DEVELOPMENT OF E-WALLET APPLICATION: SHARED WALLET AND GROUP BILL SPLITTING FEATURES

BY

JOE TAY ZHI FENG

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF INFORMATION SYSTEMS (HONOURS) INFORMATION SYSTEMS ENGINEERING

Faculty of Information and Communication Technology

(Kampar Campus)

FEBRUARY 2026

COPYRIGHT STATEMENT

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Information Systems (Honours) (Information Systems Engineering) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project report represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project report 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.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Mr. Phan Koo Yuen, who gave me this great opportunity to create the E-Wallet application: Shared Wallet and Group Bill Splitting features. This is my first step to establish a career in mobile application development and this will be very helpful for my future. Thank you very much.

ABSTRACT

This project addresses three critical gaps in existing e-wallet platforms: the lack of effective group financial management tools, the absence of integrated bill-splitting functionality for social activities, and insufficient multi-currency support for internationally mobile users. Following a systematic literature review of existing e-wallet systems and collaborative financial management tools, a fully functional mobile e-wallet application was developed targeting the iOS platform using Flutter, Node.js with Express, and MySQL. The application delivers five core modules. Individual wallet management supports nine international currencies with Stripe-integrated top-up, peer-to-peer transfers, QR code payments, and live-rate currency conversion. The shared wallet module enables group fund management with configurable transfer limits, real-time transaction tracking, and a consensus-based closure mechanism with four fund distribution strategies. The group bill splitting module supports manual entry and AI-powered receipt scanning via the Claude API, with item-level allocation and automatic in-app payment settlement. A Transfer Calculator presents real-time exchange rates and a seven-day historical trend chart via the ExchangeRate API. A contact management system prioritizes added contacts across shared wallet and bill splitting workflows. Security is enforced across two independent layers: account-level verification via Google ML Kit liveness detection, and transaction-level authentication via PIN or iOS native Face ID. Registration is secured through Firebase Phone Authentication OTP verification and bcrypt password hashing, with session tokens persisted in encrypted local storage. The system was validated through an incremental development model across four sequential increments, with functional testing conducted against all twenty-five defined use

Area of Study (Minimum 1, Maximum 2): Mobile Application, Financial Technology

Keywords (Minimum 5, Maximum 10): E-Wallet, Bill Splitting, Shared Wallet, Multi-Currency Management, Group Financial Management, Mobile Payment, Cross-Border Transactions

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ACID	Atomicity, Consistency, Isolation, Durability
ADB	Asian Development Bank
API	Application Programming Interface
ASEAN	Association of Southeast Asian Nations
AUD	Australian Dollar
BTC	Bitcoin
CAD	Canadian Dollar
CAGR	Compound Annual Growth Rate
CNY	Chinese Yuan
CORS	Cross-Origin Resource Sharing
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheets
ETH	Ethereum
EUR	Euro
GBP	British Pound Sterling
GrabPay	Grab Payment (regional e-wallet)
GUI	Graphical User Interface
HKD	Hong Kong Dollar
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I/O	Input/Output
ID	Identifier
IDE	Integrated Development Environment
IDR	Indonesian Rupiah
iOS	iPhone Operating System
IP	Internet Protocol
JSON	JavaScript Object Notation
JPY	Japanese Yen
KYC	Know Your Customer
ML	Machine Learning

MYR	Malaysian Ringgit
NFR	Non-Functional Requirements
npm	Node Package Manager
OCR	Optical Character Recognition
OS	Operating System
OTP	One-Time Password
PCI DSS	Payment Card Industry Data Security Standard
PIN	Personal Identification Number
QR	Quick Response
RESTful	Representational State Transfer
SDK	Software Development Kit
SGD	Singapore Dollar
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SSD	Solid State Drive
SUS	System Usability Scale
TAM	Technology Acceptance Model
THB	Thai Baht
TTL	Time To Live
UC	Use Case
UI	User Interface
USD	United States Dollar
UTAUT	Unified Theory of Acceptance and Use of Technology
UX	User Experience
VND	Vietnamese Dong
2FA	Two-Factor Authentication
3NF	Third Normal Form

CHAPTER 1 INTRODUCTION

The quick growth of mobile technology has changed the way people manage their money in a big way. E-wallets, or electronic wallets, have become one of the most popular financial technologies because they make it easy and safe for people to make digital transactions. Statista says that the total value of transactions in the digital payments market will reach USD 16.62 trillion by 2028, growing at a compound annual growth rate (CAGR) of 9.52% from 2024 [1]. At the same time, the number of people who use digital wallets around the world is expected to reach more than 5.2 billion by 2026. This shows that mobile-based financial services are becoming more popular [2]. Even though e-wallet apps are growing quickly, many of them still don't meet the collaborative and international financial needs of today's users.

This chapter explains why the proposed e-wallet app was made and what it is based on. The app has a Shared Wallet and Group Bill Splitting feature set. The next sections will talk about the problem statement, the goals of the project, the scope of the work, and what the work is expected to add.

1.1 Project Background

Digital payment systems have become much more popular in the last ten years. The World Bank's 2022 Global Findex Database says that 76% of adults around the world now have a financial account, up from 51% in 2011. Mobile money accounts are a big part of this growth, especially in developing economies [3]. The e-wallet market has grown the fastest in Southeast Asia. The e-Conomy SEA Report by Google, Temasek, and Bain & Company (2023) said that digital financial services in the region are on track to serve 300 million people who don't have access to them by 2025 [4].

Even though there are a lot of e-wallet apps out there, there is a big hole in group-oriented financial management. PayPal, Venmo, and GrabPay are some of the most popular platforms for individual and peer-to-peer transactions. They don't offer much built-in support for collaborative financial management [5]. When people share expenses, like when housemates split the cost of utilities or travellers split the cost of

their stay, they often have to use more than one app at the same time, which makes things more complicated and increases the chances of financial errors.

Cross-border financial transactions make the user experience even more difficult. According to Bank Negara Malaysia's Payment Systems Report (2023), retail e-payment transactions in Malaysia grew by 23.6% year-on-year to reach 11.7 billion transactions in 2022 [6]. This shows that people in Malaysia are using these systems a lot. The same report also said that cross-border retail payment interoperability is still a work in progress, and that managing multiple currencies is still hard for most people [6].

The proposed application fills these gaps by providing a single mobile platform built with Flutter on the front end and Node.js with MySQL on the back end. This platform includes individual and shared wallets that support nine international currencies, AI-powered receipt scanning for splitting bills among groups, real-time exchange rate tracking through the ExchangeRate API, QR code payment, and Stripe-integrated top-up. The app also has a system for managing contacts and recipients, face recognition liveness detection for account verification, and transaction-level security through PIN and device biometric authentication. These features work together to make a complete solution for managing money across borders, in a secure way, and with others.

1.2 Problem Statement

The development of this application is motivated by three key problem areas identified through an analysis of existing e-wallet ecosystems and user behavioral patterns.

1.2.1 Lack of effective group financial management tools

Many of the e-wallets and financial management tools that are available today were designed for individuals and don't provide much, if any, functionality for families, couples or groups to manage their joint expenses. Salmony and Harald (2010) argue that, although peer-to-peer payment systems are now mature, the collaborative aspect of group financial management, where multiple contributors and withdrawals are made from a common pot, has not been addressed by mainstream platforms [9]. It is very difficult for users to keep track of contributions fairly, monitor expenditures in real time and ensure transparency among all participants in group activities. Gupta and Arora (2017) also conducted a study which pointed out usability constraints and lack of feature completeness as important drivers of user dissatisfaction in mobile payment applications [10].

The lack of specific collaborative wallet features, such as adjustable daily or transaction-based transfer limits, group consensus protocols for wallet closure, and intelligent algorithms for funds allocation, aggravates the challenge experienced by user groups in creating a fair and verifiable shared financial environment. Without those controls groups are vulnerable to overspending, unequal contributions, and disputes that erode trust among members [9].

1.2.2 Lack of bill splitting functionality in social events

Most e-wallet apps don't have advanced features for splitting bills. Users have to do math by hand, switch between different apps, or use third-party bill-sharing services that don't work with their payment wallets. Riquelme and Rios (2010) discovered that usability complexity and the necessity to transition among various applications constitute substantial impediments to the ongoing adoption of mobile payments [11]. Most current solutions only support basic equal-share division and don't automatically take into account taxes, service charges, or item-specific selections. This makes the process less efficient for groups with different consumption patterns.

Some bill-splitting apps like Splitwise offer partial solutions, but they usually don't work with payment wallets, so users have to switch between platforms to settle their debts.

Combining Optical Character Recognition (OCR) with AI-powered document parsing is a useful way to make group cost settlement less work for people. Previous studies on scanned receipt OCR and information extraction show that modern document understanding systems can reliably extract structured receipt data, making it possible to automate the process of itemizing bills [12]. Users can easily go from capturing a receipt to settling a payment in one app by adding this kind of technology directly to the payment workflow.

1.2.3 Insufficient internationalization support and inconvenient currency management

Most e-wallet platforms don't have strong internationalization features, like support for multiple currencies and real-time integration of exchange rates. Schuh and Stavins (2010) observed that the efficacy of cross-border digital payment tools is fundamentally reliant on the clarity and availability of currency conversion systems for end users [13]. Users who do business across borders or travel internationally often have trouble with foreign currencies. They often have to use outside tools to get exchange rate information or do manual conversions, which is inconvenient and increases the risk of making a financial mistake.

The quick rise in travel within ASEAN and digital commerce across borders makes this problem even worse for Malaysian users. The Asian Development Bank (2022) says that the lack of payment tools that work with multiple currencies is still a big problem for smooth digital financial participation in the ASEAN region [14]. This limitation is directly addressed by a single platform that keeps separate currency wallets for different international currencies and tracks exchange rates in real time.

1.3 Motivation

The impetus for this project stems from the increasing disparity between users' collaborative financial practices and the individual-focused design of prevalent e-wallet applications. In everyday life, financial transactions are rarely limited to one person. For example, housemates share rent and utilities, travel groups split the costs of lodging and transportation, and dining groups often have one person pay for everything and then get reimbursed. Even though these kinds of cooperative financial behaviours are common, most e-wallet systems don't have a built-in way to support them.

Another reason is that fragmented multi-application workflows are not very efficient. When users have to switch between messaging apps, calculators, and payment apps, the whole process takes longer and is more likely to go wrong. Studies have consistently demonstrated that usability issues and disjointed workflows adversely affect user satisfaction and adoption in mobile financial applications [7].

Data specific to Malaysia further supports the importance of this project. In 2022, Bank Negara Malaysia said there were 11.7 billion e-payment transactions, which was a 23.6% increase from the previous year [6]. The Department of Statistics Malaysia also said that in 2022, Malaysian residents took 8.1 million trips outside the country, many of which involved managing money in more than one currency [8]. These trends show that there are a lot of people who use e-wallets and are active online, but current e-wallet solutions don't meet their needs.

Lastly, improvements in easily accessible third-party APIs give this project a strong technological base. The Claude API for AI-based receipt parsing, the ExchangeRate API for real-time currency data, Stripe for adding wallet top-up functionality, and Firebase for authentication make it possible to build a feature-rich and scalable application within the limits of an academic project. The main reason for this work is the coming together of user demand and technological readiness.

1.4 Project Scope

This project includes the full-cycle development of a mobile e-wallet app for iOS using the Flutter framework. The main parts of the project are as follows:

- Individual wallet management that supports nine international currencies (MYR, USD, SGD, JPY, HKD, THB, AUD, GBP, CAD). You can add money to your wallet using Stripe, send money to other people, scan a QR code to pay, or convert currencies.
- Shared Wallet module that lets you create groups, invite members from your contact list, set transfer limits, track transactions in real time with cashflow analytics (a pie chart), and close the wallet based on consensus with multiple fund distribution strategies (equal split, full transfer, proportional, and smart distribution based on contribution history).
- The Group Bill Splitting module lets you create bills by hand and scan receipts with AI (via the Claude API) to automatically get merchant information, line items, quantities, prices, and taxes. Recipients choose what they want to eat on their own, and the system automatically figures out how much to charge and takes it off their bill.
- A multi-currency transfer calculator with a seven-day historical exchange rate line chart, powered by ExchangeRate API, that lets users guess how much money they will make before they make a transaction.
- Contact and recipient management lets users add registered contacts who are at the top of the list for shared wallet member selection and bill splitting.
- Google ML Kit liveness detection checks the account by using the front camera to make sure there is a real person with open eyes. When you pass, the backend marks the account as verified, which lets you create a multi-currency wallet. Unverified accounts can only have one main currency wallet, which is based on the country where the account was created.
- Transaction-level security is required for Transfer and Convert operations. Users can set a numeric PIN and use iOS native Face ID (via local_auth) if they want to. This is completely different from the ML Kit account verification. It

uses the device's built-in biometric system and works entirely on the device without any server involvement.

- Security infrastructure that includes Gmail-based email and password authentication with bcrypt password hashing, Firebase Phone Authentication for OTP-based phone verification during registration, face recognition liveness detection for account verification, and PIN or device biometric verification for sensitive transactions.

The project doesn't include web-based interfaces, cryptocurrency transactions, managing an investment portfolio, or integrating physical point-of-sale hardware.

1.5 Project Objectives

The project is based on three main goals, each of which has its own set of sub-goals that help with the implementation of each feature.

1.5.1 Project Objective 1

Main Objective:

To develop a mobile e-wallet application that facilitates group financial management

Sub Objectives:

1. Develop a multi-person Shared Wallet feature to enhance financial collaboration and enable group users to manage shared expenditures effectively and transparently.
2. Implement real-time expense tracking and recording to minimize financial errors and ensure all group members have visibility over contributions and spending.
3. Develop a dynamic notification system to improve usability and transparency by providing timely updates on shared wallet activities and financial status.
4. Develop a settlement mechanism activated upon shared wallet closure, offering residual settlement (distribution of the remaining balance only), with user consensus required for the selected distribution strategy.
5. Develop a contact and recipient management feature enabling users to add registered contacts, who are subsequently prioritized in shared wallet member

selection and bill splitting recipient lists to reduce friction during group financial activities.

1.5.2 Project Objective 2

Main Objective:

To enable group bill splitting for social activities

Sub-Objectives:

1. Develop bill uploading and intelligent recognition functionality allowing users to photograph or upload physical and electronic receipts, with automatic extraction of merchant names, item details, quantities, prices, and taxes via the Claude API.
2. Develop a flexible manual bill creation function to accommodate complex or personalized cost-sharing scenarios across diverse social contexts.
3. Design a precise bill allocation confirmation mechanism where recipients independently select their consumed items, with the system automatically computes each recipient's payable amount inclusive of applicable taxes, excluding the bill creator who has already settled the full amount on behalf of the group.
4. Develop an automatic debit and transfer module that deducts the confirmed amount from the recipient's e-wallet and transfers it to the bill initiator's account in real time upon payment confirmation.

1.5.3 Project Objective 3

Main Objective:

To develop a wallet that supports internationalization.

Sub-Objectives:

1. Develop multi-currency support enabling users to create and operate wallets in nine international currencies to meet the needs of internationally mobile users.
2. Develop a currency transfer calculator that shows real-time exchange rates and a seven-day historical trend line chart. This will let users figure out how much money they will need to convert before they make a transaction.

3. Develop a currency wallet management system that makes separate currency wallets for each supported currency. This will make it clear what each balance is and avoid confusion when switching between currencies.
4. Develop transaction features that let you exchange currencies, add money to your wallet through Stripe, and send money to other people at any time. This will make managing your money across borders more flexible and easy.

1.6 Contribution

This project adds the following to the field of mobile financial technology:

- A single app that combines personal wallet management, group shared wallets, and AI-powered bill splitting into one platform. This means that users don't have to keep track of multiple financial tools at the same time.
- A new shared wallet design that includes democratic closure consensus, customizable transfer limits, and data-driven fund distribution strategies. One of these strategies is a smart distribution algorithm that takes into account each member's past contributions and spending.
- An AI-assisted receipt parsing pipeline that uses the Claude API to automatically pull structured billing information from photos, cutting down on manual data entry and the amount of work that end users have to do.
- A financial management system that supports nine international currencies and has live exchange rate integration. This makes it easy to manage cross-border transactions from a single mobile interface.
- A multi-layer security architecture that uses bcrypt for Gmail and password authentication, Firebase Phone Authentication OTP for registration, Google ML Kit liveness detection for account-level verification (which unlocks multi-currency access), and iOS native Face ID via local_auth for transaction-level authorization. The account verification and transaction biometric work as two separate systems.
- A contact management system that makes it easier for groups to work together on money matters by keeping a prioritized list of recipients that is used consistently across workflows for creating shared wallets and splitting bills.

1.7 Report Organization

The remainder of this report is organized as follows:

Chapter 1 introduces the project by presenting the project background, problem statement, motivation, project scope, project objectives, and contributions of the proposed e-wallet application.

Chapter 2 presents a Literature Review of existing e-wallet systems, collaborative financial management tools, bill-splitting applications, and multi-currency mobile payment platforms. Relevant academic and industry literature is critically evaluated to establish the theoretical and practical context for this work.

Chapter 3 talks about the System Methodology. It includes the system architec ture diagram, use case diagrams with their descriptions, and activity diagrams that sho w the main workflows of the application.

Chapter 4 is about System Design. It talks about the system block diagram, the specifications for the components, and how the different parts of the system work together.

Chapter 5 talks about the System Implementation, which includes setting up the software, configuring it, taking screenshots of how it works, and talking about the problems that came up during development.

Chapter 6 is about System Evaluation and Discussion. It covers testing methods, performance metrics, test results, project problems, and whether the goals were met.

Chapter 7 gives the Conclusion and Recommendations. It sums up what happened in the project and suggests ways to move forward.

CHAPTER 2 LITERATURE REVIEW

2.1 Overview

The widespread use of mobile payment technology has led to a wide range of e-wallet apps, each of which meets a different set of transactional needs. The academic literature consistently identifies a structural limitation in this ecosystem: most commercially deployed platforms are optimized for individual, bilateral transactions and have not evolved to support the collaborative and multi-currency financial scenarios that characterize modern group activities [5]. Dahlberg et al. (2008) demonstrated that the design trajectory of mobile payment systems has traditionally been influenced by the interests of banks and payment processors, rather than the collaborative needs of end-users. This elucidates the persistent underdevelopment of group-oriented features, despite the technological capability for their implementation [5].

This chapter gives a critical look at the literature and systems that are already in place that are related to the proposed application. The review is organized like this: Section 2.2 looks at the theoretical underpinnings of e-wallet technology and how it is classified. Section 2.3 critically examines six representative existing applications; Section 2.4 provides a comparative feature analysis; Section 2.5 delineates the limitations of current solutions and existing research; Section 2.6 explains how the proposed system mitigates these deficiencies; and Section 2.7 encapsulates the principal findings of this review.

2.2 E-Wallet Fundamentals

2.2.1 What is E-Wallet?

E-wallets have changed from simple payment storage systems to full-fledged financial platforms that support a wide range of transactional, identity, and financial management functions [15]. This is part of a larger trend toward full-fledged digital financial ecosystems. An e-wallet is a software-based system that safely stores payment information and lets you make electronic payments without having to enter sensitive information over and over again [15]. But this technical definition downplays how important e-wallets are to modern efforts to include everyone in the financial system.

Kim, Mirusmonov, and Lee (2010) empirically established that perceived usefulness and ease of use are the primary determinants of mobile payment adoption, with the diversity of supported use cases serving as a moderating variable [16]. This discovery has important effects on platform design: e-wallets that can handle a wider range of financial tasks, such as managing expenses with others and making transactions in multiple currencies, are more likely to keep users interested over time. This view is supported by the global adoption trend; Statista predicts that by 2026, there will be more than 5.2 billion digital wallet users around the world [2]. This means that the competitive edge between platforms will depend more on the depth of their features than on their basic payment capabilities.

Modern e-wallets now do more than just let you pay for things. They also let you verify your identity, join loyalty programs, and do financial analytics [15]. However, the literature review in this chapter shows that collaborative financial management, in which several users manage, contribute to, and withdraw from a shared pool, is still a largely unexplored area in the design of commercial e-wallets.

2.2.2 How E-Wallets Work?

To make good design choices for the proposed system, it's important to understand how e-wallets work. There are two main steps to using an e-wallet: registration and transaction processing [17]. Each step has its own security and user experience needs that affect how the platform is built.

When users sign up, they give the system their personal and financial information, which the system uses to create a secure account that is linked to a bank. At this point, modern platforms combine Know Your Customer (KYC) verification with device-based authentication [19]. From a research standpoint, Bonneau et al. (2012) identified that the friction associated with registration processes is a crucial factor influencing initial adoption rates. This finding supports the proposed system's implementation of Firebase Phone Authentication to reduce registration complexity while ensuring security [18].

There are four steps in the transaction phase: authentication, starting the transaction, processing the payment, and finishing the transaction [17]. Each step in this sequence is a possible point of friction, which is very important. Singh (2009) found that the more steps it takes to make a payment, the less satisfied users are [17]. This supports the proposed system's design goal of allowing bill splitting and payment settlement to happen in one application flow instead of making users switch between platforms.

2.2.3 Types of E-Wallets

Research delineates a distinct taxonomy of e-wallet systems predicated on operational scope, merchant network accessibility, and regulatory stipulations. According to the old way of classifying wallets, there are three main types: closed wallets, semi-closed wallets, and open wallets [19]. Closed wallets work in limited ecosystems where only certain companies can use wallets to buy their own goods and services. Examples include Amazon Pay and Starbucks' in-store payment system. Semi-closed wallets, like Paytm, let you make purchases from more than one approved merchant while still limiting cash withdrawals. Open wallets let you store money and make transactions with many merchants and platforms without any network restrictions. PayPal is a good example of this type of wallet.

Table 2.2.3.1 Advantages and Limitations of E-Wallet Types

Wallet Type	**Advantages**	**Limitations**
Closed Wallets	• High integration with company services • Enhanced loyalty programs • Simplified user experience • Lower regulatory burden	• Limited to single company ecosystem • No cash withdrawal capability • Restricted merchant acceptance • Limited interoperability
Semi-Closed Wallets	• Multiple merchant acceptance • Moderate implementation complexity • Balanced regulatory requirements • Enhanced user flexibility	• No cash withdrawal capability • Restricted to authorized networks • Limited international acceptance • Dependency on merchant agreements
Open Wallets	• Universal merchant acceptance • Full cash withdrawal capability	• Complex regulatory compliance • Higher implementation costs

	• Comprehensive functionality • International transaction support	• Extensive security requirements • Increased operational complexity

2.2.4 Summary on E-Wallet Types

The classification of e-wallet types shows a basic trade-off between having control over the ecosystem and being able to change its functions. Closed and semi-closed wallets are simple and safe because they limit their use, but this limitation makes them structurally unsuitable for group financial management situations that require participation from users in different ecosystems. Open wallets, on the other hand, let people send money to each other without any restrictions and work with merchants, which is what you need for collaborative financial use cases.

But there is a big hole in the open wallet category itself. Open wallets enable unrestricted bilateral transfers; however, they have not developed to accommodate multi-party financial structures, including shared pools, group spending limits, or consensus-driven fund distribution. Gupta and Arora (2017) said that this gap was caused by the fact that commercial platforms tend to focus on transaction volume rather than depth of collaborative financial management when developing new features [10]. The application analysis in Section 2.3 backs this up directly. It shows that none of the open wallet platforms we looked at, like PayPal and Revolut, have built-in shared wallet or item-level bill splitting features.

The Payment Systems Report (2022) from Bank Negara Malaysia also said that while more people in Malaysia are using e-payments, the platforms that are available don't have enough interoperability and collaborative features for a user base that moves around the region [6]. This context makes the proposed system even more relevant. It uses an open wallet architecture and adds purpose-built collaborative and multi-currency features that are specifically designed for Malaysian and ASEAN users.

2.3 Existing Application

This part looks closely at six existing apps that are useful for e-wallets, managing group finances, and splitting bills. Instead of just listing the features of each platform, the analysis puts each application in the context of academic literature to find out what theoretical ideas went into its design choices, what problems it has, and what needs it leaves that the proposed system meets. We chose these six platforms because they cover all the bases when it comes to useful features: N26 is for dedicated banking with shared accounts, Wallet by BudgetBakers is for financial planning with group features, Splitwise and Tricount are for splitting bills, Revolut is a full-featured multi-currency e-wallet, and PayPal is the most popular payment platform in the world.

2.3.1 N26 Shared Spaces

N26 Shared Spaces [20] is one of the few commercial uses of a collaborative sub-account in a popular banking app. With this platform, two or more people with N26 accounts can share a sub-account, add money to it, and see the total balance in real time. Its connection to individual N26 accounts gives a single view of both personal and shared finances, which meets the need for transparency that Wilkinson and Young (2002) said was key to user satisfaction in shared financial arrangements [21].

Even though N26 Shared Spaces has a strong concept, it has some big problems that make it hard to use. Participation necessitates that all members possess an active N26 account—a design choice that, although rational from a banking compliance and identity verification standpoint, poses a considerable obstacle to collaborative adoption. Ondrus and Pigneur (2006) observed that subordinating group participation to a singular platform's ecosystem significantly constrains reach [22]. This limitation is probably not an oversight in design, but rather a deliberate choice by regulators and risk managers. However, from a user experience point of view, the platform doesn't have any tools for automatically splitting expenses, managing itemized bills, or setting spending limits, so users still have to do the math themselves to figure out how much each person owes. The proposed system aims to bridge this usability gap with its shared wallet module, which automatically tracks contributions, lets users set limits on how much they can transfer per transaction, and closes transactions based on consensus.

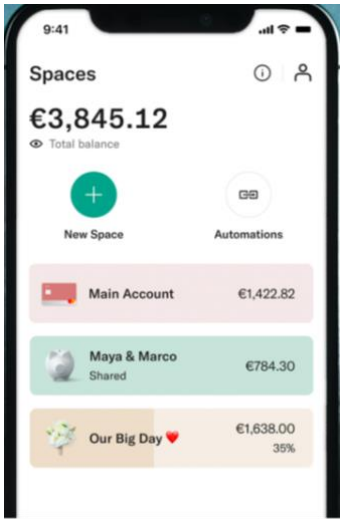

Figure 2.3.1.1 N26 Shared Spaces interface [20]

2.3.2 Wallet by BudgetBakers

BudgetBakers' Wallet [23] is a tool for managing personal finances that lets you track your budget with other people at more than 3,500 banks around the world. Brush and Inkpen (2007) said that good shared financial tools need real-time visibility, role-based access control, and an auditable transaction history. Its Group Sharing feature is an example of these principles [24]. Wallet by BudgetBakers does well on these points: participants can keep an eye on shared accounts at different times, and the difference between administrators and read-only members helps keep group governance clear.

But there is a big problem with Wallet by BudgetBakers that makes it not a good choice for an integrated payment solution: it can't hold money or make direct payments. The app only works as a financial planning tool on top of existing bank accounts. All payments must be made through separate platforms. Chawla and Joshi (2019) discovered that this type of platform separation, wherein expense tracking and payment execution transpire in distinct applications, serves as a principal catalyst for user dissatisfaction and abandonment in mobile financial management scenarios [7]. The paywall that only lets Premium subscribers share groups makes this accessibility problem even worse. The proposed system directly addresses this issue by combining fund storage, group tracking, and payment execution into a single platform. This means that Wallet by BudgetBakers doesn't have to rely on an outside settlement process.

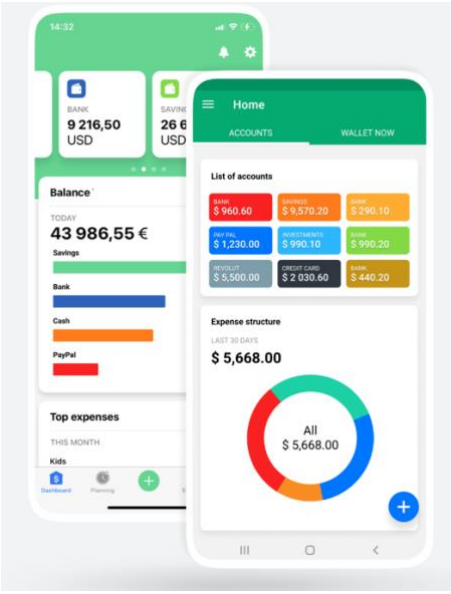

Figure 2.3.2.1 Wallet by BudgetBakers main dashboard interface [23]

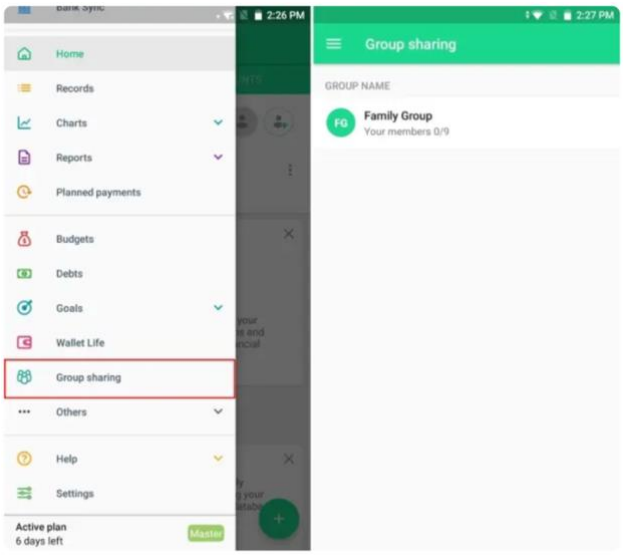

Figure 2.3.2.2 Wallet by BudgetBakers group sharing [23]

2.3.3 Splitwise

Splitwise [25] is the most popular app for sharing expenses. It automatically calculates debts, gives suggestions for settling them with the least amount of money, and supports multiple currencies. The main idea behind its design is to cut down on the number of transactions between parties needed to settle group debts. This is in line with the cognitive load reduction principle, which was found to be a key factor in getting people to use group payments [11]. Splitwise works well for simple expense tracking among groups with similar spending habits, and it is the current standard for standalone bill-splitting tools.

However, a major architectural flaw makes Splitwise less useful as a complete financial solution: it is a record-keeping system, not a payment platform. To pay off debts recorded in Splitwise, users must use separate channels. This creates the same kind of inter-application friction that Zhou, Lu, and Wang (2010) found to be a major problem for mobile financial workflows [26]. Also, with Splitwise's expense entry model, you have to type in each item and amount by hand. There is a premium receipt scanning feature, but it only extracts the total amount and not the item-level breakdown with tax apportionment. This makes it structurally inadequate for dining situations where people eat different things at different prices. The proposed system fixes both problems: it combines payment processing with expense tracking in the same workflow, and its AI-powered receipt scanning automatically extracts individual line items and taxes—things that Splitwise's architecture can't do without a major redesign.

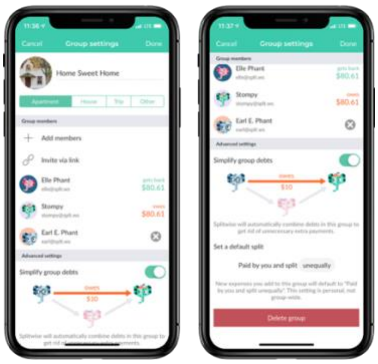

Figure 2.3.3.1 Splitwise expense tracking and balance calculation interface [25]

2.3.4 Tricount

Tricount [27] is a group expense management app that focuses on a specific area: sharing costs for travel and events where internet access may not always be available. Its offline-first architecture, which lets you record expenses even when you're not connected to the internet and automatically syncs when you reconnect, is a practical engineering solution to real-world usage conditions. Multi-currency support and the ability to export reports in PDF and Excel formats are real design strengths for international travel situations. They show that the tool is more focused on the context than most of its competitors.

Even though these contextual strengths are there, Tricount has the same structural problem as all other pure expense-tracking apps: it can't hold money or make payments. It works like a ledger instead of a wallet, and all payments must be made through other platforms. Riquelme and Rios (2010) identified this separation as a major usability issue, pointing out that making users switch platforms to finish a financial task greatly lowers the number of people who actually do it [11]. Also, the interface gets more complicated as the group size grows, and advanced multi-currency features need a paid subscription, which makes them less accessible to casual users. The suggested system gets around these problems by combining expense tracking, itemised bill splitting, and direct in-app payment settlement. This gets rid of the need for an outside platform, which limits both Tricount and Splitwise.

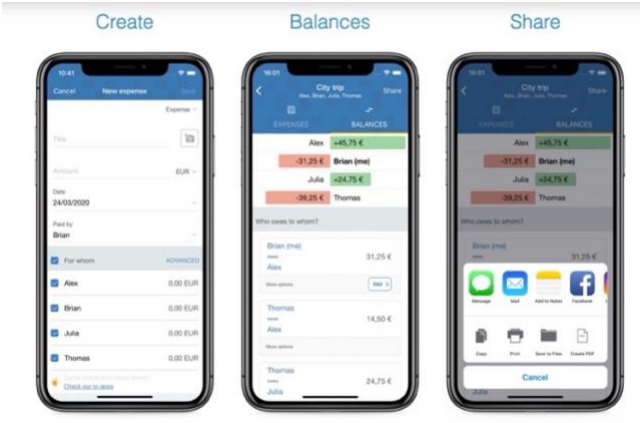

Figure 2.3.4.1 Tricount group creation, balance tracking, and sharing features [27]

2.3.5 Revolut

Revolut [28] is the most advanced platform in this comparison. It has multi-currency accounts that support up to 25 currencies, real-time interbank exchange rates, spending analytics, and a wide range of investment and savings features. Schuh and Stavins (2010) identified real-time exchange rate transparency as a critical determinant of user trust in cross-border payment instruments [13], and Revolut's implementation of live rate display directly addresses this requirement. Its full range of personal finance tools, such as instant currency conversion, virtual cards, and detailed spending breakdowns, make it a real strength that makes it the most feature-rich individual e-wallet among the platforms we looked at.

But a close look shows that Revolut's design is mostly focused on the individual. Its bill-splitting feature only lets you split the bill evenly among contacts, and it doesn't let you choose items, divide taxes, or use AI to read receipts. There is also no shared wallet feature for managing funds as a group. From a business point of view, this design choice makes sense: collaborative features add a lot of liability, regulatory, and dispute-resolution problems that Revolut may have decided to put off in order to grow its core financial services. Ondrus and Pigneur (2006) noted that the preference for high-volume individual transactions over collaborative structures is a prevalent trend among commercially dominant platforms [22]. Also, Revolut isn't licensed to offer financial services in Malaysia, which makes it less useful for the people this project is aimed at. These limitations are not just design flaws; they are well-thought-out trade-offs that leave a clear gap for a platform that is primarily focused on collaborative financial management.

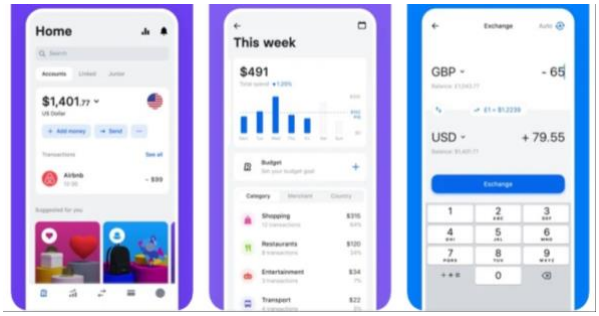

Figure 2.3.5.1 Revolut account management, spending analytics, and currency exchange [28]

2.3.6 PayPal

PayPal [29] is the most widely used digital payment platform in the world. It supports peer-to-peer transfers, payments to merchants in multiple currencies, and large-scale cross-border transactions. Dahlberg et al. (2008) found that global merchant interoperability and transaction security were two of the most important factors in the adoption of mobile payment platforms [5]. PayPal's dominant network effect makes it the standard for both of these. Its two-factor authentication, real-time fraud detection, and two-decade track record of compliance are all real security strengths that make it the most trusted payment system in the comparison set.

PayPal's architecture is limited to two-way transactions between individuals and merchants or between individuals. This is despite its size and security strengths. There is no shared wallet feature, no way to manage a group account, and no way to split bills other than asking a contact for a certain amount of money. This limitation is not accidental; it is a strategic choice that favours high-volume individual transactions and makes collaborative financial management impossible. Kim et al. (2010) observed that perceived feature completeness in relation to user needs is a crucial factor influencing platform switching intent [16]; consequently, users needing group financial tools are necessitated to augment PayPal with specialized applications, thereby sustaining the disjointed multi-platform workflow that the proposed system aims to eradicate by unifying all pertinent financial functions within a singular integrated platform.

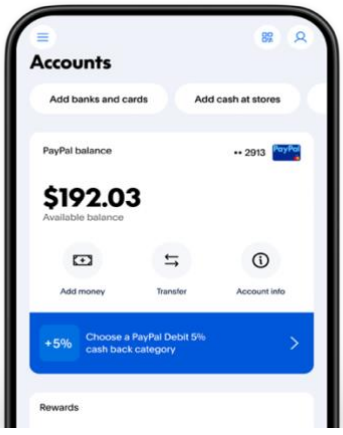

Figure 2.3.6.1 PayPal account interface showing balance management and transfer capabilities [29]

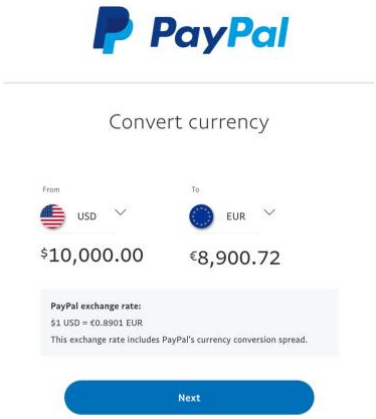

Figure 2.3.6.2 PayPal convert currency [29]

There is a clear pattern in the design of the six apps we looked at: platforms either focus on financial execution (like PayPal and Revolut) or expense coordination (like Splitwise and Tricount), but not both. This split shows that transaction processing systems and collaborative financial interfaces are built in such a way that they don't work together well, which leads to broken user workflows. From a systems design standpoint, this indicates that the limitation extends beyond mere feature omission; it reflects a fundamental deficiency in integrated design frameworks for multi-user financial interaction—a gap recognized in existing literature but not yet addressed in commercial development [22].

2.4 Comparison of Existing Applications

Table 2.4.1 shows a systematic comparison of the six applications that were reviewed with the proposed system in twelve functional areas. The assessment relies on publicly accessible feature documentation, official platform resources, and the functional analysis performed in Section 2.3. If a feature is available but only partially or superficially, it is labelled as "Basic" or "Premium" instead of being given a binary rating. This shows how different platforms implement features in different ways.

Table 2.4.1 Comparison of Existing Application and the Proposed System

Feature Aspect	N26 Shared Spaces	Wallet by Budget Bakers	Splitwise	Tricount	Revolut	PayPal	Proposed System
Shared Wallet	✓	✓	✗	✗	✗	✗	✓
Group Bill Splitting	Basic	Limited	✓	✓	Basic	✗	✓
AI Receipt Scanning	✗	✗	Premium	✗	✗	✗	✓
Multi-Currency Support	✓	✗	✓	✓	✓	✓	✓
Real-time Exchange Rate	✗	✗	✗	✗	✓	✓	✓
Direct Payments	✓	✗	✗	✗	✓	✓	✓
QR Code Payment	✗	✗	✗	✗	✗	✓	✓

Transfer Limit Control	✗	✗	✗	✗	✗	✗	✓
Consensus Wallet Closure	✗	✗	✗	✗	✗	✗	✓
Cashflow Analytics	Basic	✓	✓	Basic	✓	Basic	✓
2FA / Phone Auth	✓	✓	✗	✗	✓	✓	✓

Table 2.4.1 shows that no current platform meets all twelve functional needs at the same time. The comparison shows that the current landscape is structurally fragmented: collaborative features and payment functionality are always developed separately, not together. BudgetBakers' N26 Shared Spaces and Wallet let you share a wallet or track a group, but they don't let you pay directly, scan receipts with AI, or set transfer limits. Splitwise and Tricount are the best at splitting bills, but they don't work with any payment systems; instead, they act as ledgers instead of wallets. Revolut has the best multi-currency and analytics features, but it doesn't let you manage shared wallets or split bills at the item level, and it isn't licensed in Malaysia. PayPal is the best at reaching people all over the world and keeping their money safe, but it doesn't let people work together to manage their money.

This pattern of structural fragmentation illustrates the perspective of Ondrus and Pigneur (2006), who contended that the disparate interests of platform stakeholders generate systematic disincentives for the development of features that allocate financial control among multiple users [22]. The result for end users is that they have to rely on multi-application workflows, which can cause problems, mistakes, and delays in settling payments. The proposed system solves this problem by bringing together collaborative fund management, AI-powered bill splitting, multi-currency wallets, and payment execution into one platform.

2.5 Limitations of Previous Studies and Applications

The analysis of current e-wallet and group financial management applications uncovers significant deficiencies that collectively necessitate the creation of the proposed system.

First, there isn't a single platform that offers all of the features of a shared wallet, advanced bill splitting, and support for multiple currencies. Apps like Splitwise and Tricount are mostly for sharing expenses, but they aren't real e-wallets because they can't store money or process direct payments. On the other hand, platforms like Revolut and PayPal have strong payment options and support for multiple currencies, but they don't have a way for users to work together to manage group expenses, so users have to do it all by hand [10].

Second, the ways that people split bills now are either very simple or very strict. Most solutions use simple equal-split methods that don't take into account taxes, service fees, or how much each person uses. Riquelme and Rios (2010) discovered that the intricacy of manually tracking these variables is a considerable impediment to the ongoing utilisation of group payment functionalities [11]. Users have to make changes by hand, which makes it more likely that mistakes will be made and disagreements will arise. Also, the lack of automated settlement means that users have to rely on outside transactions, which makes the process take longer and be more of a hassle.

Third, the support for internationalization is not the same on all platforms. Revolut and PayPal let you manage more than one currency, but Wallet by BudgetBakers and N26 Shared Spaces don't let you change currencies in real time. Schuh and Stavins (2010) showed that not having clear information about exchange rates in the app is a big problem for people who do business across borders [13]. The Asian Development Bank (2022) also said that the lack of multi-currency payment tools that work together is still a big problem for people in the ASEAN region who want to use digital money [14].

Fourth, the security and access control features are not always used correctly. Two-factor authentication is not used by most bill-splitting apps, like Splitwise and Tricount. Bonneau et al. (2012) showed that using a phone to verify an OTP is a much safer way to log in to financial apps than just using a password [18]. This is a standard that many of the platforms we looked at don't meet.

The fact that features are spread out across many apps means that users have to use different tools for shared budgeting, expense tracking, currency conversion, and payment settlement. This leads to inefficiency and a bad user experience. Zhou, Lu, and Wang (2010) discovered that directly incorporating financial management into the payment workflow markedly enhances usability, resulting in increased task completion rates and user satisfaction [26]. The lack of such integration in current platforms highlights the necessity for the cohesive solution suggested in this project.

These limitations show that current systems aren't built around collaborative financial interaction as a main idea; instead, they treat it as an extra feature that can be added to systems that are focused on individuals. No one platform can handle the whole lifecycle of a group financial event, from managing shared funds to capturing itemized expenses, splitting them automatically, and settling payments all in one place. The field has created strong theoretical frameworks for how people use payments, like TAM and UTAUT, but it hasn't made similar frameworks for how to design and test multi-user financial systems. The architectural choices of current commercial platforms reflect and perpetuate this theoretical gap. The proposed system is designed to fill this gap at both the product and paradigm level.

2.6 Proposed Solution

The design requirements for the proposed system are based on the limitations found in Sections 2.3 and 2.5. Every aspect of the proposed application directly addresses a particular deficiency identified in the prior assessment. Based on what has been written about the topic, this section shows how each design choice relates to the problem it solves.

The shared wallet module directly addresses the lack of multi-party fund control mentioned in Section 2.5. None of the reviewed platforms offer a way for groups to collectively manage, hold, and monitor a shared financial pool with customisable access controls. Wilkinson and Young (2002) found this to be the main reason why users are happy with collaborative financial arrangements [21]. The proposed system solves this problem by showing everyone their balance in real time, keeping a record of all transactions, allowing users to set limits on how much they can transfer each day and each transaction, and requiring all members to agree before funds can be distributed.

These features give the structural fairness and democratic control that current platforms can't provide.

The bill-splitting module directly fixes the two-part problem that was found in Section 2.5: there is no item-level receipt parsing and expense recording is separate from payment execution. The module uses the Claude API to do end-to-end receipt parsing, which means it can take a photo of a receipt and get the merchant's name, line items, quantities, unit prices, and any taxes that apply. This is made possible by recent advances in AI-based document understanding. Huang et al. (2019) showed that today's receipt recognition systems can accurately extract item-level information from a wide range of receipt formats [12]. This provided the empirical basis for this design choice. Each recipient chooses what they want to eat, and the system figures out how much each person owes, including how much tax they owe. This level of detail is not available on any of the platforms we looked at. Most importantly, payment happens in the same application flow, which directly fixes the problem of inter-platform friction that Zhou, Lu, and Wang (2010) found to be a major problem with current mobile financial workflows [26].

The multi-currency wallet directly fixes the problems with internationalization support that were found in Section 2.5. The Transfer Calculator supports nine different currency wallets (MYR, USD, SGD, JPY, HKD, THB, AUD, GBP, and CAD). It also shows real-time exchange rates and a seven-day historical trend chart. Schuh and Stavins (2010) found that transparency in in-app exchange rates is an important factor in building trust in cross-border payment systems [13]. The proposed system solves this problem by showing live rates and historical trends in the same interface that users use to make conversions, which removes the need for users to check rates outside of the app, as current platforms require. The security architecture directly addresses the authentication gap identified in Section 2.5. It uses bcrypt hashing for Gmail and password authentication, Firebase Phone Authentication OTP verification during registration, and Stripe's PCI DSS Level 1 certified infrastructure to make sure that card-based top-ups are processed without the application ever having to handle raw cardholder data [30]. This is a compliance standard that many of the platforms reviewed do not meet.

2.7 Summary

This chapter has given a thorough look at the basics of e-wallets, how they are used now, and the academic literature that is related to them. The review shows that existing platforms and research have looked at each of the proposed system's features—peer-to-peer payment, group expense tracking, OCR-based receipt parsing, multi-currency management, and mobile security—on their own. However, there is no current commercial application that brings all of these features together into one mobile platform.

The comparative analysis in Table 2.4.1 shows that the proposed system meets twelve functional requirements at the same time. This fills the gap left by platforms that are great in one or two areas but not so great in all areas of collaborative financial management. The constraints delineated in Section 2.5 serve as the primary impetus for the design choices recorded in the ensuing chapters, which elaborate on the system architecture, implementation, and assessment of the proposed application.

CHAPTER 3 SYSTEM METHODOLOGY

This chapter outlines the system methodology employed in the development of the proposed e-wallet application. It talks about the overall system architecture, defines the actors and how they interact using use case diagrams and descriptions, and shows the main operational workflows using activity diagrams. All of these artefacts together make a complete plan for how the system should work and be built. This plan was used to guide the implementation described in the next chapters.

3.1 System Design Diagram

Three complementary artefacts show the system design of the proposed application: a system architecture diagram that shows the high-level technical structure and how the components interact; use case diagrams that show the functional limits of the system from the point of view of its actors; and activity diagrams that show the step-by-step flow of key user-initiated processes. These design artefacts show the system from three different angles: structural, functional, and behavioural.

3.1.1 System Architecture Diagram

The suggested application uses a three-tier client-server architecture with a mobile presentation layer, a RESTful application logic layer, and a persistent data layer. This architectural pattern divides responsibilities across tiers, making it possible for each layer to be developed, tested, and maintained on its own.

The Flutter framework is used to build the presentation layer, which is for iOS. Flutter's widget-based rendering model makes sure that the UI behaves the same way every time. Its stateful widget architecture lets the UI update in real time when an API response comes in, which is necessary for features like live transaction notifications and shared wallet balance updates.

The application logic layer is a single Node.js server that uses the Express framework. It has a RESTful API that only the Flutter client can use. This layer contains all of the business logic, such as managing wallets, processing transactions, figuring out how to split bills, and changing money from one currency to another. The Node.js event-driven, non-blocking I/O model works well with the way that multiple users can

make requests at the same time when they use a shared wallet. A secure session token stored on the client device using encrypted local storage manages session authentication. This token is sent with authenticated requests to identify the current user.

The data layer has a MySQL relational database that keeps track of all application state, such as user accounts, wallet balances, transaction records, shared wallet membership, bill splitting records, and currency exchange history. The relational model is right for this app because it needs referential integrity between entities. For example, a transaction record must point to valid wallet, user, and currency entities. It also needs ACID-compliant transaction processing to make sure that financial data stays the same.

The Node.js backend makes API calls to four outside services that work together: Stripe is only used for adding money to a wallet with a card (all other financial transactions, like transfers, QR payments, bill payments, and currency conversions, are done by taking money out of the wallet balance without Stripe); the ExchangeRate API is used to get real-time and historical foreign exchange rates; the Claude API is used to parse receipt images with AI in the bill splitting module; and Firebase Phone Authentication is used during the registration process to send a one-time password (OTP) to the user's registered phone number for identity verification. You use your Gmail address and password to log in after that. The Flutter local_auth package lets you use device-native Face ID instead of a PIN for transaction-level authorization of Transfer and Convert operations.

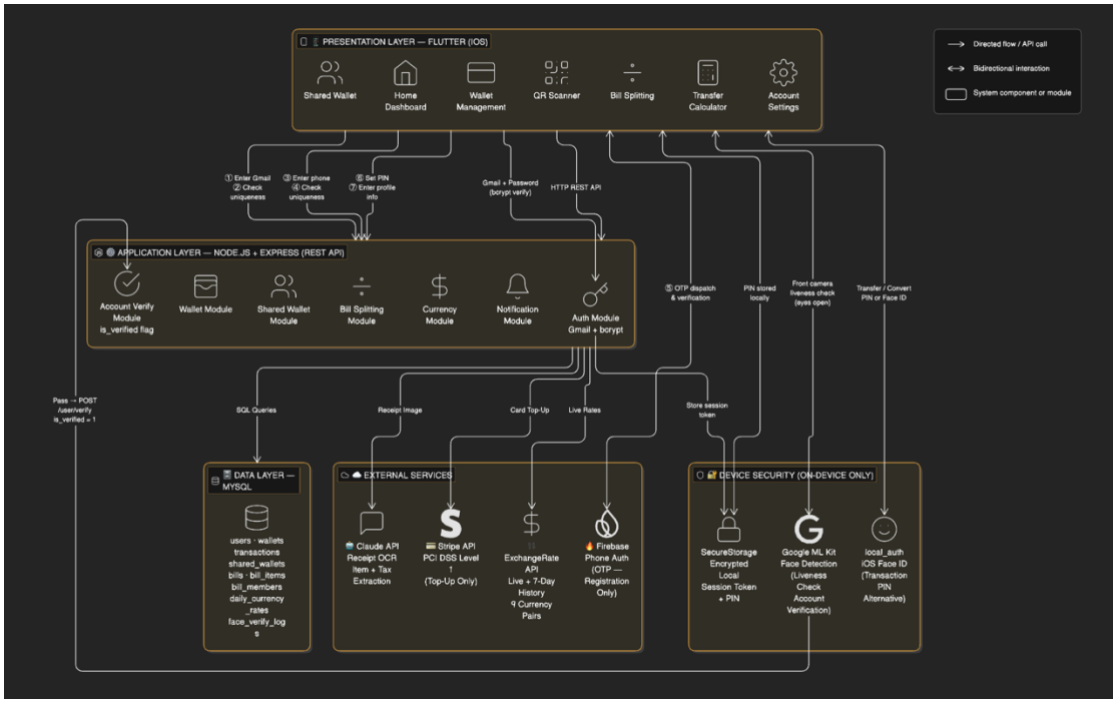

Figure 3.1.1.1 System Architecture Diagram of the Proposed E-Wallet Application

Figure 3.1.1.1 shows that the Flutter client talks to the Node.js backend over HTTP using a RESTful API. The Node.js backend is the only way for third-party APIs to talk to each other, like Stripe, ExchangeRate API, Claude API, and Firebase. This keeps API credentials and business logic on the server side and away from the client.

3.1.2 Use Case Diagram and Description

The use case diagram shows what the system can do from the point of view of its actors. The system recognizes two main actors: the Registered User, who is anyone who has logged into the application, and the System, which includes automated tasks like validating sessions, converting currencies, parsing receipts, and sending notifications. There is also an implicit actor called External Services, which includes Stripe, ExchangeRate API, Claude API, and Firebase. These services talk to the system through server-side API calls.

There are five functional modules that hold the use cases. The Authentication and Account Management module includes registration (UC-01), logging in (UC-02), managing contacts (UC-03), verifying accounts with ML Kit liveness detection (UC-04), setting up transaction security with a PIN and Face ID (UC-05), editing profiles (UC-06), logging out (UC-07), changing a PIN (UC-08), sending a password reset email (UC-09), changing a password (UC-10), changing an email address (UC-11), and

changing a phone number (UC-12). The Wallet Management module includes managing and adding to currency wallets (UC-13), setting personal transfer limits (UC-14), transferring money with a PIN or Face ID (UC-15), paying with a QR code (UC-16), converting currency (UC-17), and closing a currency wallet with an automatic balance conversion to the main currency (UC-18). The Shared Wallet Management module (UC-19 to UC-21), the Bill Splitting module (UC-22 to UC-24), and the Transfer Calculator module (UC-25) make up the full range of functions.

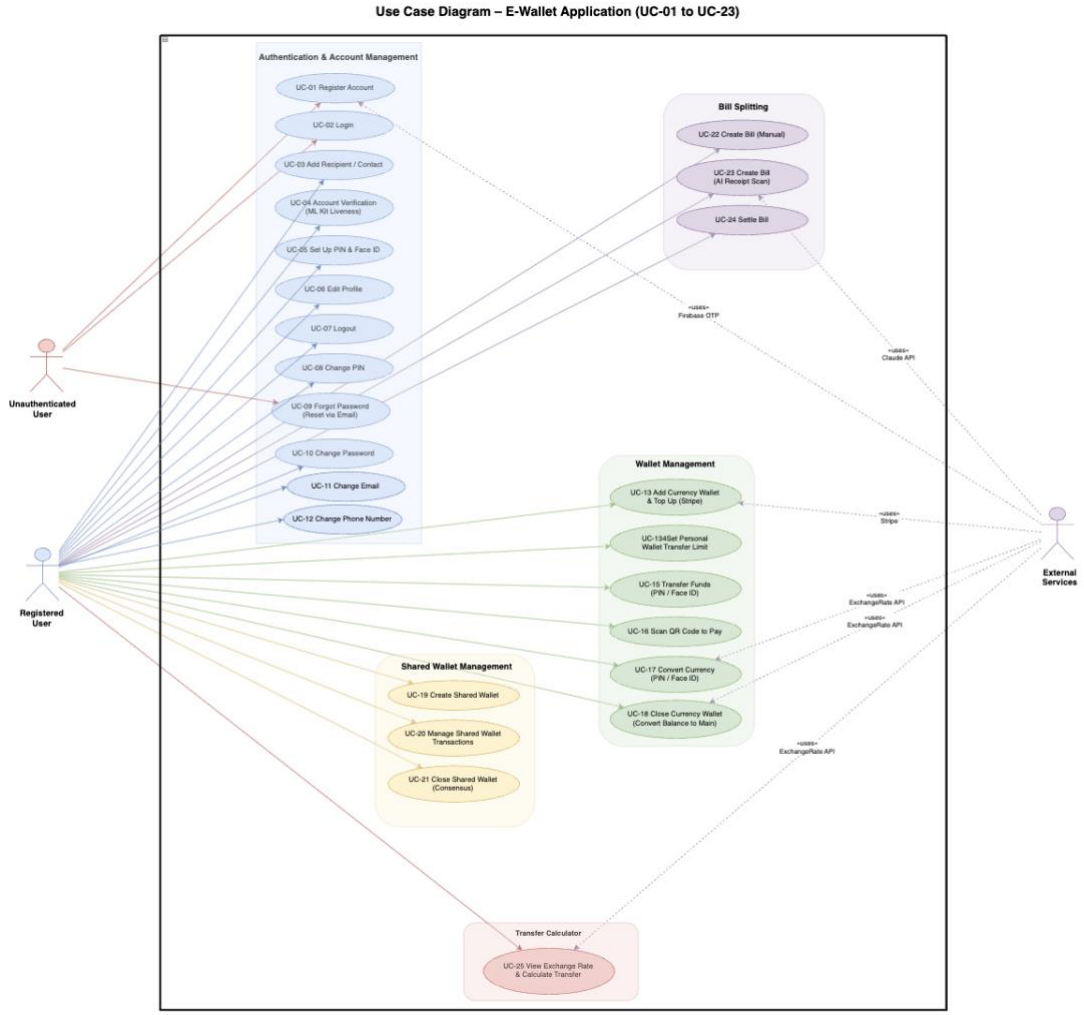

Figure 3.1.2.1 Use Case Diagram of the Proposed E-Wallet Application

The next subsections give organized descriptions of the main use cases for each module. Each description includes the actor, preconditions, postconditions, main flow, and alternative flows, which is the standard use case template used in requirements engineering.

3.1.2.1 Authentication and Account Management

Table 3.1.2.1.1 Register Account Description

Use Case ID	UC-01
Use Case Name	Register Account
Description	A new user signs up in a series of steps: Check that your Gmail address is unique, your phone number is unique, your Firebase OTP is correct, set up your PIN, and enter your personal information.
Actor	Unregistered User
Precondition	The user does not have an account yet. There is no record of this Gmail address or phone number. The Firebase Phone Authentication service is now available.
Postcondition	A user account is made with a PIN set up. The default currency wallet was set up based on the country you registered in. The user is sent to the login screen, where they can sign in with their Gmail address and password.
Main Flow	1. The user opens the app and clicks on "Register." 2. The user types in their Gmail address. The system checks to see if the Gmail account is already set up. 3. The user types in their phone number. The system checks to see if the phone number is already in use. 4. The system sends an OTP to the phone number using Firebase Phone Authentication.

	5. The user types in the OTP they got. The system checks the OTP with Firebase. 6. The user chooses a PIN number. 7. The user types in their name, date of birth, country, and password. 8. The system makes the user account with a password that has been hashed with bcrypt and a PIN that has been set up. 9. The system sets up the default currency wallet based on the country that is registered. 10. The user is sent back to the login screen.
Alternative Flow	2a. The system shows an error and stops if Gmail is already registered. Instead, the user is asked to log in. 3a. The phone number is already registered, so the system shows an error and stops. 5a. Wrong OTP entered: The system shows an error and lets you try again. 5b. If the OTP expires, the user can ask for a new one.

Table 3.1.2.1.2 Login Description

Use Case ID	UC-02
Use Case Name	Login
Description	A registered user logs in with their Gmail address and password. After the authentication is successful, the system saves a secure session token on the client's device.
Actor	Registered User
Precondition	User has a registered account with a Gmail address and password.
Postcondition	The user has been verified. Encrypted local storage (SecureStorage) stores a secure session token on the client's device.
Main Flow	1. The user types in their Gmail address and password. 2. The system checks the credentials against the database (bcrypt password comparison). 3. The system checks to see if the account is still active. 4. The system saves a secure session token on the client's device. 5. The user is sent back to the home screen.
Alternative Flow	3a. The wrong password was entered, so the system shows an error and lets you try again. 4a. The system shows an error and asks you to sign up if it can't find your account.

Table 3.1.2.1.3 Add Recipient Description

Use Case ID	UC-03
Use Case Name	Add Recipient
Description	The user adds another registered user as a contact or recipient. When you make a shared wallet or send out a bill, the people you added show up at the top of the list of people you can choose from.
Actor	Registered User
Precondition	User is authenticated. The target user is a registered account in the system.
Postcondition	The recipient is added to the user's contact list and will show up first when choosing a member of a shared wallet or a bill recipient.
Main Flow	1. The user goes to the Recipients section. 2. The user looks for another user by their email address or phone number. 3. The system checks to see if the target user is in the database. 4. The user agrees to add the contact. 5. The system saves the relationship with the contact. 6. The new recipient now shows up at the top of the list for creating a Shared Wallet and choosing a Bill Splitting recipient.
Alternative Flow	2a. User not found: The system shows an error message. 2b. The user is already on the contact list, so the system tells them.

Table 3.1.2.1.4 Account Verification Description

Use Case ID	UC-04
Use Case Name	Account Verification
Description	The user uses the front camera on the device and Google ML Kit Face Detection to do a one-time liveness check to make sure they are a real person with their eyes open. The backend marks the account as verified after the liveness check is passed. Verified accounts can make and use more currency wallets than the main wallet that was set up when they signed up. This mechanism processes images entirely on the device and is different from the transaction-level Face ID authentication used in Transfer and Convert operations (UC-05).
Actor	Registered User
Precondition	The user has been verified. The device has a working front-facing camera and has given permission for the camera. User has not yet verified their account (is_verified = false in the database).
Postcondition	The database says that the account verification status is Verified (is_verified = true, verified_at = NOW()). The verification status is synced to the client's encrypted local storage (SecureStorage). Users can now make and use more currency wallets in addition to their main currency wallet.
Main Flow	1. The user goes to Account Settings and chooses Verify Account. 2. The system turns on the front-facing camera and goes into cameraReady mode.

	3. The user taps "Start Face Scan." The system starts sending camera frames and goes into scanning mode. 4. Google ML Kit Face Detection runs on each camera frame that the system gets. 5. ML Kit sees a face in the frame. The system stops the stream of images and goes into analysis mode. 6. The system checks to see if the face is alive: (a) Makes sure that there is only one face in the frame. (b) Confirms that the chance of the left eye being open is at least 0.4. (c) Confirms that the chance of the right eye being open is at least 0.4. 7. The checks for liveness are passed. System calls POST /api/user/verify with the ID of the user. 8. The backend sets is_verified to 1 and verified_at to NOW() in the users table. 9. The client gets an HTTP 200 response and writes "face_enrolled = true" to SecureStorage. 10. The system goes to a verified state and shows a message saying it worked.
Alternative Flow	3a. If the camera doesn't start up or the system doesn't have permission to use it, an error message will appear and the system will stay in a failed state. After giving permission, the user can try again. 6a. The system shows an error message that says "Multiple faces detected" when it sees more than one face in the frame. Please make sure that only your face is visible. User can try again. 6b. If the probability of either eye being open is less than 0.4, the system shows an error message ("Please

	open your eyes and look at the camera") and goes into failed state. The user can try again. 6c. There is no face in the scanning window: The system keeps streaming until it finds a face. The user is told to stay still in good light. 7a. If the backend returns a non-200 response, the system shows the error message from the response body and goes into a failed state. User can try again. 7b. If there is a network error or timeout during POST /api/user/verify, the system shows a network error message and goes into failed state. When the connection is back up, the user can try again. User declines verification: User goes back. The account is still not verified, and the user can only use the main currency wallet.

Table 3.1.2.1.5 Set Up Transaction PIN and Biometric Description

Use Case ID	UC-05
Use Case Name	Set Up Transaction PIN and Biometric
Description	The user sets a number PIN to confirm the transaction. The user can also use biometric data (Face ID) instead of a PIN to confirm transfers and currency exchanges.
Actor	Registered User
Precondition	User is authenticated.
Postcondition	Transaction PIN is saved. Biometric option is enabled or disabled per user preference. PIN or biometric will be required for Transfer and Convert operations.
Main Flow	1. User navigates to Security Settings.

	2. User selects Set Transaction PIN. 3. User enters a numeric PIN and confirms it. 4. System validates and saves the PIN securely. 5. System prompts user to optionally enable device biometric as an alternative. 6. If enabled, system registers biometric preference. 7. PIN and/or biometric will henceforth be required when initiating Transfer or Convert transactions.
Alternative Flow	3a. PIN confirmation does not match: System prompts re-entry. 6a. Device does not support biometric: Option is disabled automatically.

Table 3.1.2.1.6 Edit Profile Description

Use Case ID	UC-06
Use Case Name	Edit Profile
Description	The user changes their country and display name. The email address and phone number fields are locked, so you can't change them directly from the profile screen. This is because they need to go through different verification processes.
Actor	Registered User
Precondition	User is authenticated.
Postcondition	Display name and/or country is updated in the database.
Main Flow	1. User navigates to Profile Settings. 2. User edits their display name and/or country.

	3. User submits the changes. 4. System validates the input. 5. System updates the name and/or country in the database. 6. System displays a success confirmation.
Alternative Flow	4a. Name is empty or invalid: System displays a validation error and rejects the change. 3a. User cancels: No changes are saved.

Table 3.1.2.1.7 Logout Description

Use Case ID	UC-07
Use Case Name	Logout
Description	The user signs out of the app. The system deletes the stored session token from the client device's encrypted local storage.
Actor	Registered User
Precondition	User is authenticated and has an active session.
Postcondition	The client device's session token is cleared. The user is sent back to the login screen.
Main Flow	1. User navigates to Profile or Settings and selects Logout. 2. System prompts confirmation. 3. User confirms logout. 4. System clears the session token from encrypted local storage on the device. 5. System redirects the user to the login screen.

Alternative Flow	3a. User cancels: Session remains active and unchanged.

Table 3.1.2.1.8 Change PIN Description

Use Case ID	UC-08
Use Case Name	Change PIN
Description	The user changes their transaction PIN by first checking the old one and then setting and confirming a new one.
Actor	Registered User
Precondition	The user is verified. A transaction PIN has already been set up.
Postcondition	Transaction PIN is updated to the new value.
Main Flow	1. User navigates to Security Settings and selects Change PIN. 2. User enters the current PIN for verification. 3. System validates the current PIN. 4. User enters the new PIN and confirms it. 5. System validates that the two entries match. 6. System saves the new PIN securely. 7. System displays a success confirmation.
Alternative Flow	3a. Current PIN incorrect (max 3 attempts): System locks the action temporarily. 5a. New PIN entries do not match: System prompts re-entry.

Table 3.1.2.1.9 Forgot Password Description

Use Case ID	UC-09
Use Case Name	Forgot Password
Description	A user who has forgotten their password starts the reset process by entering their registered Gmail address. The system sends an email with a link to reset the password. The user then sets a new password.
Actor	Unauthenticated User
Precondition	The user is on the login screen and has an account that is linked to a Gmail address.
Postcondition	We sent you a link to reset your password. When the user clicks the link, their password is changed and they are sent to Login.
Main Flow	1. The user clicks "Forgot Password" on the Login screen. 2. The system asks the user to type in their Gmail address. 3. The user sends the email. 4. The system checks to see if the email is linked to an account that is already registered. 5. The system sends a link to reset your password to the Gmail address you gave. 6. The user opens the email and clicks on the link to reset their password. 7. The system shows a form where you can enter and confirm a new password. 8. The system checks and updates the password. 9. The user is sent back to the Login screen.

Alternative Flow	4a. Email not found: The system shows a generic error but doesn't say if the account exists. 6a. The reset link has expired; the user needs to ask for a new one. 7a. The new passwords don't match, so the system asks for them again.

Table 3.1.2.1.10 Change Password Description

Use Case ID	UC-10
Use Case Name	Change Password
Description	A user who has been verified can change their account password by entering their current password and then choosing a new one.
Actor	Registered User
Precondition	User is authenticated.
Postcondition	Account password is updated to the new value.
Main Flow	1. The user goes to Security Settings and clicks on Change Password. 2. The user types in their current password to confirm it. 3. The system checks the current password against the bcrypt hash that is already stored. 4. The user types in the new password and then confirms it. 5. The system checks that the two new password entries are the same and meet the minimum requirements. 6. The system hashes the new password and saves it. 7. The system shows that the operation was successful.
Alternative Flow	3a. The current password is wrong, so the system shows an error. 5a. The new password entries don't match, so the system asks you to enter them again. 5b. The new password doesn't meet the requirements: the system shows what they are.

Table 3.1.2.1.11 Change Email Description

Use Case ID	UC-11
Use Case Name	Change Email
Description	The user updates the Gmail address they have on file. The email field is locked on the Edit Profile screen, and you can only change it through this special flow, which checks that the new email is unique and accessible.
Actor	Registered User
Precondition	The user has been verified. No other account has already registered the new email address.
Postcondition	The database gets the new Gmail address after the email verification is successful.
Main Flow	1. The user goes to Security Settings and clicks on Change Email. 2. The user types in the new Gmail address. 3. The system checks to see if another account has already registered the new email. 4. The system sends a verification link to the new Gmail address if it is unique. 5. The user opens the email and clicks on the link to verify it. 6. The system checks the link and changes the email address in the database. 7. The system shows that the operation was successful.
Alternative Flow	3a. The system shows an error and won't let you change it because the new email is already registered.

	5a. The verification link is no longer valid, so the user must ask for a new one. 2a. The email format is not valid: The system shows a validation error.

Table 3.1.2.1.12 Change Phone Number Description

Use Case ID	UC-12
Use Case Name	Change Phone Number
Description	The user updates their phone number on file. The phone number field is locked on the Edit Profile screen, and you can only change it through this special flow, which uses Firebase OTP to check the new number.
Actor	Registered User
Precondition	The user is verified. No other account has already registered the new phone number. You can use Firebase Phone Authentication.
Postcondition	The database updates the phone number after the OTP verification is successful.
Main Flow	1. The user goes to Security Settings and chooses Change Phone Number. 2. The user types in the new phone number. 3. The system checks to see if another account has already registered the new phone number. 4. If the phone number is unique, the system sends an OTP to it using Firebase Phone Authentication. 5. The user types in the OTP they got. 6. The system checks the OTP with Firebase.

	7. The system changes the phone number in the database. 8. The system shows a message that the operation was successful.
Alternative Flow	3a. The phone number is already registered: The system shows an error and won't let you make the change. 5a. Wrong OTP: The system shows an error and lets you try again. 5b. If the OTP expires, the user can ask for a new one.

3.1.2.2 Wallet Management

Table 3.1.2.2.1 Add Currency Wallet and Top Up Description

Use Case ID	UC-13
Use Case Name	Add Currency Wallet and Top Up
Description	A verified user can add a new wallet for a currency other than the main currency and use Stripe to top up any wallet with a credit or debit card. Users who haven't been verified can only add money to their main currency wallet.
Actor	Registered User
Precondition	The user has been verified. To add a new currency wallet, you need to verify your account (UC-04). For a top-up, the chosen wallet must exist and Stripe must be available.
Postcondition	A new currency wallet is created (if needed), and/or the chosen wallet gets the amount that was requested. A record of the transaction is made.
Main Flow	1. The user goes to Wallets. 2a. To add a new wallet for a currency: User clicks on "Add Wallet" and picks a currency. The system checks to see if the account has been verified. If the information is correct, a wallet is made. If the account is not verified, the system shows a message telling the user to do so first. 2b. To add money, the user picks a wallet and taps "Top Up." 3. The user types in the amount they want to add. 4. The system shows the Stripe payment interface.

	5. The user types in their card information. 6. The system sends a payment request to Stripe. 7. Stripe sends back a confirmation. 8. The system adds money to the wallet and makes a record of the transaction. 9. The system shows a message that the task was successful.
Alternative Flow	7a. Stripe says the payment failed. The wallet balance stays the same, and an error message shows up. 2a-alt. When you try to add a wallet, the system blocks the action and takes you to Account Verification (UC-04).

Table 3.1.2.2.2 Set Personal Wallet Transfer Limit Description

Use Case ID	UC-14
Use Case Name	Set Personal Wallet Transfer Limit
Description	The user sets daily and per-transaction transfer limits for a chosen personal currency wallet. The system makes sure that these limits are followed for all Transfer operations from that wallet.
Actor	Registered User
Precondition	The user is verified.The chosen personal currency wallet is real and works.
Postcondition	The user_currency_limits table stores the daily and per-transaction transfer limits for the chosen currency wallet.Before processing, all future Transfer operations from that wallet are checked against the set limits.

Main Flow	1. The user goes to the settings for the personal currency wallet they chose. 2. The user chooses Set Transfer Limits. 3. The user types in the daily transfer limit and the limit for each transaction. 4. The system checks the input values. 5. The system saves the limits to user_currency_limits for the wallet of the currency that was chosen. 6. The system shows a message that the operation was successful. 7. Before processing, all future Transfer operations from this wallet are checked against these limits.
Alternative Flow	4a. If either limit value is negative, zero, or not a number, the system shows a validation error and won't accept the input. The user is asked to enter valid values again. 4b. The per-transaction limit is higher than the daily limit: The system shows a validation error that says the per-transaction limit can't be higher than the daily transfer limit. The user is asked to fix the values.

Table 3.1.2.2.3 Transfer Funds Description

Use Case ID	UC-15
Use Case Name	Transfer Funds
Description	The user moves money from their wallet to another registered user's wallet in the same currency.
Actor	Registered User
Precondition	The user is verified. The sender has enough money. The recipient is a registered user.

Postcondition	The sender's wallet is charged. The recipient's wallet gets money. Both records of the transaction are made.
Main Flow	1. The user picks "Transfer" and looks for the person they want to send money to by phone number or username (new recipients show up at the top of the list). 2. The user chooses the wallet to send money from and types in the amount. 3. The system checks to make sure the sender has enough money. 4. The system asks for a PIN or Face ID check. 5. The user enters their PIN or Face ID to prove who they are. 6. The system takes money out of the sender's wallet and puts it into the recipient's wallet. 7. The system makes records of the transaction for both the sender and the receiver. 8. The system sends a message to the person who got it. 9. The system sends a message to the sender saying that the operation was successful.
Alternative Flow	3a. Not enough balance: The system shows an error and cancels the transfer. 1a. The system shows an error message because it can't find the recipient. 5a. Wrong PIN (up to three tries): The system locks the transaction and asks you to try again. 5b. If biometric verification fails, the system goes back to PIN verification.

Table 3.1.2.2.4 Scan QR Code to Pay Description

Use Case ID	UC-16
Use Case Name	Scan QR Code to Pay
Description	The user scans the QR code of another user to pay them. The QR code has the recipient's wallet information in it.
Actor	Registered User
Precondition	The user is verified. You have permission to use the camera. The recipient has a working QR code.
Postcondition	The sender's wallet is charged, and the recipient's wallet is credited. Both parties get records of the transaction.
Main Flow	1. The user chooses "Scan QR" from the home screen. 2. The user scans the QR code of the person they want to send the message to with the camera on their device. 3. The system decodes the QR code and gets the wallet information for the person who sent it. 4. The system shows a payment confirmation screen with the name of the person who will get the money and the amount. 5. The user agrees to the payment. 6. The system takes care of the transfer and updates the balances in both wallets. 7. The system makes records of the transaction and tells both parties.
Alternative Flow	2a. If the QR code is not valid or can't be read, the system shows an error. 5a. Not enough money: The system cancels the transaction and tells the user.

Table 3.1.2.2.5 Convert Currency Description

Use Case ID	UC-17
Use Case Name	Convert Currency
Description	The user moves a certain amount of money from one currency wallet to another at the current live exchange rate. You need to verify your account to use other currency wallets besides the main one.
Actor	Registered User
Precondition	The user is verified. User has verified their account (by face recognition) to unlock more currency wallets. You can use the ExchangeRate API.
Postcondition	The source wallet is charged. The target wallet gets money. A record of the conversion transaction is made.
Main Flow	1. The user picks Convert and then picks the wallets for the source and target currencies. 2. The user types in the amount they want to change. 3. The system gets the current exchange rate from the ExchangeRate API. 4. The system shows the user the converted amount and the exchange rate that applies. 5. The system asks for a PIN or Face ID check. 6. The user finishes the PIN or Face ID check. 7. The system takes money out of the source wallet and puts it into the target wallet at the agreed-upon rate. 8. The system makes a record of the conversion transaction.

Alternative Flow	3a. The ExchangeRate API is not available, so the system shows an error and cancels the conversion. 6a. The PIN is wrong (maximum of three tries): The system locks the transaction. 6b. If Face ID doesn't work, the system goes back to PIN verification. 5a. The user cancels, and no transaction is made.

Table 3.1.2.2.6 Close Currency Wallet Description

Use Case ID	UC-18
Use Case Name	Close Currency Wallet
Description	The user closes a wallet that doesn't hold main currency. Before the wallet is turned off, any money left over is automatically changed to the main currency wallet at the current live exchange rate.
Actor	Registered User
Precondition	The user has been verified. The wallet you chose is not a main currency wallet. The ExchangeRate API is ready to use.
Postcondition	The remaining balance was moved to the main currency wallet. The currency wallet is closed and you can't use it to make transactions anymore.
Main Flow	1. The user goes to the currency wallet they want to close. 2. The user clicks on "Close Wallet."

	3. The system shows the current balance and the estimated amount that will be converted to the main currency at the current exchange rate. 4. The system asks the user to confirm. 5. The user agrees to close. 6. The system gets the current exchange rate from the ExchangeRate API. 7. The system changes the remaining balance into the main currency wallet. 8. The system makes a record of the conversion transaction. 9. The system turns off the currency wallet. 10. The system shows a success message that shows the converted amount has been added to the main wallet.
Alternative Flow	3a. Balance is zero: The system closes the wallet right away without changing it. 6a. The ExchangeRate API is not available: The system shows an error and stops the closure. 4a. The user cancels, but the wallet stays active.

3.1.2.3 Shared Wallet Management

Table 3.1.2.3.1 Create Shared Wallet Description

Use Case ID	UC-19
Use Case Name	Create Shared Wallet
Description	To make a shared wallet, the owner gives it a name, chooses a currency, sets optional transfer limits, and invites people from their contact list.
Actor	Registered User (Owner)
Precondition	The user is verified. The user's contact list has at least one contact.
Postcondition	A shared wallet is made. People who are invited get a message telling them to join.
Main Flow	1. User picks "Create Shared Wallet." 2. The user types in the name of the wallet and chooses the currency for it. 3. The user can set a daily transfer limit and a per-transaction transfer limit if they want to. 4. The user chooses people from their contact list to invite. 5. The system makes the shared wallet and gives the creator the title of Owner. 6. The system sends invitations to all of the chosen contacts to join. 7. The system shows the new shared wallet on the home screen.
Alternative Flow	4a. The chosen contact is not a registered user, so the system leaves them out and tells the owner.

Table 3.1.2.3.2 Manage Shared Wallet Transaction Description

Use Case ID	UC-20
Use Case Name	Manage Shared Wallet Transaction
Description	Anyone who has access to a shared wallet can add money to it from their own wallet, pay bills by scanning a QR code with the shared wallet balance, and see the full transaction history and cash flow analytics.
Actor	Registered User (Member)
Precondition	The user is verified and is a member of the shared wallet. The wallet is open.
Postcondition	The shared wallet keeps track of the transaction. All members' opinions are updated right away.
Main Flow	1. The user opens the wallet that is shared. 2a. To add money, the user clicks "Top Up," types in the amount, and then clicks "Confirm." The system takes money out of the user's personal wallet and adds it to the shared wallet balance. 2b. To pay with a QR code, the user chooses "Scan QR" in the shared wallet, scans the recipient's QR code, and enters the amount they want to pay. 3. The system checks the transaction against the transfer limits set for the shared wallet. 4. The system handles the transaction and updates the balance in the shared wallet. 5. The system makes a record of the transaction that all members can see. 6. The system sends a message to all members of the wallet.

Alternative Flow	3a. The transaction goes over the transfer limit: The system turns down the transaction and shows the limit that applies. 2b-alt. The QR code is not valid or can't be read: The system shows an error. 2a-alternatively. System shows an error when there isn't enough money in the personal wallet to top up.

Table 3.1.2.3.3 Close Shared Wallet Description

Use Case ID	UC-21
Use Case Name	Close Shared Wallet
Description	The owner starts the process of closing a shared wallet. Before the wallet can be closed, all members must agree on how to divide the money and give their approval.
Actor	Registered User (Owner)
Precondition	The user is verified as the Owner. The shared wallet is on.
Postcondition	Everyone has voted. If everyone agrees, the money is given out in the agreed-upon way, and the wallet is closed.
Main Flow	1. The owner picks "Close Wallet" and then chooses how to divide the money (equal split, full transfer to one member, proportional by contribution, or smart distribution). 2. The system sends a request for approval to all members to close and distribute.

	3. Each member looks over the proposed distribution and votes to approve or deny it. 4. If everyone agrees, the system sends the funds to each wallet and marks the wallet as closed. 5. The system sends all members a message saying that the closure is complete.
Alternative Flow	3a. Any member rejects the closure: Closure is cancelled. Owner may propose a different distribution method. 3b. Member does not respond within the defined period: System sends a reminder notification.

3.1.2.4 Bill Splitting

Table 3.1.2.4.1 Create Bill (Manual) Description

Use Case ID	UC-22
Use Case Name	Create Bill (Manual)
Description	The creator types in the details of a bill for a payment they have already made in full in real life. The person who made the transaction chooses the merchant name, line items, quantities, prices, and applicable tax. They also choose the items they personally consumed (which won't be charged to others) and gives the rest of the items to the people who will pay.
Actor	Registered User (Bill Creator)
Precondition	The user has been verified. There is at least one contact. At least one currency wallet is active for the user.
Postcondition	The bill is made in the chosen currency and sent to the chosen people. People only have to pay for the things they choose. The person who made it doesn't have to pay.
Main Flow	1. The creator chooses "Create Bill" and then "Manual Entry." 2. The creator chooses the billing currency from the currencies in their currently active wallets. 3. The creator types in the merchant name, date, and each line item with its quantity and unit price. 4. The creator types in the right tax rate (if there is one). 5. The system figures out the subtotal and tax for each item.

	6. The creator picks the things they ate themselves. No one will have to pay for these things. 7. The creator picks who will get the message from their contact list. 8. The system makes the bill in the chosen currency and sends it to the people who were chosen. 9. The person who made it doesn't have to pay. Each person who gets something will have to pay for it when they get it.
Alternative Flow	7a. The chosen recipient does not have a wallet in the chosen currency. The system lets the creator know. 4a. No tax due: The creator sets the tax to zero. 2a. The user only has one active wallet, so the currency is automatically chosen.

Table 3.1.2.4.2 Create Bill (AI Receipt Scan) Description

Use Case ID	UC-23
Use Case Name	Create Bill (AI Receipt Scan)
Description	The person who made the receipt takes a picture of it or uploads it online after they have already paid for it in full. Using the Claude API, the system takes merchant names, line items, quantities, prices, and taxes and puts them into a structured bill form. The creator looks over the pre-filled-out form, picks the items they personally used (which won't be charged to anyone else), and gives the rest of the items to the people who will pay for them.
Actor	Registered User (Bill Creator)

Precondition	The user has been verified. You have access to the camera or gallery. You can use the Claude API. User has at least one currency wallet that is active.
Postcondition	A bill is made and sent to a small number of people. People only pay for the things they choose. The creator doesn't have to pay because they already paid the full bill in real life.
Main Flow	1. The creator chooses "Create Bill" and then "Scan Receipt." 2. The creator takes a picture of the receipt or uploads one from the gallery. 3. The system sends the picture to the Claude API so it can be parsed. 4. The Claude API sends back structured data that has been extracted and sorted, such as the merchant's name, line items, quantities, unit prices, and tax. 5. The system automatically fills out the bill form with the extracted data, including how the items are grouped. 6. The creator picks the billing currency from the currencies in their active wallets. 7. The creator checks the pre-filled form to make sure it is correct. If any field is wrong, the creator fixes it by hand. 8. The creator picks the things they ate themselves. No one will have to pay for these items. 9. The creator picks people from their contact list to send the message to.

	10. The system makes the bill in the chosen currency and sends it to the chosen people. 11. The creator does not have to pay. When they settle, each recipient will have to pay for the items they choose.
Alternative Flow	3a. The Claude API is not available, so the system tells the creator to switch to manual entry (UC-22). 4a. Extraction only partially works: The system fills in the available fields, marks the fields that haven't been extracted, and lets the creator finish them by hand before confirming. 6a. The user only has one active wallet, so the currency is automatically chosen. 7a. The creator cancels after reviewing: the bill is thrown away and no transaction is made.

Table 3.1.2.4.3 Settle Bill Description

Use Case ID	UC-24
Use Case Name	Settle Bill
Description	A person who gets a bill looks over the items on it, picks the ones they used, and then pays. The system takes the calculated amount out of their wallet and sends it to the person who made the bill.
Actor	Registered User (Recipient)
Precondition	The user has been verified and has received a bill. The user has enough money in their wallet.
Postcondition	The recipient's wallet is charged. The creator's wallet is credited. The status of the bill has been changed.

Main Flow	1. The recipient gets a bill notice and opens the bill. 2. The system shows all the bill items, along with their quantities and prices per unit. 3. The recipient chooses the things they ate. 4. The system figures out how much the recipient owes, including the right amount of tax. 5. The system shows the total amount due for confirmation. 6. The person receiving the payment confirms it. 7. The system takes the money out of the recipient's wallet and puts it into the creator's wallet. 8. The system updates the bill's status and lets the person who made the payment know.
Alternative Flow	6a. Not enough balance: The system shows an error. Before settling, the recipient must add money. 3a. If the recipient disagrees with the bill, they can contact the creator outside of the system.

3.1.2.5 Transfer Calculator

Table 3.1.2.5.1 View Exchange Rate and Calculate Transfer Description

Use Case ID	UC-25
Use Case Name	View Exchange Rate and Calculate Transfer
Description	The user sees the current exchange rate between two chosen currencies, looks at a chart of the rates over the past seven days, enters an amount, and gets an estimate of the converted value.
Actor	Registered User

Precondition	User is authenticated. ExchangeRate API is available.
Postcondition	The user has looked at both current and past exchange rates and gotten an estimate of the conversion. No transaction is executed.
Main Flow	1. The user goes to the Transfer Calculator. 2. The user chooses the currencies to send and receive. 3. The system gets the current exchange rate and the historical rates for the last seven days from the ExchangeRate API. 4. The system shows the current rate and the seven-day line chart. 5. The user types in the amount they want to change. 6. The system figures out and shows the estimated converted amount at the current rate. 7. The user can either do a conversion (UC-17) or leave without doing anything.
Alternative Flow	3a. The ExchangeRate API is down, so the system shows a cached rate with a warning that it may be out of date. 7a. The user leaves without making a transaction, and no changes are made to any wallet.

3.1.3 Activity Diagram

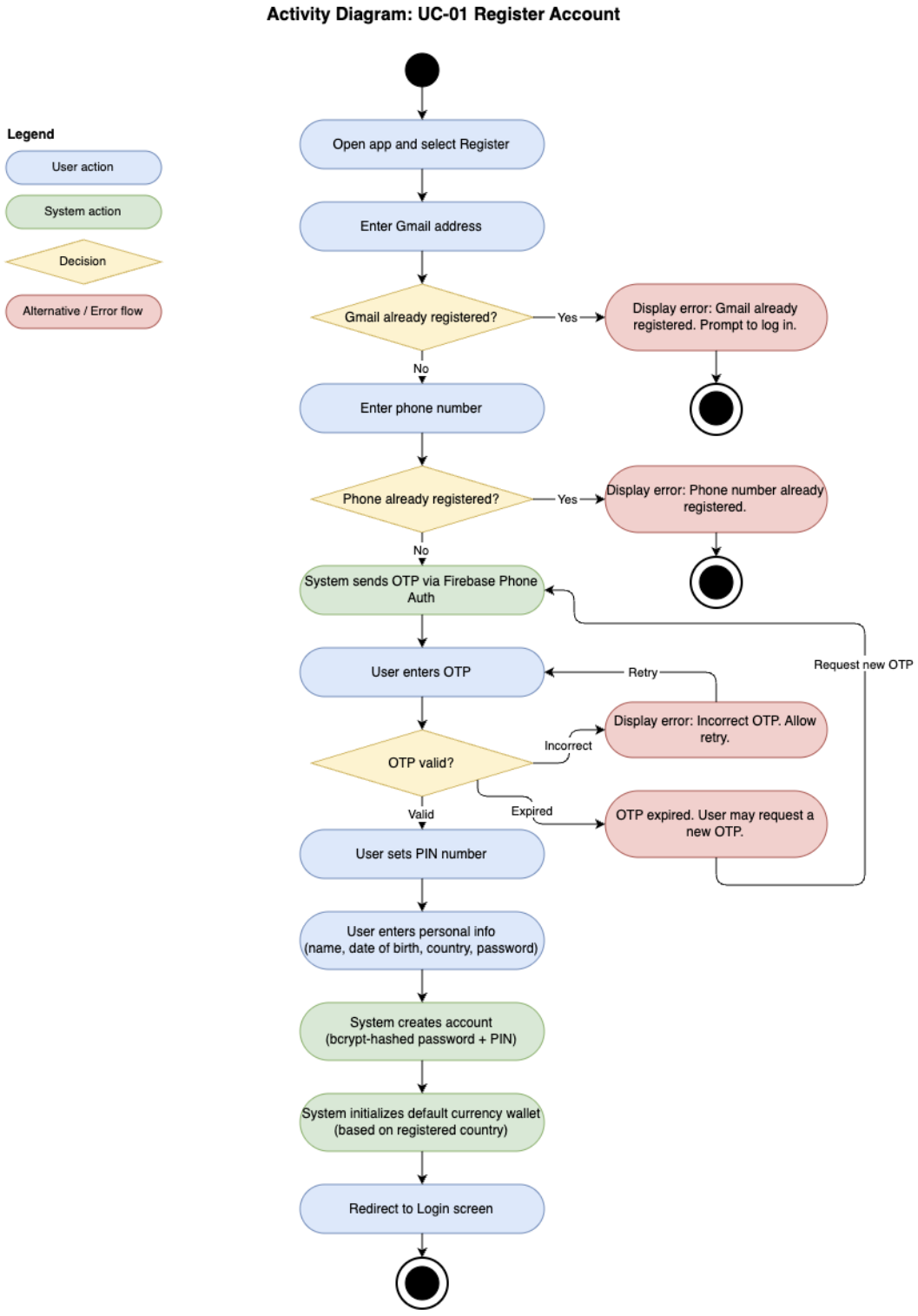

Figure 3.1.3.1 Activity Diagram UC-01 Register Account

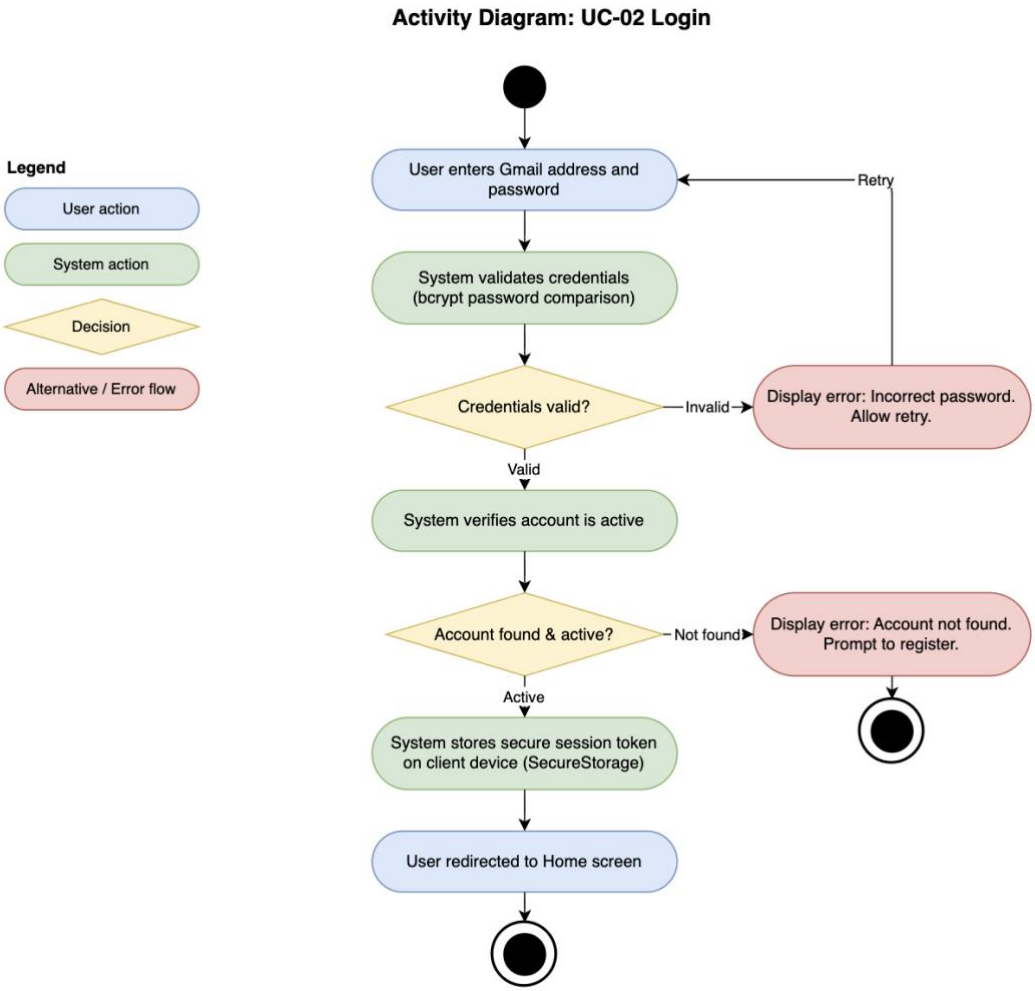

Figure 3.1.3.2 Activity Diagram UC-02 Login

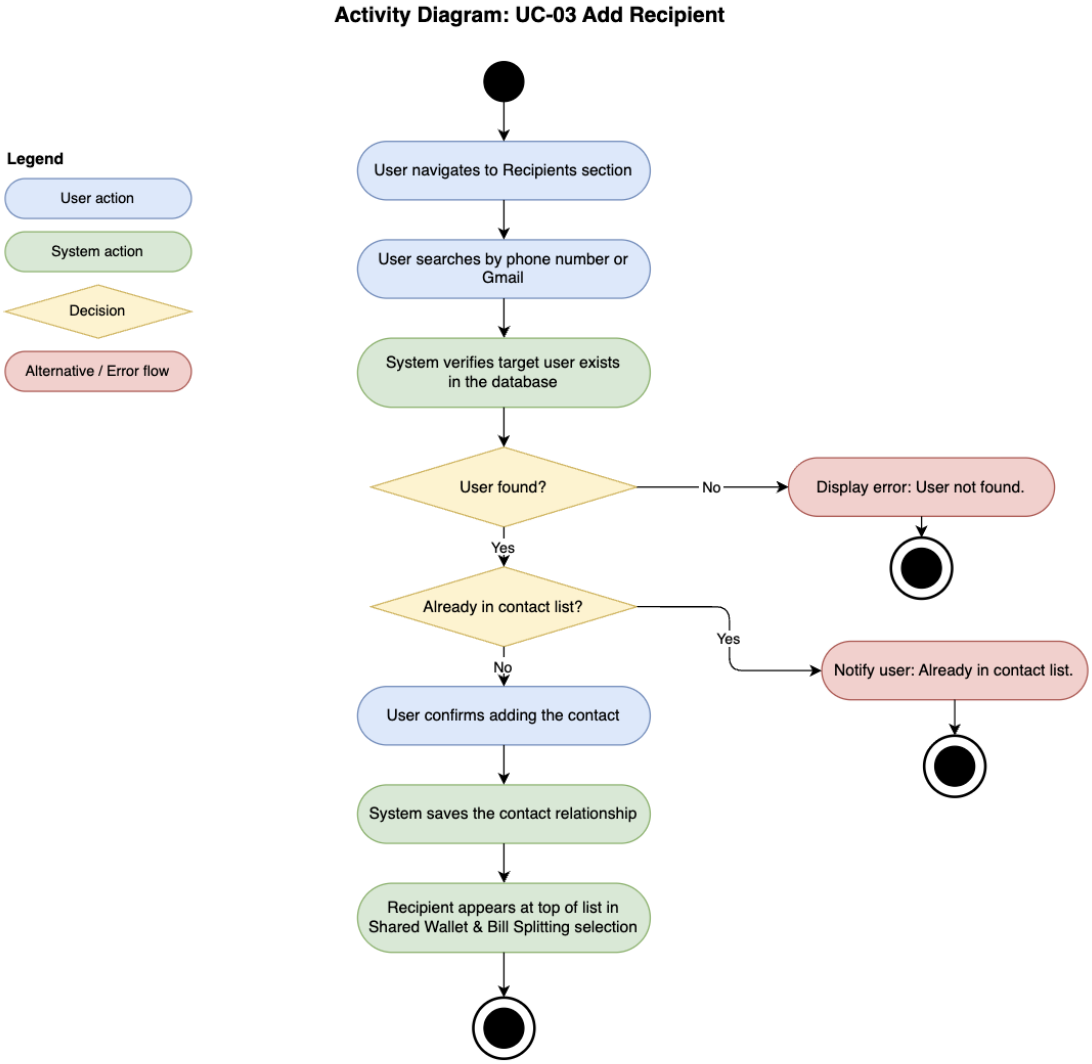

Figure 3.1.3.3 Activity Diagram UC-03 Add Recipient

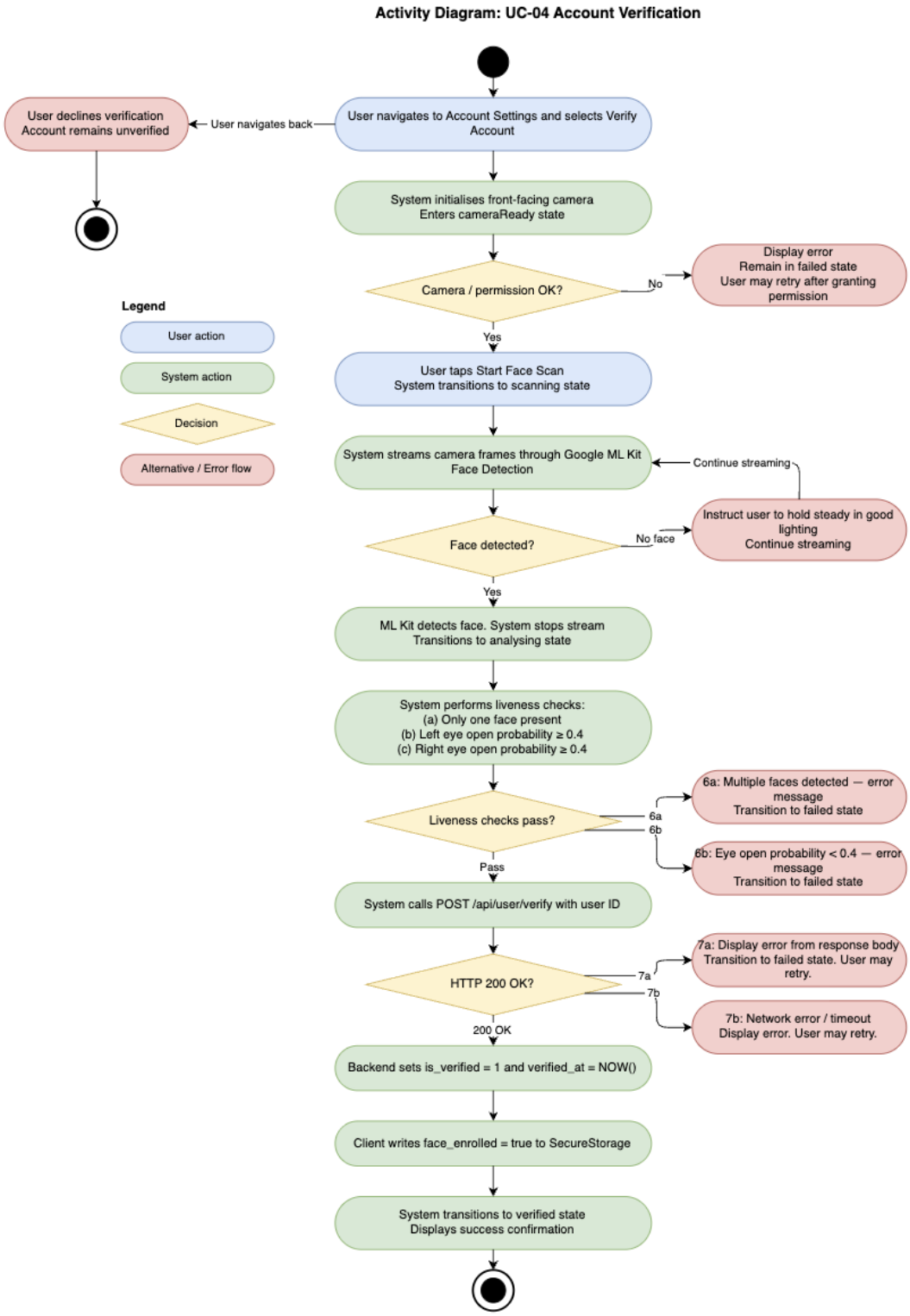

Figure 3.1.3.4 Activity Diagram UC-04 Account Verification

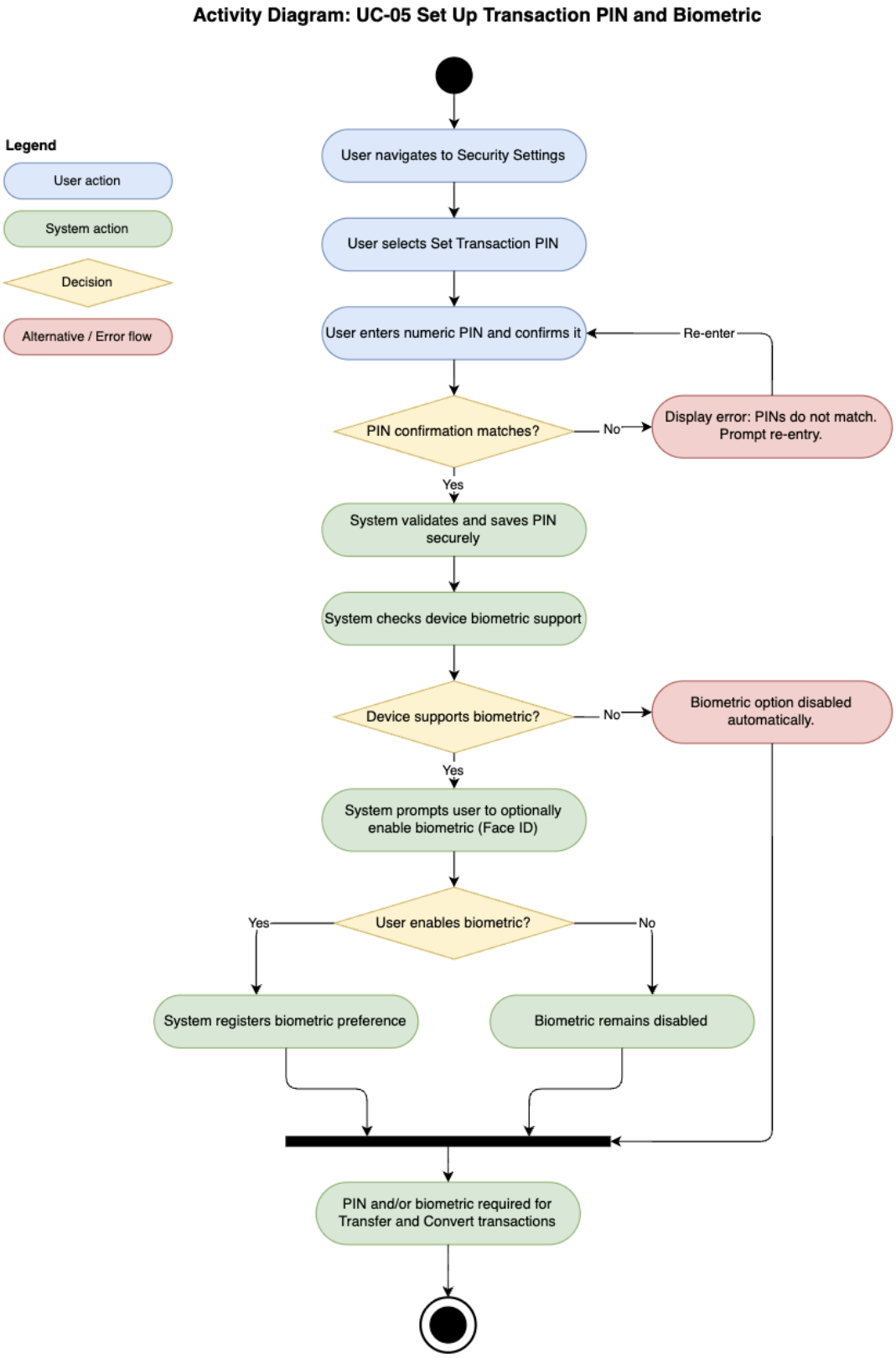

Figure 3.1.3.5 Activity Diagram UC-05 Set Up Transaction PIN and Biometric

Figure 3.1.3.6 Activity Diagram UC-06 Edit Profile

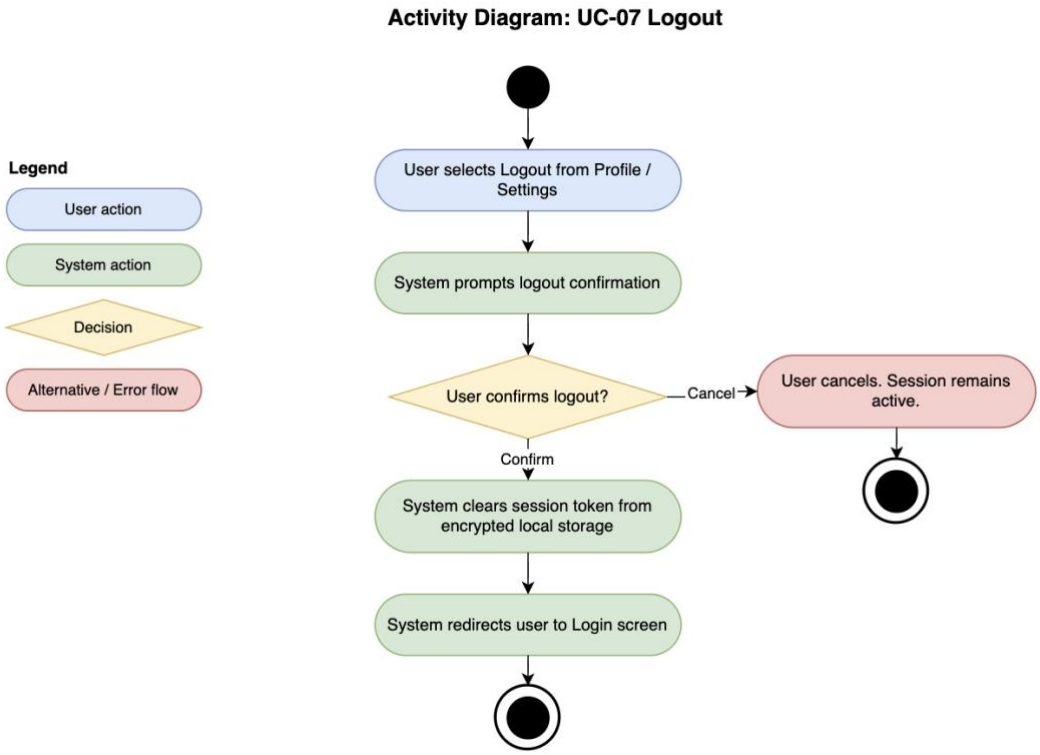

Figure 3.1.3.7 Activity Diagram UC-07 Logout

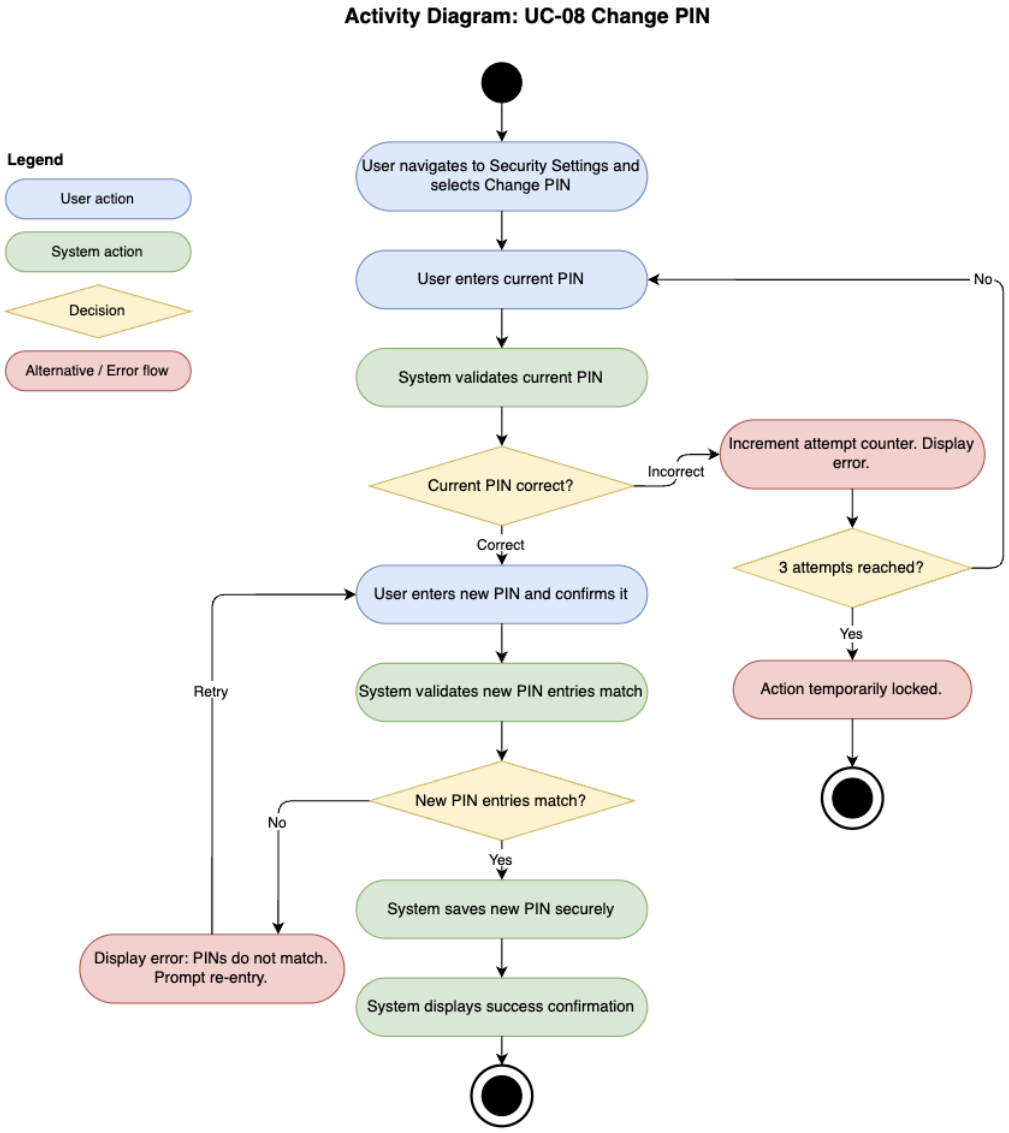

Figure 3.1.3.8 Activity Diagram UC-08 Change PIN

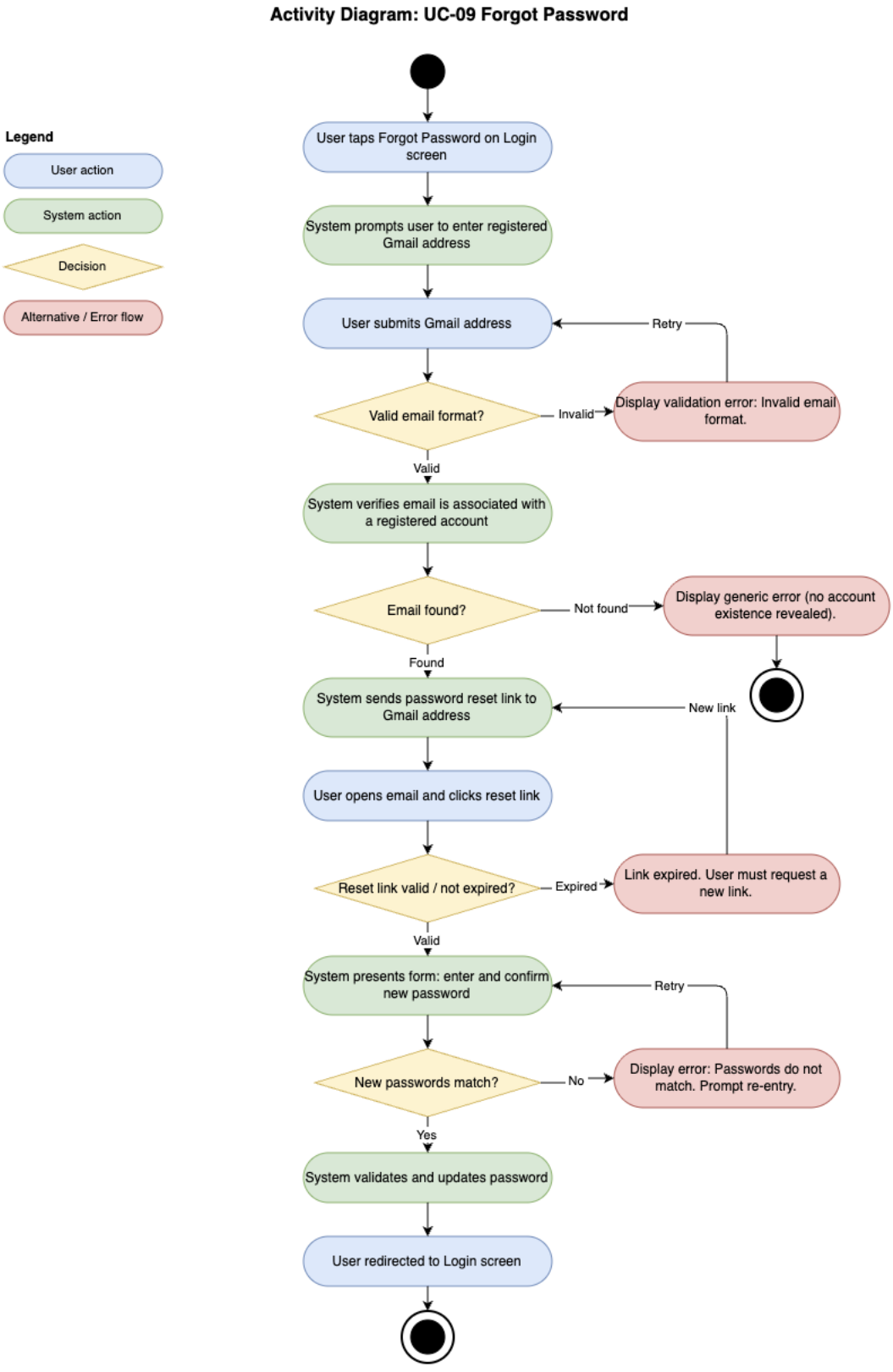

Figure 3.1.3.9 Activity Diagram UC-09 Forgot Password

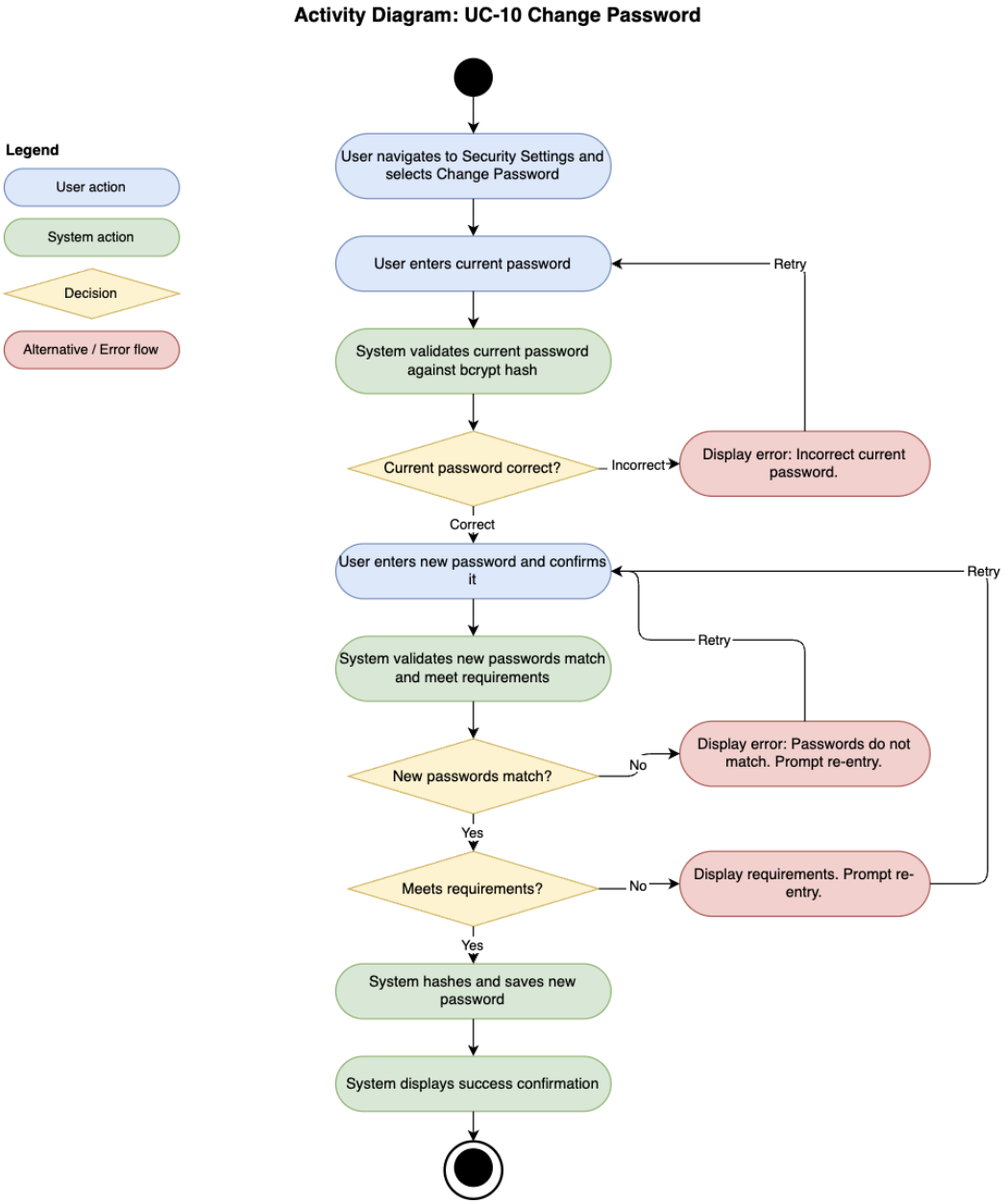

Figure 3.1.3.10 Activity Diagram UC-10 Change Password

Figure 3.1.3.11 Activity Diagram UC-11 Change Email

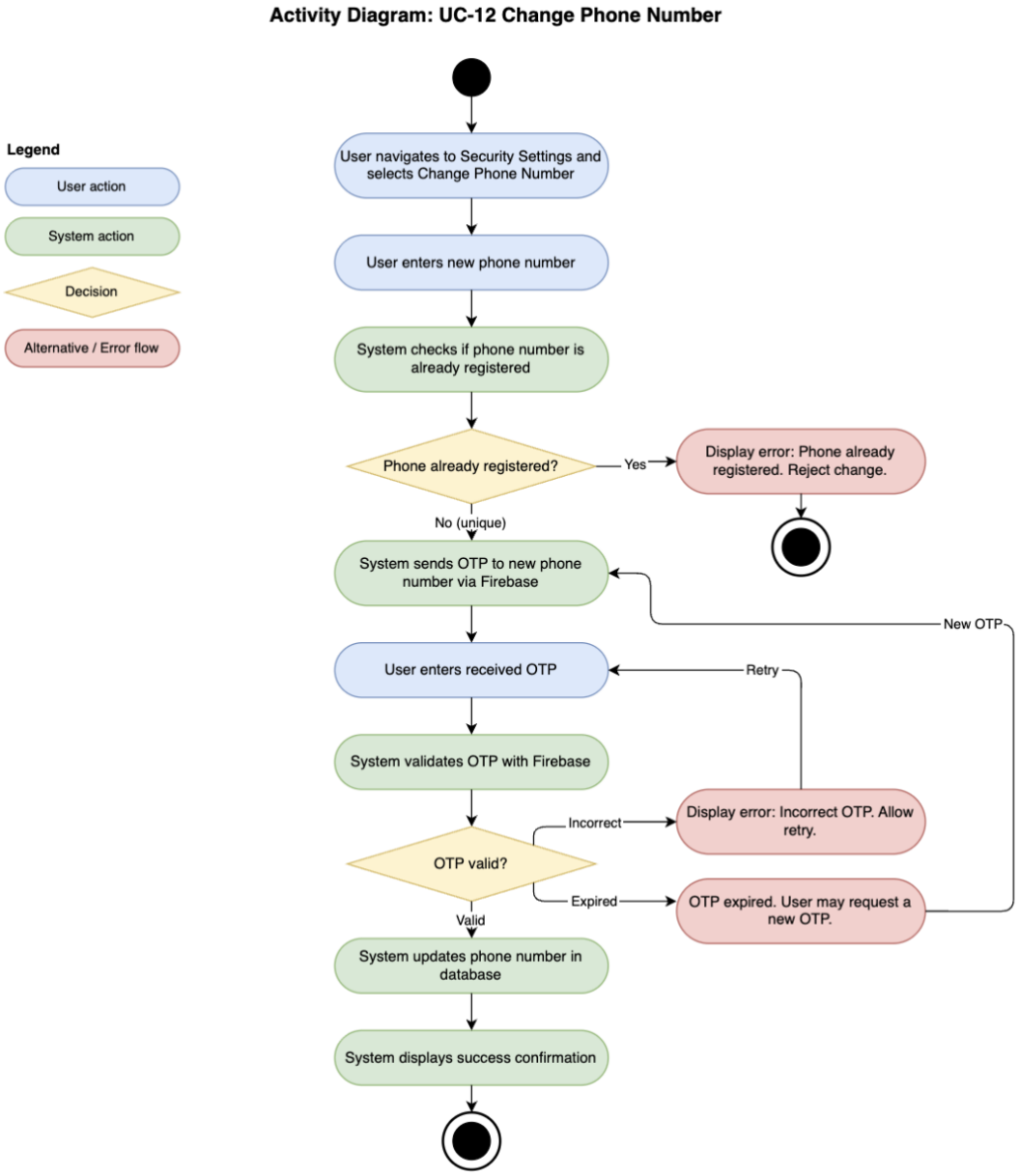

Figure 3.1.3.12 Activity Diagram UC-12 Change Phone Number

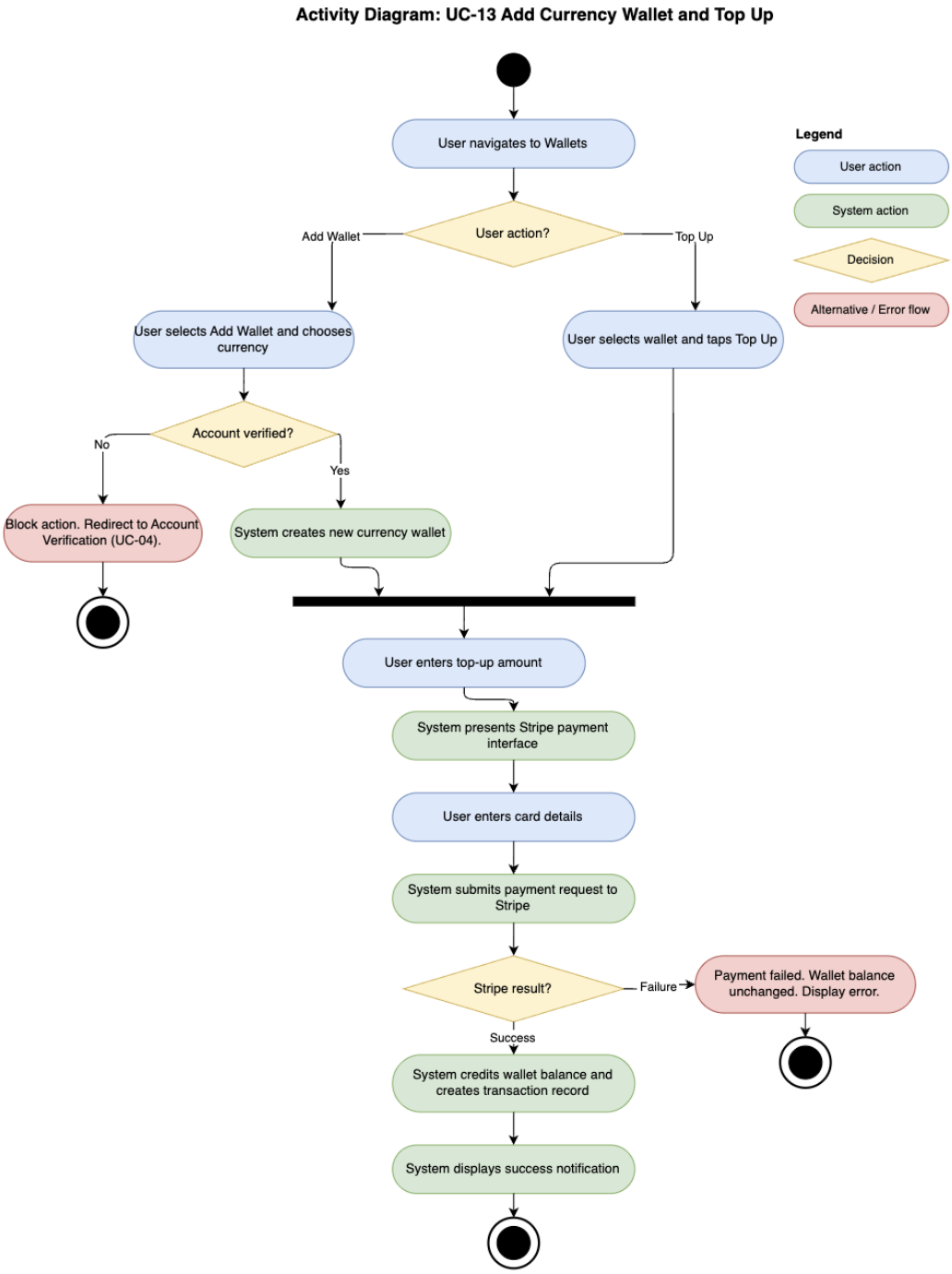

Figure 3.1.3.13 Activity Diagram UC-13 Add Currency Wallet and Top Up

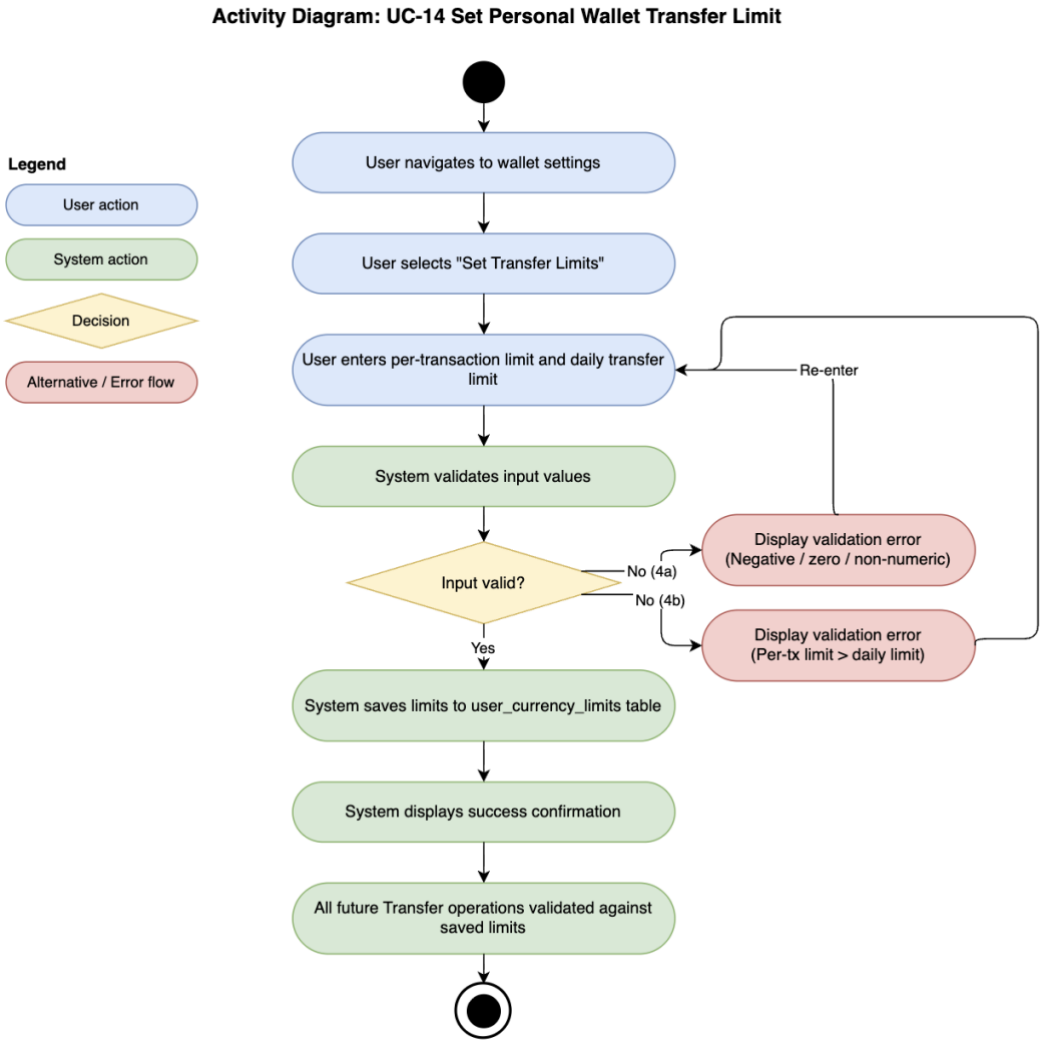

Figure 3.1.3.14 Activity Diagram UC-14 Set Personal Wallet Transfer Limit

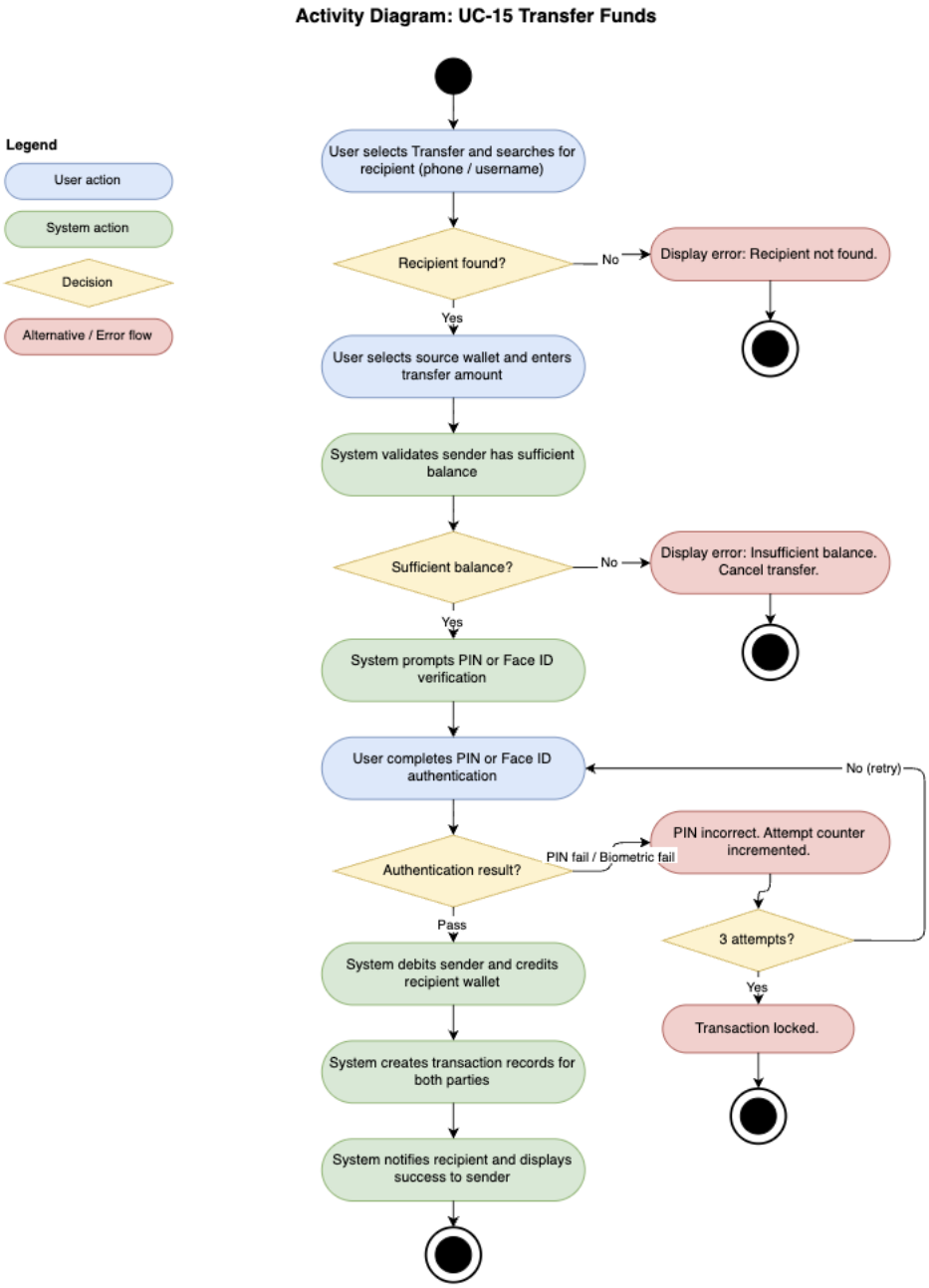

Figure 3.1.3.15 Activity Diagram UC-15 Transfer Funds

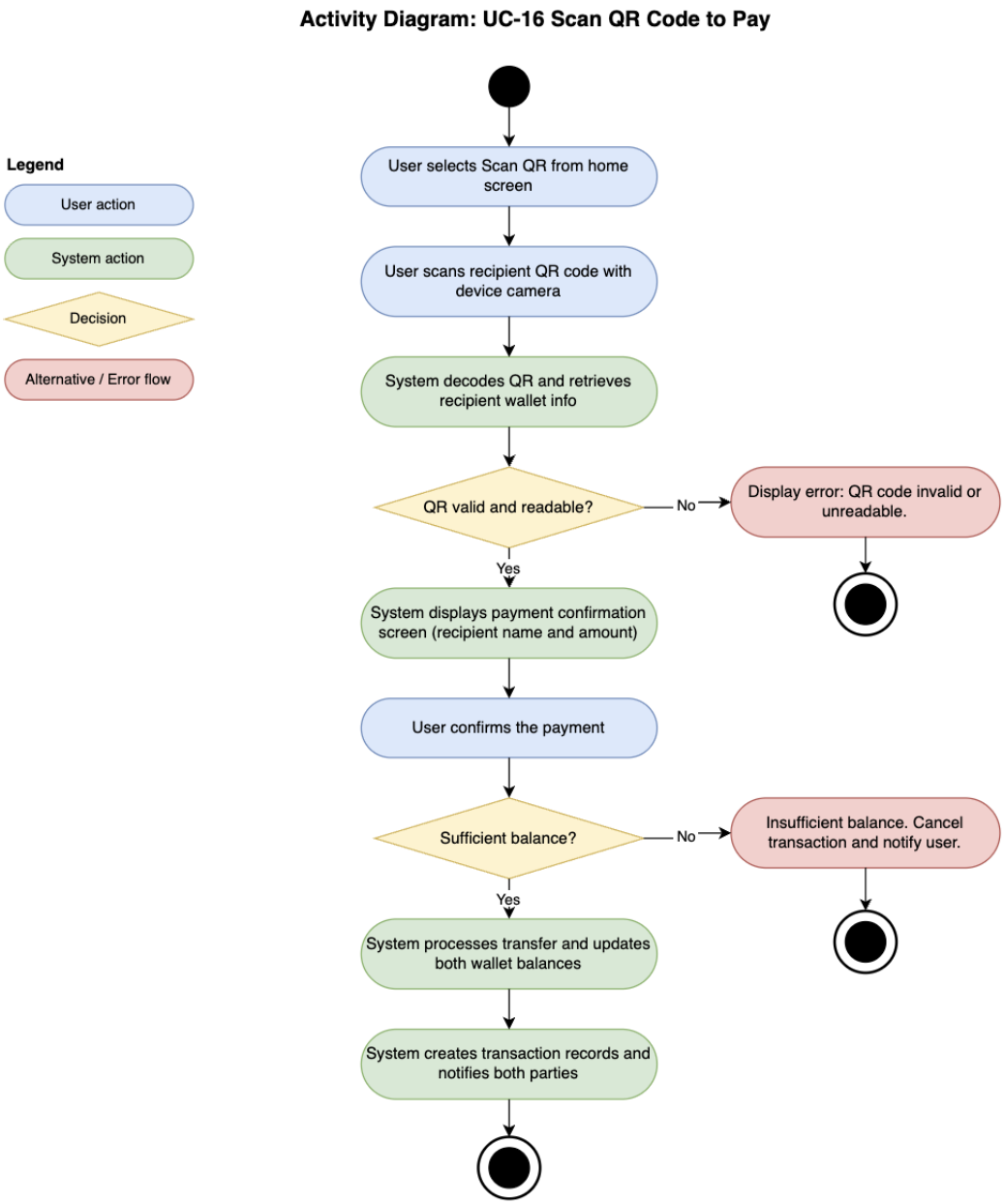

Figure 3.1.3.16 Activity Diagram UC-16 Scan QR Code to Pay

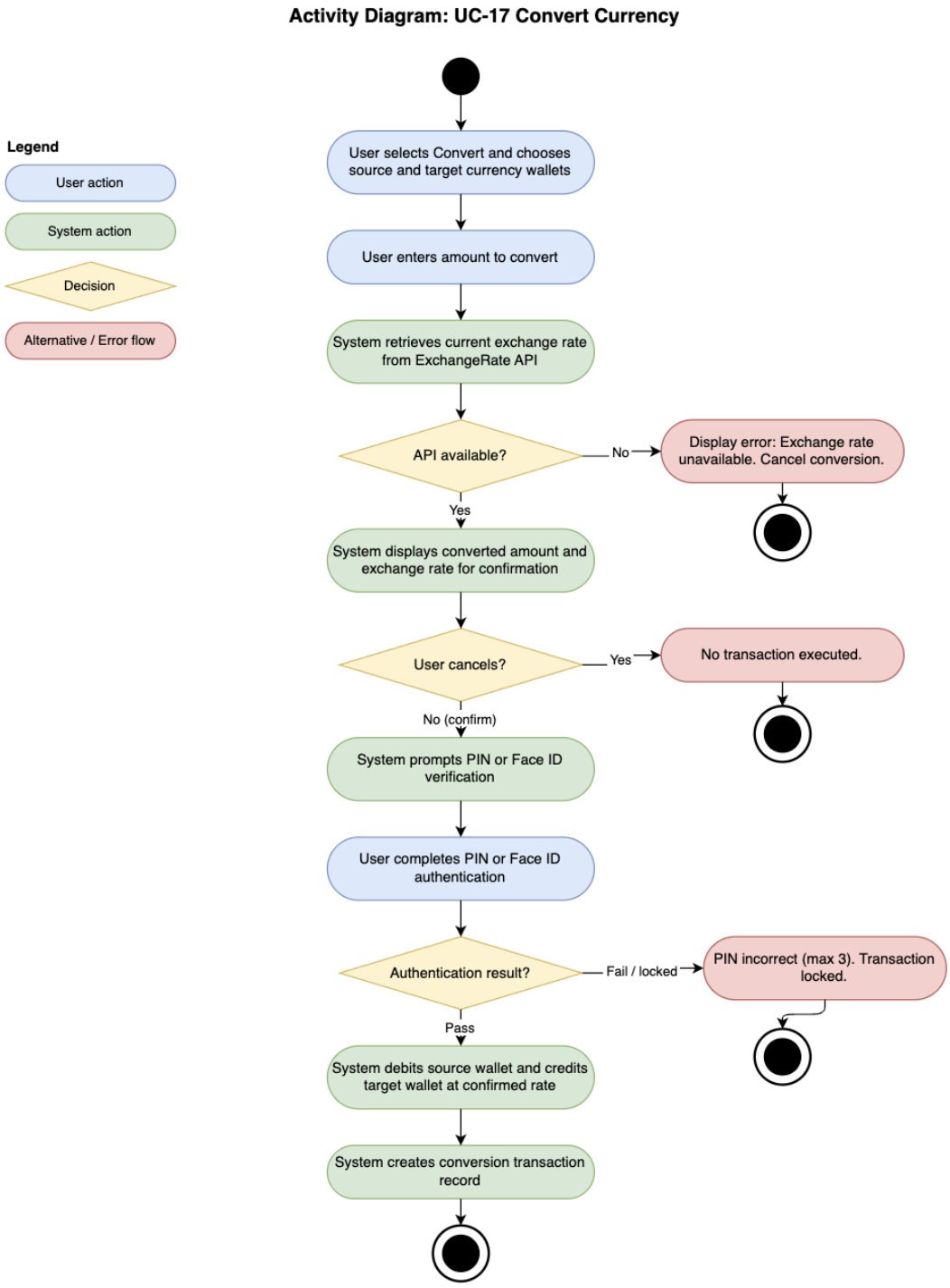

Figure 3.1.3.17 Activity Diagram UC-17 Convert Currency

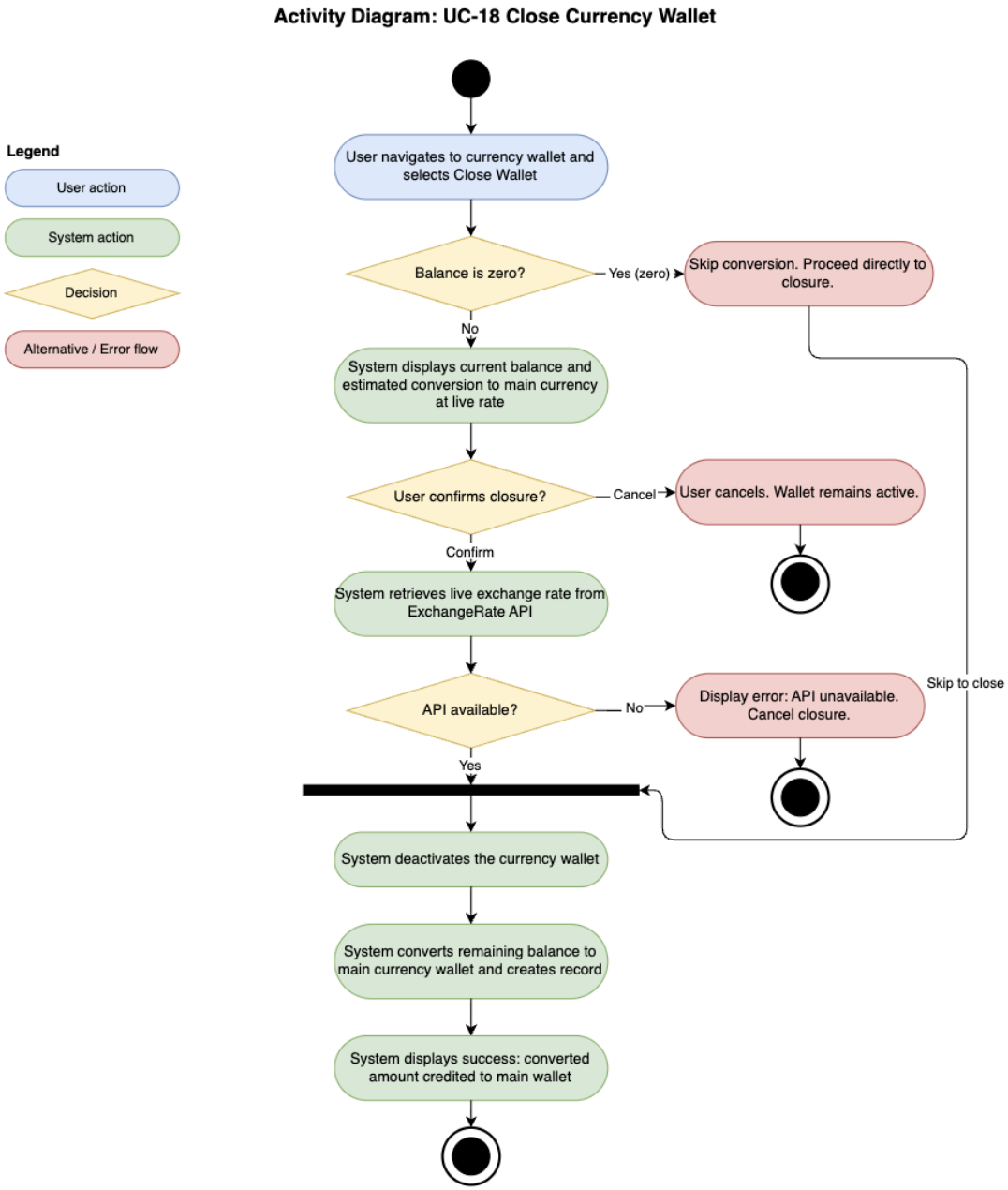

Figure 3.1.3.18 Activity Diagram UC-18 Close Currency Wallet

Figure 3.1.3.19 Activity Diagram UC-19 Create Shared Wallet

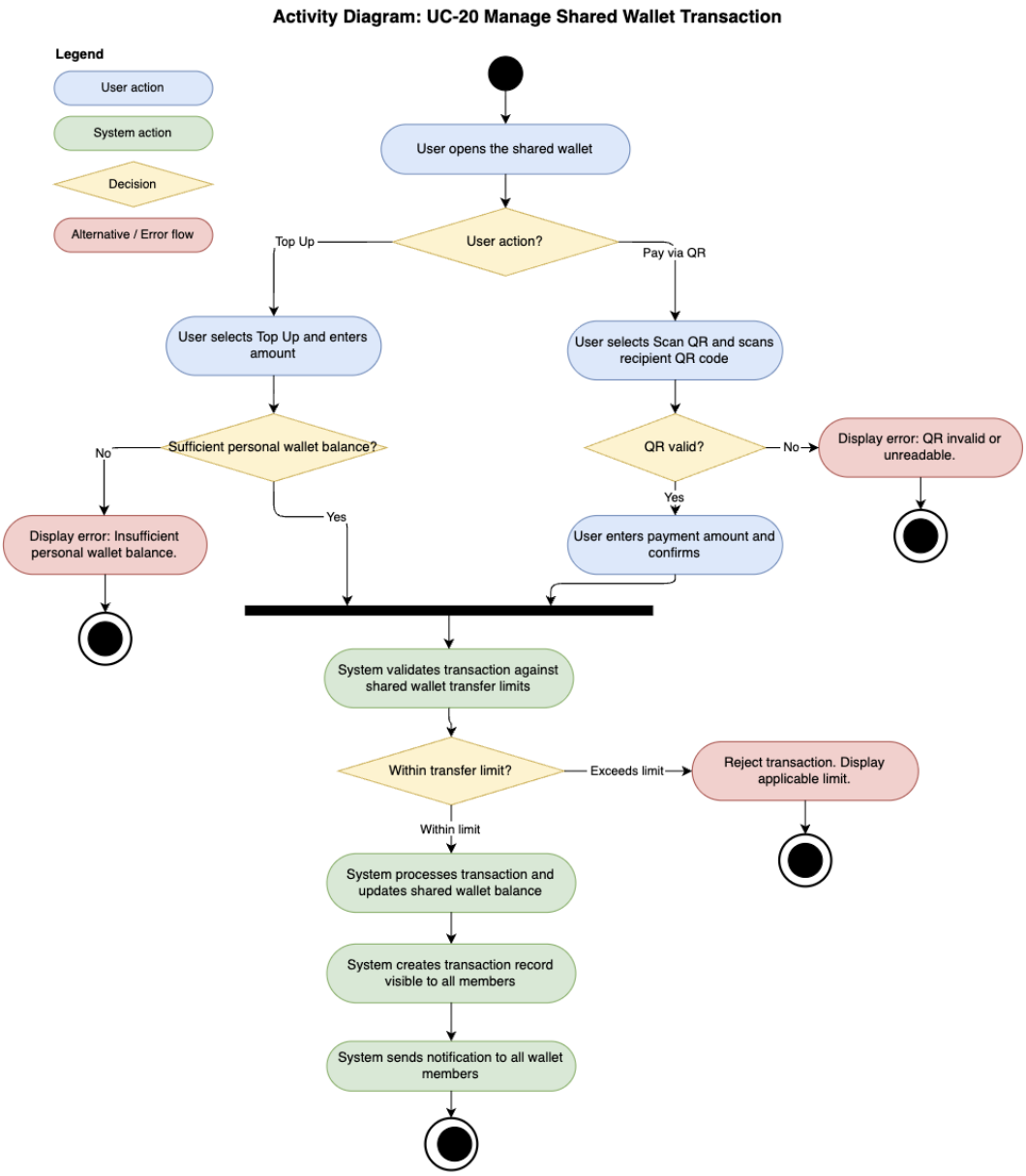

Figure 3.1.3.20 Activity Diagram UC-20 Manage Shared Wallet Transaction

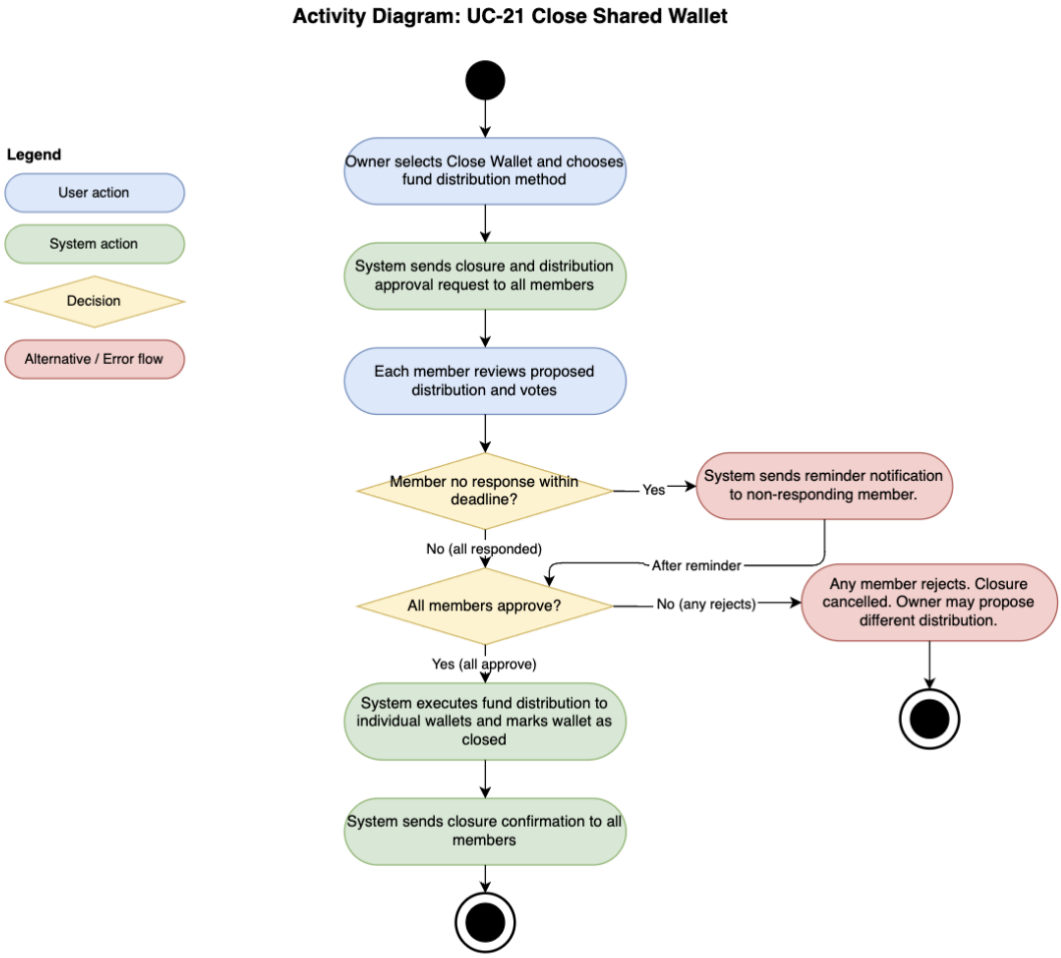

Figure 3.1.3.21 Activity Diagram UC-21 Close Shared Wallet

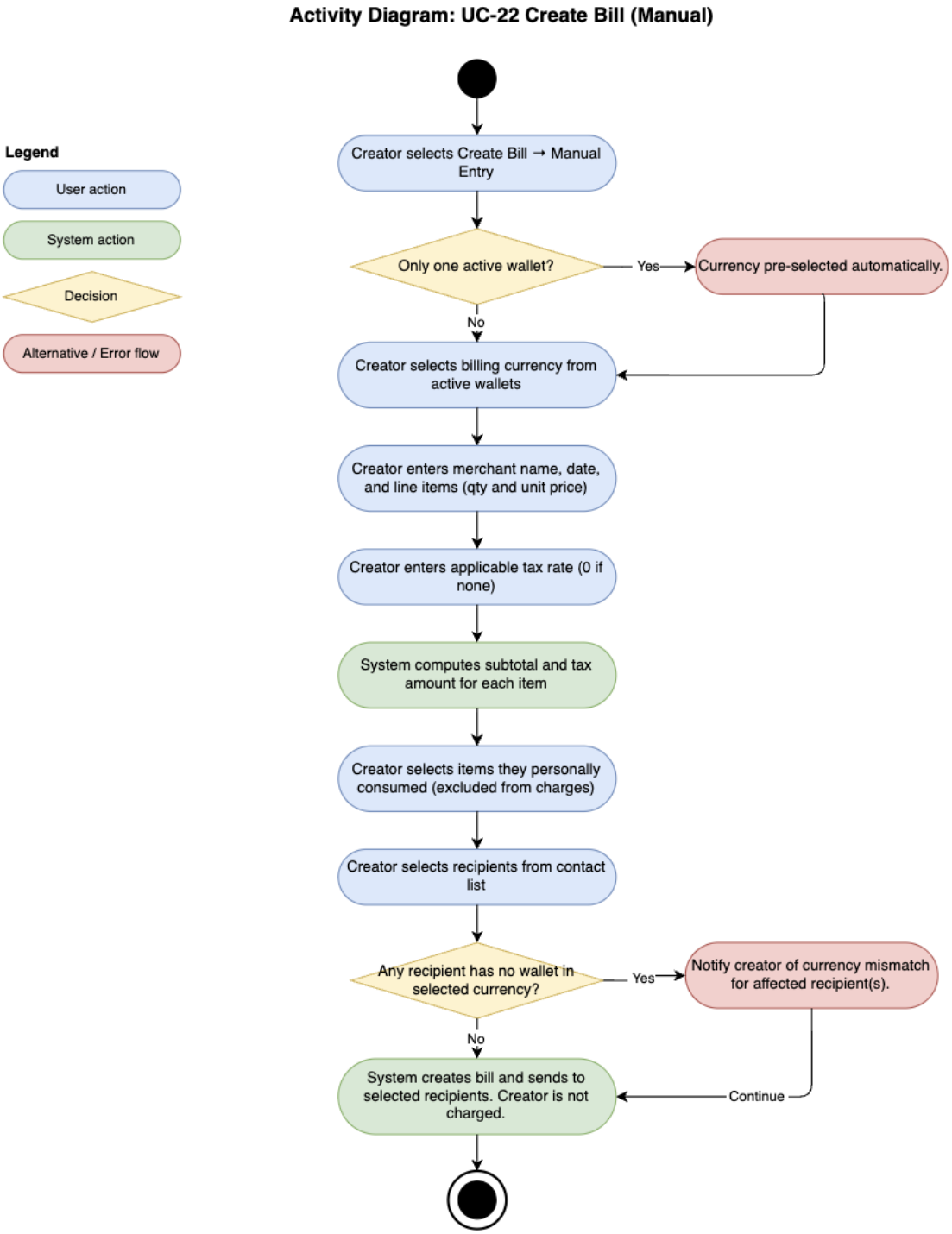

Figure 3.1.3.22 Activity Diagram UC-22 Create Bill (Manual)

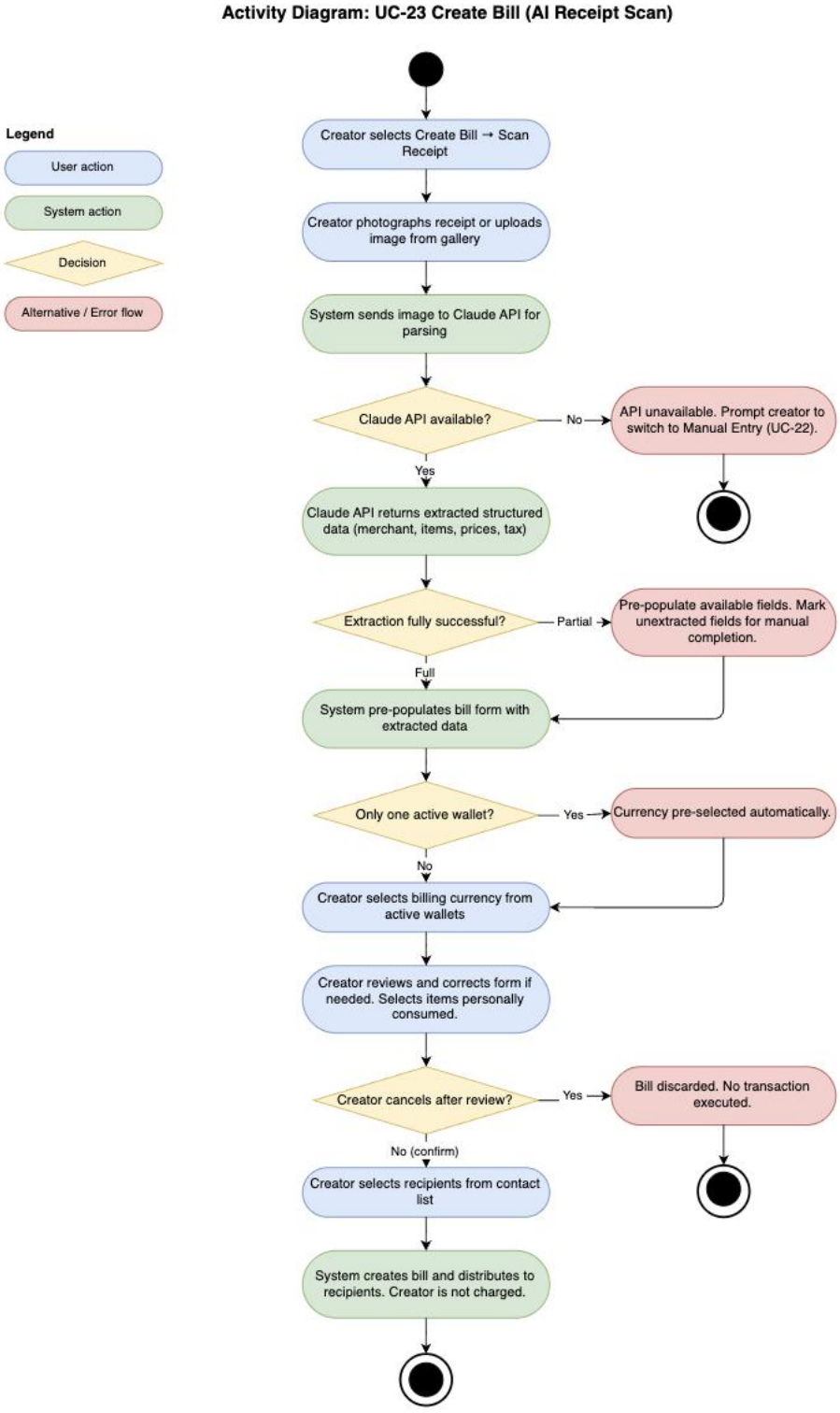

Figure 3.1.3.23 Activity Diagram UC-23 Create Bill (AI Receipt Scan)

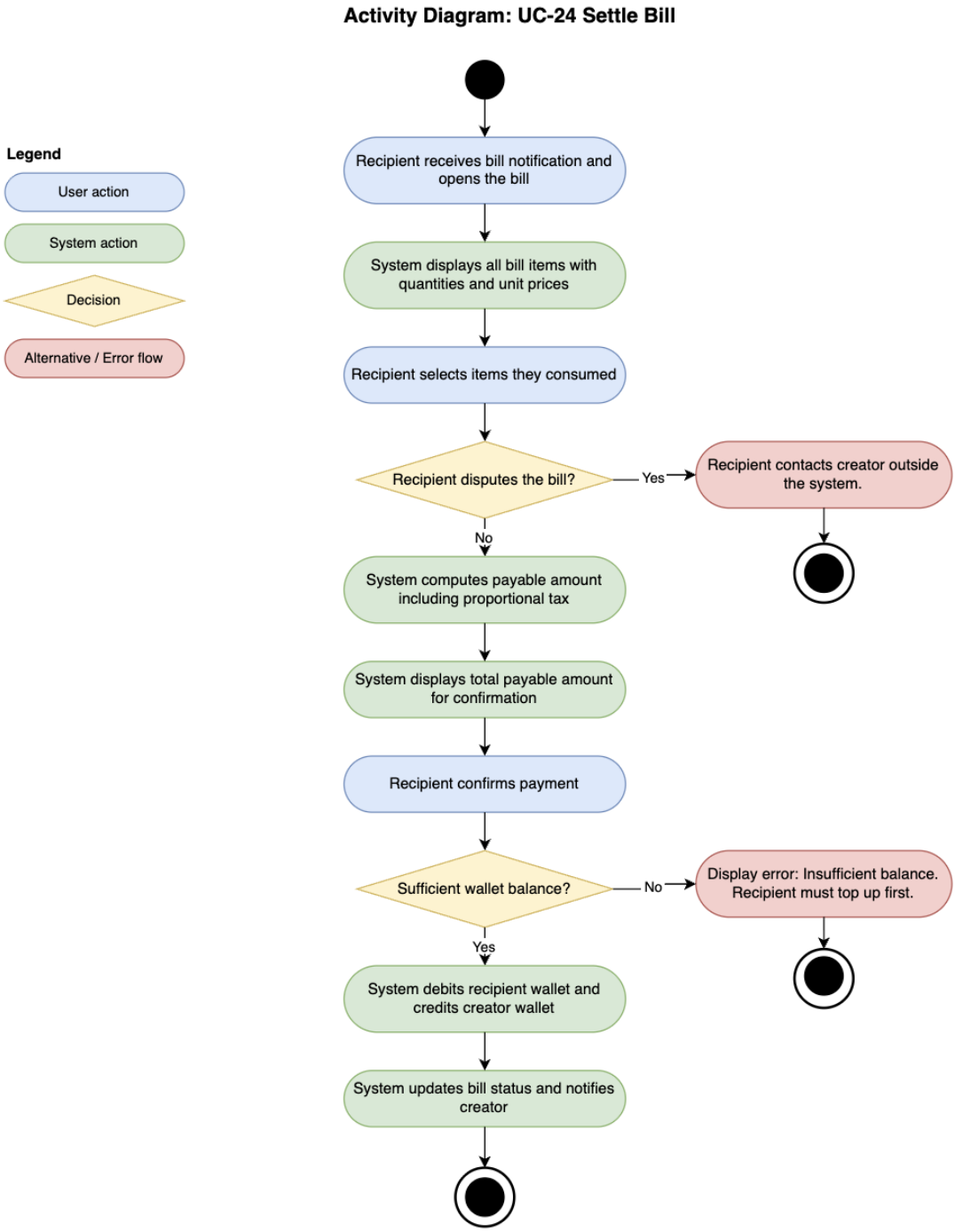

Figure 3.1.3.24 Activity Diagram UC-24 Settle Bill

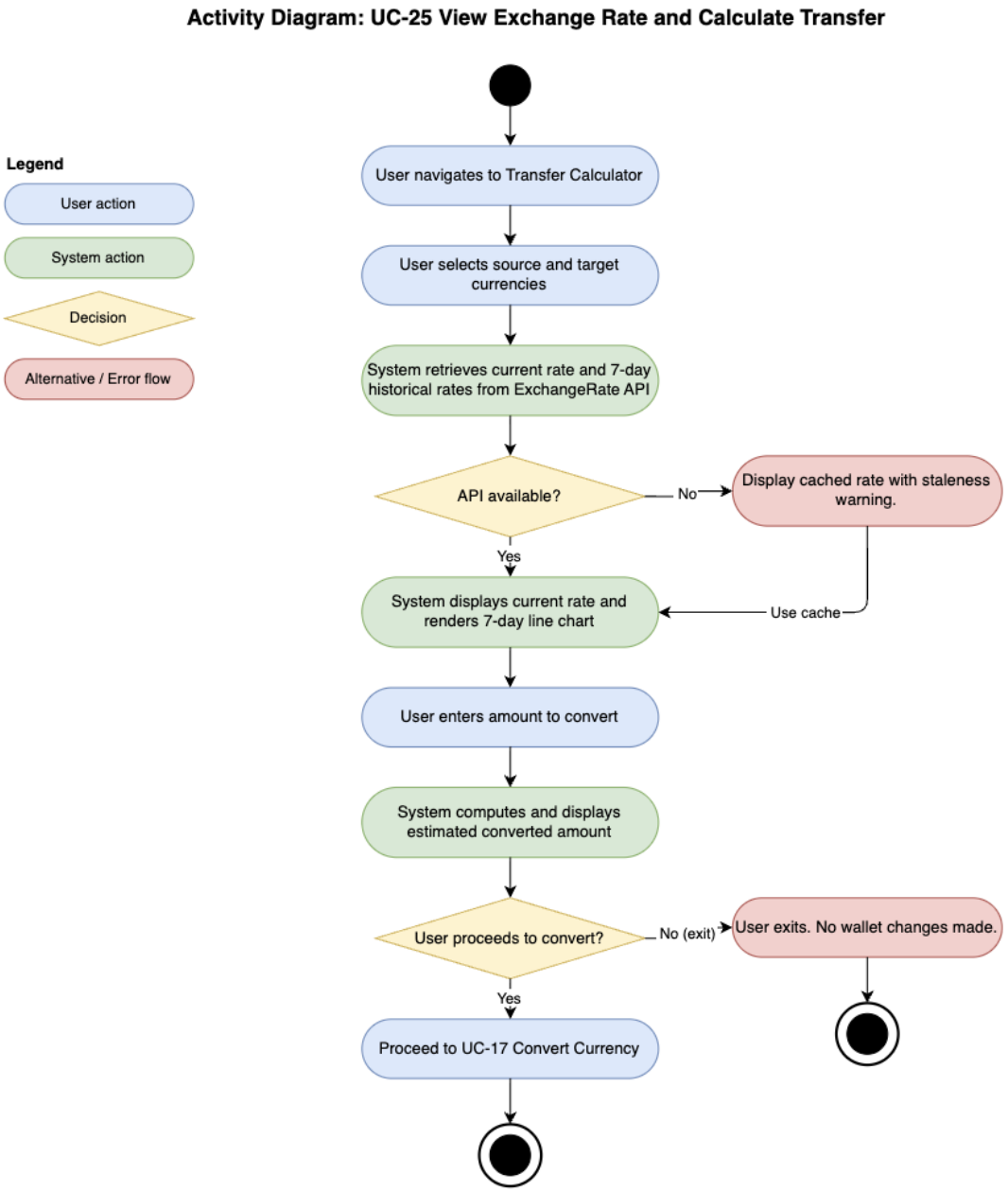

Figure 3.1.3.25 Activity Diagram UC-25 View Exchange Rate and Calculate Transfer

3.2 Methodology

The Incremental Software Development Model is the method used in this project to make the system work. This model has four steps: developing, testing, and validating the application. Each step adds a fully functional module that builds on the work done in the previous step. This method works well for the proposed e-wallet system because it has many complex, interdependent modules, such as authentication and wallet infrastructure, multi-currency management, shared wallet collaboration, and AI-assisted bill splitting. Each module can be defined, built, and tested on its own before the next one starts.

The incremental model lowers the risk of development by making sure that the core infrastructure is stable before adding more advanced features on top of it. It also makes it possible to deliver functional parts early, which lets you improve them based on feedback from testing at each step. Figure 3.2.1 shows how the whole project will work under this model.

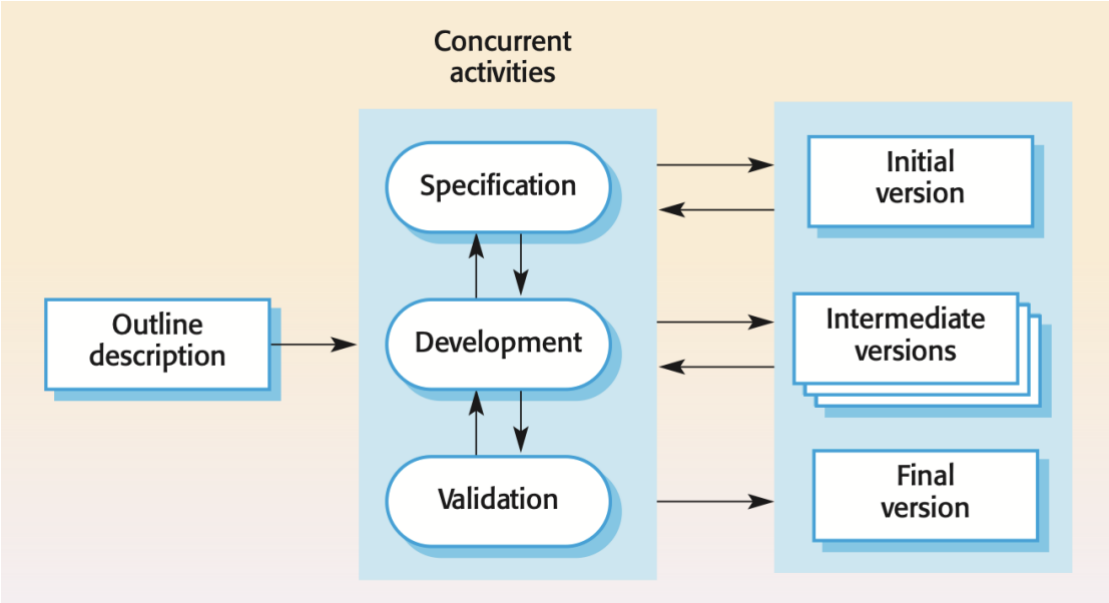

Figure 3.2.1 Project workflow in incremental development model

3.2.1 Outline Description

The system is divided into four parts, each of which is linked to a functional module found in the use case analysis. The increment boundaries are meant to take into account dependencies between modules: each increment adds features that are needed for the next one.

Increment 1 (Core Authentication and Wallet Management) sets up the basic infrastructure that all the other increments will build on. It includes signing up users with Firebase OTP verification, Gmail and bcrypt password authentication, Google ML Kit liveness detection for account verification, setting up transaction PINs and iOS Face ID, and managing the entire personal wallet lifecycle, including Stripe-integrated top-ups, peer-to-peer fund transfers with PIN or Face ID authorization, QR code payments, and security management workflows (changing PIN, password, email, and phone number). This increase is equal to UC-01 through UC-16.

Increment 2, "Multi-Currency Support and Transfer Calculator," adds multi-currency support to the wallet infrastructure. It adds live currency conversion between the nine supported currencies using the ExchangeRate API, the ability to close a currency wallet and automatically move the balance to the main wallet, and the Transfer Calculator, which has a seven-day historical exchange rate chart. This increase is based on the verified account status and wallet infrastructure set up in Increment 1, and it is equal to UC-17, UC-18, and UC-25.

Increment 3 (Shared Wallet Management) adds features that let groups work together on finances. It lets users make shared wallets with customizable limits on transactions and daily transfers, invite people from their contact list, do group top-ups and QR payments against the shared wallet balance, and start closing the wallet based on consensus with four fund distribution methods: equal split, full transfer to one member, proportional by contribution, and smart distribution. This increase is based on the contact management, wallet, and notification systems from Increment 1 and is equal to UC-19 through UC-21.

Increment 4 (Group Bill Splitting) gives you workflows for creating and settling bills with AI and by hand. When you create a bill by hand, you can enter the merchant's information, the quantity and unit price of each item, the tax that applies, and any items that shouldn't be included before sending the bill to the people you want. The AI receipt scan flow uses the Claude API to automatically fill in bill fields from a picture of a

receipt. If the API is down, it will fall back to manual entry without any problems. After choosing what to buy and confirming payment, the system automatically calculates and pays the amount owed. This increase is based on the wallet debit and credit system from Increment 1 and is equivalent to UC-22 through UC-24.

3.2.2 Specification

At the start of each increment, functional and non-functional requirements are set to help with implementation and set the criteria for validation. The functional requirements listed below are directly related to the use cases in Section 3.1.2.

Increment 1 — Core Authentication and Wallet Management (UC-01 to UC-16)

1. The system must let a new user sign up through a multi-step process that includes checking the uniqueness of their Gmail account, their phone number, their Firebase OTP, setting up a PIN, and entering their personal information (UC-01).
2. The system must verify the identity of registered users by checking their Gmail address and bcrypt-verified password. It must also store a secure session token in encrypted local storage after a successful login (UC-02).
3. The system must let users add registered contacts by phone number or Gmail. These contacts will then be put at the top of the shared wallet and bill splitting recipient lists (UC-03).
4. Google ML Kit liveness detection will be used to verify the account once, and if it is successful, the verification status will be updated in the database, allowing the creation of a multi-currency wallet (UC-04).
5. The system must let users set up a numeric transaction PIN and, if they want, use Face ID on their device as another way to verify their identity for Transfer and Convert operations (UC-05).
6. The system must allow users to edit their profiles (display name and country), log out and clear their session tokens, and manage security workflows like changing their PIN, getting a new password via email reset link, changing their email with a verification link, and changing their phone number with Firebase OTP (UC-06 to UC-12).

7. Verified users should be able to make currency wallets in any of the nine supported currencies, and all users should be able to add money to any existing wallet using a Stripe card (UC-13).
8. Users should be able to set daily and per-transaction transfer limits on their personal currency wallets. These limits should apply to all future transfers from those wallets (UC-14).
9. The system must allow peer-to-peer fund transfers between wallets that use the same currency and require a PIN or Face ID to approve the transfer. It must also allow QR code-based payments, where the recipient's wallet information is encoded in the QR code (UC-15, UC-16).

Increment 2 — Multi-Currency Support and Transfer Calculator (UC-17, UC-18, UC-25)

1. The system must let users move money between any two of the nine supported currency wallets at the live exchange rate retrieved from the ExchangeRate API. Before the transaction can happen, the user must enter their PIN or Face ID (UC-17).
2. The system should let users close a wallet that isn't the main currency wallet. Before the wallet is deactivated, it should automatically convert any remaining balance to the main currency wallet at the live exchange rate (UC-18).
3. The system must have a Transfer Calculator that shows the current exchange rate and a seven-day historical trend chart for any currency pair that the user chooses. It must also be able to calculate an estimated converted amount for a value that the user enters without making a transaction (UC-25).
4. If the ExchangeRate API is down, the system should show the most recently cached rate with a warning that it is out of date instead of failing the operation completely.

Increment 3 — Shared Wallet Management (UC-19 to UC-21)

1. The system should let verified users make a shared wallet by giving it a name, a currency, optional limits on how much money can be transferred each day and per transaction, and inviting people from their contact list (UC-19).

2. The system must let any member add money to the shared wallet by taking it from their own wallet or making QR-based payments from the shared wallet balance, as long as the wallet's transfer limits are set (UC-20).
3. The system must make a record of every transaction in the shared wallet that all members can see in real time (UC-20).
4. The system shall support consensus-based shared wallet closure initiated by the owner, requiring all members to approve the proposed fund distribution method (equal split, full transfer, proportional by contribution, or smart distribution) before executing the closure (UC-21).
5. The system will send a reminder to any member who hasn't responded to a closure request within the time frame set (UC-21).

Increment 4 — Group Bill Splitting (UC-22 to UC-24)

1. The system should let a bill creator enter merchant information, line items with quantity and unit price, the tax rate that applies, and personal item exclusions by hand. Then, the system should send the bill to the people who need it (UC-22).
2. The system must let the person who makes the bill take a picture or upload a receipt and use the Claude API to automatically pull out the merchant name, line items, quantities, unit prices and tax into a pre-filled bill form for the creator to check and fix (UC-23).2. The system must let someone who makes a bill take a picture of or upload a receipt and use the Claude API to automatically pull out the merchant's name, line items, quantities, unit prices, and tax into a pre-filled bill form for the creator to check and fix (UC-23).
3. If the Claude API is down or extraction fails partially, the system should either send the user to manual entry or fill in the available fields and mark the unextracted fields for manual completion (UC-23).
4. The system should let each bill recipient choose the items they want to buy on their own. After that, the system should figure out the total amount due, including the proportional tax, and automatically debit the wallet and credit the creator's account when payment is confirmed (UC-24).

Non-Functional Requirements

The following non-functional requirements must be followed at all times during the development process. Table 3.2.2.1 shows each requirement next to its reason for being in the system design.

Table 3.2.2.1 Non-Functional Requirements

Category	Requirement	Rationale
Security	Before processing, all financial transactions must be verified with a PIN or device Face ID.	Stops unauthorized money transfers and is in line with transaction-level security design (UC-05, UC-15, UC-17).
Security	Passwords should be stored using bcrypt hashing, and session tokens should be kept in SecureStorage, which is encrypted local storage.	Keeps credentials safe from brute-force and storage-access attacks when they are not in use and when they are being sent.
Security	After three wrong attempts in a row, the PIN entry will be locked.	Reduces the risk of brute-force PIN attacks on Transfer and Convert operations (UC-08, UC-15, UC-17).
Data Consistency	All financial transactions, such as adding money, moving money, converting money, and paying bills, must be done as ACID-compliant database transactions.	Keeps wallet balances the same even when multiple users are using the same wallet at the same time.
Availability & Resilience	When the ExchangeRate API is down, the system should show the last cached exchange rate with a warning that it is old.	Keeps the transfer calculator and conversion display working even when the API is down (UC-17, UC-25).

Availability & Resilience	If the Claude API isn't available, the system should send the user to manual bill entry (UC-22) instead of failing without saying anything.	Keeps the ability to split bills even if a third-party AI service isn't available (UC-23).
Performance	Under normal network conditions, API responses for standard wallet operations (like transferring money or checking your balance) should finish in three seconds.	Makes sure that time-sensitive financial operations have a responsive user experience.
Usability	Users should be able to complete common tasks like transferring money, adding money, and paying bills in five steps from the home screen.	Directly addresses the multi-application workflow problem mentioned in the problem statement by reducing interaction friction.
Scalability	The system must be able to handle multiple shared wallets at the same time, with transactions happening at the same time from different members without corrupting data.	Needed for the shared wallet module, which lets more than one member start a top-up or QR payment at the same time.
Platform	The app should work on iOS and use the Flutter local_auth package to call native Face ID for transaction verification.	Makes sure that biometric authentication works completely on the device without the need for a server, which protects privacy and performance.

3.2.3 Development

The same development workflow is used for each increment: backend API endpoints are built and unit-tested on the Node.js server before the Flutter UI

components that will use them are built. This API-first approach makes sure that the business logic is checked without the presentation layer and that the RESTful contract between the client and server is stable before UI integration starts.

The Flutter frontend is organized by feature-based directories, with each module (authentication, wallet, shared wallet, bill splitting, calculator) having its own directory that holds screens, widgets, and service classes. The Node.js backend is set up as a single Express app, with route controllers, service functions, and database access objects grouped by what they do. When each development increment is made, MySQL database schema migrations are applied one at a time. This makes sure that schema changes are versioned and can be undone.

The Node.js backend has separate service modules for third-party service integrations like Stripe, ExchangeRate API, Claude API, and Firebase. This keeps API credentials and business logic on the server side and out of the Flutter client. Each integration with an outside service has a clear way to handle errors that degrades gracefully to a different flow, like the cached exchange rate fallback and the manual bill entry fallback, as described in the use case alternative flows.

3.2.4 Validation

Before the next increment can be developed, each one must be validated. There are three levels of validation: unit testing of individual backend service functions and database operations; integration testing of the API endpoints against the MySQL database to make sure data is stored and retrieved correctly; and end-to-end functional testing of the entire user workflow through the Flutter client.

Validation for each increment is performed in accordance with the functional requirements delineated in Section 3.2.2, ensuring that each use case's primary flow and specified alternative flows are explicitly tested. Dedicated test scenarios are used to check the non-functional requirements in Table 3.2.2. These include security penetration tests for PIN lockout and session token handling, concurrency tests for shared wallet simultaneous transactions, and API unavailability simulations for the ExchangeRate API and Claude API fallback behaviours. Chapter 6 (System Evaluation and Discussion) talks about the results of these validation activities.

This step-by-step method, with required validation gates between steps, makes sure that any bugs that were added in earlier modules are found and fixed before they

spread to features that depend on them. This keeps the system stable throughout the development process.

CHAPTER 4 SYSTEM DESIGN

4.1 System Block Diagram

Figure 4.1.1 shows a system block diagram that breaks down the proposed e-wallet application into its main functional parts. It shows how the client layer, server layer, database layer, and external services layer depend on each other.

There are five functional UI modules in the Flutter Client layer, each of which corresponds to one of the application's five main feature sets: Auth & Account, Wallet Management, Shared Wallet, Bill Splitting, and Transfer Calculator. Each module only talks to its own backend service through HTTP REST calls. This makes sure that the presentation tier is clearly separated from the other tiers.

The Node.js/Express Server layer has five services that handle all of the business logic. The Auth Service is in charge of making session tokens, checking PINs, and checking credentials. The Wallet Service takes care of balance checks, peer-to-peer transfers, QR payments, and changing one currency into another. The Shared Wallet Service handles making group wallets, managing members, keeping track of activities, and calculating settlements. The Bill Splitting Service handles receipt parsing, item allocation, and automatic payment deduction. The Rate Service gets and stores foreign exchange data for both the Transfer Calculator and conversion operations.

The MySQL Database layer keeps track of all the application's state across five logical data groups that are related to their services: Users & Sessions, Wallets & Transactions, Shared Wallets & Members, Bills & Allocations, and Exchange Rate History.

There are four third-party integrations in the External Services layer, and they can only be accessed through the Node.js backend. Payment Intents are how Stripe handles adding money to card-based wallets. The ExchangeRate API gives the Wallet Service and the Rate Service real-time foreign exchange rates. The Bill Splitting Service sends receipt images and structured prompts to the Claude API, which then

sends back parsed item data. Firebase takes care of sending and checking OTPs during the user registration process.

Figure 4.1.1 shows that the Node.js backend handles all communication between the Flutter client and outside services. This makes sure that API credentials and business logic are never shown to the client device.

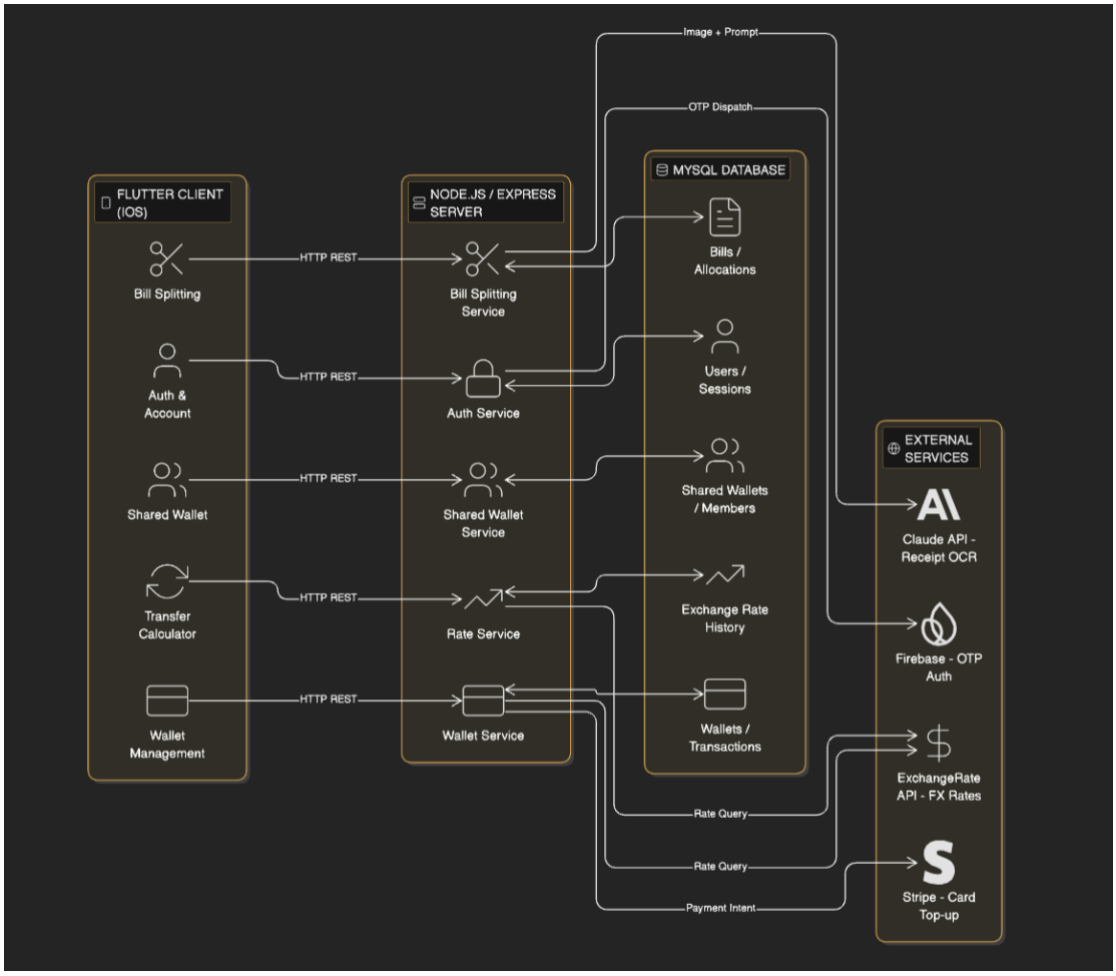

Figure 4.1.1.1 System Block Diagram

4.2 System Components Specifications

This part lists the technical details of each major software part of the system. There are four layers of components: the client layer, the server layer, the data layer, and the external services layer.

4.2.1 Client Layer — Flutter Application

The Flutter framework was used to make the client-side app, which works on iOS. Flutter uses a declarative UI model based on widgets, where the interface is set up as a tree of widgets that can be combined. The app uses stateful widgets and asynchronous state management to show real-time updates, like new shared wallet notifications and live balance changes, without having to reload the whole page.

The Flutter app only talks to the Node.js backend over HTTP using the http package. It uses the RESTful API described in Section 4.3.2. Using the flutter_secure_storage package, you can keep session authentication by keeping an encrypted session token in the device's local storage. This token is added to the top of all API requests that have been authenticated.

A dual-modality security layer enforces transaction-level authorisation for Transfer and Convert operations. This layer uses a user-defined numeric PIN or biometric authentication through Face ID. It is implemented using the local_auth package, which wraps the iOS LocalAuthentication framework. The app also uses the Google ML Kit liveness detection library to check accounts during the registration process and the qr_code_scanner package to process QR payments.

4.2.2 Server Layer — Node.js / Express Backend

The application logic layer is a single RESTful API server built with Node.js and the Express framework. The server has a number of HTTP endpoints that are grouped by functional module. Each one runs the appropriate business logic before reading from or writing to the MySQL database.

We chose Node.js because its event-driven, non-blocking I/O architecture lets the server handle multiple requests at the same time. This is especially important for

the shared wallet module, where multiple users can read and write to the same wallet record at the same time. The server runs as a single-process app on port 5001, and you can change its settings using environment variables.

The server starts up with a strict order for middleware: Express has registered the Stripe webhook endpoint (/api/stripe/webhook).raw() before the global express.json() middleware is used. This order is very important because Stripe's webhook signature verification needs access to the request body as it is. If you register express.json() first, it will parse and change the body before Stripe can check it, which will cause all webhook signature checks to fail. After that, CORS and static file serving middleware are used.

The server layer's business logic responsibilities are spread out over the following functional domains. The authentication module handles user registration, sending verification tokens via Nodemailer and Gmail SMTP, logging in with a session-based password comparison, managing PINs with bcrypt hashing, and resetting passwords through tokenised email links. The wallet module takes care of things like checking balances in nine different currencies, sending money from one person to another with user_currency_limits that let you set limits on how much you can send per transaction and per day, processing QR code payments, and converting currency using live exchange rates. The shared wallet module controls the entire lifecycle of a group wallet, from creation to inviting members, recording contributions and expenses, starting closure requests, collecting member votes, and calculating settlements. The bill splitting module takes care of sending receipt images to the Claude API, confirming item allocation, calculating taxes and service charges, and automatically deducting payments when the bill is paid. The exchange rate module uses node-cron to run a scheduled cron job that gets foreign exchange rates from the ExchangeRate API and saves them in the daily_currency_rates table. This cuts down on unnecessary external API calls for historical rate queries.

All communication between services and external APIs, like Stripe, ExchangeRate API, Claude API, and Firebase, happens on the server side. Environment variables managed by dotenv keep all API keys, database keys, and email

authentication information safe, so that sensitive settings aren't hard-coded into the source code or made available to the Flutter client.

4.2.3 Data Layer — MySQL Relational Database

A MySQL relational database is used to create the persistent data layer. The mysql2 library is used to access the database, and a connection pool is set up with a limit of ten concurrent connections. MySQL is chosen because it strongly supports referential integrity enforcement through foreign key constraints, ACID-compliant transaction processing that is necessary for financial operations, and it has a lot of experience with complex multi-table join queries that are needed for settlement computation and transaction history retrieval.

The initializeDatabase() function runs at server startup to set up the database schema. It runs CREATE TABLE IF NOT EXISTS statements for all tables and uses INFORMATION_SCHEMA to apply any column migrations that are still needed. Before adding missing columns with ALTER TABLE, COLUMNS checks. This method makes sure that the schema is the same in both development and deployment environments without needing a separate migration tool.

The schema has these tables in it. The users table holds account information and profile information, such as username, email, phone number, country, password hash, pin hash, face descriptor (JSON, reserved for a server-side face comparison subsystem; not populated by the account verification flow, which uses on-device ML Kit liveness detection without storing any facial data), is_verified, and verified_at. The user_currency_accounts table is the main wallet store. Each row represents one active currency sub-wallet for a user, which is identified by user_id and currency_code. The current balance is stored as a DECIMAL(15,2) field, and there is an is_default flag to show which currency is the user's main operating currency. The UNIQUE KEY constraint on (user_id, currency_code) makes sure that each user can only have one wallet for each currency.

The user_transactions table keeps track of all financial activities, such as deposits, withdrawals, transfers in and out, currency conversions, fees, and payments. Each row records balance_before and balance_after, creating a complete audit trail that doesn't depend on the current wallet balance. The stripe_payment_intent_id and

payment_processor columns connect deposits made through Stripe to the transaction records that go with them. The user_transfers table keeps track of peer-to-peer transfer events separately. It stores the from_user_id, to_user_id, exchange_rate, and converted_amount to support transfers between different currencies, along with a UNIQUE reference ID to make sure that the same transfer isn't counted twice.

The user_currency_limits table keeps track of configurable limits for each user and currency for per_payment_limit, daily_payment_limit, per_transfer_limit, and daily_transfer_limit. These limits are set to conservative values by default and are enforced by the Wallet Service before any outgoing transaction is processed.

The shared_wallets, shared_wallet_members, and shared_wallet_transactions tables all work together to model the group wallet subsystem. shared_wallets keeps track of the wallet's name, currency, balance, and status (pending, active, closing, or closed). shared_wallet_members keeps track of each member's role (owner or member) and whether or not they are participating. shared_wallet_close_requests and shared_wallet_close_votes use the consensus-based closure mechanism to keep track of the initiator's proposed distribution_mode and distribution_json, as well as each member's vote to approve or reject, which can include a reason for rejecting.

There are four tables that make up the bill splitting subsystem. split_bills keeps the bill header, which has the title, currency code, total amount, service charge percentage, tax percentage, status, and an optional receipt image field for AI-parsed receipts. split_bill_items keeps track of each line item, including its name, price, and quantity. split_bill_members keeps track of how much each person owes and whether they have paid. split_bill_item_selections records each member's item-level choices down to the individual quantities, which lets for accurate per-item allocation before the total amount due is calculated.

The daily_currency_rates table stores historical exchange rate snapshots for each date, and a UNIQUE KEY on (from_currency, to_currency, recorded_date) stops duplicate entries. The scheduled cron job in the Rate Service fills this table, and the Transfer Calculator uses it to make the seven-day historical trend chart without making extra calls to external APIs.

Three extra tables help with Stripe integration. stripe_customers links each user to a Stripe customer ID for each currency. The user_payment_methods table stores

tokenised card information, such as the card brand, last four digits, and expiration date, along with an is_default flag that is set by database triggers. stripe_transactions keeps a complete record of all Payment Intent lifecycle events that it gets from Stripe webhooks. This includes the status, amount, currency, and other information.

Three database triggers are set up to make sure that business rules are followed at the data layer. ensure_single_default_currency makes sure that a user can't have more than one active currency account that is marked as the default at the same time. update_default_payment_method_per_currency and insert_default_payment_method_per_currency make sure that only one payment method per user and currency can have the is_default flag. These functions run when a user_payment_methods record is updated or inserted, respectively.

The face_verify_logs table keeps track of every attempt to verify a face, including the computed similarity score, euclidean_distance, and boolean result. This makes it possible to keep track of account verification events that happen during registration through the ML Kit liveness detection flow.

4.2.4 External Services Layer

You can only use Stripe to add money to your card-based wallet. The Node.js backend makes a Stripe Payment Intent and sends the client secret back to the Flutter client. The Flutter client then uses the flutter_stripe package to show the Stripe payment sheet and confirm the payment. The backend updates the balance of the corresponding user_currency_accounts in MySQL and records the event in both user_transactions and stripe_transactions after the confirmation is successful.

ExchangeRate API gives you real-time and historical exchange rates for nine currencies: MYR, USD, SGD, JPY, HKD, THB, AUD, GBP, and CAD. The Node.js backend uses this API to get live conversion operations on demand and a scheduled cron job to save daily rate snapshots in the daily_currency_rates table for the Transfer Calculator module.

The Bill Splitting module uses Claude API (Anthropic) to read receipt images. The Flutter client takes or uploads an image and sends it to the Node.js backend. The backend sends the image to the Claude API with a structured prompt asking for the merchant's name, itemised descriptions, quantities, unit prices, and any taxes that apply.

The structured response is sent back to the Flutter client so that the user can confirm it before the bill is made.

During the registration process, Firebase Phone Authentication sends a one-time password to the user's phone number that they used to sign up. The Node.js backend sends and checks OTPs through Firebase, and then the user account is saved in MySQL. After you sign up, Firebase doesn't do any authentication. Instead, the app's own session system uses bcrypt credential comparison to handle all subsequent logins.

4.3 Database and API Design

In the context of this software project, this section shows how the two main design artefacts of the system are structured inside: the relational database schema, which defines the data structures and relationships that make all system operations possible, and the RESTful API endpoint design, which defines the communication contracts between the Flutter client and the Node.js backend. These artefacts make up the wiring of the software system, which is like the design of a circuit in a hardware system.

4.3.1 Database Schema Design

The third normal form (3NF) is used to make the database schema so that there is no duplicate data and referential integrity is enforced. When the server starts up, the schema is set up programmatically. There is also runtime migration logic that adds missing columns to existing tables without losing any data. This section talks about the main tables, their most important features, and how they relate to each other.

The users table is the main part of the schema. It has an id (primary key, auto-increment), a username (unique), an email (unique), a phone number, a country, a password hash (bcrypt), a pin hash (bcrypt), a face descriptor (JSON, nullable; as noted in Section 4.2.3, this field is not populated by the UC-04 account verification flow), an is_verified (boolean), a verified_at, a created_at, and an updated_at. To make it easier to look up usernames, email addresses, phone numbers, and countries during authentication and contact resolution, indexes are set up.

The user_currency_accounts table shows separate currency wallets. The main attributes are id, user_id (a foreign key that points to users), currency_code (VARCHAR 3), balance (DECIMAL 15,2), is_default (boolean), is_active (boolean),

payment_method_id, and payment_processor. A UNIQUE KEY on (user_id, currency_code) makes sure that each user can only have one wallet for each currency. When a new default is set, the database trigger "ensure_single_default_currency" runs on UPDATE to turn off the is_default flag on all other accounts for the same user.

The user_transactions table keeps track of all money-related events. There are a number of attributes, such as id, user_id, currency_code, transaction_type (ENUM: deposit, withdrawal, transfer_in, transfer_out, conversion, fee, payment), amount, fee_amount, balance_before, balance_after, description, reference_id, stripe_payment_intent_id, payment_processor, and created_at. The balance_before and balance_after fields make up a complete audit trail.

The user_transfers table keeps track of peer-to-peer transfers with the following fields: from_user_id, to_user_id, currency_code, amount, exchange_rate, converted_amount, converted_currency, transfer_fee, status (ENUM: pending, completed, failed, cancelled), description, reference_id (unique), created_at, and completed_at.

The user_currency_limits table sets limits on how much money each user can send and receive in each currency. It has DECIMAL(15,2) fields for per_payment_limit, daily_payment_limit, per_transfer_limit, and daily_transfer_limit, and a UNIQUE KEY on (user_id, currency_code).

The shared_wallets table has records of group wallets. Each record has a wallet_id (a unique string identifier), a name, a currency_code, a balance, a status (ENUM: pending, active, closing, closed), a created_by (a foreign key pointing to users), a created_at, and an updated_at. The shared_wallet_members table keeps track of who is a member by storing their wallet_id, user_id, role (ENUM: owner, member), status (ENUM: pending, active, left), joined_at, and a UNIQUE KEY on (wallet_id, user_id).

The shared_wallet_close_requests table keeps track of when a closure starts, including the wallet_id, who started it, the distribution_mode, the distribution_json (TEXT, serialised allocation plan), the status (ENUM: pending, completed, rejected, cancelled), and the timestamps. The shared_wallet_close_votes table keeps track of each member's vote for each request. It has a UNIQUE KEY on (request_id, user_id)

and stores the vote (ENUM: approve, reject), the time the vote was cast, the reason for the rejection, and the mode of rejection.

The shared_wallet_limits table makes sure that each wallet owner sets transaction limits for their wallet. It has DECIMAL (15, 2) fields for wallet_id (a foreign key that points to shared_wallets), currency_code, per_payment_limit, and daily_payment_limit. There is a UNIQUE KEY on (wallet_id, currency_code). When a user first accesses the site, default limits are set for each currency. The owner can change these limits at any time using a special API endpoint. The expense route calls a helper function called checkSharedWalletPayLimit() to check each transaction against the daily and per-payment limits before any money is taken out of the account.

The user_contacts table is what makes the contact management feature possible. It keeps track of the relationships between registered users. The most important fields are owner_id (a foreign key that points to the user who added the contact), contact_id (a foreign key that points to the user who was added), and an optional nickname field. A UNIQUE KEY on (owner_id, contact_id) stops people from adding the same contact more than once. To make group financial activities go more smoothly, contacts are listed in alphabetical order and are given priority in shared wallet member selection and bill splitting recipient lists.

The split_bills table has bill headers with bill_id (a unique string), created_by, title, currency_code, total_amount, service_charge_pct, tax_pct, status (ENUM: open, settled, cancelled), receipt_image (TEXT, base64, or URL for AI-parsed receipts), and timestamps. The split_bill_items table has a list of items with their names, prices, and quantities for each bill. The split_bill_members table keeps track of how much each person owes and whether they have paid (ENUM: pending, paid). The split_bill_item_selections table keeps track of each member's choices for each item, with a UNIQUE KEY on (bill_id, item_id, user_id).

The daily_currency_rates table stores daily snapshots of exchange rates with the following fields: from_currency, to_currency, rate (DECIMAL 15,6), and recorded_date (DATE). A UNIQUE KEY on (from_currency, to_currency, recorded_date) stops duplicate entries and helps the Transfer Calculator quickly find historical rates.

Stripe integration is supported by stripe_customers (which maps user_id to stripe_customer_id for each currency), user_payment_methods (which stores tokenised card details with card_brand, card_last4, expiry, and is_default for each currency, enforced by two triggers), and stripe_transactions (which logs Payment Intent lifecycle events like payment_intent_id, amount, currency, status, and metadata JSON).

The email_verification_tokens and password_reset_tokens tables help with the authentication flows. Both keep a hashed token, an expiration timestamp, an is_used flag, and a link to the user. email_verification_tokens also saves user_data as TEXT so that the registration payload stays the same between sending the OTP and making the account.

The face_verify_logs table keeps track of liveness verification events by storing the user_id, similarity, euclidean_distance, and boolean result for each attempt.

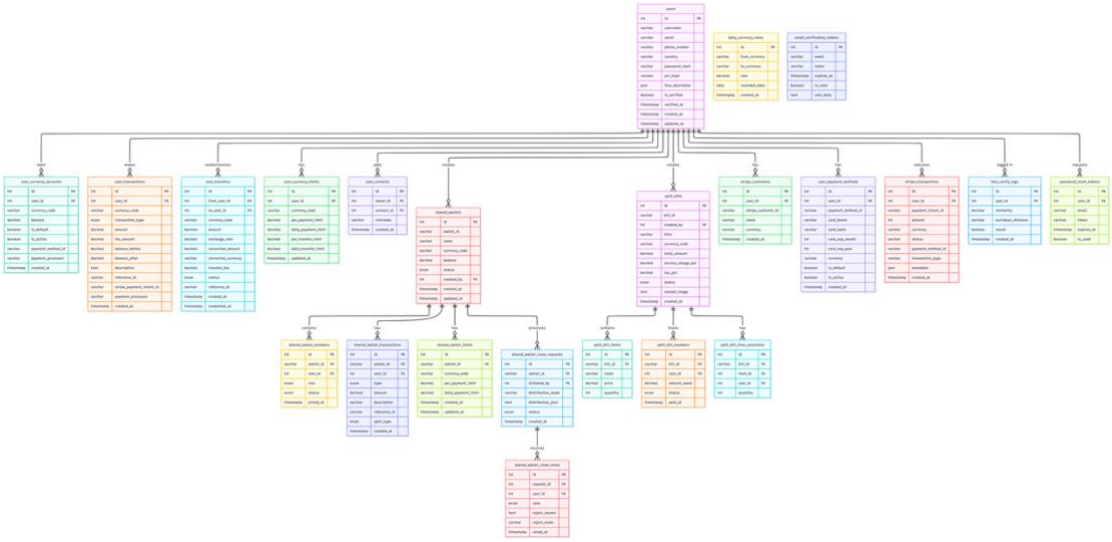

Figure 4.3.1.1 Entity Relationship Diagram

Figure 4.3.1.1 shows the database schema's entity relationship diagram, which shows the 23-table relational structure that was made in Third Normal Form (3NF). The users table is the main entity from which all functional modules come. These modules include multi-currency wallets, peer-to-peer transfers, shared wallets, bill splitting, Stripe payment integration, and caching exchange rates. Foreign key constraints and database triggers make sure that referential integrity and business rules are followed throughout the schema.

4.3.2 RESTful API Endpoint Design

The Node.js backend has RESTful HTTP endpoints that are grouped by functional module across several route handler files. All endpoints send back JSON responses and use standard HTTP status codes. The tables below show the main endpoints for each module.

Table 4.3.2.1 Authentication and User Management Endpoints

Method	Endpoint	Description
POST	/api/login-email	Gmail-based login
POST	/api/login	Username-based login
POST	/api/send-verification-email	Send email OTP for registration
GET	/api/verify-email/:token	Confirm email verification token
POST	/api/check-verification-status	Check email verification status
POST	/api/users	Create new user account
GET	/api/users	Retrieve all users
GET	/api/users/:userId	Retrieve single user profile
PUT	/api/users/:userId	Update user profile
DELETE	/api/users/:userId	Delete user account
POST	/api/check-phone-availability	Check phone number availability
POST	/api/check-user-data-availability	Check email, phone, username availability
POST	/api/user/verify	Mark account as face-verified
POST	/api/request-password-reset	Request password reset email
GET	/api/verify-reset-token/:token	Verify password reset token validity

POST	/api/reset-password	Reset password with token
GET	/reset-password/:token	Web interface for password reset

Table 4.3.2.2 Face Verification Endpoints

Method	**Endpoint**	**Description**
POST	/api/users/:userId/face-descriptor	Enrol face descriptor for account verification
POST	/api/users/:userId/verify-face	Verify face descriptor via cosine similarity
GET	/api/users/:userId/face-status	Check if face is enrolled and verified
DELETE	/api/users/:userId/face-descriptor	Remove face data and reset verification status

Table 4.3.2.3 Wallet and Currency Management Endpoints

Method	**Endpoint**	**Description**
GET	/api/users/:userId/currencies	Retrieve all active currency wallets
POST	/api/users/:userId/currencies	Create or reactivate a currency wallet
PUT	/api/users/:userId/currencies/:currencyCode	Update wallet balance or default status
DELETE	/api/users/:userId/currencies/:currencyCode	Close a currency wallet

POST	/api/users/:userId/currencies/:currencyCode/add	Deposit funds into a wallet
GET	/api/users/:userId/transactions	Retrieve transaction history
GET	/api/users/:userId/total-balance	Retrieve total balance summary across all wallets
GET	/api/users/:userId/limits	Retrieve per-currency transaction limits
PUT	/api/users/:userId/limits/:currency	Update transaction limits for a currency
POST	/api/users/:userId/expense	Record a QR payment expense
POST	/api/transfers	Execute a peer-to-peer fund transfer
GET	/api/search/users	Search users by email, phone, or username
POST	/api/conversions	Execute a currency conversion

Table 4.3.2.4 Exchange Rate Endpoints

Method	**Endpoint**	**Description**
GET	/api/rates/:baseCurrency	Retrieve real-time exchange rates for a base currency
POST	/api/convert	Convert a specified amount between two currencies

GET	/api/history/:from/:to	Retrieve seven-day historical rate data for a currency pair
GET	/api/country/:countryName/currency	Retrieve default currency for a registered country
POST	/api/calculate-total-balance	Calculate total balance in a target currency using latest rates
GET	/api/supported-currencies	Retrieve the list of all nine supported currencies
POST	/api/internal/save-daily-rates	Persist daily rate snapshot to daily_currency_rates table (cron-triggered)

Table 4.3.2.5 Stripe Payment Endpoints

Method	**Endpoint**	**Description**
POST	/api/stripe/webhook	Receive and verify Stripe Payment Intent lifecycle events
POST	/api/stripe/create-setup-intent	Create a Setup Intent for card registration
POST	/api/stripe/confirm-setup-intent	Confirm Setup Intent and save tokenised card
GET	/api/stripe/payment-methods/:userId/:currency	Retrieve saved payment methods for a currency
POST	/api/stripe/create-payment-intent	Create and confirm a Payment Intent for wallet top-up

Table 4.3.2.6 Shared Wallet Endpoints

Method	Endpoint	Description
POST	/api/shared-wallets	Create a new shared wallet
GET	/api/shared-wallets/user/:userId	Retrieve all shared wallets for a user
GET	/api/shared-wallets/invitations/:userId	Retrieve pending wallet invitations
GET	/api/shared-wallets/user/:userId/recently-closed	Retrieve recently closed wallets
GET	/api/shared-wallets/:walletId	Retrieve wallet details, members, and transactions
POST	/api/shared-wallets/:walletId/respond	Accept or decline a wallet invitation
POST	/api/shared-wallets/:walletId/invite	Invite a new member to a wallet
POST	/api/shared-wallets/:walletId/topup	Top up a shared wallet from personal balance
POST	/api/shared-wallets/:walletId/expense	Record an expense against a shared wallet
POST	/api/shared-wallets/:walletId/leave	Leave a shared wallet
GET	/api/shared-wallets/:walletId/all-transactions	Retrieve full transaction history for a wallet
GET	/api/shared-wallets/:walletId/limits	Retrieve per-payment and daily payment limits

PUT	/api/shared-wallets/:walletId/limits	Update payment limits (owner only)
POST	/api/shared-wallets/:walletId/close-request	Initiate a wallet closure request with distribution plan
GET	/api/shared-wallets/:walletId/close-request	Retrieve current closure request and vote status
POST	/api/shared-wallets/:walletId/close-vote	Submit a member vote on the closure request
POST	/api/shared-wallets/:walletId/close-cancel	Cancel an active closure request (owner only)
GET	/api/shared-wallets/:walletId/smart-refund	Preview smart distribution based on contribution history

Table 4.3.2.7 Bill Splitting Endpoints

Method	**Endpoint**	**Description**
POST	/api/split-bills	Create a new bill manually or from parsed receipt data
POST	/api/split-bills/scan-receipt	Upload and parse a receipt image via Claude API
GET	/api/split-bills/user/:userId	Retrieve all bills for a user
GET	/api/split-bills/:billId	Retrieve full bill details including items, members, and selections
POST	/api/split-bills/:billId/invite	Invite a user to a bill
POST	/api/split-bills/:billId/select-items	Submit item selections and compute amount owed

GET	/api/split-bills/:billId/payment-options/:userId	Retrieve available payment sources for a bill
POST	/api/split-bills/:billId/pay	Confirm payment of a participant's computed share
POST	/api/split-bills/:billId/cancel	Cancel a bill and auto-refund paid members

Table 4.3.2.8 Contact Management Endpoints

Method	**Endpoint**	**Description**
GET	/api/contacts/:userId	Retrieve all contacts alphabetically
POST	/api/contacts/:userId/add	Add a registered user as a contact
DELETE	/api/contacts/:userId/remove/:contactId	Remove a contact
PATCH	/api/contacts/:userId/nickname/:contactId	Update a contact's nickname
GET	/api/contacts/:userId/recent	Retrieve recent transfer recipients
GET	/api/contacts/:userId/check/:contactId	Check if a user is already a contact

Table 4.3.2.9 QR Code Endpoints

Method	**Endpoint**	**Description**
POST	/api/qr/pay-token	Generate a dynamic QR payment token (35-second TTL)
POST	/api/qr/validate-token	validate a QR payment token and retrieve the associated user ID

CHAPTER 5 SYSTEM IMPLEMENTATION

5.1 Hardware Setup

Table 5.1.1 Specifications of laptop

Component	Specifications
Model	MacBook Air
Processor	Apple M3 Chip with 8-core CPU
Operating System	macOS
Graphic	10-core GPU
Memory	16GB Unified Memory
Storage	512GB SSD

Table 5.1.2 Specifications of mobile device

Component	Specifications
Model	iPhone 14 Pro
Processor	A16 Bionic
Operating System	ios18.1.1
Graphic	Apple GPU (5-core)
Memory	6GB RAM
Storage	256GB SSD

5.2 Software Setup

To use the proposed application, a number of software tools and frameworks had to be set up and installed. These included the mobile frontend, backend server, database, and connections to third-party services.

Flutter (Dart) was used to build the mobile client because it has a cross-platform widget-based rendering model and can work with iOS natively. We used the official Flutter SDK to install Flutter, and we used Visual Studio Code as the main integrated development environment (IDE) to manage the project. The Flutter and Dart extensions helped with syntax highlighting, hot reloading, and debugging on devices.

We used Node.js with the Express framework to build the backend application logic layer. This gave us a lightweight, event-driven RESTful API server. Using the official installer, we set up Node.js, and we used npm to keep track of the project's dependencies. During development, the backend server ran on the MacBook Air and was accessed by the Flutter client over the local network.

We used MySQL as the relational database management system, which was set up on the development machine. MySQL Workbench was used as a GUI client and direct terminal access was used to create database schemas and check queries.

During the user registration process, Firebase was added to handle phone authentication by sending an OTP to the user's registered phone number. The Firebase console was used to start the Firebase project, register the iOS app, and add the GoogleService-Info.Added a plist configuration file to the Flutter project.

The google_mlkit_face_detection Flutter package added Google ML Kit to make it possible to check for liveness for account-level verification. The package works completely on the device, using the front camera of the device. There is no need for a server to be involved in the detection.

The Stripe API (in test mode) was added so that people could add money to their wallets with cards. The backend had the Stripe Node.js SDK installed to make Payment Intents and handle webhook events.

The backend now uses the Claude API (Anthropic) to help the bill splitting module read receipt images. The Node.js server used the axios library to make HTTP requests directly to the Claude API endpoint.

The ExchangeRate API was added so that you could get real-time and historical foreign exchange rates. This made it possible to use the Transfer Calculator module to convert currencies in real time and see a seven-day historical trend chart. API calls were made on the server side, and the results were stored in the daily_currency_rates table to cut down on unnecessary requests from outside sources.

5.3 Setting and Configuration

5.3.1 Environment Variable Configuration (.env)

As shown in Figure 5.3.1.1, all sensitive credentials and environment-specific parameters were stored in a .env file in the root directory of the Node.js project. The dotenv package loads the file when the server starts up, and .gitignore keeps it out of version control so that no credentials are visible in the source repository.

The server setup tells the application port (5001) and the runtime environment (development). The database block gives the mysql2/promise connection pool the MySQL connection information it needs. The email block gives Nodemailer the Gmail account information it needs to send OTP verification emails and password reset links. The @anthropic-ai/sdk client uses the ANTHROPIC_API_KEY on the server side to verify requests to the Claude API for parsing receipt images with AI.

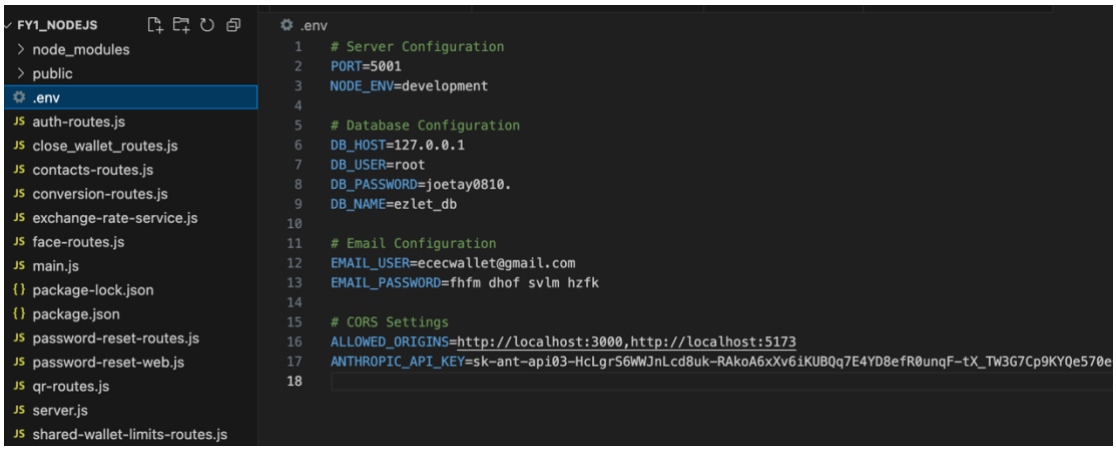

Figure 5.3.1.1 .env environment configuration file

5.3.2 MySQL Database Creation and Connection

The server's initializeDatabase() function sets up the database when the server starts up.js. The function first connects to the MySQL instance temporarily using the credentials in .env and then creates all 21 tables in the order they depend on each other. mysql2/promise keeps a connection pool of up to ten concurrent connections. It also has waitForConnections turned on, which lets requests queue up when the load is high, as shown in Figure 5.3.2.1. Using INFORMATION_SCHEMA, runtime migration logic looks for certain columns.COLUMNS checks for missing columns and applies

the ALTER TABLE statement to add them, which lets the schema change without losing any data. Figure 5.3.2 shows the database structure that was made in MySQL Workbench, which shows that all 23 tables were made successfully.

```
// Database connection configuration
const dbConfig = {
  host: process.env.DB_HOST || '127.0.0.1',
  user: process.env.DB_USER || 'root',
  password: process.env.DB_PASSWORD || 'joetay0810.',
  database: process.env.DB_NAME || 'ezlet_db',
  waitForConnections: true,
  connectionLimit: 10,
  queueLimit: 0
};

const pool = mysql.createPool(dbConfig);
```

Figure 5.3.2.1 MySQL connection pool configuration in server.js

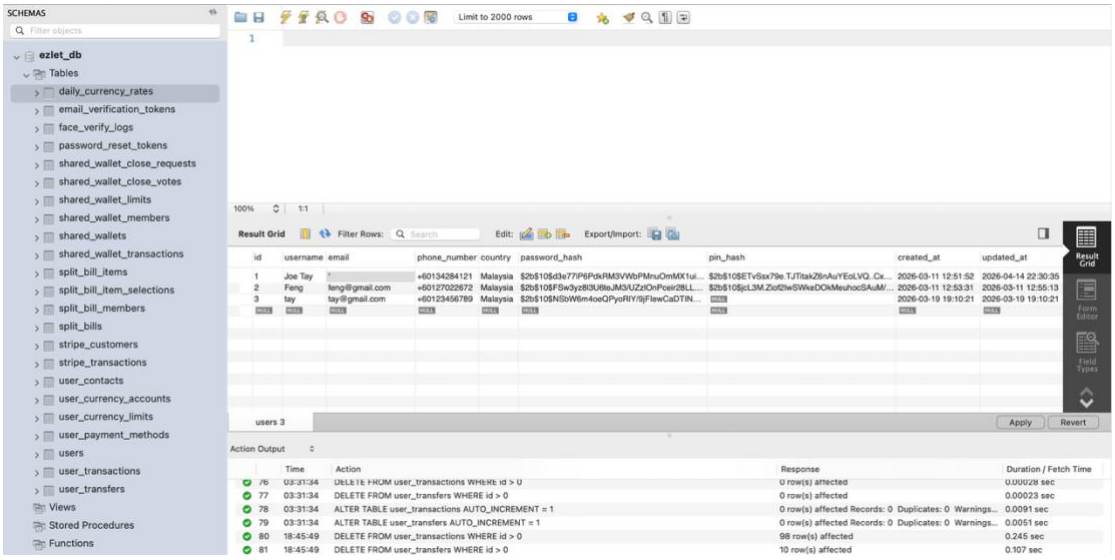

Figure 5.3.2.2 MySQL Workbench view

5.3.3 Flutter Dependency Installation (pubspec.yaml)

The pubspec.yaml file listed all of the Flutter package dependencies, and flutter pub get installed them. Key packages included http for RESTful API communication

with the Node.js backend; local_auth and local_auth_darwin for iOS native Face ID integration; google_mlkit_face_detection and camera for liveness verification via the front camera; qr_flutter and mobile_scanner for QR code generation and scanning; image_picker and file_picker for receipt image capture and file selection; flutter_secure_storage for encrypted local storage of the session token; firebase_core and firebase_auth for Firebase Phone Authentication; flutter_stripe for Stripe payment integration; flutter_local_notifications for in-app notification dispatch; and provider for application-level state management.

5.3.4 Firebase Authentication Set Up

1. Create a Firebase project.
    i. Go to the Firebase Console.
    ii. Click “Create a new Firebase project”.

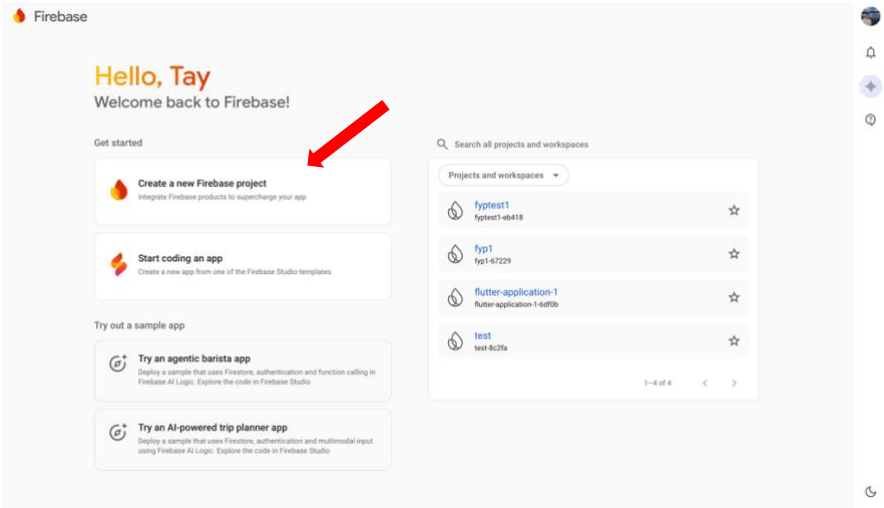

Figure 5.3.4.1 Create Firebase project

    iii. Follow the instructions to create a project in the Firebase Console.

2. Add Firebase to Flutter app.
    i. In firebase console, click on the project.

    ii. Click the iOS icon to add iOS app to the project.

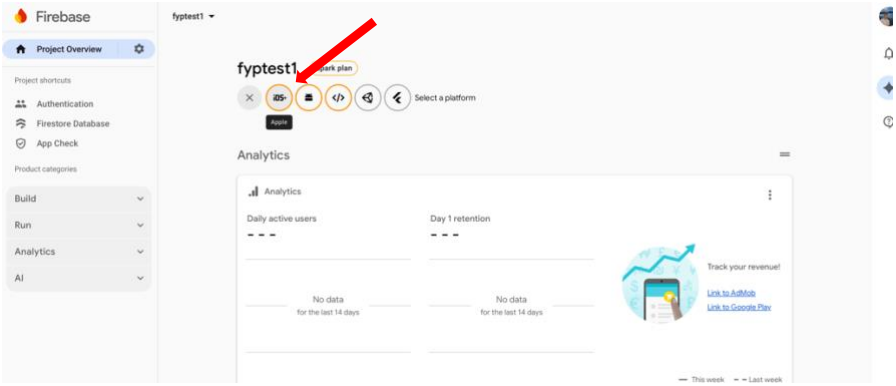

Figure 5.3.4.2 Add iOS app

iii. Enter Apple bundle ID which can be found in ios/Runner.xcodeproj/project.pbxproj file.

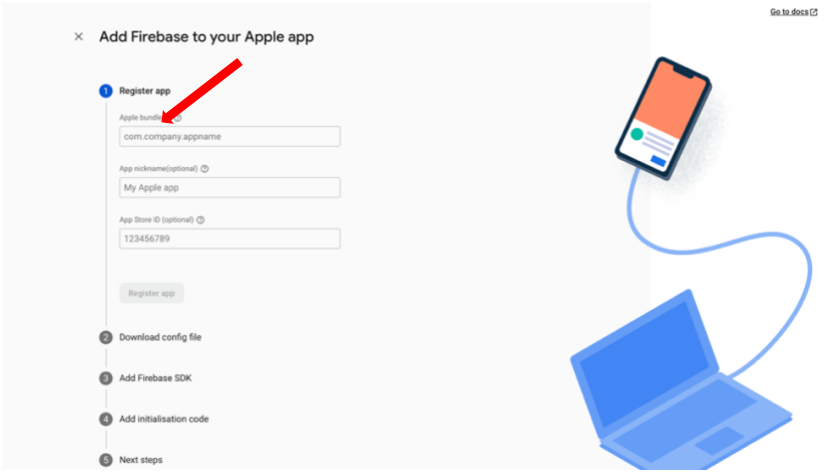

Figure 5.3.4.3 Input the Apple bundle ID

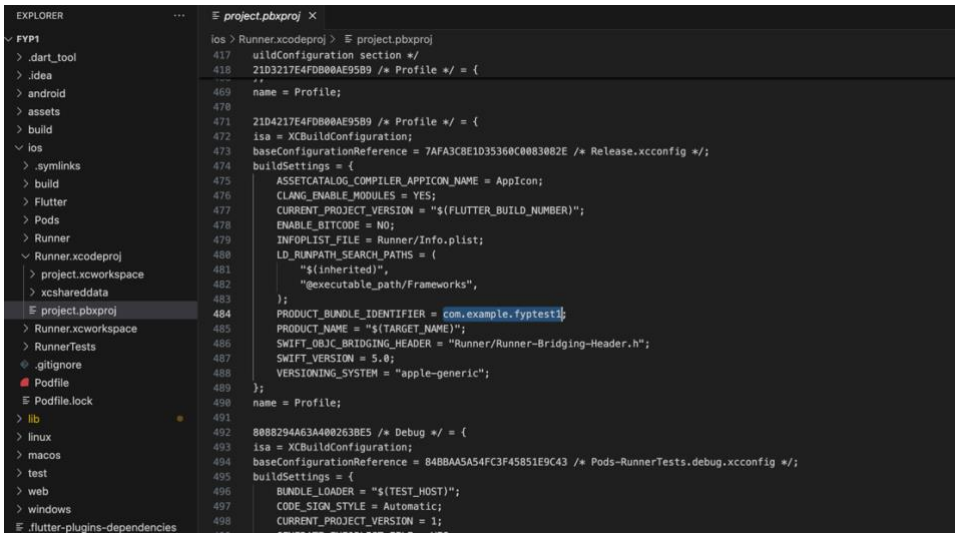

Figure 5.3.4.4 Get the ID in ios/Runner.xcodeproj/project.pbxproj file

iv. Download the GoogleService-Info.plist file and move it into Flutter project's ios/Runner/ directory.

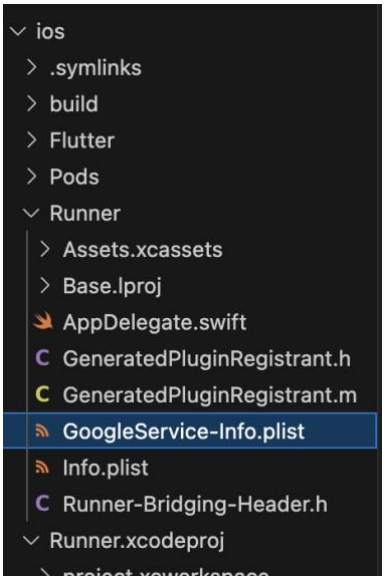

Figure 5.3.4.5 Paste the GoogleService-Info.plist in ios/Runner/ directory

3. Configure Flutter project.
4. Install Firebase packages. In Visual Studio code using Dart: Add Dependencies to add and after that it will show at pubspec.yaml file.

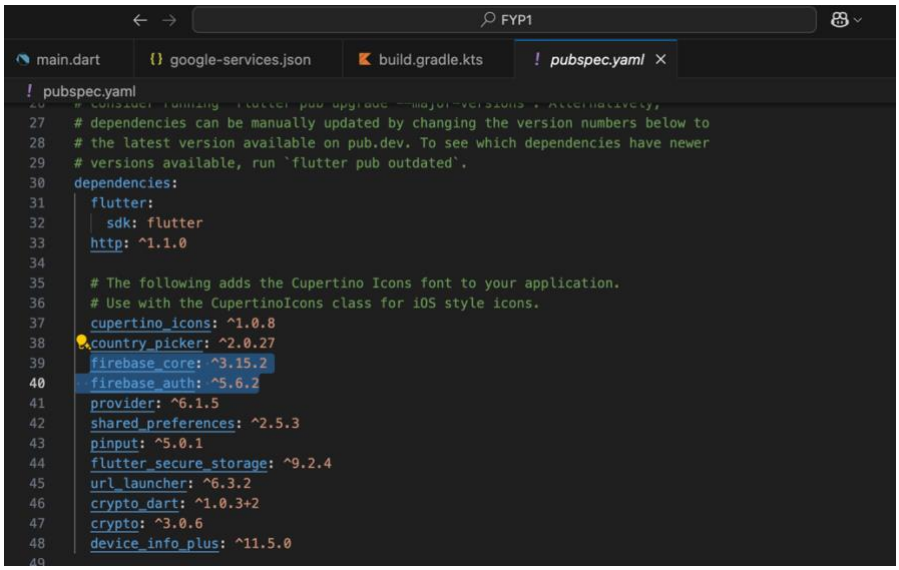

Figure 5.3.4.6 After successful adding the firebase packages in pubspec.yaml

5. Initialize the Firebase in main.dart.

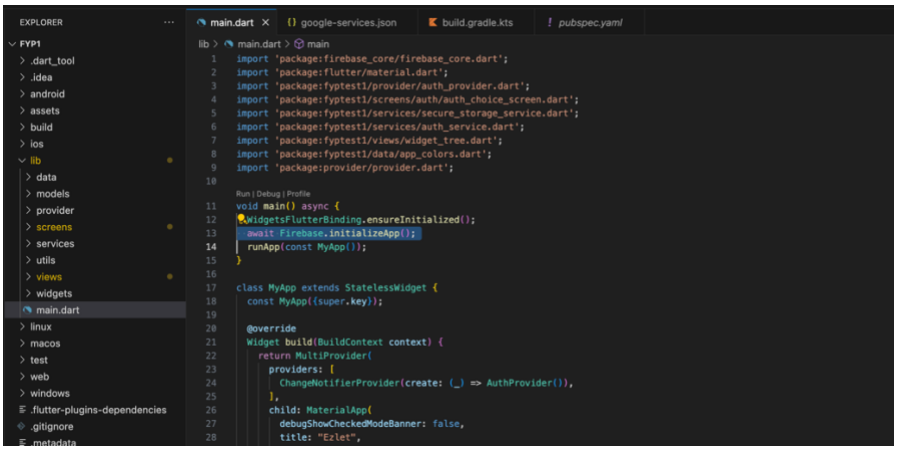

Figure 5.3.4.7 Screenshot of Firebase set up

6. Set up Firebase Authentication.
   i. In the Firebase Console, go to Build section and click “Authentication”.
   ii. Click “Get started”.
   iii. Choose the sign-in methods(Phone) that want to enable.
   iv. Create the test phone number and the modify OTP number.

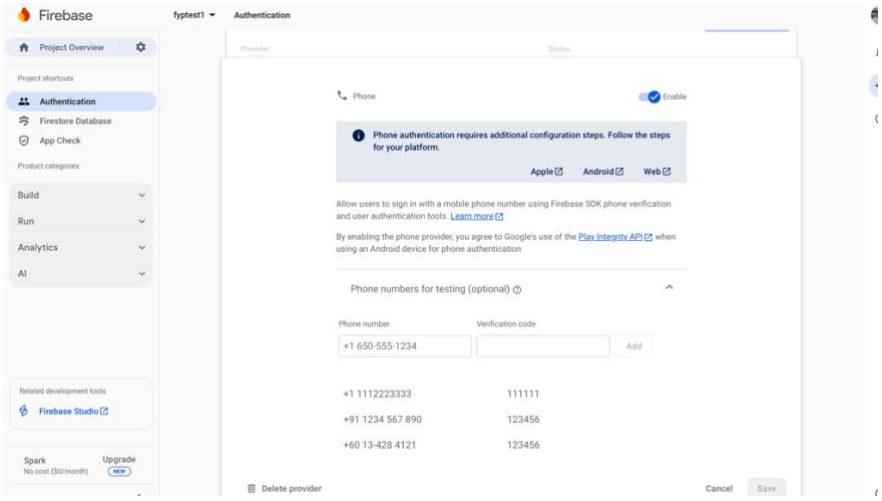

Figure 5.3.4.8 Screenshot of created tested account in Firebase Console

5.3.5 API Key

The Stripe API is in test mode, and there is a separate Stripe test account for each of the nine supported currencies: MYR, USD, SGD, JPY, HKD, THB, AUD, GBP, and CAD. Stripe-routes sets each account's secret key, publishable key, and webhook signing secret as constants.js. For each currency, a separate Stripe instance is set up using the appropriate secret key. At runtime, the getStripeConfig() helper function finds the right instance based on the requested currency. To start a top-up, the Flutter client asks the backend for a Setup Intent. The backend then sends the client_secret, ephemeral_key, and publishable_key back to the flutter_stripe package so that the card can be registered. Payment Intent creation on the backend handles subsequent payments, and the transaction result is sent asynchronously through a webhook event (payment_intent.succeeded). The webhook handler goes through all nine registered accounts to find the right signing secret for each incoming event. This makes sure that one endpoint works for all currencies. After the verification is successful, the handler adds money to the user's wallet and records the transaction in user_transactions. The Stripe Dashboard in Figure 5.3.5.1 shows that the test mode is set up correctly.

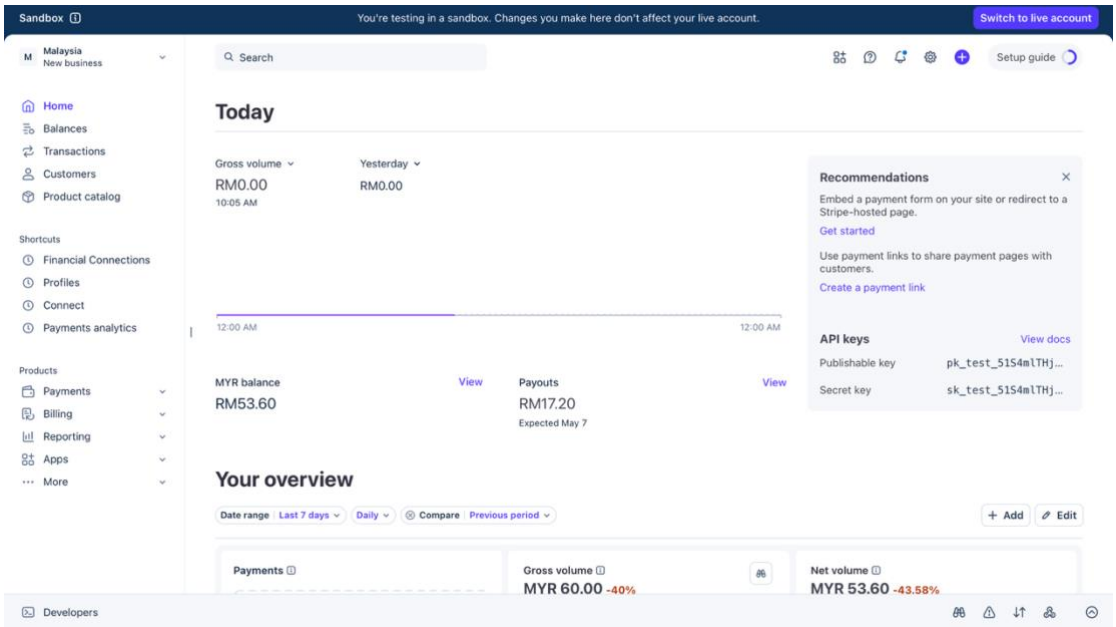

Figure 5.3.5.1 Stripe Dashboard showing test mode configuration

The .env file has the Claude API key stored as ANTHROPIC_API_KEY. The @anthropic-ai/sdk client can get to it on the server side through process.env.ANTHROPIC_API_KEY. The Claude API integration uses claude-opus-

4-5 as the model and a structured prompt template that tells the model to return receipt parsing results as a strictly formatted JSON object that includes the merchant name, line items, quantities, prices, and tax percentage values. JSON.parse() parses the response on the server side. If the output is not in the right format, a try-catch block sends a structured error response back to the client.

You can find the ExchangeRate API key and base URL (https://v6.exchangerate-api.com/v6) in the same file, exchange-rate-service.js. We only get exchange rates when we need them and store them in memory for 30 minutes to cut down on unnecessary calls to external APIs. A scheduled internal endpoint sends daily rate snapshots to the daily_currency_rates table, which supports the seven-day historical trend chart in the Transfer Calculator module. The ExchangeRate API dashboard in Figure 5.3.5.2 shows that the API key is set up correctly.

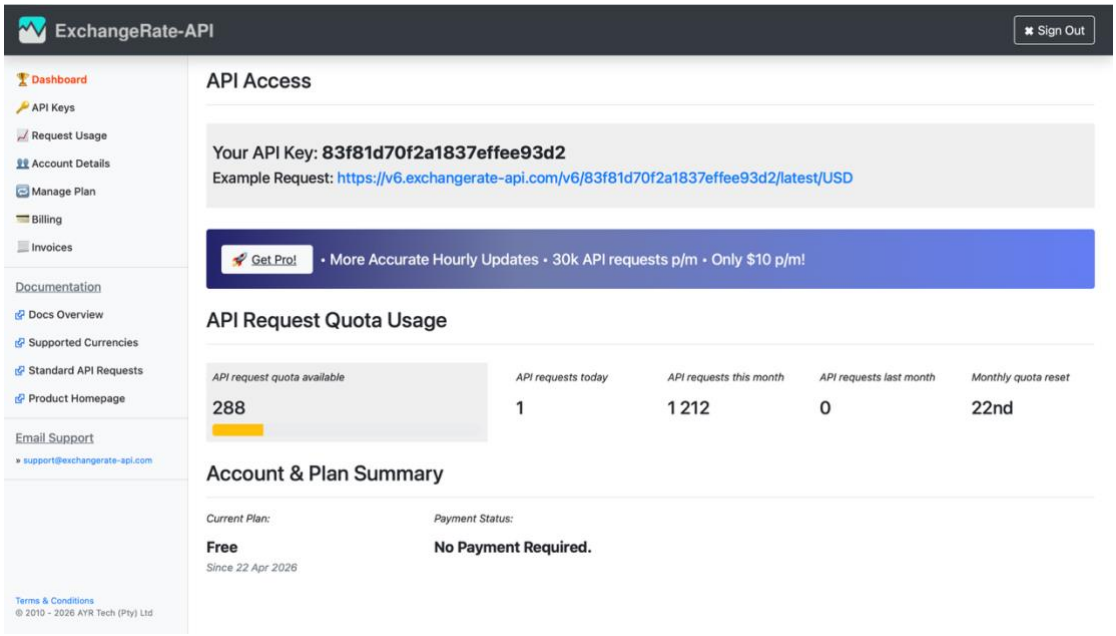

Figure 5.3.5.2 ExchangeRate API Dashboard

5.4 System Operation

5.4.1 App launch and Authentication

On staring the first time you will see a loading screen configuring up your system. In this process, it automatically checks whether the user is still in a session of his previous login or not. If the returned session is valid, then this user will be taken directly to the main dashboard with-out them having to log-in again. It saves a few steps for returning users. However, if the system does not find a valid session, the user is sent to the authentication page where they can choose between two options an existing user logs in and a new user signs up. The login process allows users to safely enter their credentials in order to regain access to their accounts. This is what we call the sign-up process which consists of potential new users who create an account and are then able to do every task that the app has available.

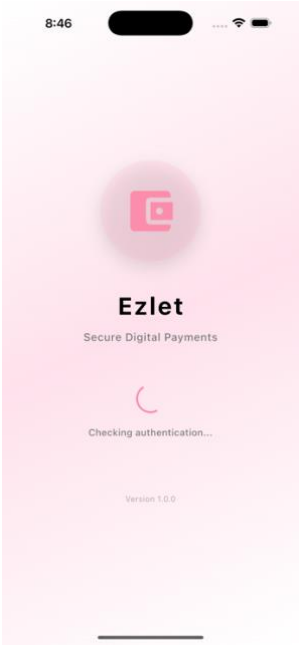

Figure 5.4.1.1 Initialize page

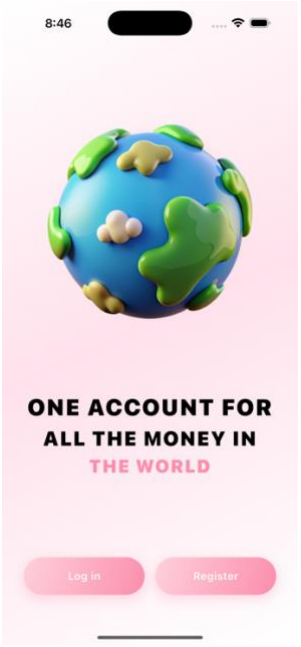

Figure 5.4.1.2 Authentication page

5.4.2 Register

The registration process starts when a user clicks on the Register button present at the authentication page. First, it takes me to a page where I could confirm my email address. In this step we make sure given email is not already used, and it belongs to user. As soon as the user sends the form, a verification link goes to his provided email address. The user needs to click on this link to proceed.

Once the email has been validated, the system proceeds to mobile verification. At that juncture, the system assigns a country or region code to the phone number automatically. Malaysia is the default choice. Users can change this if their number is from a different area. This then prompts the user for their mobile number which will then be checked to see if it has been registered. Once you try this phone not yet registered, an OTP is sent on the phone. On the right page, the user inputs OTP and can also resend code if required.

Once the mobile number has been confirmed, the system explains how to set their account security up. That includes creating a strong password and then a PIN which will come in handy later, when you need to authorize payments or even incoming money.

The final part of registration is configuring your user profile. A form appears asking for details such as your name, country, date of birth and phone number. The previously verified email address is automatically filled in and cannot be changed as it has been already confirmed. The system creates the new account after you complete and submit the profile form. Then it shows you a successful registration welcome page. After that, the user clicks on "Get Started," and they are navigated to the main dashboard where all the features of the app are available.

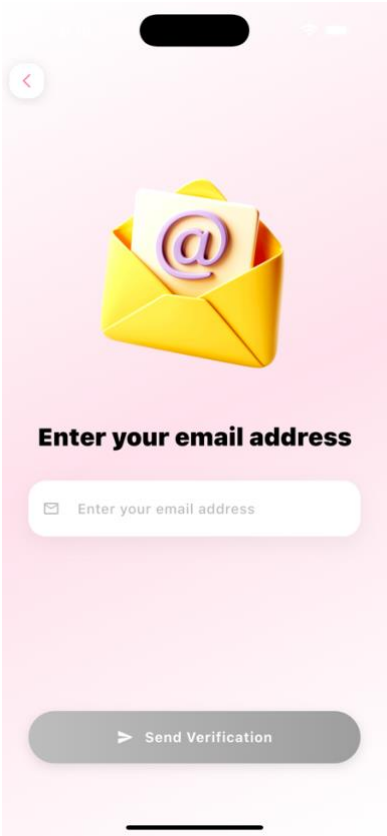

Figure 5.4.2.1 Email verification page

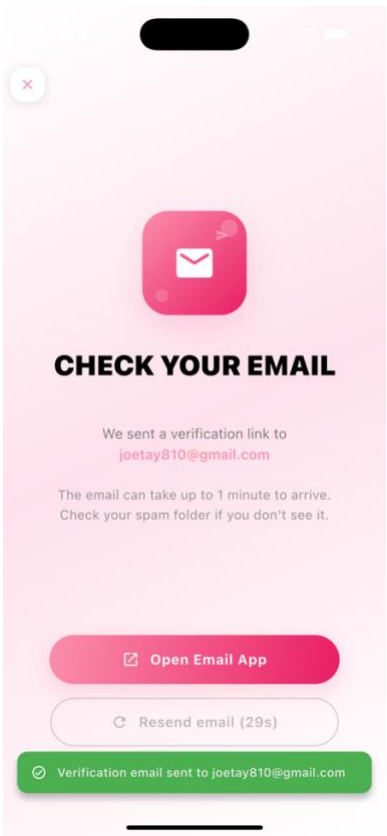

Figure 5.4.2.2 Send the verification page

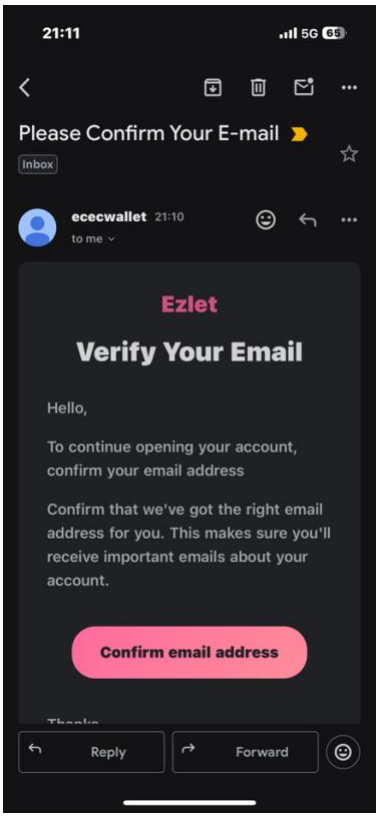

Figure 5.4.2.3 Verification link that send

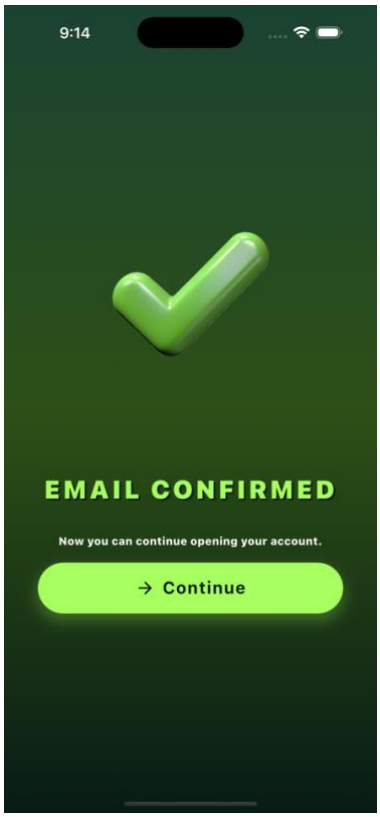

Figure 5.4.2.4 Email confirmed page

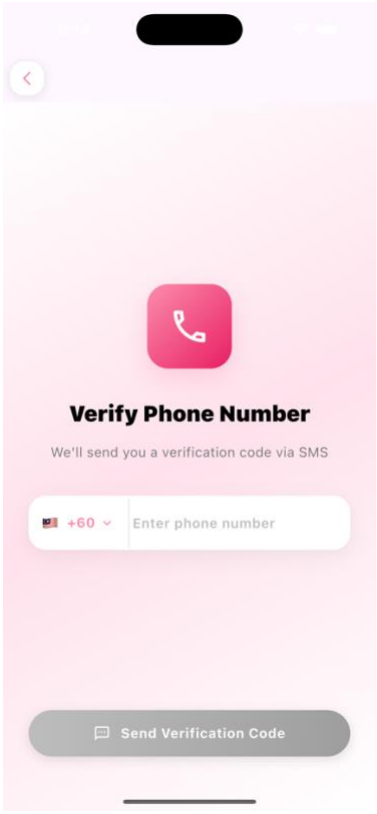

Figure 5.4.2.5 Phone number verification page

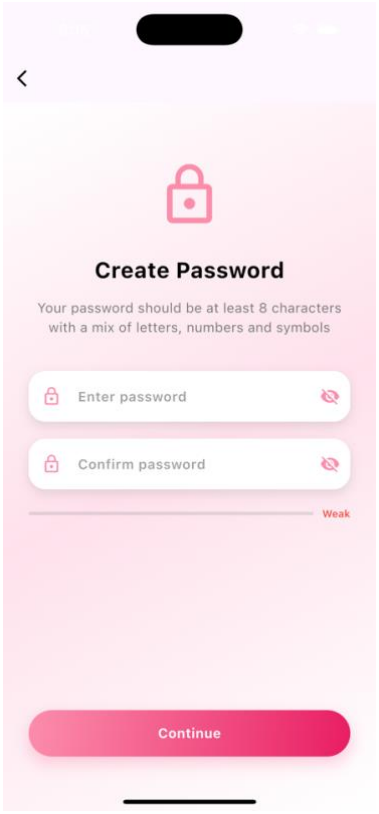

Figure 5.4.2.6 Create password page

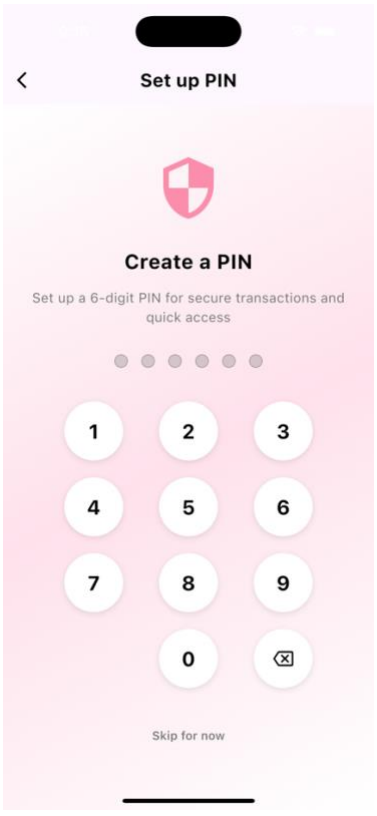

Figure 5.4.2.7 Create PIN page

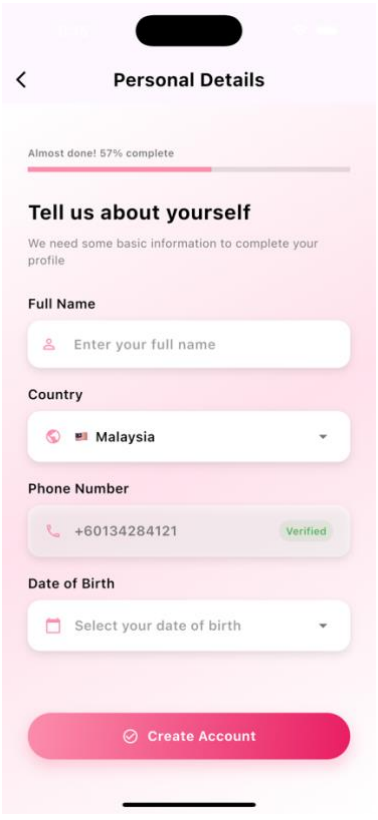

Figure 5.4.2.8 Fill-in personal details page

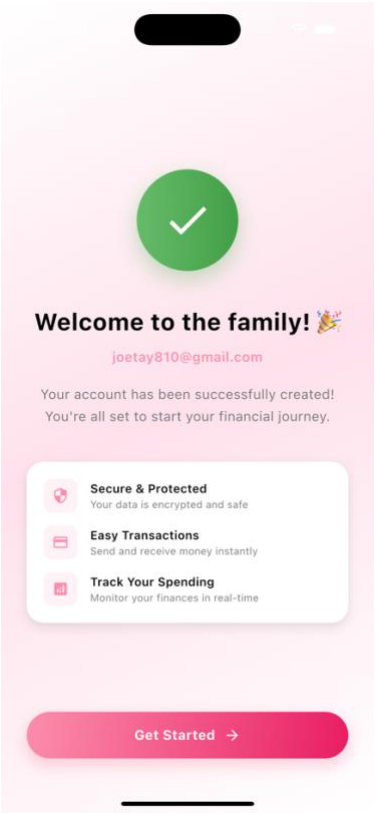

Figure 5.4.2.9 Welcome page

5.4.3 Login

The user sees the authentication page and clicks on "Login" and the system navigates them to the login page. Here, the user must input their registered email address and password prior to selecting the confirmation button. If the credentials are correct, then the system allows user in and redirects him directly to main dashboard.

The login page also offers registration for registered users who may not yet have an account. Below the login form is a "Sign Up" button that quickly gets new users to registration.

In case the users forget their credentials, the login page also has a link that reads Forgot Password to reset user info. If the user clicks this option, they will be directed to an email verification page where they need to input their registered email. After that, the system will send a link to reset your password to the email address you provided. User clicks at the link, user directs on page and here he can work with his password – reset found and write new. Once the reset is completed, the user shall login with the new credentials and hit to main dashboard.

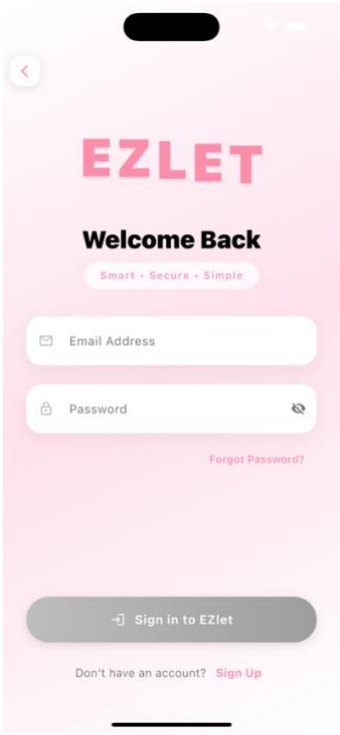

Figure 5.4.3.1 Login page

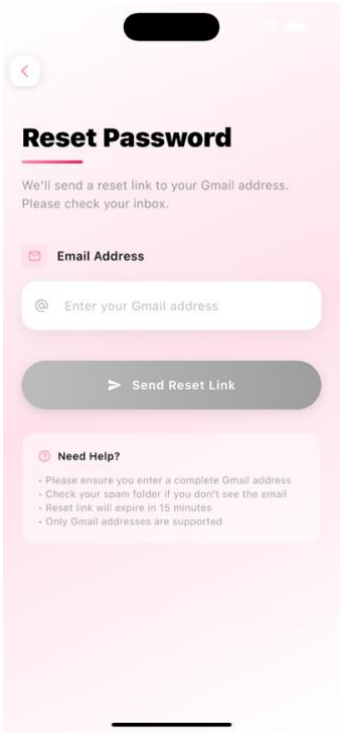

Figure 5.4.3.2 Send reset link page

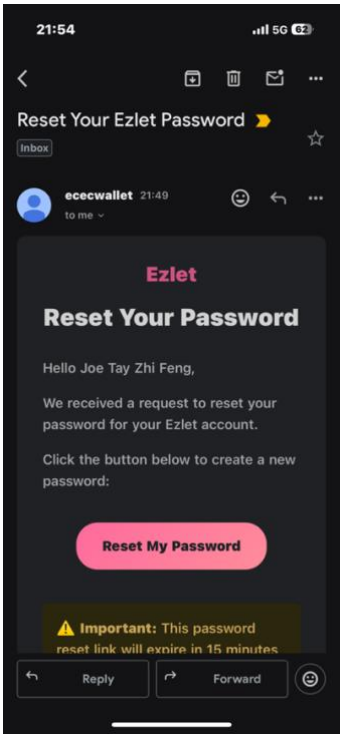

Figure 5.4.3.3 Reset link that send

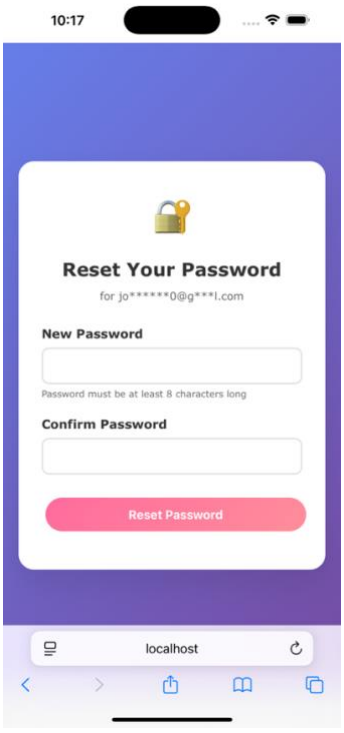

Figure 5.4.3.4 Reset password page

5.4.4 Home

The Home Page is the main dashboard of the app. It gives users a quick look at their account and easy access to important features. There is a button in the top left corner that shows the first letter of the username you used to sign up. This button takes you directly to the Profile Page. Next to the Total Balance section, a shield icon shows whether the account has been verified, which means it belongs to a real user. The system then shows the user their total account balance, which is based on the main currency that was set when they signed up with their phone number. This total balance is the result of adding up all the available balances in different currencies and converting them into the main currency. Because exchange rates change every day, the amount changes as well.

Next to the total balance, there is a statistics icon button that takes users to the Cash Flow Page for a more in-depth look at their finances. Users can see their own currency wallets below this section. The first wallet is for the main currency, and the other wallets are ones that the user has made. When you first sign up for an account, it only comes with the main currency wallet. If you need to, you have to add other currencies yourself.

The Transaction section shows the user's transaction history further down. The Home Page only shows the three most recent transactions for quick reference. To see the full list, click on "See All." There is also a Transfer Calculator on the page that lets people figure out how much money they need to send based on the most recent exchange rates. The system automatically calculates the converted value when you enter an amount and choose the source and target currencies. In addition, there is a line chart that shows how exchange rates have changed over the last seven days. This chart changes automatically based on the currency pair you choose.

A navigation bar (navbar) at the bottom of the screen lets you get to the main parts of the app, like Home, Shared Wallet, Split Bill, and Recipient. Each one takes you to its own page. The Scan icon in the middle of the navigation bar is a quick access button that takes users right to the Scan for Pay page, making transactions easy.

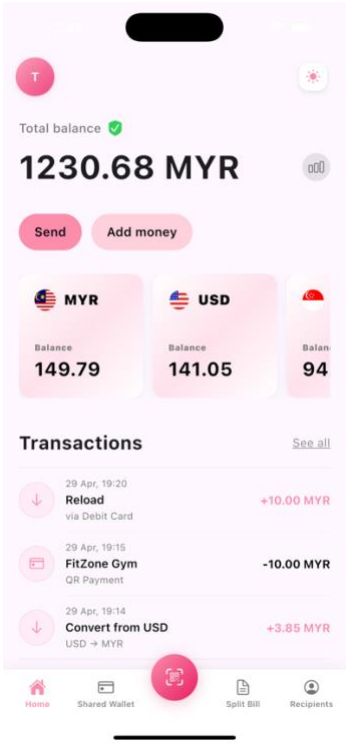

Figure 5.4.4.1 Home page

Figure 5.4.4.2 Transfer calculator in Home Page

5.4.5 Currency Wallet

This is where you can see all the wallets that were made for specific currencies. Users can add a new currency by clicking on the "Add Another Currency" button at the bottom of the list (Figure 5.4.5.1). When the user makes a choice, a list of currencies that haven't been added yet will show up for them to choose from (Figure 5.4.5.2). After you choose a currency, the system will ask you to enter your PIN to make sure it's safe. If the PIN is correct, the new currency wallet will be added (Figure 5.4.5.3). Figure 5.4.5.4 shows that the new wallet will be at the bottom of the list. Users can see the national flag, the name of the currency, and the current balance before they go into the wallet.

When a user clicks on a wallet, they are taken to the detail page for that currency wallet. At the bottom of this page is a transaction section that shows all of the transactions that have to do with that currency. The top part shows the current balance (amount). If there aren't any transactions, a message saying "No transaction yet" will show up (Figure 5.4.5.5). Users can also use three main features: Add, Convert, and Send. The Add function lets users add money to the chosen currency, the Convert function lets users change the currency into another currency, and the Send function lets users send money to other users.

Also, as seen in Figure 5.4.5.6, each currency wallet (except for the main currency) has a delete icon in the top right corner. This option lets users close the currency wallet when they choose it. If the wallet balance is zero, the wallet will be closed right away after it is checked. However, if there is still a balance, the system will let the user know that the remaining amount will be changed into the main currency at the current exchange rate before the account is closed. The app charges a 2.5% conversion fee as part of this process. There will be a summary (Figure 5.4.5.7) that shows the total amount to be received and the fees that will be charged. The wallet will be successfully closed, the converted amount will be added to the main currency, and the currency wallet will no longer be on the Home Page list if the user confirms the action and passes the verification process. The currency that was taken out will then show up again in the "Add Another Currency" option, where it can be added again later.

Figure 5.4.5.8 shows the main currency wallet without a delete icon in the top right corner. This is because it cannot be removed from the account.

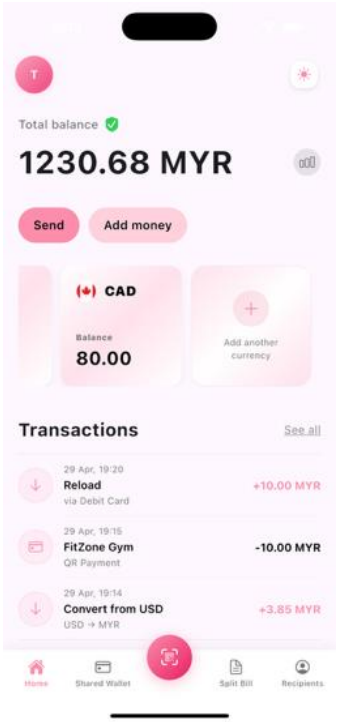

Figure 5.4.5.1 Add Another Currency

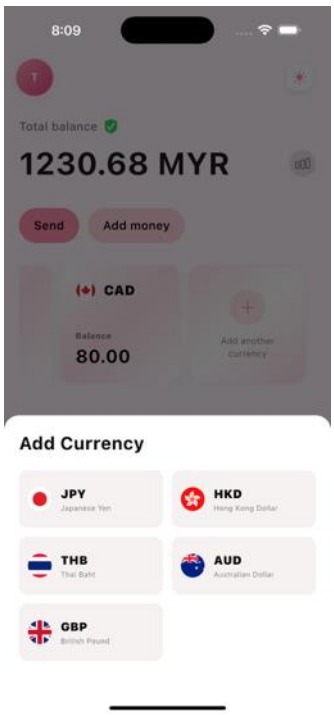

Figure 5.4.5.2 Currency List

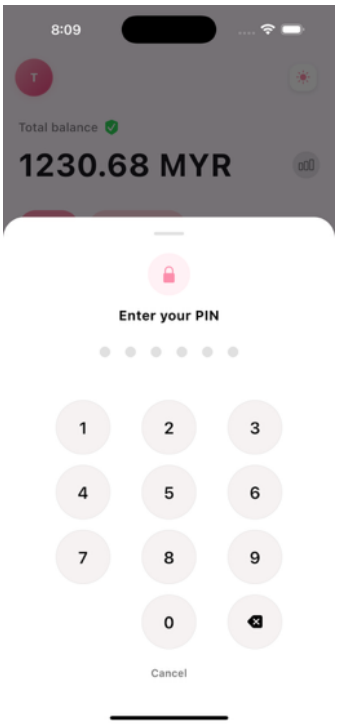

Figure 5.4.5.3 PIN Verify

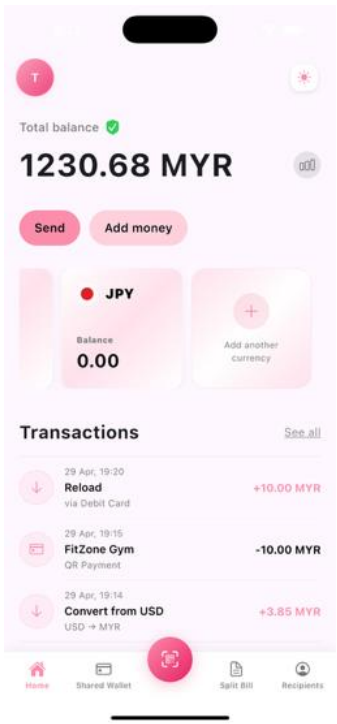

Figure 5.4.5.4 Added Currency Wallet

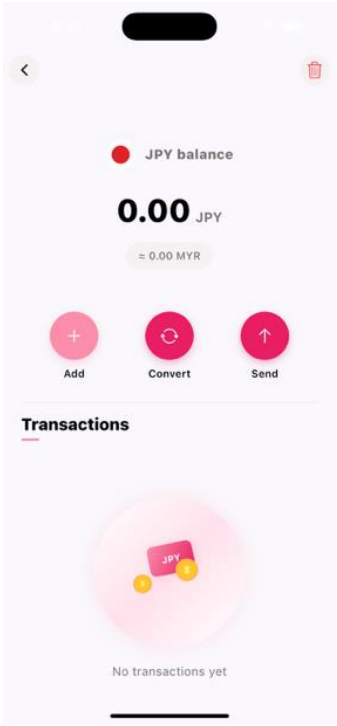

Figure 5.4.5.5 New Currency Wallet Page

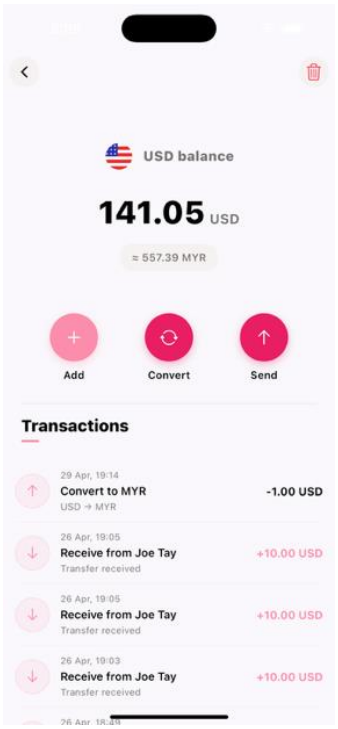

Figure 5.4.5.6 Currency Wallet

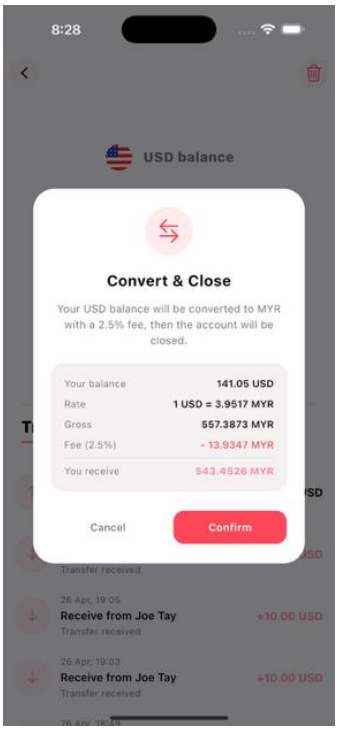

Figure 5.4.5.7 Summary Popout

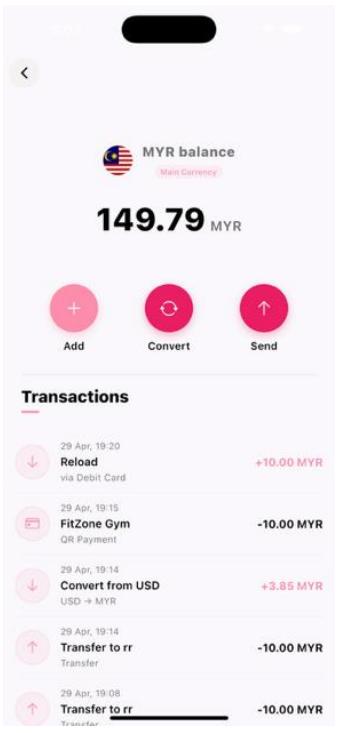

Figure 5.4.5.8 Main Currency Wallet

5.4.6 Send

When the user taps the "Send" button on the Home Page, they are taken to the Send Money screen (Figure 5.4.6.1). The goal of this page is to let people send money to a chosen person, either in the same currency or by changing it to a different one. The "You Send Exactly" field is set to the user's main currency by default, and the "Recipient Gets" field starts out following the same currency as the currency chosen for sending. Users can change the currency they send by choosing from the dropdown menu (Figure 5.4.6.2). The options are based on the currencies they already have in their wallets. The "Recipient Gets" currency, on the other hand, shows a list of all nine currencies that the app supports (Figure 5.4.6.3). If the user uses the Send function from a specific currency wallet, the "You Send Exactly" currency will be set and can't be changed, as shown in Figure 5.4.6.4.

If the user sends the same currency, the amounts in "You Send Exactly" and "Recipient Gets" will be the same. The system shows the selected currency wallet and its current balance below this. The transfer fee section shows that the app charges a service fee of 2.5%. The fee is marked as "Free" (Figure 5.4.6.5) if there is no need to change the currency. If the transaction involves different currencies, though, the fee will be shown (Figure 5.4.6.6), along with the current exchange rate at the top of the page.

After entering the amount and currency they want, the user can click "Add Recipient" to go to the next page, which shows the list of saved contacts (Figure 5.4.6.7). When you choose a recipient, a summary pop-up will show up. For transactions that don't involve changing currencies, the summary just shows the amount of the transfer (Figure 5.4.6.8). When money is being converted, the summary will include detailed information like the current exchange rate, the amount the recipient will get after the conversion, and any service fees that apply (Figure 5.4.6.9). There is also a space for users to add notes. If they don't fill it out, it will say "Fund Transfer" by default.

The user can confirm the transaction after looking over the details. This will start a PIN verification step (Figure 5.4.6.10). When the verification is successful, a screen will show that the transfer was successful (Figure 5.4.6.11). This screen will show information like the recipient, the remark, and the date and time of the transaction.

Users can also share or save the transaction receipt by clicking a button next to the confirm option. This lets them send it to other apps or save it to files or the gallery.

The system also has a way to deal with low balances. If the user tries to send more money than they have in the selected currency, the system will show the amount they are short (Figure 5.4.6.12), and the action button will change to "Top Up & Send." When the user chooses this option, the system will suggest other currencies that the user owns and figure out how much they need to convert to make up the difference (Figure 5.4.6.13). The 2.5% service charge is included in the calculation to make sure the exact amount is met. After this step, the rest of the process follows the same transfer flow as described above.

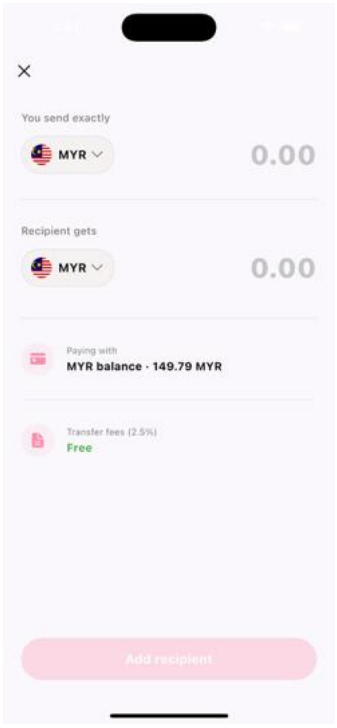

Figure 5.4.6.1 Send Page

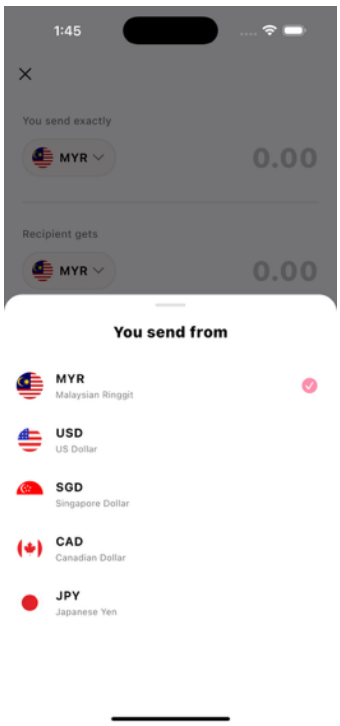

Figure 5.4.6.2 Send from currencies dropdown

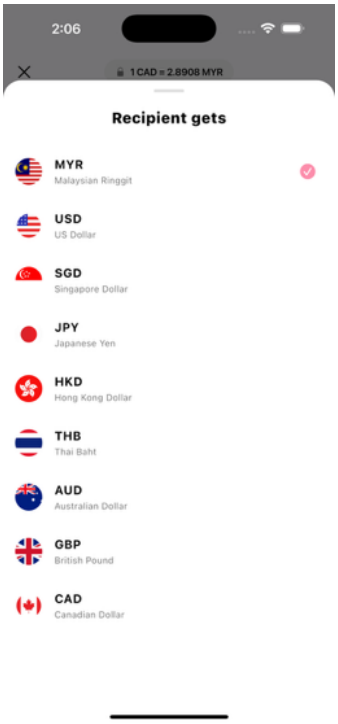

Figure 5.4.6.3 Recipients gets currencies dropdown

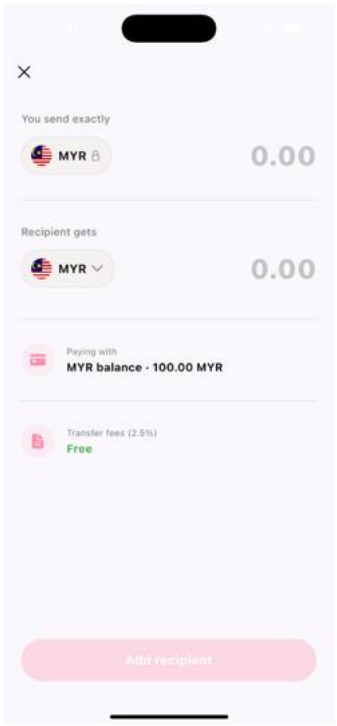

Figure 5.4.6.4 Enter by specific currency wallet

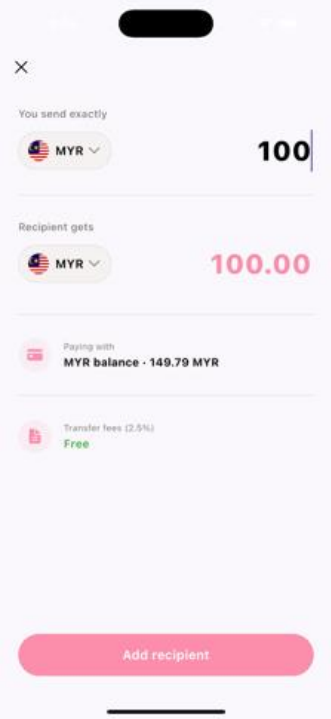

Figure 5.4.6.5 Enter Amount (you send currency and recipient get currency same)

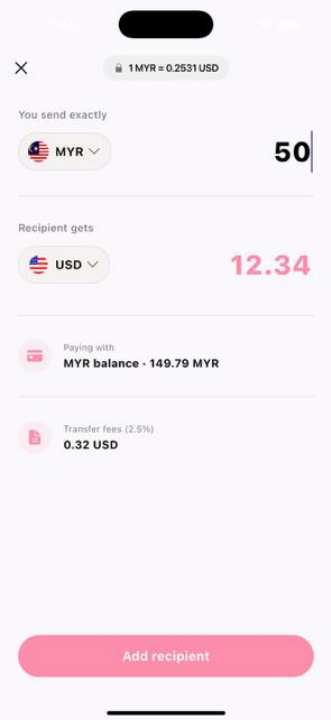

Figure 5.4.6.6 Enter Amount (you send currency and recipient get currency different)

Figure 5.4.6.7 Select Recipient

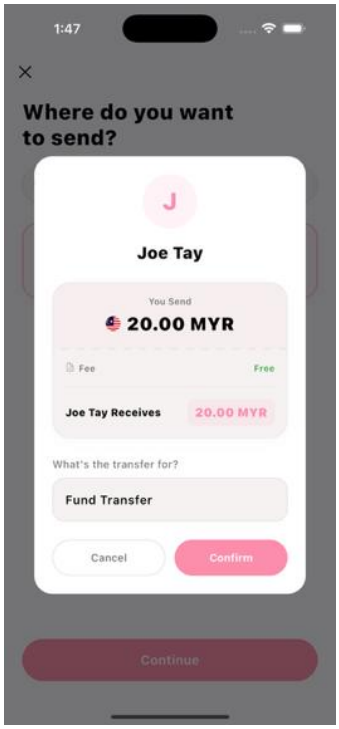

Figure 5.4.6.8 Summary Popout (same currency)

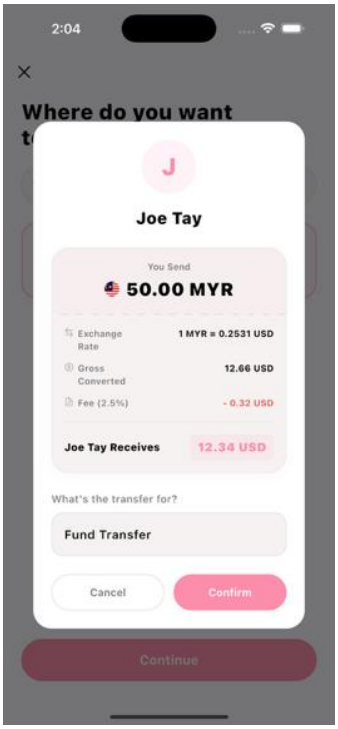

Figure 5.4.6.9 Summary Popout (different currency)

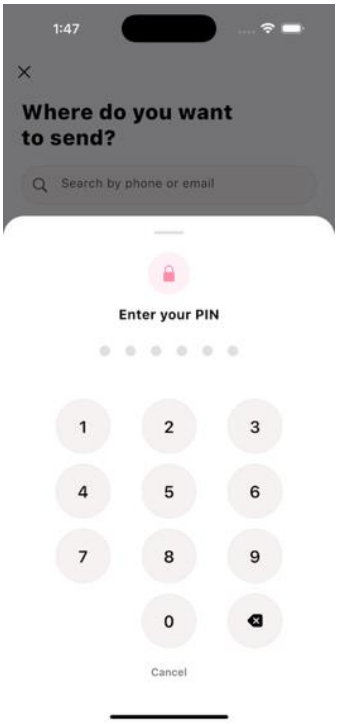

Figure 5.4.6.10 PIN Verify

Figure 5.4.6.11 Transfer Successful Page (Send)

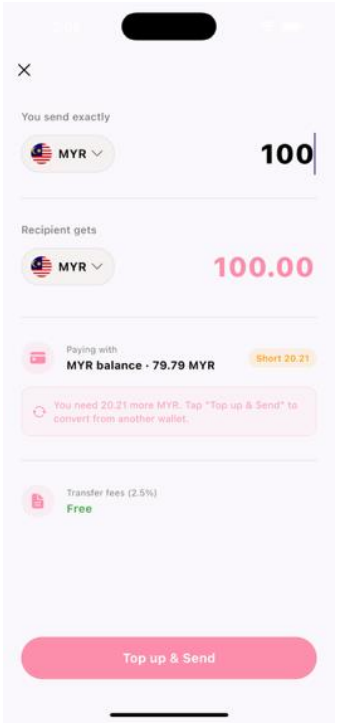

Figure 5.4.6.12 Insufficient Balance

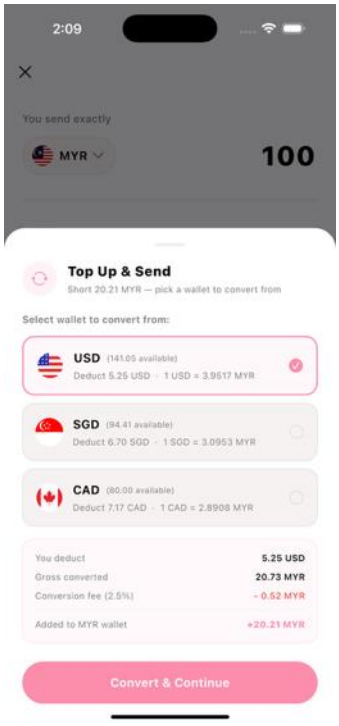

Figure 5.4.6.13 Option Currency to Convert

5.4.7 Add money

If the user clicks "Add Money" on the Home Page, they will be taken to the Add Money page (Figure 5.4.7.1). The user's main currency is the default currency for adding funds. Users can tap on the currency field to see a list of all the currency wallets they own, along with the balance for each one (Figure 5.4.7.2). Figure 5.4.7.3 shows that if the user goes to the Add Money page from a specific currency wallet, the currency will be set and cannot be changed. After choosing the currency, the user can type in the amount they want to add.

The next step is for the user to choose a way to pay. If the user has already added a bank card and used it to add money, that card will be the default way to pay. If not, the choice to "Add New Card" will show up. When the user taps on the payment method section, a selection sheet will pop up with all the cards that have already been added and the option to add a new card (Figure 5.4.7.4). If the user wants to add a new card, they will be taken to a form where they can fill out the needed card information (Figure 5.4.7.5).

After entering all the required information, the user can tap "Add Money" to see a confirmation summary (Figure 5.4.7.6). The user can click "Confirm Pay" after checking that all the information is correct. The payment will then go through the bank. Then, as shown in Figure 5.4.7.7, a message will appear saying that the transaction was successful.

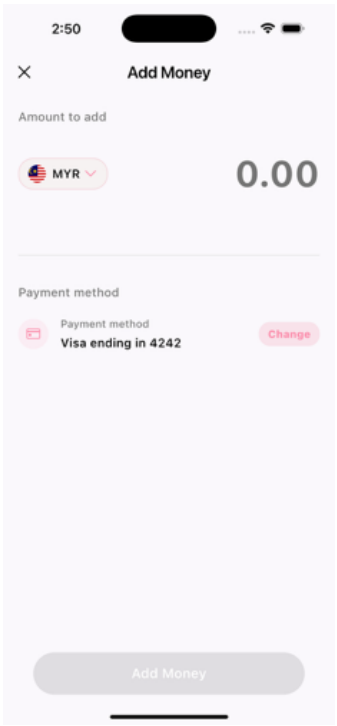

Figure 5.4.7.1 Add Money Page

Figure 5.4.7.2 Select Currency Dropdown

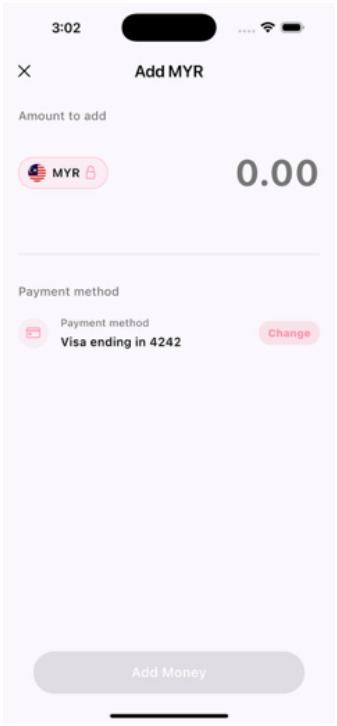

Figure 5.4.7.3 Enter from Specific Currency Wallet Money Page

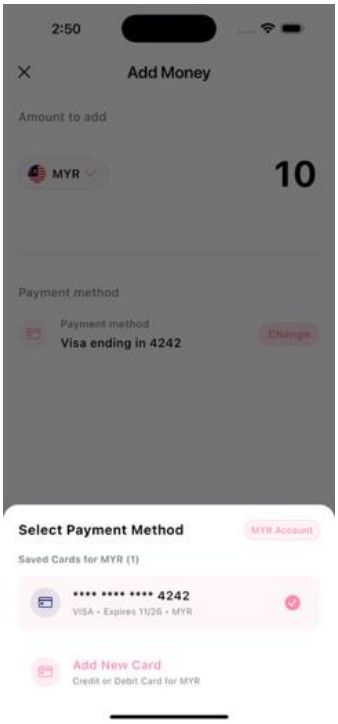

Figure 5.4.7.4 Select Payment Method

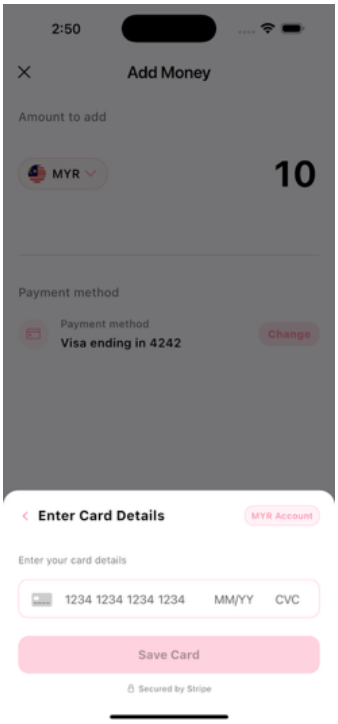

Figure 5.4.7.5 Card Details Form

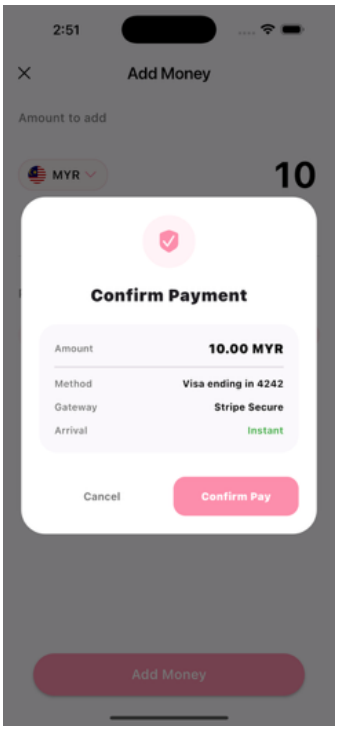

Figure 5.4.7.6 Confirm Payment Popout

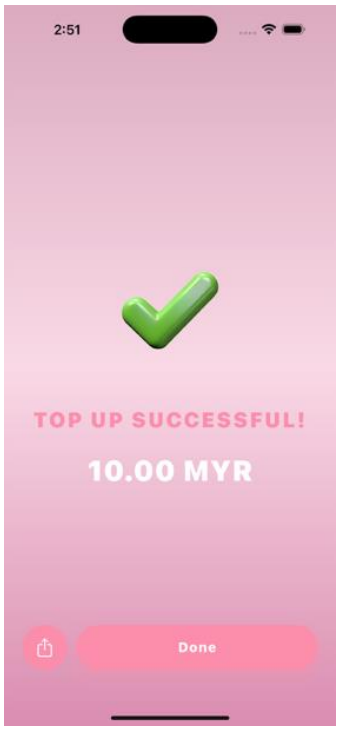

Figure 5.4.7.7 Add Money Successful Page

5.4.8 Convert

To change money, the user must first go to the detail page of the currency wallet they want to use (Figure 5.4.8.1) and then tap the "Convert" button. This will take them to the Convert page (Figure 5.4.8.2). The currency you chose on this page is set as the "You Convert" currency and can't be changed. The current rate of exchange is shown at the top of the screen. Users can only change the "You Get" currency, and the only options are the currencies that the app supports (Figure 5.4.8.3).

The user can type in the amount they want to change after choosing the currency they want to change. If the amount entered is more than the available balance, the system will show a "Insufficient Balance" message and stop any further action until the amount is changed (Figure 5.4.8.5). The system will figure out and show the user how much of the target currency they will get if the amount they entered is less than the available balance (Figure 5.4.8.4). The "You Get" amount shown is the net amount after the app's 2.5% service fee has been taken out. This is the exact amount the user will get. For the sake of openness, users can also see a breakdown below that shows the gross amount, the service fee, and the final net amount after all deductions.

After all the information has been verified, the user can tap the "Convert" button. This will bring up a confirmation pop-up (Figure 5.4.8.6) to make sure the user wants to go ahead. The system will ask for a PIN verification step after confirming again (Figure 5.4.8.7). A message saying "Conversion Successful" will appear after the verification is successful (Figure 5.4.8.8). The amount of the source currency that is left will be taken out (Figure 5.4.8.9), and the amount that was changed will be added to the wallet for the target currency (Figure 5.4.8.10).

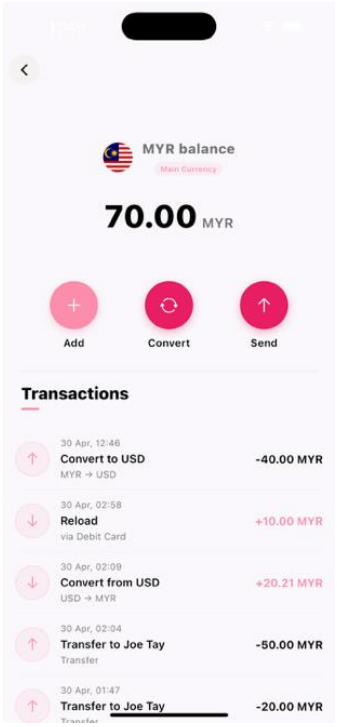

Figure 5.4.8.1 Specific Currency Wallet Page

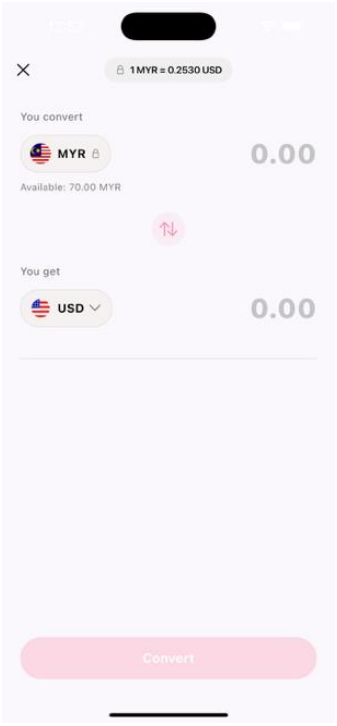

Figure 5.4.8.2 Convert Page

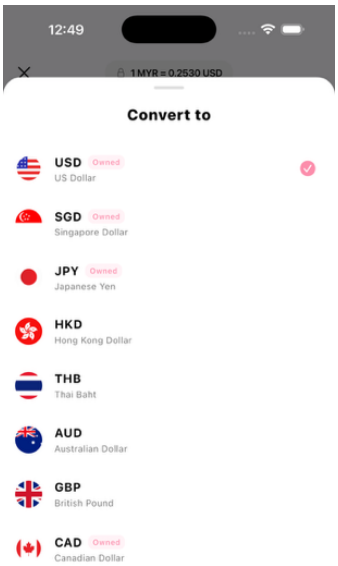

Figure 5.4.8.3 Convert to Currency Dropdown

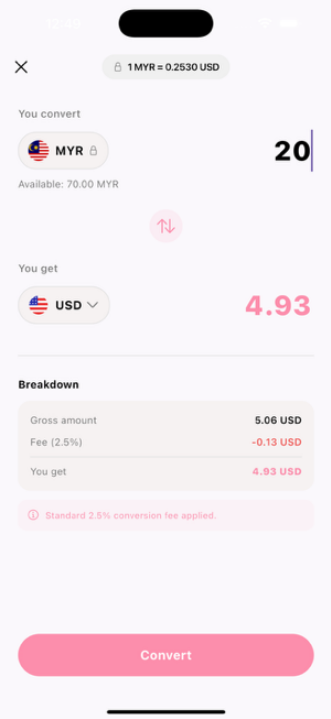

Figure 5.4.8.4 Enter Amount in Convert Page

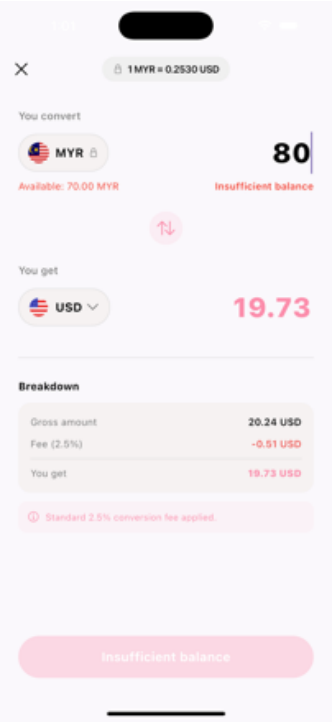

Figure 5.4.8.5 Insufficient Balance

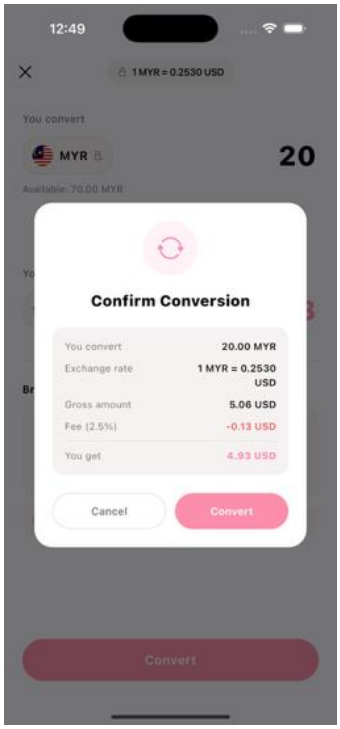

Figure 5.4.8.6 Confirm Conversion Popout

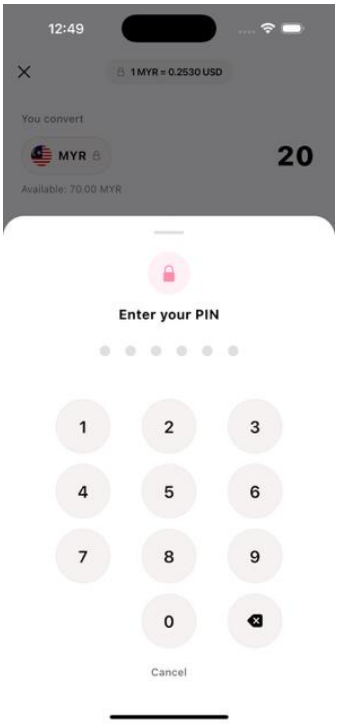

Figure 5.4.8.7 PIN Verify Convert Part

Figure 5.4.8.8 Conversion Successful Page

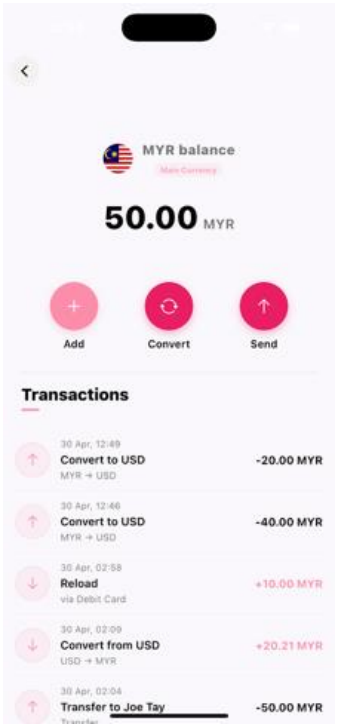

Figure 5.4.8.9 Convert from Currency Wallet

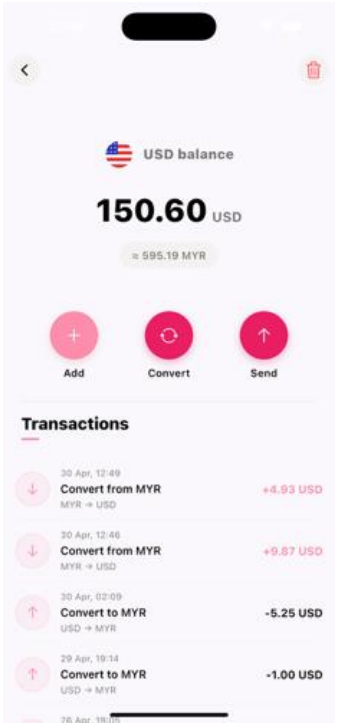

Figure 5.4.8.10 Convert to Currency Wallet

5.4.9 Transaction

The Transaction section on the Home Page only shows the three most recent records. The full transaction history page (Figure 5.4.9.1) will open when the user clicks "See All." This page shows all transactions. This section will keep track of all financial transactions, such as transfers and balance changes. These records cannot be deleted. Transactions are grouped by date, with the most recent ones (like those from today) shown at the top for easy access.

Users can use the search bar at the top of the page to find certain transactions by typing in the right keywords. Also, you can use the button in the bottom right corner of the search bar to filter your results. Depending on the data in the records, this filter lets users narrow down transaction records by type and currency (Figure 5.4.9.2).

Users can also use date filters to see transactions that happened in a certain time period (Figure 5.4.9.3). There are preset date ranges like "Today," "Yesterday," and "Last 30 Days." Users can also choose their own start and end dates. The transaction history will show records from the last 30 days by default.

You can also interact with each transaction record. Figure 5.4.9.4 shows that when you choose it, it will show detailed information such as the type of transaction, the recipient, the payment details, the reference ID, and other relevant information.

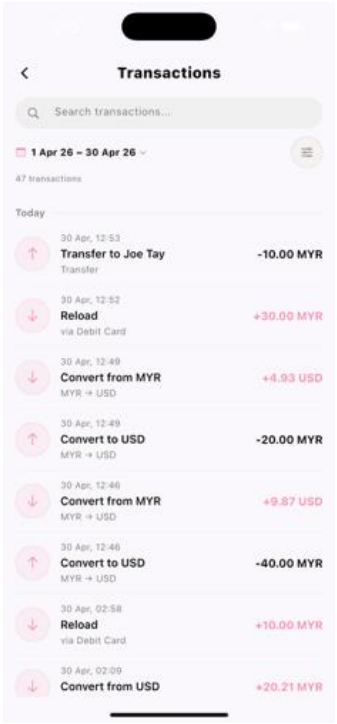

Figure 5.4.9.1 Transaction Page

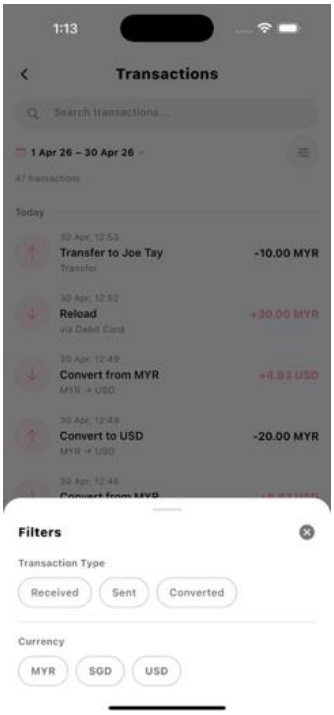

Figure 5.4.9.2 Filters Transaction Option

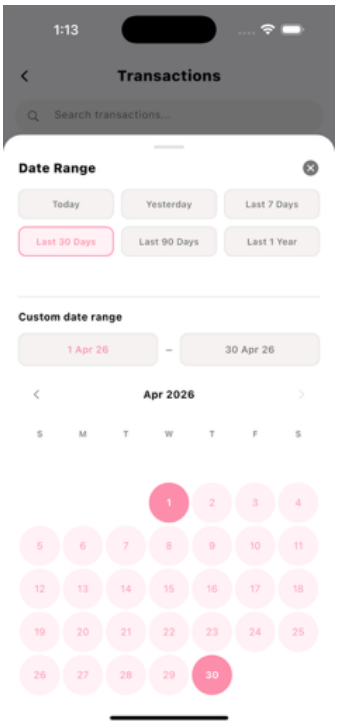

Figure 5.4.9.3 Transaction Date Filter

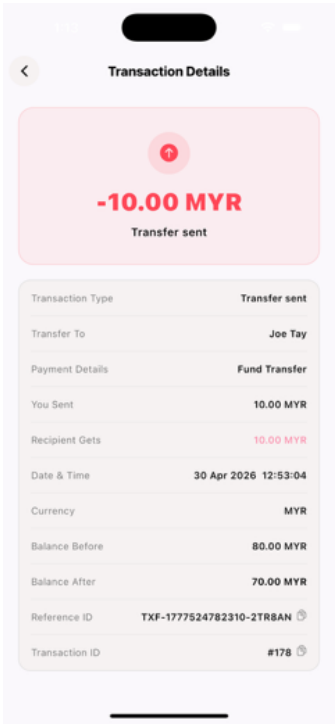

Figure 5.4.9.4 Transaction Details

5.4.10 Cash Flow

When the user taps the Cash Flow button, they will be taken to the Cash Flow page (Figure 5.4.10.1). There, a pie chart will show statistical data based on the user's transactions for the current month. As shown in Figure 5.4.10.2, the data is split into two main sections: Cash Out and Cash In. Each section has its own type of transaction. Below the chart, each type of transaction is listed with its percentage, starting with the one with the highest percentage.

There is a timeline selector at the top of the page, below the Cash Flow title. This lets users change the time period they are looking at (Figure 5.4.10.3). Users can choose to see things in different ways, like by month or by year. In the top right corner, there is also a currency filter option (Figure 5.4.10.4). The system shows data in the user's main currency by default. This means that all transactions in different currencies are combined and changed into the main currency for analysis. Users can, however, choose to filter by a specific currency. In this case, the statistics will only be based on the transaction records for that currency.

Also, you can click on each category below the chart. When you choose it, it will show all related transaction records based on the timeline and currency filter you chose, as shown in Figure 5.4.10.5.

Figure 5.4.10.1 Cash Flow Page (Money Out)

Figure 5.4.10.2 Cash Flow Page (Money In)

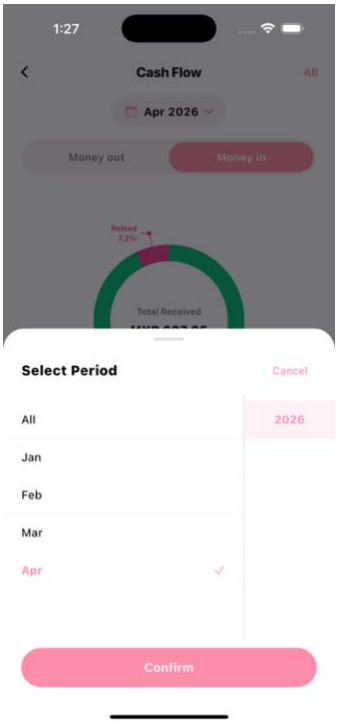

Figure 5.4.10.3 Period Filter Option

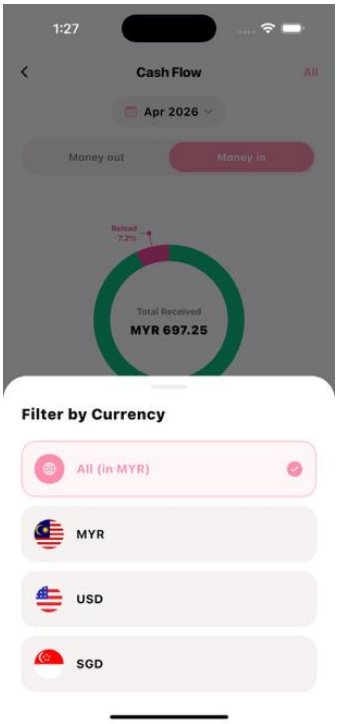

Figure 5.4.10.4 Filter Currency Option

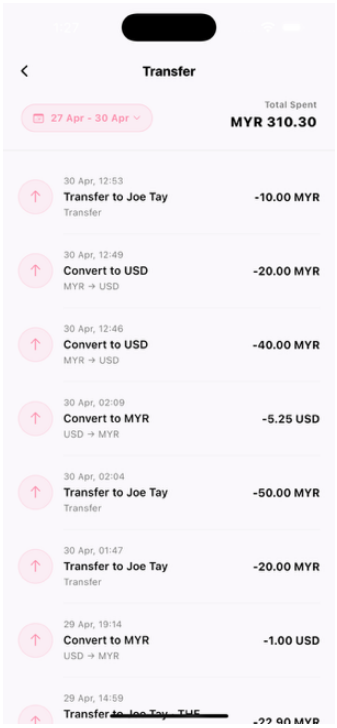

Figure 5.4.10.5 Transaction Type Details

5.4.11 Shared Wallet

The user will be taken to the Shared Wallet page (Figure 5.4.11.1) when they click on "Shared Wallet" in the bottom navigation bar. There are two parts to this page: "My Wallets" and "Invitations." The "My Wallets" section shows all the shared wallets that the user has either created or joined. The "Invitations" section (Figure 5.4.11.2) shows any invitations from other users that are still open for the user to join their shared wallets. People can choose whether or not to accept these invites. Users can also make a new shared wallet by clicking the "+ New" button in the top right corner. The current balance, status (active or not), currency, wallet name, and number of members are all shown on each shared wallet entry.

When a user chooses a shared wallet, they will be taken to its detail page (Figure 5.4.11.3). The name of the wallet is at the top, and below that are three main sections: Overview, Transaction, and Cash Flow. The Overview section shows the current balance and has buttons for actions like Top Up, Pay, and Payment Limits. It also shows the list of people who can use the shared wallet and names the owner.

The Transaction section (Figure 5.4.11.4) shows all of the transaction records for the shared wallet that you chose. Like the main transaction page, it has features like search, filtering, and date filtering. But each record makes it clear which member did the transaction. The Cash Flow section (Figure 5.4.11.5) gives you a statistical breakdown of the shared wallet you chose, just like the main Cash Flow page.

There are also two features that only the wallet owner can use. There is a button in the top right corner that says "Add Member." This lets the owner invite other people to join the shared wallet (Figure 5.4.11.6). To the right of it is a delete icon that you can use to close the shared wallet. When the user clicks on it, they are taken to a closure page (Figure 5.4.11.7), where the owner can choose how to split up any remaining balance. You can choose to split the money evenly, give it to one member, or use a smart refund method.

Users can easily enter the amount they want and choose a payment method on the Top Up page for the shared wallet (Figure 5.4.11.8). You can pay with a bank card or a personal wallet. If you choose a personal wallet, the balance will show up. Figure 5.4.11.9 shows the Pay function, which lets users pay by scanning a merchant's QR code. The current balance of the shared wallet is shown at the top.

Lastly, if the user clicks on the Payment Limits button, they will be taken to Figure 5.4.11.10, which shows the daily and per-transaction limits for the shared wallet. Only the owner can change these limits; other users can see them but can't change them.

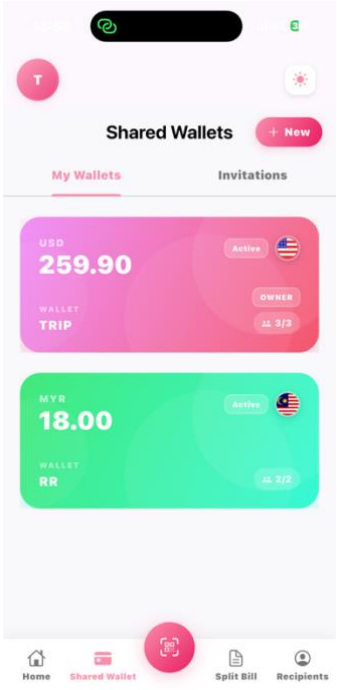

Figure 5.4.11.1 Shared Wallet Page (My Wallets)

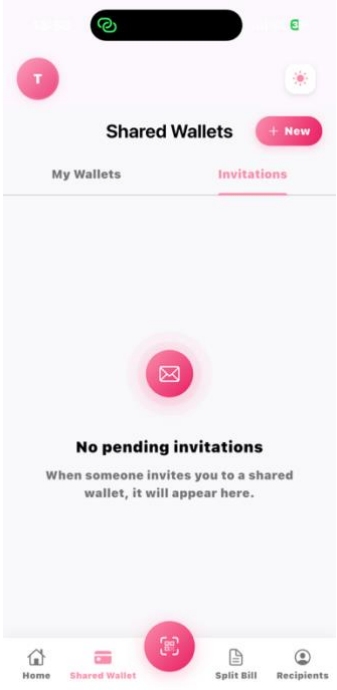

Figure 5.4.11.2 Shared Wallet Page (Invitations)

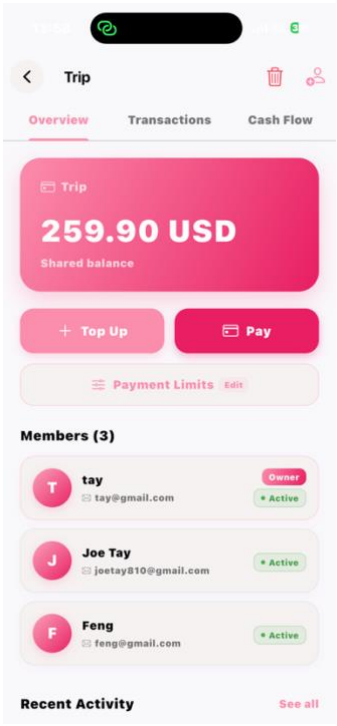

Figure 5.4.11.3 Shared Wallet Details Page

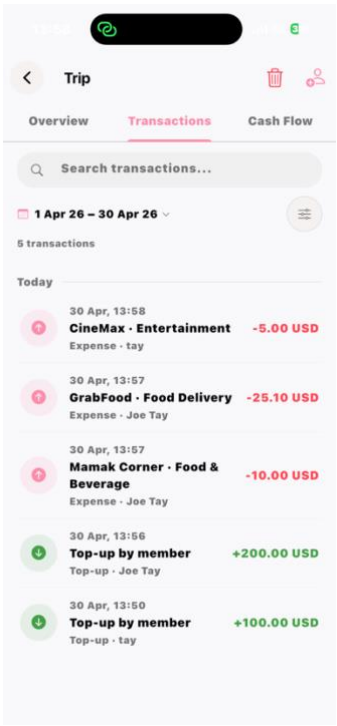

Figure 5.4.11.4 Shared Wallet Details Page (Transactions)

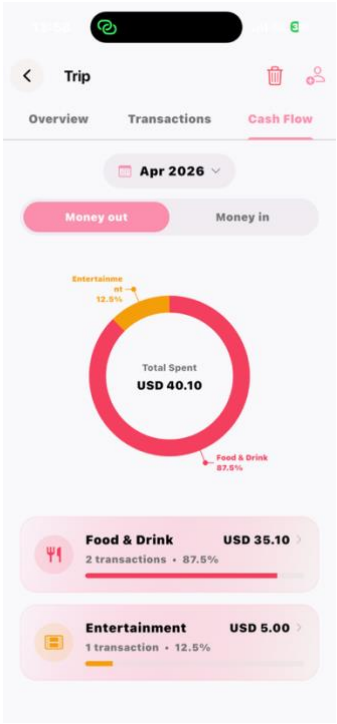

Figure 5.4.11.5 Shared Wallet Details Page (Cash Flow)

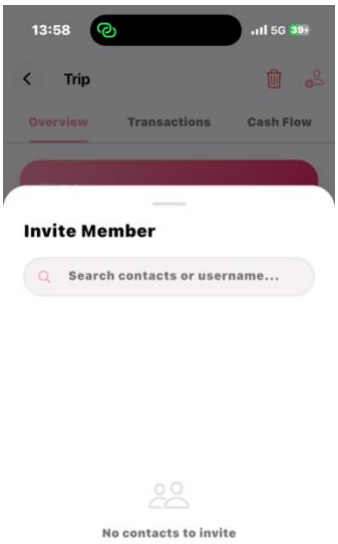

Figure 5.4.11.6 Invite Member

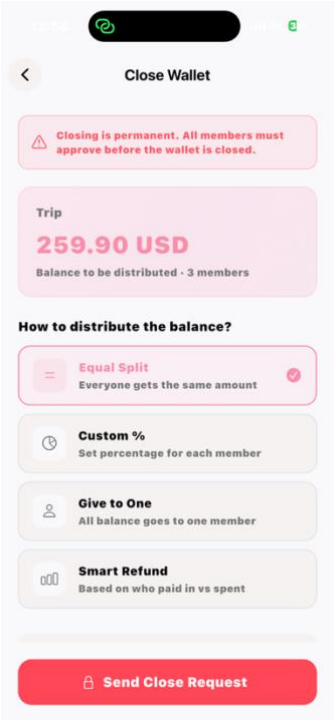

Figure 5.4.11.7 Close Wallet Page

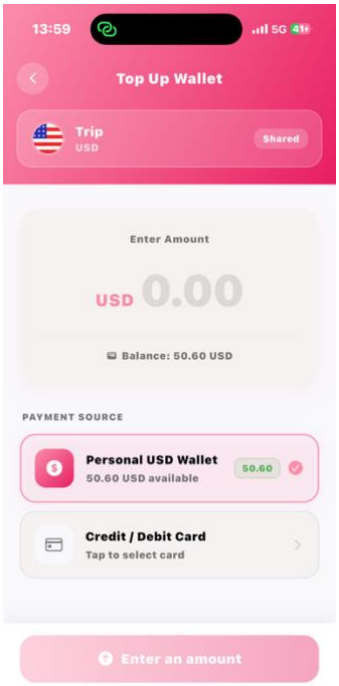

Figure 5.4.11.8 Shared Wallet Top Up Page

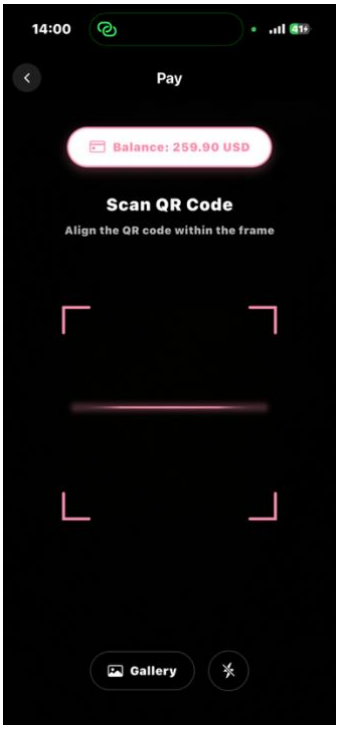

Figure 5.4.11.9 Shared Wallet Pay Page

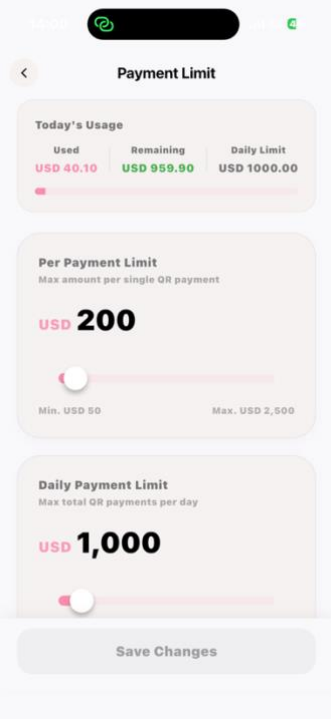

Figure 5.4.11.10 Shared Wallet Payment Limit Page

5.4.11.1 Shared Wallet (Owner)

When the owner makes a shared wallet, they start by tapping the "Add" button, which takes them to the page where they can create the wallet (Figure 5.4.11.1.1). The first thing to do is choose the currency you want for the shared wallet. Next, the user gives the wallet a name (Figure 5.4.11.1.2). After this, the user can ask people to join (Figure 5.4.11.1.3). The list shows contacts that have already been added. If the user wants to invite someone who isn't on the list, they can use the search bar to find them. A shared wallet needs at least two people to be made, so keep that in mind. The user can tap "Create Wallet" once all the settings are done.

After you make the shared wallet, it will show up as a greyed-out card with the words "Waiting for members to accept invitation" under the "My Wallets" section (Figure 5.4.11.1.4). The wallet becomes active and is shown in full color once the people who were invited accept the invitation (Figure 5.4.11.1.5).

If the owner wants to close the shared wallet in the future, they must first choose a way to handle the remaining balance (Figure 5.4.11.1.6). After picking a method, the owner can tap "Send Close Request," which will show a preview of how the balance will be split (Figure 5.4.11.1.7). If the owner agrees, they send in the request, and the system sends the proposal to all members to vote on. During this time, the shared wallet will be turned off until everyone has voted and a final decision has been made. In the Overview section (Figure 5.4.11.1.8), you can see the current voting status and what each member has chosen.

The wallet can't be closed if any member says no to the proposal. In this case, the owner will have two more choices: they can either cancel the request and keep using the wallet, or they can change the way the money is distributed based on the feedback or reasons given by the members who said no (Figure 5.4.11.1.9). But if everyone agrees to the proposal, the shared wallet will be successfully closed and taken off the "My Wallets" list. After the process is over, the user will get the specified amount, and any allocated amount shown during the process will be given out as planned.

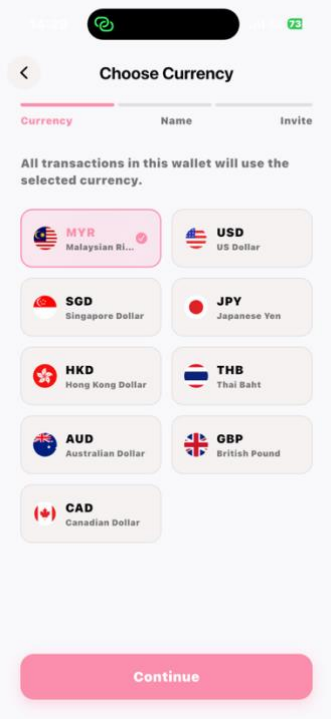

Figure 5.4.11.1.1 Shared Wallet Top Up Page

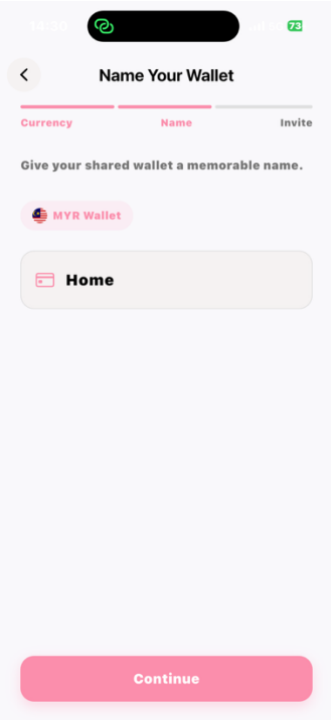

Figure 5.4.11.1.2 Shared Wallet Pay Page

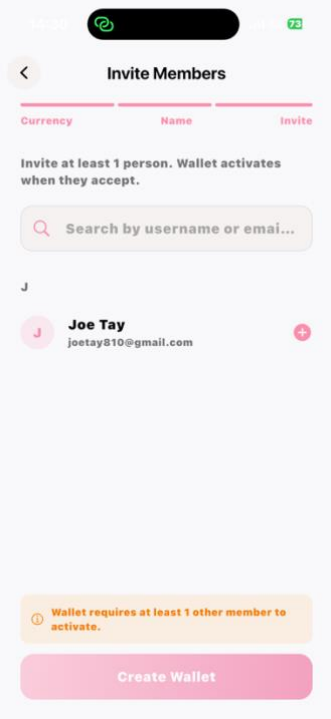

Figure 5.4.11.1.3 Shared Wallet Invite Member Page

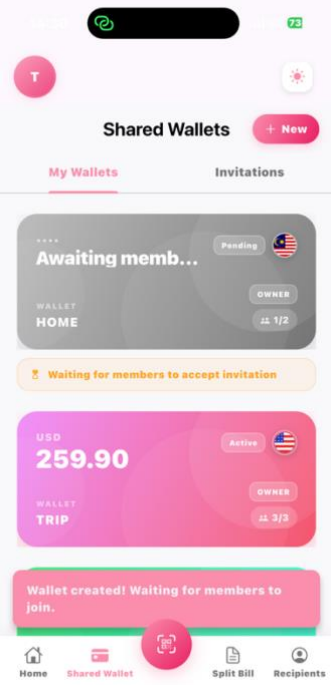

Figure 5.4.11.1.4 My Wallets New Shared Wallet Waiting Member Accept

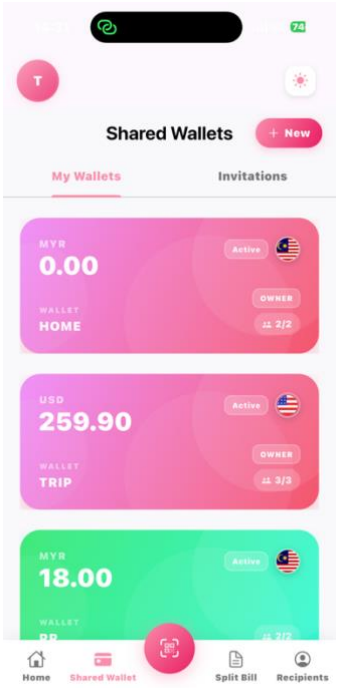

Figure 5.4.11.1.5 Shared Wallet Successful Active

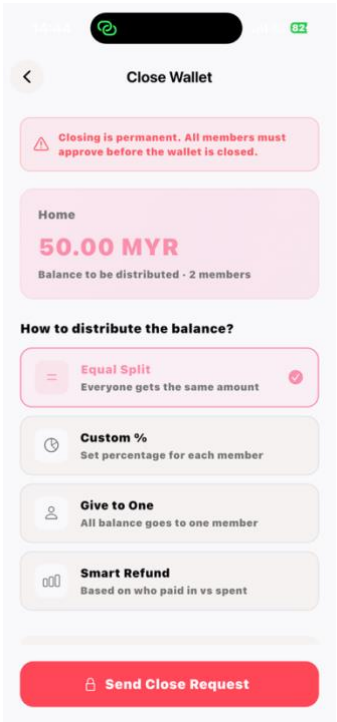

Figure 5.4.11.1.6 Close Shared Wallet Select Distribution

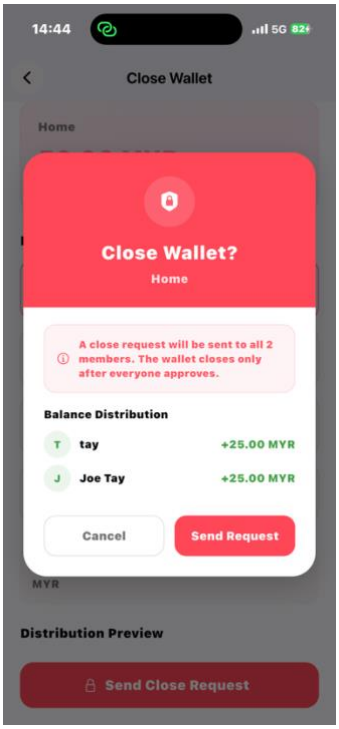

Figure 5.4.11.1.7 Close Shared Wallet Confirmation Popout

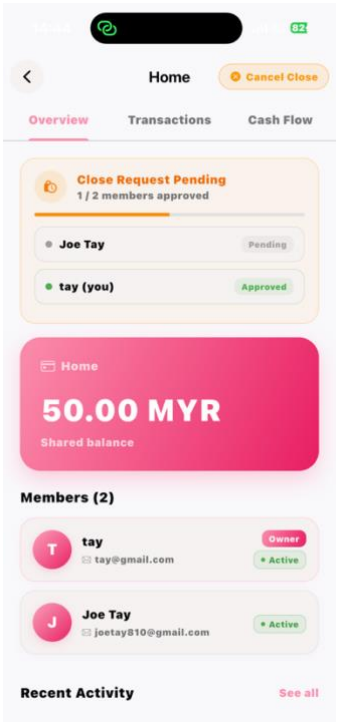

Figure 5.4.11.1.8 Close Request Pending

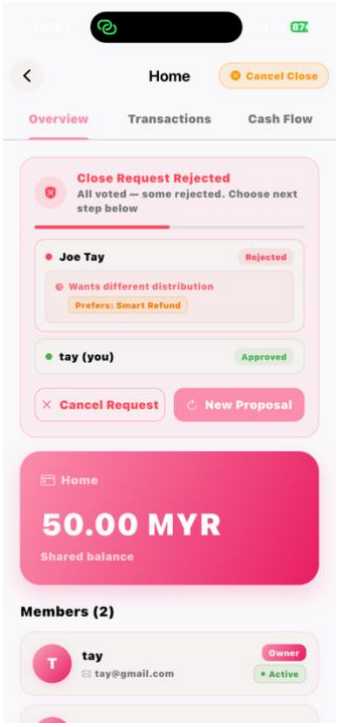

Figure 5.4.11.1.9 Member Decline the Closing Request

5.4.11.2 Shared Wallet (Member)

When the owner invites a member to join a shared wallet, the invitation will show up in the "Invitations" section. Figure 5.4.11.2.1 shows that the user can either accept or decline the invitation. The shared wallet will be added to the "My Wallets" section (Figure 5.4.11.2.2) if the user agrees. After that, the user will be able to access and use the wallet.

If the owner later asks to close the shared wallet, the Overview section will show a "Close Request Pending" status. This is where the user can also see how the voting is going (Figure 5.4.11.2.3). When the user clicks on "Review & Vote," they will be taken to a page (Figure 5.4.11.2.4) that shows the owner's suggested distribution method and the expected outcome. If the user agrees with the proposal, they can approve it, which will help close the wallet successfully. But if the user doesn't like the proposal, they can turn it down and give a reason. You can do this by either picking a general reason (Figure 5.4.11.2.5) or saying that you're unhappy with the way the distribution is done and giving a full explanation (Figure 5.4.11.2.6).

The user will be able to see the updated voting status in the Overview section (Figure 5.4.11.2.7) after they vote. The shared wallet will be temporarily unavailable during this voting period, and no transactions can be made. Until everyone has voted, the process will stay on hold. After that, the owner will choose whether to cancel the request to close the wallet and keep using it or to change the way the money is distributed and start another round of voting.

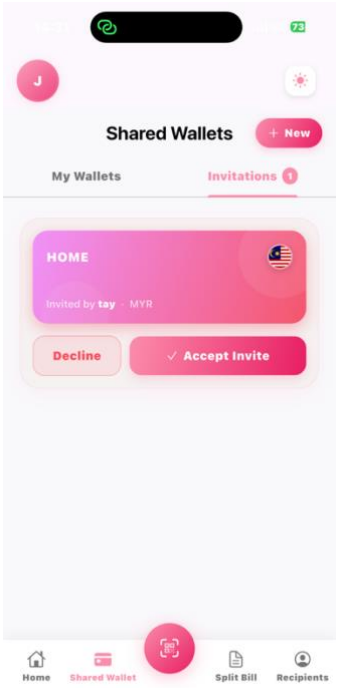

Figure 5.4.11.2.1 Invitation Shared Wallet from Creator

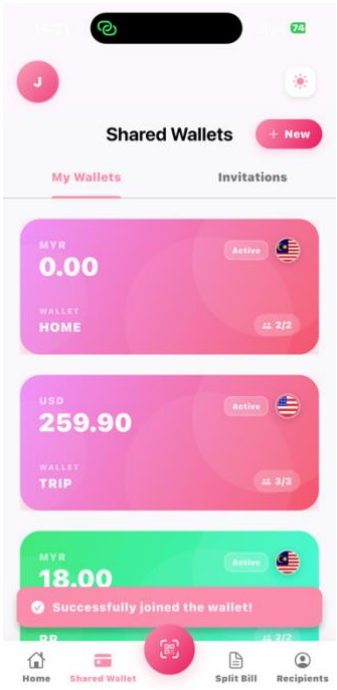

Figure 5.4.11.2.2 Accept and Successful Join Shared Wallet

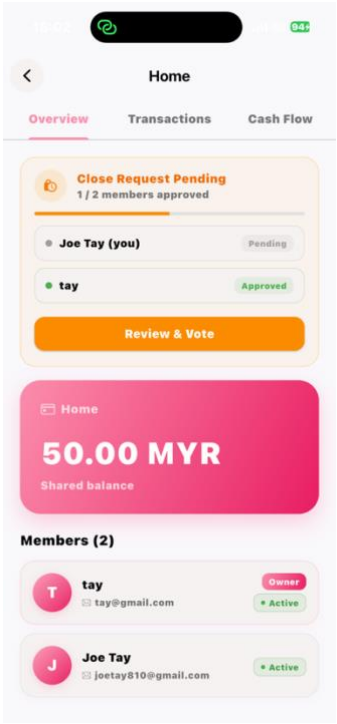

Figure 5.4.11.2.3 View of Member Close Request Pending

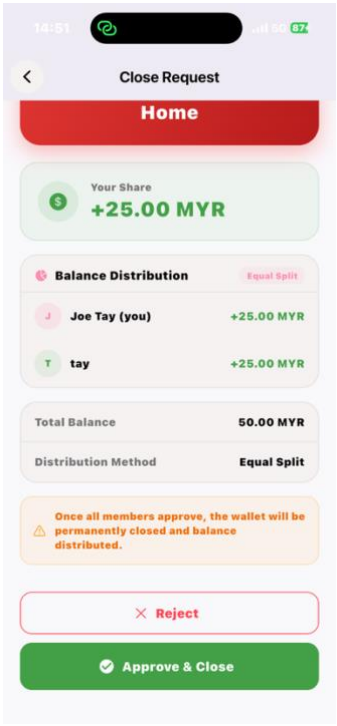

Figure 5.4.11.2.4 Review Distribution of Shared Wallet and Decision

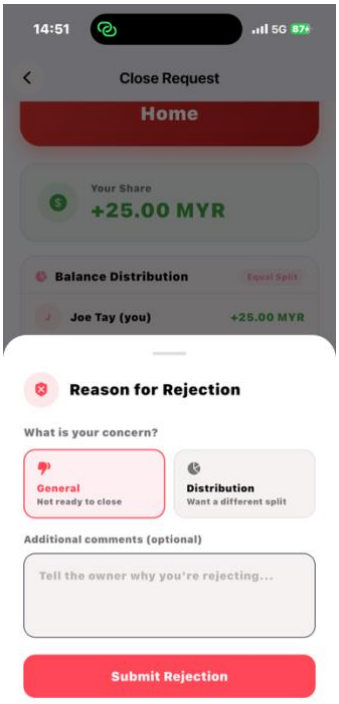

Figure 5.4.11.2.5 Reject Reason (General)

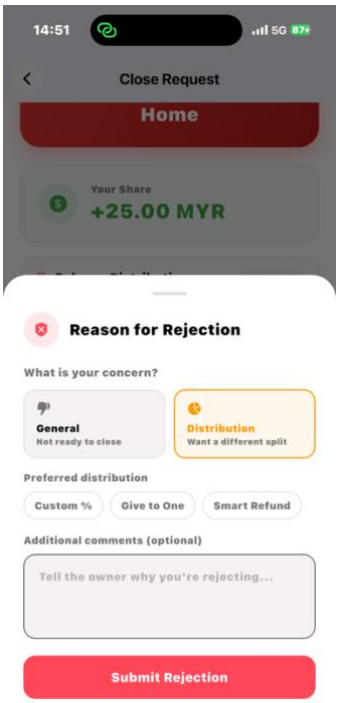

Figure 5.4.11.2.6 Reject Reason (Distribution)

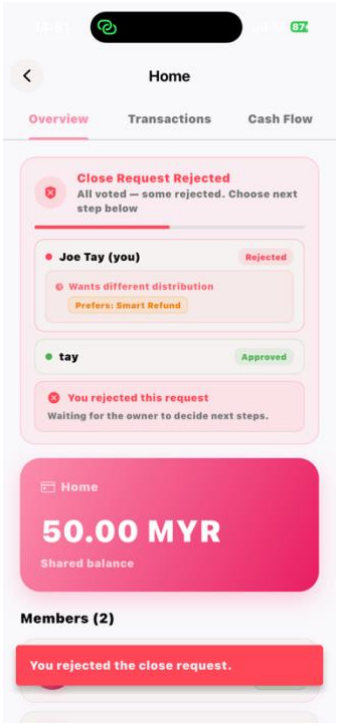

Figure 5.4.11.2.7 Submit the Reject Reason for Vote

5.4.12 Split Bill

When the user clicks on "Split Bill" in the navigation bar, they will be taken to the Split Bill page (Figure 5.4.12.1), which has three sections: Unpaid, Paid, and My Bills. The Unpaid section shows all the bills that have been sent to the user. If there aren't any, a message saying "No unpaid bills" will appear. The Paid section (Figure 5.4.12.2) has all the records of bills that the user has already paid. Figure 5.4.12.3 shows the My Bills section, which has all the bills that the user has made and sent. These are then split into two groups: bills that have not yet been fully paid and bills that have been fully paid. The name of the bill, the total amount to be collected, the amount already collected, and the amount still owed are all shown on each record. It also shows how many people have received the bill.

To make a new bill, users should tap the "Add" button in the top right corner. This will take you to a page (Figure 5.4.12.4) where you can either scan a receipt by taking a picture or choosing one from the gallery, or you can enter the bill information by hand. The currency dropdown only shows the currencies that the user has in their wallets right now. If the user wants to scan a receipt, the system will first show them how to do it (Figure 5.4.12.5). Then, it will show them the results (Figure 5.4.12.6),

which will include the bill name, items, quantities, amounts, and any taxes that apply. Before choosing recipients (Figure 5.4.12.7), users must check and, if necessary, change the information that was taken from the document. After confirming the recipients, the system will ask the user to mark which items are theirs. This means that those items won't be charged to other recipients (Figure 5.4.12.8). If there aren't any of those items, the user can move on without choosing any. When you finish, the bill will be successfully created and will show up in the My Bills section as an active (open) bill (Figure 5.4.12.9).

The person who made the bill can click on it to see all the information about it and check the payment status of each person (Figure 5.4.12.10). The system will show that a member has paid by putting a "Paid by [member name]" label on the payment (Figure 5.4.12.11). The amount collected will go up and the amount pending will go down in the My Bills overview (Figure 5.4.12.12). Also, the system will only let the creator delete a bill right away if no payments have been made. But if any member has already paid, a warning pop-up will appear (Figure 5.4.12.13) to let the user know that deleting the bill will cause refunds to be sent.

From the recipient's point of view, any bill they receive will show up in the Unpaid section, along with how long ago it was sent (Figure 5.4.12.14). After choosing the bill, the user will be able to see the itemized information and choose the things they ate (Figure 5.4.12.15). Then, the system will automatically figure out the total amount due, which includes any taxes that may apply (Figure 5.4.12.16). After looking over the amount, the user can tap "Pay," which will ask them to choose a payment method, either from their own wallet or a shared wallet with the right currency (Figure 5.4.12.17). After the payment is finished, a success screen will show up (Figure 5.4.12.18), and the bill will move to the Paid section (Figure 5.4.12.19).

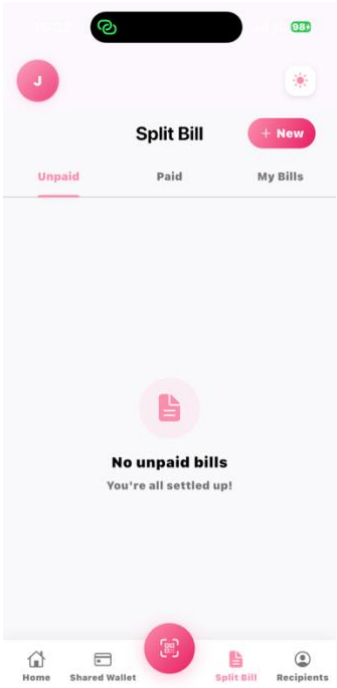

Figure 5.4.12.1 Split Bill Page

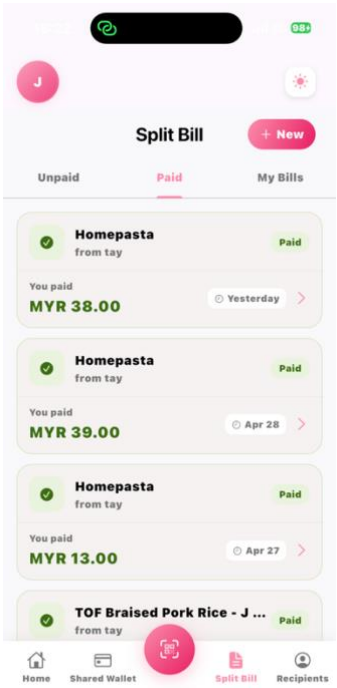

Figure 5.4.12.2 Split Bill Page (Paid)

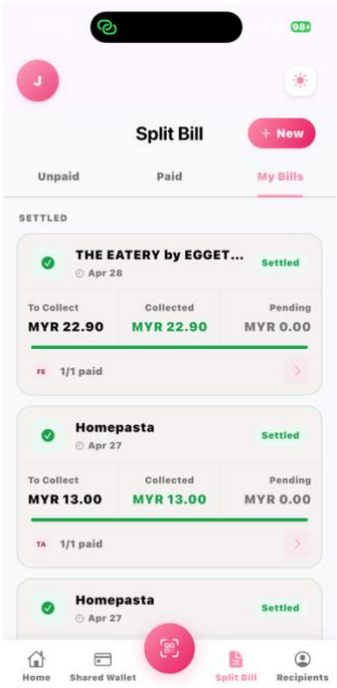

Figure 5.4.12.3 Split Bill Page (My Bills)

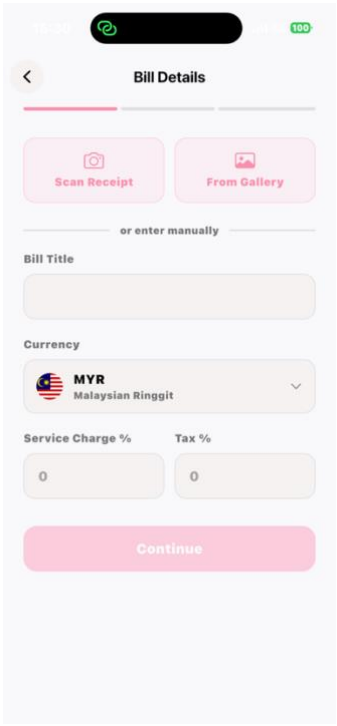

Figure 5.4.12.4 Create Bill Page

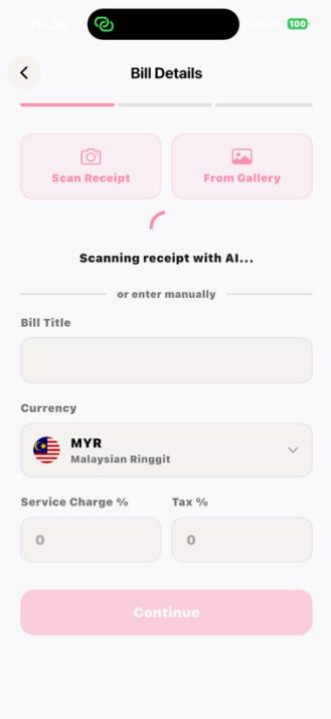

Figure 5.4.12.5 Scanning Receipt via Photo

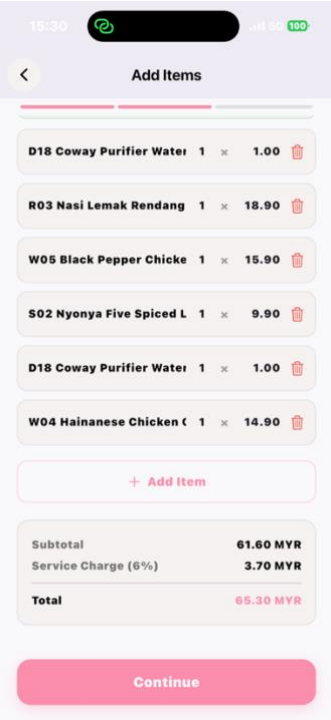

Figure 5.4.12.6 Result Scanned and Detected By AI

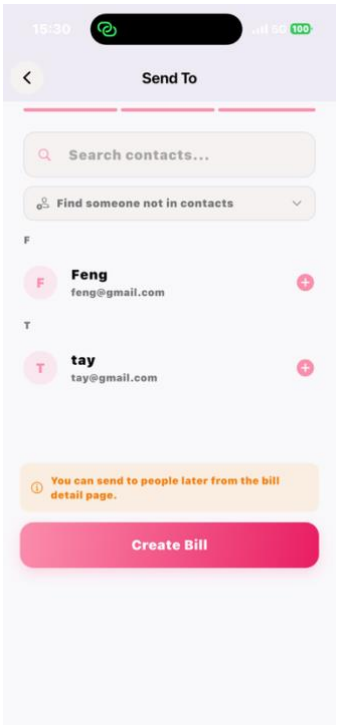

Figure 5.4.12.7 Send Bill Page

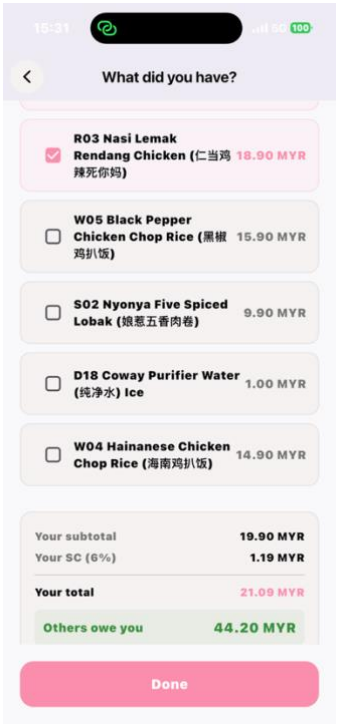

Figure 5.4.12.8 Select the Item that Recipient No Need Pay

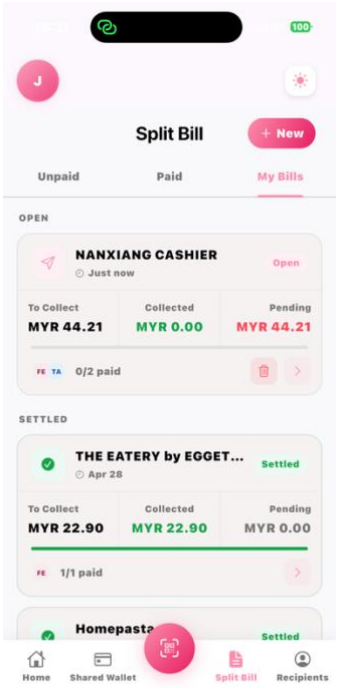

Figure 5.4.12.9 Created Bill (My Bills)

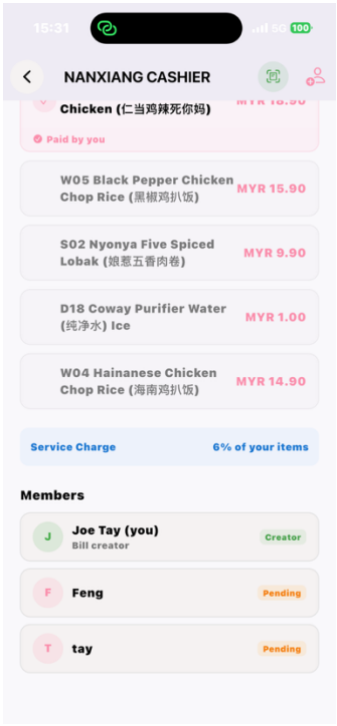

Figure 5.4.12.10 Created Bill Details Page

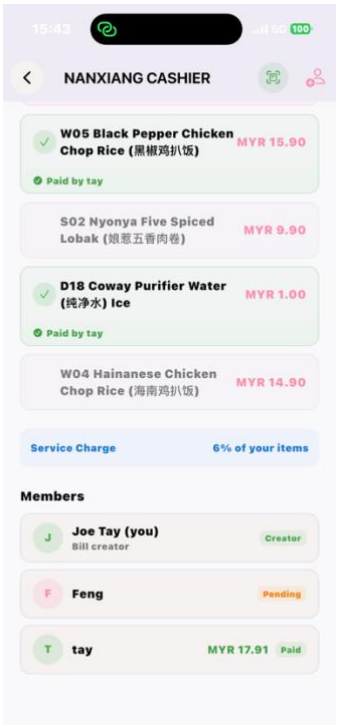

Figure 5.4.12.11 Payment Status of Member in Bill Details

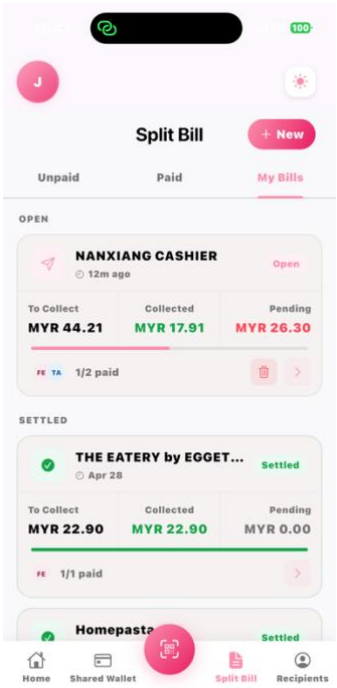

Figure 5.4.12.12 Payment Status (My Bills)

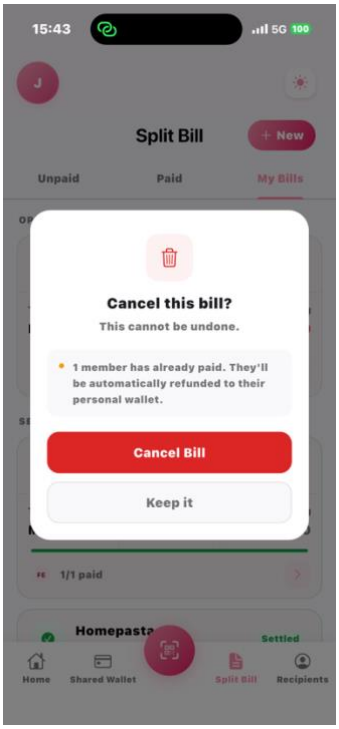

Figure 5.4.12.13 Cancel the Bill When Some Member was Paid

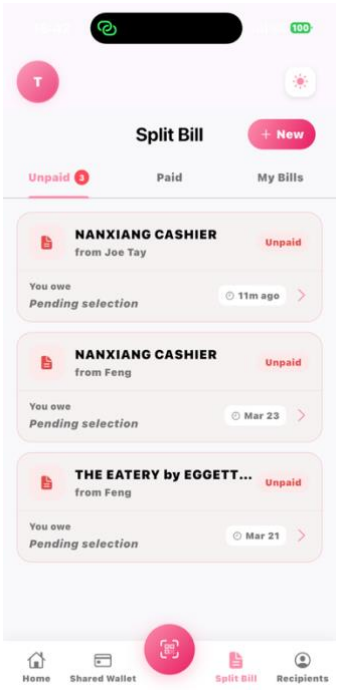

Figure 5.4.12.14 Spit Bill (Unpaid)

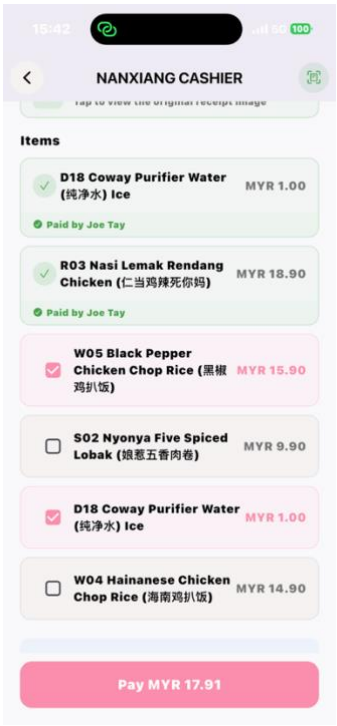

Figure 5.4.12.15 Select the Items that yours (Unpaid)

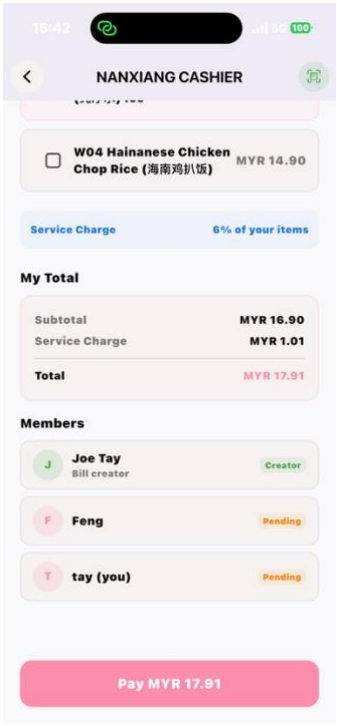

Figure 5.4.12.16 Review the Calculation (Unpaid)

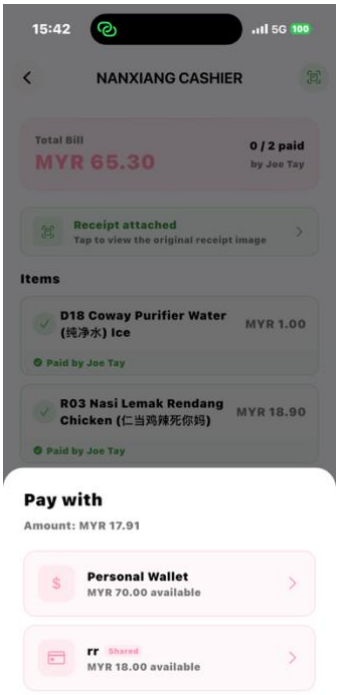

Figure 5.4.12.17 Select Payment Method (Unpaid)

Figure 5.4.12.18 Payment Successful Page (Split Bill)

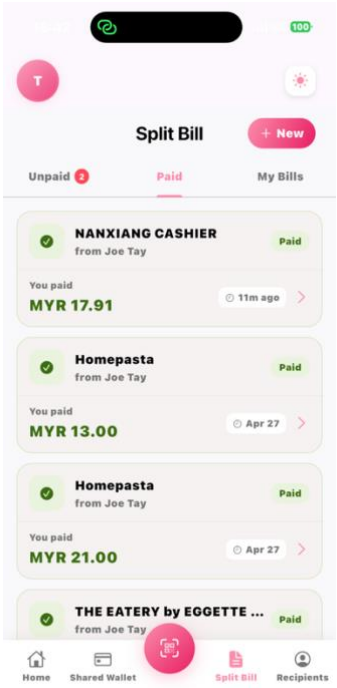

Figure 5.4.12.19 Recorded in Paid Page

5.4.13 Recipient

When the user clicks on "Recipient" in the navigation bar, they will go to the page shown in Figure 5.4.13.1. The user can see the list of recipients on this page. This list only shows recipients that the user has already added.

The search function lets users quickly find existing recipients.

Users can add a new recipient by either phone number (Figure 5.4.13.2) or email (Figure 5.4.13.3). A dropdown menu lets users choose the country code they want to use when entering a phone number. To search, the user must enter either a full phone number (Figure 5.4.13.4) or a valid email address (Figure 5.4.13.5) and then click the Search button. If the system finds a match, it will show the user; if not, it will show a message saying that the user can't be found.

When you add someone, they will show up in the recipient list (Figure 5.4.13.6), which is sorted by last name.

When the user picks a recipient, they can see more information (Figure 5.4.13.7), such as the recipient's name, phone number, email, country, and the date they were

added. Users can also change the recipient's display name as it appears in their own interface, and they can delete the recipient if they need to.

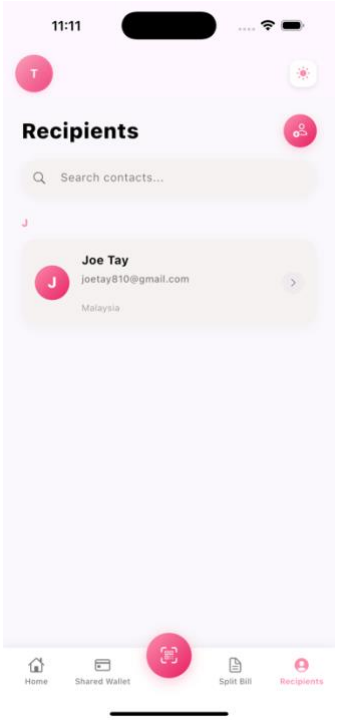

Figure 5.4.13.1 Recipients Page

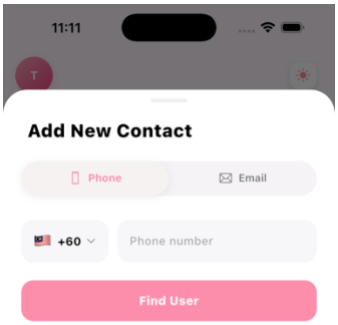

Figure 5.4.13.2 Add Recipient in Phone

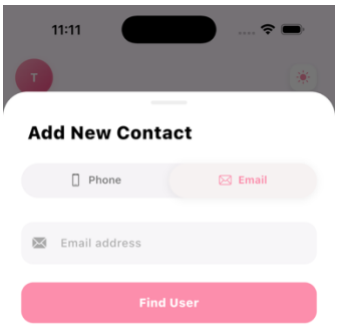

Figure 5.4.13.3 Add Recipient in Email

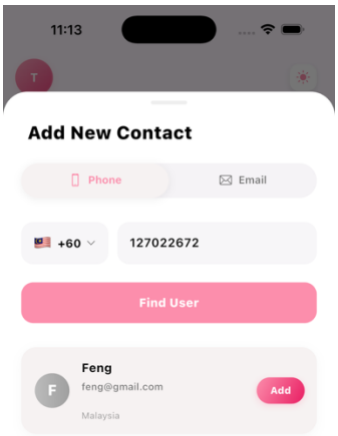

Figure 5.4.13.4 Result Search from Phone

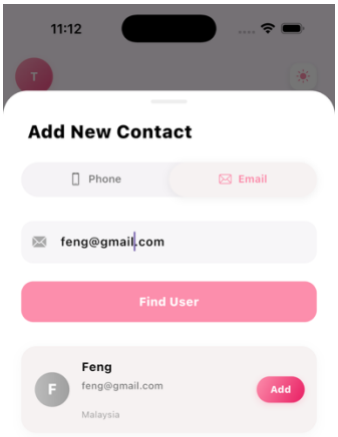

Figure 5.4.13.5 Result Search from Email

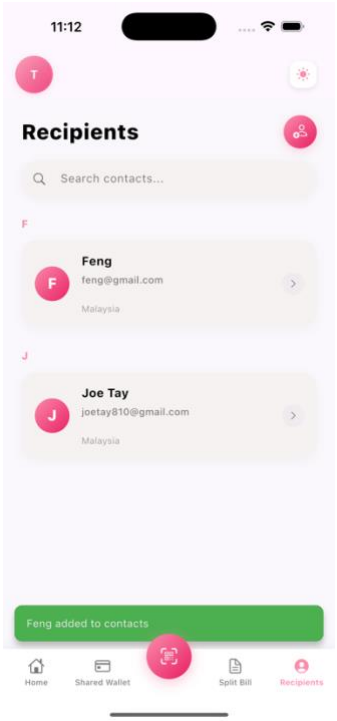

Figure 5.4.13.6 Successful Added Recipient

Figure 5.4.13.7 Recipient Details

5.4.14 Scan

When the user taps the scan icon button in the middle of the navigation bar, they will be taken to the page shown in Figure 5.4.14.1. The user can either scan a QR code from a merchant or a recipient directly on this page, or they can choose a QR code image from their photo gallery.

After the scan is successful, the user will be taken to the payment interface shown in Figure 5.4.14.2, where they can pay the person who sent the QR code. The user can choose from the currencies that are already in their wallet and type in the amount they want to pay. You can add or change a comment in the payment details section if you need to. If you don't, the default description is "Fund Transfer."

The user can tap "Send Money" once the details are correct. After that, the transaction needs to be verified with a PIN or Face ID. Figure 5.4.14.3 shows what the transaction success screen will look like after the verification is successful.

The scan page also has two buttons at the bottom: Pay and Receive.

Figure 5.4.14.4 shows the Pay function, which lets merchants scan the user's QR code to get paid. This QR code changes every 30 seconds to make it more secure.

The Receive function (Figure 5.4.14.5) shows the user's personal QR code, which they can then download and save. Users can also choose the "Enter Specific Amount" option to make a QR code that is unique to them. When the user chooses an option, they are asked to pick a currency from their available balance and enter an amount (Figure 5.4.14.6). When you tap "Generate," a QR code with the currency and amount you chose will be made (Figure 5.4.14.7). The payment information (currency and amount) is set in stone when someone else scans this QR code.

This QR code was made for security reasons and will stop working after 180 seconds.

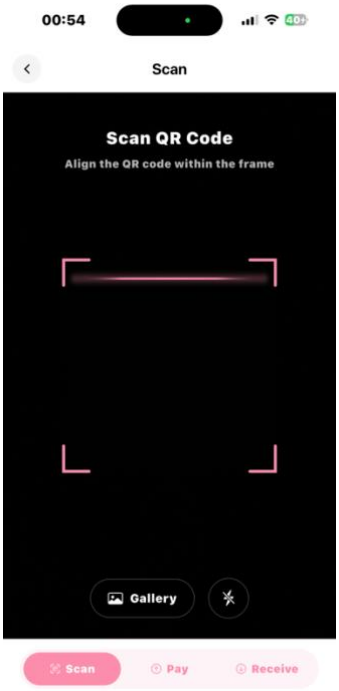

Figure 5.4.14.1 Scan Page

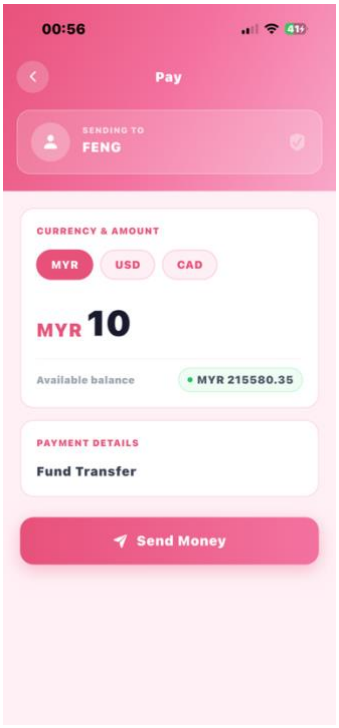

Figure 5.4.14.2 After Scan Recipient Page

Figure 5.4.14.3 Transfer Successful Page (Scan)

Figure 5.4.14.4 Pay Page

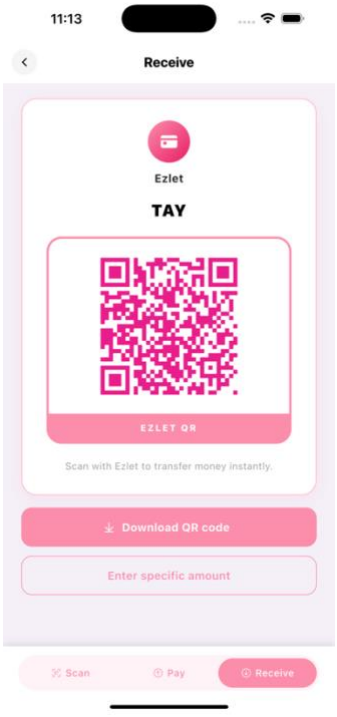

Figure 5.4.14.5 Receive Page

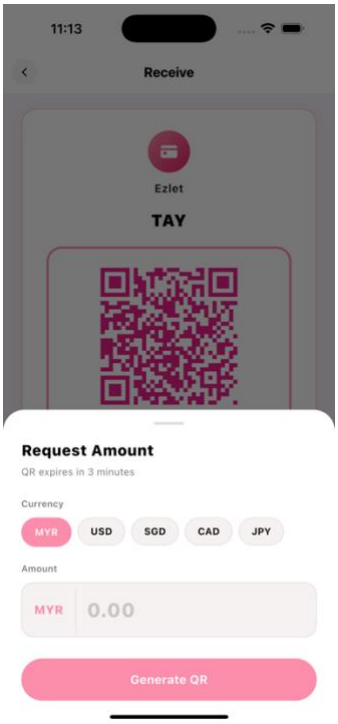

Figure 5.4.14.6 Create Specific Amount and Currency QR

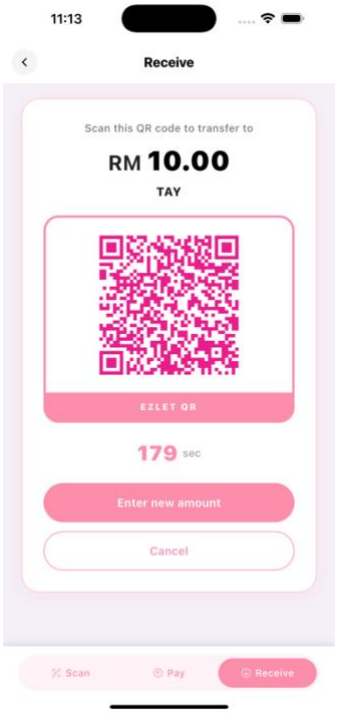

Figure 5.4.14.7 QR that Created in Specific Currency and Amount

5.4.15 Profile

Clicking the icon in the top-left corner of the home page will take the user to the profile page, as shown in Figure 5.4.15.1. This page shows the user their personal information, like their name and Gmail address.

There is an Edit button in the top right corner of the profile page. When the user clicks it, they will be taken to the edit profile page (Figure 5.4.15.2), where they can change their name and country. The email address and phone number fields, on the other hand, are locked and can't be changed. This is because they need to be verified and are used to make sure that any new email or phone number isn't already in the system. So, the only fields that can be changed in this part are the name and country.

The profile page also has a few useful sections. Users can manage their saved cards and bank accounts in the "Saved Cards & Banks" section of the Finance & Cards section. They can also set limits on how much money they can send and receive through the "Transfer Limits" option.

The "Security Settings" part of the Security & Settings section lets you change your password and make your account safer. The "Account Information" part lets you see general information about your account.

There is a Logout button at the bottom of the profile page. When the user clicks it, they will be logged out of the app. The next time they open the app, they will have to log in again with their email and password.

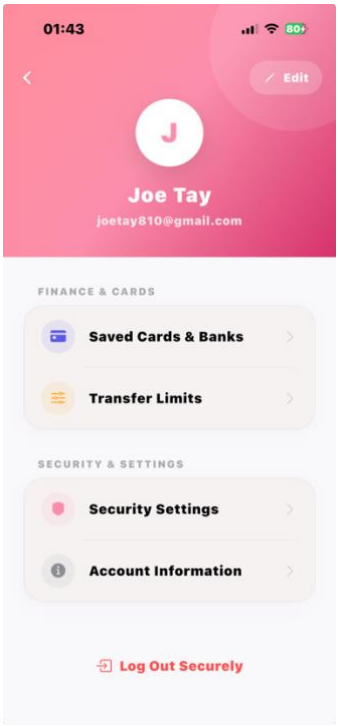

Figure 5.4.15.1 Profile Page

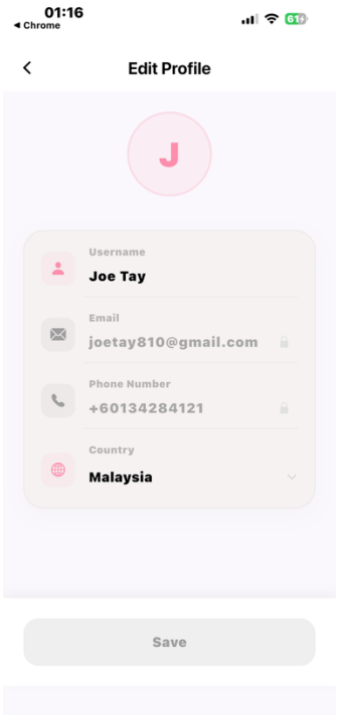

Figure 5.4.15.2 Edit Profile Page

5.4.16 Saved Cards & Banks

The page in Figure 5.4.16.1 will open when the user clicks on "Saved Cards & Banks." The user can see the card information that was added during the top-up process on this page.

As shown in Figure 5.4.16.2, the system shows the different currencies in which the user has previously added top-up cards at the top. This makes it easy for users to see which cards go with which currency.

As seen in Figure 5.4.16.3, the user can also add a new card to a certain currency category. When you add a new card, it will be saved under the currency you chose and will be an option for future top-up transactions.

Figure 5.4.16.1 Saved Cards Page

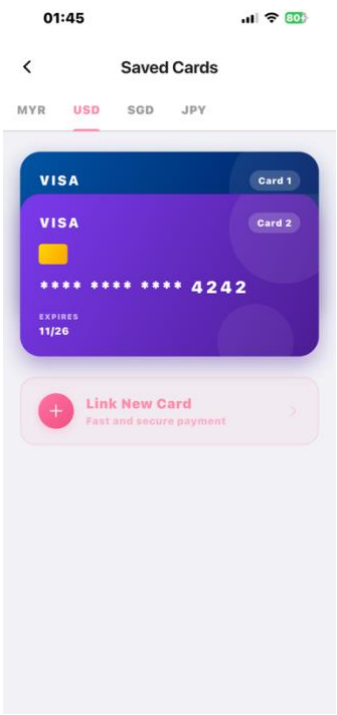

Figure 5.4.16.2 Saved Cards Page (Different Currency)

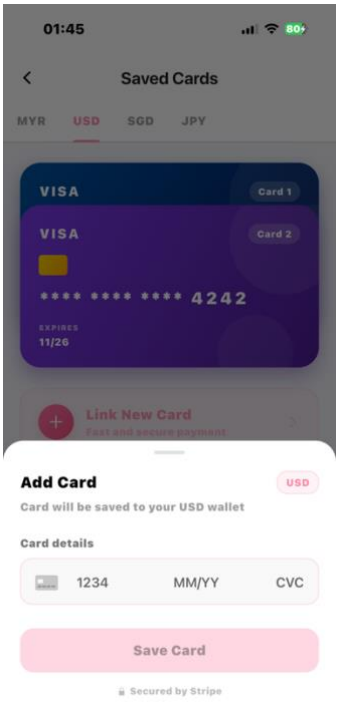

Figure 5.4.16.3 Add Cards Form

5.4.17 Transfer Limits

When the user clicks on "Transfer Limit," they will see two groups, as shown in Figure 5.4.17.1: Payment and Transfer. The system also says at the top of this page that any changes to the limits will not take effect for 12 hours.

When the user clicks on Payment, they will be taken to the page where they can set their payment limits (Figure 5.4.17.2). This part sets the limits based on the currencies that the user has in their account. The user can change both the daily payment limit and the limit for each payment. At first, these values are set to the system's default limits. Users can change them later to fit their needs and preferences.

Also, when the user clicks on Transfer, they will be taken to the page where they can set their transfer limits (Figure 5.4.17.3). This part also works with more than one currency and lets the user change the daily transfer limit and the per-transfer limit.

The main difference between these two functions is that Payment is for transactions that use QR codes, while Transfer is for manual transfers where the user types in an amount and chooses a recipient.

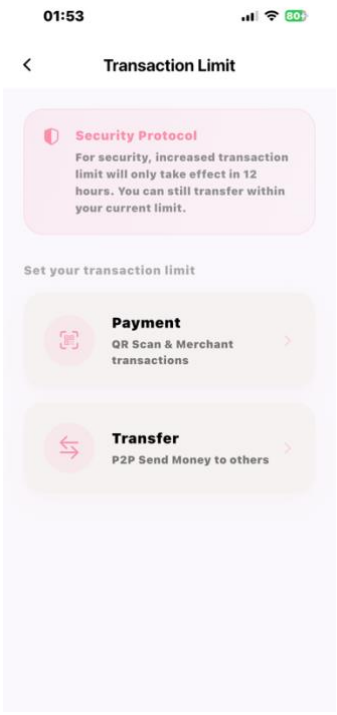

Figure 5.4.17.1 Transaction Limit Page

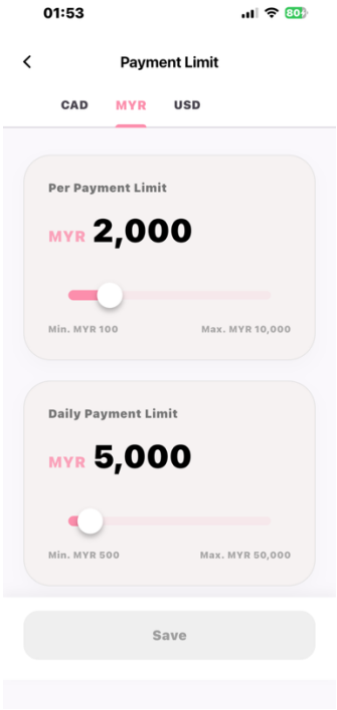

Figure 5.4.17.2 Payment Limit Page

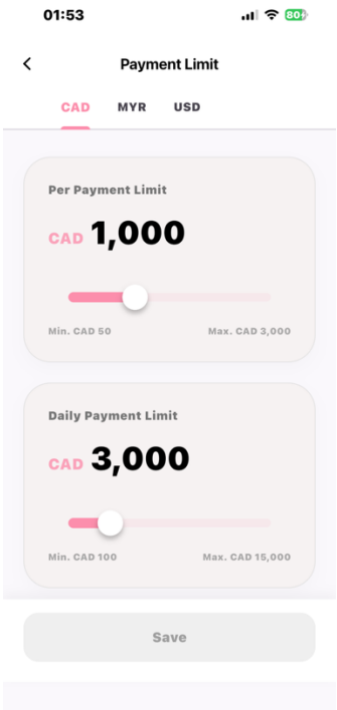

Figure 5.4.17.3 Payment Limit Page (Different Currency)

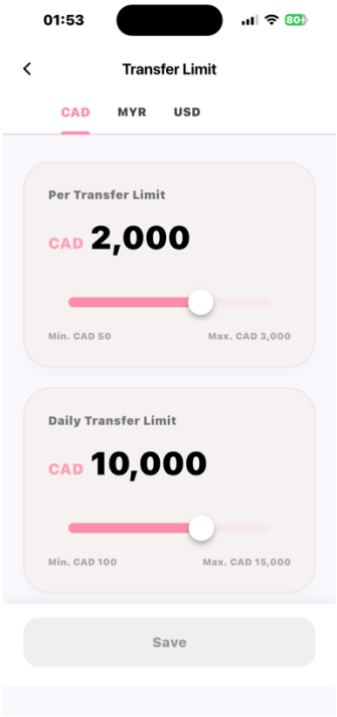

Figure 5.4.17.4 Transfer Limit Page

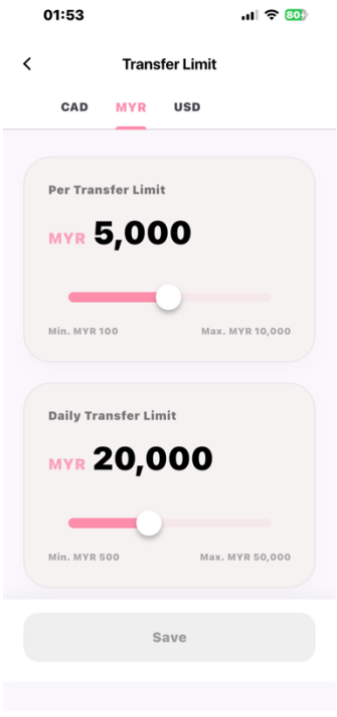

Figure 5.4.17.5 Transfer Limit Page (Different Currency)

5.4.18 Security & Settings

When the user clicks on "Security & Settings," they will be taken to the page shown in Figure 5.4.18.1. There are three main parts to this page: Account Verification, App Security, and Contact Information.

If the user has finished verification, the verification status will be shown in green in the Account Verification section. If the user hasn't been verified, they can only use one currency, which is the main currency of the app. Figure 5.4.18.2 shows that the interface is yellow and says "Verify Your Account" when it is not verified. When the user clicks on this option, they will be taken to the verification interface (Figure 5.4.18.3), which shows a message saying that the camera is ready. When the user is ready, they put their face in the frame and click "Start Face Scan." Facial scanning is what the system does next, as seen in Figure 5.4.18.4. If the verification is successful, the system will show a "Verified" status, as shown in Figure 5.4.18.5.

The user can change their password by clicking on "Change Password" in the App Security section. This will bring up the screen in Figure 5.4.18.6, where the user must enter a new password and then confirm it by entering it again. After you click "Confirm," the password will be changed and will be used for future logins.

"Change PIN" is another option in this section. When the user clicks on this option, they will be taken to Figure 5.4.18.7, where they must first enter their current PIN. After they have verified their identity, they will go to Figure 5.4.18.8 to set a new PIN and confirm it. The new PIN will be used for authentication in the future once it is finished.

The Security Authentication settings also let the user set their own authentication preferences. If you choose this option, the interface in Figure 5.4.18.9 will appear. There is a Biometric on Launch switch and a Payment Verification setting on this page. When Biometric on Launch is turned on, the user will have to use the device's built-in biometric system to recognize their face every time they open the app again (unless they logged out properly). When you choose the Payment Verification option, you will see Figure 5.4.18.10, which lets you choose between using a PIN number or Face ID to verify your identity. The chosen method will be used to check all transactions.

The user can change their phone number or email address in the Contact Information section. When the user clicks "Change Email," they will be taken to Figure 5.4.18.11, where they must type in a new email address. The system will first see if the email address is already in use. If not, a verification link will be sent to confirm ownership before the email is updated.

To change phone numbers, the user will be taken to Figure 5.4.18.12. As shown in Figure 5.4.18.13, the user must first choose the country code from a dropdown menu that lists all the countries the app supports. The system will check to see if the new phone number is already registered after you enter it. If it isn't, a one-time password (OTP) will be sent to the new number to check. The phone number will be changed after it has been successfully verified.

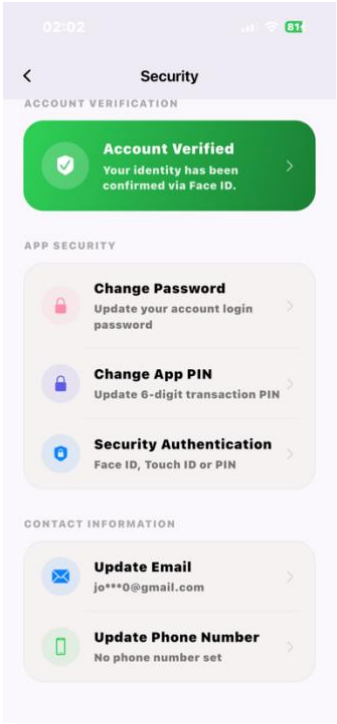

Figure 5.4.18.1 Security Page

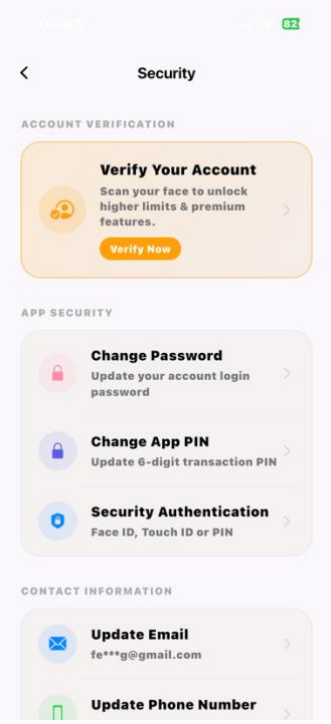

Figure 5.4.18.2 Security Page (Not Yet Verify Account)

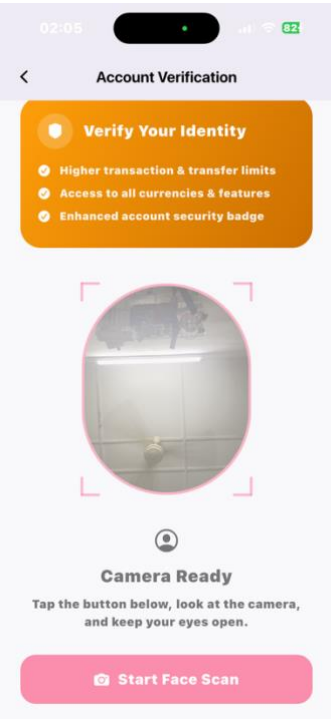

Figure 5.4.18.3 Account Verification Page (Not Yet Verify)

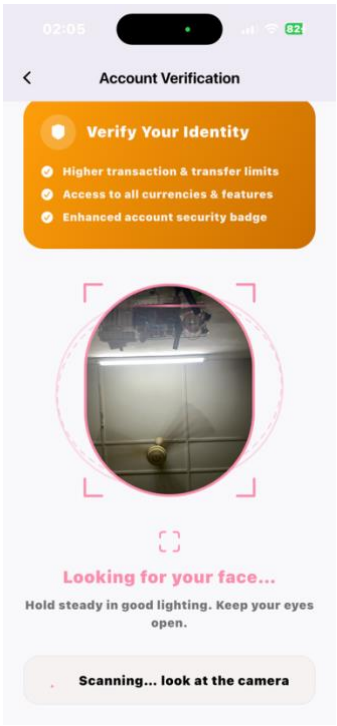

Figure 5.4.18.4 Verifying Function (Not Yet Verify)

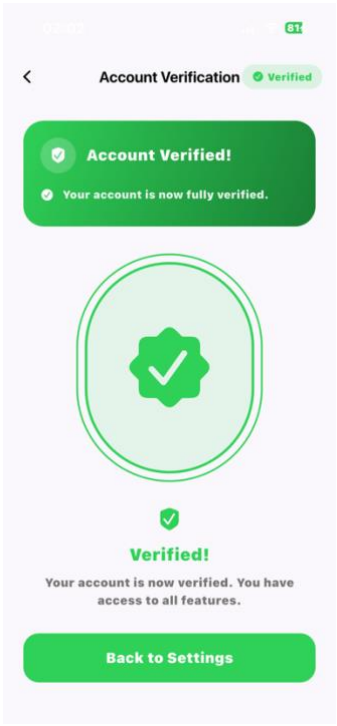

Figure 5.4.18.5 Verify Successful

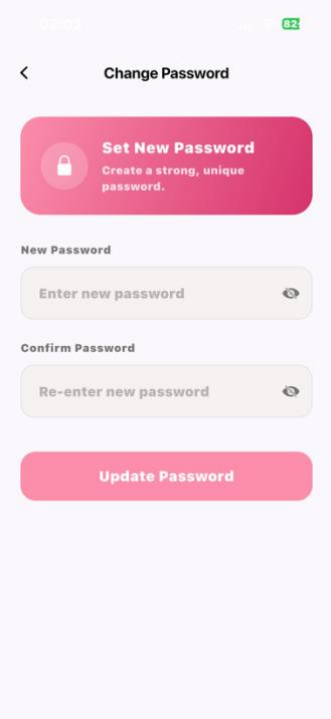

Figure 5.4.18.6 Change Password Page

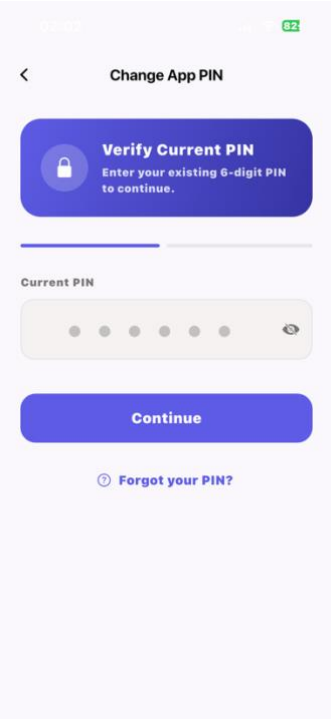

Figure 5.4.18.7 Change App PIN Page (Verify)

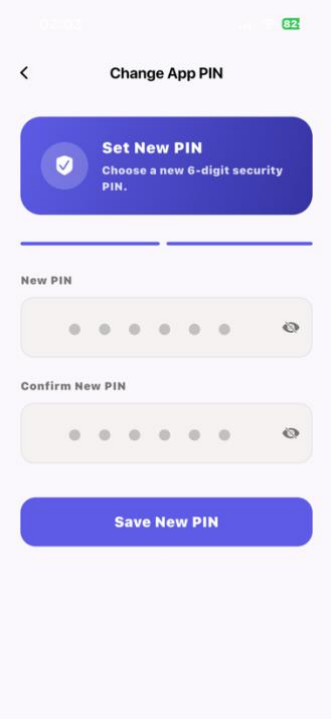

Figure 5.4.18.8 Change App PIN Page (Set PIN)

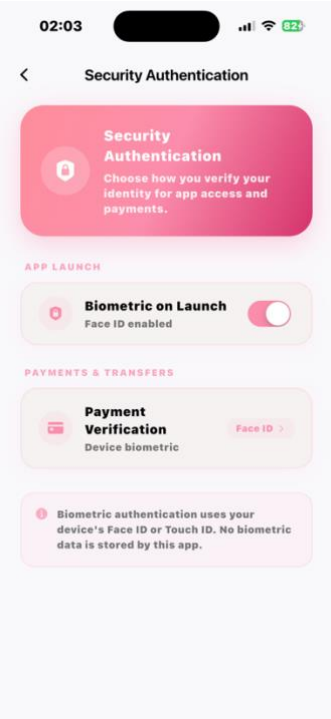

Figure 5.4.18.9 Security Authentication Page

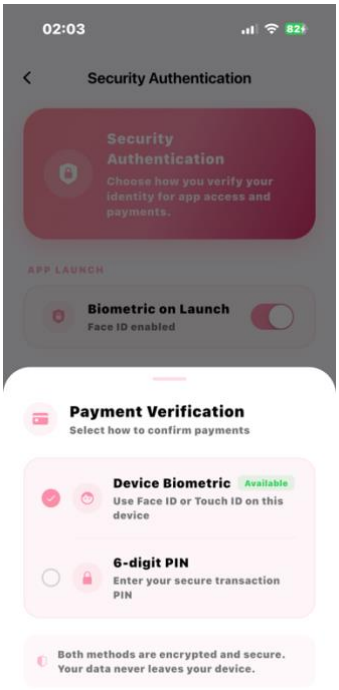

Figure 5.4.18.10 Option for Payment Verification

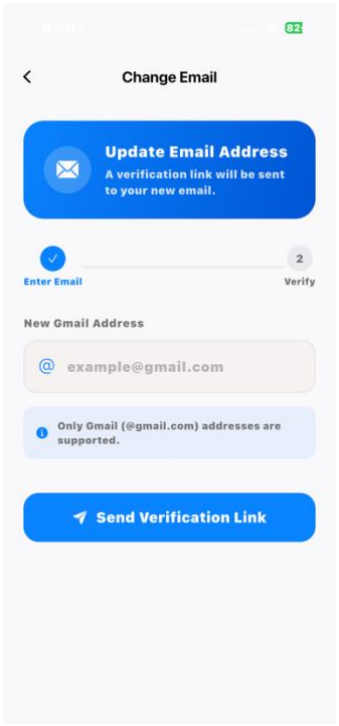

Figure 5.4.18.11 Change Email Page

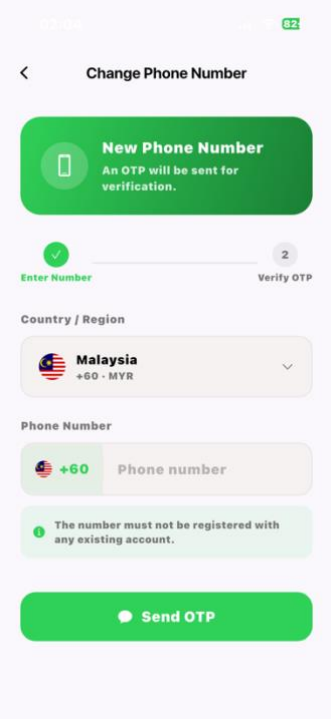

Figure 5.4.18.12 Change Phone Number Page

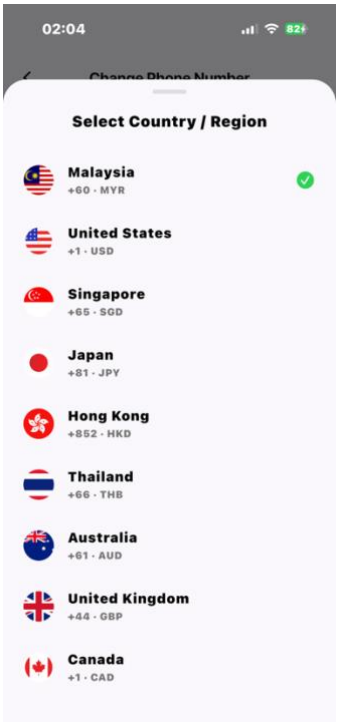

Figure 5.4.18.13 Phone Number Country Dropdown

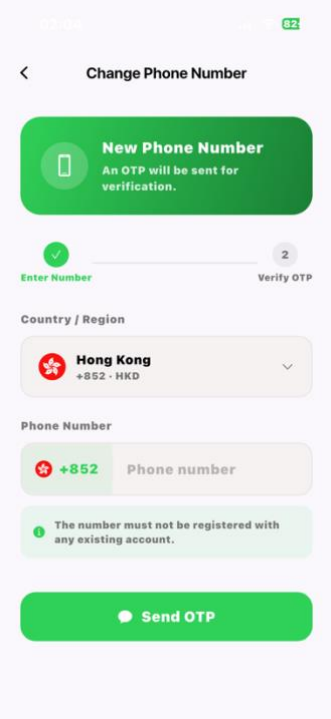

Figure 5.4.18.14 Change Phone Number Page (Selected Another Country Phone Number)

5.5 Implementation Issue and Challenges

5.5.1 Stripe Multi-Currency Webhook Routing

One of the hardest parts of implementing the project was making the Stripe payment integration work with nine different currencies. Stripe's test mode only lets each account process transactions in one currency, so the system needed nine separate Stripe test accounts—one for each currency—and nine separate Stripe instances to be set up at the same time, each with its own secret key, publishable key, and webhook signing secret.

The webhook endpoint was especially hard. Stripe sends a payment confirmation event to /api/stripe/webhook. The server must use a signing secret that is unique to each Stripe account to make sure the event is real. Because all nine accounts use the same webhook endpoint, the server can't tell which account sent the event ahead of time. The solution used was an iterative matching strategy. When a webhook request

came in, the handler went through all nine accounts and tried to verify the signature against each account's signing secret one at a time, accepting the first successful match. One important detail about the implementation was that verification exceptions that happen during failed iterations must be explicitly caught and suppressed. If they are allowed to propagate, the handler will stop working before a valid match is found.

The order of Express middleware made things even more complicated. To check the signature of a Stripe webhook, you need access to the raw, unparsed request body. When express.json() is used globally before the webhook route is registered, the request body is parsed and changed before verification, which makes all signature checks fail. Registering the webhook endpoint with Express fixed this.raw() is run before the global express.json() middleware, which makes sure that the raw request body stays the same for Stripe's verification process.

5.5.2 Duplicate Payment Processing Prevention

Stripe's official guidelines say that there is a race condition in the webhook handler that could cause Stripe to send the same payment confirmation event more than once. This would mean that a user's wallet balance would be credited multiple times for one top-up transaction if there were no deduplication mechanism. The problem was fixed by adding a two-stage idempotency check. Before crediting the wallet, the handler checks the user_transactions table for a record that matches the incoming payment_intent_id. To protect against multiple webhook deliveries coming in at the same time, a second identical check is done inside the database transaction block. The handler will only credit the wallet balance and finish the transaction if both checks show that there is no previous record.

5.5.3 ML Kit Liveness Detection Threshold Calibration

Google ML Kit Face Detection is used by the account verification feature to make sure that there is a real person with open eyes in the camera frame. The liveness check looks at two probability values that ML Kit gives back: leftEyeOpenProbability and rightEyeOpenProbability. We had to test these values in different lighting conditions and device orientations to find the right threshold. A threshold of 0.5 was found to cause false negatives in moderate indoor lighting, turning away real users

whose eyes looked partially closed in the camera frame. Lowering the threshold to 0.4 cut down on false rejections while still reliably finding eyes that were really closed. This was chosen as the final setting.

5.5.4 Claude API Response Parsing Instability

The Claude API is what the bill splitting module uses to get structured data from receipt images. The prompt template told the API to only return results as a JSON object with the merchant name, line items, quantities, unit prices, and tax values. But during testing, it was found that the API sometimes added explanatory text to the beginning of its response or wrapped the JSON output in markdown code fences, which made the raw response impossible to read with JSON.parse(). To fix this, the parse call was put in a try-catch block, and if it fails, the markdown formatting is removed before trying again. If parsing fails completely, the server sends back a structured error message that sends the user to manual bill entry. This keeps the system usable even when there isn't a clean API response.

5.5.5 Stripe Test Mode Currency Restriction

Because this project is using Stripe in test mode for development and testing, each Stripe account can only process transactions in the currency it is set up to do so. This means that a single Stripe account can't be used to top up in more than one currency. For example, a user can't top up in USD and have the system automatically credit a MYR wallet. To get around this problem, top-up operations can only be done with wallets that have the same currency as the Stripe account. After the top-up is done, the system's built-in currency conversion feature (UC-17 Convert Currency) handles moving funds between currencies.

Please keep in mind that this restriction only applies to Stripe's test mode. Stripe offers Connect and Currency Conversion features that let you make transactions in multiple currencies with a single account in a production environment. This would let the nine separate accounts be combined into a single payment architecture.

5.5.6 ExchangeRate API Call Limit and Absence of Historical Rate Queries

There were two related technical problems that came up while integrating the ExchangeRate API.

The first problem was that the free subscription tier only allowed a limited number of API calls. The system needs exchange rate data for nine different currencies. If the API were to be called every time a user requested something, like every time they visited the Transfer Calculator or changed currencies, the call quota would quickly run out. To fix this, a two-tier caching strategy was put in place. Exchange rate data is stored in memory for 30 minutes, so that repeated requests during that time don't have to make external API calls. A daily rate snapshot is also saved to the daily_currency_ratestable through a scheduled node-cron job, which cuts down on the number of live API requests needed over time.

The second problem was that the ExchangeRate API's free tier only lets you get the current live rate, not historical exchange rate data. The Transfer Calculator, on the other hand, needs a seven-day historical trend chart for each currency pair you choose. The system depends entirely on the daily cron job to collect rate snapshots over time because the API can't query historical rates. Because of this, the seven-day chart can only be fully filled out after the system has been running for at least seven days. Until then, the chart will only show the data points that are available without giving an error.

The design of the daily_currency_rates table in the database schema described in Section 4.3.1 was based on these two limitations.

5.6 Concluding Remark

This chapter has documented the entire implementation process of the proposed e-wallet application, including hardware and software configuration, integration of third-party services, system setup, and the technical challenges faced during development along with their corresponding solutions.

The system was finished following the four steps of incremental development that were explained in Chapter 3. All of the main functional modules are in place and working, such as user authentication and account management, managing a personal multi-currency wallet, working together on a shared wallet, AI-assisted group bill splitting, and the real-time exchange rate transfer calculator. It has been confirmed that integration across modules works as it should. This includes RESTful API communication between the Flutter client and the Node.js backend, data persistence through the MySQL database, and the integration of four external services: Stripe, Claude API, ExchangeRate API, and Firebase.

The technical problems that came up during development, such as routing multiple Stripe accounts through webhooks, stopping duplicate payments, calibrating the ML Kit liveness detection threshold, fixing the instability of parsing Claude API responses, and the ExchangeRate API call limit and lack of historical rate queries, were all solved with specific technical solutions, as shown in Section 5.5. Fixing these problems not only made sure that the affected features worked correctly, but it also made the system more fault-tolerant and resilient when it was running in less than ideal or unusual conditions.

The implementation work described in this chapter shows that the system methodology from Chapter 3 is possible and gives the system evaluation and testing in Chapter 6 a place to start.

CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION

6.1 System Testing and Performance Metrics

To make sure that all of the functional modules that were put in place work as they should, system testing was done. The testing method used was manual black-box functional testing. This meant that each feature was tested from the point of view of the end user by running defined test cases against the application without looking at how it was built. This method was chosen because it directly shows how the system works for the user and is a good way to check that functional workflows in a mobile app are working correctly.

Testing was done in small steps, following the four-increment development model from Section 3.2. After each increment was finished, all of the use cases in that increment, as well as any use cases that had already been implemented and depended on the new module, were run again to make sure that no regressions had been introduced. This made sure that integration between modules was tested all the time during development instead of just once at the end of the project.

Each test case had a unique identifier, a description of the scenario being tested, the conditions that had to be met before the test could be run, the steps that had to be followed during the test, the data that was used, the expected result, the actual result, and a pass or fail status. There were separate test cases for both the main flow and the defined alternative flows of each use case. This made sure that error handling and boundary conditions were checked along with the main happy-path scenarios.

The performance metric for this project is functional correctness, which means that each test case should give the expected result. Since the system is tested in a controlled development environment with a single test device and a backend server hosted locally, latency-based performance metrics like response time benchmarks are not the main way to judge it. We tested all API interactions with external services, such as Stripe, ExchangeRate API, Claude API, and Firebase, under normal network conditions using the test credentials set up in Section 5.3.

6.2 Testing Setup and Result

6.2.1 Register Account Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Register Account Module
Created By	Joc Tay Zhi Feng
Execute By	Joc Tay Zhi Feng
Date of Execution	2/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-Reg-01	Verify registration with valid new email and phone.	1. Launch App 2. Click on Register 3. Enter new email 4. Click on Send Verification 5. Click link in email 6. Enter new phone 7. Click Send OTP	User has no existing account	Email: new@test.com Phone: 134284121 OTP: 123456	Account successfully created.	Account successfully created.	Pass
TC-Reg-02	Register the account using email that is already registered.	1. Launch App 2. Click on Register 3. Enter email 4. Click on Send Verification	Email already registered	Email: new@test.com	Failed to get the email verification link. Display 'Email already registered.'	Failed to get the email verification link. Display 'Email already registered.'	Pass
TC-Reg-03	Register the account using a new email and registered phone.	1. Launch App 2. Click on Register 3. Enter new email 4. Click on Send Verification 5. Click link in email 6. Enter phone 7. Click Send OTP	1. Email is new 2. Phone already registered	Email: new123@test.com Phone: 134284121	Failed to get OTP. Display 'Phone already in use.'	Failed to get OTP. Display 'Phone already in use.'	Pass
TC-Reg-04	Verify OTP validation.	Enter invalid OTP	1. Email and phone not yet registered 2. OTP code: 123456	OTP: 234567	Registration interruption. Display 'Unvalid OTP'	Registration interruption. Display 'Unvalid OTP'	Pass
TC-Reg-05	Register the account with invalid password format.	1. Launch App 2. Click on Register 3. Enter new email 4. Click on Send Verification 5. Click link in email 6. Enter new phone 7. Click Send OTP	1. Email and phone not yet registered 2. Password in invalid format	Password: 1234	Failed to create the password. Display 'The password must be at least 8 characters.'	Display 'The password must be at least 8 characters.'	Pass

Figure 6.2.1 Test Case for Register Account Module

6.2.2 Login Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Login Module
Created By	Joc Tay Zhi Feng
Execute By	Joc Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-Login-01	To verify the login with valid email and password.	1. Launch App 2. Click on Log in 3. Enter email 4. Enter password 5. Click on Sign in to Eslet	1. Account has been created. 2. Email: jt@gmail.com 3. Password: User@1234	jt@gmail.com User@1234	Successful Login	Successful Login	Pass
TC-Login-02	To verify the login with unvalid email and valid password.	1. Launch App 2. Click on Log in 3. Enter email 4. Enter password 5. Click on Sign in to Eslet	1. Account has been created. 2. Email: jt@gmail.com 3. Password: User@1234	jt@gmail.com User@1234	Failed to Login.	1. Failed to Login. 2. Display 'No account found with this Email'.	Pass
TC-Login-03	To verify the login with valid email and invalid password.	1. Launch App 2. Click on Log in 3. Enter email 4. Enter password 5. Click on Sign in to Eslet	1. Account has been created. 2. Email: jt@gmail.com 3. Password: User@1234	jt@gmail.com User@12345	Failed to Login.	1. Failed to Login. 2. Display 'Invalid email or password. Please try again.'	Pass

Figure 6.2.2 Test Case for Login Module

6.2.3 Contact Management Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Contact Management Module
Created By	Joc Tay Zhi Feng
Execute By	Joc Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-CR-01	Add a registered user as contact by phone number.	1. Launch App 2. Login 3. Go to Recipient tab 4. Select Add Recipient 5. Choose phone number input 6. Enter phone number 7. Tap Search 8. Confirm add	1. Account logged in 2. Target user is a registered account	Phone: 0123456789 (registered)	Contact added and appears prioritised in recipient list alphabetically.	Contact added and appears prioritised in recipient list alphabetically.	Pass
TC-CR-02	Add contact with unregistered phone number.	1. Launch App 2. Login 3. Go to Recipient tab 4. Select Add Recipient 5. Enter unregistered phone number 6. Tap Search	1. Account logged in	Phone: 0000000000 (not registered)	Display error: User not found.	Display error: User not found.	Pass
TC-CR-03	Add a registered user as contact by email.	1. Launch App 2. Login 3. Go to Recipient tab 4. Select Add Recipient 5. Choos email input 6. Enter email 7. Tap Search 8. Confirm add	1. Account logged in 2. Target user is a registered account	Email: joetay810@gmail.com (registered)	Contact added and appears prioritised in recipient list alphabetically.	Contact added and appears prioritised in recipient list alphabetically.	Pass
TC-CR-04	Add contact with unregistered email.	1. Launch App 2. Login 3. Go to Recipient tab 4. Select Add Recipient 5. Enter unregistered email 6. Tap Search	1. Account logged in	Email: jj@gmail.com (not registered)	Display error: User not found.	Display error: User not found.	Pass
TC-CR-05	Add contact that is already in contact list.	1. Launch App 2. Login 3. Go to Recipient tab 4. Select Add Recipient 5. Enter already-added contact phone 6. Tap Search 7. Confirm add	1. Account logged in 2. Contact already exists in list	Phone: existing contact	System notifies user that contact is already added.	System notifies user that contact is already added.	Pass

Figure 6.2.3 Test Case for Contact Management Module

6.2.4 Account Verification Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Account Verification Module
Created By	Joc Tay Zhi Feng
Execute By	Joc Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-AV-01	Verify account with real face and both eyes open.	1. Launch App 2. Login 3. Go to Security Settings 4. Select Verify Account 5. Tap Start Face Scan 6. Hold face in front of camera with both eyes open	1. Account logged in 2. Account is not yet verified 3. Camera permission granted	User: real person, both eyes open	Liveness check passed. Account marked as verified. Multi-currency wallet creation unlocked.	Liveness check passed. Account marked as verified. Multi-currency wallet creation unlocked.	Pass
TC-AV-02	Verify account with eyes closed.	1. Launch App 2. Login 3. Go to Security Settings 4. Select Verify Account 5. Tap Start Face Scan 6. Close eyes in front of camera	1. Account logged in 2. Camera permission granted	User: eyes closed	Display error: 'Please open your eyes and look at the camera.' Remain in failed state.	Display error: 'Please open your eyes and look at the camera.' Remain in failed state.	Pass
TC-AV-03	Verify account with multiple faces in frame.	1. Launch App 2. Login 3. Go to Security Settings 4. Select Verify Account 5. Tap Start Face Scan 6. Two faces appear in camera frame	1. Account logged in 2. Camera permission granted	Two people in front of camera	Display error: 'Multiple faces detected. Please ensure only your face is visible.'	Display error: 'Multiple faces detected. Please ensure only your face is visible.'	Pass
TC-AV-04	Verify account without camera permission.	1. Launch App 2. Login 3. Go to Security Settings 4. Select Verify Account 5. Tap Start Face Scan (camera permission denied)	1. Account logged in 2. Camera permission: Denied	Camera permission: Denied	Display error. Remain in failed state. User prompted to grant permission.	Display error. Remain in failed state.	Pass

Figure 6.2.4 Test Case for Account Verification Module

6.2.5 Transaction PIN and Biometric Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Transaction PIN and Biometric Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-PIN-01	Change transaction PIN with matching confirmation.	1. Launch App 2. Login 3. Go to Security Settings 4. Select Change PIN 5. Enter current PIN 6. Enter new PIN 7. Confirm new PIN 8. Click Confirm	1. Account logged in 2. PIN previously configured	Current PIN: 123456 New PIN: 654321 Confirm PIN: 654321	PIN changed successfully. New PIN used for future transactions.	PIN changed successfully.	Pass
TC-PIN-02	Change transaction PIN with mismatched confirmation.	1. Launch App 2. Login 3. Go to Security Settings 4. Select Change PIN 5. Enter current PIN 6. Enter new PIN 7. Enter different confirm PIN	1. Account logged in	New PIN: 654321 Confirm PIN: 111111	Display error: PIN confirmation does not match. Prompt re-entry.	Display error: PIN confirmation does not match.	Pass
TC-PIN-03	Enable Face ID as alternative transaction authentication.	1. Launch App 2. Login 3. Go to Security Settings 4. Select Security Authentication 5. Enable Face ID	1. Account logged in 2. Device supports Face ID	Biometric: Face ID	Face ID registered as alternative to PIN for Transfer and Convert operations.	Face ID registered as alternative to PIN.	Pass
TC-PIN-04	Enter incorrect PIN for transaction 3 times (lockout).	1. Initiate a fund transfer 2. Enter wrong PIN 3 consecutive times	1. Account logged in 2. PIN configured	Wrong PIN: 000000 (x3)	Transaction locked after 3 incorrect attempts.	Transaction locked after 3 incorrect attempts.	Pass

Figure 6.2.5 Test Case for Transaction PIN and Biometric Module

6.2.6 Edit Profile Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Edit Profile Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-EP-01	Edit profile name and country successfully.	1. Launch App 2. Login 3. Go to Profile 4. Tap Edit 5. Change name and country 6. Submit	1. Account logged in	New name: John Doe New country: Singapore	Name and country updated. Success confirmation displayed.	Name and country updated. Success confirmation displayed.	Pass
TC-EP-02	Submit profile with empty name field.	1. Launch App 2. Login 3. Go to Profile 4. Tap Edit 5. Clear name field 6. Submit	1. Account logged in	Name: (empty)	Display validation error. Changes not saved.	Display validation error. Changes not saved.	Pass
TC-EP-03	Cancel profile edit without saving.	1. Launch App 2. Login 3. Go to Profile 4. Tap Edit 5. Modify fields 6. Tap Cancel	1. Account logged in	Action: Cancel	No changes saved. Profile remains unchanged.	No changes saved.	Pass

Figure 6.2.6 Test Case for Edit Profile Module

6.2.7 Logout Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Logout Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-LO-01	Logout and session cleared successfully.	1. Launch App 2. Login 3. Go to Profile 4. Tap Logout 5. Confirm	1. Account logged in with active session	Action: Confirm logout	Session token cleared. User redirected to login screen.	Session token cleared. User redirected to login screen.	Pass
TC-LO-02	Cancel logout confirmation.	1. Launch App 2. Login 3. Go to Profile 4. Tap Logout 5. Cancel	1. Account logged in	Action: Cancel logout	Session remains active. User stays on current screen.	Session remains active.	Pass

Figure 6.2.7 Test Case for Logout Module

6.2.8 Change PIN Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Change PIN Module
Created By	Joc Tay Zhi Feng
Execute By	Joc Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-CP-01	Change PIN with correct current PIN.	1. Launch App 2. Login 3. Security Settings 4. Change PIN 5. Enter current PIN 6. Enter new PIN 7. Confirm new PIN	1. Account logged in 2. PIN previously set	Current PIN: 123456 New PIN: 999999 Confirm: 999999	PIN changed successfully.	PIN changed successfully.	Pass
TC-CP-02	Change PIN with incorrect current PIN (max 3 attempts).	1. Launch App 2. Login 3. Security Settings 4. Change PIN 5. Enter wrong current PIN 3 times	1. Account logged in 2. PIN previously set	Current PIN: 000000 (wrong, x3)	Action temporarily locked after 3 failed attempts.	Action temporarily locked.	Pass
TC-CP-03	Change PIN with mismatched new PIN confirmation.	1. Launch App 2. Login 3. Security Settings 4. Change PIN 5. Enter correct current PIN 6. Enter new PIN 7. Enter different confirm PIN	1. Account logged in	New PIN: 111111 Confirm: 222222	Display error: PINs do not match. Prompt re-entry.	Display error: PINs do not match.	Pass

Figure 6.2.8 Test Case for Change PIN Module

6.2.9 Forgot Password Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Forgot Password Module
Created By	Joc Tay Zhi Feng
Execute By	Joc Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-FP-01	Set the new valid password for account that already registered.	1. Launch App 2. Click on Login 3. Click on 'Forgot Password?' 4. Enter email 5. Click send 'Reset link' 6. Click the link in email 7. Enter new password 8. Click 'Reset'	1. Account already registered 2. Email: test@gmail.com 3. Password: User@1234 4. Set new password in valid format	1. Email: test@gmail.com 2. New password: Test@1234	1. Password reset successful 2. User can login with new password.	1. Password reset successful 2. User can login with new password.	Pass
TC-FP-02	Set the new unvalid password for account that already registered.	1. Launch App 2. Click on Login 3. Click on 'Forgot Password?' 4. Enter email 5. Click send 'Reset link' 6. Click the link in email 7. Enter new password 8. Click 'Reset'	1. Account already registered 2. Email: test@gmail.com 3. Password: User@1234 4. Set new password in invalid format	1. Email: test@gmail.com 2. New password: 1111	Password reset unsuccessful.	Password reset unsuccessful.	Pass

Figure 6.2.9 Test Case for Forgot Password Module

6.2.10 Change Password Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Change Password Module
Created By	Joc Tay Zhi Feng
Execute By	Joc Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-CPW-01	Change password with valid current password.	1. Launch App 2. Login 3. Security Settings 4. Change Password 5. Enter current password 6. Enter new password 7. Confirm new password 8. Submit	1. Account logged in	Current: User@1234 New: NewPass@5678 Confirm: NewPass@5678	Password updated successfully.	Password updated successfully.	Pass
TC-CPW-02	Change password with incorrect current password.	1. Launch App 2. Login 3. Security Settings 4. Change Password 5. Enter wrong current password	1. Account logged in	Current: WrongPass@000	Display error: Incorrect current password.	Display error: Incorrect current password.	Pass
TC-CPW-03	Change password with mismatched new password confirmation.	1. Launch App 2. Login 3. Security Settings 4. Change Password 5. Enter correct current password 6. Enter new password 7. Enter different confirm password	1. Account logged in	New: NewPass@5678 Confirm: DiffPass@000	Display error: Passwords do not match.	Display error: Passwords do not match.	Pass

Figure 6.2.10 Test Case for Change Password Module

6.2.11 Change Email Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Change Email Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-CE-01	Change email to a new unique email and verify.	1. Launch App 2. Login 3. Security Settings 4. Change Email 5. Enter new email 6. Click Send Verification Link 7. Click link in email 8. Confirm	1. Account logged in 2. New email not already registered	New email: newuser@test.com	Verification link sent. Email updated upon confirmation.	Email updated upon confirmation.	Pass
TC-CE-02	Change email to an already registered email.	1. Launch App 2. Login 3. Security Settings 4. Change Email 5. Enter already-registered email	1. Account logged in 2. Entered email belongs to another account	New email: existing@test.com	Display error: Email already registered. Change rejected.	Display error: Email already registered.	Pass

Figure 6.2.11 Test Case for Change Email Module

6.2.12 Change Phone Number Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Change Phone Number Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-CPH-01	Change phone number to a new unique number with valid OTP.	1. Launch App 2. Login 3. Security Settings 4. Change Phone Number 5. Select country code 6. Enter new phone number 7. Enter received OTP	1. Account logged in 2. New phone not already registered 3. Firebase available	New phone: 0112345678 OTP: 123456	Phone number updated after OTP verified.	Phone number updated.	Pass
TC-CPH-02	Change phone number to an already registered number.	1. Launch App 2. Login 3. Security Settings 4. Change Phone Number 5. Enter already-registered phone	1. Account logged in 2. Entered phone belongs to another account	Phone: 0123456789 (registered)	Display error: Phone number already registered. Change rejected.	Display error: Phone already registered.	Pass
TC-CPH-03	Change phone number with invalid OTP.	1. Launch App 2. Login 3. Security Settings 4. Change Phone Number 5. Enter new phone 6. Enter wrong OTP	1. Account logged in 2. New phone not registered	OTP: 000000 (wrong)	Display error: Invalid OTP. Allow retry.	Display error: Invalid OTP.	Pass

Figure 6.2.12 Test Case for Change Phone Number Module

6.2.13 Add Currency Wallet and Top Up Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Add Currency Wallet and Top Up Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-AW-01	Verified user adds a new currency wallet.	1. Launch App 2. Login 3. Go to Wallets 4. Select Add Another Currency 5. Choose USD 6. Verify PIN	1. Account logged in 2. Account is verified 3. USD wallet not yet created	Currency: USD Account: Verified	USD wallet created and displayed in wallet list.	USD wallet created.	Pass
TC-AW-02	Unverified user attempts to add a new currency wallet.	1. Launch App 2. Login 3. Go to Wallets 4. Select Add Another Currency 5. Choose USD	1. Account logged in 2. Account is NOT verified	Currency: USD Account: Unverified	System blocks action. Prompt to complete account verification first.	System blocks and prompts verification.	Pass
TC-AW-03	Top up existing wallet with valid card via Stripe.	1. Launch App 2. Login 3. Select wallet 4. Tap Add Money 5. Enter amount 6. Select/add card 7. Confirm payment	1. Account logged in 2. Wallet exists 3. Stripe available	Amount: 100 MYR Card: valid test card	Wallet balance credited. Transaction record created.	Wallet balance credited.	Pass
TC-AW-04	Top up wallet with invalid card.	1. Launch App 2. Login 3. Select wallet 4. Tap Add Money 5. Enter amount 6. Enter invalid card 7. Confirm	1. Account logged in 2. Wallet exists	Amount: 100 Card: invalid	Payment fails. Wallet balance unchanged. Error displayed.	Payment fails. Error displayed.	Pass

Figure 6.2.13 Test Case for Add Currency Wallet and Top Up Module

6.2.14 Personal Wallet Transfer Limit Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Personal Wallet Transfer Limit Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-TL-01	Set valid per-transaction and daily transfer limit.	1. Launch App 2. Login 3. Go to Profile > Transfer Limits 4. Select Transfer 5. Enter per-transaction limit 6. Enter daily limit 7. Save	1. Account logged in 2. Currency wallet exists	Per-transaction: 500 MYR Daily: 2000 MYR	Limits saved. All future transfers validated against configured limits.	Limits saved and enforced.	Pass
TC-TL-02	Set per-transaction limit exceeding daily limit.	1. Launch App 2. Login 3. Go to Transfer Limits 4. Enter per-transaction limit > daily limit 5. Save	1. Account logged in	Per-transaction: 3000 MYR Daily: 1000 MYR	Display validation error: Per-transaction limit cannot exceed daily limit.	Display validation error.	Pass
TC-TL-03	Set limit with zero or negative value.	1. Launch App 2. Login 3. Go to Transfer Limits 4. Enter 0 or negative value 5. Save	1. Account logged in	Per-transaction: 0 Daily: -100	Display validation error. Input rejected.	Display validation error.	Pass

Figure 6.2.14 Test Case for Personal Wallet Transfer Limit Module

6.2.15 Transfer Funds Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Transfer Funds Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-TF-01	Transfer funds to registered recipient (same currency).	1. Launch App 2. Login 3. Tap Send 4. Select recipient 5. Select MYR wallet 6. Enter amount 7. Verify PIN	1. Account logged in 2. Sender has sufficient MYR balance 3. Recipient is registered	Amount: 50 MYR Recipient: registered user	Sender debited 50 MYR. Recipient credited 50 MYR. Transaction records created for both.	Both wallets updated. Transaction records created.	Pass
TC-TF-02	Transfer funds with insufficient balance.	1. Launch App 2. Login 3. Tap Send 4. Select recipient 5. Enter amount exceeding balance	1. Account logged in 2. Sender MYR balance = 10	Amount: 200 MYR Balance: 10 MYR	Display insufficient balance error. Transfer not processed.	Insufficient balance error displayed.	Pass
TC-TF-03	Transfer to non-existent recipient.	1. Launch App 2. Login 3. Tap Send 4. Search recipient by unregistered phone	1. Account logged in	Phone: 0000000000 (not registered)	Display error: Recipient not found.	Display error: Recipient not found.	Pass
TC-TF-04	Transfer with cross-currency conversion.	1. Launch App 2. Login 3. Tap Send 4. Select MYR as send currency 5. Select USD as recipient currency 6. Enter amount 7. Verify PIN	1. Account logged in 2. Sender has MYR balance 3. ExchangeRate API available	Send: 100 MYR Recipient currency: USD	MYR debited. Recipient credited equivalent USD. 2.5% fee applied.	MYR debited. USD credited. Fee applied.	Pass

Figure 6.2.15 Test Case for Transfer Funds Module

6.2.16 QR Code Payment Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	QR Code Payment Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-QR-01	Scan valid QR code and complete payment.	1. Launch App 2. Login 3. Tap Scan icon 4. Scan recipient QR code 5. Enter amount 6. Confirm 7. Verify PIN	1. Account logged in 2. Camera permission granted 3. Recipient has valid QR 4. Sender has sufficient balance	QR: valid Amount: 30 MYR	Sender wallet debited. Recipient wallet credited. Transaction records created.	Payment completed. Records created.	Pass
TC-QR-02	Scan invalid or unreadable QR code.	1. Launch App 2. Login 3. Tap Scan icon 4. Scan invalid QR code	1. Account logged in 2. Camera permission granted	QR: invalid/unreadable	Display error: QR code is invalid or unreadable.	Display error: Invalid QR.	Pass
TC-QR-03	QR payment with insufficient balance.	1. Launch App 2. Login 3. Scan QR 4. Enter amount exceeding wallet balance	1. Account logged in 2. Wallet balance insufficient	Amount: 500 MYR Balance: 10 MYR	Transaction cancelled. Insufficient balance error displayed.	Transaction cancelled. Error displayed.	Pass

Figure 6.2.16 Test Case for QR Code Payment Module

6.2.17 Currency Conversion Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Currency Conversion Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-CV-01	Convert currency between two active wallets successfully.	1. Launch App 2. Login 3. Open MYR wallet 4. Tap Convert 5. Select USD as target 6. Enter amount 7. Confirm 8. Verify PIN	1. Account logged in 2. MYR and USD wallets exist 3. ExchangeRate API available	Convert: 100 MYR to USD	MYR wallet debited. USD wallet credited at live rate. 2.5% fee deducted. Record created.	MYR debited. USD credited. Fee deducted.	Pass
TC-CV-02	Convert currency with insufficient balance.	1. Launch App 2. Login 3. Open MYR wallet 4. Tap Convert 5. Enter amount exceeding balance	1. Account logged in 2. MYR balance = 10	Convert: 500 MYR Balance: 10 MYR	Display Insufficient Balance error. Conversion not executed.	Insufficient balance error displayed.	Pass
TC-CV-03	Convert currency when ExchangeRate API is unavailable.	1. Launch App 2. Login 3. Open wallet 4. Tap Convert 5. API is unavailable	1. Account logged in 2. ExchangeRate API offline	API: unavailable	Display error. Conversion cancelled.	Display error. Conversion cancelled.	Pass

Figure 6.2.17 Test Case for Currency Conversion Module

6.2.18 Close Currency Wallet Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Close Currency Wallet Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-CW-01	Close non-main currency wallet with remaining balance.	1. Launch App 2. Login 3. Open USD wallet 4. Select Close Wallet 5. Review conversion summary 6. Confirm 7. Verify PIN	1. Account logged in 2. USD wallet exists with balance > 0 3. ExchangeRate API available	USD balance: 50 Exchange rate: live	USD balance converted to main currency at live rate minus 2.5% fee. Wallet deactivated.	Balance converted. Wallet deactivated.	Pass
TC-CW-02	Close non-main currency wallet with zero balance.	1. Launch App 2. Login 3. Open USD wallet 4. Select Close Wallet 5. Confirm	1. Account logged in 2. USD wallet balance = 0	USD balance: 0	Wallet closed immediately without conversion.	Wallet closed immediately.	Pass
TC-CW-03	Close wallet when ExchangeRate API is unavailable.	1. Launch App 2. Login 3. Open USD wallet 4. Select Close Wallet 5. API is unavailable	1. Account logged in 2. ExchangeRate API offline	API: unavailable USD balance: 20	Display error. Wallet closure cancelled.	Display error. Closure cancelled.	Pass

Figure 6.2.18 Test Case for Close Currency Wallet Module

6.2.19 Create Shared Wallet Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Create Shared Wallet Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-SW-01	Create shared wallet with valid details and invite members.	1. Launch App 2. Login 3. Go to Shared Wallet 4. Tap New 5. Select currency 6. Enter wallet name 7. Set transfer limits (optional) 8. Select members from contact list 9. Tap Create Wallet	1. Account logged in 2. At least one contact exists	Wallet name: Family Trip Currency: MYR Members: 2 contacts	Shared wallet created. Invitations sent to selected contacts. Wallet shown as pending.	Wallet created. Invitations sent.	Pass
TC-SW-02	Invited member accepts shared wallet invitation.	1. Login as invited member 2. Go to Shared Wallet 3. Select Invitations tab 4. Accept invitation	1. Member account logged in 2. Invitation received	Invitation: pending	Member joins wallet. Wallet becomes active and shown in My Wallets.	Member joined. Wallet active.	Pass
TC-SW-03	Invited member declines shared wallet invitation.	1. Login as invited member 2. Go to Shared Wallet 3. Select Invitations tab 4. Decline invitation	1. Member account logged in 2. Invitation received	Invitation: pending	Invitation declined. Wallet does not include this member.	Invitation declined.	Pass

Figure 6.2.19 Test Case for Create Shared Wallet Module

6.2.20 Shared Wallet Transaction Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Shared Wallet Transaction Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-SWT-01	Top up shared wallet from personal wallet.	1. Launch App 2. Login 3. Open shared wallet 4. Tap Top Up 5. Enter amount 6. Select personal wallet 7. Confirm	1. Account logged in and active member 2. Personal wallet has sufficient balance 3. Shared wallet is active	Top up: 100 MYR	Personal wallet debited. Shared wallet credited. Transaction visible to all members.	Shared wallet credited. All members can see record.	Pass
TC-SWT-02	Make QR payment from shared wallet.	1. Launch App 2. Login 3. Open shared wallet 4. Tap Pay 5. Scan merchant QR 6. Enter amount 7. Confirm	1. Account logged in and active member 2. Shared wallet has sufficient balance	Payment: 50 MYR via QR	Shared wallet debited. Transaction visible to all members.	Shared wallet debited. Record visible to all.	Pass
TC-SWT-03	Transaction exceeds shared wallet transfer limit.	1. Launch App 2. Login 3. Open shared wallet 4. Tap Pay 5. Enter amount exceeding limit	1. Account logged in 2. Shared wallet per-transaction limit = 100 MYR	Payment: 500 MYR Limit: 100 MYR	Transaction rejected. Applicable limit message displayed.	Transaction rejected. Limit message shown.	Pass

Figure 6.2.20 Test Case for Shared Wallet Transaction Module

6.2.21 Close Shared Wallet Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Close Shared Wallet Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-SWC-01	Owner initiates closure with equal split and all members approve.	1. Login as Owner 2. Open shared wallet 3. Tap Close Wallet 4. Select Equal Split 5. Tap Send Close Request 6. All members approve via Review & Vote	1. Owner logged in 2. Shared wallet active 3. All members approve	Distribution: Equal Split All members: Approve	Funds distributed equally to all members. Wallet closed and removed from list.	Funds distributed. Wallet closed.	Pass
TC-SWC-02	Member rejects wallet closure proposal.	1. Login as Owner 2. Open shared wallet 3. Send Close Request 4. One member rejects	1. Owner logged in 2. Shared wallet active	One member: Reject	Closure cancelled. Wallet remains active. Owner notified of rejection with reason.	Closure cancelled. Owner notified.	Pass
TC-SWC-03	Owner cancels closure request before all votes.	1. Login as Owner 2. Open shared wallet 3. Send Close Request 4. Owner selects Cancel Request	1. Owner logged in 2. Close request pending	Action: Cancel request	Closure request cancelled. Wallet becomes active again.	Closure request cancelled.	Pass

Figure 6.2.21 Test Case for Close Shared Wallet Module

6.2.22 Create Bill Manual Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Create Bill Manual Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-BM-01	Create bill manually with valid details, items, tax, and recipients.	1. Launch App 2. Login 3. Go to Split Bill 4. Tap Add 5. Select Manual Entry 6. Select currency 7. Enter merchant name, items, quantity, price, tax 8. Select personal items 9. Select recipients 10. Confirm	1. Account logged in 2. At least one contact exists 3. At least one active wallet	Merchant: Restaurant A Items: 3 items Tax: 6% Recipients: 2 contacts	Bill created. Recipients notified. Creator not charged.	Bill created. Recipients notified.	Pass
TC-BM-02	Create bill manually with no tax (0%).	1. Launch App 2. Login 3. Go to Split Bill 4. Tap Add 5. Manual Entry 6. Enter items with tax = 0 7. Select recipients 8. Confirm	1. Account logged in	Tax: 0%	Bill created without tax charge. Recipients notified.	Bill created. No tax charged.	Pass
TC-BM-03	Create bill where selected recipient has no wallet in chosen currency.	1. Launch App 2. Login 3. Go to Split Bill 4. Manual Entry 5. Select SGD as currency 6. Select recipient with no SGD wallet	1. Account logged in 2. Recipient has no SGD wallet	Currency: SGD Recipient: no SGD wallet	System notifies creator that recipient does not have a wallet in selected currency.	Creator notified of recipient wallet mismatch.	Pass

Figure 6.2.22 Test Case for Create Bill Manual Module

6.2.23 Create Bill AI Receipt Scan Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Create Bill AI Receipt Scan Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-BA-01	Scan receipt and AI extracts all data successfully.	1. Launch App 2. Login 3. Go to Split Bill 4. Tap Add 5. Select Scan Receipt 6. Take clear photo of receipt 7. Wait for extraction 8. Review pre-populated form 9. Select personal items 10. Select recipients 11. Confirm	1. Account logged in 2. Camera permission granted 3. Claude API available	Receipt: clear photo with merchant, items, prices, tax	Merchant name, items, quantities, prices, tax extracted and pre-populated. Bill created after confirmation.	All data extracted. Bill created.	Pass
TC-BA-02	Scan receipt when Claude API is unavailable.	1. Launch App 2. Login 3. Go to Split Bill 4. Tap Add 5. Select Scan Receipt 6. Claude API unavailable	1. Account logged in 2. Claude API offline	API: unavailable	System prompts user to switch to manual entry.	User redirected to manual entry.	Pass
TC-BA-03	Scan receipt with partial extraction (blurry image).	1. Launch App 2. Login 3. Go to Split Bill 4. Tap Add 5. Scan Receipt 6. Upload blurry receipt	1. Account logged in 2. Claude API available	Receipt: partially readable	System pre-populates available fields. Unextracted fields marked for manual completion.	Partial fields pre-populated. Rest flagged for manual input.	Pass

Figure 6.2.23 Test Case for Create Bill AI Receipt Scan Module

6.2.24 Settle Bill Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Settle Bill Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-BS-01	Recipient selects items and settles bill successfully.	1. Login as recipient 2. Go to Split Bill 3. Open unpaid bill 4. Select consumed items 5. Review computed amount 6. Select payment wallet 7. Confirm payment	1. Recipient logged in 2. Bill received 3. Recipient wallet has sufficient balance	Items: 2 items selected Tax: included	Recipient wallet debited. Creator wallet credited. Bill status updated. Creator notified.	Recipient debited. Creator credited. Bill updated.	Pass
TC-BS-02	Recipient settles bill with insufficient balance.	1. Login as recipient 2. Open unpaid bill 3. Select items 4. Confirm payment 5. Wallet insufficient	1. Recipient wallet balance < amount owed	Amount owed: 50 MYR Balance: 10 MYR	Display error: Insufficient balance. Prompt to top up.	Error displayed. Payment blocked.	Pass
TC-BS-03	Creator cancels bill when no payment made.	1. Login as creator 2. Go to My Bills 3. Select open bill 4. Delete bill	1. Creator logged in 2. No payments made yet	Bill: 0 payments	Bill deleted immediately.	Bill deleted immediately.	Pass
TC-BS-04	Creator cancels bill after some recipients already paid.	1. Login as creator 2. Go to My Bills 3. Select open bill with payments 4. Delete bill	1. Creator logged in 2. At least one recipient has paid	Bill: 1 payment made	Warning pop-up shown. Bill cancelled. Paid recipients automatically refunded.	Warning pop-up shown. Bill cancelled. Paid recipients automatically refunded.	Pass

Figure 6.2.24 Test Case for Settle Bill Module

6.2.25 Transfer Calculator Module

Project name	E-Wallet application: Shared Wallet and Group Bill Splitting features
Module name	Transfer Calculator Module
Created By	Joe Tay Zhi Feng
Execute By	Joe Tay Zhi Feng
Date of Execution	3/5/26

Test Case ID	Test Case Description	Test Steps	Preconditions	Test Data	Expected Result	Actual Result	Status
TC-TC-01	View real-time exchange rate and 7-day historical chart.	1. Launch App 2. Login 3. Go to Transfer Calculator on Home 4. Select MYR as source 5. Select USD as target	1. Account logged in 2. ExchangeRate API available	Source: MYR Target: USD	Current exchange rate displayed. 7-day historical line chart rendered.	Exchange rate and chart displayed.	Pass
TC-TC-02	Calculate estimated converted amount.	1. Launch App 2. Login 3. Go to Transfer Calculator 4. Select MYR to USD 5. Enter amount 100	1. Account logged in 2. ExchangeRate API available	Source: MYR Target: USD Amount: 100	Estimated converted amount displayed at current rate. No transaction executed.	Conversion estimate shown. No transaction executed.	Pass
TC-TC-03	View Transfer Calculator when API is unavailable.	1. Launch App 2. Login 3. Go to Transfer Calculator 4. API unavailable	1. Account logged in 2. ExchangeRate API offline	API: unavailable	Cached rate displayed with staleness warning.	Cached rate shown with staleness warning.	Pass

Figure 6.2.25 Test Case for Transfer Calculator Module

6.3 Project Challenges

6.3.1 Multi-Currency Framework Complexity

One of the hardest parts of the project was designing and implementing support for nine different international currencies. In a multi-currency system, each currency must have its own wallet balances, transaction records, and transfer limits. The system must also make sure that the logic for handling currencies is the same across all functional modules, such as currency conversion, bill splitting, and shared wallet management. For a project with only one developer, keeping track of nine different currency settings at the same time, testing all possible interactions between currencies, and making sure that every functional module works correctly with all supported currencies took a lot more work than we had planned. This problem directly affected the development schedule because currency-related edge cases in several modules needed to be debugged multiple times before the correct behaviour could be confirmed.

6.3.2 Stripe Test Account Configuration

Stripe's test mode only lets each account use one currency, as explained in Section 5.4.5. This means that you need to set up and register nine separate Stripe test accounts, one for each currency that Stripe supports. Setting up each account required separate steps, such as getting the API key, registering the webhook endpoint, and testing the integration locally. This meant that a lot of time was spent on configuring the payment infrastructure compared to its functional scope. Also, since Stripe doesn't let you move money directly between currency accounts, we had to add extra logic to route cross-currency fund movement through the system's internal currency conversion feature. To make sure it was working correctly, we had to check all nine currency combinations very carefully.

6.3.3 Third-Party API Availability Dependency

The system's most important parts depend on three external API services: the ExchangeRate API for real-time and historical exchange rate data, the Claude API for AI-powered receipt parsing, and Stripe for adding money to a card-based wallet. If any of these services went down or were stopped, the features that depended on them would no longer work. The ExchangeRate API and Claude API both have fallback

mechanisms in place, such as showing cached rates and allowing users to enter bills manually, as explained in Section 5.4. However, these are only temporary fixes and not full replacements for the APIs. In a production deployment, this risk would be mitigated by incorporating secondary API providers or creating an autonomous exchange rate data service to remove reliance on a single external supplier.

6.3.4 Time Management

Throughout the development lifecycle, it was always hard to manage development time well on a project of this size. The system has five functional modules, four development increments, many third-party service integrations, and a complete security architecture, which makes it hard to accurately estimate how long it will take to build. Stripe multi-account configuration, multi-currency logic debugging, and ML Kit threshold calibration all took a lot longer than planned, which put pressure on the development schedule for the next modules. To handle this, development priorities were changed on the fly based on how modules depended on each other. This made sure that core infrastructure components like authentication and wallet management stayed stable before higher-level features like shared wallet collaboration and AI-assisted bill splitting were added.

6.3.5 Learning Curve of Flutter and Mobile Application Development

This was the developer's first time making a mobile app with the Flutter framework. Before starting the project, the team learned about Dart as a programming language, Flutter's widget-based rendering model, stateful widget lifecycle management, and mobile-specific issues like handling device permissions, encrypted local storage, and native iOS integration through platform channels. It was harder to stick to the development schedule because the team had to learn a new framework while also delivering a feature-complete application on time for school. Integrating the local_auth package for iOS Face ID, setting up the google_mlkit_face_detection package for on-device liveness detection, and managing asynchronous state updates across multiple interconnected screens were all areas where I had to learn a lot on my own.

6.3.6 UI/UX Design Without Prior User Feedback

As a project with only one developer and no separate design phase or user research data, all decisions about UI and UX design were based on the developer's own ideas about how easy it is to use, how clear it is visually, and how logical the navigation flow is. Throughout development, it was hard to make an interface that was both visually clean, easy to use, and in line with the rules for financial applications, where trust and clarity are very important. During implementation, decisions about layout hierarchy, color scheme, navigation patterns, and information density were changed many times. However, without formal user testing or feedback collection, it is still unclear if the final design meets all of the needs and wants of the intended users. As discussed in Chapter 7, this limitation is recognized as an area for future enhancement.

6.3.7 Multi-User Testing Constraints

Some parts of the system, like shared wallet collaboration, peer-to-peer fund transfer, bill splitting, and inviting group members, are inherently multi-user features that need at least two accounts to be active at the same time to be properly tested. Because only the developer did the testing, multi-user scenarios were created by logging into different accounts one after the other on the same device or by keeping two accounts open at the same time when the application state allowed it. While this method is good enough to check that each transaction flow is correct, it doesn't fully replicate interactions between multiple users at the same time. Due to this, edge cases that happen when multiple users are doing things at the same time, like two members adding money to a shared wallet at the same time, couldn't be fully tested in the testing environment.

6.4 Objectives Evaluation

This part looks at how well the implemented system has met the three main goals of the project that were set out in Section 1.5.

Project Objective 1:

Main Objective:

To develop a mobile e-wallet application that facilitates group financial management

Sub Objectives:

1. Develop a Shared Wallet feature that lets more than one person use it to improve financial collaboration and let group users easily and openly manage shared expenses. (Fully)
2. Set up real-time tracking and recording of expenses to cut down on mistakes and make sure everyone in the group can see how much they are spending and contributing. (Fully)
3. Develop a dynamic notification system that sends out timely updates on shared wallet activities and financial status to make things easier to use and more open. (Fully)
4. Develop a settlement mechanism that kicks in when a shared wallet is closed. This mechanism should offer residual settlement (distribution of the remaining balance only) and require user agreement on the chosen distribution strategy. (Fully)
5. Develop a contact and recipient management feature that lets users add registered contacts. These contacts will then be given priority when choosing members of a shared wallet and when splitting bills to make group financial activities go more smoothly. (Fully)

The first main goal has been fully reached. The Shared Wallet module (UC-19 to UC-21) lets more than one person create, manage, and close a shared wallet. They can also set limits on how much money can be transferred each day and per transaction. All

members can see real-time transaction records for every top-up or payment, which makes sure that everyone knows exactly how much the group is spending. A notification system sends timely alerts to all relevant members when there are wallet activities, member invitations, or requests to close the wallet. The consensus-based wallet closure mechanism needs all members to agree before it can be used. It supports four ways to divide funds: equal split, full transfer, proportional by contribution, and smart distribution based on contribution history. The contact management feature (UC-03) keeps a list of prioritized recipients that is always visible during shared wallet member selection and bill splitting workflows. This makes it easier to coordinate group finances.

Project Objective 2:

Main Objective:
To enable group bill splitting for social activities
Sub-Objectives:

1. Develop a feature that lets users take pictures or upload physical and electronic receipts and automatically extracts merchant names, item details, quantities, prices, and taxes using the Claude API. (Fully)
2. Develop a flexible manual bill creation function that can handle complicated or personalized cost-sharing situations in a variety of social settings. (Fully)
3. Develop a precise bill allocation confirmation system that lets each recipient choose the items they used and automatically calculates the amount each person owes, including taxes. The bill creator has already paid the full amount on behalf of the group. (Fully)
4. Implement an automatic debit and transfer module that takes the agreed-upon amount out of the recipient's e-wallet and sends it to the bill initiator's account right away after payment confirmation. (Fully)

The second main goal has been completely met. The Bill Splitting module (UC-22 to UC-24) lets you create bills by hand or scan receipts with AI using the Claude API. When you create a bill by hand, you can enter the merchant's information, the quantity and unit price of each line item, and the tax rate that applies. You can also choose not to charge the recipient for items that you personally consumed. The AI receipt scanning

flow automatically pulls out and fills in the merchant name, line items, quantities, unit prices, and tax from a picture of a receipt. If the Claude API is down or returns an unparseable response, it will gracefully fall back to manual entry. Each recipient chooses what they want to eat, and the system automatically figures out how much they owe, including the right amount of tax. Once payment is confirmed, the system automatically takes money out of the recipient's wallet and puts it into the bill creator's wallet. This completes the full settlement cycle in one application flow.

Project Objective 3:

Main Objective:

To develop a wallet that supports internationalization.

Sub-Objectives:

1. Add support for multiple currencies so that users can create and use wallets in nine different international currencies to meet the needs of users who travel a lot. (Fully)
2. Create a currency transfer calculator that shows real-time exchange rates and a seven-day historical trend line chart. This will let users figure out how much they will need to convert before they make a transaction. (Fully)
3. Develop a currency wallet management system that makes separate currency wallets for each supported currency. This will make it clear how much money is in each wallet and avoid confusion when switching between currencies. (Fully)
4. Develop transaction features that let you exchange currencies, add money to your wallet through Stripe, and send money to other people at any time. This will make managing international funds more flexible and easy. (Fully)

The third main goal has been completely reached. The system supports nine international currencies: MYR, USD, SGD, JPY, HKD, THB, AUD, GBP, and CAD. Each currency is kept in its own wallet with its own balance, so it's easy to see how much money you have in each currency. The Transfer Calculator (UC-25) shows real-time exchange rates and a seven-day historical trend line chart powered by the ExchangeRate API. This lets users guess what the conversion will be before they make a transaction. Currency wallet management (UC-13, UC-18) lets you make more currency wallets for verified accounts and close non-main wallets. The balance will

automatically be converted to the main currency. All supported currencies can do full transactions, such as Stripe-integrated card top-ups, peer-to-peer transfers, QR code payments, and live-rate currency conversion with PIN or Face ID authorization.

6.5 Concluding Remark

The implemented system has fully met all three main goals and their related sub-goals. The proposed e-wallet app successfully creates a single mobile platform that combines managing individual wallets, working together on a group shared wallet, AI-assisted bill splitting, and managing finances in multiple currencies. This directly addresses the issue of fragmented multi-application workflows that was identified as the main problem in Section 1.2. The problems with implementation that were listed in Section 6.3 were fixed without affecting the functional completeness of any module. The system works as planned for all twenty-five use cases.

CHAPTER 7 CONCLUSION AND RECOMMENDATION

7.1 Conclusion

The goal of this project was to create a mobile e-wallet app that fixes three major problems with current digital payment platforms: it doesn't have built-in tools for managing group finances, it doesn't have advanced bill-splitting features for social activities, and it doesn't have enough support for managing finances in multiple currencies. During the project, a fully working mobile app was made with Flutter on the front end and Node.js with MySQL on the back end. It used four external services—Stripe, ExchangeRate API, Claude API, and Firebase—to create a financial platform that had all the features it needed and was safe.

We met all three of the main goals set out in Section 1.5. The first goal, which was to make it easier for groups to manage their money, was met by the Shared Wallet module. This module lets multiple users create, manage, and close a shared wallet with customizable transfer limits, real-time transaction visibility, a consensus-based closure mechanism, and four ways to distribute funds. The Bill Splitting module, which allows for both manual bill creation and AI-powered receipt scanning via the Claude API, with item-level allocation, proportional tax computation, and automatic in-app payment settlement, helped achieve the second goal of allowing groups to split bills for social activities. The third goal was to make a wallet that works with multiple currencies. This was done by creating a nine-currency wallet architecture with live exchange rate integration, a seven-day historical trend chart in the Transfer Calculator, and full transaction functionality for all supported currencies.

The project not only achieved functional completeness but also provided several contributions of greater importance to the field of mobile financial technology. The integrated platform combines managing individual wallets, group shared wallets, and AI-assisted bill splitting into one app. This gets rid of the separate, multi-application workflows that are common in current solutions. The shared wallet architecture adds features that no other commercial platform has, such as democratic closure consensus, configurable transfer limits, and data-driven fund distribution strategies. The AI-assisted receipt parsing pipeline takes structured billing information from photos and turns it into structured data, which makes it easier for end users to enter data by hand.

The multi-layer security architecture uses bcrypt password hashing, Firebase OTP verification, Google ML Kit liveness detection, and iOS native Face ID through local_auth to make sure that both account-level and transaction-level security are protected by separate but complementary methods.

The process of making the product also gave us a lot of chances to learn. The developer's first mobile app project using the Flutter framework required them to learn Dart programming, Flutter widget lifecycle management, iOS platform integration, and mobile-specific security practices all at once. In Chapter 3, we used the incremental development model, which worked well for managing the complexity of a multi-module system. This made sure that the core infrastructure stayed stable before we added more advanced features. We made the system more stable and fault-tolerant by making specific engineering decisions to fix each of the technical problems we ran into, such as Stripe multi-account webhook routing, preventing duplicate payments, calibrating the ML Kit threshold, fixing the Claude API response parsing instability, and fixing the ExchangeRate API limitations.

In short, the proposed e-wallet app shows that a single mobile platform that combines personal wallet management, group finance, AI-assisted bill splitting, and support for multiple currencies is both technically possible and useful in an academic project setting. The system directly addresses the structural fragmentation identified in the literature review and provides a comprehensive solution for the collaborative, secure, and cross-border financial management requirements of contemporary users.

7.2 Recommendation

The system has fully met its goals, but there are still areas that need to be improved and expanded for future development. The suggestions below deal with both the known problems with the current implementation and the chances for adding new features that would make the platform even more useful and profitable.

7.2.1 Migration to Production-Grade Stripe Integration

As explained in Section 5.4.5, the current Stripe integration is in test mode and has nine separate accounts set up for different currencies. In a production deployment, this architecture should be changed to a single Stripe account that uses Stripe Connect and Stripe's built-in Currency Conversion features. These features support transactions in multiple currencies and automatic exchange within a single account structure. This would make setting things up much easier, get rid of the nine-account webhook routing overhead, and make the payment system more scalable and easier to maintain.

7.2.2 Integration of Redundant API Providers

There are three external API services that the system's core features depend on: ExchangeRate API, Claude API, and Stripe. This makes the system vulnerable to single-point dependency risks, as discussed in Section 6.3.3. It is a good idea to add secondary API providers as backup options for each important service. Open Exchange Rates and Fixer.io are two other places you could get exchange rate data from. For receipt parsing, a secondary AI model or an on-device OCR solution could be looked at as a backup for the Claude API. Adding a provider-agnostic service abstraction layer to the Node.js backend would make it easy to switch between providers without having to change the client-side application.

7.2.3 Extension to Android Platform

The current implementation is only for iOS, as the project scope says it should be. Since Flutter natively supports cross-platform deployment, adding Android support to the app would greatly increase the number of people who could use it with only a small amount of extra development work. When moving to Android, some important

platform-specific things to think about are replacing the local_auth iOS Face ID integration with Android Biometric API compatibility, changing the Google ML Kit configuration for Android deployment, and doing thorough testing on a wide range of Android device models and OS versions.

7.2.4 Formal User Testing and UI/UX Refinement

As stated in Section 6.3.6, decisions about UI and UX design were made without any formal user research or feedback collection. It is suggested that future development include organized usability testing with a representative group of target users, using methods like think-aloud protocol testing, measuring the task completion rate, and the System Usability Scale (SUS) evaluation. The results of this kind of testing should help improve the application's navigation structure, information hierarchy, and visual design over time so that they better match what users expect and what is standard for financial applications.

7.2.5 Expansion of Supported Currency Portfolio

The current system supports nine international currencies that were chosen because they are useful to Malaysian and ASEAN users. In the future, this portfolio could grow to include more currencies that are important in the region, like the Chinese Yuan (CNY), Euro (EUR), Indonesian Rupiah (IDR), and Vietnamese Dong (VND). This would make the platform even more useful for people who do cross-border transactions within ASEAN. To add support for more currencies, you would need to register a Stripe account in production mode, set up more exchange rate APIs, and update the schema to include the new currency codes.

7.2.6 Cryptocurrency Support

The project scope makes it clear that the current system only works with fiat currency and not cryptocurrency. As more and more young people who are active online start using cryptocurrencies, adding support for popular ones like Bitcoin (BTC) and Ethereum (ETH) would make the platform much more appealing. This would necessitate the integration with a cryptocurrency exchange API, the establishment of a crypto wallet architecture that is separate from the current fiat wallet model, and

adherence to the applicable regulatory standards governing digital asset transactions in the intended market.

7.2.7 Web Platform Extension

The current version only works with a mobile app interface. A companion web application would let users manage their wallets, see their transaction history, and split bills from their desktops. This is especially useful for shared wallet management and bill creation, which may be easier to do on a bigger screen. You could use a JavaScript framework like React or Vue.js to build a web frontend that works with the existing Node.js RESTful API. This wouldn't require many changes to the backend because the API contract is already separate from the Flutter client.

7.2.8 Implementation of Formal KYC Verification

The current account verification system uses Google ML Kit's liveness detection to make sure that a real person is present with their eyes open, as explained in UC-04. This is a good way to stop automated account creation, but it doesn't follow the formal Know Your Customer (KYC) process that most financial regulatory frameworks require. It is best to add government-issued identity document verification, like MyKad for Malaysian users, to the liveness detection for a production deployment. This can be done through a third-party KYC provider like Jumio or Onfido. Formal KYC compliance would be required to run the platform as a licensed financial service provider, and it would greatly improve user trust and the platform's standing with regulators.

7.2.9 Enhanced Multi-User Concurrent Testing

As stated in Section 6.3.7, it was not possible to fully test multi-user concurrent scenarios in the testing environment for a single developer. In the future, automated load testing should be a part of development. This can be done with tools like Apache JMeter or k6 to simulate multiple users interacting with the Node.js backend at the same time. This is especially important for shared wallet operations, where race conditions can happen when two people try to top up or pay at the same time. Automated test suites that cover the whole use case matrix should also be made to replace the manual black-

box testing method used in this project. This will improve regression coverage and cut down on the time needed for validation at each development increment.

REFERENCES

[1] Statista, "Digital Payments – Worldwide: Market Forecast," Statista, Hamburg, Germany, 2024. [Online]. Available: https://www.statista.com/outlook/dmo/fintech/digital-payments/worldwide. [Accessed: Apr. 2025].

[2] Statista, "Number of digital wallet users worldwide from 2020 to 2025," Statista, Hamburg, Germany, 2024. [Online]. Available: https://www.statista.com/statistics/1228757/digital-wallet-users-worldwide. [Accessed: Apr. 2025].

[3] World Bank, *The Global Findex Database 2022: Measuring Financial Inclusion and the Fintech Revolution*. Washington, DC: World Bank Group, 2022. [Online]. Available: https://www.worldbank.org/en/publication/globalfindex. [Accessed: Apr. 2025].

[4] Google, Temasek, and Bain & Company, *e-Conomy SEA 2023: The Resilience of the Digital Decade*, 2023. [Online]. Available: https://economysea.withgoogle.com. [Accessed: Apr. 2025].

[5] T. Dahlberg, N. Mallat, J. Ondrus, and A. Zmijewska, "Past, present and future of mobile payments research: a literature review," *Electron. Commer. Res. Appl.*, vol. 7, no. 2, pp. 165–181, 2008, doi: 10.1016/j.elerap.2007.09.003.

[6] Bank Negara Malaysia, *Payment Systems Report 2022*. Kuala Lumpur: Bank Negara Malaysia, 2023. [Online]. Available: https://www.bnm.gov.my/publications/psr2022. [Accessed: Apr. 2025].

[7] D. Chawla and H. Joshi, "Consumer attitude and intention to adopt mobile wallet in India – an empirical study," *Int. J. Bank Mark.*, vol. 37, no. 7, pp. 1590–1618, 2019, doi: 10.1108/IJBM-06-2018-0158.

[8] Department of Statistics Malaysia (DOSM), *Tourism Statistics 2022*. Putrajaya: DOSM, 2023. [Online]. Available: https://www.dosm.gov.my/portal-main/release-content/tourism-statistics-2022. [Accessed: Apr. 2025].

[9] M. Salmony and B. Harald, "Extending access to the financial system: new approaches to account access," *J. Payments Strategy Syst.*, vol. 4, no. 2, pp. 166–174, 2010.

[10] S. Gupta and N. Arora, "Investigating consumer intention to accept mobile payment systems through unified theory of acceptance model," *South Asian J. Bus. Stud.*, vol. 6, no. 3, pp. 289–311, 2017, doi: 10.1108/SAJBS-08-2016-0073.

[11] H. E. Riquelme and R. E. Rios, "The moderating effect of gender in the adoption of mobile banking," *Int. J. Bank Mark.*, vol. 28, no. 5, pp. 328–341, 2010, doi: 10.1108/02652321011064872.

[12] Z. Huang et al., "ICDAR 2019 Competition on Scanned Receipt OCR and Information Extraction," in *Proc. Int. Conf. Document Analysis and Recognition (ICDAR)*, 2019, pp. 1516–1520.

[13] S. Schuh and J. Stavins, "Why are (some) consumers (finally) writing fewer checks? The role of payment characteristics," *J. Bank. Finance*, vol. 34, no. 8, pp. 1745–1758, 2010, doi: 10.1016/j.jbankfin.2010.03.001.

[14] Asian Development Bank (ADB), *Accelerating Digital Finance in ASEAN: Key Insights from Financial Sector Landscape*. Manila: ADB, 2022. [Online]. Available: https://www.adb.org/publications/digital-finance-asean. [Accessed: Apr. 2025].

[15] C. Frankenfield, "Digital wallet," *Investopedia*, 2023. [Online]. Available: https://www.investopedia.com/terms/d/digital-wallet.asp. [Accessed: Apr. 2025].

[16] C. Kim, M. Mirusmonov, and I. Lee, "An empirical examination of factors influencing the intention to use mobile payment," *Comput. Hum. Behav.*, vol. 26, no. 3, pp. 310–322, 2010, doi: 10.1016/j.chb.2009.10.013.

[17] S. Singh, "Emergence of payment systems in the age of electronic commerce: the state of art," *Global J. Int. Bus. Res.*, vol. 2, no. 2, pp. 17–36, 2009.

[18] J. Bonneau, C. Herley, P. C. van Oorschot, and F. Stajano, "The quest to replace passwords: a framework for comparative evaluation of web authentication schemes," in *Proc. IEEE Symp. Security Privacy*, San Francisco, CA, USA, 2012, pp. 553–567, doi: 10.1109/SP.2012.44.

[19] D. S. Evans and R. Schmalensee, *Paying with Plastic: The Digital Revolution in Buying and Borrowing*, 2nd ed. Cambridge, MA: MIT Press, 2005.

[20] N26 GmbH, "N26 Shared Spaces," N26. [Online]. Available: https://n26.com/en-eu/shared-spaces. [Accessed: Apr. 2025].

[21] T. Wilkinson and B. Young, "On cooperating: firms, relations and networks," *J. Bus. Ind. Mark.*, vol. 17, no. 2/3, pp. 123–135, 2002, doi: 10.1108/08858620210419210.

[22] J. Ondrus and Y. Pigneur, "Towards a holistic analysis of mobile payments: a multiple perspectives approach," *Electron. Commer. Res. Appl.*, vol. 5, no. 3, pp. 246–257, 2006, doi: 10.1016/j.elerap.2005.09.003.

[23] BudgetBakers s.r.o., "Wallet by BudgetBakers," BudgetBakers. [Online]. Available: https://budgetbakers.com. [Accessed: Apr. 2025].

[24] A. J. B. Brush and K. M. Inkpen, "Yours, mine and ours? Sharing and use of technology in domestic environments," in *Proc. UbiComp 2007*, Innsbruck, Austria, 2007, pp. 109–126, doi: 10.1007/978-3-540-74853-3_7.

[25] Splitwise Inc., "How Splitwise works," Splitwise, Providence, RI, USA. [Online]. Available: https://www.splitwise.com/how-it-works. [Accessed: Apr. 2025].

[26] T. Zhou, Y. Lu, and B. Wang, "Integrating TTF and UTAUT to explain mobile banking user adoption," *Comput. Hum. Behav.*, vol. 26, no. 4, pp. 760–767, 2010, doi: 10.1016/j.chb.2010.01.013.

[27] Tricount SAS, "Tricount – Split bills & expenses," Tricount. [Online]. Available: https://tricount.com. [Accessed: Apr. 2025].

[28] Revolut Ltd., "Revolut features," Revolut. [Online]. Available: https://www.revolut.com/features. [Accessed: Apr. 2025].

[29] PayPal Inc., "PayPal features," PayPal. [Online]. Available: https://www.paypal.com/us/webapps/mpp/home. [Accessed: Apr. 2025].

[30] Stripe Inc., "Stripe security and PCI compliance," Stripe. [Online]. Available: https://stripe.com/docs/security. [Accessed: Apr. 2025].

POSTER


UTAR
UNIVERSITI TUNKU ABDUL RAHMAN
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY
Development of E-Wallet application:
Shared Wallet and Group Bill Splitting features
Introduction
Most e-wallets focus only on individual payments and transfers, lacking group financial management, bill-splitting, and multi-currency support. This project develops an integrated e-wallet to improve fairness, transparency, and convenience in shared financial activities.
OBJECTIVE
To develop a wallet that supports internationalization.
To develop a mobile e-wallet application that facilitates group financial management
To enable group bill splitting for social activities
Methodology
Incremental development method
Flutter
Node.js REST API
MySQL
Firebase
Visual Studio Code
Stripe API, ExchangeRate API
Project Scope
Core wallet functions
Multi-currency support
Shared wallet
Group Bill Splitting
Project Developer: Joe Tay Zhi Feng
Project Supervisor: Mr. Phan Koo Yuen