SMART FINANCIAL TRACKING MOBILE APPLICATION

By

CHANG KOK SHEN

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)
Faculty of Information and Communication Technology
(Kampar Campus)

FEB 2025

COPYRIGHT STATEMENT

This Final Year Project report is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) 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, Dr Tan Joi San who has been guiding me and advising me throughout the development of my final year project. She is a wonderful mentor that has provided a lot of valuable and constructive feedback to me throughout these few semesters. I am particularly grateful for her trust and confidence in my abilities during moments of self-doubt. For me she will always serve a special place in my heart as the best FYP supervisor in UTAR. Additionally, I want to extend my gratitude to my moderator Mr Tan Chiang Kang @ Thang Chiang Kang, for moderating and evaluating my work. He is the current lead lecturer for the course UCCD3223 Mobile Applications Development, his teachings provided me with fundamental mobile application knowledge which I have built upon and applied to develop the more advanced features in my Final Year Project. Not to mention, both supervisor and moderator play an important role in providing feedback to improve the existing weaknesses of my mobile application.

I would also like to thank all my family members for their continuous support throughout the journey of my Bachelor of Computer Science (Honours) degree.

Lastly, I would like to thank all my close friends and teammates who have supported me throughout my time studying in UTAR. Without their mental support each semester, I could not have made it this far. I am sincerely grateful for that unsung friend / assignment teammate, Lim Chia Yoong, who has supported me throughout these three years. The countless all-nighters and late-night meetings to discuss assignments with her will always stick to me as a core memory. I sincerely wish her all the best in her future endeavours after graduation.

ABSTRACT

In this project, a Smart Financial Tracking Mobile Application is proposed to cater to all individuals who care about being out of debt, spending, budgeting, and financial tracking in Malaysia. Financial tracking, also known as expense tracking, is a common approach whereby an individual manages their expenses by recording their daily, monthly, and yearly expenditure through digital software such as Microsoft Excel, cross-platform budget tracking applications like the popular You Need A Budget (YNAB) or through traditional financial entries on notebooks. Most financial tracking systems have limitations, such as a lack of Asian Bank Integration, insufficient financial data insight, and mundane financial entry. The proposed system solves the common mobility issue for users who want to track their finances on the go and aims to solve the problems mentioned earlier. Moreover, it also leverages state-of-the-art AI technology, such as a seamless Integration with Google's newest Machine Learning Kit models for near real-time receipt extraction and various connections with third-party APIs such as LangChain API, to improve data extraction accuracy. The core features of the mobile application include receipt scanning with OCR, scraping email financial data, chatting with financial data leveraging Large Language Models (LLM) like OpenAI's ChatGPT, Malaysian bank app integration (Maybank, CIMB, Public Bank) and voice data recognition entry. Furthermore, the main Software Development Life Cycle (SDLC) model used in this project is Rapid Application Development (RAD). This approach enables quick creation of multiple prototype versions that can be refined based on user feedback. Lastly, the main tools used for development are IDE, such as Android Studio and Visual Studio Code, Firebase as the backend as a service, a mobile phone, and a laptop.

Area of Study: Mobile Application Development, Artificial Intelligence
Keywords: Financial Technology integration, Optical Character Recognition, Voice Recognition, Expense Tracking, Generative AI, Workflow Automation

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

YNAB	You Need A Budget
SDLC	Software Development Lifecycle
PDF	Portable Document Format
RAD	Rapid Application Development
AI	Artificial Intelligence
OCR	Optical Character Recognition
RAG	Retrieval Augmented Generation
API	Application Programming Interface
JSON	JavaScript Object Notation
LLM	Large Language Model
HNSW	Hierarchical Navigable Small World

CHAPTER 1 Introduction

1.1 Overview

In the recent advancement of technology, Artificial Intelligence (AI) has been dominating every industry including finance, education, marketing, e-commerce and so much more. AI's capabilities are vast and varied especially in the fintech sector, usually AI will aid fintech companies in cost-saving, improved customer experience, and better analytics [1]. However, most end users would not have the chance to experience these innovative AI driven financial technologies. To bridge this gap, smaller-scale solutions such as mobile and web applications are developed, to make these innovations more accessible to a broader audience.

Additionally, smaller scaled systems such as loan calculators, financial planners and budget trackers can assist end user's financial decisions. To note, the existence of these specific mobile applications cannot replicate the final financial decision of the user but guides their decision-making process. Financial tracking or also known as expense tracking is another common use case. It involves the process of documenting daily, monthly, and annual expenses. Likewise, the expense tracker can involve the process of automated data entry from receipts, bank accounts, and credit cards [2]. Some individuals maintain detailed financial records to claim tax reductions at year-end. Overtime, a well recorded financial planner will give us a clear picture of our cash flow and might allow us to forecast/predict future expenditures [3].

In the past, expenses were manually recorded in logbooks, and accountants played a vital role in calculating large arithmetic figures by hand, often leading to human error [4]. The introduction of Microsoft Excel in the 1980s revolutionised financial tracking by partially automating manual data entry, a practice which is still common today. Excel can organise data in rows and columns, but it introduces issues like data duplication and redundancy.

Moreover, recording financial transactions in Excel is often seen as a temporary fix for a long-term problem. One major drawback of using Excel for expense tracking is the lack of mobility, as users couldn't record transactions immediately after a purchase. They had to wait until they were home to enter the data. The rise of mobile applications in 2008 addressed this issue by providing lightweight, multifunctional

systems on users' phones, allowing them to record expenses on the spot and enhancing the utility of mobile devices beyond their original purpose.

In this report, a mobile application is proposed to solve the issue of mobility particularly when user seeks to record down their transactions. Not to mention, the mobile application is intended to provide an interface to show transactions in an auto categorised format and lastly focusing on the automation and seamless transition for users to record down their transaction details.

1.2 Problem Statement and Motivation

Many individuals like students, working professionals and even working entities struggle with maintaining a clear layout of their financial stance, which might lead to overspending, insufficient savings and difficulty in achieving financial goals. The development of a robust financial tracker is needed for them to maintain a healthy financial status. However, the current existing financial tracking mobile applications and methods pose a few potential issues. Firstly, not all financial trackers have built in integration of banking details. Financial trackers and banking apps are usually independent from one another which makes it difficult to keep track of expenses across 2 different apps. Secondly, manual financial tracking is often redundant and time consuming. Thirdly, financial trackers do not provide detail data insights for the user. Traditional financial trackers only show charts and graphs with statistics, they fail to provide deeper context and provide more informed financial feedback to the user. In short, this smart financial tracking based mobile application is intended to solve the following issues:

i. Lack of integrated overview of banking institutions

Most financial tracking mobile applications lack an integrated support of banking institutions. Often than not, it can be difficult for users to keep track of their banking details in a single centralised platform. Therefore, users struggle with fragmented financial data across multiple banking applications. For instance, an individual who has just spent their expenses with a debit card would only be shown their transaction history in the banking application. However, if the individual wants to document this transaction. He / She would have to do it on a separate financial tracking application.

ii. Inefficiency of manual financial record tracking

Nowadays, almost everyone is showing increased awareness about financial planning. However, there are still some individuals who continue to record down their expenses manually either in a small notebook or even record down in spreadsheet software like Excel. While the traditional manual expense tracking methods remain common, they are inefficient and error prone. Hence, users should have access to a wide range of smart tracking methods that suit different contexts or scenarios.

iii. Insufficient data insight on expenses tracking

In the past, when individuals wanted to generate insight from their monthly expenses or transactions. They are overwhelmed by at least hundred lines of numbers and text. Notwithstanding this, they had to do manual calculation to calculate statistics like average spent, most spent category and remaining balance of that month. Even with data visualisation tools created in Excel like bar charts or pie charts. Users did not really feel involved with their financial data; it is likely that charts alone could not really connect with the user. While these existing expense tracking solutions offered basic visualisations, they failed to provide deeper, personalised financial insights that truly engaged with the users.

1.3 Project Objectives

The project's main objective is developing an intuitive financial tracking mobile application to help individuals who struggle to save money, are in debt, students and even personal finance enthusiasts. Individuals who use this mobile application can benefit from the smart automation features, such as automated receipt scanning, email scraping, expense report generation, personal fine-tuned financial insights and bank integration. The project aims to solve the following:

1) Enhance financial data accessibility and comprehensiveness

An integration with banking institutions through the Finverse API, creating a robust foundation for financial data integration. The implementation will demonstrate the ability to retrieve key financial data such as account balances and transaction histories from Malaysian banks including Maybank, Public Bank, and CIMB. This integration represents a significant advancement over traditional financial trackers that operate in isolation from banking applications, allowing users to access their financial information

within a single platform. The secure bank connectivity framework established in this project enables a streamlined financial management experience that bridges the gap between expense tracking and banking information.

2) Develop an automated data entry system for financial transactions

A smart automated financial tracker which solves the burden of tedious data entry and adapts to different contexts and preferences. The financial tracker addresses the common hassle of manually data input after physical transactions with an auto capture functionality which can extract data from receipts by leveraging Optical Character Recognition (OCR) powered by Google Machine Learning Kit. By leveraging an external package like speech to text, the application further enhances user convenience through voice recognition capabilities, enabling hands-free expense logging in dynamic situations like grocery shopping. While the overall goal focuses on receipt scanning and voice input, the app also allows users to perform email transaction scraping, enabling the automatic parsing of digital invoices and other online purchases confirmations. Overall, this reduces the time and effort required for user to log their financial transactions manually and improves the overall user experience and convenience. Not to mention, this feature helps minimises human error in data entry and improves the accuracy of financial tracking for both physical and digital transactions.

3) Create an interactive and personalised financial management insight

The proposed system leverages a full-fledged personal financial analysis system to analyse and recommend the best financial practices based on user’s data. At its core, the system generates detailed monthly expense reports in PDF format, providing users with a clear summary of their spending patterns and financial activity. These reports then serve as the foundation for an intelligent conversational interface powered by ChatGPT and Retrieval Augmented Generation (RAG) technology. By implementing a LangChain framework powered by a FastAPI backend, the system enables natural language interaction with financial data while maintaining factual accuracy. Also, the RAG based financial advisory pipeline personalises insights based on user’s unique interaction with the aggregated financial PDF. The architecture enables the chatbot to extract relevant financial information directly from the user's own expense documents, significantly reducing potential hallucinations that plague traditional LLM implementations. This approach enables users to interact naturally with their financial

data through simple queries like "Where am I overspending?" or “Show me all transactions on May 1, 2025.”, receiving contextually aware responses grounded in their actual transaction history. The overall goal is to integrate a RAG financial advisory PDF based chatbot and present them with financial insights based on their query.

1.4 Project Scope

The scope of this project is to enhance the overall financial tracking experience for individuals who are trying to get out of debt, save money and budgeting. Therefore, the proposed mobile application **“Smart Financial Tracking Mobile Application”** allows these individuals to automate financial data entry such as receipts, scrape financial transactions from email, chat / interact with inputted financial data, monthly expense report generation, receipt sharing, financial chart analysis and lastly an integration of Malaysian banks. All these suggested features are unlike traditional budgeting or financial trackers, as this mobile application will leverage a lot of integration with current third-party APIs, libraries like Google Machine Learning Kit, Langchain API, n8n workflow integration and FastAPI. Below shows a brief overview of the main modules.

i. Login In / Sign Up module

To access the mobile app's full range of features, users must first create an account. Without completing the registration process, individuals will be unable to proceed to the main interface where all functionalities are available. Then, after completing the registration process, users can immediately access the application by logging in with their credentials, which are securely authenticated through Firebase's robust authentication system. This module is essential to keep track of different user’s spending and expenditure upon using the app. Also, this also caters for the saved data instance for each new user created. The Login In / Sign Up will utilise Firebase (Backend as a service) for user authentication.

ii. Receipt Scanning OCR module

Optical Character Recognition (OCR) is the process of recognising characters from images using computer vision. Also, receipts are the main source of financial data after an individual has purchased something. This functionality leverages the capabilities of Optical Character Recognition (OCR) for receipt scanning which can help automate the process of manual entry as it can directly extract key information from receipts. The high-level overview includes user first capturing an image of the receipt, then Google Machine Learning Kit will detect the bounding boxes around possible texts and finally extract the recognised text. In the context of financial tracking, the main region of interest is the total amount that is spent for that transaction that will be later stored in a database.

iii. Voice data entry module

Voice data recognition is another common way to automate tasks, it is common in voice assistant technologies such as Siri and Google Assistant. In the context of financial data entry, this approach allows convenience for recording expenses through natural language. The system would leverage a speech to text technology to capture and process verbal sentence structures. For example, user can simply say “Today, I spent RM 50 on food”. Then the system will parse the recognised text and return corresponding bullet point list showing the recorded expense, date and category. The functionality is practical in situations where manual input is troublesome such as while walking or when the user needs to input text quickly and hands-free.

iv. Scrape transactions from email module

With the advent of the internet, financial transactions are also recorded in digital formats that will be sent to the user’s email. The financial transactions that are common in an email is digital invoices, order confirmations, online subscriptions and shipping receipts. This module mainly implements an automated workflow using n8n, an open-source workflow automation platform as compared to traditional web scraping methods. This approach enables the system to connect directly to email services, identify relevant financial communications using intelligent filtering, and extract key transaction details through structured data processing workflows. The process will involve the automation of login into user’s email and scrape relevant emails based on

subject of email and date range. After scraping the relevant emails, the data is pre-processed before being displayed back on the mobile application interface. Scraping transactions directly from emails would also reduce the manual data entry for users.

v. Chat with financial data module

Users have the chance to interact with their financial data in PDF format, via a chatbot like interface. But unlike traditional Large Language Models (LLM) chatbots who solely rely on pre-trained knowledge, users are instead giving more context to the LLM based on their financial entries such as a PDF which acts as an external data source to the LLM. This implementation creates a personalised financial advisor that grounds all responses in the user's actual transaction data through PDF-based RAG technology. The interface retrieves relevant information directly from their monthly expense reports, ensuring answers are factually accurate and contextually appropriate. Overall, the chatbot significantly reduces the risk of hallucinations by anchoring all responses in documented financial data. The module presents insights in conversational language, making complex financial information accessible while delivering actionable recommendations tailored to each user's unique financial behaviour and goals.

vi. Financial analytics module

This module delivers intuitive financial visualization through a comprehensive dashboard interface. Users can access an overview of their spending patterns with interactive charts displaying expense breakdowns across various categories. The system offers flexible time-based filtering options, allowing users to analyse specific monthly periods and view key performance indicators such as total expenditures, average transaction, transaction counts, and spending categories. Lastly, for deeper financial analysis and record-keeping, users can generate detailed PDF financial reports that present category-based percentage breakdowns alongside a complete chronological transaction history, providing both summary insights and granular transaction details in a structured, easily accessible format.

vii. Bank app integration module

Many financial trackers and budgeting tools reviewed only accommodate to US or European banks. However, in the context of Malaysian banks. Most financial trackers cannot / unable to import and connect to existing Malaysian banks. In this proposed

mobile application, Finverse API is integrated into the financial tracker. As this is one of the very few third-party APIs which can integrate with up to 3 local Malaysian banks namely Maybank, Public bank and CIMB to extract end user's financial data such as currency, total amount of savings account, statement balance and other various financial data.

1.5 Contributions

The main contribution of the proposed application is to solve the issues stated in the above problem statement. The proposed system acts as an enhancement on top of many existing financial tracking mobile applications. Another core idea is to make the application free of charge, as many financial trackers would charge users for using extra features such as bank integration. Notably, users who cannot afford financial management tools like Document360, Mint and Mvelopes can utilise this mobile application instead.

A key technical innovation in this project is establishing the interfacing connection between the mobile application and n8n's workflow automation platform for email transaction processing. This novel integration addresses a significant gap in the current mobile development landscape, where direct communication between mobile apps and external automation tools like n8n has historically been challenging. By creating and documenting this implementation approach, the project not only solves a specific functional requirement but also provides the broader developer community with a valuable reference architecture that can be adapted for similar integration scenarios across other workflow solutions.

Beyond automated data entry using OCR, users benefit from comprehensive financial insights powered by AI through the RAG-based chatbot system. The application's approach of generating structured financial reports that serve as the foundation for conversational analytics creates an interaction model that makes financial data more accessible and actionable for everyday users without technical expertise. This implementation demonstrates how emerging AI technologies can be practically applied to solve real-world problems in personal finance management.

1.6 Report Organisation

The whole FYP 2 report consists of six main chapters including Chapter 1 which briefly introduces the problems of existing financial trackers and how AI could be used to enhance the user's financial tracking ability. In Chapter 2, a vigorous literature review process is performed to evaluate the strengths and weaknesses of existing systems. Next, in Chapter 3 showcases the proposed method / approaches when developing the mobile application, backend hosting and database storages. In Chapter 4, the main system development methodology, system requirements, implementation bottlenecks and timeline are discussed in detail. In Chapter 5, various functionalities of the mobile application are demonstrated in detail. Lastly, Chapter 6 concludes the overall findings of this project and future work.

1.7 Summary

In chapter 1, a brief overview of the history of traditional financial tracking is examined. Not everyone is entitled to use latest state of the art financial technologies used in fintech companies. The innovation of mobile application financial trackers helps tackle this issue. Next, problems were examined from the perspective of existing financial mobile trackers such as lack of integration between banking institutions, hard to keep track of financial expenses on the go and the insufficient data insight in expense tracking. Therefore, a proposed cross platform android mobile application aims to solve these issues and improve the overall user experience when using a financial tracker. Users can view banking details in a centralised application, automate data entry with OCR and understand their financial expenses more insightfully.

CHAPTER 2 Literature Review

This chapter showcases past works and similar projects for finance related apps. A review of past work is essential when developing a new system that aims to solve problems that are aligned with the problem statement. Also, similar systems are reviewed to get an overview of the strengths and weaknesses of each respective platform. Then after reviewing the strengths and weaknesses of each system, it is summarised in a table for easy comparison.

2.1 Previous Works on Finance related apps

2.1.1 Touch ‘n Go

TnG eWallet is an eWallet platform to conduct payments and transactions in stores, tolls and even parking payments. The main initiative of TnG eWallet was to encourage a cashless society. Obviously the 2 common use cases would be transferring money and scanning for payment as shown in Figure 2.1.1.1. In a way or another, TnG eWallet also exhibits the functionality of tracking transactions. Users can view their transaction history of up to 90 days Figure 2.1.1.2. Not to mention, they can check the transaction type, merchant and other specific payment details. Just recently, TnG eWallet launched a new supplementary feature for financial tracking named Gofinance which is an all-in-one financial hub for convenience and accessibility. Gofinance allows eWallet users to seamlessly manage their expenses and budget their spending within the eWallet. Thus, offering valuable financial insight into user’s financial habits as shown in Figure 2.1.1.3.

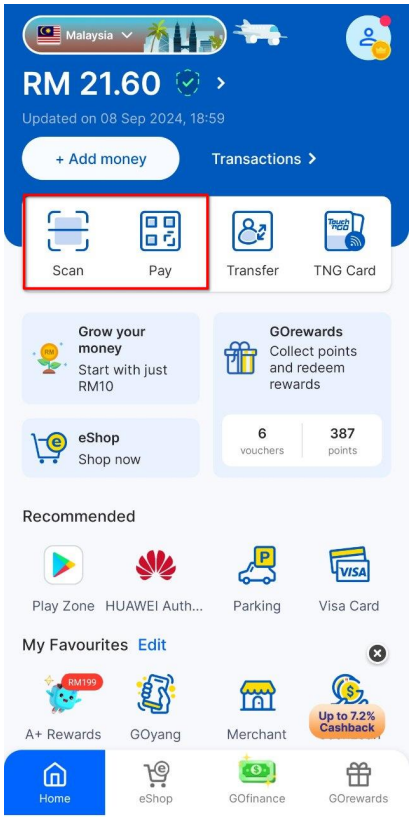


Figure 2.1.1.1 Scan and Pay feature

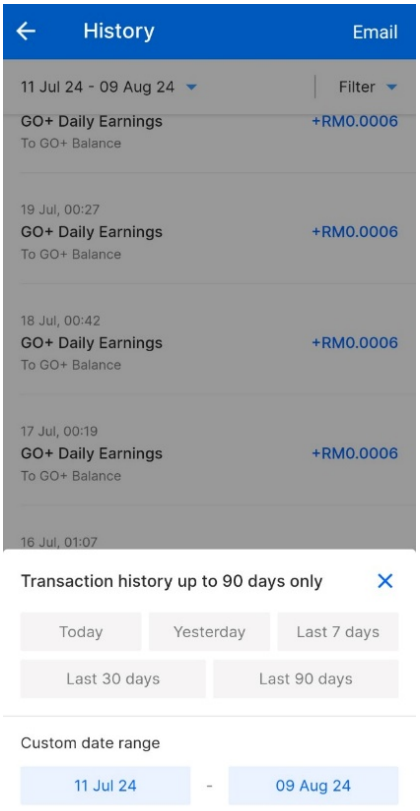


Figure 2.1.1.2 View transaction history

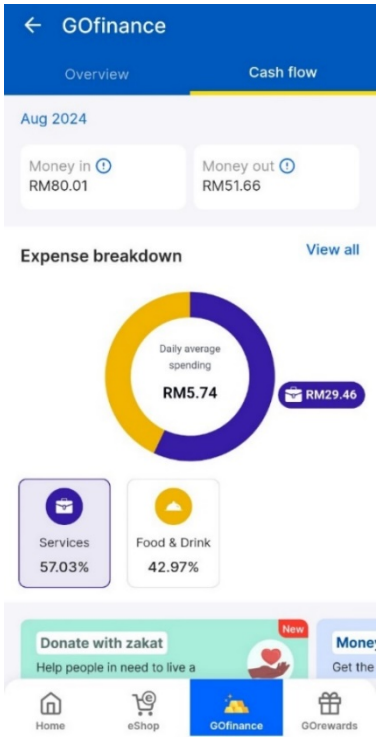


Figure 2.1.1.3 Gofinance dashboard

Strengths

i. Cashless payment

Touch ‘n Go is an E-wallet which streamlines transactions by eliminating the need for physical cash or debit cards. This increases the convenience for busy individuals who take time to withdraw money from ATM machines or banks. Users of Touch ‘n Go have 2 ways the ability to make purchases at a Merchant’s outlet which are using the scan feature and pay feature as shown in Figure 2.1.1.4 and Figure 2.1.1.5. The scan feature is where the user scans the Merchant’s QR similar to Figure 2.1.1.6 code and enters the amount whereas the pay feature is where the merchant/cashier scans the user’s QR code just like Figure 2.1.1.7, and the amount is deducted from their balance.

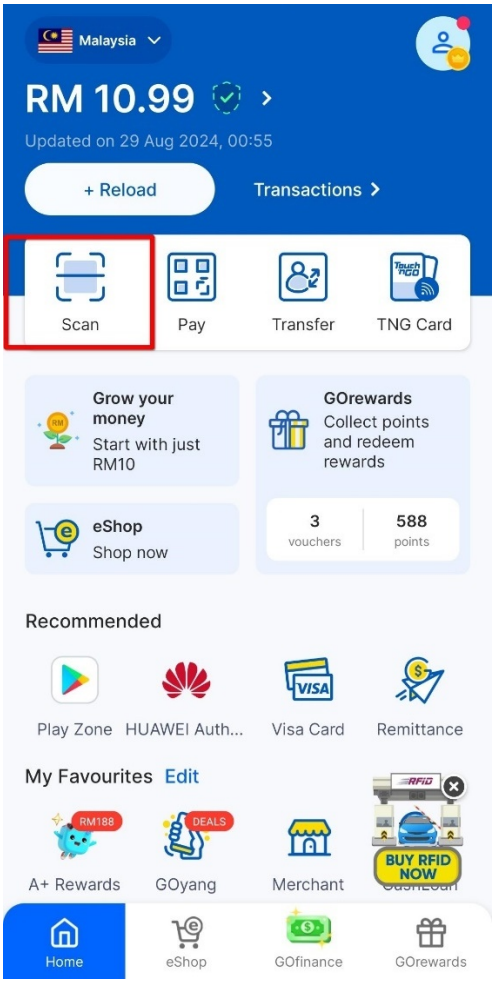


Figure 2.1.1.4 Scan feature

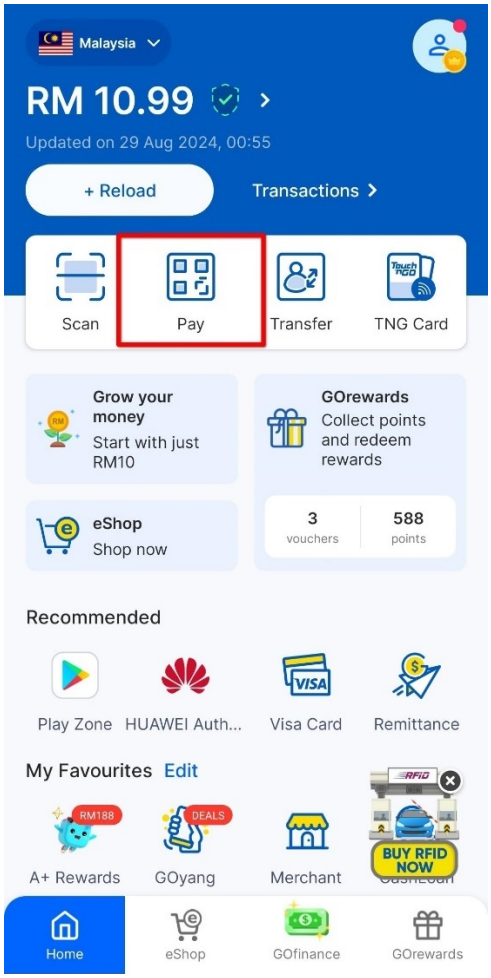


Figure 2.1.1.5 Pay feature

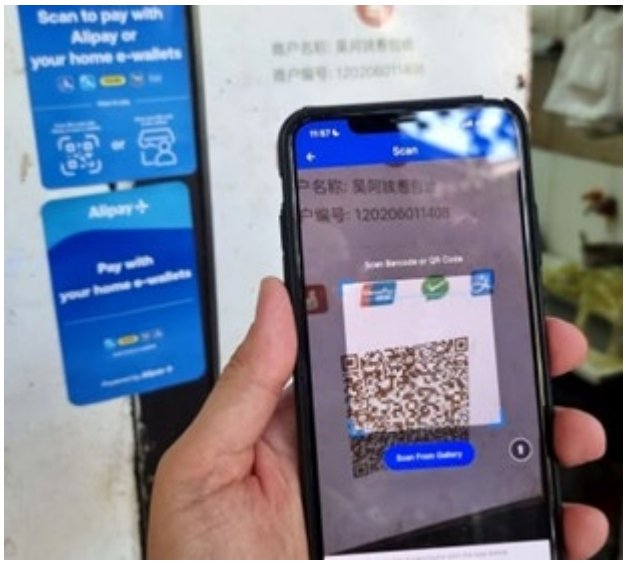


Figure 2.1.1.6 Scanning merchant's qr code

Figure 2.1.1.7 Paying with qr code

ii. Loyalty program

Touch ‘n go also offers a loyalty program with different tiers which are Lite, Pro and Premium respectively as shown in Figure 2.1.1.8. When a user first registers for an account, they are automatically entitled to the Lite tier, after verification process, they will be upgraded to the pro tier and finally if they wish to be on the premium tier. They will have to onboard the premium page by paying a small fee. Each tier offers different benefits including different monthly transaction limits, annual transaction limits and additional features. For the basic lite tier, user is only able to make annual transaction

of up to 24,000 MYR whereas the premium tier users can spend up to 360,000 MYR a year. Some extra features for the premium tier include sending payments and premium product benefits.

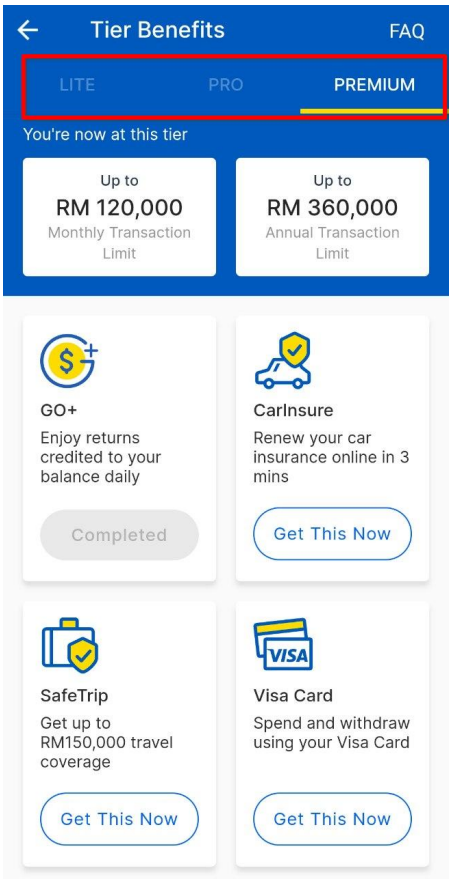


Figure 2.1.1.8 Loyalty tiers

iii. Gofinance (all in one financial hub)

Recently in June 2024, Touch ‘n Go officially launched a new feature called Gofinance which acts as a financial hub which shows summary of aggregated statistics of different expenses breakdown like services, transfer and food & drink as shown in Figure 2.1.1.9. The new feature aims to empower users by providing financial insight through interactive graphs. Also, the feature encourages users to take full control of their financial goals. Consequently, Gofinance also provides a variety of financial services like applying for cash loan, insurance and credit score management through a digital and convenient process [5]. One key point of Gofinance is the low entry barrier which makes these financial services available to everyone regardless of financial status.

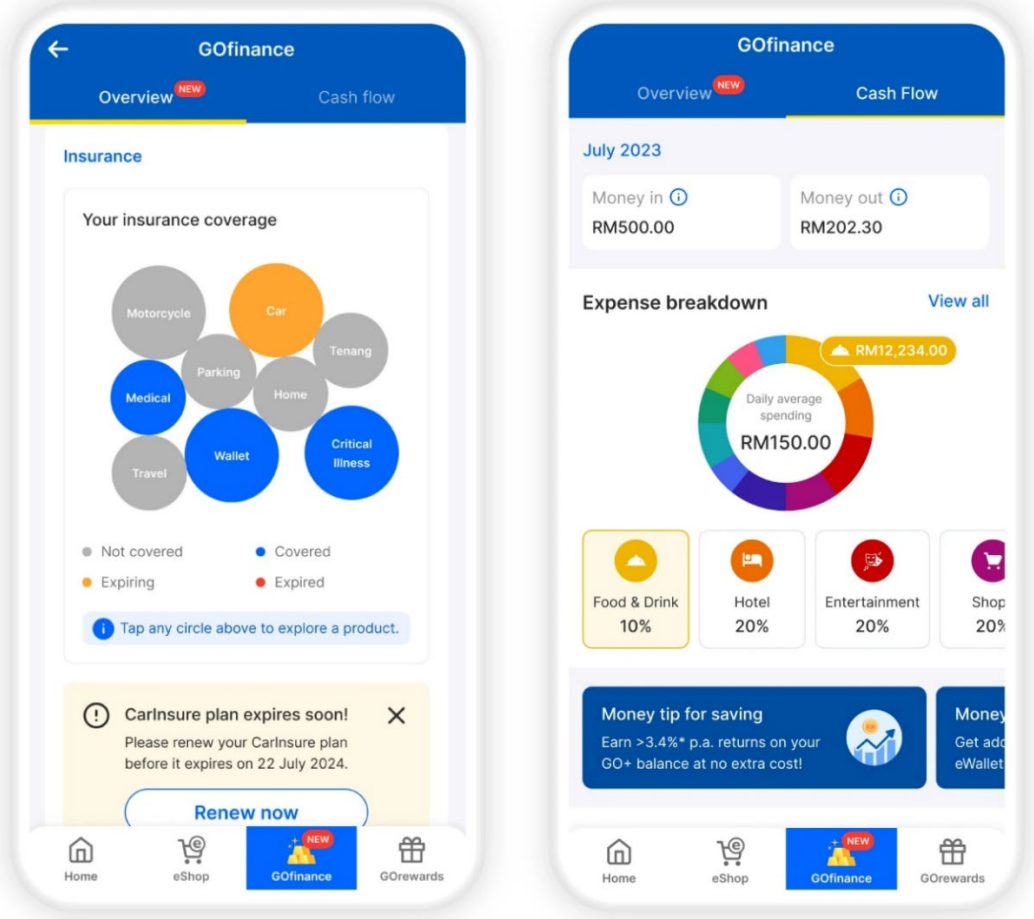


Figure 2.1.1.9 Gofinance functionality overview

Weakness

i. **Non-transferable balance**

In 2022, Touch n' go introduced 2 main types of balance which were transferable and non-transferable eWallet balance as shown in Figure 2.1.1.10. A transferable eWallet balance can be used for all the normal payment use cases such as QR code payments, DuitNow QR payments and peer-to-peer transfer. In contrast, a non-transferable eWallet balance can cater to most payment cases similar to transferable amount except for peer-to-peer transfer, GOpinjam loan repayment and ATM withdrawal through Touch 'n Go. To note, TNG eWallet reloads made via credit cards, reload pins, government initiatives fall under non-transferable balance. Additionally for credit card reloads, a monthly quota of RM1000 will be treated as transferable eWallet balance. After exceeding RM1000, a 1% fee will be charged for each reload. The introduction of non-transferable balance has caused inconvenience for multiple users. In a case where an individual bought a TNG reload pin intending to transfer the credit to their elderly parents would fail as reload pins are non-transferable amount.

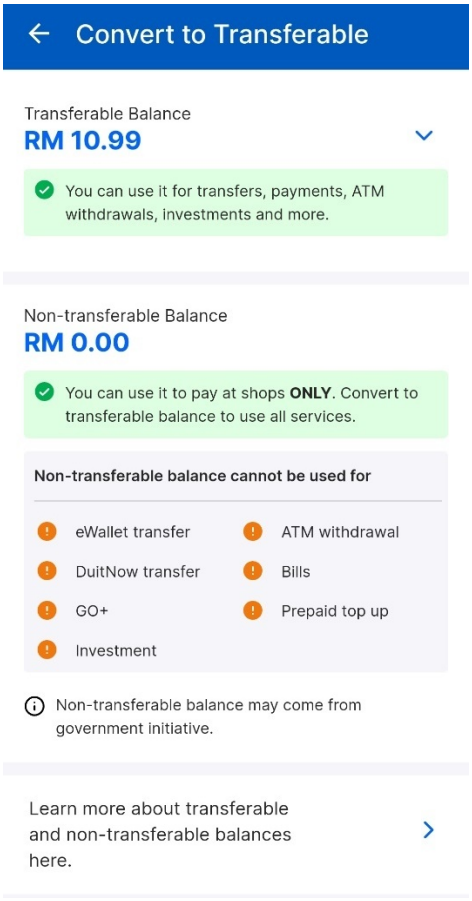


Figure 2.1.1.10 Convert to non-transferable balance

ii. Counter intuitive User Interface

The user base of TNG is wide, there are more than 1 million active users daily that use the eWallet. While the app is useful for making transactions to merchants seamlessly. The appearance and access can be sometimes difficult to follow for non-technical users. As shown in the Figure 2.1.1.12 below, on the homepage itself. There are too many icons and too many unnecessary buttons on a single page (view) as shown in the red borders. Furthermore, in the middle section and bottom navigation bar highlighted in light blue are repeated elements for instance eShop and GOrewards. Touch ‘n Go is trying to highlight these are 2 important features but in return, it leads to redundancy and confusion for the user. Another thing to highlight is the bottom navigation bar (prominent blue), in most cases many users would not even notice it at first glance due to the attention drawn towards the red icon such as the profile section at top right corner as shown in Figure 2.1.1.11.

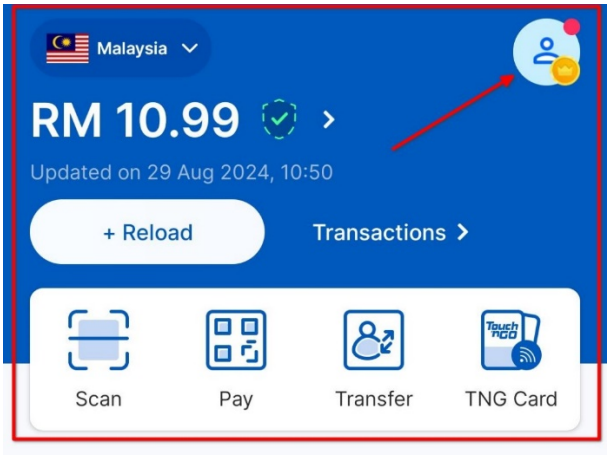


Figure 2.1.1.11 Upper right corner icon

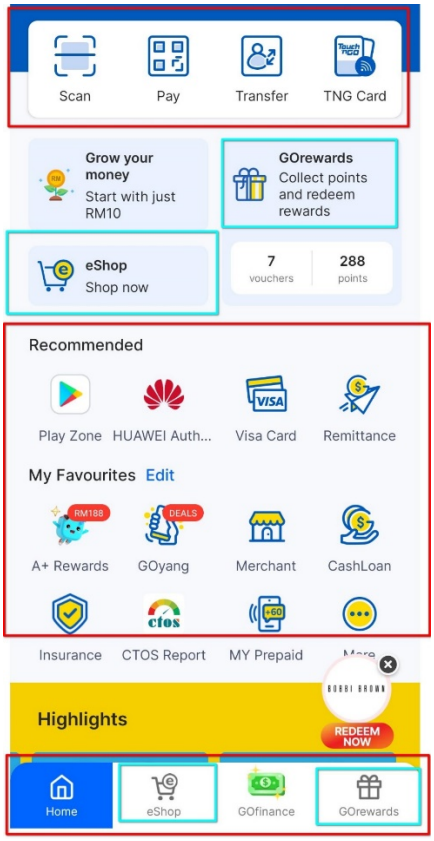


Figure 2.1.1.12 Too many cluttered and repeated icons

2.1.2 You Need a Budget (YNAB)

You Need A Budget (YNAB) is a popular budgeting platform that lets users take charge of their finances. YNAB has taught millions of people how to set habits and change their relationship with money for good. The main key features of YNAB not only allow individuals to create a budget but also allow bank integration, goal setting, spending & net worth reports and much more. YNAB follows 4 rules which are given every dollar a job, embrace your real expenses, roll with the punches and age your money. Despite all these features, strictly speaking YNAB is not a monthly spending tracker. Instead, the app forces users to decide where the money should go beforehand. Hence, users understand where their expenses are going into, and this concept helps them save money. YNAB offers cross platform availability which consists of both the website version and mobile app version.

Strengths

i. Visual clarity of finances

Although YNAB's main feature is setting budgets for different types of allowances. Users are also given the chance to view a summary of their outflow cash through the reflection feature. However, the reflection feature is only offered on the web app version. The mobile app and web app versions are always in sync, which means that if the user has added a new transaction known as utilities. The category and the amount of the transaction will also be synced to the web version. Visual clarity of finances is one of many strengths of YNAB. A clear, visual representation of spending breakdown is shown to the user instead of going through endless spreadsheets of financial data. The overall spending totals for a certain month can be represented as a donut graph as shown in the Figure 2.1.2.1 below. Each section represents different categories such as dining out, phone, utilities and bills.

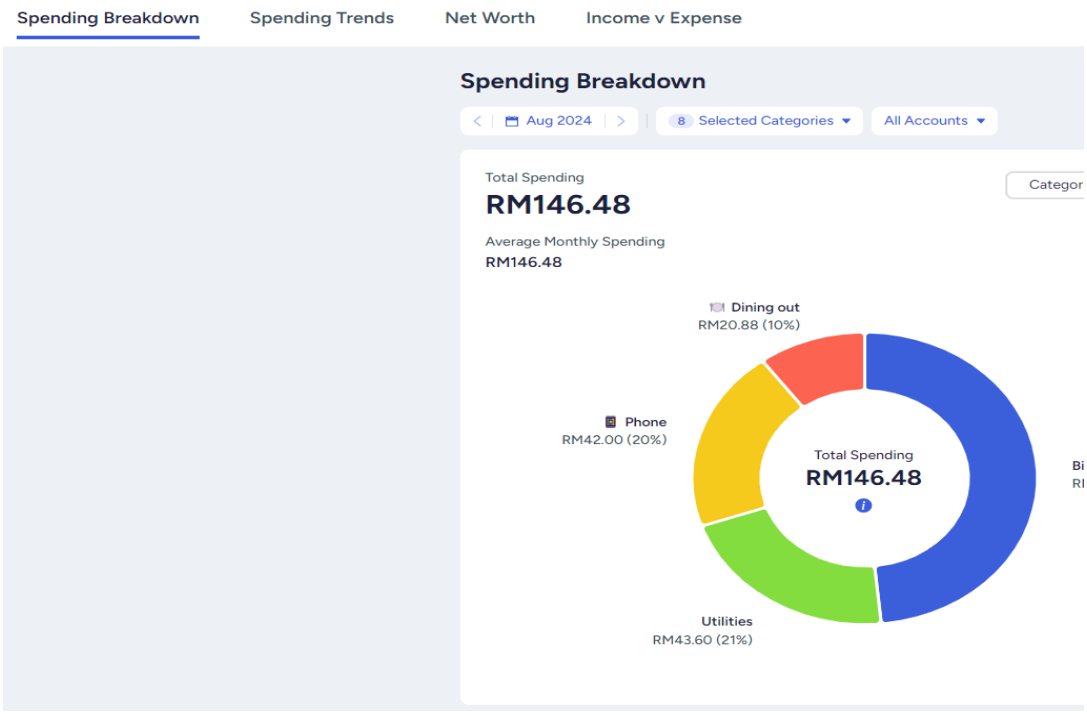


Figure 2.1.2.1 Donut graph spending breakdown

ii. Active engagement

YNAB is unlike other traditional budgeting apps, YNAB chooses a more active approach. Before starting the budgeting process, the user must create an imaginary account which denotes the amount that will be spent. Users are given the choice to either import data from banking institutions or create a custom balance as indicated in Figure 2.1.2.2 – Figure 2.1.2.3. As an example, if the user creates a custom budget called balance_1 with an amount of RM5000 as shown in Figure 2.1.2.4. This means that the user chooses which transactions or categories to spend this RM5000 on. This approach is different than traditional budget trackers or financial trackers where the user blindly documents down their expenditure without the consideration of overspending. Hence, YNAB is fostering a deeper understanding of connecting with one's finances. User is not passively tracking what they spent, instead they are actively deciding beforehand what to spend on.

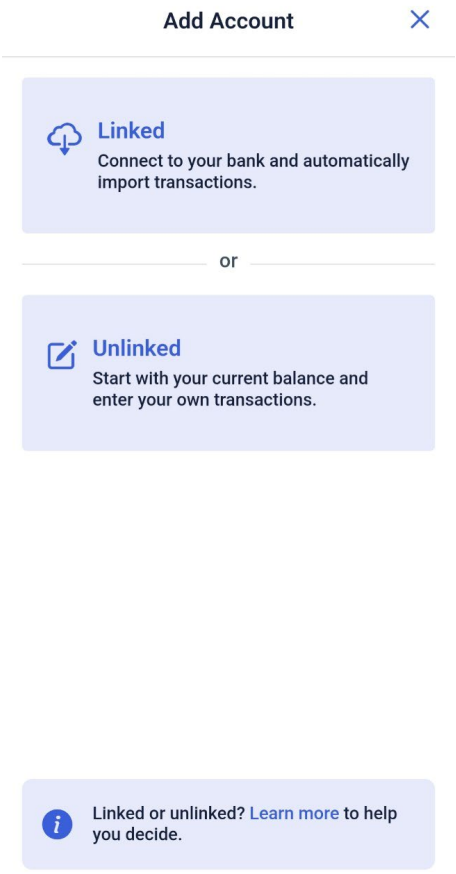


Figure 2.1.2.2 Link bank account

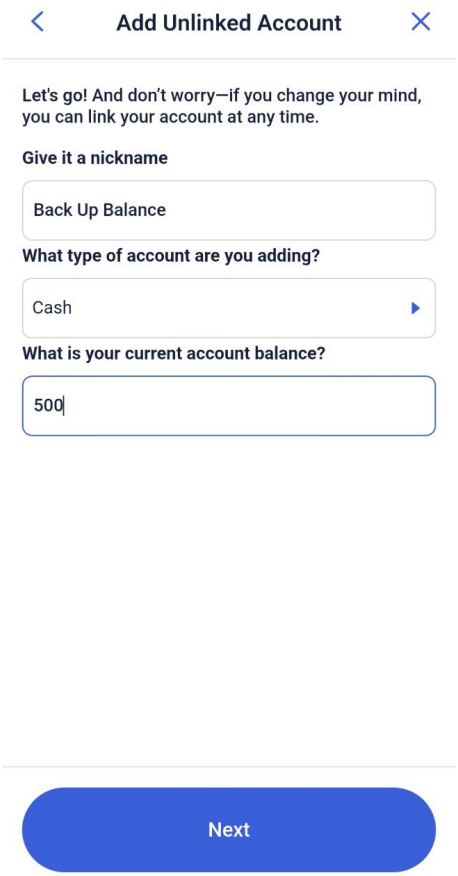


Figure 2.1.2.3 Unlinked account

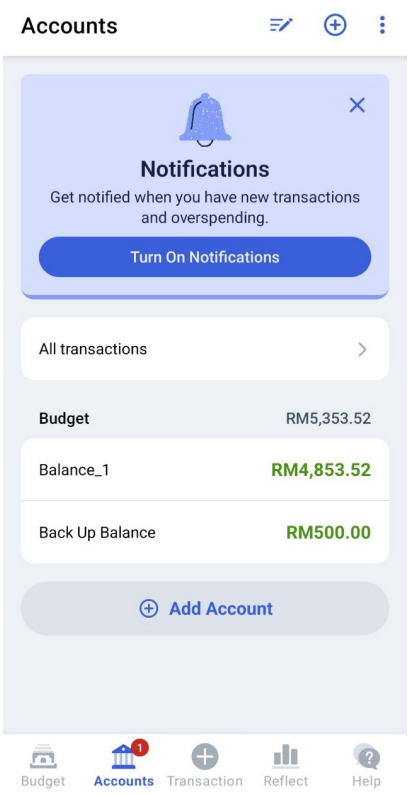


Figure 2.1.2.4 Custom amount set for unlinked account

iii. Add transaction

Another prominent feature of YNAB, is the ability to add transactions either outflow or inflow as shown in Figure 2.1.2.5. This feature allows user to keep track of transactions that are going out and into their set balance. Furthermore, user can choose payee, the category, type of account to add or deduct from and date. Another interesting point about this feature is the split category section which allows a single transaction to fall under different categories as shown in Figure 2.1.2.6.

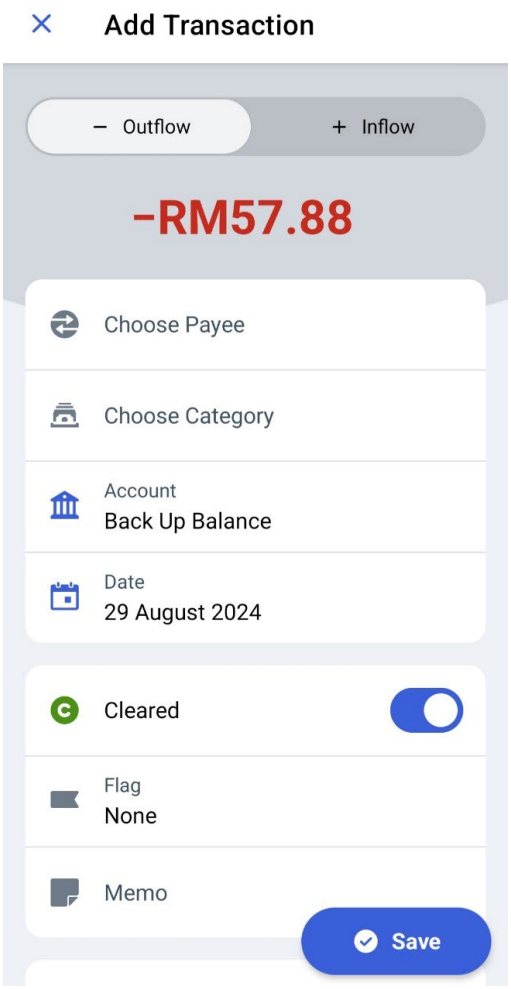


Figure 2.1.2.5 Adding transaction

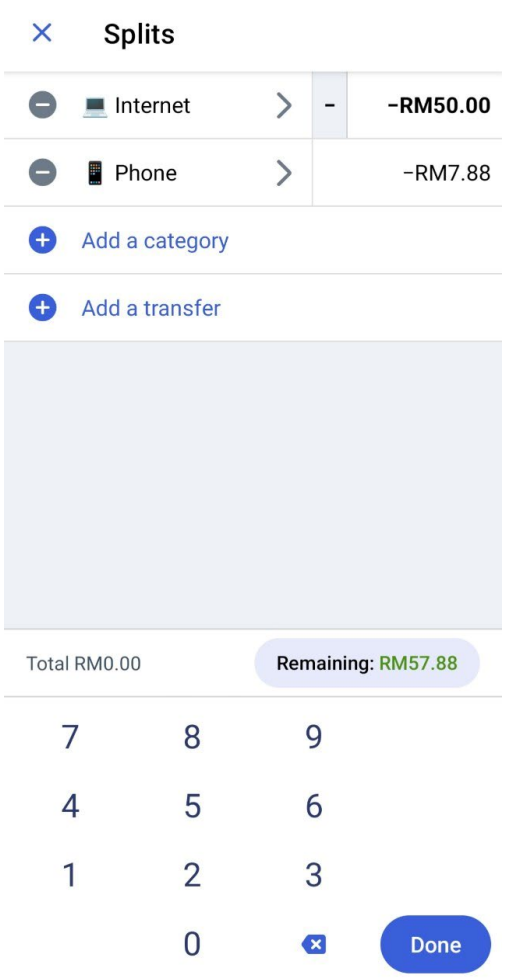


Figure 2.1.2.6 Split into more than one category

Weakness

i. Lack of Malaysian bank integration

YNAB allows the import of banking data from supported banks. However, not all banks are supported in the context of Malaysia banks such as Maybank, Public Bank and CIMB. Most of the banks being supported are either United States or European banks as shown from Figure 2.1.2.7 – 2.1.2.8.

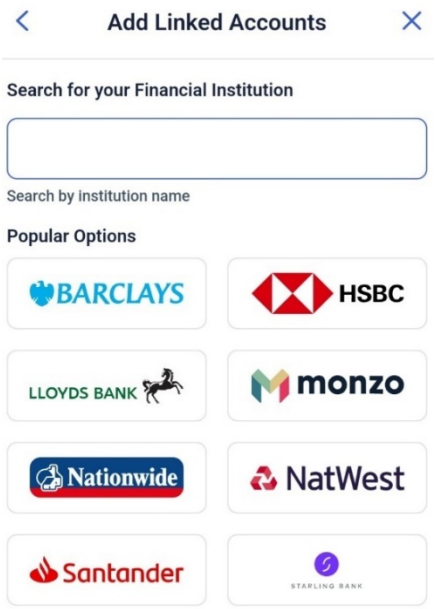


Figure 2.1.2.7 European bank availability dominating

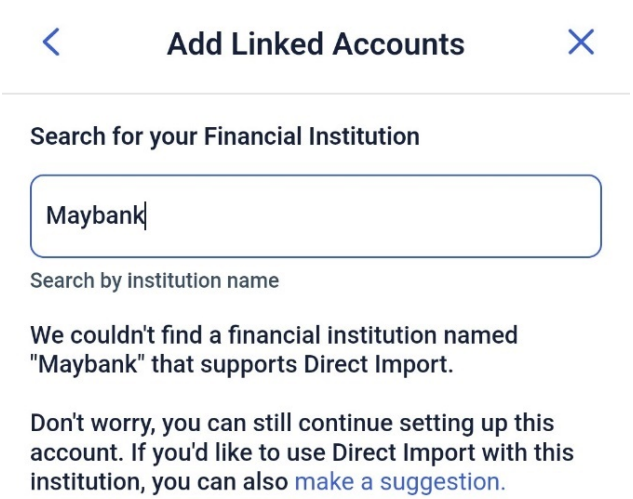


Figure 2.1.2.8 Lack of Asian banks integration

ii. Charges for app after trial

YNAB is not entirely free to use, the user is only given a 34-day trial to test run the app and see if YNAB fits their money management methods as indicated in Figure 2.1.2.9. Within the trial period, users are entitled full access to YNAB's powerful software features including the loan planner, reports, money plans, spending targets and auto import. Nevertheless, after the trial has ended. User is immediately prompted to either pay a monthly or annual subscription to continue using the features. It is arguable that individuals who use budgeting apps want to save money instead of spending more money. In this case, YNAB is instilling a new burden of RM444.99 per year or RM66.99 per month for a tool that encourages users to get out of debt and reduce expenditures on unnecessary subscriptions.

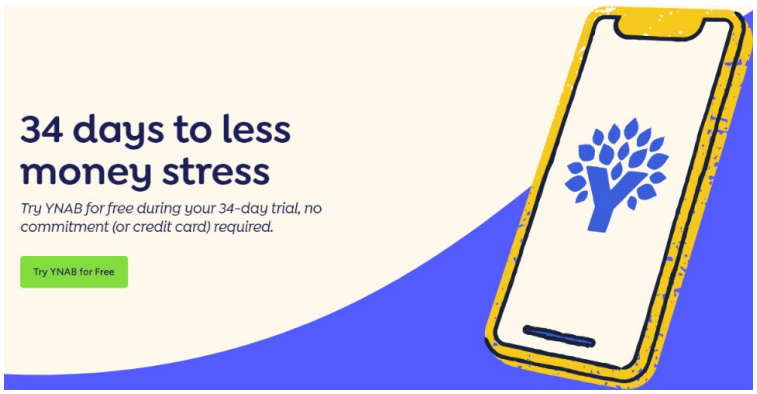


Figure 2.1.2.9 A 34-day trial period

2.1.3 Meow Money Manager

Meow Money Manager is also another expense/budget tracker but this time it focuses on cute cat icons for personal asset management. Also, it acts as an expense tracker to record financial expenditure and income activities. The app claims that it will not save any information of users, thus protecting their digital piracy. Overall, the Figure 2.1.3.1 below shows that Meow Money Manager focuses on a cute User Interface design focusing on cat-based themes which makes the overall experience budgeting experience more appealing and fun.

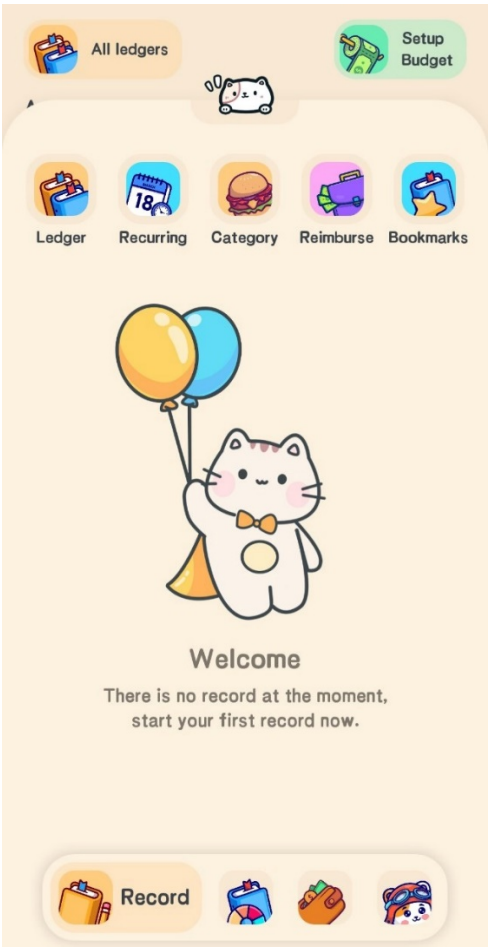


Figure 2.1.3.1 Main interface of meow money manager

Strengths

i. Bill export

The added expenditures can be exported as a csv file as shown from Figure 2.1.3.2 – 2.1.3.3. Also, user is also able to select categories, assets, ledgers and time to filter the records to be exported. CSV files are compatible with almost all software applications that work with data.

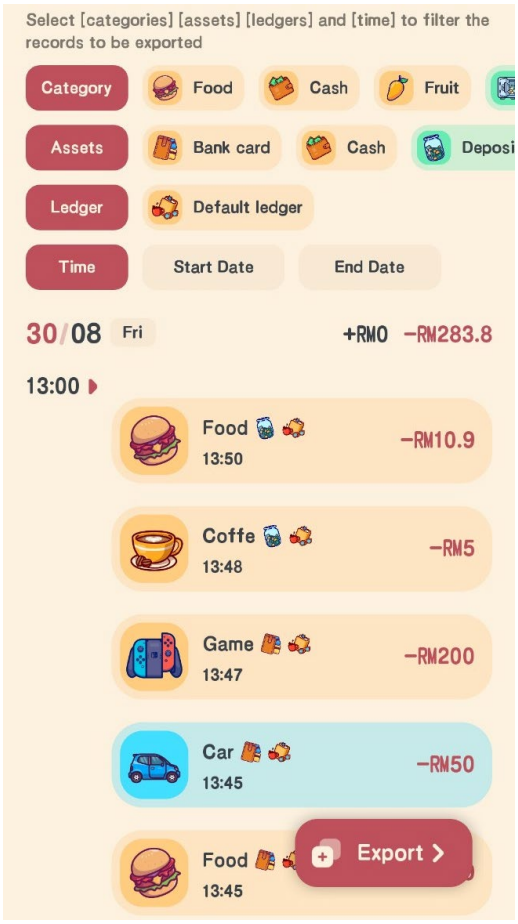
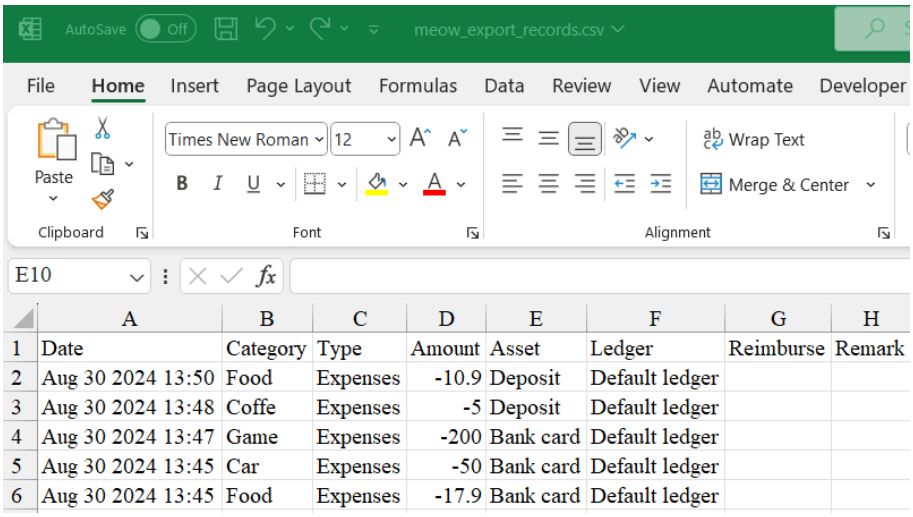


Figure 2.1.3.2 Bill export functionality



	A	B	C	D	E	F	G	H
1	Date	Category	Type	Amount	Asset	Ledger	Reimburse	Remark
2	Aug 30 2024 13:50	Food	Expenses	-10.9	Deposit	Default ledger		
3	Aug 30 2024 13:48	Coffe	Expenses	-5	Deposit	Default ledger		
4	Aug 30 2024 13:47	Game	Expenses	-200	Bank card	Default ledger		
5	Aug 30 2024 13:45	Car	Expenses	-50	Bank card	Default ledger		
6	Aug 30 2024 13:45	Food	Expenses	-17.9	Bank card	Default ledger		

Figure 2.1.3.3 Exported csv with expenses

ii. Recurring transactions

As shown from Figure 2.1.3.4 – Figure 2.1.3.5, Meow Money Manager allows user to set recurring transactions for different categories on daily, weekly, monthly and yearly basis.

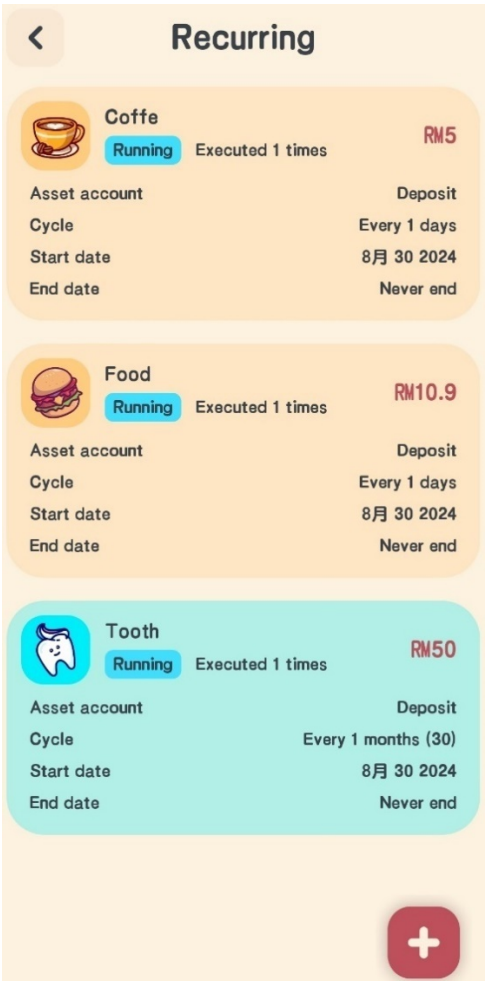


Figure 2.1.3.4 Recurring expenses for different categories

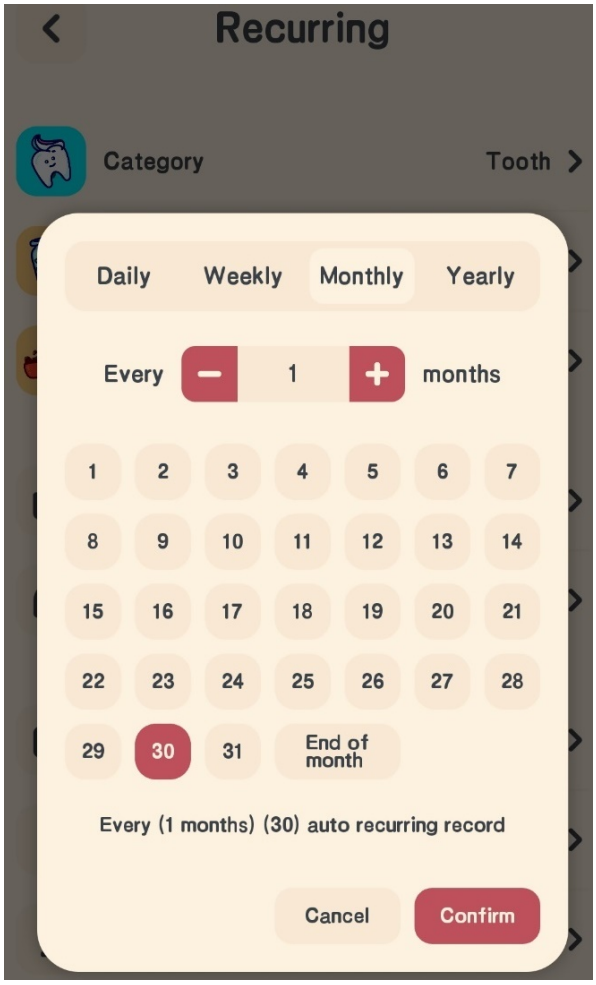


Figure 2.1.3.5 Set recurring transactions on monthly basis

iii. Assets management

In Meow Money Manager, users can create their own asset accounts such as bank card, cash, deposit and side hustle as shown in Figure 2.1.3.6. All of these contribute to the overall asset value. Also in this section, users can view their overall assets and liabilities. Assets indicate the overall asset value on hand such as having a deposit of RM4819.10. The negative (-) sign would indicate a liability which happens when the user spends more than they have. As shown in Figure 2.1.3.7 below, bank card and cash are liabilities because the user has overspent thus contributing to an overall liability of RM316.90. Notably, donut graphs area also shown for both assets and liabilities for easier visualization as demonstrated in Figure 2.1.3.7.

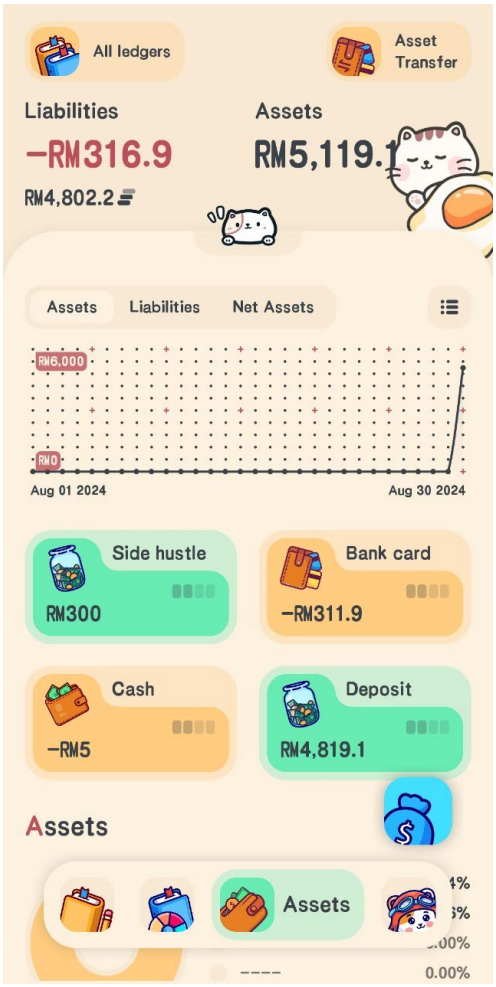


Figure 2.1.3.6 Managing different assets

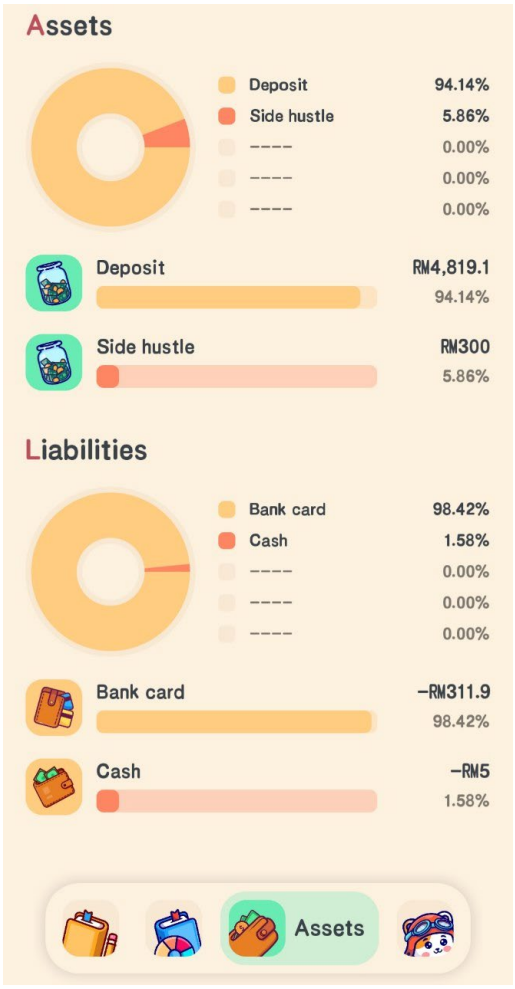


Figure 2.1.3.7 Donut graph of assets and liabilities

Weakness

i. Incompatible language support

In the settings section of Meow Money Manager, there is an option for users to change to a different language as shown in Figure 2.1.3.8. However, the app shows compatibility issues with other supported languages such as Mandarin, Malay and French. After selecting any other language besides English. The main text views of other sections remained in English. The entire app was not changed at all, except for a specific custom section which was choosing the first day of the week as shown in Figure 2.1.3.9 below.

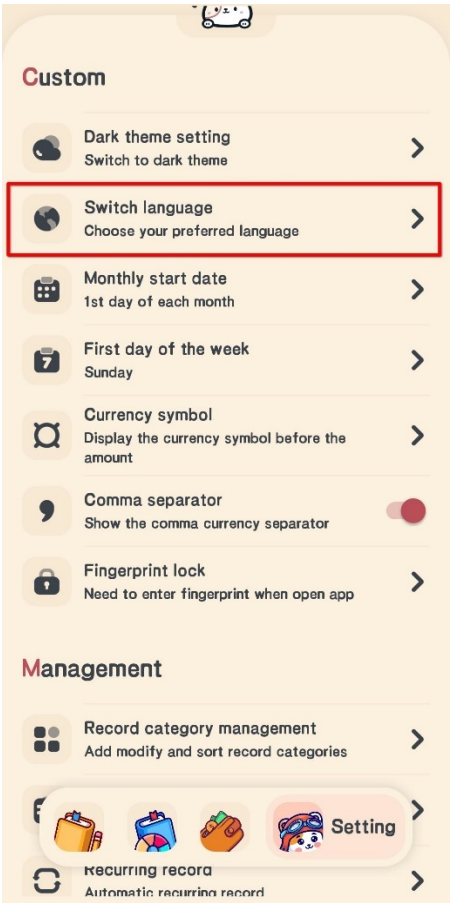


Figure 2.1.3.8 Switching languages

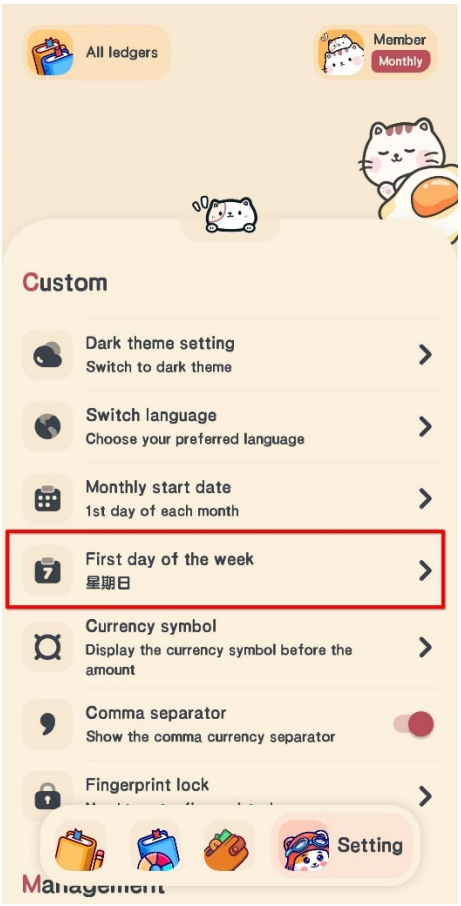


Figure 2.1.3.9 Only a single option is changed

2.1.4 Easy Expense

Easy Expense is an expense management mobile application that is catered for both individual and teams. It eliminates the tedious task of manual receipt organization into a streamlined, automated process. The core functionality of Easy Expense is receipt tracking which focuses on individuals who want to claim tax reduction or reimbursements. Users can keep their physical receipts in a digital format to prevent the misplacement or losing of receipts as shown in 2.1.4.1. Consequently, it leverages AI to automatically categorise labels for receipts such as tax category, vendor, total, tax, date and payment method. Besides, the app helps users automatically import email receipts by connecting to their Gmail account to auto scan for receipts. Consequently, user can connect to their bank accounts and credit cards. Overall, the app claims to help an average user save up to $2,192 (RM9470.55) per year on taxes.

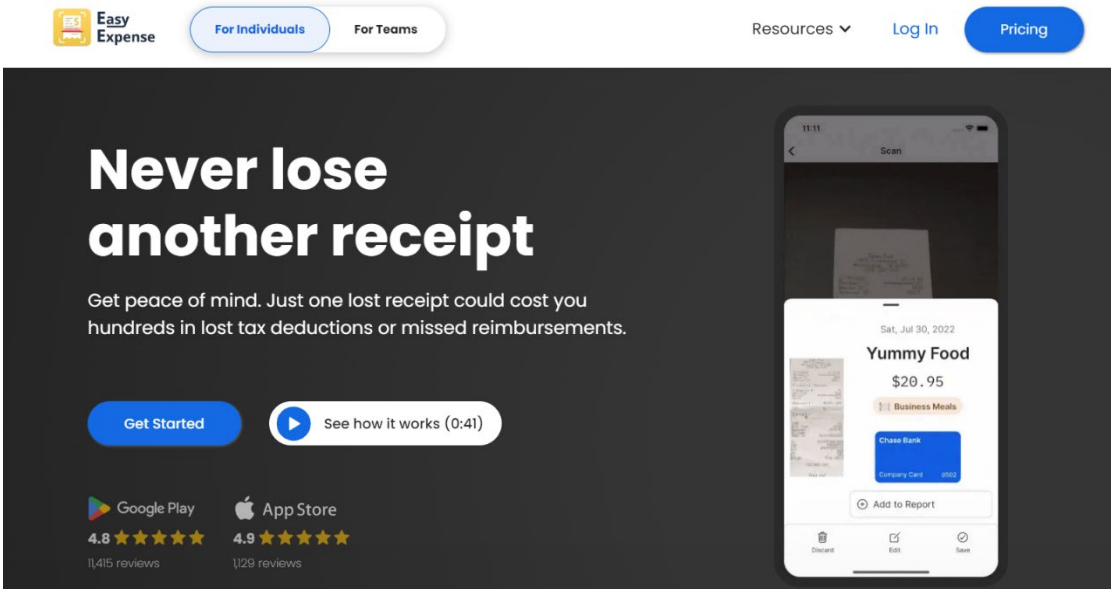


Figure 2.1.4.1 Easy Expense's webpage motto

Strengths

i. Scan receipts using OCR

Manually entering data from paper receipts can be troublesome. But Easy Expense has this core functionality which allows users to swiftly scan receipts within a few seconds as indicated from Figure 2.1.4.2. The app can detect, crop and extract pertinent data such as tax, total and date of transaction. Notably, there is an auto sync capability which saves the processed receipts into the cloud. As shown in the Figure 2.1.4.3 below, these are the 2 physical receipts scanned using the application. In Figure 2.1.4.4, the scanned receipts is blurry, but the application still managed to accurately determine the total amount. In Figure 2.1.4.5, users are also able to select their payment method and request a refund expense that will not be accounted in the expense tracking process.

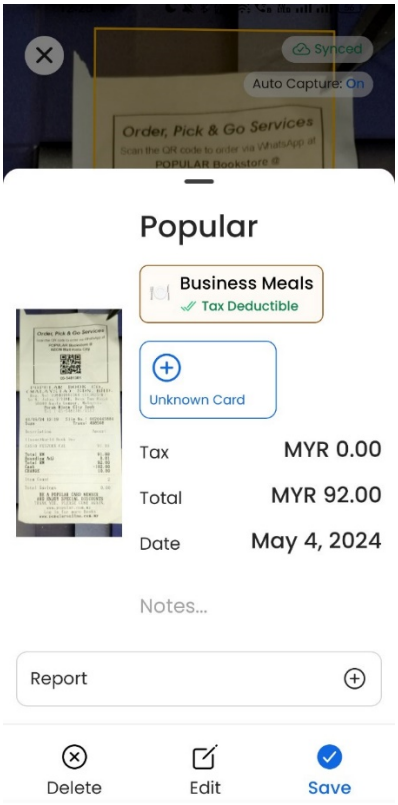


Figure 2.1.4.2 Example of scanning a receipt from Popular bookstore

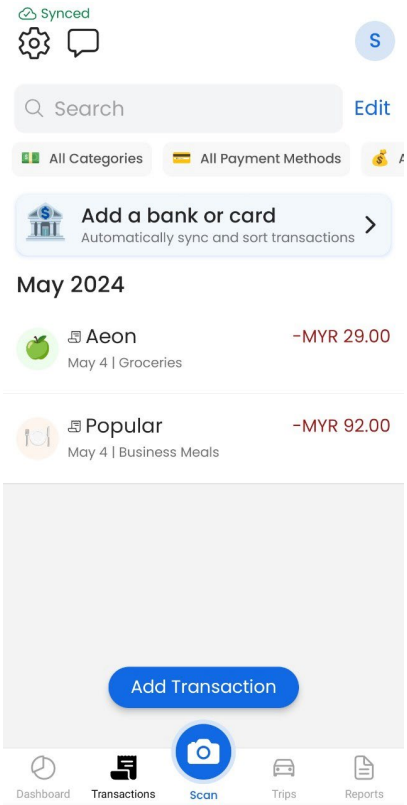


Figure 2.1.4.3 Main homepage interface

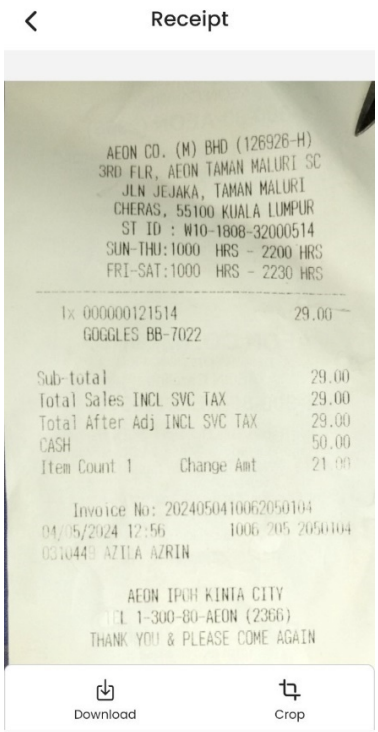


Figure 2.1.4.4 Blurry receipt still yields accurate result

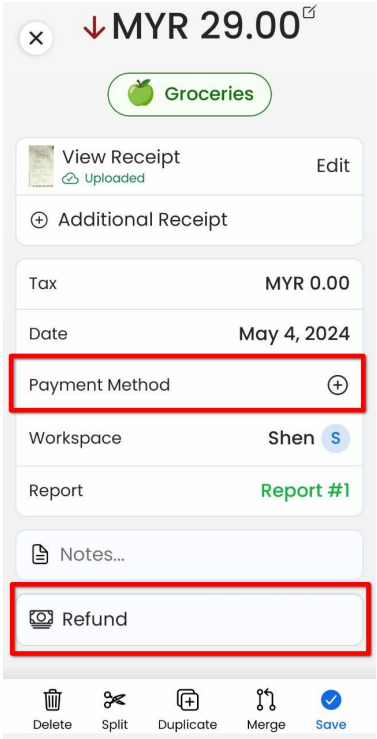


Figure 2.1.4.5 Able to add payment method and do refund

ii. Import email receipts

In some cases, receipts are sent to the user's email address. Easy expense uses third party APIs to connect with user's email address for instance Gmail. Once the application connects to user's email address, it will auto scan for receipts in the past 30 days for review as shown in Figure 2.1.4.6 – 2.1.4.7. Additionally, users can forward receipts to their supported email address which is upload@easy-expense.com to automatically add them to their account. Below Figure, shows one of the digital receipts that was extracted from user's personal email.

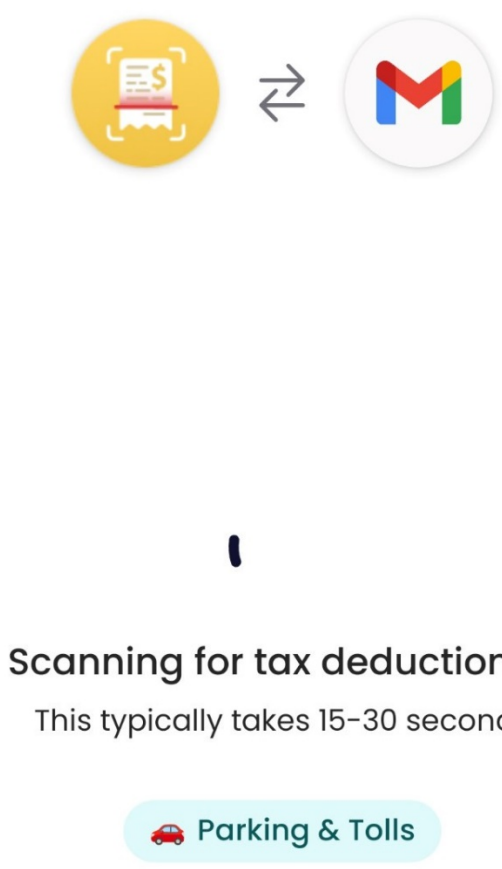


Figure 2.1.4.6 Scanning digital receipts from Gmail

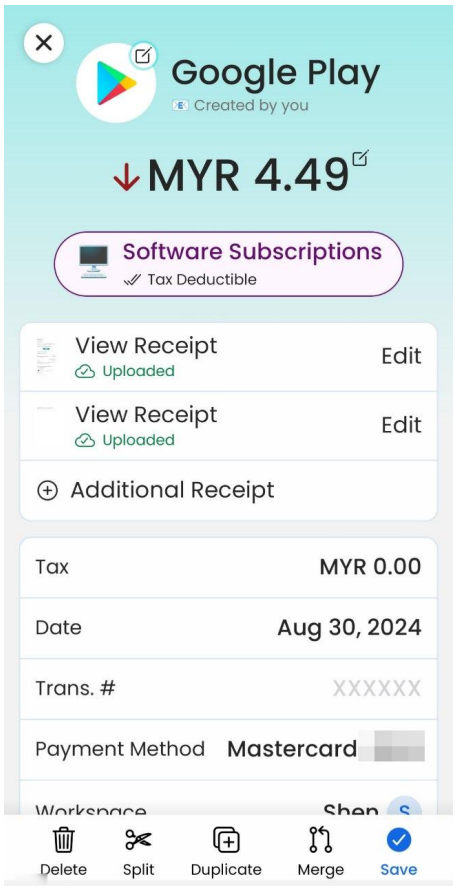


Figure 2.1.4.7 Google Play receipt found in Gmail

iii. Report generation

Besides having the option to share or download the expenses that are recorded. User is also able to generate different reports based on all their spendings as shown in Figure 2.1.4.8 below. User is also presented with a summary of the type of category and amount needed to pay. Also, they are presented with screenshots of each receipt for personal reference. These reports can be downloaded and exported via the tap of a button, also the summary reports in csv or pdf format can make filling taxes easier.

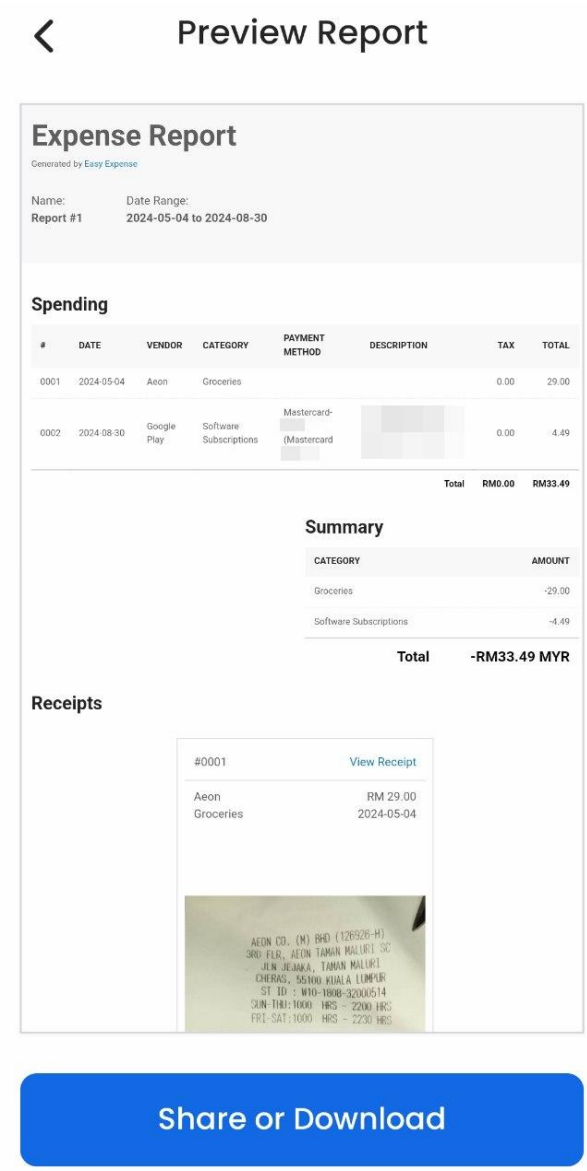


Figure 2.1.4.8 PDF expense report generated

Weakness

i. Lack of business document management

Besides, the basic document extraction like receipts. The application lacks in more sophisticated document management such as contracts, invoices or company reports. These additional documents are essential for more comprehensive financial management.

2.1.5 n8n

n8n is a popular open-source low code workflow automation tool as shown in Figure 2.1.5.1 below. n8n helps simplifies and streamline task by allowing users to connect to various application and services as shown in the Figure below. In the context of financial applications, users can leverage this open-source tool and create their own custom workflow for easier financial management and budgeting purposes. Also, it leverages automation and AI for mundane tasks like Figure 2.1.5.2 below. Hence, it is very suitable for repetitive financial tasks such as data entry, invoice processing and expense tracking. Overall, reducing manual errors and saves time.

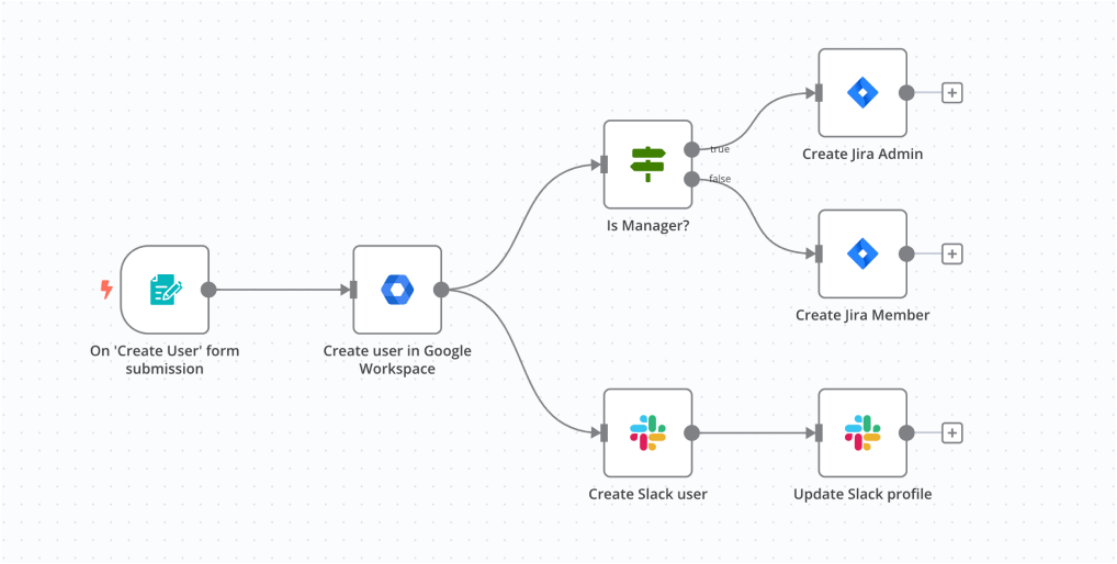


Figure 2.1.5.1 Example n8n low code workflow

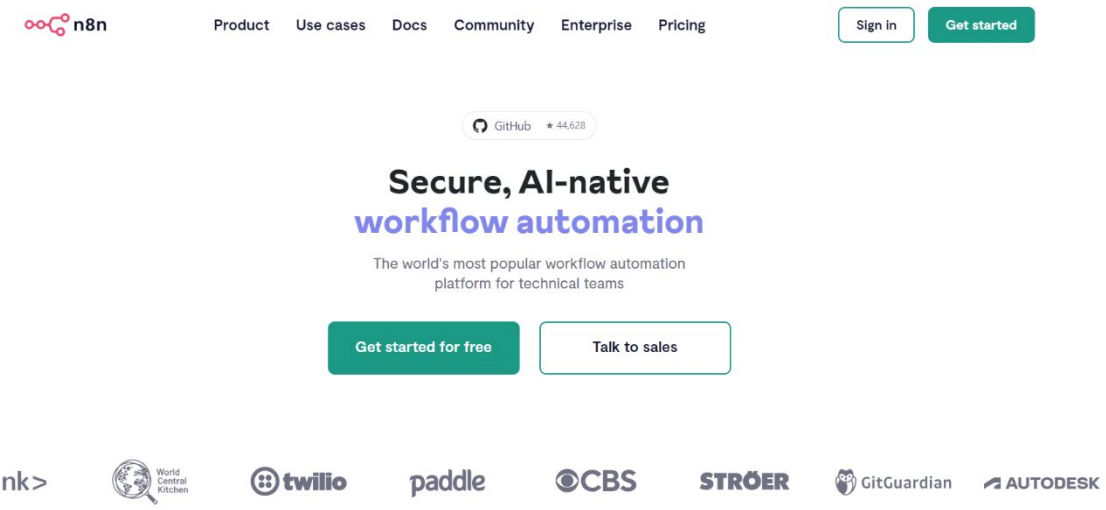


Figure 2.1.5.2 Integrate AI-native workflow

Strengths

i. Customisable workflow for financial management

The beauty of this workflow automation tool is that users can create their own custom workflow based on their needs. For instance, if the user wanted to extract expenses from emails and add to Google Sheets. They can create this workflow and even schedule it daily as shown in Figure 2.1.5.3 below. For such a workflow, it primarily involves the checking of the mailbox for new emails and checks if the Subject contains expense or receipt. It will then send the attachment to the external 3rd party API provider for processing. Finally updating the Google Sheet with corresponding values.

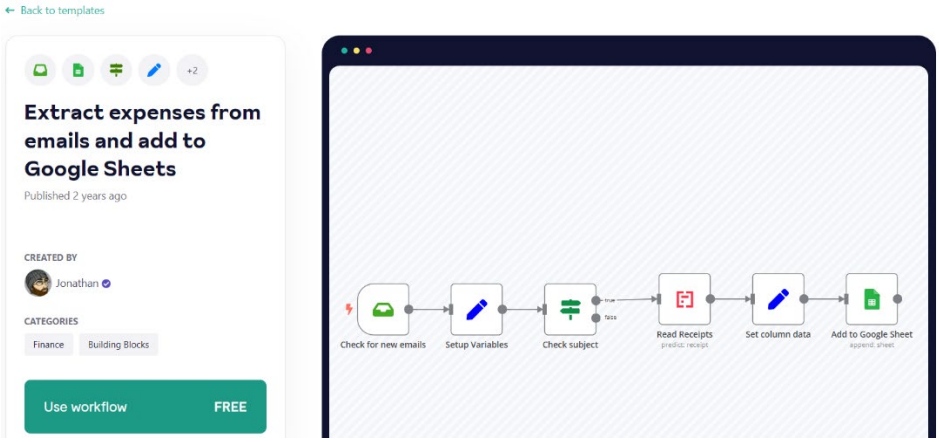


Figure 2.1.5.3 An example of finance workflow

ii. Many built in integration of financial tools

n8n supports more than 20 built in integration of financial tools as shown in the Figure 2.1.5.4 below. Besides, n8n also supports accounting software, payment gateways and banking APIs. This helps ensure for seamless data synchronisation and comprehensive financial tracking.

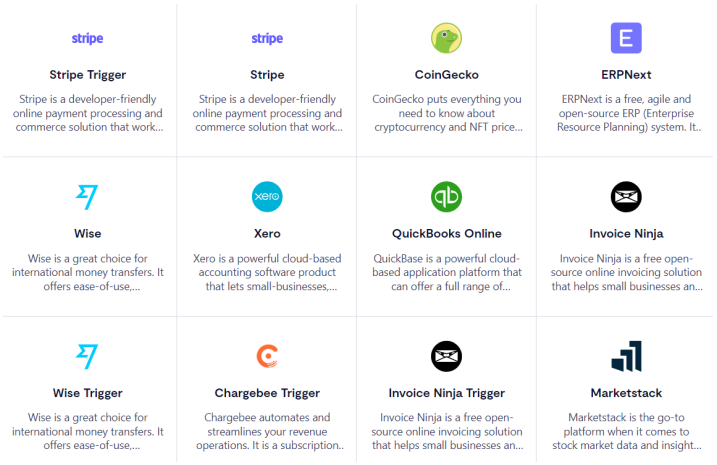


Figure 2.1.5.4 More than 20+ financial tools available

iii. Self-hosting ability

There are main options for using n8n, this includes self-hosting and the cloud option. Cloud options involve the workflows that are entirely hosted by n8n; however, a monthly fee needs to be incurred which ranges from $20 up to $50 as shown in Figure 2.1.5.5 below. But n8n also allows users or small businesses entities the option to self-host their n8n instances and workflows. This grants the users more superior data control.

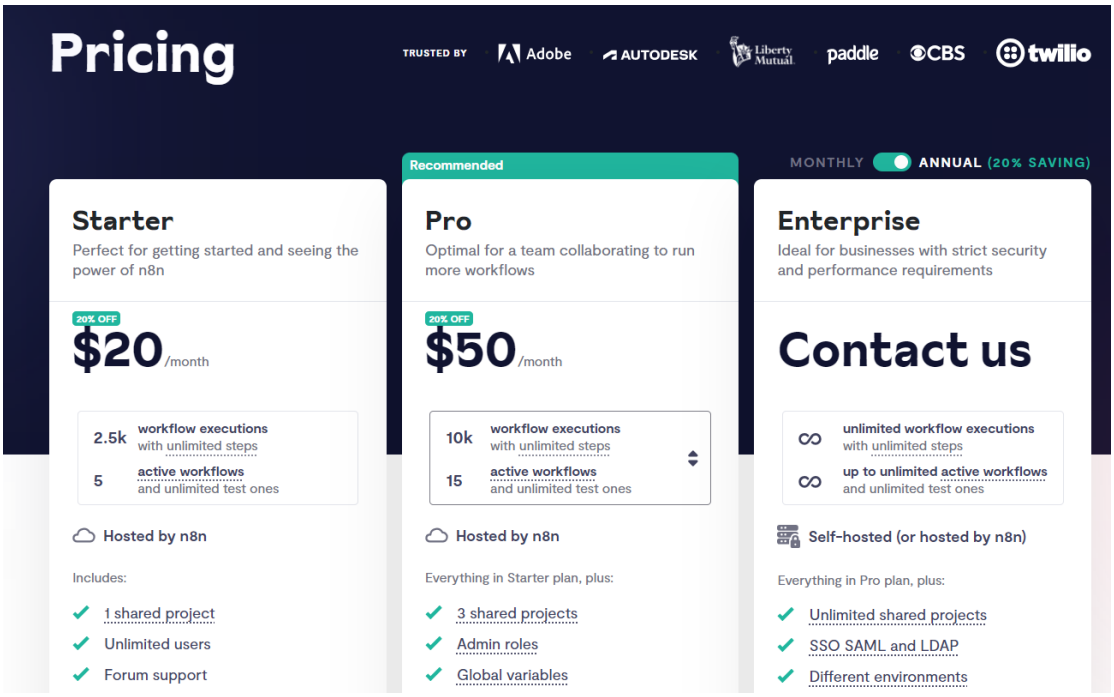


Figure 2.1.5.5 Pricing tier of cloud version of n8n

Weakness

i. Lack of integration with native android mobile applications

n8n is more inclined towards a web-based workflow automation tool. Hence, there is limited integration with the native mobile applications. The only way to integrate an n8n workflow into a mobile application is by going through a separate service by triggering specific n8n nodes. For instance, setting up a web hook in android which can accept HTTP POST requests from an n8n workflow.

2.2 Summary and table of comparison

In this chapter, 5 systems related to financial tracking, budgeting and planning were evaluated which were **Touch 'n Go**, **You Need a Budget (YNAB), Meow Money Manager**, **Easy Expense** and **n8n**. The overall strengths and weakness of the systems were summarised in Table 2.2.1 as shown below.

System / **Features**	**Touch 'n Go**	**You Need a Budget (YNAB)**	**Meow Money Manager**	**Easy Expense**	**n8n**
Supported platforms	Android, IOS	Android, IOS, Web	Android, IOS	Android, IOS, Web	Web based
Login/Signup	Yes	Yes	Yes	Yes	Yes
Nature of app	eWallet	Budgeting	Expense tracking, budgeting	Expense tracking with focus on receipt scanning	Workflow automation tool for financial management
Cost / Pricing Model	Free	Subscription-based (has free trial)	Subscription-based (has free trial)	Subscription-based (has free trial)	Free or paid cloud version
Connectivity requirement	Requires internet connection	Requires internet connection for bank	Does not require internet connection	Requires internet connection for bank synchronisation	Requires internet connection

		and web app synchronisation			
Data Export / Import	Limited import and export options (only can export recent transactions)	CSV export, import and bank details	CSV export	CSV export, import and bank details	CSV, JSON and other formats depending on workflow
Security features	Two-factor authentication (2FA), biometric login	Two-factor authentication (2FA), encryption	Pin protection, biometric login	Syncs and secures data to the cloud	Self-hosting which offers full control over data security
Customisable options	Limited customisation	Customisable budgets and categories	Customisable categories and themes	Customizable expense categories	Customisable workflows
Integration with other services	Limited integration	Integration with bank accounts	Integration with Google Drive for backup	Integration with bank, email scanning services	Extensive integration of APIs
Overall user interface and design	Complex design, not suitable for novice users. Too many	Cluttered design in the budget section and	Appealing design but has too many icons	Minimalist design and easy to navigate.	Simple interface design, but not so

	icons cause confusion.	hard to follow for beginners	which confuses the user.		easy to navigate when placing nodes.
Collaboration	Individual	Individual or family members	Individual	Individual or with team	Individual or with team
Learning curve	Moderate, requires familiarity of eWallets	High, not beginner friendly	Low	Low	High, requires knowledge of setting up workflows

Table 2.2.1 Comparison between systems

CHAPTER 3 Proposed Method/Approach

3.1 Use Case Diagram

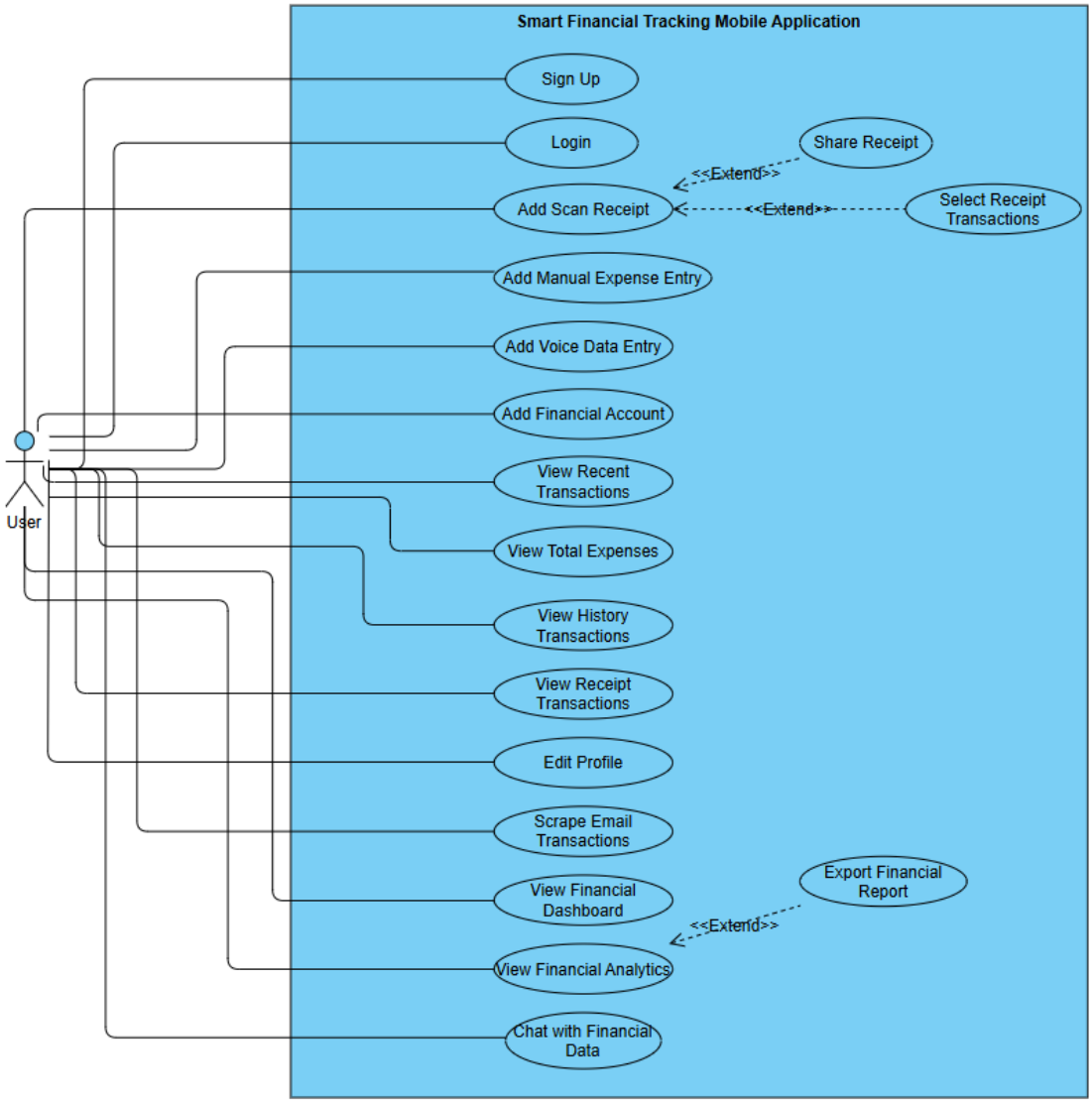


Figure 3.1.1 Use case diagram for Smart Financial Tracking Mobile Application

3.2 Use Case Description

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	1	
Use Case	Sign Up	
Use Case Description	This use case describes how user will register an account for the first time before using the mobile application.	
Actor	User	
Trigger	User clicks on the Sign-Up button	
Precondition	User has not created an account	
Scenario Name	**Step**	**Action**
Main Flow	1	User enters full name
	2	User enters full email
	3	User fills in password and confirm password
	4	User clicks on Sign Up button
	5	The system validates email, password fields entered.
	6	User is redirected to Login screen
Sub Flow	S-1a: Authenticate User Data in Firebase 1. The system securely saved user's credentials such as username, email and password	
Alternate Flow – Invalid email format	5a.1 User enters an invalid email without @ and .com 5a.2 System validates all other user fields 5a.3 System displays error message: "Please enter a valid email"	
Alternate Flow – Passwords do not match	5b.1 User enters password and confirm password do not match 5b.2 System validates all other user fields 5b.3 System displays error message: "Passwords do not match".	
Alternate Flow – Password is empty	5c.1 User did not enter password 5c.2 System validates other user fields	

	5c.3 System displays error message: "Please enter your password".
Alternate Flow – Email is empty	5d.1 User did not enter email 5d.2 System validates all other user fields 5d.3 System displays error message: "Please enter your email".
Alternate Flow – Password is weak	5e.1 User enters password and confirm password 5e.2 System validates all other user fields 5e.3 System displays error message: "Password must be at least 6 characters".

Table 3.2.1 Sign Up Use Case Description

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	2	
Use Case	Login	
Use Case Description	This use case describes how user will login an existing account upon registration.	
Actor	User	
Trigger	User clicks on the Login button	
Precondition	User has created an account but has not logged in	
Scenario Name	**Step**	**Action**
Main Flow	1	User enters email address
	2	User enters password
	3	System validates if account with inputted email address and password exists
	4	System redirects user to user to home screen
Sub Flow	-	
Alternate Flow – Invalid email address	3a.1 3a.2 3a.3	User enters an invalid email address System validates all other user fields System displays error message: “Enter a valid email address”
Alternate Flow – Invalid password	3b.1 3b.2 3b.3	User enters an invalid password System validates all other user fields System displays error message: "The password is invalid. Please try again".
Alternate Flow – Password is empty	3c.1 3c.2 3c.3	User did not enter password System validates other user fields System displays error message: "Please enter your password".
Alternate Flow – Email is empty	3d.1 3d.2 3d.3	User did not enter email System validates all other user fields System displays error message: "Please enter your email".

Table 3.2.2 Login use case description

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	3	
Use Case	Add Scan Receipt	
Use Case Description	This use case describes how user will scan and then add receipts expense entry	
Actor	User	
Trigger	User clicks on the centred camera icon	
Precondition	User is logged in and on receipts screen	
Relationship	Extended by Share Receipt use case Extended by Select Receipt Transactions use case	
Scenario Name	**Step**	**Action**
Main Flow	1	User clicks on the centred camera icon
	2	User captures the receipt with phone camera
	3	System processes receipts and extract merchant name, tax and date of transaction
	4	Scanned receipt is stored in receipts section
Sub Flow	S-1a: Student chooses manual camera capture 1. The user manually captures the receipt with the phone camera S-2a: Student chooses auto camera capture 1. The user automatically captures the receipt with the phone camera S-2a: Student chooses an image from gallery 1. The user chooses a receipt from gallery	
Alternate Flow – Cancel scan receipt	2a.1 User chooses to cancel the scan of receipt 2a.2 User exits the camera interface 2a.3 Receipt entry is not saved in receipts section	

Table 3.2.3 Add Scan Receipt use case description

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	4	
Use Case	Add Manual Expense Entry	
Use Case Description	This use case describes how user will manually add expense entry according to different categories	
Actor	User	
Trigger	User clicks on the plus button	
Precondition	User is logged in and on history screen	
Scenario Name	**Step**	**Action**
Main Flow	1	User clicks on the plus floating action button
	2	User clicks on the font shaped icon
	3	User selects a date for the expense
	4	User enters the expense in Ringgit Malaysia
	5	User selects from different categories
	6	Scanned receipt is stored in receipts section
Sub Flow	-	
Alternate Flow – Cancel manual expense entry	2a.1 2a.2 2a.3	User chooses to cancel the adding of expense Manual expense is not recorded Manual expense entry is not saved in history section

Table 3.2.4 Add Manual Expense Entry use case description

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	5	
Use Case	Add Voice Data Entry	
Use Case Description	This use case describes how user will use voice recognition to add expenses	
Actor	User	
Trigger	User clicks on the plus button	
Precondition	User is logged in and on history screen	
Scenario Name	**Step**	**Action**
Main Flow	1	User clicks on the plus floating action button
	2	User clicks on the microphone shaped button
	3	System displays dialog box and user proceeds to record voice
	4	System performs voice to text recognition and parses the text into expense, category and date
	5	Extracted text recognition result is displayed in history section
Sub Flow	-	
Alternate Flow – Cancel voice data expense entry	2a.1 2a.2 2a.3	User chooses to cancel the voice recognition entry Voice data expense is not recorded Voice data expense entry is not saved in history section

Table 3.2.5 Add Voice Data Entry use case description

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	6	
Use Case	Add Financial Account	
Use Case Description	This use case describes how user will add their financial accounts such as debit cards, credit cards and saving accounts	
Actor	User	
Trigger	User clicks on the plus button	
Precondition	User is logged in and on home screen	
Scenario Name	**Step**	**Action**
Main Flow	1	User clicks on the plus button near the card's header
	2	System displays add new card pop up
	3	User clicks on connect bank account button
	4	System links user to embedded web UI of Finverse API.
	5	User chooses test bank for data retrieval
	6	User enters test credentials provided by the API
	7	Finverse API provider validates test credentials and starts connecting user to test bank account
	8	Financial Info such as demo debit card, credit card, and statement savings are retrieved
Sub Flow	-	
Alternate Flow – Connection failure of Finverse API	7a.1	Finverse API service provider fails connection to the banking backend server
	7a.2	Finverse API service provider does not return any financial data
	7a.3	Bank account financial info is not saved in cards section

Table 3.2.6 Add Financial Account use case description

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	7	
Use Case	View Recent Transactions	
Use Case Description	This use case describes how user will view top 4 aggregated transactions from history screen and receipt screen	
Actor	User	
Trigger	User must have at least 1 added expense	
Precondition	User is logged in and on home screen	
Scenario Name	**Step**	**Action**
Main Flow	1	User must have added at least 1 expense either at history screen or receipt screen
	2	System validates if there are active expenses to be displayed on recent transactions section
	3	The recent transactions are sorted to most recent date of the transaction
	4	User can view the top available recent transactions at the home screen
Sub Flow	-	
Alternate Flow – Delete recent transactions	2a.1 2a.2	User deletes the most recent transaction from history screen or receipt screen Expense will be deleted from recent transactions section
Alternate Flow – Recent transactions is empty	2b.1 2b.2	User has not added any expenses at history screen and receipt screen System displays empty state to user with message “No Transactions Yet” and “Start tracking your expenses”

Table 3.2.7 View Recent Transactions use case description

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	8	
Use Case	View Total Expenses	
Use Case Description	This use case describes how user will view total expenses spent in Ringgit Malaysia	
Actor	User	
Trigger	User must have at least 1 added expense	
Precondition	User is logged in and on home screen	
Scenario Name	**Step**	**Action**
Main Flow	1	User must have added at least 1 expense either at history screen or receipt screen
	2	System validates there exists increment of total expenses from history screen and receipt screen
Sub Flow	-	
Alternate Flow – Decrement of total expenses	2a.1 2a.2	User deletes the transaction from history screen or receipt screen Expense will be deducted from total expenses
Alternate Flow – Total expenses is null	2b.1 2b.2	User has not added any expenses at history screen and receipt screen System displays RM 0.00 at total expenses section

Table 3.2.8 View Total Expenses use case description

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	9	
Use Case	View History Transactions	
Use Case Description	This use case describes how user will view expenses from manual data expenses and voice data recognition	
Actor	User	
Trigger	User must have at least 1 added expense from manual data expenses or voice data recognition	
Precondition	User is logged in and on history screen	
Scenario Name	**Step**	**Action**
Main Flow	1	User must have added at least 1 expense from manual data expenses or voice data recognition
	2	System validates if there exist at least 1 expense added into history screen
	3	User can view the expenses from manual data entry and voice data recognition entry
Sub Flow	-	
Alternate Flow – Delete expense	2a.1 2a.2	User deletes the transaction from history screen or receipt screen Expense will be deducted from total expenses
Alternate Flow – History screen is empty	2b.1 2b.2	User has not added any expenses at history screen System displays empty state to user with message “No Expenses Yet” and “Add your first expense by tapping the + button”

Table 3.2.9 View History Transactions use case description

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	10	
Use Case	View Receipt Transactions	
Use Case Description	This use case describes how user will view expenses from scanned receipts	
Actor	User	
Trigger	User must have at least 1 added receipt	
Precondition	User is logged in and on receipt screen	
Scenario Name	**Step**	**Action**
Main Flow	1	User must have added at least 1 receipt
	2	System validates if there exists at least 1 receipt scanned in the receipt screen
	3	User can view receipt information such as merchant name, data of transaction and total tax
Sub Flow	-	
Alternate Flow – Delete receipt expense	2a.1 2a.2	User deletes the receipt from receipt screen Expense will be deducted from total expenses
Alternate Flow – Receipts screen is empty	2b.1 2b.2	User has not added any receipts at receipts screen System displays empty state to user with message “No Receipts Yet” and “Start by scanning your first receipt”

Table 3.2.10 View Receipt Transactions use case description

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	11	
Use Case	Edit Profile	
Use Case Description	This use case describes how user will edit their profile such as profile picture	
Actor	User	
Trigger	User must click on the profile icon	
Precondition	User is logged in and on profile screen	
Scenario Name	**Step**	**Action**
Main Flow	1	User clicks on the profile icon
	2	System checks permission with the user before accessing the gallery
	3	User chooses a picture from gallery and selects it as profile picture
Sub Flow	-	
Alternate Flow – Declines gallery permission	2a.1 2a.2	User declines gallery permission User is not able to access gallery and upload profile picture

Table 3.2.11 Edit Profile use case descriptions

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	12	
Use Case	Scrape Email Transactions	
Use Case Description	This use case describes how user will connect to their Gmail account and scrape financial transactions	
Actor	User	
Trigger	User must click on Connect & Fetch Transactions button	
Precondition	User is logged in and on Email Scrape screen	
Scenario Name	**Step**	**Action**
Main Flow	1	User clicks on Connect & Fetch Transactions button
	2	System presents Gmail account connection interface
	3	User selects and authorises their Gmail account
	4	System initiate HTTP POST request to webhook
	5	External platform (n8n) processes the request and executes email scraping logic
	6	External platform extracts financial transaction data from emails and returns as JSON
	7	System receives and processes the JSON data
	8	System displays the scraped email transactions to the user
Sub Flow	-	
Alternate Flow – Authentication Error	3a.1	User declines to authorise Gmail access or authentication fails
	3a.2	System displays error message: "Authentication error. Please sign in again"
	3a.3	User remains back on Email Scrape Screen
Alternate Flow – Network Connection Error	4a.1	System fails to connect to external webhook
	4a.2	System displays error message: "Network error. Please check your connection"
	4a.3	User remains back on Email Scrape Screen
Alternate Flow – No Transactions	6a.1	External system processes emails but finds no transaction data

Found	6a.2	System presents empty state UI with option to try again
	6a.3	User may try to reconnect different Gmail account

Table 3.2.12 Scrape Email Transactions use case descriptions

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	13	
Use Case	View Financial Dashboard	
Use Case Description	This use case describes how user will view their aggregated current month expenses, spending trend (last 6 months), all time expenses breakdown, weekly breakdown and top spending categories	
Actor	User	
Trigger	User must click on the analytics button	
Precondition	User is logged in and on home screen	
Scenario Name	**Step**	**Action**
Main Flow	1	User clicks on the analytics button on home screen
	2	System loads the analytics screen with three tabs: Dashboard, Analytics, and Chat
	3	User selects Dashboard tab
	4	System loads user's financial data from the database
	5	System aggregates and calculates current month's total expenses
	6	System processes spending trends for the last 6 months
	7	System calculates expense breakdown by category for all time
	8	System generates weekly spending breakdown for the current month
	9	System identifies and sorts top spending categories
	10	System displays all the aggregated financial data in different sections
	11	User views their financial information in charts and graphs
Sub Flow	-	
Alternate Flow – No Financial Data Found	4a.1	System is unable to retrieve any financial data from database

	4a.2	System displays an empty state with message “No financial data available”
	4a.3	System suggests adding expenses before viewing analytics

Table 3.2.13 View Financial Dashboard use case descriptions

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	14	
Use Case	View Financial Analytics	
Use Case Description	This use case describes how user will view a summary analysis of selected month, year spending distribution and category breakdown.	
Actor	User	
Trigger	User must click on the analytics button	
Precondition	User is logged in and on home screen	
Relationship Type	Extended by Export Financial Report use case	
Scenario Name	**Step**	**Action**
Main Flow	1	User clicks on the analytics button on home screen
	2	System loads the analytics screen with three tabs: Dashboard, Analytics, and Chat
	3	User selects Analytics tab
	4	System displays month and year filter dropdown
	5	User selects desired month and year for analysis
	6	System retrieves financial data for the selected month, year
	7	System calculates and displays total spend amount, displays average transaction amount, displays total number of transactions
	8	System identifies and displays highest spending category
	9	System generates spending distribution visualisation and category breakdown
	10	System displays Export Financial Report button
Sub Flow	S-5a: Filtering by Different Month 1. User taps on month dropdown 2. System displays list of months 3. User selects a different month 4. System refreshes analytics data for the selected month 5. System updates all metrics and visualisation S-5b: Filtering by Different Year	

	1.	User taps on year dropdown
	2.	System displays list of years
	3.	User selects a different year
	4.	System refreshes analytics data for the selected year
	5.	System updates all metrics and visualisation
Alternate Flow – Error loading data	6a.1	System has an error while loading data
	6a.2	System displays error message: "Unable to load analytics data"
	6a.3	User can try loading data by choosing a different month, year
Alternate Flow – No data for selected month, year	7a.1	System finds no financial data for the month, year
	7a.2	System displays RM 0.00 for Total Spend
	7a.3	System displays RM 0.00 for Average Transaction
	7a.4	System displays 0 for number of Transactions
	7a.5	System displays "None" for Highest Category
	7a.6	System shows "No spending distribution to show" message in distribution chart placement
	7a.7	System shows "No category breakdown available" message in category breakdown placement
	7a.8	Export Financial Report button remains inactive
Alternate Flow – Unable to Export Report with no data	10a.1	User taps on Export Financial Report button when no data is available
	10a.2	System validates that there is no financial data to export
	10a.3	Export Financial Report button remains inactive

Table 3.2.14 View Financial Analytics use case descriptions

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	15	
Use Case	Chat with Financial Data	
Use Case Description	This use case describes how user will interact with the PDF RAG based chatbot and query based on the generated PDF Financial Report	
Actor	User	
Trigger	User must click on the analytics button	
Precondition	User is logged in and on home screen	
Scenario Name	**Step**	**Action**
Main Flow	1	User clicks on the analytics button on home screen
	2	System loads the analytics screen with three tabs: Dashboard, Analytics, and Chat
	3	User selects Chat tab
	4	System displays PDF document upload interface and chat query interface
	5	User clicks Select PDF button
	6	System opens default file browser for PDF document selection
	7	User selects the generated PDF Financial Report
	8	System uploads the selected document and displays the document name
	9	User enters a question in chat field
	10	The system sends the query and document reference for processing at FastAPI backend
	11	Backend processes PDF document
	12	System displays the generated answer to the user
Sub Flow	-	
Alternate Flow – No PDF Selected	5a.1	User attempts to ask a question without uploading a PDF document
	5a.2	Chat interface is inactive upon press send button
	5a.3	User must first select a PDF document to proceed

Alternate Flow – PDF Processing Error	11a.1	Backend encounters an error while processing the PDF document
	11a.2	Backend returns error code response to the system
	11a.3	System displays error message: “Unable to process the document. Please try again”
	11a.4	User can try reuploading same or different PDF Financial Report

Table 3.2.15 Chat with Financial Data

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	16	
Use Case	Share Receipt	
Use Case Description	This use case describes how user will share the receipt to different applications upon successful scanned receipt	
Actor	User	
Trigger	User clicks on scanned receipt from receipts screen	
Precondition	User is logged in and on receipts screen	
Relationship Type	Extends Add Scan Receipt use case	
Scenario Name	**Step**	**Action**
Main Flow	1	User clicks on a scanned receipt from the receipts screen
	2	User taps on the Share button at the bottom of the screen
	3	System displays sharing options / applications
	4	User selects a sharing option (messaging app, email, social media etc)
	5	System prepares the receipt image with default text “Shared Receipt”
	6	Selected application receives the receipt image
	7	User completes the sharing process in the selected application
Sub Flow	-	
Alternate Flow – Sharing Cancelled	4a.1 4a.2	User decides not to share and cancels the sharing dialog User returns to the original receipt dialog screen

Table 3.2.16 Share Receipt use case descriptions

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	17	
Use Case	Select Receipt Transactions	
Use Case Description	This use case describes how user will select the correct receipt transactions before saving the receipt or sharing the receipt	
Actor	User	
Trigger	User clicks on scanned receipt from receipts screen	
Precondition	User is logged in and on receipts screen	
Relationship Type	Extends Add Scan Receipt use case	
Scenario Name	**Step**	**Action**
Main Flow	1	User clicks on a scanned receipt from the receipts screen
	2	User taps on the Select button at the bottom of the screen
	3	System displays a list of detected transactions from the receipt
	4	User selects checkboxes for chosen transactions
	5	System calculates and displays the selected total amount
	6	User taps the Update button to confirm selections
	7	System updates the newest total amount
Sub Flow	S-4a: Edit Transaction Details 1. User taps on the edit icon at the top of the transaction list 2. System displays transaction items in editable format 3. User can modify description or amount 4. User taps update button to save changes S-4a: Add New Transaction Item 1. User taps on Add Item button 2. System displays new empty transaction fields 3. User enters new description and price 4. System adds new item to transaction list	

Alternate Flow – Remove Transaction Item	4a.1	User taps the remove (minus) button next to a transaction
	4a.2	System removes the item from the selection list
	4a.3	System recalculates the selected total amount
	4a.4	User continues selecting other items to delete or confirms with Update button
Alternate Flow – Cancel Transaction Selection	6a.1	User decides not to proceed with transaction selection
	6a.2	User taps the cancel button
	6a.3	System closes the transaction selection dialog
	6a.4	User returns to original receipt view screen

Table 3.2.17 Select Receipt Transactions use case descriptions

System	**Smart Financial Tracking Mobile Application**	
Use Case ID	18	
Use Case	Export Financial Report	
Use Case Description	This use case describes how user will export a financial report in PDF format from the analytics tab	
Actor	User	
Trigger	User must click on the analytics tab	
Precondition	User is logged in and on dashboard screen	
Relationship Type	Extends View Financial Analytics use case	
Scenario Name	**Step**	**Action**
Main Flow	1	User clicks on the analytics button on home screen
	2	System loads the analytics screen with three tabs: Dashboard, Analytics, and Chat
	3	User selects Analytics tab
	4	System displays month and year filter dropdown
	5	User selects desired month and year for analysis
	6	User clicks on Export Financial Report Button
Sub Flow	-	
Alternate Flow – Cancel PDF Export	6a.1 6a.2 6a.3	User decides not to proceed with PDF export User taps the back arrow User returns to original analytics tab

Table 3.2.18 Edit Profile use case descriptions

3.3 Mobile App Development

3.3.1 Mobile Emulator Configuration

Before the mobile application is deployed and tested on a real physical device. An Android Virtual Device (AVD) is created in Android Studio. To begin with, Android 14.0 SDK is installed as shown in Figure 3.3.1.1. The higher sdk versions of android provide access to latest android platform APIs. Then, as shown in Figure 3.3.1.2 select a device with suitable dimensions and install it will the corresponding system image. Lastly, as shown in Figure 3.3.1.3 rename the Android Virtual Device (AVD).

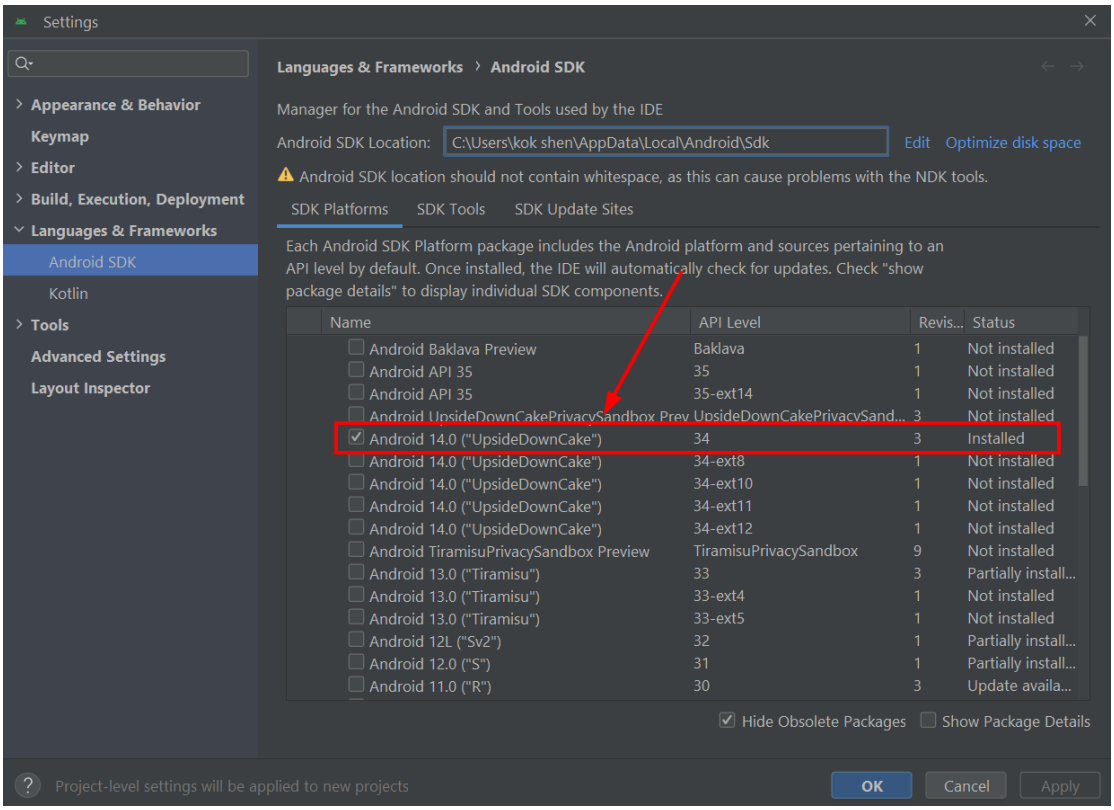


Figure 3.3.1.1 Installing Android 14.0 SDK

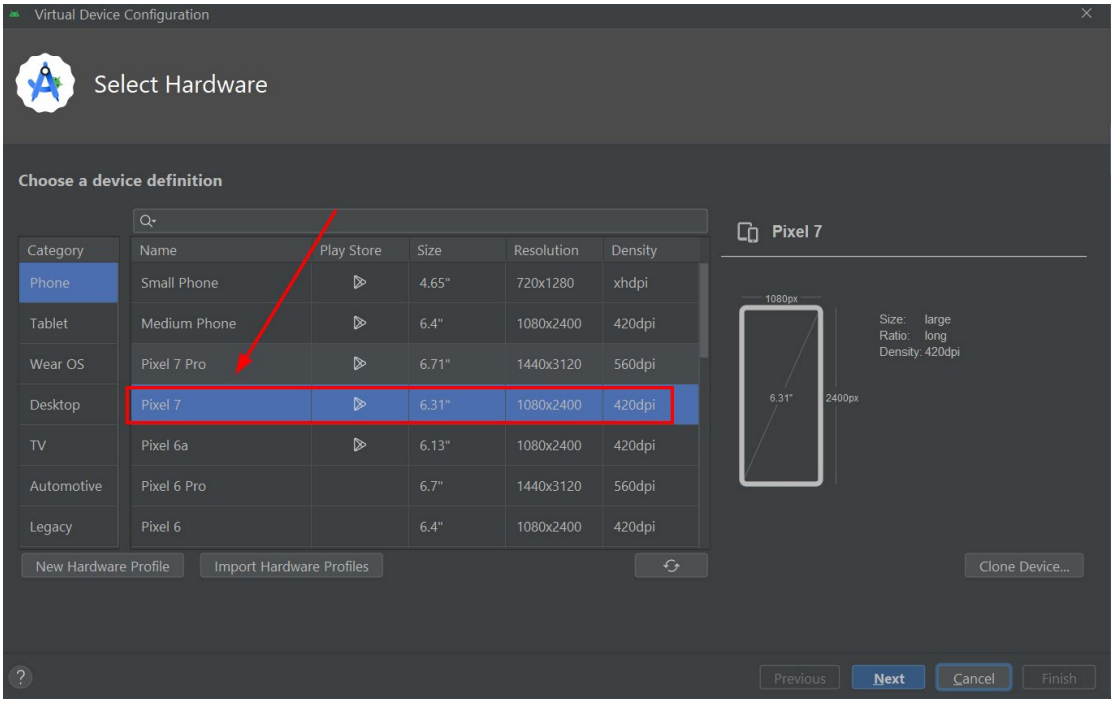


Figure 3.3.1.2 Choosing mobile emulator

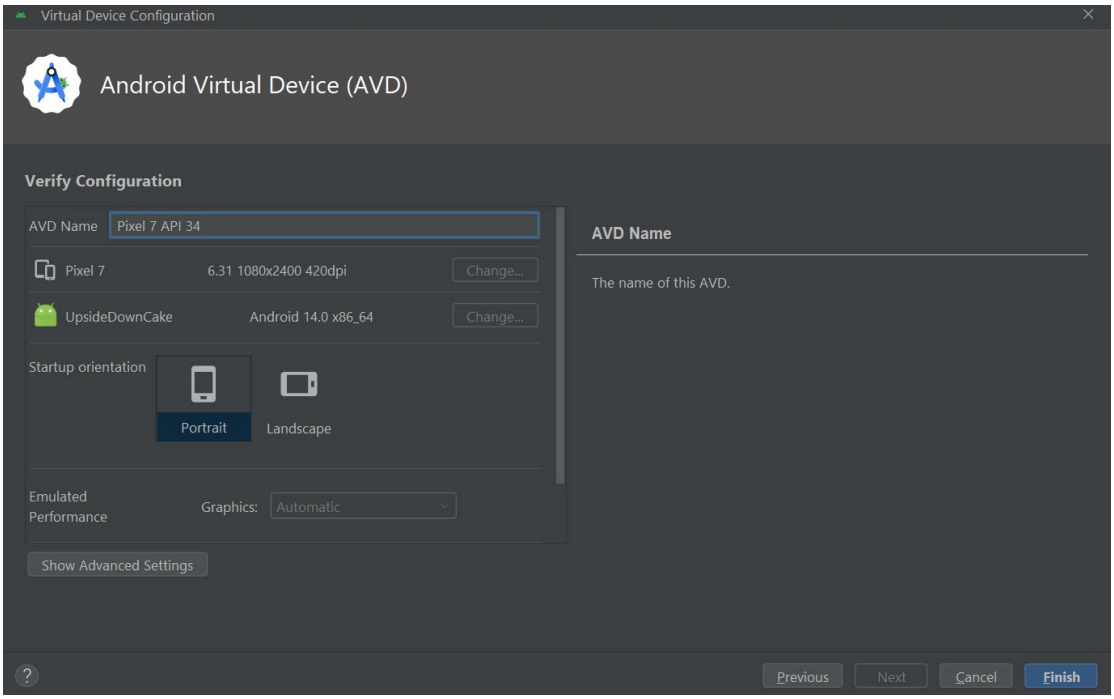


Figure 3.3.1.3 Configuring AVD name

3.3.2 Main Development Framework

In this project, the main development framework chosen was Flutter. Flutter is a cross-platform open-source UI framework that was developed by Google. The beauty of flutter is that developers can build apps using a single codebase for different platforms including IOS, Android and Web. Additionally, Flutter is suitable for rapid development, native OS performance and designing expressive user interface. To compensate the UI framework, dart programming language is used for development. Dart is a programming language which drives the Flutter framework, also it possesses a syntax like other object-oriented languages like Java and C# [7]. To note, Flutter combined with Dart can only fulfil the requirements of a frontend system. Instead of building the entire backend from scratch, Backend-as-a-service (Baas) cloud service tools such as Firebase Authentication and Cloud Firestore will support the basic backend requirements such as new user creation, user credential validation and data storage.

Figure 3.3.2.1 Flutter logo

3.3.3 Signup and Login Function Development

Upon entering the smart financial tracker, the user is redirected to the SignUp screen. To use the application, user must first enter their full name, email address, password and confirm password fields within the provided form. Upon pressing the SignUp button, the system validates the input fields to ensure all required information is provided, with correct email format and password meets the minimum length, also the confirmed password matches original password. The partial code snippets from Figure 3.3.3.1 to Figure 3.3.3.3 illustrate the logic of a new user signing up.

If the input is valid, the app will register the user with Firebase Authentication using the provided email and password. Furthermore, the new user profile with the provided name and email address is created with a new entry in Cloud Firestore based on the UserModel created as shown in Figure 3.3.3.1 below. UserModel ensures that the easy conversion of a suitable format for storage in Cloud Firestore. Finally. When the registration of the user is successful, a toast message informs the user of their success, and the user is redirected to the Login Screen.

```dart
class UserModel {
  final String? id;
  final String fullName;
  final String email;

  const UserModel({
    this.id,
    required this.email,
    required this.fullName,
  });

  // Convert to JSON for Firestore
  Map<String, dynamic> toJson() {
    return {
      "FullName": fullName,
      "Email": email,
    };
  }

  // Create from Firestore JSON
  factory UserModel.fromJson(Map<String, dynamic> json) {
    return UserModel(
      id: json['id'],
      fullName: json['FullName'] ?? '',
      email: json['Email'] ?? '',

    );
  }
}
```

Figure 3.3.3.1 user_model.dart

```dart
// Name Field
TextFormField(
  onSaved: (name) {
    if (name != null) widget.onNameChanged(name);
  },
  onChanged: widget.onNameChanged,
  validator: (value) {
    if (value == null || value.isEmpty) {
      return 'Please enter your name';
    }
    return null;
  },
  textInputAction: TextInputAction.next,
  decoration: InputDecoration(
    hintText: "Full name",
    prefixIcon: Padding(
      padding:
          const EdgeInsets.symmetric(vertical: defaultPadding * 0.75),
      child: Icon(
        Icons.person_outline,
        color: Theme.of(context)
            .textTheme
            .bodyLarge!
            .color!
            .withOpacity(0.3),
      ), // Icon
    ), // Padding
  ), // InputDecoration
), // TextFormField
```

Figure 3.3.3.2 sign_up_form.dart

```dart
Future<void> _signUp() async {
  if (_formKey.currentState!.validate()) {
    if (_password != _confirmPassword) {
      _showToast("Passwords do not match", isError: true);
      return;
    }
    try {
      // Try to create auth account
      UserCredential userCredential =
          await _auth.createUserWithEmailAndPassword(
        email: _email.trim(),
        password: _password.trim(),
      );
      if (userCredential.user != null) {
        // Update auth profile
        await userCredential.user!.updateDisplayName(_name.trim());
        // Create user model
        final user = UserModel(
          id: userCredential.user!.uid,
          email: _email.trim(),
          // password: _password.trim(),
          fullName: _name.trim(),
          // phoneNo: '',
        );
        // Save to Firestore using service
        await FirestoreService.createUser(user, userCredential.user!.uid);
        _showToast(
          "Account created successfully! Please login.",
          isError: false,
        );
        Future.delayed(const Duration(seconds: 2), () {
          Navigator.pushReplacementNamed(context, logInScreenRoute);
        }); // Future.delayed
      }
```

Figure 3.3.3.3 signup_screen.dart

Next, for the login function development, the LoginForm is a reusable widget designed to collect user credentials such as email and password. It uses a TextFormField widget for input, with built-in validation to ensure the email format is correct and the password field is not left empty as shown in Figure 3.3.3.4.

As for the main login screen, it will facilitate the login process combined with Firebase Authentication. When the screen loads, the app checks the user's login status by verifying both Firebase authentication and a locally stored flag in shared preferences as shown in code snippet in Figure 3.3.3.5. If the user is already logged in, the app navigates directly to the home screen without requiring manual input. Finally, when user submits their credentials, the login method handles the authentication process. It validates the form, authenticates the user with Firebase, and retrieves additional user information from Cloud Firestore.

```
TextFormField(
  onSaved: (email) {
    if (email != null) onEmailChanged(email);
  },
  onChanged: onEmailChanged,
  validator: emailValidator,
  textInputAction: TextInputAction.next,
  keyboardType: TextInputType.emailAddress,
  decoration: InputDecoration(
    hintText: "Email address",
    prefixIcon: Padding(
      padding: const EdgeInsets.symmetric(
          vertical: defaultPadding * 0.75), // EdgeInsets.symmetric
      child: SvgPicture.asset(
        "assets/icons/Message.svg",
        height: 24,
        width: 24,
        colorFilter: ColorFilter.mode(
            Theme.of(context)
                .textTheme
                .bodyLarge!
                .color!
                .withOpacity(0.3),
            BlendMode.srcIn), // ColorFilter.mode
      ), // SvgPicture.asset
    ), // Padding
  ), // InputDecoration
), // TextFormField
```

Figure 3.3.3.4 login_form.dart

```dart
Future<void> _login() async {
  if (_formKey.currentState!.validate()) {
    _formKey.currentState!.save();
    try {
      // First authenticate with Firebase Auth
      UserCredential userCredential = await _auth.signInWithEmailAndPassword(
        email: _email.trim(),
        password: _password.trim(),
      );
      // Get user data from Firestore
      final userData = await FirestoreService.getUser(userCredential.user!.uid);
      if (userData != null) {
        // Save login state with user data
        final SharedPreferences prefs = await SharedPreferences.getInstance();
        await prefs.setBool('isLoggedIn', true);
        await prefs.setString('userEmail', _email.trim());
        await prefs.setString('userId', userCredential.user?.uid ?? '');
        await prefs.setString('userName', userData.fullName); // Store user's name
        // Show success message
        _showToast("Login successful!", isError: false);
        // Navigate to Home screen
        if (mounted) {
          await Future.delayed(const Duration(seconds: 1));
          if (mounted) {
            Navigator.of(context).pushAndRemoveUntil(
              MaterialPageRoute(
                builder: (context) => const Home(),
                settings: const RouteSettings(name: 'Home'),
              ), // MaterialPageRoute
              (route) => false,
            );
          }
        }
```

Figure 3.3.3.5 login_screen.dart

3.3.4 Scan Receipt Function Development

The scan receipt feature is only triggered upon the toggle of camera button. The underlying algorithm combines 2 packages / libraries and requires the assistance of an external API provider named Langchain. Figure 3.3.4.1 below shows the 3 dependencies added in pubspec.yaml for the libraries and API provider to work.

pubspec.yaml

```
# the latest version available on pub.dev. To see which dependencies have newer
# versions available, run `flutter pub outdated`.
dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.8
  flutter_svg: ^2.0.10+1
  form_field_validator: ^1.1.0
  firebase_core: ^3.6.0
  firebase_auth: ^5.3.1
  fluttertoast: ^8.2.8
  http: ^1.2.2
  webview_flutter: ^4.10.0
  shared_preferences: ^2.3.3
  google_mlkit_document_scanner: ^0.3.0
  google_mlkit_text_recognition: ^0.14.0
  langchain: ^0.7.7+1
  langchain_openai: ^0.7.2+5
  timezone: ^0.10.0
  loading_animation_widget: ^1.3.0
  speech_to_text: ^7.0.0
  google_generative_ai: ^0.4.6
  provider: ^6.1.2
  iconsax: ^0.0.8
  calendar_date_picker2: ^1.1.7
  animations: ^2.0.11
  smooth_page_indicator: ^1.2.0+3
  image_picker: ^1.1.2
  permission_handler: ^11.3.1
  cloud_firestore: ^5.5.0
```

Figure 3.3.4.1 Added important dependencies

The main logic of the receipt scanning extraction comes from the document scanner service dart file as shown in the Figure 3.3.4.2 below. The startScanning method initializes the document scanner, captures the document image, and processes the image to extract text. Then the processImage method processes the scanned image to extract and structure text using the bounding box position of text lines as shown in the Figure 3.3.4.3 below. The method transforms the imagePath to an InputImage object, then proceeds to process the input image using a text recognizer to extract text blocks, lines and bounding box information. Finally, the lines are grouped together if their vertical positions are within 10 pixels of each other. This means that text lines close together on the vertical axis belong to the same paragraph.

```dart
Future<Map<String, dynamic>> startScanning() async {
  try {
    // Start document scanner
    final documentScanner = DocumentScanner(
      options: DocumentScannerOptions(
        pageLimit: 1,
        mode: ScannerMode.full,
        isGalleryImport: true,
        documentFormat: DocumentFormat.jpeg,
      ), // DocumentScannerOptions
    ); // DocumentScanner

    final result = await documentScanner.scanDocument();

    if (result.images.isNotEmpty) {
      final imagePath = result.images.first;
      final String text = await _processImage(imagePath);

      return {
        'success': true,
        'imagePath': imagePath,
        'scannedText': text,
      };
    }

    return {
      'success': false,
      'error': 'No image captured'
    };
```

Figure 3.3.4.2 document_scanner_service.dart - 1

```dart
Future<String> _processImage(String imagePath) async {
  try {
    final inputImage = InputImage.fromFilePath(imagePath);
    final recognizedText = await _textRecognizer.processImage(inputImage);

    // Use smaller threshold (3-5px) for vertical grouping
    Map<double, List<TextLine>> lineGroups = {};

    for (TextBlock block in recognizedText.blocks) {
      for (TextLine line in block.lines) {
        double y = line.boundingBox.top;
        // Group only very close lines (10px threshold)
        double key = lineGroups.keys.firstWhere(
          (k) => (k - y).abs() < 10,
          orElse: () => y,
        );
        lineGroups.putIfAbsent(key, () => []).add(line);
      }
    }
    // Sort and process groups efficiently
    StringBuffer processedText = StringBuffer();
    for (var lines in lineGroups.values) {
      lines.sort((a, b) => a.boundingBox.left.compareTo(b.boundingBox.left));
      processedText.writeln(lines.map((l) => l.text.trim()).join(' '));
    }
    return processedText.toString();
  } catch (e) {
    debugPrint('Error: $e');
    return 'Error processing image: $e';
  }
}
```

Figure 3.3.4.3 document_scanner_service.dart - 2

Figure 3.3.4.4 below shows the overall block diagram of how the scan receipt functionality is implemented. User will toggle the camera button and is allowed to choose an existing receipt from the gallery or capture the receipt on the spot. The receipt will then be processed by Google Mlkit Document Scanner for enhanced preprocessing such as auto cropping for accurate edge detection, perspective transform for different angled receipts, apply filters, remove shadows or stains. Then the preprocessed receipt jpeg is further fed into Google Mlkit Text recognition library for text extraction. Consequently, all the extracted text from the receipt is sent to Langchain API for high level processing. Langchain API will return structured receipt details such as merchant name, tax, total and date of transaction. The overall workflow leverages Google's pretrained machine learning libraries and Langchain API to automate receipt tracking effortlessly.

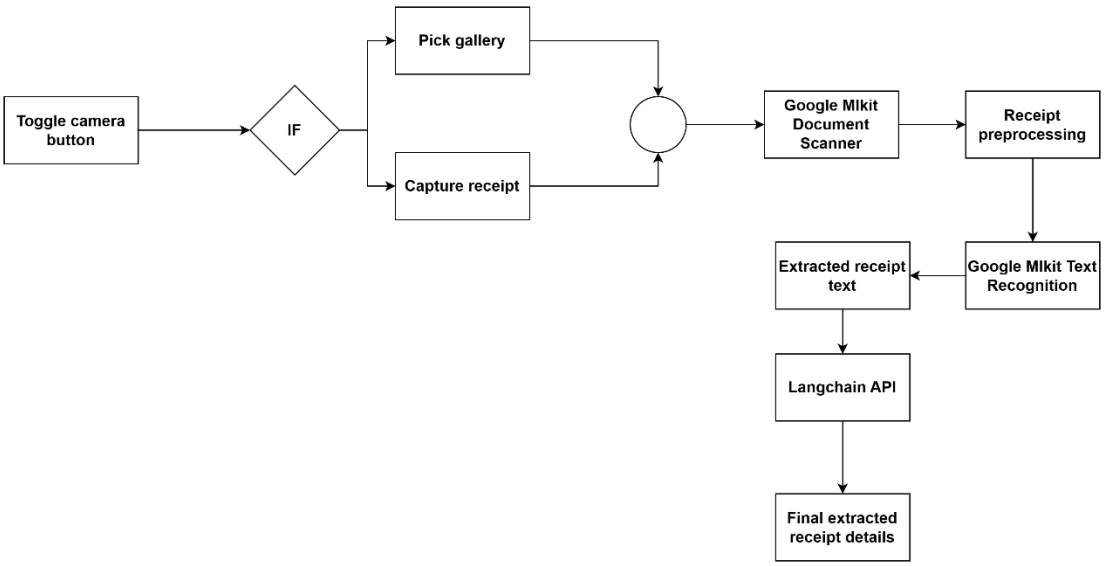


Figure 3.3.4.4 Block diagram of receipt scanning algorithm

3.3.5 Manual Expense Entry Development

The manual expense entry feature is triggered upon user select plus pop-up floating action button and then proceed to press the font size icon. The main screen developed was a custom ExpensesPage which allowed users to input, edit and categorise different expenses. Also, they can select their preferred date. Partial code snippets of the manual expense entry development are shown from Figure 3.3.5.1 to Figure 3.3.5.2.

```
Expanded(
  child: GridView.count(
    crossAxisCount: 5,
    padding: const EdgeInsets.symmetric(
        horizontal: 12, vertical: 8), // EdgeInsets.symmetric
    mainAxisSpacing: 8,
    crossAxisSpacing: 8,
    childAspectRatio: 1.1,
    children: [
      _CategoryItem(
        icon: Icons.restaurant,
        label: 'Food',
        color: Colors.red,
        isSelected: _selectedCategory == 'Food',
        onTap: () => _handleCategorySelected('Food'),
      ), // _CategoryItem
      _CategoryItem(
        icon: Icons.local_cafe,
        label: 'Drinks',
        color: Colors.teal,
        isSelected: _selectedCategory == 'Drinks',
        onTap: () => _handleCategorySelected('Drinks'),
      ), // _CategoryItem
      _CategoryItem(
        icon: Icons.directions_bus,
        label: 'Transport',
        color: Colors.blue,
        isSelected: _selectedCategory == 'Transport',
        onTap: () => _handleCategorySelected('Transport'),
      ), // _CategoryItem
      _CategoryItem(
        icon: Icons.shopping_bag,
        label: 'Shopping',
        color: Colors.purple,
        isSelected: _selectedCategory == 'Shopping',
        onTap: () => _handleCategorySelected('Shopping'),
      ), // _CategoryItem
```

Figure 3.3.5.1 expenses_screen.dart - 1

```dart
void _handleConfirm() {
  try {
    final numericAmount = double.parse(_amount) * (_isNegative ? -1 : 1);

    // Check for zero amount
    if (numericAmount == 0) {
      // Calculate bottom padding based on keypad height
      final keypadHeight = 5 * 56.0; // 4 rows * button height
      final bottomPadding =
          MediaQuery.of(context).viewInsets.bottom + keypadHeight;

      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: const Text('Amount cannot be zero'),
          behavior: SnackBarBehavior.floating,
          margin: EdgeInsets.only(
            bottom: bottomPadding,
            left: 16,
            right: 16,
          ), // EdgeInsets.only
        ), // SnackBar
      );
      return;
    }
```

Figure 3.3.5.2 expenses_screen.dart

3.3.6 Voice Recognition Entry Development

The voice data entry feature is triggered upon user select plus pop-up floating action button and then proceed to press the microphone icon. The dart files developed mainly handled the voice to text recognition of the user. Once the user's voice is parsed into text format as shown in Figure 3.3.6.1. The text will undergo additional text processing such as regular expression filtering to identify common expenses, date and category patterns as shown in Figure 3.3.6.2.

```
void _initSpeech() async {
  _speechEnabled = await _speechToText.initialize(
    onStatus: (status) {
      print('Speech status: $status');
      if (status == 'done' || status == 'notListening') {
        setState(() => _isListening = false);

        // Process text only when the mic stops
        if (_recognizedText.isNotEmpty) {
          _processText(_recognizedText);
        }
      }
    },
    onError: (error) {
      print('Speech error: $error');
      setState(() => _isListening = false);
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text(error.errorMsg)),
      );
    },
  );
  setState(() {});
}
```

Figure 3.3.6.1 voice_recognition.dart

```dart
static String? extractAmount(String text) {
  // Match RM amounts e.g., RM50.30, RM50
  final regexRM = RegExp(r'\bRM\s?(\d{1,3}(?:\.\d{1,2})?)\b', caseSensitive: false);
  final matchRM = regexRM.firstMatch(text);
  if (matchRM != null) return matchRM.group(1);

  // Match formats like "RM50 90" or "rm50 90"
  final regexRMSeparated = RegExp(r'\bRM\s?(\d{1,3})\s?(\d{1,2})\b', caseSensitive: false);
  final matchRMSeparated = regexRMSeparated.firstMatch(text);
  if (matchRMSeparated != null) {
    double total = double.parse(matchRMSeparated.group(1)!) +
        double.parse(matchRMSeparated.group(2)!) / 100;
    return total.toStringAsFixed(2);
  }

  // Match amounts like "50 ringgit 90 sen" or "50 ringgit 90 cents"
  final regexRinggitCents = RegExp(
      r'\b(\d+)\s*ringgits?\s*(?:and|&)?\s*(\d+)\s*(?:cents|sen)\b', caseSensitive: false);
  final matchRinggitCents = regexRinggitCents.firstMatch(text);
  if (matchRinggitCents != null) {
    double total = double.parse(matchRinggitCents.group(1)!) +
        double.parse(matchRinggitCents.group(2)!) / 100;
    return total.toStringAsFixed(2);
  }
```

Figure 3.3.6.2 text_processing.dart

3.3.7 Add Financial Account Development

The add financial account development feature allows users to simulate the adding of demo / test financial accounts such as statement savings, checking accounts and credit card balance. This feature requires 3rd party integration of an open banking financial API called Finverse. The main dart files included the file to store API credentials for testing purpose such as Figure 3.3.7.1, another dart file was created to facilitate the process of data retrieval. Below Figure 3.3.7.2, illustrates an overview of exchange of excess tokens and account data retrieval. Similarly, for the usage of a live bank account identical workflow can be applied but strong encryption practices and a robust backend must be implemented to secure user data retrieval.

```dart
// API credentials for test account

class ApiConstants {
  static const String apiHost = "https://api.prod.finverse.net";
  static const String clientId = [redacted]
  static const String clientSecret =
  [redacted]
  static const String redirectUri = "https://developer.prod.finverse.net/sink";
  static const String appName = "Smart Financial Tracker";
  static const String customerAppId = [redacted]
}
```

Figure 3.3.7.1 api_constants.dart

```dart
Future<String> getCustomerAccessToken() async {
  print("Starting getCustomerAccessToken");
  if (_customerAccessToken != null) {
    print("Returning cached customer access token");
    return _customerAccessToken!;
  }

  try {
    final response = await http.post(
      Uri.parse('${ApiConstants.apiHost}/auth/customer/token'),
      headers: {'Content-Type': 'application/json'},
      body: jsonEncode({
        'client_id': ApiConstants.clientId,
        'client_secret': ApiConstants.clientSecret,
        'grant_type': 'client_credentials',
      }),
    );

    print("Customer access token response status: ${response.statusCode}");
    print("Customer access token response body: ${response.body}");

    if (response.statusCode == 200) {
      final jsonResponse = jsonDecode(response.body);
      _customerAccessToken = jsonResponse['access_token'];
      print("Successfully obtained customer access token");
      return _customerAccessToken!;
    } else {
      throw Exception('Failed to get customer access token: ${response.body}');
    }
  } catch (e) {
    print("Error in getCustomerAccessToken: $e");
    rethrow;
  }
}
```

Figure 3.3.7.2 finverse_service.dart

Below Figure 3.3.7.3 shows an overall sequence diagram indicating the high-level flow of the client and the Finverse API. The client app first authenticates with Finverse API to obtain the Customer Access Token. Then, user logs in by linking the test institution through the generated link. After user completes the linking process, the client app exchanges the authorisation code for a Login Identity Access Token. Finally, the client app uses the Login Identity Access Token to retrieve user financial data from Finverse API. The data is further processed and displayed to the user.

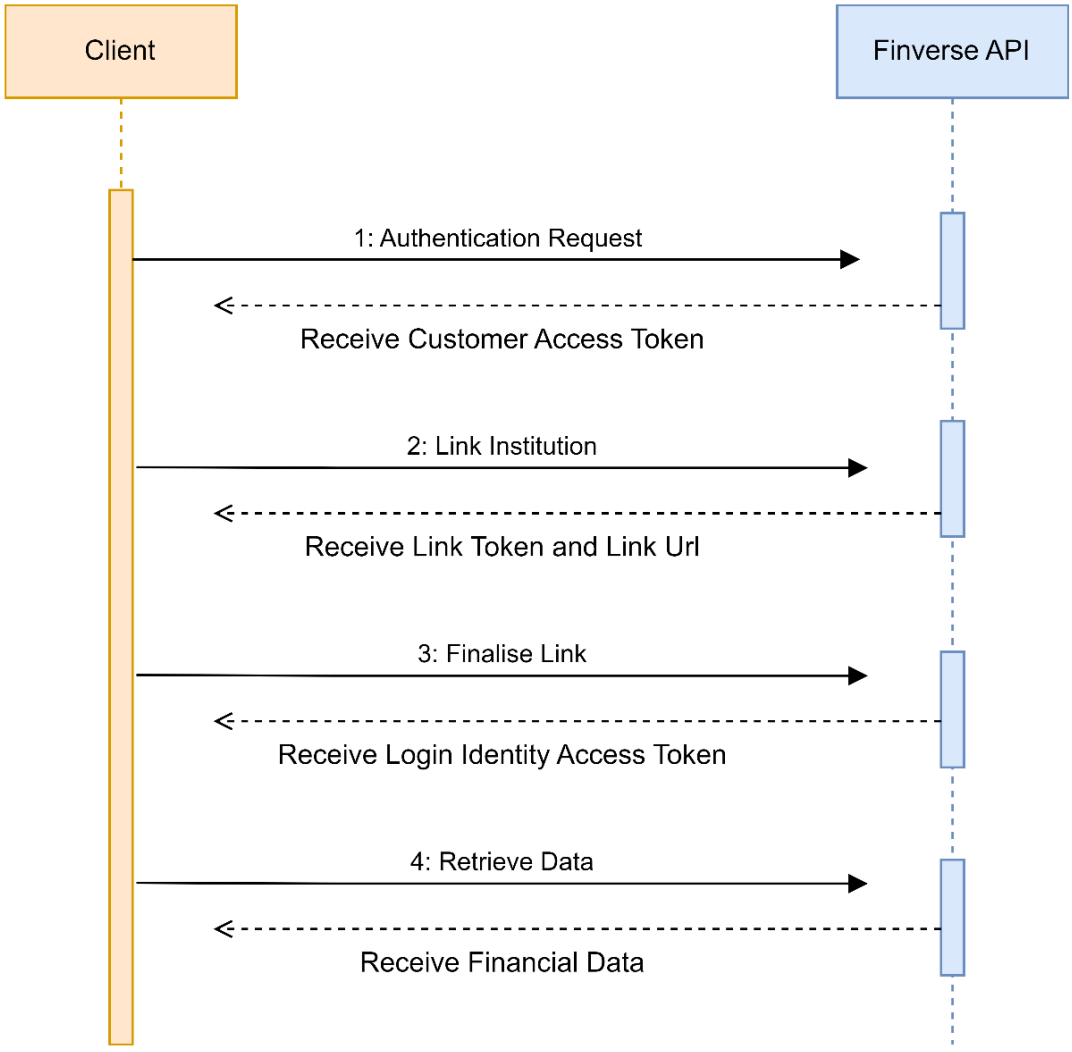


Figure 3.3.7.3 High level sequence diagram between Client and Finverse API

3.3.8 Receipt Sharing Development

The main logic of the Receipt Sharing is wrapping the receipt fields using a Screenshot widget. As shown in the partial code snippet in Figure 3.3.8.1 below, the application will essentially capture the whole page including the widget fields with the newly updated details such as Tax, Total, and Date.

```
// WIDGET TO SCREENSHOT
Screenshot(
  controller: _screenshotController,
  child: Container(
    // Explicit container for white background
    color: Colors.white,
    padding: const EdgeInsets.symmetric(
        vertical:
            16.0), // Add vertical padding inside screenshot if needed // EdgeInsets.symmetric
    child: Column(
      mainAxisSize: MainAxisSize
          .min, // Important for Column height fitting content
      crossAxisAlignment: CrossAxisAlignment.stretch,
      children: [
        // Merchant name
        Padding(
          padding: const EdgeInsets.symmetric(
              horizontal: 16), // EdgeInsets.symmetric
          child: TextField(
            controller: _merchantNameController,
            textAlign: TextAlign.center,
            style: const TextStyle(
              fontSize: 32,
              fontWeight: FontWeight.w500,
            ), // TextStyle
```

Figure 3.3.8.1 Screenshot widget wrapping entire container fields

Then, after wrapping the whole container, upon the user press the Share button it will trigger the (onPressed: _shareReceipt) method as shown in Figure 3.3.8.2 below.

```
          _buildActionButton(
            icon: Icons.share,
            label: 'Share',
            onPressed: _shareReceipt, // Share
            color: Colors.orange,
          ),
        ],
      ), // Row
    ), // Container
  ],
), // Column
```

Figure 3.3.8.2 _buildActionButton code snippet

Below Figure 3.3.8.3 shows the _shareReceipt method used to develop the logic for sharing the receipt image to other applications on the user's device. To note, this is one of the methods for sharing content which is like accessing the native android API such as Intents (Implicit Intent).

```
Future<void> _shareReceipt() async {
  try {
    // Capture the widget as PNG bytes
    Uint8List? bytes = await _screenshotController.capture(pixelRatio: 3.0);
    if (bytes == null) return;

    // Write bytes to a temp file
    final dir = await getTemporaryDirectory();
    final file = await File(
            '${dir.path}/receipt_${DateTime.now().millisecondsSinceEpoch}.png')
        .writeAsBytes(bytes);

    // Share via the SharePlus.instance.share() API
    final result = await SharePlus.instance.share(ShareParams(
      text: 'Shared receipt',
      files: [XFile(file.path)], // wrap path in XFile
    )); // ShareParams

  } catch (e) {
    print('Share error: $e');
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('Couldn\'t share receipt')),
    );
  }
}
```

Figure 3.3.8.3 _shareReceipt method code snippet

3.3.9 Financial Analytics Development

Below Figure 3.3.9.1 illustrates a partial code snippet of the implementation of the custom tab navigation interface for the financial analytics dashboard. The structure mainly utilises a row of interactive tab headers where each tab is built with an Expanded widget containing a GestureDetector for handling user taps. When a tab is selected, the setState function updates the _selectedTabIndex variable to reflect the current selection.

```
children: [
  // Tabs row
  Padding(
    padding: const EdgeInsets.fromLTRB(16, 16, 16, 0),
    child: Row(
      children: [
        Expanded(
          child: GestureDetector(
            onTap: () => setState(() => _selectedTabIndex = 0),
            child: Container(
              padding: const EdgeInsets.symmetric(vertical: 8),
              decoration: BoxDecoration(
                border: Border(
                  bottom: BorderSide(
                    color: _selectedTabIndex == 0
                        ? Colors.black
                        : Colors.transparent,
                    width: 2.0,
                  ), // BorderSide
                ), // Border
              ), // BoxDecoration
              child: Text(
                'Dashboard',
                textAlign: TextAlign.center,
                style: TextStyle(
                  fontWeight: FontWeight.bold,
                  color: _selectedTabIndex == 0
                      ? Colors.black
                      : Colors.grey,
                ), // TextStyle
              ), // Text
            ), // Container
          ), // GestureDetector
        ), // Expanded
```

Figure 3.3.9.1 Financial analytics tab navigation code snippet

As shown from Figure 3.3.9.2 to Figure 3.3.9.5, the _processDataForAnalysis method transforms raw expense data into structured analytics through a comprehensive pipeline, starting with the initialization and total calculation in Figure 3.3.9.2, the method creates a dynamic results container and calculates the overall spending using fold operations, filtering current month expenses for separate analysis. Then, a comparative analytics segment from Figure 3.3.9.2 calculates the month-over-month changes with careful handling of edge cases, while simultaneously categorizing expenses to identify top spending areas.

Next, the weekly data aggregation as shown in Figure 3.3.9.4 divides the current month into four weeks based on transaction dates, creating formatted data structures for visualisation. Finally, the historical trend analysis in Figure 3.3.9.5 generates a six-month spending history by iterating backwards through time periods, filtering expenses by month, and calculating period totals. Overall, the comprehensive data processing pipeline enables users to get immediate insights and historical context for their spending patterns.

```
312   // Data processing for unified expense model
313   Map<String, dynamic> _processDataForAnalysis(List<Expense> expenses) {
314     final results = <String, dynamic>{};
315     final now = DateTime.now();
316
317     // Calculate total spending (overall)
318     double totalSpent =
319         expenses.fold(0.0, (sum, expense) => sum + expense.amount);
320     results['totalSpent'] = totalSpent;
321
322     // Current month totals
323     final currentMonthExpenses = expenses
324         .where((e) => e.date.month == now.month && e.date.year == now.year)
325         .toList();
326
327     double currentMonthTotal =
328         currentMonthExpenses.fold(0.0, (sum, expense) => sum + expense.amount);
329     results['currentMonthTotal'] = currentMonthTotal;
330
331     // Previous month totals for comparison
332     final lastMonthDateTime = DateTime(now.year, now.month - 1, now.day);
333     final lastMonthExpenses = expenses
334         .where((e) =>
335             e.date.month == lastMonthDateTime.month &&
336             e.date.year == lastMonthDateTime.year)
337         .toList();
338
339     double lastMonthTotal =
340         lastMonthExpenses.fold(0.0, (sum, expense) => sum + expense.amount);
341     results['lastMonthTotal'] = lastMonthTotal;
```

Figure 3.3.9.2 Data processing calculation code snippet - 1

```
// Month-over-month change
double monthlyChange = 0;
double changePercent = 0;
if (lastMonthTotal > 0) {
  monthlyChange = currentMonthTotal - lastMonthTotal;
  changePercent = (monthlyChange / lastMonthTotal) * 100;
} else if (currentMonthTotal > 0) {
  changePercent = 100.0;
  monthlyChange = currentMonthTotal;
}
results['monthlyChange'] = monthlyChange;
results['changePercent'] = changePercent;

// Categorize ALL expenses
Map<String, double> categorizedExpensesAllTime = {};
for (var expense in expenses) {
  final category = expense.category.isEmpty ? 'Other' : expense.category;
  categorizedExpensesAllTime[category] =
      (categorizedExpensesAllTime[category] ?? 0) + expense.amount;
}
results['categorizedExpenses'] = categorizedExpensesAllTime;

// Top spending categories (All Time)
List<MapEntry<String, double>> sortedCategoriesAllTime = [];
if (categorizedExpensesAllTime.isNotEmpty) {
  sortedCategoriesAllTime = categorizedExpensesAllTime.entries.toList()
    ..sort((a, b) => b.value.compareTo(a.value));
}
results['topCategories'] = sortedCategoriesAllTime.take(3).toList();
```

Figure 3.3.9.3 Data processing calculation code snippet - 2

```
// Weekly spending for CURRENT month
List<Map<String, dynamic>> weeklyDataCurrentMonth = [];
Map<int, double> weeklyTotals = {1: 0.0, 2: 0.0, 3: 0.0, 4: 0.0};
for (var expense in currentMonthExpenses) {
  int weekNumber;
  if (expense.date.day <= 7)
    weekNumber = 1;
  else if (expense.date.day <= 14)
    weekNumber = 2;
  else if (expense.date.day <= 21)
    weekNumber = 3;
  else
    weekNumber = 4;
  weeklyTotals[weekNumber] =
      (weeklyTotals[weekNumber] ?? 0) + expense.amount;
}
weeklyDataCurrentMonth = [
  {'name': 'Week 1', 'amount': weeklyTotals[1] ?? 0.0},
  {'name': 'Week 2', 'amount': weeklyTotals[2] ?? 0.0},
  {'name': 'Week 3', 'amount': weeklyTotals[3] ?? 0.0},
  {'name': 'Week 4', 'amount': weeklyTotals[4] ?? 0.0},
];
results['weeklyData'] = weeklyDataCurrentMonth;
```

Figure 3.3.9.4 Data processing calculation code snippet - 3

```
395     results['weeklyData'] = weeklyDataCurrentMonth;
396
397     // Monthly data for trend chart (last 6 months)
398     List<Map<String, dynamic>> monthlyTrend = [];
399     for (int i = 5; i >= 0; i--) {
400       final monthDateTime = DateTime(now.year, now.month - i, 1);
401       final monthName = DateFormat('MMM').format(monthDateTime);
402       final monthExpenses = expenses
403           .where((e) =>
404               e.date.month == monthDateTime.month &&
405               e.date.year == monthDateTime.year)
406           .toList();
407
408       double monthTotal =
409           monthExpenses.fold(0.0, (sum, expense) => sum + expense.amount);
410       monthlyTrend.add({'month': monthName, 'amount': monthTotal});
411     }
412     results['monthlyTrend'] = monthlyTrend;
413
414     return results;
415   }
```

Figure 3.3.9.5 Data processing calculation code snippet - 4

3.3.10 Expense Report Generation

The main development logic of expense report generation is users can export their selected month, year transaction into a PDF document. Below Figure 3.3.10.1 shows the partial code snippet for the development of custom button widget for user to press before exporting the PDF. As for Figure 3.3.10.2, it represents the main ReportGenerator class which is responsible for generating PDF expense reports in the Flutter application. The required parameters which provide all necessary data for generating a financial report include expenses, receipts, categorised Expenses, month and year. At the beginning of the PDF document, it is also initialised with metadata such as title, author, creator, subject, and relevant keywords for enhancing searchability and organisation.

```
727  Widget _buildExportButton(
728    BuildContext context,
729    List<Expense> filteredExpenses,
730    List<Expense> receiptExpenses,
731    Map<String, double> categorizedExpenses,
732  ) {
733    return Container(
734      width: double.infinity,
735      margin: const EdgeInsets.only(top: 10, bottom: 20),
736      child: ElevatedButton.icon(
737        icon: const Icon(Icons.picture_as_pdf, size: 18),
738        label: const Text(
739          'Export Financial Report',
740          style: TextStyle(fontSize: 15, fontWeight: FontWeight.bold),
741        ), // Text
742        onPressed: (filteredExpenses.isEmpty && receiptExpenses.isEmpty)
743            ? null
744            : () async {
745                final messenger = ScaffoldMessenger.of(context);
746                messenger.showSnackBar(const SnackBar(
747                  content: Row(children: [
748                    CircularProgressIndicator(strokeWidth: 3),
749                    SizedBox(width: 15),
750                    Text('Generating report...')
751                  ]), // Row
752                  duration: Duration(seconds: 10))); // SnackBar
```

Figure 3.3.10.1 _buildExportButton widget code snippet

```dart
import 'dart:io';
import 'package:pdf/pdf.dart';
import 'package:pdf/widgets.dart' as pw;
import '../../models/expense.dart';
import '../../services/save_and_open_document.dart';

class ReportGenerator {
  static Future<File> generateExpenseReport({
    required List<Expense> expenses,
    required List<dynamic> receipts, // Changed to dynamic for compatibility
    required Map<String, double> categorizedExpenses,
    required String month,
    required int year,
  }) async {
    final pdf = pw.Document(
      title: 'Financial Report - $month $year',
      author: 'Expense Tracker App',
      creator: 'Expense Tracker App',
      subject: 'Monthly Expense Analysis',
      keywords: 'finance, expenses, budget, report, $month, $year',
    );
```

Figure 3.3.10.2 ReportGenerator class code snippet

3.4 n8n Workflow

n8n is an open-source low code workflow automation tool which mainly uses drag-n-drop nodes for customising personal / business uses cases. As the current trend of AI is increasing, this workflow automation platform has also incorporated native AI capabilities for creating Agentic workflows, allowing users to build multi-step AI agents that can call custom tools and integrate with any Large Language Model (LLM). The current platform currently supports self-hosting, cloud deployment and integration with more than 400+ APIs. n8n gives users complete control over their data and deployment options while providing technical teams the ability to switch between visual building and custom code such as JavaScript or Python. With the convenience of rapid development using n8n, n8n was used for implanting one of the core features of the financial tracker – Email Transaction Scraping.

Figure 3.4 n8n Supports code and visual building

3.4.1 Scrape Email Transactions Workflow

One of the core features of this mobile app is that it allows users to scrape their monthly transactions present in their Gmail. The Figure 3.4.1.1 below shows a brief workflow of Scraping Gmail Transactions using n8n platform. For easier visualisation it is further separated into 3 segments which are**:**

1. Authentication and Initial Email Retrieval
2. Email Parsing
3. Final Parsing of JSON Output

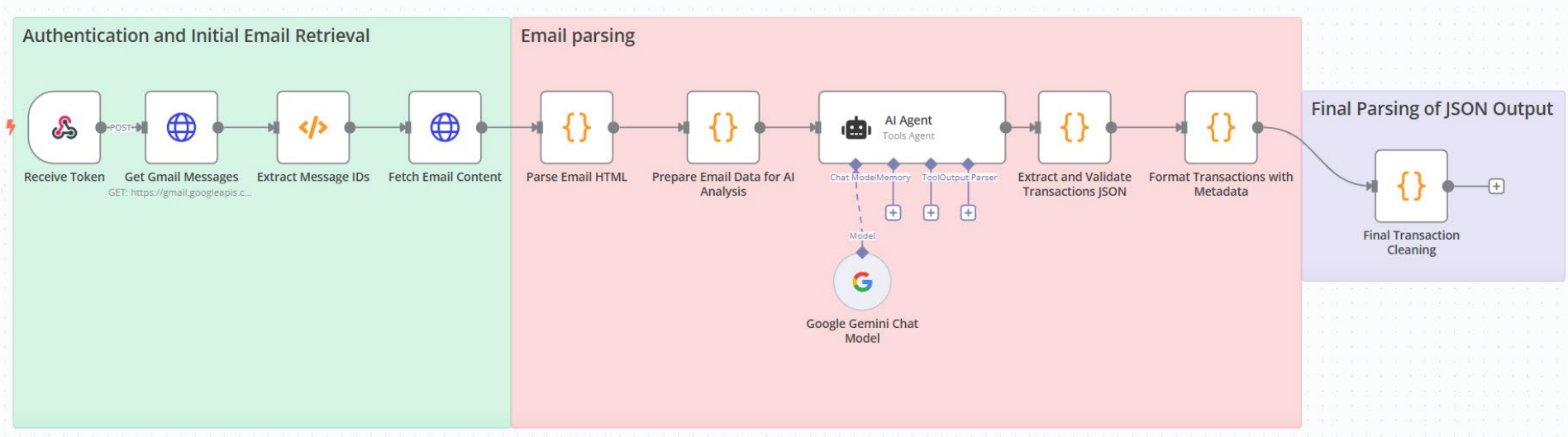


Figure 3.4.1.1 Scraping Email Transactions Workflow using n8n

The first segment handles the authentication and retrieval of emails from Gmail. In the frontend Flutter application, the OAuth Authentication process is initiated which is then triggers the entire workflow once the user has chosen their respective Gmail account. Once the user authorises the app, Flutter application receives the OAuth access token, and the token is then sent to the webhook endpoint in n8n via secure HTTP Post request. The webhook "Receive Token node" then accepts the OAuth Access Token originating from the user. Once authenticated, the "Get Gmail Messages node" performs a Gmail API request to fetch recent emails using specific search query such as looking for emails after 1st April 2025 but before May 1st, 2025, with subjects containing "Receipt", "Invoice", "Order", "Subscription", or "Transaction", while excluding promotional emails or subscriptions. The "Extract Message IDs node" then processes the search results and extracts individual message IDs, which are passed to the "Fetch Email Content node" to retrieve the full content of each email.

In the second segment, the workflow processes the raw email content to make it suitable for AI analysis. The "Parse Email HTML node" extracts key email metadata (subject, sender, date) and the HTML content from the email body, converting it from base64 format. Then, the "Prepare Email Data for AI Analysis node" cleans up the HTML by removing redundant scripts, styles and html tags, leaving only essential elements like tables and paragraphs. The cleaned-up HTML is then passed into the "AI Agent node", which uses Google Gemini to analyse the email and extract specific transaction information such as (date, type, amount, description) in a structured JSON format. The JSON undergoes 2 more steps of parsing in the "Extract and Validate Transactions JSON" node parsing the AI's output, extracting valid JSON and filtering out incomplete entries. Finally, the "Format Transactions with Metadata" node enhances the data with information such as (formatted currencies) and calculates summary statistics of parsed transactions.

In the last segment, the workflow processes the previously parsed transactions for the purpose of deduplication, removing zero-value transactions, and calculates subtotals grouped by different potentially different / same currencies. After the entire workflow has ended, the initial webhook sends back the cleaned JSON to Flutter frontend completing the entire workflow. At the frontend, JSON can be rendered using widgets like transaction list.

3.4.2 Flutter Frontend Setup

For the frontend setup, n8n workflow is constantly waiting for a request via webhook call. The below Flutter frontend code snippet as shown in figure 3.4.2.1 is very crucial in ensuring for different deployments including live usage, local usage such as testing on emulator and physical device. For live usage, code line 181 will be uncommented. However, for the rest of local testing purposes code line 182 or 183 are mainly used. For the Uri on line 181, this is the production Uri that would be used in the final demonstration. It points to a live n8n instance hosted on Railway (cloud platform service). The Uri is accessible from anywhere on the internet and is secured with HTTPS protocol. On line 182, this Uri is used for testing with a physical device. The IP address 192.168.0.8 represents a local development machine on the same Wi-Fi network as the physical device, it will use port 5678 which is the port n8n server is running on. As for line 183, this Uri is specifically for android emulator testing, the IP address of 10.0.2.2 is how android emulators reference the host machine's localhost. Therefore, allowing the emulated app to connect to services running on the development machine.

```
176 Future<void> _hitWebhook(String token) async {
177   if (_userId == null) return;
178
179   // Call webhook endpoint
180   final resp = await http.post(
181     // Uri.parse('https://[redacted]/webhook/receive-token'), // Live usage
182     // Uri.parse('http://192.168.0.8:5678/webhook-test/receive-token'),   // physical device
183     Uri.parse('http://10.0.2.2:5678/webhook-test/receive-token'), // emulator
184     headers: {'Content-Type': 'application/json'},
185     body: jsonEncode({'access_token': token}),
186   );
187
```

Figure 3.4.2.1 Different Uri for different deployment options

Additionally, in Figure 3.4.2.2 shows the main token acquisition process for webhook authentication. The method mainly handles the OAuth authentication flow for Google Sign-In to obtain an access token for Gmail API access. As for Figure 3.4.2.3, the connectAndFetch method mainly serves as the primary interface for initiating the entire email scraping process in the frontend such as checking for user authentication, managing state changes, token acquisition and webhook interaction.

```dart
// Fetch Google access-token (sign-in if needed)
Future<String?> _getAccessToken() async {
  try {
    GoogleSignInAccount? acct = _googleSignIn.currentUser;
    acct ??= await _googleSignIn.signInSilently();
    acct ??= await _googleSignIn.signIn();
    if (acct == null) return null;

    final auth = await acct.authentication;
    return auth.accessToken;
  } catch (e) {
    _handleError('Google sign-in failed', e);
    return null;
  }
}
```

Figure 3.4.2.2 Access token acquisition process

```dart
Future<void> connectAndFetch() async {
  if (_userId == null) {
    _errorMessage = 'Please sign in to continue';
    notifyListeners();
    return;
  }

  _isLoading = true;
  _isUpdating = false;
  _errorMessage = null;
  notifyListeners();

  try {
    await _googleSignIn.signOut(); // force fresh login

    final token = await _getAccessToken();
    if (token == null) {
      _errorMessage = 'Sign-in cancelled';
      _isLoading = false;
      notifyListeners();
      return;
    }

    await _hitWebhook(token);

    // After successful fetch, load all email expenses
    _localConvertedExpenses = await getAllEmailScrapedExpenses();

    Fluttertoast.showToast(
        msg: 'Email data fetched successfully',
        backgroundColor: Colors.green,
        textColor: Colors.white);
  } catch (e) {
    _handleError('Connection failed', e);
  } finally {
    _isLoading = false;
    _isBackgroundProcessing = false;
    notifyListeners();
  }
}
```

Figure 3.4.2.3 Initiating entire email scraping process

3.5 RAG Based PDF Chatbot

The RAG Based PDF Chatbot mainly works on idea of Retrieval Augmented Generation which essentially helps to reduce the possibility of LLM hallucination. To support the inferencing of the chatbot, the chatbot is created in python FastAPI and deployed on Modal which is a serverless cloud for developers to deploy and run generative AI models, large-scale batch workflows, and job queues.

3.5.1 FastAPI Setup

As shown in Figure 3.5.1.1, the first step in establishing the entire process is by setting up a FastAPI endpoint for the Flutter frontend to access the URL endpoint. This setup initialises a FastAPI application for Modal Live deployment, including security countermeasures such as CORS configuration.

```
12  app = App("rag-backend")
13  app.image = Image.debian_slim().poetry_install_from_file("./pyproject.toml")
14
15  @app.function(secrets=[Secret.from_dotenv()])
16  @asgi_app()
17  def endpoint():
18
19      app = FastAPI(
20          title="RAG API",
21          description="A simple RAG API",
22          version="0.1",
23      )
24
25      origins = [
26          "http://localhost:3000",
27          "http://10.0.2.2:8000",
28          "http://10.0.2.2",
29          "*"  # For development only
30      ]
31
32      app.add_middleware(
33          CORSMiddleware,
34          allow_origins=origins,
35          allow_methods=["*"],
36          allow_headers=["*"],
37      )
```

Figure 3.5.1.1 Initial Setup for FastAPI endpoint

From Figure 3.5.1.2 – Figure 3.5.1.3 shows the websocket configuration, in Figure 3.5.1.2 the code demonstrates the websocket endpoints such as '/async_chat' which enables real-time, bidirectional communication, allowing the application to stream responses incrementally as they are generated. In the frontend Flutter, this creates a more interactive user experience where answers appear word-by-word like chat interfaces. Then, in '/chat' HTTP POST endpoint It includes the answer and supporting document chucks which is returned in a single JSON response after processing finishes. In Figure 3.4.1.3, the '/upload_pdf' endpoint handles the document ingestion for RAG. The POST endpoint accepts multipart form data containing both the PDF file and a document name identifier. After handling the upload file process. The endpoint then calls the vector database for processing the PDF, and chunks it into segments, creating vector embeddings, and stores them in the Qdrant vector database.

```
42      # websocket endpoint
43      @app.websocket('/async_chat')
44      async def async_chat(websocket: WebSocket):
45          await websocket.accept()
46          while True:
47              question = await websocket.receive_text()
48              async for event in async_get_answer_and_docs(question):
49                  if event["event_type"] == "done":
50                      await websocket.close()
51                      return
52                  else:
53                      await websocket.send_text(json.dumps(event))
54
55      @app.post("/chat")
56      def chat(message: Message):
57          # Extract the string message from the Message object
58          question = message.message
59
60          # Now pass the string to get_answer_and_docs
61          response = get_answer_and_docs(question)
62          response_content = {
63              "question": question,
64              "answer": response["answer"],
65              "documents":[doc.dict() for doc in response["context"]]
66          }
67          return JSONResponse(content=response_content, status_code=200)
```

Figure 3.5.1.2 Websocket endpoints - 1

```python
@app.post("/upload_pdf")
async def upload_pdf(
    file: UploadFile = File(...),
    document_name: str = Form(...)
):
    try:
        # Create a temporary file
        with tempfile.NamedTemporaryFile(delete=False, suffix=".pdf") as temp_file:
            # Write uploaded file content to temp file
            content = await file.read()
            temp_file.write(content)
            temp_file_path = temp_file.name

        # Process the PDF file
        response = upload_pdf_to_collection(temp_file_path, document_name)

        # Clean up the temporary file
        os.unlink(temp_file_path)

        return JSONResponse(
            content={
                "filename": file.filename,
                "document_name": document_name,
                "response": response
            },
            status_code=200
        )
    except Exception as e:
        return JSONResponse(content={"error": str(e)}, status_code=500)

return app
```

Figure 3.5.1.3 Websocket endpoints – 1

As shown from Figure 3.5.1.4 – 3.5.1.7 below, it demonstrates the vector retrieval pipeline that powers the RAG system. In Figure 3.5.1.4 shows the retriever configuration which ensures the system fetches the top four document chunks with similarity score only exceeding 0.80, focusing on only PDF and using enhanced HNSW parameters for deeper recall in Qdrant. In Figure 3.5.1.5, depicts the text chunking strategy implementing a RecursiveCharacterTextSplitter that segments documents into 1000-character chunks with 20-character overlaps to maintain integrity between adjacent sections. Moreover, Figure 3.5.1.6 presents the embedding generation function, which transforms text into numerical vector representations using OpenAI's text embedding-ada-002 model, for similarity comparisons. Finally, Figure 3.5.1.7 illustrates the vector search implementation, which converts user queries into the same vector space for retrieval of the most semantically relevant document chunks from Qdrant collection.

```
retriever = vector_store.as_retriever(
    search_kwargs={
        "k": 4,                    # top 4 chunks
        "score_threshold": 0.80    # only keep very similar hits
    },
    filter={"source_type": "pdf"},  # PDF
    search_params={"hnsw_ef": 128}  # deeper HNSW recall in Qdrant
)
```

Figure 3.5.1.4 Retriever configuration

```
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size=1000,
    chunk_overlap=20,
    length_function=len
)
```

Figure 3.5.1.5 Text Chunking configuration

```
def get_embedding(text: str):
    return client.embeddings.create(
        model="text-embedding-ada-002",
        input=text,
    ).data[0].embedding
```

Figure 3.5.1.6 Embedding generation

```
def qdrant_search(query: str):
    vector_search = get_embedding(query)
    docs = client.search(
        collection_name=collection_name,
        query_vector=vector_search,
        limit=4
    )
    return docs
```

Figure 3.5.1.7 Vector search Implementation

3.5.2 Flutter Frontend Setup

As shown in Figure 3.5.2.1, the uploadPDF function implements an upload workflow, the function builds a multipart HTTP request targeting the backend endpoint, packaging the PDF file with appropriate content type specifications and metadata. It also implements validation by checking for both file selection and document name before initiating the upload process. Then in Figure 3.5.2.2, shows the basic question input interface that users interact with to query their uploaded documents. The UI features styled text fields with appropriate visual feedback states, clear placeholder text, and conditional enabling based on the application's streaming or loading state.

```dart
/// returns true on success
Future<bool> uploadPDF(String documentName) async {
  if (_selectedFile == null) {
    _setUploadStatus('Please select a PDF file first');
    return false;
  }
  if (documentName.trim().isEmpty) {
    _setUploadStatus('Please enter a document name');
    return false;
  }

  _setUploading(true);
  _setUploadStatus('Uploading…');

  try {
    final req = http.MultipartRequest(
      'POST',
      Uri.parse('$baseUrl/upload_pdf'),
    )
      ..files.add(await http.MultipartFile.fromPath(
        'file',
        _selectedFile!.path!,
        filename: _selectedFile!.name,
        contentType: MediaType('application', 'pdf'),
      ))
      ..fields['document_name'] = documentName.trim();

    final res = await http.Response.fromStream(await req.send());

    if (res.statusCode == 200) {
      final data = jsonDecode(res.body);
      _setUploadStatus('Successfully uploaded: ${data['response']}');
      setSelectedFile(null); // clears and notifies
      return true;
    } else {
      _setUploadStatus('Error ${res.statusCode}: ${res.body}');
      return false;
    }
  } catch (e) {
    _setUploadStatus('Error: $e');
    return false;
  } finally {
    _setUploading(false);    // ALWAYS executed
  }
}
```

Figure 3.5.2.1 uploadPDF functionality

```
margin: EdgeInsets.zero,
shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(12)),
elevation: 2,
color: Colors.grey[100],
child: Padding(
  padding: const EdgeInsets.all(16),
  child: Column(
    children: [
      TextField(
        controller: _questionController,
        // added
        onChanged: (_) => setState(() {}),
        enabled: !chatProvider.isLoading && !chatProvider.isStreaming,
        decoration: InputDecoration(
          hintText: 'Ask a question about your documents',
          hintStyle: TextStyle(color: Colors.grey[600]),
          border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(10),
            borderSide: BorderSide(color: Colors.grey.shade400),
          ), // OutlineInputBorder
          focusedBorder: OutlineInputBorder(
            borderRadius: BorderRadius.circular(10),
            borderSide:
                BorderSide(color: Theme.of(context).colorScheme.primary),
          ), // OutlineInputBorder
          filled: true,
          fillColor: Colors.white,
          contentPadding:
              const EdgeInsets.symmetric(horizontal: 16, vertical: 12),
          suffixIcon: Icon(Icons.search, color: Colors.grey[600]),
        ), // InputDecoration
```

Figure 3.5.2.2 Chat with PDF interface

3.6 Backend Hosting

Since the development of the mobile application requires an extensive use of APIs. A few backend options were used, this included Railway, Modal and FastAPI for creating a web deployable API.

3.6.1 Railway

Railway as shown in Figure 3.6.1.1 – 3.6.1.2 was mainly used for deploying the n8n workflow instance, once Railway has deployed the n8n workflow instance. The flutter frontend application can call the URL endpoint as an API. Railway provides a scalable and reliable platform for hosting these workflows, offering automatic scaling and robust logging capabilities that simplify monitoring and debugging.

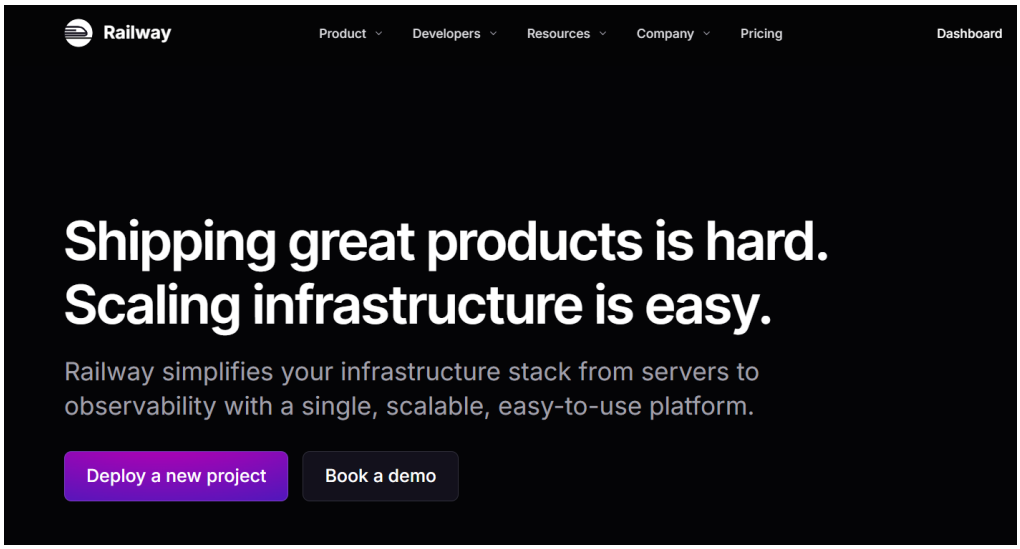


Figure 3.6.1.1 Railway dashboard

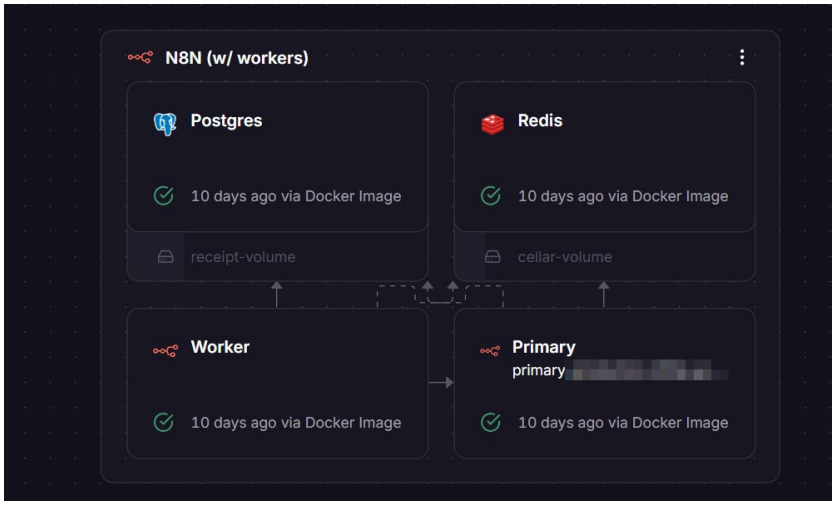


Figure 3.6.1.2 n8n workers

3.6.2 FastAPI

As shown in Figure 3.6.2.1, FastAPI is the main python framework for quickly writing and creating an API endpoint for the RAG process before deploying it to the cloud. Its high performance and automatic documentation generation significantly accelerate development cycles, while the built-in data validation through Pydantic models ensures robust request handling. FastAPI's asynchronous capabilities make it particularly suitable for the RAG process.

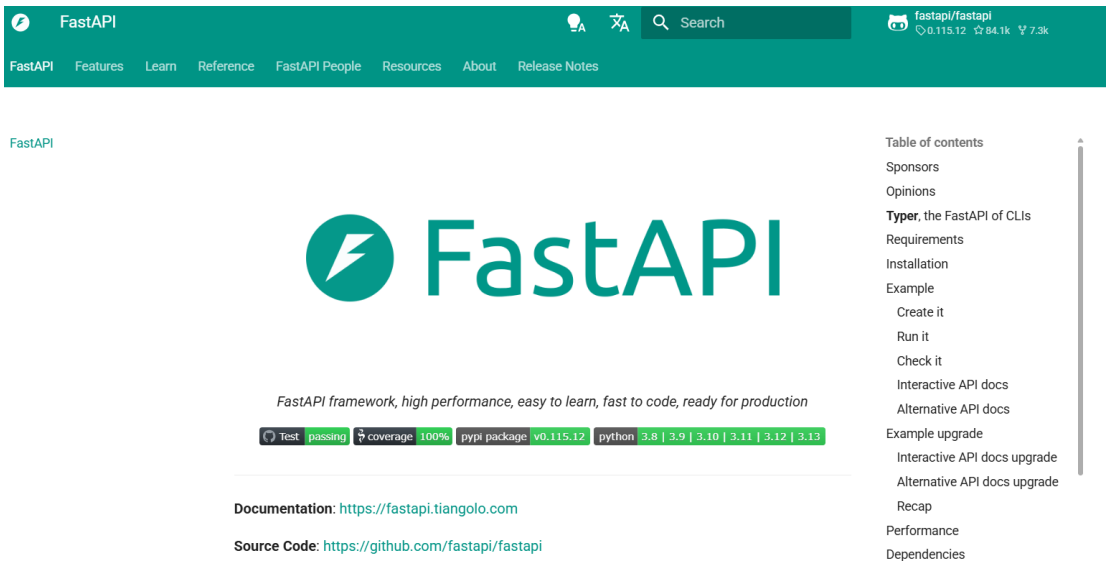


Figure 3.6.2.1 Official FastAPI documentation

3.6.3 Modal

As shown from Figure 3.6.3.1 – 3.6.3.2, Modal is primarily used for deploying the RAG model on the cloud. Modal will help create a publicly accessible URL for the Flutter application to send HTTP requests to. The platform's integrated GPU support is especially valuable for the resource-intensive RAG model, providing the necessary computational power without requiring specialised infrastructure management for the end user's device.

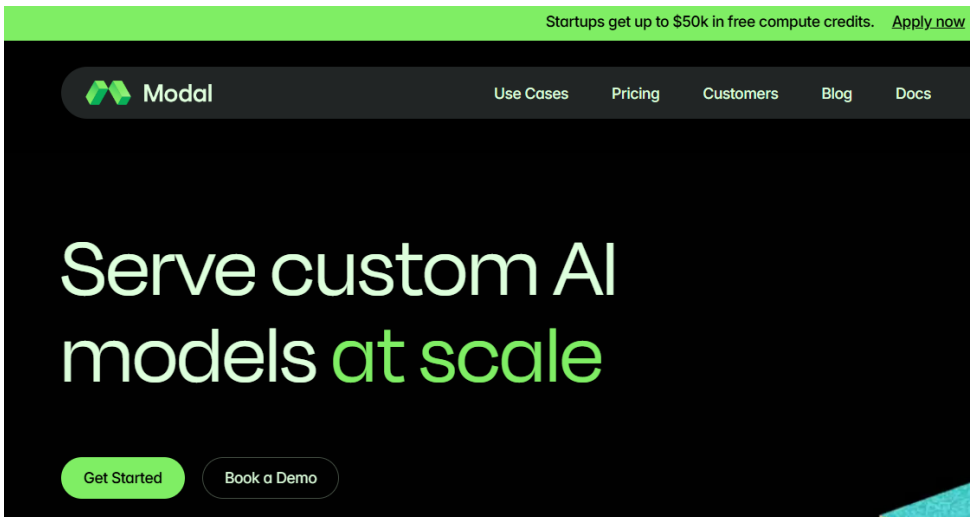


Figure 3.6.3.1 Modal dashboard

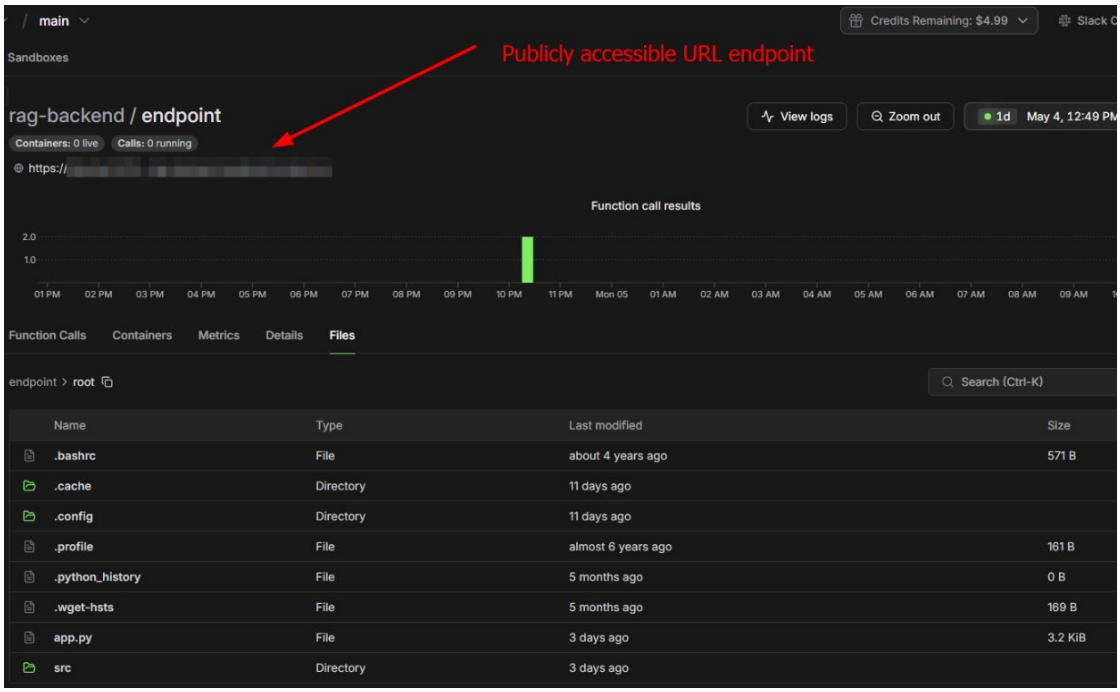


Figure 3.6.3.2 Modal's publicly accessible URL endpoint

3.7 Database Storage Options

The main data storage used throughout the mobile application is Firebase Real Time Database. Firebase is the best option if we want to continuously sync real time data. Below Figures will demonstrate the basic cloud firestore and authentication for different users and the basic connection for firebase in Flutter through CLI is also shown.

3.7.1 Firebase Realtime Firebase

In Figure 3.7.1.1 shows the Firebase Firestore database interface with users collection that contain multiple document IDS representing different users. For each document ID, holds 2 subcollections such as cards collection and expenses collection. In the users document IDs, it also holds fields such as email, FullName and profilePictureUrl.

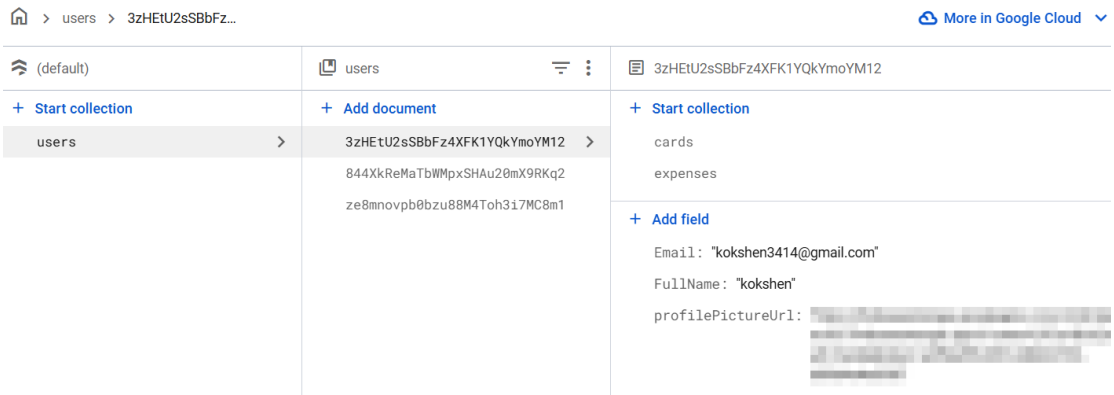


Figure 3.7.1.1 Cloud firestore collections and subcollections

In Figure 3.7.1.2 displays the Firebase Authentication user management interface showing the example of authenticated users in the system. The users will have matching User UIDs that correspond to the document IDs in the Firestore database.

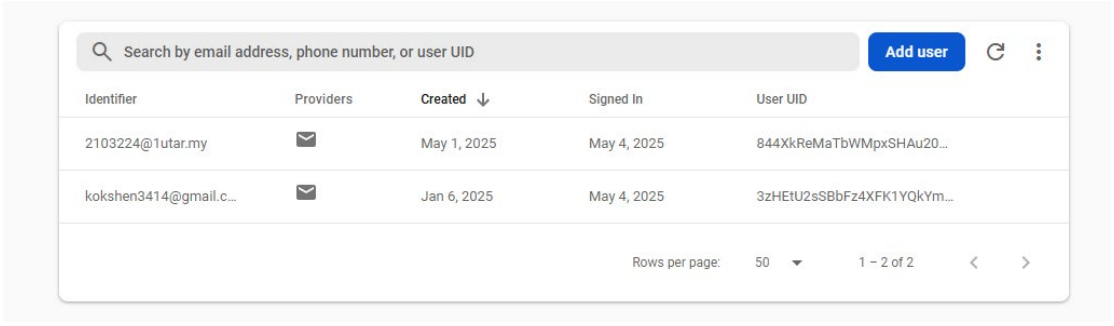


Figure 3.7.1.2 Firebase authentication

3.7.2 Set up Firebase in Flutter (CLI)

Since the mobile application uses Firebase as the backend storage for all user data. Figure 3.7.2.1 – 3.7.2.3 illustrates the clear steps on adding Firebase to the Flutter app. As shown in Figure 3.7.2.1, the steps mainly involving installing the Firebase CLI and SDK. Then, Figure 3.7.2.2 demonstrates how to initialise and run the Flutter Fire CLI before it can register a firebase_options.dart file in the application. Finally, as shown in Figure 3.7.2.3, import the required packages and then initialise Firebase by modifying the firebase_options.dart file.

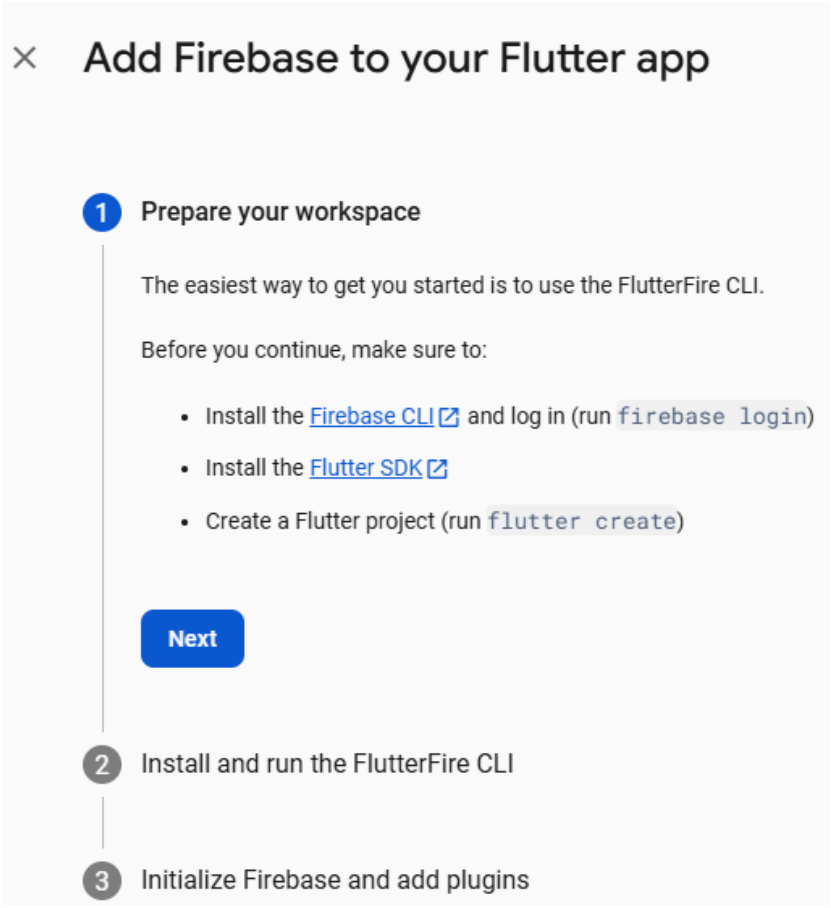


Figure 3.7.2.1 Preparing workspace

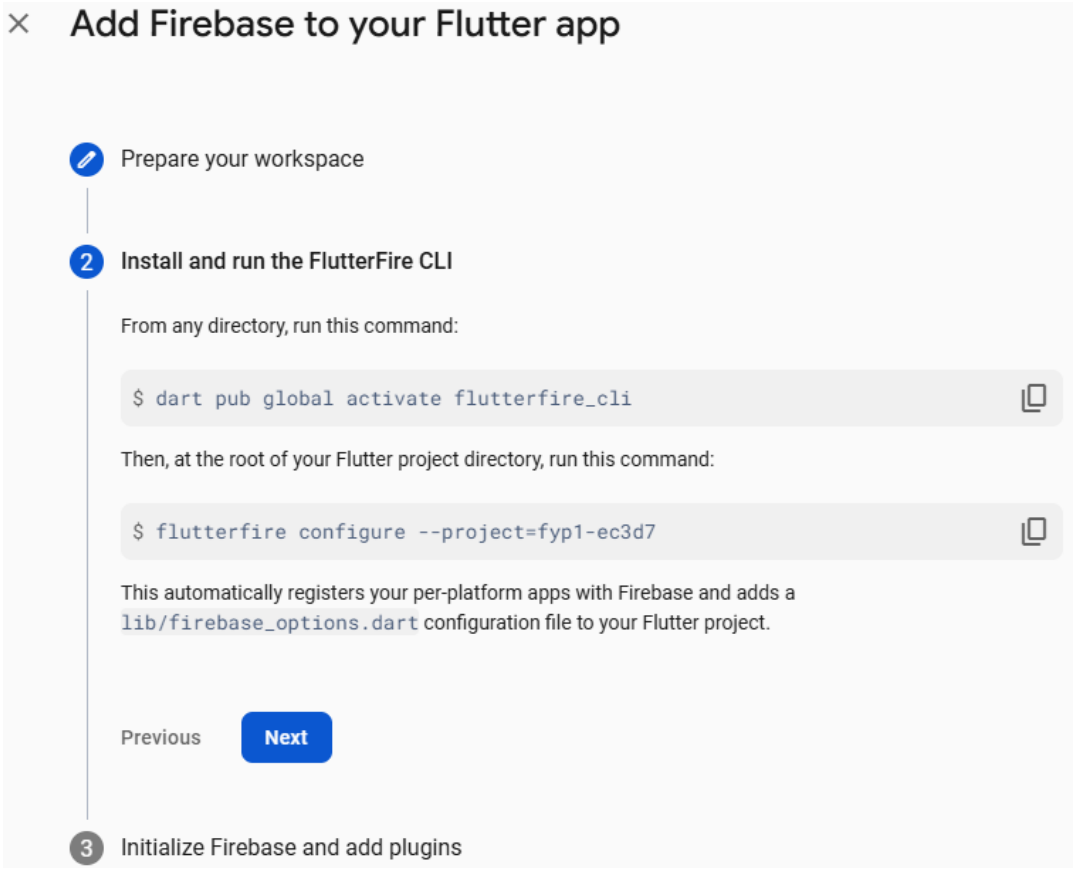


Figure 3.7.2.2 Install and run FlutterFire CLI

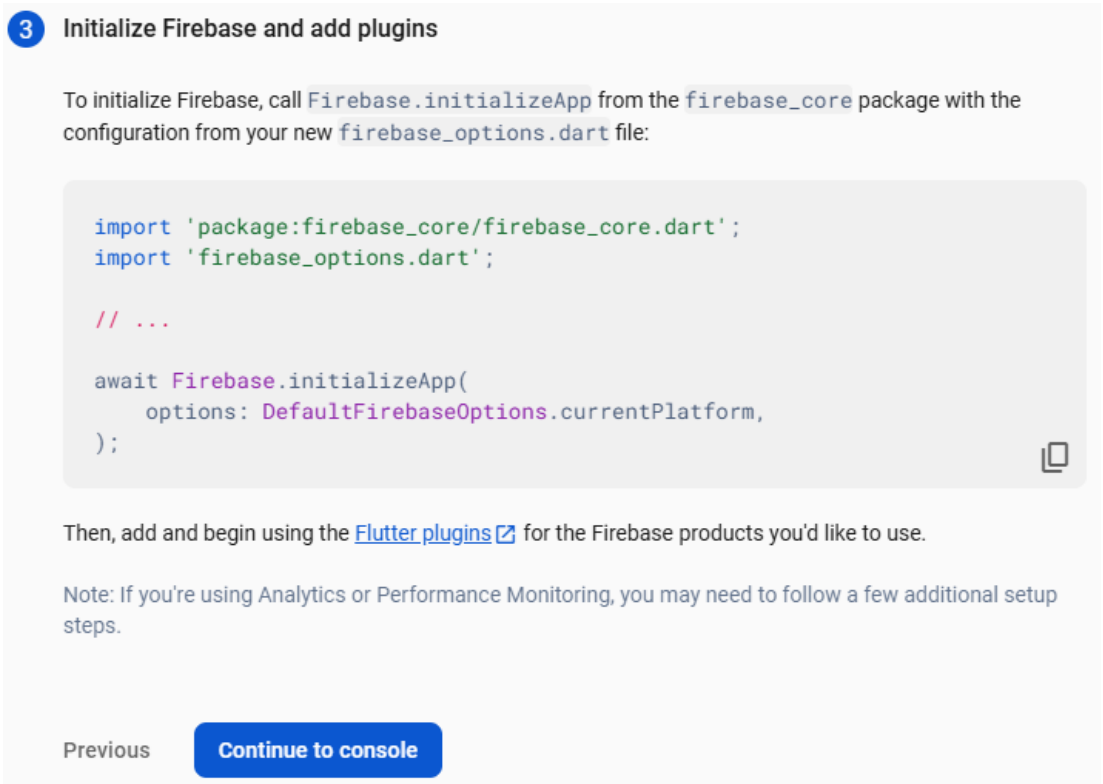


Figure 3.7.2.3 Initialise Firebase and add plugins

3.8 Summary

This chapter detailed the methodological approach for the Smart Financial Tracking Mobile Application, beginning with a comprehensive use case diagram illustrating the system's functionality from user registration to financial analytics. Detailed use case descriptions were provided for all major features including sign-up, login, receipt scanning, manual expense entry, voice recognition entry, financial account integration, and analytics.

The development framework focused on Flutter for cross-platform compatibility, with Firebase serving as the backend solution for authentication and data storage. Several innovative technical implementations were described, including the receipt scanning algorithm using Google MLKit, voice recognition for expense entry, and integration with external APIs like Finverse for financial data retrieval.

Two specialised components were developed using external technologies such as n8n workflow for email transaction scraping, and a RAG-based PDF chatbot implemented with FastAPI and deployed on Modal. The backend infrastructure utilised multiple cloud hosting options including Railway for the n8n workflow and Modal for the RAG system, overall creating a robust financial tracking pipeline while maintaining user experience.

CHAPTER 4 Methodology and Tools

This chapter presents a comprehensive overview of the project's methodology and implementation tools for the Smart Financial Tracking Mobile Application. The discussion begins with examining the main system development methodology, detailing the Rapid Application Development (RAD) framework chosen to facilitate iterative prototype creation and refinement. Then in 4.2 outlines the system requirements, tabulating both hardware and software specifications essential for the system's development. The chapter proceeds to examine the key challenges and implementation issues encountered during development. The chapter is concluded with a Gantt chart depicting the project timeline spanning across both FYP 1 and FYP 2.

4.1 System Development Methodology

For this project, the most suitable methodology is Rapid Application Development (RAD). RAD-based methodologies were an attempt to address both weakness of structured design methodologies by adjusting the SDLC phases to get some part of a system developed quickly and into the hands of users [6]. As shown in Figure 4.1 below, RAD mainly consists of 4 phases namely requirements planning, user design, construction and cutover.

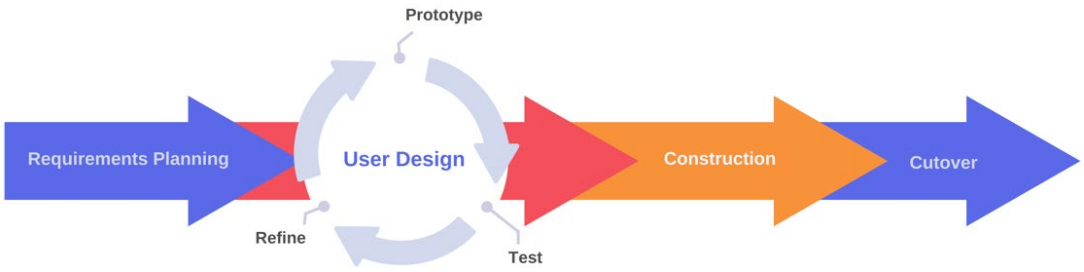


Figure 4.1 Four phases in Rapid Application Development

4.1.1 Requirements Planning

In the first phase of RAD which is requirements planning, a well-defined and appropriate project topic is chosen and then the existing problems related to the chosen topic is reviewed. The initial idea is to develop a financial tracker for individuals including teens, adults or even students who seek to track their expenses in an effective way. After having an overall of the project, literature review on related financial tracking systems is reviewed to analyse the strengths and weaknesses. The existing

systems are Touch ‘n Go, You Need a Budget (YNAB), Meow Money Manager, Easy Expense and n8n. The overall project scope and objectives are outlined clearly.

Then other requirements such as hardware specifications for the laptop and mobile phone were also identified. Not to mention, the type of developing tools such as the IDE. These additional requirements were essential for determining if the mobile application could be developed without compatibility or dependency issues.

4.1.2 User Design

In the subsequent phase, actual development commences. This stage involves rapidly creating various throwaway prototypes with different features and functions. These basic prototypes are then presented to users for feedback on design appropriateness. For example, the initial prototype will primarily focus on fundamental UI design elements, such as a basic button for user authentication in the login module, or a camera icon for receipt capture, as illustrated in Figure 4.1.2.1. UI design is streamlined due to Flutter's extensive library of pre-designed and customizable widgets, which facilitate the creation of interactive and unique interfaces while minimising development time. Prior to user presentation, each prototype undergoes iterative testing to ensure it achieves the minimal features set for that version. The first prototype, for instance, will incorporate at least two main functionalities, such as receipt scanning and Malaysian Bank Integration as shown in Figure 4.1.2.2. This iterative testing is crucial for validating whether the mobile application meets the minimal feature requirements for each prototype developed. It also aids in identifying errors or bugs before user presentation. During the testing phase, potential issues like logic errors may arise. An example would be displaying a green (+) sign instead of a red (-) sign when a user records a transaction. Based on these testing results, the prototype is refined, and such logic errors are addressed.

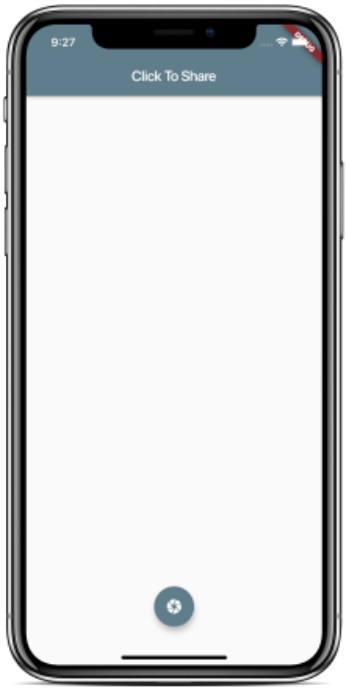


Figure 4.1.2.1 User interface design with a single camera icon

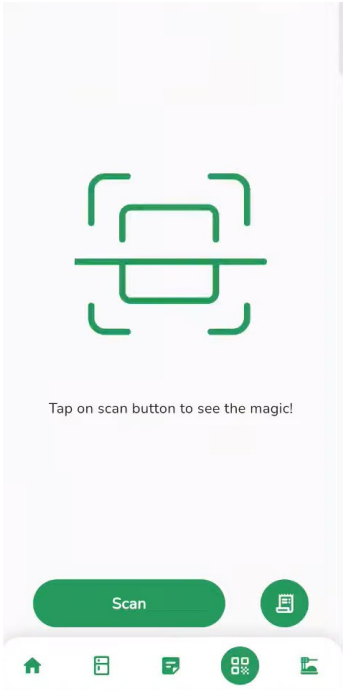


Figure 4.1.2.2 An example of refined user interface design with receipt scanning

4.1.3 Construction

In the third phase of the methodology, the focus shifts from prototyping to building a fully functional financial tracking mobile application. The refined prototypes are transformed into more workable models. With a significant portion of issues addressed during the iterative design phase, focus now shifts to constructing a final working model (a fully functional system). At this stage, the process moves beyond prototype creation, instead concentrating on integrating various refined prototype versions into a single, cohesive functional system. Key activities include implementing core modules such as receipt scanning functionality using optical character recognition (OCR), developing the Malaysian bank integration module to securely connect with various local banks' APIs, and creating a robust transaction categorisation system via voice recognition, RAG based Chatbot and email transaction scraping. Likewise, the user interface is finalised based on earlier feedback, with responsive layouts implemented to ensure compatibility across various Android devices. This phase also demands intensive coding efforts and employs more formal testing methods, including unit, integration, and system testing to verify overall functionality and performance. Overall, this phase involves coding using the Flutter framework and Dart programming

language, with the goal of transforming the conceptual designs and prototypes into a fully functional financial tracking mobile application that meets the project objectives outlined in Chapter 1.

4.1.4 Cutover

This represents the final phase of the methodology, where the fully functional model is completed and prepared for release and deployment. At this stage, the mobile application has achieved full operational capabilities, fulfilling all modules outlined in Chapter 1. A comprehensive integrated system test will be conducted to ensure all components work together seamlessly. Additionally, the project documentation, including use case diagrams, and use case description along with the full report, will be finalised. Upon completion, the project will be ready for deployment across various mobile devices.

4.2 System Requirement

4.2.1 Hardware Specification

The two tables below illustrate the technical hardware that will be used throughout the project which mainly involves a laptop and an android mobile device. Table 4.2.1.1 and Table 4.2.1.2 shows the specifications of the laptop and mobile phone for this project. The laptop uses the latest windows operating system and is also equipped with enough storage and RAM to run most resource intensive programs like the android studio IDE. Also, an android mobile phone would be used to test and deploy the proposed mobile application.

Components	**Specifications**
Model	Acer Nitro 5 (AN515-57-78PJ)
Processor	Intel® CoreTM i7-11800H
Operating System	Windows 11
Graphic	NVIDIA® GeForce® RTX™ 3060 with 6GB of dedicated GDDR6 VRAM
Memory	32GB DDR4 RAM
Storage	512GB SSD (NVMe) + 2 TB SSD

Table 4.2.1.1 Specifications of laptop

Components	**Specifications**
Model	OPPO A95 (CPH2365)
Processor	Qualcomm SM6115 Snapdragon 662
Operating System	Android 13, ColorOS 13.0
Graphic	Adreno 610
Memory	8 GB + 6 GB RAM
Storage	128 GB internal storage
Display	6.43-inch AMOLED, 1080 x 2400 pixels
Battery	5000 mAh
Main Camera	Triple Rear Camera (48 MP main, 2 MP macro, 2 MP depth)
Front Camera	16 MP
Connectivity	4G LTE

Table 4.2.1.2 Specifications of phone

4.2.2 Software Specification

Table 4.2.2.1 shows the basic software requirements to develop the project. Two IDE tools were mainly used such as Android Studio and Visual Studio Code. Android Studio is the most well-known IDE tool for developing native android based mobile applications in different programming languages such as Java, Dart, Kotlin and React Native. Notably, Android Studio comes with pre-built android emulators that allow developers to create app on different API levels and android versions.

Developers can create dynamic screen sizes that can accommodate to all screen sizes because they can test on emulators instead of testing on the physical device itself. As for Visual Studio Code, it is mainly utilised to test the HTTP API connection with the various third-party APIs such as Finverse API. Firebase (NoSQL database) is the main backend as a service for most flutter applications, since FYP 1 and FYP 2 has a strict time constraint, developing a complete and comprehensive backend using Java, Golang or PHP from scratch is not ideal. Instead, the mobile app leveraged multiple specialised backend services instead such as FastAPI for implementing the RAG model deployment hosted on Modal's serverless platform. For the n8n workflow automation instance, it was hosted on Railway. This distributed approach allowed for rapid development within the time constraints of FYP 1 and FYP 2 while still providing robust backend functionality.

The project employed multiple programming languages for different components: Flutter (Dart) for the cross-platform mobile application frontend, Python for the RAG model implementation and FastAPI services, and JavaScript for n8n workflow configurations.

Components	**Requirements**
IDE	Android Studio Hedgehog 2023.1.1 Patch 1, Visual Studio Code 1.9.2
Android Studio Emulator	Pixel 7 API 31
Software Environment	Java JDK 11, Java (TM) SE Runtime Environment 18.9, Node.js v20.9.0, Postman v11
Server-Side (Backend)	Firebase (Authentication & Database), FastAPI (RAG Model), Modal (Serverless Hosting), Railway (n8n Workflow Hosting)

Client-Side (Frontend)	Dart (Flutter)
Programming Languages	Dart, Python, JavaScript

Table 4.2.2.1 Software requirements

4.3 Implementation Issues and Challenges

Below shows the development challenges along the development of Smart Financial Tracker Mobile Application.

4.3.1 Retrieval of demo financial accounts

One of the core features of the mobile application is that it can allow users to simulate the adding of dummy financial accounts through the 3rd party Finverse API. However, documentation on this official website was scarce and not resourceful for mobile application developers. An extensive amount of time was spent on figuring out the correct workflow and sequence to extract the required access tokens before Finverse returns the financial data.

Additionally, a prerequisite knowledge of REST API basics is mandatory. HTTP methods such as GET, POST, PUT and DELETE for testing the API before integration. APIs involves fetching data over a network asynchronous programming must be implemented in Dart using Future, sync and await keywords.

4.3.2 Finding libraries and packages

Some of the core features of the mobile application require extensive support from external libraries and packages. It was an initial challenge to determine the appropriate packages needed for the application's use cases such as text recognition and voice to text recognition. After much research, Pub.dev was discovered for easy installation and usage of libraries that were already maintained by other developers. Pub.dev is a well organised package manager for the Dart programming language.

4.4 Timeline

The overall project timeline is planned according to Rapid Application Development methodology. The 3 phases are namely Requirements Planning Phase, User Design and Construction Phase, Test and Implementation Phase. Figure 4.4.1 below shows the Gantt Chart for FYP 1 and FYP 2. The overall duration of the project is estimated to be completed around 21 weeks.

FYP 1 is conducted during a short trimester, which includes the Requirements Planning, User Design and Construction Phase. In the first 3 weeks, the overall project scope is defined. The functional requirements and non-functional requirements are drafted accordingly which determine the overall functionality and performance of the mobile application. Software development tools such like Visual Studio Code, android emulator and Git are configured. A first initial dummy prototype was created and received feedback from supervisor. After consideration of Supervisor feedback, a more functional prototype is further developed. In week 5 and week 6, the FYP 1 report is written and finalised. Finally, on week 7 a presentation and demo on the prototype was presented to the moderator.

FYP 2 is conducted during a long trimester, which is the continuation of User Design and Construction Phase. The long trimester is 14 weeks long, in the first 10 weeks the remaining modules are implemented such as Financial Insight Module, Low Code Email Scraping, Expense Report Generation, and RAG PDF Chatbot. As for the remaining weeks from week 11 to week 13, the mobile application will be iteratively tested, FYP 2 is finalised, and the final working application is presented during week 14.

Task name	Duration	Start date	End date	Week																					
				Project 1							Project 2														
				1	2	3	4	5	6	7	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Requirements Planning Phase																									
Review project plan	**1**	10/27/2024	10/28/2024																						
Define project scope	**1**	10/29/2024	10/30/2024																						
Draft functional and non-functional requirements	**2**	10/31/2024	11/2/2024																						
Create use case diagrams and descriptions	**2**	11/3/2024	11/5/2024																						
Determine software setup and tools	**3**	11/6/2024	11/9/2024																						
User Design and Construction Phase																									
Setup IDE and android emulator	**1**	11/10/2024	11/11/2024																						
Setup version control system using Git	**1**	11/12/2024	11/13/2024																						
Start building initial prototype	**2**	11/14/2024	11/16/2024																						
Reevaluate feedback from supervisor	**1**	11/17/2024	11/18/2024																						
Start building functional prototype	**12**	11/19/2024	12/1/2024																						
FYP 1 Report Finalisation	**4**	12/2/2024	12/6/2024																						
FYP 1 Presentation	**1**	12/10/2024	12/11/2024																						
Integrate robust backend for prototype	**13**	2/10/2025	2/23/2025																						
Financial Insight Module	**6**	2/24/2025	3/2/2025																						
n8n Email Scraping	**13**	3/3/2025	3/16/2025																						
Expense Report Generation	**13**	3/17/2025	3/30/2025																						
Langchain API Integration	**20**	3/24/2025	4/13/2025																						
Test and Implementation Phase																									
Test mobile app	**13**	4/14/2025	4/27/2025																						
FYP 2 Report Finalisation	**11**	4/28/2025	5/9/2025																						
FYP 2 Presentation	**6**	5/10/2025	5/16/2025																						

Figure 4.4.1 Gantt chart for FYP 1 and FYP 2

4.5 Summary

The system development methodology chosen was Rapid Application Development. It is a suitable methodology because, rapid changes can be made throughout the lifecycle of the project. Prototypes can be created and abandoned at any stage but would not affect the overall development progress. Consequently, the main hardware needed for this project is a mobile phone and a laptop whereas the main development framework is Flutter and uses Dart programming language for frontend development. The backend leverages multiple specialised services including Firebase for authentication and database, FastAPI for RAG model implementation, and n8n for workflow automation, all hosted on cloud platforms like Modal and Railway. For the system design, both use case diagrams and use case description are created to map the functional and non-functional requirements of the system. Several implementation challenges were encountered, particularly with third-party API integration (Finverse) and finding appropriate Flutter packages for core functionality like OCR and voice recognition. Lastly, the entire project timeline for FYP 1 and FYP 2 is shown with a Gantt chart.

CHAPTER 5 Implementation and Testing

In this chapter, an overview of all the working modules implemented during FYP 2 is demonstrated. Namely, signup and login module, scan receipt module, manual expense module, voice recognition expense module, add financial accounts module, financial transaction analytics, and PDF RAG based chatbot.

5.1 Signup and Login

The smart financial tracker will utilise Firebase Authentication and Cloud Firestore for saving and validating new users upon registration and login. Below Figure 5.1.1, and Figure 5.1.2 shows the main interface of the signup screen and login screen. In the signup screen, new users must enter their full name, email address, password and confirmed password. As for the login screen, user must enter credentials created during registration. Upon registration, new user is authenticated by Firebase Authentication as shown in Figure 5.1.3. Consequently, basic user information such as full name and email address is stored inside Cloud Firestore to keep track of different users as shown in Figure 5.1.4.

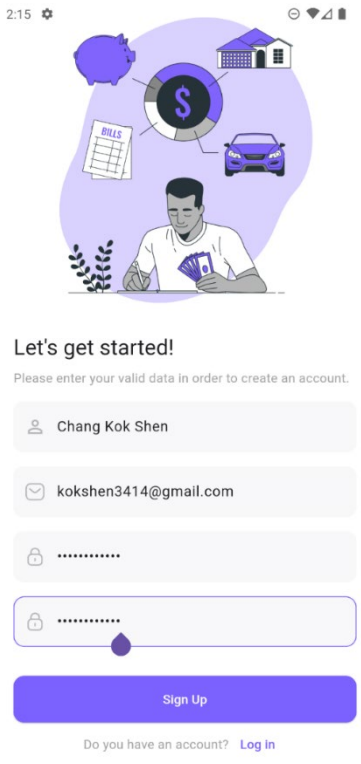


Figure 5.1.1 Signup screen

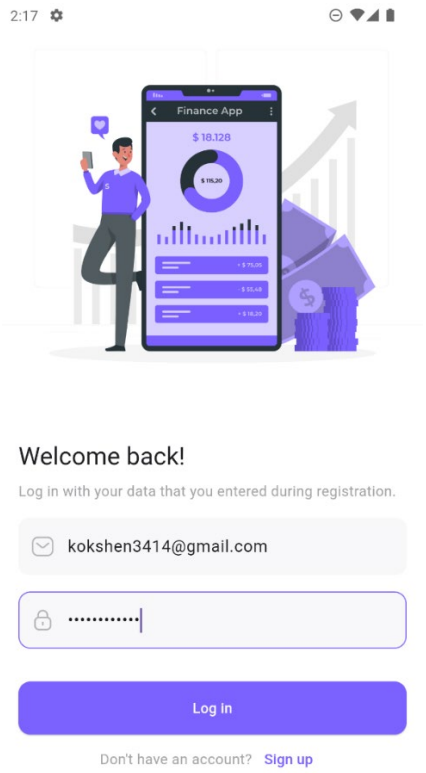


Figure 5.1.2 Login screen

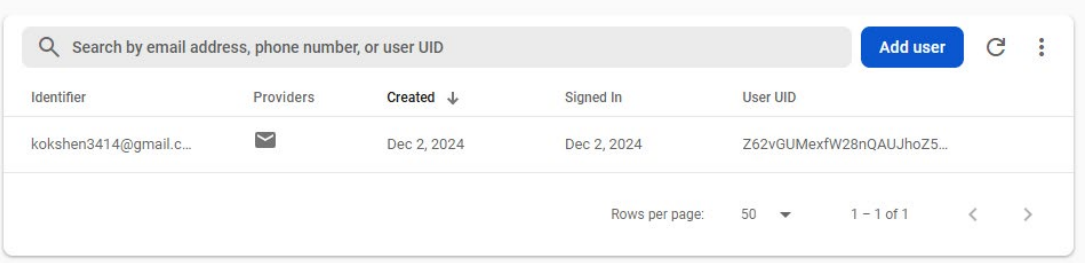


Figure 5.1.3 Firebase Authentication

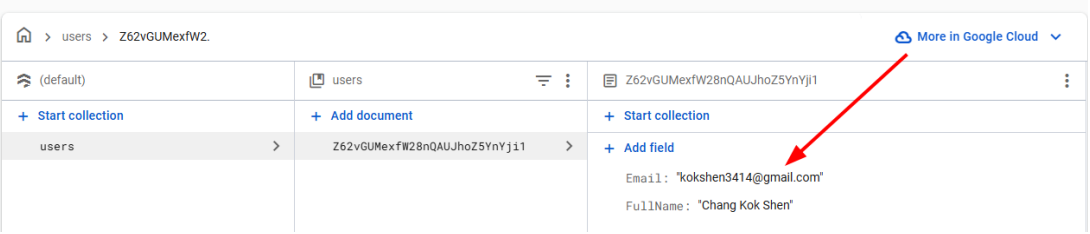


Figure 5.1.4 Firebase Cloud Firestore

5.2 Scan Receipt

The smart financial tracker offers the user the capability to automatically capture and recognise key insights from physical receipts. But user must first click on the middle camera icon to start the receipt scanning process. Figure 5.2.1 below shows an empty in the Receipts screen before the scans and adds any new receipts whereas Figure 5.2.2 below indicates the newly scanned receipts arranged in most recently added receipt at the top.

As for Figure 5.2.3 indicates 3 options whereby user can select an image from gallery, manually capture the image or auto capture the image. Then in Figure 5.2.4 below show the different options for user to enhance the receipt image such as cropping extra edges, applying filters for contrast improvement and eraser which allows users to remove extra smudges on the receipt image.

Next, the selected receipt image is selected and processed to extract key details such as merchant name, tax, total amount and transaction date as shown from Figure 5.2.5 to Figure 5.2.6. Finally, users can choose to save the receipt or delete the extracted receipt details.

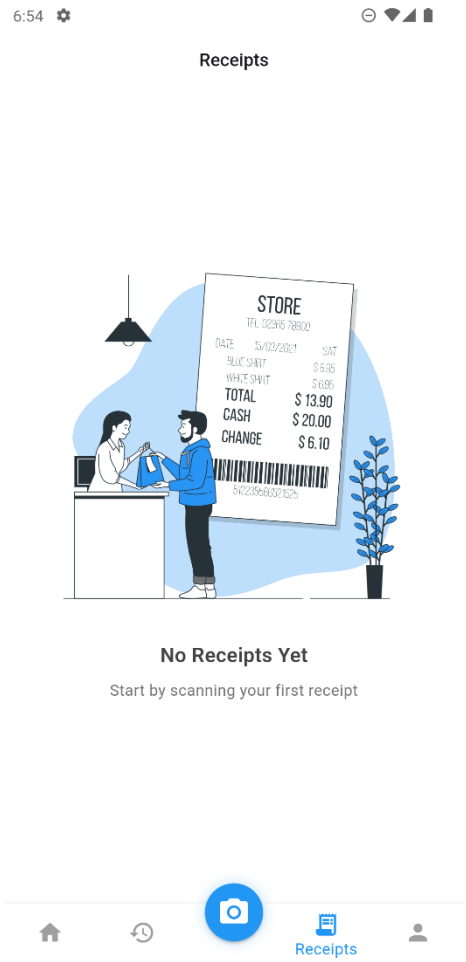


Figure 5.2.1 Empty state - no receipts added

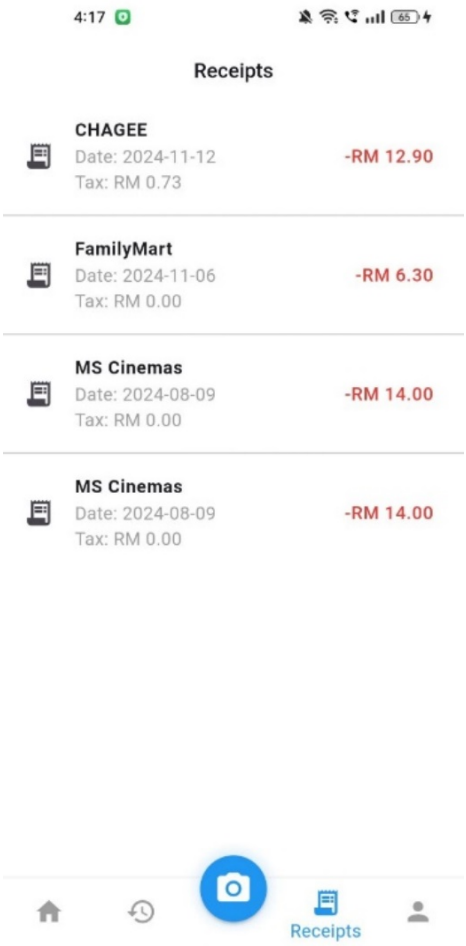


Figure 5.2.2 Scanned and saved receipts

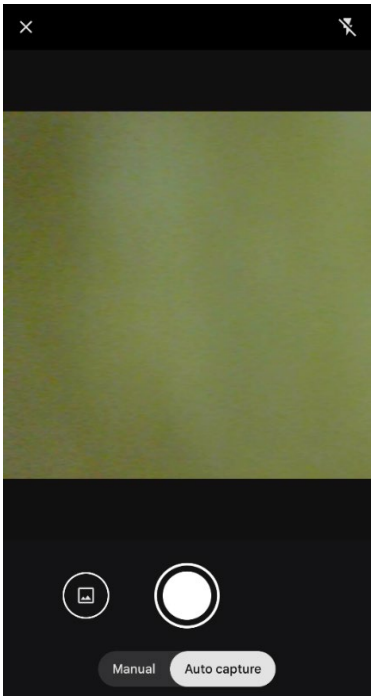


Figure 5.2.3 Options - gallery, manual and auto capture

Figure 5.2.4 Edit processed receipt

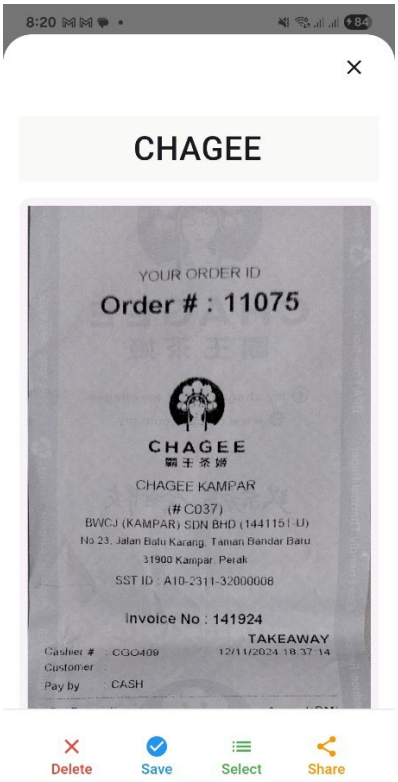


Figure 5.2.5 Receipt details - 1

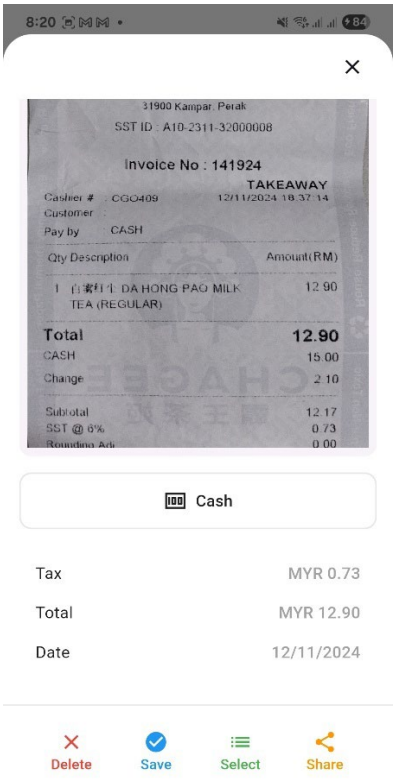


Figure 5.2.6 Receipt details - 2

5.3 Select Receipt Transactions & Share Receipt

The smart financial tracker offers the user the ability to select receipt transactions and share the receipt to their friends, family members, colleagues etc. As shown in Figure 5.3.1, it shows the scanned receipt details before editing the receipt transactions. To note, editing receipt transactions is essential because within a receipt not all item transactions belong to us. Consequently, in Figure 5.3.2 the user of the application can select their own / their friend's respective transactions. In the below example, we are replicating the logic that we are selecting their food items on their behalf. Once we have selected their food items. We will update the receipt details and then be able to share the newly updated that reflects their transaction. As shown in Figure 5.3.3, the user must tap on the share button which will bring up a native UI sharing dialog which allows users to send this receipt image to their selected application as also demonstrated in Figure 5.3.4. As illustrated in Figure 5.3.5, in this given example, the newly updated receipt image was tested by sending out to a WhatsApp contact. Before confirming to send the new receipt image, user can also edit the description from the original "Shared Receipt" and expanding the description into their own favour such as "Shared Receipt – Hiii, this is how much you need to pay".

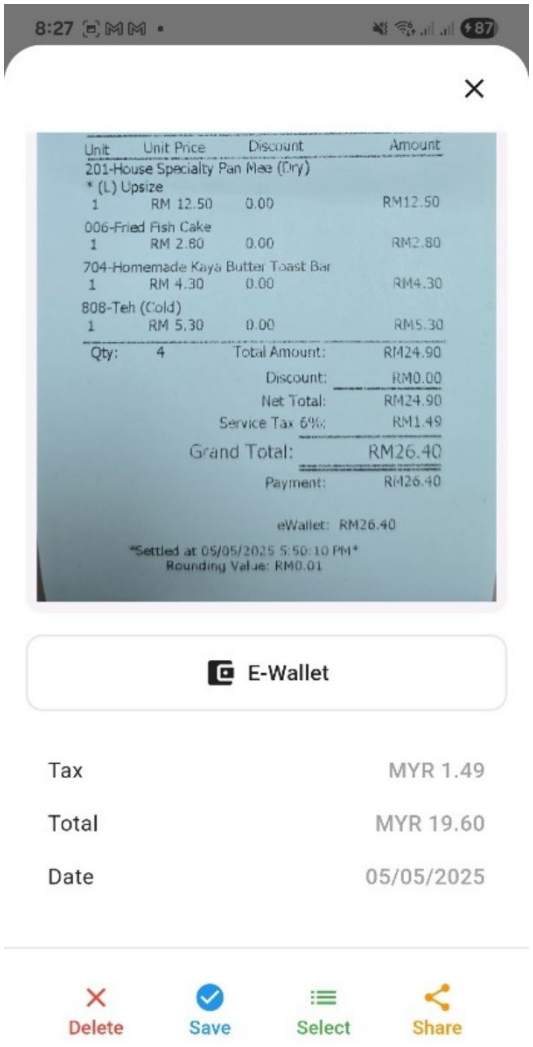


Figure 5.3.1 Original receipt details before edit

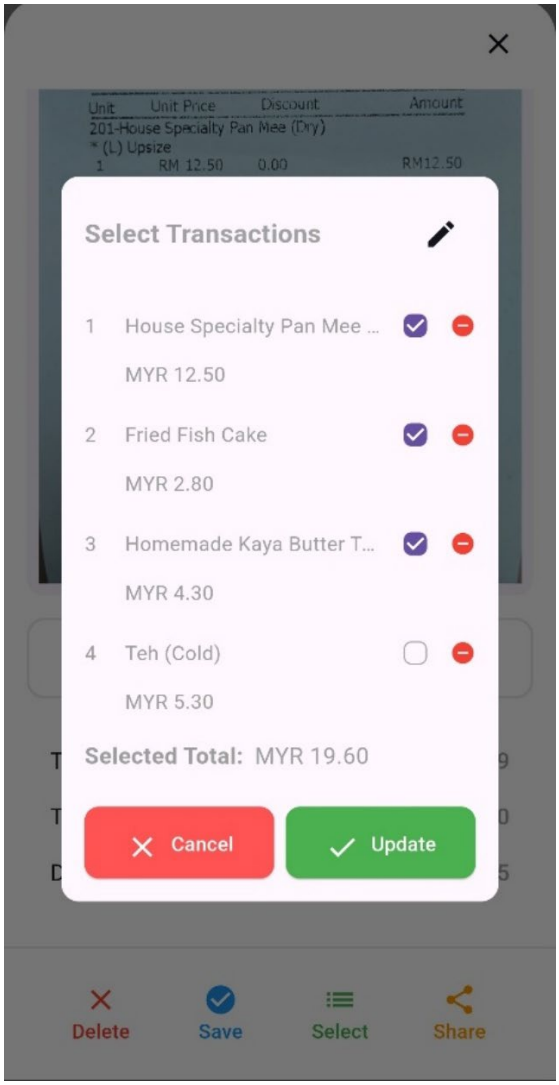


Figure 5.3.2 Selecting transactions

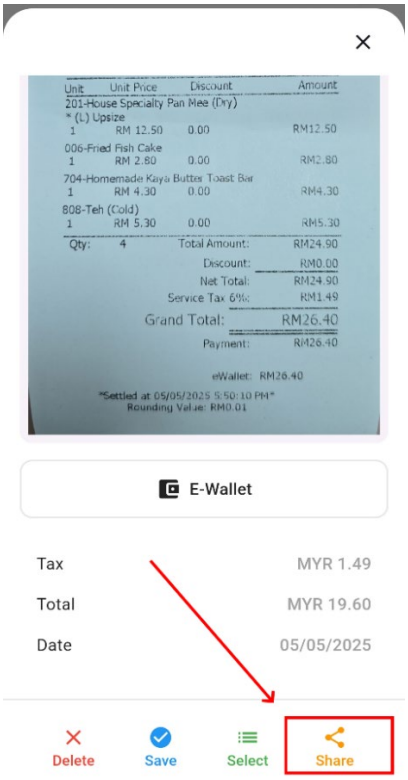


Figure 5.3.3 Share newly updated receipt details

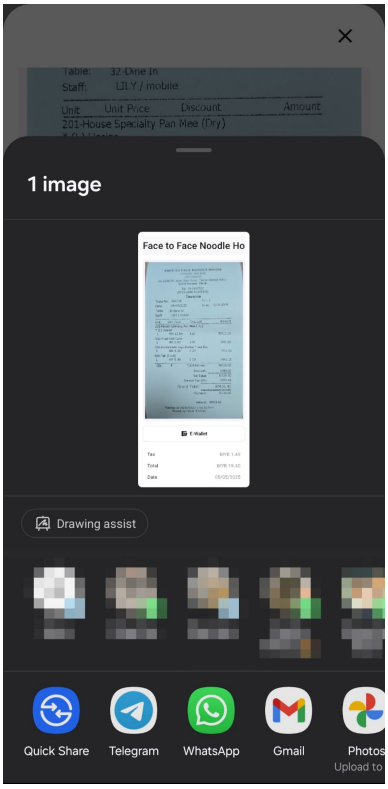


Figure 5.3.4 Share receipt image to available apps

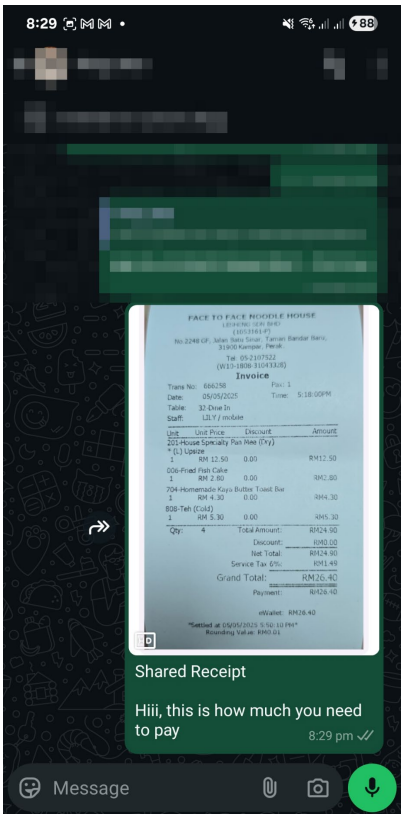


Figure 5.3.5 Share receipt image to WhatsApp contact

5.4 Edit Receipt Transactions & Add Receipt Transactions

The smart financial tracker also offers the user the flexibility to edit their receipt transactions such as modifying the item price. As shown from the Figure 5.4.1 – Figure 5.4.2, user can select an item and modify the item price, once user has finished modifying the numeric value. They must proceed to tap on green update button, as depicted in Figure 5.4.3 they will be redirected to the previous select transactions dialog whereby they can update their new selected total.

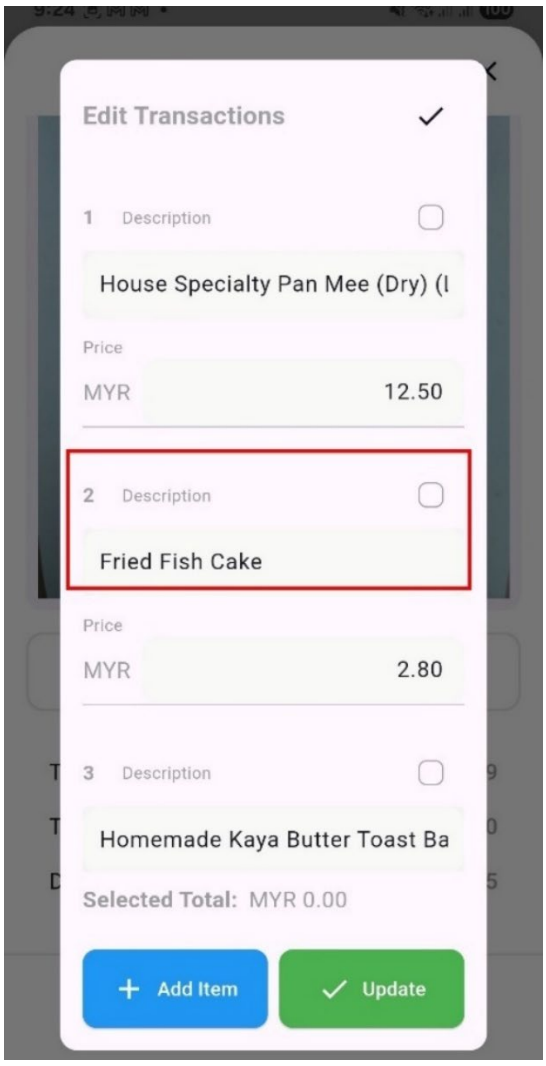


Figure 5.4.1 Select transaction to edit

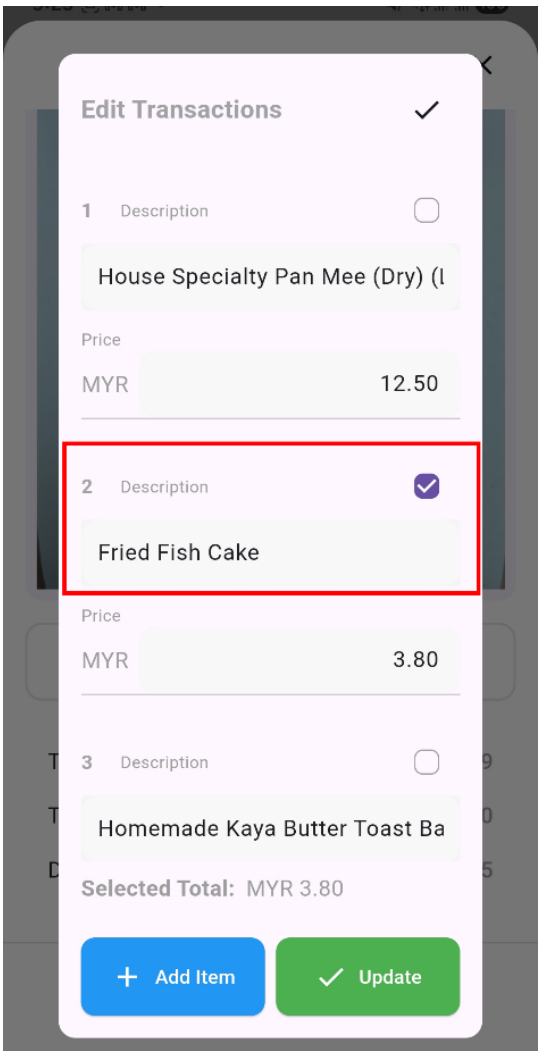


Figure 5.4.2 Select transaction to edit - 2

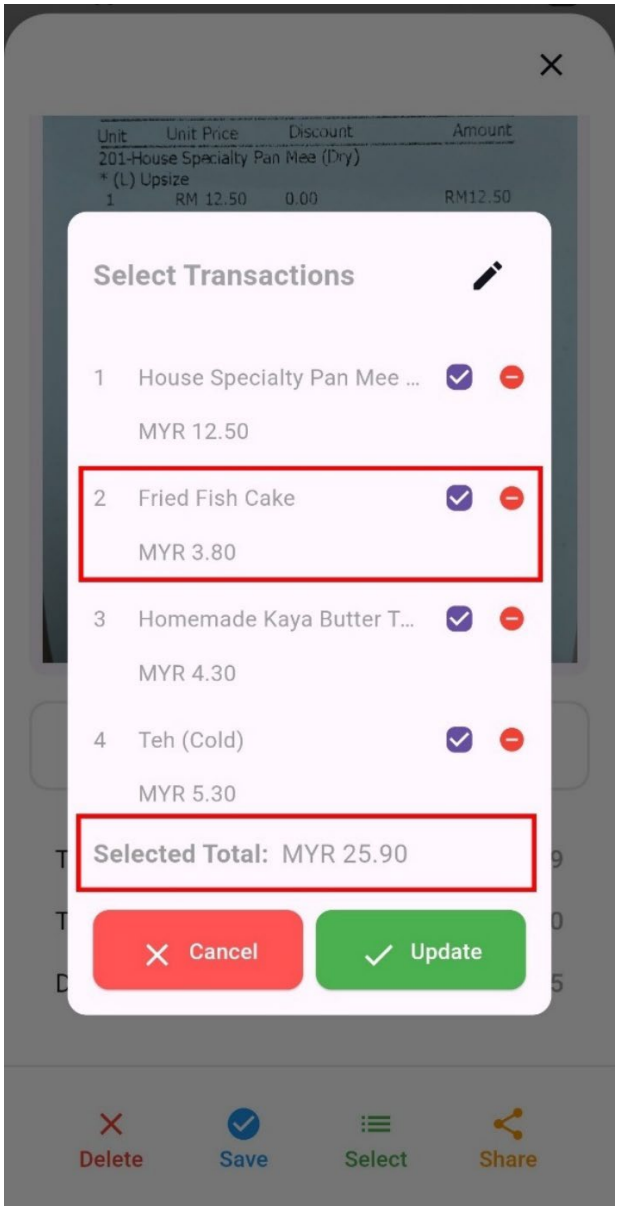


Figure 5.4.3 Edited transaction reflects in new selected total

As shown from Figure 5.4.4 – 5.4.5 below, user can add new items into the receipt transaction list. This is a fallback in case the scanning receipt functionality fails to correctly parse all the correct transactions.

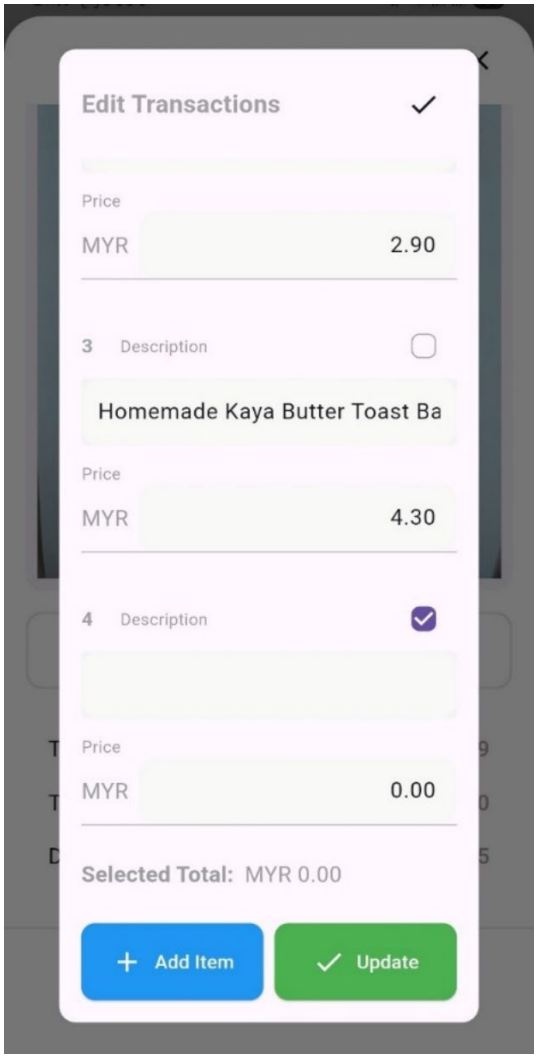


Figure 5.4.4 User adds new item

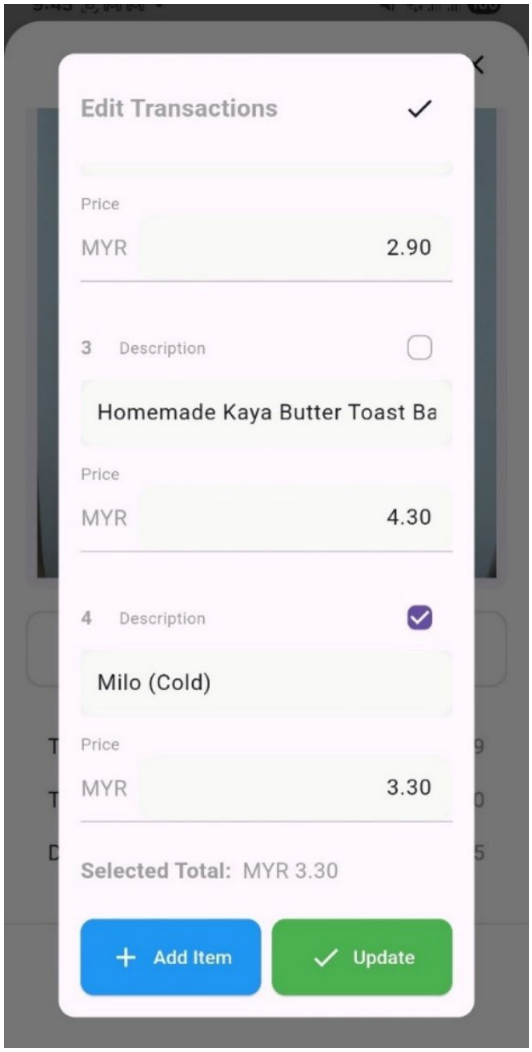


Figure 5.4.5 User adds new item - 2

5.5 Manual Expense

The smart financial tracker also allows users to manually add their expenses according to different categories. Like the receipts screen, as shown in the Figure 5.5.1 below, manual expenses are all stored inside history screen and the user interface will be triggered upon user press on text font like icon. As shown the Figure 5.5.2 below, in the current interface implementation, there are 9 main categories for the user to choose from. For any categories not mentioned, the user should select the miscellaneous category. To note, the current expense tracker has only the option to record in Ringgit Malaysia currency.

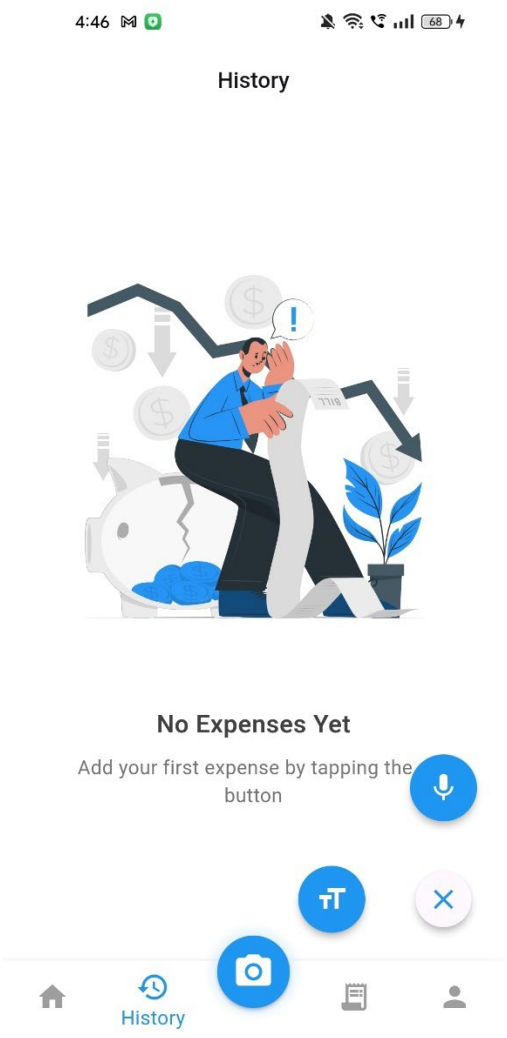


Figure 5.5.1 Empty state - no expenses added

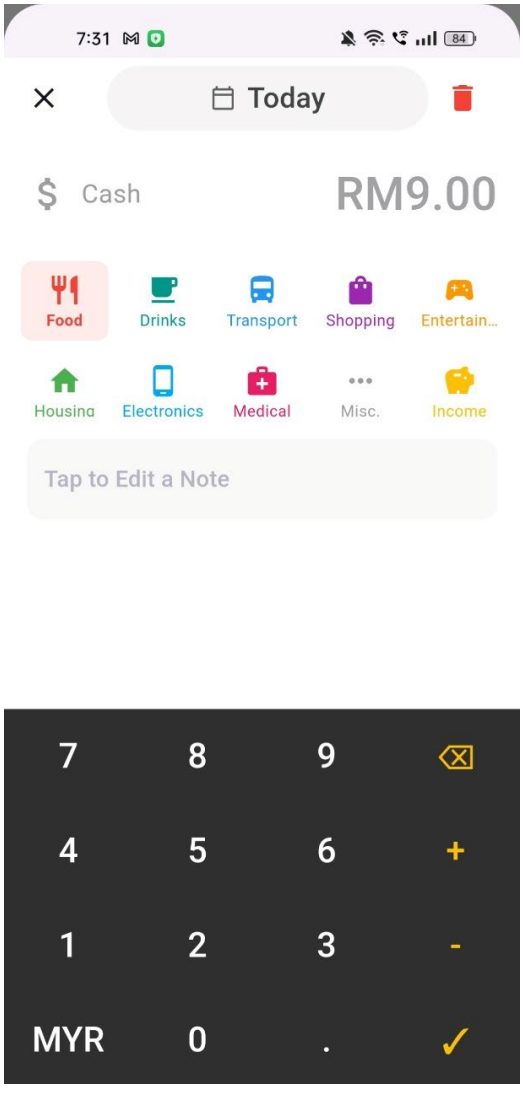


Figure 5.5.2 Adding expense according to category

5.6 Voice Recognition Expense

The smart financial tracker also allows users to add their expenses using voice recognition. The voice recognition entry will be triggered upon pressing the mic button after toggling the floating action button. As shown the Figure 5.6.1 below, user can express how they spent their expenses in natural language upon pressing and continuous holding the mic. Then, in Figure 5.6.2 user must keep holding the mic while they articulate their sentence structure. Finally, in Figure 5.6.3 the complete sentence structure of the user is shown along with the parsed output showing expense, category and date.

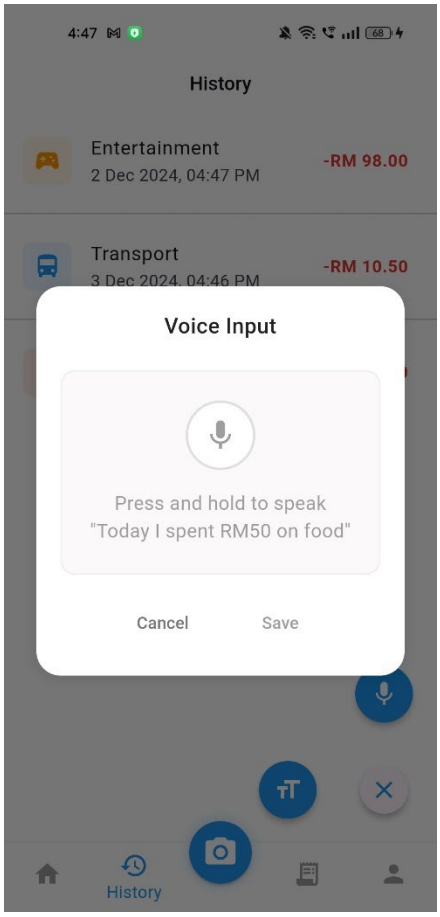


Figure 5.6.1 Voice input - 1

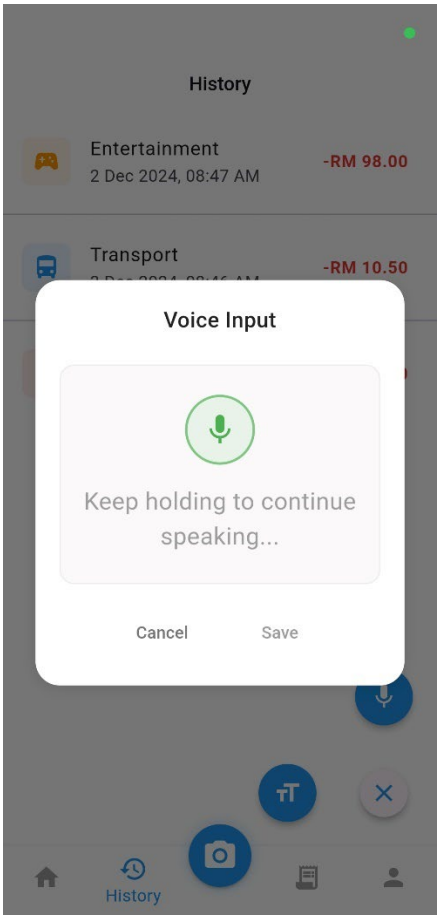


Figure 5.6.2 Voice input - 2

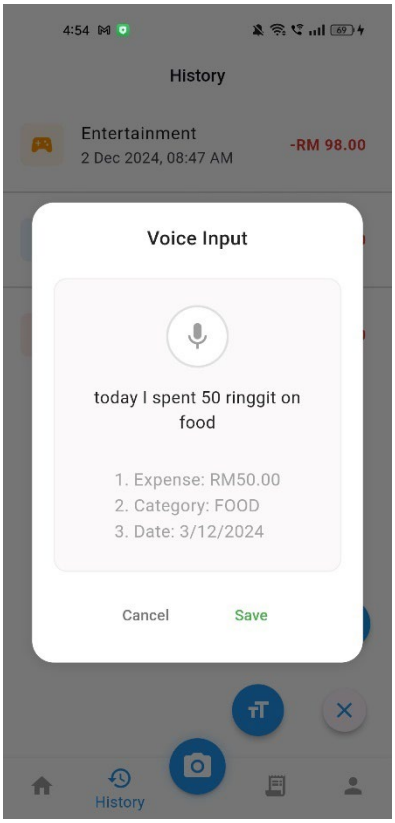


Figure 5.6.3 Voice recognition result

5.7 Add Financial Accounts

The smart financial tracker also allows users to simulate the adding of test / demo financial accounts provided by Finverse 3rd party API provider. In Figure 5.7.1, user first tap on the plus icon and proceeds to connect bank account. Next, from Figure 5.7.2 to Figure 5.7.4 demonstrates the process of linking a test bank account through Finverse API. Finally, the test banking financial info is displayed as shown in Figure 5.7.5. To note, Finverse API can be extended to retrieve real financial bank accounts, in the context of Malaysia. 3 known banks have collaborated with Finverse namely Public Bank, Maybank and CIMB. Due to security concerns and lack of a robust backend to support the retrieval and sending of access tokens. In FYP 1 and 2, only the demo bank account is retrieved to prove the possibility of retrieval real time financial accounts. As shown in the last Figure 5.7.6, retrieval of real bank account statements for Public Bank is achieved but will not be demonstrated in the current working mobile application for security concerns.

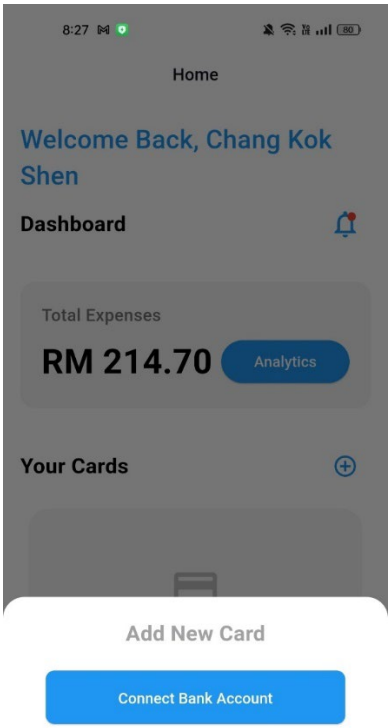


Figure 5.7.1 Add bank account - 1

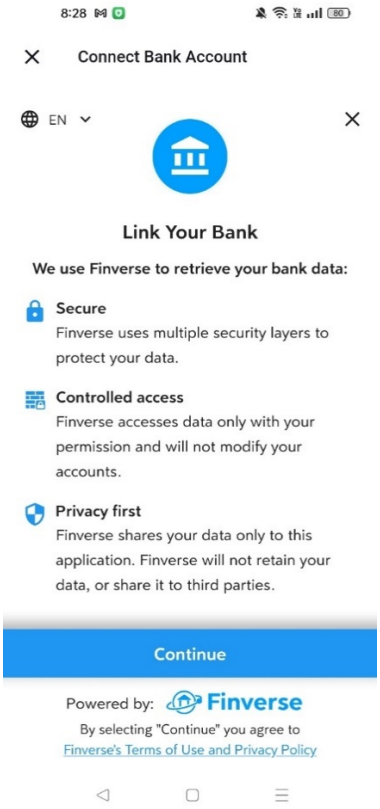


Figure 5.7.2 Add bank account - 2

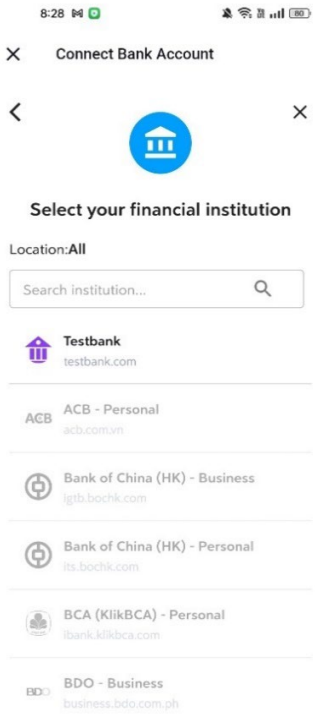


Figure 5.7.3 Select financial institution

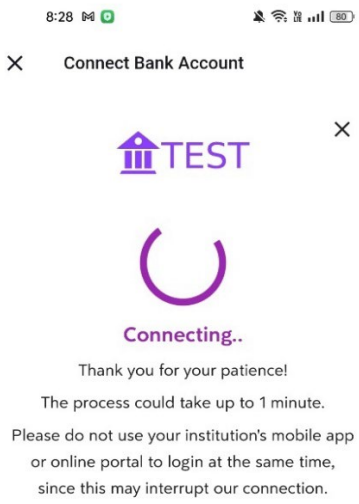


Figure 5.7.4 Connecting with Finverse API

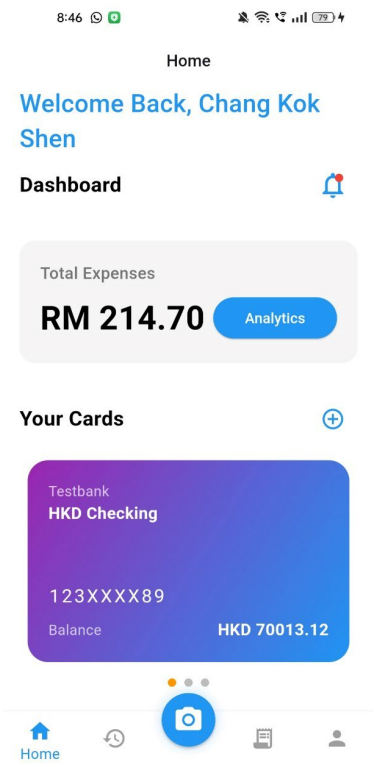


Figure 5.7.5 Test bank account data retrieved shown in cards

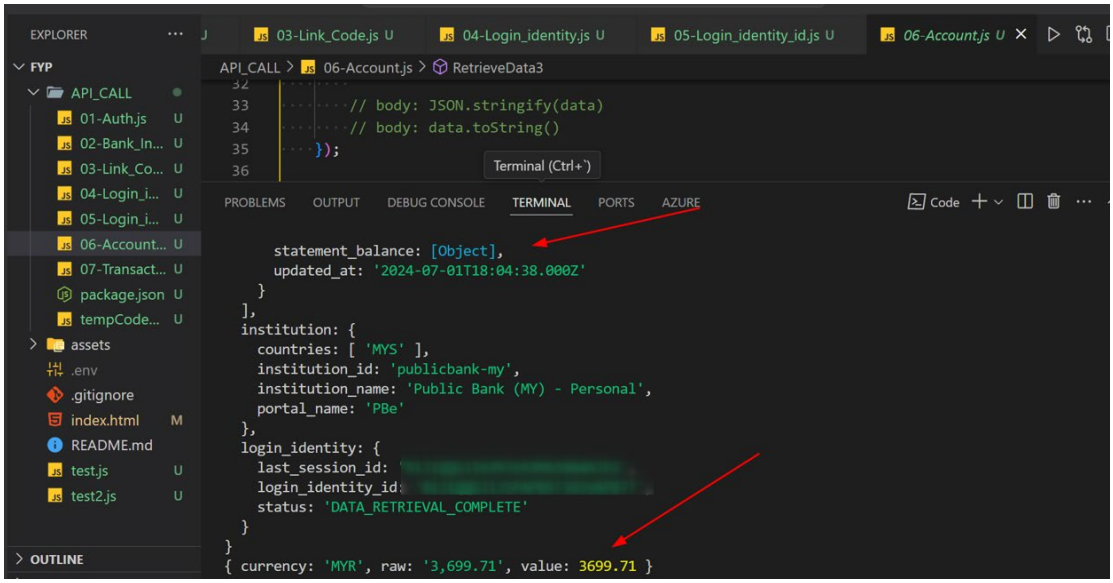


Figure 5.7.6 Retrieval of real bank data - Public Bank

5.8 View Recent Transactions and Total Expenses

Below Figure 5.8.1 shows the aggregated total expenses across manual data entry, voice recognition data entry and receipts scanned, whereas in Figure 5.8.2 the top 4 most recent transactions are showed from the recent transaction up to the oldest transaction.

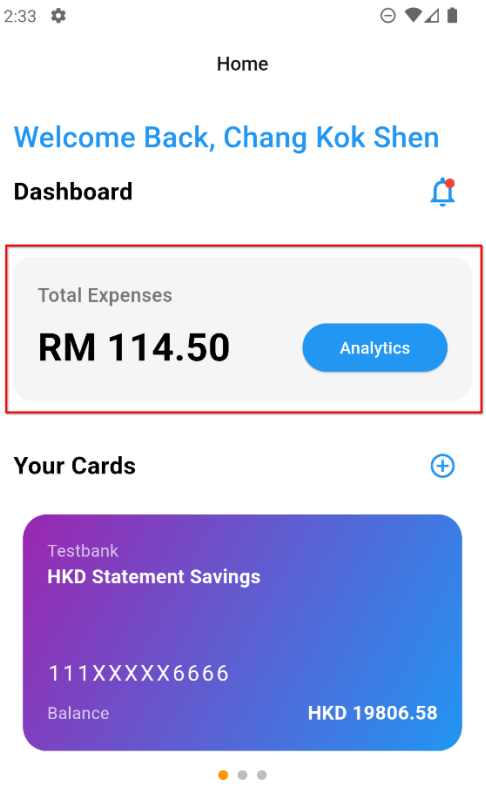


Figure 5.8.1 Total aggregated expenses

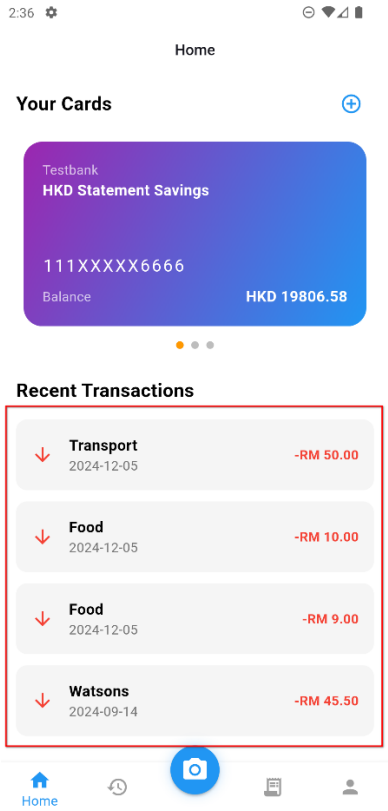


Figure 5.8.2 Top Recent Transactions

5.9 Transaction Dashboard

Another important feature of the smart financial tracker is that it allows users to understand their data by giving them visualisation through a dashboard divided into a few main segments. But before this, user must first tap on the analytics button as shown in Figure 5.9.1 in the home screen. Then, they are defaulted to the first tab which shows different segments of analytical data as shown from Figure 5.9.2 – Figure 5.9.3 including the current spending of the month, the spending trend of the last 6 months, all time expenses breakdown, weekly breakdown of the current month and the all-time spending categories.

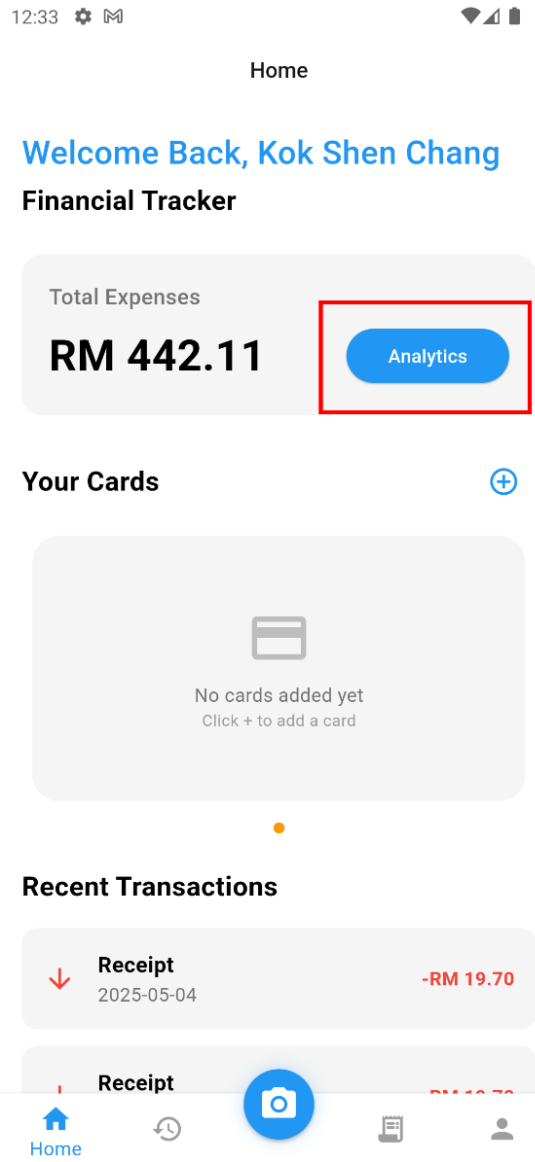


Figure 5.9.1 Home Screen – analytics button

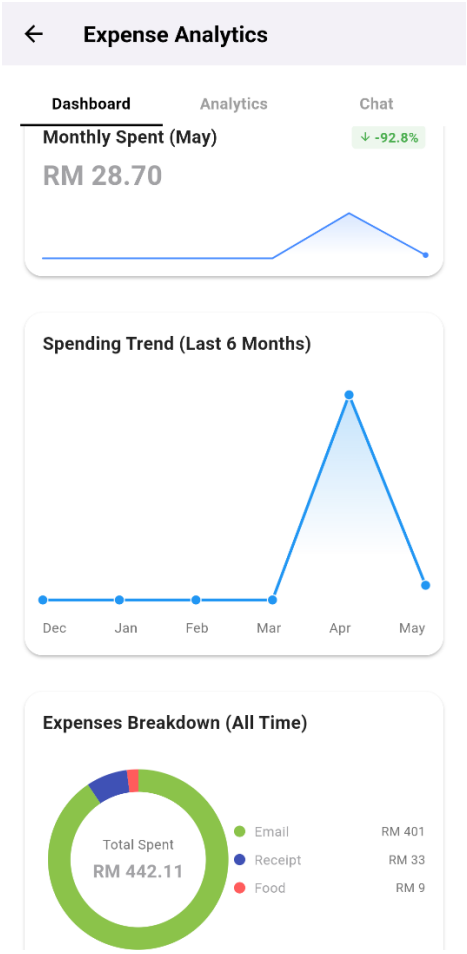


Figure 5.9.2 Home Screen – analytics button

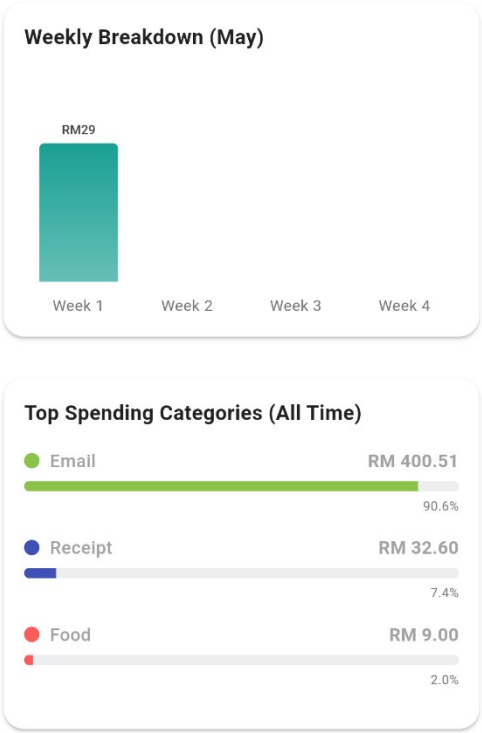


Figure 5.9.3 Home Screen – analytics button

5.10 Transaction Analytics

Besides, below Figure 5.10.1 and Figure 5.10.2 represent more specialised analytics whereby user can filter the month, year to see their total expenses. At the end of the page, they can export it as a financial report for RAG purpose. In the below example, only scraped email transactions were considered. However, other expenses such as receipts, manual expenses and voice data entry would also appear in the centralised Financial PDF report. As shown in Figure 5.10.3 - Figure 5.10.4, important metrics such as total spend, average transactions, number of total transactions for the month, highest category are shown to the user. On the 2nd page of the Report, the total transaction history is formatted into a structured table with headers such as No, Date, Description, Category and Amount.

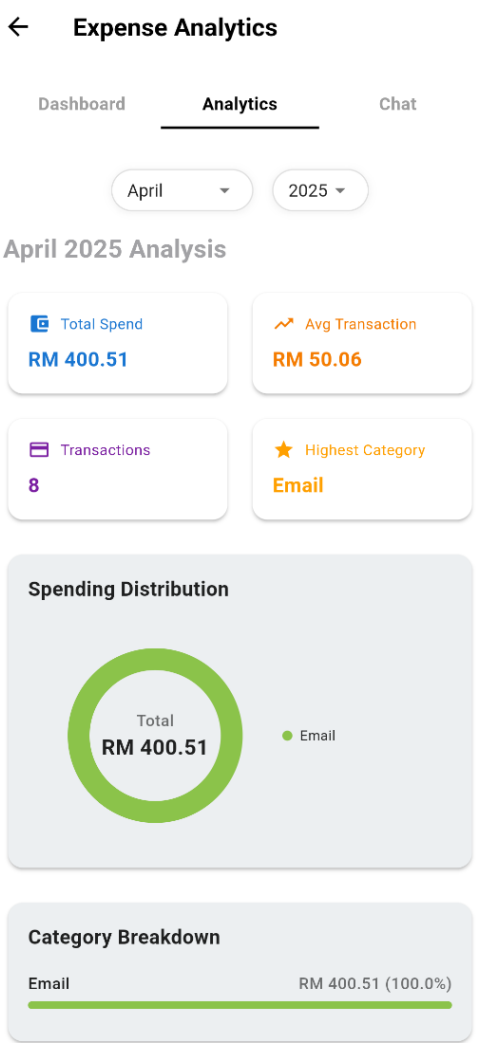


Figure 5.10.1 Analytics tab - 1

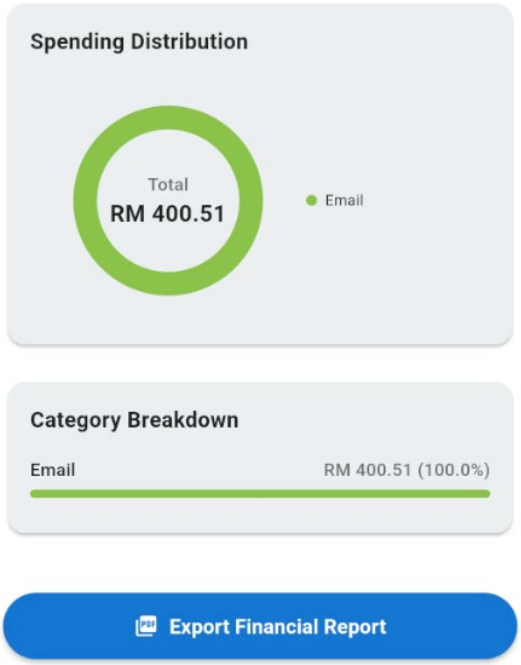


Figure 5.10.2 Analytics tab - 2

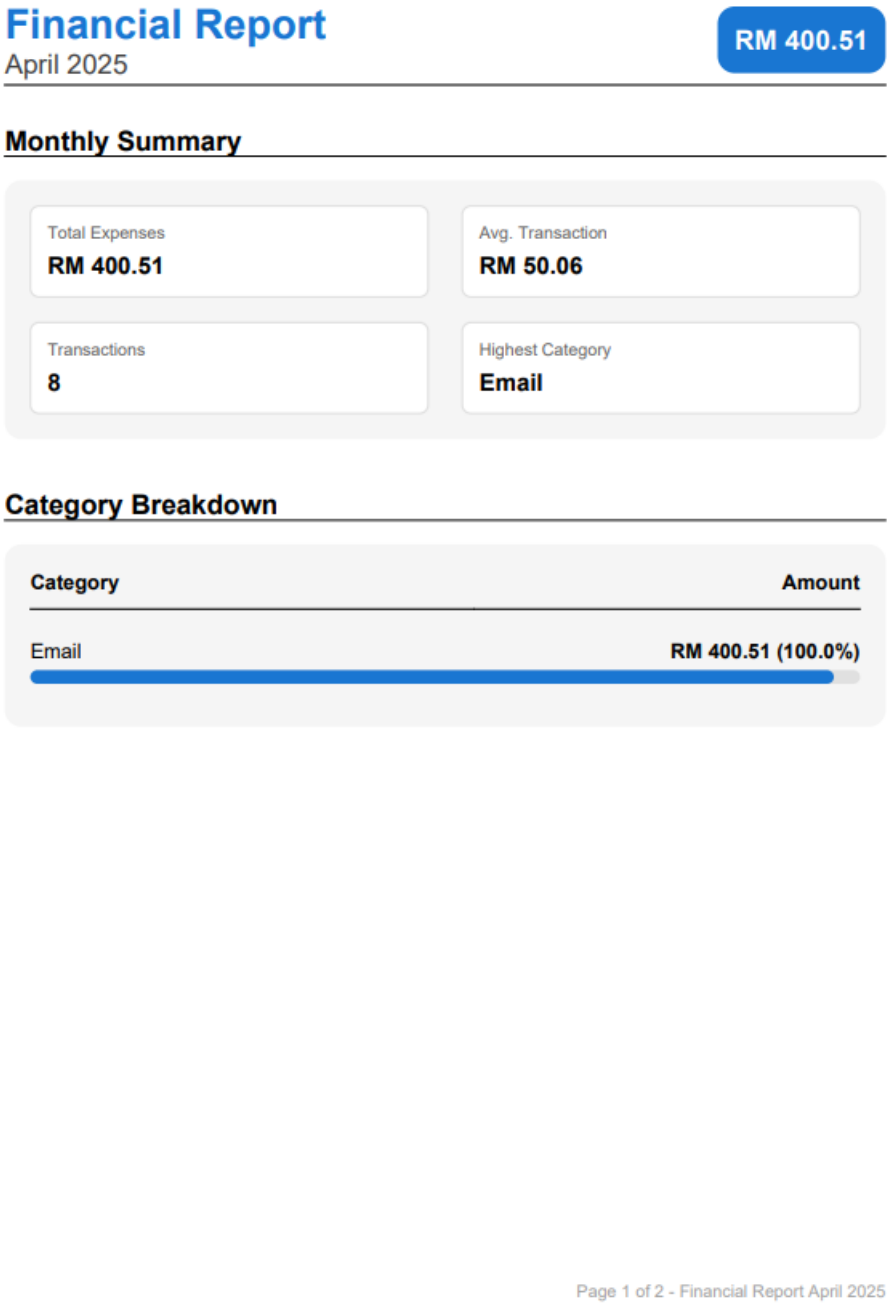
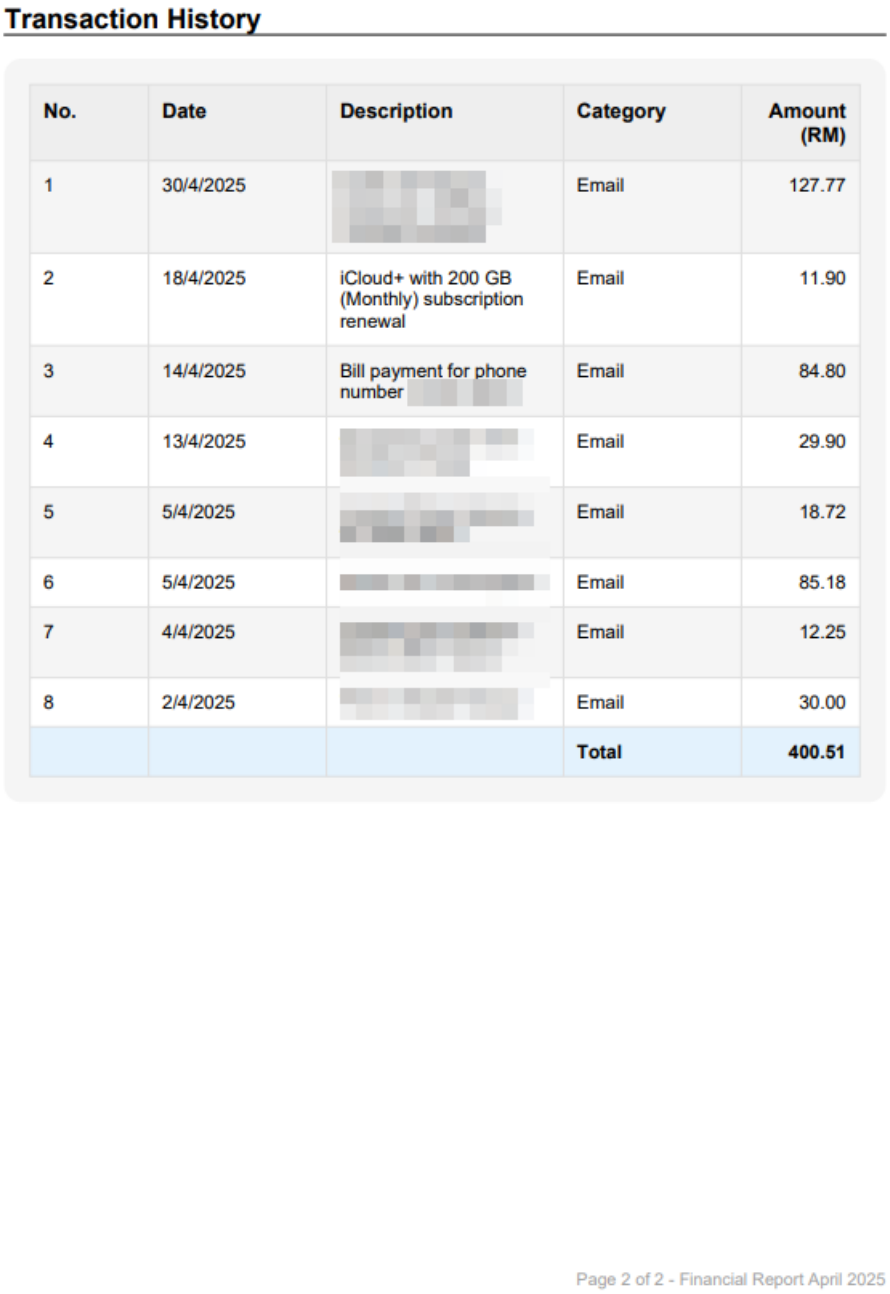


Figure 5.10.3 Financial Report April Page 1



Transaction History

No.	Date	Description	Category	Amount (RM)
1	30/4/2025	[redacted]	Email	127.77
2	18/4/2025	iCloud+ with 200 GB (Monthly) subscription renewal	Email	11.90
3	14/4/2025	Bill payment for phone number [redacted]	Email	84.80
4	13/4/2025	[redacted]	Email	29.90
5	5/4/2025	[redacted]	Email	18.72
6	5/4/2025	[redacted]	Email	85.18
7	4/4/2025	[redacted]	Email	12.25
8	2/4/2025	[redacted]	Email	30.00
			Total	**400.51**

Page 2 of 2 - Financial Report April 2025

Figure 5.10.4 Financial Report April Page 2

5.11 PDF RAG Chatbot

The Chat tab of the expense analytics screen enables intelligent document interaction through a streamlined process. As shown in Figure 5.11.1, users upload their financial PDF reports which undergo RAG processing. Figure 5.11.2 demonstrates how users can ask natural language questions about their financial data, such as spending insights requests. Figure 5.11.3 displays the system's response, providing structured financial analysis including total expenses (RM400.51), categorization (100% "Email" expenses), transaction details (8 transactions averaging RM50.06), and actionable recommendations for reducing email-related costs. The RAG pipeline supports its answers by referencing specific pages from the source document, allowing users to verify information directly while making financial analysis more accessible and actionable.

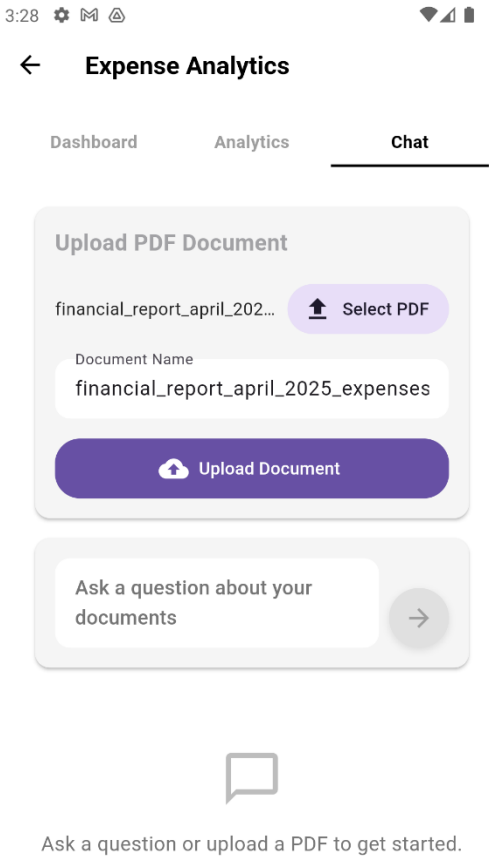


Figure 5.11.1 Upload generated Financial Report PDF

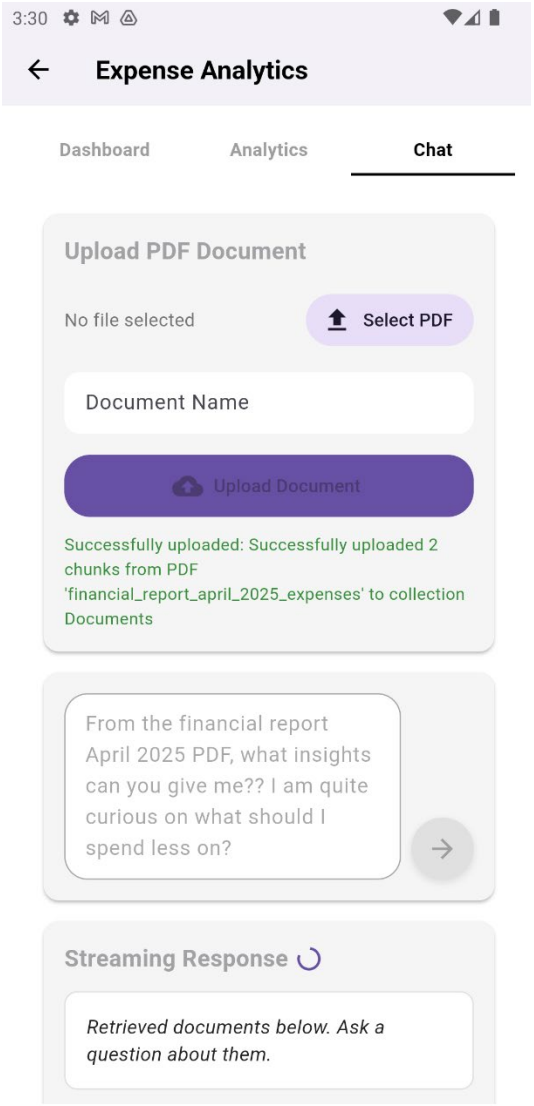


Figure 5.11.2 Streaming chat response

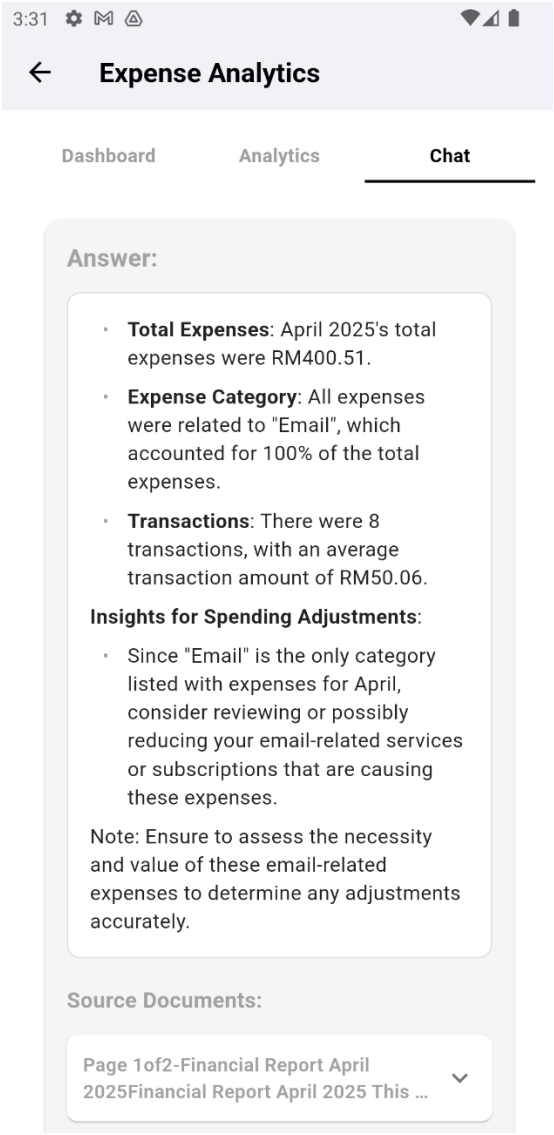


Figure 5.11.3 Chatbot returned answer

5.12 Scrape Email Transactions

The smart financial tracker also allows users to scrape their monthly email transactions. In the example above, the email scraping logic will be involved scraping transactions from April 1st to April 30th. As shown in Figure 5.12.1 below, upon tapping of the Connect & Fetch Transactions button, user is allowed to choose their desired Gmail Account for scraping. To note, the user's Gmail account must contain at least 1 transaction related email or else the scraping workflow will terminate early without retrieving any data. Then in Figure 5.12.2, the scraped email transactions are returned to the user. Depending on the number of emails containing transactions, the process could take between 1 minute to 1 minute 30 seconds. Lastly as shown in Figure 5.12.3 – Figure 5.12.4, the existence of 2 emails were verified from manually searching in Gmail.

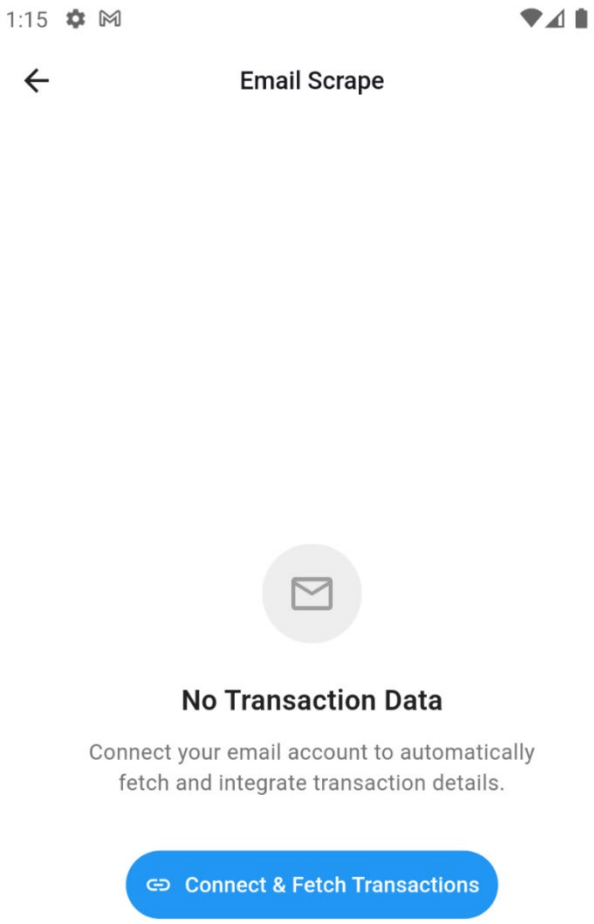


Figure 5.12.1 User is prompted to Connect & Fetch Transactions

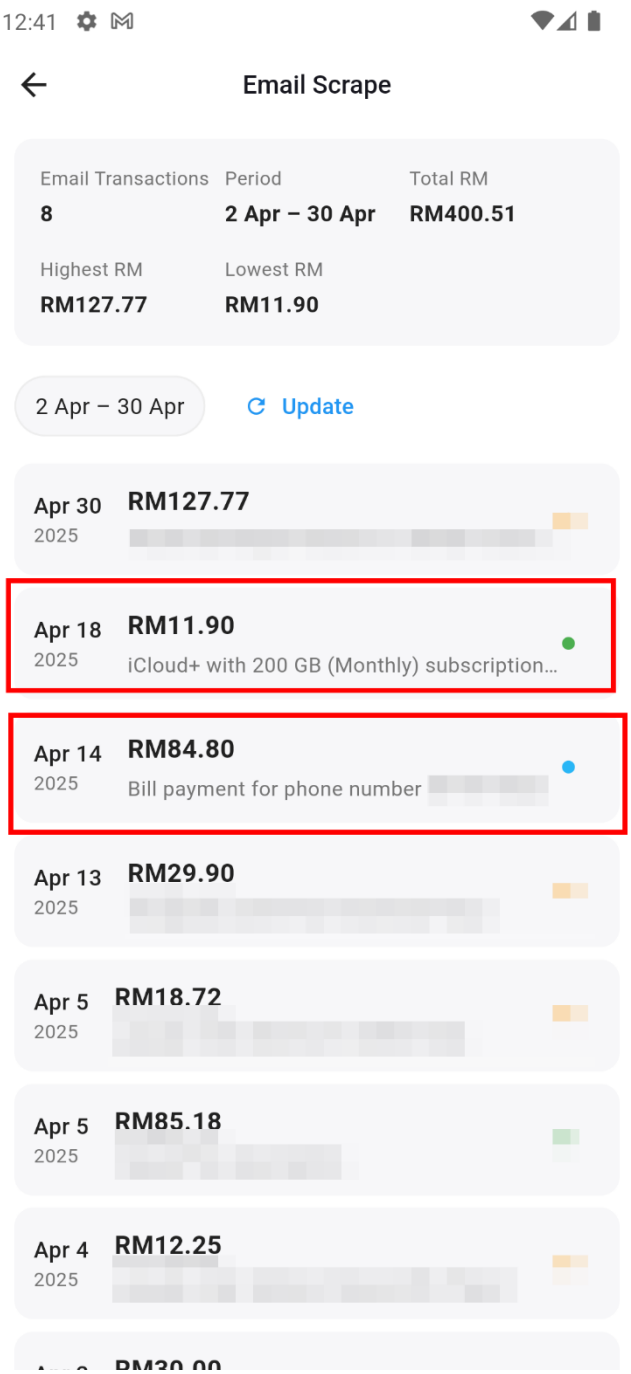


Figure 5.12.2 User retrieves scraped email transactions from 1st – 30th April

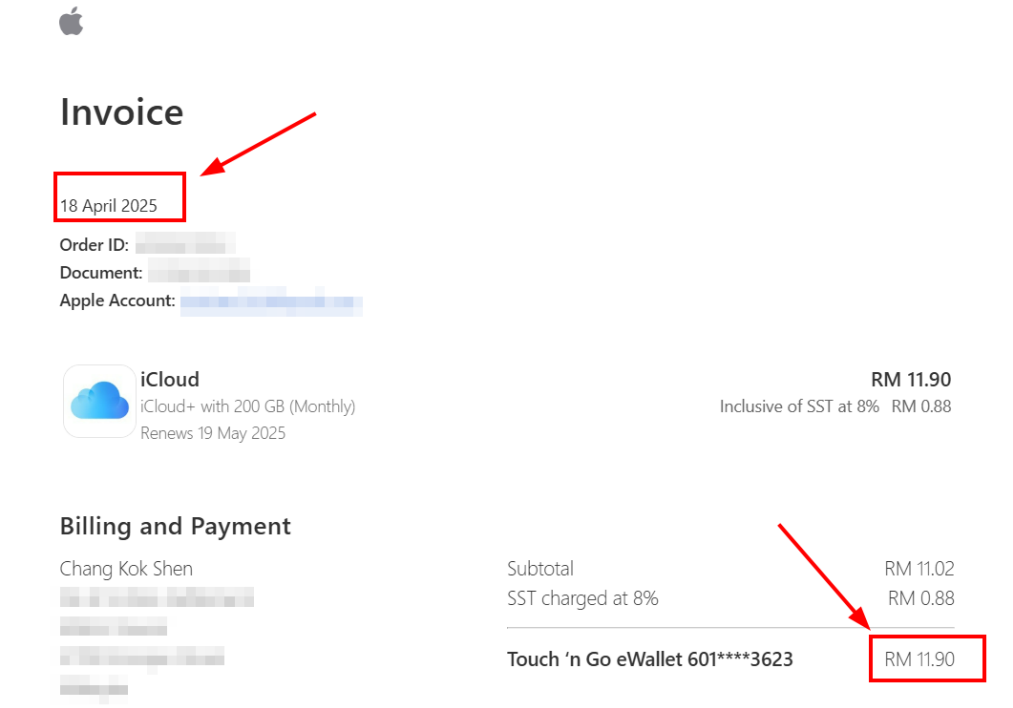


Figure 5.12.3 Apple Invoice email

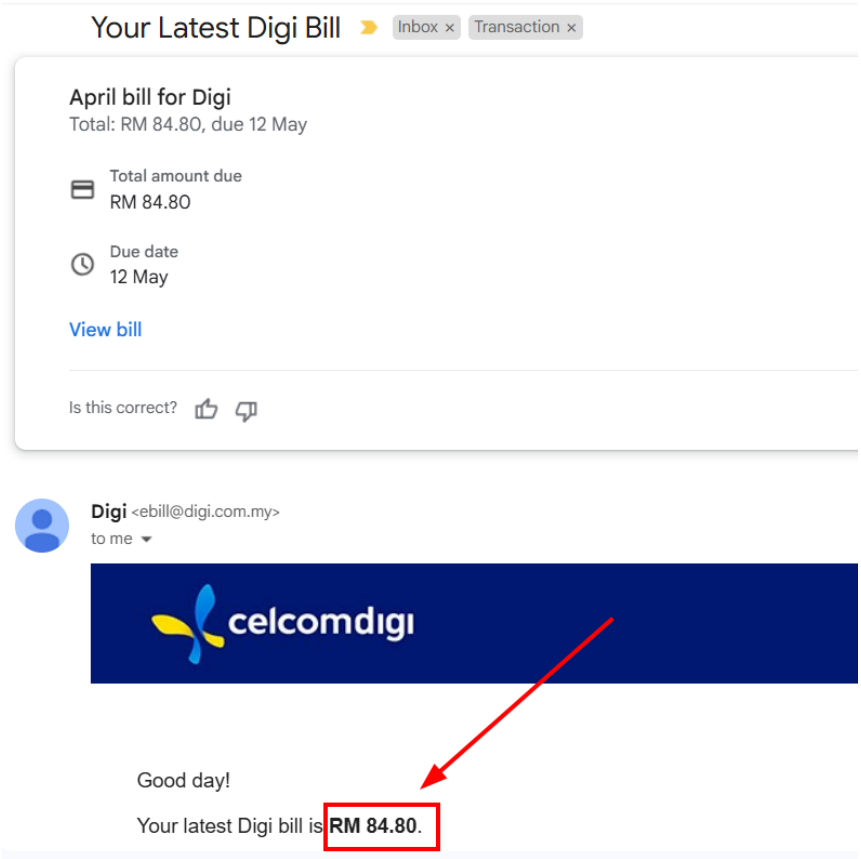


Figure 5.12.4 Digit Payment email

5.13 Summary

In summary, the full working mobile application showcases all the main modules developed in FYP 2. The main modules included login and sign up authentication for different users, auto categorisation of receipts, manual expense entry, voice recognition expense entry, simulation of adding financial accounts, receipt sharing, financial analytics and scrape email transactions. Some additional miscellaneous features not shown included modify profile settings function, edit and delete existing expenses operation. All the modules developed serve to tackle all the objective mentioned in Chapter 1.

CHAPTER 6 Conclusion and Future Work

6.1 Conclusion

Financial trackers aim to serve the purpose of tracking a user's daily expenditure, most financial trackers do not allow their user base to effortlessly and intelligently track their expenses. With the evolvement of Artificial Intelligence, programming frameworks, extensive open-source libraries, open banking APIs. More intelligent financial trackers are developed to cater to different users' needs. For instance, scanning receipts or invoices and automatically extracting key insights without manual intervention. Another example would be integrating financial bank accounts into a centralised mobile application, which allows users to have a more aggregated overview of info such as bank account savings, banking statements and total balance.

The ideation of a Smart Financial Tracker Mobile Application allows users lessen the burden of traditional manual data entry for expenses with more fast and intuitive features such as voice recognition data entry and automatic receipt data entry. On top of that, users are still able resort to manual data expense entry according to specific scenarios and preference. Besides, the mobile application also allows users to connect to their personal banking account with the help of 3rd party open banking API service provider Finverse. Not to mention, an innovative approach of scraping relevant email transactions through n8n workflow platform was introduced. Then, after all the user's data is inputted, it is aggregated into a summarised monthly financial report. More importantly, user can gain personalised financial insights by simply asking questions about their spending patterns in everyday language.

In summary, this Smart Financial Tracking Mobile Application represents a significant advancement over traditional financial trackers by providing an integrated, automated, and intelligent platform tailored to Malaysian users, meeting all stated objectives and successfully addressing the limitations identified in existing systems.

6.2 Future work and recommendations

The smart financial tracker has significant potential for expansion. In the future, we can integrate real-time WhatsApp notifications to deliver personalised daily, weekly, or monthly summaries of financial transactions directly to users. This enhancement will provide timely insights without requiring users to open the application.

Following the emerging trend of AI agentic workflows, Google's Agent Development Kit (ADK) can be leveraged to create multi-agent applications and seamlessly integrate them into the mobile platform. These intelligent agents could proactively analyse spending patterns, suggest budget optimizations, and execute routine financial tasks. Like the RAG-based financial pipeline mentioned, ADKs can transform the current query-response model into an ecosystem of specialised financial agents working in parallel. For example, one agent could focus on categorizing ambiguous transactions from scraped emails, while another could analyse voice entries for spending pattern anomalies, which extends the current analytics capabilities with more financial guidance.

Lastly, the bank integration module established with Finverse API could be enhanced through advanced communication standards such as Model Context Protocol and Agent2Agent Protocol. These protocols would build upon the current Malaysian bank connections to create a more robust and secure data exchange pipeline, addressing the current limitations in displaying comprehensive financial information while maintaining the strong user privacy standards established in the current implementation.

REFERENCES

[1] D. Efimova, “AI in Fintech 2024 | EPAM Startups & SMBs,” *startups.epam.com*, Jul. 24, 2024. https://startups.epam.com/blog/ai-in-fintech

[2] Volopay, “Features to Look for in Business Expense Tracker,” *Volopay*, Nov. 04, 2022.https://www.volopay.com/expense-management/features-to-consider-in-business-expense-tracker/

[3] D. Fuscaldo, “Financial Tracking 101: Best Practices,” *business.com*, Feb. 02, 2024. https://www.business.com/articles/financial-tracking-101/

[4] M. Boyle, “Financial History: The Evolution of Accounting,” *Investopedia*, Apr. 24, 2020. https://www.investopedia.com/articles/08/accounting-history.asp

[5] H. Kamel, “TnG eWallet Gains New GOfinance Feature,” *Lowyat.NET*, Jul. 09, 2024. https://www.lowyat.net/2024/326118/tng-ewallet-gains-new-gofinance-feature/

[6] A. Dennis, Barbara Haley Wixom, David Paul Tegarden, and E. Seeman, *System analysis & design: an object-oriented approach with UML*, 5th ed. Hoboken, Nj: Wiley, 2015.

[7] K. Bhimani and N. Sorathiya, “Flutter vs Dart: Revolutionizing App Development,” *www.dhiwise.com*, May 02, 2024. https://www.dhiwise.com/post/flutter-vs-dart-insights-into-the-future-of-app-development

APPENDIX A

A.1 POSTER

SMART FINANCIAL TRACKER

UTAR
UNIVERSITI TUNKU ABDUL RAHMAN

INTRODUCTION

A Smart Financial Tracker solves everyday financial tracking challenges through Receipt Recognition, Voice Recognition, Financial Account Retrieval, and AI Powered Insights. This app streamlines expense management by combining these modern technologies into one user-friendly mobile application solution.

KEY FEATURES

Receipt Recognition

Voice Recognition

Financial Account Retrieval

AI Powered Insights

PROBLEMS SOLVED

Afraid of lost receipts

Receipt Recognition digitises and stores them instantly

Too busy to log expenses

Voice Recognition lets you record on-the-go, as simple as speaking to your phone

Scattered bank platforms

Centralise financial account retrieval in one place

Confused about spending habits

AI Insights analyses and guides your finances with personalised recommendations

Project Developer : Chang Kok Shen