

Development of Personal Finance Management Application

BY

WONG ZHI XUAN

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)

JUNE 2024

REPORT STATUS DECLARATION FORM

Title: Development of Personal Finance Management
Application

Academic Session: JUNE 2024

I WONG ZHI XUAN
(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,



(Author's signature)



(Supervisor's signature)

Address:

C-23, Kampung Baru
Tanah Mas, 35500, Bidor
Perak

Tan Joi San

Supervisor's name

Date: 12 SEPTEMBER 2024

Date: 12 Sep 2024

Universiti Tunku Abdul Rahman			
Form Title : Sample of Submission Sheet for FYP/Dissertation/Thesis			
Form Number: FM-IAD-004	Rev No.: 0	Effective Date: 21 JUNE 2011	Page No.: 1 of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNIVERSITI TUNKU ABDUL RAHMAN

Date: 12 SEPTEMBER 2024

SUBMISSION OF FINAL YEAR PROJECT

It is hereby certified that WONG ZHI XUAN (ID No: 21ACB06564) has completed this final year project entitled “ Development of Personal Finance Management Application ” under the supervision of DR TAN JOI SAN (Supervisor) from the Department of Computer Science , Faculty of Information And Communication Technology .

I understand that University will upload softcopy of my final year project in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,




WONG ZHI XUAN
(Student Name)

*Delete whichever not applicable

DECLARATION OF ORIGINALITY

I declare that this report entitled “**Development of Personal Finance Management Application**” is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature :  _____

Name : WONG ZHI XUAN

Date : 12 SEPTEMBER 2024

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor Dr Tan Joi San for giving me this great opportunity to get involved in mobile application development. This is my first step to establish a career in mobile application development and this will be very helpful for my future. Thank you very much.

Apart from this, I would also like to thank my parents and my family for giving me their love and support throughout the course. I would also like to thank my course's lecturers and coursemates who have given me a lot of help and knowledge in this field. Last but not least, I would like to thank my friends for their encouragement and motivation.

ABSTRACT

Efficient personal finance management is crucial for individuals to achieve their financial goals and secure their future. Existing approaches often face challenges such as inadequate expense tracking, unorganised transaction categorisation, and limited financial awareness. Inadequate expense tracking results from the manual recording of daily expenses, leading to incomplete records and overspending. Unorganized transaction categorization is time-consuming and prone to errors, hindering the analysis of spending patterns. Limited financial awareness arises from a lack of financial education and a focus on short-term needs. Several research on existing applications were conducted to identify the strengths and weaknesses of the applications. By addressing these challenges, this project aims to empower individuals with the tools and knowledge to manage their finances effectively. This project adopts the Rapid Application Development (RAD) methodology, emphasizing prototyping and rapid feedback. The project encompasses phases for determining requirements, user design, construction, and implementation. This iterative approach ensures the application meets user expectations and functions seamlessly. The proposed solution includes modules which are user authentication and profile management module, virtual wallet module, transaction tracking and categorization module, income or saving recording module, manual transaction input module, budget management and calculation module, goal-setting and tracking module, analytics module as well as receipt OCR module. Ultimately, this project led to better financial control and decision-making.

TABLE OF CONTENTS

TITLE PAGE	i
REPORT STATUS DECLARATION FORM	ii
FYP THESIS SUBMISSION FORM	iii
DECLARATION OF ORIGINALITY	iv
ACKNOWLEDGEMENTS	v
ABSTRACT	vi
TABLE OF CONTENTS	vii
LIST OF FIGURES	x
LIST OF TABLES	xv
LIST OF ABBREVIATIONS	xvii
CHAPTER 1 INTRODUCTION	1
1.1 Problem Statement	1
1.2 Background and Motivation	3
1.3 Project Objectives	4
1.4 Project Scope and Direction	5
1.5 Proposed Approach / Study	8
1.6 Highlight of what have been archived	9
1.7 Report Organization	10
1.8 Summary	10
CHAPTER 2 LITERATURE REVIEW	11
2.1 Overview	11
2.2 Existing Applications	11
2.2.1 Spendee	11
2.2.2 Mint	17
2.2.3 YouNeedABudget (YNAB)	21
2.4.4 Goodbudget	25
2.3 Comparision of Existing Application	28
2.4 Limitations of Previous Studies	29

2.5	Proposed Solution	29
2.6	Summary	30
CHAPTER 3 SYSTEM DESIGN		31
3.1	Overview	31
3.2	Use-Cases Diagram	31
3.3	Use Case Description	32
3.4	Activity Diagram	52
3.5	Program Development	80
	3.5.1 Front-End and Back-End Development	80
	3.5.2 User Authentication and Profile Management Development	82
	3.5.3 Top-Up Development	86
	3.5.4 Scan QR Development	87
	3.5.5 Analytics Development	89
	3.5.6 Budget Development	91
	3.5.7 Goal Development	94
	3.5.8 Manual Input Transaction Development	97
	3.5.9 Transaction History Development	99
	3.5.10 Receipt OCR Development	102
3.6	Summary	104
CHAPTER 4 METHODOLOGY AND TOOLS		105
4.1	Overview	105
4.2	Methodology	105
	4.2.1 Determining Requirements Phase	106
	4.2.2 User Design Phase	106
	4.2.3 Construction Phase	106
	4.2.4 Implementation Phase	107
4.3	Tools to Use	107
4.4	Timeline	109
	4.4.1 Overview	109
	4.4.2 Gantt Chart	110

4.5	Summary	110
CHAPTER 5 SYSTEM IMPLEMENTATION		111
5.1	Overview	111
5.2	User Authentication Screens	111
5.3	Home Screen	114
5.4	Top Up Screen	115
5.5	Scan QR Code Screen	117
5.6	Analytics Screen	119
5.7	Budget Screen	121
5.8	Goal Screen	124
5.9	Manual Input Transaction Screen	127
5.10	All Transaction Screen	129
5.11	Receipt OCR Screen	132
5.12	Profile Screen	134
5.13	Summary	135
CHAPTER 6 CONCLUSION		136
6.1	Project Review, Discussion and Conclusion	136
6.2	Novelties and Contributions	137
6.3	Future Work	138
REFERENCES		139
WEEKLY LOG		142
POSTER		148
PLAGIARISM CHECK RESULT		153
FYP2 CHECKLIST		154

LIST OF FIGURES

Figure Number	Title	Page
Figure 1.5.1	Flowchart of Personal Finance Management Application	8
Figure 2.2.1.1	Spendee's Add Transaction Feature	13
Figure 2.2.1.2	Spendee's Syn with Bank Features	13
Figure 2.2.1.3	Spendee's Overview Section	14
Figure 2.2.1.4	Spendee's Categorization Features	14
Figure 2.2.1.5	Spendee's Packages and Their Features	15
Figure 2.2.2.1	The Monthly Cashflow in Mint	18
Figure 2.2.2.2	All Account in One Place Feature by Mint	19
Figure 2.2.2.3	Mint's Bill Negotiation Feature	19
Figure 2.2.2.4	Mint's Credit Score Feature	20
Figure 2.2.3.1	YNAB's Auto-Assigns Funds	22
Figure 2.2.3.2	YNAB's Color-coding to Indicate Each Category Status	23
Figure 2.2.3.3	YNAB's Loan Payoff Simulator and The Effect on The Loan	23
Figure 2.2.4.1	The Envelops Tab in Goodbudget	26
Figure 2.2.4.2	The Transaction Tab in Goodbudget	26
Figure 2.2.4.3	The Reports Tab in Goodbudget	27
Figure 2.2.4.4	The Differences Between Goodbudget Free Version and Goodbudget "Plus" Version	27
Figure 3.2.1	Use Case Diagram for Personal Finance Management Application System	31
Figure 3.4.1	Activity Diagram for Login	52
Figure 3.4.2	Activity Diagram for Log Out	53
Figure 3.4.3	Activity Diagram for Create Profile	54
Figure 3.4.4	Activity Diagram for View Profile	55
Figure 3.4.5	Activity Diagram for Edit Profile	56
Figure 3.4.6	Activity Diagram for Top Up	57
Figure 3.4.7	Activity Diagram for Make Payment	58

Figure 3.4.8	Activity Diagram for Categorize Transaction	59
Figure 3.4.9	Activity Diagram for Manual Input Transaction	60
Figure 3.4.10	Activity Diagram for Set Budget	61
Figure 3.4.11	Activity Diagram for Edit Budget	62
Figure 3.4.12	Activity Diagram for Duplicate Budget	63
Figure 3.4.13	Activity Diagram for Download Budget	64
Figure 3.4.14	Activity Diagram for Delete Budget	65
Figure 3.4.15	Activity Diagram for Add Transaction to Budget	66
Figure 3.4.16	Activity Diagram for Set Goal	67
Figure 3.4.17	Activity Diagram for Edit Goal	68
Figure 3.4.18	Activity Diagram for Duplicate Goal	69
Figure 3.4.19	Activity Diagram for Download Goal	70
Figure 3.4.20	Activity Diagram for Delete Goal	71
Figure 3.4.21	Activity Diagram for Add Transaction to Goal	72
Figure 3.4.22	Activity Diagram for View Analytics	73
Figure 3.4.23	Activity Diagram for View Transaction for Admin	74
Figure 3.4.24	Activity Diagram for View Transaction for User	75
Figure 3.4.25	Activity Diagram for View Balance for Admin	76
Figure 3.4.26	Activity Diagram for View Balance for User	77
Figure 3.4.27	Activity Diagram for View User Authentication	78
Figure 3.4.28	Activity Diagram for Perform OCR	79
Figure 3.5.1.1	Application's screen in Screen Folder	81
Figure 3.5.1.2	"auth_provider.dart"	81
Figure 3.5.1.3	"main.dart"	82
Figure 3.5.2.1	Functions for User Authentication_1	83
Figure 3.5.2.2	Functions for User Authentication_2	83
Figure 3.5.2.3	Functions for User Authentication_3	84
Figure 3.5.2.4	Functions for Update User Information in "auth_provider.dart"	85
Figure 3.5.2.5	Functions for Update User Information in "editProfileScreen.dart"	85
Figure 3.5.2.6	Functions for Sign Out	86
Figure 3.5.3.1	Functions for Update and Fetch Wallet Balance	86

Figure 3.5.3.2	Functions to Start a Payment for Top Up	87
Figure 3.5.3.3	“_handleTopUpSuccess” Function	87
Figure 3.5.4.1	“_onQRViewCreated” Function for QR Code Scanner Set Up	88
Figure 3.5.4.2	“_confirmPayment” Function	88
Figure 3.5.4.3	“_updateBudget” Function	89
Figure 3.5.5.1	Function to Get Value for Monthly Spending Bar Chart	90
Figure 3.5.5.2	Function to Get Value for Spending Category Breakdown Pie Chart	90
Figure 3.5.6.1	Function to Calculate Daily Advise	91
Figure 3.5.6.2	Function to Add or Update Budget	92
Figure 3.5.6.3	Function to Duplicate Budget	92
Figure 3.5.6.4	Function to Delete Budget	93
Figure 3.5.6.5	Function to Fetch Transaction for Budget	93
Figure 3.5.6.6	Function to Download Budget as PDF	94
Figure 3.5.7.1	Function to Add or Update Goal	95
Figure 3.5.7.2	Function to Duplicate Goal	95
Figure 3.5.7.3	Function to Delete Goal	96
Figure 3.5.7.4	Function to Fetch Transaction for Goal	96
Figure 3.5.7.5	Function to Download Goal as PDF	97
Figure 3.5.8.1	Function to Fetch Budget and Goal to Manual Transaction Input Screen	98
Figure 3.5.8.2	“_submitForm” Function to Save a Transaction	98
Figure 3.5.9.1	Function to Fetch Transaction History	100
Figure 3.5.9.2	Function to Fetch Transaction History to All Transaction Screen	100
Figure 3.5.9.3	Function to Fetch Budget and Goal to Transaction Details Screen	100
Figure 3.5.9.4	Function to Upload Image	101
Figure 3.5.9.5	Function to Load the Image	101
Figure 3.5.9.6	Function to Update the Transaction with Budget or Goal	102
Figure 3.5.10.1	Function to Call the Asprise API	103
Figure 3.5.10.2	Extract Text Result in JSON Format	103

Figure 3.5.10.3	Function to Show the Selected Item	104
Figure 3.5.10.4	Function to Save the Selected Item to Transaction	104
Figure 4.2.1	The Phase of Methodology	105
Figure 4.4.2.1	Project Timeline Gantt Chart	110
Figure 5.2.1	Welcome Screen	112
Figure 5.2.2	Phone Entry Screen for Login or Register	112
Figure 5.2.3	Verification Screen	113
Figure 5.2.4	Create Profile Screen	113
Figure 5.3.1	Home Screen	115
Figure 5.4.1	Top Up Screen	116
Figure 5.4.2	Payment Process by Stripe	117
Figure 5.5.1	QR Code Scanner Screen	118
Figure 5.5.2	Confirm Button Lock	118
Figure 5.5.3	Confirm Button Unlock	119
Figure 5.6.1	Monthly Spending Bar Chart Graph	120
Figure 5.6.2	Spending Category Breakdown Pie Chart	120
Figure 5.7.1	Budget Screen	121
Figure 5.7.2	Add Budget Screen	122
Figure 5.7.3	Edit Budget Screen	122
Figure 5.7.4	Duplicate of the Budget	123
Figure 5.7.5	Transaction List of the Budget	123
Figure 5.8.1	Goal Screen	124
Figure 5.8.2	Add Goal Screen	125
Figure 5.8.3	Edit Goal Screen	125
Figure 5.8.4	Duplicate of the Goal	126
Figure 5.8.5	Transaction List of the Goal	126
Figure 5.9.1	Manual Input Transaction Floating Button	127
Figure 5.9.2	Expense Transaction Entry Form	128
Figure 5.9.3	Income/Savings Transaction Entry Form	128
Figure 5.10.1	“More” Icon to Navigate to All Transaction Screen	129
Figure 5.10.2	All Transaction Screen with the Filters	130
Figure 5.10.3	Income/Savings Transaction Details Screen	130
Figure 5.10.4	Expense Transaction Details Screen	131

Figure 5.10.5	Transaction Details with Image Uploaded	131
Figure 5.11.1	Extract Text Button After an Image Uploaded	132
Figure 5.11.2	Extract Text with Checkboxes	133
Figure 5.11.3	Selected Item Screen	133
Figure 5.12.1	User Profile Screen	134
Figure 5.12.2	Edit Profile Screen	135

LIST OF TABLES

Table Number	Title	Page
Table 2.3.1	Comparison of Existing Applications	28
Table 3.3.1	Login Use Case Description	32
Table 3.3.2	Log Out Use Case Description	32
Table 3.3.3	Create Profile Use Case Description	33
Table 3.3.4	View Profile Use Case Description	34
Table 3.3.5	Edit Profile Use Case Description	34
Table 3.3.6	Top Up Use Case Description	35
Table 3.3.7	Make Payment Use Case Description	35
Table 3.3.8	Categorize Transaction Use Case Description	36
Table 3.3.9	Manual Input Transaction Use Case Description	37
Table 3.3.10	Set Budget Use Case Description	38
Table 3.3.11	Edit Budget Use Case Description	39
Table 3.3.12	Duplicate Budget Use Case Description	40
Table 3.3.13	Download Budget Use Case Description	40
Table 3.3.14	Delete Budget Use Case Description	41
Table 3.3.15	Add Transaction to Budget Use Case Description	42
Table 3.3.16	Set Goal Use Case Description	43
Table 3.3.17	Edit Goal Use Case Description	44
Table 3.3.18	Duplicate Goal Use Case Description	44
Table 3.3.19	Download Goal Use Case Description	48
Table 3.3.20	Delete Goal Use Case Description	49
Table 3.3.21	Add Transaction to Goal Use Case Description	49
Table 3.3.22	View Analytics Use Case Description	47
Table 3.3.23	View Transaction Use Case Description	48
Table 3.3.24	View Balance Use Case Description	49
Table 3.3.25	View User Authentication Use Case Description	50
Table 3.3.26	Perform OCR Use Case Description	51

Table 4.3.1	Hardware Component and Requirement for Application Development	107
Table 4.3.2	Software Component and Requirement for Application Development	108

LIST OF ABBREVIATIONS

<i>AI</i>	Artificial Intelligence
<i>API</i>	Application Programming Interface
<i>iOS</i>	iPhone Operating System
<i>OCR</i>	Optical Character Recognition
<i>OTP</i>	One-time password
<i>QR Code</i>	Quick Response Code
<i>RAD</i>	Rapid Application Development
<i>URL</i>	Uniform Resource Locator
<i>YNAB</i>	YouNeedABudget

Chapter 1 Introduction

In this chapter, we present the problem statement, background and motivation, project objectives, project scope, and the outline of the thesis.

1.1 Problem Statement

Efficient management of personal finances is essential for individuals to achieve their financial goals and secure their future. However, existing approaches to managing personal finances often encounter a variety of challenges that lead to inefficiencies in financial planning and decision-making. These problems include inadequate expense tracking, unorganized transaction categorization and limited financial awareness and future risk preparedness.

i. Inadequate Expense Tracking and Overbudgeting

Inadequate expense tracking is due to the fact that individuals struggle to record every financial transaction they make consistently. This problem is caused by traditional or existing personal finance management applications which require individuals to record their daily expenses immediately and manually. This includes small daily expenses as well as larger irregular expenses. Individuals often rely on memory or sporadic note-taking to record their expenses. This delayed approach can result in forgetting or inaccurately recalling transactions, leading to incomplete expense records and an inaccurate view of overall spending. Over time, this can lead to incomplete records each month. Moreover, modern life involves a multitude of transactions across various payment methods such as credit cards, cash, mobile payments, and more. This complexity can make it challenging to accurately track and record every single transaction in a timely manner. Without an easy and efficient way to track expenses, individuals may underestimate how much they spend on certain categories, leading to overspending and financial strain. On the other hand, overbudgeting occurs when individuals allocate more money to certain categories than they actually need. This can be due to a lack of clarity about their actual spending patterns or a failure to account for unforeseen expenses. If individuals do not have access to real-time information about their financial behaviour, they are more likely to set unrealistic budgets, leading to frustration and difficulty in sticking to their financial plans.

ii. Unorganized Transaction Categorization

In today's digital age, individuals engage in numerous financial transactions daily from purchase to bill payment. Thus, the sheer volume of these transactions can make manually categorizing each one a time-consuming task. Moreover, financial transactions span a wide range of categories, including groceries, entertainment, bills, savings, investments, and more. Users need to navigate through numerous categories, each with its specific subcategories, making the process time-consuming. Although individuals can add some references or details during every transaction for future categories, they still need to manually track each transaction and assign it to a category, which can be tedious and error-prone. This process becomes even more inconvenient when dealing with a large number of transactions. Some transactions can be ambiguous in terms of categorization. For instance, a purchase at a department store might include both clothing and household items, creating uncertainty about which category to assign it to. Without an accurate categorization, individuals are unable to analyse their spending patterns, they cannot track which category they spent the most or overspent in certain unnecessary categories.

iii. Limited Financial Awareness and Future Risk Preparedness

This problem arises from a lack of comprehensive understanding of one's financial situation, including emergency funds, retirement planning, and potential investment opportunities. Many individuals have not received formal financial education that equips them with the necessary knowledge to manage their finances effectively. Without a strong foundation in financial concepts, individuals may struggle to make informed decisions and plan for the future. People often prioritize short-term needs over long-term financial planning. The immediate demands of daily life can overshadow the importance of setting aside funds for emergencies, retirement, or other future financial needs. Without accessible tools and resources that provide insights into financial health and future projections, individuals may find it challenging to assess their situation accurately and plan for potential risks.

1.2 Background and Motivation

According to the CFI Team [1], personal finance refers to the process of managing and planning personal activities which include saving, income generation, investment and personal protection. A financial plan or budget can serve as a summary of the process of managing one's own finances. Personal finance management enables individuals to make informed decisions based on their financial goals and aspirations, fostering a stable and secure financial future.

In the modern world, effective management of personal finances has become a critical skill for individuals to navigate the complexities of their financial lives. The management of one's financial resources, including saving, investing, budgeting and protecting, plays a key role in determining one's financial health and future security. This dynamic process is critical to achieving financial goals, whether it's buying a dream home, funding an education, or ensuring a comfortable retirement. The traditional approach to personal finance management, often relying on manual recording of expenses, sporadic note-taking, and fragmented transaction categorization, presents numerous challenges. These challenges can range from incomplete expense tracking leading to overspending, to a lack of financial awareness hampering future risk preparedness.

In light of these challenges, this project seeks to provide a solution that empowers individuals to take control of their financial destinies. By leveraging technology, data analysis, and user-friendly interfaces, our personal finance management application aims to streamline the management of personal finances. It offers a comprehensive suite of features designed to address the common pain points encountered in financial planning and decision-making. The proposed modules include user authentication and profile management module, virtual wallet module, transaction tracking and categorization module, income or saving recording module, manual transaction input module, budget management module, budget calculation module, goal-setting and tracking module, analytics module as well as receipt OCR module. The application strives to make personal finance management more accessible, efficient, and insightful. With these tools, individuals can gain a deeper understanding of their financial habits, track progress towards achieving their goals, and ultimately achieve greater financial stability and peace of mind.

The motivation behind developing this personal finance management application arises from the recognition of the numerous challenges individuals face in effectively managing their finances. These challenges include inadequate expense tracking, unorganised transaction categorisation, limited financial awareness and future risk preparedness. As such, there is a compelling need for a solution that addresses these issues and empowers individuals to take control of their financial well-being.

1.3 Project Objectives

This project aims to develop an application and efficient algorithms to increase effectiveness in personal finance management and provide users with advanced tools and features for better financial control and decision-making.

i. To Achieve Comprehensive Expense Tracking and Budget Control

The objective is to create a personal finance management application that automates expense tracking and budget calculations. The application aims to enable users to effortlessly record and categorize expenses while receiving real-time insights into their spending patterns by implementing e-wallets within the application. This can reduce reliance on memory or note-taking, ensuring that every transaction is accurately captured. The application will not only allow users to set daily, weekly, or monthly budgets but will also incorporate intelligent budget calculations. For instance, the application will dynamically adjust the remaining daily budgets based on ongoing spending patterns to prevent exceeding the monthly limit. This mechanism ensures that users maintain an accurate overview of their financial allocation and reduce the risk of overspending.

ii. To Implement Intelligent Transaction Categorization using AI

The objective is to integrate machine learning algorithms that can analyze transaction data and assign them to appropriate predefined categories. Users can conveniently review and import transaction details into the application. With the power of artificial intelligence, the application will recognize transaction keywords, patterns, and contextual information, ensuring accurate and automated categorization. The integration of AI algorithms will effectively eliminate the need for users to manually categorize each transaction. This is particularly convenient and

efficient as it can save users time as well as reduce categorisation errors when dealing with a high volume of transactions. The algorithms' ability to intelligently assign transactions to appropriate categories ensures that users can analyse their spending patterns and take further action to enhance their financial plans.

iii. To Enhance Financial Insights and Facilitate Risk Planning

This objective is to equip users with an advanced suite of tools for bolstering their financial awareness and conducting strategic risk planning by implementing goal-setting and analytics features. These features enable users to set and track financial goals encompassing various aspects of their financial lives. By offering goal-setting capabilities within the application, users will gain a panoramic view of their financial journey, including savings progress, retirement planning, investments, and risk mitigation. The goal-setting feature will prompt users to establish specific financial objectives aligned with their aspirations, such as building an emergency fund, paying off debt, or making an investment. Additionally, users will be able to visualize their spending habits through graphs in the analytics feature. Over the long term, users can enhance their financial insight and will be better equipped to facilitate risk planning.

1.4 Project Scope and Direction

The project's scope is to enhance personal financial management by providing individuals with user-friendly applications to effectively manage their financial activities. Features include user authentication and profile management module, virtual wallet module, transaction tracking and categorization module, income or saving recording module, manual transaction input module, budget management and calculation module, goal-setting and tracking module, analytics module as well as receipt OCR module.

i. User Authentication and Profile Management Module:

This module handles user registration and login. It ensures that each user has a unique account and manages their personal information securely. Users can create an account, log in with their credentials, and update their profile details if needed.

ii. Virtual Wallet Module

The virtual wallet module offers users a convenient way to conduct transactions. Users can utilize the virtual wallet for payments including features like scanning QR codes. Additionally, users can top up their balance within the wallet module. All transactions made by the user are automatically recorded without the need for manual record-keeping.

iii. Transaction Tracking and Categorization Module:

The Transaction Tracking module provides users with a seamless way to monitor their financial transactions directly from their transaction history. Users can conveniently view transactions recorded through QR code payments or manual entry. The module automatically captures essential details such as transaction amount, payment details, date, and merchant information. During the transaction process, users can select the appropriate category from a predefined list. This will simplify the tracking process and offer valuable insights into spending habits. This feature ensures accurate and up-to-date financial records.

iv. Income or Saving Recording Module:

Beyond monitoring expenses, the income or saving recording module empowers users to record their various income or saving sources. This encompasses regular monthly salaries, as well as additional income streams such as bonuses and investments. By accommodating both primary and supplementary sources of income, the application provides a holistic view of users' financial inflows.

v. Manual Transaction Input Module:

This module allows users to manually input transactions that may not be automatically captured, such as cash payments or other offline transactions. Users can enter details such as merchant information, transaction amount, date, category, and a brief description. This feature offers users complete control over their financial records even when they are not processed through the virtual wallet.

vi. Budget Management and Calculation Module:

The Budget Management and Calculation module empowers users to take control of their spending. Users can set budgets for specific timeframes which are daily, weekly or monthly to

manage their expenses effectively. The application tracks spending against these budgets in real-time, providing insights into financial habits. This module calculates and manages daily budgets based on the overall monthly budget. If a user surpasses their daily budget on a specific day, the remaining budgets for the remaining days are adjusted to ensure the monthly limit will not be exceeded.

vii. Goal-Setting and Tracking Module:

With this module, users can set financial goals based on their aspirations. For instance, saving for a holiday, paying off debt, or achieving investment milestones, the application helps users stay motivated by tracking progress and providing a visual representation of achievements.

viii. Analytics Module:

The Analytics module helps users gain deeper insights into their financial habits by providing visual representations of their spending patterns. Users can access various charts and graphs that display trends such as money spending and category spending breakdown. This feature allows users to identify areas where they can save, manage their budget more effectively, and make informed financial decisions based on clear, data-driven visualizations.

ix. Receipt OCR Module:

The Receipt OCR (Optical Character Recognition) module simplifies the process of recording purchases by scanning and extracting data from physical receipts. Users can upload an image of a receipt, and the module will automatically extract key details such as the merchant's name, transaction amount, and date. This information is then categorized and added to the user's transaction history, minimizing manual input and improving the accuracy of financial records.

1.5 Proposed Approach / Study

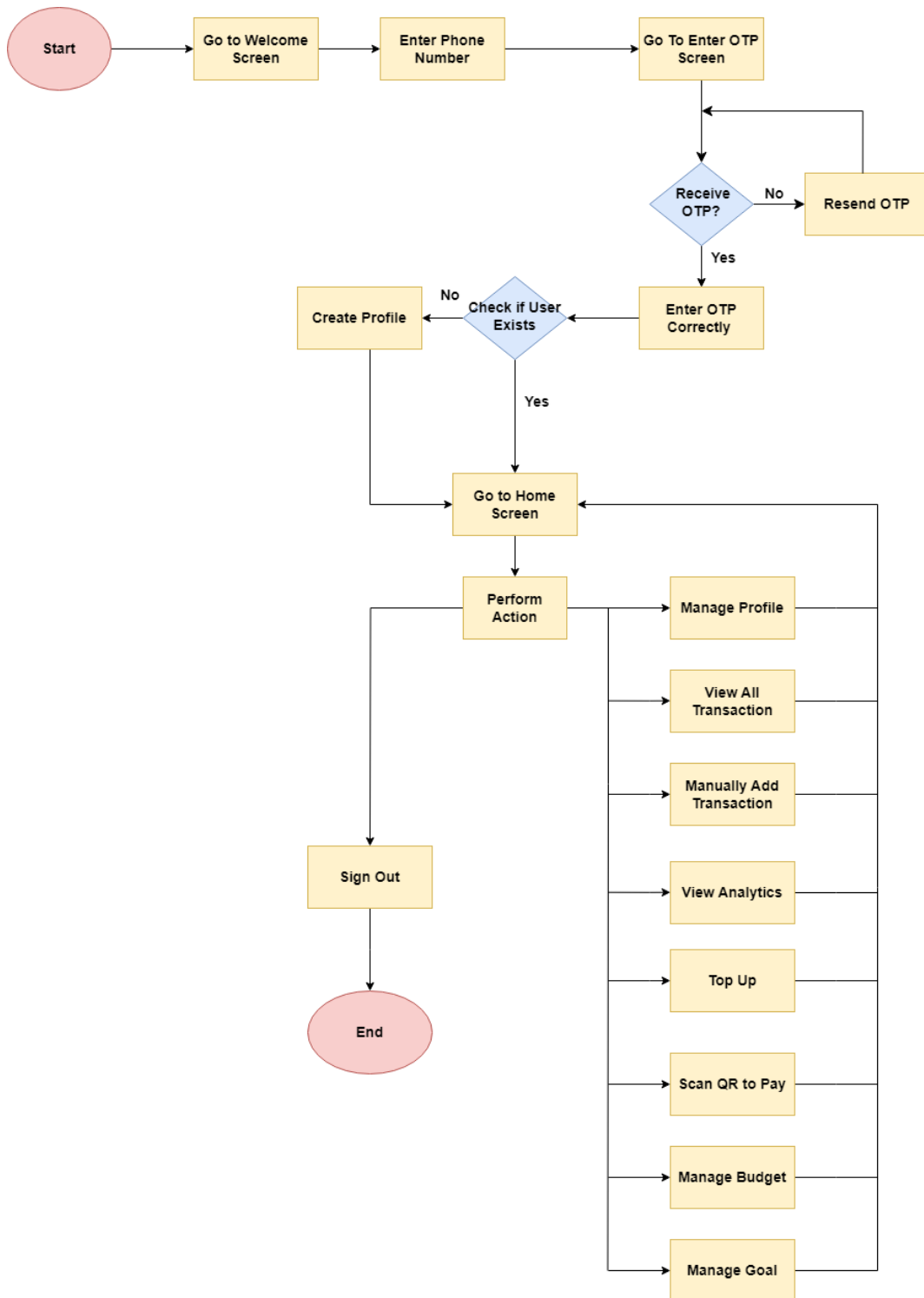


Figure 1.5.1 Flowchart of Personal Finance Management Application

1.6 Highlight of what have been archived

The project has achieved several significant milestones, delivering a robust personal finance management solution with a wide array of features that enhance the user experience. One of the key achievements is the User Authentication and Profile Management Module which ensures secure user registration, login, and profile management. This module enables users to create personalized accounts and securely manage their personal details. Another notable achievement is the development of the Virtual Wallet Module which offers users a seamless platform for conducting transactions. Through this module, users can make payments, scan QR codes, and top up their wallet balance. All transactions are automatically recorded eliminating the need for manual input and ensuring accurate financial tracking. The project also successfully implemented the Transaction Tracking and Categorization Module, providing users with a streamlined way to monitor their transactions. Whether through QR code payments or manual input, transactions capture essential details such as amounts, dates, and merchants, with the ability to categorize each transaction. This feature gives users a clear overview of their spending habits. The Income and Saving Recording Module has been integrated, enabling users to log various income sources such as salaries, bonuses, and investments. This feature offers users a complete financial overview, helping them track both expenses and income for better financial planning. A further achievement is the Manual Transaction Input Module which allows users to record transactions not captured through the virtual wallet such as cash payments. This feature ensures comprehensive tracking of all financial activity, giving users full control over their records. The development of the Budget Management and Calculation Module is another key success. Users can set and manage budgets for various timeframes with real-time tracking and automatic adjustments. This ensures that users stay within their financial limits and manage their spending effectively. The project also introduced the Goal-Setting and Tracking Module, which allows users to set and track financial goals such as saving for a major purchase or paying off debts. The progress tracking feature motivates users and helps them stay focused on their financial aspirations. The Analytics Module provides users with detailed visual insights into their spending patterns. Users can review their expenses and make data-driven financial decisions through charts and graphs to improve their budgeting strategies. Finally, the Receipt OCR Module was successfully developed, enabling users to upload receipts and extract relevant data such as item descriptions, prices, and dates. This feature reduces the need for manual entry to ensure accurate and efficient transaction recording.

1.7 Report Organization

The details of this research are shown in the following chapters. In Chapter 1, the problem statement, background and motivation are discussed. Moreover, the objectives are proposed to solve the problem statement. The project scope which includes the developed module will also be mentioned. In Chapter 2, the applications related to personal finance management are reviewed. Then, Chapter 3 discusses system design which includes the use case diagram, use case description as well as activity diagram. In addition, the project development along with the code is also explained in Chapter 3. Moreover, Chapter 4 represents the methodology, software and hardware tools used to develop the application. The timeline is also mentioned in the Gantt Chart. In addition, Chapter 5 discussed the overview of the implementation of the project. Furthermore, Chapter 6 reports the conclusion, project review, novelties and contribution. Chapter 6 also includes the future work of the project.

1.8 Summary

The definition and significance of personal finance management are addressed in the introduction. The outline of the personal finance management application is also briefly explained in the introduction. However, there are three major problems faced in traditional personal finance management applications. Thus, it is important to develop an application including modules such as user authentication and profile management module, virtual wallet module, transaction tracking and categorization module, income or saving recording module, manual transaction input module, budget management and calculation module, goal-setting and tracking module, analytics module as well as receipt OCR module for better financial control and decision-making.

Chapter 2 Literature Review

2.1 Overview

In this chapter, existing applications for personal finance applications are reviewed. Similar existing applications are reviewed to find out the Strengths and weaknesses of each application. It also includes a summary and a comparison of the existing application.

2.2 Existing Applications

2.2.1 Spendee

Spendee is a user-friendly expense tracking and budget management application that was co-founded by David and Jakub [2]. Spendee is available on both IOS and Android platforms. It is designed to empower users with tools to monitor their spending, set financial goals, and gain insights into their financial habits. Spendee offers features such as transaction tracking, budget creation, expense categorization, and synchronization with bank accounts and credit cards. Spendee originated as a side project within the app development agency Cleevio. In the early stages, they possessed extensive knowledge in app development but lacked experience in the sustained management of apps, continuous improvement, and other post-release activities. As a result, Spendee was eventually spun off from Cleevio and transformed into an independent company [3].

Spendee users can keep a monthly record of their financial transactions, tracking both their income sources and expenses. This gives them insights into their financial inflow and outflow. Users can choose to input transactions manually as shown in Figure 2.2.1.1 from [4] or sync Spendee with their bank accounts, e-wallet or crypto-wallet for automatic tracking as shown in Figure 2.2.1.2 from [4]. Manual input allows for more informed spending decisions. Spendee also utilizes the “Overview” section to gain a deeper understanding of users’ financial behaviour. Spendee provides a comprehensive list of users’ transactions, account balances, income versus expenses, and visual graphics that categorize where their money is spent each month as shown in Figure 2.2.1.3 from [4]. With the help of Spendee, users can plan their future expenses with confidence. Spendee assists them in staying within their budget by allowing them to customize budgets for different spending categories such as foods, sports and entertainment Figure 2.2.1.4 from [5]. Spendee employs machine learning algorithms to

automatically categorize expenses, reducing the need for manual categorization. Users can also set daily spending limits to help them stay within their budget on a day-to-day basis, reducing unnecessary expenses. There is also a shared wallet feature to manage not only users' finances but also their family's budget effectively. Users can set up alerts to notify them when they are nearing their budget limit and track their daily spending in real time. Users can allocate their money toward various savings goals, including long-term and short-term savings, emergency funds, and more. Moreover, there is an international expense tracking that allows users to keep track of expenses in different currencies, especially when travelling or doing business abroad. Spendee handles currency conversion for the users, ensuring accurate financial management. Spendee offers its users the capability to link their accounts with over 2,500 financial institutions worldwide [6].

However, users will have the choice of three different packages which are Spendee Basic, Spendee Plus, and Spendee Premium. Some features are offered for the specific package only. Spendee Basic is a free option, offering essential financial tracking features. Users can manually input their expenses, create budgets, and get a detailed overview of their finances. This package includes one complimentary budget and one cash wallet only. For users that looking for more flexibility, Spendee Plus is available at a cost of \$14.99 per year. It provides unlimited cash wallets and budgets as well as shared wallets with others, making it a less restrictive choice compared to the Basic package. For a comprehensive financial management experience, Spendee Premium is offered at a price of \$22.99 per year. This package includes automatic expense categorization, backup and sync features, unlimited cash wallets and budgets, the ability to share wallets with other Spendee users, transaction import and export capabilities, secure data storage, and a detailed overview of their expenses. Figure 2.2.1.5 from [7] illustrates the Spendee's packages and their features.

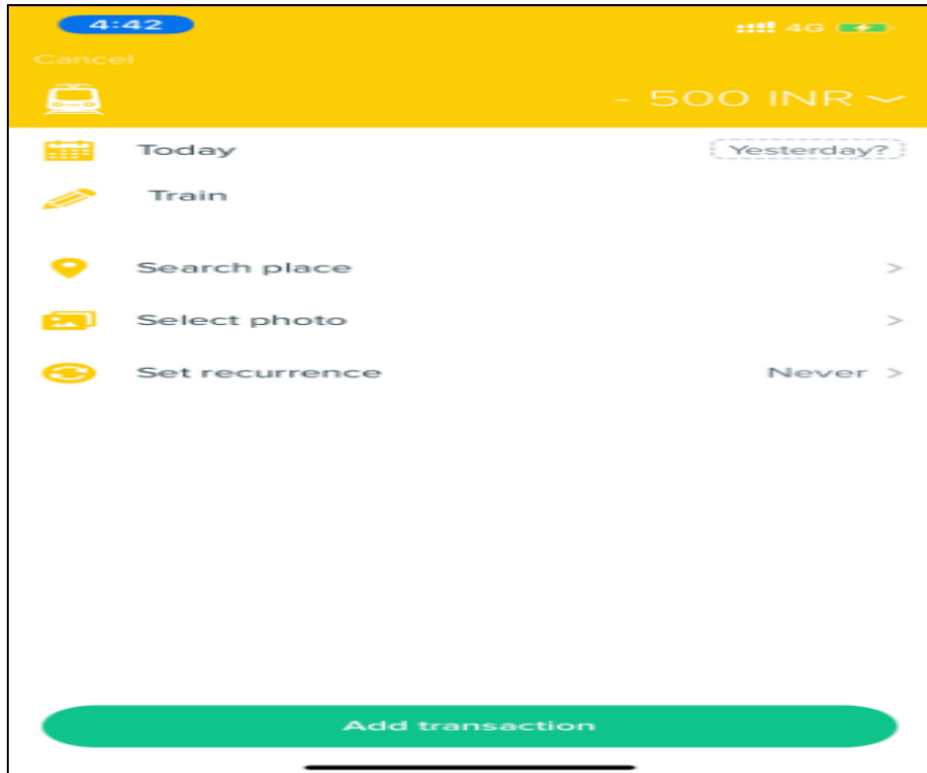


Figure 2.2.1.1 Spende's Add Transaction Feature [4]

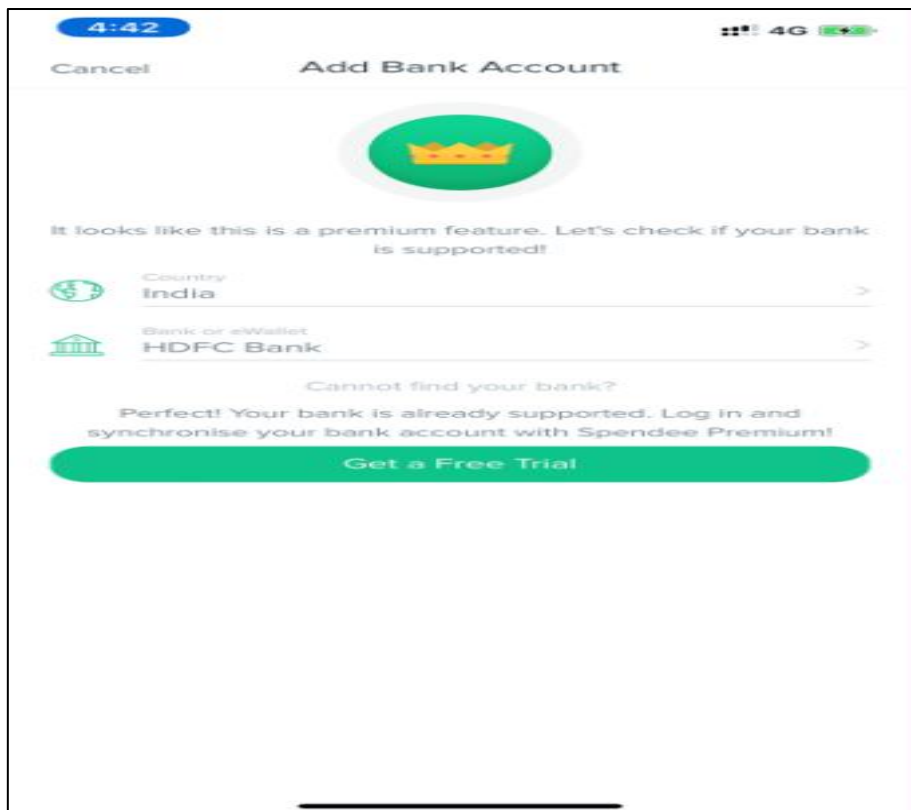


Figure 2.2.1.2 Spende's Syn with Bank Features [4]

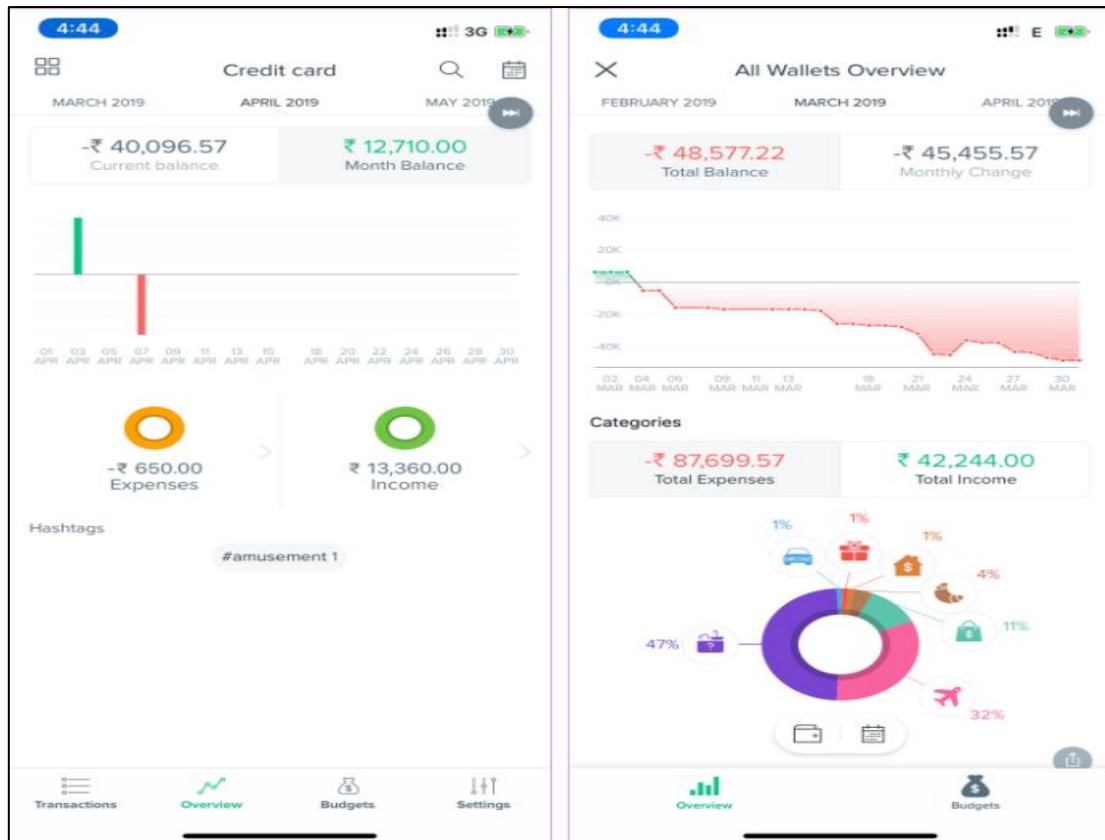


Figure 2.2.1.3 Spendee's Overview Section [4]

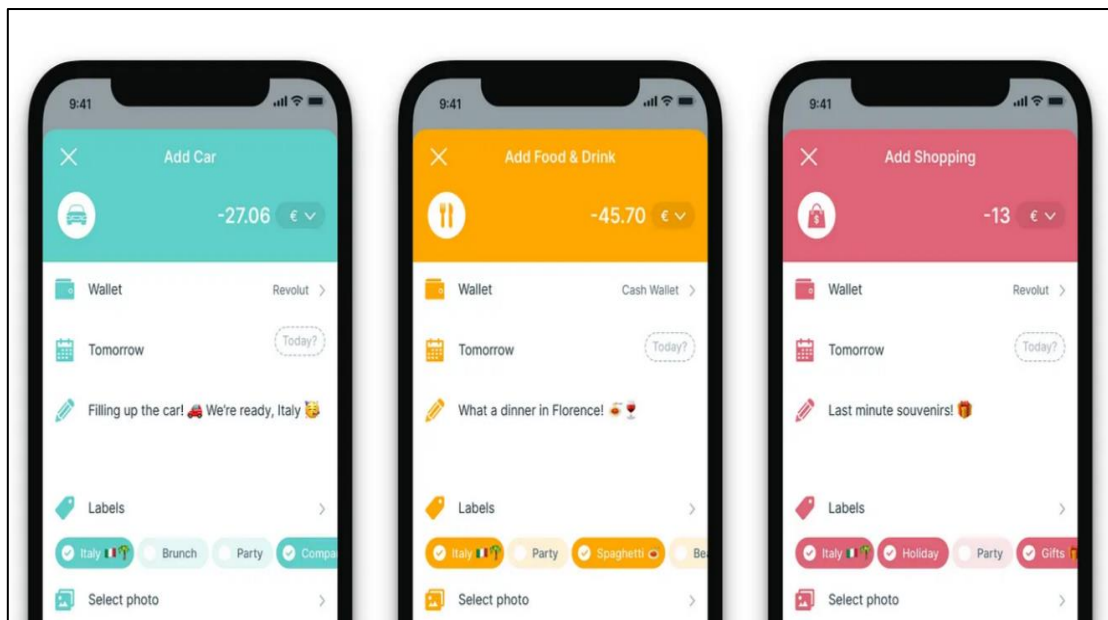


Figure 2.2.1.4 Spendee's Categorization Features [6]

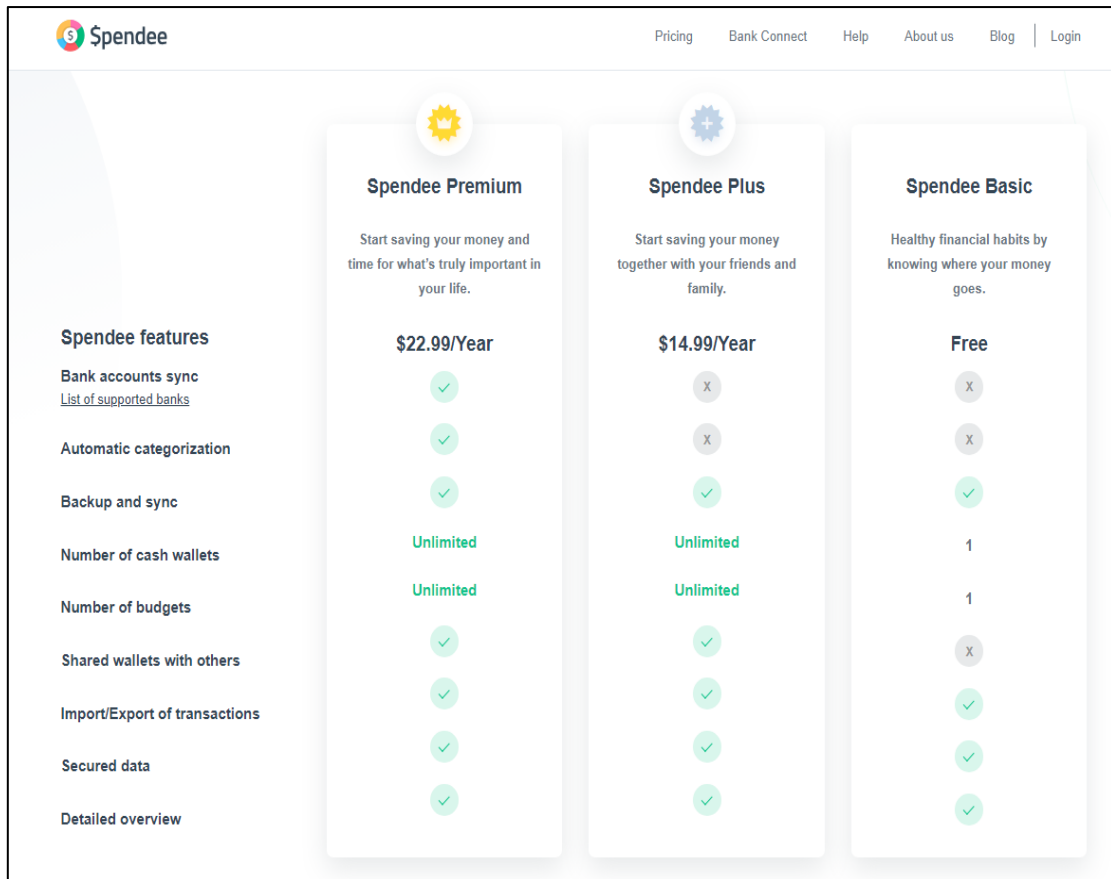


Figure 2.2.1.5 Spendeo’s Packages and Their Features [7]

Strengths

- Effortless management of all the financial information, including cryptocurrency, on a single device.
- It provides both automatic and manual spending categorization.
- It supports multiple currencies.
- A user-friendly interface featuring colourful infographic charts, making it easy to visualize and analyze the spending patterns.
- The ability to link multiple checking accounts, such as a family plan, ensuring clarity on expenses and tracking total expenditures.
- It provides shared wallets for dual budgeting.

Weaknesses

- The cost of the premium package can be a burden for some users.
- There are limited free features. Users have to pay to enjoy advanced features.
- There are no credit monitoring features in the application.
- Does not sync with all banks. This makes users have to manually input their transactions if their bank account is not supported by Spendee.

2.2.2 Mint

Mint is a personal budgeting application developed by Intuit, that offers both free and premium versions. Mint was introduced in 2006 and founded by Aaron Patzer. It stands out as one of the most popular budgeting apps currently available, providing users with a comprehensive platform for managing all their financial accounts in one place. The key features of Mint include bill payment tracking, subscription management, budgeting tools, access to free credit scores, alert notifications, and investment tracking. Mint has earned recognition on Forbes Advisors' list of the Best Budgeting Apps, making it a valuable option for individuals seeking budgeting assistance. Users appreciate Mint's intuitive interface, which includes features such as bill tracking, customizable budgets, and the ability to establish specific savings targets. However, some users have reported technical issues [8].

Mint offers a free version and a premium version called Mint Premium, priced at \$4.99 per month but it is only available exclusively on IOS devices. The premium version includes benefits such as the removal of in-app ads, subscription cancellation, and advanced data visualization. With the input of minimal information, such as online banking login details, Mint provides an overview of users' inflows and outflows as shown in Figure 2.2.2.1 from [9]. Mint allows user to view their linked account in one place such as cash, credit cards, investments and property as shown in Figure 2.2.2.2 from [9]. Mint will automatically assign categories and subcategories to imported transactions, which users can customize their preferences. Users can create multiple budgets for various spending categories, allowing for control over monthly expenditures in areas like entertainment, dining out, or personal care. Visual indicators, such as coloured bars and numbers, illustrate how close users are to reaching their budget limits. Budgets are flexible and adjustable, accommodating changes based on evolving spending patterns. Mint also allows users to manage bills by creating bill records that display due dates and amounts. Users can mark bills as paid and even set up similar records for offline payments, such as payments to Netflix and YouTube Premium. Mint offers bill negotiation services for select billers, aiming to reduce user's bill payments as shown in Figure 2.2.2.3 from [9]. Mint is that it does not provide online bill payment services so users can only track their bills through the app. While Mint provides access to user's credit scores as shown in Figure 2.2.2.4 from [10], but it offers minimal information on this topic. Users receive a single credit score which updates approximately twice a month. Mint also offers basic investment tracking features,

especially on its mobile apps, where it primarily displays transaction data from users' investment accounts.

However, some users may find the mobile app's interface a bit crowded due to its extensive set of features. While the browser-based version employs a navigation menu to address this issue, the mobile app's organization may require users to scroll extensively to discover all available features. For instance, setting the budget is more intuitive in the browser-based version but less so in the mobile app, where users need to access the Notifications icon to find this feature. Mint simplifies the tracking of your financial transactions, providing a list of recent transactions and their current balances once users have connected their online banking and financial accounts. While the connection process may encounter occasional difficulties, this is a common issue among financial apps that integrate with banks [10].

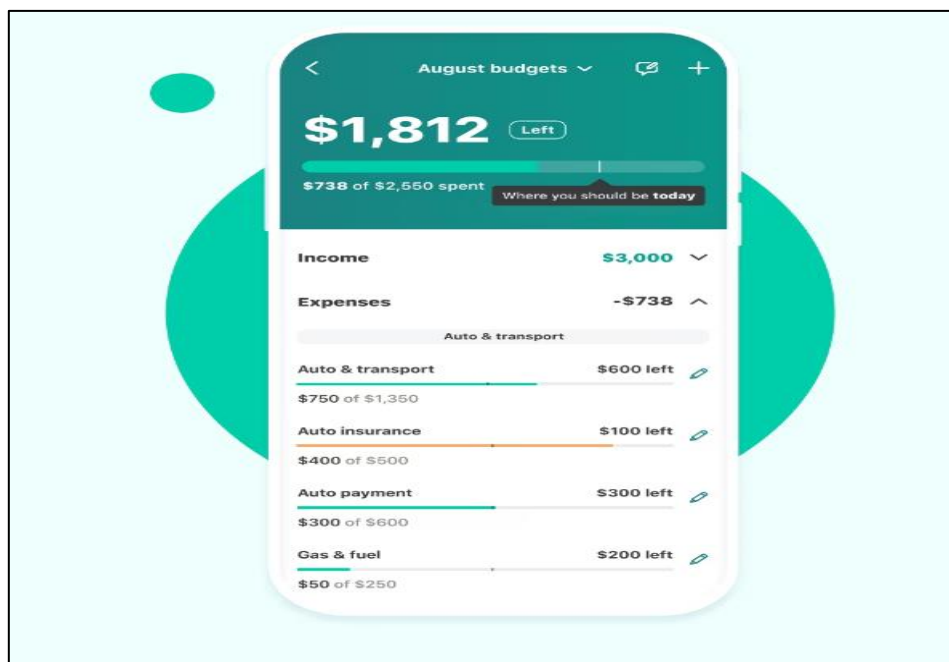


Figure 2.2.2.1 The Monthly Cashflow in Mint [9]

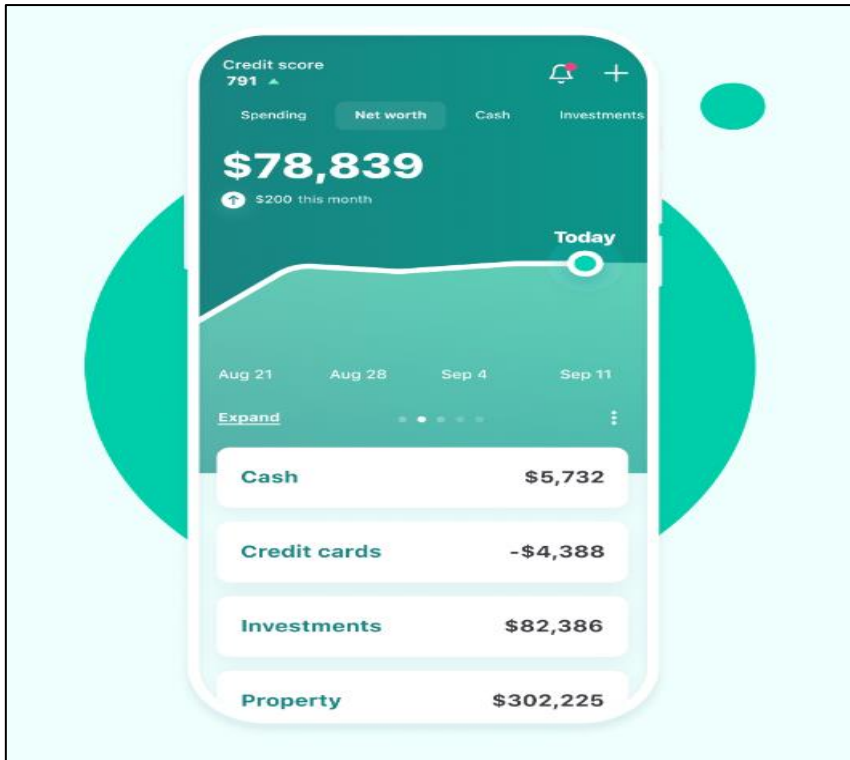


Figure 2.2.2.2 All Account in One Place Feature by Mint [9]

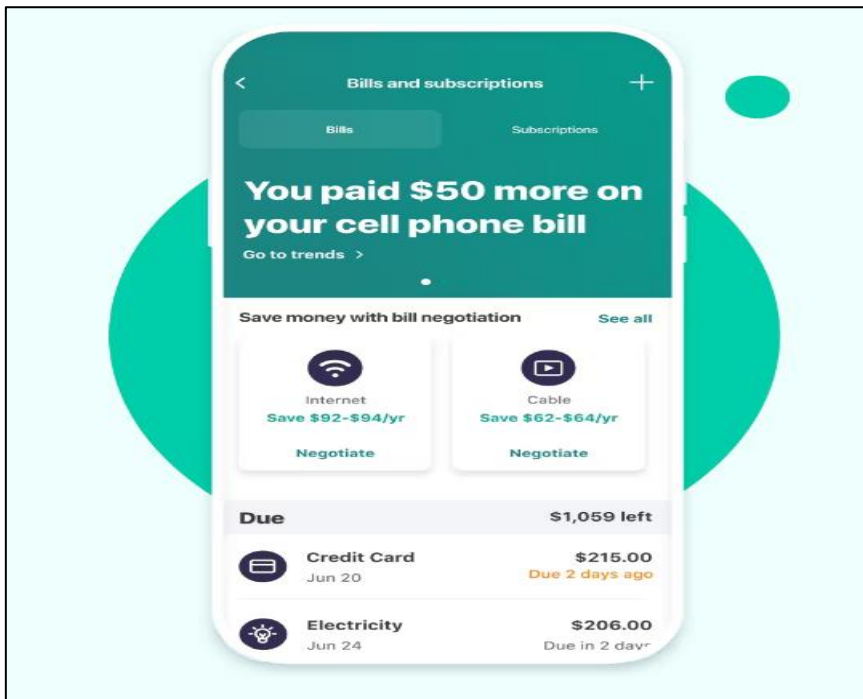


Figure 2.2.2.3 Mint's Bill Negotiation Feature [9]

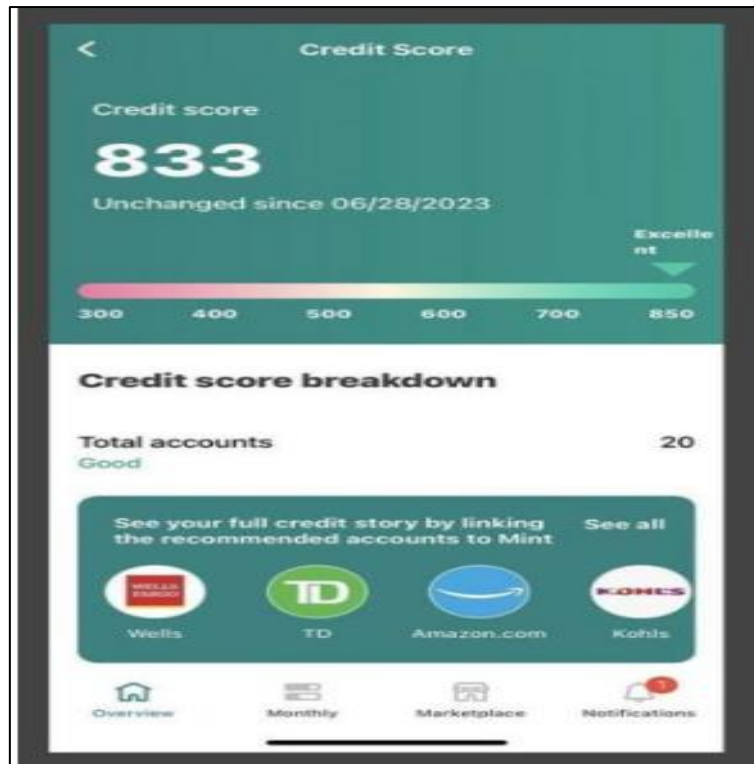


Figure 2.2.2.4 Mint's Credit Score Feature [10]

Strengths

- It includes the credit score feature.
- This application is free to download and use.
- Able to view all the financial accounts in one place.
- It provides good transaction tracking, trend graphs, and budgeting tools

Weaknesses

- There is an occasional bank connection problem.
- This application contains a lot of ads.
- It does not support joint accounts.

2.2.3 You Need a Budget (YNAB)

YNAB, short for "You Need a Budget," is an online financial application that presents a unique approach to personal budgeting. This application emphasizes the philosophy that every dollar users expect to earn should have a specific purpose, whether it's allocated for spending or saving. YNAB connects seamlessly with users' financial accounts to import balances and transactions directly into users' budgets, offering instant insights into their financial standing. Although YNAB may not be easy to master, it provides a wealth of educational resources, including tutorials and budgeting principles, to help users understand and manage their saving and spending habits more effectively. However, users have to pay for a monthly or annual subscription after a 34-day free trial [11].

After creating an account and logging in, users are greeted with a budget template featuring spending categories such as Bills, Frequent Expenses, and Goals. These categories can be customized, edited, or deleted as needed. Firstly, users can set a target for their spending categories. However, the onboarding tool might lack detailed explanations in this regard. As users progress, they can add connections to their financial accounts, either linked or unlinked. Linked accounts require users to provide login credentials, and YNAB supports multiple connection providers so users can add as many accounts as they want. Users can also manually enter data or import files for unlinked accounts. Notably, YNAB does not automatically categorize transactions, unlike some other personal finance services. Moreover, YNAB may offer auto-assign suggestions as shown in Figure 2.2.3.1 from [11], but users can make their own decisions, and an Undo option is available. Once onboarding is complete, users must continue by adding all their financial accounts, ensuring an accurate reflection of their financial status. Users can build a budget that begins with assessing their income, which they will assign to various spending categories and savings goals. YNAB encourages users to allocate every dollar, promoting responsible spending and saving practices. The budget template resembles a spreadsheet with columns for categories, assigned amounts, activity, and spending in the current funding period. Users must ensure that every dollar they allocate to spending categories aligns with their budgeted amounts. Throughout the month, they can monitor their progress and make adjustments as needed.

To gauge progress, YNAB employs colour coding as shown in Figure 2.2.3.2 from [11]. Green signifies categories that have been funded, yellow indicates categories that require further attention, and red signifies significant issues that need resolution. YNAB emphasizes setting new budgets every month, allowing for evolving financial plans. Two valuable features include Recent Moves, which functions as an audit trail, and Progress Bars on the Budget screen, offering a visual overview of category status. YNAB also provided a Loan Payoff Simulator that functions to show the effect on the loan as shown in Figure 2.2.3.3 from [11]. For instance, if the user wishes to make an extra payment on their car loan beyond the regular monthly instalment, they can do so. YNAB's Loan Payoff Simulator provides a clear illustration of how this additional payment will impact their loan. Users can access a net worth report and an income-versus-expenses table. These reports are customizable based on date range, category, and account preferences [11].

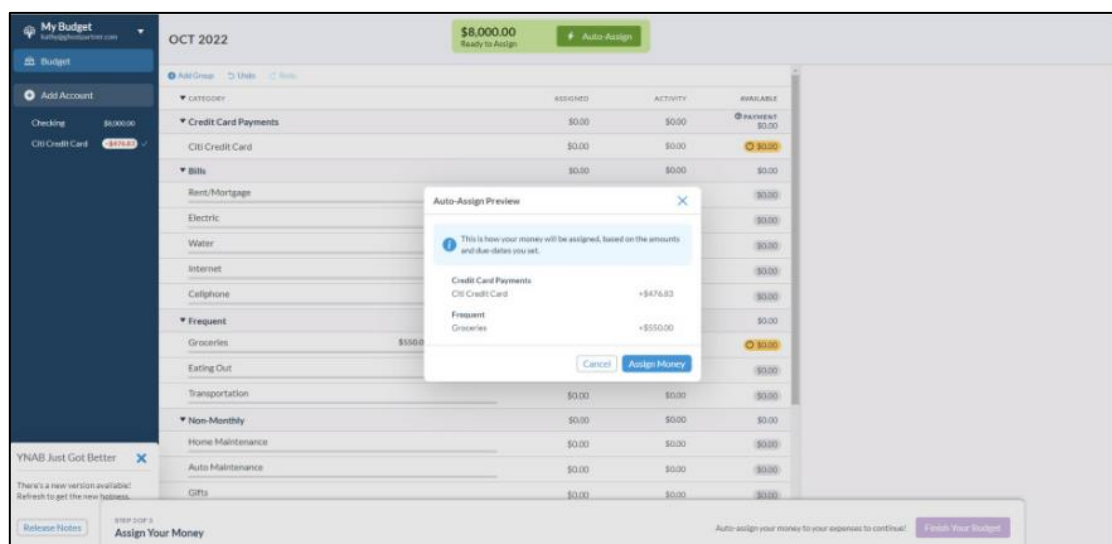


Figure 2.2.3.1 YNAB's Auto-Assigns Funds [11]

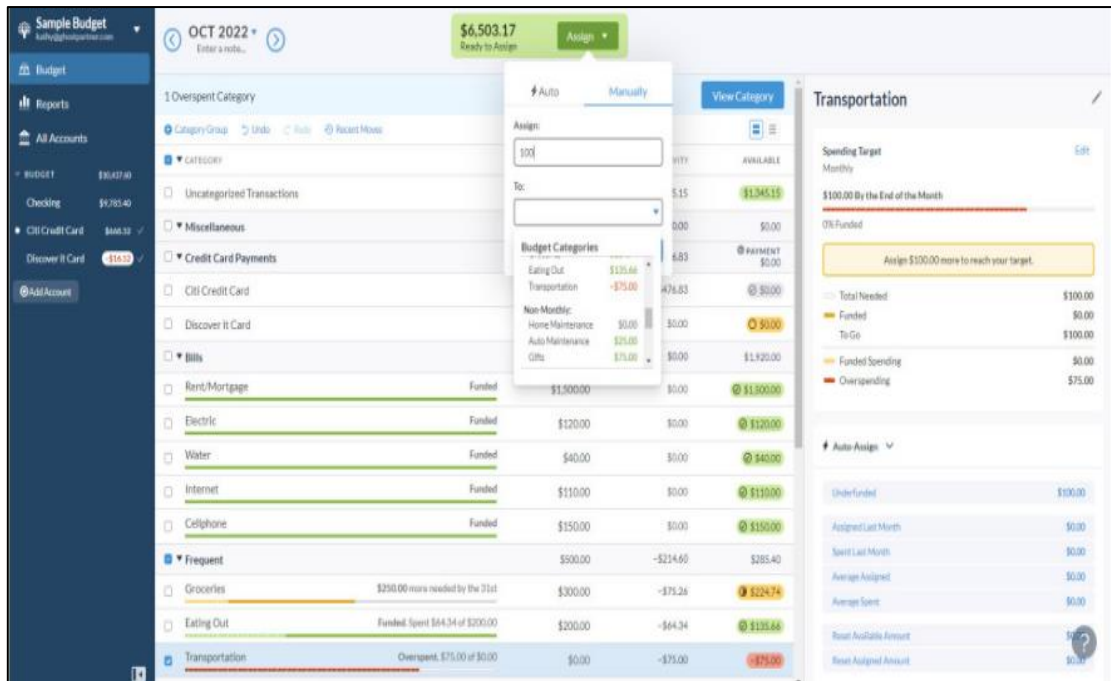


Figure 2.2.3.2 YNAB’s Color-coding to Indicate Each Category Status [11]

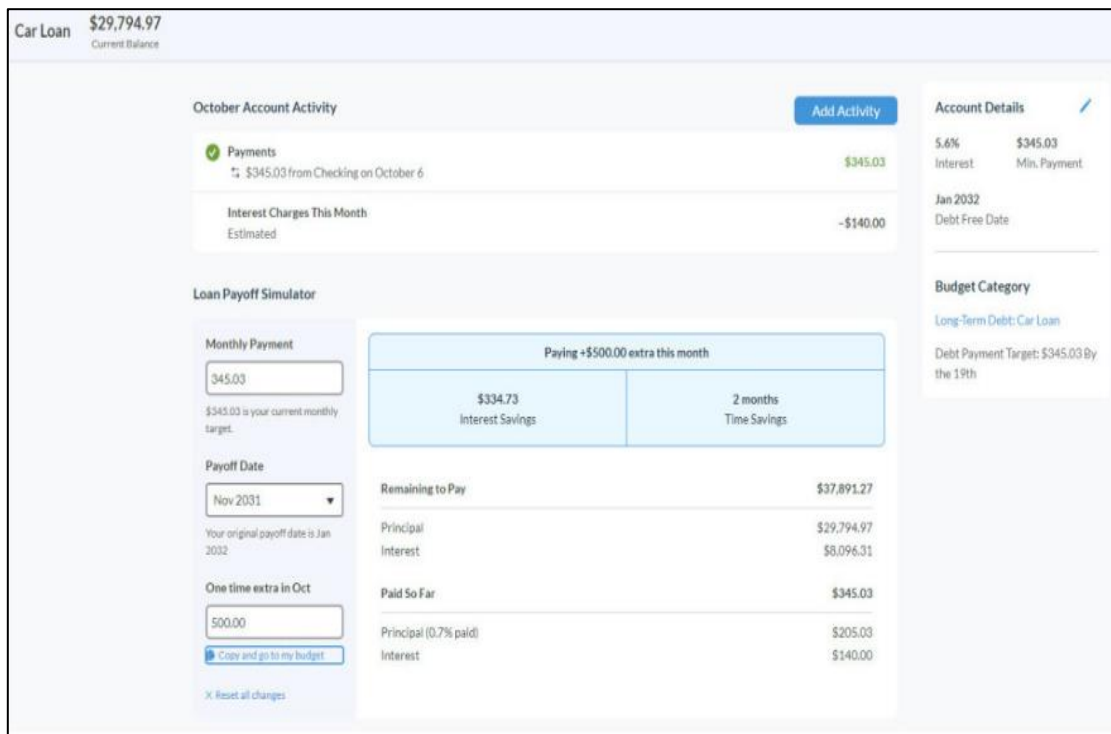


Figure 2.2.3.3 YNAB’s Loan Payoff Simulator and The Effect on The Loan [11]

Strengths

- It allows manual account linking which users can choose to link or unlink their account.
- Users can have multiple bank connection options.
- It provides voluminous educational material and tutorials.
- No ads to enhance user experience.

Weaknesses

- It takes time to understand and use.
- No bill tracking features.
- Users need to pay to use the application since it has no free version.

2.2.4 Goodbudget

Goodbudget is a user-friendly expense tracking and budget management application designed to assist users in their budget management. It employs the "envelope method" philosophy of personal finance, where users allocate specific amounts of money to different virtual envelopes, each designated for a particular purpose. This approach ensures that individuals are less likely to overspend unintentionally, as exceeding a budget would require adding more funds to the respective envelope. In addition to its practicality, the envelope method has a psychological component. In today's digital age, many people manage their budgets electronically, which can sometimes lead to a disconnect between the money and the spender. The envelope method helps individuals visualize and track their spending more effectively [12].

Goodbudget transforms the conventional envelope budgeting method into a user-friendly mobile app, introducing virtual envelopes for a digital budgeting experience. Users can allocate predetermined amounts of money to these digital envelopes for various spending categories they choose as shown in Figure 2.2.4.1 from [13]. As expenditures occur, the allocated funds in each envelope decrease, enabling users to monitor their spending accurately. Goodbudget also offers the flexibility to create additional envelopes for specific financial goals, whether it is paying down debt or savings. Users can input their transactions manually as shown in Figure 2.2.4.2 from [13] or import a summary of recent transactions from their bank, typically available through the bank's online portal. If necessary, users can adjust and customize their envelopes to better align with their financial goals. Goodbudget provides visual cues, such as green or red bars, to indicate whether the budget is on track, making it easy for users to assess their financial situation at a glance. Users can also view the income versus spending report. Figure 2.2.4.3 from [13] illustrates the reports tab in Goodbudget.

Goodbudget offers two versions which are a free and a "Plus" version. The free version allows users to create up to ten envelopes, link a single account across up to two devices, and store spending history for up to one year. While the "Plus" version, available for \$5 per month or \$45 per year, provides more extensive features. Users have the flexibility to create an unlimited quantity of envelopes, utilize the app on as many as five devices, connect an unlimited number of accounts, and retain spending history for up to five years. "Plus" version's users also enjoy comprehensive customer support. While the free version suits individuals with straightforward financial needs and a limited number of savings goals, those seeking a more extensive selection

of spending categories may find the “Plus” version more suitable for their budgeting requirements. Figure 2.2.4.4 illustrates the differences between the free version and the “Plus” version from [14].

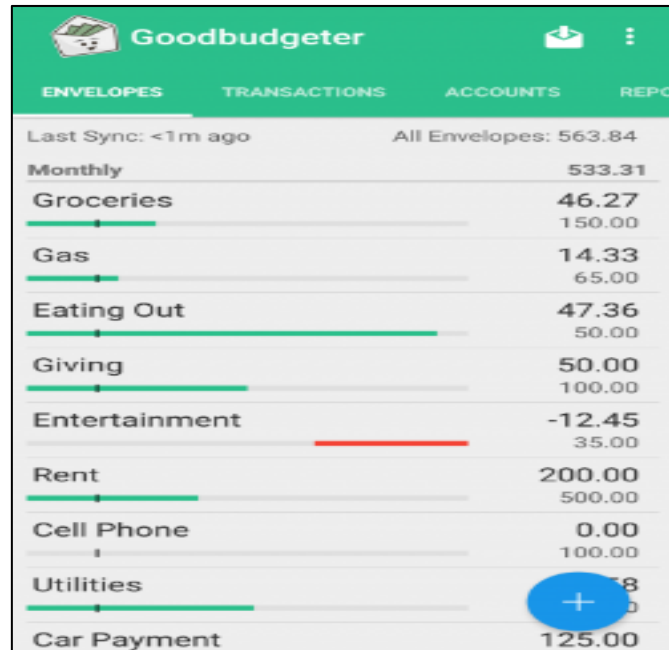


Figure 2.2.4.1 The Envelopes Tab in Goodbudget [13]

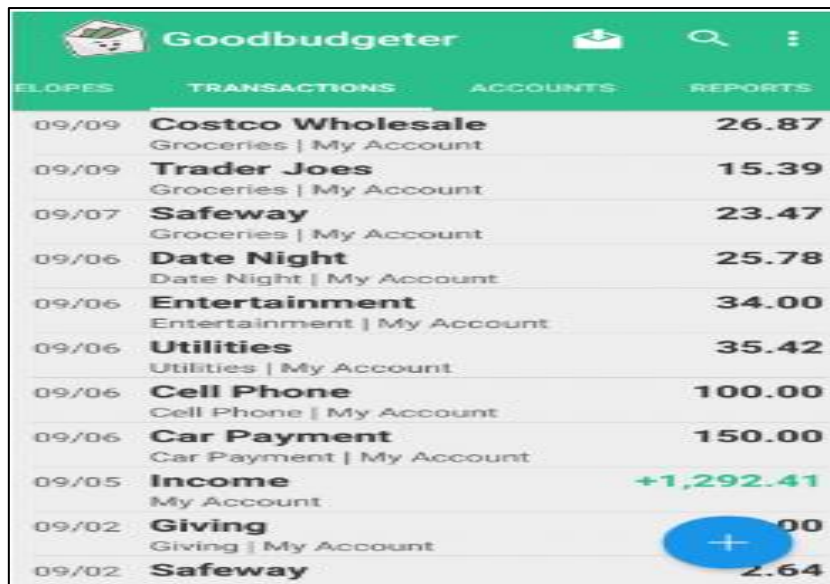


Figure 2.2.4.2 The Transaction Tab in Goodbudget [13]

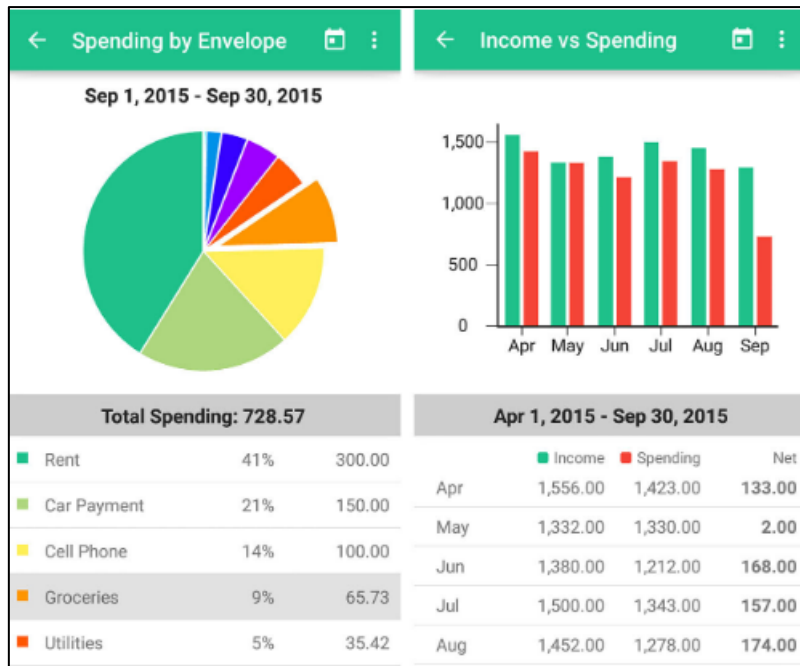


Figure 2.2.4.3 The Reports Tab in Goodbudget [13]

Goodbudget	Goodbudget Plus
20 Envelopes	Unlimited Envelopes
1 Account	Unlimited Accounts
2 Devices	5 Devices
1 Year of Transaction Tracking	7 Years of Transaction Tracking
Community Support	Email Support

Figure 2.2.4.4 The Differences Between Goodbudget Free Version and Goodbudget “Plus” Version [14]

Strengths

- A free version and a paid version are available for users.
- Enables multiple users within a household to synchronize their applications, allowing them to collectively track and manage their budgets in real-time.
- Users can import bank transaction files.
- It provides reports to analyze income and spending trends

Weaknesses

- The app does not have a direct bank account syncing feature, users need to manually record every transaction.
- It does not provide investing resources or tools.
- Some features are limited to the paid version.

2.3 Comparison of Existing Applications

This section includes the comparison of existing applications which are Spendee, Mint YNAB and Goodbudget. Table 2.3.1 illustrates the comparison in various aspects.

Table 2.3.1 Comparison of Existing Applications

Feature Aspect	Spendee	Mint	YNAB	Goodbudget
Pricing	Free with premium options	Free	Subscription-based	Free with premium option
Budgeting	Yes	Yes	Yes	Yes
Expense Tracking	Yes	Yes	Yes	Yes
Income Tracking	Yes	Yes	Yes	Yes
Bank Account Sync	Yes	Yes	Yes	No (Manual Entry)
Investment Tracking	Limited	Yes	Limited	No
Debt Tracking	Limited	Yes	Yes	No
Goal Setting	Yes	Yes	Yes	Yes
Bill Payment Reminders	No	Yes	No	No
Mobile Apps	iOS, Android	iOS, Android	iOS, Android	iOS, Android
Web Version	Yes	Yes	Yes	Yes
User-Friendly Interface	Yes	Yes	Yes	Yes

Multi-Currency Support	Yes	Yes	Yes	Yes
-------------------------------	-----	-----	-----	-----

2.4 Limitations of Previous Studies

Previous research has often not sufficiently investigated the depth and breadth of expense-tracking capabilities within personal finance applications. This includes real-time tracking, advanced forecasting, and tools to prevent overbudgeting, all of which are crucial for effective financial management. Secondly, the precision and accuracy of transaction categorization have not been extensively explored in previous studies. While some studies mention categorization, there is a lack of in-depth analysis of how well these systems categorize transactions, especially in complex financial scenarios. Thirdly, previous research has not adequately emphasized the role of personal finance applications in enhancing users' financial awareness. Additionally, there is a limited focus on tools that assist users in preparing for future financial risks, such as unexpected expenses or economic downturns.

2.5 Proposed Solutions

This project aims to propose an advanced expense-tracking system that provides users with real-time insights into their spending patterns. This system will include features such as real-time transaction tracking for enhanced budget control, customizable alerts to notify users of potential overspending and Intelligent forecasting tools to prevent overbudgeting and facilitate better financial planning. Secondly, integrate artificial intelligence and machine learning algorithms to enhance transaction categorization accuracy and efficiency. An AI-driven algorithm that automatically and accurately categorizes transactions based on context to reduce the reliance on manual transaction categorization, making financial management more seamless for users. It also aims to provide users with the ability to customize categorization rules to suit their unique financial situations. Lastly, introduce a comprehensive goal-setting feature within the application to equip users with an advanced suite of tools to enhance their financial awareness, and facilitate strategic risk planning. The proposed solution includes the implementation of a robust goal-setting feature that empowers users to set, track, and manage financial goals across various aspects of their financial lives. Users will be prompted to establish specific financial objectives that align with their aspirations, such as building an

emergency fund, paying off debt, saving for a major purchase, or investing in their retirement. Users will have the ability to visualize their progress towards achieving their set financial goals. In this chapter, the existing applications including Spendee, Mint YNAB and Goodbudget are reviewed and their strengths and weaknesses are identified. Table 2.3.1 illustrates the comparison in various aspects.

2.6 Summary

In this chapter, the existing applications including Spendee, Mint YNAB and Goodbudget are reviewed and their strengths and weaknesses have been identified. The comparison of these existing applications in various aspects is also illustrated by a table. Moreover, the limitation of previous studies is identified and the proposed solution is explained in this chapter.

Chapter 3 System Design

3.1 Overview

The chapter will discuss the overview of the system by using the use case diagram and descriptions. In addition, the chapter will briefly explain the application development through the project's code.

3.2. Use-Cases Diagram

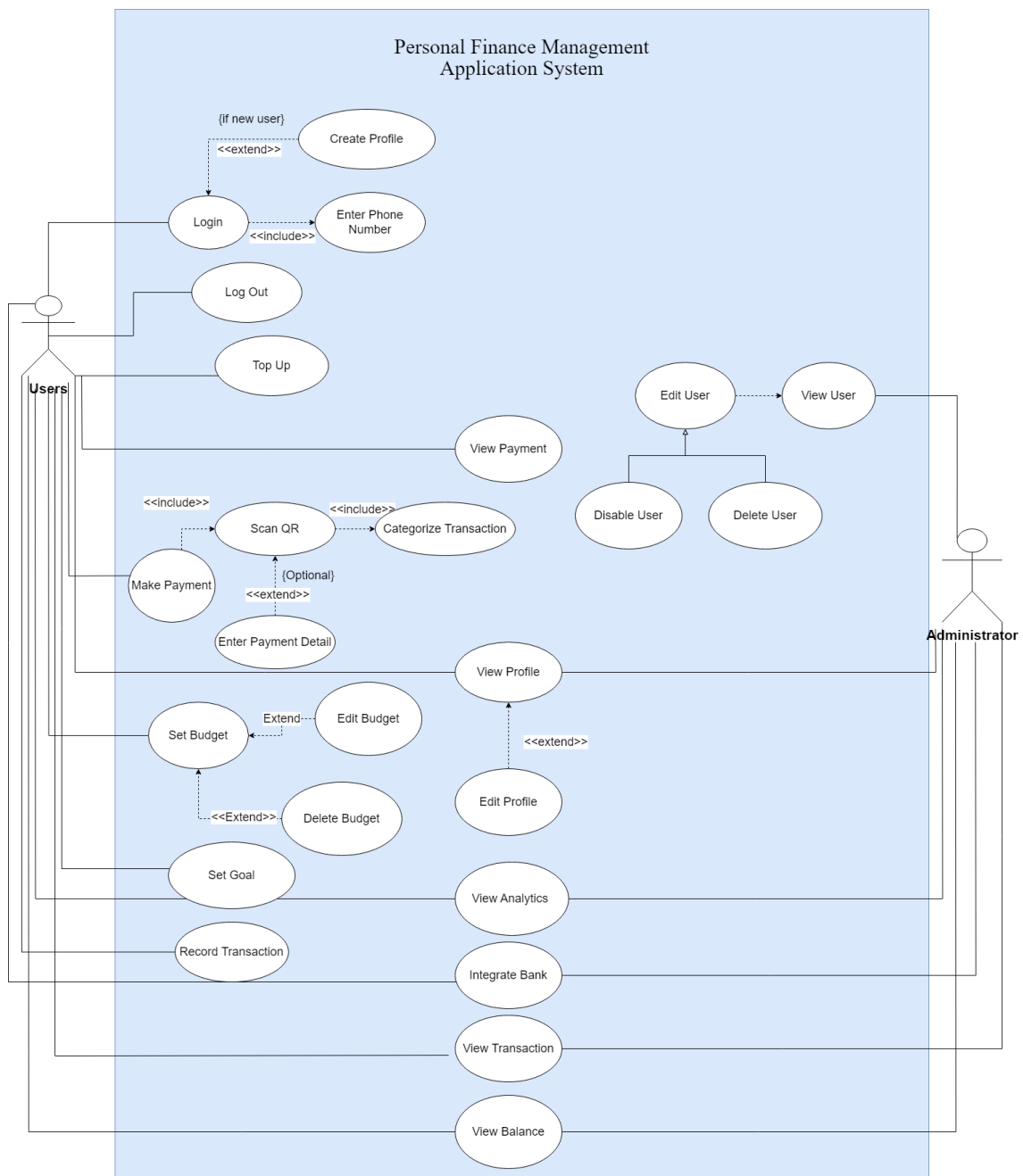


Figure 3.2.1 Use Case Diagram for Personal Finance Management Application System

3.3 Use Case Description

Table 3.3.1 Login Use Case Description

Use Case ID	00001
Use Case Name	Login
Brief Description	User is allowed to access the application.
Actor	User
Trigger	The user navigates to the login page of the application.
Precondition	The user owns a valid phone number to receive OTP.
Normal flow of events	<ol style="list-style-type: none"> 1. The user enters their phone number. 2. The system sends an OTP to the user's phone. 3. The system validates whether the OTP entered by the user is correct. 4. The system checks if the user exists in the system. 5. The system will direct the user to the home screen.
Sub flows	
Alternate flow	4a. If the user does not exist in the system, they are prompted to create a profile by entering their name and email.

Table 3.3.2 Log Out Use Case Description

Use Case ID	00002
Use Case Name	Log out
Brief Description	The user is able to log out of the application.
Actor	User
Trigger	The user decides to log out of the application.
Precondition	The user must be logged in to the application.
Normal flow of events	<p>Logout via Interface</p> <ol style="list-style-type: none"> 1. The user clicks on the logout icon located in the upper right corner of the application interface.

	<p>2. The user will logout and navigate to the welcome screen.</p> <p>Logout via Profile Screen</p> <p>1. The user navigates to the profile screen.</p> <p>2. The user clicks on the logout button located on the profile screen.</p> <p>3. The user will logout and navigate to the welcome screen</p>
Sub flows	-
Alternate flow	-

Table 3.3.3 Create Profile Use Case Description

Use Case ID	00003
Use Case Name	Create Profile
Brief Description	A new user creates a profile in the application
Actor	User
Trigger	The user navigates to create a new profile screen.
Precondition	The user does not have an existing profile in the system.
Normal flow of events	<p>1. The system prompts the user to enter their personal information.</p> <p>2. The user provides the required details, which are name and email.</p> <p>3. The system validates the entered information.</p> <p>4. The system creates a new user profile.</p> <p>5. The user will be directed to home screen.</p>
Sub flows	
Alternate flow	

Table 3.3.4 View Profile Use Case Description

Use Case ID	00004
Use Case Name	View Profile
Brief Description	User views their account profile information
Actor	User
Trigger	The users intend to view their profile information.
Precondition	The user is logged into their account.
Normal flow of events	<ol style="list-style-type: none"> 1. The user navigates to the profile management screen. 2. The system displays the user's details such as name and email. 3. The user can review their information.
Sub flows	-
Alternate flow	3a. The user can select to edit their profile if desired.

Table 3.3.5 Edit Profile Use Case Description

Use Case ID	00005
Use Case Name	Edit Profile
Brief Description	User modifies their account profile information.
Actor	User
Trigger	The users intend to edit their profile information.
Precondition	The user is logged into their account.
Normal flow of events	<ol style="list-style-type: none"> 1. The user navigates to the profile management screen. 2. The system displays the user's details such as name and email. 3. The user clicks on the “Edit Profile”. 4. The system displays editable fields of the user's profile. 5. The user makes desired changes to their information such as name and email.

	6. The system saves the updated profile information.
Sub flows	-
Alternate flow	-

Table 3.3.6 Top Up Use Case Description

Use Case ID	00006
Use Case Name	Top Up
Brief Description	Users can add funds to their virtual wallets.
Actor	User
Trigger	The user intends to add funds to their virtual wallet.
Precondition	The user must have a valid payment method.
Normal flow of events	<ol style="list-style-type: none"> 1. The user clicks on the "Top Up" button within the application. 2. The user enters the desired amount to top up and confirms the transaction. 3. The system initiates the payment process. 4. The user is prompted to enter payment information such as credit card details to complete the transaction. 5. The system updates the user's account balance
Sub flows	-
Alternate flow	-

Table 3.3.7 Make Payment Use Case Description

Use Case ID	00007
Use Case Name	Make Payment
Brief Description	Users are able to make payments within the application.
Actor	User

Trigger	The user intends to make a payment.
Precondition	The user must have a sufficient balance in their virtual wallet.
Normal flow of events	<ol style="list-style-type: none"> 1. The user clicks on the "Scan" button on the homepage. 2. The user scans the QR code associated with the payment. 3. The user enters the payment details, which include the amount, and payment details (optional), selects a category and adds to the existing budget (optional). 4. The user confirms the payment. 5. The system processes the transaction. 6. The system updates the user's balance
Sub flows	-
Alternate flow	4a. The system will display an error message indicating insufficient balance if the user does not have enough balance.

Table 3.3.8 Categorize Transaction Use Case Description

Use Case ID	00008
Use Case Name	Categorize Transaction
Brief Description	The user assigns a category to a transaction, either through QR payment or manual entry.
Actor	User
Trigger	The user selects a transaction to categorize.
Precondition	The user must log in to the application
Normal flow of events	<ol style="list-style-type: none"> 1. The user conducts a transaction. 2. The system prompts the user to fill in the transaction details, including an option to assign a category.

	<ol style="list-style-type: none"> 3. The user selects the appropriate category from the list of predefined categories. 4. The system saves the transaction with the selected category. 5. The transaction is categorized and recorded in the transaction history.
Sub flows	<p>QR Payment:</p> <ol style="list-style-type: none"> 1a. The user initiates a payment by scanning a QR code (via the Make Payment use case). 1b. The system retrieves payment details from the QR code. <p>Manual Entry:</p> <ol style="list-style-type: none"> 1a. The user clicks on the floating blue button at the bottom right to manually add a transaction. 1b. The user selects "Expense"
Alternate flow	-

Table 3.3.9 Manual Input Transaction Use Case Description

Use Case ID	00009
Use Case Name	Manual Input Transaction
Brief Description	User manually enters a financial transaction.
Actor	User
Trigger	The user intends to manually input a transaction.
Precondition	The user must log in to the application
Normal flow of events	<ol style="list-style-type: none"> 1. The user clicks on the floating blue button at the bottom right to manually add a transaction. 2. The user selects whether the transaction is an "Expense" or "Income / Savings".

	<ol style="list-style-type: none"> 3. The system prompts the user to fill in the transaction details. 4. The user confirms and saves the transaction details.
Sub flows	<p>Expense:</p> <ol style="list-style-type: none"> 2a. The user selects a category of the transaction. 2b. The user can select whether to add the transaction to an existing budget or not. <p>Income/Savings:</p> <ol style="list-style-type: none"> 2a. The user can select whether to add the transaction to an existing budget or not.
Alternate flow	-

Table 3.3.10 Set Budget Use Case Description

Use Case ID	00010
Use Case Name	Set Budget
Brief Description	Users are able to set a budget within the application.
Actor	User
Trigger	The user decides to establish a budget.
Precondition	The user is logged into their account
Normal flow of events	<ol style="list-style-type: none"> 1. The user navigates to the budget management screen of the application. 2. The user clicks the “+” symbol at the right corner top right corner of the screen which is next to the word "Budget" or clicks the “+” symbol which is centered in a grey rectangular area. 3. The user enters the budget details such as title, amount, category and duration of the budget 4. The user confirms and saves the budget settings. 5. The system validates the entered information.

	6. The system confirms successful budget creation and adds the budget to the screen.
Sub flows	-
Alternate flow	5a. If the entered information is invalid, the system prompts the user to correct it.

Table 3.3.11 Edit Budget Use Case Description

Use Case ID	00011
Use Case Name	Edit Budget
Brief Description	User modifies an existing budget.
Actor	User
Trigger	The user decides to edit a budget.
Precondition	The user has at least one budget set up.
Normal flow of events	<ol style="list-style-type: none"> 1. The user navigates to the budget management screen of the application. 2. The system displays the current budget details. 3. The user clicks the edit icon next to the budget title. 4. The user modifies the budget title, amount, category, or period as needed. 5. The user confirms and updates the budget settings. 6. The system validates the changes. 7. The system saves the updated budget information.
Sub flows	-
Alternate flow	6a. If the changes are invalid, the system prompts the user to correct them.

Table 3.3.12 Duplicate Budget Use Case Description

Use Case ID	00012
Use Case Name	Duplicate Budget
Brief Description	User creates a copy of an existing budget.
Actor	User
Trigger	The user decides to duplicate a budget.
Precondition	The user has at least one budget set up.
Normal flow of events	<ol style="list-style-type: none"> 1. The user navigates to the budget management screen of the application. 2. The system displays the current budget details. 3. The user clicks the duplicate icon next to the budget title. 4. The system duplicates the budget information and the spend amount and start date will be reset.
Sub flows	-
Alternate flow	-

Table 3.3.13 Download Budget Use Case Description

Use Case ID	00013
Use Case Name	Download Budget
Brief Description	User downloads their budget information.
Actor	User
Trigger	The user decides to download the budget information.
Precondition	The user has at least one budget set up.
Normal flow of events	<ol style="list-style-type: none"> 1. The user navigates to the budget management screen of the application. 2. The system displays the current budget details. 3. The user clicks on the budget they wish to download.

	<ol style="list-style-type: none"> 4. The system retrieves the user's budget information. 5. The user clicks the download icon at the top right corner which is next to the budget title. 6. The system generates a PDF containing budget details. 7. The system initiates the file download to the user's device. 8. The system confirms successful download.
Sub flows	-
Alternate flow	-

Table 3.3.14 Delete Budget Use Case Description

Use Case ID	00014
Use Case Name	Delete Budget
Brief Description	User removes an existing budget.
Actor	User
Trigger	The user decides to delete an existing budget.
Precondition	The user has at least one budget set up.
Normal flow of events	<ol style="list-style-type: none"> 1. The user navigates to the budget management screen of the application. 2. The system displays the current budget details. 3. The user clicks on the delete icon on the budget they wish to delete. 4. The system removes the budget from the user's account.
Sub flows	-
Alternate flow	-

Table 3.3.15 Add Transaction to Budget Use Case Description

Use Case ID	00015
Use Case Name	Add Transaction to Budget
Brief Description	The user adds a transaction to the budget.
Actor	User
Trigger	The user selects a transaction to add to an existing budget.
Precondition	The user must have at least one transaction and a budget.
Normal flow of events	<ol style="list-style-type: none"> 1. The user navigates to the home screen. 2. The system displays the available transactions or allows the user to create a new transaction. 3. The user selects or creates a transaction and chooses to add it to an existing budget. 4. The system updates the budget's progress with the transaction amount. 5. The transaction is recorded as part of the selected budget.
Sub flows	<p>QR Payment:</p> <ol style="list-style-type: none"> 3a. The user initiates a payment by scanning a QR code. 3b. The system retrieves payment details from the QR code. 3c. During the transaction details entry step, the user selects an existing budget from the list. <p>Manual Entry:</p> <ol style="list-style-type: none"> 3a. The user manually adds a transaction by clicking the floating blue button on the screen. 3b. The system prompts the user to fill in the transaction details and select it as an "Expense". 3c. The user selects a budget from the lists. <p>Adding an Existing Transaction to a Budget:</p>

	<p>3a. The user clicks on an existing “Expense” transaction from the transaction history.</p> <p>3b. The system shows the transaction details.</p> <p>3c. If the transaction has not yet been added to a budget, the user is given the option to add it to a budget.</p>
Alternate flow	-

Table 3.3.16 Set Goal Use Case Description

Use Case ID	00016
Use Case Name	Set Goal
Brief Description	This use case enables users to set financial goals within the application.
Actor	User
Trigger	The user decides to establish a financial goal.
Precondition	-
Normal flow of events	<ol style="list-style-type: none"> 1. The user navigates to the goal management screen of the application. 2. The user clicks the “+” symbol at the right corner top right corner of the screen which is next to the word "Goals" or clicks the “+” symbol which is centered in a grey rectangular area. 3. The user enters the goal details such as title, amount and duration of the goal. 4. The user confirms and saves the goal settings. 5. The system validates the entered information. 6. The system confirms successful goal creation adds the budget to the screen.
Sub flows	-
Alternate flow	5a. If the entered information is invalid, the system prompts the user to correct it.

Table 3.3.17 Edit Goal Use Case Description

Use Case ID	00017
Use Case Name	Edit Goal
Brief Description	User modifies an existing goal.
Actor	User
Trigger	The user decides to edit a goal.
Precondition	The user has at least one goal set up.
Normal flow of events	<ol style="list-style-type: none"> 1. The user navigates to the goal management screen of the application. 2. The system displays the current goal details. 3. The user clicks the edit icon next to the goal title. 4. The user modifies the goal title, amount or period as needed. 5. The user confirms and updates the goal settings. 6. The system validates the changes. 7. The system saves the updated goal information.
Sub flows	-
Alternate flow	6a. If the changes are invalid, the system prompts the user to correct them.

Table 3.3.18 Duplicate Goal Use Case Description

Use Case ID	00018
Use Case Name	Duplicate Goal
Brief Description	User creates a copy of an existing goal.
Actor	User
Trigger	The user decides to duplicate a goal.
Precondition	The user has at least one goal set up.

Normal flow of events	<ol style="list-style-type: none"> 1. The user navigates to the goal management screen of the application. 2. The system displays the current goal details. 3. The user clicks the duplicate icon next to the goal title. 4. The system duplicates the goal information and the saved amount and start date will be reset.
Sub flows	-
Alternate flow	-

Table 3.3.19 Download Goal Use Case Description

Use Case ID	00019
Use Case Name	Download Goal
Brief Description	User downloads their goal information.
Actor	User
Trigger	The user decides to download the goal information.
Precondition	The user has at least one goal set up.
Normal flow of events	<ol style="list-style-type: none"> 1. The user navigates to the goal management screen of the application. 2. The system displays the current goal details. 3. The user clicks on the goal they wish to download. 4. The system retrieves the user's goal information. 5. The user clicks the download icon at the top right corner which is next to the goal title. 6. The system generates a PDF containing goal details. 7. The system initiates the file download to the user's device. 8. The system confirms successful download.
Sub flows	-
Alternate flow	-

Table 3.3.20 Delete Goal Use Case Description

Use Case ID	00020
Use Case Name	Delete Goal
Brief Description	User removes an existing goal.
Actor	User
Trigger	The user decides to delete an existing goal.
Precondition	The user has at least one goal set up.
Normal flow of events	<ol style="list-style-type: none"> 1. The user navigates to the goal management screen of the application. 2. The system displays the current goal details. 3. The user clicks on the delete icon on the goal they wish to delete. 4. The system removes the goal from the user's account.
Sub flows	-
Alternate flow	-

Table 3.3.21 Add Transaction to Goal Use Case Description

Use Case ID	00021
Use Case Name	Add Transaction to Goal
Brief Description	The user adds a transaction to the goal.
Actor	User
Trigger	The user selects a transaction to add to an existing goal.
Precondition	The user must have at least one transaction and a goal.
Normal flow of events	<ol style="list-style-type: none"> 1. The user navigates to the home screen. 2. The system displays the available transactions or allows the user to create a new transaction. 3. The user selects or creates a transaction and chooses to add it to an existing goal.

	<p>4. The system updates the goal's progress with the transaction amount.</p> <p>5. The transaction is recorded as part of the selected goal.</p>
Sub flows	<p>Manual Entry:</p> <p>3a. The user manually adds a transaction by clicking the floating blue button on the screen.</p> <p>3b. The system prompts the user to fill in the transaction details and select it as “Income / Savings”.</p> <p>3c. The user selects a goal from the lists.</p> <p>Adding an Existing Transaction to a Goal:</p> <p>3a. The user clicks on an existing “Income / Savings” transaction from the transaction history.</p> <p>3b. The system shows the transaction details.</p> <p>3c. If the transaction has not yet been added to a goal, the user is given the option to add it to a goal.</p>
Alternate flow	-

Table 3.3.22 View Analytics Use Case Description

Use Case ID	00022
Use Case Name	View Analytics
Brief Description	This use case allows users to view their financial analytics.
Actor	User
Trigger	Users to view analytics.
Precondition	For User: The user must log in to their account.
Normal flow of events	<p>1. The user navigates to the home screen of the application.</p> <p>2. The user clicks on the “Analytics” button on the home screen.</p>

	3. The system generates visual representations of data such as charts and graphs.
Sub flows	-
Alternate flow	3a. The user can interact with the analytics by changing periods or categories.

Table 3.3.23 View Transaction Use Case Description

Use Case ID	00023
Use Case Name	View Transaction
Brief Description	This use case allows users and administrators to view transactions.
Actor	User, Administrator
Trigger	Users and administrators want to view transaction details.
Precondition	For Administrator: The administrator must have access to view the users' transactions. For User: The user must log in to their account.
Normal flow of events	For Administrator: <ol style="list-style-type: none"> 1. Log in to Firebase using their credentials. 2. Navigate to the Firestore database in Firebase. 3. Select the user's transactions they wish to view. For User: <ol style="list-style-type: none"> 1. Navigate to the home screen of the application. 2. The user can view the transaction history on the home screen or click the "more" icon next to the Transactions on the home screen. 3. The system will display all the transactions. 4. The user can interact with the transaction by changing periods or categories.

Sub flows	<p>For User:</p> <ul style="list-style-type: none"> 3a. The user can click on the transaction history to view the details such as amount, transaction type, payment details and date. 3b. The user can add the transaction to the existing budget or goal. 3c. The user can upload a receipt to the transaction details screen. 3d. The User can perform OCR if a receipt is uploaded.
Alternate flow	-

Table 3.3.24 View Balance Use Case Description

Use Case ID	00024
Use Case Name	View Balance
Brief Description	This use case allows users and administrators to view the wallet balance.
Actor	User, Administrator
Trigger	Users and administrators want to view the wallet balance.
Precondition	<p>For Administrator: The administrator must have access to view the users' wallet balances.</p> <p>For User: -</p>
Normal flow of events	<p>For Administrator:</p> <ul style="list-style-type: none"> 1. Log in to Firebase using their credentials. 2. Navigate to the Firestore database in Firebase. 3. Select the user's wallet balance they wish to view. <p>For User:</p> <ul style="list-style-type: none"> 1. Navigate to the home screen of the application.

	2. View the balance of the virtual wallet.
Sub flows	-
Alternate flow	-

Table 3.3.25 View User Authentication Use Case Description

Use Case ID	00025
Use Case Name	View User Authentication
Brief Description	This use case involves administrators viewing users' authentication.
Actor	Administrator
Trigger	The administrator wants to view user authentication.
Precondition	The administrator must have appropriate permissions and access rights.
Normal flow of events	<ol style="list-style-type: none"> 1. The administrator logs in to the Firebase console using their credentials. 2. The administrator navigates to the user authentication section. 3. The administrator can view and check the user authentication.
Sub flows	<ol style="list-style-type: none"> 3a. The administrator can disable the user account temporarily if desired. 3b. The administrator can delete the user account permanently if desired.
Alternate flow	-

Table 3.3.26 Perform OCR Use Case Description

Use Case ID	00026
Use Case Name	Perform OCR
Brief Description	This use case involves the user extracting text from uploaded receipts.
Actor	User
Trigger	The user wishes to assign the item without manual entry.
Precondition	The user must log in to the application and an image must be uploaded to a transaction.
Normal flow of events	<ol style="list-style-type: none"> 1. Navigate to the home screen of the application. 2. The user can view the transaction history on the home screen or click the “more” icon next to the Transactions on the home screen. 3. The system will display all the transactions. 4. The user can click on the transaction history. 5. The user uploads a receipt to the transaction details screen. 6. The User clicks the “Extract Text” button after the receipt is uploaded. 7. The system will start performing OCR. 8. The system will display the result of the OCR. 9. The user selects the result item to be assigned and recorded. 10. The user selects the category for each item. 11. The item will be recorded and assigned to the transaction history
Sub flows	-
Alternate flow	-

3.4 Activity Diagram

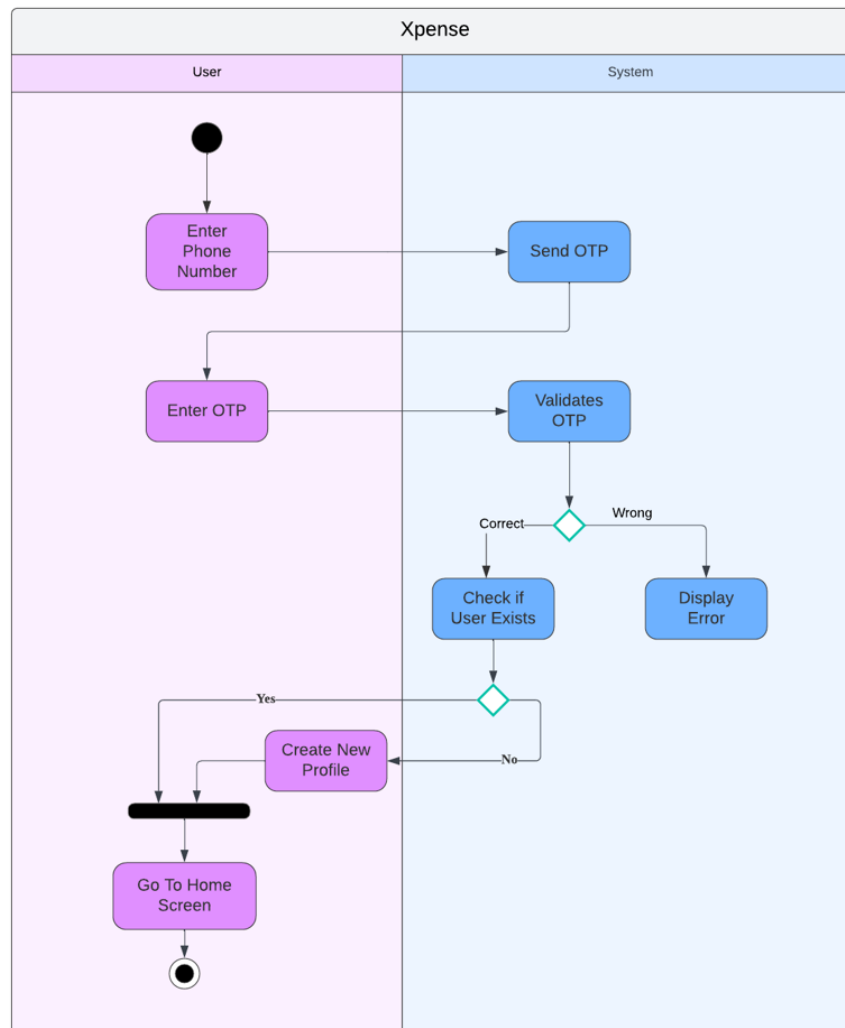


Figure 3.4.1 Activity Diagram for Login

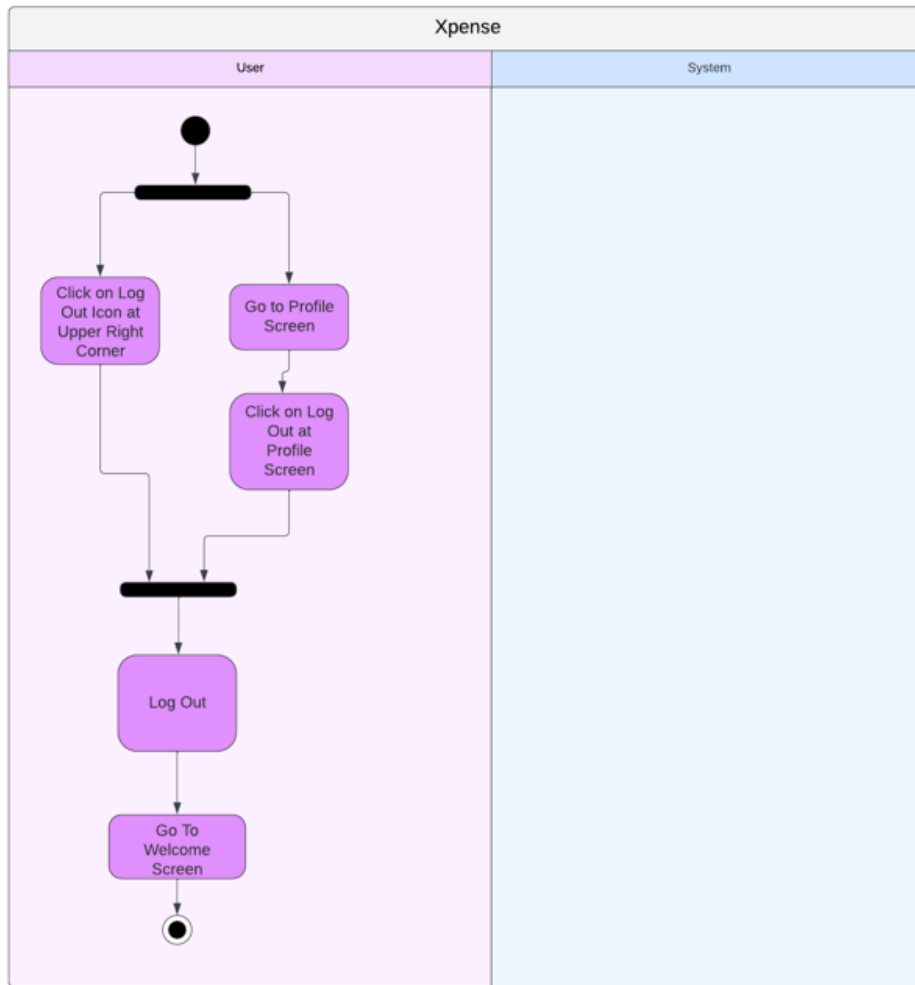


Figure 3.4.2 Activity Diagram for Log Out

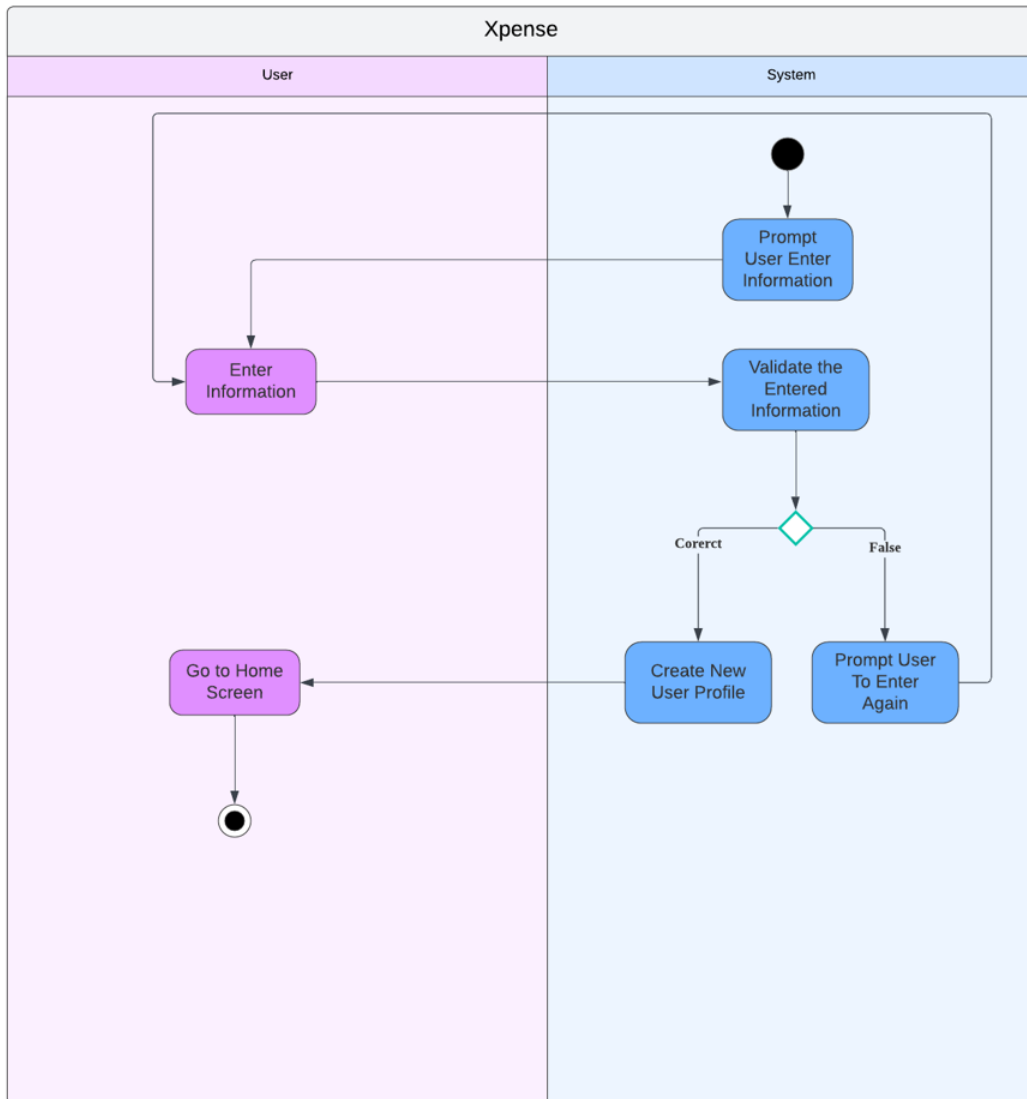


Figure 3.4.3 Activity Diagram for Create Profile

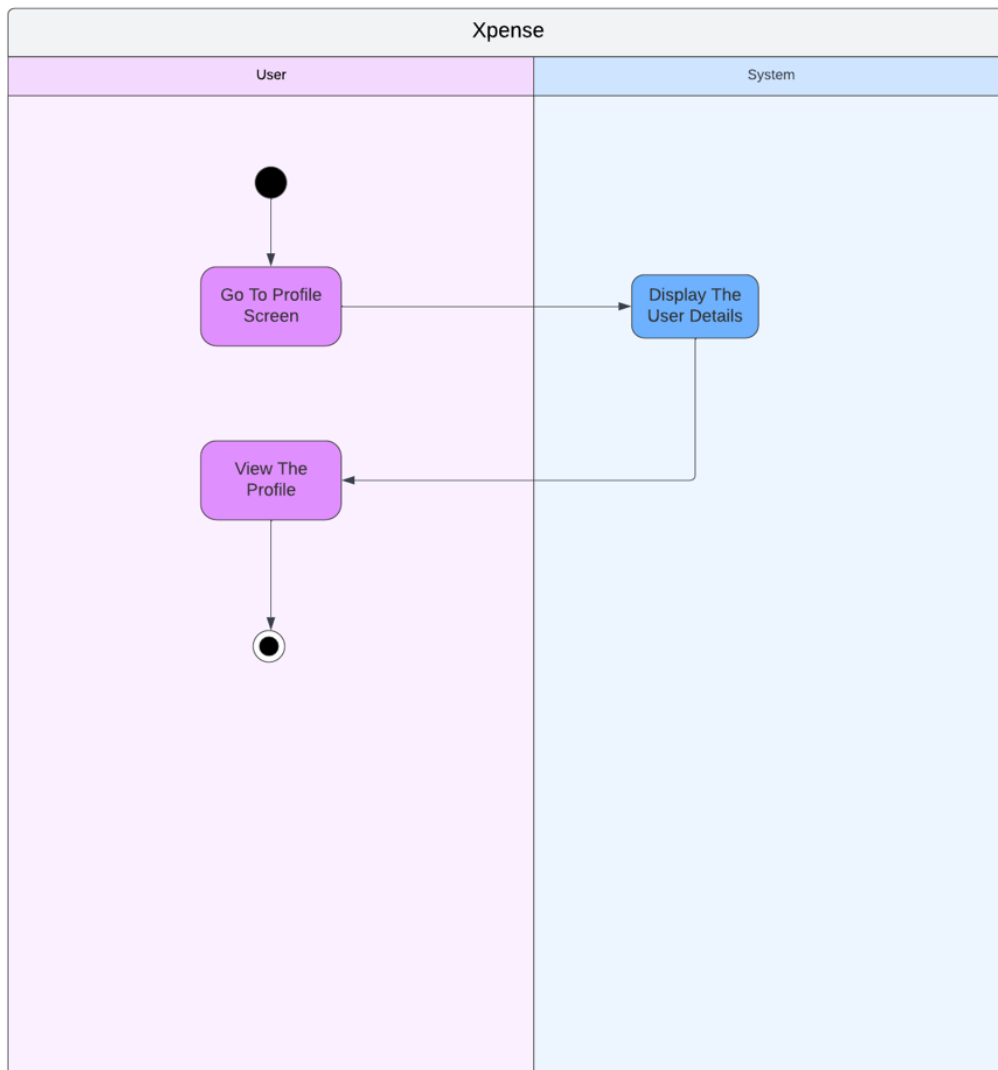


Figure 3.4.4 Activity Diagram for View Profile

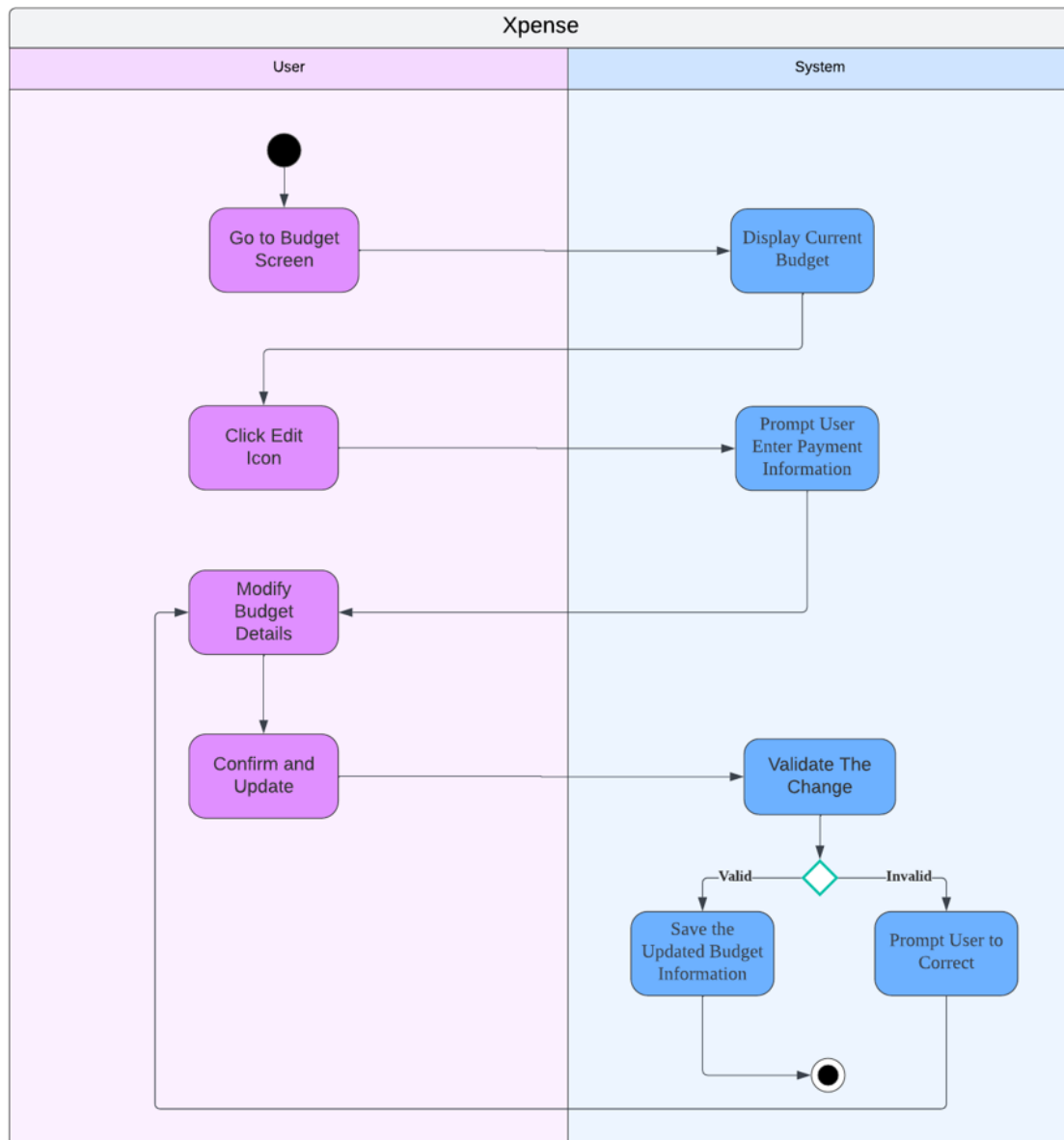


Figure 3.4.5 Activity Diagram for Edit Profile

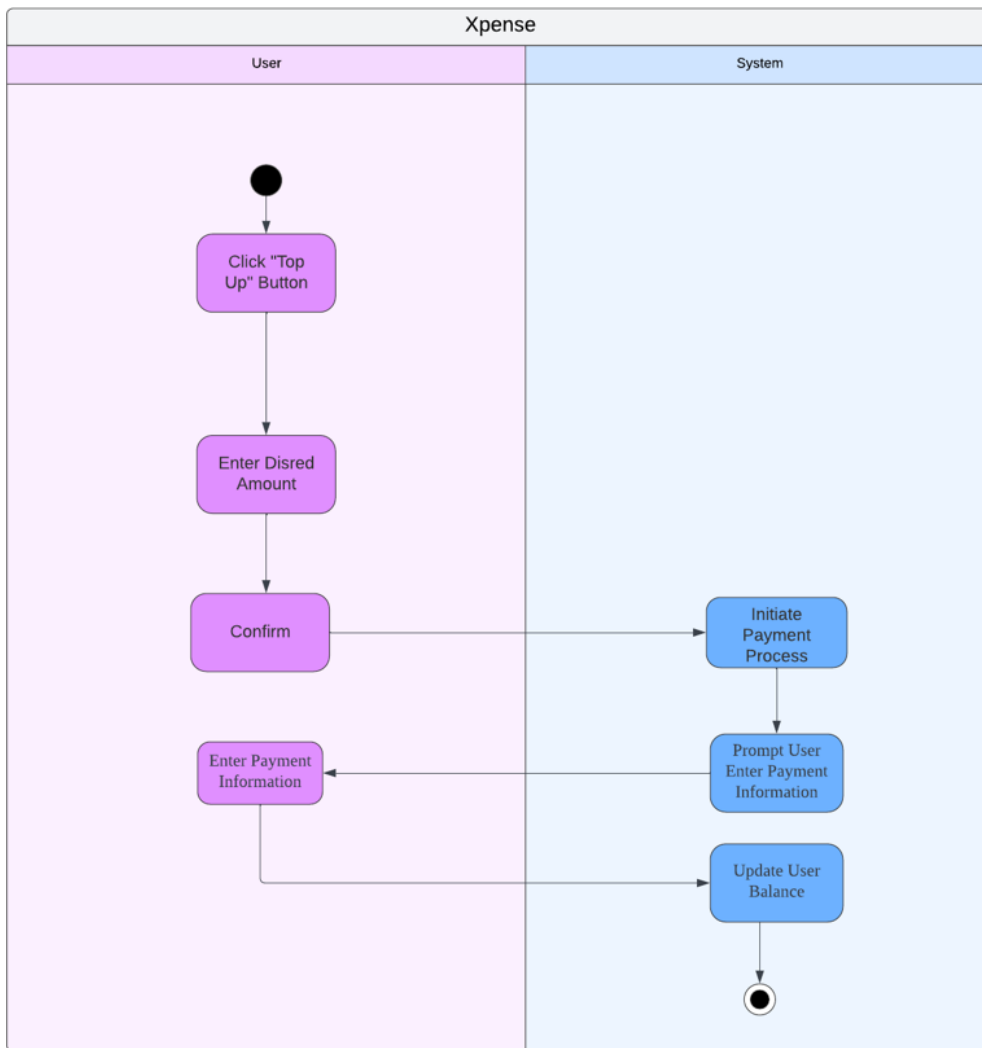


Figure 3.4.6 Activity Diagram for Top Up

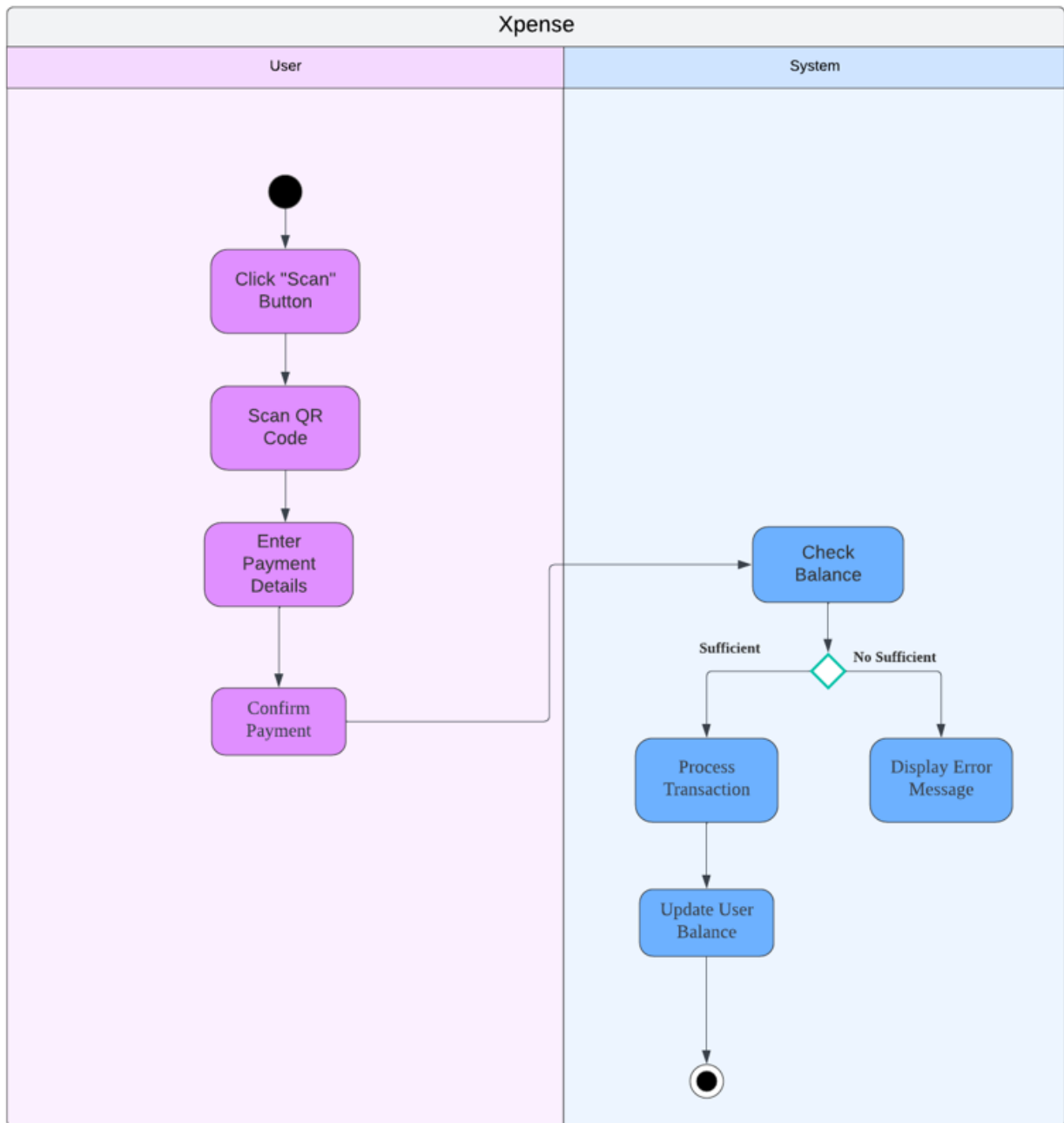


Figure 3.4.7 Activity Diagram for Make Payment

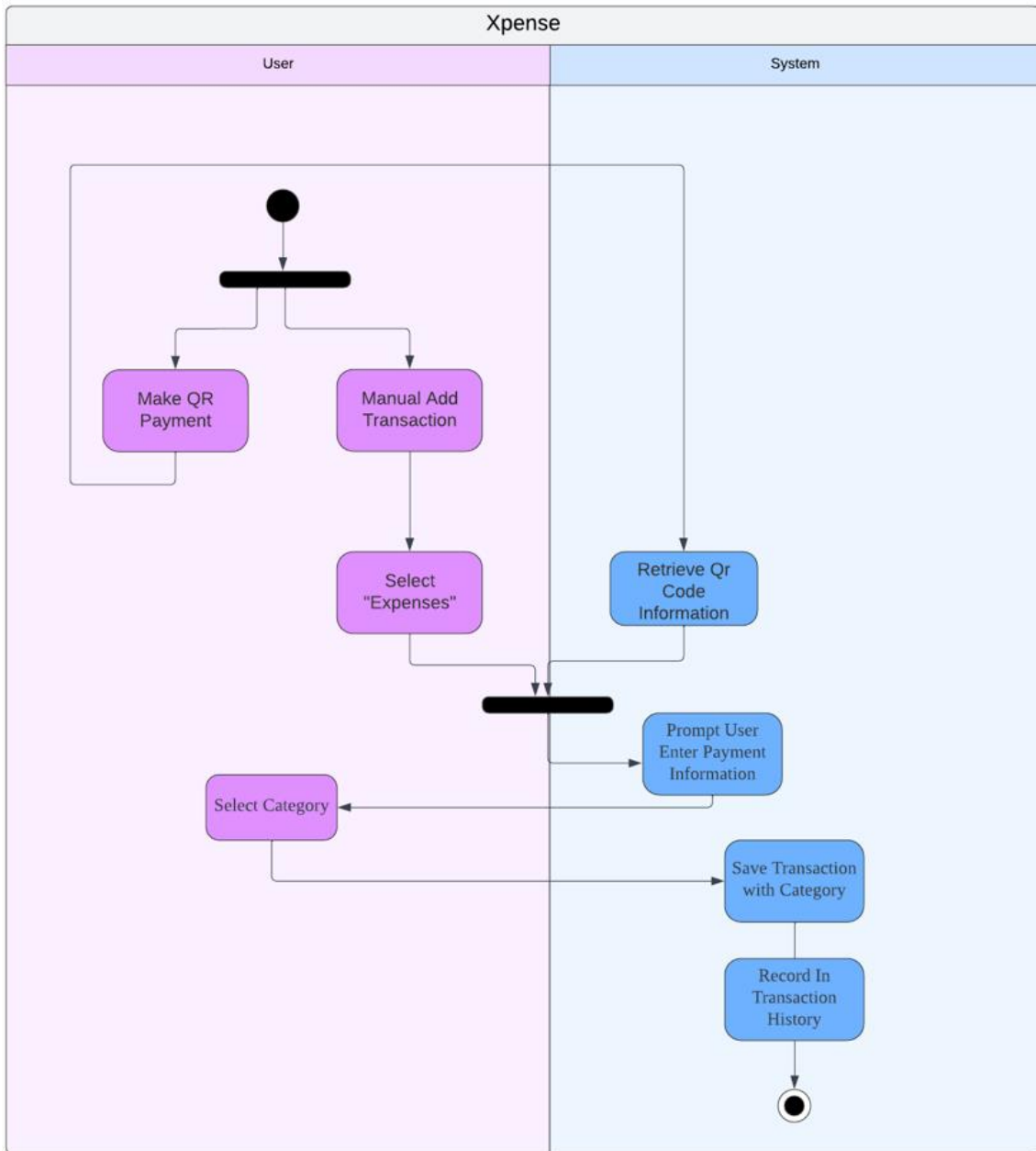


Figure 3.4.8 Activity Diagram for Categorize Transaction

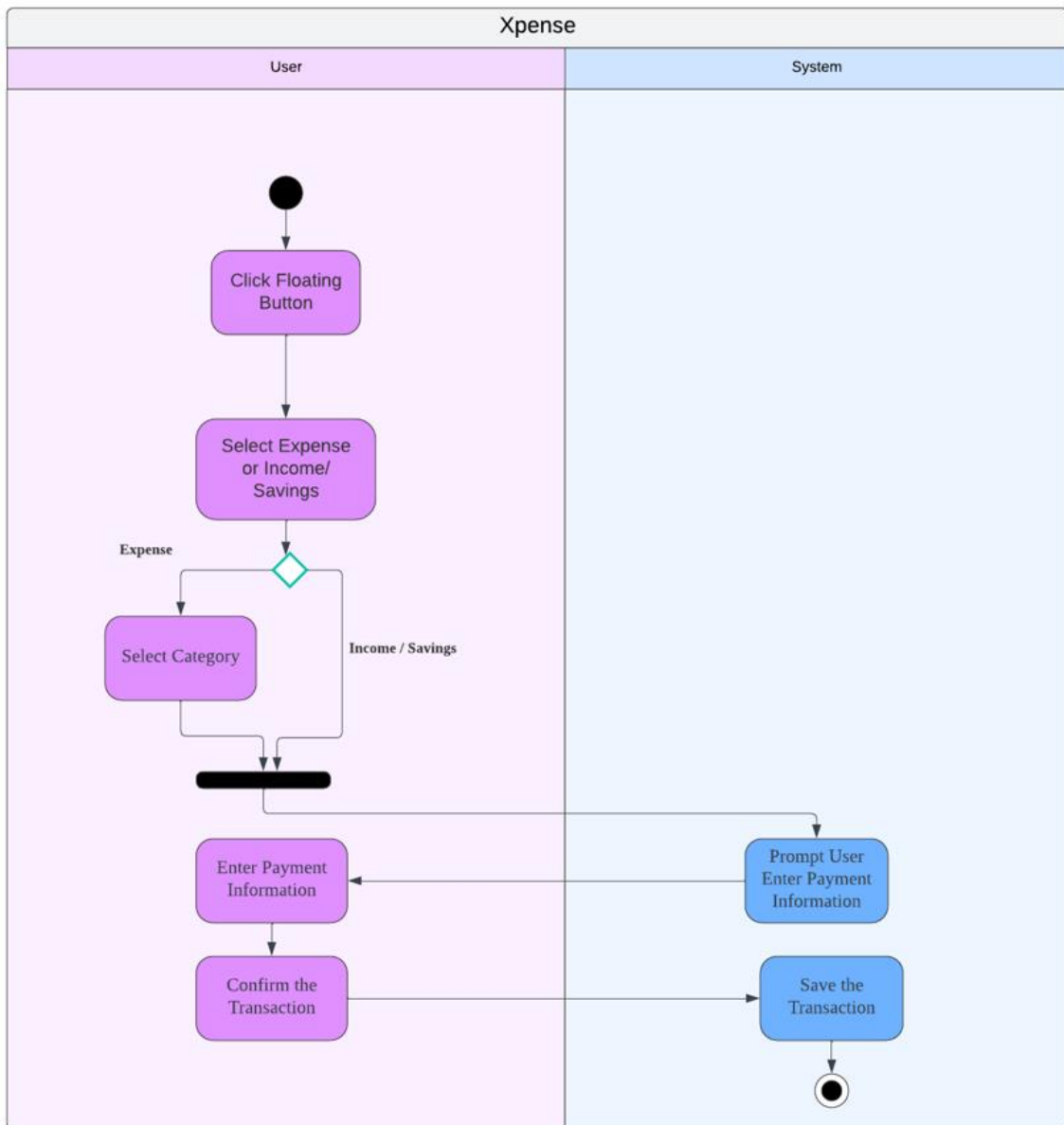


Figure 3.4.9 Activity Diagram for Manual Input Transaction

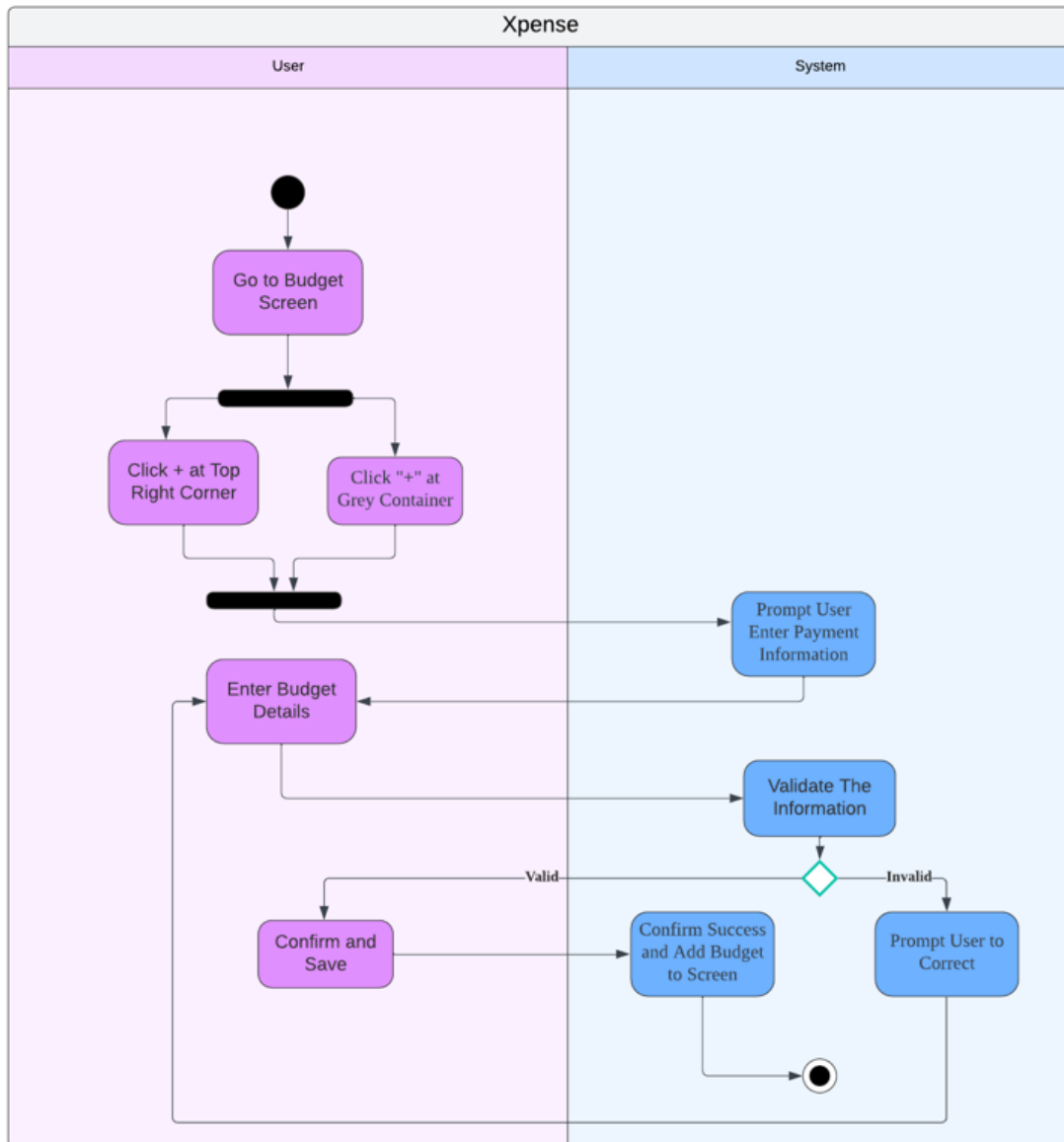


Figure 3.4.10 Activity Diagram for Set Budget

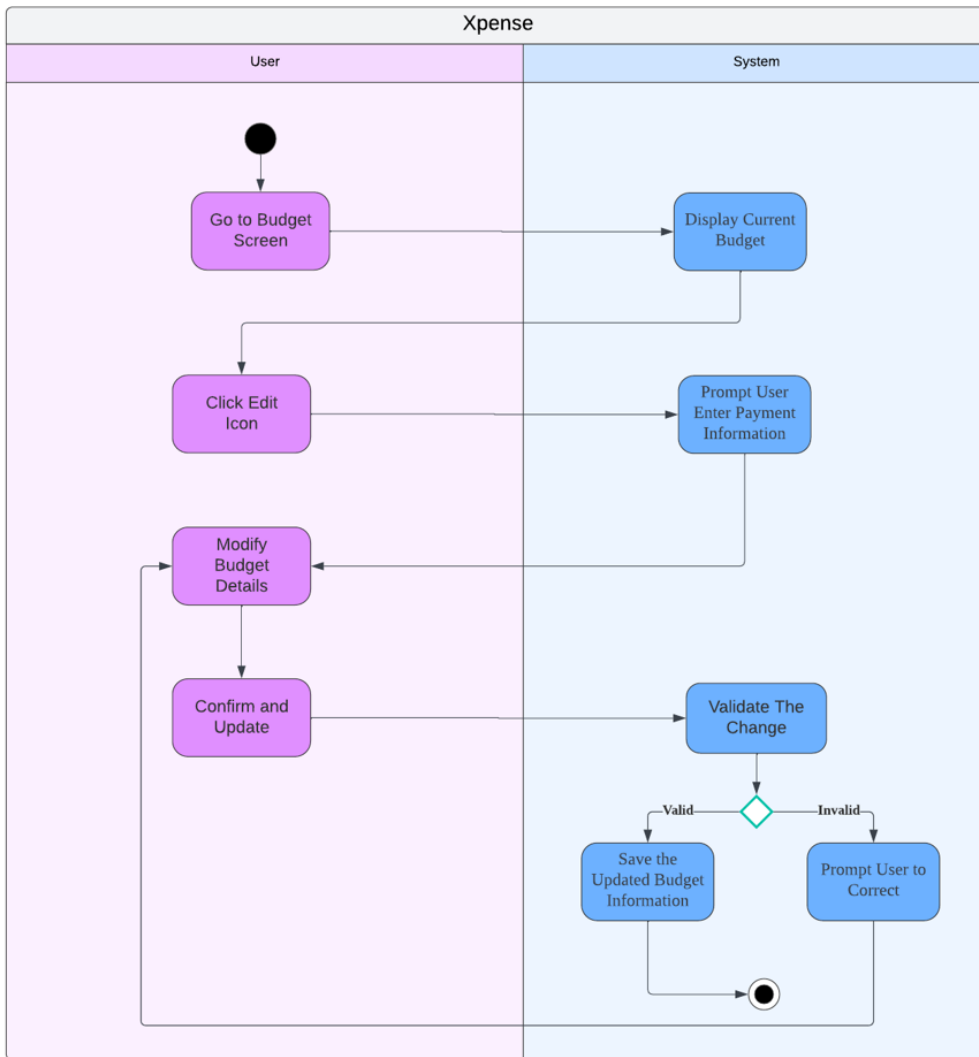


Figure 3.4.11 Activity Diagram for Edit Budget

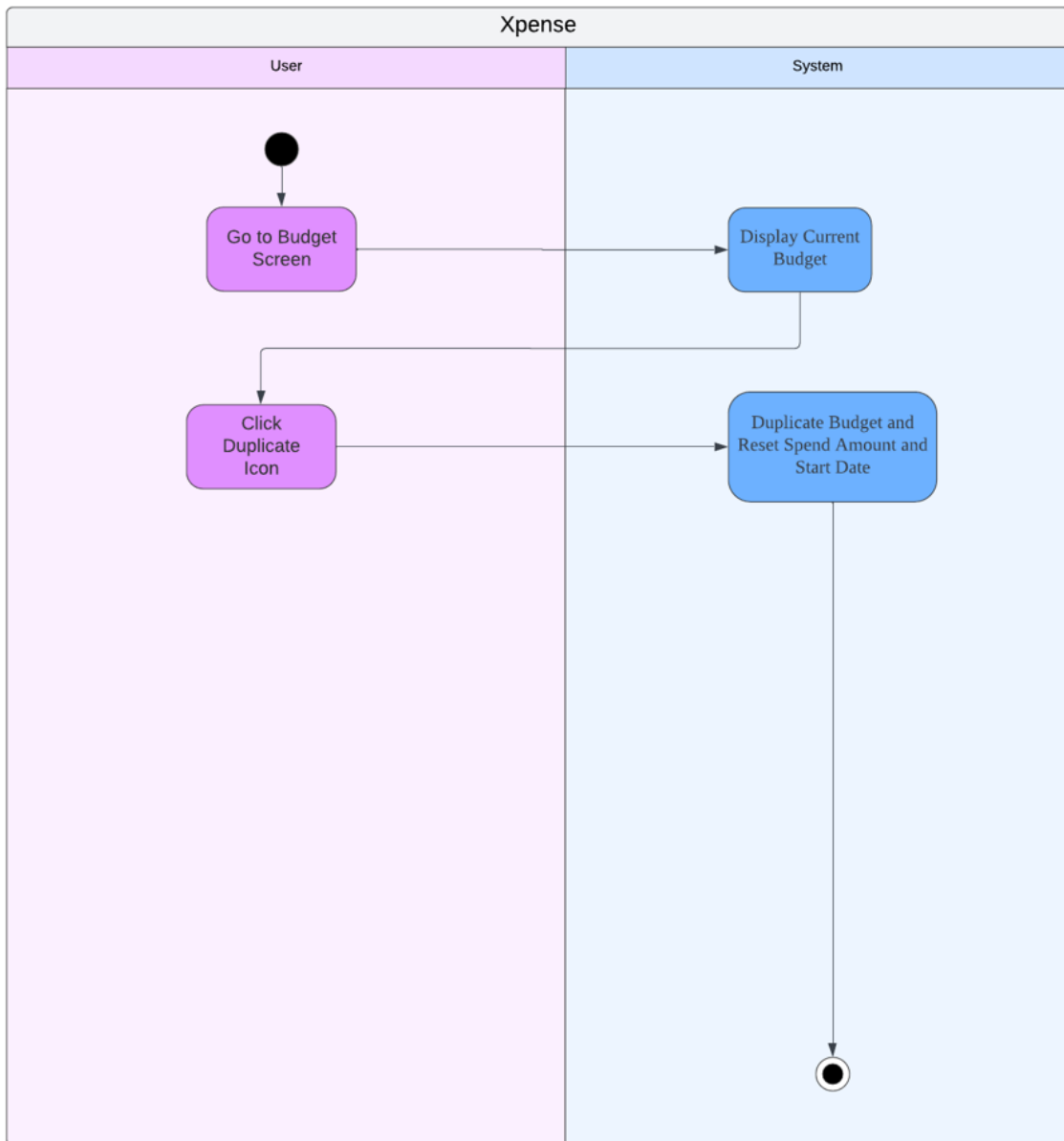


Figure 3.4.12 Activity Diagram for Duplicate Budget

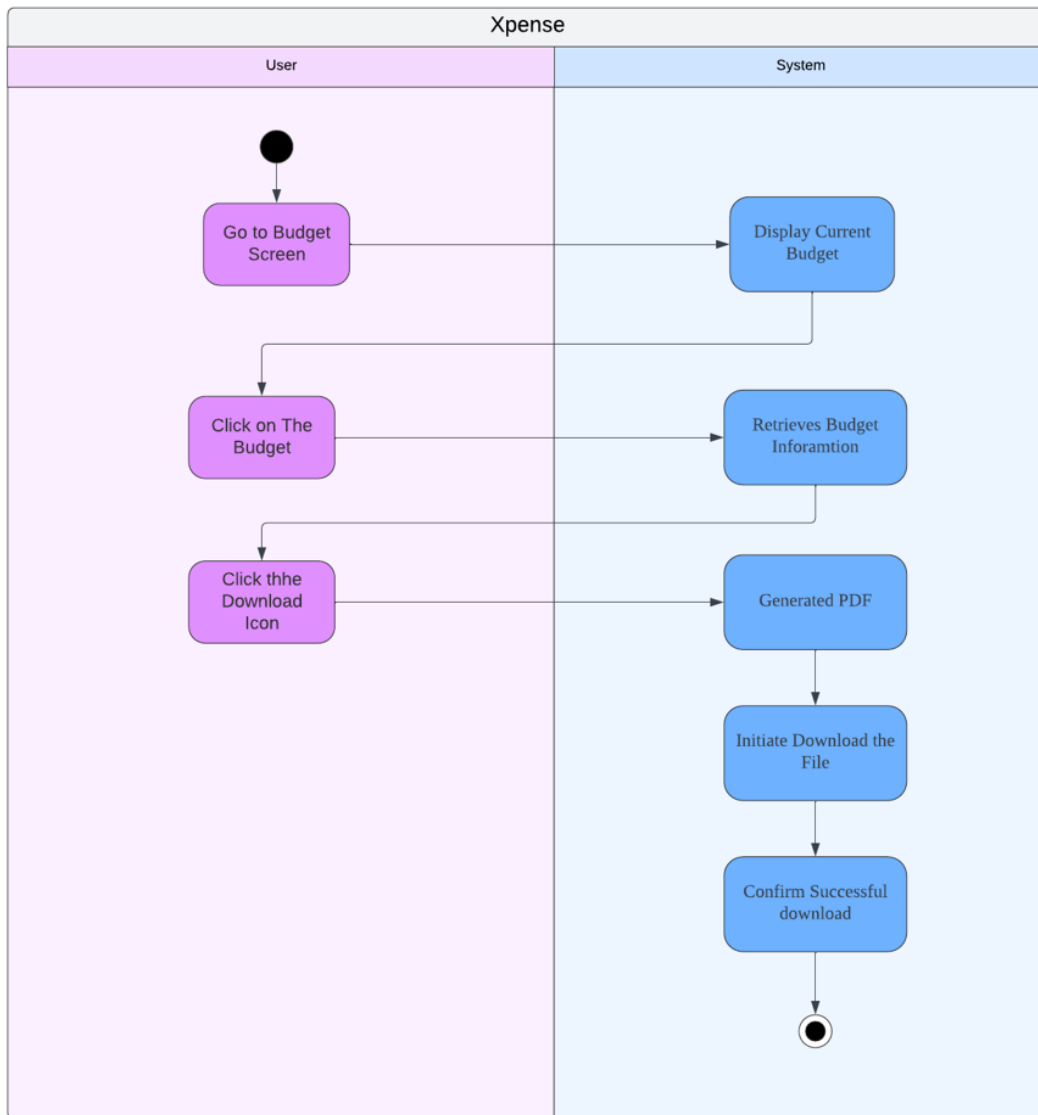


Figure 3.4.13 Activity Diagram for Download Budget

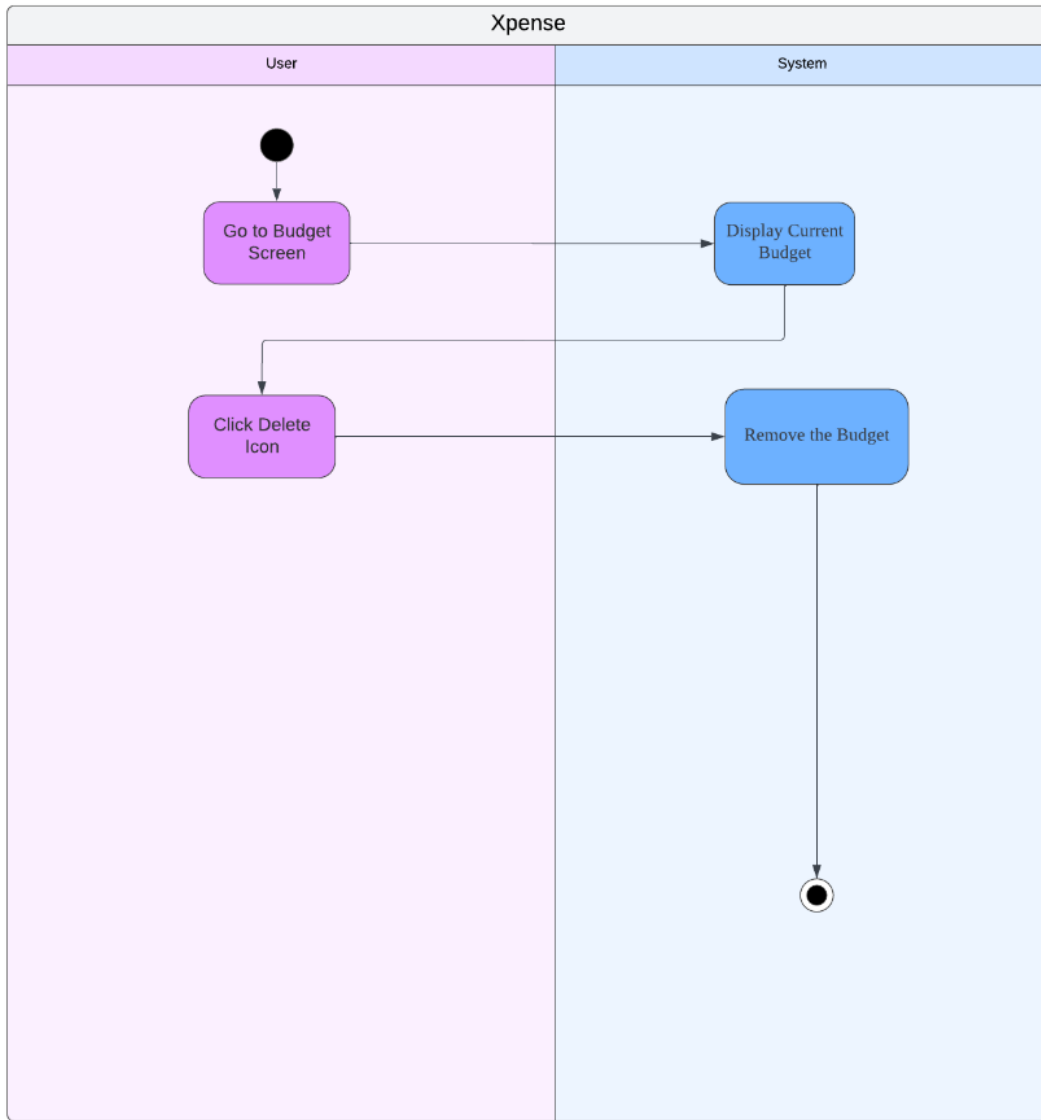


Figure 3.4.14 Activity Diagram for Delete Budget

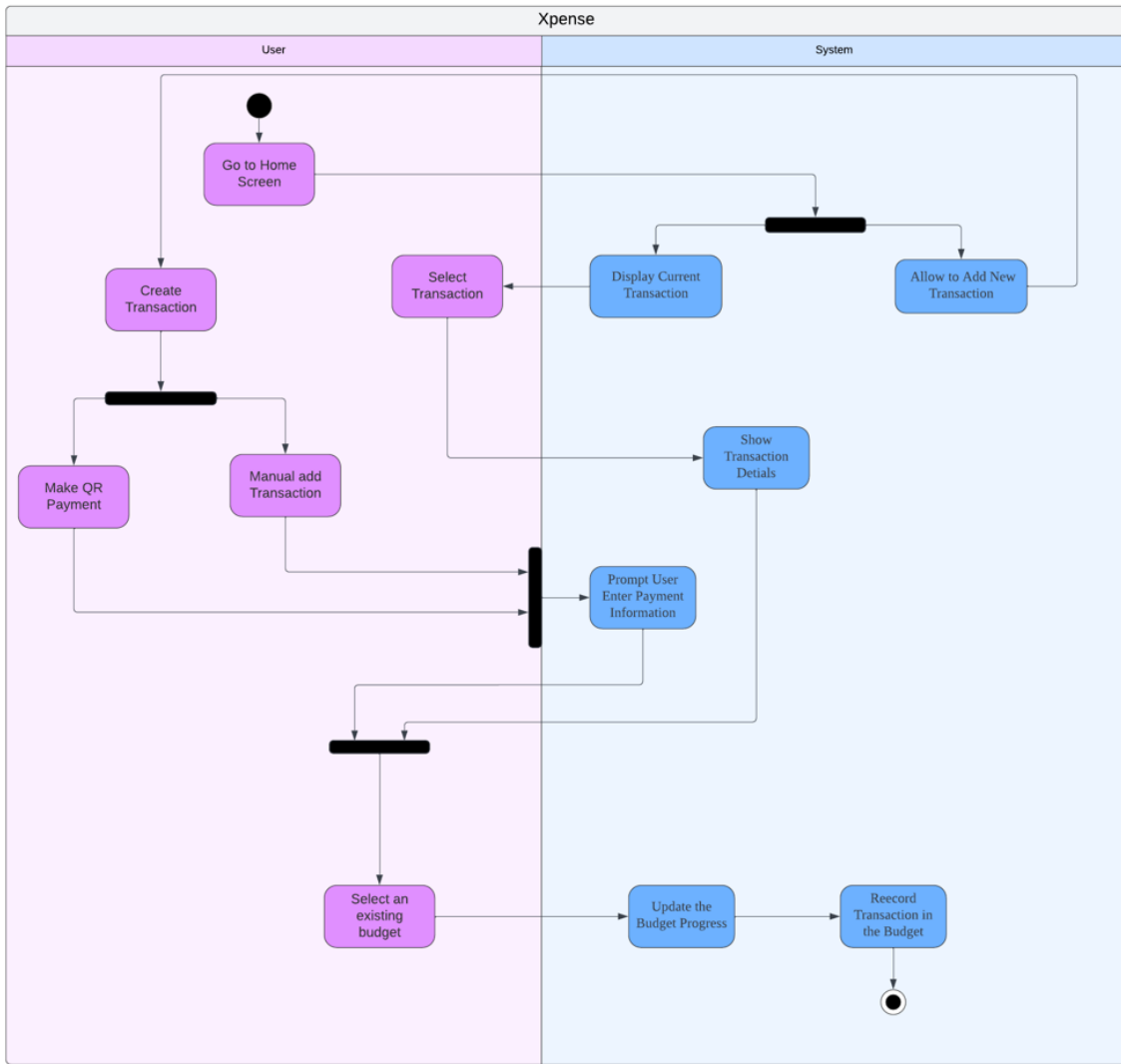


Figure 3.4.15 Activity Diagram for Add Transaction to Budget

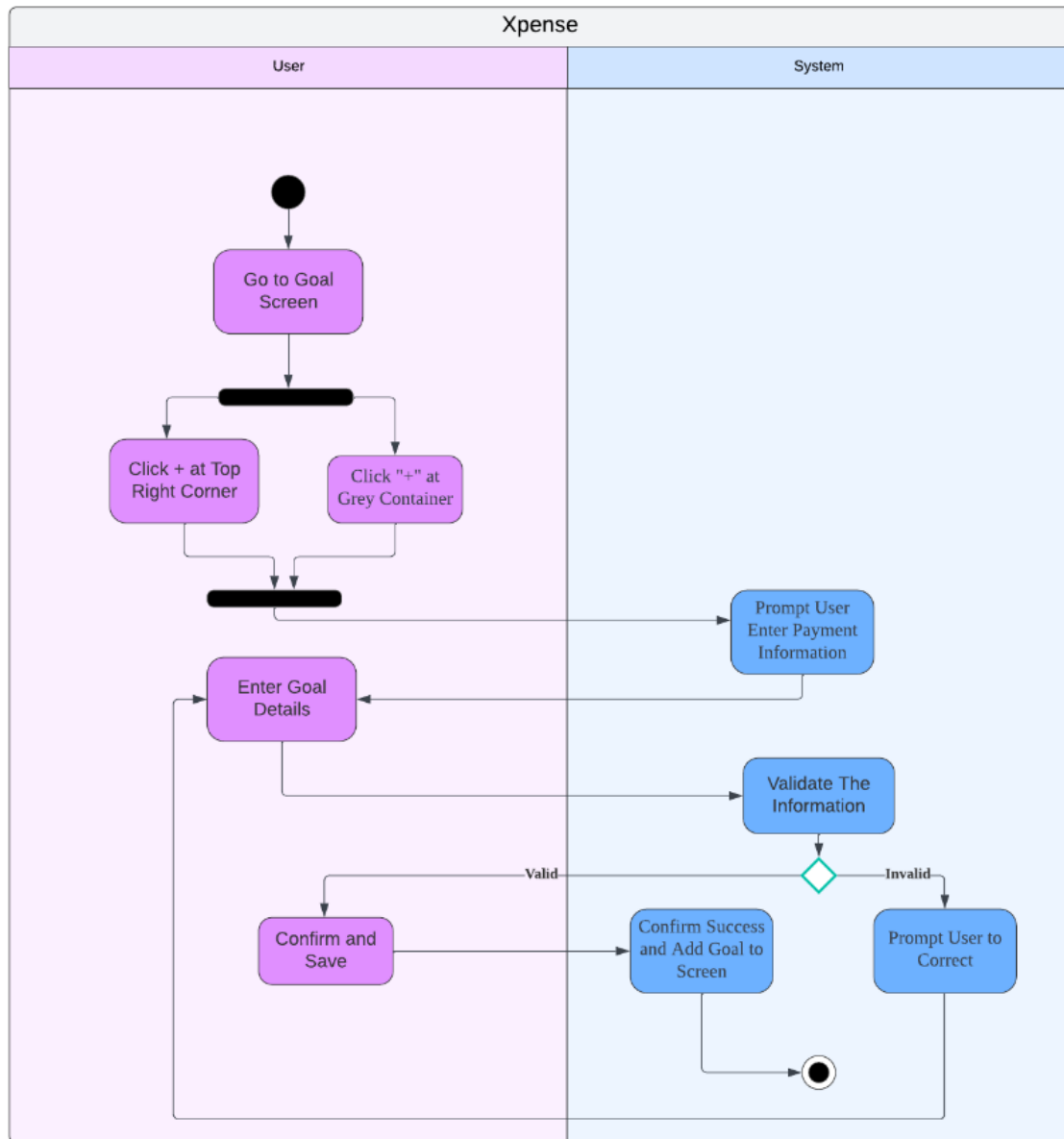


Figure 3.4.16 Activity Diagram for Set Goal

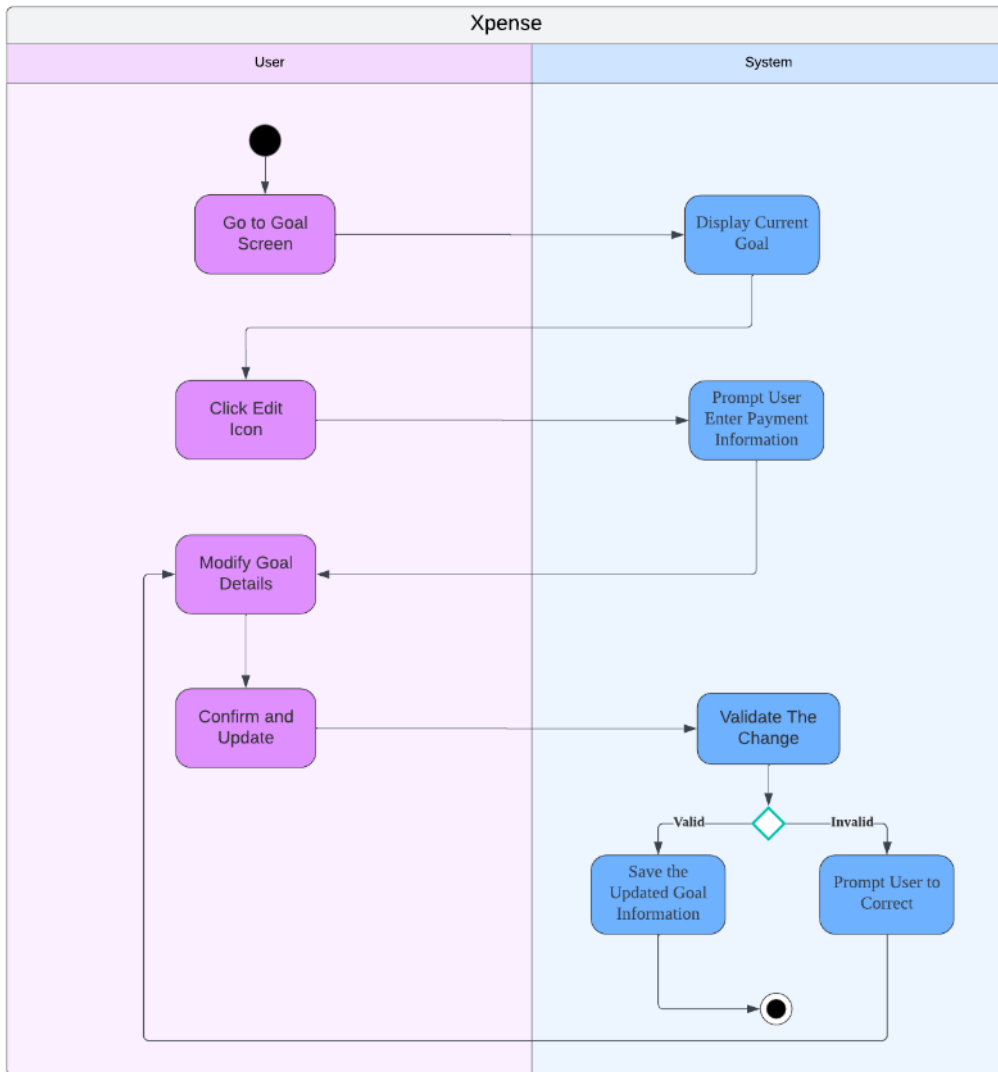


Figure 3.4.17 Activity Diagram for Edit Goal

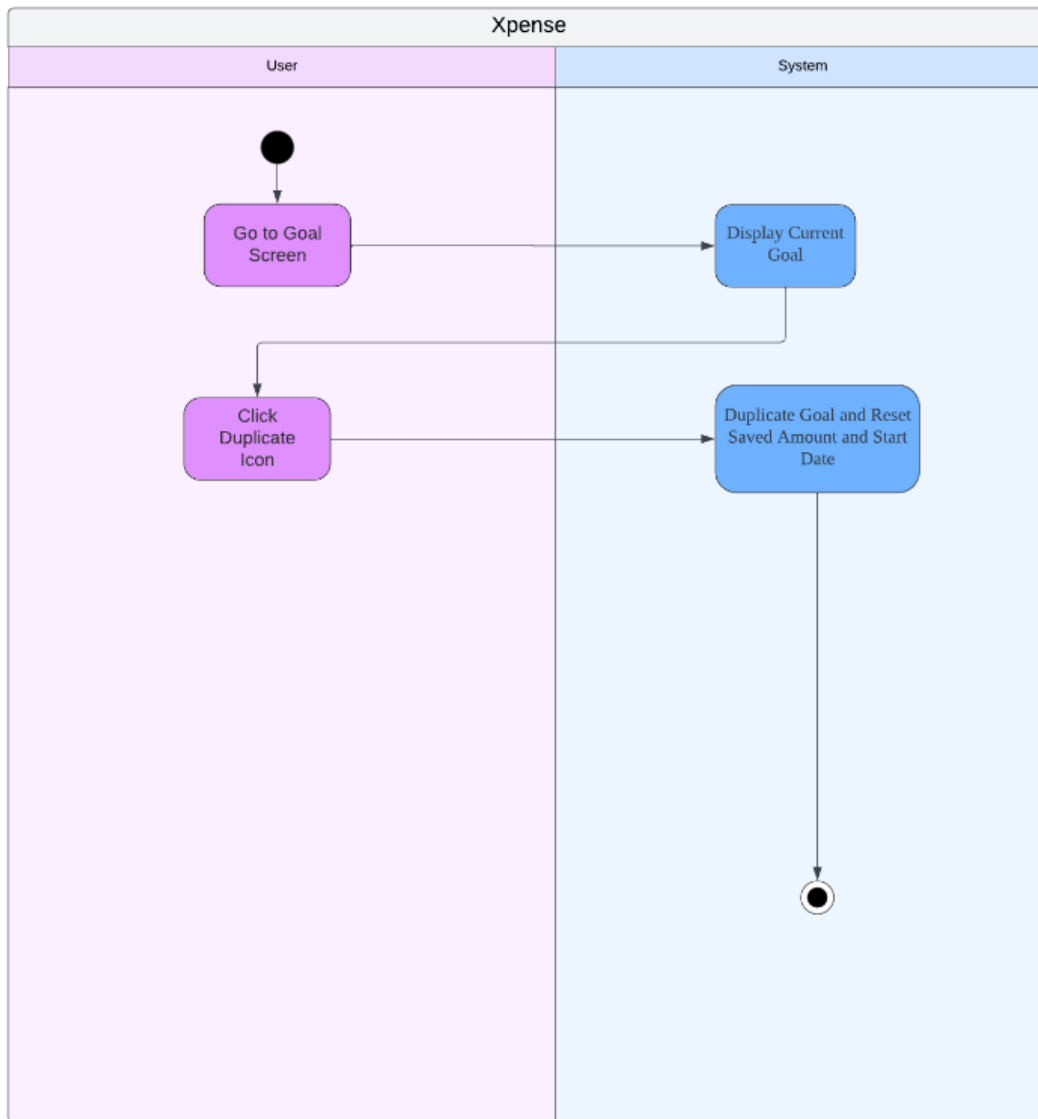


Figure 3.4.18 Activity Diagram for Duplicate Goal

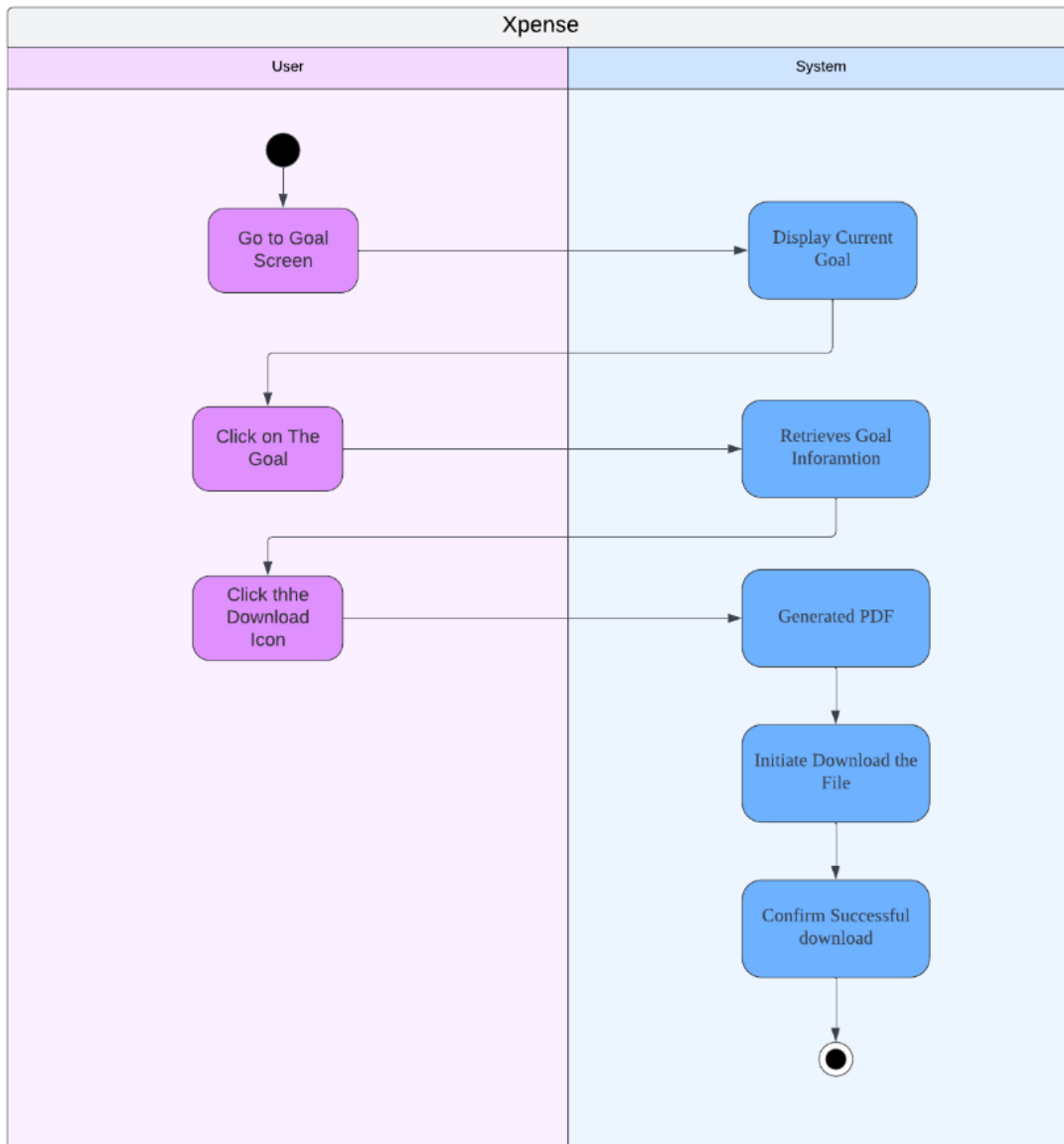


Figure 3.4.19 Activity Diagram for Download Goal

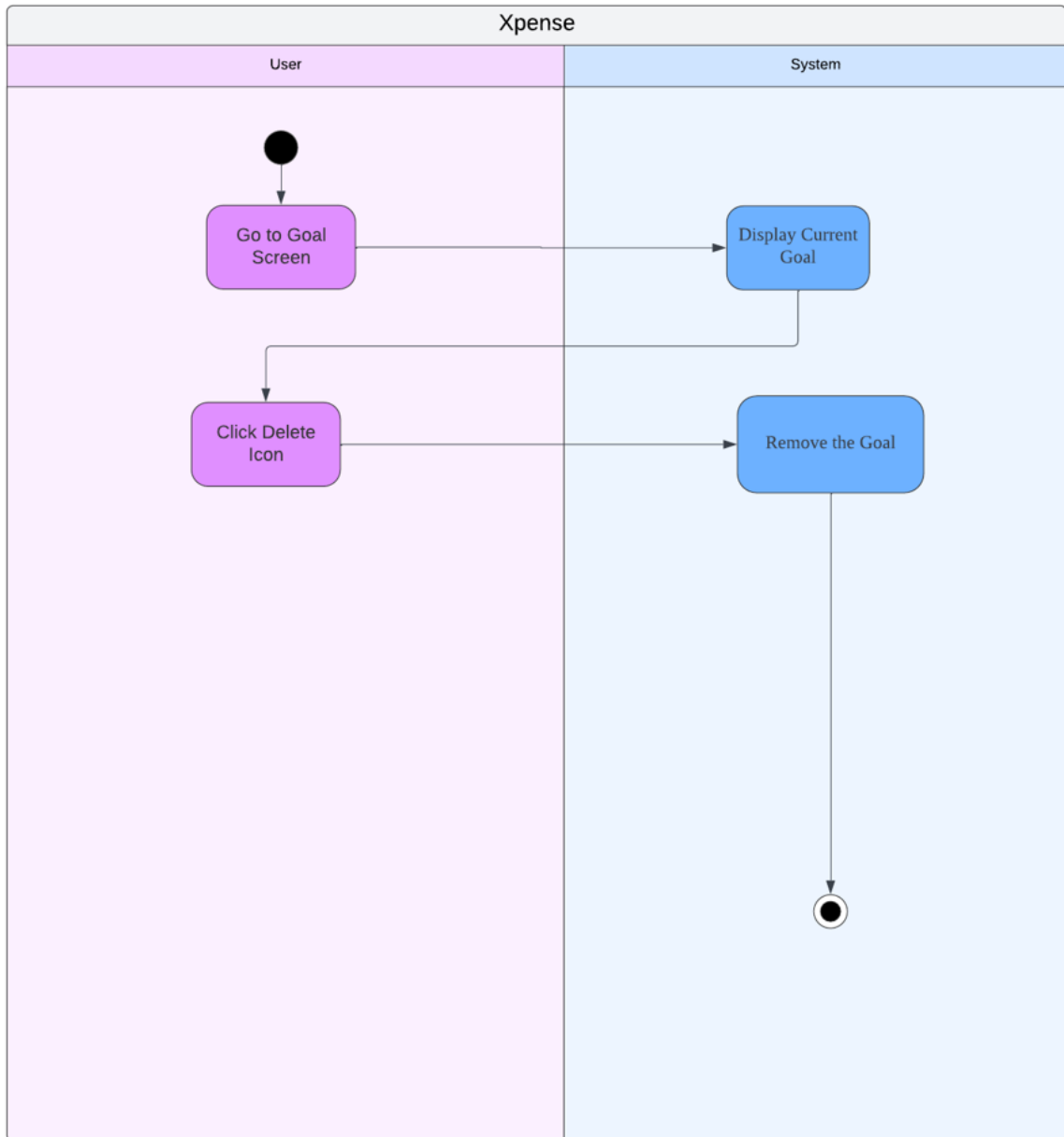


Figure 3.4.20 Activity Diagram for Delete Goal

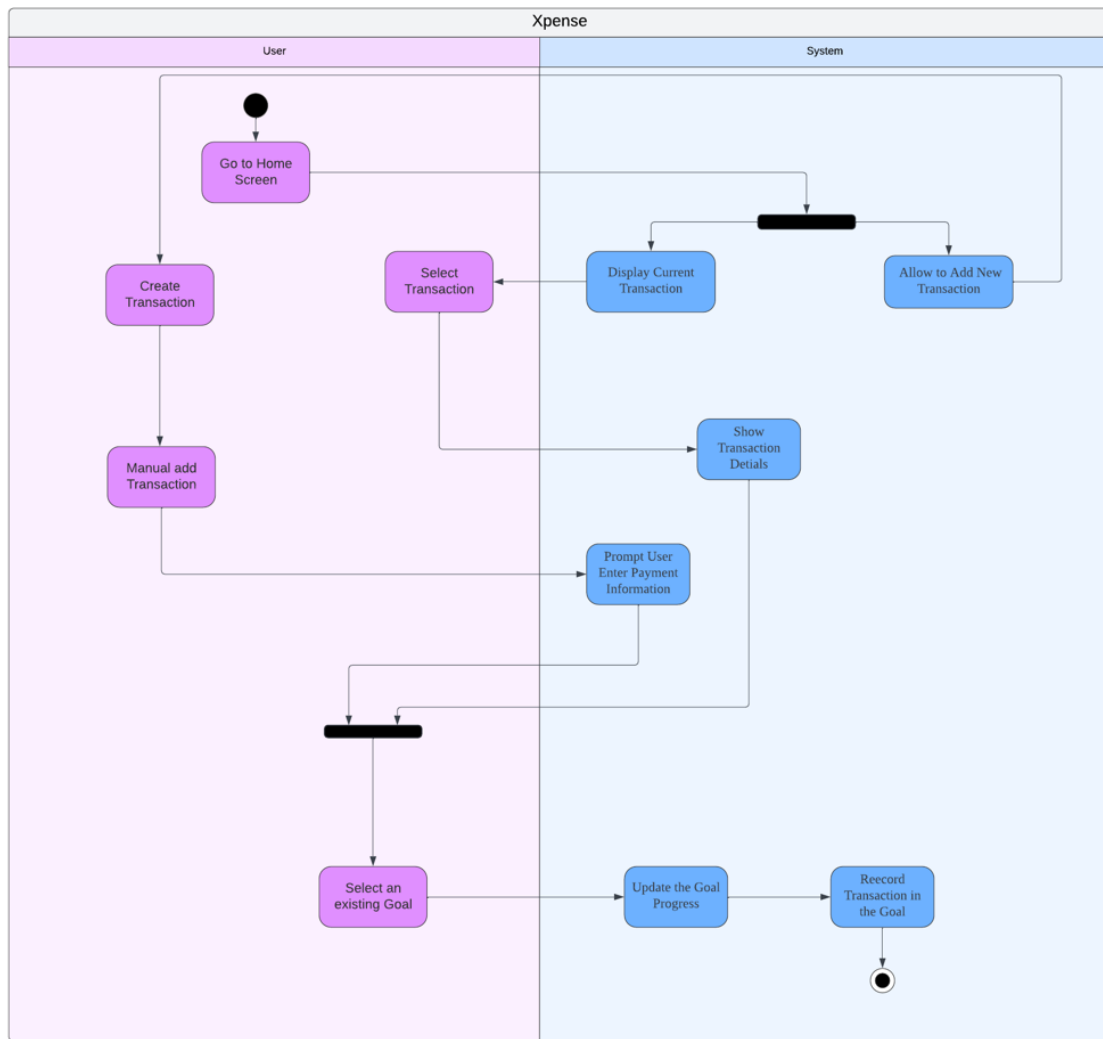


Figure 3.4.21 Activity Diagram for Add Transaction to Goal

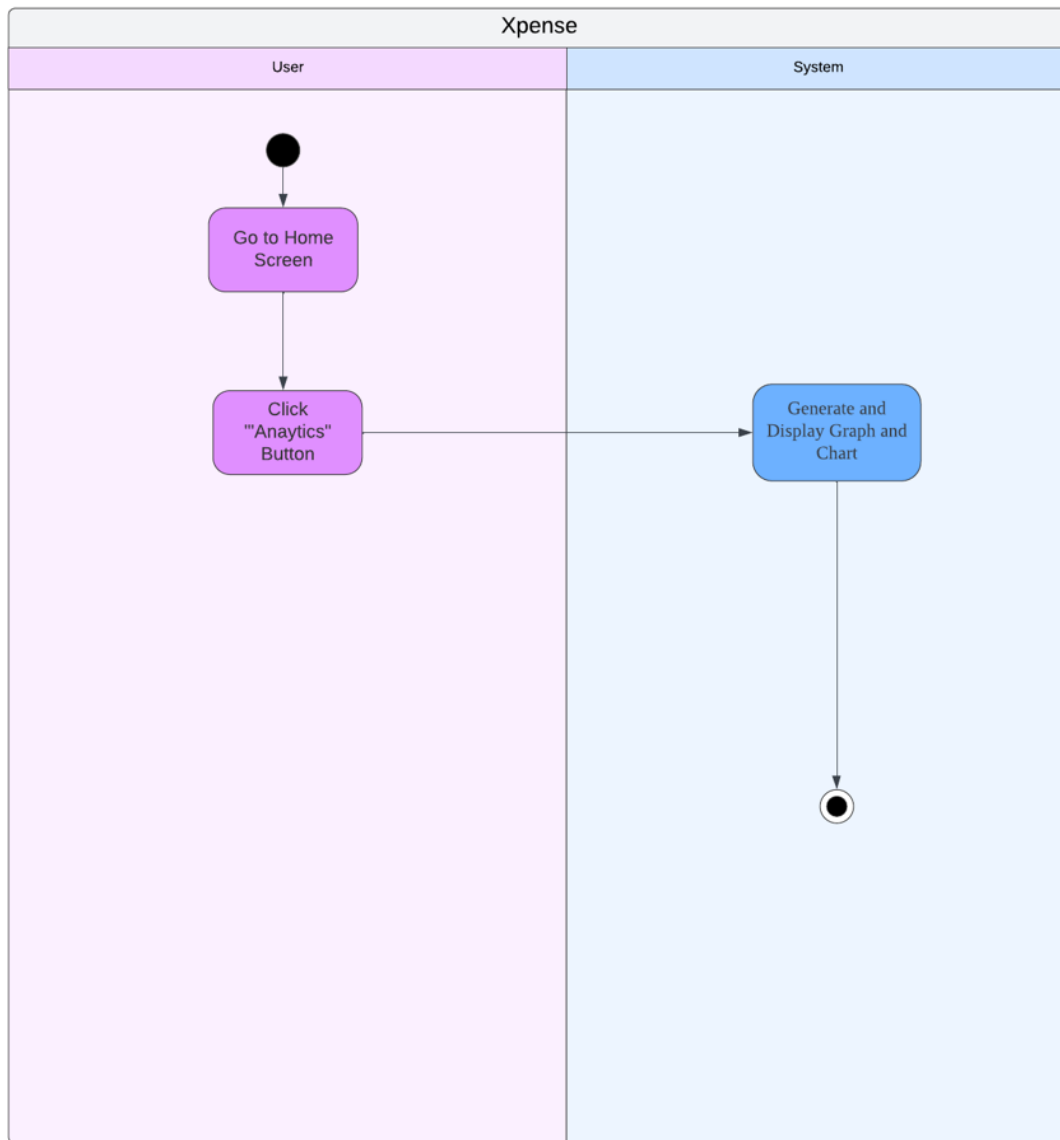


Figure 3.4.22 Activity Diagram for View Analytics

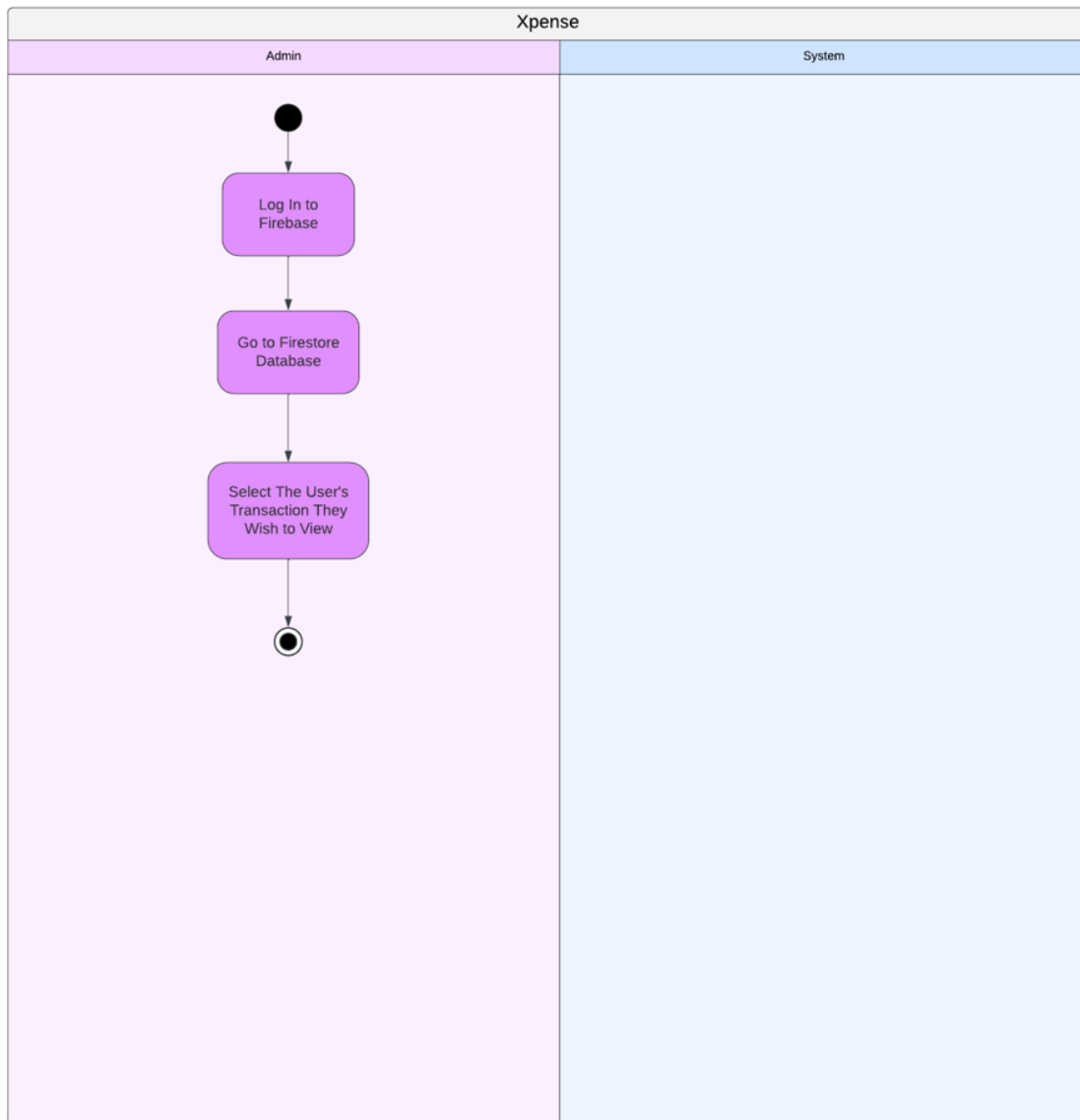


Figure 3.4.23 Activity Diagram for View Transaction for Admin

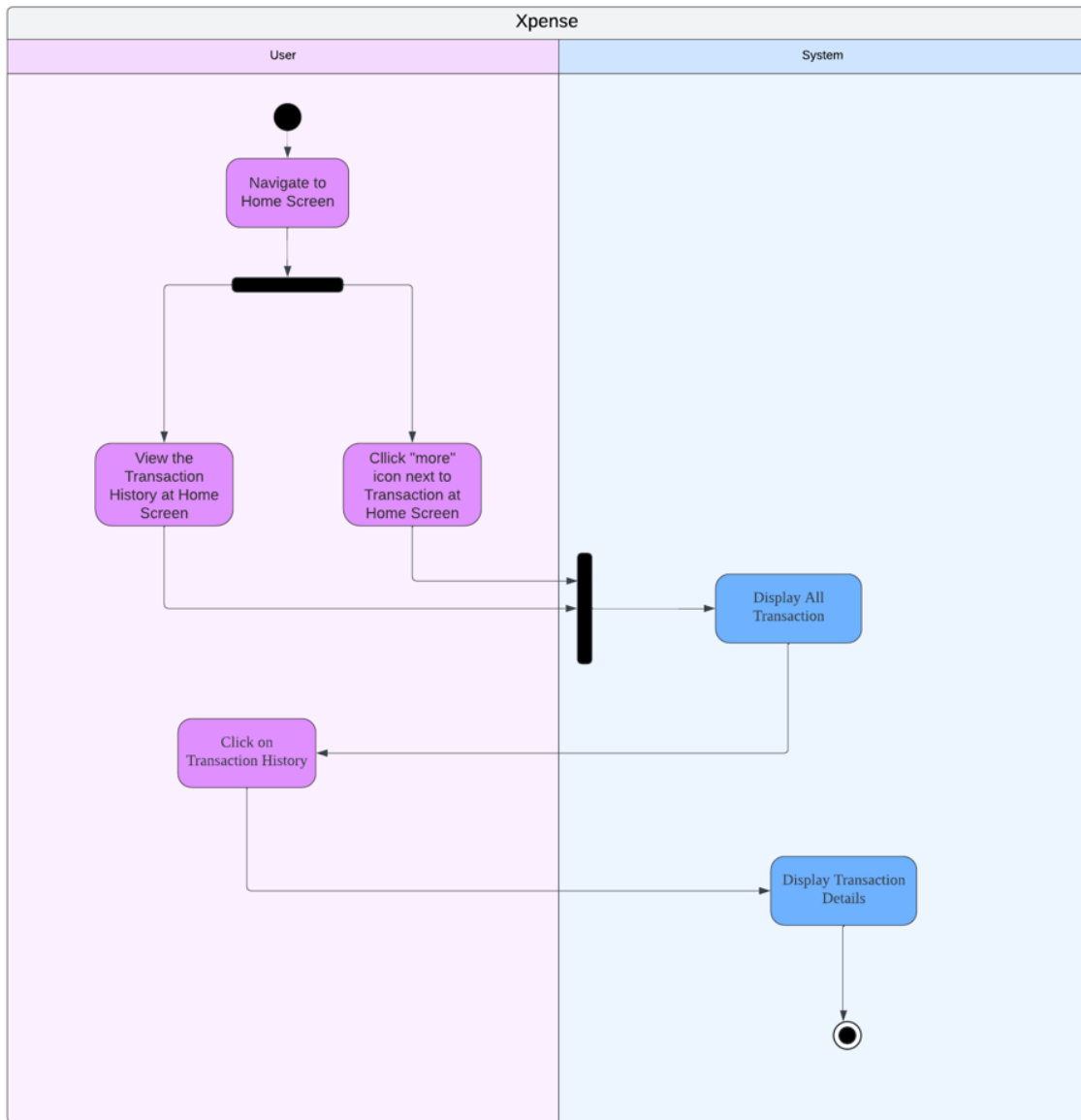


Figure 3.4.24 Activity Diagram for View Transaction for User

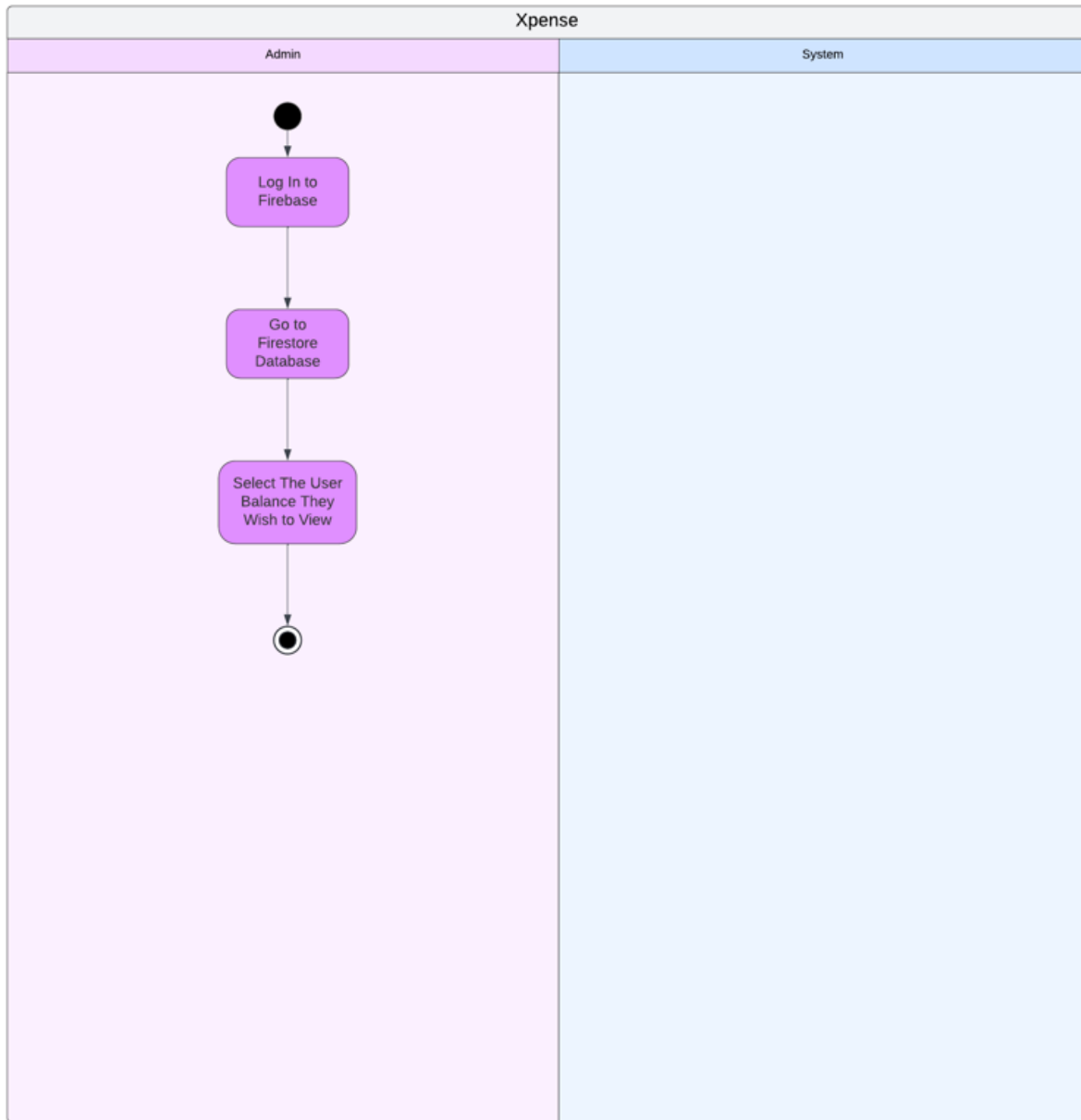


Figure 3.4.25 Activity Diagram for View Balance for Admin

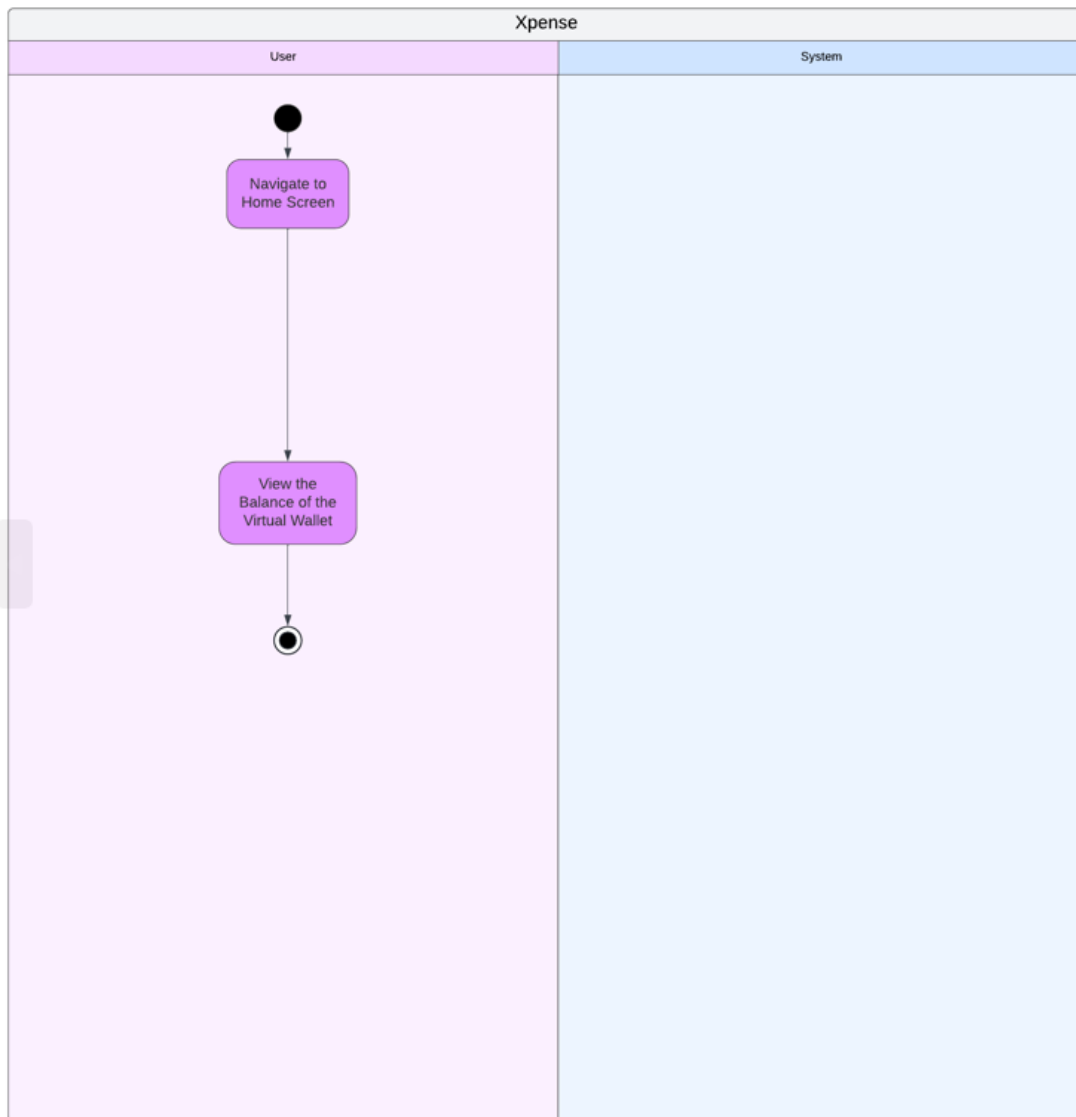


Figure 3.4.26 Activity Diagram for View Balance for User

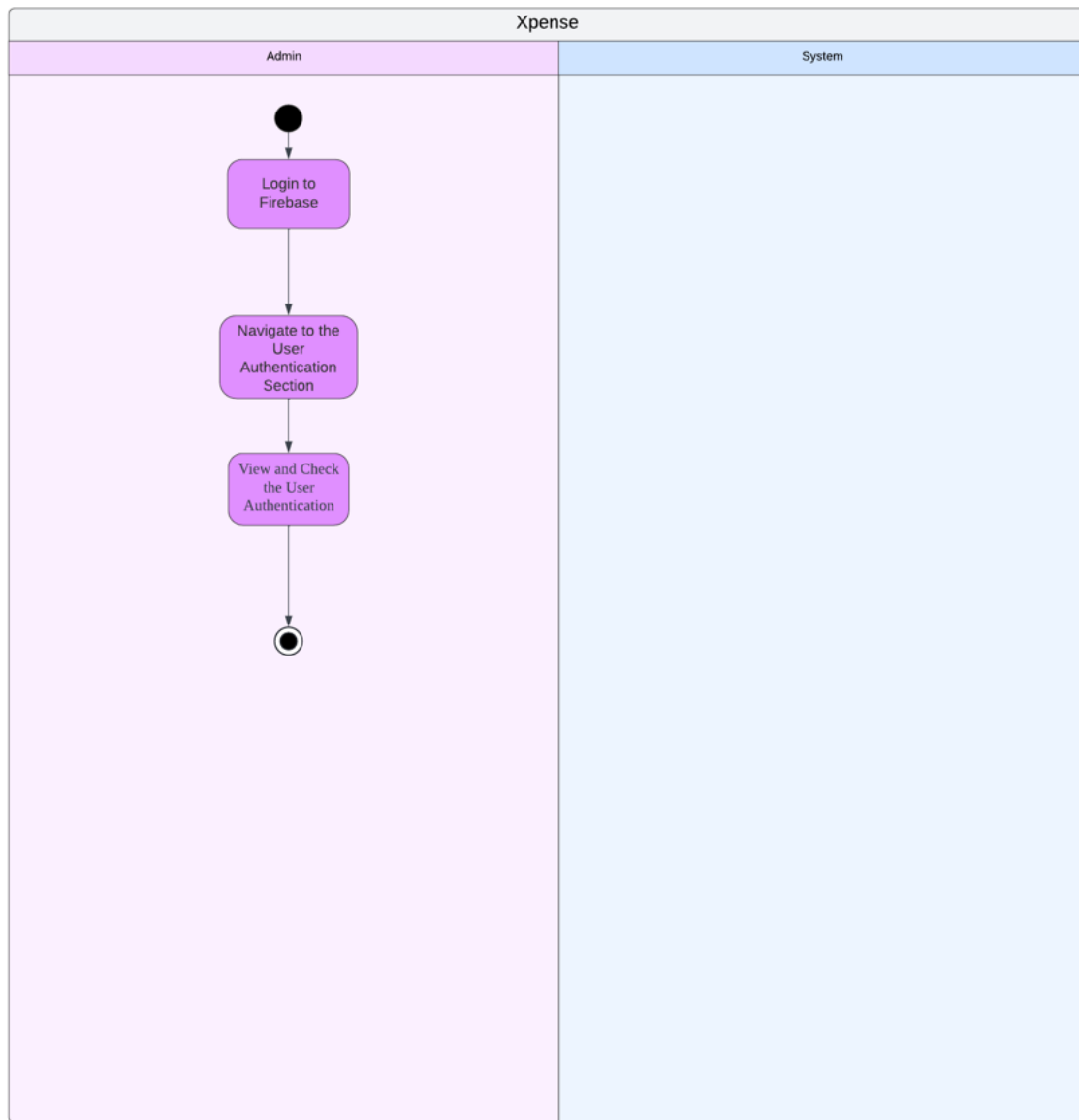


Figure 3.4.27 Activity Diagram for View User Authentication

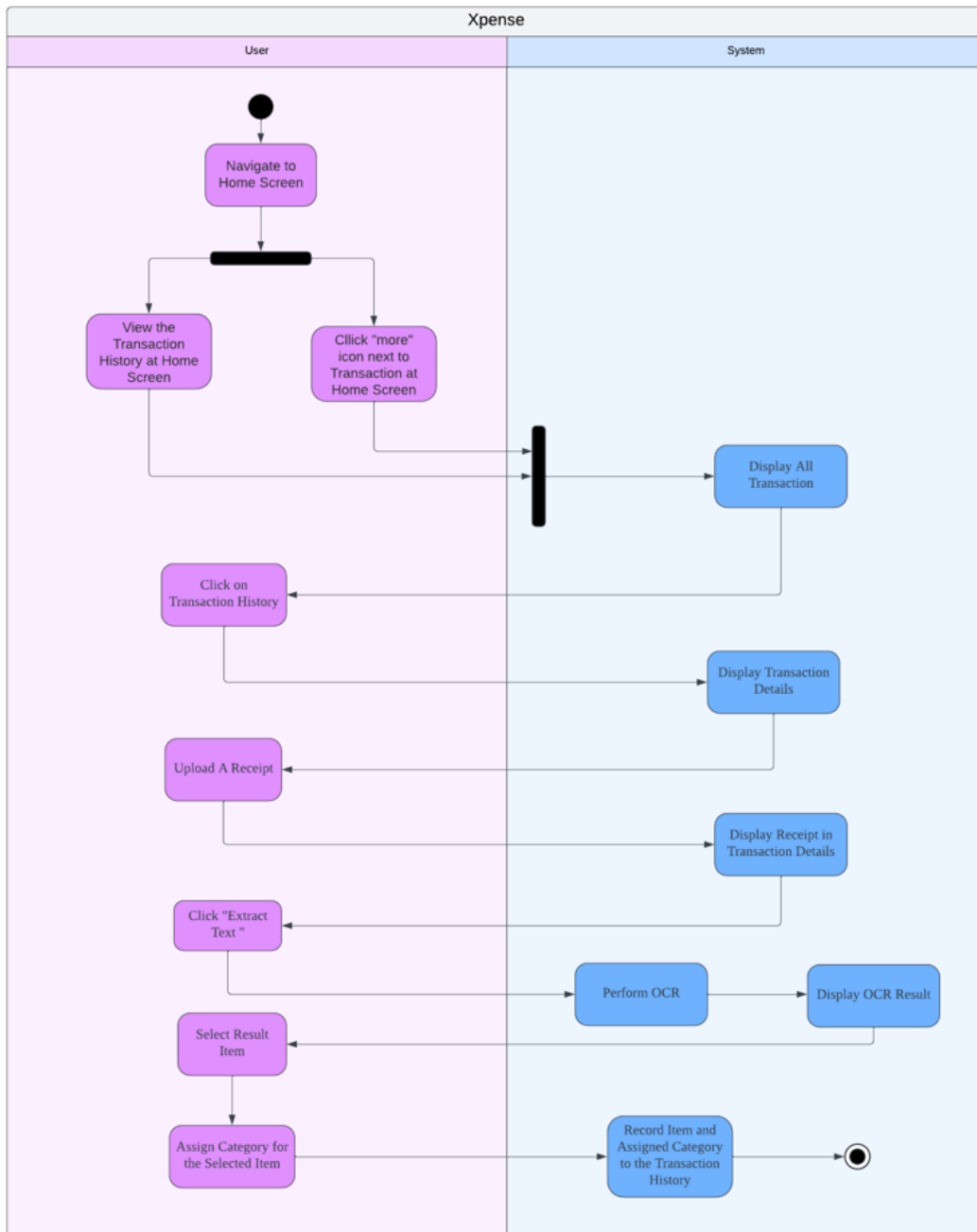


Figure 3.4.28 Activity Diagram for Perform OCR

3.5 Program Development

3.5.1 Front-End and Back-End Development

The mobile application is developed exclusively for client-side operations. For front-end development, Flutter is utilized as the open-source framework with Dart as the programming language. On the backend, Firebase is employed to manage user information and store data in collections such as user profiles, transaction history, and wallet balances for the personal finance management application. Additionally, Stripe is integrated into the application to facilitate payment processing through API calls. Moreover, Asprise is also integrated into the application to handle the OCR process to extract the text in the uploaded receipt.

One of the key advantages of using Flutter is its seamless integration with Firebase, Stripe and Asprise, which allows for efficient communication between the frontend and backend systems. Moreover, Flutter's dependency management system enables developers to utilize various packages and functionalities by simply downloading dependencies.

In terms of application structure, the Dart files located under the screen folder represent all screens within the application as shown in Figure 3.5.1.1. These screens are interconnected and provide a structured and organized navigation flow for users. The “AuthProvider” class is created in the “auth_provider.dart” as shown in Figure 3.5.1.2. This file plays an important role as it handles authentication processes, managing user data, and executing database operations seamlessly.

This Flutter code initializes a mobile application using Firebase and Stripe in the “main.dart”. It starts by importing necessary packages for Firebase, Stripe, and other application functionalities. The main function is asynchronous and ensures that Flutter bindings are initialized. It then configures Firebase with platform-specific options and activates Firebase App Check using the Play Integrity provider for Android. The Stripe API is set up with a publishable key, allowing for payment processing.

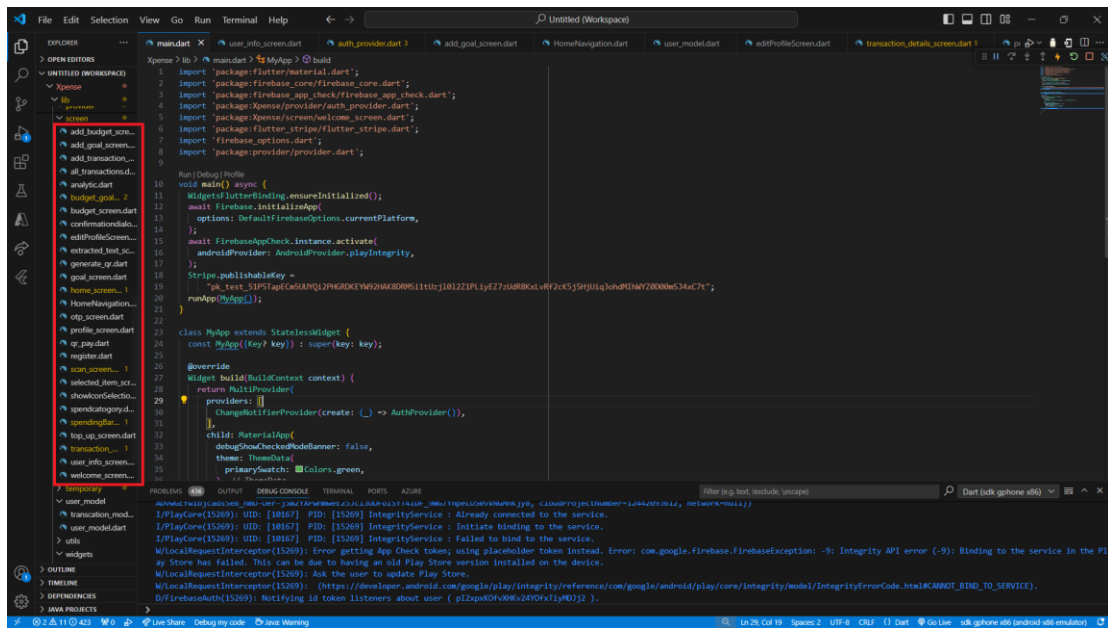


Figure 3.5.1.1 Application's screen in Screen Folder

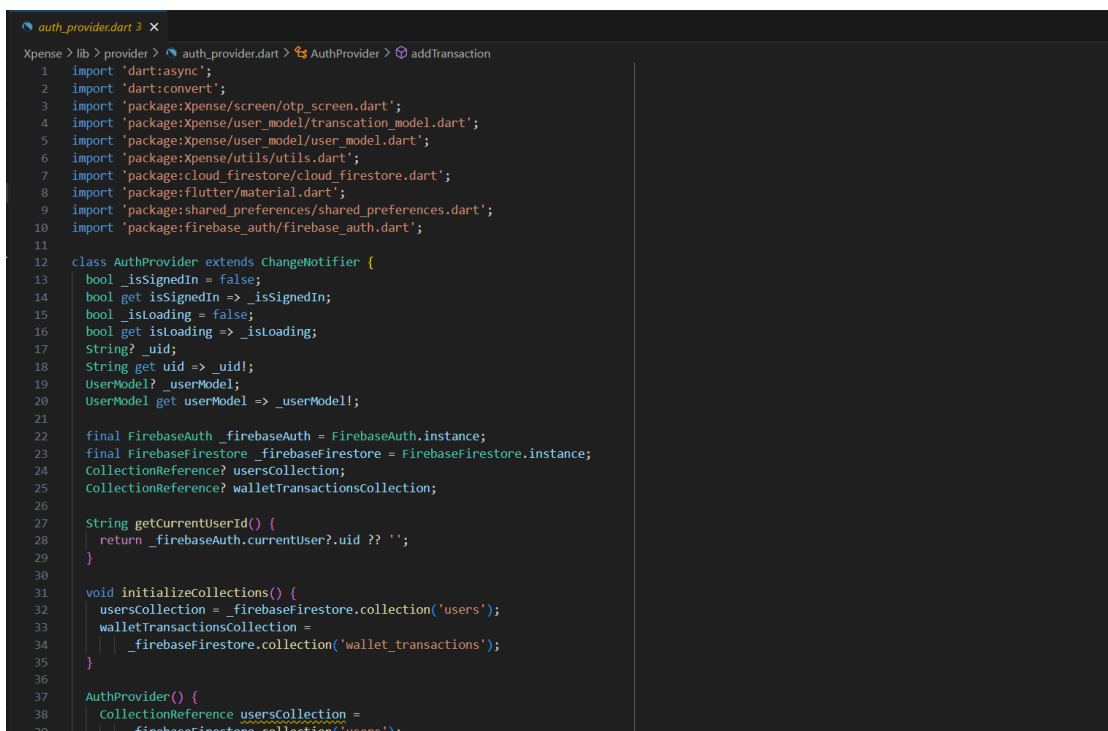


Figure 3.5.1.2 "auth_provider.dart"

```

Xpense > lib > main.dart > MyApp > build
1 import 'package:flutter/material.dart';
2 import 'package:firebase_core/firebase_core.dart';
3 import 'package:firebase_app_check/firebase_app_check.dart';
4 import 'package:Xpense/provider/auth_provider.dart';
5 import 'package:Xpense/screen/welcome_screen.dart';
6 import 'package:flutter_stripe/flutter_stripe.dart';
7 import 'firebase_options.dart';
8 import 'package:provider/provider.dart';
9
10 Run | Debug | Profile
11 void main() async {
12   WidgetsFlutterBinding.ensureInitialized();
13   await Firebase.initializeApp(
14     options: DefaultFirebaseOptions.currentPlatform,
15   );
16   await FirebaseAppCheck.instance.activate(
17     androidProvider: AndroidProvider.playIntegrity,
18   );
19   Stripe.publishableKey =
20     "pk_test_51P5TapECm5UUVQ12PHGRDKEYW92HAK8DRMSi1tUzj1012Z1PLiyEZ7zUdR8KxLvRf2cK5j5HjUiq3ohdMIhWYZ0D00msJ4xC7t";
21   runApp(MyApp());
22 }
23
24 class MyApp extends StatelessWidget {
25   const MyApp({Key? key}) : super(key: key);
26
27   @override
28   Widget build(BuildContext context) {
29     return MultiProvider(
30       providers: [
31         ChangeNotifierProvider(create: (_) => AuthProvider()),
32       ],
33       child: MaterialApp(
34         debugShowCheckedModeBanner: false,
35         theme: ThemeData(
36           primarySwatch: Colors.green,
37         ), // ThemeData
38         home: WelcomeScreen(),
39       ), // MaterialApp
40     ); // MultiProvider
41   }
42 }

```

Figure 3.5.1.3 “main.dart”

3.5.2 User Authentication and Profile Management Development

The user authentication process in the Flutter application begins with the user entering their phone number. Firebase Authentication sends an OTP to the user's provided phone number and the user needs to enter this OTP into the application for verification. If the OTP verification succeeds, the application stores the user's unique ID (UID) obtained during verification. Subsequently, the application checks if this UID corresponds to an existing user in the Firestore database. If the user exists, the application retrieves the user's data from Firestore and marks them as signed in. Additionally, the user's data is saved to SharedPreferences, ensuring that the user remains logged in across sessions without the need for repeated authentication. However, if the UID does not correspond to an existing user, the user needs to proceed to create their profile.

```

void checkSignIn() async {
  final SharedPreferences s = await SharedPreferences.getInstance();
  _isSignedIn = s.getBool("is_signedin") ?? false;
  notifyListeners();
}

Future setSignIn() async {
  final SharedPreferences s = await SharedPreferences.getInstance();
  s.setBool("is_signedin", true);
  _isSignedIn = true;
  notifyListeners();
}

//sign in
void signInWithPhone(BuildContext context, String phoneNumber) async {
  try {
    await _firebaseAuth.verifyPhoneNumber(
      phoneNumber: phoneNumber,
      verificationCompleted: (PhoneAuthCredential phoneAuthCredential) async {
        await _firebaseAuth.signInWithCredential(phoneAuthCredential);
      },
      verificationFailed: (error) {
        throw Exception(error.message);
      },
      codeSent: (verificationId, forceResendingToken) {
        Navigator.push(
          context,
          MaterialPageRoute(
            builder: (context) => OtpScreen(
              verificationId: verificationId,
              phoneNumber: phoneNumber,
            ), // OtpScreen
          ), // MaterialPageRoute
        );
      },
      codeAutoRetrievalTimeout: (verificationId) {},
      timeout: const Duration(seconds: 60),
    );
  } on FirebaseAuthException catch (e) {
    showSnackBar(context, e.message.toString());
  }
}

```

Figure 3.5.2.1 Functions for User Authentication_1

```

void verifyOtp({
  required BuildContext context,
  required String verificationId,
  required String userOtp,
  required Function onSuccess,
}) async {
  _isLoading = true;
  notifyListeners();

  try {
    PhoneAuthCredential creds = PhoneAuthProvider.credential(
      verificationId: verificationId, smsCode: userOtp);

    User? user = (await _firebaseAuth.signInWithCredential(creds)).user!;

    if (user != null) {
      _uid = user.uid;

      onSuccess();
    }
    _isLoading = false;
    notifyListeners();
  } on FirebaseAuthException catch (e) {
    showSnackBar(context, e.message.toString());
    _isLoading = false;
    notifyListeners();
  }
}

Future saveUserDataToSP() async {
  SharedPreferences s = await SharedPreferences.getInstance();
  await s.setString("user_model", jsonEncode(userModel.toMap()));
}

Future getDataFromSP() async {
  SharedPreferences s = await SharedPreferences.getInstance();
  String data = s.getString("user_model") ?? "";
  userModel = UserModel.fromMap(jsonDecode(data));
  _uid = userModel!.uid;
  notifyListeners();
}

Future userSignOut() async {
  SharedPreferences s = await SharedPreferences.getInstance();
}

```

Figure 3.5.2.2 Functions for User Authentication_2

```

Future<bool> checkExistingUser() async {
  DocumentSnapshot snapshot =
    await _firebaseFirestore.collection("users").doc(_uid).get();
  if (snapshot.exists) {
    print("USER EXISTS");
    return true;
  } else {
    print("NEW USER");
    return false;
  }
}

void saveUserDataToFirebase({
  required BuildContext context,
  required UserModel userModel,
  required Function onSuccess,
}) async {
  _isLoading = true;
  notifyListeners();
  try {
    userModel.createdAt = DateTime.now().millisecondsSinceEpoch.toString();
    userModel.phoneNumber = _firebaseAuth.currentUser!.phoneNumber!;
    userModel.uid = _firebaseAuth.currentUser!.phoneNumber!;

    _userModel = userModel;
    //upload data to database
    await _firebaseFirestore
      .collection("users")
      .doc(_uid)
      .set(userModel.toMap())
      .then((value) {
        onSuccess();
        _isLoading = false;
        notifyListeners();
      });
  } on FirebaseAuthException catch (e) {
    showSnackBar(context, e.message.toString());
    _isLoading = false;
    notifyListeners();
  }
}

Future getDataFromFirestore() async {
  await _firebaseFirestore
    .collection("users")

```

Figure 3.5.2.3 Functions for User Authentication_3

The profile management feature is designed to allow users to view and manage their personal information within the application. Once the user successfully signs in, they can access the profile screen which displays key details such as their name and email address. This information was fetched from Firestore. In addition to viewing their information, users are provided with two key options on the profile screen which are edit profile and log out. The "Edit Profile" option allows users to update their personal information such as their name and email. Upon selecting this option, the user is directed to a screen where they can enter new values for their profile information. Once the user submits their changes by tapping the "Update" button, the `_updateUserInfo` function is triggered. If valid inputs are provided, the `AuthProvider` is used to update the user information in Firestore. Additionally, the "Log Out" option allows users to sign out from the application. By tapping the "Log Out" button in the profile screen or the "Log Out" icon at the top right corner of the application, the `userSignOut()` function is triggered, which signs the user out of Firebase and redirects them to the welcome screen. When the user signs out, the user's data saved to `SharedPreferences` will be clear and the user needs to perform authentication again.

```

Future<void> updateUserInfo({
  required String name,
  required String email,
}) async {
  try {
    // Query the users collection for a document with matching phoneNumber
    QuerySnapshot querySnapshot = await _firebaseFirestore
      .collection("users")
      .where('phoneNumber', isEqualTo: _userModel.phoneNumber)
      .limit(1)
      .get();

    if (querySnapshot.docs.isEmpty) {
      throw Exception("User document not found");
    }

    // Get the first (and should be only) matching document
    DocumentReference userDocRef = querySnapshot.docs.first.reference;

    // Update the document
    await userDocRef.update({
      'name': name,
      'email': email,
    });

    // Update local UserModel
    _userModel = _userModel.copyWith(
      name: name,
      email: email,
    );

    // Update SharedPreferences
    await saveUserDataToSP();

    notifyListeners();

    print('User info updated successfully');
  } catch (e) {
    print('Error updating user info: $e');
    print('Current UID: $_uid');
    print('Current Phone Number: ${_userModel?.phoneNumber}');
    throw e;
  }
}

```

Figure 3.5.2.4 Functions for Update User Information in “auth_provider.dart”

```

void _updateUserInfo(AuthProvider ap) async {
  if (_nameController.text.trim().isEmpty ||
      _emailController.text.trim().isEmpty) {
    showSnackBar(context, "Please fill in all fields");
    return;
  }

  try {
    await ap.updateUserInfo(
      name: _nameController.text.trim(),
      email: _emailController.text.trim(),
    );
    showSnackBar(context, "Profile updated successfully");
    Navigator.pushAndRemoveUntil(
      context,
      MaterialPageRoute(builder: (context) => const HomeNavigationScreen()),
      (route) => false,
    );
  } catch (e) {
    showSnackBar(context, "Failed to update profile: $e");
  }
}

```

Figure 3.5.2.5 Functions for Update User Information in “editProfileScreen.dart”

```

Future userSignOut() async {
  SharedPreferences s = await SharedPreferences.getInstance();
  await _firebaseAuth.signOut();
  _isSignedIn = false;
  notifyListeners();
  s.clear();
}

```

Figure 3.5.2.6 Functions for Sign Out

3.5.3 Top-Up Development

The top-up feature is seamlessly integrated into the home screen of the application, allowing users to initiate the top-up process with a single click. When the user taps the top-up button on the home screen, they are redirected to the dedicated top-up screen. They can enter the desired amount they wish to add to their account balance. Upon entering the top-up amount and tapping the "Top Up" button on the top-up screen, the payment process begins. Users are prompted to enter their payment information, typically involving credit card details for payment processing. Once the top-up transaction is completed, the `_handleTopUpSuccess` function is triggered. This function will update the wallet balance in Firestore. This ensures that the user's account balance is accurately maintained and updated in the backend database. Next, the application fetches the updated wallet balance from Firestore. This retrieval of the latest wallet balance ensures that the home screen reflects the most current and accurate account balance to the user.

```

void fetchTransactionHistoryFromFirestore() async {
  try {
    List<TransactionModel> transactionHistory =
      await Provider.of<AuthProvider>(context, listen: false)
        .fetchTransactionHistory();

    setState(() {
      _transactionList = transactionHistory;
    });
  } catch (e) {
    print("Error fetching transaction history: $e");
  }
}

void _fetchUserBalance() async {
  try {
    double balance = await Provider.of<AuthProvider>(context, listen: false)
      .getUserWalletBalance();

    setState(() {
      availableBalance = balance;
    });
  } catch (e) {
    print("Error fetching balance: $e");
  }
}

void _updateBalance(double newBalance) async {
  try {
    await Provider.of<AuthProvider>(context, listen: false)
      .updateWalletBalance(newBalance); // Update balance in Firestore

    setState(() {
      availableBalance = newBalance;
    });
  } catch (e) {
    print("Error updating balance: $e");
  }
}

```

Figure 3.5.3.1 Functions for Update and Fetch Wallet Balance

```

void makePayment() async {
  try {
    if (int.parse(paymentAmount) < 10) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
          content: Text('You must top up a minimum of RM10.'),
          duration: Duration(seconds: 2),
        ), // SnackBar
      );
      return;
    }
    paymentIntent = await createPaymentIntent();
    var gpay = PaymentSheetGooglePay(
      merchantCountryCode: "MYR",
      currencyCode: "MYR",
      testEnv: true,
    );
    await Stripe.instance.initPaymentSheet(
      paymentSheetParameters: SetupPaymentSheetParameters(
        paymentIntentClientSecret: paymentIntent!["client_secret"],
        merchantDisplayName: "Xpense",
        googlePay: gpay,
      ),
    );
    displayPaymentSheet();
  } catch (e) {
    print("Error making payment: $e");
  }
}

```

Figure 3.5.3.2 Functions to Start a Payment for Top Up

```

void _handleTopUpSuccess(double topUpAmount) {
  double currentBalance = widget.availableBalance;
  double newBalance = currentBalance + topUpAmount;
  widget.onBalanceUpdated(newBalance);
  Navigator.pop(context); // Close the top-up screen
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    backgroundColor: const Color(0xffffffff),
    appBar: AppBar(
      elevation: 4,
      centerTitle: true,
      automaticallyImplyLeading: false,
      backgroundColor: const Color(0xffd1f1fd),
      shape: const RoundedRectangleBorder(
        borderRadius: BorderRadius.zero,
      ), // RoundedRectangleBorder
      title: const Text(
        "Top Up",
        style: TextStyle(
          fontWeight: FontWeight.w700,
          fontStyle: FontStyle.normal,
          fontSize: 18,
          color: const Color(0xff000000),
        ), // TextStyle
      ), // Text
      leading: IconButton(
        icon: const Icon(Icons.arrow_back),
        onPressed: () {

```

Figure 3.5.3.3 “_handleTopUpSuccess” Function

3.5.4 Scan QR Development

The QR code scanning feature is developed on the homepage of the application. When users tap on the “Scan” button, it will trigger the QR scanner functionality. This function enables users to scan QR codes associated with merchant names. Users will be directed to a screen where they can enter payment details which are the payment amount, select a spending category, and the option to select an existing budget which is fetched from Firestore. If the user selects add to a budget, the “_updateBudget” function will be called to update the budget total spend amount in the Firestore. The payment confirmation process is managed by the

“_confirmPayment” method. This method validates the payment amount against the user's available balance. After that, this function will update the balance in Firestore and add a record to the transaction history in Firestore.

```

@override
void _onQRViewCreated(QRViewController controller) {
  this.controller = controller;
  bool isNavigating = false; // Flag to track navigation state

  controller.scannedDataStream.listen((scanData) {
    if (!isNavigating) {
      // Check if navigation is already in progress
      setState(() {
        result = scanData;
        if (scanData.code != null) {
          isNavigating =
            true; // Set the flag to true to prevent further navigation
          controller.pauseCamera(); // Pause the camera when a code is scanned

          Navigator.push(
            context,
            MaterialPageRoute(
              builder: (context) => QrPay(qrData: scanData.code!),
            ), // MaterialPageRoute
          ).then(() {
            // Reset the flag and resume the camera when returning from the QrPay screen
            isNavigating = false;
            controller.resumeCamera();
          });
        } else {
          print('Error: Scanned data code is null.');
```

Figure 3.5.4.1 “_onQRViewCreated” Function for QR Code Scanner Set Up

```

void _confirmPayment() async {
  if (isPaymentValid()) {
    try {
      String transactionId = Uuid().v4();
      double paymentAmount = double.parse(amountController.text);
      double currentBalance =
        await Provider.of<AuthProvider>(context, listen: false)
          .getUserWalletBalance();

      if (paymentAmount <= currentBalance) {
        double newBalance = currentBalance - paymentAmount;
        await Provider.of<AuthProvider>(context, listen: false)
          .updateWalletBalance(newBalance);

        String merchantName = widget.qrData.split('.').last;
        String description = detailsController.text.isNotEmpty
          ? detailsController.text
          : "No Payment Details";
        String spendCategory = selectedCategory ?? "Others";

        await Provider.of<AuthProvider>(context, listen: false)
          .addTransactionFromQR(
            transactionId: transactionId,
            transactionDate: DateTime.now(),
            amount: paymentAmount,
            merchantName: merchantName,
            description: description,
            spendCategory: spendCategory,
            source: "QR",
            isExpense: true,
            budgetId: selectedBudget,
          );

        if (selectedBudget != null) {
          _updateBudget(selectedBudget!, paymentAmount);
        }

        ScaffoldMessenger.of(context).showSnackBar(
          const SnackBar(content: Text('Payment successful!')),
        );

        Navigator.pushAndRemoveUntil(
          context,
          MaterialPageRoute(
            builder: (context) => const HomeNavigationScreen(), // MaterialPageRoute
            (route) => false,
```

Figure 3.5.4.2 “_confirmPayment” Function

```

void _updateBudget(String budgetId, double amount) async {
  try {
    Map<String, dynamic> budget =
      budgets.firstWhere((b) => b['docId'] == budgetId);
    double currentSpentAmount = budget['spentAmount'] ?? 0.0;
    double newSpentAmount = currentSpentAmount + amount;

    await Provider.of<AuthProvider>(context, listen: false)
      .updateBudget(budgetId, {
        'spentAmount': newSpentAmount,
      });
  } catch (e) {
    print('Error updating budget: $e');
  }
}

```

Figure 3.5.4.3 “_updateBudget” Function

3.5.5 Analytics Development

The development of the analytics feature is seamlessly integrated into the home screen of the application. The users can access this feature by tapping on the “Analytics” button on the home screen which will then redirect them to the Analytics Screen. This screen provides a comprehensive overview of financial insights through two distinct graphical representations which are a monthly spending bar chart graph and a spending category breakdown in a pie chart. Both graphs are implemented using the “fl_chart” Flutter package which offers robust charting capabilities. The data for these graphs is dynamically fetched from the Firestore Firebase's transaction history to ensure real-time and accurate representations of the user's financial activity. The monthly spending bar graph utilizes the total transaction amount for each month. This has offered a visual timeline of expenditure trends over time. Furthermore, the pie chart breaks down spending categories based on their total amounts has provides a clear and concise breakdown of where the user's money is being allocated.

```

void fetchTransactionHistoryFromFirestore() async {
  try {
    List<TransactionModel> transactionHistory =
      await Provider.of<AuthProvider>(context, listen: false)
        .fetchTransactionHistory();

    List<TransactionModel> expenseTransactions = transactionHistory
      .where((transaction) => transaction.isExpense)
      .toList();

    // Filter transactions by current year
    int currentYear = DateTime.now().year;
    expenseTransactions = expenseTransactions.where((transaction) {
      Timestamp timestamp = transaction.transactionDate;
      int transactionYear = DateTime.fromMillisecondsSinceEpoch(
        timestamp.millisecondsSinceEpoch)
        .year;
      return transactionYear == currentYear;
    }).toList();
    List<double> monthlyTotal = List.filled(12, 0);

    expenseTransactions.forEach((transaction) {
      Timestamp timestamp = transaction.transactionDate;
      int month = DateTime.fromMillisecondsSinceEpoch(
        timestamp.millisecondsSinceEpoch)
        .month;
      double amount = transaction.amount;

      monthlyTotal[month - 1] += amount;
    });

    // Calculate the maximum expense
    maxExpense =
      monthlyTotal.reduce((max, value) => max > value ? max : value);

    // Round up the maxExpense to the nearest 100 for better chart scaling
    maxExpense = (maxExpense / 100).ceil() * 100.0;

    List<BarChartData> updatedData =
      monthlyTotal.asMap().entries.map((entry) {
        int monthIndex = entry.key;
        double totalAmount = entry.value;
        return BarChartData(
          x: monthIndex,
          barRods: [

```

Figure 3.5.5.1 Function to Get Value for Monthly Spending Bar Chart

```

void fetchTransactionHistoryFromFirestore() async {
  try {
    List<TransactionModel> transactionHistory =
      await Provider.of<AuthProvider>(context, listen: false)
        .fetchTransactionHistory();

    // Filter transactions by current year
    int currentYear = DateTime.now().year;
    List<TransactionModel> filteredTransactions =
      transactionHistory.where((transaction) {
        Timestamp timestamp = transaction.transactionDate;
        int transactionYear = DateTime.fromMillisecondsSinceEpoch(
          timestamp.millisecondsSinceEpoch)
          .year;
        return transactionYear == currentYear;
      }).toList();

    setState(() {
      _transactionList = _filterTransactions(filteredTransactions);
    });
  } catch (e) {
    print('Error fetching transaction history: $e');
  }
}

List<TransactionModel> _filterTransactions(
  List<TransactionModel> transactions) {
  return transactions.where((transaction) {
    bool passesCategoryFilter = _selectedCategory == 'All' ||
      transaction.spendCategory == _selectedCategory;
    bool passesMonthFilter = _selectedMonth == 'All' ||
      transaction.transactionDate.toDate().month ==
        _getMonthNumber(_selectedMonth);
    return passesCategoryFilter && passesMonthFilter;
  }).toList();
}

```

Figure 3.5.5.2 Function to Get Value for Spending Category Breakdown Pie Chart

3.5.6 Budget Development

The budget feature is seamlessly integrated into the application which provides users with a clear and intuitive interface. Users can view their set budgets when they navigate to the budget screen. The system allows users to add a new budget by prompting them to input essential details, such as the budget title, amount, colour, category, and budget period. The budget information will then be saved to Firestore. Each budget is displayed on a budget card. The budget card will show details like the budget title, remaining amount, and a progress bar that visually indicates the percentage of the budget used. Additionally, the system offers daily spending advice based on the remaining budget and the number of days left to help users maintain their financial goals. The feature also provides a detailed transaction view for each budget. The users can see the associated transactions which are retrieved from Firestore by clicking on a budget. These transactions are linked to the budget by a matching budget ID. For enhanced analysis, users have the option to download PDF reports of their budget transactions. Users can further manage their budgets by performing actions such as duplicating, editing, or deleting them by referencing the budget's "docId". This ensures that users have complete control over their financial planning and can adjust as needed.

```
String calculateDailyAdvice() {  
  int currentYear = DateTime.now().year;  
  DateTime start = DateFormat('MMM d yyyy').parse('$startDate $currentYear');  
  DateTime end = DateFormat('MMM d yyyy').parse('$endDate $currentYear');  
  
  if (end.isBefore(start)) {  
    end = DateTime(currentYear + 1, end.month, end.day);  
  }  
  
  int daysLeft = end.difference(DateTime.now()).inDays;  
  
  if (daysLeft <= 0) return 'Budget period ended';  
  
  double remainingAmount = totalAmount - spentAmount;  
  double dailyAmount = remainingAmount / daysLeft;  
  return 'You can spend RM${dailyAmount.toStringAsFixed(2)} per day for $daysLeft days';  
}
```

Figure 3.5.6.1 Function to Calculate Daily Advise

```

void _addOrUpdateBudget() {
    final String name = _nameController.text;
    final double amount = double.tryParse(_amountController.text) ?? 0;
    final DateTime endDate = _calculateEndDate();

    if (name.isEmpty || amount <= 0) {
        ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(content: Text('Please enter a valid budget name and amount')),
        );
        return;
    }

    if (selectedCategoryLabel == 'Select Category') {
        ScaffoldMessenger.of(context).showSnackBar(
            SnackBar(content: Text('Please select a category')),
        );
        return;
    }

    final Budget budget = Budget(
        title: name,
        totalAmount: amount,
        spentAmount: _currentSpentAmount,
        startDate: _selectedStartDate,
        endDate: endDate,
        cardColor: selectedColor,
        category: selectedCategoryLabel,
        categoryIcon: selectedCategoryIcon,
        categoryColor: selectedCategoryColor,
    );

    widget.onAddBudget(budget);
}

```

Figure 3.5.6.2 Function to Add or Update Budget

```

Future<void> duplicateBudget(String docId) async {
    try {
        String userId = _firebaseAuth.currentUser!.uid;
        DocumentSnapshot budgetDoc = await _firebaseFirestore
            .collection('users')
            .doc(userId)
            .collection('budgets')
            .doc(docId)
            .get();

        if (budgetDoc.exists) {
            Map<String, dynamic> budgetData =
                budgetDoc.data() as Map<String, dynamic>;
            budgetData['title'] = 'Copy of ${budgetData['title']}';

            // Reset the start date to today and recalculate the end date
            DateTime today = DateTime.now();
            DateTime endDate = DateTime.parse(budgetData['endDate']);
            int durationInDays =
                endDate.difference(DateTime.parse(budgetData['startDate'])).inDays;
            budgetData['endDate'] =
                today.add(Duration(days: durationInDays)).toIso8601String();
            budgetData['startDate'] = today.toIso8601String();

            // Reset the spent amount
            budgetData['spentAmount'] = 0;

            await _firebaseFirestore
                .collection('users')
                .doc(userId)
                .collection('budgets')
                .add(budgetData);

            print('Budget duplicated successfully.');
```

Figure 3.5.6.3 Function to Duplicate Budget

```

Future<void> deleteBudget(String docId) async {
  try {
    String userId = _firebaseAuth.currentUser!.uid;
    await _firebaseFirestore
      .collection('users')
      .doc(userId)
      .collection('budgets')
      .doc(docId)
      .delete();
    print('Budget deleted successfully from Firebase.');
```

Figure 3.5.6.4 Function to Delete Budget

```

Future<List<Map<String, dynamic>>> fetchTransactionsForBudgetOrGoal(
  String id) async {
  try {
    String userId = _firebaseAuth.currentUser!.uid;
    DocumentSnapshot userDoc =
      await _firebaseFirestore.collection('users').doc(userId).get();

    if (userDoc.exists) {
      List<dynamic> allTransactions = userDoc['transactionHistory'];
      List<Map<String, dynamic>> filteredTransactions = allTransactions
        .where((transaction) =>
          transaction['budgetId'] == id || transaction['goalId'] == id)
        .cast<Map<String, dynamic>>()
        .toList();

      return filteredTransactions;
    } else {
      throw Exception('User document not found');
    }
  } catch (e) {
    print('Error fetching transactions for budget/goal: $e');
    throw e;
  }
}

```

Figure 3.5.6.5 Function to Fetch Transaction for Budget

```
Future<void> _downloadPDF(BuildContext context) async {
  final pdf = pw.Document();
  final transactions = await Provider.of<AuthProvider>(context, listen: false)
    .fetchTransactionsForBudgetOrGoal(id);

  pdf.addPage(
    pw.Page(
      build: (pw.Context context) => pw.Column(
        crossAxisAlignment: pw.CrossAxisAlignment.start,
        children: [
          pw.Text(
            'Transactions for $title',
            style: pw.TextStyle(
              fontSize: 28,
              fontWeight: pw.FontWeight.bold,
            ), // pw.TextStyle
          ), // pw.Text
          pw.SizedBox(height: 20),
          pw.Table(
            columnWidths: {
              0: pw.FixedColumnWidth(100),
              1: pw.FixedColumnWidth(100),
              2: pw.FlexColumnWidth(),
            },
            border: pw.TableBorder.all(),
            children: [
              pw.TableRow(
                children: [
                  pw.Padding(
                    padding: const pw.EdgeInsets.all(8.0),
                    child: pw.Text(
                      'Date',
                      style: pw.TextStyle(
                        fontWeight: pw.FontWeight.bold,
                        fontSize: 16,
                      ), // pw.TextStyle
                    ), // pw.Text
                  ), // pw.Padding
                  pw.Padding(
                    padding: const pw.EdgeInsets.all(8.0),
                    child: pw.Text(
                      'Amount',
                      style: pw.TextStyle(
                        fontWeight: pw.FontWeight.bold,
                        fontSize: 16,
```

Figure 3.5.6.6 Function to Download Budget as PDF

3.5.7 Goal Development

The development of the goal feature follows a similar process to the budget feature, offering a user-friendly and integrated experience within the application. On the goal screen, users can easily view their existing goals. When adding a new goal, users are prompted to provide key information such as the goal title, target amount, image icon, and goal period. Once completed, the goal details are saved in Firestore. Each goal is displayed on a goal card which will present important information like the title, duration, and a progress bar that visually tracks the percentage of the goal achieved. The feature also includes a detailed transaction view for each goal which allows users to see all related transactions. These transactions are retrieved from Firestore and are associated with the corresponding goal through a unique “goalID”. For deeper insights, users have the option to download PDF reports of their goal-related transactions. Additionally, users can manage their goals by duplicating, editing, or deleting them. These actions are performed by utilizing the goal's document ID (goal['docId']), giving users full control over their savings objectives.

```

void _addOrUpdateGoal() {
  final goalName = goalNameController.text;
  final goalAmount = goalAmountController.text;

  if (goalName.isNotEmpty && goalAmount.isNotEmpty) {
    final newGoal = {
      'id': widget.goal?['id'] ??
        DateTime.now().millisecondsSinceEpoch.toString(),
      'title': goalName,
      'startDate': '${selectedStartDate.toLocal()}.split(' ')[0]',
      'endDate': '${selectedEndDate.toLocal()}.split(' ')[0]',
      'savedAmount': widget.goal?['savedAmount'] ?? 0.0,
      'goalAmount': double.tryParse(goalAmount) ?? 0.0,
      'icon': selectedIcon.codePoint,
    };

    print('New Goal: $newGoal');

    Navigator.pop(context, newGoal);
  } else {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Please fill in all fields.')),
    );
  }
}

```

Figure 3.5.7.1 Function to Add or Update Goal

```

Future<void> duplicateGoal(String docId) async {
  try {
    String userId = _firebaseAuth.currentUser!.uid;
    DocumentSnapshot goalDoc = await _firebaseFirestore
      .collection('users')
      .doc(userId)
      .collection('goals')
      .doc(docId)
      .get();

    if (goalDoc.exists) {
      Map<String, dynamic> goalData = goalDoc.data() as Map<String, dynamic>;
      goalData['title'] = 'Copy of ${goalData['title']}';
      DateTime today = DateTime.now();
      DateTime endDate = DateTime.parse(goalData['endDate']);
      int durationInDays =
        endDate.difference(DateTime.parse(goalData['startDate'])).inDays;
      goalData['endDate'] = today
        .add(Duration(days: durationInDays))
        .toLocal()
        .toString()
        .split(' ')[0]; // Only keep the date part

      goalData['startDate'] =
        today.toLocal().toString().split(' ')[0]; // Only keep the date part

      // Reset the saved amount
      goalData['savedAmount'] = 0;

      await _firebaseFirestore
        .collection('users')
        .doc(userId)
        .collection('goals')
        .add(goalData);

      print('Goal duplicated successfully.');
```

Figure 3.5.7.2 Function to Duplicate Goal


```

Future<void> deleteGoal(String goalId) async {
  try {
    String userId = _firebaseAuth.currentUser!.uid;
    await _firebaseFirestore
      .collection('users')
      .doc(userId)
      .collection('goals')
      .doc(goalId)
      .delete();
    print('Goal deleted successfully from Firebase.');
```

Figure 3.5.7.3 Function to Delete Goal

```

Future<List<Map<String, dynamic>>> fetchTransactionsForBudgetOrGoal(
  String id) async {
  try {
    String userId = _firebaseAuth.currentUser!.uid;
    DocumentSnapshot userDoc =
      await _firebaseFirestore.collection('users').doc(userId).get();

    if (userDoc.exists) {
      List<dynamic> allTransactions = userDoc['transactionHistory'];
      List<Map<String, dynamic>> filteredTransactions = allTransactions
        .where((transaction) =>
          transaction['budgetId'] == id || transaction['goalId'] == id)
        .cast<Map<String, dynamic>>()
        .toList();

      return filteredTransactions;
    } else {
      throw Exception('User document not found');
    }
  } catch (e) {
    print('Error fetching transactions for budget/goal: $e');
    throw e;
  }
}

```

Figure 3.5.7.4 Function to Fetch Transaction for Goal

```

Future<void> _downloadPDF(BuildContext context) async {
  final pdf = pw.Document();
  final transactions = await Provider.of<AuthProvider>(context, listen: false)
    .fetchTransactionsForBudgetOrGoal(id);

  pdf.addPage(
    pw.Page(
      build: (pw.Context context) => pw.Column(
        crossAxisAlignment: pw.CrossAxisAlignment.start,
        children: [
          pw.Text(
            'Transactions for $title',
            style: pw.TextStyle(
              fontSize: 28,
              fontWeight: pw.FontWeight.bold,
            ), // pw.TextStyle
          ), // pw.Text
          pw.SizedBox(height: 20),
          pw.Table(
            columnWidths: {
              0: pw.FixedColumnWidth(100),
              1: pw.FixedColumnWidth(100),
              2: pw.FlexColumnWidth(),
            },
            border: pw.TableBorder.all(),
            children: [
              pw.TableRow(
                children: [
                  pw.Padding(
                    padding: const pw.EdgeInsets.all(8.0),
                    child: pw.Text(
                      'Date',
                      style: pw.TextStyle(
                        fontWeight: pw.FontWeight.bold,
                        fontSize: 16,
                      ), // pw.TextStyle
                    ), // pw.Text
                  ), // pw.Padding
                  pw.Padding(
                    padding: const pw.EdgeInsets.all(8.0),
                    child: pw.Text(
                      'Amount',
                      style: pw.TextStyle(
                        fontWeight: pw.FontWeight.bold,
                        fontSize: 16,

```

Figure 3.5.7.5 Function to Download Goal as PDF

3.5.8 Manual Input Transaction Development

The manual input feature is smoothly integrated into the application to enable users to record offline transactions such as cash payments. Users can add a new transaction by clicking the floating blue button located at the bottom right of the app interface. Users can specify whether it is an expense or income/savings when adding a transaction. During recording the transaction for expenses, users are prompted to input the details such as the merchant's name, the date of the transaction, the additional notes or details about the transaction and the amount spent. While recording the transaction for income/savings, the process is simpler and requires only the transaction amount, details and date. In addition, users have the option to associate the transaction with a specific budget (for expenses) or a savings goal (for income). This feature allows for more accurate tracking of progress toward financial targets and ensures that all transactions are properly categorized. Before saving the transaction, the application will prompt users to confirm that all entered information is correct because all the transactions cannot be edited later. Once the user confirms the details, the app saves the transaction and updates any linked budget or goal if applicable. A confirmation message is displayed to notify the user that the transaction has been successfully saved. To enhance usability, the app includes a calendar

for selecting dates easily and a dropdown list for quick category selection. Moreover, the app ensures that all required fields are correctly filled in before allowing the user to save the transaction. Finally, the app automatically returns to the home screen after saving. The application will display updated financial information which includes the transaction history and any changes to budgets or goals.

```
Future<void> _fetchBudgetsAndGoals() async {
  List<Map<String, dynamic>> fetchedBudgets =
    await Provider.of<AuthProvider>(context, listen: false).fetchBudgets();
  List<Map<String, dynamic>> fetchedGoals =
    await Provider.of<AuthProvider>(context, listen: false).fetchGoals();
  setState(() {
    _budgets = fetchedBudgets;
    _goals = fetchedGoals;
  });
}
```

Figure 3.5.8.1 Function to Fetch Budget and Goal to Manual Transaction Input Screen

```
void _submitForm() {
  if (_formKey.currentState!.validate()) {
    String transactionId = Uuid().v4(); // Generate a unique ID

    String merchantName =
      _isExpense ? _merchantController.text : "Income/Savings";
    String details = _detailsController.text.isNotEmpty
      ? _detailsController.text
      : "No Payment Details";
    double amount = double.parse(_amountController.text);
    String category = _isExpense ? _selectedCategory! : "Income/Savings";
    bool isExpense = _isExpense;

    String? budgetId = _isSelectedItemBudget ? _selectedBudgetOrGoal : null;
    String? goalId = !_isSelectedItemBudget ? _selectedBudgetOrGoal : null;

    Provider.of<AuthProvider>(context, listen: false)
      .manualAddTransaction(
        transactionId: transactionId,
        merchantName: merchantName,
        details: details,
        amount: amount,
        category: category,
        isExpense: isExpense,
        transactionDate: _selectedDate,
        source: "Manual",
        budgetId: budgetId,
        goalId: goalId,
      )
      .then((_) {
        if (_selectedBudgetOrGoal != null) {
          _updateBudgetOrGoal(_selectedBudgetOrGoal!, amount, isExpense);
        }
      });

    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text("Transaction saved!")),
    );

    Navigator.pushAndRemoveUntil(
      context,
      MaterialPageRoute(builder: (context) => HomeNavigationScreen()),
      (Route<dynamic> route) => false,
    );
  }).catchError((error) {
    ScaffoldMessenger.of(context).showSnackBar(

```

Figure 3.5.8.2 “_submitForm” Function to Save a Transaction

3.5.9 Transaction History Development

The transaction history feature is developed to provide users with a clear overview of their financial activity within the application. Users are presented with a list of recent transactions on the home screen. Each transaction displays key information which includes the date and time, the merchant's name, the spending category, and the transaction amount. Expenses are shown with a minus sign, while income is displayed with a plus sign, making it easy for users to distinguish between the two. There are two primary types of transactions which are QR payments and manual entries. QR payment transactions are displayed in black text, while manual entries are highlighted in blue. This will allow users to easily identify the type of transaction immediately. Apart from that, users can view more detailed information by tapping on a transaction. The information includes the transaction amount, the transaction type (QR or manual entry), transaction details, the exact date and time, and the transaction status. This detailed view provides users with all relevant information about their financial activities. In addition, users are also allowed to upload an image, such as a receipt associated with a particular transaction. The image will upload to Firebase Storage and the image URL will then be saved in the specific transaction in Firestore and then loaded to the transaction details. This feature includes the ability to extract text from the uploaded image using OCR which can be helpful for record-keeping and ensuring accuracy in transaction data. For better financial tracking, users can associate their transactions with specific budgets or savings goals by using the unique "budgetID" or "goalID". This ensures that expenses contribute to budget tracking, while income aligns with savings goals to help users stay on top of their financial targets. An "All Transactions" screen provides users with access to their complete transaction history. Users can filter transactions based on date ranges and categories to make it easier to locate specific transactions and analyze spending patterns. The app automatically updates the user's available balance, ensuring that the displayed balance always reflects the most current state of the user's finances.

```

Future<List<TransactionModel>> fetchTransactionHistory() async {
  try {
    String userId = _firebaseAuth.currentUser!.uid;
    DocumentSnapshot userDoc = await usersCollection!.doc(userId).get();

    if (userDoc.exists) {
      List<dynamic> transactions = userDoc['transactionHistory'];

      // Filter out transactions with null transactionDate
      transactions = transactions
        .where((transaction) => transaction['transactionDate'] != null)
        .toList();

      // Sort transactions in descending order by date
      transactions.sort((a, b) {
        Timestamp dateA = a['transactionDate'] as Timestamp;
        Timestamp dateB = b['transactionDate'] as Timestamp;
        return dateB.compareTo(dateA);
      });

      List<TransactionModel> transactionList = transactions
        .map((transaction) => TransactionModel.fromMap(transaction))
        .toList();

      return transactionList;
    } else {
      throw Exception('User document not found');
    }
  } catch (e) {
    print('Error fetching transaction history: $e');
    throw e;
  }
}

```

Figure 3.5.9.1 Function to Fetch Transaction History

```

void _fetchAllTransactions() async {
  try {
    List<TransactionModel> transactions =
      await Provider.of<AuthProvider>(context, listen: false)
        .fetchTransactionHistory();
    setState(() {
      _allTransactions = transactions;
      _filteredTransactions = transactions;
      _categories = ['All', ..._extractCategories(transactions)];
    });
  } catch (e) {
    print('Error fetching all transactions: $e');
  }
}

```

Figure 3.5.9.2 Function to Fetch Transaction History to All Transaction Screen

```

Future<void> _fetchBudgetsAndGoals() async {
  final authProvider = Provider.of<AuthProvider>(context, listen: false);
  List<Map<String, dynamic>> fetchedBudgets =
    await authProvider.fetchBudgets();
  List<Map<String, dynamic>> fetchedGoals = await authProvider.fetchGoals();
  setState(() {
    _budgets = fetchedBudgets;
    _goals = fetchedGoals;
  });
}

```

Figure 3.5.9.3 Function to Fetch Budget and Goal to Transaction Details Screen

```

Future<void> _pickImage() async {
  setState(() => _isLoading = true);
  final ImagePicker _picker = ImagePicker();
  final XFile? image = await _picker.pickImage(source: ImageSource.gallery);
  if (image != null) {
    final file = File(image.path);
    final imageUrl = await _uploadImageToStorage(file);
    await Provider.of<AuthProvider>(context, listen: false)
      .updateTransactionImage(
        widget.transaction.transactionId,
        imageUrl,
      );
    setState(() {
      _image = file;
      _isLoading = false;
    });
    // Refresh the transaction data
    widget.transaction.imageUrl = imageUrl;
  } else {
    setState(() => _isLoading = false);
  }
}

Future<String> _uploadImageToStorage(File image) async {
  final storageRef = FirebaseStorage.instance
    .ref()
    .child('transaction_images')
    .child('${widget.transaction.transactionId}.jpg');
  final uploadTask = storageRef.putFile(image);
  final snapshot = await uploadTask.whenComplete(() {});
  final downloadUrl = await snapshot.ref.getDownloadURL();
  return downloadUrl;
}

```

Figure 3.5.9.4 Function to Upload Image

```

Future<void> _loadImage() async {
  if (widget.transaction.imageUrl != null &&
    widget.transaction.imageUrl!.isNotEmpty) {
    setState(() => _isLoading = true);
    try {
      final ref =
        FirebaseStorage.instance.refFromURL(widget.transaction.imageUrl!);
      final url = await ref.getDownloadURL();
      final response = await http.get(Uri.parse(url));
      final bytes = response.bodyBytes;
      final directory = await getApplicationDocumentsDirectory();
      final filePath =
        '${directory.path}/transaction_${widget.transaction.transactionId}.jpg';
      final file = File(filePath);
      await file.writeAsBytes(bytes);
      setState(() {
        _image = file;
        _isLoading = false;
      });
    } catch (e) {
      print('Error loading image: $e');
      setState(() => _isLoading = false);
    }
  }
}

```

Figure 3.5.9.5 Function to Load the Image

```

void _updateTransactionWithBudgetOrGoal() async {
  if (_selectedBudgetOrGoal != null) {
    final authProvider = Provider.of<AuthProvider>(context, listen: false);
    try {
      await authProvider.updateTransactionBudgetOrGoal(
        widget.transaction.transactionId,
        _selectedBudgetOrGoal!,
        widget.transaction.isExpense,
      );
      // Update the transaction amount in the budget or goal
      if (widget.transaction.isExpense) {
        await authProvider.updateBudget(_selectedBudgetOrGoal!, {
          'spentAmount': FieldValue.increment(widget.transaction.amount),
        });
      } else {
        await authProvider.updateGoal(_selectedBudgetOrGoal!, {
          'savedAmount': FieldValue.increment(widget.transaction.amount),
        });
      }
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text("Transaction updated successfully")),
      );
      Navigator.pop(context, true);
    } catch (e) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text("Failed to update transaction: $e")),
      );
    }
  }
}

```

Figure 3.5.9.6 Function to Update the Transaction with Budget or Goal

3.5.10 Receipt OCR Development

The Receipt OCR feature is seamlessly integrated into the application which allows users to upload a receipt. After the image is uploaded, users can tap a button labelled "Extract Text" to initiate text recognition. This action will call the Aspire OCR API. The Asprise OCR service analyzes the uploaded image and identifies all the text within the image. It is a highly efficient tool that can read and extract content from a receipt swiftly. Once the OCR process is complete, the service returns the recognized text in a structured JSON format. This JSON data includes key information such as item descriptions, prices, and other relevant details. Once the app receives the JSON data, it processes the information and displays it on a new screen. On this screen, the text is broken down line by line with items that resemble descriptions and prices displayed as individual checkboxes. Users can then go through the list and select the specific items they wish to keep track of. This process makes it easy to record expenses without having to manually enter each item. This feature significantly streamlines the expense-tracking process by allowing users to quickly and efficiently extract data from receipts and add the relevant items to their expense tracker.

```

static Future<Map<String, dynamic>> _processImageInIsolate(
  String imagePath) async {
  try {
    String receiptOcrEndpoint = "https://ocr.asprise.com/api/v1/receipt";
    var request =
      http.MultipartRequest('POST', Uri.parse(receiptOcrEndpoint));

    request.fields['api_key'] = 'TEST';
    request.fields['recognizer'] = 'auto';
    request.fields['ref_no'] =
      'ocr_flutter_${DateTime.now().millisecondsSinceEpoch}';

    request.files.add(await http.MultipartFile.fromPath('file', imagePath));

    var streamedResponse =
      await request.send().timeout(Duration(seconds: 30));
    var response = await http.Response.fromStream(streamedResponse);

    print('Response status: ${response.statusCode}');
    print('Response body: ${response.body}');

    if (response.statusCode == 200) {
      var jsonResponse = json.decode(response.body);
      String ocrText = jsonResponse['receipts']?[0]?['ocr_text'] ?? '';

      if (ocrText.isEmpty) {
        print('OCR text is empty. Full response: $jsonResponse');
        return {'success': false, 'error': 'OCR text is empty'};
      } else {
        return {'success': true, 'text': ocrText};
      }
    } else {
      return {
        'success': false,
        'error':
          'Failed to process image. Status code: ${response.statusCode}'
      };
    }
  } catch (e) {
    print('Error in isolate: $e');
    return {'success': false, 'error': e.toString()};
  }
}

```

Figure 3.5.10.1 Function to Call the Asprise API

```

I/flutter (15269): Response body: {
I/flutter (15269):   "ocr_type": "receipts",
I/flutter (15269):   "request_id": "P_175.144.109.96_m0vfp0ui_eth",
I/flutter (15269):   "ref_no": "ocr_flutter_1725912418990",
I/flutter (15269):   "file_name": "transaction_daf995af-7ca5-4628-b56e-af85354899fe.jpg",
I/flutter (15269):   "request_received_on": 1725912423306,
I/flutter (15269):   "success": true,
I/flutter (15269):   "image_width": 808,
I/flutter (15269):   "image_height": 1590,
I/flutter (15269):   "image_rotation": 0,
I/flutter (15269):   "recognition_completed_on": 1725912423667,
I/flutter (15269):   "receipts": [ {
I/flutter (15269):     "merchant_name": "4200 SAMPLE",
I/flutter (15269):     "merchant_address": "SELANGOR.",
I/flutter (15269):     "merchant_phone": "+60 7134",
I/flutter (15269):     "merchant_website": null,
I/flutter (15269):     "merchant_tax_reg_no": null,
I/flutter (15269):     "merchant_company_reg_no": null,
I/flutter (15269):     "region": null,
I/flutter (15269):     "mall": null,
I/flutter (15269):     "country": "MY",
I/flutter (15269):     "receipt_no": "180124/10100/01/39729",
I/flutter (15269):     "date": "2024-01-18",
I/flutter (15269):     "time": "18:46",
I/flutter (15269):     "items": [ {
I/flutter (15269):       "amount": 5,
I/flutter (15269):       "category": null,
I/flutter (15269):       "description": "NO LOT 218 & 219, JALANKS",
I/flutter (15269):       "flags": "",
I/flutter (15269):       "qty": null,
I/flutter (15269):       "remarks": null,
I/flutter (15269):       "tags": null,
I/flutter (15269):       "unitPrice": null
I/flutter (15269):     }, {
I/flutter (15269):

```

Figure 3.5.10.2 Extract Text Result in JSON Format


```

void _showSelectedItems() {
  List<Map<String, String>> selectedItems = [];
  for (int i = 0; i < lines.length; i++) {
    if (checkedItems[i]) {
      final match = RegExp(r'(.+?)\s+([\d.-]+)$').firstMatch(lines[i]);
      if (match != null) {
        selectedItems.add({
          'description': match.group(1) ?? '',
          'price': match.group(2) ?? '',
        });
      }
    }
  }
  Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) => SelectedItemsScreen(selectedItems: selectedItems),
    ), // MaterialPageRoute
  );
}

```

Figure 3.5.10.3 Function to Show the Selected Item

```

void _saveTransactions(AuthProvider authProvider) async {
  for (int i = 0; i < widget.selectedItems.length; i++) {
    String transactionId = const Uuid().v4();
    String description = widget.selectedItems[i]['description'] ?? '';
    double amount =
      double.tryParse(widget.selectedItems[i]['price'] ?? '0') ?? 0.0;
    String category = selectedCategories[i] ?? 'Unknown';
    bool isExpense = true;
    DateTime transactionDate = DateTime.now();
    String source = 'Manual';

    await authProvider.manualAddTransaction(
      transactionId: transactionId,
      merchantName: description,
      details: description,
      amount: amount,
      category: category,
      isExpense: isExpense,
      transactionDate: transactionDate,
      source: source,
    );
  }

  Navigator.pushAndRemoveUntil(
    context,
    MaterialPageRoute(builder: (context) => HomeNavigationScreen()),
    (Route<dynamic> route) => false,
  );
}

```

Figure 3.5.10.4 Function to Save the Selected Item to Transaction

3.6 Summary

The use case diagram and description for both the user and administrator are included to provide a clear overview of all the functions that can be performed within the system. The system development process and the functionality of the code are explained in detail.

Chapter 4 Methodology and Tools

4.1 Overview

The methodology and the technologies to develop the project will be proposed in this chapter. The system requirement which includes the hardware and software specifications, and the methodology will be explained. The timeline of the project is proposed to ensure the project is delivered on time.

4.2 Methodology

Rapid Application Development (RAD) methodology is proposed for this project. According to Kissflow [15], RAD is a flexible approach to software development that focuses on prototyping and rapid feedback rather than specific planning. RAD enables developers to rapidly iterate and update software without having to start from scratch. This ensures the outcome is more focused on quality and meets the requirements of the end-user so it is suitable to develop the application. Figure 4.2.1 illustrates the phase of RAD methodology from the Quixy Editorial Team [16].

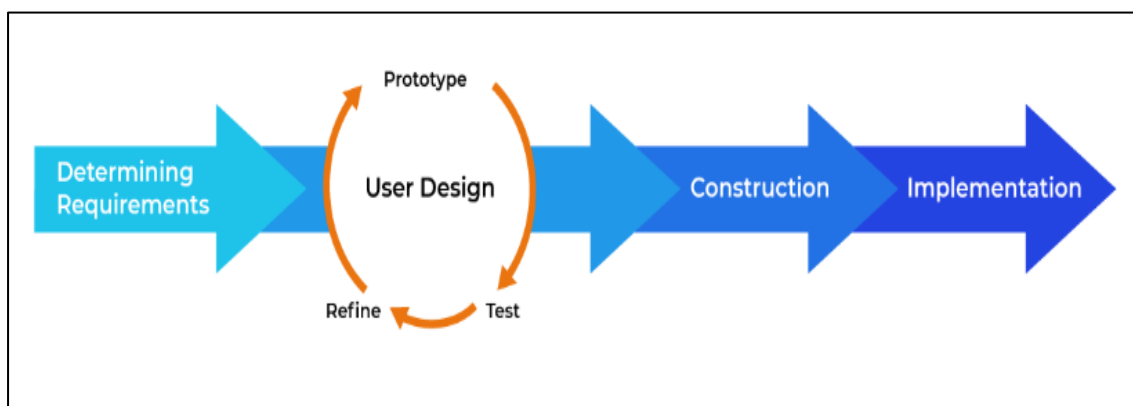


Figure 4.2.1 The Phase of Methodology [16]

4.2.1 Determining Requirements Phase

In the determining requirement phases, the primary goal is to have a comprehensive understanding of the requirements for the personal finance management application. This phase involves conducting research on existing personal finance management applications to understand their features and functionalities. After understanding the problems, the project's requirements and objectives will be defined. The specific functionalities and features will be defined in this phase such as user authentication, transaction categorization criteria, budget calculation logic, and goal-setting parameters that will be incorporated into the application. It also includes identifying technical requirements such as integration with external financial services and machine learning libraries. The system requirements including hardware and software will be determined in this phase.

4.2.2 User Design Phase

The module prototypes of the application will be built during this phase. The module prototypes include the user authentication and profile management module, virtual wallet module, transaction tracking and categorization module, income or saving recording module, manual transaction input module, budget management and calculation module, goal-setting and tracking module, analytics module as well as receipt OCR module. Then the user interface and the functionalities will be developed by using Dart and Flutter. During this phase, each prototype will be tested to ensure it meets the requirements. The program will then be tested, and all the bugs will be worked out in an iterative process. This process will keep repeating until the prototypes reach a satisfactory design.

4.2.3 Construction Phase

The Construction Phase involves developing the core functionality of the application based on the requirements and user interface designs. This phase takes the prototypes from the design phase and converts the prototypes into working models. This phase includes the coding development for the application, which includes the user authentication and profile management module, virtual wallet module, transaction tracking and categorization module, income or saving recording module, manual transaction input module, budget management and calculation module, goal-setting and tracking module, analytics module as well as receipt OCR module. Then create a database for storing transaction records. The coding, testing and development work together to ensure the personal finance management application works

smoothly. The feedback from users will be collected for any suggestions, alterations or new ideas until it satisfies the users' expectations.

4.2.4 Implementation Phase

The implementation phase involves integrating the various modules, conducting thorough testing, preparing for deployment, and monitoring post-launch performance.

In this phase, it will ensure seamless integration between modules, enabling them to work cohesively within the application. It will also perform unit testing for each module to identify and resolve bugs. The testing will be conducted thorough testing, including unit testing, integration testing, and user acceptance testing, to identify and address any issues in the personal finance management application. After that, the application is ready to launch. It will continuously monitor the application's performance post-launch and collect user feedback for future improvements.

4.3 Tools to Use

i. Hardware Requirements

Table 4.3.1 Hardware Component and Requirement for Application Development

Component	Requirements
Model	MSI Katana GF66 12UD
Processor	Intel® Core™ i7-12650H @ 2.70 GHz
Operating System	Windows 11 64-bit
Graphic	NVIDIA GeForce RTX 3050
Memory	16GB DDR4 3200MHz
Storage	455 GB SSD
Input	Keyboard and mouse

ii. Software Requirements

Table 4.3.2 Software Component and Requirement for Application Development

Component	Requirements
Tools	<p><u>Visual Studio Code</u></p> <p>Visual Studio Code is a lightweight but powerful source code editor developed by Microsoft. It is highly customizable and supports various programming languages and extensions. It is a popular choice for web and software development due to its versatility and active community support [17].</p>
	<p><u>Android Studio</u></p> <p>Android Studio is the official integrated development environment for Android app development. It offers tools for designing user interfaces, writing code, and debugging Android applications. It is built on top of IntelliJ IDEA and supports Java, Kotlin, and more [18].</p>
	<p><u>Firebase</u></p> <p>Google Firebase is a suite of cloud-based development tools designed to assist mobile app developers in creating, deploying, and scaling their applications [19].</p>
Languages, Libraries and Frameworks	<p><u>Java</u></p> <p>Java is a robust and platform-independent programming language often used for building cross-platform mobile apps, web applications, enterprise software, and more. It is known for its portability and strong community support [20].</p>
	<p><u>Kotlin</u></p> <p>Kotlin is a modern programming language developed by JetBrains and officially supported for Android app development. It is designed to be concise, expressive, and fully interoperable with Java, making it an excellent choice for Android development [21].</p>
	<p><u>Flutter</u></p>

	Flutter is an open-source technology developed by Google that enables developers to build mobile, desktop, and web applications using a single codebase [22].
	<p><u>Dart</u></p> <p>Dart is an open-source, object-oriented programming language with a C-style syntax and serves as a versatile tool for various programming tasks [23].</p>

4.4 Timeline

4.4.1 Overview

The planning of the project's timeline is alignment with the methodology of the project. During the determining requirement phases, the primary goal is to have a comprehensive understanding of the requirements for the personal finance management application. This phase involves researching existing personal finance management applications to understand their features and functionalities. After understanding the problems, the project's requirements and objectives will be defined. The specific functionalities and features will be defined in this phase such as user authentication, transaction categorization criteria, budget calculation logic, and goal-setting parameters that will be incorporated into the application. It also includes identifying technical requirements such as integration with external financial services and machine learning libraries. The system requirements including hardware and software will be determined in this phase.

During the user design and construction phase, the user authentication module is completed first. This is the most important module to authenticate the users for them to access the application. After that, the virtual wallet module is done which includes, top-up, and scanning QR code. Furthermore, this phase also includes the Final Year Project 1 Report preparation and presentation.

In Final Year Project 2, an income or saving recording module, manual transaction input module, budget management and calculation module, goal-setting and tracking module, analytics module as well as receipt OCR module will be developed. During the implementation phase, testing for every module will be conducted to ensure seamless integration between

modules, enabling them to work cohesively within the application. Lastly, the Final Year Project 2 Report preparation and presentation will be conducted.

4.4.2 Gantt Chart

Task Name	Duration(week)	Start Date	End Date	Week																											
				Project 1														Project 2													
				1	2	3	4	5	6	7	8	9	10	11	12	13	14	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Requirement Planning Phase																															
Review Proposal of Project	1	29/1/2024	4/2/2024	█																											
Define Problem Statement	1	29/1/2024	4/2/2024	█																											
Identify Objective and Project Scopes	1	29/1/2024	4/2/2024	█																											
Literature Review	1	29/1/2024	4/2/2024	█																											
Plan Project Timeline	1	29/1/2024	4/2/2024	█																											
Plan Tools	1	29/1/2024	4/2/2024	█																											
Project Use Case and Description	2	5/2/2024	18/2/2024		█	█																									
User Design and Development Phase																															
User Authentication Module	3	19/2/2024	10/3/2024			█	█	█																							
Analytics Module	3	19/2/2024	10/3/2024			█	█	█																							
Virtual Wallet Module	4	11/3/2024	7/4/2024					█	█	█	█																				
Final Year Project 1 Report Preparation	2	8/4/2024	21/4/2024																												
Final Year Project 1 Presentation	2	22/4/2024	5/5/2024																												
Budget Management and Calculation Module	2	17/6/2024	30/6/2024																												
Profile Management Module	2	17/6/2024	30/6/2024																												
Goal-setting and Tracking module	2	17/6/2024	30/6/2024																												
Income and Recording Module	2	17/6/2024	30/6/2024																												
Manual Transaction Input	5	17/6/2024	21/7/2024																												
Transaction Tracking and Categorization Module	5	17/6/2024	21/7/2024																												
Receipt OCR Module	5	1/7/2024	4/8/2024																												
Improve Overall Design and Feature	3	29/7/2024	18/8/2024																												
Finalise Product and Implementation Phase																															
Project Testing	3	19/8/2024	8/9/2024																												
Final Year Project 2 Preparation	2	26/8/2024	8/9/2024																												
Final Year Project 2 Presentation	2	9/9/2024	22/9/2024																												

Figure 4.4.2.1 Project Timeline Gantt Chart

4.5 Summary

This chapter discusses the methodology utilized for this project, which is the RAD methodology. The RAD methodology is chosen for its ability to reduce project risks by incorporating testing in nearly every phase and the error and bugs can be detected at the early stage. Additionally, the hardware used to develop the application is defined in this chapter. Furthermore, this chapter includes use case diagrams and detailed descriptions for users and administrators. This will provide a comprehensive overview of the application's functionalities. The development process of the application is also outlined in this chapter. A well-planned timeline is illustrated in a Gantt chart to ensure project progress without delays.

Chapter 5 Implementation

5.1 Overview

The modules include user authentication and profile management module, virtual wallet module, transaction tracking and categorization module, income or saving recording module, manual transaction input module, budget management and calculation module, goal-setting and tracking module, analytics module as well as receipt OCR module are developed in the application. The application is named Xpense and it features a light green theme and is designed to be fully mobile responsive. There is a consistent bottom navigation bar that enables users to easily navigate between different screens within the application. The top navigation bar includes a back icon across all screens within the application. This app bar will enable users to navigate to the previous tab or screen. There is also a log out icon located at the app bar for the main screen to allow users to sign out of the application. In addition, there is also a consistent floating blue button which allows users to manually input the transactions. Users have access to all the screens and the features of the application.

5.2 User Authentication Screens

When a user launches the application, the users are greeted with a welcoming screen prompting them to begin by clicking on the "Get Started" button. After that, the user will be directed to the phone number entry page. The user is required to input their phone number. Once the phone number is submitted, an OTP is sent to the user's phone for verification purposes. Following this, users are directed to the verification screen where they enter the received OTP to authenticate their phone number. The system then performs a check to determine if the user already exists within the system. If the user is an existing user, they are swiftly redirected to the home screen. In the case of a new user, they are guided to create a profile screen. They are prompted to provide essential information such as their name and email address. Once the profile creation process is complete, they are seamlessly granted access to the home screen.

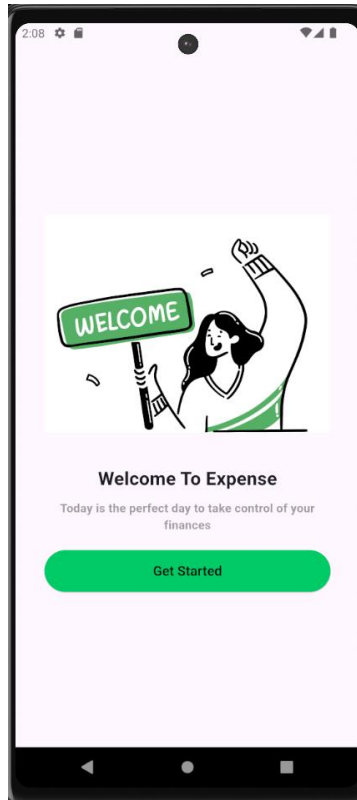


Figure 5.2.1 Welcome Screen

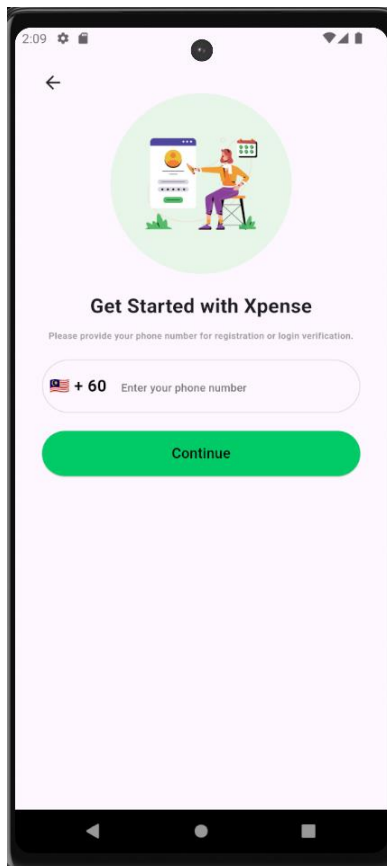


Figure 5.2.2 Phone Entry Screen for Login or Register

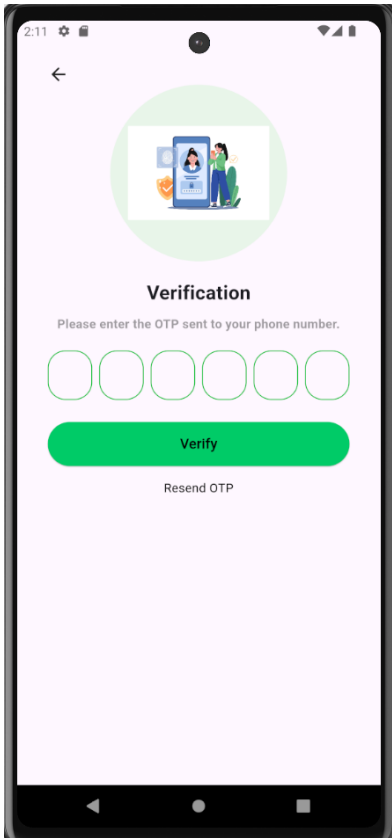


Figure 5.2.3 Verification Screen

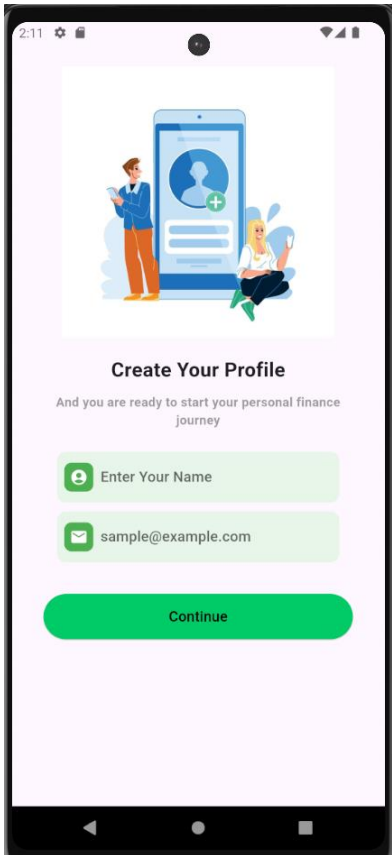


Figure 5.2.4 Create Profile Screen

5.3 Home Screen

After a user login successfully to the application, the users are directed to the homepage. The application features a consistent bottom navigation bar that enables users to easily navigate between different screens. Additionally, a sign-out icon is located at the top right corner which allows users to log out. Furthermore, a blue floating action button is consistently available to provide users with quick access to manually input transactions. There are three sections on the homepage. The first section displays the virtual wallet which provides users with immediate visibility of their current balance. This serves as a quick reference point for their available funds within the application. The second section titled "Operations," presents users with three key functionalities. The first is the "Top Up" feature which enables users to add funds to their wallet seamlessly. The second functionality is "Scan QR" which can facilitate quick and efficient QR code scanning for transactions. Lastly, the "Analytics" feature empowers users with insightful data visualization regarding their spending patterns and financial trends. The third and final section is dedicated to displaying the user's transaction history. This section provides a comprehensive overview of past transactions, including details such as the transaction date, merchant name, category of expenditure, and the corresponding transaction amount. There are two primary types of transactions which are QR payments and manual entries. QR payment transactions are displayed in black text, while manual entries are highlighted in blue, allowing users to easily identify the type of transaction at a glance.

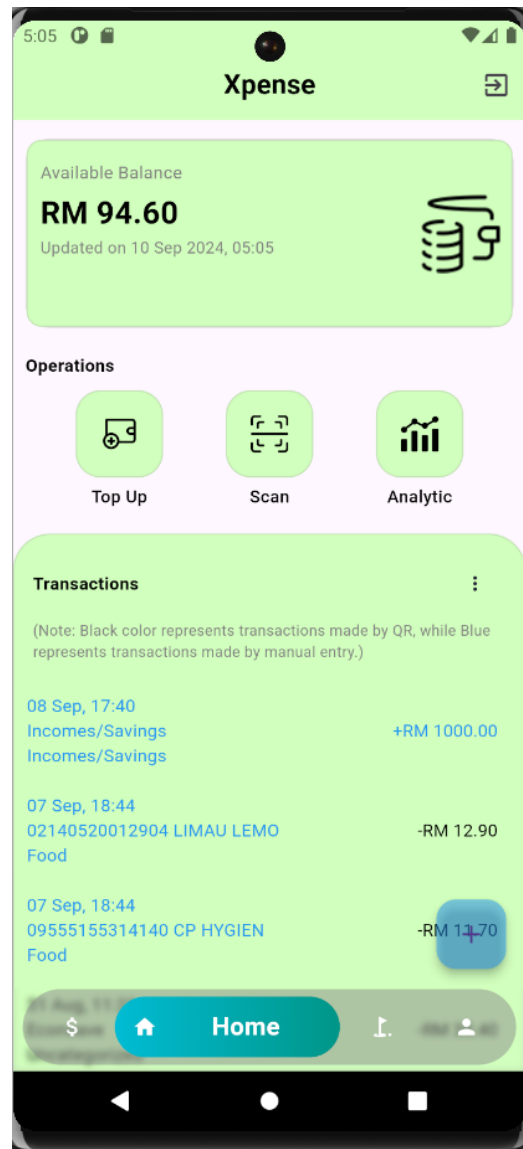


Figure 5.3.1 Home Screen

5.4 Top Up Screen

When the user clicks the "Top Up" button on the home screen, users are directed to the top-up page. The user needs to input the desired amount for topping up their wallet. The preset buttons are available which allows users to quickly autofill specific amounts. The system enforces that only positive integer amounts can be entered, and the minimum top-up amount is set at RM10. When the user enters the top-up amount and clicks the "Top Up" button, the payment process will be initiated. The user is prompted to input their credit card information to complete the payment securely. Upon successful payment completion, the system redirects the user back to the home screen. Their wallet balance is instantly updated to reflect the top-up amount to ensure that users can immediately access and utilize their updated funds within the application.

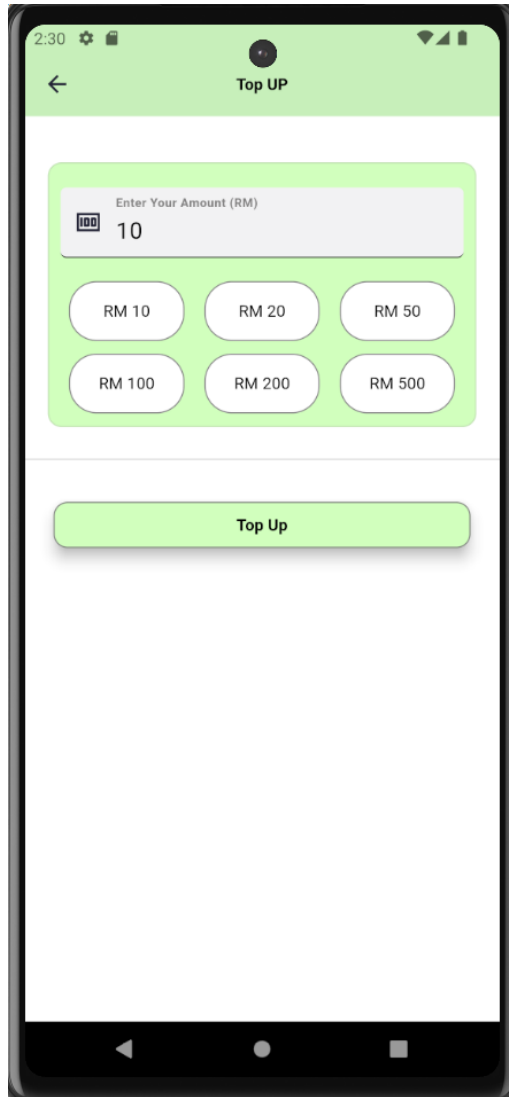


Figure 5.4.1 Top Up Screen

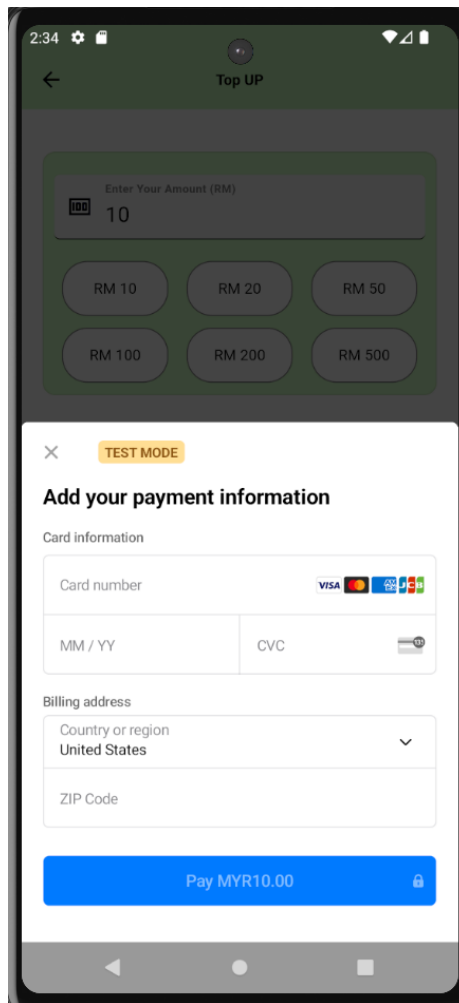


Figure 5.4.2 Payment Process by Stripe

5.5 Scan QR Code Screen

When the user clicks the "Scan" button on the home screen, they are directed to a scanner feature that enables them to scan a QR code to initiate a transaction. Upon successful scanning of the QR code, the user is redirected to the QR payment page. They can input the transaction amount, and payment details (optional), and select a category for the transaction. The system ensures that both the payment amount and category selection are completed before enabling the user to click the "Confirm" button. Moreover, users have the option to select an existing budget from the list if one is available. When users select an existing budget, the transaction will be associated with that budget and automatically added to the budget's transaction history. After the payment process is completed, the user is redirected back to the home screen and their wallet balance is updated to reflect the transaction.

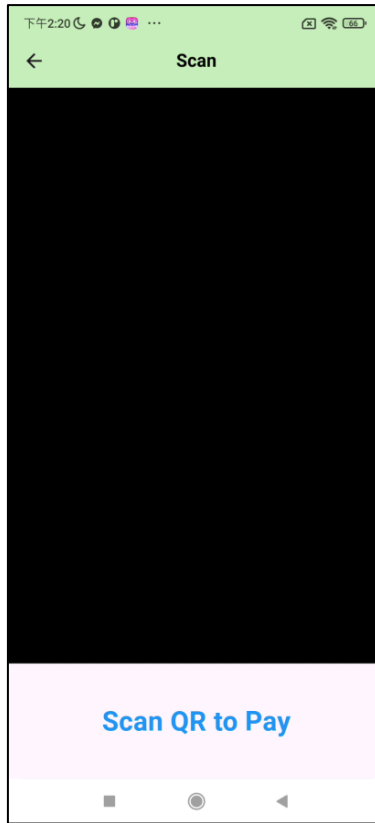


Figure 5.5.1 QR Code Scanner Screen

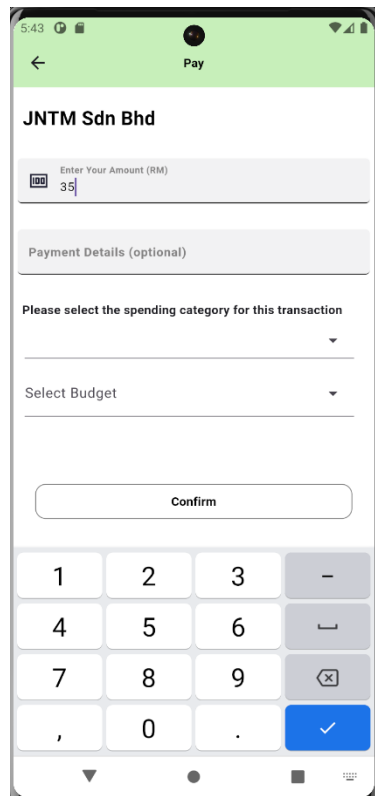


Figure 5.5.2 Confirm Button Lock

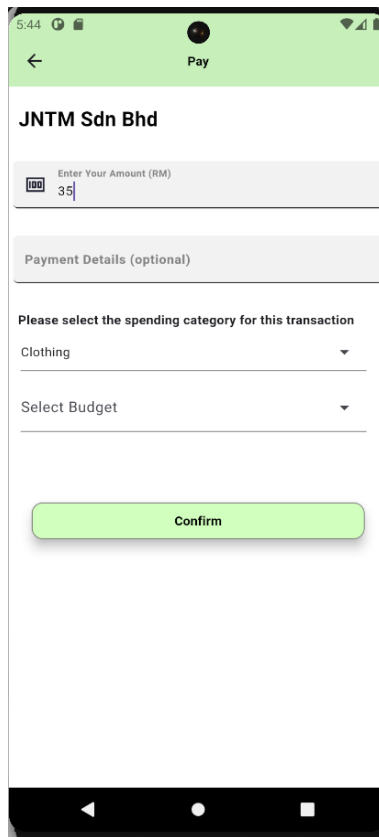


Figure 5.5.3 Confirm Button Unlock

5.6 Analytics Screen

When the user clicks on the "Analytics" button on the home screen, they are redirected to the Analytics page. This page offers a comprehensive view of spending analytics, presenting two key visualizations. The first visualization is a monthly spending bar chart which represents the total transaction amounts of each month. This graph provides a clear overview of spending trends and patterns over time. Additionally, the Analytics page includes a spending category breakdown presented in a pie chart format. This pie chart illustrates the distribution of total expenditures across specific spending categories. Each segment of the pie represents a distinct spending category, allowing users to easily identify where their money is being allocated. Additionally, users can filter the pie chart by month or by category, enabling them to analyze their spending more precisely.

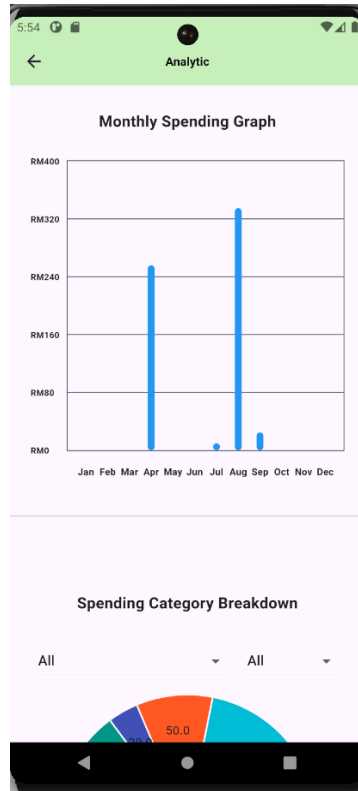


Figure 5.6.1 Monthly Spending Bar Chart Graph

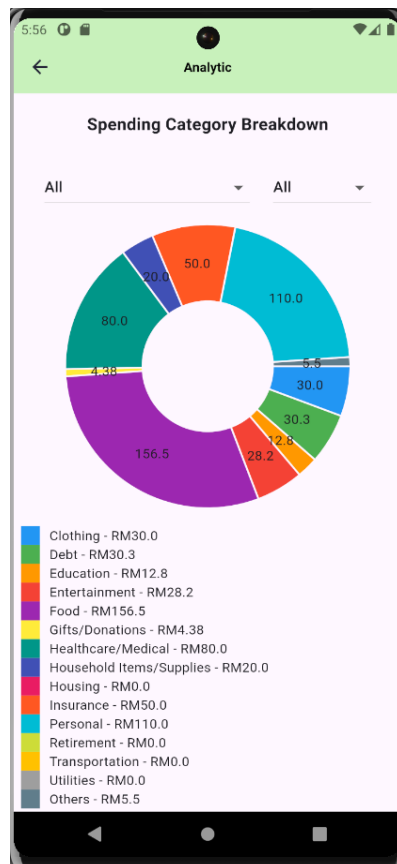


Figure 5.6.2 Spending Category Breakdown Pie Chart

5.7 Budget Screen

When the user clicks on the "Budget" option in the bottom navigation bar, they will be directed to the budget screen. In the budgets screen, users can view their budgets displayed as cards and show key information such as the budget title with its category, the remaining amount, a progress bar indicating the amount spent, and daily spending advice. Users can add a new budget by clicking the "+" icon at the top right corner next to "Budget" or within the grey container. This action redirects them to the "Add Budget" screen, where they are prompted to fill in details like the title, budget amount, duration, colour, and category. Once the information is entered correctly, the budget is added to the budget screen. The users can click the "Edit" icon next to the budget title, which takes them to the "Edit Budget" screen. After making changes and clicking "Update Budget," the updates will be reflected on the budget screen. When users click the "Duplicate" icon, a copy of the budget will be created. The spent amount will be reset, and the start date updated to the current date, setting one month by default. In addition, the user can delete the budget by clicking the "Delete" icon next to the "Edit" icon. Moreover, users can view the transactions associated with the budget by simply tapping on that budget card. Additionally, they can click on the "Download" icon to export the budget transaction history as a PDF.

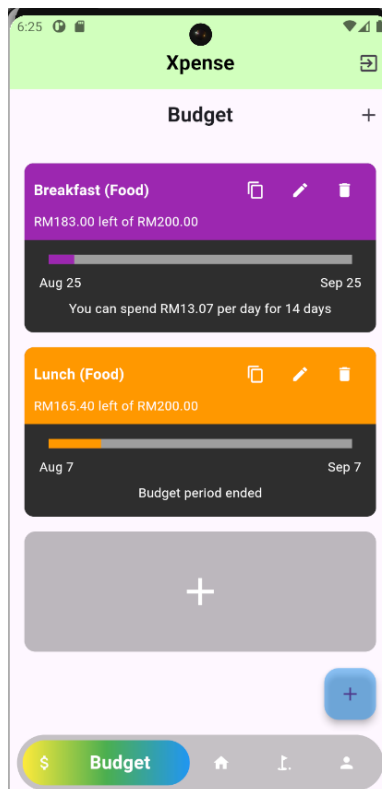


Figure 5.7.1 Budget Screen

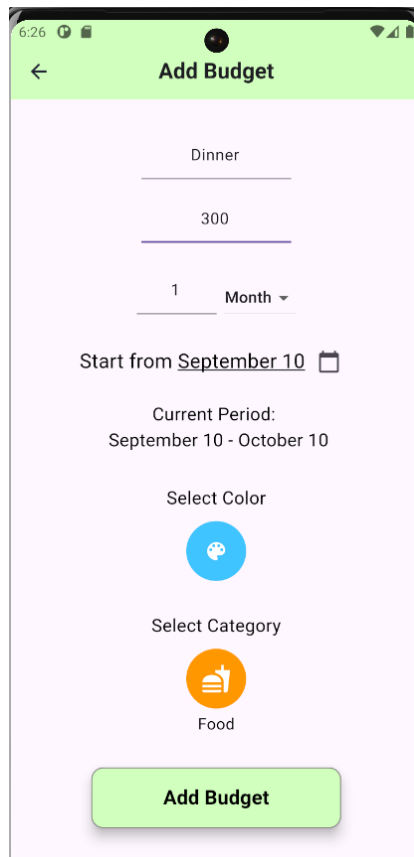


Figure 5.7.2 Add Budget Screen

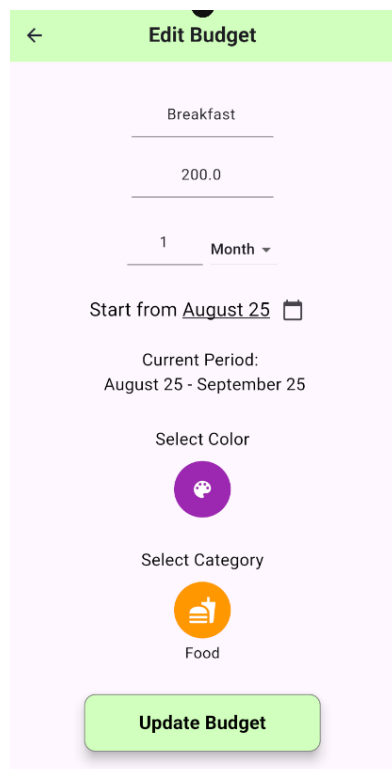


Figure 5.7.3 Edit Budget Screen

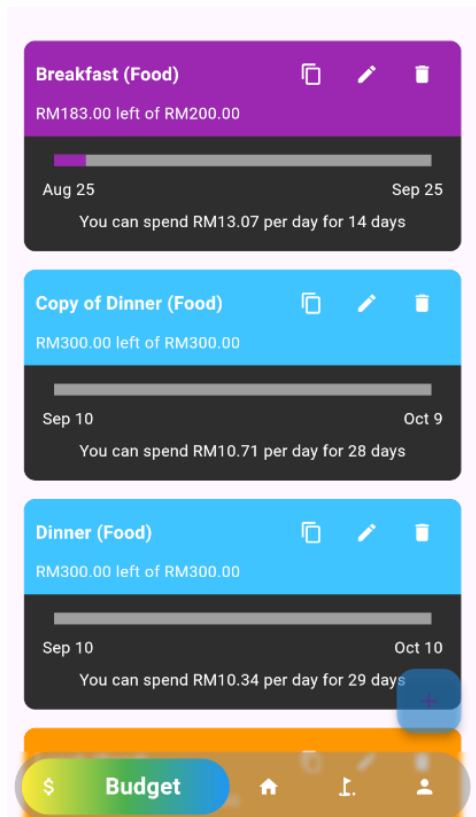


Figure 5.7.4 Duplicate of the Budget



Figure 5.7.5 Transaction List of the Budget

5.8 Goal Screen

When the user clicks on the ‘Goal’ option in the bottom navigation bar, they will be directed to the goal screen. From the goal screen, users can view their goals displayed as cards with key information such as the goal title, the duration, and a progress bar showing progress towards the goals. Users can add a new goal by clicking on the ‘+’ icon next to ‘Goal’ in the top right corner or inside the grey container. This action redirects the user to the Add Goal screen, where they are prompted to fill in details such as Title, Goal Amount, Duration, and Budget Image Icon. Once the information has been entered correctly, the goal will be added to the goal screen. The user can click on the ‘Edit’ icon next to the goal title to access the ‘Edit Goal’ screen. Once changes are made, clicking ‘Update Goal’ will reflect the updates on the goal screen. When the user clicks on the ‘Copy’ icon, a duplicate of the goal will be created. The saved amount will be reset, and the start date will be updated to the current date, with a default one-month duration. Additionally, users can delete a goal by clicking the ‘Delete’ icon next to the ‘Edit’ icon. Users can also view the transactions associated with a goal by clicking on the goal card. Moreover, they can click on the ‘Download’ icon to export the goal’s transaction history in PDF format.

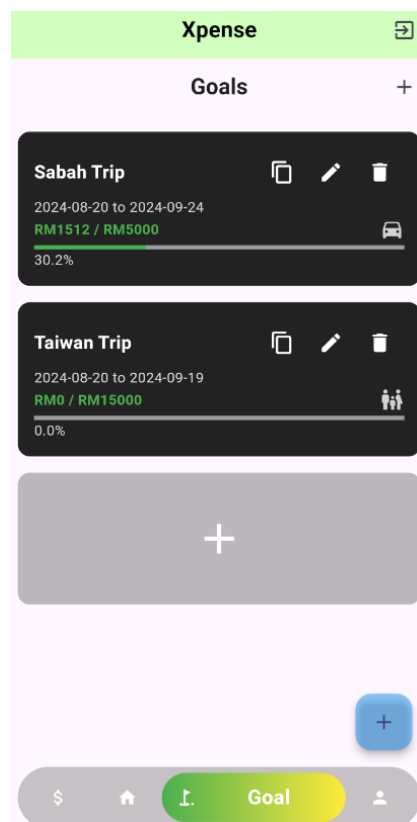


Figure 5.8.1 Goal Screen

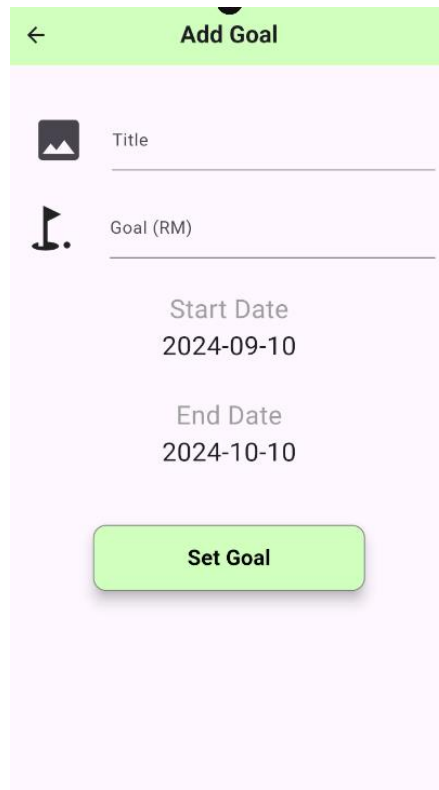


Figure 5.8.2 Add Goal Screen

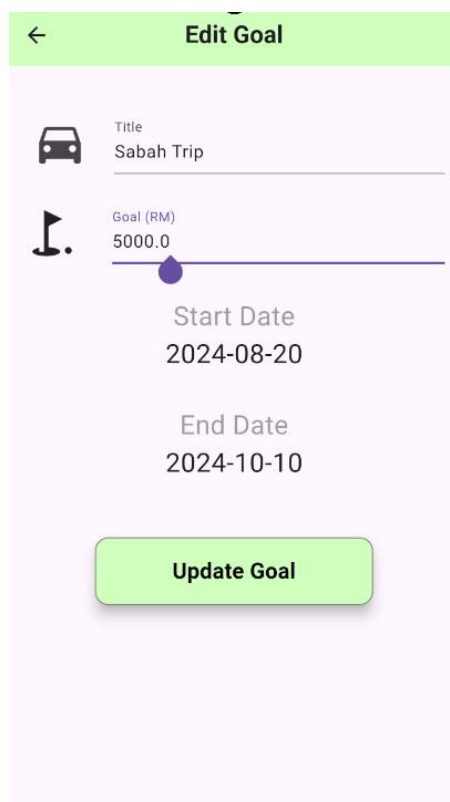


Figure 5.8.3 Edit Goal Screen

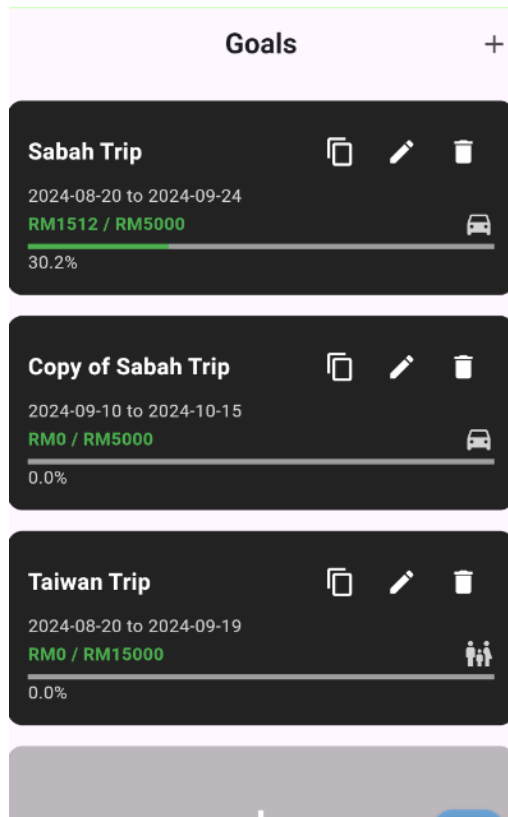


Figure 5.8.4 Duplicate of the Goal

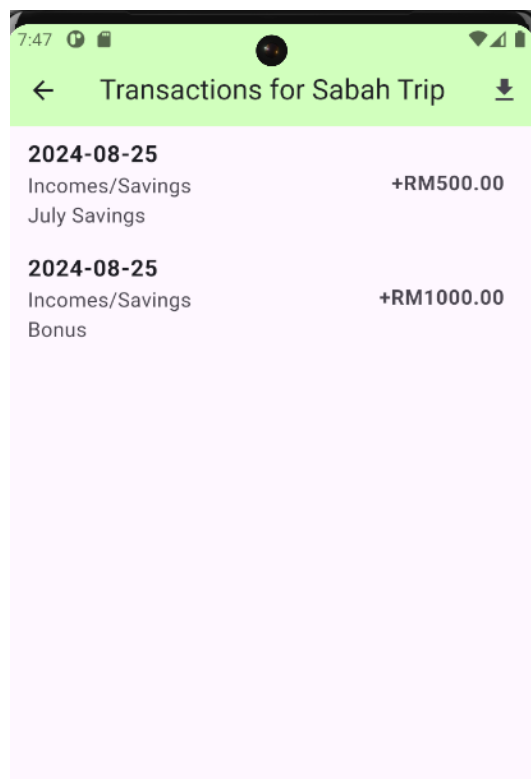


Figure 5.8.5 Transaction List of the Goal

5.9 Manual Input Transaction Screen

There is a floating blue button located at the bottom right of the application. When the user clicks on it, they will be navigated to the "Add Transaction" screen. On this screen, there is a toggle switch button that allows the user to choose whether they are adding a transaction as an expense or income/savings. For an expense transaction, the user is required to input the merchant's name, date, category, details, and amount. If there is a budget available, the user can select it from the list to associate the transaction with that budget. For income/savings transactions, the user needs to select the date, enter the details, and input the amount. If there is a goal available, the user can choose it from the list to link the transaction to that goal. Once the user enters and confirms the information, the transaction is added to the transaction list and the associated budget or goal transaction list if applicable. The transaction cannot be edited after it is saved. Manual input transactions are displayed in blue, while QR payment transactions are shown in black to distinguish between them.

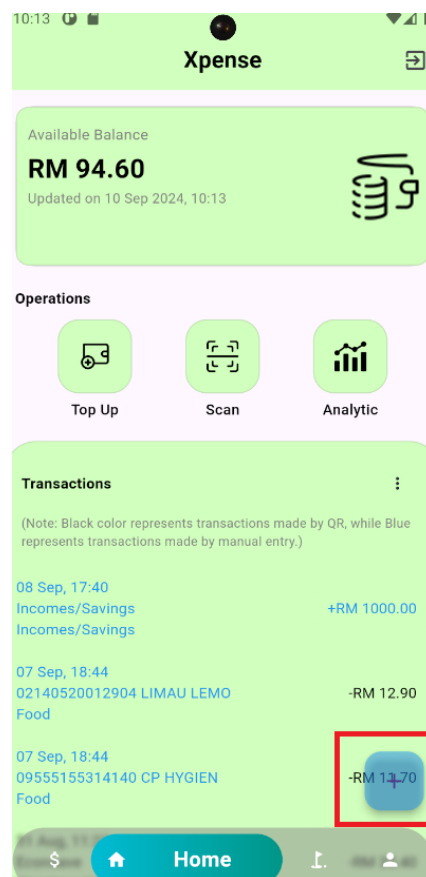


Figure 5.9.1 Manual Input Transaction Floating Button

10:15

← Add Transaction

Merchant Name

Date: September 10, 2024

Category

Details

Select Budget

\$ Amount

Expense

Save Transaction

Figure 5.9.2 Expense Transaction Entry Form

10:17

← Add Transaction

Date: September 10, 2024

Details

Select Goal

\$ Amount

Income / Savings

Save Transaction

Figure 5.9.3 Income/Savings Transaction Entry Form

5.10 All Transaction Screen

Apart from viewing the transaction history on the home screen, users can click the "More" icon next to the "Transactions" title to navigate to the "All Transactions" screen. This screen displays a comprehensive list of all transactions. Users can filter transactions by date or category for easier navigation. By clicking on a specific transaction, users can view detailed information, including the transaction type, details, date/time, and status. The screen also provides an option to upload an image related to the transaction. Once uploaded, this image will be displayed in the transaction details each time the user accesses it. If a transaction is an expense and has not yet been associated with a budget, users will have the option to select a budget from the list if available. Conversely, if the transaction is income/savings and has not yet been linked to a goal, users can select a goal from the list if one is available.

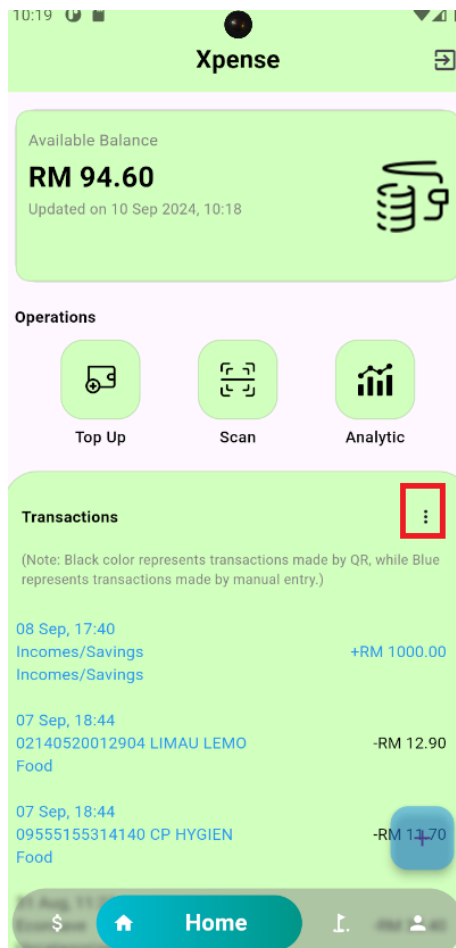


Figure 5.10.1 “More” Icon to Navigate to All Transaction Screen

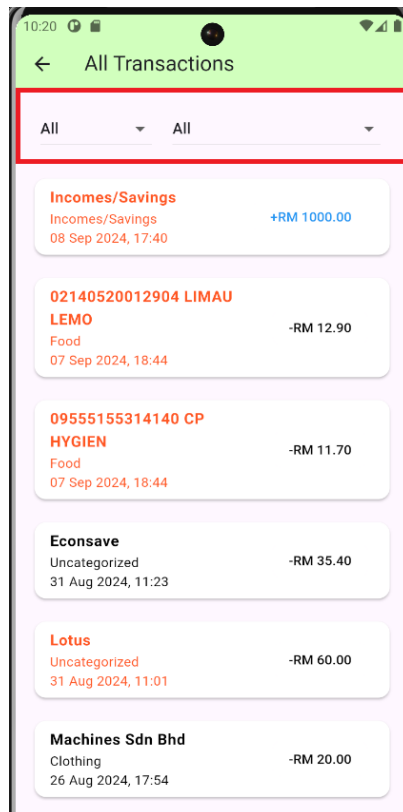


Figure 5.10.2 All Transaction Screen with the Filters

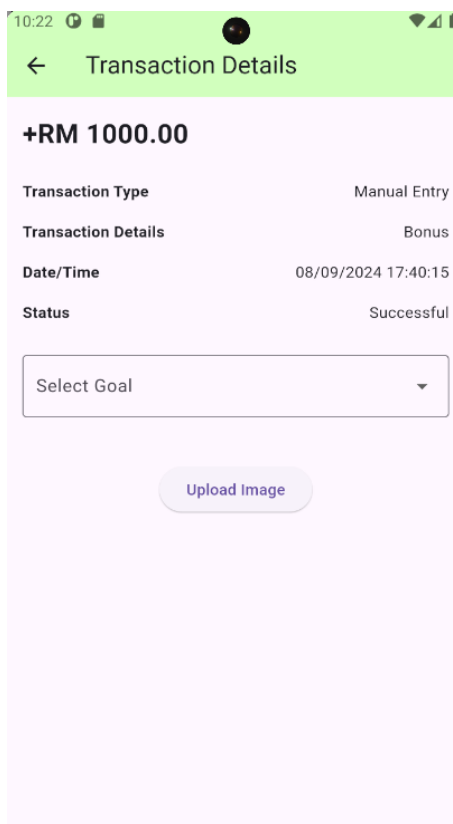


Figure 5.10.3 Income/Savings Transaction Details Screen

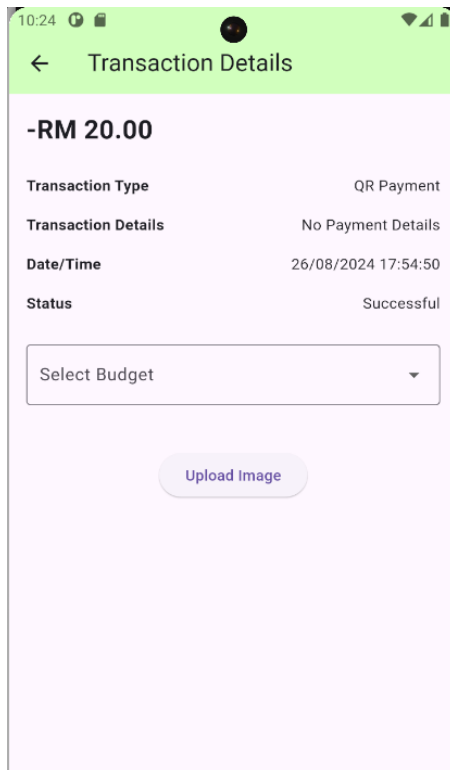


Figure 5.10.4 Expense Transaction Details Screen

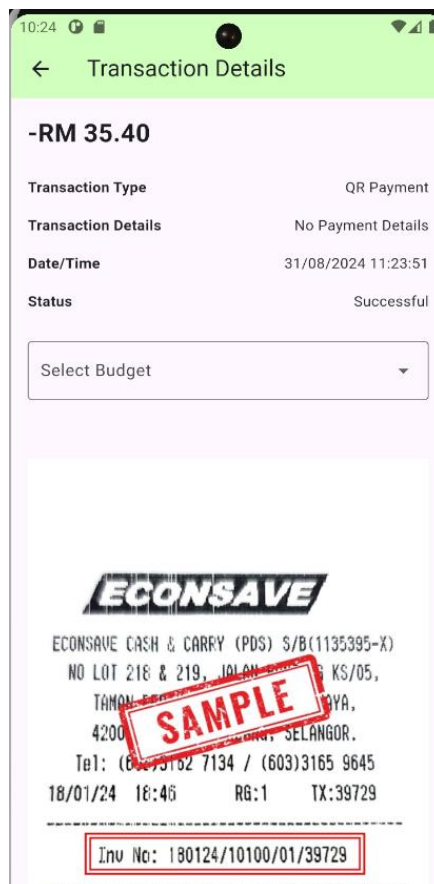


Figure 5.10.5 Transaction Details with Image Uploaded

5.11 Receipt OCR Screen

Users can perform the OCR process within the transaction details screen. When the user navigates to the transaction details screen, they have the option to upload an image. Once an image is uploaded, an "Extract Text" button will appear. Upon clicking this button, the OCR process will start the app will redirect the user to the "Extract Text" screen if the process is successful. On the "Extract Text" screen, the extracted items from the image will be displayed in a list format with checkboxes next to each item which allows the user to select the relevant ones. After selecting the items, the app navigates to a "Selected Items" screen where the chosen items and their corresponding prices are shown. The user is then required to assign a category to each selected item. Once the categories are assigned, the user can click the "Save" button and the selected items will be added to the transaction list. This feature is particularly useful in situations where the user pays the bill upfront or when a transaction involves multiple categories as it allows for a more convenient and time-saving method of categorizing items accurately.

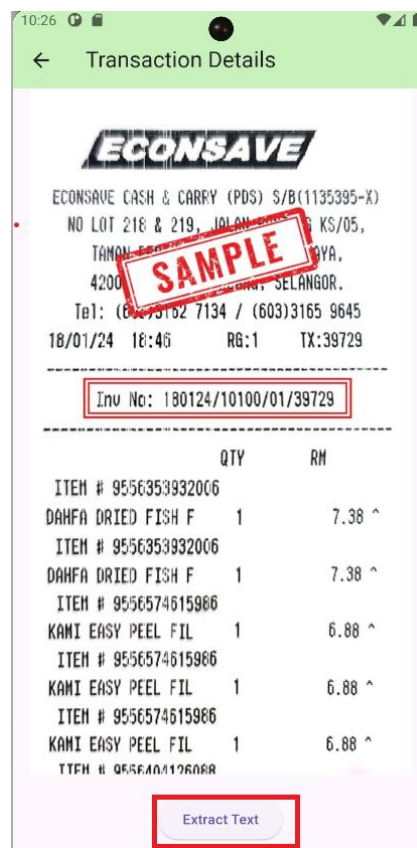


Figure 5.11.1 Extract Text Button After an Image Uploaded

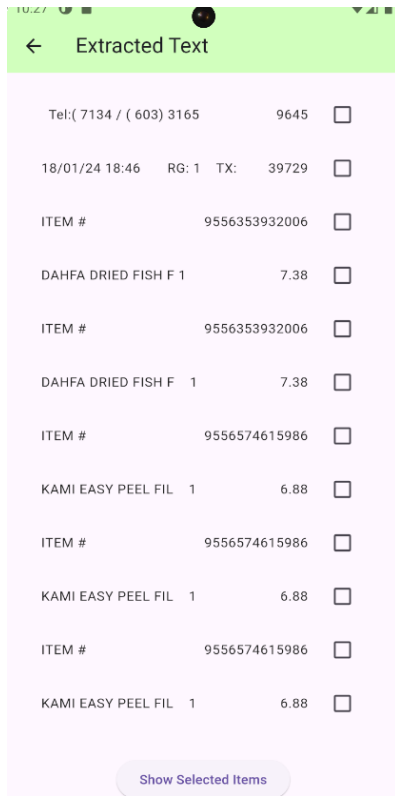


Figure 5.11.2 Extract Text with Checkboxes

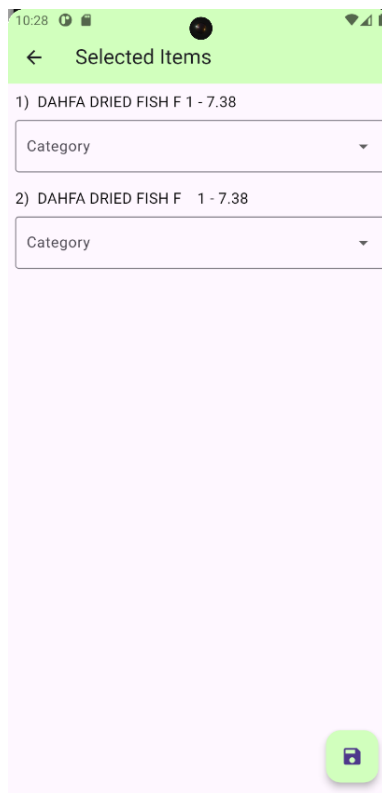


Figure 5.11.3 Selected Item Screen

5.12 Profile Screen

Users can access their profile by clicking the “Profile” icon in the bottom navigation bar. On the profile screen, they can view their personal information such as their name and email. There are two available options which are “Edit Profile” and “Log Out.” When users click on “Edit Profile”, they will be directed to the “Edit Profile” screen where they can update their information such as name and email. Once the edits are validated, the changes will be reflected on the profile screen. Users will be signed out of the application and redirected to the Welcome screen by clicking “Log Out”.

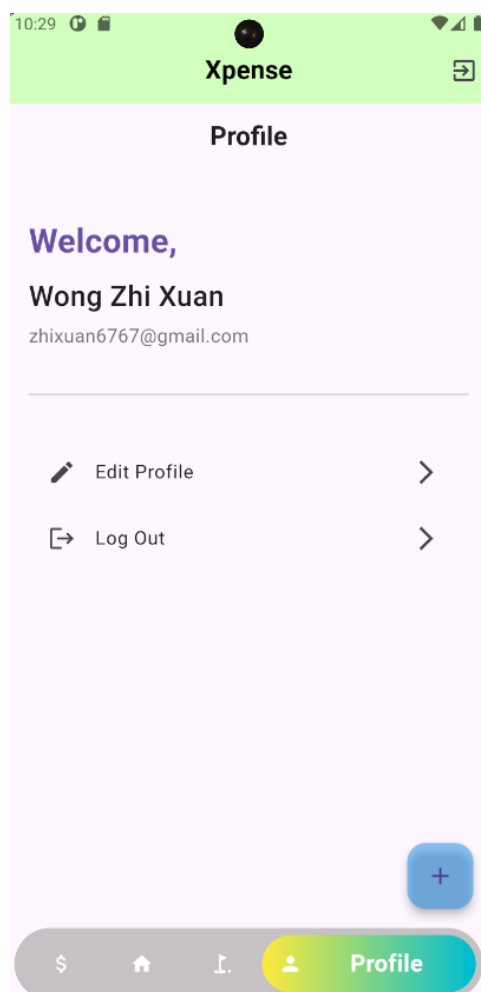


Figure 5.12.1 User Profile Screen

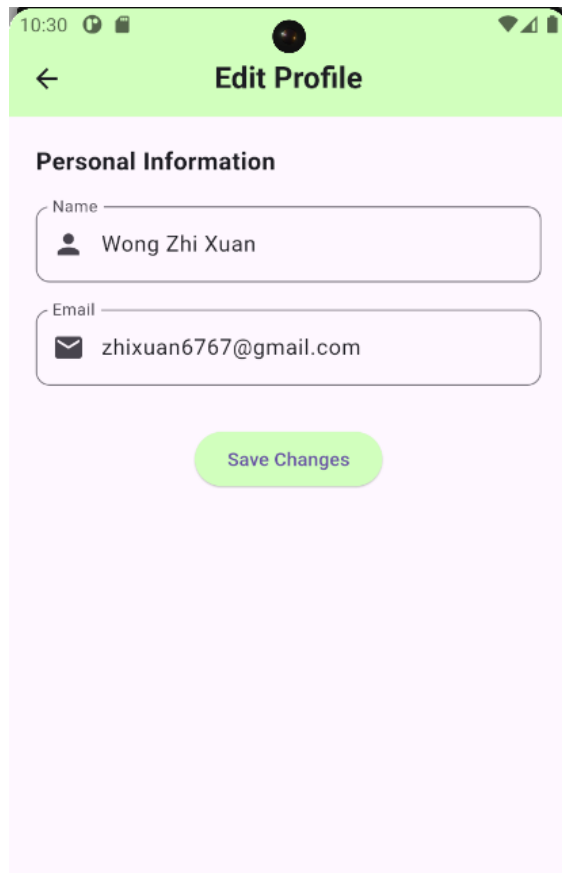


Figure 5.12.2 Edit Profile Screen

5.13 Summary

All the modules, including user authentication and profile management, virtual wallet, transaction tracking and categorization, income or savings recording, manual transaction input, budget management and calculation, goal-setting and tracking, analytics, and receipt OCR, have been successfully developed in the application. Each module's functionality is fully operational and the corresponding screens of the application are demonstrated in this chapter.

Chapter 6 Conclusion

6.1 Project Review, Discussion and Conclusion

Effective management of personal finances has become a critical skill for individuals to navigate the complexities of their financial lives. However, the traditional approach to personal finance management, often relying on manual recording of expenses, sporadic note-taking, and fragmented transaction categorization, presents numerous challenges. These challenges can range from incomplete expense tracking leading to overspending, to a lack of financial awareness hampering future risk preparedness. There is a compelling need for a solution that addresses these issues and empowers individuals to take control of their financial well-being.

This proposed application aims to improve the efficiency of managing individuals' financial activities. The RAD methodology is utilised in this development of the application. In alignment with the first objective of achieving comprehensive expense tracking and budget control, several key modules have been successfully implemented which include the Virtual Wallet Module, Transaction Tracking and Categorization Module, Manual Transaction Input Module, and Budget Management and Calculation Module. Users can utilize the virtual wallet to make payments with categories assigned during the transaction and ensure the records are updated in real time. Additionally, users can manually record offline payments such as cash payments which include expenses and income. The budget management feature allows users to set budgets with daily spending advice, helping them monitor their spending patterns and ensuring they stay within their budget limits. Regarding the second and third objectives, the Goal-Setting and Tracking Module, Analytics Module, and Receipt OCR Module have been successfully developed. The goal-setting and tracking functionality enables users to define and monitor their aspirations. The Analytics Module offers data visualization through bar charts for monthly spending and pie charts for spending category breakdowns. The Receipt OCR Module streamlines the process of categorizing items, saving time and ensuring accuracy by extracting the text from the receipt.

During the development phase, several challenges were encountered. One of the major issues was with user authentication, specifically the inability to receive OTP codes when using an emulator. Testing was only possible with testing numbers in Firebase or on physical devices. While the OTP functioned correctly on physical devices, this was only a temporary solution

for debugging. After some research, the issue was resolved and it was determined that the Play Integrity API had not been properly initialized during the application setup which was the root cause. Moreover, challenges were faced with the Receipt OCR feature. Initially, the MLKit Flutter package successfully extracted text from receipts, but the varying formats of receipts made it difficult to manage a standard structure. After further research, the Asprise API was identified as a solution as it formats the extracted data into a structured JSON format. This improved the organization of extracted items and prices, making the selection and categorization process much smoother for the user, especially in the case of users paying the bill upfront or when a transaction involves multiple categories.

6.2 Novelties and Contributions

The project introduces several novelties and significant contributions to enhance personal finance management, making it more efficient and user-friendly. The Virtual Wallet Module integrates QR code payments and wallet top-up functions, with all transactions being automatically recorded. This eliminates the need for manual entry and ensures accurate real-time tracking of expenses. The project also reduces manual categorization efforts and increases accuracy to provide users with a clear view of their spending habits. The Income or Saving Recording Module adds another layer of comprehensiveness by enabling users to record multiple income sources such as salaries, bonuses, and investments. This feature promotes better financial planning by offering a complete overview of users' financial inflows. A notable Manual Transaction Input Modul allows users to enter transactions that were not captured through the virtual wallet such as cash payments. This ensures comprehensive financial tracking and gives users full control over their financial records. The Budget Management and Calculation Module is designed to help users manage their finances by setting daily, weekly, or monthly budgets. This module includes real-time budget tracking and dynamic adjustments based on spending patterns to help users stay within their financial limits and foster better spending habits. Additionally, the Goal-Setting and Tracking Module motivates users to set and achieve financial goals, such as saving for specific purchases or paying off debt. The ability to track progress visually reinforces financial discipline and helps users stay focused on their objectives. The Analytics Module further enhances the user experience by offering data-driven insights through charts and graphs. These visual tools break down expenses and allow users to make informed decisions about their finances to improve budgeting and financial planning. Lastly, the project introduces an OCR-Based Receipt Scanning Module that simplifies the

process of recording purchases. By extracting relevant data from physical receipts, such as the item, amount, and date, this feature reduces manual input and improves the accuracy of transaction records.

6.3 Future Work

There are several improvements that can be made to enhance the current project. Currently, the payment gateway in the project is only in test mode and functions with test cards due to the costs and the need for business registration. Future work includes fully integrating a live payment gateway and making the virtual wallet functional for real transactions. This step would significantly enhance the utility of the virtual wallet by allowing users to carry out real financial transactions securely.

Another area for improvement is the addition of a notification system. This feature would notify users of important events, such as low balance alerts, budget limit exceedances, and goal achievements. These notifications will ensure that users stay informed about their financial activities and avoid missing critical updates related to their budget and financial goals.

The current Optical Character Recognition (OCR) system can be further improved. It struggles to extract receipt data in a well-structured format which affects the accuracy of financial records. In future, the OCR algorithm needs to be revised and improved to ensure that receipts are processed accurately and extracting information in a structured format to maintain precise transaction records.

REFERENCES

- [1] “Personal Finance,” Corporate Finance Institute.
<https://corporatefinanceinstitute.com/resources/wealth-management/personal-finance>
(accessed Aug. 05, 2023).
- [2] S. a s www.spendee.com, “Meet Spendee | Spendee,” www.spendee.com.
<https://www.spendee.com/about> (accessed Aug. 05, 2023).
- [3] Spendee, “Story of Spendee: Interview with Head of Product,” Spendee: when your money talks, May 27, 2019. <https://medium.com/spendee/story-of-spendee-interview-with-head-of-product-5ff3394f9e99> (accessed Aug. 06, 2023).
- [4] P. Shah, “Spendee vs Wallet: Which App Is Better for Finance Management,” Guiding Tech, Jun. 08, 2019. <https://www.guidingtech.com/spendee-vs-wallet-finance-management-app-comparison/> (accessed Aug. 19, 2023).
- [5] Spendee, “Labels Explained: Detailed & Simple Categorization of Transactions,” Medium, Feb. 12, 2021. <https://spendeeapp.medium.com/labels-explained-detailed-simple-categorization-of-transactions-8bab4479dc5e> (accessed Aug. 07, 2023).
- [6] “Spendee Review 2023,” WealthRocket.
<https://www.wealthrocket.com/budgeting/spendee-> (accessed Aug. 17, 2023).
- [7] S. a s www.spendee.com, “Pricing - Start any Plan with a 7 day Free Trial | Spendee,” www.spendee.com. <https://www.spendee.com/pricing> (accessed Aug. 18, 2023).
- [8] K. Payne, “Mint Budgeting App Review,” Forbes Advisor, May 05, 2021.
<https://www.forbes.com/advisor/banking/mint-budgeting-app-review/> (accessed Aug. 20, 2023).
- [9] “Budget Tracker & Planner | Free Online Money Management | Mint,” mint.intuit.com.
<https://mint.intuit.com/> (accessed Aug. 23, 2023).

[10] “Mint Review,” PCMAG. <https://www.pcmag.com/reviews/mintcom> (accessed Aug. 23, 2023).

[11] “YNAB Review,” PCMAG. <https://www.pcmag.com/reviews/ynab> (accessed Aug. 23, 2023).

[12] “Good Budget Review: Personal Budgeting Software | WealthRocket 2023,” WealthRocket. <https://www.wealthrocket.com/budgeting/goodbudget-review/> (accessed Aug. 23, 2023).

[13] “How to Get Started on Android,” Goodbudget. <https://goodbudget.com/help/mobile-apps/get-started-android/> (accessed Aug. 23, 2023).

[14] B. Holzhauer, “Goodbudget Budgeting App Review,” Forbes Advisor, May 06, 2021. <https://www.forbes.com/advisor/banking/goodbudget-budgeting-app-review/> (accessed Aug. 23, 2023).

[15] kissflow, “Rapid Application Development (RAD) | Definition, Steps & Full Guide,” kissflow.com, Oct. 31, 2022. <https://kissflow.com/application-development/rad/rapid-application-development/> (accessed Aug. 27, 2023).

[16] “Rapid Application Development (RAD) Guide | Definition, Steps,” Aug. 09, 2023. <https://quixy.com/blog/rapid-application-development/> (accessed Aug. 29, 2023).

[17] M. Heller, “What is Visual Studio Code? Microsoft’s extensible code editor,” InfoWorld, Jul. 08, 2022. <https://www.infoworld.com/article/3666488/what-is-visual-studio-code-microsofts-extensible-code-editor.html> (accessed Aug. 25, 2023).

[18] Google Developers, “Meet Android Studio | Android Developers,” Android Developers, 2019. <https://developer.android.com/studio/intro> (accessed Aug. 25, 2023).

[19] K. T. Hanna and L. Rosencrance, "What is Google Firebase?," Mobile Computing, <https://www.techtarget.com/searchmobilecomputing/definition/Google-Firebase> (accessed Apr. 18, 2024).

[20] NAGENDRAG, "What Is Java?," CloudFoundation | Blog, May 23, 2023. <https://cloudfoundation.com/blog/what-is-java/> (accessed Aug. 25, 2023).[25] Kotlin, 2020.

[21] Kotlin, 2020. <https://kotlinlang.org/> (accessed Aug. 25, 2023).

[22] "What is the Flutter Framework and why you should learn it today: By perforce," Perfecto by Perforce, <https://www.perfecto.io/blog/what-is-flutter-framework> (accessed Apr. 18, 2024).

[23] "Flutter: What is Dart Programming - Javatpoint," www.javatpoint.com, <https://www.javatpoint.com/flutter-dart-programming> (accessed Apr. 18, 2024).

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3T3	Study week no.: 2
Student Name & ID: Wong Zhi Xuan 21ACB06564	
Supervisor: Dr Tan Joi San	
Project Title: Development Of Personal Finance Management Application	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

Meeting with the supervisor to discuss the task to be done. Doing research on the development of the module within the application.

2. WORK TO BE DONE

Planning to start the development of the budget management module.

3. PROBLEMS ENCOUNTERED

-

4. SELF EVALUATION OF THE PROGRESS

The progress is going well so far.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3T3	Study week no.: 4
Student Name & ID: Wong Zhi Xuan 21ACB06564	
Supervisor: Dr Tan Joi San	
Project Title: Development Of Personal Finance Management Application	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

The budget module has been completed. The budget module includes functions such as set a budget, edit a budget, duplicate a budget and delete a budget. Moreover, users can click on the budget to check the transaction associated with the budget and users are able to download the budget transaction history.

2. WORK TO BE DONE

Planning to start the development of the goal management module.

3. PROBLEMS ENCOUNTERED

-

4. SELF EVALUATION OF THE PROGRESS

The progress is going well and on the track.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3T3	Study week no.: 6
Student Name & ID: Wong Zhi Xuan 21ACB06564	
Supervisor: Dr Tan Joi San	
Project Title: Development Of Personal Finance Management Application	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

The goal module has been completed. The goal module includes functions such as set a goal, edit a goal, duplicate a goal and delete a goal. Moreover, users can click on the goal to check the transaction associated with the goal and users are able to download the goal transaction history.

2. WORK TO BE DONE

Planning to start the development of Manual Input Transaction Module.

3. PROBLEMS ENCOUNTERED

-

4. SELF EVALUATION OF THE PROGRESS

The progress is going well and on the track.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3T3	Study week no.: 8
Student Name & ID: Wong Zhi Xuan 21ACB06564	
Supervisor: Dr Tan Joi San	
Project Title: Development Of Personal Finance Management Application	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

The Manual Input Transaction had been done. Users can manually enter the transaction history made offline such as cash payments. Users can input an expense transaction or income/savings.

2. WORK TO BE DONE

Planning to start the development of the Receipt Module.

3. PROBLEMS ENCOUNTERED

-

4. SELF EVALUATION OF THE PROGRESS

The progress is going well and on track.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3T3	Study week no.: 10
Student Name & ID: Wong Zhi Xuan 21ACB06564	
Supervisor: Dr Tan Joi San	
Project Title: Development Of Personal Finance Management Application	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

The transaction details screen had been done. Users can click on the transaction details and upload an image or select add to goal or budget.

2. WORK TO BE DONE

Continue the development of OCR.

3. PROBLEMS ENCOUNTERED

The OCR cannot function well. The image text had been extracted but cannot manage the standard format of the extract item due to the complex structure of different receipt

4. SELF EVALUATION OF THE PROGRESS

The progress is stuck due to the OCR function.



Supervisor's signature



Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: Y3T3	Study week no.: 12
Student Name & ID: Wong Zhi Xuan 21ACB06564	
Supervisor: Dr Tan Joi San	
Project Title: Development Of Personal Finance Management Application	

1. WORK DONE

[Please write the details of the work done in the last fortnight.]

The OCR feature had been done. Now the user can upload a receipt and start the OCR process. After extracting the item, the user can select the desired item and add it to the transaction history. A profile screen is also developed.

2. WORK TO BE DONE

Check and test the overall application and prepare to make any modifications to the application.

3. PROBLEMS ENCOUNTERED

-

4. SELF EVALUATION OF THE PROGRESS

The progress is going well.



Supervisor's signature



Student's signature

POSTER

DEVELOPMENT OF PERSONAL FINANCE MANAGEMENT APPLICATION



INTRODUCTION **1**

In the modern world, effective management of personal finances has become a critical skill for individuals to navigate the complexities of their financial lives. The management of one's financial resources, including saving, investing, budgeting and protecting, plays a key role in determining one's financial health and future security.

PROBLEM STATEMENT **2**

- Inadequate Expense Tracking and Overbudgeting
- Unorganized Transaction Categorization
- Limited Financial Awareness and Future Risk Preparedness

OBJECTIVE **3**

- To Achieve Comprehensive Expense Tracking and Budget Control
- To Implement Intelligent Transaction Categorization using AI
- To Enhance Financial Insights and Facilitate Risk Planning

METHODOLOGY **4**

- RAD (Determining Requirements Phase, User Design Phase, Construction Phase, Implementation Phase)
- Flutter, Dart
- Visual Studio Code, Firebase

PROJECT SCOPE **5**

- User Authentication and Profile Management module
- Virtual Wallet module
- Transaction Tracking and Categorization module
- Income or Saving Recording module
- Manual Transaction Input module
- Budget management and Calculation module
- Goal-setting and Tracking module
- Analytics module
- Receipt OCR module

UTAR UNIVERSITI TUNKU ABDUL RAHMAN

BACHELOR OF COMPUTER SCIENCE (HONS)
UNIVERSITI TUNKU ABDUL RAHMAN

DONE BY WONG ZHI XUAN
SUPERVISED BY DR TAN JOI SAN

PLAGIARISM CHECK RESULT

fyp2forturnitin.docx

ORIGINALITY REPORT

4%	2%	2%	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Mohamed E. Fayad. " Stable Design Patterns for Software and Systems", CRC Press, 2017 Publication	1%
2	open-innovation-projects.org Internet Source	1%
3	K Chayashree, U K Harshitha, Mohammed Sayeed, L P Nagaditya, Sumalatha Aradhya, K R Subramanya Navada. "Early Detection of Dry Eyes Through Blink", 2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT), 2022 Publication	<1%
4	123dok.com Internet Source	<1%
5	www.matellio.com Internet Source	<1%
6	Katarina Milojković, Miodrag Živković, Nebojša Bačanin Džakula. "Agile Multi-user Android Application Development With Firebase: Authentication, Authorization, and	<1%

Profile Management", Proceedings of the
International Scientific Conference - Sinteza
2024, 2024

Publication

7	discuss.ilw.com Internet Source	<1 %
8	vocal.media Internet Source	<1 %
9	financesyrup.com Internet Source	<1 %
10	dspace.daffodilvarsity.edu.bd:8080 Internet Source	<1 %
11	Susan Traynor, Michael A. Wellens, Venki Krishnamoorthy. "SAP SuccessFactors Talent: Volume 1", Springer Science and Business Media LLC, 2021 Publication	<1 %
12	Sivaraj Selvaraj. "Mastering REST APIs", Springer Science and Business Media LLC, 2024 Publication	<1 %
13	www.daytrading.com Internet Source	<1 %
14	origin.geeksforgeeks.org Internet Source	<1 %

scholar.ppu.edu

15	Internet Source	<1 %
16	read.bookcreator.com Internet Source	<1 %
17	Kefas Bagastio, Raymond Sunardi Oetama, Arief Ramadhan. "Development of stock price prediction system using Flask framework and LSTM algorithm", Journal of Infrastructure, Policy and Development, 2023 Publication	<1 %
18	summit.sfu.ca Internet Source	<1 %
19	www.adaface.com Internet Source	<1 %
20	www.coursehero.com Internet Source	<1 %
21	Howard Anderson, Sharon Yull, Bruce Hellingsworth. "Higher National Computing - Core Units for BTEC Higher Nationals in Computing and IT", Newnes, 2004 Publication	<1 %
22	nakedelement.co.uk Internet Source	<1 %
23	text-id.123dok.com Internet Source	<1 %

24	D. Yogi Goswami. "The CRC Handbook of Mechanical Engineering", CRC Press, 2019 Publication	<1 %
25	Lennart Andersson, Kyla Farrell, Oleg Moshkovich, Cheryle Cranbourne. "Implementing Virtual Design and Construction using BIM - Current and future practices", Routledge, 2016 Publication	<1 %
26	Ronald Compesi. "Video Field Production and Editing", Routledge, 2019 Publication	<1 %
27	Yong Gu Ji. "Advances in Affective and Pleasurable Design", CRC Press, 2019 Publication	<1 %
28	digitalcommons.odu.edu Internet Source	<1 %
29	fortune.com Internet Source	<1 %
30	strathprints.strath.ac.uk Internet Source	<1 %
31	ukdiss.com Internet Source	<1 %
32	www.ijraset.com Internet Source	<1 %

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1 of 1



FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	WONG ZHI XUAN
ID Number(s)	21ACB06564
Programme / Course	BACHELOR OF COMPUTER SCIENCE (HONOURS)
Title of Final Year Project	DEVELOPMENT OF PERSONAL FINANCE MANAGEMENT APPLICATION

Similarity	Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)
Overall similarity index: <u> 4 </u> % Similarity by source Internet Sources: <u> 2 </u> % Publications: <u> 2 </u> % Student Papers: <u> 0 </u> %	
Number of individual sources listed of more than 3% similarity: <u> 0 </u>	
Parameters of originality required and limits approved by UTAR are as Follows: (i) Overall similarity index is 20% and below, and (ii) Matching of individual sources listed must be less than 3% each, and (iii) Matching texts in continuous block must not exceed 8 words <i>Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.</i>	

Note Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

Signature of Supervisor

Name: Tan Joi San

Date: 12 Sep 2024

Signature of Co-Supervisor

Name: _____

Date: _____



UNIVERSITI TUNKU ABDUL RAHMAN

**FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY
(KAMPAR CAMPUS)**

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	21ACB06564
Student Name	WONG ZHI XUAN
Supervisor Name	DR TAN JOI SAN

TICK (√)	DOCUMENT ITEMS
	Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
N/A	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
N/A	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

(Signature of Student)

Date: 12 SEPTEMBER 2024