and design, while FYP2 emphasizes execution through development, testing, deployment, and documentation.

## 3.5 Estimated Cost

The project incurred no direct monetary cost during development and evaluation, as all tools and services operated within free or open-source tiers. Development used the Flutter SDK, Android Studio, and Visual Studio Code, while testing was performed on a personal laptop with an Android emulator and a personal Android phone, avoiding additional hardware purchases. The backend ran on the Supabase Free plan (authentication, PostgreSQL, Storage, RPC/PostGIS, and email OTP). The AI-driven Nutrition Assistant used Dify's free tier; mapping and geocoding relied on Google Maps Platform within its monthly free credits with OpenRouteService as a free fallback; OCR was performed on-device (e.g., ML Kit/Tesseract); and monitoring utilized Flutter DevTools and the Supabase dashboard. Accordingly, the estimated total cost for this phase was **RM0**, with potential future costs only if usage exceeds free-tier quotas (e.g., LLM tokens, geocoding requests, or a Supabase plan upgrade).

# CHAPTER 4

# System Design

## 4.1 System Architecture Diagram

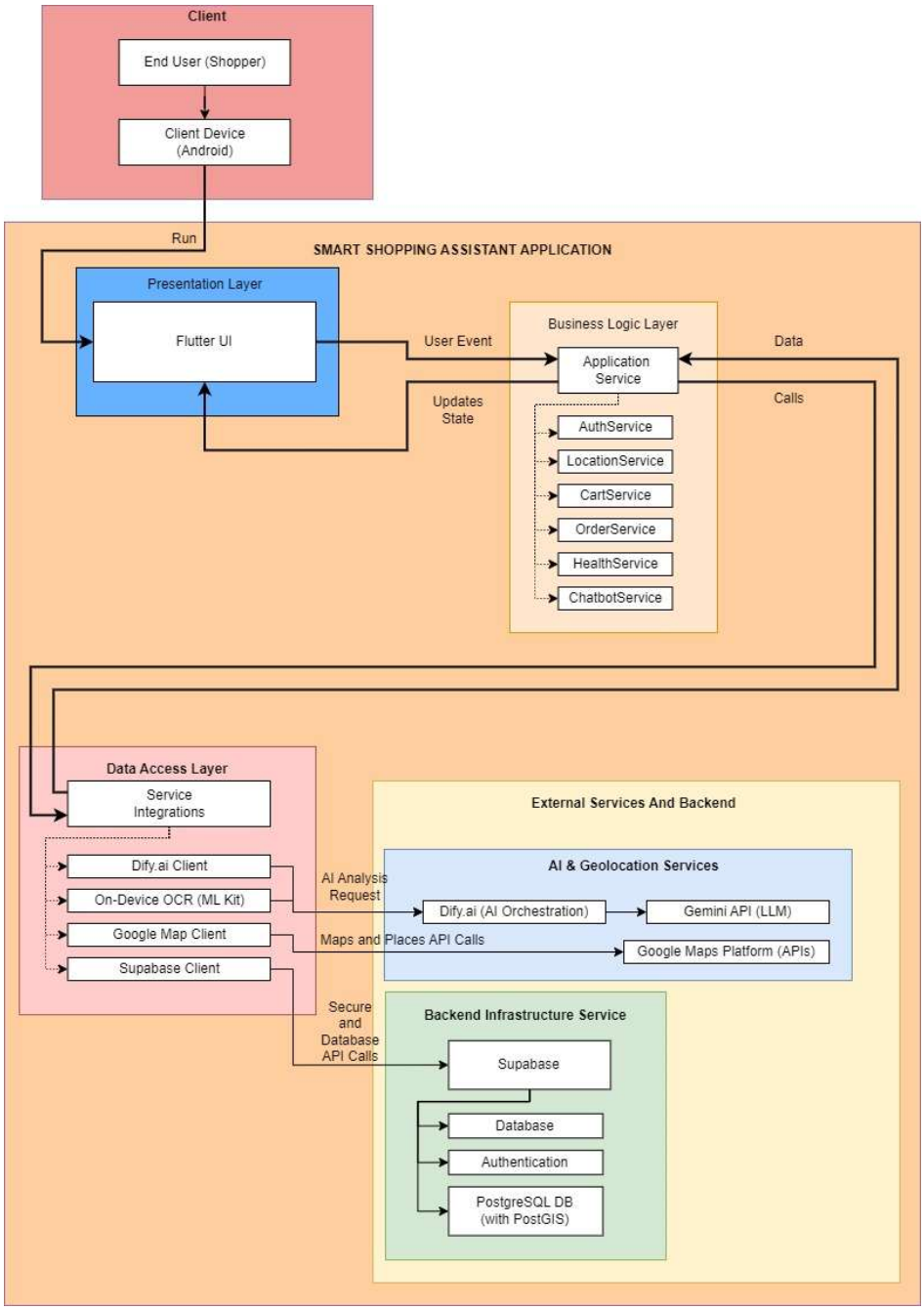



Figure 4.1 Smart Shopping Assistant Application System Architecture

The system architecture of the Smart Shopping Assistant application is predicated on a modern, multi-layered paradigm, as depicted in the provided diagram. This design enforces a strict separation between the client-side application and the server-side backend services, a methodology that ensures modularity, enhances scalability, and simplifies maintenance. The architecture is logically partitioned into three principal domains: the Client, the Smart Shopping Assistant Application, and the External Services and Backend.

The Client domain constitutes the initiation point for all system interactions. It is composed of the primary actor, the End User (Shopper), and the physical medium through which they interface with the system, their Client Device (Android). The core application is executed entirely within this domain.

Residing on the client device is the Smart Shopping Assistant Application, a front-end system developed using the Flutter framework. It is internally structured according to a three-tier architectural pattern to facilitate a clear separation of concerns. The Presentation Layer is the top-most tier, responsible for rendering the graphical user interface (GUI) via Flutter widgets and capturing all user-generated events. Subjacent to this is the Business Logic Layer, which serves as the application's central processing unit. This layer contains a suite of services, such as AuthService, CartService, and HealthService, that are responsible for executing the application's core business rules and managing its state. The foundational tier is the Data Access Layer, which functions as an abstraction layer that decouples the business logic from the underlying data sources. It manages all data communication, containing clients for Supabase and Dify.ai, and also orchestrates on-device processes like Optical Character Recognition (OCR) via ML Kit.

The External Services and Backend domain comprises all server-side components that provide data persistence, computation, and specialized functionalities to the client application. The Backend Infrastructure Service is provided by Supabase, which functions as the primary backend provider, offering a secure Authentication service and a PostgreSQL Database with PostGIS for data persistence and complex geospatial operations. This is complemented by the AI & Geolocation Services, which

are specialized third-party platforms. Dify.ai serves as an AI orchestration engine managing the invocation of the Gemini Large Language Model (LLM) for nutritional analysis, while the Google Maps Platform provides a suite of Application Programming Interfaces (APIs) for advanced geolocation tasks.

The overall data flow within the architecture is systematic and unidirectional. A user event captured by the Presentation Layer initiates a request to the Business Logic Layer. This layer processes the request and delegates the task of data retrieval or submission to the Data Access Layer, which then communicates with the appropriate Backend or External Service via a secure API call. The resulting data or confirmation is propagated back through the layers, enabling the Business Logic Layer to update the application's state, which in turn triggers the Presentation Layer to render the updated information to the user.

## 4.2 Use Case Diagram

### 4.2.1 Use Case Diagram Design for Smart Shopping Assistant Application

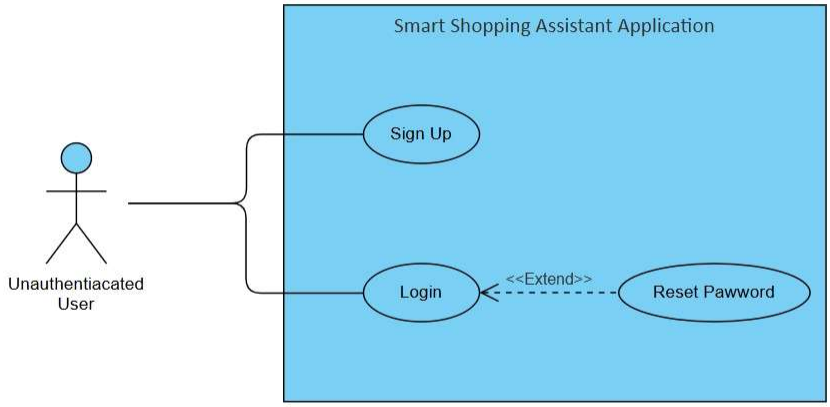



Figure 4.2 Use Case Diagram for Unauthenticated User

The Figure 4.2 shows the authentication process of the Smart Shopping Assistant Application, highlighting the actions available to an unauthenticated user. It illustrates how a new user can create an account through the Sign-Up process, or an existing user can access the Login page to become authenticated. In cases where a user has forgotten their password, the Login use case is extended by a Reset Password option, providing a secure account recovery method. This ordered authentication process serves as the secure entry point into the application's main features, which become accessible only after successful login.

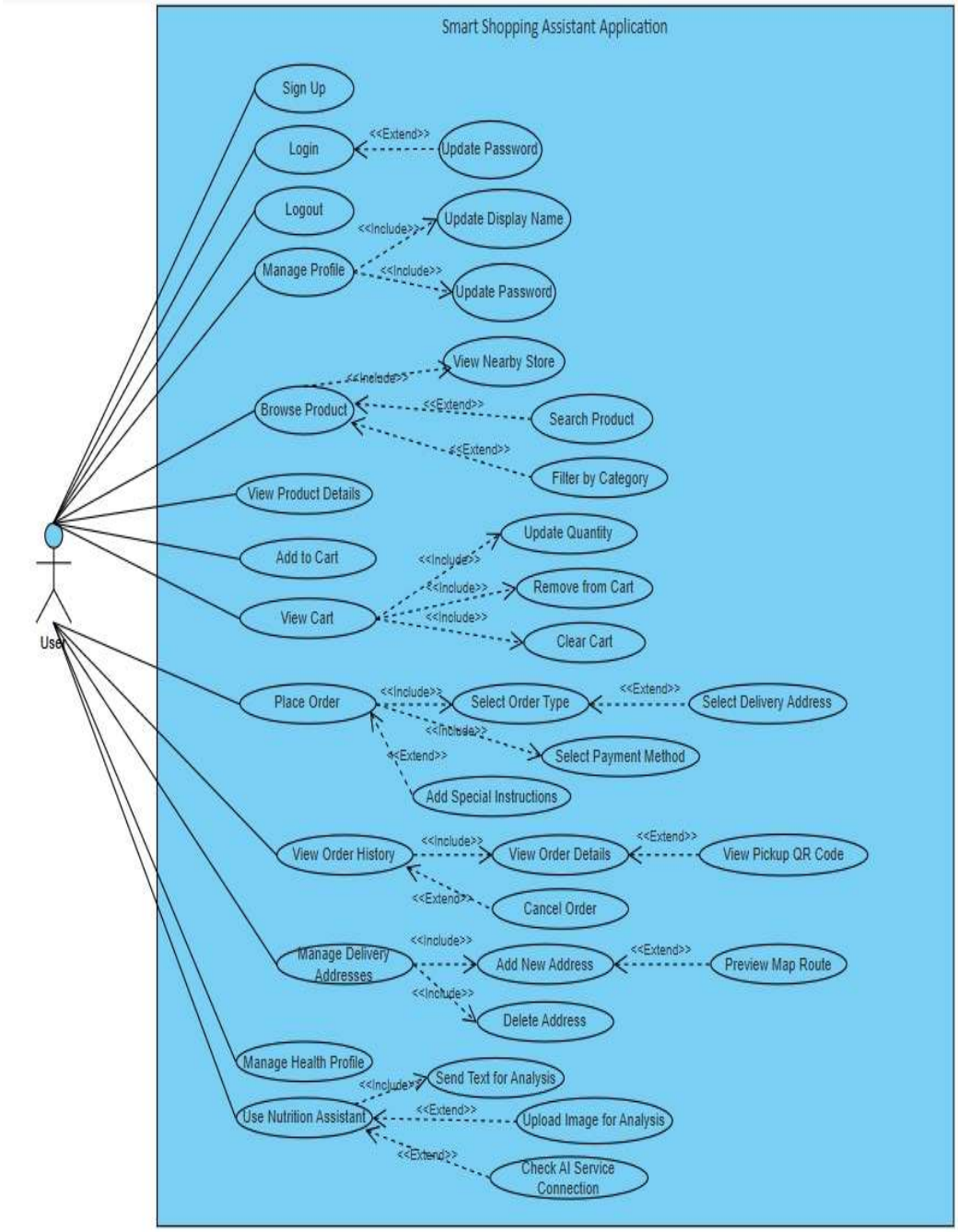



Figure 4.3 Use Case Diagram for Authenticated User

The diagram in Figure 4.3 provides a detailed feature decomposition of the Smart Shopping Assistant Application, outlining the functional capabilities available to an authenticated user. Upon successful authentication, the user is presented with a suite of integrated modules designed to create a seamless, intelligent, and personalized shopping experience.

The Account and Profile Management module serves as the foundation for user interaction. It facilitates secure access through Sign Up and Login functions. The Manage Profile use case empowers users to control their personal information, incorporating relationships to Update Display Name and Update Password, thereby ensuring account integrity and personalization.

Central to the application is the Product Discovery and Cart Management module. Users can Browse Products, a feature that includes viewing nearby stores and can be extended with Search Product and Filter by Category functionalities for a more refined search. The workflow continues with options to View Product Details and Add to Cart. The View Cart use case acts as a control center for selected items, including essential sub-functions to Update Quantity, Remove from Cart, and Clear Cart.

The Checkout and Order Management module streamlines the entire transaction process. The Place Order use case orchestrates the checkout, which includes mandatory steps like Select Order Type and Select Payment Method. This process can be optionally extended to Select Delivery Address or Add Special Instructions. Post-transaction, users can View Order History, which includes viewing Order Details and can be extended to View Pickup QR Code or Cancel Order under appropriate conditions.

Finally, the AI-Powered Nutrition Assistant represents a key innovative feature. The Use Nutrition Assistant use case provides personalized dietary advice. This includes a primary function to Send Text for Analysis and is extended with the advanced capability to Upload Image for Analysis for automated ingredient assessment and to Check AI Service Connection, ensuring the feature's reliability. These integrated modules collectively deliver a sophisticated and user-centric digital toolset for an intelligent and efficient shopping administration.

### 4.2.2 Use Case Description for Account Management

Table 4.1 Use Case Description for "Account Management"

| **Use Case ID** | UC 1 | **Use Case Name** | Account Management |
|---|---|---|---|
| **Actors** | | User | |
| **Purpose** | | To allow users to create, securely access, and maintain their personal account details. | |
| **Preconditions** | | The user has the application installed and has internet access. | |
| **Basic Flow:**<br>1. The user opens the app and chooses to "Sign Up" or "Login".<br>2. For new users, they fill out the registration form (name, email, password) to create an account.<br>3. Existing users enter their credentials to log in.<br>4. Once logged in, the user can navigate to their profile to view or update their information.<br>5. The user can modify their display name or change their password.<br>6. The user can log out of their account. | | | |
| **Postconditions** | | User account is created, accessed, or updated successfully. User session is started or ended. | |
| **Exceptions** | | • Invalid or missing input data during registration or login.<br>• Email is already in use during sign-up.<br>• Incorrect password.<br>• User attempts to reset a password for a non-existent email. | |

### 4.2.3 Use Case Description for Product Browsing & Search

Table 4.2 Use Case Description for "Product Browsing & Search"

| **Use Case ID** | UC 2 | **Use Case Name** | Product Browsing & Search |
|---|---|---|---|
| **Actors** | | User | |
| **Purpose** | | To enable users to discover available products, find nearby stores, and view detailed product information. | |
| **Preconditions** | | The user is logged into the application. | |
| **Basic Flow:**<br>1. The user navigates to the main products page.<br>2. The system displays a list of nearby stores based on the user's location.<br>3. The user browses the general list of products.<br>4. Optionally, the user enters keywords into the search bar to find specific products.<br>5. Optionally, the user selects a category to filter the product list.<br>6. The user selects a product to view its detailed information page. | | | |
| **Postconditions** | | The user has found and viewed information about one or more products. | |
| **Exceptions** | | • No products match the search query or filter criteria.<br>• The selected product is no longer available.<br>• Failed to fetch location to show nearby stores. | |

### 4.2.4 Use Case Description for Cart Management

Table 4.3 Use Case Description for "Cart Management"

| **Use Case ID** | UC 3 | **Use Case Name** | Cart Management |
|---|---|---|---|
| **Actors** | | User | |
| **Purpose** | | To allow users to select, review, and modify items they intend to purchase. | |
| **Preconditions** | | The user is logged in and is browsing products. | |
| **Basic Flow:**<br>1. The user adds a product to the shopping cart from the product list or detail page.<br>2. The user opens the cart to review the selected items, quantities, and total price.<br>3. The user modifies the quantity of an item in the cart.<br>4. The user removes an item they no longer wish to purchase from the cart.<br>5. Optionally, the user clears all items from the cart to start over. | | | |
| **Postconditions** | | The user's cart is updated with the desired items and quantities, ready for checkout. | |
| **Exceptions** | | • The user tries to add an item from a different store to a non-empty cart, and the system prompts for action.<br>• A product's stock runs out before the user can check out. | |

### 4.2.5 Use Case Description for Ordering and Checkout

Table 4.4 Use Case Description for "Ordering and Checkout"

| **Use Case ID** | UC 4 | **Use Case Name** | Ordering and Checkout |
|---|---|---|---|
| **Actors** | | User | |
| **Purpose** | | To enable users to finalize their purchase by creating a formal order for pickup or delivery. | |
| **Preconditions** | | The user is logged in and has one or more items in their shopping cart. | |
| **Basic Flow:**<br>1. The user proceeds to check out the cart.<br>2. The user selects an order type: "Delivery" or "Pickup".<br>3. If "Delivery" is chosen, the user selects a delivery address from their saved addresses.<br>4. The user chooses a payment method.<br>5. Optionally, the user adds special instructions for the order.<br>6. The user reviews the final order summary and confirms to place the order. | | | |
| **Postconditions** | | A new order is created in the system with a "Pending" or "Confirmed" status. The user's cart is cleared. | |
| **Exceptions** | | • A delivery address is not selected for a delivery order.<br>• The payment method is invalid or fails.<br>• An item in the cart becomes unavailable during the checkout process. | |

### 4.2.6 Use Case Description for Order Management

Table 4.5 Use Case Description for "Order Management"

| **Use Case ID** | UC 5 | **Use Case Name** | Order Management |
|---|---|---|---|
| **Actors** | | User | |
| **Purpose** | | To allow users to track the status of their current and past orders. | |
| **Preconditions** | | The user has previously placed at least one order. | |
| **Basic Flow:**<br>1. The user navigates to the "My Orders" section.<br>2. The system displays lists of active and historical orders.<br>3. The user selects an order to view its full details, including items, cost, and status.<br>4. If the order is a "Pickup" order, the user can view a QR code for collection.<br>5. If the order status allows, the user can choose to cancel the order. | | | |
| **Postconditions** | | The user is informed about the status and details of their orders. An order may be cancelled if conditions permit. | |
| **Exceptions** | | • The user attempts to cancel an order that is already being processed or completed.<br>• The QR code is not available for the selected order type. | |

### 4.2.7 Use Case Description for Address Management

Table 4.6 Use Case Description for "Address Management"

| **Use Case ID** | UC 6 | **Use Case Name** | Address Management |
|---|---|---|---|
| **Actors** | | User | |
| **Purpose** | | To allow users to save and manage multiple delivery addresses for a faster checkout experience. | |
| **Preconditions** | | The user is logged in. | |
| **Basic Flow:**<br>1. The user accesses the "Manage Addresses" section from their profile or the checkout page.<br>2. The user views a list of their saved addresses.<br>3. The user chooses to add a new address via current location, map selection, or manual input.<br>4. When adding via map, the user can preview the route from the store.<br>5. The user selects an existing address to delete it. | | | |
| **Postconditions** | | The user's list of saved delivery addresses is updated. | |
| **Exceptions** | | • The address entered is invalid or cannot be geocoded.<br>• Location services are disabled when trying to use the current location. | |

### 4.2.8 Use Case Description for Health & Nutrition Analysis

Table 4.7 Use Case Description for "Health & Nutrition Analysis"

| **Use Case ID** | UC 7 | **Use Case Name** | Health & Nutrition Analysis |
|---|---|---|---|
| **Actors** | | User | |
| **Purpose** | | To provide users with AI-driven nutritional insights and advice based on their personal health data and product ingredients. | |
| **Preconditions** | | The user is logged in. For analysis features, the user's health profile must be completed. | |
| **Basic Flow:**<br>1. The user first navigates to their profile to fill out or update their health information (height, weight, allergies, etc.)<br>2. The user opens the "Nutrition Assistant" chat interface.<br>3. The user sends a text-based question about nutrition.<br>4. Alternatively, the user uploads a photo of a product's ingredient list for analysis.<br>5. The system processes the input and provides a detailed nutritional analysis and suitability report. | | | |
| **Postconditions** | | The user receives personalized nutritional advice based on their query and health profile. | |
| **Exceptions** | | • The user attempts to get an analysis without completing their health profile.<br>• The AI service is unavailable or fails to respond.<br>• The uploaded image is unclear, and the OCR fails to extract text | |

## 4.3 Activity Diagram

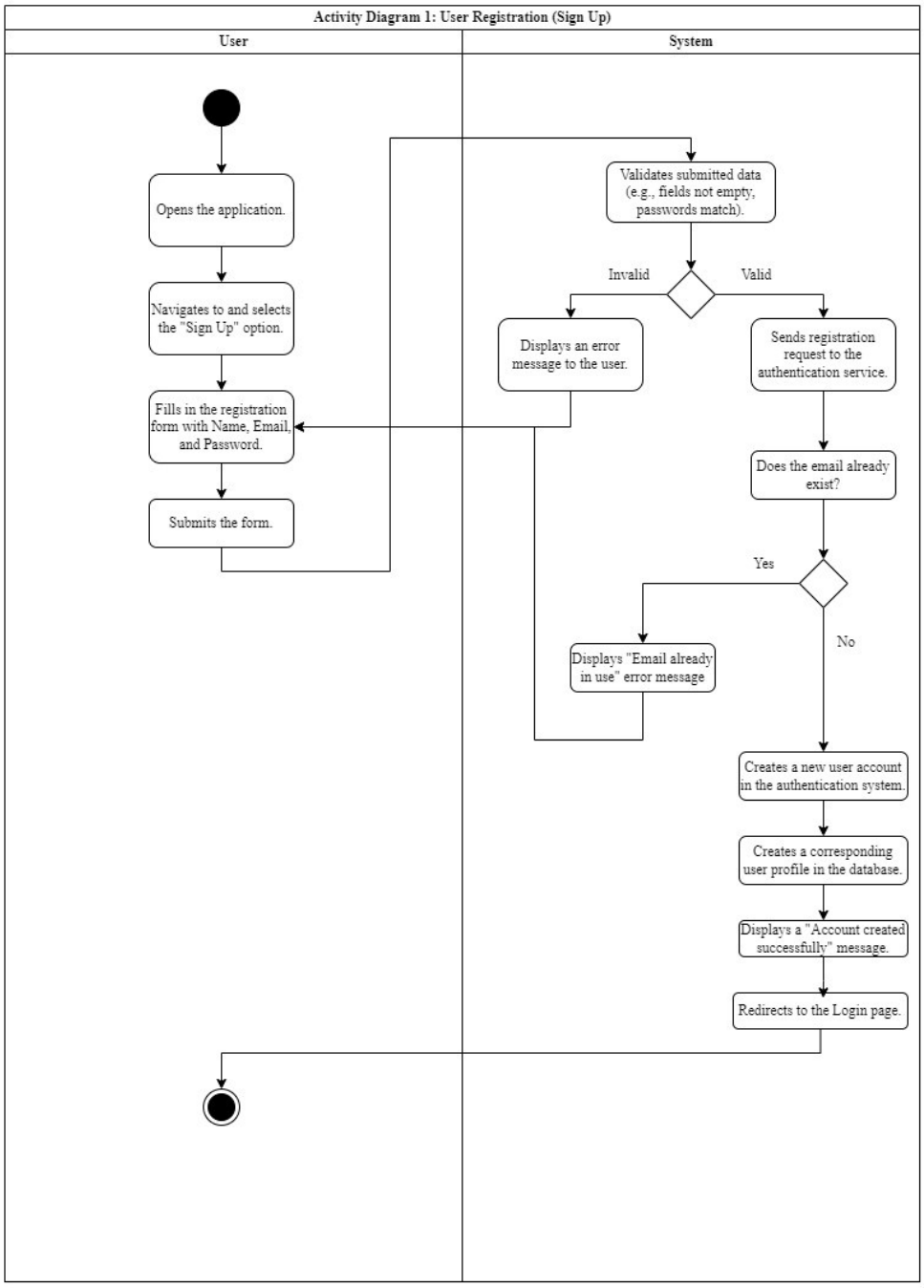



Figure 4.4 Activity diagram for User Registration (Sign Up)

The User Registration (Sign-Up) activity begins when the user opens the application, navigates to the Sign-Up option, completes the registration form with name, email, and password, and submits it. The system validates the submitted data to ensure that all required fields are provided and that the password satisfies policy requirements; if any validation fails, an error message is displayed, and the user is returned to the form to correct the input. When the submission is valid, the system forwards the request to the authentication service, which checks whether the email is already registered. If the email exists, the system presents an “email already in use” message and returns the user to the form. If the email is unique, the system creates a new account in the authentication service, generates a corresponding user profile in the application database, and confirms completion with an “account created successfully” message. The user is then redirected to the login page, at which point the registration activity concludes.

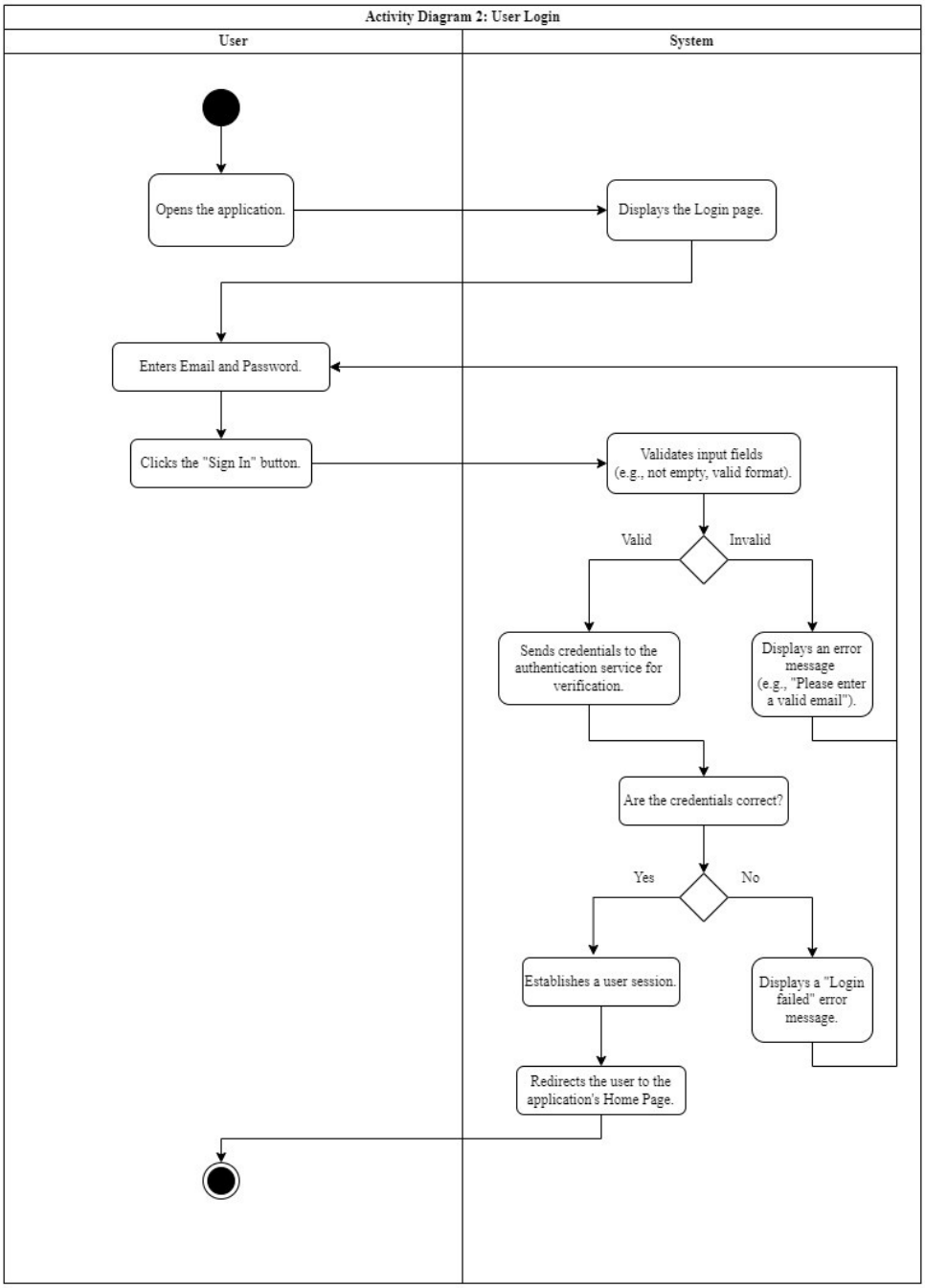



Figure 4.5 Activity diagram for User Login

The User Login activity begins when the user opens the application, and the system displays the login page. The user enters an email and password and selects Sign In. The system validates the inputs for completeness and correct format (e.g., a valid email); if validation fails, an error message is shown, and the user is returned to the form to correct the entries. When the inputs are valid, the system submits the credentials to the authentication service for verification. If the credentials are incorrect, the system displays a “Login failed” message and returns the user to the form. If the credentials are correct, the system establishes an authenticated user session and redirects the user to the application’s Home page, concluding the login

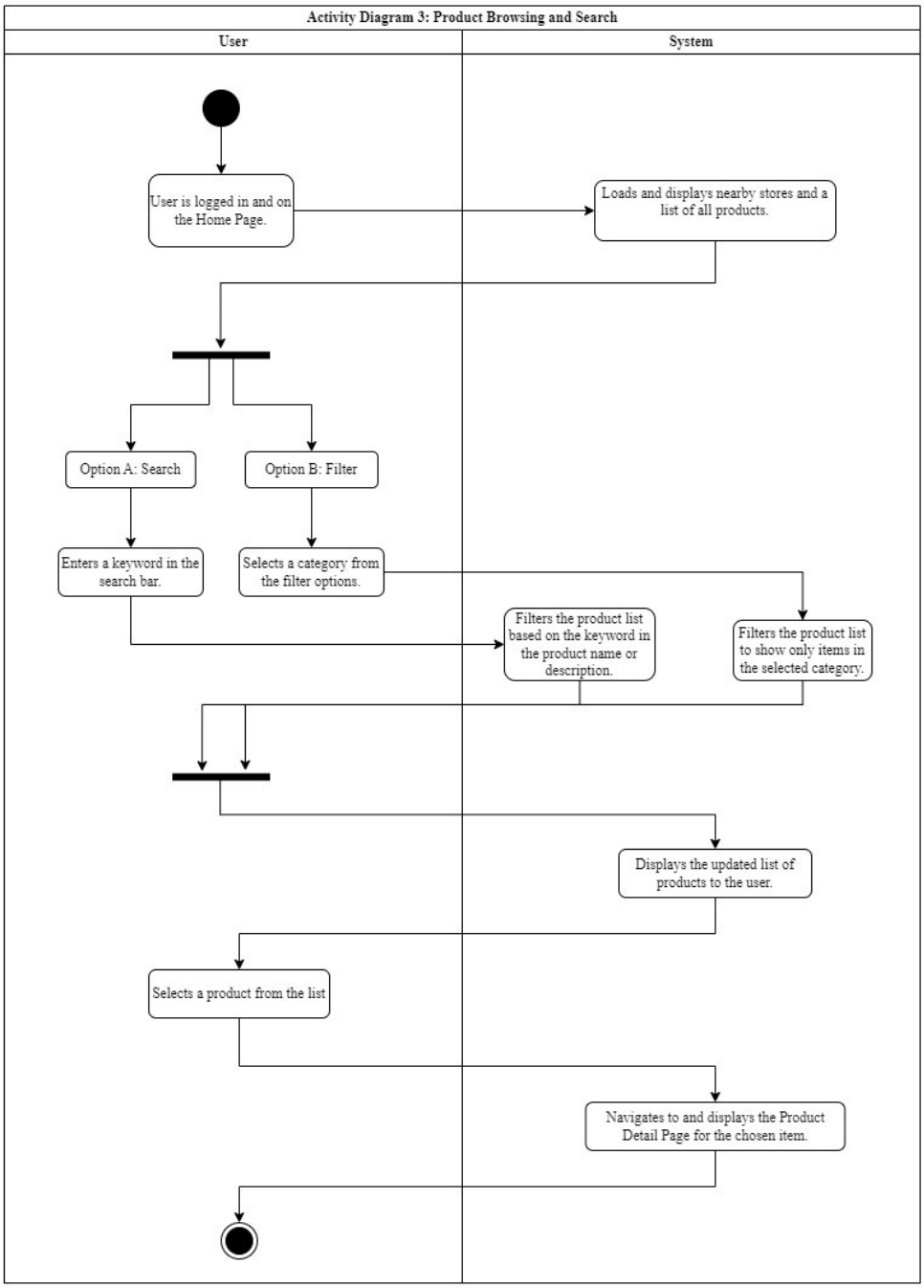



Figure 4.6 Activity diagram for Product Browing and Search

The Product Browsing and Search activity starts with an authenticated user on the Home page, where the system loads and displays nearby stores and an initial catalogue of all products. The user may then refine the catalogue by either entering a keyword in the search bar or selecting a category via the filter options. For a keyword, the system filters the list by matching the term against product names and descriptions; for a category, it restricts the list to items within the chosen category. After applying the selected criterion, the system presents the updated list of products to the user. The user selects an item from this list, and the system navigates to and displays the Product Detail page for the chosen product, concluding the activity.

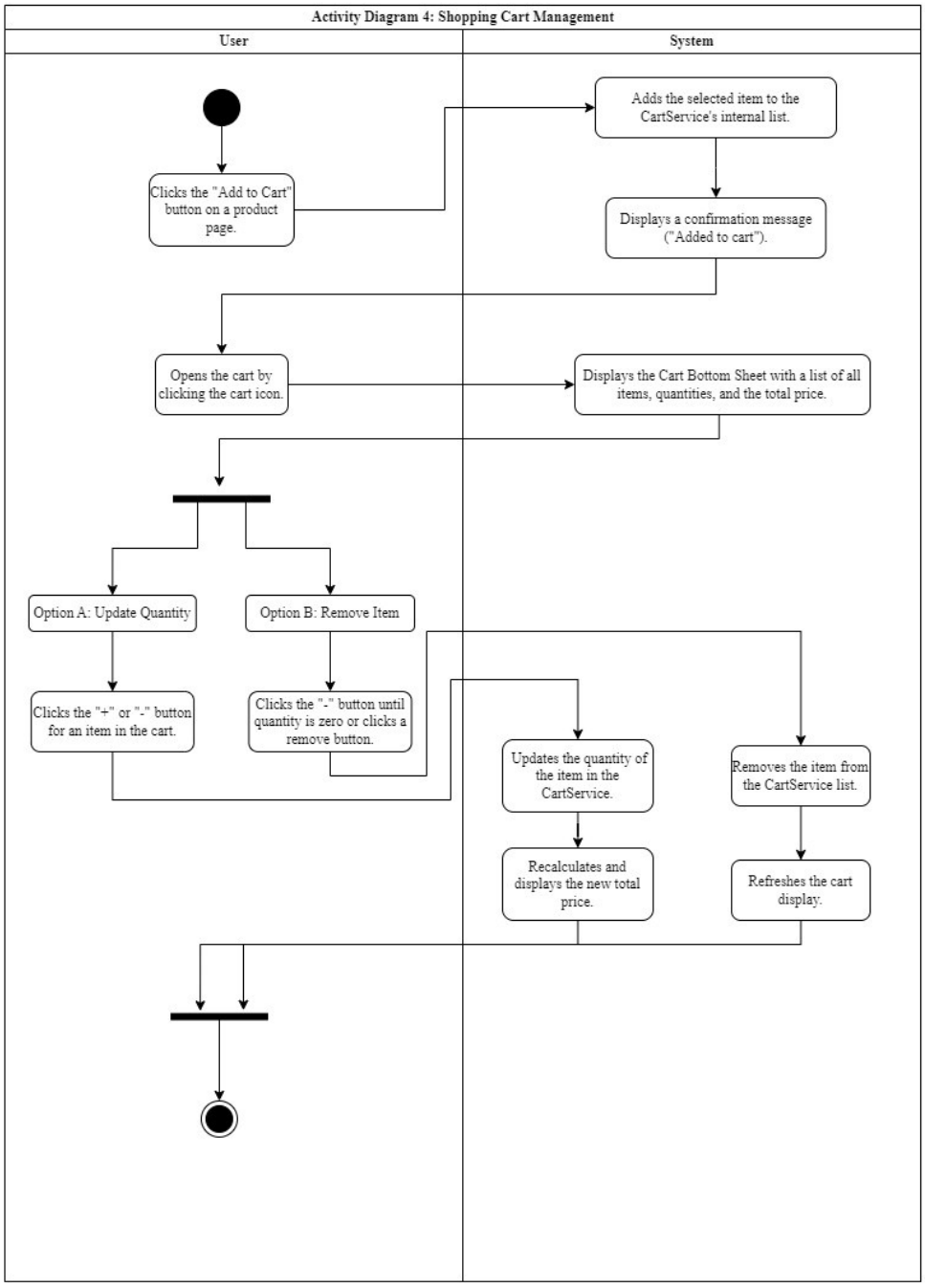



Figure 4.7 Activity diagram for Shopping Cart Management

The Shopping Cart Management activity begins when the user taps add to cart on a product page. The system adds the selected item to the cartService's internal list and displays a confirmation ("Added to cart"). When the user opens the cart via the cart icon, the system presents a bottom sheet showing all items, their quantities, and the current total price. The user can either update quantities or remove items: pressing plus icon or minus icon to adjusts the quantity, upon which the system updates the item in CartService and recalculates and displays the new total; alternatively, pressing minus icon until the quantity reaches zero or selecting a remove action deletes the item, and the system refreshes the cart view. The activity ends once the user has finished managing the cart and exits the cart interface.

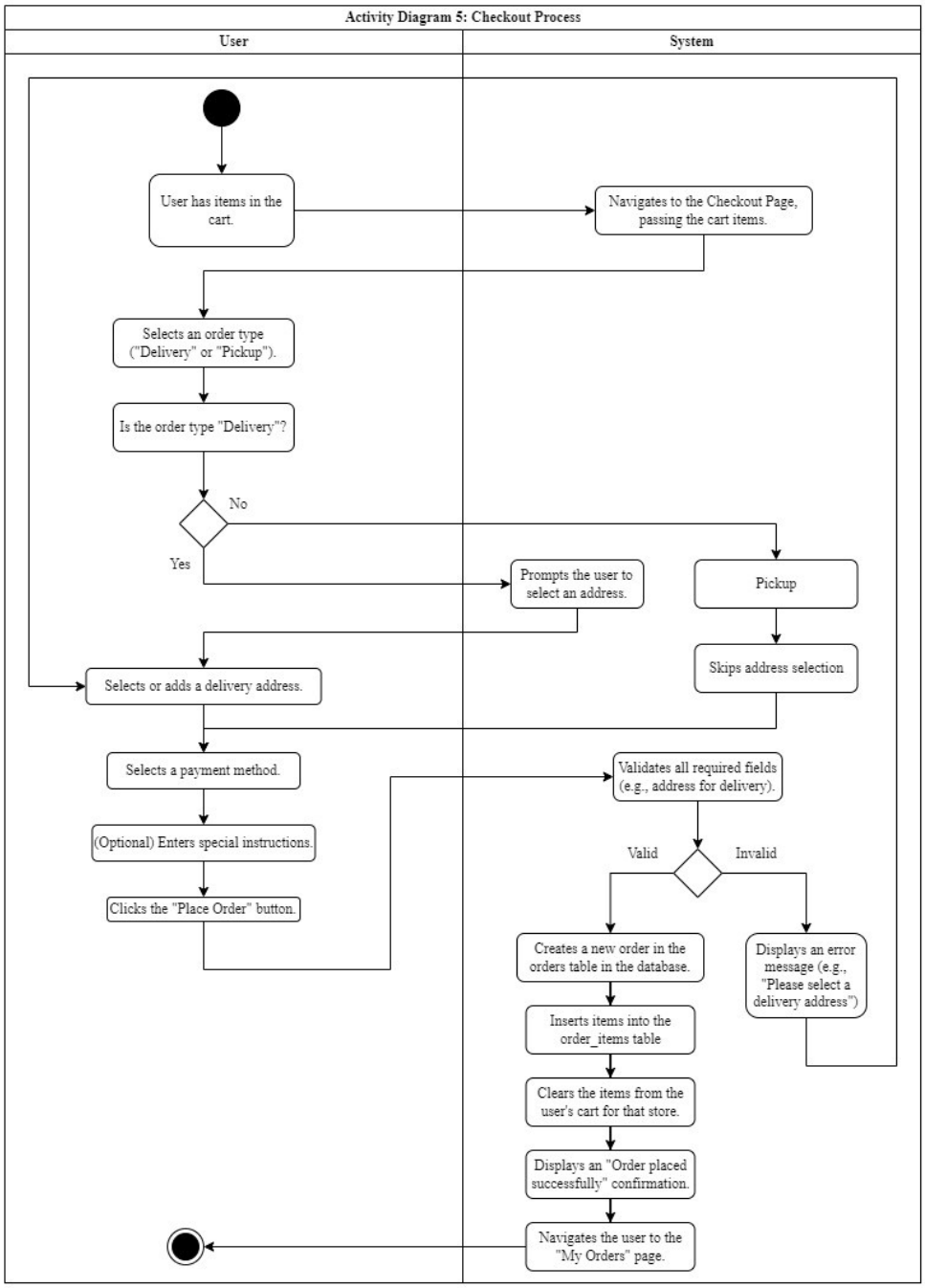



Figure 4.8 Activity diagram for Checkout Process

The Checkout Process activity begins when a user with items in the cart proceeds to the checkout page, where the system loads the cart contents. The user selects an order type via delivery or pickup. For delivery, the system prompts the user to select or add a delivery address; for pickup, address selection is skipped. The user then chooses a payment method and may enter optional special instructions before clicking place order. The system validates all required fields (e.g., presence of a delivery address for delivery orders). If validation fails, an error message is shown, and the user is returned to correct the input. If validation succeeds, the system creates a new order record, inserts the corresponding order items, clears the cart for that store, and displays an "Order placed successfully" confirmation. The user is then navigated to my orders page, concluding the checkout activity.

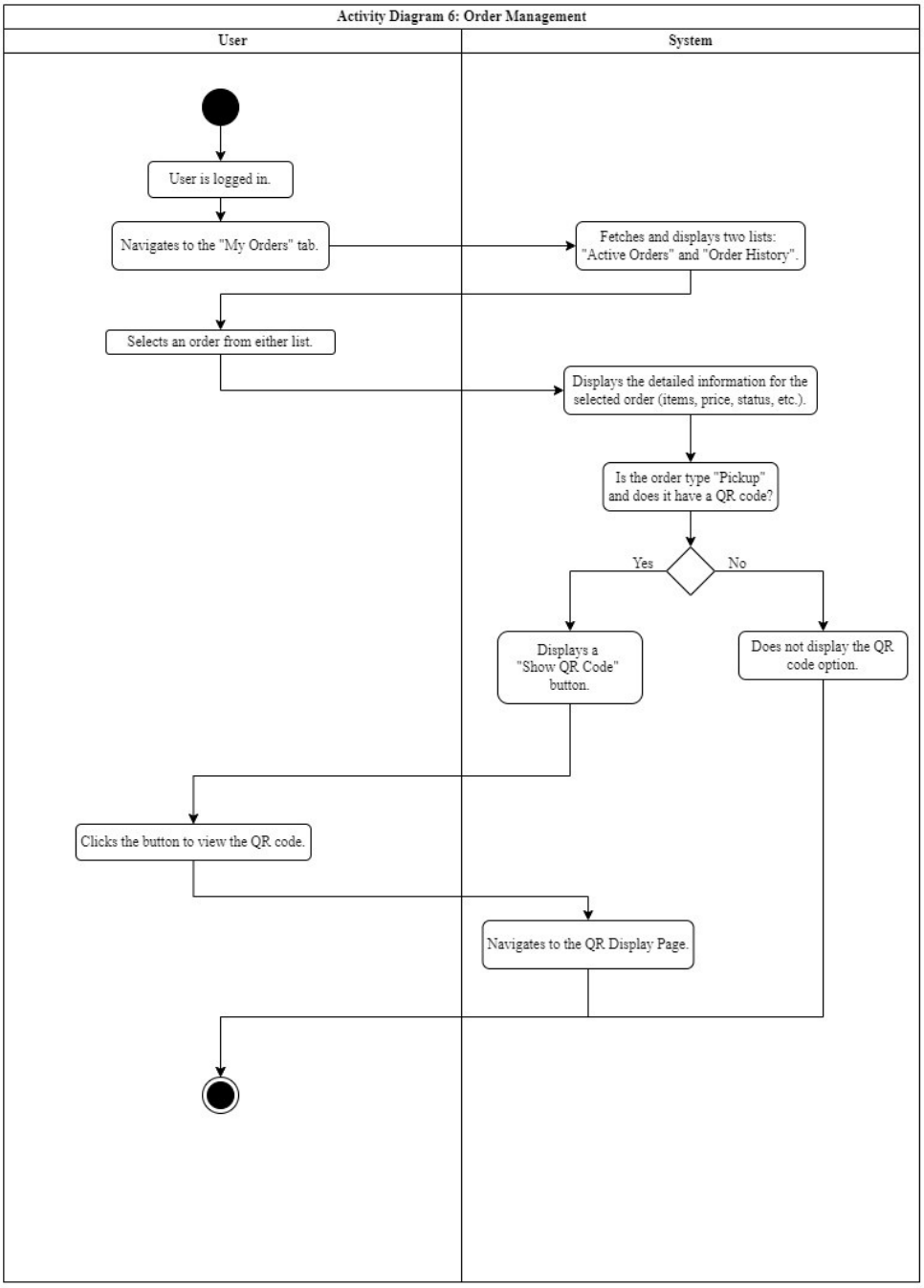



Figure 4.9 Activity diagram for Order Management

The Order Management activity begins with an authenticated user navigating to my orders tab. The system retrieves and displays two lists active orders and order history. When the user selects an order from either list, the system presents the order's details, including items, prices, and status. The system then evaluates whether the order is of type pickup and whether a QR code is available. If both conditions are met, a show QR code button is displayed; otherwise, the QR option is not shown. When the user chooses to view the QR code, the system navigates to the QR display page, concluding the activity.

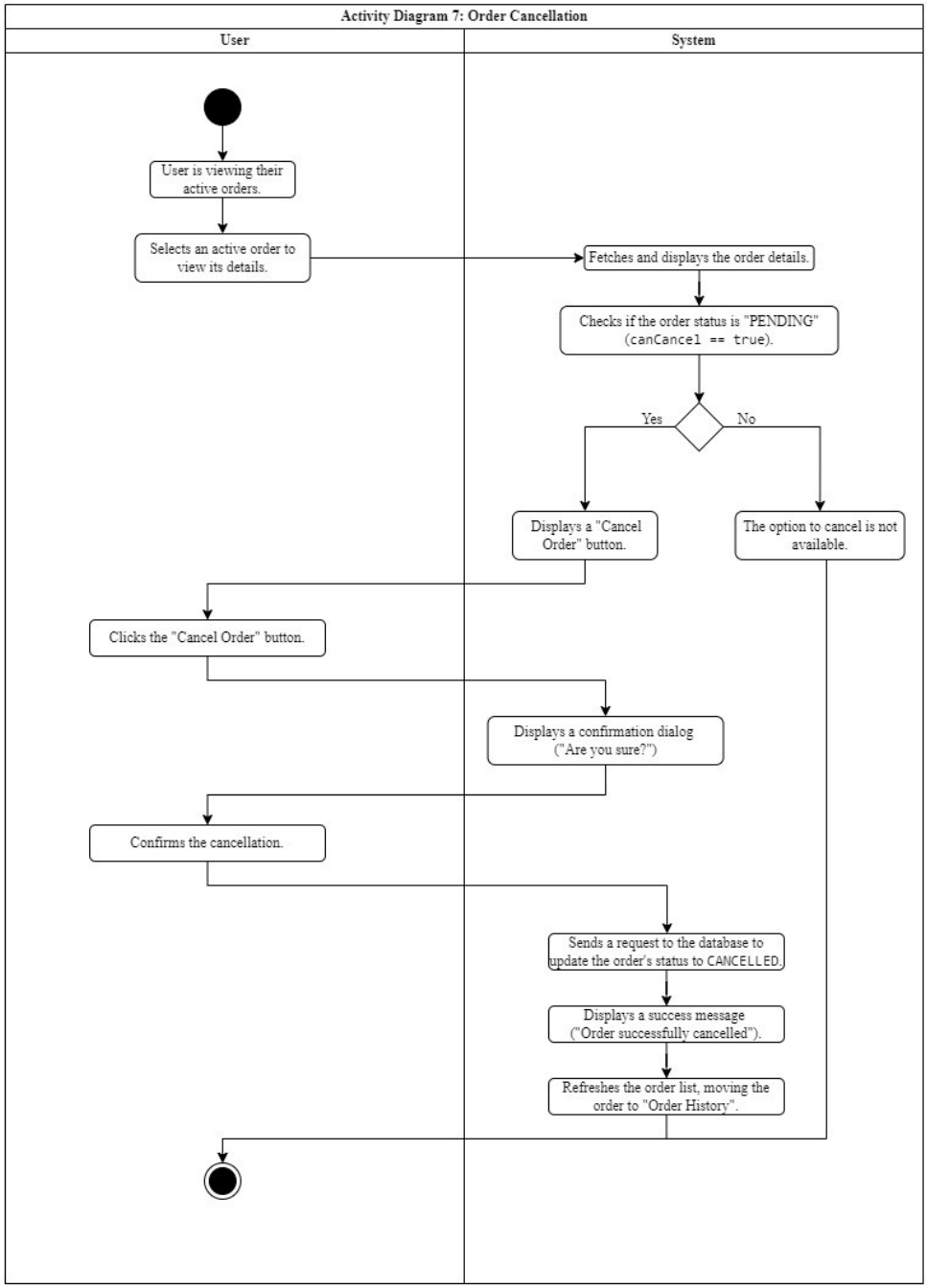



Figure 4.10 Activity diagram for Order Cancellation

The Order Cancellation activity begins when a logged-in user views active orders and selects one to see its details. The system fetches and displays the order information and evaluates whether the order is still cancelable. If the order is not eligible, the cancellation option is not shown. If it is eligible, the system displays a cancel order action; when the user selects it, a confirmation dialog ("Are you sure?") is presented. Upon confirmation, the system updates the order's status to cancelled in the database, shows a success message, and refreshes the lists to remove the order from active orders and move it to order history thereby concluding the process.

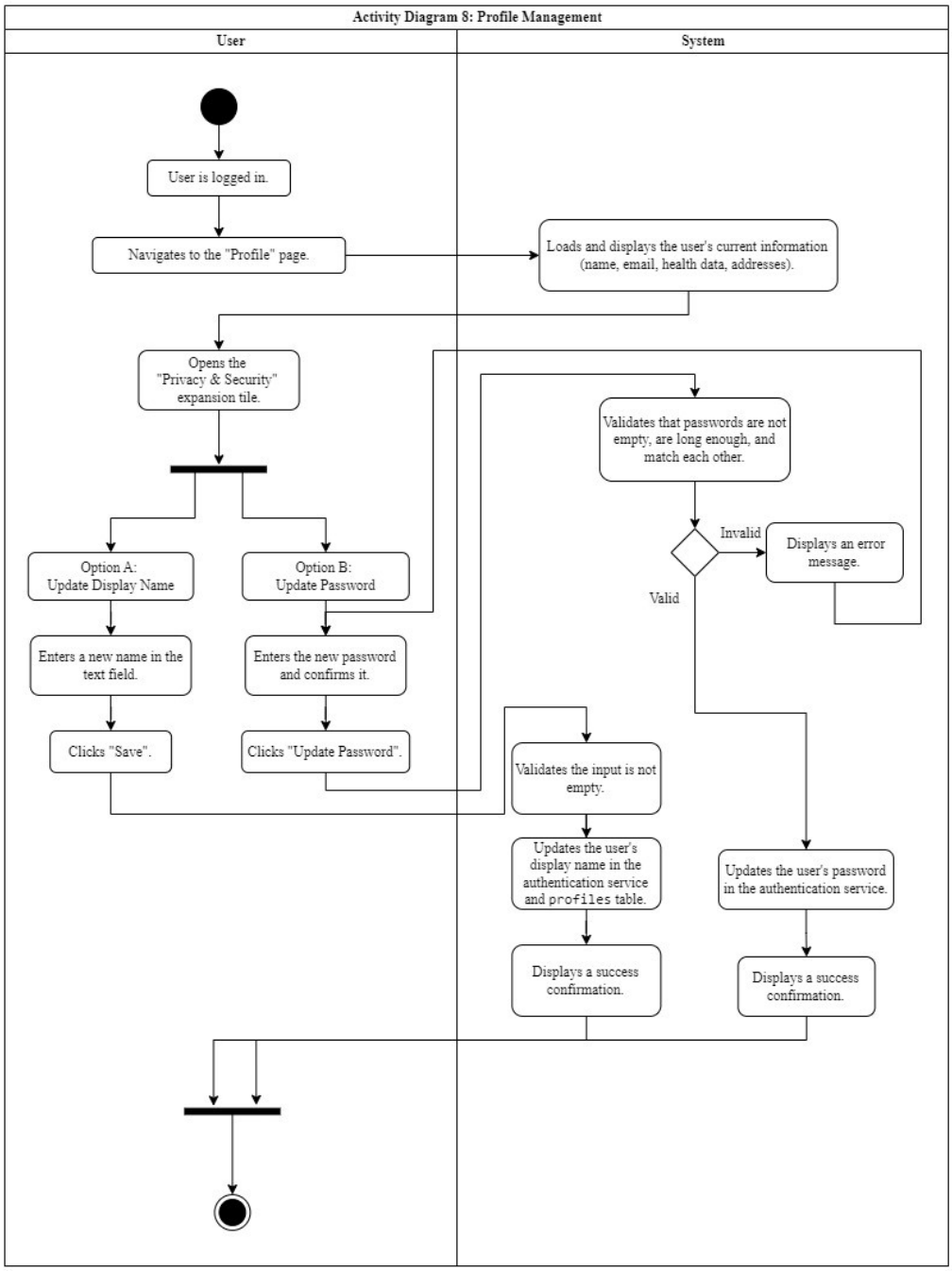



Figure 4.11 Activity diagram for Profile Management

The Profile Management activity begins with an authenticated user navigating to the Profile page, where the system loads and displays the user's current information (name, email, health data, and saved addresses). The user opens the privacy & security section and chooses either to update display name or update password. For a display-name change, the user enters a new name and selects save then the system validates that the field is not empty, updates the display name in both the authentication service and the profiles table, and returns a success confirmation. For a password change, the user enters and confirms a new password; the system verifies that the entries are non-empty, satisfy length requirements, and match. If validation fails, an error message is shown; if it succeeds, the system updates the password in the authentication service and displays a success confirmation. The process concludes upon confirmation and returns to the profile view.

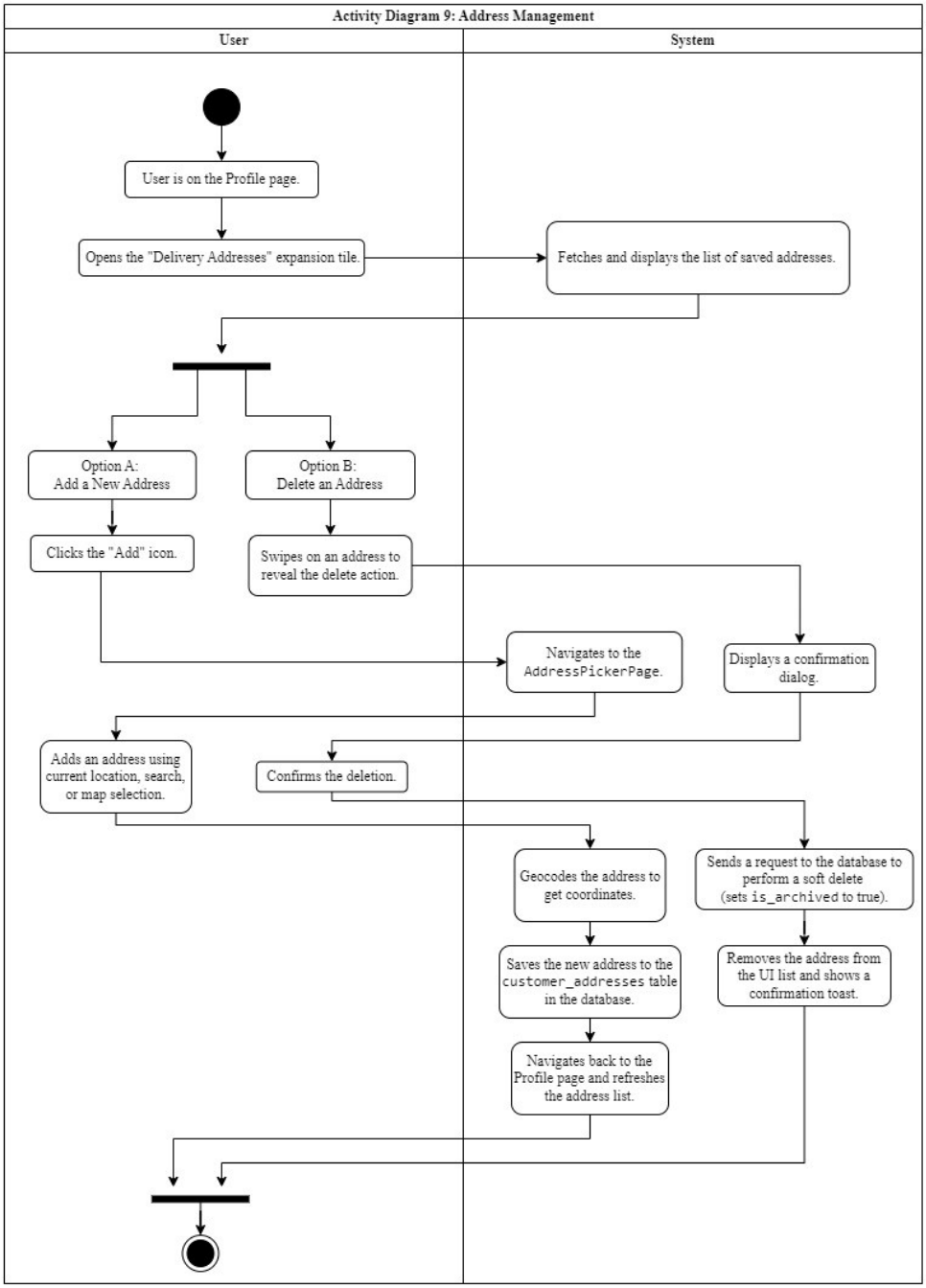



Figure 4.12 Activity diagram for Address Management

The Address Management activity begins with an authenticated user opening the profile page and expanding delivery addresses, upon which the system retrieves and displays the saved address list. The user may add a new address by tapping add, which navigates to the AddressPicker page to provide an address via current location, search, or map selection. The system geocodes the input to coordinates, stores the new record in the customer_addresses table, and returns to the profile page with the refreshed list and confirmation. Alternatively, the user may delete an address by swiping to reveal delete; the system shows a confirmation dialog and, upon confirmation, performs a soft delete, removes the item from the UI, and displays a success toast. The process concludes when the updated list is shown.

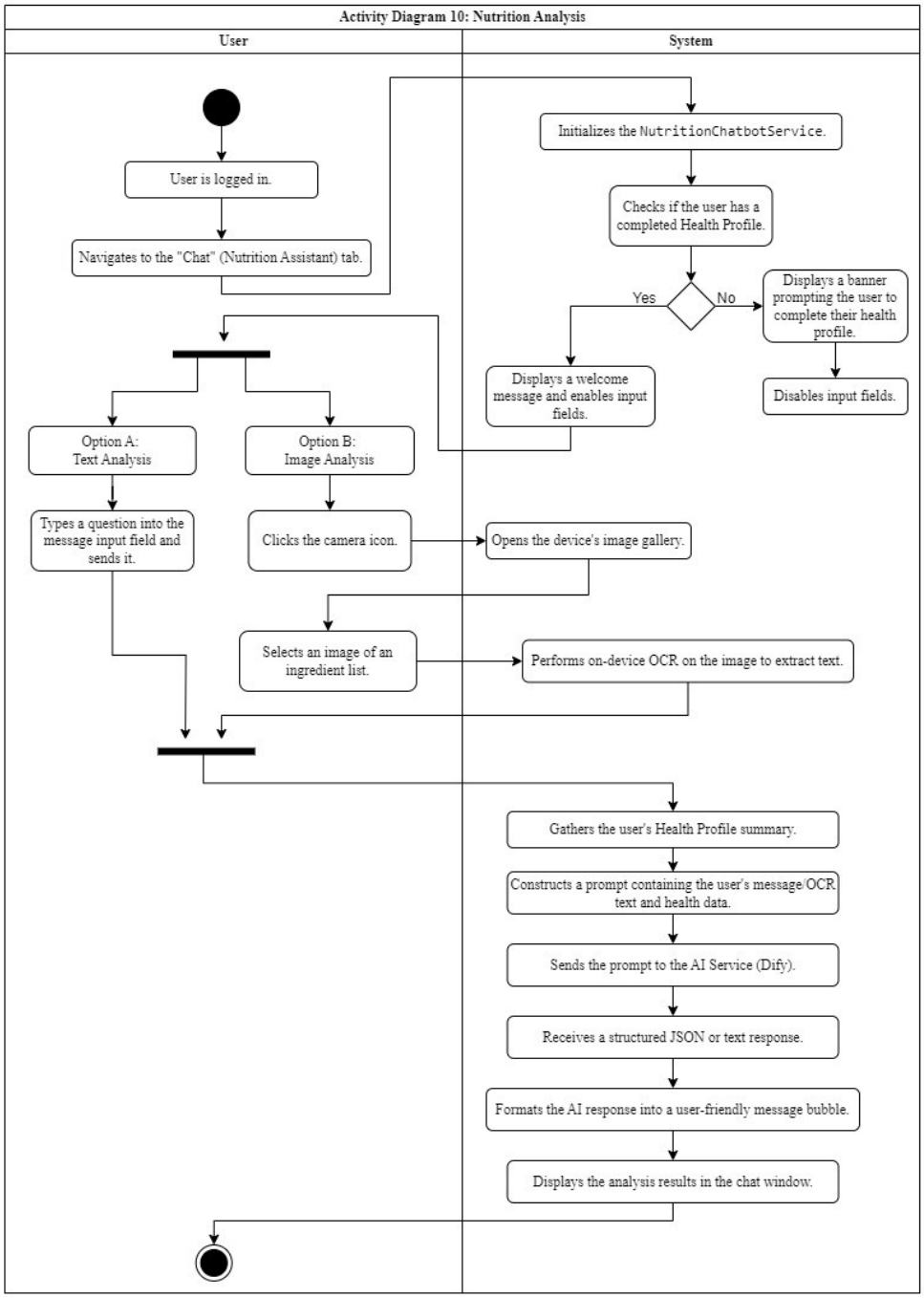



Figure 4.13 Activity diagram for Nutrition Analysis (Chatbot)

The Nutrition Analysis activity begins when an authenticated user opens the Chat (Nutrition Assistant) tab. The system initializes the NutritionChatbotService and checks whether the user has a completed Health Profile. If the profile is incomplete, a banner prompts completion and input fields are disabled; if complete, a welcome message is shown, and inputs are enabled. The user can then proceed via two modes: text analysis, by typing a question and sending it, or image analysis, by tapping the camera icon, selecting an image of an ingredient or nutrition list from the device gallery, after which the system performs on-device OCR to extract text. The system then gathers the user's Health Profile summary and constructs a prompt that includes the typed question or OCR text plus relevant health data, submits this prompt to the AI service (Dify), receives a structured JSON or text reply, formats it into a user-friendly chat bubble, and displays the analysis in the chat window, concluding the activity.

## 4.4 Database Design

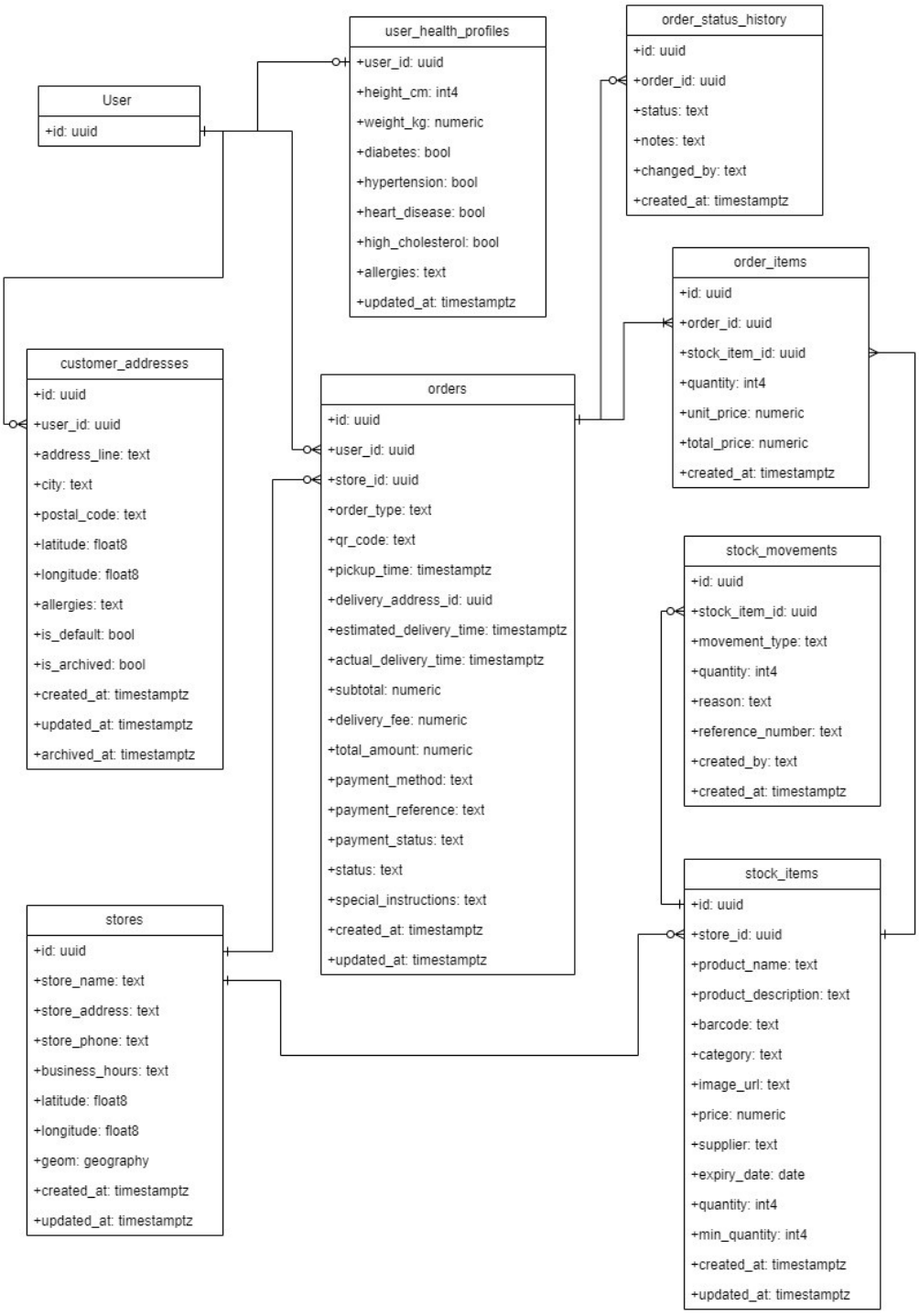



Figure 4.14 Entity–Relationship Diagram (ERD)

Table 4.8 Entity Description

| Entity | Description |
|---|---|
| **User** | Stores the unique ID for each app user (links to orders, health profile, addresses). |
| **user_health_profiles** | Height/weight and boolean health flags for nutrition features. |
| **customer_addresses** | Saved delivery addresses per user, including geolocation and defaults. |
| **stores** | Store metadata (name, contact, hours, location, geometry). |
| **orders** | Customer orders (pickup/delivery), timing, totals, payment info, status. |
| **order_items** | Line items of an order with quantity and pricing. |
| **order_status_history** | Time-stamped status changes for an order (audit trail). |
| **stock_items** | Store inventory catalog with pricing, barcode, category, images. |
| **stock_movements** | Inventory changes (in/out/adjustment) with reasons and references. |

Table 4.9 Data Dictionary of User

| Field Name | Data Type | Null | PK/FK | Description |
|---|---|---|---|---|
| id | uuid | No | PK | User identifier (maps to auth user). |

Table 4.10 Data Dictionary of user_health_profiles

| Field Name | Data Type | Null | PK/FK | Description |
|---|---|---|---|---|
| user_id | uuid | No | PK, FK → User.id | Owner of this health profile. |
| height_cm | int4 | Yes | – | User height in centimeters. |
| weight_kg | numeric | Yes | – | User weight in kilograms. |
| diabetes | bool | Yes | – | Diabetes flag. |
| hypertension | bool | Yes | – | Hypertension flag. |
| heart_disease | bool | Yes | – | Heart disease flag. |
| high_cholesterol | bool | Yes | – | High cholesterol flag. |
| allergies | text | Yes | – | Free-text allergy list. |
| updated_at | timestamptz | No | – | Last update timestamp. |

Table 4.11 Data Dictionary of Stores

| Field Name | Data Type | Null | PK/FK | Description |
|---|---|---|---|---|
| id | uuid | No | PK | Store identifier. |
| store_name | text | No | – | Store display name. |
| store_address | text | No | – | Physical address. |
| store_phone | text | Yes | – | Contact number. |
| business_hours | text | Yes | – | Opening hours text. |
| latitude | float8 | Yes | – | Latitude (WGS-84). |
| longitude | float8 | Yes | – | Longitude (WGS-84). |
| geom | geography | Yes | – | PostGIS geography POINT for spatial queries. |
| created_at | timestamptz | No | – | Creation time. |
| updated_at | timestamptz | No | – | Last update time. |

Table 4.12 Data Dictionary of customer_addresses

| Field Name | Data Type | Null | PK/FK | Description |
|---|---|---|---|---|
| id | uuid | No | PK | Address identifier. |
| user_id | uuid | No | FK → User.id | Address owner. |
| address_line | text | No | – | Street/line details. |
| city | text | No | – | City. |
| postal_code | text | No | – | Postal/ZIP code. |
| latitude | float8 | Yes | – | Latitude (WGS-84). |
| longitude | float8 | Yes | – | Longitude (WGS-84). |
| allergies | text | Yes | – | (Optional) allergy note for deliveries. |
| is_default | bool | No | – | True if default delivery address. |
| is_archived | bool | No | – | Soft delete/archive flag. |
| created_at | timestamptz | No | – | Creation time. |
| updated_at | timestamptz | No | – | Last update time. |
| archived_at | timestamptz | Yes | – | When archived. |

Table 4.13 Data Dictionary of order_items

| Field Name | Data Type | Null | PK/FK | Description |
|---|---|---|---|---|
| id | uuid | No | PK | Order item identifier. |
| order_id | uuid | No | FK → orders.id | Parent order. |
| stock_item_id | uuid | No | FK → stock_items.id | Product (snapshot reference). |
| quantity | int4 | No | – | Units ordered (≥1). |
| unit_price | numeric | No | – | Price per unit at order time. |
| total_price | numeric | No | – | Calculated quantity × unit_price. |
| created_at | timestamptz | No | – | Creation time. |

Table 4.14 Data Dictionary of order_status_history

| Field Name | Data Type | Null | PK/FK | Description |
|---|---|---|---|---|
| id | uuid | No | PK | Status change record. |
| order_id | uuid | No | FK → orders.id | Related order. |
| status | text | No | – | New status after change. |
| notes | text | Yes | – | Optional remarks. |
| changed_by | text | Yes | – | Who changed it (staff/system). |
| created_at | timestamptz | No | – | Timestamp of change. |

Table 4.15 Data Dictionary of stock_items

| Field Name | Data Type | Null | PK/FK | Description |
|---|---|---|---|---|
| id | uuid | No | PK | Inventory item identifier. |
| store_id | uuid | No | FK → stores.id | Owning store. |
| product_name | text | No | – | Display name. |
| product_description | text | Yes | – | Short description. |
| barcode | text | Yes | – | GTIN/UPC/EAN if available. |
| category | text | Yes | – | Product category. |
| image_url | text | Yes | – | Product image URL. |
| price | numeric | No | – | Unit retail price. |
| supplier | text | Yes | – | Supplier name. |
| expiry_date | date | Yes | – | For perishables. |
| quantity | int4 | No | – | On-hand quantity (≥0). |
| min_quantity | int4 | No | – | Reorder threshold. |
| created_at | timestamptz | No | – | Creation time. |
| updated_at | timestamptz | No | – | Last update time. |

Table 4.16 Data Dictionary of orders

| Field Name | Data Type | Null | PK/FK | Description |
|---|---|---|---|---|
| id | uuid | No | PK | Order identifier. |
| user_id | uuid | No | FK → User.id | Customer who placed the order. |
| store_id | uuid | No | FK → stores.id | Fulfilling store. |
| order_type | text | No | – | PICKUP or DELIVERY. |
| qr_code | text | Yes | – | Encoded pickup token (pickup orders). |
| pickup_time | timestamptz | Yes | – | Scheduled pickup time. |
| delivery_address_id | uuid | Yes | FK → customer_addresses.id | Destination (delivery orders). |
| Estimated_delivery_time | timestamptz | Yes | – | Estimated delivery time. |
| actual_delivery_time | timestamptz | Yes | – | Actual completion time. |
| subtotal | numeric | No | – | Sum of item totals (pre-fees). |
| delivery_fee | numeric | Yes | – | Delivery charge. |
| total_amount | numeric | No | – | Final payable amount. |
| payment_method | text | Yes | – | Card / e-wallet / cash, etc. |
| Payment_reference | text | Yes | – | Gateway/reference number. |
| Payment_status | text | Yes | – | PAID, PENDING, FAILED, etc. |

| | | | | |
|---|---|---|---|---|
| status | text | No | – | PLACED, PREPARING, READY, COMPLETED, CANCELLED, etc. |
| Special _instructions | text | Yes | – | Buyer notes for store/courier. |
| created_at | timestamptz | No | – | Creation time. |
| updated_at | timestamptz | No | – | Last update time. |

Table 4.17 Data Dictionary of stock_movements

| Field Name | Data Type | Null | PK/FK | Description |
|---|---|---|---|---|
| id | uuid | No | PK | Movement record identifier. |
| stock_item_id | uuid | No | FK → stock_items.id | Affected inventory item. |
| movement_type | text | No | – | IN, OUT, or ADJUST. |
| quantity | int4 | No | – | Units moved (positive integer). |
| reason | text | Yes | – | Reason (purchase, sale, spoilage, etc.). |
| reference_number | text | Yes | – | Related doc/ref. |
| created_by | text | Yes | – | Who performed the movement. |
| created_at | timestamptz | No | – | Timestamp of entry. |

# CHAPTER 5

# System Implementation

## 5.1 Software Setup and Configuration

### 5.1.1 Android Studio

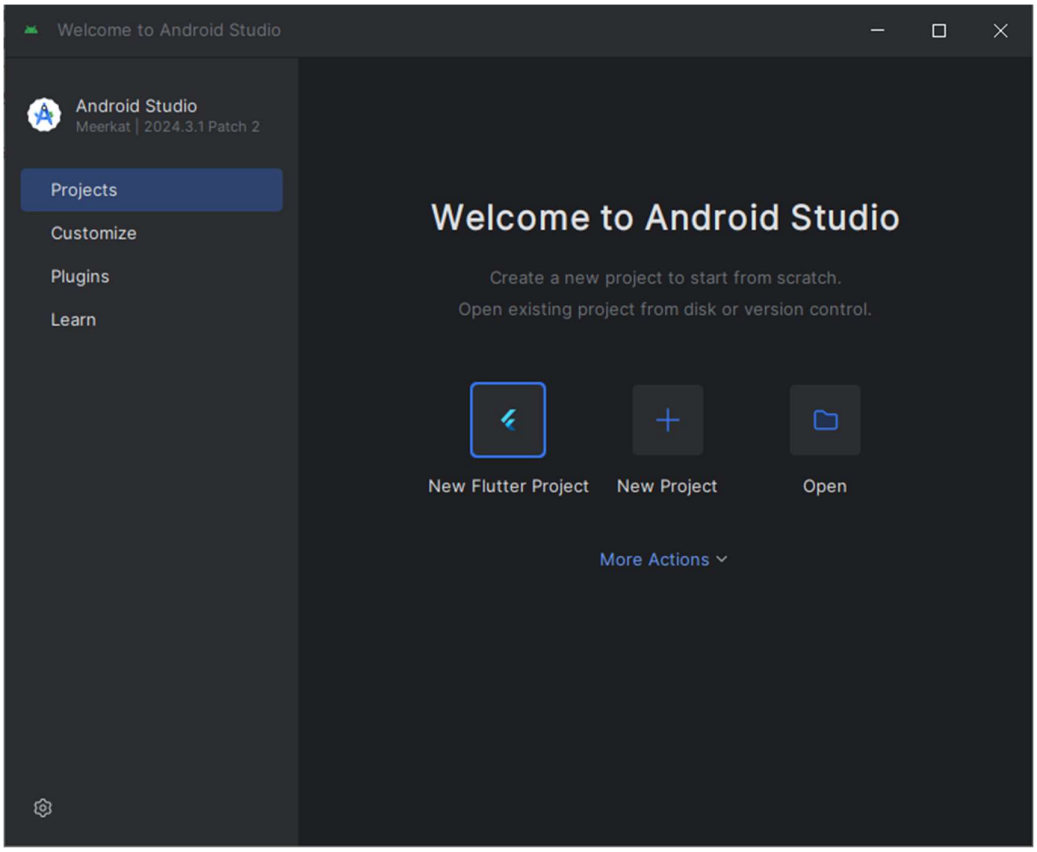



Figure 5.1 Android Studio

Android Studio are the free open-source development tools that can be downloaded from the official site. Android Studio Meerkat has been selected for project usage. When the installation process is complete, the Android Studio interface will appear as presented in Figure 5.1. Android Studio serves as a support environment to the base environment, Visual Studio Code, to a great extent by its implementation as an emulator for Android. Set up for the emulator of Android in Android Studio follows that.

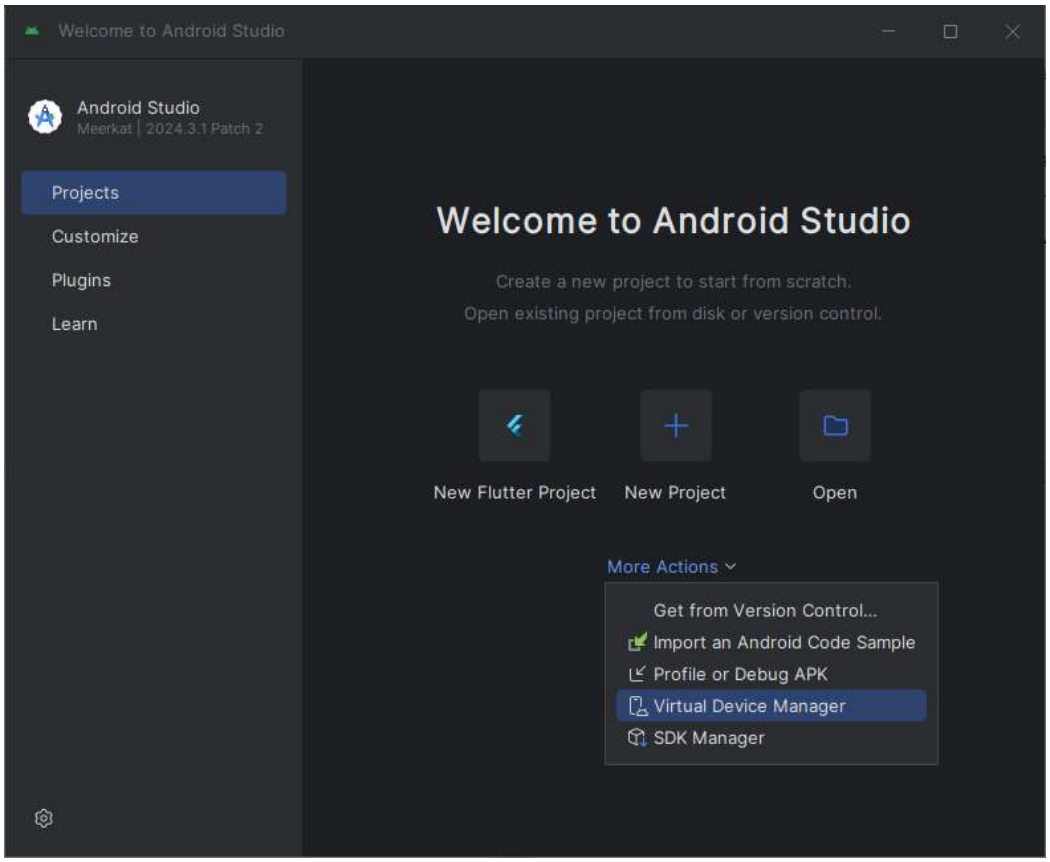



Figure 5.2 Android Emulator Setup (Step 1)

To configure the Android emulator within Android Studio, select the "More Actions" option as indicated in Figure 5.2. A dropdown menu will appear; from this menu, choose "Virtual Device Manager."

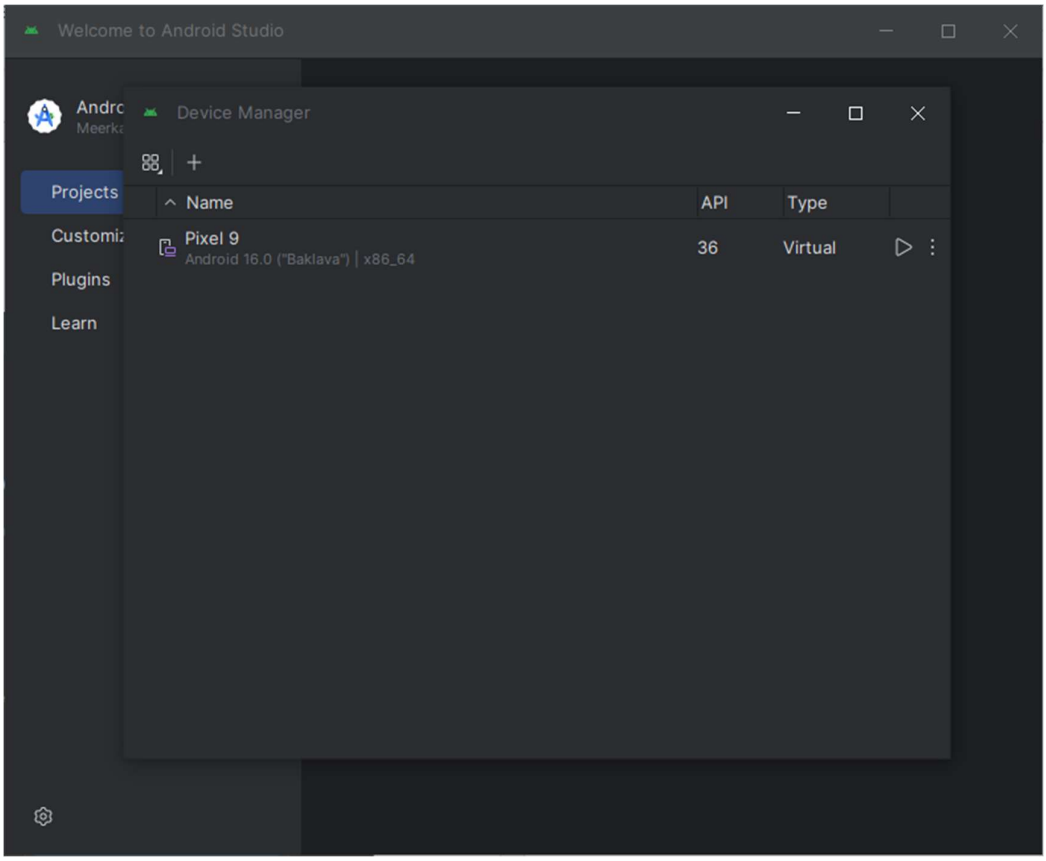



Figure 5.3 Android Emulator Setup (Step 2)

Upon clicking the "Virtual Device Manager," the Device Manager window will appear. Next, click the "+" icon, located as the second option in the top toolbar of the window, as illustrated in Figure 5.3.

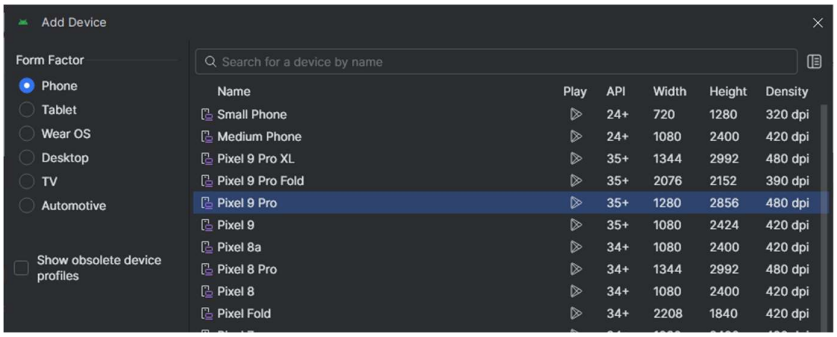



Figure 5.4 Android Emulator Setup (Step 3)

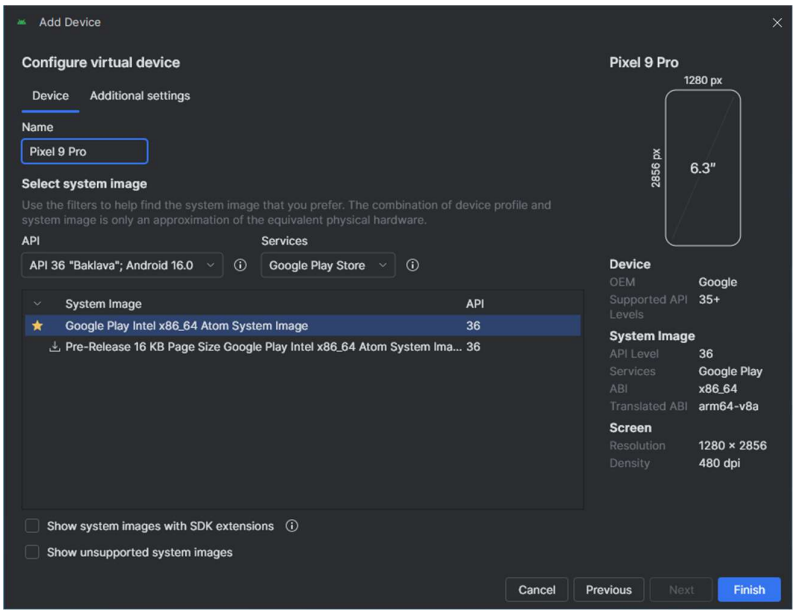



Figure 5.5 Android Emulator Setup (Step 4)

The virtual device configuration window will appear, as depicted in Figure 5.4, showing the Android Emulator Setup (Step 3). For this project, the selected device configuration is "Pixel 9 Pro," and the API level is set to Android 16.0 "Baklava," as illustrated in Figure 5.5 Android Emulator Setup (Step 4). Other settings remain at their default values. After completing the configuration, the Android Emulator can be launched via Android Studio's Virtual Device Manager (AVD Manager).

### 5.1.2 Visual Studio Code

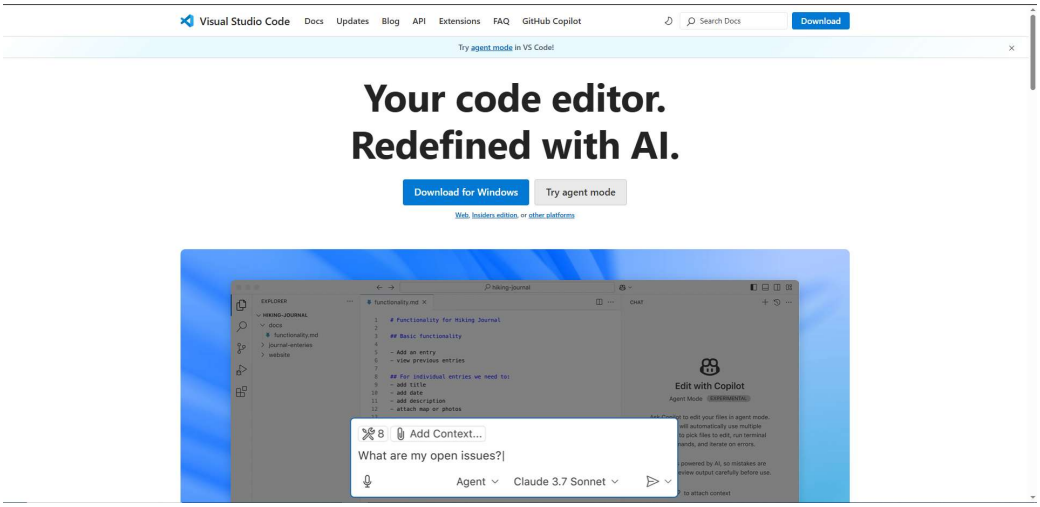



Figure 5.6 Visual Studio Code (Step 1)

Visual Studio Code serves as the primary code editor for this project. To install it, visit the official website (Download Visual Studio Code – Mac, Linux, Windows). Once the installation is complete, the Visual Studio Code Build Tools setup window will appear. From there, choose the "Desktop development with C++" option and proceed with the default installation settings.

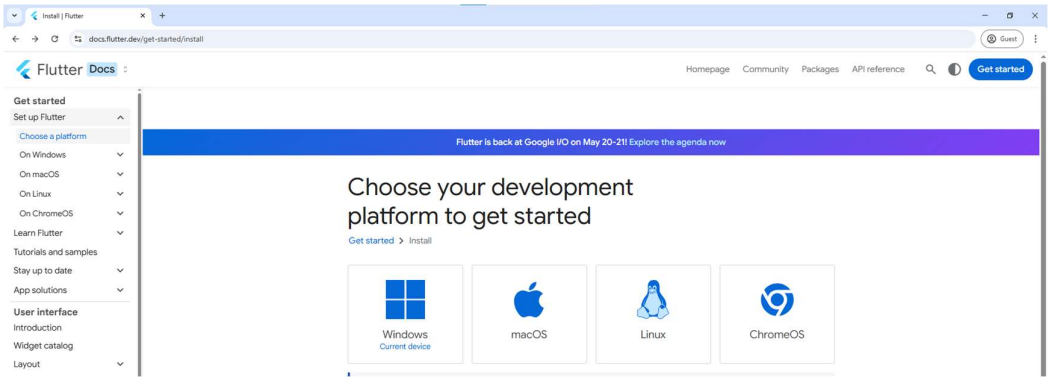



Figure 5.7 Visual Studio Code (Step 2)

After installing Visual Studio Code, the next step is to set up Flutter using its official online platform. As shown in Figure 5.7 (Visual Studio Setup – Step 2), visit the Flutter installation page and select your development platform—Windows, macOS, Linux, or ChromeOS—to begin the setup process.


Figure 5.8 Visual Studio Code (step 3)

Following that, proceed to choose your first target platform as illustrated in Figure 5.8 (Visual Studio Setup – Step 3). For this project, "Android" is recommended as the initial app type. This selection will help configure the necessary Flutter tools for Android development.

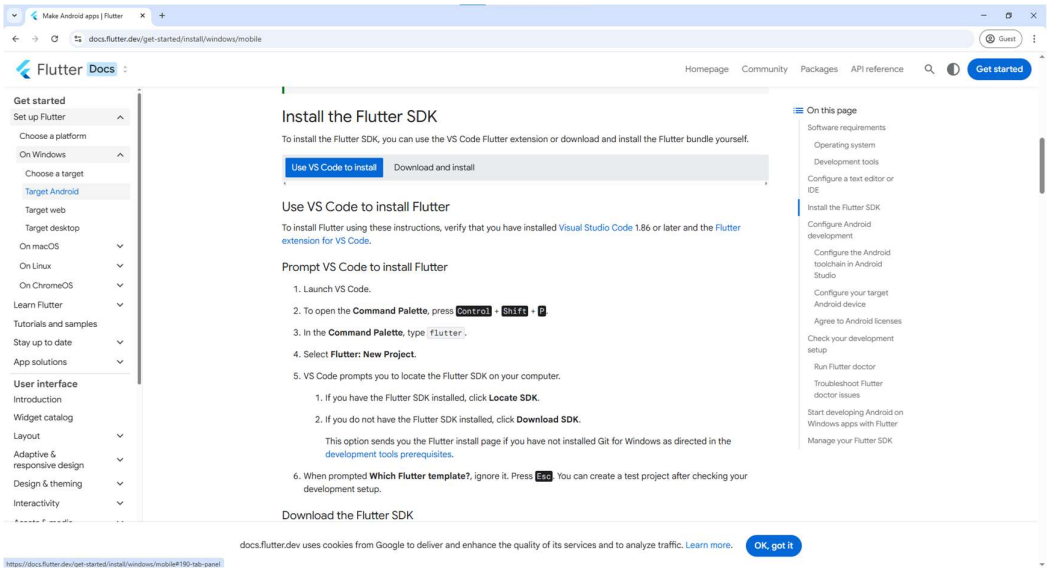



Figure 5.9 Visual Studio Code (step 4)

In Step 4, as shown in Figure 5.9, follow the on-screen instructions to install Flutter using Visual Studio Code. This involves opening the Command Palette (Ctrl + Shift + P), searching for "Flutter: New Project," and allowing VS Code to download or locate the Flutter SDK accordingly.

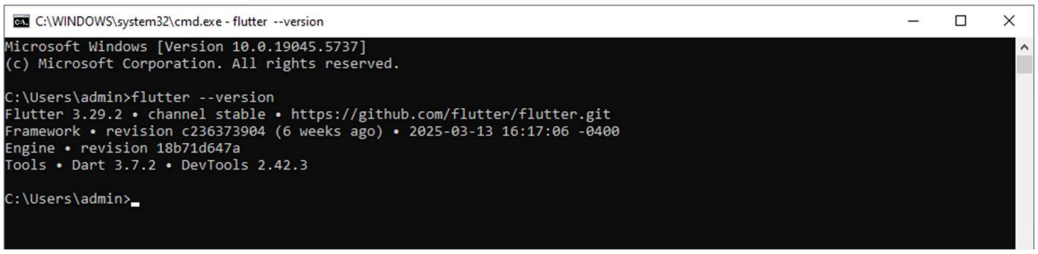



Figure 5.10 Visual Studio Code (step 5)

Once the installation is complete, proceed to Step 5, as illustrated in Figure 5.10. Open the command prompt and type “flutter –version” to confirm that Flutter has been installed correctly and to view the current version details.

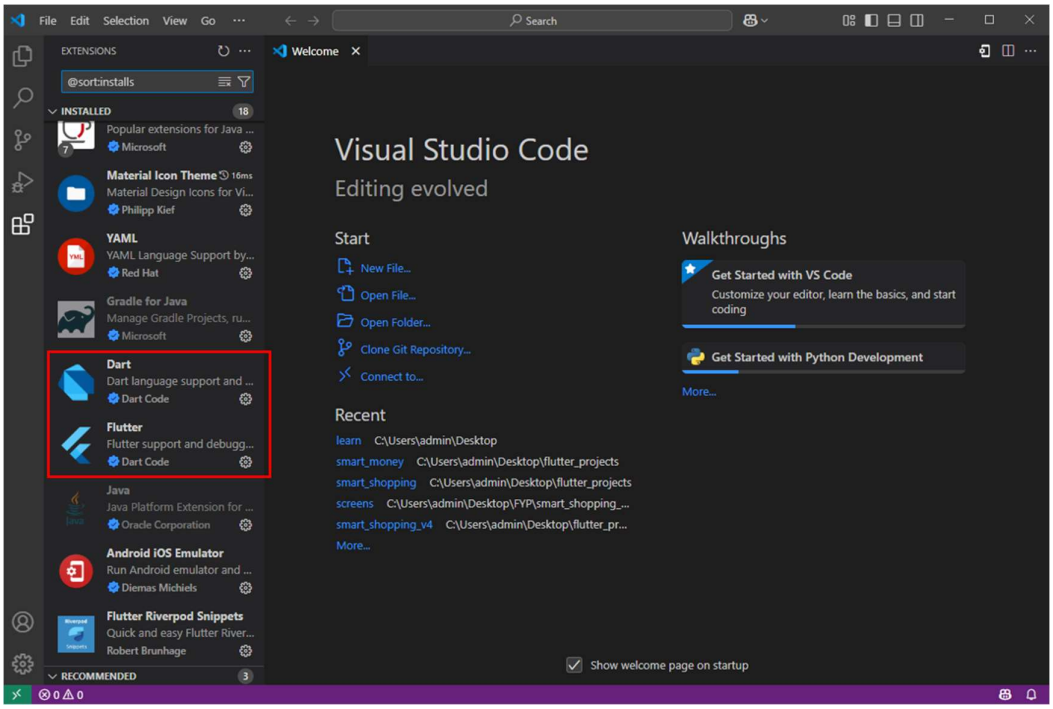



Figure 5.11 Visual Studio Code (step 6)

In Visual Studio Code, install the required extensions to enable Flutter development. As shown in Figure 5.11 (Visual Studio Setup – Step 6), install both the Dart and Flutter extensions from the Extensions Marketplace. These plugins provide essential language support and debugging tools necessary for building and running Flutter applications.

### 5.1.3 Supabase

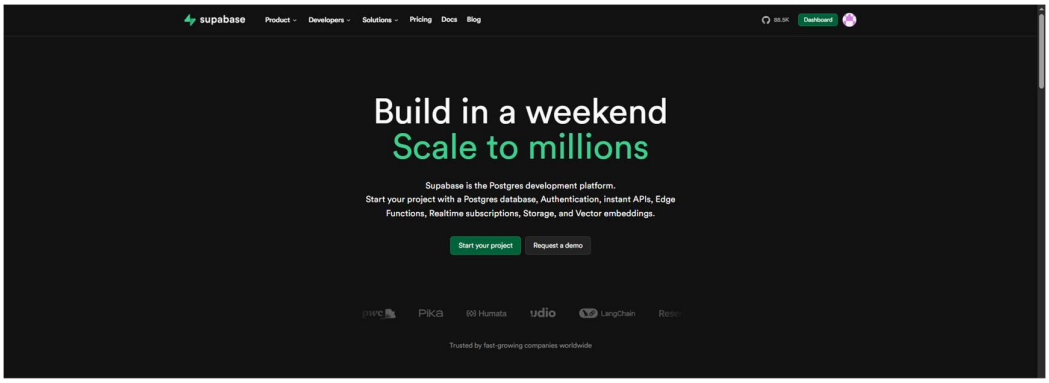



Figure 5.12 Supabase (Step 1)

To begin setting up a Supabase backend, navigate to the official Supabase website and sign in with your account credentials. On the landing page, click the “Start your project” button as illustrated in Figure 5.12 (Supabase – Step 1). This action opens the Supabase Dashboard, where you can proceed to create a new organization and project for your application, selecting the region, pricing plan, and database credentials as required.

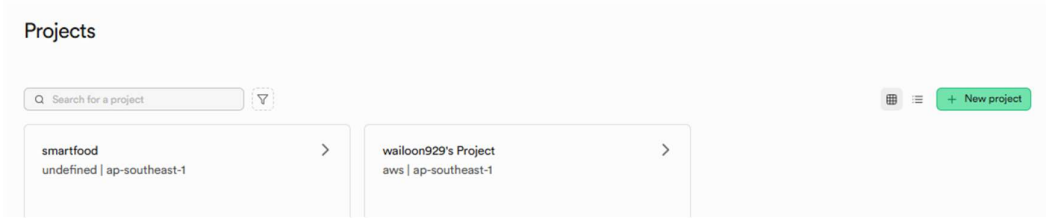



Figure 5.13 Supabase (Step 2)

After signing in to the Supabase Dashboard, navigate to the projects page and click new project, as illustrated in Figure 5.13. This opens the project-creation workflow where you will specify the basic details for your backend. If you already have existing projects, they will appear in the grid; otherwise, the interface presents the option to create a new one to begin provisioning a dedicated PostgreSQL instance for your application.

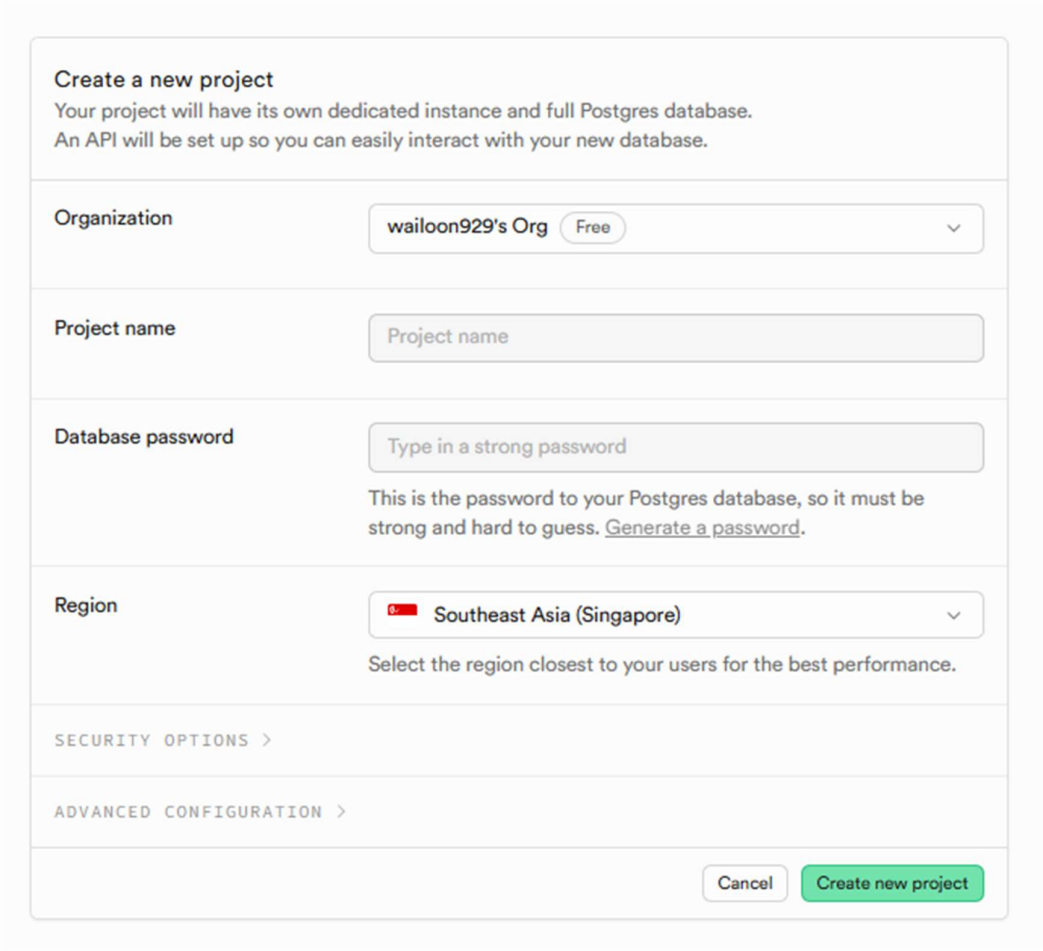



Figure 5.14 Supabase (Step 3)

In the Create a new project dialog (Figure 5.14), select or create an organization, enter a clear Project name, and set a strong Database password for the managed PostgreSQL database. Choose the appropriate Region (e.g., *Southeast Asia – Singapore* / *ap-southeast-1*) to minimize latency for your users, then review the plan and optional security/advanced settings. Click Create new project to start provisioning; once completed, the dashboard will confirm that the project is ready for further configuration.

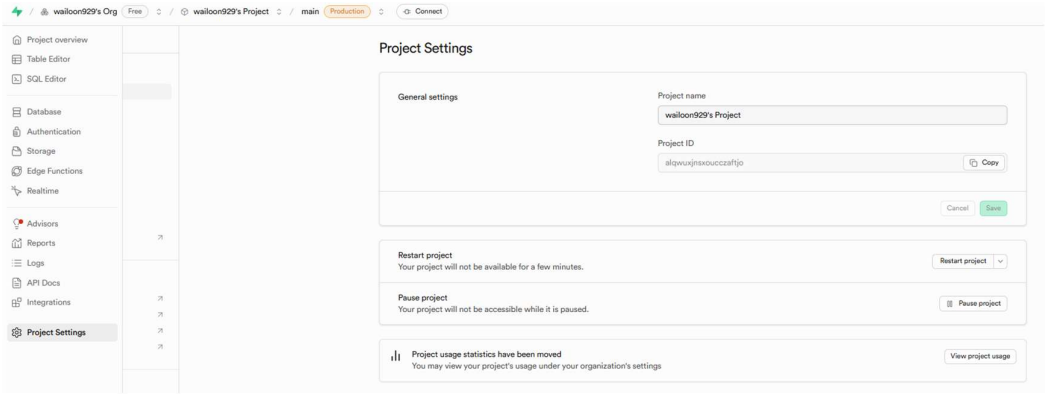



Figure 5.15 Supabase (Step 4)

Once the Supabase project has been created, open Project Settings as shown in Figure 5.15. This page provides the general project details (project name, ID, and controls such as pause or restart) and serves as the entry point to essential configuration pages. From here, proceed to Settings → API to obtain the connection parameters that will be used by the Flutter application.

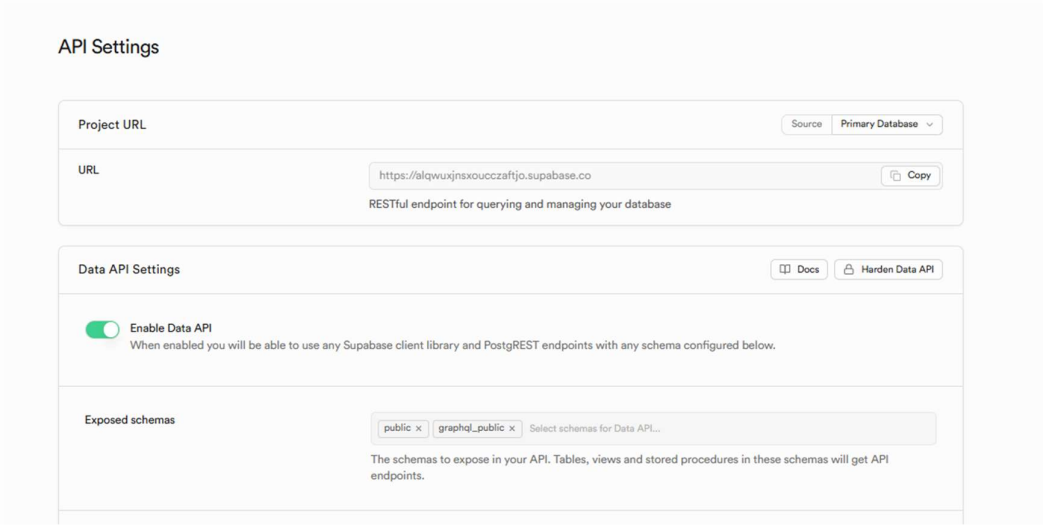



Figure 5.16 Supabase (Step 5)

Navigate to Settings → API to access the API Settings panel, as illustrated in Figure 5.16. Copy the Project URL, then enable the Data API to expose REST endpoints (PostgREST) for database. Under Exposed schemas, select the schemas that your client should access—typically public (and graphql_public if GraphQL is required). Save the changes to make these endpoints available for your app.

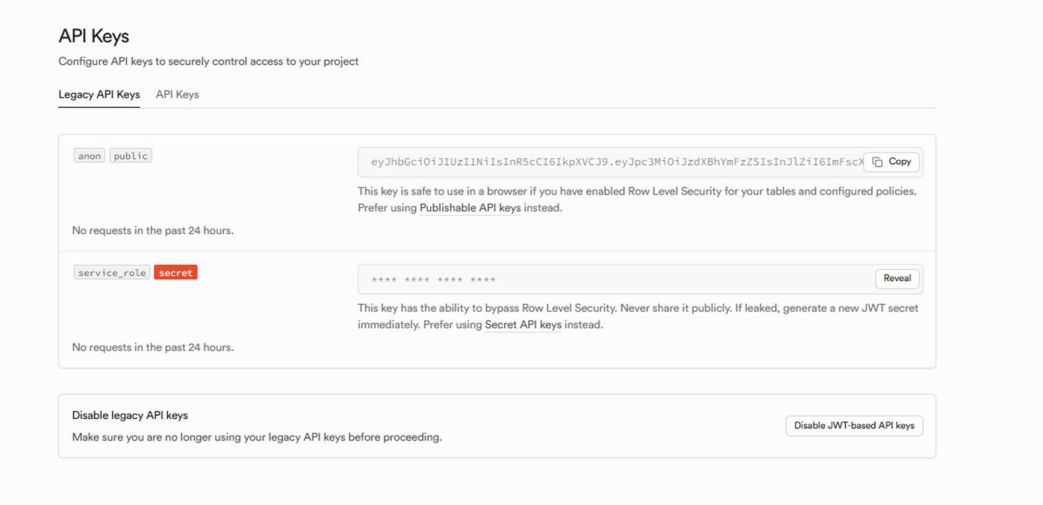



Figure 5.17 Supabase (Step 6)

.

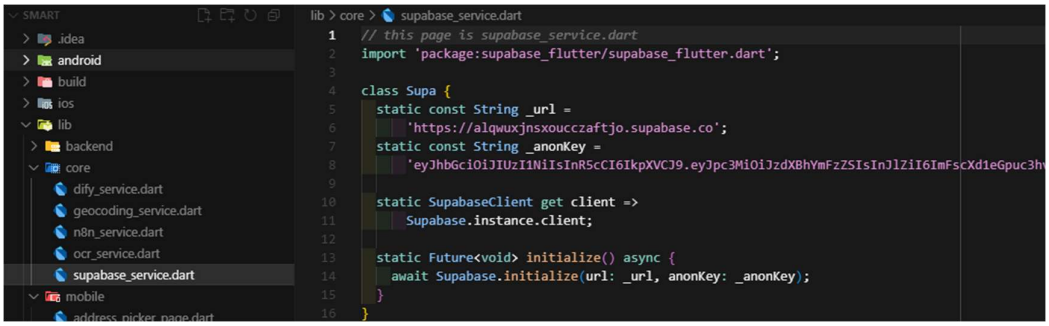



Figure 5.18 Supabase (Step 7)

Under the same API section, open API Keys as shown in Figure 5.17. Copy the anon (public) key; this key is intended for use in client applications and respects Row Level Security policies. Do not use or expose the service_role key in any client app, as it bypasses RLS and is meant only for secure server environments. Rotate keys immediately if they are ever leaked. With the Project URL and anon key ready, integrate Supabase into Flutter project as depicted in Figure 5.18. Add the dependency (supabase_flutter) and initialize the client in your app startup code using the copied Project URL and anon key. After initialization, the Flutter app can securely access Supabase services—including Authentication, PostgreSQL (via REST/GraphQL), Storage, and Realtime—according to the policies have configure in the project

### 5.1.4 Dify.Ai

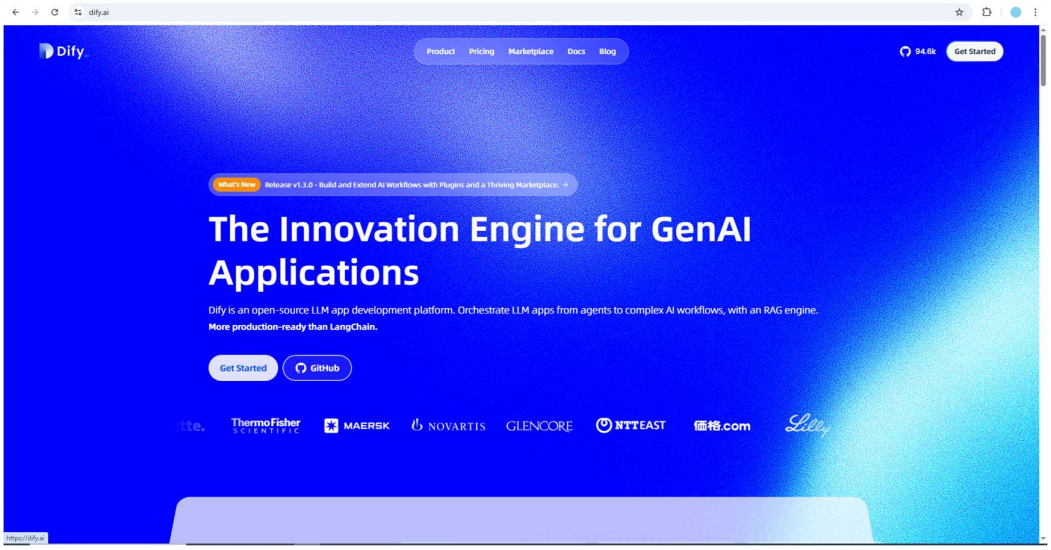



Figure 5.19 Dify AI Official Webpage Overview

To begin integrating the chatbot functionality, the development started by accessing the Dify AI official webpage (Figure 5.19). Dify AI is an open-source LLM app orchestration platform that provides users with tools to easily create, manage, and deploy AI-driven applications without the need for extensive backend development. The platform supports a wide range of AI models, allowing flexibility in building customized intelligent systems.

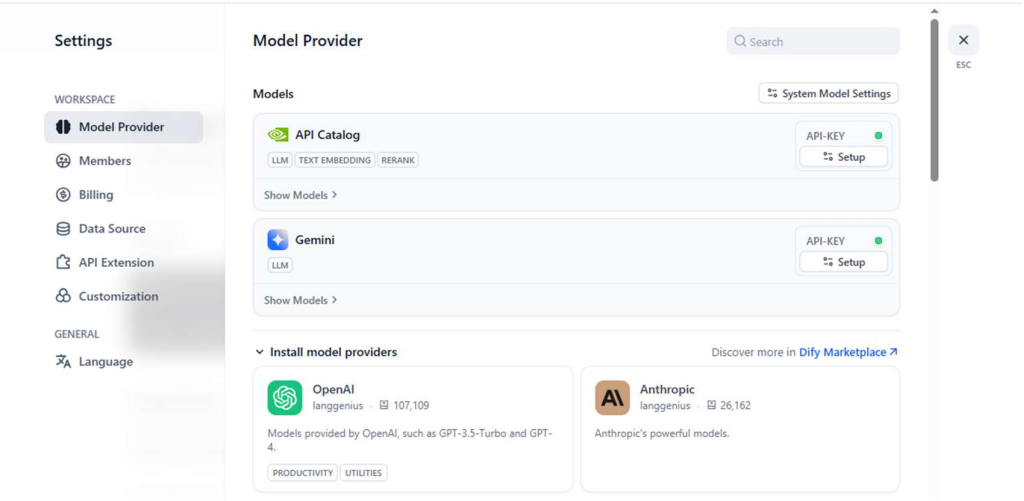



Figure 5.20 Setting Up Gemini API Key in Dify AI

After entering the platform, the next step was to configure the model provider settings (Figure 5.20). In the settings page, users can select different LLM providers such as OpenAI, Gemini, or Anthropic. For this project, the Gemini API was chosen by setting up and inserting the API key, enabling Dify AI to leverage Gemini's capabilities for chatbot responses.

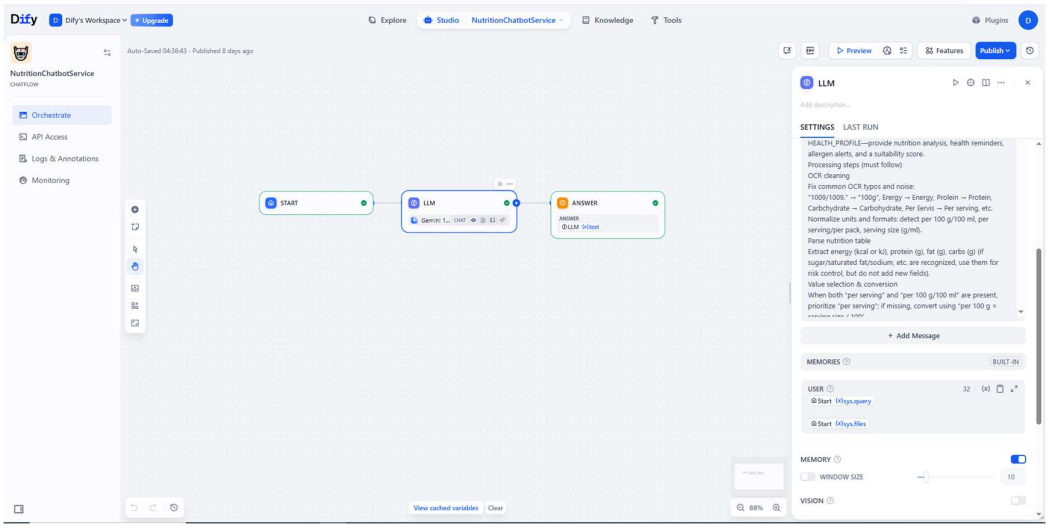



Figure 5.21 Creating a New Chatbot in Dify AI

Following the model configuration, the chatbot creation process was initiated by selecting the “Chatbot” option under “Create from Blank” (Figure 5.21). The application’s basic metadata—name, description, and icon—was completed, establishing the chatbot’s identity and intended purpose. This step prepared the workspace for subsequent node configuration and testing.

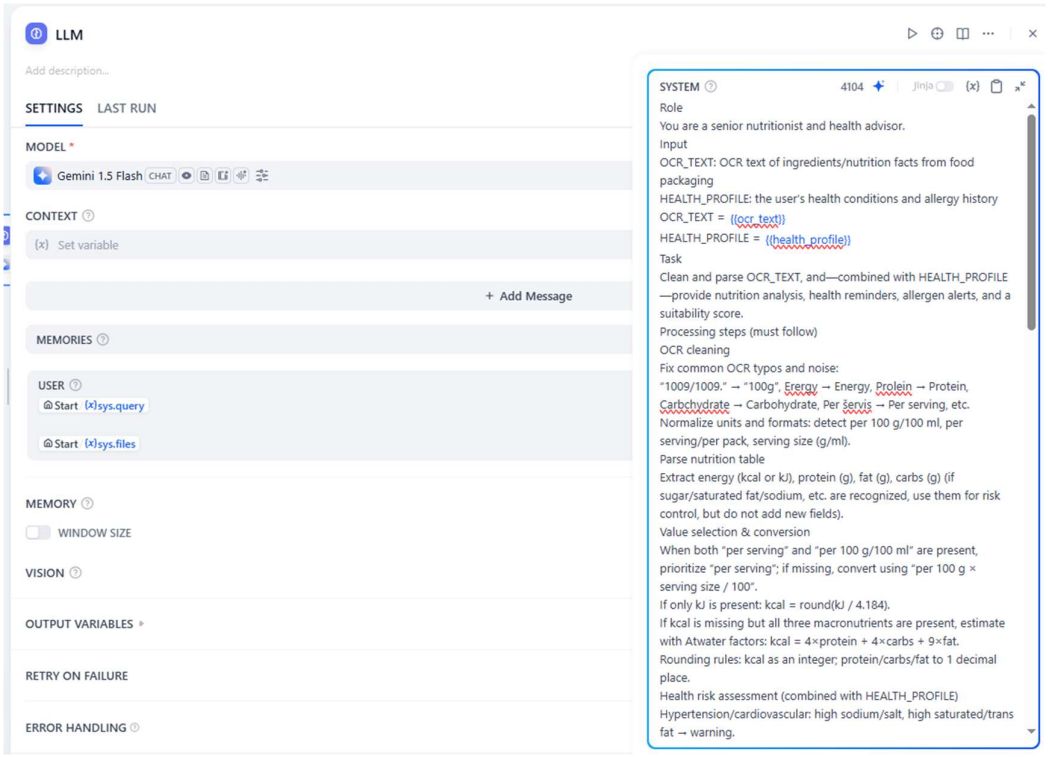



Figure 5.22 Defining System Instructions and Model

After the canvas was created, an Large Language Models (LLM) node was added, and the system prompt was entered to govern the chatbot’s behaviour (Figure 5.22). The prompt specified inputs (e.g., OCR text and health profile) and processing rules to ensure consistent, domain-appropriate responses. The Gemini model was then selected, finalizing the model settings required for controlled and reliable inference.

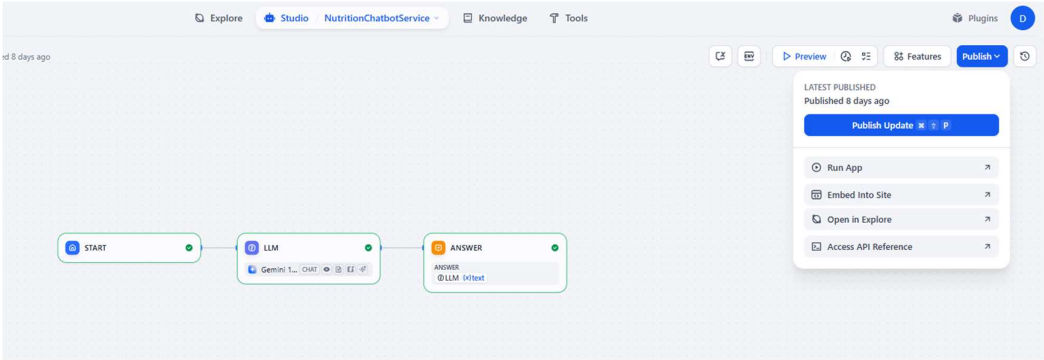



Figure 5.23 Publishing the Chatbot and Accessing API Reference

Once the instructions and model selection were verified, the workflow was published by clicking “Publish Update” (Figure 5.23). Publication generated a stable deployment and exposed the API Reference, which provided the request format, authentication scheme, and endpoint details necessary for integrating external applications through REST APIs.

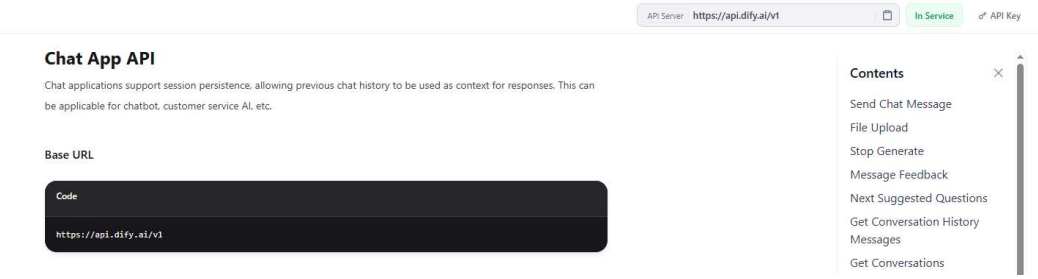



Figure 5.24 Dify API Key Usage Instructions

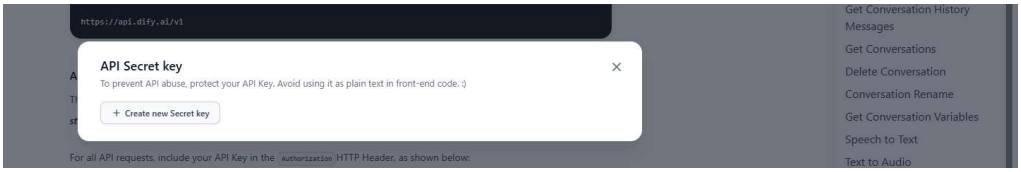



Figure 5.25 Generating a New Dify API Key

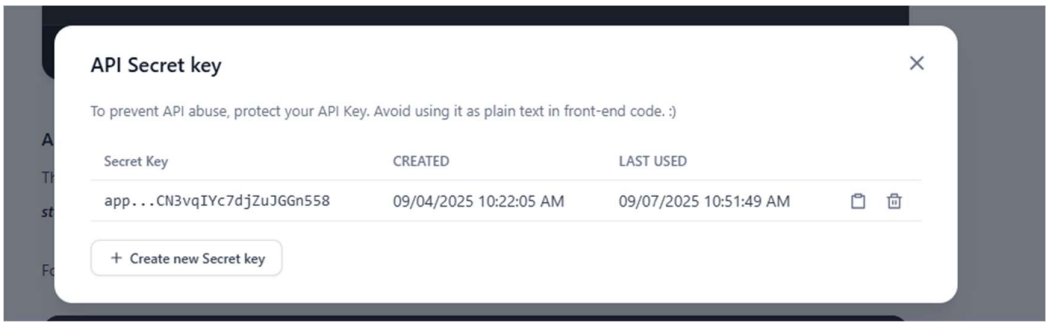



Figure 5.26 Copying the Generated API Key

The API reference provided clear guidelines for using the Dify API key to interact securely with the chatbot (Figure 5.24). It explained the necessary endpoint (https://api.dify.ai/v1) and how to format the Authorization header by including the Bearer token to maintain secure access.

Before actual API integration, it was necessary to generate a secret API key on the platform (Figure 5.25). This key acts as a credential to authenticate external requests made to the Dify server. Once the API key was generated, it was copied and prepared for integration into the application's source code (Figure 5.26).

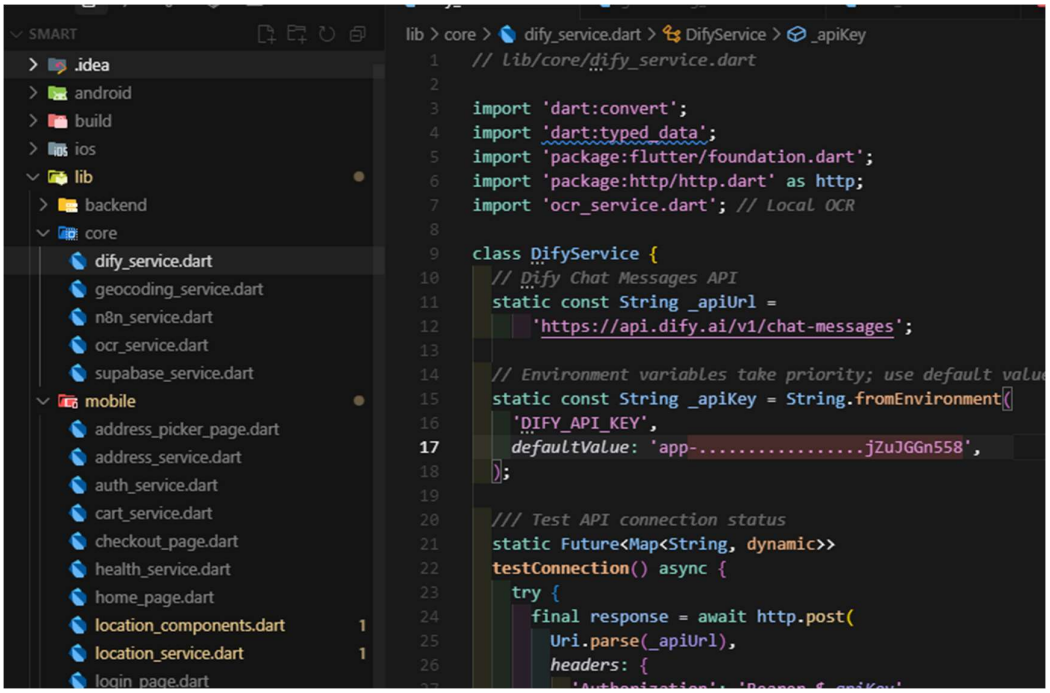



Figure 5.27 Connecting Flutter Application with Dify API

Within the Flutter project, the Dify API key was injected into the codebase as a compile-time value (for example, via String.fromEnvironment) and used when issuing HTTP requests (Figure 5.27). The application communicated with the Chat Messages endpoint (https://api.dify.ai/v1/chat-messages) using headers configured with Content-Type: application/json and Authorization: Bearer <API_KEY>. This setup enabled secure, programmatic interaction between the Flutter interface and the Dify AI backend while minimizing accidental exposure of sensitive credentials.

### 5.1.5 Google ML Kit & Google Maps Platform

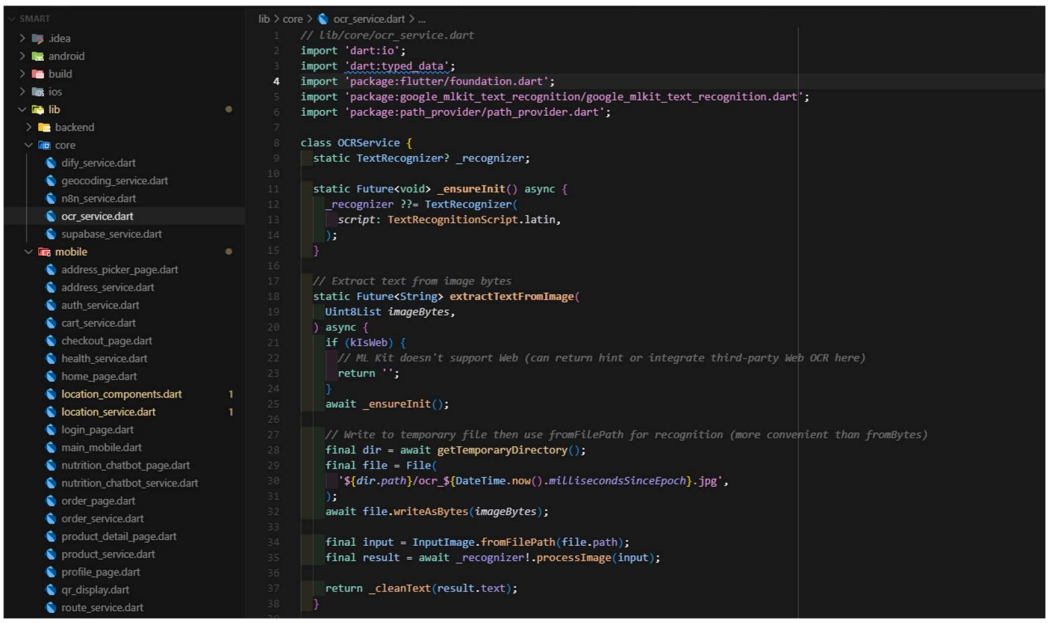



Figure 5.28 Local OCR with Google ML Kit

The application's OCR module was implemented using Google ML Kit Text Recognition. As shown in Figure 5.28, the OCRService class initialized a TextRecognizer with the Latin script and exposed an extractTextFromImage method that accepted Uint8List image bytes. For convenience and stability, the bytes were written to a temporary file (via path_provider) and processed as an InputImage.fromFilePath. The method then returned cleaned text for downstream use. A guard for kIsWeb was included because ML Kit did not support web targets; in such cases, the method exited early. This design enabled fast, on-device extraction while keeping the OCR boundary clearly separated from the rest of the app logic.

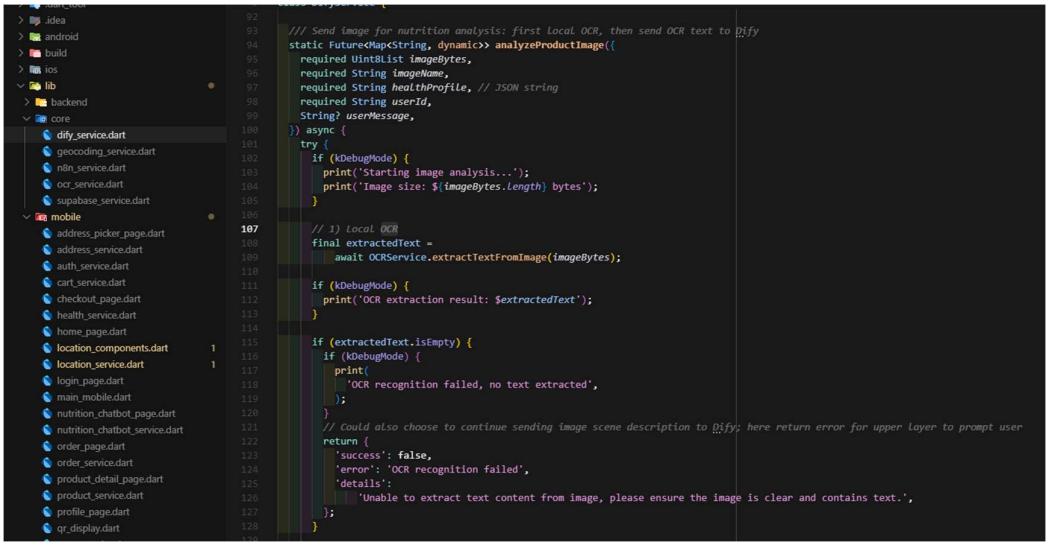



Figure 5.29 Orchestrating OCR and Analysis Pipeline

The end-to-end analysis flow was coordinated in DifyService.analyzeProductImage. As shown in Figure 5.29, the method received the image bytes and contextual parameters (image name, health profile JSON, user ID, and an optional user message). It first invoked OCRService.extractTextFromImage to perform local OCR. If no text was extracted, the method returned a structured failure response with a descriptive error, prompting the upper layer to request a clearer image. When text was available, the pipeline proceeded (subsequent lines not shown) to prepare the payload for the Dify Chat Messages API, sending the extracted nutrition text—rather than the raw image—to reduce bandwidth and limit exposure of sensitive data. This two-stage approach (local OCR → remote LLM analysis) improved responsiveness, privacy, and overall reliability.

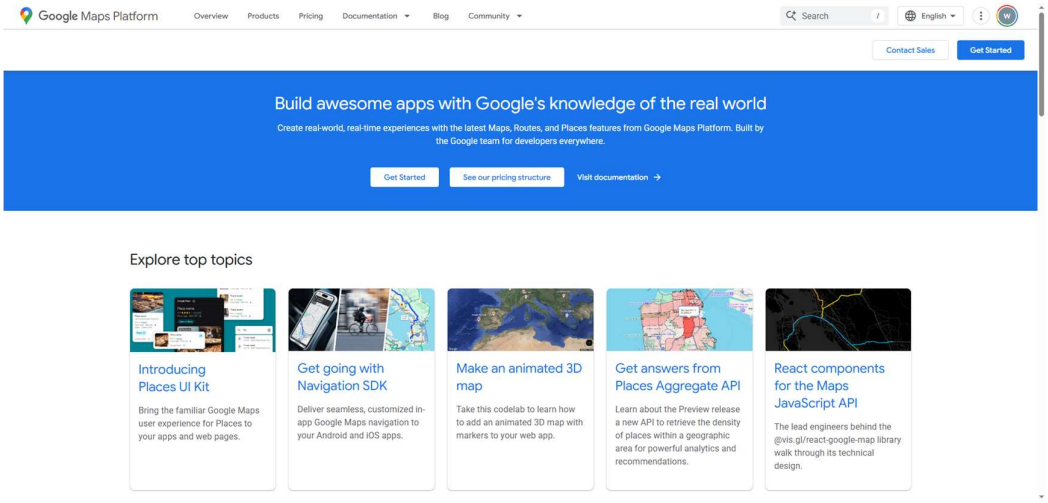



Figure 5.30 Google Maps Platform Landing Page

The integration began on the **Google Maps Platform** site, where documentation and product options were reviewed before proceeding with setup (Figure 5.30). From this page, the **Get Started** workflow was launched to link a Google Cloud project and prepare the required mapping and places services for the application.

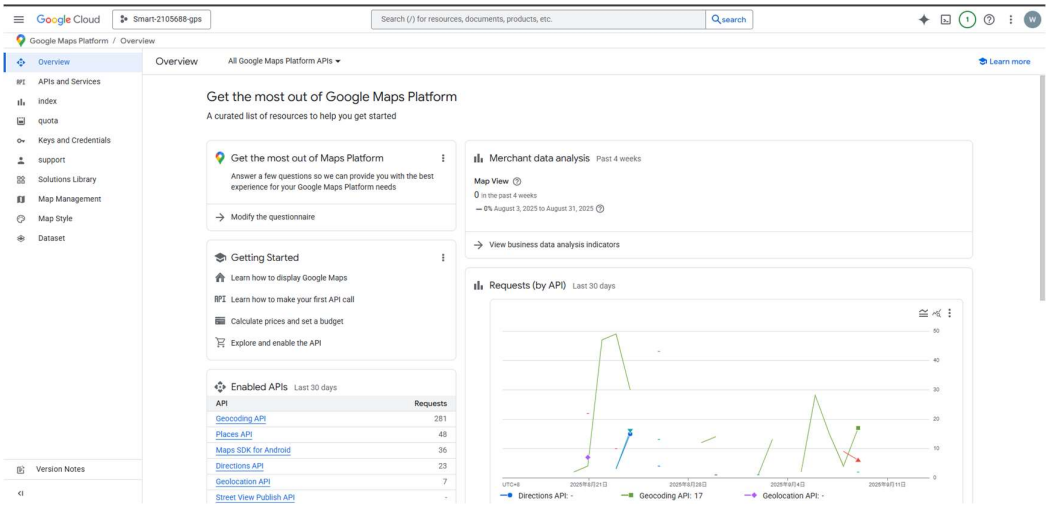



Figure 5.31 Google Maps Platform Overview & Enabled APIs

Within the Google Cloud Console, the Maps Platform → Overview dashboard summarized onboarding resources and usage metrics (Figure 5.31). The required services were then enabled for the project, including Geocoding API, Places API, Maps SDK for Android, Directions API, Geolocation API, and Street View Publish API. The dashboard subsequently listed these under Enabled APIs and displayed request activity for verification.

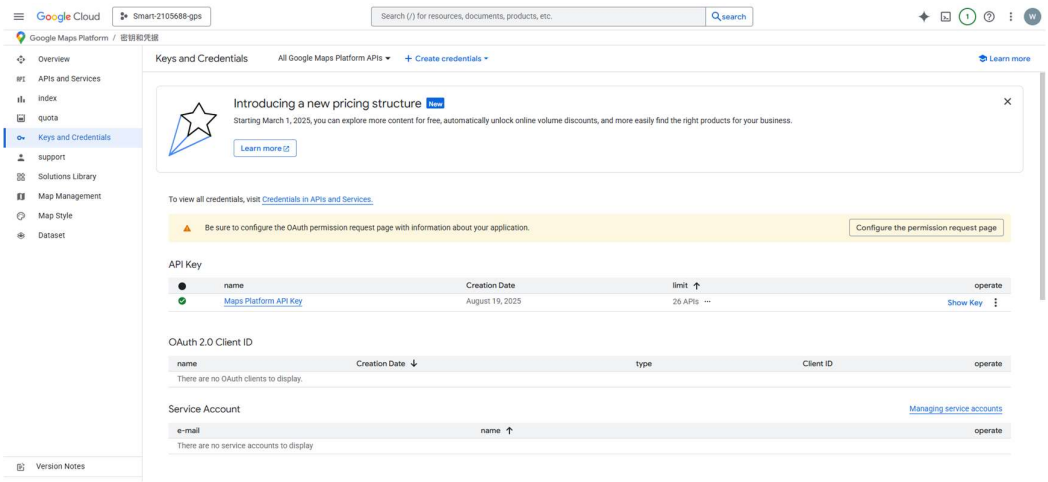



Figure 5.32 Keys and Credentials

Credentials were provisioned under Maps Platform → Keys and Credentials (Figure 5.32). An API key was created for the mobile application, with usage limits and key restrictions configured to protect the project (e.g., restricting allowed APIs and platform origins). The key was managed centrally so that rotation and auditing could be performed when necessary.

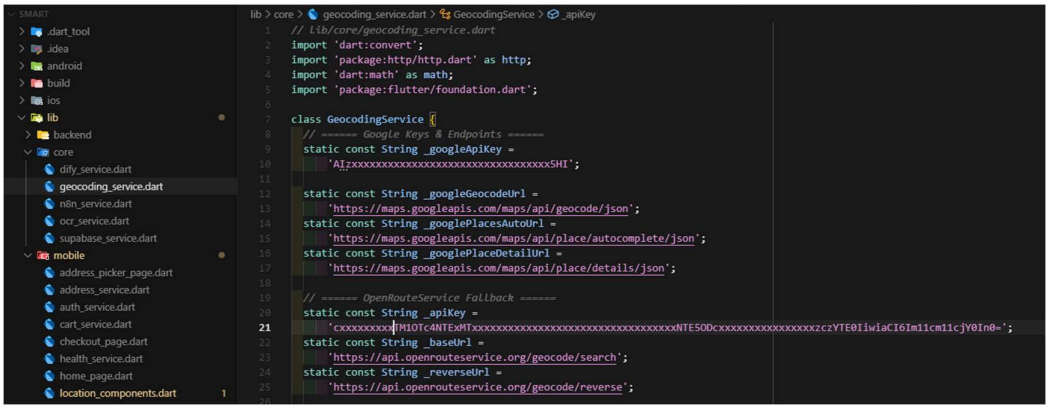
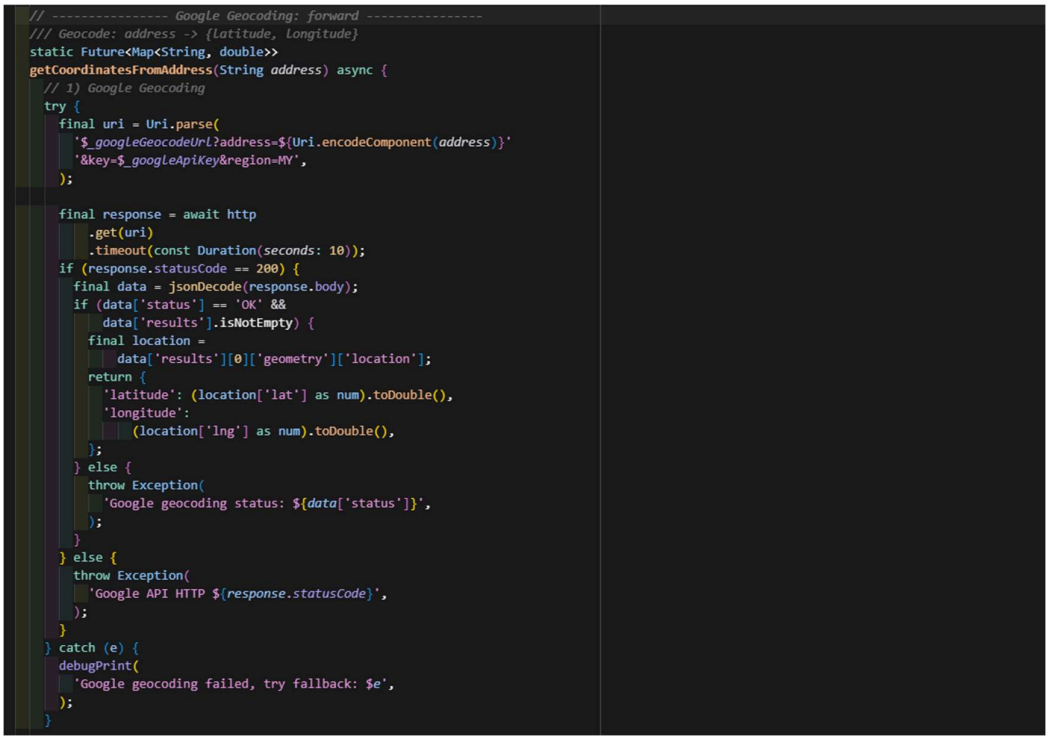



Figure 5.33 Centralizing Endpoints in Flutter

In the Flutter codebase, a GeocodingService class consolidated configuration for Google endpoints and the project API key (Figure 5.33). Constants were defined for Geocoding, Places Autocomplete, and Place Details URLs, while an OpenRouteService block was added as a fallback provider. This organization simplified maintenance and supported key injection through environment variables during build.



```dart
// ---------------- Google Geocoding: forward ----------------
/// Geocode: address -> {Latitude, Longitude}
static Future<Map<String, double>>
getCoordinatesFromAddress(String address) async {
  // 1) Google Geocoding
  try {
    final uri = Uri.parse(
      '$_googleGeocodeUrl?address=${Uri.encodeComponent(address)}'
      '&key=$_googleApiKey&region=MY',
    );

    final response = await http
        .get(uri)
        .timeout(const Duration(seconds: 10));
    if (response.statusCode == 200) {
      final data = jsonDecode(response.body);
      if (data['status'] == 'OK' &&
          data['results'].isNotEmpty) {
        final location =
            data['results'][0]['geometry']['location'];
        return {
          'latitude': (location['lat'] as num).toDouble(),
          'longitude':
              (location['lng'] as num).toDouble(),
        };
      } else {
        throw Exception(
          'Google geocoding status: ${data['status']}',
        );
      }
    } else {
      throw Exception(
        'Google API HTTP ${response.statusCode}',
      );
    }
  } catch (e) {
    debugPrint(
      'Google geocoding failed, try fallback: $e',
    );
  }
```

Figure 5.34 Forward Geocoding Implementation

Forward geocoding was implemented in getCoordinatesFromAddress, which constructed the Google Geocoding request using the encoded address, project API key, and the region=MY hint (Figure 5.34). The method executed an HTTP GET with a timeout, parsed the JSON response, and returned the first result's latitude and longitude when the API status was OK. Non-OK responses and network errors were surfaced as exceptions, and the catch block logged failures and prepared a fallback path to the secondary provider.

## 5.2 System Operation

### 5.2.1 Welcome, Login, Sign-up

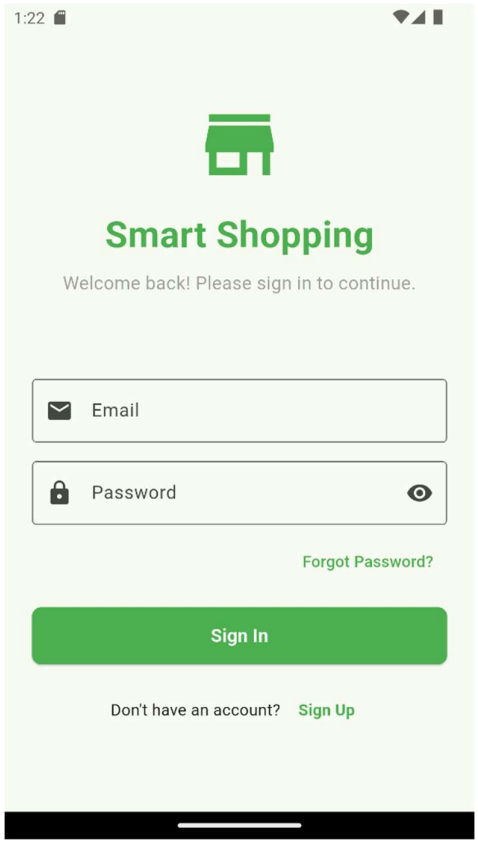



Figure 5.35 Welcome Page and Login Page

Figure 5.35 illustrated the Welcome Page of the Smart Shopping Assistant mobile application. The page was designed to provide users with access to the system through secure authentication. It displayed the application logo and name at the top, followed by a welcoming message prompting users to sign in. Two input fields were provided for entering an email address and a password, with an additional option to reveal or hide the password for verification purposes. A “Forgot Password?” link was included to facilitate password recovery. The primary “Sign In” button allowed users to log in to the application, while a secondary link at the bottom redirected new users to the registration process through the “Sign Up” option.

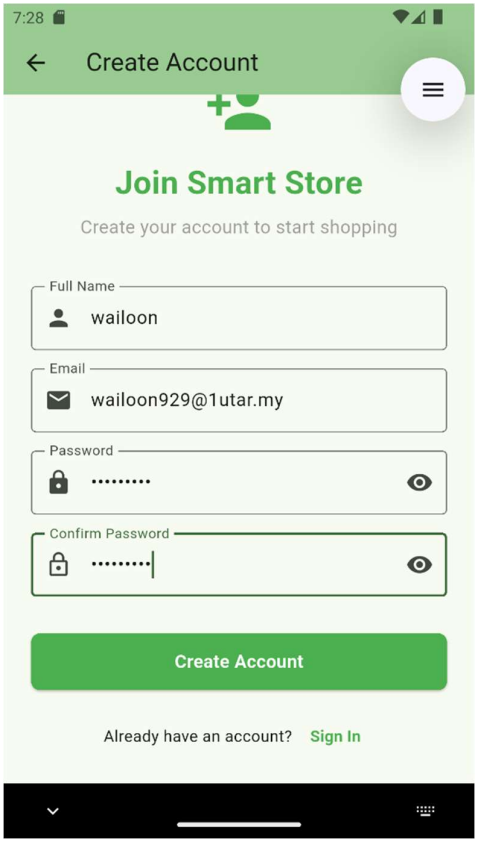



Figure 5.36 Sign Up Page

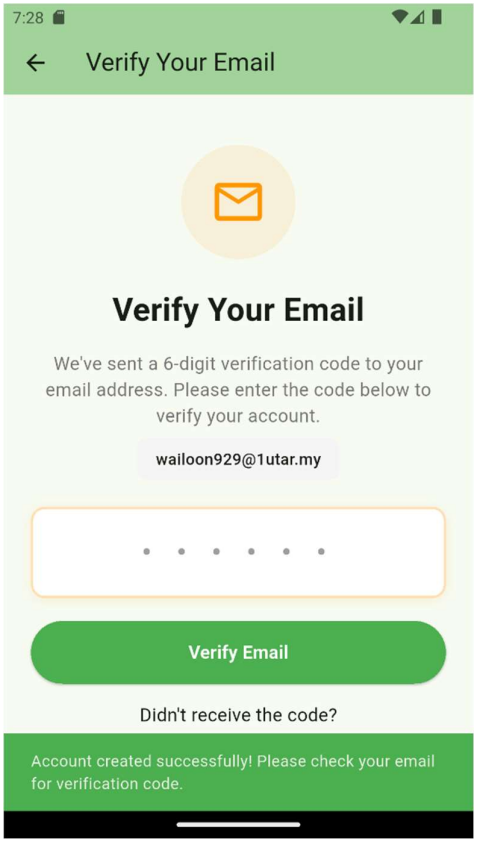



Figure 5.37 Email Verification Page

Figure 5.36 presented the Sign-Up Page of the Smart Store mobile application. This interface was designed to facilitate new user registration. It included input fields for the user's full name, email address, password, and confirmation of the password to ensure accuracy. A "Create Account" button was provided to complete the registration process. Additionally, a link was displayed at the bottom of the page to redirect existing users back to the Sign In page. Figure 5.37 illustrated the Email Verification Page of the Smart Store mobile application. This page prompted users to verify their newly created account through a six-digit verification code sent to the registered email address. The page displayed the user's email address and provided a field for entering the verification code. A "Verify Email" button was placed below the input field to confirm the verification process. Furthermore, the interface included a message confirming successful account creation and guidance for users who did not receive the verification code.

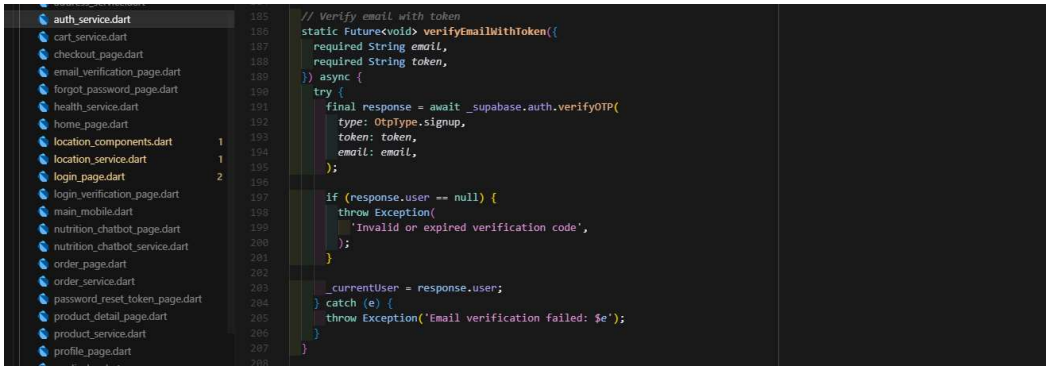



Figure 5.38 Email Verification Code Implementation

Figure 5.38 demonstrated the implementation of the email verification code function within the authentication service of the Smart Store application. The function verifyEmailWithToken was defined to verify a newly registered account by validating the one-time password (OTP) sent to the user's email address. The implementation utilized the Supabase authentication service to perform verification, where an exception was thrown if the token was invalid or expired. Upon successful validation, the current user session was updated accordingly.


Figure 5.39 Login Verification Code Implementation

Figure 5.39 illustrated the login verification code implementation in the authentication service of the Smart Store application. The function verifyLoginWithToken was developed to validate login attempts using a one-time token, supporting secure login processes such as passwordless authentication or two-factor login. Like the email verification function, the system checked the token validity through Supabase authentication. If the token was invalid or expired, an exception was raised, while a successful response updated the current user session. Additional logic was included to check whether the user's email was verified and to expose the authentication state stream for real-time status tracking.

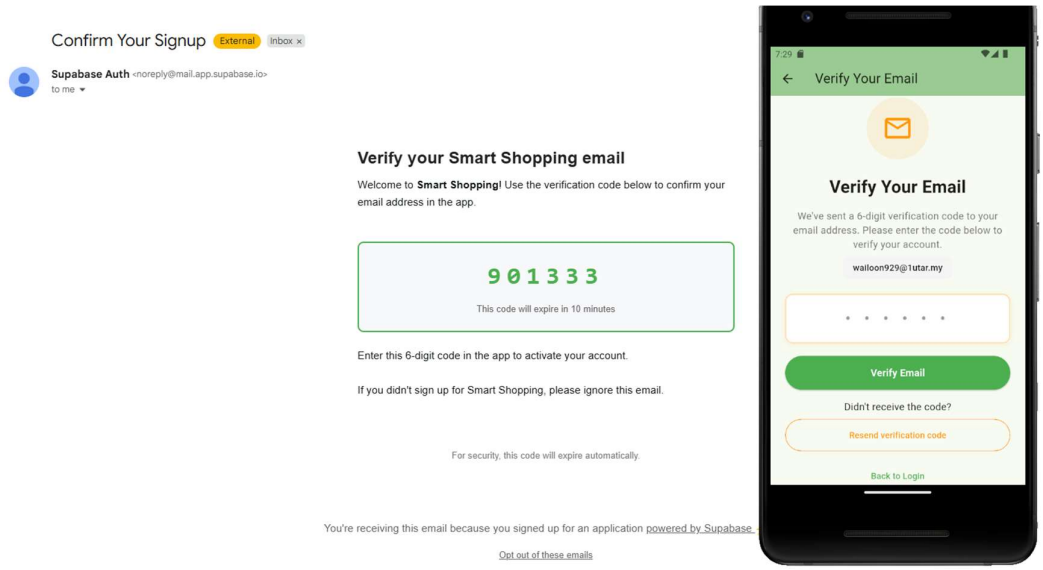



Figure 5.40 Email Verification Process

Figure 5.40 illustrated the email verification process in the Smart Shopping application. When a new account was created, a six-digit verification code was automatically sent to the user's registered email address by the Supabase authentication service. The email contained instructions for entering the code into the application, with a note that the code would expire after ten minutes for security purposes. On the mobile application interface, users were prompted to input the received verification code into the provided field. The page included a "Verify Email" button to complete the verification process, along with options to resend the code or return to the login page if required. This process ensured that only valid and accessible email addresses were used for account registration, thereby enhancing account security and authenticity.

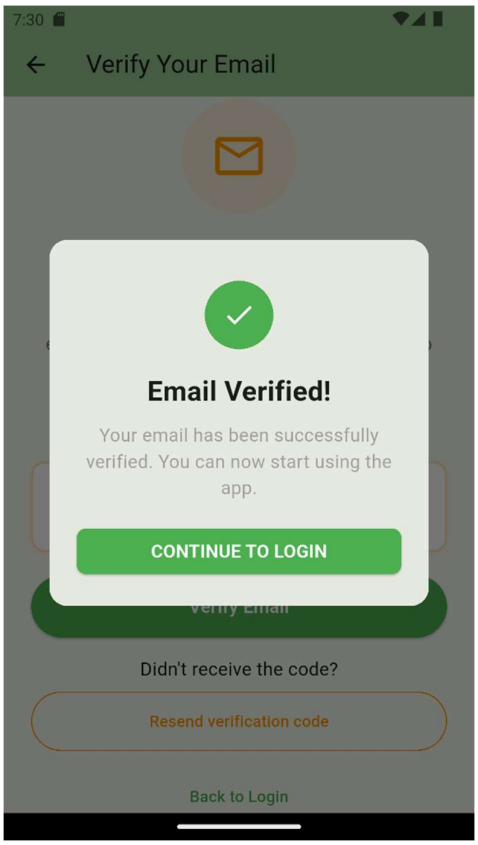



Figure 5.41 Email Verified Page

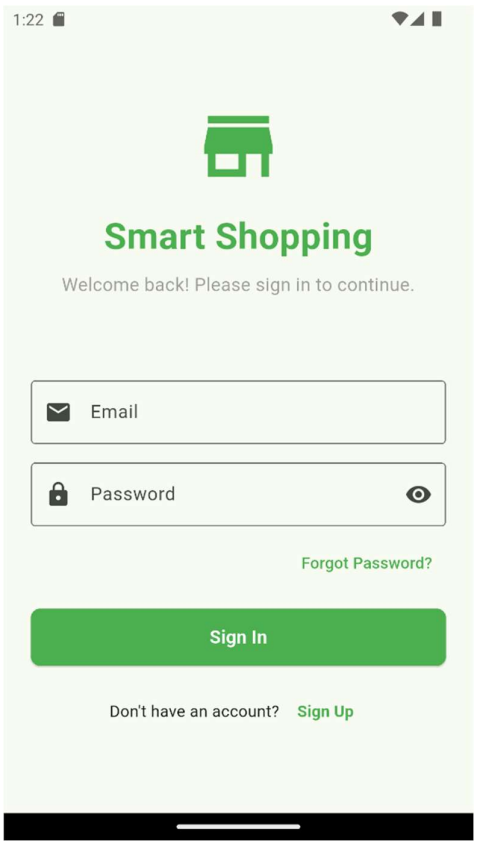



Figure 5.42 Login Page

Figure 5.41 showed the Email Verified Page of the Smart Shopping application. After the verification code was successfully entered, the system displayed a confirmation message indicating that the email address had been verified. The page included a green checkmark icon and a message stating that the verification process was completed. A "Continue to Login" button was provided to redirect users back to the login interface, allowing them to proceed with accessing their accounts. Figure 5.42 presented the Login Page of the Smart Shopping application. This page enabled registered users to access their accounts securely by entering their email address and password. A visibility toggle was provided for the password input field to reduce input errors. Additionally, the interface included a "Forgot Password?" link to initiate password recovery and a "Sign Up" link for new users who had not yet created an account. The central "Sign In" button was prominently displayed to authenticate users and grant them access to the application's features.

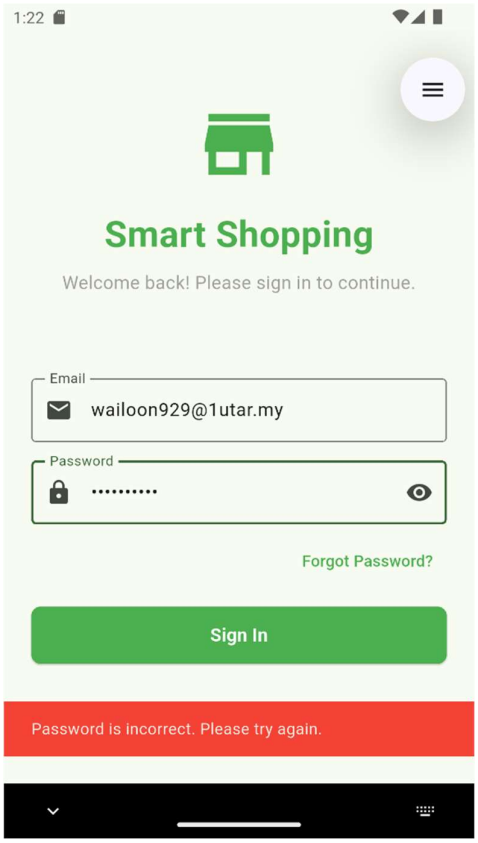



Figure 5.43 Invalid Login Attempt Page

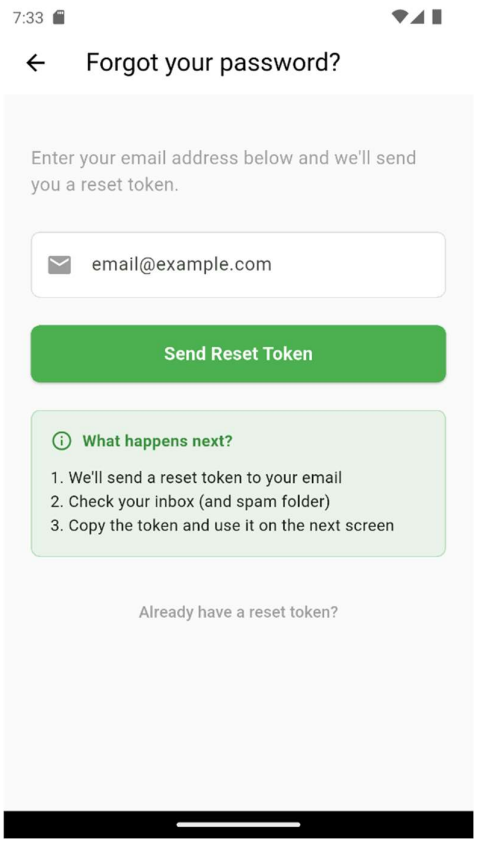



Figure 5.44 Forgot Password Page

Figure 5.43 illustrated the Invalid Login Attempt Page of the Smart Shopping application. When a user entered an incorrect email address or password, the system displayed an error message highlighted in red at the bottom of the page, notifying the user that the login attempt was unsuccessful. This feature was implemented to enhance security and provide immediate feedback, prompting users to re-enter the correct credentials or utilize the “Forgot Password?” option if necessary. Figure 5.44 presented the Forgot Password Page of the Smart Shopping application. This interface allowed users to recover account access by requesting a password reset. Users were instructed to enter their registered email address, after which a reset token was sent to their inbox. This process ensured a secure and user-friendly method for handling forgotten passwords.

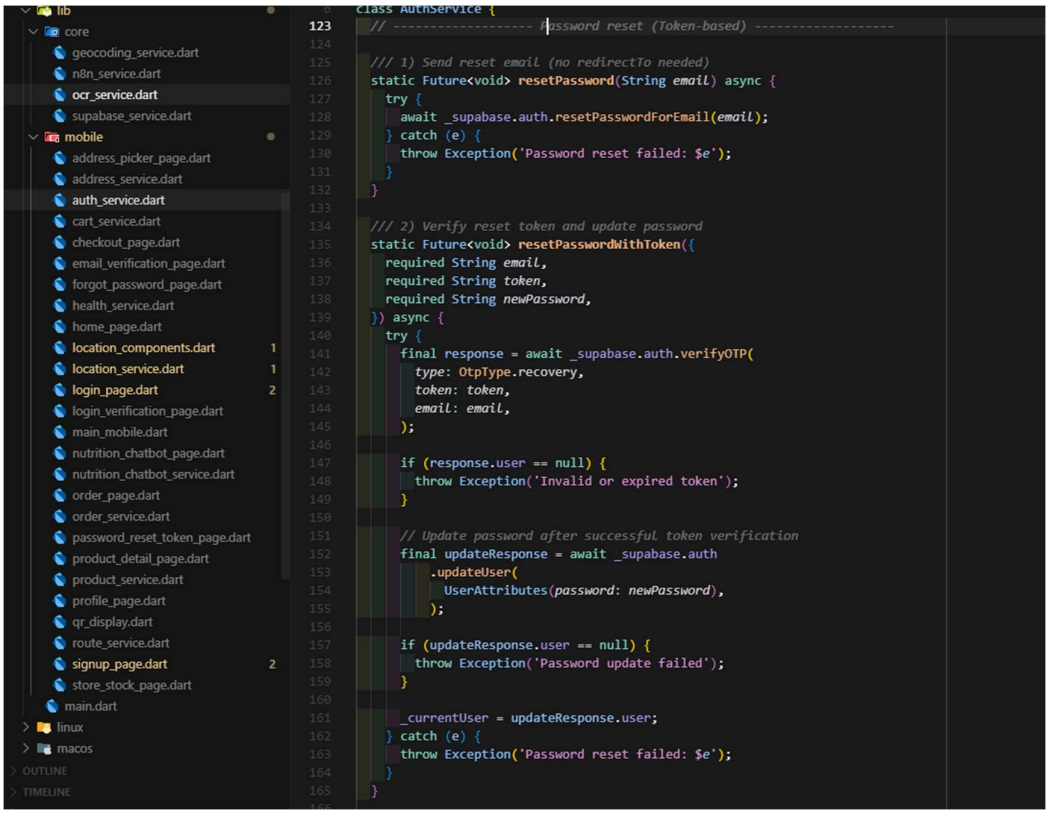



Figure 5.45 Password Reset Code Implementation

Figure 5.45 demonstrated the implementation of the password reset functionality within the authentication service of the Smart Shopping application. The code included two key methods: resetPassword and resetPasswordWithToken. The resetPassword method was responsible for sending a password reset email to the user's registered email address. The resetPasswordWithToken method handled the verification of the reset token and the subsequent update of the password. The process utilized Supabase authentication services, where invalid or expired tokens triggered exception handling to ensure security. Once the token was verified, the new password was updated in the system, and the current user session was refreshed. This implementation ensured a secure and reliable mechanism for account recovery.

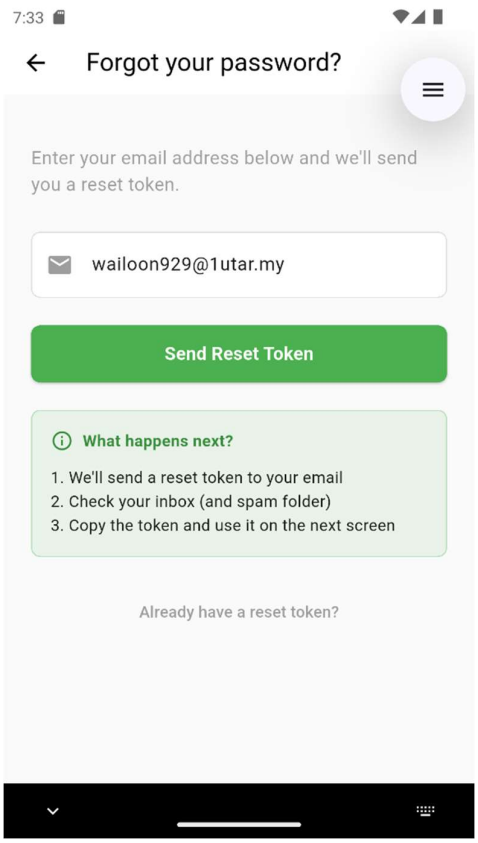



Figure 5.46 Send Reset Token Page

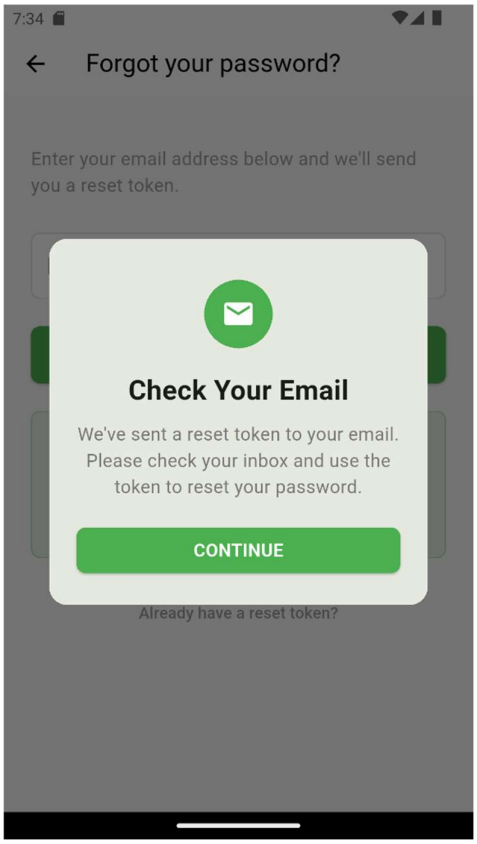



Figure 5.47 Check Your Email Page

Figure 5.46 displayed the Send Reset Token Page of the Smart Shopping application. This page allowed users to initiate the password recovery process by entering their registered email address. Once submitted, a reset token was sent to the specified email. The interface also included an instructional box that outlined the subsequent steps, namely checking the inbox or spam folder, retrieving the reset token, and using it in the following screen. This ensured that users were clearly guided throughout the reset process. Figure 5.47 illustrated the Check Your Email Page of the Smart Shopping application. After a reset token was successfully requested, a confirmation message was displayed, instructing the user to check their email inbox for the token. The page reinforced the recovery process by highlighting the need to retrieve the token before proceeding. A “Continue” button was provided to guide users to the next stage, where the reset token would be entered and validated.

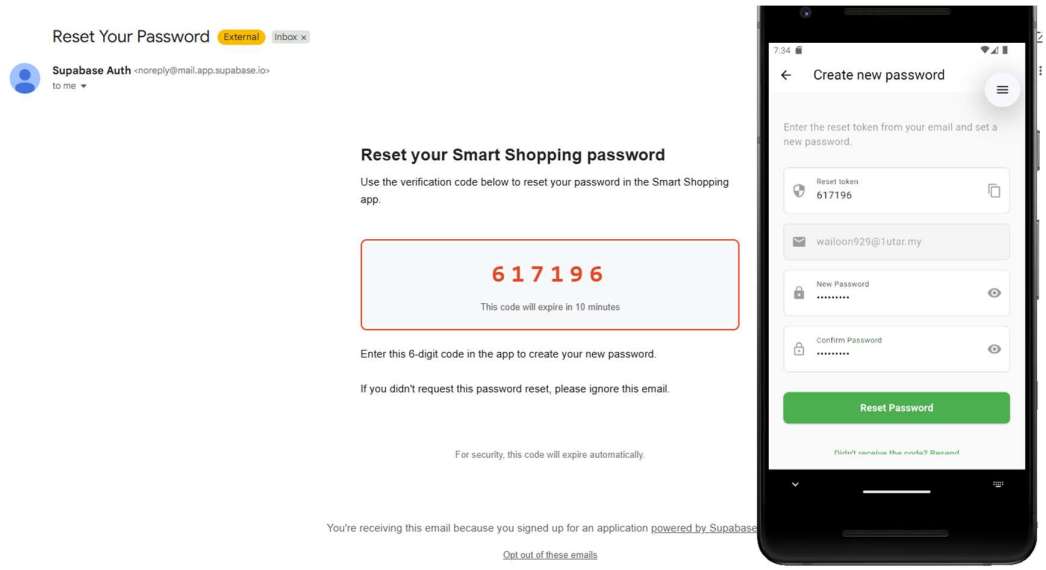



Figure 5.48 Token for Reset Password send by Email and Create New Password Page

Figure 5.48 illustrated the process of resetting a password in the Smart Shopping application. On the left, the system-generated email contained a six-digit reset token sent by the Supabase authentication service. This token was required to verify the user's identity and was valid for ten minutes to enhance security. On the right, the mobile interface displayed the Create New Password Page, where the user was prompted to enter the received token along with their registered email address. Two additional fields were provided to input and confirm the new password. A "Reset Password" button was included to complete the process, enabling the user to securely update their login credentials.

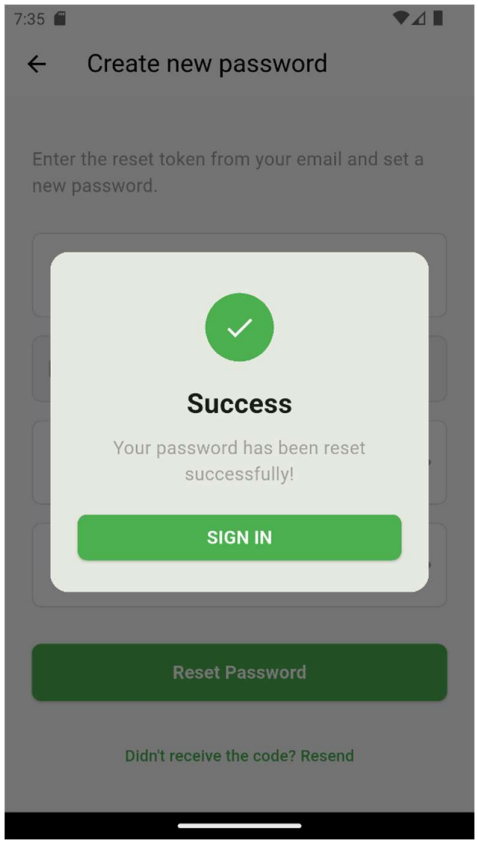



Figure 5.49 Password Reset Success Page

Figure 5.49 showed the Password Reset Success Page of the Smart Shopping application. After a valid reset token was entered and a new password was successfully created, the system displayed a confirmation message to notify the user that the password reset process had been completed. A green checkmark icon reinforced the success status, while a “Sign In” button was provided to redirect the user back to the login page. This confirmation ensured that the recovery process was clearly communicated and that users could immediately proceed to access their accounts with the updated credentials.

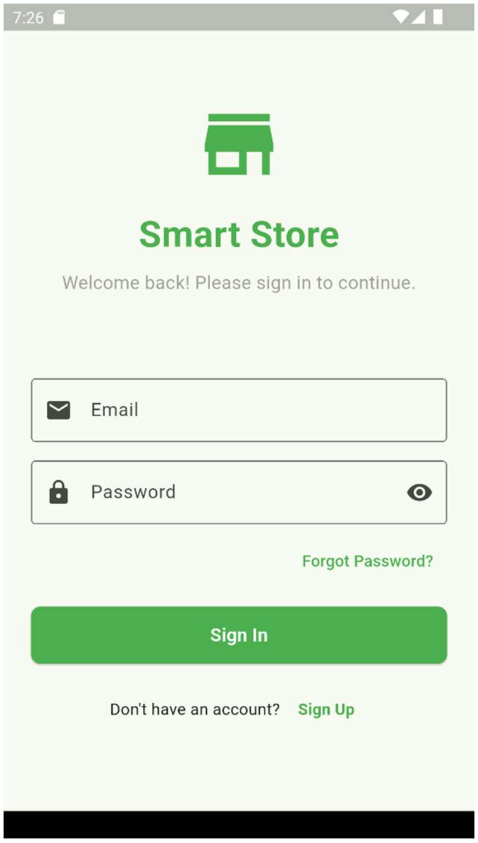



Figure 5.50 Redirection to Login Page

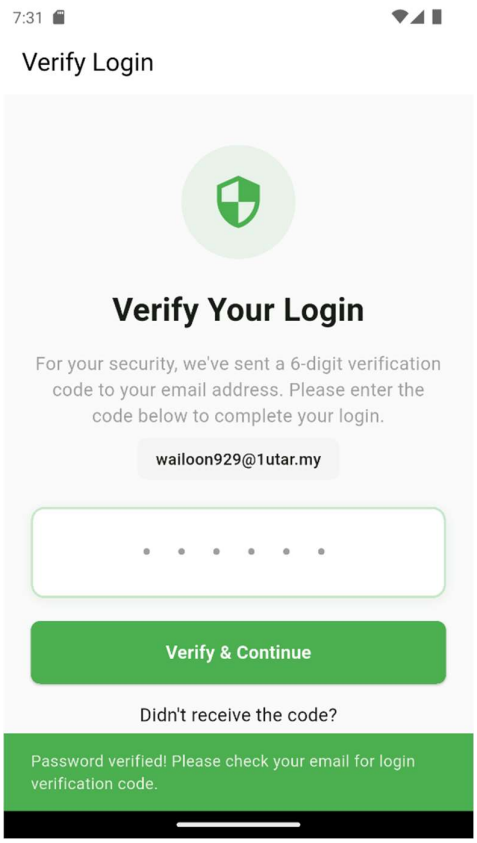



Figure 5.51 Login Verification Page

Figure 5.50 displayed the Redirection to Login Page of the Smart Shopping application. After completing previous authentication or recovery actions, users were redirected to zthis interface to access their accounts. The page required the input of an email address and password, with an option to toggle password visibility. Additional links for password recovery and new account registration were also provided, while the central "Sign In" button initiated the authentication process. Figure 5.51 illustrated the Login Verification Page of the Smart Shopping application. As part of the security process, the system required users to verify their login by entering a six-digit verification code sent to their registered email address. The interface displayed the user's email and provided a field for code input. A "Verify & Continue" button confirmed the login attempt, while a link was included to resend the verification code if necessary. This step added an additional layer of security to ensure account protection.

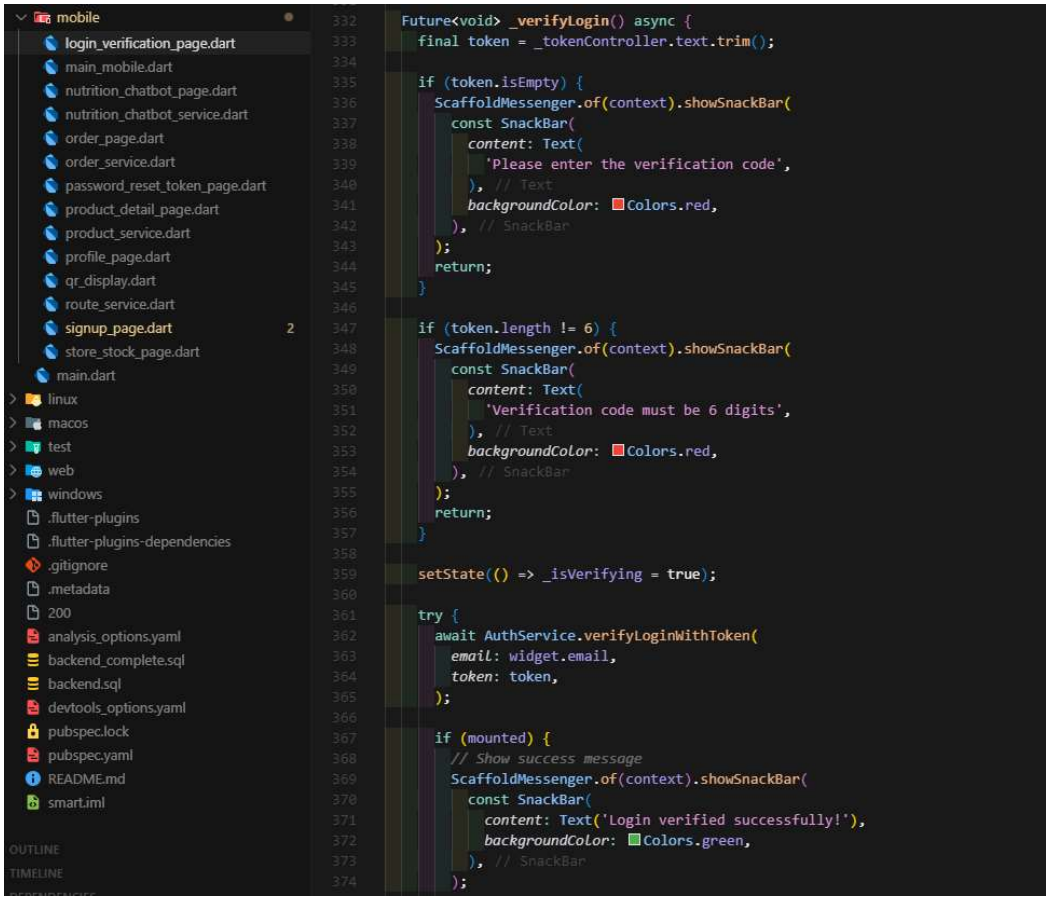



Figure 5.52 Verify Login Token Code Implementation

Figure 5.52 illustrated the Verify Login Token Code Implementation within the Smart Shopping application. The verifyLogin method validated the user's six-digit verification code. The implementation first checked whether the token was empty or invalid in length, providing immediate feedback to the user via error messages. If valid, the token was passed to the authentication service for verification. Upon success, a confirmation message was displayed to notify the user that the login had been verified successfully. Exception handling and user feedback mechanisms were integrated to ensure both security and usability throughout the login verification process.


Figure 5.53 Send Login Verification Token Code Implementation

Figure 5.53 presented the Send Login Verification Token Code Implementation in the Smart Shopping application. The method sendLoginVerificationToken was implemented to generate and send a one-time verification code to the user's registered email address using the Supabase authentication service. Exception handling was included to manage cases where the token could not be sent, thereby ensuring system reliability and user awareness during the login verification process.


Figure 5.54 Login Verification Email and In-App Verification Page

Figure 5.54 illustrated the login verification process through both email and in-app validation in the Smart Shopping application. On the left, the system-generated email from the Supabase authentication service contained a six-digit verification code, valid for ten minutes, to enhance security. The page displayed the registered email address, an input field for the verification code, and a "Verify & Continue" button to confirm access. This dual-step mechanism ensured secure authentication by combining email-based verification with in-app confirmation.

### 5.2.2 Home Page and Product Page

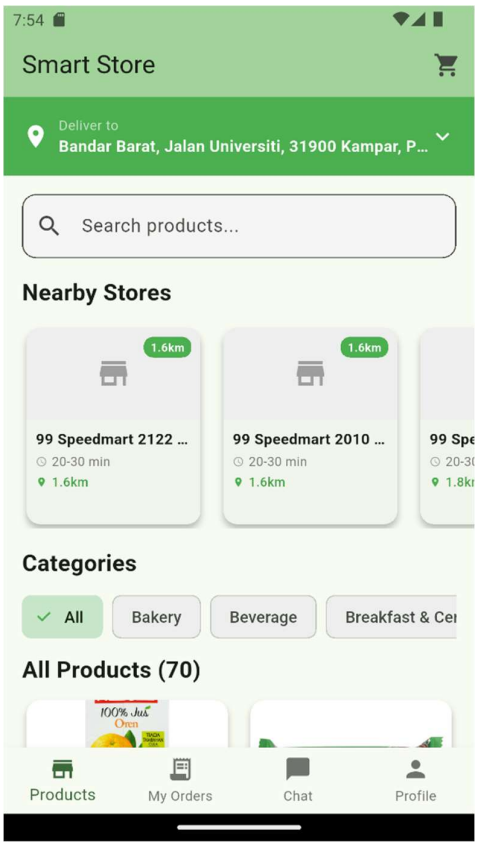
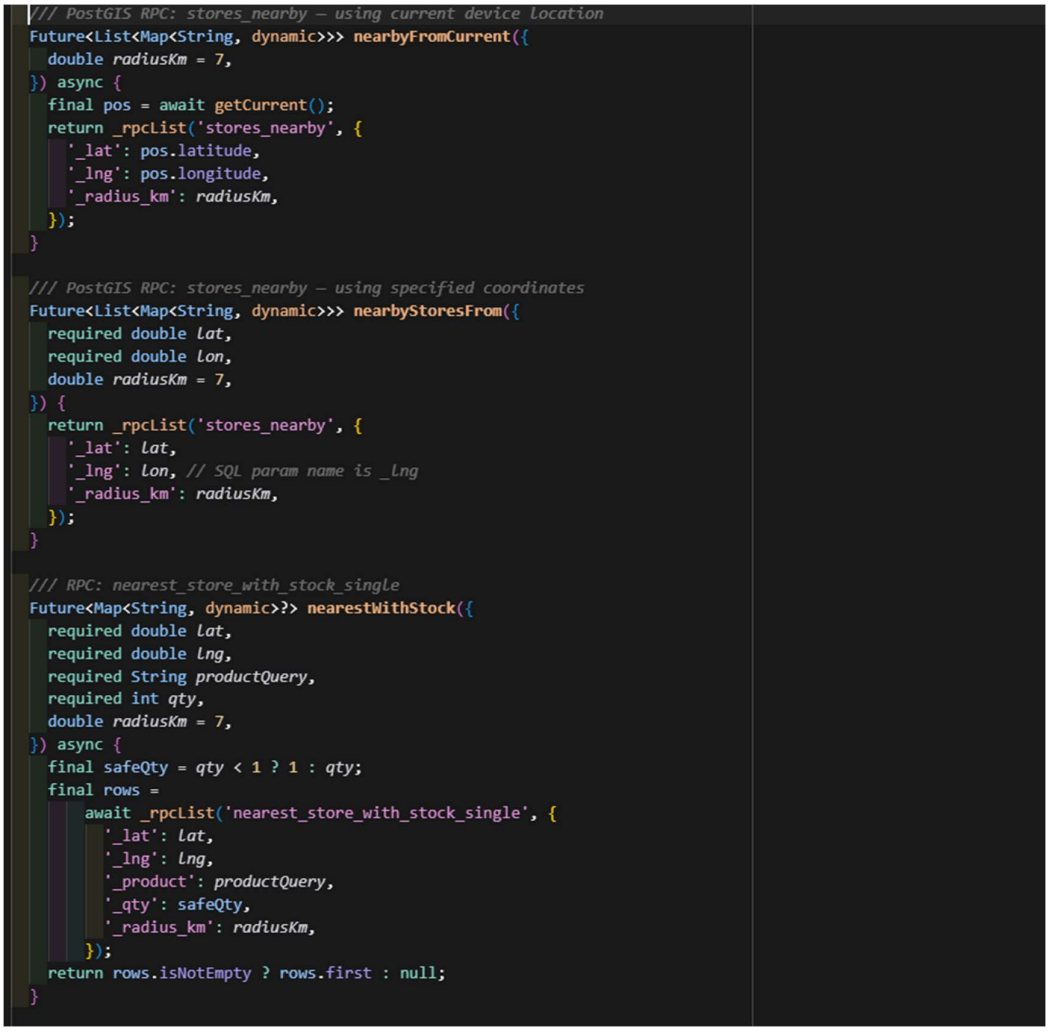



Figure 5.55 Home Page

Figure 5.55 illustrated the Home Page of the Smart Shopping application. This interface served as the central entry point for browsing available stores and products. At the top, the user's delivery address was displayed with an option to update the location. Below, a search bar allowed users to quickly find products. The "Nearby Stores" section presented a list of stores with details such as distance and estimated delivery time. Categories were displayed beneath for filtering, followed by a list of products with images and details. The bottom navigation bar provided quick access to Products, My Orders, Chat, and Profile.


Figure 5.56 Location Service Code Implementation

Figure 5.56 demonstrated the Location Service Code Implementation used in the Smart Shopping application. The method getCurrent() was implemented to retrieve the user's current geographical coordinates. The function included checks for location service availability, permission requests, and fallbacks to last known location if available. Exception handling was used to manage cases of denied permissions or disabled services. When valid, the method requested a high-accuracy real-time position, ensuring precise location data for store and delivery operations.



```dart
/// PostGIS RPC: stores_nearby — using current device location
Future<List<Map<String, dynamic>>> nearbyFromCurrent({
  double radiusKm = 7,
}) async {
  final pos = await getCurrent();
  return _rpcList('stores_nearby', {
    '_lat': pos.latitude,
    '_lng': pos.longitude,
    '_radius_km': radiusKm,
  });
}

/// PostGIS RPC: stores_nearby — using specified coordinates
Future<List<Map<String, dynamic>>> nearbyStoresFrom({
  required double lat,
  required double lon,
  double radiusKm = 7,
}) {
  return _rpcList('stores_nearby', {
    '_lat': lat,
    '_lng': lon, // SQL param name is _lng
    '_radius_km': radiusKm,
  });
}

/// RPC: nearest_store_with_stock_single
Future<Map<String, dynamic>?> nearestWithStock({
  required double lat,
  required double lng,
  required String productQuery,
  required int qty,
  double radiusKm = 7,
}) async {
  final safeQty = qty < 1 ? 1 : qty;
  final rows =
      await _rpcList('nearest_store_with_stock_single', {
        '_lat': lat,
        '_lng': lng,
        '_product': productQuery,
        '_qty': safeQty,
        '_radius_km': radiusKm,
      });
  return rows.isNotEmpty ? rows.first : null;
}
```

Figure 5.57 Nearby Stores and Stock Checking Code Implementation

Figure 5.57 presented the Nearby Stores and Stock Checking Code Implementation. The methods nearbyFromCurrent and nearbyStoresFrom retrieved nearby store data either using the device's current coordinates or specified coordinates, respectively. Both functions used PostGIS remote procedure calls (RPC) to fetch results based on latitude, longitude, and search radius. The method nearestWithStock further extended functionality by checking for stores with available stock for a specified product. These implementations ensured accurate store suggestions and real-time stock availability checks.

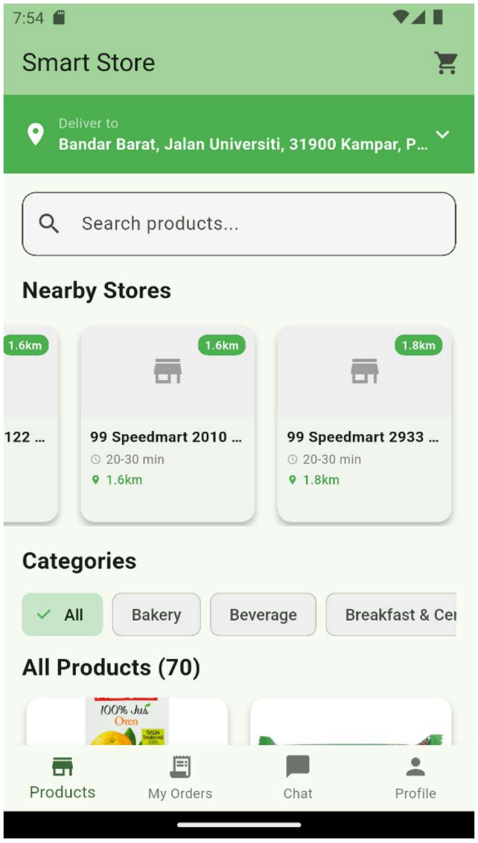



Figure 5.58 Select Category Page

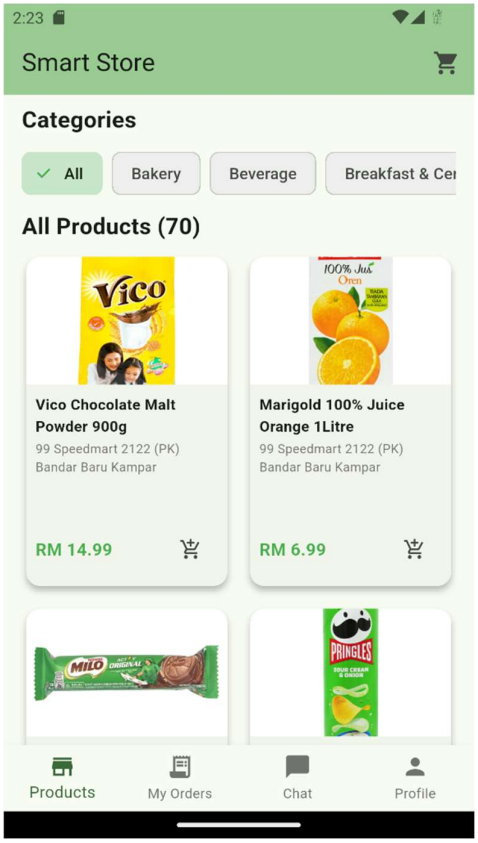



Figure 5.59 Product Viewing

Figure 5.58 displayed the Select Category Page of the Smart Shopping application. Users could filter products by selecting predefined categories such as Bakery, Beverage, or Breakfast & Cereal. The interface combined store information and product filtering options, helping users refine their search within nearby stores. The page also allowed switching between "All" products and specific categories for a more tailored browsing experience. Figure 5.59 illustrated the Product Viewing Page of the Smart Shopping application. This interface presented detailed product listings with names, images, store information, and prices. Each product card included an "Add to Cart" icon to simplify the shopping process. The design ensured clarity by grouping products by store and enabling easy comparison. This page was essential for providing users with a seamless browsing and selection experience.

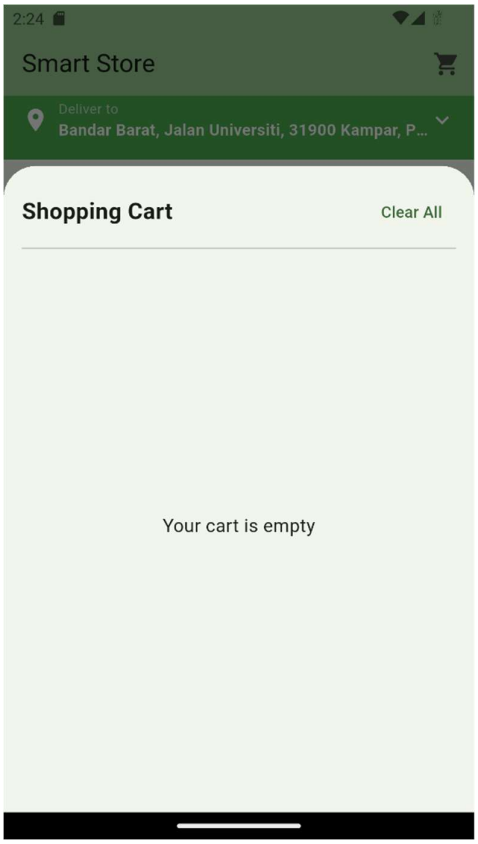



Figure 5.60 Empty Shopping Cart Page

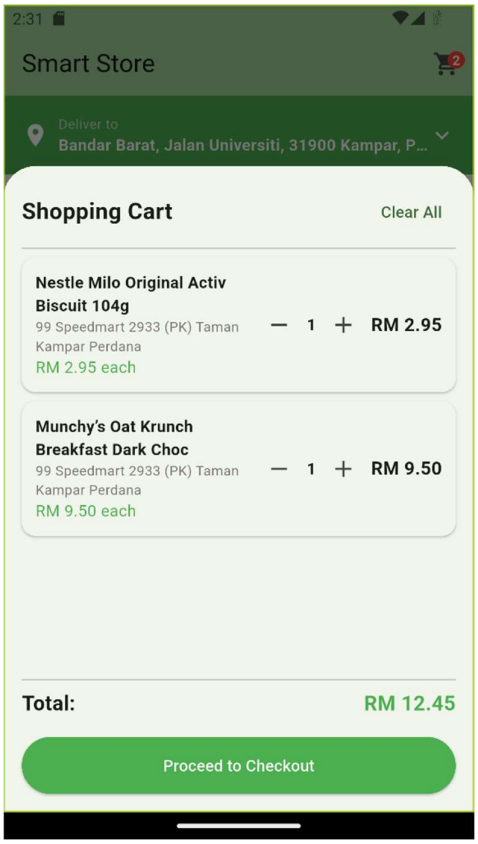



Figure 5.61 Filled Shopping Cart Page

Figure 5.60 illustrated the Empty Shopping Cart Page of the Smart Shopping application. When no items had been added, the cart displayed a simple message indicating that it was empty. The page also included a "Clear All" option at the top, allowing users to reset the cart in case any items were previously stored. This design provided clarity to the user while maintaining a minimal interface when no products were selected. Figure 5.61 presented the Filled Shopping Cart Page of the Smart Shopping application. This interface displayed a list of items selected by the user, including product names, quantities, store details, and individual prices. Increment and decrement buttons allowed quantity adjustments directly within the cart. At the bottom, the total cost was calculated automatically, and a "Proceed to Checkout" button was provided to move forward with the purchase. This feature streamlined the shopping process by summarizing selected items and enabling efficient cart management.

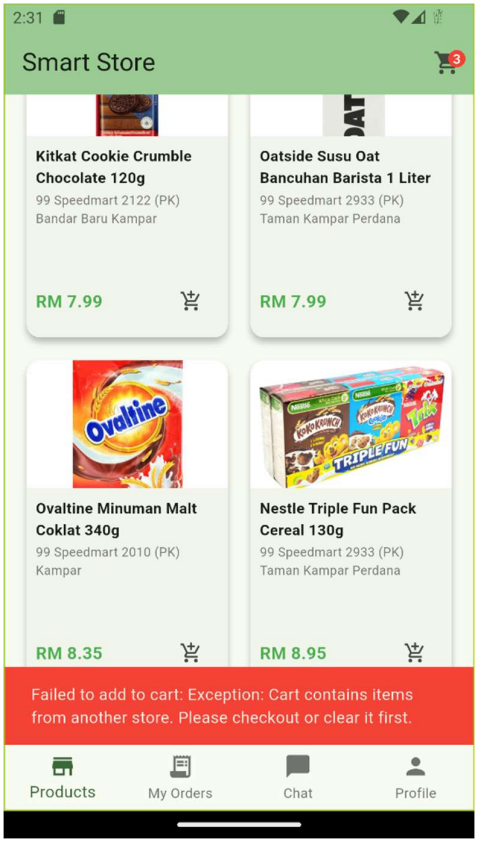



Figure 5.62 Failed Add to Cart Notification

Figure 5.62 illustrated the Failed Add to Cart Notification in the Smart Shopping application. This error occurred when a user attempted to add products from multiple stores into the same shopping cart. A red notification banner appeared at the bottom of the page, informing the user that the cart already contained items from another store and prompting them to either complete the current checkout or clear the cart before adding new items. This feature was implemented to maintain order consistency by restricting each cart to a single store.

```
Future<void> _addToCart(Map<String, dynamic> p) async {
  try {
    // Build a Product model from the fetched map
    final product = Product.fromMap(p);
    await CartService().addItem(
      product,
      quantity: quantity,
    );

    if (!mounted) return;
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text(
          'Added $quantity × ${p['product_name'] ?? 'item'} to cart',
        ),
        backgroundColor: Colors.green,
      ),
    );
  } catch (e) {
    if (!mounted) return;
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('Failed to add to cart: $e'),
        backgroundColor: Colors.red,
      ),
    );
  }
}
```

Figure 5.63 Add to Cart Code Implementation

Figure 5.63 demonstrated the Add to Cart Code Implementation. The addToCart method handled the process of adding items to the shopping cart. The function constructed a product object from the provided map and added it using the CartService. User feedback was delivered through a SnackBar notification, which displayed a success message when an item was added successfully or an error message if the operation failed. Exception handling ensured robust error management and improved user experience by clearly communicating the outcome of each action.

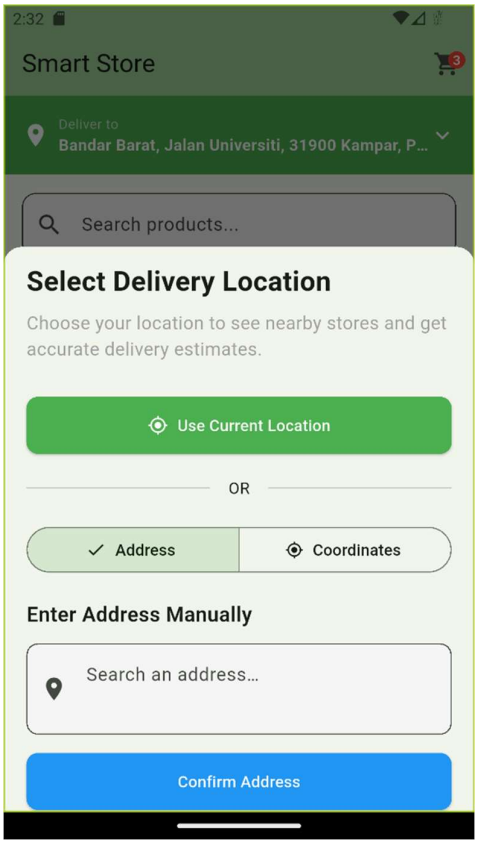



Figure 5.64 Select Delivery Location Page

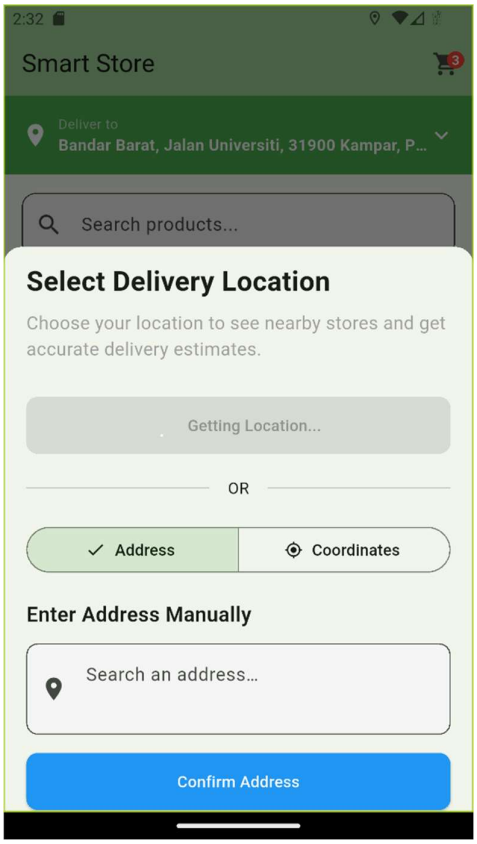



Figure 5.65 Getting Current Location Page

Figure 5.64 illustrated the Select Delivery Location Page of the Smart Shopping application. This page allowed users to define their preferred delivery location by either using their current device location or entering an address manually. The interface provided options to input by address or geographic coordinates, with a “Confirm Address” button to proceed once details were entered. This design ensured flexibility in location selection for accurate delivery estimates. Figure 5.65 presented the Getting Current Location Page. When the user selected the option to use the current device location, the system initiated the process of retrieving GPS coordinates. The button displayed a loading state, informing the user that the location was being obtained. This feature enhanced usability by providing real-time feedback during the retrieval process.

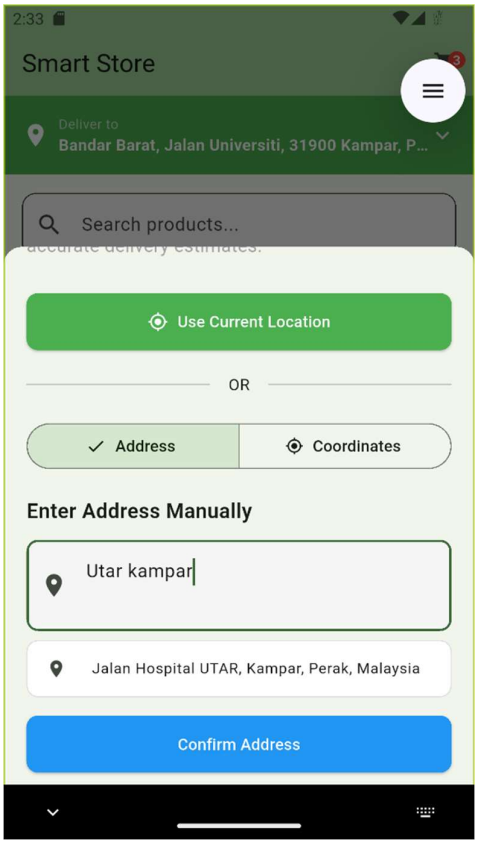



Figure 5.66 Manually Input Address Page

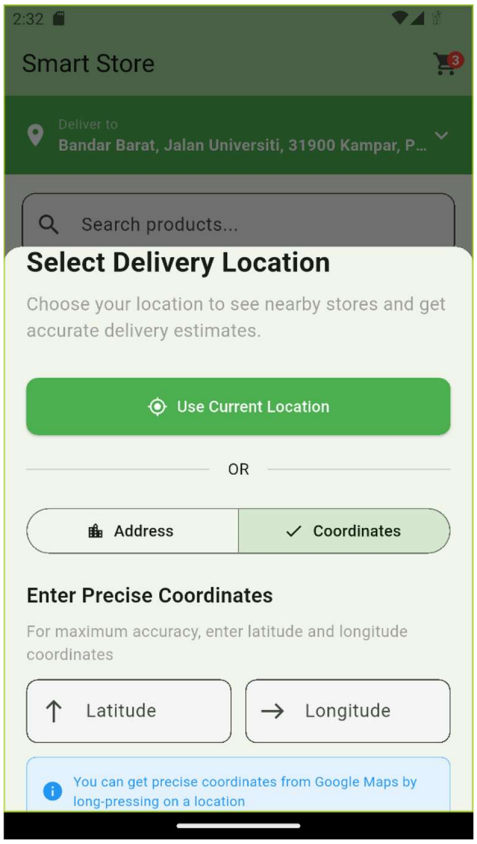



Figure 5.67 Manually Input Coordinates Page

Figure 5.66 showed the Manually Input Address Page. In cases where users preferred to type their address instead of relying on GPS, the interface allowed manual entry of location details. As the user typed, address suggestions appeared, helping them to select the correct location quickly. This ensured accuracy while accommodating scenarios where GPS access was unavailable or restricted. Figure 5.67 demonstrated the Manually Input Coordinates Page. For maximum accuracy, users could enter latitude and longitude values directly. This feature was particularly useful for rural areas or custom locations where addresses were not easily recognized. The page provided labelled input fields for latitude and longitude, with guidance on obtaining coordinates from Google Maps.

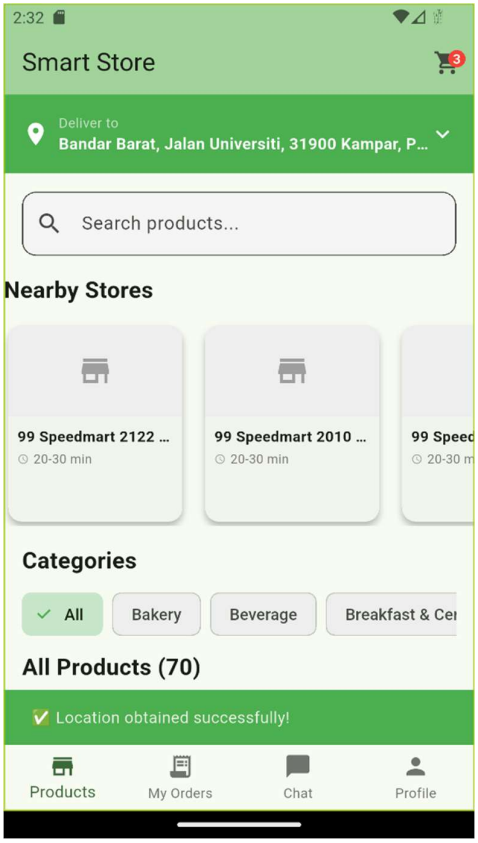



Figure 5.68 Location Obtained Successfully Notification

Figure 5.68 displayed the Location Obtained Successfully Notification. Once the system retrieved a valid location, a green confirmation message appeared at the bottom of the screen, indicating success. This notification reassured users that their location had been captured accurately and that nearby stores and delivery estimates would be updated accordingly.

### 5.2.3 My Order Page

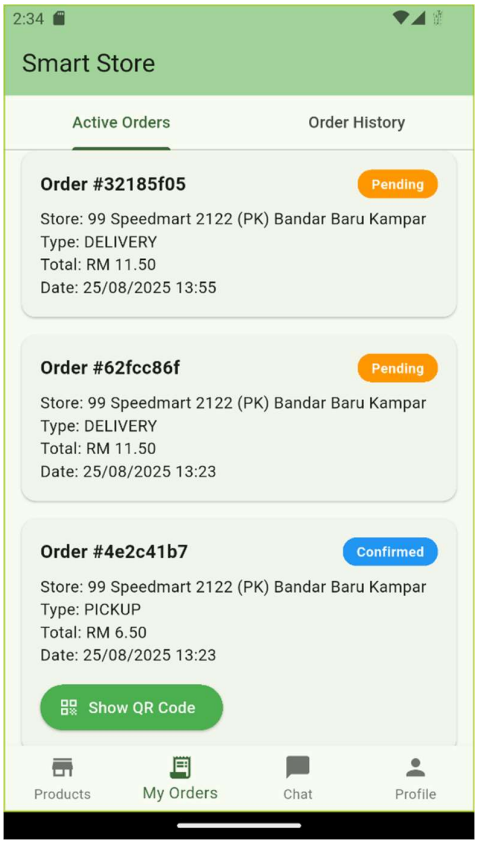



Figure 5.69 Delivery Orders Page

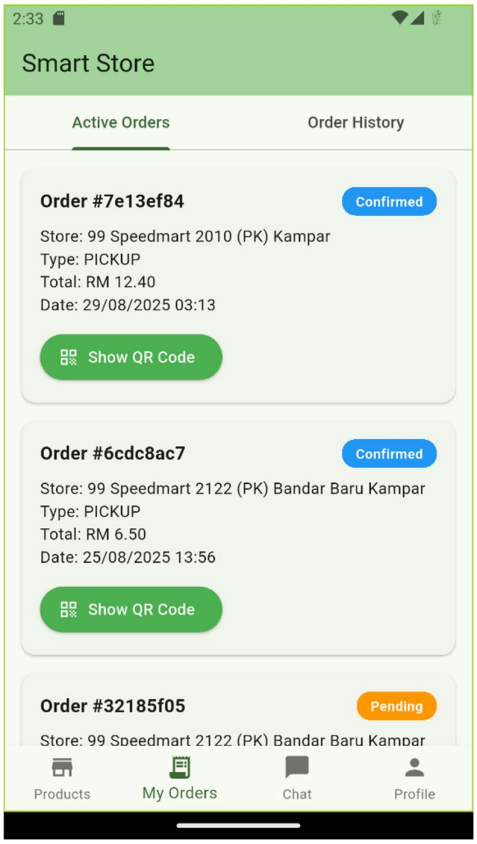



Figure 5.70 Pickup Orders Page

Figure 5.69 illustrated the Delivery Orders Page of the Smart Shopping application. This interface listed all active delivery orders with details such as store name, order type, total amount, and timestamp. Each order was labelled with a status indicator, such as “Pending” or “Confirmed,” to inform users of the progress. A “Show QR Code” button was also included for quick access to order verification during collection or delivery confirmation. Figure 5.70 presented the Pickup Orders Page of the Smart Shopping application. Like delivery orders, this page displayed active pickup orders along with store details, total cost, and order timestamps. Orders were marked with a confirmation status, and users were provided with a “Show QR Code” option to facilitate in-store pickup. This page enabled efficient management and tracking of pickup-based transactions.

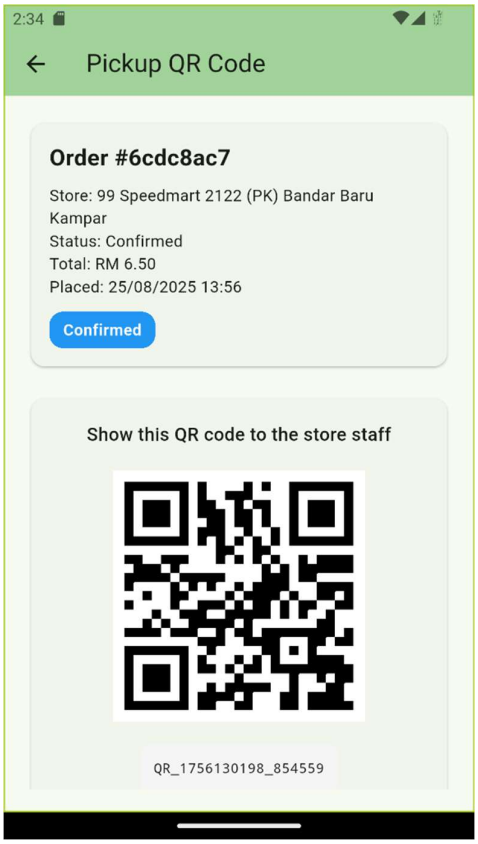



Figure 5.71 Pickup QR Code Page

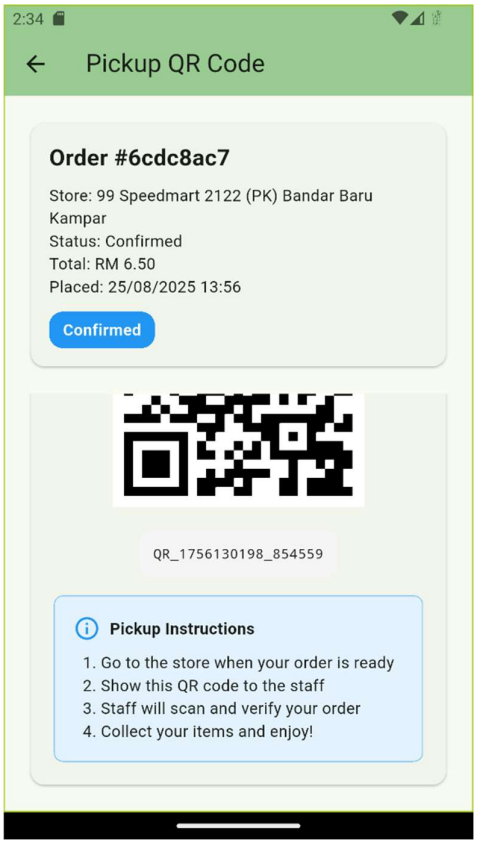



Figure 5.72 Pickup Instructions Page

Figure 5.71 demonstrated the Pickup QR Code Page. Once a pickup order was confirmed, a unique QR code was generated and displayed to the user. This QR code served as digital proof of purchase, which could be scanned by store staff to validate and release the order. Displaying the QR code directly within the app streamlined the verification process, reducing the need for manual checks. Figure 5.72 showed the Pickup Instructions Page of the Smart Shopping application. Alongside the QR code, this page included clear instructions for completing the pickup process. Users were guided to visit the store when their order was ready, present the QR code to staff, and collect their items after verification. The combination of QR code and step-by-step instructions ensured a smooth and user-friendly pickup experience.

### 5.2.4 Chatbot

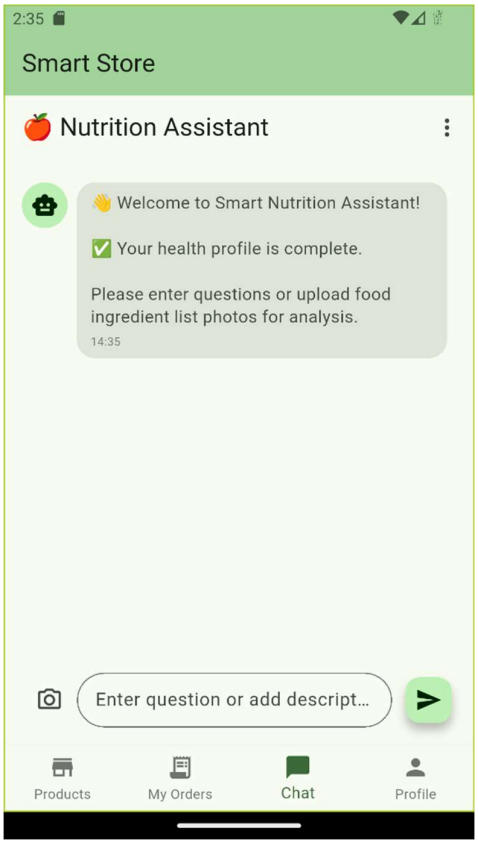



Figure 5.73 Chatbot Page

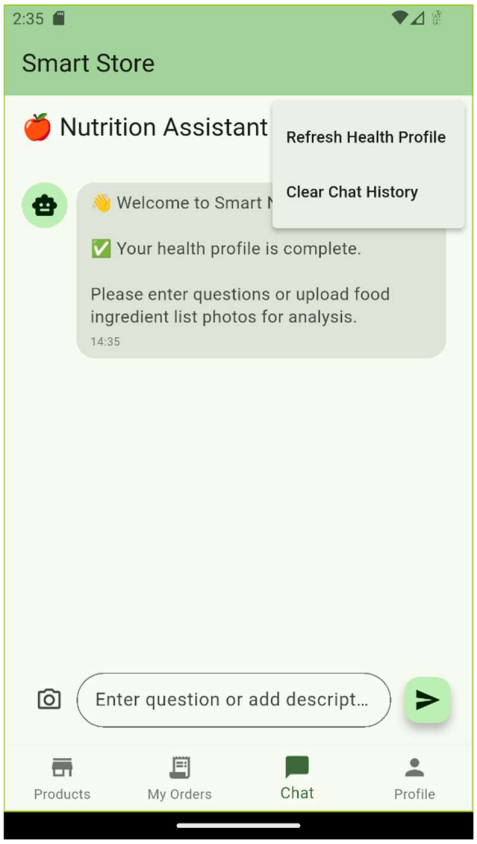



Figure 5.74 Chatbot Options Menu Page

Figure 5.73 illustrated the Chatbot Page of the Smart Shopping application, referred to as the Smart Nutrition Assistant. The chatbot provided users with personalized nutrition support by answering questions and analysing food ingredient lists. The interface included a text input field, an upload option for photos, and displayed system messages to guide users. This feature enhanced the application by offering interactive health-related advice. Figure 5.74 presented the Chatbot Options Menu Page. The options menu allowed users to manage chatbot functionality, including refreshing the health profile or clearing chat history. These options ensured that users could update their dietary data for more accurate analysis or reset the chatbot conversation for a fresh interaction.

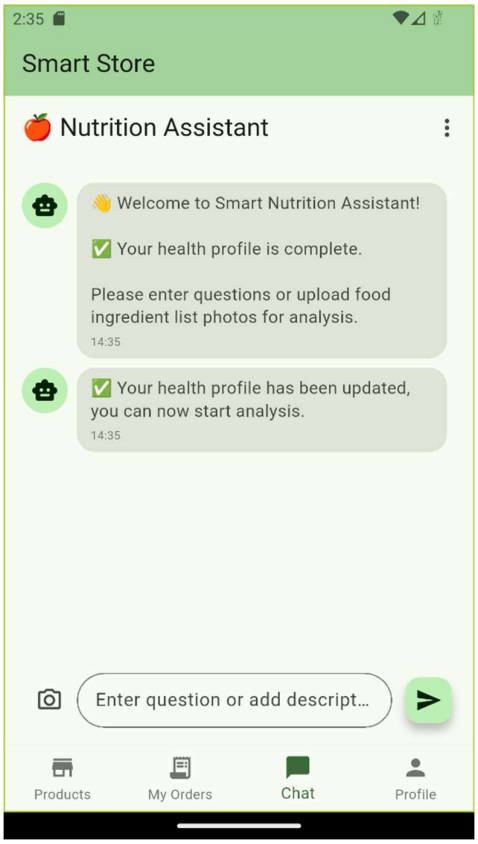



Figure 5.75 After Refresh Health Profile

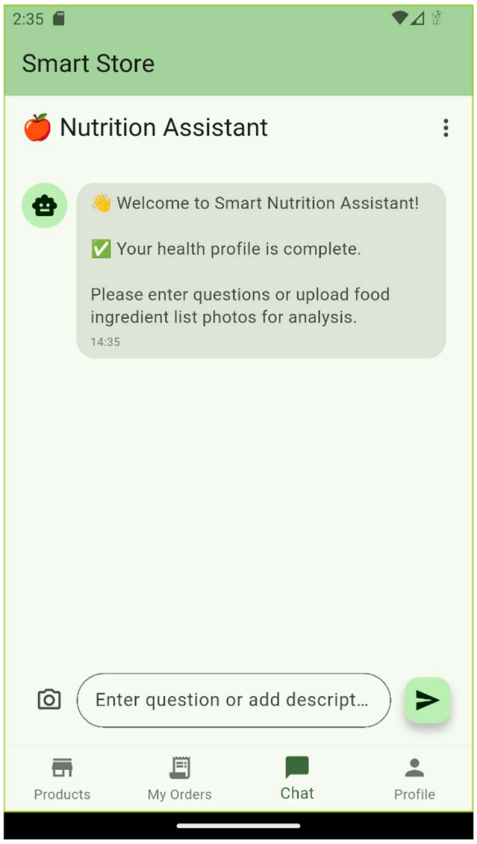



Figure 5.76 After Clear Chat History

Figure 5.75 showed the Chatbot Page after the Refresh Health Profile option was selected. The chatbot confirmed that the health profile had been successfully updated, allowing users to proceed with nutrition analysis using the latest health data. This ensured that recommendations were aligned with the user's most recent health information. Figure 5.76 illustrated the Chatbot Page after the Clear Chat History option was selected. The previous conversation was removed, and the interface returned to the default welcome state. This feature allowed users to start new sessions without being affected by past interactions, ensuring clarity and improved usability.

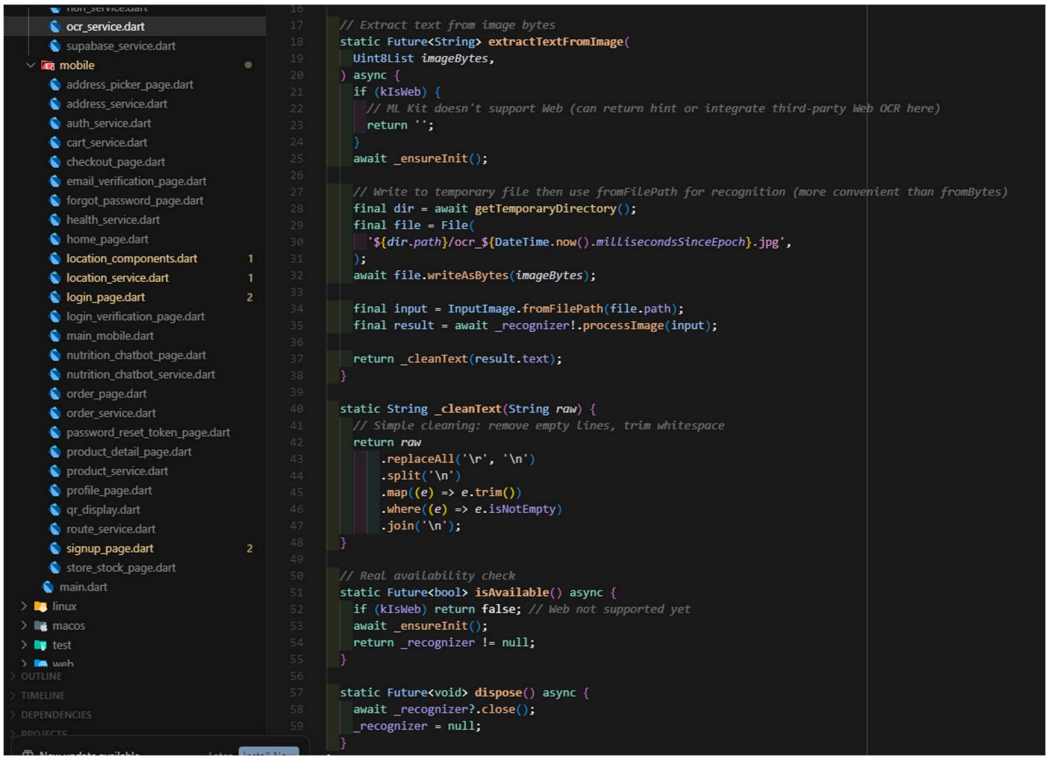



Figure 5.77 OCR Service Code Implementation

Figure 5.77 demonstrated the OCR Service Code Implementation. The extractTextFromImage function processed uploaded images by converting them into temporary files and running optical character recognition (OCR) for text extraction. The code also included a cleanText method to remove formatting noise and empty lines, and an isAvailable check to ensure service compatibility. This implementation enabled the chatbot to analyse nutrition labels and ingredient lists directly from user-uploaded images.

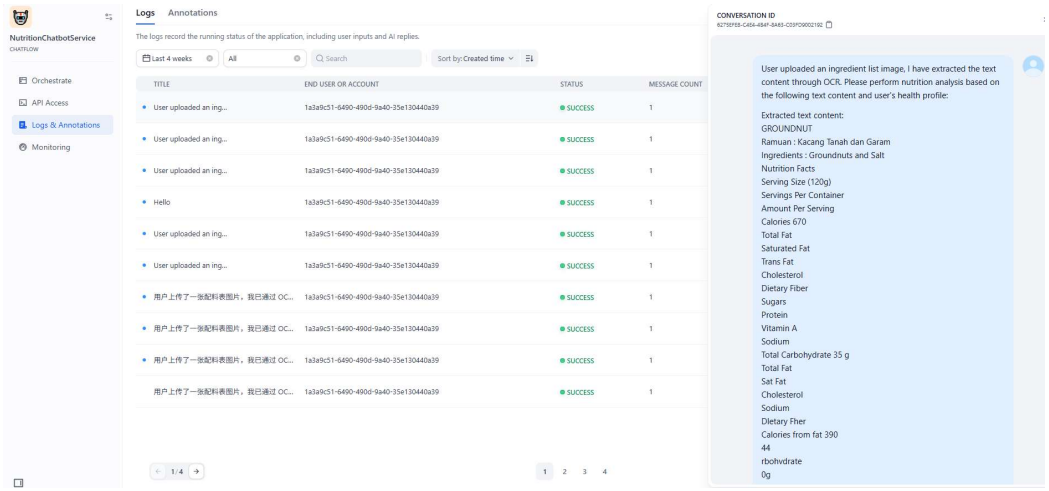



Figure 5.78 Nutrition Chatbot Service Logs

Figure 5.78 illustrated the Nutrition Chatbot Service Logs, which recorded the running status of the chatbot system. The log entries included user interactions such as text inputs and image uploads, along with corresponding system responses. Each entry displayed the end-user ID, timestamp, status, and message count. The example on the right showed an OCR-extracted ingredient list where the chatbot processed food label text, enabling further nutritional analysis based on the user's health profile.

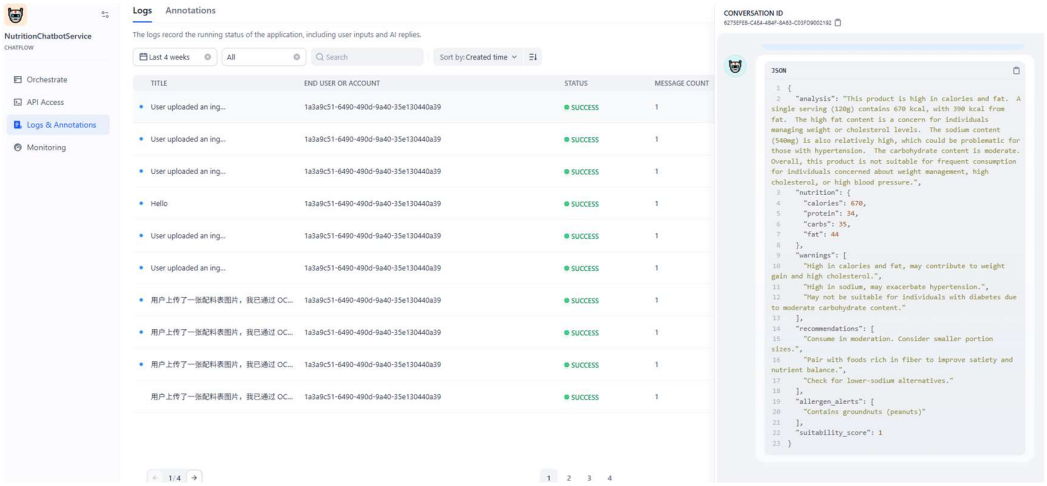



Figure 5.79 Nutrition Chatbot OCR Extracted Nutrition Content

Figure 5.79 presented the Nutrition Chatbot OCR Extracted Nutrition Content. After OCR successfully extracted the nutrition label data, the chatbot generated structured output in JSON format. The content included nutritional facts such as calories, fat, carbohydrates, sodium, and relevant health warnings. Additional sections provided personalized recommendations, allergen alerts, and suitability scores based on the user's health profile. This demonstrated how the chatbot transformed raw OCR data into actionable nutritional insights.

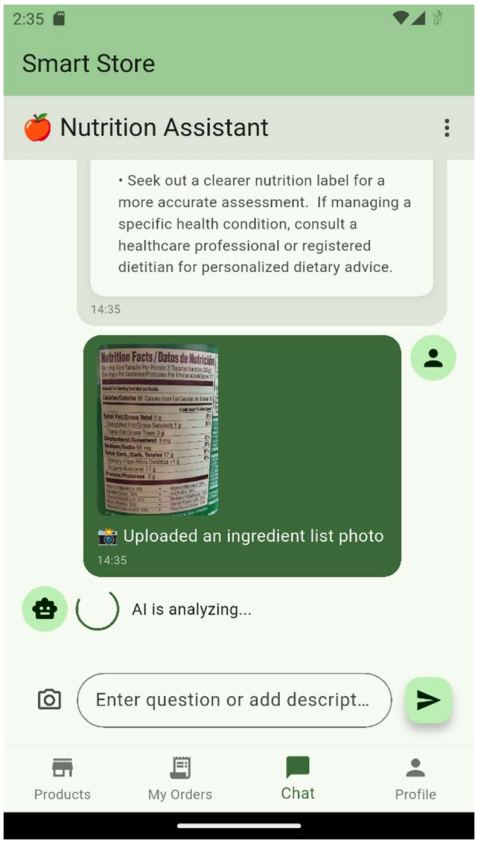



Figure 5.80 Upload Ingredient List Photo

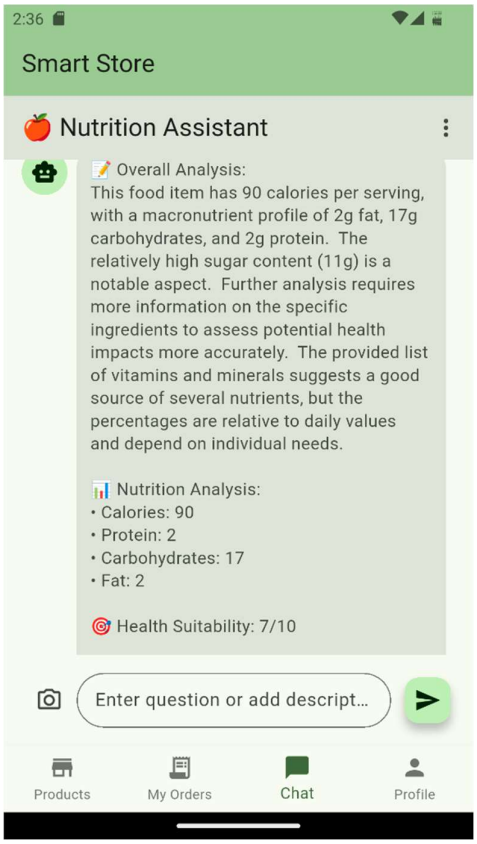



Figure 5.81 Nutrition Analysis Reply

Figure 5.80 illustrates the Upload Ingredient List Photo Page of the Smart Nutrition Assistant. Users were able to upload an image of a nutrition label or ingredient list, which was then processed by the system's OCR and AI modules. The chatbot displayed a confirmation message indicating that the uploaded photo was being analyzed. This functionality provides a convenient way for users to receive nutritional insights without manually entering data. Figure 5.81 presented the Nutrition Analysis Reply generated by the chatbot after processing an uploaded food label. The system provided an overall analysis, including calorie count, macronutrient composition, and notable remarks such as high sugar content. The reply also included a health suitability score, offering users a quick assessment of how well the food item fit within general dietary guidelines.

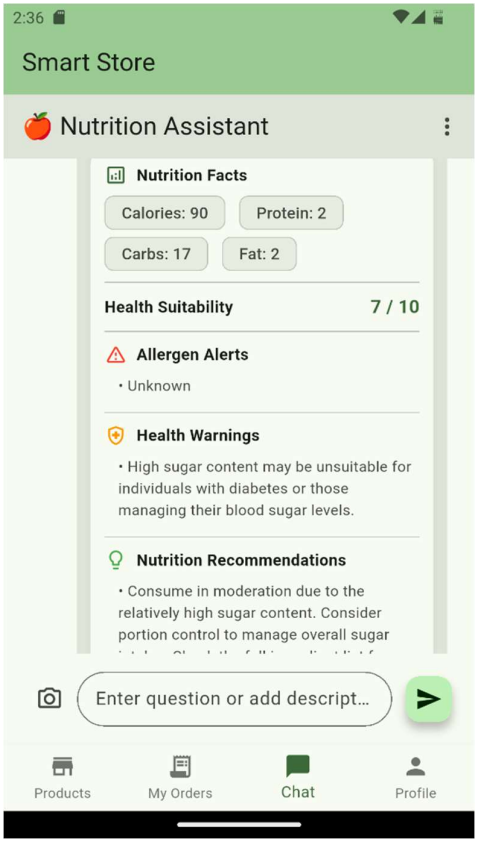



Figure 5.82 Nutrition Analysis with Alerts and Recommendations

Figure 5.82 demonstrated the Nutrition Analysis with Alerts and Recommendations Page. Beyond displaying macronutrient data, the chatbot generated additional sections such as allergen alerts, health warnings, and personalized nutrition recommendations. These insights helped users identify potential dietary risks and adopt healthier eating habits by following system-generated advice.

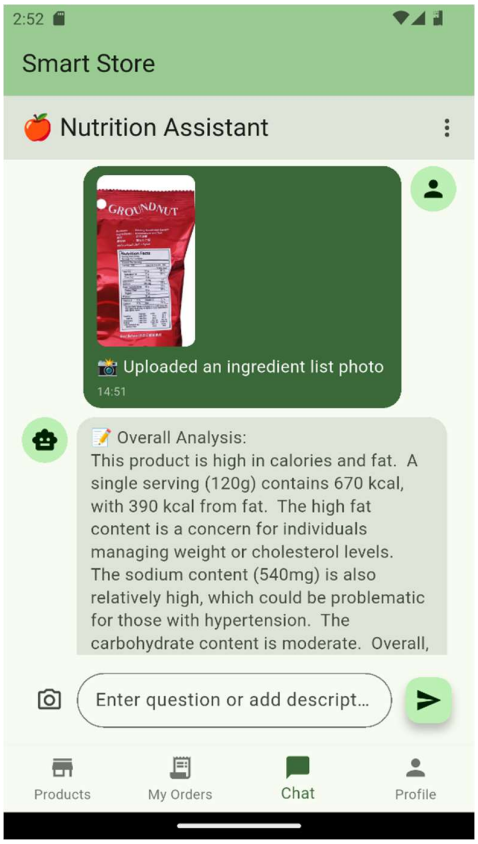



Figure 5.83 Nutrition Analysis Test with Alerts

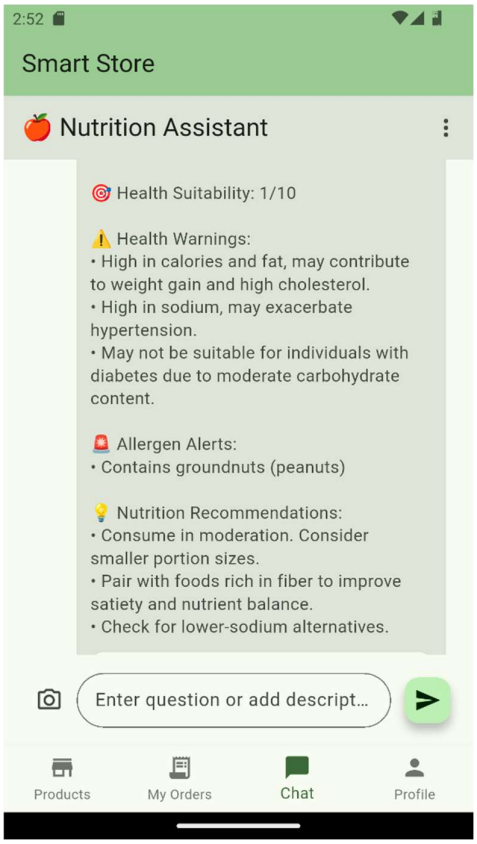



Figure 5.84 Detailed Alerts and Recommendations

Figure 5.83 showed the Nutrition Analysis Test with Alerts for a high-calorie and high-fat food item. The system identified excessive calorie and fat levels, alongside high sodium content, which could be problematic for individuals with hypertension or cholesterol issues. The chatbot provided an overall analysis highlighting these health concerns, reinforcing its role as a preventive dietary assistant. Figure 5.84 illustrated the Detailed Alerts and Recommendations generated by the chatbot. The analysis flagged allergen risks, such as groundnut content, and provided health warnings for conditions like diabetes and hypertension. Additionally, the chatbot recommended smaller portion sizes, increased fiber intake, and lower-sodium alternatives to support balanced nutrition. This detailed guidance enabled users to make informed food choices tailored to their health profiles.

### 5.2.5 Profile

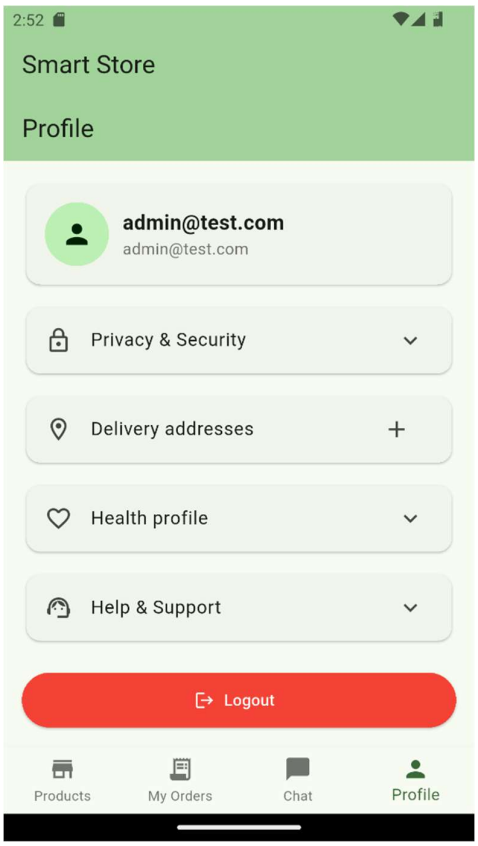



Figure 5.85 Profile Main Page

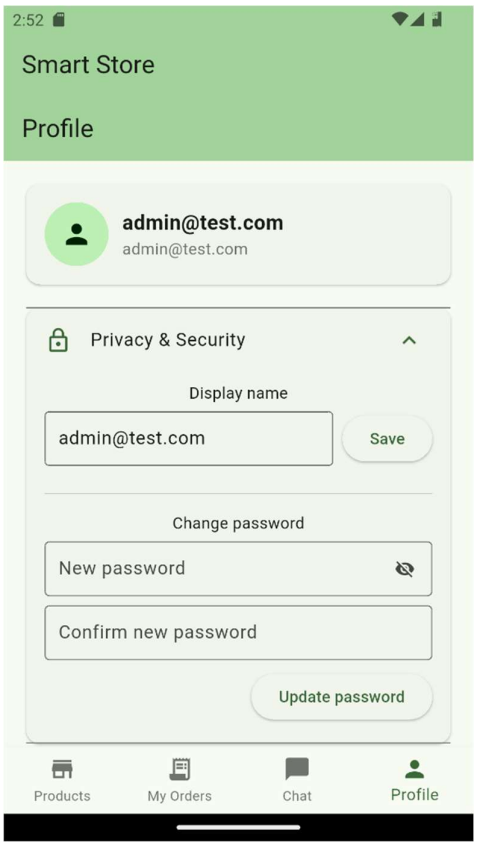



Figure 5.86 Edit Profile and Password Page

Figure 5.85 illustrated the Profile Main Page of the Smart Shopping application. This section enabled users to manage their account settings, including privacy and security, delivery addresses, health profile, and help and support options. A prominent logout button was also included to allow users to securely exit their accounts. The design provided a centralized location for accessing personal and account-related features. Figure 5.86 presented the Edit Profile and Password Page under the Privacy & Security section. Users were able to update their display name and change their account password by entering and confirming a new password. A "Save" and "Update password" button was provided to finalize changes. This feature enhanced account security and allowed users to maintain control over their profile credentials.

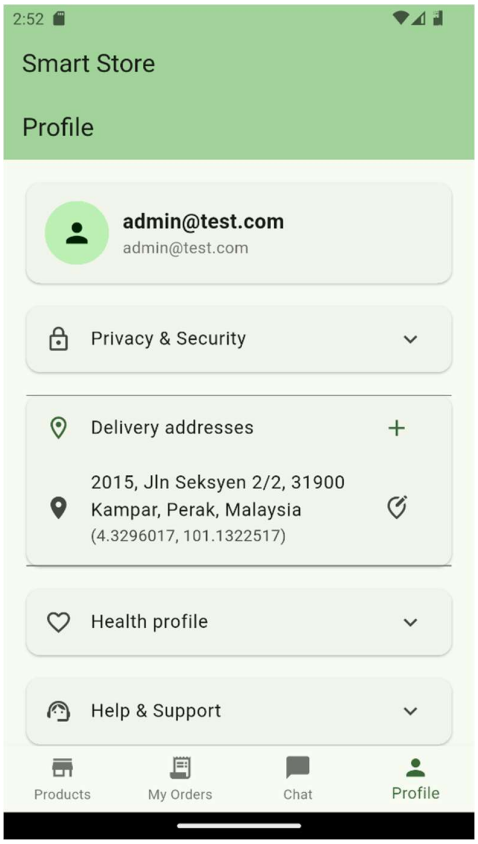



Figure 5.87 Delivery Address Page

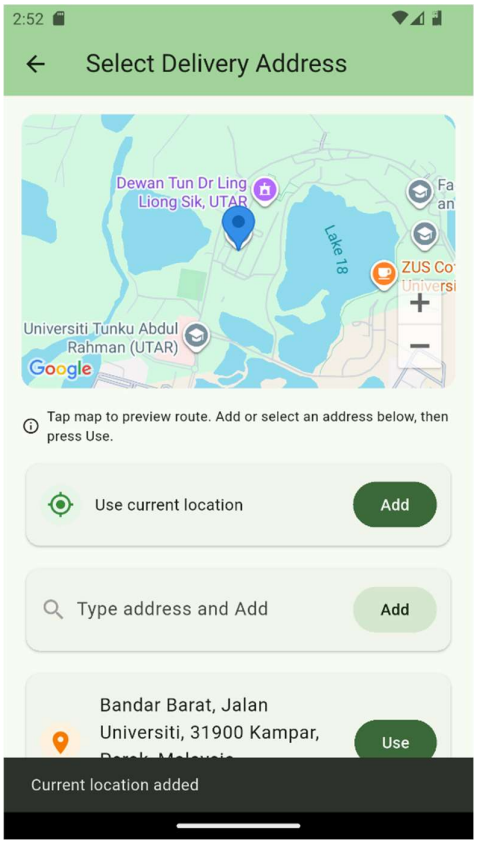
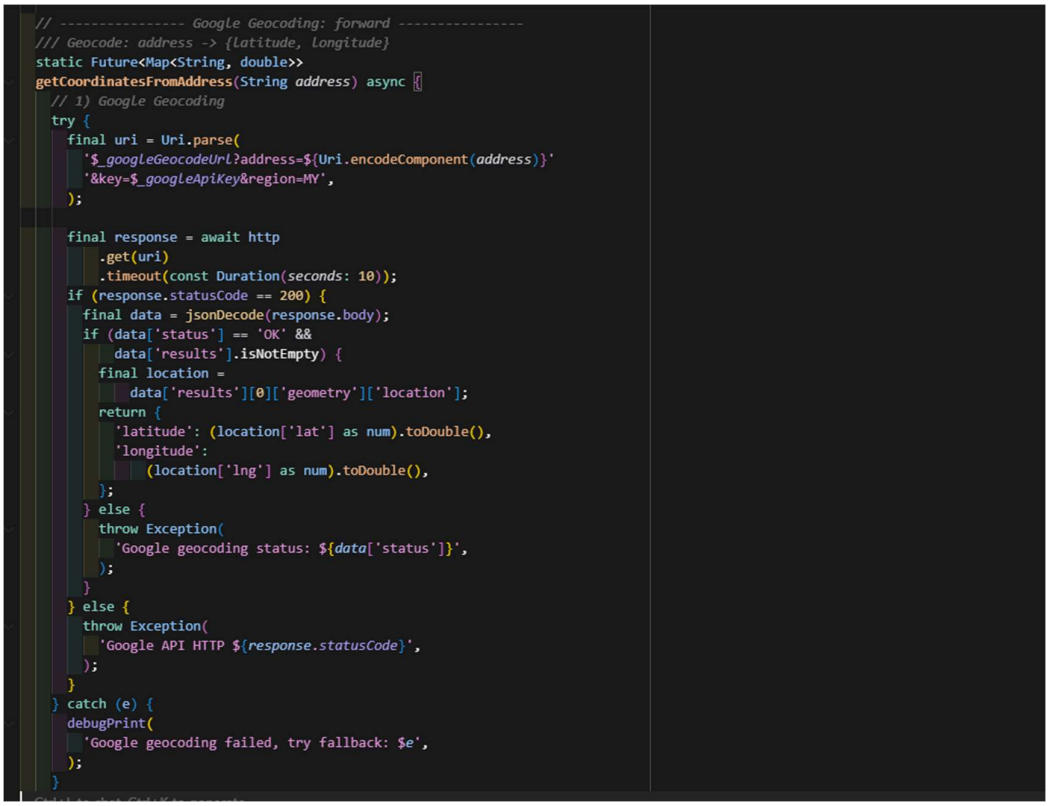



Figure 5.88 Select Delivery Address Page

Figure 5.87 showed the Delivery Address Page, where users could view and manage their saved delivery addresses. The page displayed the full address along with geographic coordinates for accuracy. An edit option was provided for modifying existing entries, ensuring that delivery details remained up to date and precise. Figure 5.88 demonstrated the Select Delivery Address Page. Users were able to add a new address either by using the current device location or by typing in the address manually. The page also displayed a map preview to confirm the location visually. Once added, the address could be set as active for deliveries. This functionality improved flexibility and user convenience in managing delivery preferences.



```dart
// ---------------- Google Geocoding: forward ----------------
/// Geocode: address -> {latitude, longitude}
static Future<Map<String, double>>
getCoordinatesFromAddress(String address) async {
  // 1) Google Geocoding
  try {
    final uri = Uri.parse(
      '$_googleGeocodeUrl?address=${Uri.encodeComponent(address)}'
      '&key=$_googleApiKey&region=MY',
    );

    final response = await http
        .get(uri)
        .timeout(const Duration(seconds: 10));
    if (response.statusCode == 200) {
      final data = jsonDecode(response.body);
      if (data['status'] == 'OK' &&
          data['results'].isNotEmpty) {
        final location =
            data['results'][0]['geometry']['location'];
        return {
          'latitude': (location['lat'] as num).toDouble(),
          'longitude':
              (location['lng'] as num).toDouble(),
        };
      } else {
        throw Exception(
          'Google geocoding status: ${data['status']}',
        );
      }
    } else {
      throw Exception(
        'Google API HTTP ${response.statusCode}',
      );
    }
  } catch (e) {
    debugPrint(
      'Google geocoding failed, try fallback: $e',
    );
  }
```

Figure 5.89 Google Geocoding Code Implementation

Figure 5.89 illustrated the Google Geocoding Code Implementation, which converted user-provided addresses into geographical coordinates (latitude and longitude). The implementation made use of the Google Geocoding API to send HTTP requests and parse the response. The code verified whether the request returned a valid status and extracted the location values from the JSON response. In case of errors, appropriate exceptions were raised to ensure reliability and accuracy.

```
Ctrl+I to chat, Ctrl+K to generate
  // 2) Fallback: ORS
  try {
    final uri = Uri.parse(
      '$_baseUrl?api_key=$_apiKey&text=${Uri.encodeComponent(address)}',
    );
    final response = await http.get(uri);
    if (response.statusCode != 200) {
      throw Exception(
        'ORS geocoding HTTP ${response.statusCode}',
      );
    }
    final data = jsonDecode(response.body);
    final feats = (data['features'] as List?) ?? [];
    if (feats.isEmpty) {
      throw Exception(
        'No coordinates found for "$address"',
      );
    }
    final coordinates =
        feats[0]['geometry']['coordinates'];
    final lon = (coordinates[0] as num).toDouble();
    final lat = (coordinates[1] as num).toDouble();
    return {'latitude': lat, 'longitude': lon};
  } catch (e) {
    throw Exception('Failed to geocode "$address": $e');
  }
}
```

Figure 5.90 ORS Geocoding Code Implementation

Figure 5.90 presented the ORS (OpenRouteService) Geocoding Code Implementation, which acted as a fallback mechanism when Google Geocoding failed or was unavailable. This implementation followed a similar structure by sending a request to the ORS API, decoding the response, and extracting coordinate values. The fallback ensured service continuity, enabling the application to retrieve location data from an alternative source when primary geocoding failed.

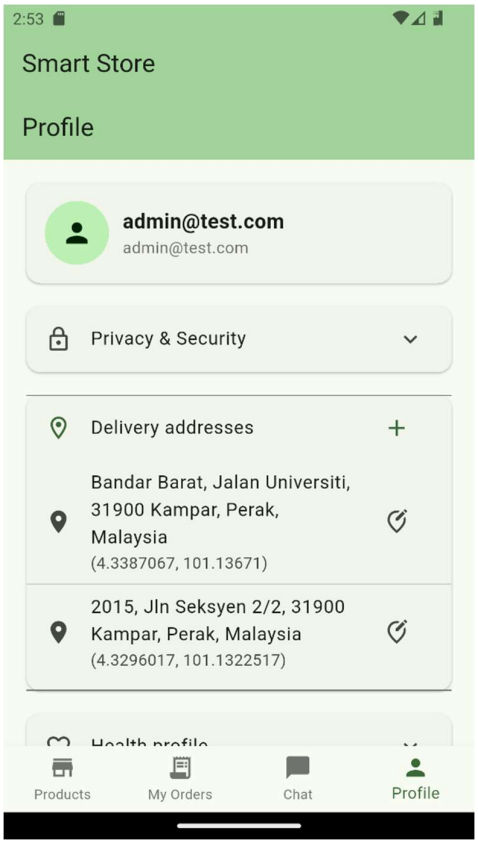



Figure 5.91 Update Delivery Address Successful

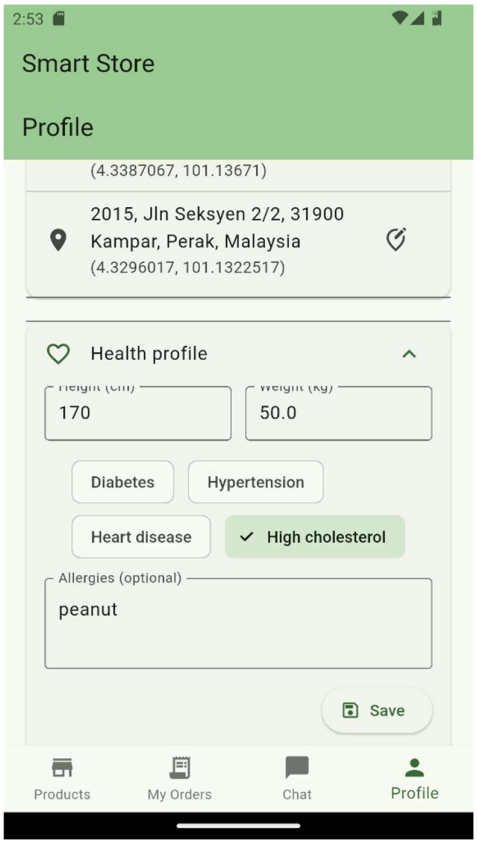



Figure 5.92 Update Health Profile

Figure 5.91 demonstrated the successful update of a delivery address within the profile management module. The interface displayed multiple saved addresses, each with detailed location information including street, city, state, and geographical coordinates. This allowed users to manage multiple delivery destinations conveniently, ensuring that their orders could be directed to accurate and preferred addresses. Figure 5.92 illustrated the health profile update page, where users were able to input their personal health-related information. The form included fields such as height, weight, medical conditions (e.g., diabetes, hypertension, heart disease, high cholesterol), and optional allergy details. The updated health profile served as a critical input for the Nutrition Assistant module, enabling personalized nutritional analysis and dietary recommendations tailored to individual health needs.

### 5.2.6 Check Out

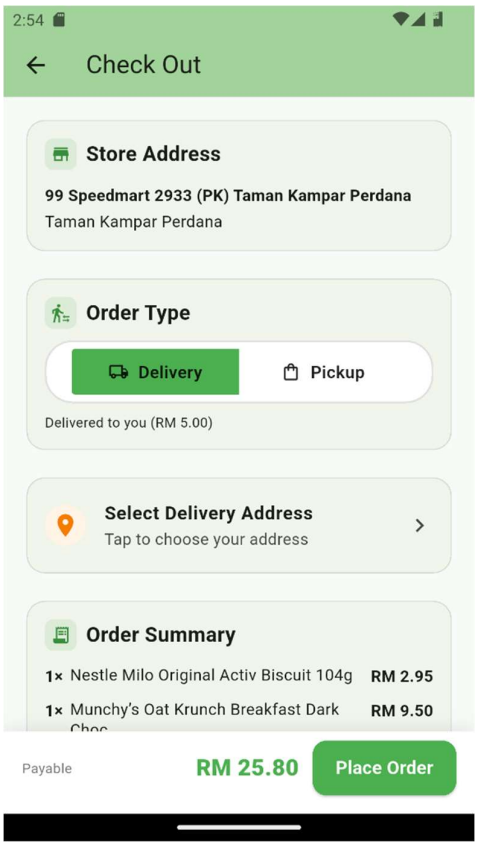



Figure 5.93 Check Out Page

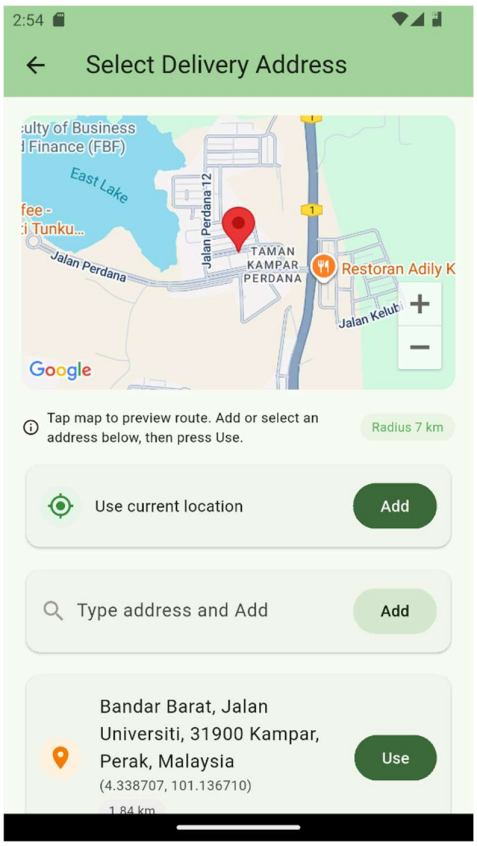



Figure 5.94 Select Delivery Address Page

Figure 5.93 illustrated the Check Out page, where users were able to review their order details before confirming the purchase. The page displayed the store address, provided an option to select the order type (delivery or pickup), and included a section for choosing a delivery address. An order summary with item details and total cost was presented at the bottom, along with a prominent "Place Order" button to proceed with the transaction. Figure 5.94 showed the Select Delivery Address page, which enabled users to specify their preferred delivery location. The page incorporated an interactive map for previewing routes, as well as options to use the current location or manually enter an address. Additional controls allowed users to confirm and save the selected address, ensuring accuracy and convenience in managing delivery information.

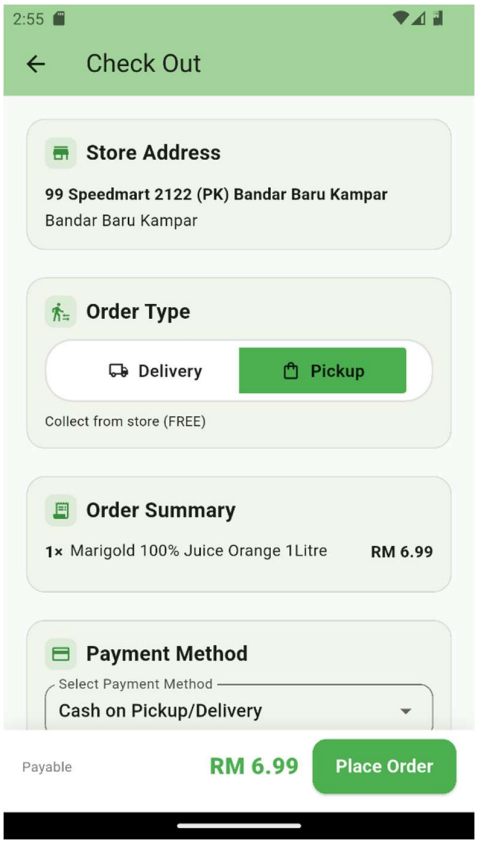



Figure 5.95 Check Out with Pickup Order

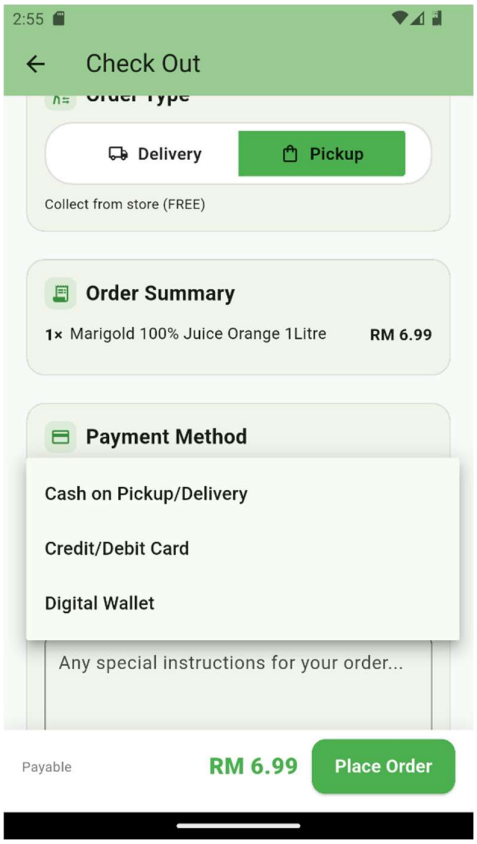



Figure 5.96 Payment Method Selection

Figure 5.95 illustrated the Check Out page configured for a pickup order. The interface displayed the store address, order type set to “Pickup,” and an order summary with the selected product details and total cost. The page also included a section for selecting a payment method, ensuring that users could confirm and finalize their purchase before proceeding with the order. Figure 5.96 showed the Payment Method Selection page, where users were able to choose from multiple options, including cash on pickup/delivery, credit or debit card, and digital wallet. This flexibility allowed the system to accommodate different user preferences, ensuring a convenient and user-friendly checkout process.

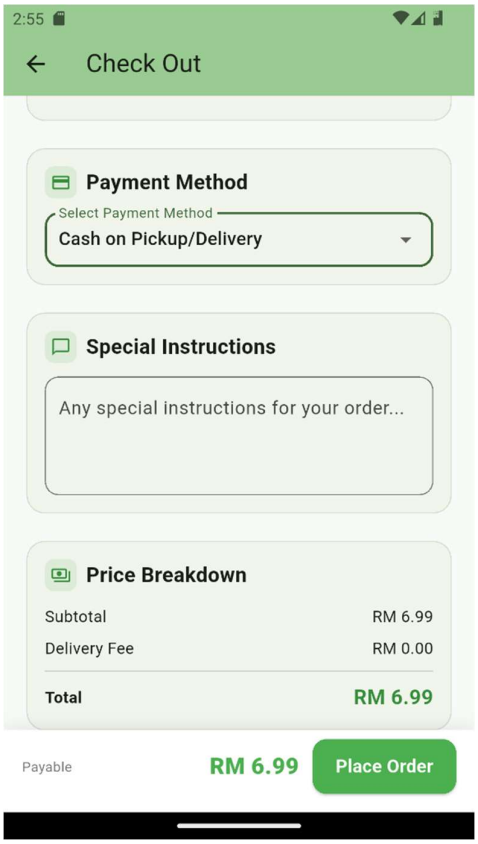



Figure 5.97 Special Instructions in Checkout Page

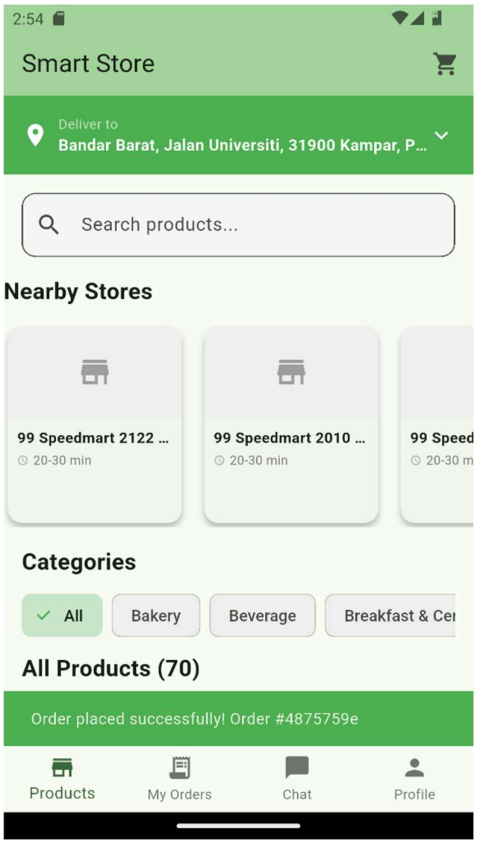



Figure 5.98 Order Placed Successfully Notification

Figure 5.97 illustrates the Special Instructions section within the Checkout page. This feature enabled users to provide additional notes or specific requests related to their order, such as delivery preferences or product handling instructions. The page also presented a clear price breakdown, including subtotal, delivery fee, and total payable amount, ensuring transparency for the user before finalizing the order. Figure 5.98 showed the Order Placed Successfully notification displayed after a purchase was confirmed. The system generated a success message along with an order reference number, which served as proof of confirmation. This notification assured users that their transaction was successfully completed, and their order had been recorded in the system.

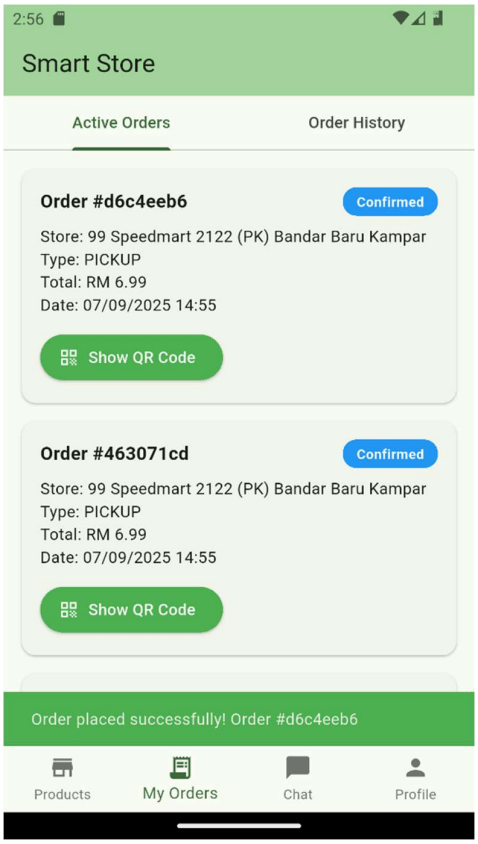



Figure 5.99 Order Confirmation in My Orders Page

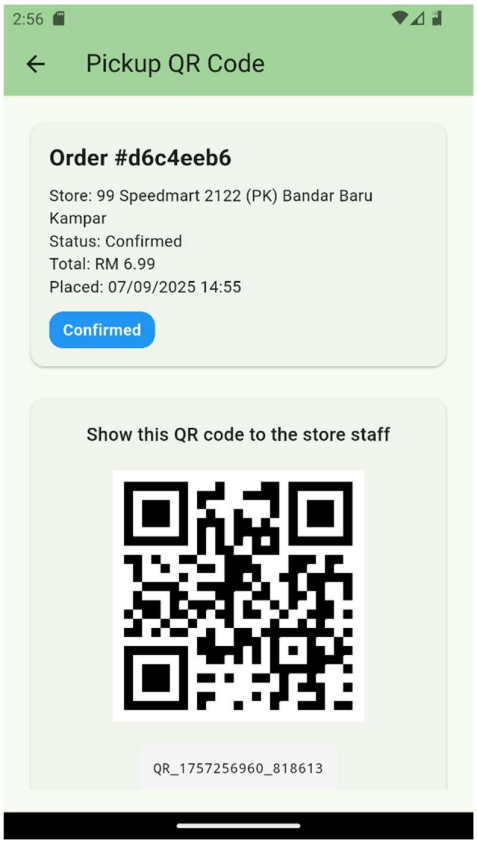



Figure 5.100 In-Store Pickup QR Code Page

Figure 5.99 illustrates the Order Confirmation page within the My Orders section. This page displayed the list of confirmed orders, showing details such as the order ID, store location, order type (pickup or delivery), total amount, and the date and time of placement. Each order included a "Show QR Code" button, allowing users to quickly access their digital pickup code for store verification. Figure 5.100 depicted the In-Store Pickup QR Code page, which generated a unique QR code linked to the specific order. Users were required to present this QR code to store staff during collection, ensuring a secure and efficient verification process. This feature streamlined the pickup workflow and minimized manual errors by providing a digital proof of purchase.

# CHAPTER 6

# System Evaluation and Discussion

## 6.1 System Performance Evaluation

The performance of the Smart Shopping Assistant mobile application was evaluated based on responsiveness, resource usage, and reliability. Testing showed that the application loaded within a few seconds and provided smooth navigation across modules such as product browsing, cart management, and nutrition analysis. Resource utilization remained efficient, with stable performance on mid-range devices and minimal battery impact even when using features like location tracking and OCR. Backend testing under simulated high-traffic conditions confirmed that concurrent requests, including order placement and profile updates, were processed reliably. Overall, the system demonstrated fast response times, low resource consumption, and stable scalability, ensuring a seamless and efficient user experience.

### 6.1.1 Network Performance

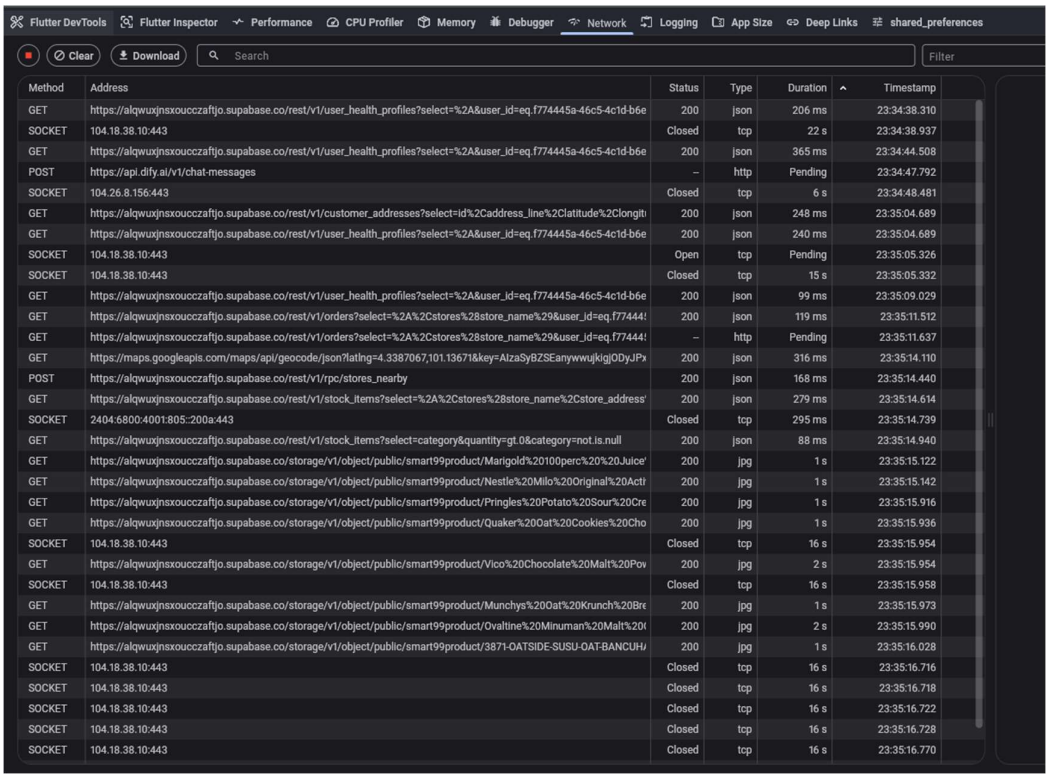



Figure 6.1 Chatbot Module Network Activity (Flutter DevTools)

Figure 6.1 illustrated the network activity of the chatbot module, captured using Flutter DevTools' Network panel. The trace recorded HTTPS requests and socket connections made during a chat session, including REST calls to the Supabase backend (e.g., user health profile, orders, stores, stock items, and image assets from Supabase Storage), a Google Maps geocoding request, and the chatbot service endpoint for message processing. For each request, the tool listed method, address, status, payload type, duration, and timestamp, enabling evaluation of response times, verification of successful 200 responses, and identification of any potential bottlenecks during chatbot interactions.

**Response Time:**
From the capture, core REST calls completed quickly: user_health_profiles (206–365 ms), orders (99–119 ms), rpc/stores_nearby (168 ms), v1/stock_items (88–279 ms), and Google geocoding (≈316 ms). Image fetches from Supabase Storage were slower at ~1–2 s each. One external chat POST (Dify) remained **Pending**, indicating longer server-side processing.

**Consistency:**
Nearly all requests returned **200** with stable sub-300 ms times for JSON APIs. Repeated profile/order queries showed low variance. No failed requests observed; several sockets show "Closed" after use, consistent with short-lived connections.

**Data Types:**
Mixed traffic: JSON for Supabase REST/RPC, JPG for product images (CDN/storage), TCP socket entries for HTTPS connections, and standard HTTP POST for the chatbot endpoint.

**Variations in Request Duration:**
Shortest times were lightweight JSON queries (≤300 ms). Longer durations correlated with large payloads (image downloads ~1–2 s) and third-party processing (chat POST). Network conditions and cache misses likely contribute to the spread.

**Overall Performance:**
API responsiveness is good for real-time UX; image loading is the main latency source. To optimize: cache and resize thumbnails, enable/verify CDN caching with ETag/Cache-Control, prefetch critical images, batch REST queries where possible, and stream chatbot responses to avoid long "Pending" waits.

### 6.1.2 Retrieve Product Picture (Supabase)

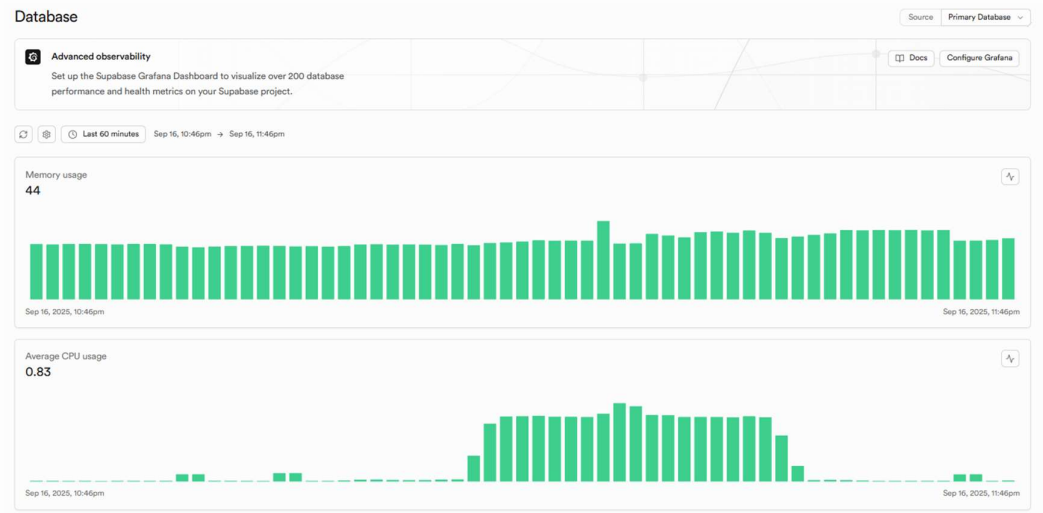



Figure 6.2 Database Metrics — Memory Usage and Average CPU Usage

Figure 6.2 illustrated the database's memory usage and average CPU usage on the Supabase dashboard during the evaluation period. Memory consumption remained generally stable with minor fluctuations and no signs of progressive growth, indicating the absence of leaks. Average CPU usage showed a distinct activity window with sustained moderate utilization during testing, returning to near-idle levels outside that window.

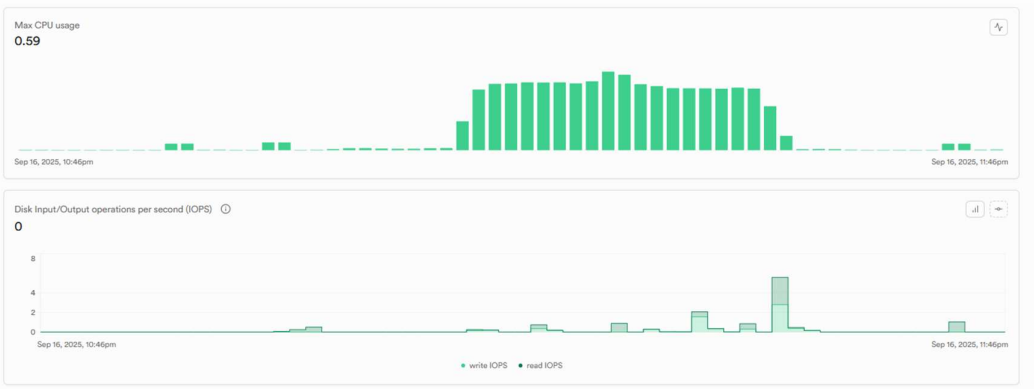



Figure 6.3 Database Metrics — Max CPU Usage and Disk I/O Operations per Second

Figure 6.3 presented the database's maximum CPU usage and disk I/O operations per second (IOPS). Maximum CPU mirrored the same activity window as Figure 6.2, peaking during load and tapering afterward. Disk IOPS stayed low overall with short, isolated spikes, suggesting brief bursts of read/write activity without prolonged saturation, thereby indicating available performance headroom.

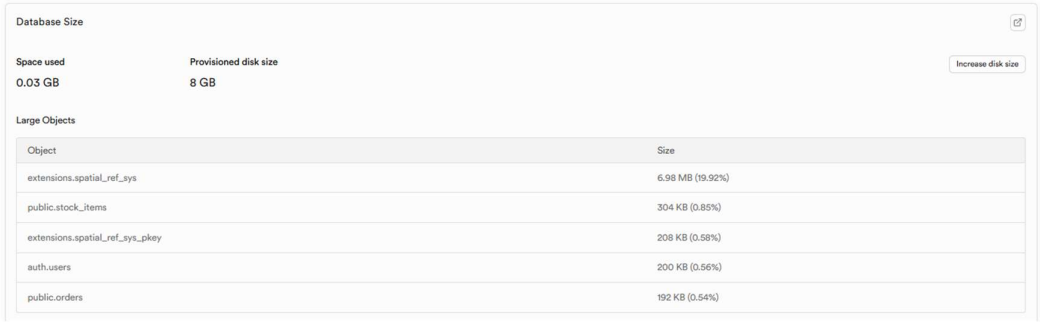



Figure 6.4 Database Size and Large Objects (Supabase Dashboard)

Figure 6.4 illustrated the database storage utilization. The instance used 0.03 GB out of a provisioned 8 GB disk, indicating utilization of well under one percent. The “Large Objects” list showed that space was primarily occupied by extensions.spatial_ref_sys (6.98 MB, 19.92%), followed by small application tables such as public.stock_items (304 KB, 0.85%), auth.users (200 KB, 0.56%), and public.orders (192 KB, 0.54%). Overall, storage consumption remained minimal with substantial headroom for future data growth.

## 6.2 System Testing Setup and Result

### 6.2.1 User Registration (Sign Up) Testing

Table 6.1 User Registration (Sign Up) Testing

| **User Registration (Sign Up) Test Case** | | | | |
|---|---|---|---|---|
| **Use Case:** User Registration (Sign Up) | | | | |
| **Function Id:** UC 1 | | | | |
| **Date Created:** 9/9/2025 | | | | |
| **Role:** User | | | | |
| **No** | **Input Values** | **Expected Results** | **Actual Results** | **Pass/Fail** |
| 1 | Tap Sign Up on Login page | System requests user details | System requests user details | Passed |
| 2 | Enter valid name, email, password → tap Register | Account created, success message shown | Account created, success message shown | Passed |
| 3 | Enter invalid email format | Error asking to correct email | Error asking to correct email | Passed |
| 4 | Password length < 6 | Error indicating weak password | Error indicating weak password | Passed |
| 5 | Leave Name empty | Error asking to fill name | Error asking to fill name | Passed |
| 6 | Leave Email empty | Error asking to fill email | Error asking to fill email | Passed |
| 7 | Leave Password empty | Error asking to fill password | Error asking to fill password | Passed |

### 6.2.2 User Login Testing

Table 6.2 User Login Testing

| **User Login Test Case** | | | | |
|---|---|---|---|---|
| **Use Case:** User Login | | | | |
| **Function Id:** UC 2 | | | | |
| **Date Created:** 9/9/2025 | | | | |
| **Role:** User | | | | |
| **No** | **Input Values** | **Expected Results** | **Actual Results** | **Pass/Fail** |
| 1 | Open app → enter valid email & password → Sign In | User authenticated; redirected to Home | Authenticated; redirected to Home | Passed |
| 2 | Wrong password | "Invalid credentials" error | Error shown | Passed |
| 3 | Invalid email format | Validation error shown | Validation error shown | Passed |
| 4 | Leave fields empty | "Required field" errors | Errors shown | Passed |
| 5 | After password check, enter correct 6-digit login OTP | Login verified; proceed | Verified; proceed | Passed |
| 6 | Enter wrong/expired OTP | "Invalid/expired code" error | Error shown | Passed |
| 7 | Tap Logout | Session cleared; back to Login | Session cleared; back to Login | Passed |

### 6.2.3 Product Browsing and Search Testing

Table 6.3 Product Browsing and Search Testing

| **Product Browsing and Search Test Case** | | | | |
|---|---|---|---|---|
| **Use Case:** Product Browsing and Search | | | | |
| **Function Id:** UC 3 | | | | |
| **Date Created:** 9/9/2025 | | | | |
| **Role:** User | | | | |
| **No** | **Input Values** | **Expected Results** | **Actual Results** | **Pass/Fail** |
| 1 | Open Home | Nearby stores and product feed displayed | Displayed | Passed |
| 2 | Enter keyword “milo” in search | List filtered with matching items | Filtered list shown | Passed |
| 3 | Select category Beverage | List shows only Beverage items | Category filter applied | Passed |
| 4 | Search for non-existing term | Empty state / “No results” displayed | “No results” displayed | Passed |
| 5 | Tap a product card | Product Detail page opens | Detail page opens | Passed |
| 6 | Clear filters | Full product list restored | Restored | Passed |

### 6.2.4 Shopping Cart Management Testing

Table 6.4 Shopping Cart Management Testing

| **Shopping Cart Management Test Case** | | | | |
|---|---|---|---|---|
| **Use Case:** Shopping Cart Management | | | | |
| **Function Id:** UC 4 | | | | |
| **Date Created:** 9/9/2025 | | | | |
| **Role:** User | | | | |
| **No** | **Input Values** | **Expected Results** | **Actual Results** | **Pass/Fail** |
| 1 | From product detail, tap Add to Cart | Item added; confirmation shown | Added; confirmation shown | Passed |
| 2 | Add items from different store while cart has items | Error: “Cart contains items from another store” | Error shown | Passed |
| 3 | Open Cart → increase quantity (+) | Quantity & total update correctly | Updated correctly | Passed |
| 4 | Decrease quantity (−) to 0 or tap Remove | Item removed; totals updated | Removed; totals updated | Passed |
| 5 | Tap Clear All | Cart emptied | Emptied | Passed |
| 6 | Navigate away and back | Cart state persists for same store | State persisted | Passed |

### 6.2.5 Checkout Process Testing

Table 6.5 Checkout Process Testing

| **Checkout Process Test Case** | | | | |
|---|---|---|---|---|
| **Use Case:** Checkout Process | | | | |
| **Function Id:** UC 5 | | | | |
| **Date Created:** 9/9/2025 | | | | |
| **Role:** User | | | | |
| **No** | **Input Values** | **Expected Results** | **Actual Results** | **Pass/Fail** |
| 1 | From Cart → Checkout | Checkout page shows store, order type, address, payment | Shown | Passed |
| 2 | Select Delivery without address | Validation prompts to select/add address | Prompt shown | Passed |
| 3 | Use current location → select saved address | Address displayed and selectable | Address selectable | Passed |
| 4 | Select Pickup | Address skipped; pickup allowed | Skipped as expected | Passed |
| 5 | Choose payment (Cash/Card/Digital Wallet) | Payment method saved | Saved | Passed |
| 6 | Enter special instructions | Instructions included in order | Included | Passed |
| 7 | Tap Place Order with valid data | Order created; success banner shown | Created; success shown | Passed |

### 6.2.6 Order Management Testing

Table 6.6 Order Management Testing

| **Order Management Test Case** | | | | |
|---|---|---|---|---|
| **Use Case:** Order Management | | | | |
| **Function Id:** UC 6 | | | | |
| **Date Created:** 9/9/2025 | | | | |
| **Role:** User | | | | |
| **No** | **Input Values** | **Expected Results** | **Actual Results** | **Pass/Fail** |
| 1 | Open My Orders | Active Orders and Order History lists displayed | Displayed | Passed |
| 2 | Open an active pickup order | Order details shown (items, total, status) | Shown | Passed |
| 3 | Tap Show QR Code (pickup) | QR code screen opens for scanning | Open with QR | Passed |
| 4 | Open a delivered/completed order | History details shown; QR not available | Shown as expected | Passed |
| 5 | Verify totals vs. cart at purchase time | Totals match | Match | Passed |

### 6.2.7 Order Cancellation Testing

Table 6.7 Order Cancellation Testing

| **Order Cancellation Test Case** | | | | |
|---|---|---|---|---|
| **Use Case:** Order Cancellation | | | | |
| **Function Id:** UC 7 | | | | |
| **Date Created:** 9/9/2025 | | | | |
| **Role:** User | | | | |
| **No** | **Input Values** | **Expected Results** | **Actual Results** | **Pass/Fail** |
| 1 | Opening a Pending order | Cancel Order button visible | Visible | Passed |
| 2 | Tap Cancel Order → confirm dialog | Dialog appears requesting confirmation | Dialog appears | Passed |
| 3 | Confirm cancellation | Order status updated to CANCELLED; moved to History | Updated; moved to History | Passed |
| 4 | Try to cancel Confirmed/Completed order | Cancel option hidden or error shown | Not allowed as expected | Passed |

### 6.2.8 Profile Management Testing

Table 6.8 Profile Management Testing

| **Profile Management Test Case** | | | | |
|---|---|---|---|---|
| **Use Case:** Profile Management | | | | |
| **Function Id:** UC 8 | | | | |
| **Date Created:** 9/9/2025 | | | | |
| **Role:** User | | | | |
| **No** | **Input Values** | **Expected Results** | **Actual Results** | **Pass/Fail** |
| 1 | Open Profile → Privacy & Security | Current info displayed (email, name) | Displayed | Passed |
| 2 | Update Display Name → Save | Name updated; success message | Updated; success shown | Passed |
| 3 | Enter mismatched new/confirm password | Validation error shown | Error shown | Passed |
| 4 | Enter weak password (<6) | Strength/length error shown | Error shown | Passed |
| 5 | Enter valid new password → Update password | Password updated; success message | Updated; success shown | Passed |

### 6.2.9 Address Management Testing

Table 6.9 Address Management Testing

| Address Management Test Case | | | | |
|---|---|---|---|---|
| **Use Case:** Address Management | | | | |
| **Function Id:** UC 9 | | | | |
| **Date Created:** 9/9/2025 | | | | |
| **Role:** User | | | | |
| **No** | **Input Values** | **Expected Results** | **Actual Results** | **Pass/Fail** |
| 1 | Open Profile → Delivery addresses | Current address list displayed | Displayed | Passed |
| 2 | Tap Add | Navigates to Select Delivery Address page | Navigated | Passed |
| 3 | Tap Use current location → Add | Coordinates resolved; address card added; success toast | Added; toast shown | Passed |
| 4 | Enter valid address text → Add | Address geocoded; card added with lat/lng | Added; lat/lng stored | Passed |
| 5 | Map pin select → Use | Selected pin converted to address and returned | Returned to list with new entry | Passed |
| 6 | Enter invalid/nonsense address | Geocoding error shown; nothing saved | Error shown; no save | Passed |
| 7 | Swipe an address → Delete | Shows confirmation dialog | Dialog shown | Passed |
| 8 | Confirm delete | Record soft-deleted (is_archived=true); card removed | Removed; flag set | Passed |
| 9 | Cancel delete | No change to list | No change | Passed |
| 10 | Tap Use on an address from checkout flow | Address returned to checkout and populated | Populated in checkout | Passed |

### 6.2.10 Nutrition Analysis (Chatbot) Testing

Table 6.10 Nutrition Analysis (Chatbot) Testing

| **Nutrition Analysis (Chatbot) Test Case** | | | | |
|---|---|---|---|---|
| **Use Case:** Nutrition Analysis (Chatbot) | | | | |
| **Function Id:** UC 10 | | | | |
| **Date Created:** 9/9/2025 | | | | |
| **Role:** User | | | | |
| **No** | **Input Values** | **Expected Results** | **Actual Results** | **Pass/Fail** |
| 1 | Open Chat (Nutrition Assistant) with completed health profile | Welcome message; input enabled | As expected | Passed |
| 2 | Open Chat with missing health profile | Banner prompts to complete profile; input disabled | As expected | Passed |
| 3 | Send text question about a food | AI returns analysis summary + nutrition facts | Returned summary + facts | Passed |
| 4 | Upload ingredient image (clear) | OCR extracts text; AI returns analysis | Extracted; analysis shown | Passed |
| 5 | Upload blurry/low-contrast image | Helpful error/guide to retake or provide clearer image | Guidance shown | Passed |
| 6 | Food contains profile allergen | Response flags allergen alert in results | Allergen alert shown | Passed |
| 7 | Refresh Health Profile from menu | Profile reloaded; subsequent analyses use new data | Reloaded; used in next reply | Passed |
| 8 | Clear Chat History | Conversation cleared; only system welcome remains | Cleared | Passed |
| 9 | Temporary network/AI timeout | Non-blocking error shown with option to retry | Error + retry shown | Passed |
| 10 | Very long user message | Handled without crash; trimmed or processed with notice | Handled safely | Passed |

## 6.3 Project Challenges

Implementing OCR for nutrition labels was challenging because image quality, lighting, and label layouts varied widely. I had to add preprocessing and retry guidance, then convert the extracted text into a clean, structured payload. Only after that could the app issue reliable HTTP requests to the analysis endpoint; otherwise, minor OCR noise (e.g., “O” vs “0”) broke downstream parsing and produced inconsistent results.

Linking the application database to Dify AI for nutrition analysis required a careful data contract. I needed to assemble a minimal profile (allergies, conditions, height/weight) and recent inputs, sanitize them, and pass them to Dify with strict prompt and schema controls. Handling timeouts and malformed responses meant adding retries, result validation, and safe fallbacks so the chat stayed responsive.

Account verification was solved with a request-token authentication flow. I implemented secure token generation, delivery, and expiry handling for sign-up, login verification, and password reset. Edge cases—like resending tokens, expired links, or app restarts mid-flow—were addressed with clear error states, cooldown timers, and idempotent API calls to keep the user journey reliable.

Capturing nearby stores depended on accurate geolocation and robust geocoding. I built a permission-aware location service with fallbacks from last known position to high-accuracy fixes and added a manual address/coordinate entry path. On the back end, PostGIS queries were tuned to return stores within a radius quickly, ensuring the app could display relevant options even on lower-end devices or weak signals.

## 6.4 Objectives Evaluation

### 6.4.1 Achievement of Project Objectives

**Objective 1: To establish a security and privacy-centric architecture for handling sensitive user data.**

Evaluation: Achieved. The app enforces a two-step sign-in flow combining password authentication with an email One-Time Token (OTP). After a valid password is submitted, a 6-digit code is sent to the registered email; only when the user enters the correct code is the session established (see Figures 5.50–5.54 and the User Login tests in Table 6.1). This "password + email OTP" approach mitigates credential-stuffing and shared-device risks. On the data layer, access to user records (profiles, addresses, orders) is restricted through backend policies and authenticated requests; images and messages are tied to the signed-in user. Together, these measures protect identity, limit data exposure, and satisfy the project's privacy goal.

**Objective 2: To create an artificial intelligence–driven decision support component that leverages OCR-parsed nutrition data and user context to deliver personalized recommendations during shopping.**

Evaluation.: Achieved. The Nutrition Assistant pipeline extracts text from ingredient/label photos via on-device OCR, cleans the text, and sends it—along with the user's Health Profile (e.g., diabetes, hypertension, allergies)—to the AI service for analysis. The assistant returns a structured summary (nutrition facts, health suitability score), allergen flags, warnings, and practical suggestions tailored to the user (Figures 5.73–5.84, 5.77–5.79). Logs show successful end-to-end runs, and the Nutrition Analysis (Chatbot) test suite (Table 6.10) passed common and edge scenarios (clear vs. blurry images, profile refresh, long inputs, temporary timeouts). This demonstrates reliable OCR → AI reasoning → personalized feedback during shopping.

**Objective 3: To design and implement a user-friendly interface with integrated Location-Based Services (LBS) to streamline the shopping process.**

Evaluation: Achieved. The Home and product experiences surface nearby stores with distance/ETA chips and filterable product feeds (Figures 5.55, 5.58–5.59). The Location Service obtains the current position with permission guidance and fallbacks, and server RPCs locate stores within a radius and the nearest store with stock (Figures 5.56–5.57). Users can select delivery locations via current location, typed address (geocoded), map pin, or precise coordinates (Figures 5.64–5.68, 5.89–5.90). Cart rules prevent mixing items from different stores and clearly notify users (Figures 5.60–5.63). Checkout supports Delivery or Pickup, payment selection, special instructions, and produces a pickup QR code for in-store fulfilment (Figures 5.93–5.100). All related use-case tests—Product Browsing & Search (Table 6.3), Cart Management (Table 6.4), Checkout (Table 6.5), Order Management (Table 6.6), Order Cancellation (Table 6.7), Profile and Address Management (Tables 6.8–6.9)—passed, confirming an intuitive LBS-enabled flow from discovery to fulfilment.

Overall conclusion. Across security, AI-assisted nutrition guidance, and LBS-enhanced shopping, the implemented features met the stated objectives and were verified by comprehensive system tests and in-app evidence.

### 6.4.2 User Acceptance Testing

User Acceptance Testing (UAT) is the final validation phase in the development lifecycle, involving direct participation from the system's intended end-users. This stage is critical to project success, as it ensures the application aligns with user requirements and expectations. For this evaluation, a group of three (3) testers was recruited to perform the acceptance test. The results from their evaluations are presented in the following section.

Below show three (3) results evaluated by tester after the user acceptance test:

**Application Tester 1**

Table 6.11 User Acceptance Testing Form (T001)

| **User Acceptance Testing**<br>Smart Shopping Assistant Mobile Application | | | |
|---|---|---|---|
| Name: Tan Kok Fu | | Position: IT student | |
| Date: 20 Sep 2025 | | Tester ID: T001 | |
| ** Rating 1 as lowest (worst), rating 5 as highest (excellent)<br>** The higher scale of rating indicates higher level of satisfaction | | | |
| **No.** | **Acceptance Criteria** | **Rating** | **Remarks** |
| **System Content** | | | |
| 1 | Readability | 4 | - |
| 2 | Position | 4 | - |
| 3 | Layout Consistency | 5 | The user interface (UI) and user experience (UX) are very well done. The icons and theme colours are consistent and visually appealing. |
| **System Functionality** | | | |
| 4 | Overall Functionalities | 4 | Functioning is good and easy to use. |
| 5 | Ease of Accessibilities | 4 | - |
| 6 | System Accuracy | 4 | - |
| 7 | Learnability of System | 4 | - |
| **System Performance** | | | |
| 8 | Response Time | 5 | Quick Response. |
| 9 | Processing Speed | 5 | - |
| 10 | Application Crash | 4 | - |
| 11 | Bugs free | 4 | - |
| 12 | Quality | 5 | High quality in UI design. |
| 13 | Reliability | 4 | - |
| 14 | Efficiency | 4 | AI chatbot play a good role in analysis the product nutrition. |
| **Others** | | | |
| 15 | Meeting Objectives | 4 | - |
| 16 | Security aspect | 5 | Secure login. |
| 17 | User friendliness | 5 | Application function is easy to use. |
| Comments/Suggestion:<br><br>The app works well. For future growth, consider partnering with local stores to offer special deals. Also, adding multilingual support (Malay, Chinese) would greatly improve usability for the local community. | | | |
| Actions taken by developer:<br><br>The suggestion has been noted for consideration in future system enhancements. | | | |

**Application Tester 2**

Table 6.12 User Acceptance Testing Form (T002)

| **User Acceptance Testing**<br>Smart Shopping Assistant Mobile Application | | | |
|---|---|---|---|
| Name: Tan Pei Ru | | Position: PR student | |
| Date: 20 Sep 2025 | | Tester ID: T002 | |
| ** Rating 1 as lowest (worst), rating 5 as highest (excellent)<br>** The higher scale of rating indicates higher level of satisfaction | | | |
| **No.** | **Acceptance Criteria** | **Rating** | **Remarks** |
| **System Content** | | | |
| 1 | Readability | 4 | - |
| 2 | Position | 3 | - |
| 3 | Layout Consistency | 5 | Theme color and icon is consistency |
| **System Functionality** | | | |
| 4 | Overall Functionalities | 4 | - |
| 5 | Ease of Accessibilities | 4 | - |
| 6 | System Accuracy | 4 | - |
| 7 | Learnability of System | 4 | - |
| **System Performance** | | | |
| 8 | Response Time | 4 | - |
| 9 | Processing Speed | 4 | - |
| 10 | Application Crash | 4 | - |
| 11 | Bugs free | 4 | - |
| 12 | Quality | 5 | Beautiful UI |
| 13 | Reliability | 4 | - |
| 14 | Efficiency | 4 | - |
| **Others** | | | |
| 15 | Meeting Objectives | 4 | - |
| 16 | Security aspect | 5 | Verification tokens secure the login process |
| 17 | User friendliness | 5 | Easy to use |
| Comments/Suggestion:<br><br>The application is very intuitive. To enhance the user experience further, I suggest implementing multilingual support. This would make the tool accessible to non-English speaking users and significantly increase its potential user base. | | | |
| Actions taken by developer:<br><br>The suggestion has been noted for consideration in future system enhancements. | | | |

**Application Tester 3**

Table 6.13 User Acceptance Testing Form (T003)

| **User Acceptance Testing**<br>Smart Shopping Assistant Mobile Application | | | |
|---|---|---|---|
| Name: Ng Hai Rou | | Position: Marketing student | |
| Date: 20 Sep 2025 | | Tester ID: T003 | |
| ** Rating 1 as lowest (worst), rating 5 as highest (excellent)<br>** The higher scale of rating indicates higher level of satisfaction | | | |
| **No.** | **Acceptance Criteria** | **Rating** | **Remarks** |
| **System Content** | | | |
| 1 | Readability | 4 | - |
| 2 | Position | 4 | - |
| 3 | Layout Consistency | 5 | - |
| **System Functionality** | | | |
| 4 | Overall Functionalities | 5 | - |
| 5 | Ease of Accessibilities | 5 | - |
| 6 | System Accuracy | 4 | - |
| 7 | Learnability of System | 4 | - |
| **System Performance** | | | |
| 8 | Response Time | 4 | - |
| 9 | Processing Speed | 4 | - |
| 10 | Application Crash | 4 | - |
| 11 | Bugs free | 4 | - |
| 12 | Quality | 5 | - |
| 13 | Reliability | 4 | |
| 14 | Efficiency | 5 | Application is function with efficiency. |
| **Others** | | | |
| 15 | Meeting Objectives | 5 | - |
| 16 | Security aspect | 5 | Login token increases the security for the user data. |
| 17 | User friendliness | 5 | Application is easy to use. |
| Comments/Suggestion:<br><br>The chatbot's ability to analyse nutrition is a powerful and highly useful function. To build on this success, I recommend two key enhancements: first, establishing collaborations with local grocery retailers to create a more integrated user experience. Second, setting up a direct user feedback channel within the application to streamline issue resolution and gather ideas for continuous improvement. | | | |
| Actions taken by developer:<br><br>The suggestion has been noted for consideration in future system enhancements. | | | |

# CHAPTER 7

# Conclusion and Recommendation

## 7.1 Conclusion

The development of the Smart Shopping Assistant mobile application was completed successfully and met all stated objectives. The system delivers a secure, privacy-centric shopping experience, combining password plus email one-time token authentication, user profile management, and protected data flows with a clean, task-oriented interface. Location-Based Services (LBS) reliably surface nearby stores and delivery options, while curated product feeds, category filters, and search streamline product discovery. End-to-end shopping is supported through a consistent cart, store-aware validation, an intuitive checkout (delivery or pickup), and QR-based pickup for in-store fulfillment.

A core contribution of the project is the AI-driven Nutrition Assistant, which transforms the app from a static catalogue into intelligent shopping assistance. Using on-device OCR to read ingredients and nutrition labels and fusing those signals with the user's health profile and cart context, the assistant proactively flags allergens and risks, recommends healthier or in-stock alternatives, and explains trade-offs at the moment of choice. This directly addresses the previously identified absence of intelligent shopping assistance, delivering personalized, in-flow guidance that helps users make healthier, better-informed purchases without leaving the shopping flow.

Comprehensive system testing across critical use cases—registration and login (with OTP verification), product browsing and search, cart management with cross-store constraints, checkout, order management and QR display, address management, profile updates, and chatbot interactions consistently implemented results. Observability via Flutter DevTools and backend metrics confirmed responsive network behavior and stable resource usage under typical loads, supporting a smooth user experience on target devices.

The project also navigated and resolved several technical challenges, including integrating OCR with AI analysis, connecting the database to the AI service, implementing token-based account verification, and leveraging geolocation for nearby-store discovery. Addressing these issues strengthened the solution's robustness and the team's fluency with Flutter, Supabase, and modern mobile AI patterns.

In summary, the Smart Shopping Assistant addressed security and privacy concerns through password and email one-time-token verification, authenticated data access, and protected data flows; resolved the absence of intelligent shopping assistance via an OCR- and AI-driven nutrition assistant that provided real-time, personalized guidance; and improved user experience and design efficiency with location-based store and address selection, single-store cart integrity, and a streamlined checkout. Comprehensive testing and operational metrics validated these outcomes across target devices and network conditions.

## 7.2 Recommendation

While the Smart Shopping Assistant met its stated objectives, the following targeted enhancements are recommended to strengthen impact and readiness for wider adoption.

**Partnership with Local Stores:**

Formalize integrations with partner retailers to enable real-time stock, price, and promotion sync (via POS/ERP or API/webhook), support loyalty IDs and digital receipts at checkout, and streamline in-store pickup through QR verification at the counter. Begin with a limited pilot (one to two stores), define SLAs and data-sharing agreements, add an admin "sync health" dashboard (last sync time, failures, retries), and implement queue-backed ingestion to ensure resilience during peak periods.

**Continuous Performance Optimization:**

Adopt a performance budget and ongoing monitoring (e.g., Crashlytics/Sentry, Supabase metrics). Prioritize image and asset delivery via CDN with responsive thumbnails, pagination and selective projections for product queries, Postgres indexing (BTREE/GIN/ GiST for geospatial), and caching of geocoding and repeated AI analyses. Reduce cold start cost with deferred initialization, prefetch critical assets, and conduct periodic load tests to validate latency SLOs under growth.

**Multilingual Support:**

Introduce full internationalization with dynamic language switching (e.g., Malay, English, Chinese, Tamil). Localize units, currency, dates, and nutrition terminology; avoid hard-coded strings by using ARB resource files and a translation workflow. Ensure search and category filters are language-aware and provide accessibility-ready labels/alt text to maintain usability across locales.

**User Feedback:**

Embed lightweight in-app feedback (bug report + screenshot), task-based micro-surveys after key flows (checkout, address add, chatbot reply), and periodic NPS. Instrument analytics for funnel diagnostics (search → add-to-cart → checkout), publish

release notes/changelog, and use feature flags with A/B tests to validate UI or recommendation changes before full rollout.

These recommendations focus on operational robustness, inclusivity, and evidence-driven iteration, positioning the application for sustainable scale and improved user satisfaction.

# References

[1] D. Faulds, W. Mangold, P. Raju, and S. Valsalan, "The mobile shopping revolution: Redefining the consumer decision process," *Business Horizons*, vol. 61, pp. 323–338, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0007681317301672?via%3Dihub

[2] S. Sanna, D. Soi, D. Maiorca, G. Fumera, and G. Giacinto, "A risk estimation study of native code vulnerabilities in Android applications," *ArXiv*, abs/2406.02011, 2024. [Online]. Available: https://academic.oup.com/cybersecurity/article/10/1/tyae015/7744932

[3] T. M. *et al.*, "Threats and vulnerabilities of mobile applications," *Research Anthology on Securing Mobile Technologies and Applications*, 2021. [Online]. Available: https://doi.org/10.4018/978-1-7998-3479-3.ch034

[4] U. Kishnani, N. Noah, S. Das, and R. Dewri, "Assessing Security, Privacy, User Interaction, and Accessibility Features in Popular E-Payment Applications," *Proceedings of the 2023 European Symposium on Usable Security*, 2023. [Online]. Available: https://doi.org/10.1145/3617072.3617102

[5] R. Pillai, B. Sivathanu, and Y. Dwivedi, "Shopping intention at AI-powered automated retail stores (AIPARS)," *Journal of Retailing and Consumer Services*, vol. 57, p. 102207, 2020. [Online]. Available: https://doi.org/10.1016/j.jretconser.2020.102207

[6] V. Chattaraman, W. Kwon, K. Ross, J. Sung, K. Alikhademi, B. Richardson, and J. Gilbert, "'Smart' choice? Evaluating AI-based mobile decision bots for in-store decision-making," *Journal of Business Research*, 2024. [Online]. Available: https://doi.org/10.1016/j.jbusres.2024.114801

[7] Q. Chen, C. Chen, S. Hassan, Z. Xing, X. Xia, and A. E. Hassan, “How should I improve the UI of my app? A study of user reviews of popular apps in the Google Play,” ACM Trans. Softw. Eng. Methodol., vol. 30, no. 3, Art. no. 37, 2021. [Online]. Available: https://doi.org/10.1145/3447808

[8] Ontrack Digital Sdn Bhd, "Hargapedia - Compare Prices!" *App Store*. [Online]. Available: https://apps.apple.com/my/app/hargapedia-compare-prices/id1315874967. [Accessed: Aug. 30, 2024].

[9] Foodpanda, "foodpanda: Food & Groceries," *App Store*. [Online]. Available: https://apps.apple.com/us/app/foodpanda-food-groceries/id758103884. [Accessed: Aug. 30, 2024].

[10] Maplebear Inc, "Instacart Shopper: Earn money," *App Store*. [Online]. Available: https://apps.apple.com/us/app/instacart-shopper-earn-money/id1454056744. [Accessed: Aug. 30, 2024].

[11] Figma, "Figma – The collaborative interface design tool." [Online]. Available: https://www.figma.com/. [Accessed: Aug. 30, 2024].

[12] Flutter, "Flutter - Build apps for any screen." [Online]. Available: https://flutter.dev/. [Accessed: Aug. 30, 2024].

[13] Supabase, "The Postgres Development Platform." [Online]. Available: https://supabase.com/. [Accessed: July 11, 2025].

[14] Dify, "The Open-Source LLM App Development Platform." [Online]. Available: https://dify.ai/. [Accessed: Feb. 2, 2025].

[15] Google, "ML Kit." [Online]. Available: https://developers.google.com/ml-kit. [Accessed: July 22, 2025].

[16] Google, "Google Maps Platform." [Online]. Available: https://mapsplatform.google.com/. [Accessed: July 22, 2025].

## Appendix A: Poster