

Personal Finance Management Application

BY

Tan Le Jie

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

**BACHELOR OF INFORMATION SYSTEMS (HONOURS) BUSINESS INFORMATION
SYSTEMS**

Faculty of Information and Communication Technology

(Kampar Campus)

MAY 2024

REPORT STATUS DECLARATION FORM

Title: Personal Finance Management Application

Academic Session: MAY 2024

I TAN LE JIE

(CAPITAL LETTER)

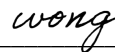
declare that I allow this Final Year Project Report to be kept in
Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

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:

17, Jalan 1/3 A,

Taman Kerian Permai,

34200, Parit Buntar Perak

Ts Dr Wong Pei Voon

Supervisor's name

Date: 13 September 2024

Date: 13/9/2024

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/INSTITUTE* OF INFORMATION AND COMMUNICATION TECHNOLOGY
UNIVERSITI TUNKU ABDUL RAHMAN

Date: 13 September 2024

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that Tan Le Jie (ID No: 20ACB04995) has completed this final year project entitled “Personal Finance Management Application” under the supervision of Dr Wong Pei Voon (Supervisor) from the Department of Digital Economy Technology, 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,



(Tan Le Jie)

DECLARATION OF ORIGINALITY

I declare that this report entitled “**METHODOLOGY, CONCEPT AND DESIGN OF A 2-MICRON CMOS DIGITAL BASED TEACHING CHIP USING FULL-CUSTOM DESIGN STYLE**” 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 : Tan Le Jie

Date : 13 September 2024

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisors, Ts Dr.Wong Pei Voon for the invaluable opportunity to undertake this significant project. Your guidance and, support have been instrumental in my academic journey. A million thanks for your enduring patience and insightful contributions.

ABSTRACT

Managing personal finances can be challenging due to the lack of personalized feedback, real-time insights, and effective forecasting tools. Individuals often struggle to track their financial health, make informed decisions, and plan for future expenses, especially with the complexity of modern financial trends. After reviewing existing technologies like OpenAI GPT for natural language processing and Facebook Prophet for time series forecasting, along with financial management applications such as Spendee, Money Lover, Expensify, Mint and YNAB, gaps in personalization and dynamic financial insights were identified. These existing tools often lack the capability to provide users with tailored financial advice and accurate, long-term forecasting. In response, this project offers a solution by developing a mobile application that integrates a Personalized Financial Health Score System, an AI-powered chatbot for interactive financial guidance, and predictive analytics to help users better understand and manage their financial health. This app aims to empower individuals to make more informed decisions, offering real-time financial insights and a proactive approach to personal finance management.

TABLE OF CONTENTS

TITLE PAGE	I
REPORT STATUS DECLARATION FORM.....	II
DECLARATION OF ORIGINALITY	IV
ACKNOWLEDGEMENTS	V
ABSTRACT.....	VI
LIST OF FIGURES	XI
LIST OF TABLES	XV
LIST OF ABBREVIATIONS	XVI
CHAPTER 1 INTRODUCTION.....	1
1.1 PROBLEM STATEMENT AND MOTIVATION	3
1.1.1 <i>Inadequacies in Current Financial Management Tools</i>	3
1.1.2 <i>Limited Interactive Support in Personal Finance Management Apps</i>	3
1.1.3 <i>Inadequate Financial Forecasting and Planning</i>	4
1.2 OBJECTIVES	4
1.2.1 <i>Develop a Personalized Financial Health Score System</i>	5
1.2.2 <i>Implement an AI Chatbot for Interactive Financial Guidance</i>	5
1.2.3 <i>Implement Predictive Analytics for Financial Forecasting</i>	5
1.3 PROJECT SCOPE AND DIRECTION	5
1.4 CONTRIBUTIONS	6
1.5 REPORT ORGANIZATION.....	6
CHAPTER 2 LITERATURE REVIEW	8
2.1 REVIEW OF THE TECHNOLOGIES.....	8
2.1.1 <i>OpenAI GPT for Chatbots</i>	8
2.1.2 <i>Facebook Prophet for Time Series Forecasting</i>	9
2.1.3 <i>OCR for Receipt Scanning</i>	11
2.1.4 <i>Summary of the Technologies Review</i>	13
2.2 REVIEW OF THE EXISTING APPLICATIONS.....	14
2.2.1 <i>Existing System A: Money Lover</i>	14
2.2.2 <i>Existing System B: Expensify</i>	16
2.2.3 <i>Existing System C: Spendee</i>	18
2.2.4 <i>Existing System D: YNAB</i>	20

2.2.5 Existing System E: Mint	22
2.2.6 Comparative Summary of Existing Systems and Proposed Application.....	25
CHAPTER 3 SYSTEM METHODOLOGY/APPROACH	1
3.1 SYSTEM DESIGN DIAGRAM.....	2
3.1.1 System Architecture Diagram.....	2
3.1.2 Use Case Diagram and Description.....	3
3.1.3 Activity Diagram.....	16
CHAPTER 4 SYSTEM DESIGN	28
4.1 SYSTEM BLOCK DIAGRAM	28
4.2 SYSTEM FLOW DIAGRAM	30
4.3 SYSTEM FLOW DESCRIPTION (CODE).....	30
4.3.1 Login.tsx.....	30
4.3.2 Register.tsx.....	31
4.3.4 About.tsx.....	33
4.3.5 CreateWallet.tsx.....	34
4.3.6 MyMenu.tsx.....	35
4.3.7 TransactionAdd.tsx	36
4.3.8 MyLocation.tsx.....	37
4.3.9 ContactList.tsx.....	38
4.3.10 CategoryList.tsx	39
4.3.11 CategoryAdd.tsx.....	40
4.3.12 TransactionList.tsx.....	41
4.3.13 TransactionDetail.tsx.....	43
4.3.14 TransactionEdit.tsx	46
4.3.15 Budget.tsx.....	48
4.3.16 BudgetAdd.tsx	49
4.3.17 BudgetDetail.tsx.....	50
4.3.18 UpdateGoals.tsx.....	53
4.3.19 TrackAchievements.tsx.....	54
4.3.20 LoanAdd.tsx.....	56
4.3.21 AIChatbot.tsx.....	57
4.3.22 FinancialAnalysis.tsx & finance_forecast.py.....	61
4.3.23 ReceiptOCR.tsx	65
CHAPTER 5 SYSTEM IMPLEMENTATION	69

5.1 HARDWARE SETUP.....	69
5.2 SOFTWARE SETUP.....	69
5.3 SETTING AND CONFIGURATION	70
5.4 SYSTEM OPERATION (WITH SCREENSHOT).....	74
5.5 IMPLEMENTATION ISSUES AND CHALLENGES	95
5.6 CONCLUDING REMARK	96
CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION	98
6.1 SYSTEM TESTING AND PERFORMANCE METRICS	98
6.1.1 Login Module	98
6.1.2 Register Module	98
6.1.3 Forget Password Module.....	99
6.1.4 Add Transaction Module.....	99
6.1.5 View Transaction List Module	100
6.1.6 View Transaction Detail Module	100
6.1.7 Edit Transaction Module	101
6.1.8 Add Category Module	101
Figure 6- 9 Add Category Module.....	101
6.1.9 View Category List Module.....	102
6.1.10 Budget Module	102
6.1.11 Add Budget Module.....	103
6.1.12 View Budget Detail Module	103
6.1.13 Create Wallet Module	104
6.1.14 Menu Module	104
6.1.15 Report Module.....	105
6.1.16 View Report Detail Module.....	106
6.1.17 Scan Receipt Module.....	107
6.1.18 Track Achievements Module	108
6.1.20 Set Location Module	109
6.1.21 Add Loan Module.....	110
6.1.22 Select Contact List Module	110
6.1.23 AI Chatbox Module	110
6.1.24 Facebook Prophet Forecast Module.....	111
6.2 PROJECT CHALLENGE	112
6.3 OBJECTIVE EVALUATION.....	113
6.4 CONCLUDING REMARK.....	114

CHAPTER 7 CONCLUSION AND RECOMMENDATION	116
7.1 CONCLUSION.....	116
7.2 RECOMMENDATION	117
REFERENCES	119
FINAL YEAR PROJECT WEEKLY REPORT	123
FINAL YEAR PROJECT WEEKLY REPORT	123
FINAL YEAR PROJECT WEEKLY REPORT	123
FINAL YEAR PROJECT WEEKLY REPORT	124
POSTER	127
PLAGIARISM CHECK RESULT.....	128

LIST OF FIGURES

<i>Figure 2- 1 Facebook Prophet</i>	10
<i>Figure 2- 2 Overview of Google Cloud Vision OCR</i>	12
<i>Figure 2- 3 Money Lover Mobile Application</i>	14
<i>Figure 2- 4 Expensify Mobile Application</i>	16
<i>Figure 2- 5 Spendee Mobile Application</i>	19
<i>Figure 2- 6 YNAB Mobile Application.....</i>	21
<i>Figure 2- 7 Mint Mobile Application</i>	23
<i>Figure 3- 1 Phase of Prototyping Methodology</i>	1
<i>Figure 3- 2 System Architecture Diagram of the proposed system</i>	2
<i>Figure 3- 3 Use Case Diagram for the proposed application</i>	3
<i>Figure 3- 4 Activity Diagram of Register and User Login</i>	16
<i>Figure 3- 5 Activity Diagram of Create Wallet</i>	17
<i>Figure 3- 6 Activity Diagram of Add Transaction</i>	18
<i>Figure 3- 7 Activity Diagram of Manage Transaction</i>	19
<i>Figure 3- 8 Activity Diagram of Create Budget</i>	20
<i>Figure 3- 9 Activity Diagram of Manage Budget</i>	21
<i>Figure 3- 10 Activity Diagram of Add Loan</i>	22
<i>Figure 3- 11 Activity Diagram of Track Achievements.....</i>	23
<i>Figure 3- 12 Activity Diagram of View Reports</i>	24
<i>Figure 3- 13 Activity Diagram of Input Query</i>	25
<i>Figure 3- 14 Activity Diagram of Make Prophet Forecast</i>	26
<i>Figure 3- 15 Activity Diagram of Capture Image (OCR).....</i>	27
<i>Figure 4- 1 Block Diagram of the proposed system</i>	28
<i>Figure 4- 2 System Flow Diagram</i>	30
<i>Figure 4- 3 Code Snippet for User Login Process</i>	30
<i>Figure 4- 4 Code Snippet for User Registration</i>	31
<i>Figure 4- 5 Code Snippet for Password Reset.....</i>	32
<i>Figure 4- 6 Code Snippet for User Wallet Navigation</i>	33
<i>Figure 4- 7 Code Snippet for Creating a Wallet</i>	34
<i>Figure 4- 8Code Snippet for MainTabs.....</i>	35
<i>Figure 4- 9 Code Snippet for Adding Transactions.....</i>	36
<i>Figure 4- 10 Code Snippet for Location Selection</i>	37
<i>Figure 4- 11 Code Snippet for Contact List</i>	38

<i>Figure 4- 12 Code Snippet for Category List</i>	39
<i>Figure 4- 13 Code snippet for adding a new category</i>	40
<i>Figure 4- 14 Code snippet for parsing transaction dates</i>	41
<i>Figure 4- 15 Code snippet for fetching transactions</i>	42
<i>Figure 4- 16 Code snippet for fetching wallet currency</i>	43
<i>Figure 4- 17 Code snippet for QR code generation</i>	44
<i>Figure 4- 18 Code snippet for generating and sharing transaction details</i>	45
<i>Figure 4- 19 Code Snippet for Handling Transaction Update</i>	46
<i>Figure 4- 20 Code Snippet for Budget Retrieval and Update</i>	48
<i>Figure 4- 21 Code Snippet for Budget Creation</i>	49
<i>Figure 4- 22 Code Snippet for Budget Data Retrieval and Overspent Update</i>	51
<i>Figure 4- 23 Code Snippet for Budget Deletion</i>	52
<i>Figure 4- 24 Code Snippet for Fetching and Updating Financial Goals</i>	53
<i>Figure 4- 25 Code Snippet for Tracking Achievements and Sending Notifications</i>	54
<i>Figure 4- 26 Code Snippet for Unlocking Achievements and Sending Notifications</i>	55
<i>Figure 4- 27 Code Snippet for Loan Calculation and Advice Generation</i>	56
<i>Figure 4- 28 Code Snippet for API Configuration and Tutorial Initialization</i>	57
<i>Figure 4- 29 Code Snippet for Handling User Messages</i>	57
<i>Figure 4- 30 Code Snippet for Handling Quick Replies</i>	58
<i>Figure 4- 31 Code Snippet for Sending Financial Health Score</i>	59
<i>Figure 4- 32 Code Snippet for Sending Financial Tips</i>	60
<i>Figure 4- 33 Flask API for Time Series Forecasting with Prophet (finance_forecast.py)</i>	61
<i>Figure 4- 34 Code Snippet for Fetching and Processing Transaction Data (FinancialAnalysis.tsx)</i> ..	62
<i>Figure 4- 35 Code Snippet for Time Series Data Preparation (FinancialAnalysis.tsx)</i>	63
<i>Figure 4- 36 Code Snippet for Prophet Forecast Functionality (FinancialAnalysis.tsx)</i>	64
<i>Figure 4- 37 Code Snippet for Generating Financial Insights (FinancialAnalysis.tsx)</i>	64
<i>Figure 4- 38 Code Snippet for Receipt Capture and OCR Processing</i>	65
<i>Figure 4- 39 Code Snippet for Data Extraction and Field Population</i>	66
<i>Figure 4- 40 Code Snippet for Saving Receipt Data to Firebase</i>	67
<i>Figure 5- 1 Firebase Initialization and App Configuration</i>	71
<i>Figure 5- 2 MainStack Navigation Setup</i>	72
<i>Figure 5- 3 Type Definitions for Navigation Parameters</i>	73
<i>Figure 5- 4 Login</i>	74
<i>Figure 5- 5 Register</i>	75
<i>Figure 5- 6 Forgot Password</i>	76

<i>Figure 5- 7 Create Wallet</i>	77
<i>Figure 5- 8 Financial Dashboard</i>	78
<i>Figure 5- 9 Financial Report</i>	79
<i>Figure 5- 10 Financial Health</i>	80
<i>Figure 5- 11 Add Transaction</i>	81
<i>Figure 5- 12 Add Transaction - Select Category</i>	82
<i>Figure 5- 13 Add Transaction - Select Category - Add Category</i>	83
<i>Figure 5- 14 Add Transaction - Additional Fields</i>	84
<i>Figure 5- 15 Transaction List - Transaction Detail</i>	85
<i>Figure 5- 16 Transaction Detail - Transaction Edit and Delete</i>	86
<i>Figure 5- 17 Add Budget</i>	87
<i>Figure 5- 18 Notifications Alert</i>	88
<i>Figure 5- 19 Budget Detail</i>	89
<i>Figure 5- 20 Track Achievements</i>	90
<i>Figure 5- 21 Chatbox with Predefined Options for Financial Assistance</i>	91
<i>Figure 5- 22 Retrieving Financial Data from Firebase in Chatbox</i>	92
<i>Figure 5- 23 Facebook Prophet Forecasting in Financial Analysis</i>	93
<i>Figure 5- 24 Receipt OCR</i>	94
<i>Figure 5- 25 TensorFlow.js Integration Failure and Non-std C++ Exception</i>	96
<i>Figure 6- 2 Login Module</i>	98
<i>Figure 6- 3 Register Module</i>	98
<i>Figure 6- 4 Forget Password Module</i>	99
<i>Figure 6- 5 Add Transaction Module</i>	99
<i>Figure 6- 6 View Transaction List Module</i>	100
<i>Figure 6- 7 View Transaction Detail Module</i>	100
<i>Figure 6- 8 Edit Transaction Module</i>	101
<i>Figure 6- 9 Add Category Module</i>	101
<i>Figure 6- 10 View Category List Module</i>	102
<i>Figure 6- 11 Budget Module</i>	102
<i>Figure 6- 12 Add Budget Module</i>	103
<i>Figure 6- 13 View Budget Detail Module</i>	103
<i>Figure 6- 14 Create Wallet Module</i>	104
<i>Figure 6- 15 Menu Module</i>	104
<i>Figure 6- 16 Report Module</i>	105
<i>Figure 6- 17 View Report Detail Module</i>	106

<i>Figure 6- 18 Scan Receipt Module</i>	107
<i>Figure 6- 19 Track Achievements Module</i>	108
<i>Figure 6- 20 Update Goals Module</i>	109
<i>Figure 6- 21 Set Location Module</i>	109
<i>Figure 6- 22 Add Loan Module</i>	110
<i>Figure 6- 23 Select Contact List Module</i>	110
<i>Figure 6- 24 AI Chatbox Module</i>	111
<i>Figure 6- 25 Facebook Prophet Forecast Module</i>	112
<i>Figure 6- 26 Uncaught Error</i>	113

LIST OF TABLES

<i>Table 2- 1 Comparative Overview of Key Functionalities and Weaknesses of Popular Money Management Mobile Apps</i>	25
<i>Table 3- 1 Use Case Description of Login Module</i>	4
<i>Table 3- 2 Use Case Description of Register Account Module</i>	4
<i>Table 3- 3 Use Case Description of Change Password Module</i>	5
<i>Table 3- 4 Use Case Description of Create Wallet Module</i>	6
<i>Table 3- 5 Use Case Description of Add Transaction Module</i>	6
<i>Table 3- 6 Use Case Description of View Transaction Module</i>	7
<i>Table 3- 7 Use Case Description of Transaction Detail Module</i>	8
<i>Table 3- 8 Use Case Description of Edit Transaction Module</i>	8
<i>Table 3- 9 Use Case Description of Add Category Module</i>	9
<i>Table 3- 10 Use Case Description of Add Budget Module</i>	9
<i>Table 3- 11 Use Case Description of View Budget Module</i>	10
<i>Table 3- 12 Use Case Description of Budget Detail Module</i>	10
<i>Table 3- 13 Use Case Description of Track Achievements Module</i>	11
<i>Table 3- 14 Use Case Description of Add Loan Module</i>	12
<i>Table 3- 15 Use Case Description of View Report Module</i>	13
<i>Table 3- 16 Use Case Description of View Financial Health Score Module</i>	13
<i>Table 3- 17 Use Case Description of Input Query</i>	14
<i>Table 3- 18 Use Case Description of Capture Image</i>	14
<i>Table 3- 19 Use Case Description of Verify Receipt Data</i>	15
<i>Table 3- 20 Use Case Description of Make Prophet Forecast</i>	15
<i>Table 5- 1 Specifications of laptop</i>	69
<i>Table 5- 2 Development Tools and Software Requirements</i>	69

LIST OF ABBREVIATIONS

<i>API</i>	Application Programming Interface
<i>PFM</i>	Personal Finance Management
<i>AI</i>	Artificial Intelligence
<i>GPT</i>	Generative Pre-trained Transformer
<i>OCR</i>	Optical Character Recognition

Chapter 1 Introduction

The landscape of personal financial management (PFM) has seen profound changes over the past few decades, driven by technological advancements and shifting consumer expectations. Historically, managing personal finances involved manual bookkeeping with individuals tracking their income, expenses, and savings using paper-based methods like ledgers and simple budgets. This approach was not only labor-intensive but offered limited insights into financial health and planning [1].

However, the advent of digital technology in the last two decades has significantly shifted this paradigm towards more automated and sophisticated financial management tools [2]. These modern PFM tools have revolutionized the way individuals manage their finances, providing features such as automated transaction tracking, enhanced budgeting capabilities, and real-time financial insights. They have become integral in helping users make informed financial decisions, thereby enhancing overall financial literacy and management capabilities

Despite these advancements, contemporary PFM tools still face significant challenges, particularly in delivering real-time, personalized feedback and interactive financial planning. These limitations affect user engagement and the overall effectiveness of financial management tools, highlighting the need for innovative solutions that cater to today's tech-savvy consumers and their individual financial needs [2].

This project places a strong emphasis on personal financial health, aiming to transform passive financial management into an interactive, real-time experience. The development of this new Personal Finance Management Mobile Application focuses on providing users with enhanced financial awareness and decision-making capabilities through more tailored and responsive tools. The goal is to foster better financial health and empower users to make smarter financial decisions.

The focus on personal financial health is supported by research highlighting the crucial role of financial literacy, effective financial behavior, and the adverse impacts of financial stress. Recent studies underscore the importance of financial literacy in navigating today's intricate financial landscape. For instance, research reveals that financial literacy is crucial for making informed decisions about saving, investing, and debt management, particularly in times of economic turmoil, such as the post-pandemic period of high inflation [3]. The decline in the financial health of U.S. adults from 34% in 2021 to 31% in 2022 further emphasizes the need for tools that support financial well-being, as financial stress and inadequate financial management can lead to significant declines in overall quality of life [4].

Moreover, specific research focusing on low-income groups in Malaysia highlights the direct correlation between financial behavior, literacy, and financial well-being. This study shows that improving financial literacy and behavior can significantly enhance financial stability and reduce stress, which is especially critical for marginalized communities [5]. Additionally, integrating financial education through internet-based platforms has proven effective in increasing financial awareness and fostering positive financial behaviors among students, demonstrating the potential of technology in financial education [6].

Further supporting this focus, the 2023 RinggitPlus Malaysian Financial Literacy Survey reveals significant insights into financial behaviors and literacy levels among Malaysians. For instance, 75% of female respondents save less than RM500 per month compared to 66% of men, with only 44% of female respondents having started investing [7]. Additionally, a national strategy report highlights that one in three Malaysians consider themselves to have "low financial knowledge," particularly among low-income households, with one in ten admitting to a lack of discipline in managing finances [8]. These statistics underscore the critical need for tools that can enhance financial literacy and management, particularly among vulnerable groups.

By integrating insights from these studies into the app, the project aims to provide users with a comprehensive tool to manage their finances effectively, improve their financial literacy, and ultimately achieve better **personal financial health**. This approach not only addresses current

inadequacies in financial management tools but also pioneers the integration of AI and predictive analytics to offer a more interactive and personalized financial management experience.

1.1 Problem Statement and Motivation

1.1.1 Inadequacies in Current Financial Management Tools

Current financial management tools often fall short in providing real-time, personalized feedback, essential for effective financial management. Most tools available today deliver static and generic financial information, failing to meet the dynamic needs of users aiming to improve their financial health actively. This gap in the financial technology landscape highlights a critical need for a more responsive and individualized approach to financial management. The increasing complexity of financial needs suggests a growing demand for tools that provide personalized insights to help manage finances more effectively [9].

This project is motivated by the need to provide users with enhanced financial awareness and decision-making capabilities through more tailored and responsive tools. The aim is to transform passive financial management into an interactive, real-time experience that adapts to individual user needs, fostering better financial health and smarter financial decisions.

1.1.2 Limited Interactive Support in Personal Finance Management Apps

Another significant challenge in the domain of personal finance management is the general lack of user-engaging interactive elements, leading to user disengagement. The predominant use of static forms and data dashboards does not support an engaging or intuitive user experience, making financial management seem more daunting than it should be. Personalized, real-time interactions based on user data are essential to enhance user engagement and satisfaction in digital platforms, including financial services [10].

The second motivation of this project is to seek to revolutionize the user experience in personal finance apps by making financial guidance more interactive and accessible. The goal is to make the management of personal finances not just simpler but also more engaging, thereby encouraging users to take an active interest in their financial health.

1.1.3 Inadequate Financial Forecasting and Planning

Many users struggle with long-term financial planning due to the absence of advanced forecasting tools in existing personal finance apps. Current tools are mainly focused on tracking capabilities without offering the predictive insights that are crucial for effective planning and decision-making. Real-time feedback systems that integrate AI can significantly enhance decision-making capabilities, which is crucial for effective long-term financial planning [11].

The third motivation of this project is driven by the need to provide users with tools that offer predictive insights, enabling them to better anticipate future financial conditions. By enhancing the forecasting capabilities, the project will support users in achieving greater financial stability and making more informed decisions, addressing a critical gap in existing financial management tools.

1.2 Objectives

This mobile application development aims to fundamentally transform the landscape of personal finance management by introducing innovative solutions that respond to the evolving needs of today's users. The project is centered around enhancing interactivity, personalization, and predictive capabilities within a mobile application environment. By focusing on these key areas, the application seeks to address existing gaps in financial management tools, providing users with a more intuitive, engaging, and forward-thinking financial management experience. Below are the specific objectives that this project aims to achieve:

1.2.1 Develop a Personalized Financial Health Score System

This project aims to enhance the personal financial management experience by developing a Personalized Financial Health Score System within the app. This system will leverage real-time data and personalized insights to provide users with immediate feedback tailored to their specific financial activities. The objective is to create a dynamic, user-centric tool that helps individuals proactively manage their finances, enhancing their ability to make informed financial decisions swiftly.

1.2.2 Implement an AI Chatbot for Interactive Financial Guidance

The second objective focuses on significantly improving user interaction within the personal finance management application by implementing an AI-powered chatbot. Utilizing OpenAI's GPT model, this chatbot will facilitate real-time conversations, offering basic financial guidance and responses based on the analysis of user data stored in Firebase. The chatbot aims to make financial management more accessible and engaging, thereby increasing user engagement and satisfaction. However, this objective does not extend to providing complex financial advice or replacing human financial advisors.

1.2.3 Implement Predictive Analytics for Financial Forecasting

The third objective of this project is to integrate predictive analytics using Facebook Prophet, a time series forecasting model, within the personal finance app. By analyzing historical financial data stored in Firebase's Cloud Firestore, the app will generate expense forecasts to help users anticipate future financial trends. This feature will enhance users' ability to plan and make informed decisions about their financial goals. However, this objective does not extend to providing highly accurate or long-term financial projections beyond typical patterns.

1.3 Project Scope and Direction

This project focuses on delivering a mobile application for personal financial management, integrating enhanced functionalities such as interactive user experience, predictive analytics, and personalized financial assessments. The key features include a Personalized Financial Health Score System using real-time data to assess financial well-being, an AI-Powered Chatbot providing

interactive guidance via OpenAI's GPT model, and a Predictive Analytics Module using Facebook Prophet to forecast financial trends.

All components are integrated into a unified mobile application, emphasizing user-friendly design and financial foresight. The scope covers the design, development, and integration of these software features but does not extend to additional services like stock trading or investment management. The project remains strictly focused on software solutions aimed at improving personal financial management, excluding any hardware development.

1.4 Contributions

This project introduces an innovative mobile application designed to transform personal financial management through innovative technologies, aiming to make a significant societal impact by enhancing financial literacy and decision-making capabilities across diverse populations. By integrating features such as the Personalized Financial Health Score System, an AI-powered chatbot, and predictive analytics using Facebook Prophet, the application empowers users to take control of their finances with tailored insights and proactive management tools. These contributions democratize financial literacy, reduce economic disparities, and provide personalized guidance that encourages better financial decision-making.

The significance of the project goes beyond individual empowerment. It sets a new standard for personal finance management by integrating artificial intelligence and predictive analytics into an interactive platform. This contribution marks a key advancement in the fintech industry, offering solutions that are both responsive and user-friendly, while setting a precedent for future financial management applications.

1.5 Report Organization

This report is organized into several chapters to comprehensively cover the project. Chapter 1 introduces the problem statement, objectives, scope, contributions, and report structure. Chapter 2

focuses on reviewing existing literature and technologies relevant to personal financial management applications. Chapter 3 presents the system methodology and design, including key diagrams such as the system architecture and use case diagrams. In Chapter 4, the system's components, design, and interaction operations are explained. Chapter 5 details the implementation of the system, including hardware and software setups. Chapter 6 evaluates the system through testing, performance metrics, and objective evaluation. Finally, Chapter 7 concludes the report with a summary of findings and recommendations for future work.

Chapter 2 Literature Review

2.1 Review of the Technologies

2.1.1 OpenAI GPT for Chatbots

The implementation of AI-driven chatbots in the fintech industry has been transformative, providing businesses with powerful tools to enhance customer support and operational efficiency. OpenAI's GPT-based models are particularly effective due to their natural language processing capabilities, enabling them to understand and respond to complex financial queries in a human-like manner. Unlike traditional scripted chatbots, GPT-powered systems can converse dynamically, providing personalized financial advice and tailored responses based on user data. This makes them an invaluable asset for financial institutions, allowing them to improve customer experience and streamline support processes [12] [13].

This project is motivated by the need to provide users with enhanced financial awareness and decision-making capabilities through more tailored and responsive tools. The aim is to transform passive financial management into an interactive, real-time experience that adapts to individual user needs, fostering better financial health and smarter financial decisions.

Strengths of GPT-powered Chatbots

1. Understands complex financial queries and provides tailored advice.
2. Reduces operational costs by automating routine customer support tasks.
3. Offers real-time assistance, improving user satisfaction and loyalty.
4. Enhances staff productivity by handling repetitive tasks efficiently.
5. Easy integration into existing systems with minimal setup.
6. Strong security features, including multi-factor authentication and end-to-end encryption for safe user data management.

Weaknesses of GPT-powered Chatbots

1. Prone to generating incorrect or irrelevant responses due to its inability to reason or make rational judgments.
2. May reflect biases from its training data, leading to skewed or potentially harmful outputs.
3. Regular updates and fine-tuning are required to maintain accuracy and minimize the risk of AI hallucinations, where the model confidently produces unsupported information.

2.1.2 Facebook Prophet for Time Series Forecasting

Facebook Prophet is an open-source tool designed to simplify time series forecasting, making it accessible for users with limited statistical expertise. Built to handle various challenges like trends, seasonality, and holidays, it provides an intuitive interface for predicting future data trends based on historical data. Unlike traditional methods such as ARIMA, Prophet automates much of the process, offering ease of use while incorporating modern machine learning techniques to improve forecasting accuracy [14].

Prophet's model decomposes time series into three components: trend, seasonality, and exceptional events (holidays). This enables it to handle complex data patterns with minimal user intervention. Its ability to adjust for seasonal trends and special events makes it particularly useful in industries such as finance, healthcare, and e-commerce.



Figure 2- 1 Facebook Prophet

Strengths of Facebook Prophet

1. User -friendly and accessible to non-experts.
2. Automatically adjusts for trends, seasonality, and holidays.
3. Handle missing data and outliers by removing them.
4. Suitable for multiple types of seasonality (daily, weekly, yearly).
5. Requires minimal parameter tuning.
6. Available in R or Python

Weaknesses of Facebook Prophet

1. Assume additive or multiplicative models, limiting flexibility for certain data types.
2. Sensitive to significant outliers that can distort predictions.
3. Struggles with complex, non-linear dependencies.
4. May require preprocessing for irregular, high-frequency data.

Model Fitting and Forecasting with Prophet

1. **Data Preparation:** Prepare input data in a two-column data frame with a time variable and observed values.
2. **Model Initialization:** Initialize the Prophet model with desired parameters like growth (linear/logistic) and seasonality mode.
3. **Model Fitting:** Fit the model to historical data using the fit() method.
4. **Forecast Generation:** Generate forecasts using predict(), providing uncertainty intervals.
5. **Model Evaluation:** Perform diagnostics, cross-validation, and assess performance metrics with built-in tools.

2.1.3 OCR for Receipt Scanning

Optical Character Recognition (OCR) plays a crucial role in financial applications, transforming printed and handwritten texts into machine-readable formats. This technology is especially beneficial in automating tasks like extracting data from receipts and invoices, which improves the efficiency of data entry and financial reporting. Google Cloud Vision OCR is a powerful tool in this space, offering deep learning techniques to recognize and interpret text from various images, including documents and complex forms. The platform provides two modes: Text Annotation for sparse text and Document Text Annotation for dense text, allowing for high accuracy in extracting financial details such as transaction amounts and vendor information [15].

Incorporating OCR into financial systems significantly reduces the risk of manual data entry errors and enhances operational efficiency. Google Cloud Vision OCR leverages machine learning to improve the accuracy of text detection across different lighting conditions and languages, making it a suitable solution for large-scale processing of financial documents. The system's scalability, combined with its cloud-based infrastructure, allows it to handle large volumes of documents efficiently, making it ideal for use in sectors such as retail, banking, and personal finance management [15].

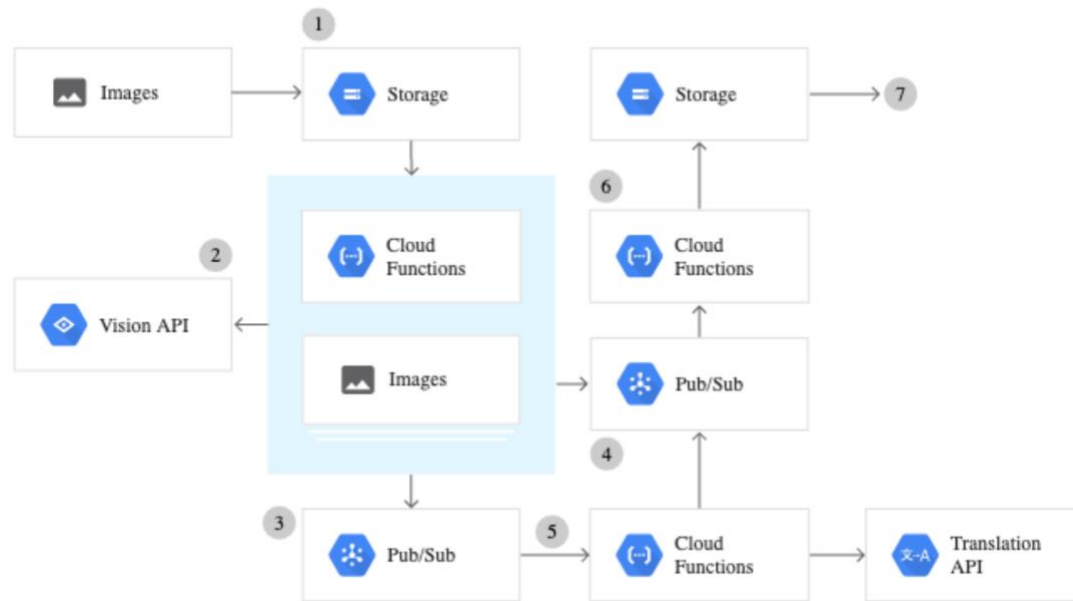


Figure 2- 2 Overview of Google Cloud Vision OCR

Strengths of Google Cloud Vision OCR

1. High accuracy in both printed and handwritten text detection across different lighting conditions.
2. Scalability for processing large amounts of data via cloud-based infrastructure.
3. Seamless integration with Google Cloud services such as Cloud Storage and BigQuery.
4. Support for multiple languages, enabling global application in various industries.
5. Secure and compliant with data protection laws
6. Cost-effective pay-as-you-go model, suitable for various business sizes.

Weaknesses of Google Cloud Vision OCR

1. Limited accuracy for distorted or noisy images, especially in non-standard conditions.
2. Dependency on high-quality images for optimal performance.

3. Cloud-based, requiring an internet connection for processing and not available offline.
4. Vulnerable to misalignment issues, particularly when dealing with improperly rotated or off-center text.

2.1.4 Summary of the Technologies Review

The three key technologies reviewed—OpenAI GPT, Facebook Prophet, and OCR (Optical Character Recognition)—highlight distinct strengths and limitations in the context of financial applications. OpenAI GPT has demonstrated significant capabilities in automating customer interactions via chatbots, leveraging natural language processing to respond to complex financial queries and provide personalized advice. This enhances customer service efficiency and reduces operational costs, but the technology still faces challenges related to biases in training data and potential inaccuracies. Regular fine-tuning is required to address these concerns and mitigate the risks of incorrect or skewed responses.

Facebook Prophet offers a user-friendly and accessible solution for time series forecasting, making it particularly useful for non-experts in handling trends and seasonality in financial data. Its strengths lie in its ability to handle missing data, outliers, and multiple seasonalities with minimal parameter tuning. However, its assumption of additive or multiplicative models and sensitivity to significant outliers can limit its flexibility in handling certain data types, especially those with complex, non-linear dependencies.

OCR, particularly through Google Cloud Vision OCR, plays a crucial role in automating data extraction from financial documents, such as receipts and invoices. It significantly reduces manual data entry errors and enhances operational efficiency, especially in large-scale financial processing environments. The technology excels in text recognition under various lighting conditions and supports multiple languages, making it a versatile solution. However, its reliance on high-quality images and cloud-based processing

introduces challenges in non-standard conditions and offline scenarios.

In summary, while each of these technologies brings unique strengths to the table, they also have limitations that need to be managed, particularly in terms of data quality, regular updates, and contextual accuracy. Their integration into financial systems offers great potential for improving efficiency, reducing costs, and enabling more accurate data analysis, but careful consideration of their weaknesses is essential for optimal performance.

2.2 Review of the Existing Applications

2.2.1 Existing System A: Money Lover

Money Lover is an intuitive expense-tracking application that serves as an excellent alternative for individuals who find traditional budgeting methods challenging or cumbersome. Developed by the Vietnamese company Finsify in 2011, Money Lover seamlessly integrates a variety of functions into a single, user-friendly platform. These functions include detailed expense tracking, efficient budgeting, comprehensive reporting, and financial planning tools [16].

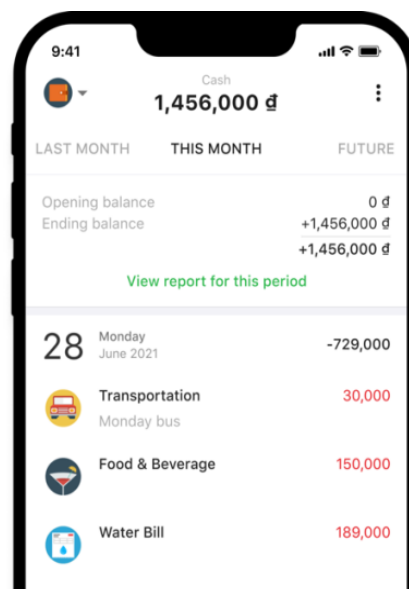


Figure 2- 3 Money Lover Mobile Application

Basic Functionalities of Money Lover

1. Users can track daily transactions.
2. Users can manage budgets and get financial overviews.
3. Users can develop one or more budget plans.
4. Users can schedule bill payments for recurring transactions.

Additional Features of Money Lover

1. Users can synchronize data across multiple devices.
2. Users have travel mode, saving plans, and receipt-scanning features.

Strengths of Money Lover

1. Money Lover is available across various platforms, including mobile and web applications, and is even compatible with the Apple Watch.
2. Money Lover offers detailed tutorials and guidance for new users to facilitate a smooth onboarding process.
3. Money Lover employs SSL encryption to ensure the security and privacy of user data.
4. Money Lover caters to a global audience by offering functionality in several languages and supporting various currencies.
5. Money Lover does provide advanced predictive analytics for financial forecasting.

Weaknesses of Money Lover

1. Money Lover predominantly supports banks located in Asian countries, offering limited options for others.
2. Users are unable to transfer their existing financial data into the app.
3. Money Lover does not provide a personalized financial health score system.
4. Money Lover does not have an AI chatbot for interactive financial guidance.
5. Money Lover lacks elements designed to enhance user interaction and provide motivational incentives for financial management.

2.2.2 Existing System B: Expensify

Launched in 2008 by a team of University of Michigan students, Expensify has evolved into a key application for managing travel and expenses, particularly adept at scanning receipts, tracking both personal and business expenses, and handling bill payments. It stands out for its utility to individuals who tend to lose paper receipts, offering a digital solution for storing and analyzing such slips. The application doesn't just benefit individuals; it's also highly appropriate for large corporations that need to manage and verify their employees' financial documents, including receipts, thereby streamlining expense tracking and reimbursements. Over the years, Expensify has been recognized with a host of awards, like App Partner of the Year and Innovation Partner of the Year, and has been consistently rated as a top-tier expense management software [17].

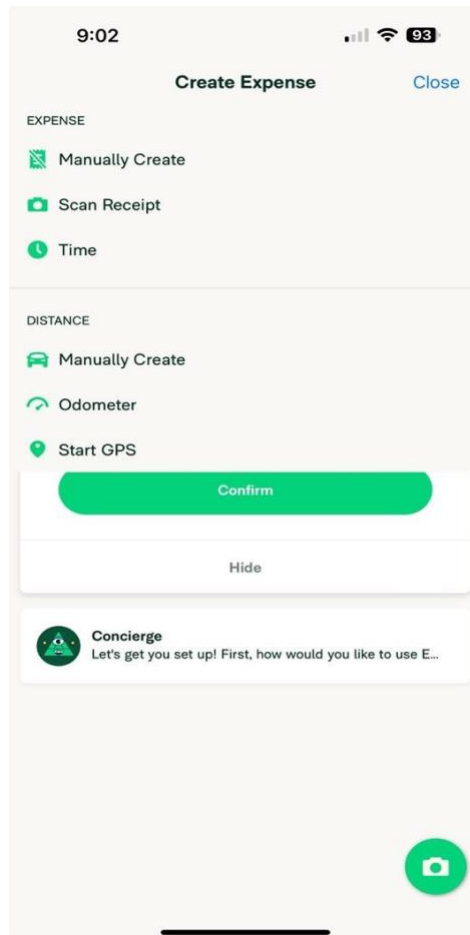


Figure 2- 4 Expensify Mobile Application

Basic Functionalities of Expensify

1. Users can scan and digitize receipts for tax and personal expenses.
2. Users can send their expense reports to others.
3. Users can control company spending and manage expenses.
4. Users can collect receipts from teams and clients.
5. Users can handle reimbursements and pay bills.

Additional Features of Expensify

1. Expensify enables users to create invoices for vendors.
2. Users can track mileage via GPS, by inputting the distance manually, or by logging odometer readings.
3. Expensify offers users the opportunity to earn cash back through the use of a premier corporate credit card.
4. Expensify provides an offline mode feature for continuous functionality without internet access.
5. Expensify provides a built-in free chat tool, Expensify Chat, which streamlines collaboration for self-employed professionals and small businesses with their teams and external partners like customers and suppliers.

Strengths of Expensify

1. Expensify enhances the management of group expenses and streamlines employee reimbursement processes for improved efficiency.
2. Expensify boasts high accessibility, offering both a mobile app and a web platform for user convenience.
3. Expensify provides continuous, 24/7 customer support through an AI-driven chatbot.

Weaknesses of Expensify

1. Users may encounter inconsistencies with the accuracy of Expensify's SmartScan feature.
2. Users have limited banking options with Expensify, which partners with fewer than 20 banks, primarily in the United States.
3. Expensify does not provide a personalized financial health score system.
4. Expensify does not offer advanced predictive analytics for financial forecasting.
5. Expensify does not incorporate user engagement or motivational features to enhance the user experience.

2.2.3 Existing System C: Spendee

Spendee, a mobile application for budgeting and tracking money, was established in 2017 by a team of young entrepreneurs and quickly gained prominence in the FinTech industry, particularly after two years of its release. The app, available on both Android and iOS platforms, has been recognized with several achievements, including features in the Apple App Store and Google Play, and winning the Mobile UX 2017 award [18].

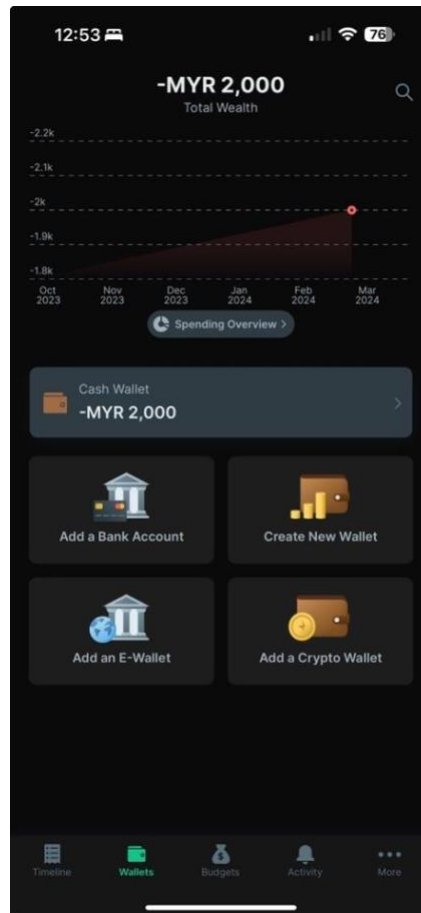


Figure 2- 5 Spende Mobile Application

Basic Functionalities of Spende

1. Spende enables users to monitor their cash flow on a daily, weekly, monthly, and yearly basis.
2. Users have the capability to import and export data.
3. Spende offers the functionality of scheduling transactions for future dates.

Additional Features of Spende

1. Spende allows for connection to both cryptocurrency wallets and e-wallets.
2. Spende enables the merging of two or more expense or income categories for streamlined categorization.
3. Spende offers a bulk editing feature, allowing users to modify categories, dates, hashtags, and notes across multiple transactions with a single action.

Strengths of Spendee

1. Spendee is available across various platforms, including mobile and web applications.
2. Spendee ensures a high level of security with features like biometric authentication for app access and encrypted connections and storage for banking data.
3. Users have the capability to add and manage cryptocurrency wallets within Spendee.
4. Spendee offers comprehensive tutorials and guidance, ensuring new users can navigate the system without feeling overwhelmed or confused.

Weaknesses of Spendee

1. Spendee does not offer a feature for monitoring credit.
2. Spendee lacks a bill payment functionality within the app.
3. Spendee is not compatible with syncing data from all banks.
4. Spendee does not provide a personalized financial health score system.
5. Spendee does not have an AI chatbot for interactive financial guidance.
6. Spendee lacks elements designed to enhance user interaction and provide motivational incentives for financial management.

2.2.4 Existing System D: YNAB

"You Need a Budget," commonly referred to as YNAB, was established in 2004 by a husband-and-wife team, Julie and Jesse, and offers a distinctive approach to personal budgeting software. Diverging from traditional budgeting methods that depend on preset categories, YNAB encourages users to assign every dollar of their income a specific job in the budget, closely aligning with the zero-based budgeting technique. This strategy is foundational to YNAB's philosophy, which is built upon four main rules aimed at helping

users live within their means, eliminate debt, save money, and break the cycle of living paycheck-to-paycheck, differentiating it from other budgeting tools in the market [19].

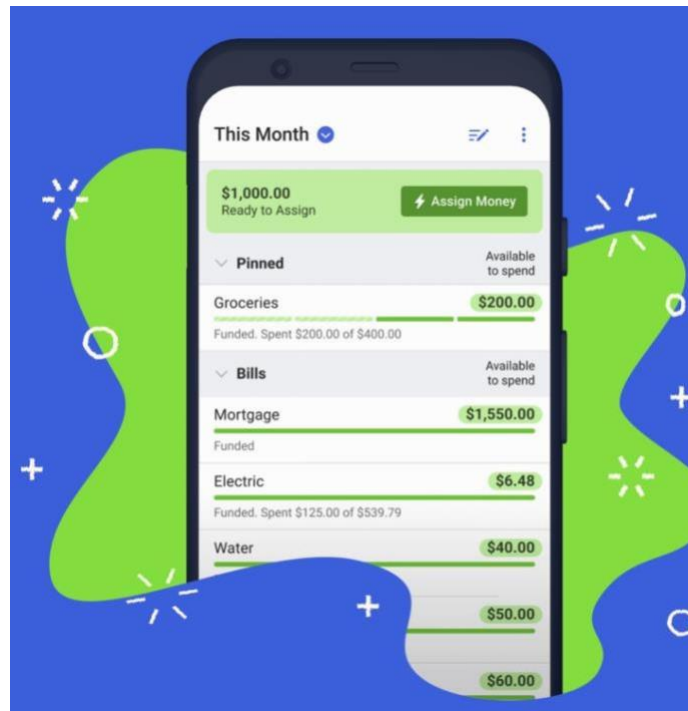


Figure 2- 6 YNAB Mobile Application

Basic Functionalities of YNAB

1. Users receive visual charts for a comprehensive view of their financial status.
2. YNAB offers three methods for users to set and achieve financial goals, including monthly funding goals for long-term objectives.
3. Users can use YNAB's target tracking feature to set and monitor specific financial targets, such as saving a particular amount or eliminating credit card debt.
4. YNAB smart reminders notify users when they are overspending, helping them stay on track.

Additional Features of YNAB

1. YNAB enables cloud syncing, allowing users to back up their budget files and

keep transactions updated.

2. Users can secure their YNAB account and financial information with a PIN code.
3. YNAB customer support includes a comprehensive and user-friendly self-help platform.

Strengths of YNAB

1. YNAB is capable of synchronizing with over 12,000 banking institutions.
2. YNAB allows for seamless transitions between multiple budget plans without the need to start over.
3. YNAB supports synchronization across multiple devices for user convenience.
4. Users can tailor YNAB's categories to fit their personal budgeting needs.
5. Users have the flexibility to export data from YNAB as needed.

Weaknesses of YNAB

1. YNAB does not include a feature for tracking investments.
2. YNAB lacks both bill tracking and bill payment features.
3. YNAB does not include a personalized financial health score system.
4. YNAB does not have an AI chatbot for interactive financial guidance.
5. YNAB lacks elements designed to enhance user interaction and provide motivational incentives for financial management.

2.2.5 Existing System E: Mint

Mint, a pioneering budgeting application from Intuit since 2006, offers a comprehensive financial overview by consolidating all personal finance accounts in one place. Recognized for its user-friendly interface, Mint provides real-time insights into your finances, empowering you with automatic updates and a customizable budgeting system for optimal financial management [20].

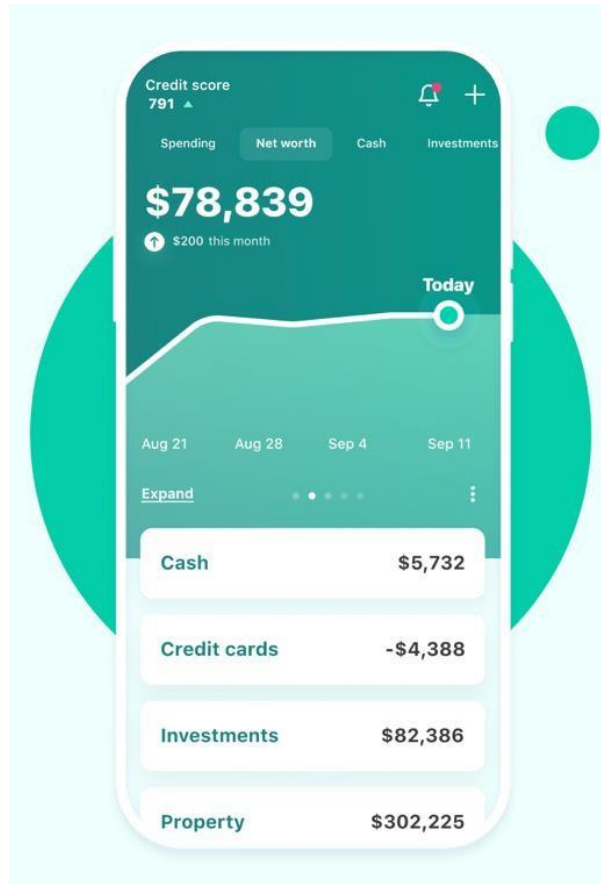


Figure 2- 7 Mint Mobile Application

Basic Functionalities of Mint

1. Users can create and manage a personalized budget.
2. Users receive reminders for bill payments within the app.
3. Mint aggregates all user financial accounts for a unified view.
4. Mint tracks spending automatically, categorizing transactions for easy review.

Additional Features of Mint

1. Users can view the performance of their investments through Mint.
2. Users can track expenditures by category to analyze monthly cash flow.
3. Mint monitors users' credit scores regularly.

4. Mint sends automatic alerts to users about financial matters such as late fees and over-budget spending.

Strengths of Mint

1. Users receive financial summaries and alerts via email or text message from Mint.
2. Users can set alerts for unusual account activity, bill reminders, and low balances through Mint.
3. Mint provides users with a free credit score, courtesy of Equifax.
4. Mint allows for easy customization and provides users with digestible financial reports.
5. Mint automatically categorizes transactions, simplifying budget tracking for users.

Weaknesses of YNAB

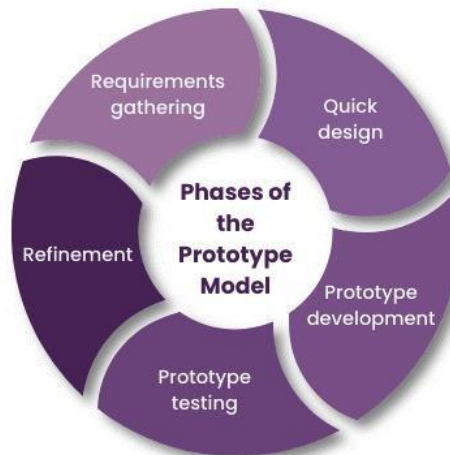
1. Users are unable to assign multiple savings goals to a single account within Mint.
2. Users cannot pay bills directly through Mint.
3. Mint sometimes has issues with account synchronization.
4. Mint lacks features for tracking investments.
5. Mint does not support multiple currencies for transactions.
6. Mint does not feature advanced predictive analytics for financial forecasting.
7. Mint does not have an AI chatbot for interactive financial guidance.
8. Mint lacks elements designed to enhance user interaction and provide motivational incentives for financial management.

2.2.6 Comparative Summary of Existing Systems and Proposed Application

Table 2- 1 Comparative Overview of Key Functionalities and Weaknesses of Popular Money Management Mobile Apps

Functionalities	Money Lover	Expensify	Spendee	YNAB	Mint	Proposed Application
Basic Functionalities						
Track daily transactions	✓	X	✓	X	✓	✓
Manage budgets	✓	X	X	✓	✓	✓
Schedule bill payments	✓	✓	X	X	✓	✓
Financial overviews	✓	X	✓	X	✓	✓
Expense tracking	✓	✓	✓	X	✓	✓
Synchronize data across devices	✓	X	X	✓	X	X
Offers additional features like travel mode, saving plans.	✓	✓	X	X	X	X
personalized financial health score system	X	X	X	X	X	✓
AI chatbot	X	✓	X	X	X	✓
Support Multicurrency	✓	✓	✓	✓	✓	✓
Loan calculations features	X	X	X	X	X	✓
Track and Unlock Achievements	✓	✓	✓	✓	✓	✓
predictive analytics for financial forecasting	X	X	X	X	X	✓
Receipt Scanning	✓	✓	X	X	X	✓

Chapter 3 System Methodology/Approach



*Figure 3- 1 Phase of Prototyping Methodology
Source: (theknowledgeacademy, 2023)*

The project utilizes the Prototype Model, which is ideal for an iterative approach that focuses on developing and refining an operational prototype [21]. The initial phase, requirement gathering, is specifically tailored to define and isolate functionalities critical for the Personalized Financial Health Score System. This phase leverages secondary research and internal analysis to identify essential features, ensuring a targeted approach to system development.

Following the requirement gathering, a preliminary design is established, outlining the basic structure and user interface that supports the integration of the Personalized Financial Health Score System. This stage sets the foundation for the initial prototype, concentrating on embedding and evaluating this primary feature for effectiveness and user interaction. Subsequent testing of the prototype identifies usability and functional gaps, leading to iterative refinements based on the feedback received. Each cycle aims to enhance the prototype's performance and user engagement, progressively advancing towards a robust final product.

3.1 System Design Diagram

3.1.1 System Architecture Diagram

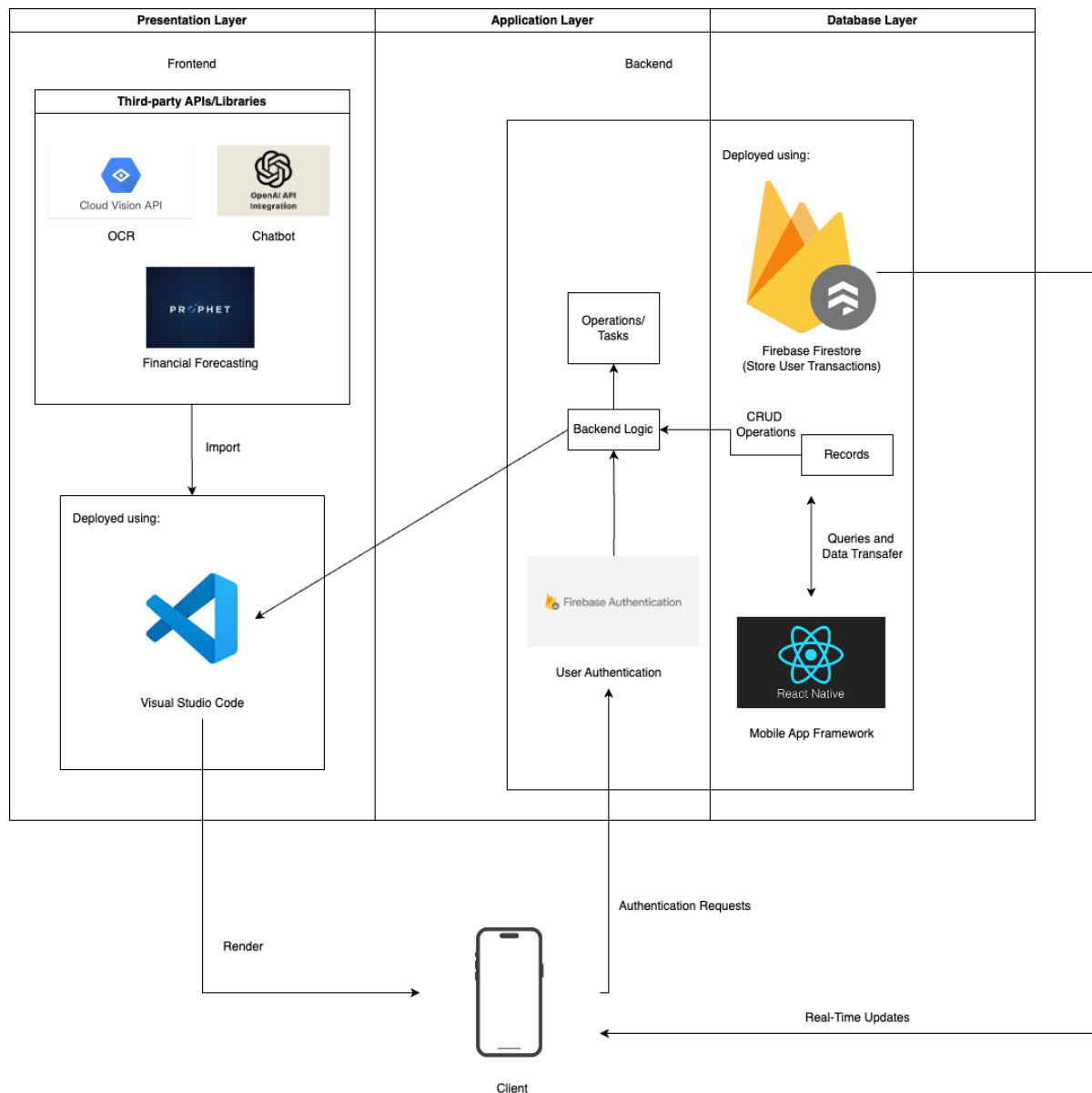


Figure 3- 2 System Architecture Diagram of the proposed system

Figure 3- 2 showcases the interaction between various layers of the system: Presentation Layer, Application Layer, and Database Layer. In the Presentation Layer, third-party APIs (OpenAI, Cloud Vision, and Prophet) handle chat, OCR, and financial forecasting, while Visual Studio Code is the development environment. The Application Layer includes Firebase Authentication for user verification and backend logic for operations. The Database Layer utilizes Firebase Firestore for CRUD operations and real-time data updates, with the Mobile App Framework (React Native) handling data transfers and app functions.

3.1.2 Use Case Diagram and Description

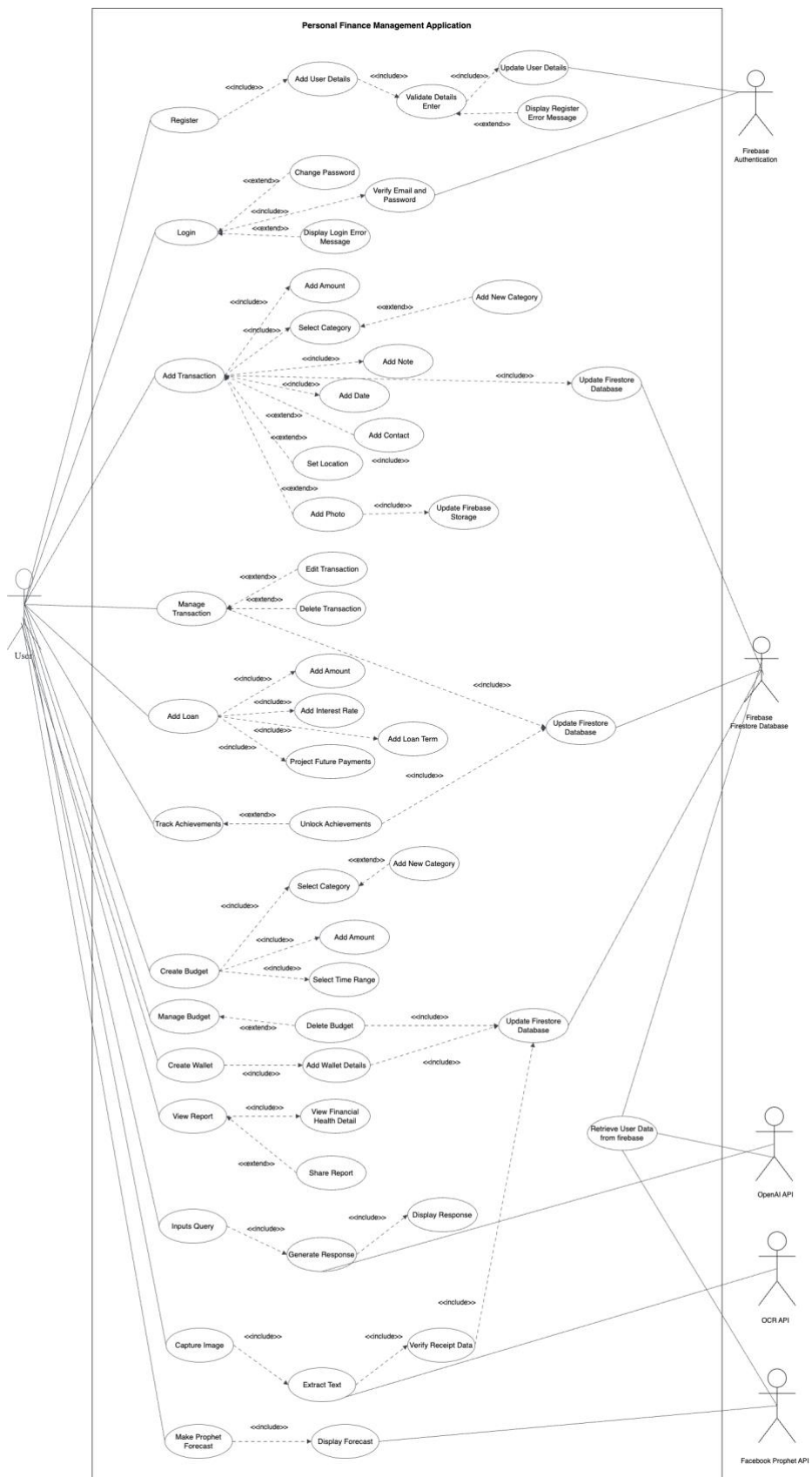


Figure 3- 3 Use Case Diagram for the proposed application

Figure 3.3 above illustrates the interactions between the system's six actors: the user, Firebase Authentication, Firebase Firestore Database, OpenAI API, OCR API, and Facebook Prophet API. Each actor engages with the application through a series of specific use cases, which are detailed further in the sections below.

Table 3- 1 Use Case Description of Login Module

Use case	Login	ID	1
Aim	To authenticate a user to grant access to the application	Importance Level	High
Actor	User and Firebase Authentication		
Trigger	The user initiates the login process by submitting email and password credentials.		
Pre-condition	The user must have an internet connection and a pre-existing account in the Firebase system.		
Main flow	<ol style="list-style-type: none"> 1. The system presents fields for the user to enter email and password. 2. It validates the provided credentials with Firebase Authentication. 3. Upon successful validation, the user is navigated to the 'About' screen with their user ID. 		
Alternative flow	<ol style="list-style-type: none"> 2a. If the credentials are incorrect, the system alerts the user. 2b. The user remains on the login screen to attempt to enter their credentials again or navigate to registration or password recovery options. 		

Table 3- 2 Use Case Description of Register Account Module

Use case	Register Account	ID	2
Aim	To create a new user account within the application	Importance Level	High
Actor	User and Firebase Authentication		
Trigger	The new user starts the registration process by filling out the registration form.		
Pre-condition	The new user must have a valid email address and access to the internet.		

Main flow	<ol style="list-style-type: none"> 1. The system displays input fields for display name, email, password, confirm password, gender selection, and birth date. 2. Validates input completeness and password confirmation match. 3. Registers the user using Firebase Authentication and updates the profile. 4. Stores additional user information in Firebase Firestore.
Alternative flow	<ol style="list-style-type: none"> 2a. In case of any input validation failure, the user is alerted with the specific issue to correct. 3a. If Firebase authentication or Firestore operations fail, the user is informed of the error. <p>The user remains on the registration screen until all criteria are met and the registration is successful or the user navigates away.</p>

Table 3- 3 Use Case Description of Change Password Module

Use case	Change Password	ID	3
Aim	To provide users with a means to reset their password if they have forgotten it.	Importance Level	High
Actor	User and Firebase Authentication		
Trigger	The user clicks the Forget Password option.		
Pre-condition	The user must have a valid email address that is registered with the app.		
Main flow	<ol style="list-style-type: none"> 1. User enters their email address into the TextInput field. 2. User clicks the 'Send email' button to request a password reset. 3. System validates the email address and sends a password reset email if it is associated with an account. 4. User receives a confirmation alert that the password reset email has been sent. 5. The user is navigated back to the 'Login' screen. 		
Alternative flow	2a. If an error occurs (email not being recognized or a network issue), the user is alerted with the error message.		
Exception Flows	If the email field is empty or not valid, prevent the password reset email from being sent and alert the user to correct it.		

Table 3- 4 Use Case Description of Create Wallet Module

Use case	Wallet Creation	ID	4
Aim	To enable users to create a new wallet within the app, specify its name, select a currency, set financial goals, and define target savings.	Importance Level	High
Actor	User		
Trigger	The user navigates to the "CreateWallet" screen and initiates the wallet creation process		
Pre-condition	The user must be authenticated and have an account within the app.		
Main flow	<ol style="list-style-type: none"> 1. User is prompted to enter a wallet name and select a currency from a dropdown list. 2. User can select a financial goal from another dropdown list. 3. User enters the target savings amount they aim to achieve. 4. Upon submission, the system validates the input and creates a wallet document in the database linked to the user's account. 5. On successful creation, the user is shown a success message and navigated to the "MyMenu" screen. 		
Alternative flow	4a. If the wallet name is empty or currency is not selected, the system displays an error message in a modal.		

Table 3- 5 Use Case Description of Add Transaction Module

Use case	Add Transaction	ID	5
Aim	To record a new financial transaction within the application.	Importance Level	High
Actor	User and Firebase Firestore database		
Trigger	The user chooses to add a new transaction.		
Pre-condition	The user must be logged in and have access to transaction details.		
Main flow	<ol style="list-style-type: none"> 1. User inputs the transaction amount and selects a category. 		

	<p>2. User can add a note, select a date, and add additional details such as a person involved or location.</p> <p>3. User can upload an image related to the transaction.</p> <p>4. The system saves the transaction to the Firebase Firestore database.</p>
Alternative flow	3a. If image upload fails, the user is notified and can attempt to re-upload or proceed without an image.
Exception Flows	If required fields are missing or there's an error saving the transaction, the user is alerted to correct the issues.

Table 3- 6 Use Case Description of View Transaction Module

Use case	Displaying User's Transaction List	ID	6
Aim	To provide users with a list of their transactions, grouped by date, with the ability to navigate to the details of each transaction.	Importance Level	Moderate
Actor	User		
Trigger	The user navigates to the "TransactionList" screen within the app.		
Pre-condition	The user must be logged in and the app must have access to the internet to fetch transaction data.		
Main flow	<p>1. On component mount, set the loading state to true and fetch wallet and transaction data from Firestore.</p> <p>2. Group transactions by date and set the sections state with the grouped transactions.</p> <p>3. Once data is fetched, set the loading state to false and display the transactions in a SectionList.</p> <p>4. User can tap on a transaction to navigate to a detailed view of that transaction.</p>		
Alternative flow	1a. If the fetch operation fails, display an error alert to the user.		
Exception Flows	1b. If there are no transactions to display, show a "No transactions" message.		

Table 3- 7 Use Case Description of Transaction Detail Module

Use case	Transaction Detail Viewing and Operations	ID	7
Aim	To display detailed information about a transaction and provide options to export the details, edit, or delete the transaction.	Importance Level	Moderate
Actor	User		
Trigger	The user selects a transaction from the transaction list to view its details.		
Pre-condition	The user must have selected a transaction from the transaction list and transaction details must exist and be passed to the component.		
Main flow	<ol style="list-style-type: none"> 1. On loading the TransactionDetail screen, fetch the wallet's currency and display transaction details. 2. Provide a feature to show/hide a QR code representing the transaction details. 3. Provide a print option that exports the transaction details to a PDF file and shares it using the device's sharing options. 4. Offer an edit button that navigates to a screen for editing transaction details. 5. Include a delete button that prompts the user to confirm before deleting the transaction. 		
Alternative flow	1a. If the transaction details are not available, display a message indicating that transaction details are not available.		

Table 3- 8 Use Case Description of Edit Transaction Module

Use case	Edit Transaction	ID	8
Aim	To modify details of an existing financial transaction.	Importance Level	Moderate
Actor	User and Firebase Firestore database		
Trigger	The user selects an existing transaction to edit.		
Pre-condition	The user must be logged in and select an existing transaction to modify.		

Main flow	<ol style="list-style-type: none"> 1. The user can edit the transaction amount, category, and type (income/expense). 2. The user can update notes, the date of the transaction, and add additional details (person involved, location). 3. The user can change the associated transaction image or upload a new one. 4. The system saves the updated transaction details to Firebase Firestore and updates the image in Firebase Storage if a new image is uploaded.
Alternative flow	<ol style="list-style-type: none"> 4a. If required fields are missing or there's an error during the update process, the user is alerted to correct the issues. 4b. If the user does not save changes, the original transaction details remain unchanged.

Table 3- 9 Use Case Description of Add Category Module

Use case	Add Category	ID	9
Aim	To allow the user to add a new category for transaction classification.	Importance Level	Moderate
Actor	User and Firebase Firestore database		
Trigger	The user decides to add a new category.		
Pre-condition	The user must be logged in and have access to the category creation form.		
Main flow	<ol style="list-style-type: none"> 1. User inputs the category name. 2. User selects the type of category: either expense or income. 3. The system validates the input and saves the new category to Firebase Firestore. 		
Alternative flow	3a. The system prevents the operation from completing until all validation checks pass.		

Table 3- 10 Use Case Description of Add Budget Module

Use case	Add Budget	ID	10
-----------------	------------	-----------	----

Aim	To enable the user to set a budget for a selected category within a specified time range.	Importance Level	High
Actor	User and Firebase Firestore database		
Trigger	The user chooses to create a new budget.		
Pre-condition	The user must be authenticated and have access to the budget creation form.		
Main flow	<ol style="list-style-type: none"> 1. User inputs the budget amount. 2. User selects a category for the budget. 3. User chooses a time range (week, month, or year) for the budget. 4. System records the budget in Firebase Firestore with a server timestamp. 		
Alternative flow	4a. If the user fails to input all required fields or encounters an error, the system alerts them to correct the information or try again.		

Table 3- 11 Use Case Description of View Budget Module

Use case	View Budget	ID	11
Aim	To display the user's budget allocations and spending in various categories over specified time periods.	Importance Level	Moderate
Actor	User		
Trigger	The user accessing the budget section of the app.		
Pre-condition	The user must be logged in and have existing budget data.		
Main flow	<ol style="list-style-type: none"> 1. Retrieves and displays a list of budget categories and amounts. 2. Shows the time range for each budget (e.g., week, month, year). 3. Indicates whether the user is over or under the set budget. 4. Provides the functionality to navigate to detailed budget views. 5. Includes the ability to alert the user when they exceed budget limits. 		
Alternative flow	1a. If no budgets are set or an error occurs during retrieval, the user is prompted to create a budget or try accessing the information again.		

Table 3- 12 Use Case Description of Budget Detail Module

Use case	Budget Detail	ID	12
-----------------	---------------	-----------	----

Aim	To display detailed information on a specific budget, including trend analysis and transaction details over a specified time range.	Importance Level	Moderate
Actor	User		
Trigger	The user selecting a budget from the list of budgets.		
Pre-condition	The user must be logged in and have have budgets created.		
Main flow	<ol style="list-style-type: none"> 1. Retrieves detailed data of a selected budget including category, amount, and time range. 2. Displays a line chart visualizing the budget amount versus actual spending over time. 3. Provides alerts for overspending with detailed comparisons. 4. Offers options to delete or modify budget entries. 		
Alternative flow	1a. If no detailed data is available or an error occurs, prompts to check network or data settings.		

Table 3- 13 Use Case Description of Track Achievements Module

Use case	Tracking User Achievements	ID	13
Aim	To monitor user activities and unlock achievements accordingly, providing visual representation and notification alerts for unlocked achievements.	Importance Level	Moderate
Actor	User		
Trigger	The system checks for user achievements upon user entering the "TrackAchievements" screen.		
Pre-condition	The user must be logged in, system must have the ability to access transaction and budget data for the user, and the achievements must be defined and available to be unlocked.		

Main flow	<ol style="list-style-type: none"> 1. Upon accessing the "TrackAchievements" screen, the system fetches transactions and budgets from the database. 2. The system checks if the user has met the criteria for any achievements. 3. If criteria are met, achievements are marked as unlocked in the user's profile, and a notification is sent. 4. The UI updates to display achievements, highlighting those that are unlocked. 5. User receives a notification for each newly unlocked achievement.
Alternative flow	1a. If there are no new achievements to unlock, the system simply displays the current status of all achievements.

Table 3- 14 Use Case Description of Add Loan Module

Use case	Adding and Calculating Loan Details	ID	14
Aim	To enable users to input loan details, calculate monthly payments based on the input, provide financial advice based on the calculated payment, and save loan information.	Importance Level	Moderate
Actor	User		
Trigger	The user navigates to the "LoanAdd" screen to add a new loan.		
Pre-condition	The user must be authenticated and have access to the loan addition feature within the app.		
Main flow	<ol style="list-style-type: none"> 1. User inputs the loan amount, interest rate, and loan term into the provided text fields. 2. User can switch on the option to project future payments. 3. The system calculates the monthly payment when the user presses the "Calculate" button, given the input fields are filled with valid numbers. 4. The system displays financial advice based on the calculated monthly payment. 5. User saves the loan information by pressing the "Save Loan" button, which persists the loan to the database. 6. User can clear all input fields by pressing the "Clear" button. 		

Alternative flow	1a. If the user inputs invalid numbers, an error alert is displayed prompting for valid input.
-------------------------	--

Table 3- 15 Use Case Description of View Report Module

Use case	View Report	ID	15
Aim	To display and analyze financial data visually, allowing users to understand their spending habits, budget utilization, and overall financial health.	Importance Level	High
Actor	User		
Trigger	The user navigates to the report section of the application.		
Pre-condition	The user must be authenticated and have financial data recorded.		
Main flow	<ol style="list-style-type: none"> 1. Generates and displays financial reports including net income, spending patterns, and budget status. 2. Allows users to interact with data via charts and graphs for a detailed view. 3. Provides options to print or share the report in PDF format. 		
Alternative flow	1a. If no data is available or an error occurs, it prompts to input data or check connectivity.		

Table 3- 16 Use Case Description of View Financial Health Score Module

Use case	View Financial Health Score	ID	16
Aim	To present a comprehensive view of the user's financial health, including income, expenses, and net cash flow.	Importance Level	High
Actor	User		
Trigger	The user navigates to the financial health score section in the app.		
Pre-condition	The user must be authenticated and have financial data recorded.		

Main flow	<ol style="list-style-type: none"> 1. Calculates and displays net cash flow, total income, and total expenses. 2. Provides interactive sliders to simulate different financial scenarios. 3. Offers personalized financial tips based on the user's financial data.
Alternative flow	3a. Handles lack of data by prompting the user to input necessary financial details.

Table 3- 17 Use Case Description of Input Query

Use case	Input Query	ID	17
Aim	To allow the user to input a query for financial advice or other inquiries.	Importance Level	High
Actor	User		
Trigger	The user types a query for AIChatbox interaction.		
Pre-condition	The user must be authenticated and have access to the query input field.		
Main flow	<ol style="list-style-type: none"> 1. Users enters a query in the provided field. 2. The system validates the input and forwards the query to OpenAI API. 3. The system receives the response from the API and displays it to the user. 		
Alternative flow	2a. If input validation fails, the user is prompted to correct the input (e.g., empty field).		

Table 3- 18 Use Case Description of Capture Image

Use case	Capture Image	ID	18
Aim	To capture an image of a receipt for data extraction using OCR.	Importance Level	High
Actor	User, OCR API and Firebase Firestore database		
Trigger	The user selects an option to scan a receipt and captures the image.		
Pre-condition	The user must have access to the device camera and permissions enabled.		

Main flow	<ol style="list-style-type: none"> 1. Users opens the receipt-scanning feature. 2. The system prompts the user to capture the image using the device camera. 3. The image is sent to the OCR API for text extraction. 4. The extracted data is displayed for user confirmation.
Alternative flow	3a. If image capture fails, the system prompts the user to retry.

Table 3- 19 Use Case Description of Verify Receipt Data

Use case	Verify Receipt Data	ID	19
Aim	To verify the extracted data from the receipt image, allowing users to confirm and make adjustments if necessary.	Importance Level	High
Actor	User and OCR API		
Trigger	The system completes text extraction from the scanned receipt image.		
Pre-condition	The user must capture an image of the receipt, and the OCR API should have processed it successfully.		
Main flow	<ol style="list-style-type: none"> 1. The system displays the extracted text from the receipt for user confirmation. 2. The user can verify and make any necessary adjustments to the data. 3. Once verified, the system stores the confirmed receipt data into the database. 		
Alternative flow	3a. If any discrepancies are found, the user can correct the text manually.		

Table 3- 20 Use Case Description of Make Prophet Forecast

Use case	Make Prophet Forecast	ID	20
Aim	To generate a financial forecast based on the user's historical data using Facebook Prophet, allowing users to predict future trends and financial conditions.	Importance Level	High
Actor	User, Facebook Prophet API and Firebase Firestore database		
Trigger	The user selects an option to generate a financial forecast.		

Pre-condition	The user must be logged in and have sufficient historical financial data in the system.
Main flow	<ol style="list-style-type: none"> 1. Users requests a financial forecast. 2. The system gathers the user’s financial history and sends it to the Facebook Prophet API for processing. 3. The Prophet model generates the forecast and returns the results. 4. The system displays the forecast results, showing projected trends.
Alternative flow	<ol style="list-style-type: none"> 3a. If there is insufficient historical data, the user is prompted to input more financial information. 3b. If the API request fails, the user is alerted to try again later.

3.1.3 Activity Diagram

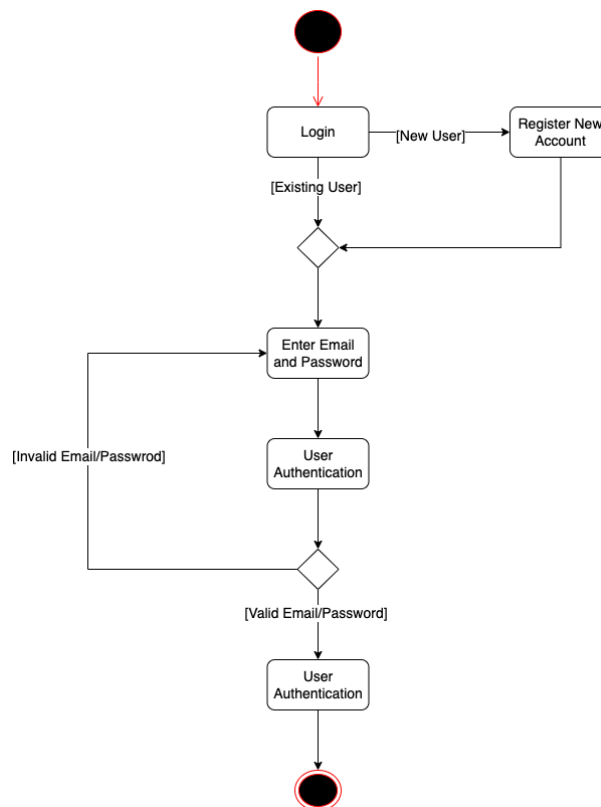


Figure 3- 4 Activity Diagram of Register and User Login

Figure 3- 4 delineates the login activity flow within the app. Upon launching the application, users are directed to the login interface. New users encounter a prompt to register and are navigated to the account registration interface to set up a new account. Existing users proceed to enter their registered email and password. The system then initiates user authentication, and

Bachelor of Information Systems (Honours) Business Information Systems
Faculty of Information and Communication Technology (Kampar Campus), UTAR

a successful match grants access to their account. Conversely, if the credentials are incorrect, an error notification is displayed, prompting the user to re-enter their login details.

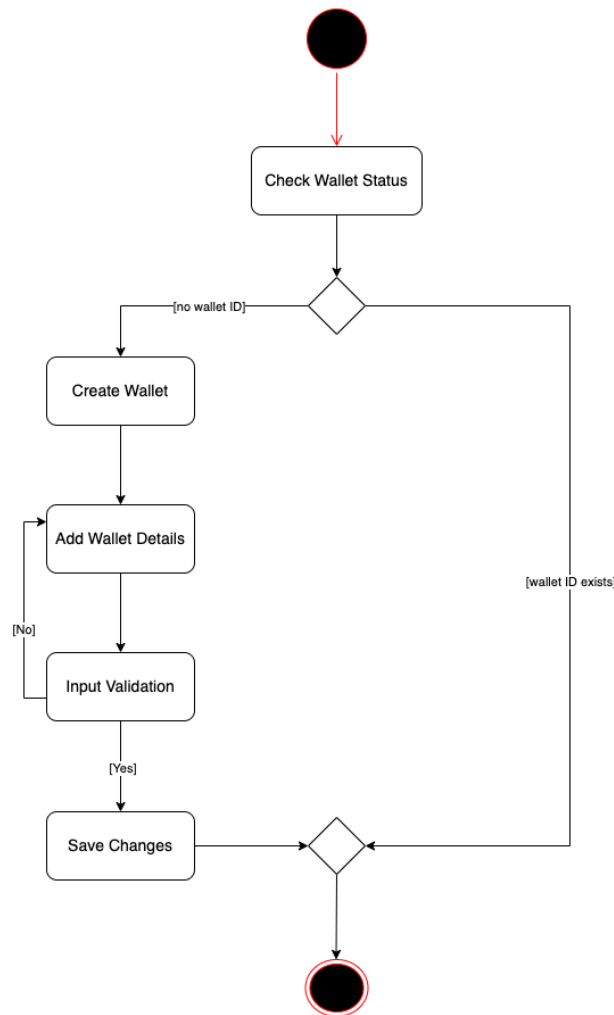


Figure 3- 5 Activity Diagram of Create Wallet

Figure 3- 5 presents an activity diagram for the Create Wallet process. The process begins with a decision point where the system checks the wallet status of the user. If a wallet exists, the flow ends. If no wallet exists, the flow continues to the "Create Wallet" activity, where the user can add wallet details. Subsequent to this, an input validation is performed; if validation fails, it loops back to the "Add Wallet Details" step for correction. Once the inputs are validated, the changes are saved, concluding the process. This depiction clearly outlines the conditional flow for new wallet creation, ensuring that only users without an existing wallet can create one.

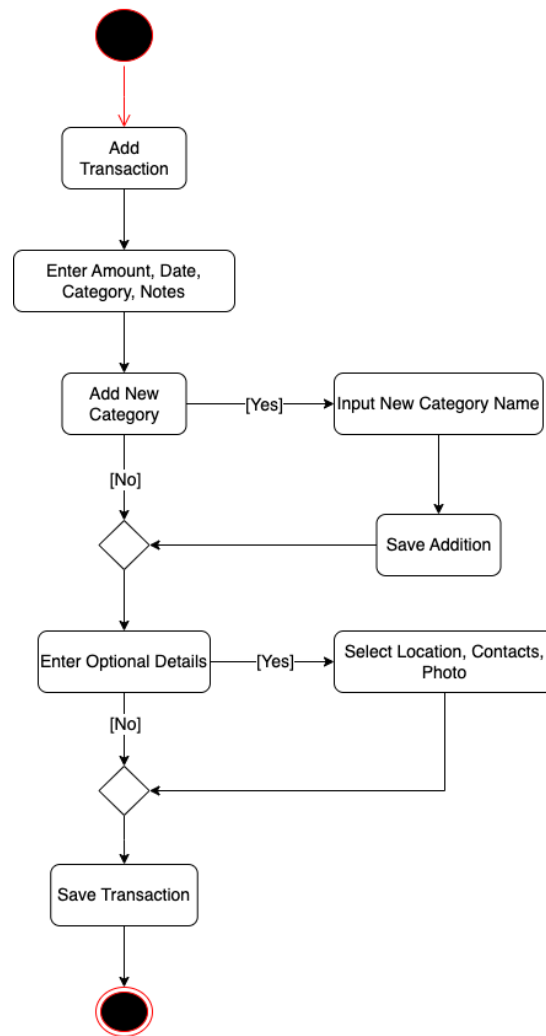


Figure 3- 6 Activity Diagram of Add Transaction

Figure 3- 6 presents the activity flow for adding a transaction. The user begins by entering the core transaction details including the amount, date, category, and any relevant notes. If necessary, the user can also create a new transaction category. After entering the main details, the user has the choice to include extra information like location, contacts, and photos. The process is completed once the user saves the transaction.

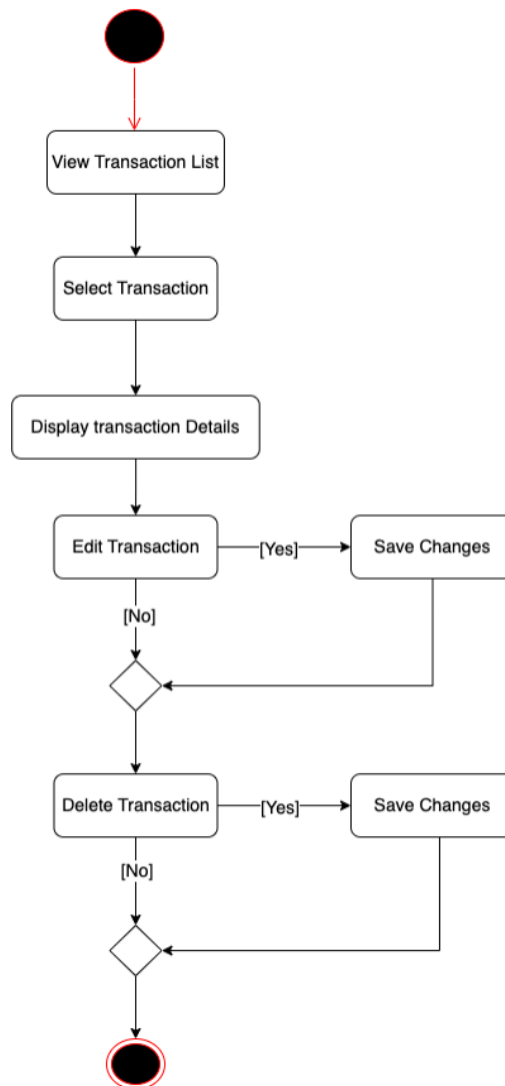


Figure 3- 7 Activity Diagram of Manage Transaction

Figure 3- 7 illustrates the flow for managing a transaction. The user starts by viewing a list of transactions and selecting one to review. Once a transaction is selected, its details are displayed. The user then has the option to edit the transaction. If edits are made, the user saves these changes. Additionally, there is an option to delete the transaction; if this option is chosen and confirmed, the changes are saved, completing the management process.

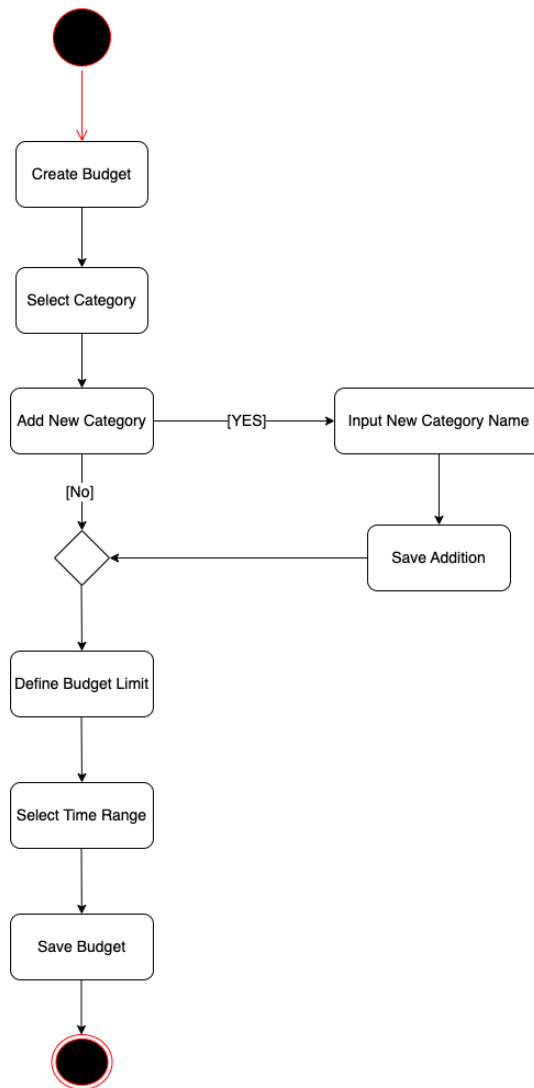


Figure 3- 8 Activity Diagram of Create Budget

Figure 3- 8 illustrates the steps a user takes to create a budget within the app. Initially, the user begins by selecting a budget category. If a new category is needed, they can add and save it. Next, the user sets a budget limit and chooses a time range for the budget. The process concludes with the user saving the budget details.

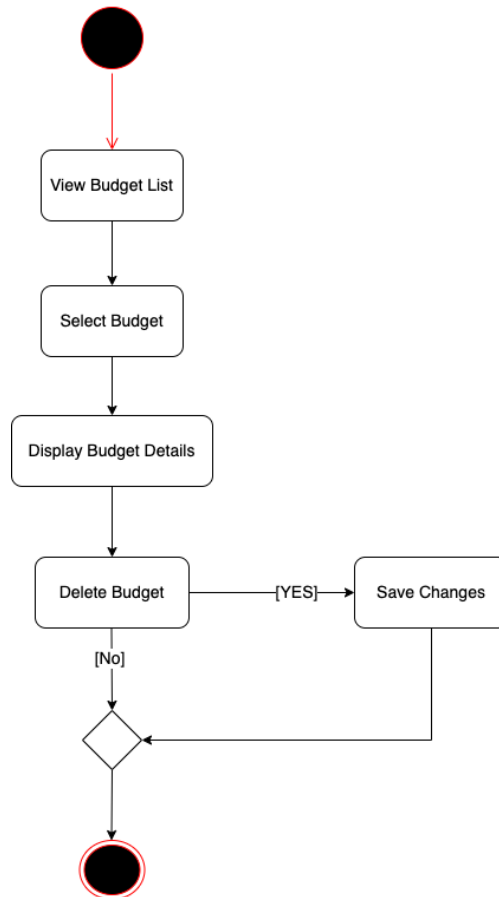


Figure 3- 9 Activity Diagram of Manage Budget

Figure 3- 9 outlines the process for managing a budget within the application. It begins with the user viewing a list of their budgets. Upon selecting a budget, they can see its details. The user then has the option to delete the budget. If changes are made, they can be saved. The process concludes with either the saving of edits or the confirmation of budget deletion.

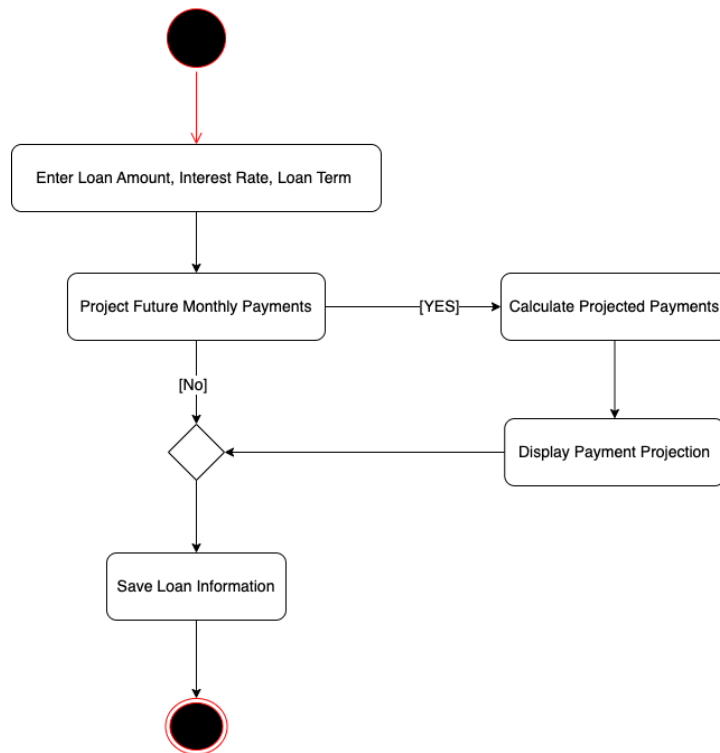


Figure 3- 10 Activity Diagram of Add Loan

Figure 3- 10 details the procedure for adding a loan within the application, beginning with inputting the loan amount, interest rate, and term. Users can opt to project future payments, which, if selected, triggers the calculation and display of payment projections. If not, the process moves directly to saving the loan information, concluding the entry process. This flow ensures users can manage their loans effectively, with an option for forward-looking financial planning.

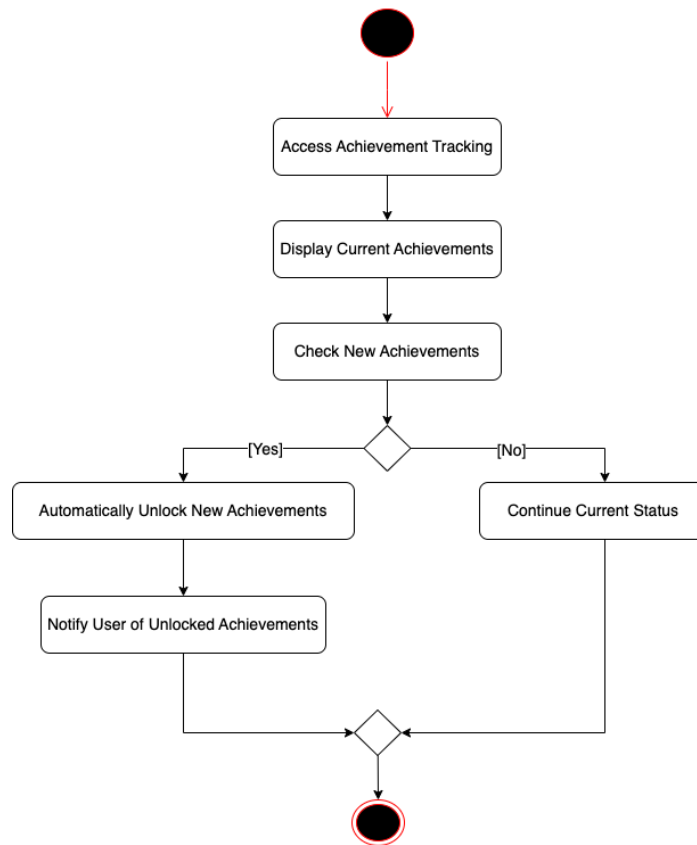


Figure 3- 11 Activity Diagram of Track Achievements

Figure 3- 11 represents the automated tracking of achievements in the application. Initially, the user accesses the achievement tracking feature, where current achievements are displayed. The system then checks for any new achievements. If found, these achievements are automatically unlocked, and the user is informed through a notification. In the absence of new achievements, the user's status remains unchanged.

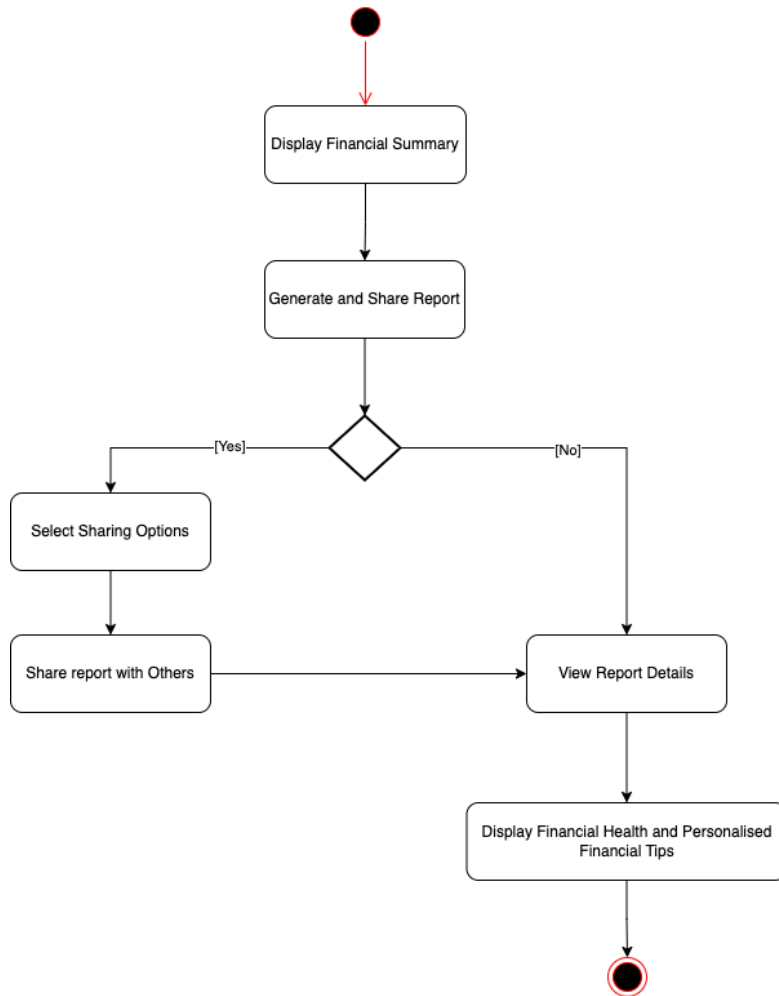


Figure 3- 12 Activity Diagram of View Reports

Figure 3- 12 presents an activity diagram of the View Reports process. The workflow begins with displaying a financial summary. Users then have the option to generate and share the report. If they choose to share, they are prompted to select sharing options and proceed to share the report with others. If not, or once sharing is completed, they have the option to view report details, which leads to the display of financial health and personalized financial tips. The process flow ends following the user's review of these details.

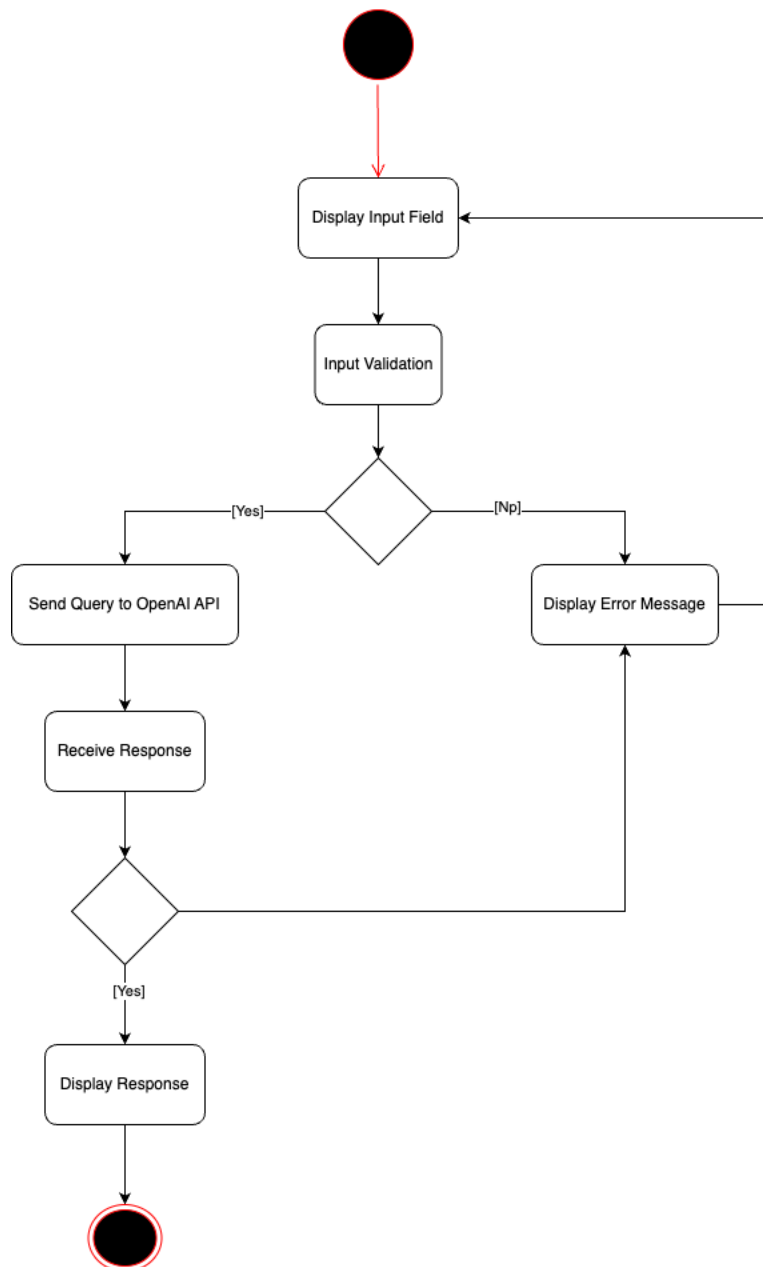


Figure 3- 13 Activity Diagram of Input Query

Figure 3- 13 illustrates the activity flow of the "Input Query" process. The user begins by entering a query in the input field. The system performs input validation, ensuring the query meets necessary conditions. If the validation fails, the system displays an error message and returns to the input field. If successful, the query is sent to the OpenAI API for processing. The system receives the response, and if the response is valid, it is displayed to the user. This completes the process.

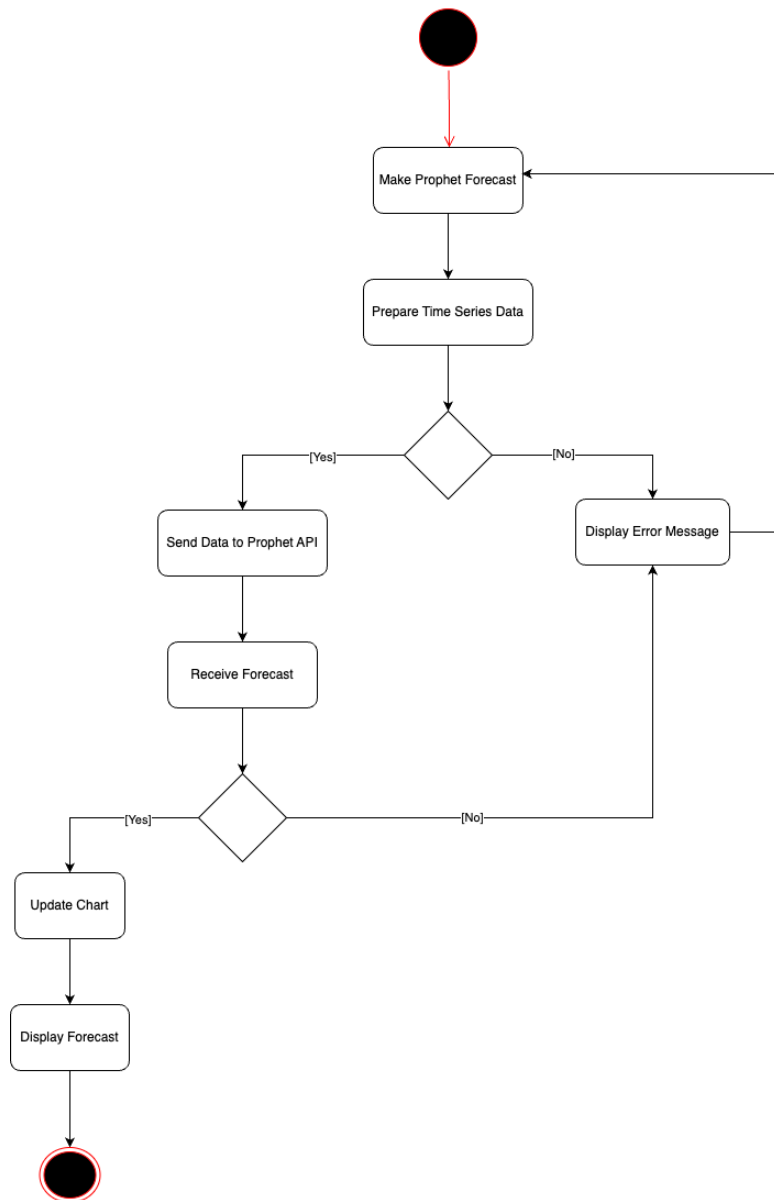


Figure 3- 14 Activity Diagram of Make Prophet Forecast

Figure 3-14 illustrates the process flow for making a financial forecast using Facebook Prophet. The user begins by initiating the Make Prophet Forecast activity. The system then prepares the time series data. If data preparation is successful, the data is sent to the Prophet API. If it fails, an error message is displayed. After receiving the forecast from the API, the system checks if the forecast is valid. If valid, the chart is updated, and the forecast is displayed; otherwise, an error is shown.

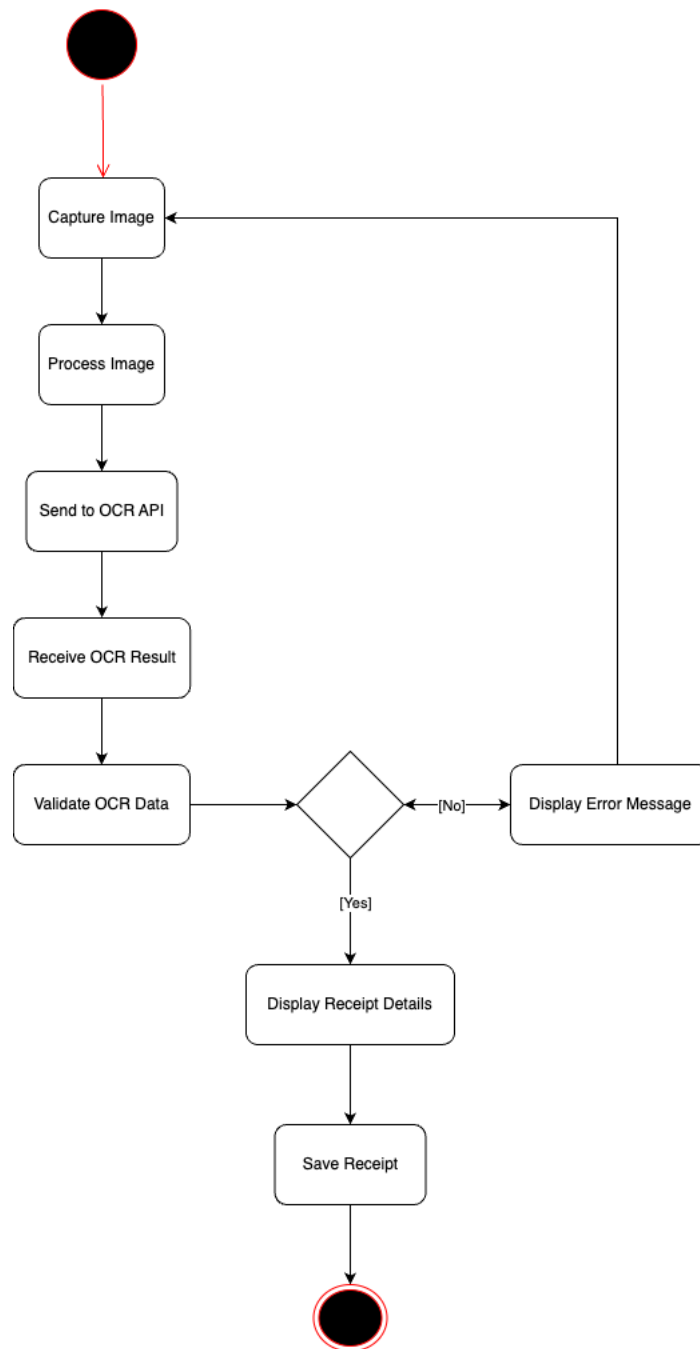


Figure 3- 15 Activity Diagram of Capture Image (OCR)

Figure 3-15 Activity Diagram of Capture Image (OCR) describes the workflow for scanning a receipt using OCR. The process starts with the user capturing an image of the receipt. After the image is processed, it is sent to the OCR API for text recognition. The system receives the OCR result and validates the data. If validation fails, an error message is displayed; otherwise, the receipt details are shown for user confirmation. Finally, the receipt is saved, completing the process.

Chapter 4 System Design

4.1 System Block Diagram



Figure 4- 1 Block Diagram of the proposed system

Figure 4-1 illustrates the block diagram of the financial mobile application system. The architecture encompasses several interconnected modules, each performing specific functions. The Auth Module manages user authentication processes such as login, registration, and password recovery, leveraging Firebase Authentication. The Transaction Module allows users to add transactions through receipt scanning or manual input, utilizing the Cloud Vision API

for OCR, and stores this data in Firebase Cloud Storage. The Budget Module enables users to create and track budgets, while the Financial Forecasting Module uses the Facebook Prophet API for predictive analytics and future financial projections.

The Achievements Module tracks user progress and unlocks achievements based on specific actions, providing a motivational tool for financial milestones. The Dashboard Module offers a comprehensive overview of the user's financial health, displaying spending trends and overall financial status. The Reporting Module generates detailed reports, allowing users to export them in PDF format. The Category Module organizes user transactions by categories, enhancing the clarity and simplicity of financial management.

The Personal Finance Health Score System assesses the user's financial status based on factors such as income, expenses, debt, and savings. It generates a personalized score and offers insights for improving financial health, serving as the first core feature of the system.

The AI Chatbox Module incorporates OpenAI's GPT API to provide personalized financial advice to users, responding to queries related to financial health, savings strategies, and spending habits through natural language interaction. The chatbox dynamically generates real-time responses based on user input, enhancing user engagement.

Lastly, the Financial Analysis Module allows users to analyze historical transaction data and predict future expenses. Users can view graphical representations of spending trends, and receive actionable insights. The entire system relies on Firebase for real-time data synchronization, ensuring a responsive and efficient user experience. The system is designed to offer a comprehensive tool for managing both short-term financial activities and long-term planning.

4.2 System Flow Diagram

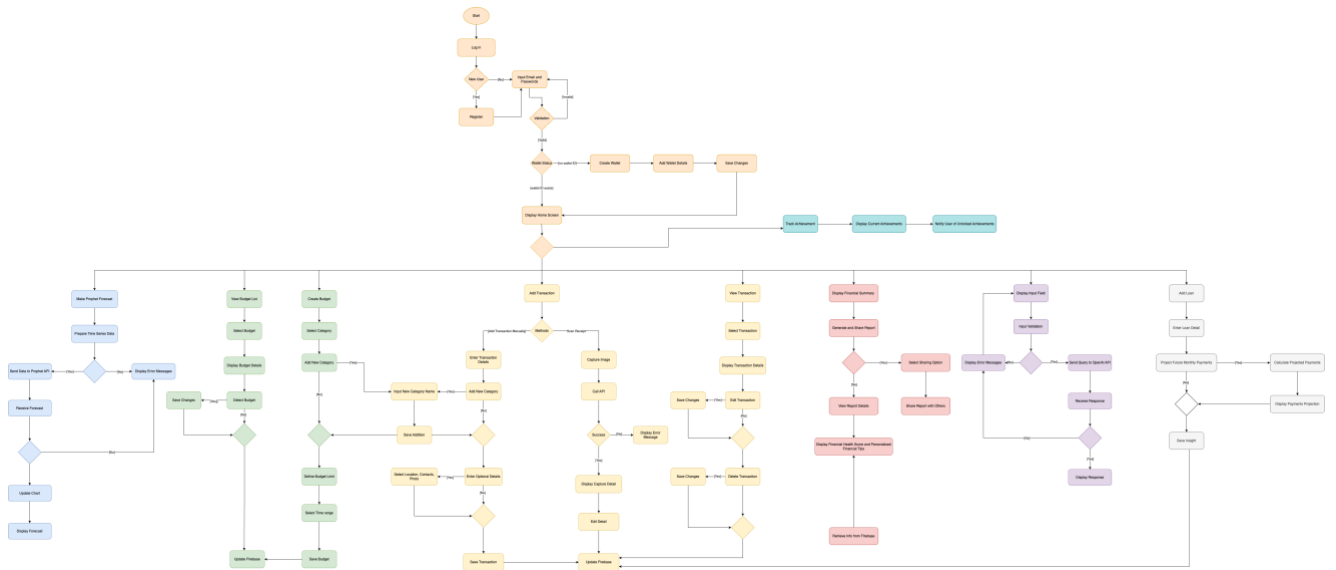


Figure 4- 2 System Flow Diagram

4.3 System Flow Description (Code)

4.3.1 Login.tsx

```

async function login() {
  setLoading(true);
  try {
    const userCredential = await signInWithEmailAndPassword(
      auth,
      email,
      password
    );
    const user = userCredential.user;

    // Navigate to About with user ID
    navigation.navigate("About", { userId: user.uid });
  } catch (error) {
    alert(error.message);
  } finally {
    setLoading(false);
  }
}

```

Figure 4- 3 Code Snippet for User Login Process

The Login.tsx file handles the user login process by utilizing Firebase Authentication. It includes input fields for email and password, which are validated before using the

signInWithEmailAndPassword method. Upon successful login, the user is navigated to the "About" page with their user ID. Error handling ensures proper feedback for login failures, and a loading indicator is displayed during the authentication process. Additionally, users can switch between light and dark themes, or navigate to registration and password recovery screens.

4.3.2 Register.tsx

```

async function register() {
  if (!display_name) {
    if (Platform.OS === "ios" || Platform.OS === "android")
      Alert.alert("Display name is required");
    else alert("Display name is required");
  } else if (!email) {
    if (Platform.OS === "ios" || Platform.OS === "android")
      Alert.alert("Email is required");
    else alert("Email is required");
  } else if (!password) {
    if (Platform.OS === "ios" || Platform.OS === "android")
      Alert.alert("Password is required");
    else alert("Password is required");
  } else if (!confirm_password) {
    if (Platform.OS === "ios" || Platform.OS === "android")
      Alert.alert("Confirm password is required");
    else alert("Confirm password is required");
  } else if (password !== confirm_password) {
    if (Platform.OS === "ios" || Platform.OS === "android")
      Alert.alert("Confirm password not match to password");
    else alert("Confirm password not match to password");
  } else if (!genderValue) {
    if (Platform.OS === "ios" || Platform.OS === "android")
      Alert.alert("Gender is required");
    else alert("Gender is required");
  } else if (!birth_date) {
    if (Platform.OS === "ios" || Platform.OS === "android")
      Alert.alert("Birth Date is required");
    else alert("Birth Date is required");
  } else {
    setLoading(true);
    await createUserWithEmailAndPassword(auth, email, password)
      .then(() => {
        if (auth.currentUser) {
          const currentUser: User = auth.currentUser;
          updateProfile(currentUser, {
            displayName: display_name,
            photoURL: "-",
          })
            .then(() => {
              setDoc(doc(db, "users", currentUser.uid), {
                email: currentUser.email,
                displayName: currentUser.displayName,
                photoURL: currentUser.photoURL,
                gender: genderValue,
                birthDate: birth_date,
                tokenId: "-",
              });
            })
            .catch((error) => {
              var errorCode = error.code;
              var errorMessage = error.message;
              setLoading(false);
              if (Platform.OS === "ios" || Platform.OS === "android")
                Alert.alert(errorMessage);
              else alert(errorMessage);
            });
        }
      })
      .catch(function (error: any) {
        var errorCode = error.code;
        var errorMessage = error.message;
        setLoading(false);
        if (Platform.OS === "ios" || Platform.OS === "android")
          Alert.alert(errorMessage);
        else alert(errorMessage);
      });
  }
}

```

Figure 4- 4 Code Snippet for User Registration

Figure 4-3 illustrates the code responsible for user registration in the application. The register function checks multiple required fields like display name, email, password, and gender before proceeding with Firebase Authentication. If any of the fields are missing or incorrect, it raises an alert. Once the user successfully registers, the code updates the user's profile and stores additional details like gender and birth date in Firestore. Error handling is integrated to ensure the process is smooth across platforms.

4.3.3 ForgetPassword.tsx


```
async function forget() {
  setLoading(true);
  await sendPasswordResetEmail(auth, email)
    .then(function () {
      setLoading(false);
      navigation.navigate("Login");
      alert("Your password reset has been sent to your email");
    })
    .catch(function (error) {
      setLoading(false);
      alert(error);
    });
}
```

Figure 4- 5 Code Snippet for Password Reset

Figure 4-4 demonstrates the password reset process in the ForgetPassword.tsx file. It utilizes Firebase Authentication to send a password reset email based on the provided email address. Once the email is sent, the user is notified, and upon success, the user is redirected to the login screen. In case of any errors, appropriate alerts are displayed to the user to ensure they are informed about the process's success or failure.

4.3.4 About.tsx

```
useEffect(() => {
  if (userId) {
    const fetchWalletData = async () => {
      if (userId) {
        console.log("Fetching wallets for user ID:", userId);
        const db = getFirestore();
        const walletsCol = collection(db, "wallets");
        const q = query(walletsCol, where("userId", "==", userId));
        try {
          const querySnapshot = await getDocs(q);
          console.log(
            `Found ${querySnapshot.size} wallets for user ID ${userId}`
          );
          if (!querySnapshot.empty) {
            console.log("Navigating to MyMenu");
            navigation.navigate("MyMenu");
          } else {
            console.log("Navigating to CreateWallet");
            navigation.navigate("CreateWallet");
          }
        } catch (error) {
          console.error("Error fetching wallet data:", error);
        }
      } else {
        console.log("No user ID provided");
      }
    };

    fetchWalletData();
  }
}, [userId, navigation]);
```

Figure 4- 6 Code Snippet for User Wallet Navigation

Figure 4-5 presents the code responsible for navigating the user to the appropriate screen based on the existence of wallets in Firebase. After fetching the wallet data for the authenticated user, the code either redirects to the MyMenu screen if wallets exist, or to the CreateWallet screen if no wallets are found. It uses Firebase Firestore queries and handles navigation within the app.

4.3.5 CreateWallet.tsx

```
const createWallet = async () => {
  if (!walletName.trim()) {
    setModalMessage("Wallet name cannot be empty.");
    setModalMessageType("error");
    setIsModalVisible(true);
    return;
  }

  if (!currency) {
    setModalMessage("Please select a currency.");
    setModalMessageType("error");
    setIsModalVisible(true);
    return;
  }

  if (!auth.currentUser) {
    setModalMessage("No authenticated user found.");
    setModalMessageType("error");
    setIsModalVisible(true);
    return;
  }

  setIsLoading(true);
  const db = getFirestore();

  try {
    // Create a new wallet document in 'wallets' collection
    const newWalletRef = doc(collection(db, "wallets"));
    await setDoc(newWalletRef, {
      walletName: walletName,
      currency: currency,
      userId: auth.currentUser.uid,
      financialGoal: selectedGoal,
      targetSavings: targetSavings,
    });

    // Link the walletId to the user's document
    const userRef = doc(db, "users", auth.currentUser.uid);
    await setDoc(
      userRef,
      {
        walletId: newWalletRef.id,
      },
      { merge: true }
    );

    setModalMessage("Your wallet has been created.");
    setModalMessageType("success");
    setIsModalVisible(true);
    setTimeout(() => {
      setIsModalVisible(false);
      navigation.navigate("MyMenu");
    }, 2000);
  } catch (error) {
    console.error("Error creating wallet:", error);
    setModalMessage("There was an error creating your wallet.");
    setModalMessageType("error");
    setIsModalVisible(true);
  } finally {
    setIsLoading(false);
  }
};
```

Figure 4- 7 Code Snippet for Creating a Wallet

Figure 4-6 illustrates the code that handles the creation of a wallet in Firebase. It checks for necessary inputs such as the wallet name, currency, and user authentication status. Upon validation, a wallet is created in the "wallets" collection, and the wallet ID is linked to the user's profile. If the wallet is successfully created, the user is navigated to the "MyMenu" screen, and a modal is displayed to confirm the action.

4.3.6 MyMenu.tsx

```
const MainTabs = ({ navigation }) => {
  const { isDarkmode } = useTheme();

  const CustomTabBarButton = ({ children, onPress }) => (
    <TouchableOpacity
      style={{
        top: -30,
        justifyContent: "center",
        alignItems: "center",
        ...styles.shadow,
      }}
      onPress={onPress}
    >
      <View
        style={{
          width: 70,
          height: 70,
          borderRadius: 35,
          backgroundColor: isDarkmode ? themeColor.dark : themeColor.white100,
          borderWidth: 2,
          borderColor: isDarkmode ? themeColor.white100 : themeColor.black,
          justifyContent: "center",
          alignItems: "center",
        }}
      >
        <Ionicons
          name="add"
          size={40}
          color={isDarkmode ? themeColor.white100 : themeColor.black}
        />
      </View>
    </TouchableOpacity>
  );

  return (
    <Tabs.Navigator
      screenOptions={{
        headerShown: false,
        tabBarShowLabel: false,
        tabBarStyle: {
          height: 60,
          backgroundColor: isDarkmode ? themeColor.dark : themeColor.white100,
          ...styles.shadow,
        },
        tabBarActiveTintColor: isDarkmode
          ? themeColor.white100
          : themeColor.black,
        tabBarInactiveTintColor: isDarkmode
          ? themeColor.gray100
          : themeColor.black,
      }}
    >
      { /* Tab Screens */ }
      <Tabs.Screen
        name="Home"
        component={Home}
        options={{
          tabBarIcon: ({ focused }) => (
            <TabBarIcon focused={focused} icon={"md-home"} />
          ),
        }}
      />
      <Tabs.Screen
        name="TransactionList"
        component={TransactionList}
        options={{
          tabBarIcon: ({ focused }) => (
            <TabBarIcon focused={focused} icon={"list"} />
          ),
        }}
      />
      <Tabs.Screen
        name="AddTransaction"
        component={View}
        options={{
          tabBarIcon: ({ focused }) => (
            <Ionicons
              name="add"
              size={40}
              color="#FFF"
              style={{ marginTop: -15 }}
            />
          ),
          tabBarButton: (props) => (
            <CustomTabBarButton
              {...props}
              onPress={() => navigation.navigate("TransactionAdd")}
            />
          ),
        }}
      />
      <Tabs.Screen
        name="Budget"
        component={Budget}
        options={{
          tabBarIcon: ({ focused }) => (
            <TabBarIcon focused={focused} icon={"wallet"} />
          ),
        }}
      />
      <Tabs.Screen
        name="Account"
        component={Account}
        options={{
          tabBarIcon: ({ focused }) => (
            <TabBarIcon focused={focused} icon={"person"} />
          ),
        }}
      />
    </Tabs.Navigator>
  );
};

export default MainTabs;
```

Figure 4- 8Code Snippet for MainTabs

Figure 4-7 shows the code for the main navigation menu of the app, implementing a bottom tab navigator using react-navigation and custom styling. It includes five screens: Home, Bachelor of Information Systems (Honours) Business Information Systems Faculty of Information and Communication Technology (Kampar Campus), UTAR

TransactionList, TransactionAdd (custom center button), Budget, and Account. Each tab has an icon, with a custom floating button in the center for adding transactions. The isDarkmode theme adjusts the color of the tab bar and buttons dynamically based on user preference.

4.3.7 TransactionAdd.tsx

```

const handlePress = async () => {
  if (!amount || !selectedCategory.CategoryName) {
    Alert.alert("Error", "Please fill in all required fields.");
    return;
  }

  setLoading(true);

  const auth = getAuth();
  const db = getFirestore();

  try {
    let downloadURL = null;

    if (image) {
      const storage = getStorage();
      const response = await fetch(image);
      const blob = await response.blob();

      let u =
        Date.now().toString(16) + Math.random().toString(16) + "0".repeat(16);
      let guid = [
        u.substr(0, 8),
        u.substr(8, 4),
        "4000-8" + u.substr(13, 3),
        u.substr(16, 12),
      ].join("-");
      const spaceRef = ref(storage, "TransactionImages/" + guid);

      try {
        const snapshot = await uploadBytes(spaceRef, blob);
        downloadURL = await getDownloadURL(snapshot.ref);
      } catch (error) {
        console.error("Error uploading image: ", error);
        Alert.alert("Upload failed", "Failed to upload image");
        setLoading(false);
        return; // Prevent further execution
      }
    }

    const startDate = new Date().getTime();

    if (auth.currentUser) {
      const currentUser = auth.currentUser;

      const transactionData = {
        amount,
        category: selectedCategory.CategoryName,
        type: transactionType,
        notes,
        date,
        image: downloadURL,
        withPerson,
        location,
        startDate: startDate,
        updatedAt: startDate,
        CreatedUser: {
          uid: currentUser.uid,
          displayName: currentUser.displayName || "",
        },
      };

      await addDoc(collection(db, "Transactions"), transactionData);
      emptyState();
      Alert.alert("Added successfully.", "", [
        { text: "OK", onPress: () => navigation.navigate("TransactionAdd") },
      ]);
    } else {
      throw new Error("Authentication failed. User not found.");
    }
  } catch (err) {
    console.error("Error adding transaction: ", err);
    Alert.alert("There is something wrong!", err.message);
  }

  setLoading(false);
};

```

Figure 4- 9 Code Snippet for Adding Transactions

Figure 4- 8 illustrates the code for adding a new transaction to the app. It allows users to input transaction details such as the amount, category, and additional notes. Users can also attach an image, select a contact, and set a location for the transaction. The data, including images, are uploaded to Firebase Firestore and Firebase Storage. It also features a date picker and handles category selection dynamically, enhancing the user's transaction management experience.

4.3.8 MyLocation.tsx

```
useEffect(() => {
  (async () => {
    let { status } = await Location.requestForegroundPermissionsAsync();
    if (status !== "granted") {
      setErrorMsg("Permission to access location was denied");
      return;
    }

    let location = await Location.getCurrentPositionAsync({});
    setLocation({
      latitude: location.coords.latitude,
      longitude: location.coords.longitude,
      latitudeDelta: 0.0922,
      longitudeDelta: 0.0421,
    });
  })();
}, []);

const handleConfirmLocation = async () => {
  if (!location) {
    Alert.alert(
      "Location not selected",
      "Please select a location on the map."
    );
    return;
  }

  try {
    // Get readable address from coordinates
    const places = await Location.reverseGeocodeAsync({
      latitude: location.latitude,
      longitude: location.longitude,
    });

    if (places.length > 0) {
      const place = places[0];
      const address = `${place.name}, ${place.city}, ${place.region}, ${place.country}`;
      route.params?.onLocationSelect(address);
    } else {
      route.params?.onLocationSelect("Address not found");
    }
  } catch (error) {
    console.error(error);
    Alert.alert("Error", "Could not fetch the address.");
  }

  navigation.goBack();
};

const handleSelectLocation = (e) => {
  const { latitude, longitude } = e.nativeEvent.coordinate;
  setLocation({ latitude, longitude });
};
```

Figure 4- 10 Code Snippet for Location Selection

Figure 4- 9 demonstrates the functionality of selecting and confirming a location using the app's map interface. It requests location permission, retrieves the current location, and displays it on a map using the MapView component. Users can choose a new location by tapping on the map.

Once selected, the app converts the coordinates into a readable address using reverse geocoding and sends it back to the previous screen for further use.

4.3.9 ContactList.tsx

```
useEffect(() => {
  (async () => {
    const { status } = await Contacts.requestPermissionsAsync();
    if (status === "granted") {
      const { data } = await Contacts.getContactsAsync({
        fields: [Contacts.Fields.PhoneNumbers],
      });

      if (data) {
        const processedData = preprocessContacts(data);
        setSections(processedData); // Set the grouped data for SectionList
      }
    }
  })();
}, []);

const scrollToIndex = (letter: string) => {
  // Ensure the sections array is not empty
  if (sections.length > 0) {
    const section = sections.findIndex((s) => s.title === letter);
    if (section !== -1) {
      sectionListRef.current?.scrollToLocation({
        animated: true,
        sectionIndex: section,
        itemIndex: 0,
        viewPosition: 0, // Position at the start of the viewport
      });
    }
  }
};

const renderAlphabetIndex = () => {
  return (
    <View style={styles.alphabetIndexContainer}>
      {alphabetIndex.map((letter) => (
        <TouchableOpacity
          key={letter}
          onPress={() => scrollToIndex(letter)}
          style={styles.alphabetIndexItem}
        >
          <Text style={styles.alphabetIndexText}>{letter}</Text>
        </TouchableOpacity>
      ))}
    </View>
  );
};
```

Figure 4- 11 Code Snippet for Contact List

Figure 4- 10 illustrates the implementation of the contact list functionality. The `useEffect` hook requests permission to access the user's contacts and retrieves their phone numbers using the `expo-contacts` library. The contacts are then processed and grouped alphabetically for display in a `SectionList`. Additionally, an alphabet index is provided for quick navigation through the contact list, allowing users to jump to specific sections. The `scrollToIndex` function ensures smooth navigation between sections by scrolling to the selected letter in the contact list.

4.3.10 CategoryList.tsx

```
useEffect(() => {
  const unsubscribe = onSnapshot(collection(db, "Category"), (snapshot) => {
    const expenses = [];
    const incomes = [];

    snapshot.docs.forEach((doc) => {
      const category = {
        name: doc.data().CategoryName,
        icon: "list",
        id: doc.id,
        type: doc.data().Type,
      };

      if (category.type === "expense") {
        expenses.push(category);
      } else if (category.type === "income") {
        incomes.push(category);
      }
    });

    // Only update the state relevant to the selected index
    if (selectedIndex === 0) {
      setCategories([...predefinedExpenseCategories, ...expenses]);
    } else {
      setCategories([...predefinedIncomeCategories, ...incomes]);
    }
  });

  return unsubscribe;
}, [selectedIndex]);

return (
  <Layout>
    <TopNav>
      middleContent="Select Category"
      leftContent=(
        <Ionicons name="chevron-back" size={20} color={themeColor.gray200} />
      )
      leftAction={() => navigation.goBack()}
      rightContent={<RightContent />}
    </TopNav>
    <SegmentedControlTab>
      values={["Expenses", "Income"]}
      selectedIndex={selectedIndex}
      onTabPress={(index) => {
        setSelectedIndex(index);
      }}
      tabsContainerStyle={styles.segmentedControlContainer}
      tabStyle={styles.segmentedControlTab}
      tabTextStyle={styles.segmentedControlTabText}
      activeTabStyle={styles.segmentedControlActiveTab}
      activeTabTextStyle={styles.segmentedControlActiveTabText}
    </SegmentedControlTab>
    <FlatList>
      data={selectedIndex === 0 ? categories : predefinedIncomeCategories}
      renderItem={renderCategory}
      keyExtractor={(item) => item.id || item.name}
      contentContainerStyle={styles.list}
      ListEmptyComponent={
        loading && selectedIndex === 0 ? <ActivityIndicator /> : null
      }
    </FlatList>
  </Layout>
);
```

Figure 4- 12 Code Snippet for Category List

Figure 4-11 presents the code for managing categories within the CategoryList.tsx component. It retrieves real-time category data from Firestore, distinguishing between income and expense categories. The SegmentedControlTab allows users to toggle between these two categories. When a category is selected, it navigates back to the previous screen with the chosen category.

4.3.11 CategoryAdd.tsx

```
const switchLabelStyle = {
  fontSize: 16,
  color: isDarkmode ? themeColor.white100 : themeColor.dark,
  marginHorizontal: 10,
};

async function addCategory() {
  if (categoryName.trim() === "") {
    Alert.alert("Validation", "Please enter a category name.");
    return;
  }

  setLoading(true);
  const db = getFirestore();
  const auth = getAuth();

  if (auth.currentUser) {
    try {
      await addDoc(collection(db, "Category"), {
        CategoryName: categoryName.trim(),
        CreatedUser: {
          uid: auth.currentUser.uid,
          displayName: auth.currentUser.displayName || "",
        },
        Type: isExpense ? "expense" : "income",
        startDate: new Date().getTime(),
        updatedDate: new Date().getTime(),
      });

      setLoading(false);
      setCategoryName("");
      setIsExpense(true); // Reset the switch to 'expense'
      navigation.goBack();
    } catch (error) {
      setLoading(false);
      Alert.alert("Error", "There was an issue adding the category.");
    }
  } else {
    setLoading(false);
    Alert.alert("Authentication Error", "User not found.");
  }
}
```

Figure 4- 13 Code snippet for adding a new category

Figure 4-12 showcases the implementation of the *CategoryAdd.tsx* functionality, allowing users to add new categories for transactions. Users can toggle between "Income" and "Expense" types using a switch and input a category name. The system verifies the input, then stores the category in Firestore, linked to the authenticated user's profile. If the process is successful, the input fields reset, and the user is navigated back to the previous screen.

4.3.12 TransactionList.tsx

```
const parseDate = (dateValue) => {
  if (dateValue instanceof Date) {
    return isNaN(dateValue.getTime()) ? new Date() : dateValue;
  }
  if (typeof dateValue === 'object' && dateValue.seconds && dateValue.nanoseconds) {
    return new Date(dateValue.seconds * 1000 + dateValue.nanoseconds / 1000000);
  }
  if (typeof dateValue === 'number') {
    return new Date(dateValue);
  }
  if (typeof dateValue === 'string') {
    const parsedDate = new Date(dateValue);
    if (!isNaN(parsedDate.getTime())) {
      return parsedDate;
    }
    const parts = dateValue.split('/');
    if (parts.length === 3) {
      const [day, month, year] = parts;
      const parsedDate = new Date(parseInt(year), parseInt(month) - 1, parseInt(day));
      return isNaN(parsedDate.getTime()) ? new Date() : parsedDate;
    }
    const isoDate = parseISO(dateValue);
    return isNaN(isoDate.getTime()) ? new Date() : isoDate;
  }
  console.warn('Invalid date format:', dateValue);
  return new Date();
};
```

Figure 4- 14 Code snippet for parsing transaction dates

Figure 4 -13 defines the parseDate function used to handle multiple date formats when working with transaction data. It processes date inputs, whether they are JavaScript Date objects, Firestore Timestamp objects, Unix timestamps, or date strings in various formats, ensuring compatibility with the Firestore data retrieved from transactions.

```

useEffect(() => {
  const fetchWalletData = async () => {
    if (auth.currentUser) {
      const userRef = doc(db, "users", auth.currentUser.uid);
      try {
        const userDoc = await getDoc(userRef);
        if (userDoc.exists()) {
          const walletId = userDoc.data().walletId;
          if (walletId) {
            const walletRef = doc(db, "wallets", walletId);
            const walletDoc = await getDoc(walletRef);
            if (walletDoc.exists()) {
              setCurrency(walletDoc.data().currency || "");
            }
          }
        }
      } catch (error) {
        console.error("Error fetching wallet data:", error);
      }
    }
  };

  const fetchTransactions = async () => {
    if (auth.currentUser) {
      const q = query(
        collection(db, "Transactions"),
        where("CreatedUser.uid", "==", auth.currentUser.uid)
      );

      try {
        const querySnapshot = await getDocs(q);
        const transactions = [];

        querySnapshot.forEach((doc) => {
          const data = doc.data();
          let transactionDate, updatedDate;
          try {
            transactionDate = parseDate(data.date);
            updatedDate = parseDate(data.updatedDate);
          } catch (error) {
            console.error("Error parsing date:", error);
            transactionDate = new Date();
            updatedDate = new Date();
          }
          transactions.push({
            ...data,
            key: doc.id,
            date: transactionDate,
            updatedDate: updatedDate,
          });
        });
        transactions.sort((a, b) => b.date.getTime() - a.date.getTime());

        const groupedTransactions = {};
        transactions.forEach((transaction) => {
          const dateString = format(transaction.date, "yyyy-MM-dd");
          if (!groupedTransactions[dateString]) {
            groupedTransactions[dateString] = {
              title: isToday(transaction.date)
                ? "Today"
                : isYesterday(transaction.date)
                ? "Yesterday"
                : format(transaction.date, "PPP"),
              data: [],
            };
          }
          groupedTransactions[dateString].data.push(transaction);
        });
        const sectionsArray = Object.values(groupedTransactions);
        setSections(sectionsArray);
      } catch (error) {
        console.error("Error fetching transactions:", error);
      } finally {
        setLoading(false);
      }
    }
  };

  fetchWalletData();
  fetchTransactions();
}, []);

```

Figure 4- 15 Code snippet for fetching transactions

Figure 4- 14 illustrates the retrieval of a user's wallet and transactions. Since a user only has one wallet, transactions are linked to that specific wallet, which is stored in Firestore. The transactions are fetched, processed, and grouped by date to display them in a structured format. Each transaction is parsed using the parseDate function for consistent date handling.

4.3.13 TransactionDetail.tsx

```
useEffect(() => {
  const auth = getAuth();
  const db = getFirestore();
  const user = auth.currentUser;

  let unsubscribeCurrency;
  if (user) {
    const userRef = doc(db, "users", user.uid);
    unsubscribeCurrency = onSnapshot(
      userRef,
      async (userDoc) => {
        if (userDoc.exists()) {
          const walletId = userDoc.data().walletId;
          if (walletId) {
            const walletRef = doc(db, "wallets", walletId);
            const walletDoc = await getDoc(walletRef);
            if (walletDoc.exists()) {
              setCurrency(walletDoc.data().currency || "");
            }
          }
        }
      },
      (error) => {
        console.error("Error fetching wallet currency:", error);
      }
    );
  }

  // Clean up the subscription on component unmount
  return () => {
    if (unsubscribeCurrency) {
      unsubscribeCurrency();
    }
  };
}, []);
```

Figure 4- 16 Code snippet for fetching wallet currency

Figure 4-15 shows the useEffect hook used to fetch the user's wallet currency in real-time via Firebase Firestore. It identifies the current authenticated user, retrieves their walletId, and listens for updates to the wallet's data using the onSnapshot method. The currency state is dynamically updated, ensuring that the user views the correct currency for transactions. The implementation includes error handling and a cleanup function to prevent memory leaks by unsubscribing from the listener when the component unmounts.

```
const prepareQRValue = () => {
  const details = {
    title: transaction.displayName,
    amount: transaction.amount,
    category: transaction.category,
    date: new Date(transaction.startDate).toLocaleDateString(),
    notes: transaction.notes || "No notes provided",
    imageURL: transaction.image,
    withPerson: transaction.withPerson,
    location: transaction.location,
  };
  return JSON.stringify(details);
};
```

Figure 4- 17 Code snippet for QR code generation

Figure 4-16 shows the `prepareQRValue` function, which serializes key transaction details into a JSON format to generate a QR code. The function includes information such as the transaction amount, category, date, notes, image URL, and any associated people or locations. This serialized data can then be easily shared or scanned to display transaction details elsewhere, providing users with a quick and efficient way to share or save transaction information via QR code technology.

```

const printToPDF = async () => {
  const htmlContent = `
<html>
<head>
  <style>
    body {
      font-size: 16px;
      font-family: Arial, Helvetica, sans-serif;
      padding: 20px;
      background-color: #fff;
    }
    h1 {
      color: #333;
      text-align: center;
      margin-bottom: 10px;
    }
    img {
      width: 100%;
      height: auto;
      max-height: 400px;
      border-radius: 10px;
      object-fit: cover;
      margin-bottom: 15px;
    }
    .info {
      margin-bottom: 8px;
    }
    footer {
      font-size: 12px;
      text-align: center;
      margin-top: 20px;
    }
  </style>
</head>
<body>
  <h1>Transaction Details</h1>
  
  <div class="info"><strong>Amount:</strong> ${currency} ${
transaction.amount
}</div>
  <div class="info"><strong>Category:</strong> ${
transaction.category
}</div>
  <div class="info"><strong>Date:</strong> ${new Date(
transaction.startDate
).toLocaleDateString()}</div>
  ${
transaction.notes
? `<div class="info"><strong>Notes:</strong> ${transaction.notes}</div>`
: ""
}
  ${
transaction.withPerson
? `<div class="info"><strong>With:</strong> ${transaction.withPerson}</div>`
: ""
}
  ${
transaction.location
? `<div class="info"><strong>Location:</strong> ${transaction.location}</div>`
: ""
}
  <footer>Generated on: ${new Date().toLocaleDateString()}</footer>
</body>
</html>`;

  try {
    const { uri } = await Print.printToFileAsync({ html: htmlContent });
    console.log("PDF saved at:", uri);
    await Sharing.shareAsync(uri, {
      UTI: ".pdf",
      mimeType: "application/pdf",
    });
    setModalVisible(true);
  } catch (error) {
    console.error("Error generating PDF: ", error);
    Alert.alert("Export Failed", "There was a problem exporting the PDF.");
  }
};

```

Figure 4- 18 Code snippet for generating and sharing transaction details

Figure 4-17 showcases a function that generates a PDF of transaction details and shares it via the device's share sheet. The printToPDF function creates an HTML structure containing dynamic transaction information, formatted neatly using HTML and CSS. This includes transaction images, amounts, categories, dates, and more. The PDF is generated using Expo's Print and Sharing libraries, enabling users to export and share transaction data for record-keeping. Error handling ensures a smooth user experience, notifying users if the export fails.

4.3.14 TransactionEdit.tsx

```
const handlePress = async () => {
  if (!amount || !selectedCategory.CategoryName) {
    Alert.alert("Error", "Please fill in all required fields.");
    return;
  }

  setLoading(true);
  const db = getFirestore();
  const auth = getAuth();

  try {
    let downloadURL = image;

    // Upload only if it's a new image
    if (isNewImage) {
      const storage = getStorage();
      const response = await fetch(image);
      const blob = await response.blob();
      const spaceRef = ref(storage, `TransactionImages/${transaction.key}`);
      const snapshot = await uploadBytes(spaceRef, blob);
      downloadURL = await getDownloadURL(snapshot.ref);
    }

    // Update the Firestore document
    if (auth.currentUser) {
      const transactionRef = doc(db, "Transactions", transaction.key);
      await updateDoc(transactionRef, {
        amount,
        category: selectedCategory.CategoryName,
        type: transactionType,
        notes,
        date,
        image: downloadURL,
        withPerson,
        location,
        updatedAt: new Date().getTime(),
      });

      Alert.alert("Update Successful", "", [
        { text: "OK", onPress: () => navigation.navigate("TransactionList") },
      ]);
    } else {
      throw new Error("Authentication failed. User not found.");
    }
  } catch (err) {
    console.error("Error updating transaction: ", err);
    Alert.alert("There is something wrong!", err.message);
  }

  setLoading(false);
};
```

Figure 4- 19 Code Snippet for Handling Transaction Update

Figure 4-18 illustrates the `handlePress` function, which processes updates to the transaction in `TransactionEdit.tsx`. The function first checks for required fields (amount and category) before proceeding. If the image is new, it uploads the image to Firebase Storage. Afterward, it updates the transaction document in Firestore with the new details such as amount, category, type, and any changes to the image, location, or person involved. Upon successful update, it navigates the user back to the transaction list, ensuring a seamless editing experience.

4.3.15 Budget.tsx

```
useEffect(() => {
  const auth = getAuth();
  const db = getFirestore();
  const user = auth.currentUser;

  if (user) {
    setLoading(true);

    const fetchWalletData = async () => {
      if (user) {
        const userRef = doc(db, "users", user.uid);
        try {
          const userDoc = await getDoc(userRef);
          if (userDoc.exists()) {
            const walletId = userDoc.data().walletId;
            if (walletId) {
              const walletRef = doc(db, "wallets", walletId);
              const walletDoc = await getDoc(walletRef);
              if (walletDoc.exists()) {
                setCurrency(walletDoc.data().currency || "USD");
              }
            }
          }
        } catch (error) {
          console.error("Error fetching wallet data:", error);
        }
      }
    };
    fetchWalletData();

    const unsubscribeBudgets = onSnapshot(
      query(
        collection(db, "Budgets"),
        where("CreatedUser.uid", "==", user.uid)
      ),
      async (budgetSnapshot) => {
        const fetchedBudgets = budgetSnapshot.docs.map((doc) => ({
          id: doc.id,
          ...doc.data(),
        })) as BudgetItem[];

        const unsubscribeTransactions = onSnapshot(
          query(
            collection(db, "Transactions"),
            where("CreatedUser.uid", "==", user.uid)
          ),
          async (transactionSnapshot) => {
            const fetchedTransactions = transactionSnapshot.docs.map(
              (doc) => ({
                id: doc.id,
                ...doc.data(),
              })
            ) as Transaction[];

            // Check each budget for overspending and update isOverspent property
            for (const budget of fetchedBudgets) {
              const budgetAmount = parseFloat(budget.amount);
              const spentAmount = fetchedTransactions
                .filter(
                  (transaction) => transaction.category === budget.category
                )
                .reduce(
                  (acc, transaction) => acc + parseFloat(transaction.amount),
                  0
                );

              budget.spentAmount = spentAmount;
              budget.isOverspent = spentAmount > budgetAmount;

              // Send notification if overspent
              if (budget.isOverspent) {
                await sendNotification(
                  budget.category,
                  budgetAmount,
                  spentAmount
                );

                const budgetRef = doc(db, "Budgets", budget.id);
                await updateDoc(budgetRef, {
                  isOverspent: true,
                  spentAmount: spentAmount,
                });
              }
            }

            setBudgets(fetchedBudgets);
            setLoading(false);
          },
          (error) => {
            console.error("Error fetching transactions: ", error);
            Alert.alert("Error", "Could not fetch transactions.");
          }
        );

        return () => unsubscribeTransactions();
      },
      (error) => {
        console.error("Error fetching budgets: ", error);
        Alert.alert("Error", "Could not fetch budgets.");
        setLoading(false);
      }
    );

    return () => {
      unsubscribeBudgets();
    };
  }
}, []);
```

Figure 4- 20 Code Snippet for Budget Retrieval and Update

Figure 4-19 demonstrates the useEffect function, which manages budget and transaction data retrieval in Budget.tsx. It first fetches the user's wallet information to set the correct currency. Next, it uses an onSnapshot listener to track budgets and transactions in real time. For each budget, the spent amount is calculated from transactions, and if the user exceeds the budget, a notification is triggered. The corresponding budget document in Firestore is also updated to reflect the overspending, ensuring the user is promptly alerted about financial limits.

4.3.16 BudgetAdd.tsx

```
const handleSaveBudget = async () => {
  if (!amount || !selectedCategory.CategoryName || !selectedTimeRange) {
    Alert.alert("Error", "All fields are required.");
    return;
  }

  setLoading(true);
  const auth = getAuth();
  const db = getFirestore();

  const currentDate = new Date();
  let startDate, endDate;

  switch (selectedTimeRange) {
    case "week":
      startDate = startOfWeek(currentDate);
      endDate = endOfWeek(currentDate);
      break;
    case "month":
      startDate = startOfMonth(currentDate);
      endDate = endOfMonth(currentDate);
      break;
    case "year":
      startDate = startOfYear(currentDate);
      endDate = endOfYear(currentDate);
      break;
    default:
      Alert.alert("Error", "Invalid time range selected.");
      setLoading(false);
      return;
  }

  if (auth.currentUser) {
    const currentUser = auth.currentUser;

    try {
      const budgetData = {
        amount,
        category: selectedCategory.CategoryName,
        timeRange: selectedTimeRange,
        startDate: startDate.getTime(),
        endDate: endDate.getTime(),
        CreatedUser: {
          uid: currentUser.uid,
          displayName: currentUser.displayName || "",
        },
        createdAt: serverTimestamp(),
      };

      const docRef = await addDoc(collection(db, "Budgets"), budgetData);
      console.log("Document written with ID: ", docRef.id);
      navigation.goBack();
    } catch (error) {
      console.error("Error adding budget: ", error);
      Alert.alert("Error", "There was an issue saving the budget.");
    } finally {
      setLoading(false);
    }
  } else {
    Alert.alert("Error", "User not authenticated.");
    setLoading(false);
  }
};
```

Figure 4- 21 Code Snippet for Budget Creation

Figure 4-20 demonstrates how the `handleSaveBudget` function saves a new budget entry in `BudgetAdd.tsx`. It first validates the form inputs (amount, category, and time range) before proceeding. Depending on the selected time range (week, month, or year), the function calculates the budget's start and end dates using `date-fns` library. It then retrieves the authenticated user's information and saves the budget data to Firebase Firestore, ensuring the correct time range, amount, and category are stored. Finally, the document is created with a server timestamp to maintain consistency.

4.3.17 BudgetDetail.tsx

```

useEffect(() => {
  const auth = getAuth();
  const db = getFirestore();
  const user = auth.currentUser;
  setLoading(true);

  const fetchCurrency = async () => {
    if (user) {
      const userRef = doc(db, "users", user.uid);
      const userDoc = await getDoc(userRef);
      if (userDoc.exists()) {
        const walletId = userDoc.data().walletId;
        if (walletId) {
          const walletRef = doc(db, "wallets", walletId);
          const walletDoc = await getDoc(walletRef);
          if (walletDoc.exists()) {
            setCurrency(walletDoc.data().currency || "");
          }
        }
      }
    }
  };

  let unsubscribe;
  if (user) {
    const qBudget = query(
      collection(db, "Budgets"),
      where("CreatedUser.uid", "=", user.uid),
      where("category", "=", category)
    );

    unsubscribe = onSnapshot(
      qBudget,
      (snapshot) => {
        if (!snapshot.empty) {
          const budgetData = snapshot.docs[0].data();
          const formattedStartDate = formatDate(budgetData.startDate);
          const formattedEndDate = formatDate(budgetData.endDate);
          const budgetLineValue = parseFloat(budgetData.amount);

          setChartData({
            labels: [formattedStartDate, formattedEndDate],
            datasets: [
              {
                data: [budgetAmount, budgetAmount],
                color: (opacity = 1) => `rgba(255, 0, 0, ${opacity})`,
                strokeWidth: 2,
              },
              {
                data: [budgetAmount, budgetAmount + overspentAmount],
                color: (opacity = 1) => `rgba(0, 255, 0, ${opacity})`,
                strokeWidth: 2,
              },
            ],
          });

          setOverspent(
            budgetData.amount < overspentAmount
              ? overspentAmount - budgetData.amount
              : 0
          );
        }
        setLoading(false);
      },
      (error) => {
        console.error("Error fetching budget data:", error);
        Alert.alert("Error", "Could not fetch budget data.");
        setLoading(false);
      }
    );
  }

  fetchCurrency();

  // Clean up the subscription on component unmount
  return () => unsubscribe && unsubscribe();
}, [
  category,
  overspentAmount,
  setLoading,
  setCurrency,
  setChartData,
  setOverspent,
]);

```

Figure 4- 22 Code Snippet for Budget Data Retrieval and Overspent Update

Figure 4-21 showcases the useEffect hook in BudgetDetail.tsx, which dynamically fetches and updates the budget and spending data for the user. It first retrieves the user's wallet currency and sets up a real-time snapshot listener on the selected budget using Firestore's onSnapshot. The chart is populated with budget and overspending data based on the fetched information. The useEffect also includes cleanup functionality to unsubscribe from Firestore when the component is unmounted. This approach ensures real-time updates and accurate representation of the user's financial data.

```
const handleDeletePress = async () => {
  Alert.alert(
    "Delete Budget",
    "Are you sure you want to delete this budget?",
    [
      {
        text: "Cancel",
        style: "cancel",
        onPress: () => setIsDeleting(false),
      },
      {
        text: "Delete",
        onPress: async () => {
          if (!isDeleting) {
            setIsDeleting(true);
            const db = getFirestore();
            if (route.params.key) {
              try {
                await deleteDoc(doc(db, "Budgets", route.params.key));
                navigation.goBack();
              } catch (error) {
                Alert.alert("Error", "Failed to delete budget.");
                console.error("Delete operation failed:", error);
              }
            } else {
              Alert.alert("Error", "No budget key found for deletion.");
            }
            setIsDeleting(false);
          }
        },
      },
    ],
  );
};
```

Figure 4-23 Code Snippet for Budget Deletion

Figure 4-22 showcases the delete functionality in BudgetDetail.tsx. The handleDeletePress function prompts users to confirm budget deletion with an alert. Upon confirming, the function deletes the budget document from Firestore using deleteDoc if the budget key exists. It prevents duplicate deletion by disabling the button during the process and includes error handling in case the deletion fails. This ensures that users have a safe and efficient way to manage and remove their budgets.

4.3.18 UpdateGoals.tsx

```
useEffect(() => {
  const fetchWalletId = async () => {
    const userRef = doc(db, "users", auth.currentUser?.uid);
    const docSnap = await getDoc(userRef);
    if (docSnap.exists() && docSnap.data().walletId) {
      const walletRef = doc(db, "wallets", docSnap.data().walletId);
      const walletSnap = await getDoc(walletRef);
      if (walletSnap.exists()) {
        setSelectedGoal(walletSnap.data().financialGoal);
        setTargetSavings(walletSnap.data().targetSavings);
      }
    }
  };
  fetchWalletId();
}, []);

const updateGoals = async () => {
  const userRef = doc(db, "users", auth.currentUser?.uid);
  const docSnap = await getDoc(userRef);
  if (docSnap.exists() && docSnap.data().walletId) {
    setIsLoading(true);
    try {
      await setDoc(
        doc(db, "wallets", docSnap.data().walletId),
        {
          financialGoal: selectedGoal,
          targetSavings: targetSavings,
        },
        { merge: true }
      );
      Alert.alert("Success", "Your goals have been updated.");
      navigation.goBack();
    } catch (error) {
      Alert.alert("Error", "Failed to update goals.");
      console.error("Error updating goals:", error);
    } finally {
      setIsLoading(false);
    }
  } else {
    Alert.alert("Error", "No wallet linked to your account.");
  }
};
```

Figure 4- 24 Code Snippet for Fetching and Updating Financial Goals

Figure 4-23 shows the logic for fetching and updating the user's financial goals in UpdateGoals.tsx. The useEffect hook retrieves the user's wallet and sets their current financial goal and target savings in the state. The updateGoals function then allows the user to modify these fields and persist the changes in Firestore. The use of setDoc with merge: true ensures existing data is preserved. This process ensures that users can customize and track their financial goals easily.

4.3.19 TrackAchievements.tsx

```
useEffect(() => {
  const db = getFirestore();
  const auth = getAuth();
  const currentUser = auth.currentUser;

  if (currentUser) {
    checkAndUnlockAchievements(db, currentUser);
  }
}, []);

const checkAndUnlockAchievements = async (
  db: Firestore,
  currentUser: firebase.User
) => {
  try {
    const transactionsSnapshot = await getDocs(
      collection(db, "Transactions")
    );
    const budgetsSnapshot = await getDocs(collection(db, "Budgets"));
    const hasTransactions = !transactionsSnapshot.empty;
    const hasBudgets = !budgetsSnapshot.empty;

    let updatedAchievements = [...achievements];

    for (let achievement of updatedAchievements) {
      let isUnlocked = achievement.unlocked;

      if (achievement.id === "first_transaction" && hasTransactions) {
        isUnlocked = true;
      }
      if (achievement.id === "first_budget" && hasBudgets) {
        isUnlocked = true;
      }

      if (isUnlocked && !achievement.unlocked) {
        await unlockAchievement(db, currentUser, achievement);
        sendNotification(
          "Achievement Unlocked",
          `${achievement.title}: ${achievement.description}`
        );
        achievement.unlocked = true;
      }
    }

    setAchievements(updatedAchievements);
  } catch (error) {
    console.error("Error fetching data:", error);
  }
};
```

Figure 4- 25 Code Snippet for Tracking Achievements and Sending Notifications

Figure 4-24 demonstrates the core logic for checking and unlocking achievements based on user activity in TrackAchievements.tsx. The checkAndUnlockAchievements function evaluates the user's transaction and budget data to determine which achievements are unlocked. Once an achievement is unlocked, it updates the achievement state and triggers a notification using Expo's notification system, enhancing the user's engagement.

```

const unlockAchievement = async (
  db: Firestore,
  currentUser: firebase.User,
  achievement: {
    id: any;
    title: any;
    description: any;
    icon?: string;
    unlocked?: boolean;
  }
) => {
  try {
    await addDoc(collection(db, "Achievements"), {
      CreatedUser: {
        uid: currentUser.uid,
        displayName: currentUser.displayName || "",
      },
      achievementId: achievement.id,
      title: achievement.title,
      description: achievement.description,
      unlockedDate: new Date(),
    });
  } catch (error) {
    console.error("Error saving achievement:", error);
  }
};

// Function to send notification
const sendNotification = async (title: string, body: string) => {
  await Notifications.scheduleNotificationAsync({
    content: {
      title: title,
      body: body,
    },
    trigger: null, // sends it right away
  });
};

```

Figure 4-26 Code Snippet for Unlocking Achievements and Sending Notifications

Figure 4-25 focuses on the specific function used to persist unlocked achievements in Firestore. The unlockAchievement function records each achievement with its associated user, ensuring it is saved in the database. The snippet also includes sendNotification, which immediately alerts users of their unlocked achievements, adding a responsive user experience.

4.3.20 LoanAdd.tsx

```
const calculateMonthlyPayment = () => {
  // Ensure all values are numbers
  const principal = parseFloat(loanAmount) || 0;
  const rate = parseFloat(interestRate) || 0;
  const term = parseFloat(loanTerm) || 0;

  if (!principal || !rate || !term) {
    Alert.alert("Error", "Please enter valid numbers for all fields.");
    return;
  }

  const monthlyRate = rate / 1200; // Convert annual rate to monthly and percentage to decimal
  const payment =
    principal * (monthlyRate / (1 - Math.pow(1 + monthlyRate, -term * 12)));
  setMonthlyPayment(payment.toFixed(2));
};

const getLoanAdvice = () => {
  if (!monthlyPayment) return "Enter loan details to get advice.";

  const payment = parseFloat(monthlyPayment);
  if (payment > loanAmount / 2) {
    return "This loan has a very high monthly payment. Consider a longer term or a lower amount.";
  } else if (payment < loanAmount / 100) {
    return "Your monthly payment is low, which may mean a longer term and more interest paid over time.";
  } else {
    return "Your loan terms seem balanced. Ensure the monthly payment fits your budget.";
  }
};
```

Figure 4- 27 Code Snippet for Loan Calculation and Advice Generation

Figure 4.26 demonstrates the loan calculation and advice generation functionality. The `calculateMonthlyPayment` function computes the monthly payment based on the loan amount, interest rate, and loan term, providing users with a precise financial estimate. Following this, the `getLoanAdvice` function evaluates the computed payment and offers personalized advice, helping users assess if the loan terms are appropriate based on their budget. Together, these functions ensure users have accurate loan information and clear guidance for financial decision-making.

4.3.21 AIChatbot.tsx

```
const API_URL = "https://api.openai.com/v1/chat/completions";
const YOUR_API_KEY = "sk-PRMrUKaFKyLW0H9wTnNt3BlbkFJbIspCN8nzgVTkMpC0FAB";
const MAX_TOKENS = 2048;

useEffect(() => {
  console.log("AIChatbox mounted with userId:", userId);
  clearMessages().then(() => {
    loadInitialMessages();
  });
}, [userId]);

const firstMessage = () => {
  const greetingMessage = {
    _id: new Date().getTime().toString() + Math.random().toString(36).substring(7),
    text: "Hello! I'm your financial assistant. How can I help you today?",
    createdAt: new Date(),
    user: {
      _id: 2,
      name: "Chatbot",
    },
  };
  return [greetingMessage];
};

const startTutorial = () => {
  const tutorialMessages = [
    {
      _id: new Date().getTime().toString() + Math.random().toString(36).substring(7),
      text: "Hello! I'm your financial assistant. How can I help you today?",
      createdAt: new Date(),
      user: {
        _id: 2,
        name: "Chatbot",
      },
      quickReplies: {
        type: 'radio',
        keepIt: true,
        values: [
          {
            title: 'Check my financial health',
            value: 'financial_health_score',
          },
          {
            title: 'Give me financial tips',
            value: 'financial_tips',
          },
          {
            title: 'How to save more money?',
            value: 'save_money_tips',
          },
        ],
      },
    },
  ];
  return tutorialMessages;
};
```

Figure 4- 28 Code Snippet for API Configuration and Tutorial Initialization

Figure 4- 27 how the OpenAI API is configured and used to initialize the chatbot tutorial messages. It includes setting up the API URL, the secret key for authentication, and the token limit for requests. The firstMessage function is responsible for sending an initial greeting, while the startTutorial function guides first-time users by presenting quick reply options like "Check my financial health" or "Give me financial tips," allowing for an interactive and user-friendly experience.

```
const onSend = useCallback((messages = []) => {
  if (isLoading) return;
  setMessages((previousMessages) =>
    GiftedChat.append(previousMessages, messages)
  );
  storeMessages(GiftedChat.append(messages, messages));
  const value = messages[0].text;
  callApi(value, userId); // Pass userId to callApi
}, [userId, isLoading]);
```

Figure 4- 29 Code Snippet for Handling User Messages

Figure 4- 28 demonstrates how user messages are handled. The onSend function captures user input and appends it to the message state, storing it for continuity in conversations. The user's input is then passed to the callApi function, where it is processed by OpenAI's GPT model, which generates responses based on the user's financial data.

```
const handleQuickReply = useCallback(async (replies) => {
  if (replies.length === 1 && !isButtonDisabled) {
    setIsButtonDisabled(true);
    setIsTyping(true);

    const reply = replies[0];
    console.log("Quick reply selected:", reply.value, reply.title);

    if (reply.value === 'financial_health_score') {
      await sendFinancialHealthScore();
    } else if (reply.value === 'financial_tips') {
      await sendFinancialTips();
    } else if (reply.value === 'save_money_tips') {
      await sendMoneySavingTips();
    } else if (reply.value === 'spending_habits') {
      await analyzeSpendingHabits();
    } else if (reply.value === 'no_thanks') {
      addNewMessage("You're welcome! If you have any other questions later, feel free to ask.");
    }

    setIsButtonDisabled(false);
    setIsTyping(false);
  }
}, [userId, isButtonDisabled]);
```

Figure 4- 30 Code Snippet for Handling Quick Replies

Figure 4- 29 illustrates how quick replies are handled within the chatbot. The user selects predefined options such as financial health scores or money-saving tips. The handleQuickReply function triggers specific actions based on the selected reply. It ensures the chatbot remains responsive to user preferences by activating various functions like sendFinancialHealthScore or sendFinancialTips, giving users personalized guidance.

```

const sendFinancialHealthScore = async () => {
  const userFinancialData = await getUserFinancialData(userId);
  const financialHealthScore = calculateFinancialHealthScore(
    userFinancialData.transactionsData,
    userFinancialData.walletsData
  );

  const message = {
    _id: Math.random().toString(36).substring(7),
    text: `Your current financial health score is ${financialHealthScore}.`,
    createdAt: new Date(),
    user: {
      _id: 2,
      name: "Chatbot",
    },
  };

  setMessages((previousMessages) =>
    GiftedChat.append(previousMessages, [message])
  );
  storeMessages(GiftedChat.append(messages, [message]));

  setTimeout(() => {
    const followUpMessage = {
      _id: Math.random().toString(36).substring(7),
      text: "Do you have any other questions or need further assistance?",
      createdAt: new Date(),
      user: {
        _id: 2,
        name: "Chatbot",
      },
      quickReplies: {
        type: 'radio',
        keepIt: true,
        values: [
          {
            title: 'Give me financial tips',
            value: 'financial_tips',
          },
          {
            title: 'Analyze my spending habits',
            value: 'spending_habits',
          },
          {
            title: 'No, thank you',
            value: 'no_thanks',
          },
        ],
      },
    };

    setMessages((previousMessages) =>
      GiftedChat.append(previousMessages, [followUpMessage])
    );
    storeMessages(GiftedChat.append(messages, [followUpMessage]));
  }, 1000);
};

```

Figure 4- 31 Code Snippet for Sending Financial Health Score

In this figure, the sendFinancialHealthScore function calculates a user's financial health score based on transaction and wallet data. The calculated score is returned as a message, followed by a suggestion for further queries. The score helps users understand their financial stability, allowing for a personalized financial advisory experience.

```

const sendFinancialTips = async () => {
  try {
    const userFinancialData = await getUserFinancialData(userId);
    const prompt = `Provide personalized money-saving tips based on the following financial data: ${JSON.stringify(userFinancialData)}`;

    const messagePayload = {
      model: "gpt-3.5-turbo",
      messages: [
        {
          role: "user",
          content: prompt,
        }
      ],
      max_tokens: MAX_TOKENS,
    };

    const response = await fetch(API_URL, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
        Authorization: `Bearer ${YOUR_API_KEY}`,
      },
      body: JSON.stringify(messagePayload),
    });

    if (!response.ok) {
      const errorData = await response.json();
      console.error("API Response Error:", errorData);
      Alert.alert("API Error", errorData.error.message);
      throw new Error(`API Error: ${errorData.error.message}`);
    }

    const data = await response.json();

    if (data.choices && data.choices.length > 0 && data.choices[0].message) {
      const reply = data.choices[0].message.content.trim();
      if (reply) {
        const finalMessage = {
          _id: Math.random().toString(36).substring(7),
          text: reply,
          createdAt: new Date(),
          user: {
            _id: 2,
            name: "Chatbot",
          },
        };
        setMessages(previousMessages) =>
          GiftedChat.append(previousMessages, finalMessage);
      }
    } else {
      throw new Error("No response from AI");
    }
  } catch (error) {
    console.error("API call failed", error);
    Alert.alert("Error", "Failed to get money-saving tips from AI.");
  }
};

```

Figure 4- 32 Code Snippet for Sending Financial Tips

Figure 4- 31 focuses on generating personalized financial tips using OpenAI’s GPT-3 model. The sendFinancialTips function sends a prompt to the GPT API, which uses the user's financial data to return tailored money-saving tips. This enables the chatbot to offer detailed and customized financial advice, enhancing the user’s financial management journey.

While the chatbot's primary focus is on financial assistance, the integration with OpenAI’s GPT-3.5 enables it to handle a broader range of topics. The model is capable of engaging with user-generated queries outside of predefined financial messages, ensuring adaptability to various user needs. This versatile capability allows the chatbot to provide detailed, dynamic responses, making it a well-rounded assistant for more than just finance-related conversations.

4.3.22 FinancialAnalysis.tsx & finance_forecast.py

```
from flask import Flask, request, jsonify
from prophet import Prophet
import pandas as pd
from flask_cors import CORS

app = Flask(__name__)
CORS(app)

@app.route('/forecast', methods=['POST'])
def forecast():
    try:
        data = request.json
        df = pd.DataFrame(data)
        df['ds'] = pd.to_datetime(df['ds'])

        model = Prophet()
        model.fit(df)

        future = model.make_future_dataframe(periods=180)
        forecast = model.predict(future)

        forecast['ds'] = forecast['ds'].dt.strftime('%Y-%m-%d')

        return jsonify({
            'status': 'success',
            'forecast': forecast[['ds', 'yhat', 'yhat_lower', 'yhat_upper']].to_dict('records')
        })
    except Exception as e:
        return jsonify({
            'status': 'error',
            'message': str(e)
        }), 400

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5001)
```

Figure 4- 33 Flask API for Time Series Forecasting with Prophet (*finance_forecast.py*)

Figure 4-32 shows the implementation of a forecasting service using Flask and the Prophet library for time series prediction. The forecast function accepts a POST request with JSON data representing the time series, fits the Prophet model to it, and generates future predictions for 180 periods. The results include predicted values (yhat) along with confidence intervals (yhat_lower, yhat_upper). The forecast data is returned as a JSON response to the client. The code handles errors by returning an error message in JSON format if the forecasting process fails.

```

const fetchTransactionData = async () => {
  try {
    setLoading(true);
    setError(null);
    const q = query(
      collection(db, "Transactions"),
      where("CreatedUser.uid", "==", userId),
      orderBy("date", "asc")
    );
    const querySnapshot = await getDocs(q);
    const fetchedTransactions = [];

    querySnapshot.forEach((doc) => {
      const data = doc.data();
      if (data.date && data.amount && data.category !== "Receipt") {
        fetchedTransactions.push({
          ...data,
          date: new Date(data.date.seconds * 1000),
          amount: Number(data.amount)
        });
      }
    });

    setTransactions(fetchedTransactions);
    prepareTimeSeriesData(fetchedTransactions);
    generateInsights(fetchedTransactions);
  } catch (error) {
    console.error("Error fetching transactions:", error);
    setError("Failed to fetch transaction data. Please try again later.");
  } finally {
    setLoading(false);
  }
};

```

Figure 4- 34 Code Snippet for Fetching and Processing Transaction Data (FinancialAnalysis.tsx)

The code snippet in this figure 4- 33 is responsible for fetching user transaction data from Firebase Firestore. It queries transactions belonging to the authenticated user, processes the data into a structured format, and excludes any receipts. Once fetched, the data is used to prepare time-series and insights for financial analysis.

```

const prepareTimeSeriesData = (transactions) => {
  const timeSeriesData = {};

  transactions.forEach((transaction) => {
    // Filter by category
    if (selectedCategory !== 'all' && transaction.category !== selectedCategory) return;

    const month = transaction.date.getMonth();
    const year = transaction.date.getFullYear();

    if (isNaN(month) || isNaN(year)) {
      console.error("Invalid date detected:", transaction.date);
      return;
    }

    const key = `${year}-${month}`;

    if (!timeSeriesData[key]) {
      timeSeriesData[key] = { income: 0, expenses: 0 };
    }

    if (transaction.type === "income") {
      timeSeriesData[key].income += transaction.amount || 0;
    } else {
      timeSeriesData[key].expenses += transaction.amount || 0;
    }
  });

  const timeSeriesArray = Object.keys(timeSeriesData)
    .sort()
    .map((key) => {
      const [year, month] = key.split('-');
      return {
        date: new Date(Number(year), Number(month), 1),
        income: timeSeriesData[key].income,
        expenses: timeSeriesData[key].expenses,
      };
    });

  setTimeSeries(timeSeriesArray);

  const barChartData = {
    labels: timeSeriesArray.map((data) => `${data.date.getFullYear()}-${data.date.getMonth() + 1}`),
    datasets: [{ data: timeSeriesArray.map((data) => data.expenses || 0) }],
  };

  setBarChartData(barChartData);
};

```

Figure 4- 35 Code Snippet for Time Series Data Preparation (FinancialAnalysis.tsx)

Figure 4- 34 shows the function for preparing time-series data. It organizes the transaction data into a time series format based on the user's selected category. The code processes the income and expenses for each month, storing the data in timeSeriesArray, which is later used for visualization.


```

const handleMakePrediction = async () => {
  if (timeSeries.length === 0) {
    console.error("No time series data available for prediction.");
    return;
  }

  try {
    const prophetData = timeSeries.map(item => ({
      ds: item.date.toISOString().split('T')[0],
      y: item.expenses
    }));

    const response = await axios.post('http://localhost:5001/forecast', prophetData);

    if (response.data.status === 'success') {
      const forecastResult = response.data.forecast;

      setForecast(forecastResult);

      const updatedBarChartData = {
        labels: [...barChartData.labels, ...forecastResult.map(d => d.ds.slice(0, 7))],
        datasets: [{
          data: [...barChartData.datasets[0].data, ...forecastResult.map(d => d.yhat)]
        }]
      };
      setBarChartData(updatedBarChartData);
    } else {
      throw new Error(response.data.message || 'Forecast failed!');
    }
  } catch (error) {
    console.error("Error making prediction:", error);
    setError("Failed to make forecast. Please try again later.");
  }
};

```

Figure 4- 36 Code Snippet for Prophet Forecast Functionality (FinancialAnalysis.tsx)

The handleMakePrediction function in this figure 4- 35 uses Prophet (via the Flask API) to make a future prediction based on time-series data. It sends a POST request to the Flask server, and once the forecast is received, the data is used to extend the chart's visualization with the predicted values.

```

const generateInsights = (transactions) => {
  const now = new Date();
  const currentMonth = now.getMonth();
  const currentYear = now.getFullYear();

  // Filter transactions to only include those from the current month
  const currentMonthTransactions = transactions.filter(transaction => {
    const transactionDate = transaction.date;
    return transactionDate.getMonth() === currentMonth && transactionDate.getFullYear() === currentYear;
  });

  const totalSpending = currentMonthTransactions.reduce((sum, transaction) => sum + transaction.amount, 0);
  const topCategories = {};

  currentMonthTransactions.forEach(transaction => {
    if (!topCategories[transaction.category]) {
      topCategories[transaction.category] = 0;
    }
    topCategories[transaction.category] += transaction.amount;
  });

  const topCategory = Object.keys(topCategories).reduce((a, b) => topCategories[a] > topCategories[b] ? a : b);

  setInsights({
    totalSpending,
    topCategory,
    topCategoryAmount: topCategories[topCategory],
  });
};

```

Figure 4- 37 Code Snippet for Generating Financial Insights (FinancialAnalysis.tsx)

Figure 4- 36 describes the function for generating financial insights based on user transactions. It calculates the total spending for the current month and determines the top spending category. These insights are displayed in the financial analysis view.

To conclude the analysis of both FinancialAnalysis.tsx and finance_forecast.py, these two files work together to provide an integrated solution for financial forecasting and insights. FinancialAnalysis.tsx is responsible for fetching and visualizing transaction data, generating time-series data, and creating insights on the user's financial habits. The finance_forecast.py service adds predictive analytics by leveraging Prophet to forecast future trends based on the time-series data sent from the FinancialAnalysis component. Together, these pages achieve the project's objective of financial forecasting and analysis, helping users make informed decisions.

4.3.23 ReceiptOCR.tsx

```
const handleCaptureImage = async () => {
  try {
    let result = await ImagePicker.launchCameraAsync({
      mediaTypes: ImagePicker.MediaTypeOptions.Images,
      allowsEditing: true,
      quality: 1,
    });

    if (!result.canceled) {
      const uri = result.assets[0].uri;
      setImageUri(uri);
      processImage(uri);
    }
  } catch (error) {
    console.error("Error launching camera: ", error);
    Alert.alert("Error", "Failed to launch camera. Please ensure all permissions are granted.");
  }
};

const processImage = async (uri: string) => {
  setProcessingImage(true); // Start processing the image
  const apiKey = 'AizaSyBvLQ4ctFosCTqCg4u0Fv0FNyXGHlqt00s';

  const base64Image = await convertImageToBase64(uri);
  const base64Data = base64Image.replace(/^data:image\/[a-z]+;base64/, "");

  const requestBody = {
    requests: [
      {
        image: {
          content: base64Data,
        },
        features: [
          {
            type: 'TEXT_DETECTION',
          },
        ],
      },
    ],
  };

  try {
    const response = await axios.post(
      'https://vision.googleapis.com/v1/images:annotate?key=${apiKey}',
      requestBody
    );
    const detectedText = response.data.responses[0].fullTextAnnotation?.text;
    setOcrResult(detectedText || 'No text detected');
    extractDetails(detectedText || '');
  } catch (error) {
    console.error('Error processing image:', error.response?.data || error.message);
    setOcrResult('Failed to process image');
  } finally {
    setProcessingImage(false); // End processing the image
  }
};
```

Figure 4- 38 Code Snippet for Receipt Capture and OCR Processing

This code segment allows users to capture receipt images using their device's camera and processes them using Google Cloud Vision API for Optical Character Recognition (OCR). The image is converted to a base64 string and sent to the Vision API, where the text is extracted. The processImage() function handles this, using TEXT_DETECTION to analyze the image, and then extracts key details like the total amount, vendor name, and date.

```
const extractDetails = (text: string) => {
  const totalMatch = text.match(/(?:TOTAL|Total|total|SUBTOTAL|Subtotal|subtotal|Amount Due|amount due)[^d]*(\d,+\.\d{2})/);
  setAmount(totalMatch ? totalMatch[1].replace(',', '') : '');

  const dateMatch = text.match(/(\d{2}[\-\/]\d{2}[\-\/]\d{4})/ || text.match(/(\d{4}[\-\/]\d{2}[\-\/]\d{2})/);
  setDate(dateMatch ? dateMatch[0] : '');

  const vendorMatch = text.split('\n')[0];
  setVendor(vendorMatch || '');
};
```

Figure 4- 39 Code Snippet for Data Extraction and Field Population

The extractDetails() function processes the detected text from the OCR result and extracts important details such as the total amount, date, and vendor. Regular expressions are used to match patterns in the text for amounts and dates, which are then displayed in the appropriate input fields for verification and potential editing by the user.

```

const handleSaveToFirebase = async () => {
  console.log("Started saving the receipt");

  if (!amount || !date || !vendor) {
    Alert.alert("Please complete all the fields.");
    console.log("Form is incomplete. Fields:", { amount, date, vendor });
    return;
  }
  setLoading(true);

  try {
    console.log("Starting to save receipt. Date:", date);

    const dateParts = date.split('/');
    if (dateParts.length !== 3) {
      throw new Error("Invalid date format");
    }
    const [month, day, year] = dateParts.map(num => parseInt(num, 10));
    console.log("Parsed date components:", { month, day, year });

    const dateObject = new Date(year, month - 1, day);
    console.log("Created Date object:", dateObject);

    if (isNaN(dateObject.getTime())) {
      console.error("Invalid date:", date);
      throw new Error("Invalid date");
    }

    const minDate = new Date('1/1/1970');
    const maxDate = new Date('12/31/2999');
    if (dateObject < minDate || dateObject > maxDate) {
      console.error("Date out of valid range (1970-2999):", dateObject);
      throw new Error("Date is out of valid range (1970-2999)");
    }

    const auth = getAuth();
    const db = getFirestore();
    const storage = getStorage();

    let downloadURL = null;
    if (imageUri) {
      console.log("Uploading image from URI:", imageUri);
      try {
        const response = await fetch(imageUri);
        console.log("Image fetched:", response);
        const blob = await response.blob();
        console.log("Image blob created. Uploading to Firebase Storage...");

        const imageRef = ref(storage, `ReceiptImages/${auth.currentUser?.uid || 'unknown'}/${Date.now()}.jpg`);
        console.log("Firebase Storage ref created:", imageRef);

        const snapshot = await uploadBytes(imageRef, blob);
        console.log("Image uploaded successfully. Snapshot:", snapshot);

        downloadURL = await getDownloadURL(snapshot.ref);
        console.log("Download URL obtained:", downloadURL);
      } catch (uploadError) {
        console.error("Error uploading image:", uploadError);
        console.log("Continuing without image due to upload error");
      }
    }

    const startDate = new Date().getTime();
    console.log("Starting timestamp:", startDate);

    if (auth.currentUser) {
      const currentUser = auth.currentUser;
      console.log("Authenticated user:", currentUser.uid);

      const receiptData = {
        amount: parseFloat(amount) || 0,
        category: "Receipt",
        type: "expense",
        vendor: vendor || "Unknown Vendor",
        date: Timestamp.fromDate(dateObject),
        image: downloadURL,
        startDate: Timestamp.fromMillis(startDate),
        updatedAt: Timestamp.fromMillis(startDate),
        CreatedUser: {
          uid: currentUser.uid,
          displayName: currentUser.displayName || "",
        },
      };

      console.log("Receipt data prepared:", receiptData);

      await addDoc(collection(db, "Transactions"), receiptData);
      console.log("Receipt saved successfully to Firestore");

      Alert.alert("Receipt saved successfully.", "", [
        { text: "OK", onPress: () => navigation.navigate("TransactionAdd") },
      ]);
    } else {
      console.error("Authentication failed. No user found.");
      throw new Error("Authentication failed. User not found.");
    }
  } catch (error) {
    console.error("Error in handleSaveToFirebase:", error);
    Alert.alert("Failed to save receipt.", error.message);
  } finally {
    setLoading(false);
    console.log("Finished saving the receipt.");
  }
};

```

Figure 4- 40 Code Snippet for Saving Receipt Data to Firebase

For Figure 4- 39, the `handleSaveToFirebase` function provides a detailed process for saving a scanned receipt to Firebase Firestore. The function first validates the receipt details, ensuring the amount, date, and vendor fields are correctly filled. The date is parsed and validated, ensuring it's within an acceptable range (1970–2999). If an image is captured, it is uploaded to Firebase Storage, and the download URL is stored along with the receipt data. The receipt is then saved as a transaction in Firestore, including user details for tracking.

Chapter 5 System Implementation

5.1 Hardware Setup

The hardware essential for this project includes a computer and mobile devices (Android and iOS). The computer is used for the development tasks such as coding, application design, and initial testing, employing tools like React Native and Visual Studio Code. The Android and iOS mobile devices are crucial for real-world testing, ensuring the app functions seamlessly across different platforms.

Table 5- 1 Specifications of laptop

Description	Specifications
Model	MacBook Pro (16-inch, 2019)
Processor	2.6 GHz 6-Core Intel Core i7
Operating System	macOS Sonoma
Graphic	AMD Radeon Pro 5300M 4 GB Intel UHD Graphics 630 1536 MB
Memory	16 GB 2667 MHz DDR4
Storage	512GB SSD

5.2 Software Setup

This project makes use of a variety of development tools and software to create a reliable and efficient mobile application. The tools chosen are essential to many aspects of the development process, from coding to testing and deployment.

Table 5- 2 Development Tools and Software Requirements

Tool/Software	Purpose
React Native	A JavaScript framework for building mobile applications for iOS and Android, allowing single codebase development for both platforms [22].
Firebase	A comprehensive platform for mobile and web application development, offering services like databases, authentication, and hosting [23].

Visual Studio Code	An open-source code editor supporting a wide range of programming languages and frameworks, ideal for application development [24].
Node.js	A JavaScript runtime environment for server-side programming, known for its efficiency and scalability [25].
Expo	A platform for developing and deploying universal native apps, simplifying the React Native development process [26].
Python	A versatile programming language used for developing the machine learning model and performing data analysis [27].

Following Table 5.2, the rationale behind the selection of each development tool and software is as follows:

The combination of React Native, Firebase, Visual Studio Code, Node.js, Expo, and Python is strategically chosen to align with the specific objectives of this project. React Native's cross-platform capability is essential for a unified user experience across Android and iOS devices. Firebase provides robust backend infrastructure, crucial for dynamic app features. Visual Studio Code supports complex coding requirements with extensive language support. Node.js ensures server-side efficiency for real-time updates, while Expo simplifies the app's building and deploying phases.

Python is utilized specifically to implement the Flask API for time series forecasting with the Prophet library, enabling the prediction of future financial data.

5.3 Setting and Configuration

Several key components are required for proper app setup and testing. First, a stable internet connection is crucial to ensure smooth communication with cloud services, such as Firebase. Additionally, the development and testing phases rely heavily on the iOS simulator and a physical device (either Android or iOS) for real-world testing. Expo is used to manage the app during these processes, allowing developers to test on both simulators and physical devices with ease, ensuring cross-platform functionality.

```

import React, { useContext } from "react";
import { getApps, initializeApp } from "firebase/app";
import { AuthContext } from "../provider/AuthProvider";

import { NavigationContainer } from "@react-navigation/native";

import Main from "./MainStack";
import Auth from "./AuthStack";
import Loading from "../screens/utils/Loading";

const firebaseConfig = {
  apiKey: "AIzaSyBVLQ4ctFosCTqCg4uQFv0FNYXGHIqt00s",
  authDomain: "myminiproject-4a563.firebaseio.com",
  projectId: "myminiproject-4a563",
  storageBucket: "myminiproject-4a563.appspot.com",
  messagingSenderId: "37475478601",
  appId: "1:37475478601:web:31e4f7720c4878ec406398",
};

if (getApps().length === 0) {
  initializeApp(firebaseConfig);
}

export default () => {
  const auth = useContext(AuthContext);
  const user = auth.user;
  return (
    <NavigationContainer>
      {user == null && <Loading />}
      {user == false && <Auth />}
      {user == true && <Main />}
    </NavigationContainer>
  );
};

```

Figure 5- 1 Firebase Initialization and App Configuration

Besides configuring the physical devices, setting up the Firebase environment is essential for app functionality. Figure 5- 1 illustrates the initialization of Firebase in the app. It shows how Firebase is configured with project-specific details, such as API keys and storage bucket information. This setup ensures that the app can connect to Firebase for backend services like authentication, storage, and real-time data synchronization. The code emphasizes the importance of the connection between the app and Firebase to support core features like user authentication and data storage.


```

import TransactionList from "../screens/MyMoneyManagement/TransactionList";
import TransactionDetail from "../screens/MyMoneyManagement/TransactionDetail";
import TransactionEdit from "../screens/MyMoneyManagement/TransactionEdit";

import CategoryList from "../screens/MyMoneyManagement/CategoryList";
import CategoryAdd from "../screens/MyMoneyManagement/CategoryAdd";
import ContactList from "../screens/MyMoneyManagement/ContactList";

import LoanAdd from "../screens/MyMoneyManagement/LoanAdd";
import TrackAchievements from "../screens/MyMoneyManagement/TrackAchievements";
import ManageAlerts from "../screens/MyMoneyManagement/ManageAlerts";

import AIChatbox from "../screens/MyMoneyManagement/AIChatbox";

import ReceiptOCR from "../screens/MyMoneyManagement/ReceiptOCR";
import FinancialAnalysis from "../screens/MyMoneyManagement/FinancialAnalysis"

const MainStack = createNativeStackNavigator();
const Main = () => {
  return (
    <MainStack.Navigator
      screenOptions={{
        headerShown: false,
      }}
    >
      <MainStack.Screen name="MainTabs" component={MainTabs} />
      <MainStack.Screen name="About" component={About} />
      <MainStack.Screen name="CreateWallet" component={CreateWallet} />
      <MainStack.Screen name="MyMenu" component={MyMenu} />
      <MainStack.Screen name="TransactionAdd" component={TransactionAdd} />
      <MainStack.Screen name="MyLocation" component={MyLocation} />
      <MainStack.Screen name="Home" component={Home} />
      <MainStack.Screen name="Report" component={Report} />
      <MainStack.Screen name="ReportDetail" component={ReportDetail} />
      <MainStack.Screen name="UpdateGoals" component={UpdateGoals} />
      <MainStack.Screen name="Budget" component={Budget} />
      <MainStack.Screen name="BudgetAdd" component={BudgetAdd} />
      <MainStack.Screen name="BudgetDetail" component={BudgetDetail} />
      <MainStack.Screen name="Account" component={Account} />
      <MainStack.Screen name="TransactionList" component={TransactionList} />
      <MainStack.Screen
        name="TransactionDetail"
        component={TransactionDetail}
      />
      <MainStack.Screen name="TransactionEdit" component={TransactionEdit} />
      <MainStack.Screen name="CategoryList" component={CategoryList} />
      <MainStack.Screen name="CategoryAdd" component={CategoryAdd} />
      <MainStack.Screen name="ContactList" component={ContactList} />
      <MainStack.Screen name="LoanAdd" component={LoanAdd} />
      <MainStack.Screen
        name="TrackAchievements"
        component={TrackAchievements}
      />
      <MainStack.Screen name="ManageAlerts" component={ManageAlerts} />
      <MainStack.Screen name="AIChatbox" component={AIChatbox} />
      <MainStack.Screen name="ReceiptOCR" component={ReceiptOCR} />
      <MainStack.Screen name="FinancialAnalysis" component={FinancialAnalysis} />
    </MainStack.Navigator>
  );
};

export default Main;

```

Figure 5- 2 MainStack Navigation Setup

After describing Firebase initialization in Figure 5- 1, it's important to highlight how the app's navigation is configured. Figure 5- 2 shows the navigation structure of the app, defining various screens such as AIChatbox, FinancialAnalysis, and ReceiptOCR. This setup ensures smooth transitions between the app's features, allowing users to easily access different functionalities. The MainStack navigator is essential for organizing the app's interface and ensuring a seamless user experience across multiple screens.

```

export type MainStackParamList = {
  MainTabs: { userId: string };
  MyMenu: { userId: string };
  TransactionAdd:
  | {
    category?: string;
    type?: "income" | "expense";
    userId: string;
  }
  | undefined;
  CreateWallet: undefined;
  Home: undefined;
  Report: {
    currency: string;
    totalSpentThisMonth: number;
    totalIncomeThisMonth: number;
    totalSpentLastMonth: number;
    totalWalletValue: number;
    financialGoal?: string;
    targetSavings?: number;
    currentSavings?: number;
  };
  ReportDetail: {
    netCashFlow?: number;
    totalIncome?: number;
    totalExpenses?: number;
    totalDebtPayments?: number;
    financialGoal?: string;
    targetSavings?: number;
    currentSavings?: number;
    currency: string;
  };
  UpdateGoals: undefined;
  MyLocation:
  | {
    onLocationSelect: (address: string) => void;
  }
  | undefined;
};

```

Figure 5- 3 Type Definitions for Navigation Parameters

Figure 5- 3 demonstrates how navigation types, such as `MainStackParamList`, define the parameters for each screen. This setup is important for ensuring that the correct data, such as user IDs or transaction details, is passed between different pages, enhancing the app's stability by preventing type-related errors. Together, these navigation configurations play a key role in managing and maintaining the app's functionality.

5.4 System Operation (with Screenshot)

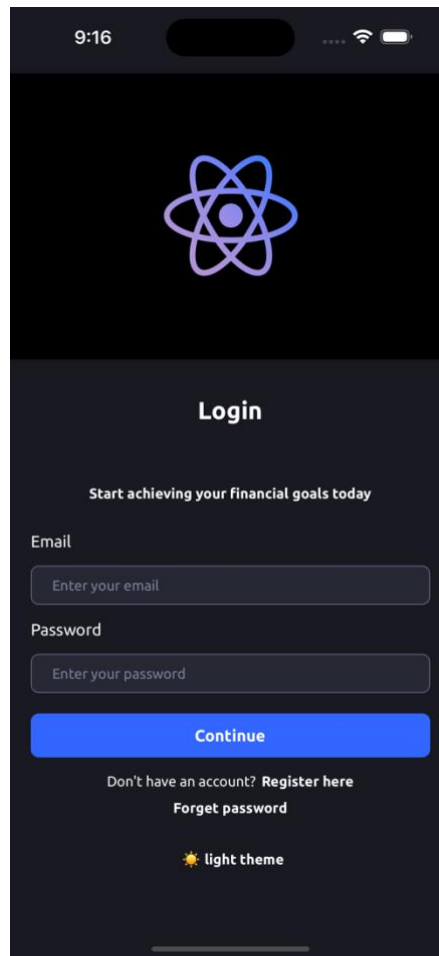


Figure 5- 4 Login

Figure 5- 4 showcases the login screen of the app with a dark theme, including fields for email and password and options to register or recover a password. The interface allows theme customization for user preference.

9:47

Register

Display Name
Enter your name

Email
Enter your email

Password
Enter your password

Confirm Password
Enter your confirm password

Gender
Select gender

Birth Date
09/10/20

Create an account

Already have an account? [Login here](#)

Figure 5- 5 Register

Figure 5- 5 presents the registration interface for users to create an account within the application. It features input fields for essential details such as display name, email, password, gender, and birth date. A dark mode option is available, and additional navigation links are provided for convenience.

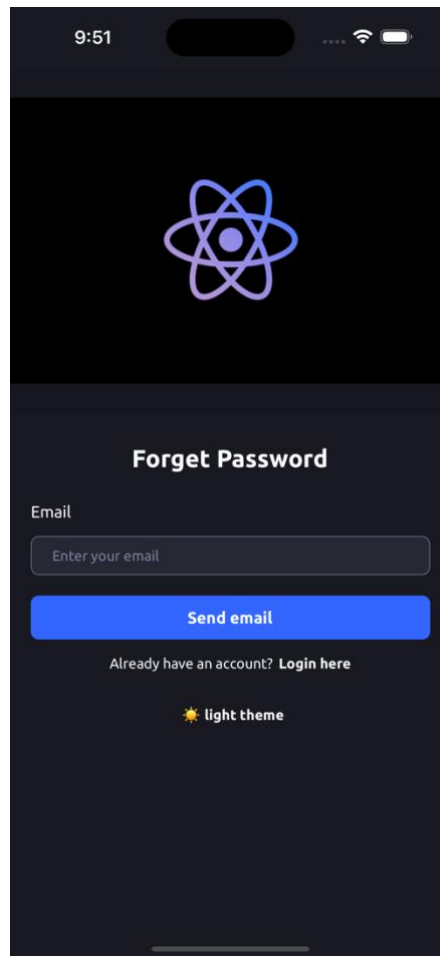


Figure 5- 6 Forgot Password

Figure 5- 6 presents a straightforward interface where users can request a password reset. The screen features a singular text input for the user's email address and a button to submit the request. Once the email is submitted, the app navigates back to the login screen and alerts the user that a password reset email has been sent. The code implements the `sendPasswordResetEmail` function from Firebase Authentication to handle the password reset process. The dark mode theme is an option, indicated by the toggle at the bottom of the screen.

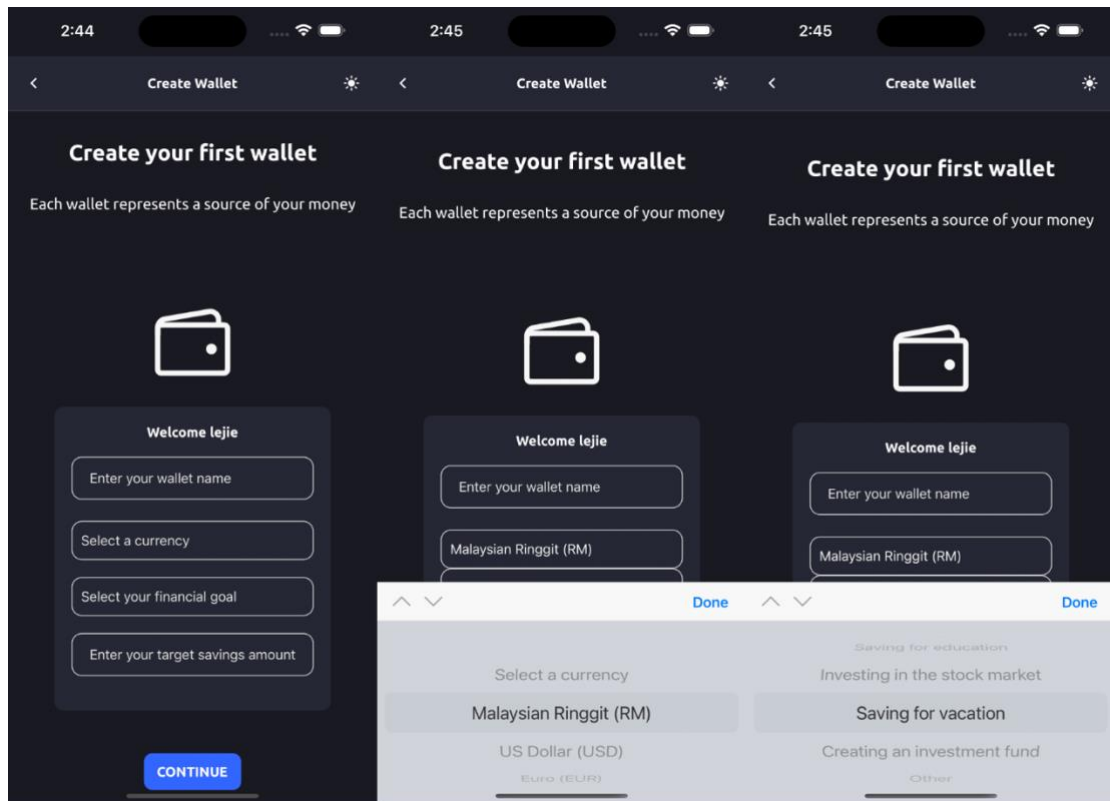


Figure 5- 7 Create Wallet

Figure 5- 7 illustrate the process for creating a new wallet, which would be the user's initial step after logging in for the first time or when they have not set up any wallets. The steps involve naming the wallet, selecting a currency, setting a financial goal and target savings amount.

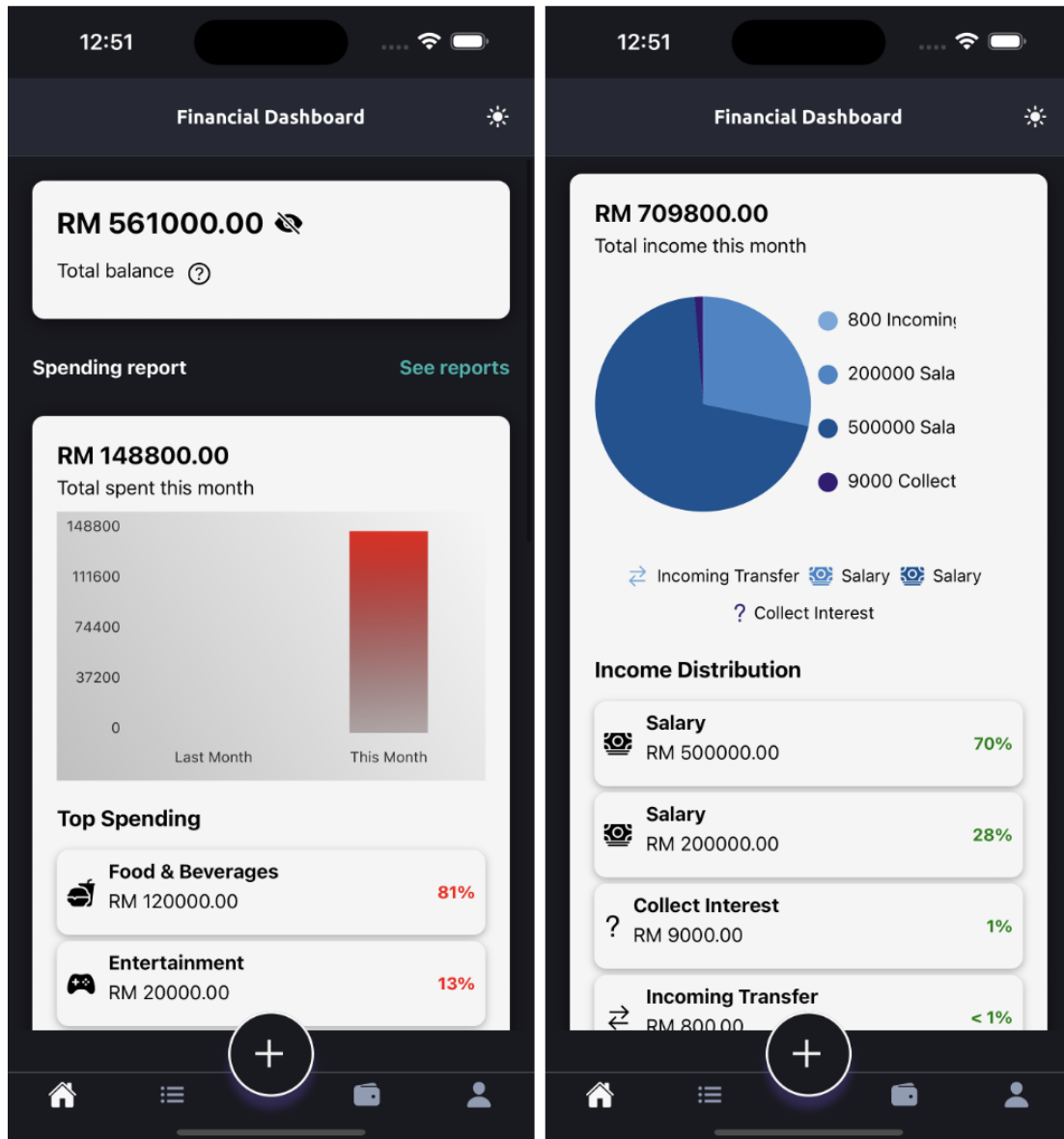


Figure 5- 8 Financial Dashboard

Figure 5- 8 shows the Dashboard screen of the app with a dark theme interface. It displays financial information including the total balance, spending report with a bar chart comparing last month's and this month's expenses, top spending categories, and a pie chart for income distribution. Besides, the bottom navigation bar enables swift movement through the app's main functions: dashboard, transactions list, budgeting, and account management, with the central button facilitating easy transaction entries.

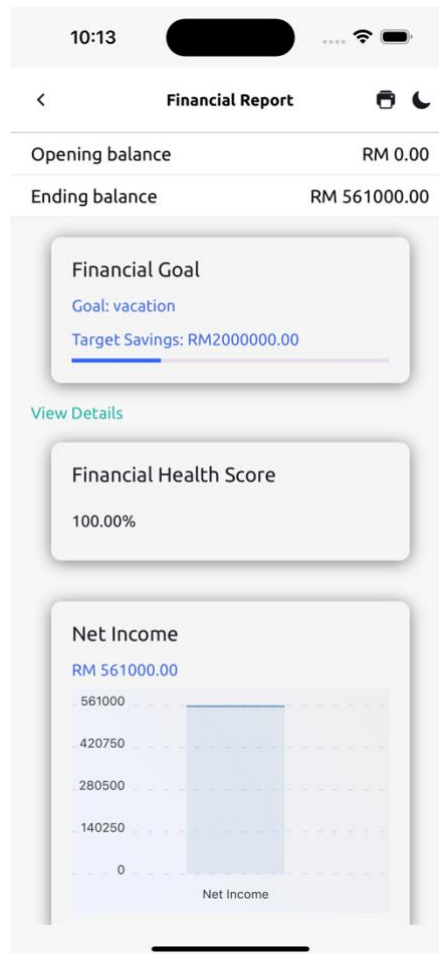


Figure 5- 9 Financial Report

Figure 5- 9 illustrates a financial report page, accessible by clicking the 'See reports' option from the Financial Dashboard. The screen is segmented into several sections providing a snapshot of the user's financial status, including opening and closing balances, financial goals with progress indicators, financial health scores depicted in percentages, and a bar chart reflecting net income. This informative layout is engineered to offer a snapshot of the user's financial health and progress toward set goals.

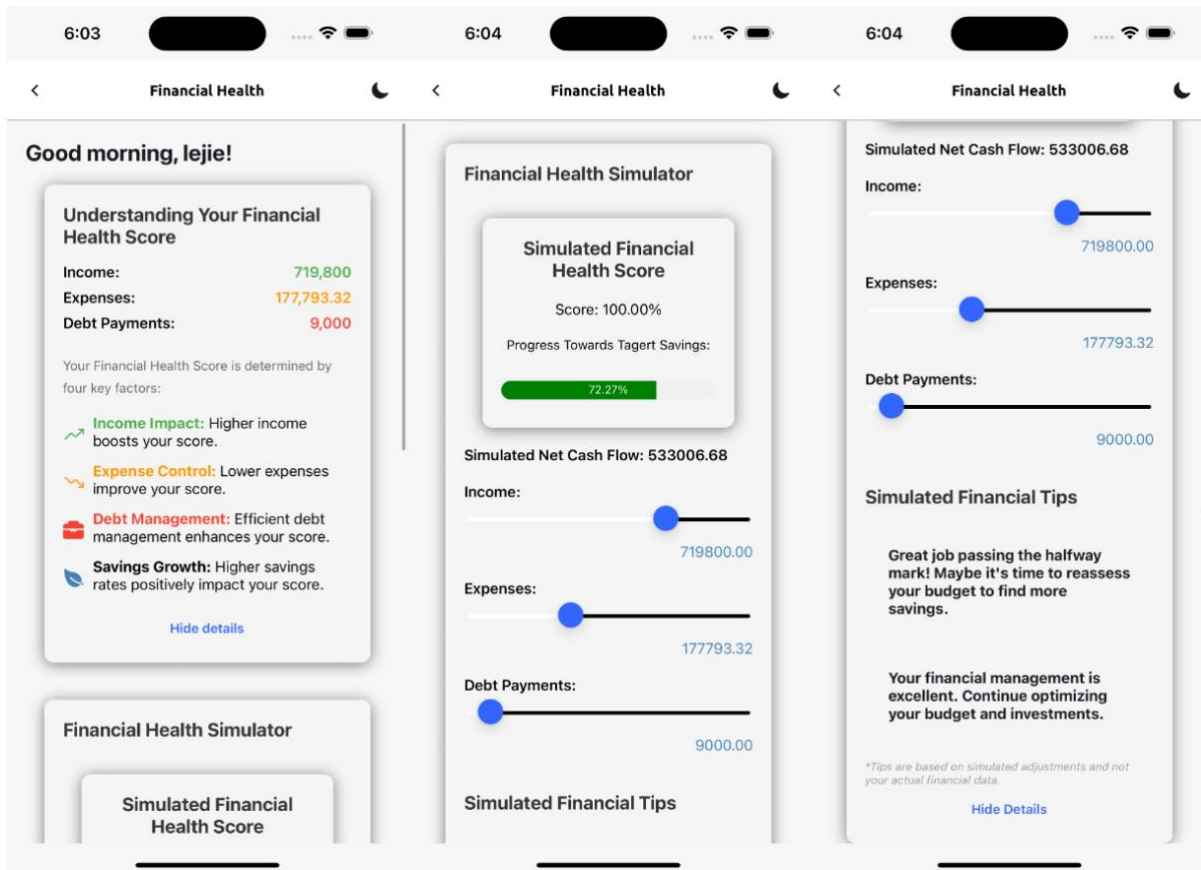


Figure 5- 10 Financial Health

Figure 5- 10 displays the Financial Health page, which users can access from the Financial Dashboard by clicking 'View Details'. This screen provides a detailed breakdown of the user's income, expenses, and debt payments, along with an automatically calculated Financial Health Score. Additionally, it includes a Financial Health Simulator where users can adjust financial variables using sliders to see their potential impact on their overall score. The page also offers personalized financial tips based on the user's performance and progress toward set savings goals."

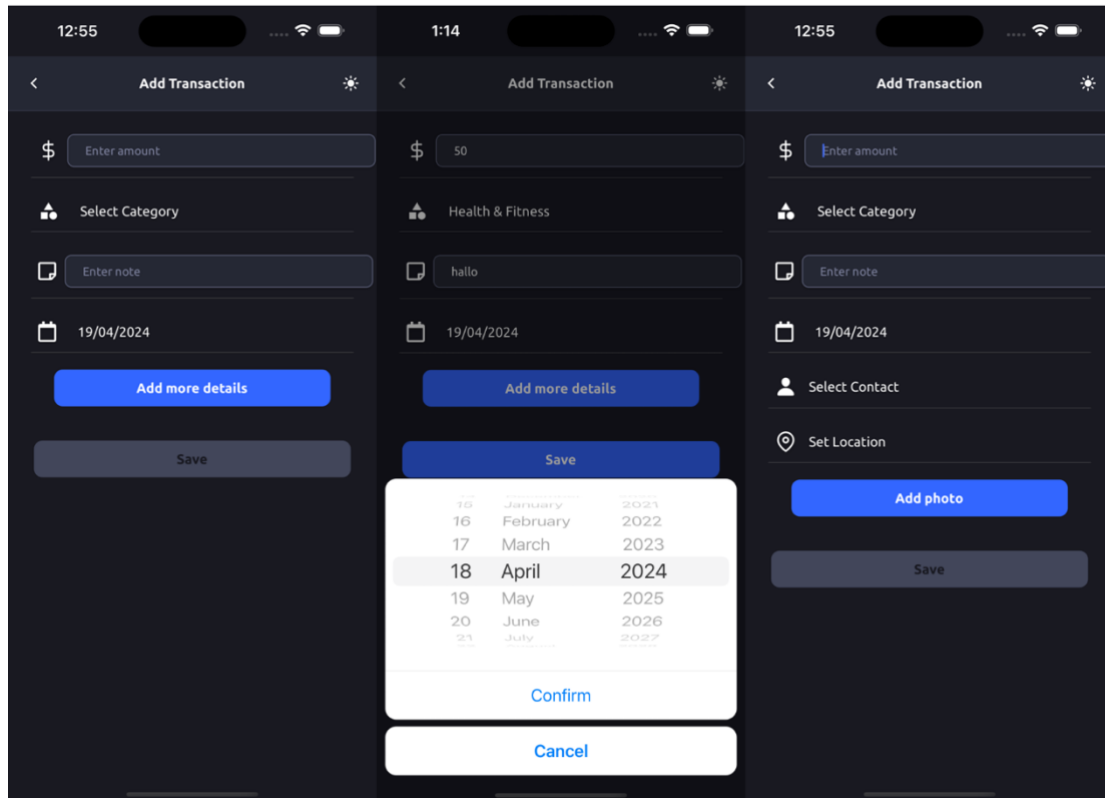


Figure 5- 11 Add Transaction

Figure 5- 11 showcases the 'Add Transaction' functionality within the app, demonstrating a streamlined process for users to record their financial activities. The first and second showcases the initial details to be inputted for a transaction, including amount, category, and a note, along with a date selector. The third expands on the initial detail entry with optional fields to personalize the transaction record, such as adding an associated contact, setting a location, and attaching a photo, enhancing the user's ability to record the context of their transactions.

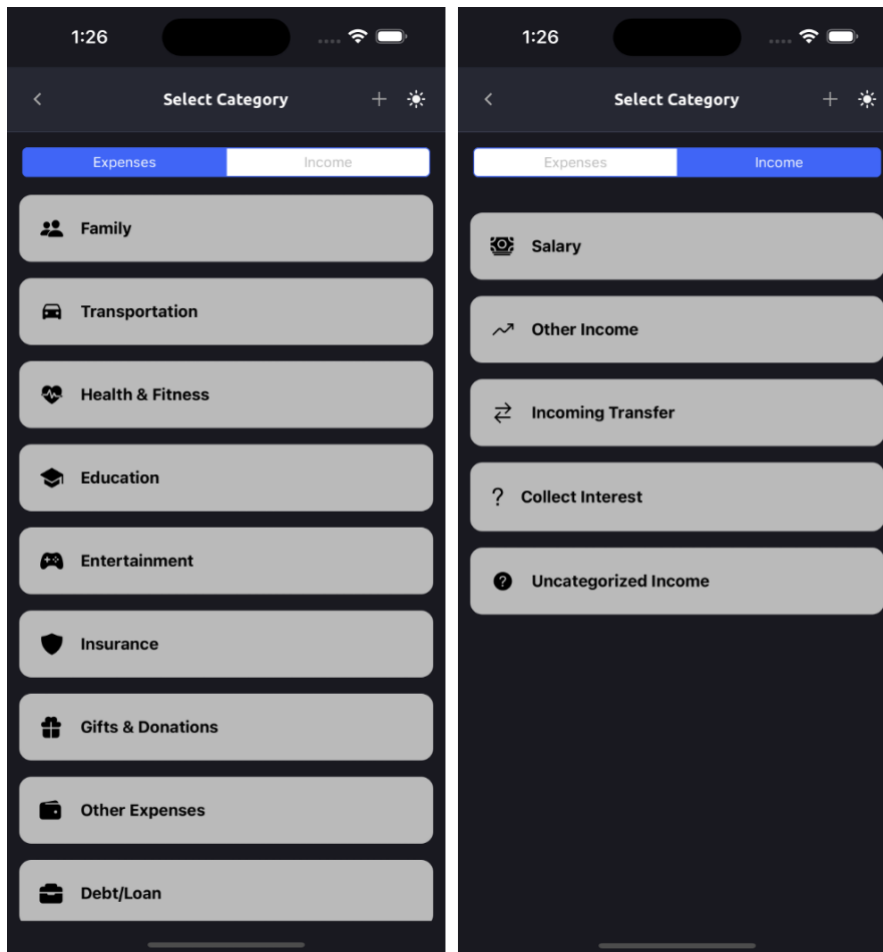


Figure 5- 12 Add Transaction - Select Category

Figure 5- 12 exhibits a user interface for selecting a financial category within the 'Add Transaction' process. The interface is neatly divided into two tabs: 'Expenses' and 'Income', allowing users to categorize their transaction with precision. Each tab unveils a list of predefined categories such as 'Food & Beverages', 'Bill & Utilities' for expenses, or 'Salary', 'Collect Interest' for income, to be selected to best match the nature of the transaction being recorded.

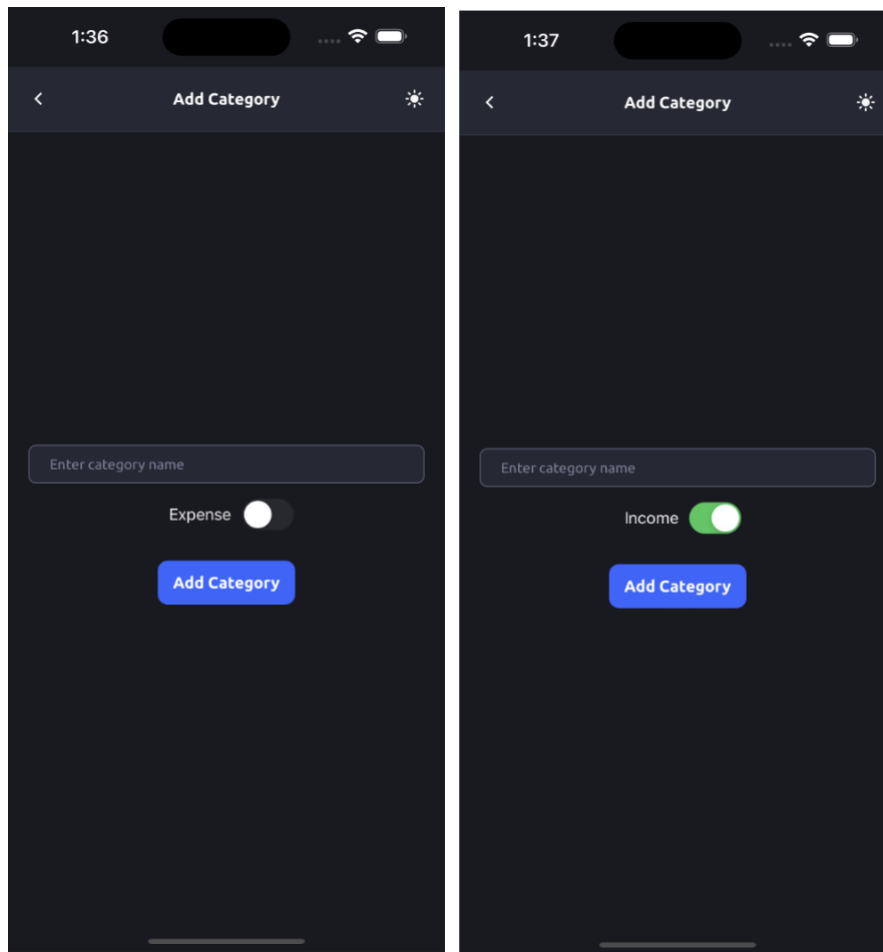


Figure 5- 13 Add Transaction - Select Category - Add Category

Figure 5- 13 illustrates the user interface for adding a new category. Users can personalize their category list by entering the name of the new category and toggling between 'Expense' and 'Income' to classify it correctly.

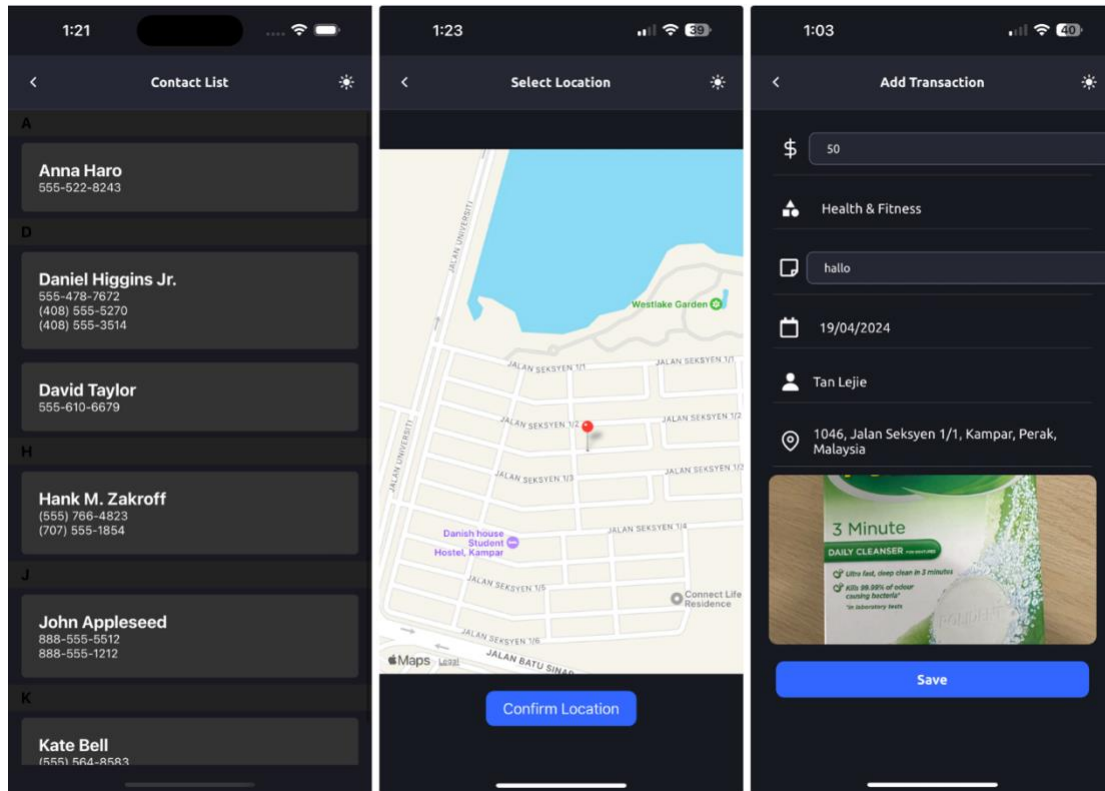


Figure 5- 14 Add Transaction - Additional Fields

Figure 5- 14 encapsulates the additional fields in the Add Transaction process of an application, enabling users to enrich transaction records with personalized details. On selecting 'Add more details' within the transaction addition interface, users are guided to a contact list where they can associate the transaction with an individual from their contacts. They also have the option to set a transaction-specific location through an interactive map interface, as well as attach a photo for additional context, thus providing a comprehensive and detailed transaction logging experience.

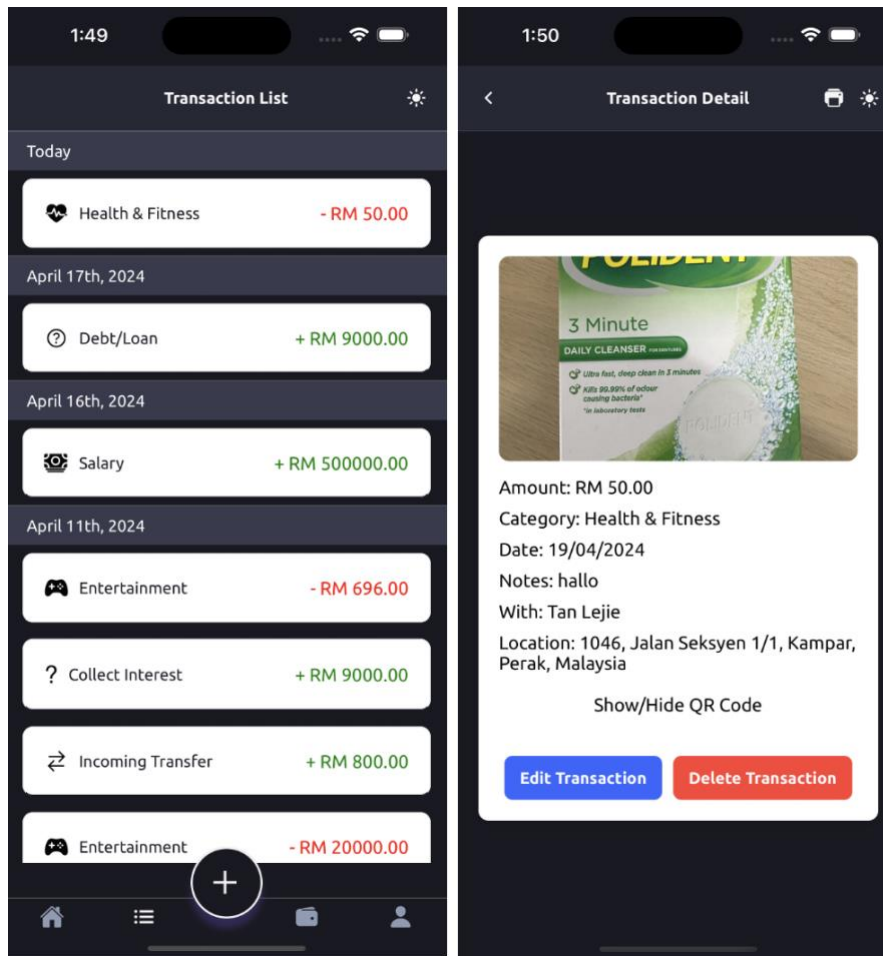


Figure 5- 15 Transaction List - Transaction Detail

Figure 5- 15 illustrates two main interfaces within the application: the Transaction List and the Transaction Detail. The Transaction List provides a comprehensive view of a user's recent financial activities, displaying each item with a corresponding icon, the transaction category, the date, and the amount spent or received. It facilitates a quick overview and efficient tracking of personal finances.

The Transaction Detail screen delves into the specifics of an individual transaction when selected from the list. It showcases detailed information such as the amount, category, date, and additional notes. It also displays a photo associated with the transaction, enhancing the visual context. Users can interact with this screen to edit or delete transactions. This detailed view ensures users have full visibility into their spending and can manage their records accurately.

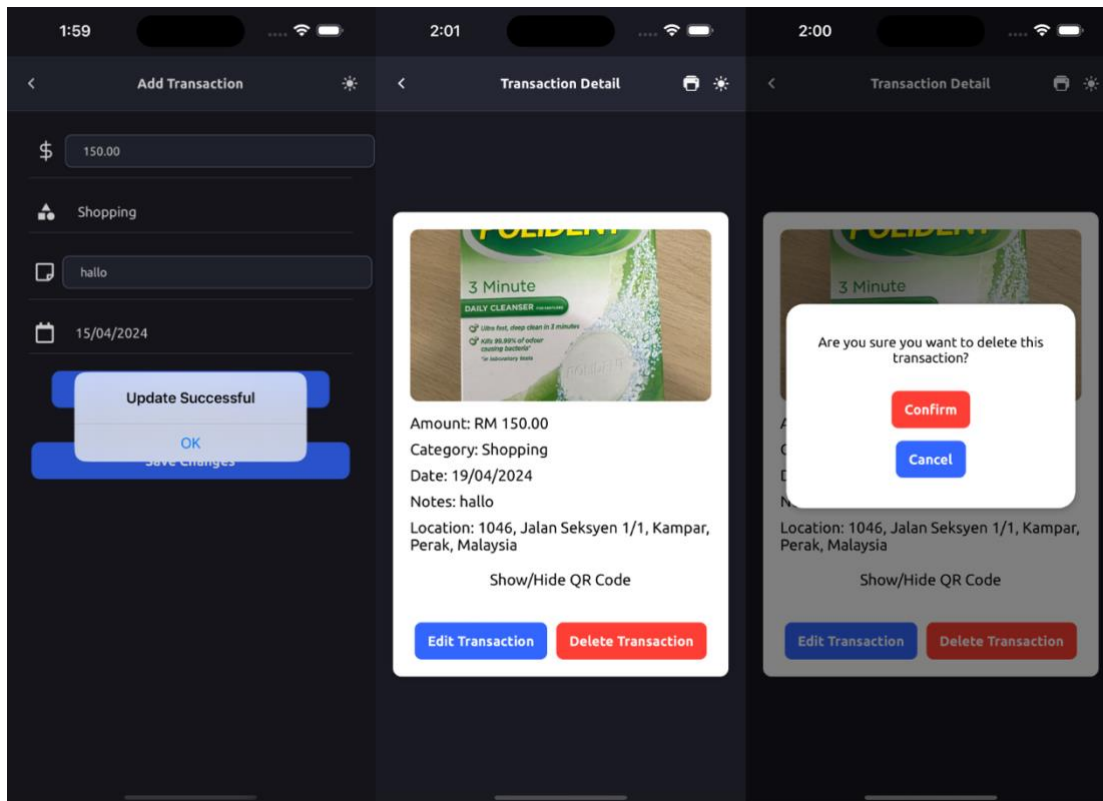


Figure 5- 16 Transaction Detail - Transaction Edit and Delete

Figure 5- 16 demonstrates how users can edit their transaction details. Additionally, the figure shows a deletion process, where a confirmation dialogue box appears to ensure the user wants to proceed with removing the transaction from the record, prevent errors by confirming actions that cannot be undone, such as this transaction deletion.

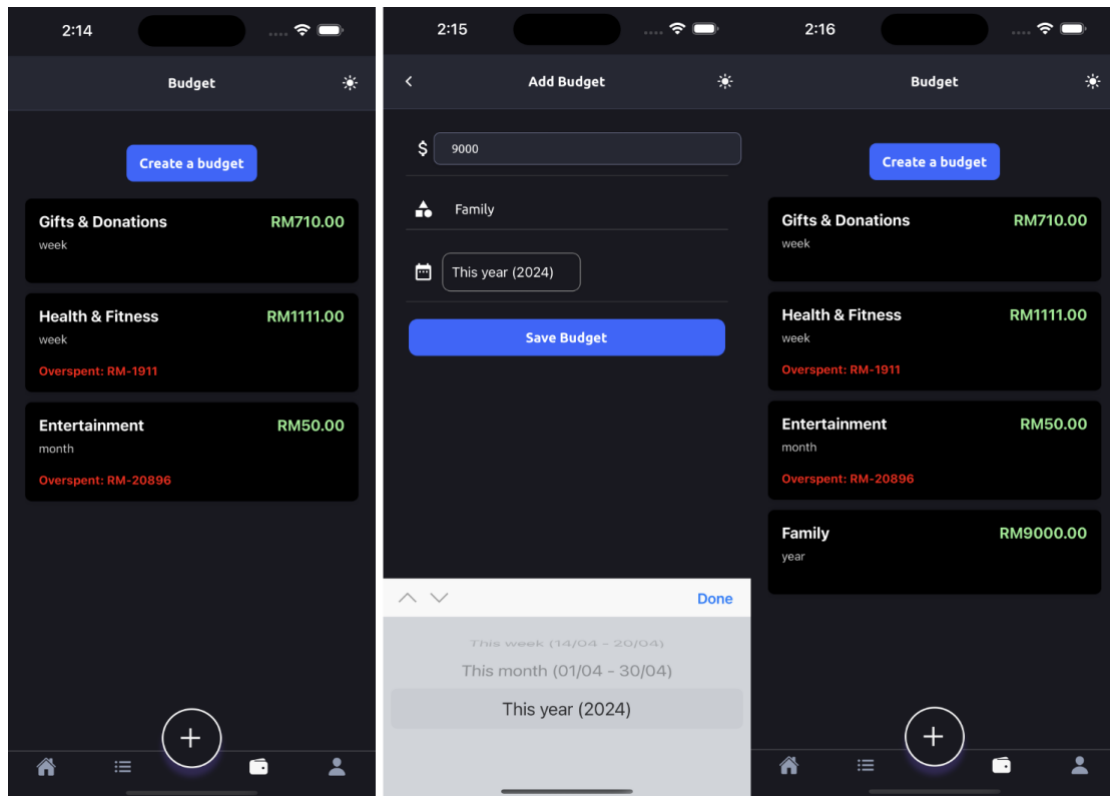


Figure 5- 17 Add Budget

Figure 5- 17 illustrates the process of creating a budget. Users can see their budget list, which displays categories alongside the allocated budget amounts, and they have the option to create a new budget. This process includes selecting a category, setting an amount, and defining a time range for the budget. This feature is crucial for users to plan and monitor their spending within set financial constraints.

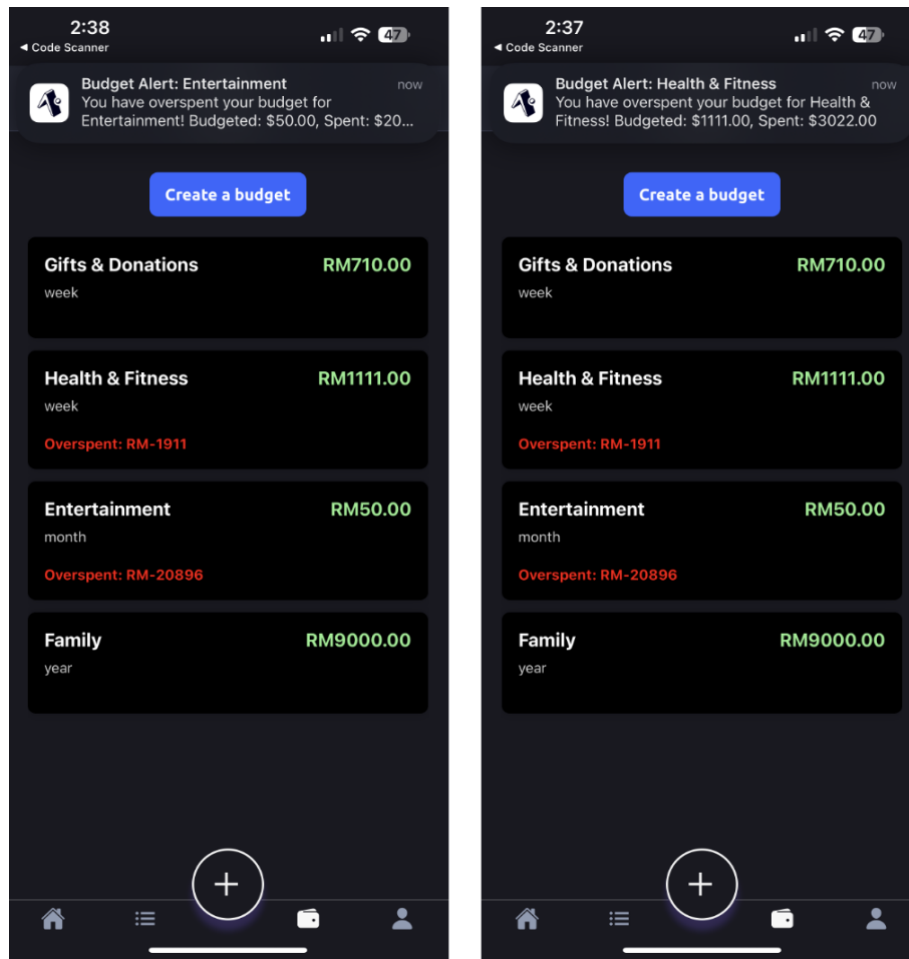


Figure 5- 18 Notifications Alert

In Figure 5- 18, when a user exceeds their budget limit, the system is designed to send immediate notifications. These alerts inform the user of the category where the overspending occurred, the budgeted amount versus the actual spending, underscoring the need for financial adjustments.

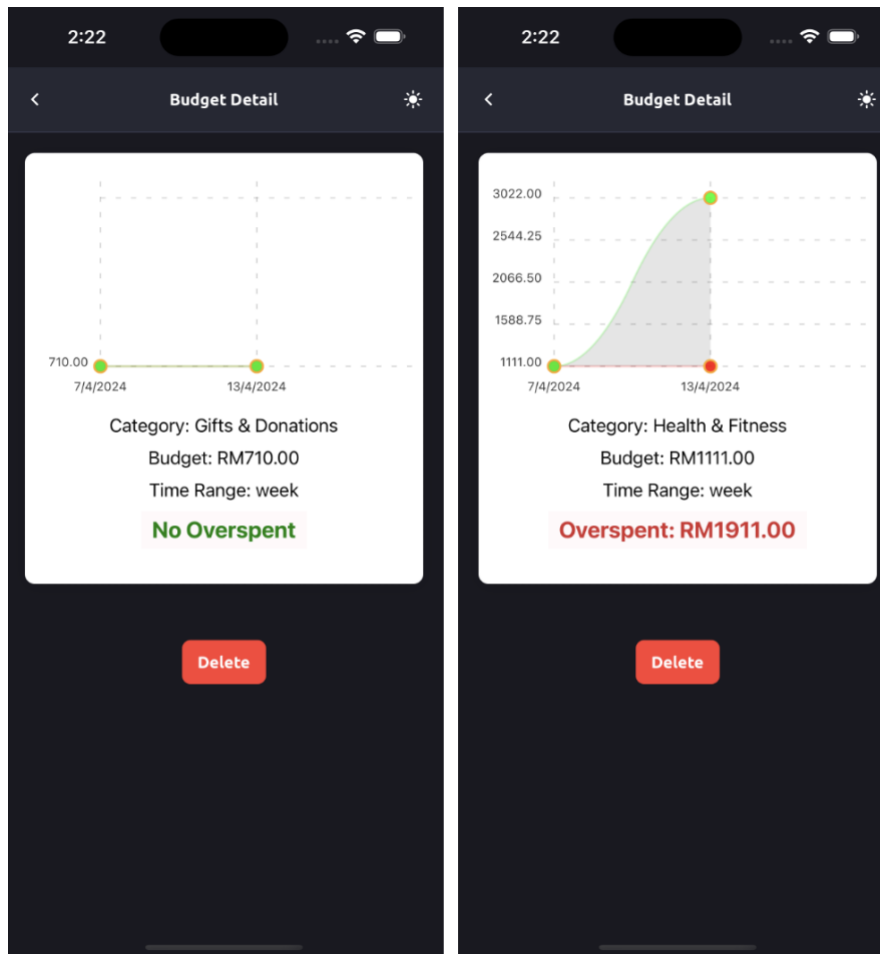


Figure 5- 19 Budget Detail

Figure 5- 19 presents a comprehensive view of a user's budgetary constraints and spending, including visuals like charts to represent financial data over time. Users can assess their budgeting for specific categories and time ranges, with indications of any overspending. If overspending occurs, it's clearly displayed, allowing users to adjust their spending habits accordingly. A button is available to delete the budget if needed, facilitating easy management of the user's financial plan.

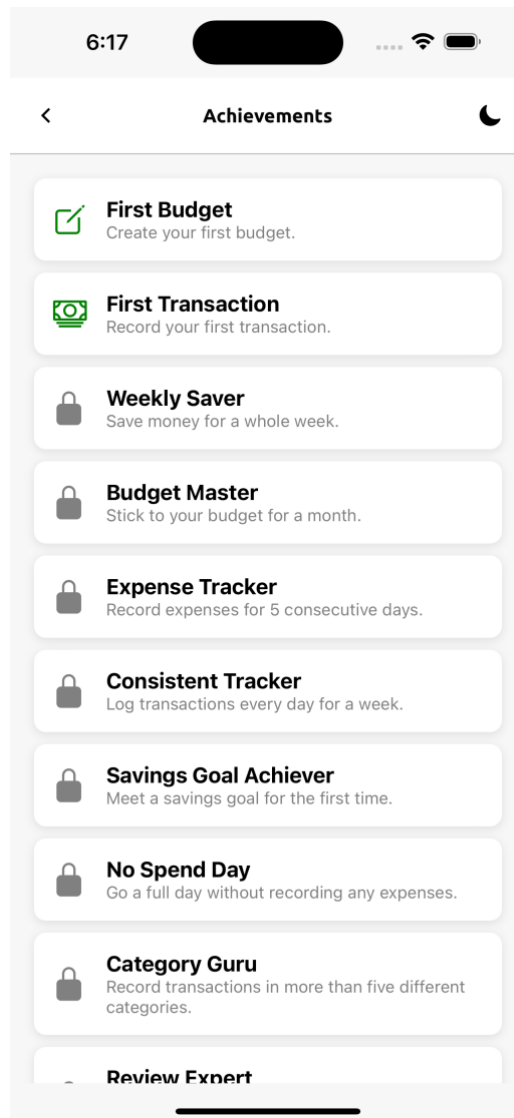


Figure 5- 20 Track Achievements

Figure 5- 20 highlights the 'Track Achievements' feature, showcasing the various milestones users can unlock based on their financial behavior. The achievements are categorized by actions like creating a budget, recording transactions, and saving money consistently. Each achievement is displayed with an icon and description, with locked achievements shown in grey and unlocked ones in green. This feature helps users stay motivated by recognizing their progress and encouraging better financial habits through gamification elements like the 'First Budget' and 'Savings Goal Achiever' badges.

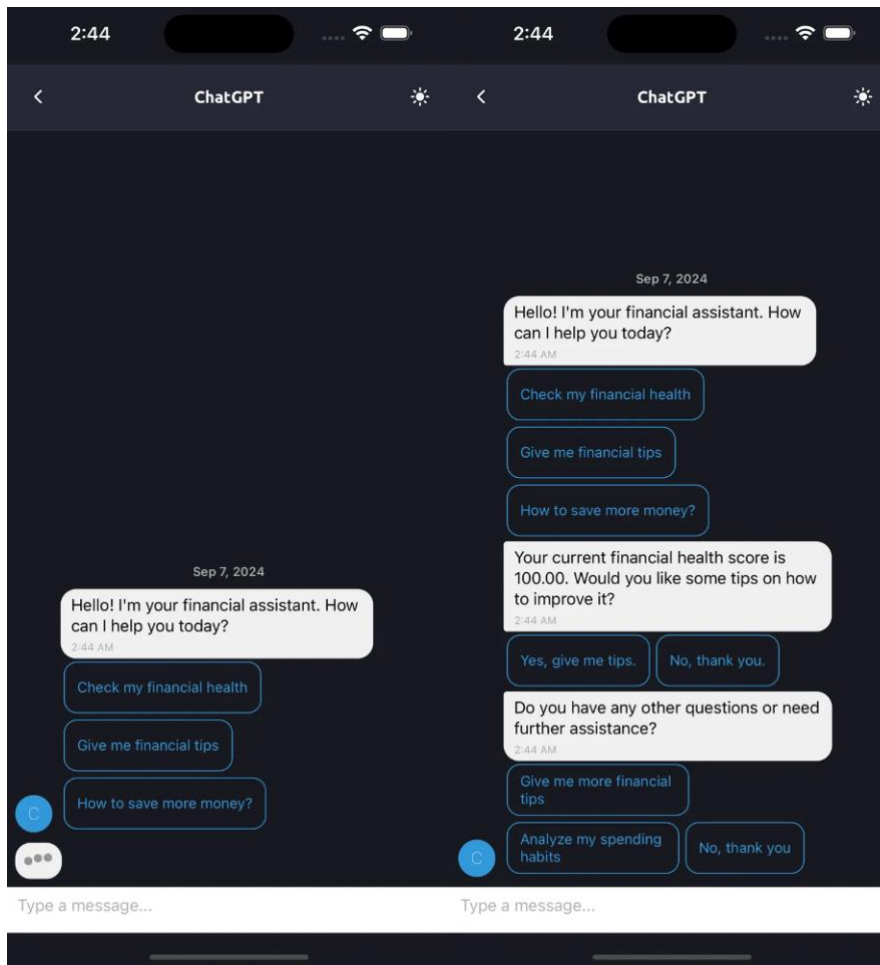


Figure 5- 21 Chatbox with Predefined Options for Financial Assistance

Figure 5- 21 highlights the use of predefined buttons within the chatbox interface, enabling users to quickly select from options like “Check my financial health,” “Give me financial tips,” or “How to save more money?” These predefined responses streamline user interactions by providing fast access to relevant financial assistance queries without needing manual input.

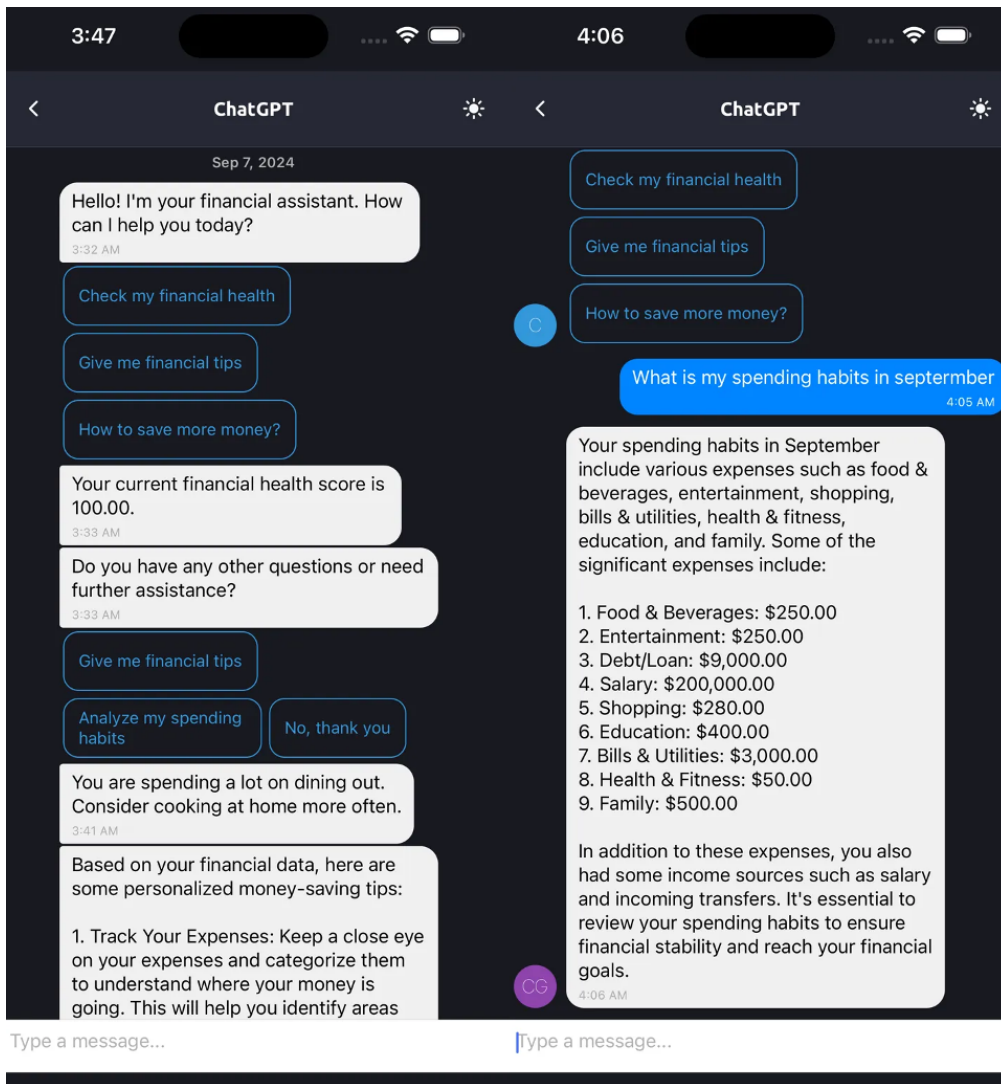


Figure 5- 22 Retrieving Financial Data from Firebase in Chatbox

Figure 5- 22 showcases the chatbox retrieving and displaying real-time financial data from Firebase. When a user requests their spending habits for September, the system fetches categorized transaction data (e.g., food, entertainment, loans) and provides a detailed analysis based on their financial activity for the month. This feature enhances the chatbox’s ability to offer personalized financial insights.

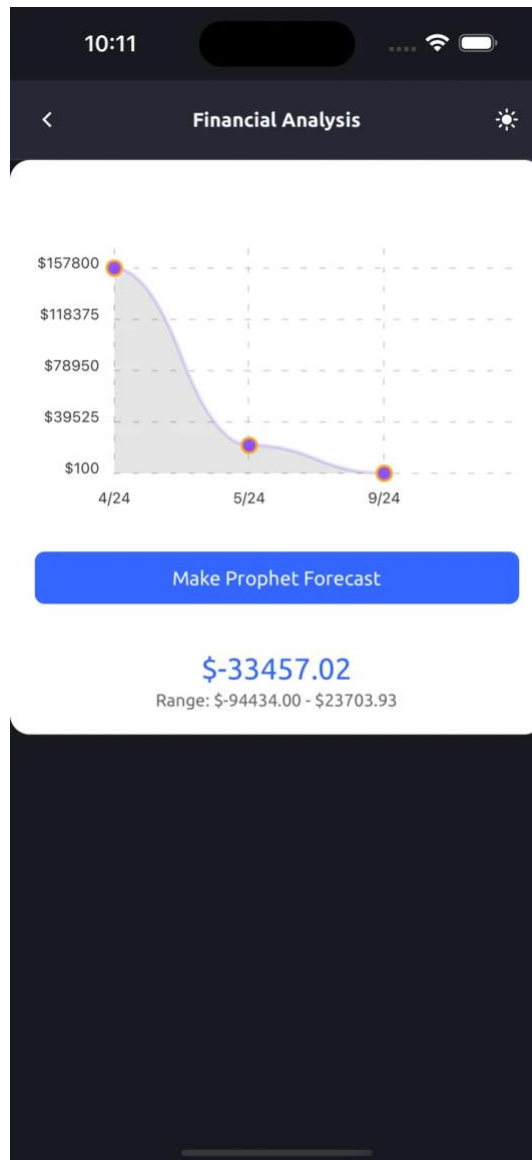


Figure 5- 23 Facebook Prophet Forecasting in Financial Analysis

Figure 5- 23 illustrates the integration of Facebook Prophet into the Financial Analysis feature of the app. Users can select specific date ranges and categories to visualize their financial data over time. In the graph shown, the y-axis displays the amounts, and the x-axis represents the dates. The line graph includes a forecast based on the Prophet model, showing expected financial trends with a shaded uncertainty range. A “Make Prophet Forecast” button allows users to trigger a new forecast based on recent transaction data. Below the chart, the forecasted amount is displayed, providing insights into financial predictions.

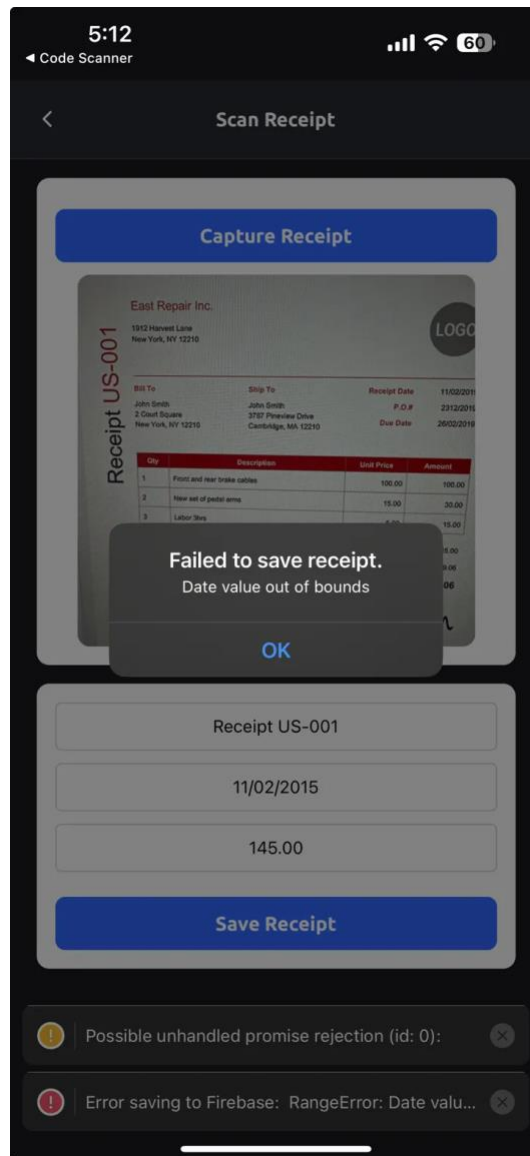


Figure 5- 24 Receipt OCR

Figure 5-24 shows the Receipt OCR feature of the application, where users can capture an image of a receipt using their device's camera. Once the image is captured, it is processed using OCR (Optical Character Recognition) technology to extract key details such as vendor, date, and amount. The figure demonstrates a successful capture of a receipt image, with extracted details shown below the image. Users can manually edit these details before saving them to Firebase for record-keeping, enhancing ease of financial tracking.

5.5 Implementation Issues and Challenges

The implementation of the Receipt OCR feature in the personal finance management app presented several significant challenges. One of the major issues was app instability when saving receipts, which primarily arose from problems with date formatting and validation. Extracted date values frequently fell outside Firestore's acceptable range, leading to crashes (Figure 5- 24). To resolve this, the date parsing logic was overhauled, and rigorous validation checks were introduced. Additionally, extracting useful data from receipts using the Google Cloud Vision API demanded complex parsing algorithms to accurately capture relevant information from diverse receipt formats. This required significant effort to refine, as receipt structures varied widely across different vendors.

The process of capturing, uploading, and storing receipt images in Firebase Storage while maintaining proper linkage to transaction data added another layer of complexity to the feature's development. This required a balance between frontend user interaction and robust backend data handling, particularly in maintaining smooth third-party API integrations for the mobile finance application. Proper structuring of the data within Firestore was also essential for ensuring consistency between newly scanned receipts and existing transaction records.


```

non-std C++ exception

RCTFatal
__28-[RCTCxxBridge handleError:]_block_invoke
_dispatch_call_block_and_release
_dispatch_client_callout
_dispatch_main_queue_drain
_dispatch_main_queue_callback_4CF
__CFRunLoop_IS_SERVICING_THE_MAIN_DISPATCH_QUEUE
__CFRunLoopRun
CFRunLoopRunSpecific
GSEventRunModal
-[UIApplication _run]
UIApplicationMain
main
start_sim

Unable to resolve module react-native-fs
from /Users/lejie/Downloads/
MyMoneyManagementOwn/node_modules/
@tensorflow/tfjs-react-native/dist/
bundle_resource_io.js: react-native-fs could
not be found within the project or in these
directories:
  node_modules/@tensorflow/tfjs-react-native/
  node_modules
  node_modules
  77 | // never hit this code path.
  78 | // tslint:disable-next-line:
no-require-imports
> 79 |   const RNFS = require('react-
native-fs');
      |               ^
  80 |   const modelJson =
this.modelJson;
  81 |   const modelArtifacts =
Object.assign({}, modelJson);
  82 |   modelArtifacts.weightSpecs =
modelJson.weightsManifest[0].weights;

RCTFatal
__28-[RCTCxxBridge handleError:]_block_invoke
_dispatch_call_block_and_release
_dispatch_client_callout
_dispatch_main_queue_drain
_dispatch_main_queue_callback_4CF
__CFRunLoop_IS_SERVICING_THE_MAIN_DISPATCH_QUEUE
__CFRunLoopRun

```

Figure 5- 25 TensorFlow.js Integration Failure and Non-std C++ Exception

Figure 5-25 demonstrates the error encountered while attempting to integrate TensorFlow.js into the app. The error includes a non-std C++ exception and a react-native-fs module issue, both of which led to the entire application crashing. This required a rollback to a stable version of the app, highlighting the severe impact such errors can have on development. The crash significantly delayed the project and emphasized the fragile nature of the environment. As a result, TensorFlow.js was replaced with Facebook Prophet for financial forecasting. Prophet's seamless integration proved to be a more stable and effective solution for achieving the forecasting objectives outlined in FYP 1.

5.6 Concluding Remark

The development of this project, which began on the 6th of March 2024 and concluded on the 13th of September 2024, presented numerous technical challenges but also offered significant learning opportunities. The core features, such as the AIChatbox, Receipt OCR, and predictive

analytics, were designed to enhance personal financial management by automating processes and providing insightful financial forecasts.

One of the key components, the Receipt OCR, involved integrating the Google Cloud Vision API for extracting data from receipts. This process encountered several challenges, particularly with date formatting and validation issues, which led to crashes during receipt storage in Firebase. To resolve these, the date parsing logic was revised, and stricter validation was implemented, ensuring better stability. Additionally, parsing algorithms were developed to manage diverse receipt formats, which required ongoing refinement.

Initially, TensorFlow.js was chosen for predictive analytics and financial forecasting as proposed in FYP 1. However, this integration encountered significant challenges, including module conflicts and issues with react-native-fs, as illustrated by the non-std C++ exceptions (Figure 5-25). These errors not only caused the app to crash but also delayed development and forced a rollback to previous stable versions. Due to the complexity of resolving these issues, TensorFlow.js was replaced with Facebook Prophet, which proved to be a more stable solution for time series forecasting. This switch enabled the app to meet its objectives for providing reliable financial predictions based on users' transaction data.

The project also incorporated the OpenAI API within the AIChatbox to deliver personalized financial advice and forecasting features. This integration allowed the app to offer users real-time insights into their financial health, along with actionable tips on saving money and analyzing spending patterns.

In conclusion, despite the technical challenges encountered, particularly with third-party APIs and machine learning model integration, the project successfully achieved its goal of enhancing personal financial management through automation and predictive analytics.

Chapter 6 System Evaluation And Discussion

6.1 System Testing and Performance Metrics

6.1.1 Login Module

Test Case	Input	Expected Output	Actual Output	Result
1. User enters valid email and password	Valid email (e.g., "user@example.com") and correct password	Navigate to the "About" page with the user ID	Navigate to the "About" page with the user ID	Pass
2. User enters invalid email or password	Invalid email or incorrect password	Display alert message with error	Display alert message with error	Pass
3. User clicks "Register here"	Clicks on "Register here" link	Navigate to the "Register" page	Navigate to the "Register" page	Pass
4. User clicks "Forget password"	Clicks on "Forget password" link	Navigate to the "ForgetPassword" page	Navigate to the "ForgetPassword" page	Pass

Figure 6- 1 Login Module

6.1.2 Register Module

Test Case	Input	Expected Output	Actual Output	Result
1. User enters valid email, password, confirm password, and other fields	Valid name, email, password, gender, and birth date	Register the user and navigate to "Login"	Register the user and navigate to "Login"	Pass
2. User enters mismatched passwords	Valid email and mismatched passwords	Display alert "Passwords do not match"	Display alert "Passwords do not match"	Pass
3. User omits required field	Missing any required field (e.g., email, password)	Display alert with corresponding missing field	Display alert with corresponding missing field	Pass

Figure 6- 2 Register Module

6.1.3 Forget Password Module

Test Case	Input	Expected Output	Actual Output	Result
1. User enters valid email	Valid email address (e.g., "user@example.com")	Display success alert and navigate to "Login"	Display success alert and navigate to "Login"	Pass
2. User enters invalid email	Invalid email format (e.g., "invalid.com")	Display error message	Display error message	Pass

Figure 6- 3 Forget Password Module

6.1.4 Add Transaction Module

Test Case	Input	Expected Output	Actual Output	Result
1. User adds a transaction with valid details	Enter valid amount, category, date, and notes	Transaction is saved and added to the transaction list	Transaction is saved and added to the transaction list	Pass
2. User adds a transaction without selecting a category	Leave category empty and submit	Show error message: "Please fill in all required fields."	Show error message: "Please fill in all required fields."	Pass
3. User adds an image for the transaction	Select an image from the gallery	Image is uploaded and saved with the transaction	Image is uploaded and saved with the transaction	Pass
4. User selects a contact for the transaction	Choose a contact from the contact list	Contact is saved with the transaction	Contact is saved with the transaction	Pass

Figure 6- 4 Add Transaction Module

6.1.5 View Transaction List Module

Test Case	Input	Expected Output	Actual Output	Result
1. View transactions list	Navigate to the transactions list screen	Display all transactions grouped by date	Display all transactions grouped by date	Pass
2. No transactions available	No transactions exist for the user	Display "No transactions" message	Display "No transactions" message	Pass
3. Select transaction	Click on a transaction from the list	Navigate to the transaction detail screen	Navigate to the transaction detail screen	Pass

Figure 6- 5 View Transaction List Module

6.1.6 View Transaction Detail Module

Test Case	Input	Expected Output	Actual Output	Result
1. View transaction details	Open a specific transaction from the list	Display transaction details with the correct amount, category, date, and notes	Display transaction details with the correct amount, category, date, and notes	Pass
2. Edit transaction	Click on the "Edit Transaction" button	Navigate to the edit screen for the selected transaction	Navigate to the edit screen for the selected transaction	Pass
3. Delete transaction	Click on the "Delete Transaction" button and confirm	Transaction is deleted and removed from the list	Transaction is deleted and removed from the list	Pass

Figure 6- 6 View Transaction Detail Module

6.1.7 Edit Transaction Module

Test Case	Input	Expected Output	Actual Output	Result
1. Edit transaction details	Modify the amount, category, or notes, and save	Updated transaction is saved and reflected in the list	Updated transaction is saved and reflected in the list	Pass
2. Edit transaction image	Change the transaction image	New image is uploaded and saved with the transaction	New image is uploaded and saved with the transaction	Pass
3. Cancel edit	Click on the back button without saving	Discard changes and return to the previous screen	Discard changes and return to the previous screen	Pass

Figure 6- 7 Edit Transaction Module

6.1.8 Add Category Module

Test Case	Input	Expected Output	Actual Output	Result
1. Add a valid category	Enter a valid category name and click on "Add Category"	Category is successfully added to Firebase and user navigates back to the previous screen	Category is successfully added to Firebase and user navigates back to the previous screen	Pass
2. Add category without a name	Leave the category name field empty and click on "Add Category"	Display an alert asking the user to enter a category name	Display an alert asking the user to enter a category name	Pass
3. Switch between expense and income	Toggle the switch to change the category type	The displayed label changes between "Expense" and "Income"	The displayed label changes between "Expense" and "Income"	Pass

Figure 6- 8 Add Category Module

6.1.9 View Category List Module

Test Case	Input	Expected Output	Actual Output	Result
1. View Expense Categories	Open the Category List page with "Expenses" tab selected	Display a list of predefined expense categories along with user-added expense categories from Firebase	Display a list of predefined expense categories along with user-added expense categories from Firebase	Pass
2. View Income Categories	Switch to the "Income" tab	Display a list of predefined income categories along with user-added income categories from Firebase	Display a list of predefined income categories along with user-added income categories from Firebase	Pass
3. Select a Category	Tap on a category item	Navigate back to the previous screen, passing the selected category data	Navigated back to the previous screen with the selected category data	Pass

Figure 6- 9 View Category List Module

6.1.10 Budget Module

Test Case	Input	Expected Output	Actual Output	Result
1. View Budgets	Open the Budget page	Display a list of all budgets with category, amount, and time range	List of budgets displayed correctly	Pass
2. Create a New Budget	Press "Create a budget" button	Navigate to the budget creation screen	Navigates to budget creation screen	Pass
3. Overspent Notification	A budget is overspent	Notification sent to user about overspending	Notification sent	Pass
4. Check for Overspent Budgets	Budgets with overspending conditions	Display the overspent amount in red text	Overspent budget correctly displayed in red	Pass

Figure 6- 10 Budget Module

6.1.11 Add Budget Module

Test Case	Input	Expected Output	Actual Output	Result
1. Enter Budget Amount	Enter a numerical value for the budget amount	Budget amount is correctly entered and displayed	Amount entered correctly	Pass
2. Select Category	Choose a category from the category list	Selected category is displayed in the input field	Category selected and displayed correctly	Pass
3. Select Time Range	Choose a time range (week, month, year)	Selected time range is displayed correctly	Time range selected and displayed correctly	Pass
4. Save Budget	Fill in all fields and click "Save Budget"	Budget is saved to Firebase and user is navigated back	Budget saved successfully and user navigated back	Pass
5. Validation Check	Leave one or more fields empty and attempt to save	Alert appears indicating all fields are required	Error alert shown, preventing save	Pass

Figure 6- 11 Add Budget Module

6.1.12 View Budget Detail Module

Test Case	Input	Expected Output	Actual Output	Result
1. Display Budget Details	Load a budget detail page with category, amount, and time range	Budget details are displayed correctly with no overspent	Budget details displayed as expected	Pass
2. Display Overspent	A budget with overspending is loaded	Overspent amount is shown in red and chart reflects the overspend	Overspent displayed correctly	Pass
3. Delete Budget	Press "Delete" button	Budget is deleted and user navigated back	Budget successfully deleted	Pass
4. Data Point Click	Click on a data point in the chart	Alert appears with the correct data for that point	Alert shown with correct data	Pass

Figure 6- 12 View Budget Detail Module

6.1.13 Create Wallet Module

Test Case	Input	Expected Output	Actual Output	Result
1. Create Wallet	Enter valid wallet details and press "Continue"	Wallet is created, confirmation message is shown	Wallet created successfully, confirmation shown	Pass
2. Create Wallet (Error)	Leave wallet name empty and press "Continue"	Error message stating wallet name is required	Error message shown	Pass
3. Set Currency	Select a currency from dropdown	Selected currency is saved	Currency saved correctly	Pass
4. Set Financial Goal	Select financial goal from dropdown	Financial goal is saved	Goal saved correctly	Pass
5. Target Savings Input	Input a valid savings target	Target savings are saved	Target savings saved successfully	Pass
6. Push Notification	Register for push notifications	Device receives notification token	Token registered and saved in the database	Pass

Figure 6- 13 Create Wallet Module

6.1.14 Menu Module

Test Case	Input	Expected Output	Actual Output	Result
1. Display Wallet Value	User logs in, navigates to the dashboard	Correct wallet balance displayed	Wallet balance displayed correctly	Pass
2. Toggle Balance View	User clicks the eye icon to hide/show balance	Balance hides/shows with toggle	Toggle worked as expected	Pass
3. Pie Chart Income Data	Load income distribution chart	Pie chart reflects income categories accurately	Pie chart displayed with correct values	Pass
4. Bar Chart Spending	Load spending report for current and previous month	Bar chart accurately shows the total spent	Bar chart displayed with correct spending values	Pass
5. Navigation to Reports	Click "See reports" button	Navigates to the report page	Navigated to the report page	Pass

Figure 6- 14 Menu Module

6.1.15 Report Module

Test Case	Input	Expected Output	Actual Output	Result
1. Display Report	Load report with opening balance, ending balance, net income, and charts	All values and charts are displayed correctly	Data and charts displayed as expected	Pass
2. Update Goals	Click on "Update Goals" button	Navigates to "Update Goals" page	Navigation worked as expected	Pass
3. View Details	Click on "View Details"	Navigates to detailed financial report page	Navigation worked as expected	Pass
4. Print to PDF	Press "Print" button to generate PDF report	PDF is generated and sharing prompt appears	PDF generated and sharing worked	Pass
5. Progress Notification	Wallet progress reaches 90% of target savings	Notification for nearing savings goal is triggered	Notification triggered correctly	Pass

Figure 6- 15 Report Module

6.1.16 View Report Detail Module

Test Case	Input	Expected Output	Actual Output	Result
1. Display Financial Health	Load report details with financial health score and progress bars	Financial health score and progress bars displayed correctly	Data and UI displayed as expected	Pass
2. View Details Toggle	Press "View Details" button	Details section expands to show additional financial information	Additional details shown correctly	Pass
3. Slider Adjustment	Adjust income, expense, and debt payment sliders	Sliders adjust values and recalculate health score and savings	Sliders adjust simulation values and update the metrics	Pass
4. Show Financial Tips	Press "View Details" button under financial tips	Financial tips section expands and displays correct tips	Tips displayed based on simulated progress and health score	Pass
5. Progress Bar Display	Simulated savings progress bar updates based on slider adjustments	Progress bar updates and reflects the percentage of target savings	Progress bar updated and displayed as expected	Pass

Figure 6- 16 View Report Detail Module

6.1.17 Scan Receipt Module

Test Case	Input	Expected Output	Actual Output	Result
1. Launch Camera	Press "Capture Receipt" button	Camera launches successfully	Camera launched as expected	Pass
2. Image Capture	Capture a receipt image	Image is captured and displayed	Image captured and displayed correctly	Pass
3. OCR Processing	Process captured image	OCR result is displayed, showing extracted receipt details	OCR result displayed correctly	Pass
4. Extract Details	Extract vendor, date, and amount from OCR	Vendor, date, and amount fields are auto-filled based on OCR	Details extracted and populated correctly	Pass
5. Save Receipt	Enter valid vendor, date, and amount, and press "Save Receipt"	Receipt saved successfully to Firebase	Receipt saved to Firebase	Pass
6. Missing Details Warning	Press "Save Receipt" without filling all fields	Warning message is displayed, indicating incomplete form	Warning displayed as expected	Pass

Figure 6- 17 Scan Receipt Module

6.1.18 Track Achievements Module

Test Case	Input	Expected Output	Actual Output	Result
1. View Achievements	Open "Track Achievements" screen	List of achievements displayed	List of achievements displayed correctly	Pass
2. Unlock First Budget	Create first budget	"First Budget" achievement unlocked and notification sent	"First Budget" achievement unlocked, notification sent	Pass
3. Unlock First Transaction	Add first transaction	"First Transaction" achievement unlocked and notification sent	"First Transaction" achievement unlocked, notification sent	Pass
4. Locked Achievement Display	Open achievements screen without completing any task	Achievements shown as locked (grey icon)	Achievements displayed as locked correctly	Pass
5. Unlock Multiple Achievements	Create budget and record transaction	Multiple achievements unlocked and notifications sent	Achievements unlocked, notifications sent	Pass

Figure 6- 18 Track Achievements Module

6.1.19 Update Goals Module

Test Case	Input	Expected Output	Actual Output	Result
1. View Existing Goals	Open "Update Goals" screen	Current financial goal and target savings are displayed	Current goal and target savings displayed correctly	Pass
2. Update Financial Goal	Select new financial goal and target savings, click "Update"	Goals updated successfully, confirmation alert shown	Goals updated, confirmation alert displayed	Pass
3. Incomplete Form Submission	Leave financial goal or target savings field blank and submit	Error alert shown indicating missing fields	Error alert shown as expected	Pass
4. Load Without Wallet	Open screen without linked wallet in user profile	Error alert shown indicating no wallet linked	Error alert displayed correctly	Pass
5. Successful Navigation	Press back button	User navigated back to the previous screen	Navigation works as expected	Pass

Figure 6- 19 Update Goals Module

6.1.20 Set Location Module

Test Case	Input	Expected Output	Actual Output	Result
1. View Current Location	Open "Set Location" screen	User's current location is displayed on the map	User's location is shown correctly on map	Pass
2. Select New Location	Select a location on the map	Marker added on selected location	Marker added correctly	Pass
3. Confirm Selected Location	Press "Confirm Location"	Selected location address is sent to the previous screen	Address sent back correctly	Pass
4. No Location Selected	Press "Confirm Location" without selecting a location	Error message: "Location not selected"	Error message displayed correctly	Pass
5. Location Permission Denied	Deny location access when prompted	Error message: "Permission to access location was denied"	Permission denied message displayed	Pass

Figure 6- 20 Set Location Module

6.1.21 Add Loan Module

Test Case	Input	Expected Output	Actual Output	Result
1. View Loan Form	Open "Loan Calculator" screen	Loan input fields are displayed	Input fields for loan amount, interest, term shown	Pass
2. Calculate Monthly Payment	Enter valid loan amount, interest rate, and term, click "Calculate"	Monthly payment calculated and displayed	Monthly payment correctly calculated and shown	Pass
3. Incomplete Form Submission	Leave loan amount or interest rate or term blank, click "Calculate"	Error alert shown indicating missing fields	Error message shown correctly	Pass
4. Clear Fields	Enter loan details, then click "Clear"	All input fields cleared	Fields cleared successfully	Pass
5. Display Loan Advice	Enter loan details and calculate payment	Appropriate loan advice displayed	Loan advice displayed correctly	Pass

Figure 6- 21 Add Loan Module

6.1.22 Select Contact List Module

Test Case	Input	Expected Output	Actual Output	Result
1. View Contact List	Open "Contact List" screen	Contact list is displayed with names and phone numbers	Contact list with names and numbers shown correctly	Pass
2. Select Contact	Select a contact from the list	Contact is selected, and navigation returns to the previous screen	Contact selected and navigation works as expected	Pass
3. No Contacts Permission	Deny contacts permission when prompted	Error message is displayed indicating permission is required	Permission error message shown as expected	Pass
4. Toggle Dark/Light Mode	Toggle the theme from light to dark mode and vice versa	UI updates to reflect the selected theme	UI theme changes correctly	Pass

Figure 6- 22 Select Contact List Module

6.1.23 AI Chatbox Module

Test Case	Input	Expected Output	Actual Output	Result
1. Load Initial Messages	Open AI Chatbox with valid userID	Initial messages with tutorial shown	Tutorial and first messages loaded as expected	Pass
2. Send Message	User sends a message to the chatbot	AI responds to the message	AI responds correctly	Pass
3. Handle Quick Reply	Select a quick reply option (e.g., "Give me financial tips")	AI processes the quick reply and gives relevant tips	AI responds with the correct financial tips	Pass
4. Load Previous Conversation	Re-open the chatbox after closing	Previous conversation is loaded from AsyncStorage	Previous messages are loaded correctly	Pass
5. API Call Failure	API key is invalid or expired	Error alert indicating API failure	API error alert shown as expected	Pass
6. Theme Switch	Toggle theme between light and dark mode	UI theme switches correctly between dark and light	Theme switches correctly between light and dark mode	Pass
7. Display Typing Indicator	AI processing a response	Typing indicator shows while the AI is processing	Typing indicator works as expected	Pass
8. Financial Health Calculation	Request "Check my financial health" from AI	AI calculates and returns a financial health score based on user data	Financial health score returned accurately	Pass
9. Analyze Spending Habits	Request "Analyze my spending habits" from AI	AI responds with an analysis of user spending habits	Spending habits analysis provided	Pass

Figure 6- 23 AI Chatbox Module

6.1.24 Facebook Prophet Forecast Module

Test Case	Input	Expected Output	Actual Output	Result
1. Fetch Transaction Data	Open the financial analysis screen	Transactions are fetched and displayed in a time series	Transactions fetched and displayed correctly in the time series chart	Pass
2. Make Forecast	Press "Make Prophet Forecast" button	Forecast data is generated and displayed on the chart	Forecast data displayed correctly, extended time series chart	Pass
3. Category Selection	Select a specific transaction category	Time series chart is updated to display only transactions for the selected category	Transactions filtered correctly by category	Pass
4. Display Insights	Open the financial analysis screen	Insights based on current month's transactions are displayed	Insights displayed correctly, total spending and top category shown	Pass
5. Forecast API Failure	Server returns an error when making a forecast	Error message is displayed	Error message shown as expected	Pass
6. No Transactions Available	Open financial analysis without transactions	A "No expense data available" message is shown	No data message displayed correctly	Pass

Figure 6- 24 Facebook Prophet Forecast Module

6.2 Project Challenge

In addition to the implementation issues discussed earlier, several challenges were encountered during the project. Firstly, the accuracy of the OCR (Optical Character Recognition) used in receipt scanning is heavily reliant on the quality of the image. Factors such as lighting, clarity, and text legibility significantly affect OCR precision, leading to potential errors in data extraction. Another issue is the limited availability of free, high-quality OCR tools. While some are available, many come with costs, creating constraints for projects with tight budgets.

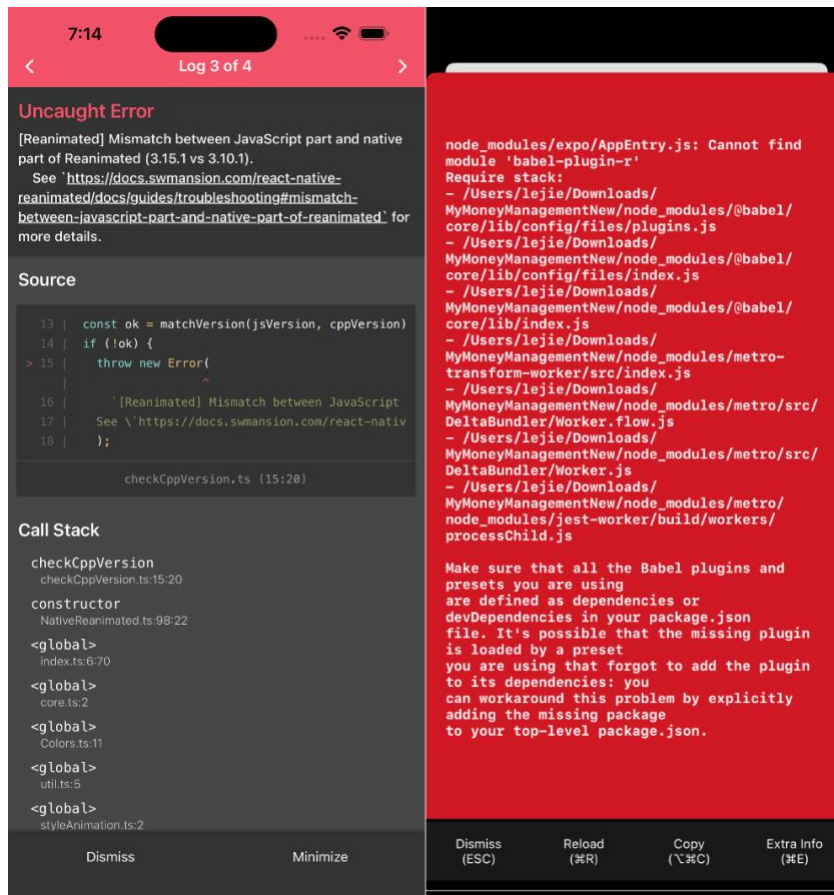


Figure 6- 25 Uncaught Error

In addition to the previously mentioned issues, several technical errors were encountered during the project. A Babel plugin error and a mismatch between the JavaScript and native parts of the Reanimated library (3.15.1 vs 3.10.1) significantly hindered the progress of the project. I attempted several fixes, including updating Reanimated, configuring Babel plugins, clearing the Metro cache, and ensuring Hermes compatibility, but the problem still persisted.

6.3 Objective Evaluation

This project was designed with three main objectives, all of which were successfully achieved as outlined below:

1. **Developing a Personalized Financial Health Score System:** The project incorporated a financial health score system that provides personalized, real-time feedback based on user transactions and financial data stored in Firebase. The system dynamically evaluates financial activities and generates scores reflecting users' overall financial

health. This allows individuals to proactively manage their finances and make informed decisions swiftly. The final system also offers insights into areas such as income, expenses, and debt payments, giving users a clearer understanding of their financial stability.

2. **Implementing an AI Chatbot for Interactive Financial Guidance:** One of the key features developed for this project is the integration of an AI-powered chatbot, which leverages OpenAI's GPT model to provide users with personalized financial guidance. The chatbot can hold real-time conversations, analyze user data stored in Firebase, and offer basic financial advice. While the chatbot doesn't replace professional financial advisors, it significantly improves user engagement and accessibility, allowing users to interactively manage their finances through conversational prompts, such as requesting financial health scores or receiving personalized tips on managing expenses.
3. **Implementing Predictive Analytics for Financial Forecasting:** To further enhance financial planning, the project integrated predictive analytics using Facebook Prophet, a time series forecasting model. By analyzing users' historical financial data, the system generates expense forecasts, helping users anticipate future financial trends. The forecasting tool supports users in setting more realistic financial goals based on predicted cash flows.

These three objectives collectively improve the user experience, providing individuals with dynamic, personalized tools for better financial management. The integration of AI and predictive models makes financial planning more interactive, accessible, and insightful for users of the application.

6.4 Concluding Remark

In Chapter 6, the system's performance across various modules was evaluated, with insights gained from login, transaction management, and AI Chatbox modules. Despite some challenges, such as OCR accuracy and technical errors, the project successfully achieved its core objectives. The personalized financial health system, AI chatbot for guidance, and Facebook Prophet predictive analytics collectively offer users an enhanced, interactive

experience. These features transform how users manage and forecast their finances, laying a solid foundation for future developments in financial management applications.

Chapter 7 Conclusion and Recommendation

7.1 Conclusion

The final output of this project is a personal finance management mobile application focused on improving personal financial health. The app integrates features such as a Personalized Financial Health Score System, an AI-driven chatbot for financial guidance, and predictive analytics for financial forecasting. The primary objective was to provide users with a dynamic platform to manage their finances effectively, offering real-time insights into their financial status, enabling smarter decision-making, and supporting long-term financial health.

In Chapter 1, the introduction highlighted the challenges individuals face in managing personal finances, including a lack of personalized financial feedback and the complexity of forecasting financial trends. This chapter also defined the objectives of the project, focusing on building innovative solutions to address these issues by enhancing personal financial health through AI, financial health scoring, and predictive analytics.

Chapter 2 reviewed both key technologies and existing financial management applications. The technologies examined include OpenAI GPT for developing AI-powered chatbots and Facebook Prophet for time series forecasting. While Google Cloud Vision OCR was reviewed, it was not part of the project's objectives. Additionally, the chapter explored existing applications such as Money Lover, Spendee, Expensify, Mint and YNAB, comparing their strengths and weaknesses with the proposed system. This comprehensive literature review provided insights into where current financial management tools excel and where there is room for improvement, particularly in delivering personalized financial insights and real-time forecasting capabilities.

Chapter 3 detailed the system methodology using the Prototype Model for iterative development. The chapter included system architecture diagrams, use case diagrams, and activity diagrams, explaining how the system's components interact to provide a seamless user experience. These diagrams established the foundation for the app's functionality and how each module contributes to meeting the project's objectives.

Chapter 4 discussed the system design, presenting key components like block diagrams, system flow diagrams, and code descriptions. The chapter explained how the various modules were implemented to ensure that users could access personalized financial health scores, AI-driven financial guidance, and predictive analytics in a user-friendly and responsive manner.

Chapter 5 covered the implementation of the system, focusing on the use of technologies such as React Native, Firebase, OpenAI, and Facebook Prophet. It showcased the development timeline, system configuration, and challenges faced during the implementation phase. Screenshots of the user interface were also provided to demonstrate the app's core functionalities.

In Chapter 6, system evaluation and performance testing were documented. Each core module, including the Personalized Financial Health Score System, AI chatbot, and Predictive Analytics for Financial Forecasting, was tested to ensure that it met the project's objectives. The chapter also discussed challenges encountered during the project, such as technical issues with API integrations and model accuracy, and how these challenges were addressed.

In conclusion, the development of this personal finance management app successfully met its goals by offering a comprehensive solution for improving personal financial health. The application allows users to manage their finances efficiently through features such as financial health scoring, AI-driven interactive guidance, and predictive analytics for future financial trends. This project lays the groundwork for further development and sets a new standard for personal financial management tools.

7.2 Recommendation

While the personal finance management app developed in this project successfully meets its objectives, there are several areas where future enhancements could improve the overall user experience, performance, and feature set. First, expanding the financial health insights with more advanced analytics could provide users with a deeper understanding of their financial situation. By incorporating additional reports, such as detailed cash flow projections or debt management plans, users could gain a clearer perspective on their long-term financial health. Additionally, improving the predictive analytics by integrating more sophisticated models like

LSTM (Long Short-Term Memory) would increase the accuracy of long-term financial forecasts, helping users make more informed decisions about their future financial plans.

Another area for improvement is implementing multi-device syncing and an offline mode. This feature would allow users to seamlessly manage their finances across multiple devices, ensuring a consistent experience whether on mobile or desktop. An offline mode would enable users to input transactions and view their financial data even when they are not connected to the internet, with the app syncing their data once the connection is restored. These additions would greatly enhance the app's usability and accessibility, making it more robust for a wider range of users.

REFERENCES

- [1] V. L. Andrews, "The Early History of Financial Management," in JSTOR, [Online]. Available: <https://www.jstor.org/stable/3665436> [Accessed: Mar. 4, 2024]
- [2] T. Montez, "Transforming personal financial management with AI-driven insights," VentureBeat, May 8, 2018. [Online]. Available: <https://venturebeat.com/ai/transforming-personal-financial-management-with-ai-driven-insights-vb-live/> [Accessed: Mar. 4, 2024]
- [3] A. Lusardi and J. L. Streeeter, "Financial literacy and financial well-being: Evidence from the US," *Journal of Financial Literacy and Wellbeing*, Cambridge Core, 2022. [Online]. Available: <https://www.cambridge.org/core/journals/journal-of-financial-literacy-and-wellbeing/article/financial-literacy-and-financial-wellbeing-evidence-from-the-us/318307008828D2D7932C13E04B90DD88>. [Accessed: Jul. 10, 2024]
- [4] A. Dunn, A. Warren, N. Celik, Ph.D., and W. Chege, "Financial Health Pulse® 2022 U.S. Trends Report," Financial Health Network, 2022. [Online]. Available: <https://finhealthnetwork.org/research/financial-health-pulse-2022-u-s-trends-report/>. [Accessed: Jul. 10, 2024]
- [5] M. Rahman, C. R. Isa, M. M. Masud, M. Sarker, and N. T. Chowdhury, "The role of financial behaviour, financial literacy, and financial stress in explaining the financial well-being of B40 group in Malaysia," *Future Business Journal*, vol. 6, no. 1, 2021. [Online]. Available: <https://fbj.springeropen.com/articles/10.1186/s43093-021-00099-0>. [Accessed: Jul. 10, 2024]
- [6] H.-C. Liu and J.-S. Lin, "Impact of Internet Integrated Financial Education on Students' Financial Awareness and Financial Behavior," *Frontiers in Psychology*, 2023. [Online]. Available: <https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2021.751709/full>. [Accessed: Jul. 10, 2024]

[7] RinggitPlus, "RMFLS Survey Reveals Unexpected Financial Resilience Among Malaysians," 2023. [Online]. Available: <https://ringgitplus.com/en/blog/ringgitplus/rmfls-survey-reveals-unexpected-financial-resilience-among-malaysians.html>. [Accessed: Jul. 10, 2024]

[8] Alliance for Financial Inclusion, "Malaysia's PM launches National Strategy for Financial Literacy (2019-2023)," 2023. [Online]. Available: <https://www.afi-global.org/newsroom/news/malaysias-pm-launches-national-strategy-for-financial-literacy-2019-2023/>. [Accessed: Jul. 10, 2024]

[9] J. Bhagat, "Consumers Need Advanced Financial Management Tools," The Financial Brand, 2024. [Online]. Available: <https://thefinancialbrand.com/financial-wellness-demands/>. [Accessed: Mar. 4, 2024]

[10] Tinybird, "Real-time Personalization: Choosing the right tools," Tinybird, 2024. [Online]. Available: <https://www.tinybird.co/real-time-personalization>. [Accessed: Mar. 4, 2024]

[11] B. Porter and F. Grippa, "A Platform for AI-Enabled Real-Time Feedback to Promote Digital Collaboration," *Sustainability*, vol. 12, no. 24, pp. 10243, Dec. 2020. [Online]. Available: <https://www.mdpi.com/2071-1050/12/24/10243>. [Accessed: Aug. 4, 2024]

[12] "Fintech customer support chatbot powered by Open AI GPT," KindGeek, [Online]. Available: <https://kindgeek.com/blog/post/fintech-customer-support-chatbot-powered-by-open-ai-gpt>. [Accessed: Aug. 15, 2024].

[13] "How to Fine-Tune ChatGPT for Financial Applications," Promptly Engineering, [Online]. Available: <https://promptly.engineering/resources/how-to-fine-tune-chatgpt-for-financial-applications>. [Accessed: Aug. 15, 2024].

[14] M. Melanie, "Facebook Prophet: All you need to know," *DataScientest*, Mar. 20, 2024. [Online]. Available: <https://datascientest.com/en/facebook-prophet-all-you-need-to-know>. [Accessed: Aug. 15, 2024].

[15] T. Cheng, "Introduction to Google Cloud Vision OCR," Nanonets Blog, Jun. 20, 2022. [Online]. Available: <https://nanonets.com/blog/google-cloud-vision/>. [Accessed: Aug. 15, 2024].

[16] Money Lover, "Money Lover: Personal Finance, Budget, Expense Tracker," Money Lover, 2023. [Online]. Available: <https://moneylover.me/>. [Accessed: Aug. 12, 2024]

[17] Expensify, "Expensify: Expense Reports, Receipts, Mileage, Time Entry, Travel," Expensify, 2023. [Online]. Available: <https://www.expensify.com/>. [Accessed: Aug. 12, 2024]

[18] S. P. E. N. D. E. E. a.s. www.spendee.com, "Money manager & budget planner," Spendee. [Online]. Available: <https://www.spendee.com/>. [Accessed: Aug. 13, 2024]

[19] YNAB, "YNAB Loan Planner: Plan for Your Loans," YNAB, 2023. [Online]. Available: <https://www.ynab.com/blog/ynab-loan-planner>. [Accessed: Aug. 13, 2024].

[20] "Mint Software Reviews, Pros and Cons - 2023 Software Advice," <https://www.softwareadvice.com/erp/mint-profile/reviews/> [Accessed: Aug. 13, 2024].

[21] The Knowledge Academy, "Prototype Model in Software Engineering," The Knowledge Academy, 2023. [Online]. Available: <https://www.theknowledgeacademy.com/blog/prototype-model-in-software-engineering/>. [Accessed: Aug. 13, 2024].

[22] React Native, "Introduction to React Native," React Native Documentation. [Online]. Available: <https://reactnative.dev/docs/getting-started>. [Accessed: Aug. 13, 2024].

[23] Firebase, "Firebase Documentation," Firebase Official Website. [Online]. Available: <https://firebase.google.com/docs>. [Accessed: Aug. 13, 2024].

[24] Visual Studio Code, "Visual Studio Code Documentation," Visual Studio Code Official Website. [Online]. Available: <https://code.visualstudio.com/docs>. [Accessed: Aug. 13, 2024].

[25] Node.js, "About Node.js," Node.js Official Website. [Online]. Available: <https://nodejs.org/en/about/>. [Accessed: Aug. 13, 2024].

[26] Expo, "Expo Documentation," Expo Official Website. [Online]. Available: <https://docs.expo.dev/>. [Accessed: Aug. 13, 2024].

[27] Python Software Foundation, "Python," 2023. [Online]. Available: <https://www.python.org/>. [Accessed: Aug. 13, 2024].

FINAL YEAR PROJECT WEEKLY REPORT

13/9/2024

(Project II)

Trimester, Year: T3, Y3	Study week no.: week 3
Student Name & ID: Tan Le Jie 20ACB04995	
Supervisor: Dr Wong Pei Voon	
Project Title: Personal Finance Management Application	

1. WORK DONE

- Completed intro, narrow down the focus area to Personal Financial Health

2. WORK TO BE DONE

- Continue development of the AI chatbot for financial guidance.
- Modify the Personalized Financial Health Score System module

3. PROBLEMS ENCOUNTERED

Nope

4. SELF EVALUATION OF THE PROGRESS

So far so good

wong

Supervisor's signature

Le Jie

Student's signature

134
FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	Study week no.: week 6
Student Name & ID: Tan Le Jie 20ACB04995	
Supervisor: Dr Wong Pei Voon	
Project Title: Personal Finance Management Application	

1. WORK DONE

- Completed literature review on existing financial management applications and key technologies.
- Slightly refined the user interface for Home pages.

2. WORK TO BE DONE

- Begin to integrate TensorFlow.js for predictive analytics and financial forecasting to help users visualize future financial trends.
- Start researching OCR technology

3. PROBLEMS ENCOUNTERED

Some delays due to optimizing Firebase data fetching, which impacted real-time updates.

4. SELF EVALUATION OF THE PROGRESS

So far so good

wong

Supervisor's signature

Le Jie

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	Study week no.: week 9
Student Name & ID: Tan Le Jie 20ACB04995	
Supervisor: Dr Wong Pei Voon	
Project Title: Personal Finance Management Application	

1. WORK DONE

- Complete fyp2 report Chapter 1, 2 and 3

2. WORK TO BE DONE

- Maybe need some modification of the report if the tech study don't fit well inside the app

3. PROBLEMS ENCOUNTERED

Nope

4. SELF EVALUATION OF THE PROGRESS

So far so good

wong

Supervisor's signature

Le Jie

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3, Y3	Study week no.: week 12
Student Name & ID: Tan Le Jie 20ACB04995	
Supervisor: Dr Wong Pei Voon	
Project Title: Personal Finance Management Application	

1. WORK DONE

- Completed AI chatbot integration. Finalized the layout and improved user interaction flow.
- Completed Google Cloud Vision OCR API integration

2. WORK TO BE DONE

Finalize predictive analytics integration and complete the testing phase for all features.

3. PROBLEMS ENCOUNTERED

- The integration of TensorFlow.js for predictive analytics failed due to module conflicts and performance issues. Currently seeking an alternative approach with Facebook Prophet for financial forecasting.
- Faced issues with the accuracy of the OCR during receipt scanning

4. SELF EVALUATION OF THE PROGRESS

Progress has slowed due to technical issues with TensorFlow.js, but overall system development remains steady. A more stable forecasting tool is required.




Supervisor's signature



Student's signature

POSTER



Development of Personal Finance Mobile Application

Introduction

Exploring the frontier of personal finance management, this project introduces an innovative mobile application designed to transform users' financial health. Emphasizing real-time, personalized financial tracking, it addresses the gap in current tools and offers a sophisticated yet user-friendly solution.

Objective

1. Develop a Personalized Financial Health Score System
2. Implement an AI Chatbot for Interactive Financial Guidance
3. Implement Predictive Analytics for Financial Forecasting

Discussion

The core of the project is the Personalized Financial Health Score System, meticulously developed through the Prototype Model for an interactive user experience. Despite integration challenges with the Google Cloud Vision OCR API, iterative testing and refinements have led to a functional prototype, ready for further enhancement.

Conclusion

This project marks a significant advancement in personal finance management by providing users with personalized financial health assessments and interactive guidance. Despite initial API integration challenges, the app successfully integrates AI-driven financial insights and predictive analytics, paving the way for future enhancements.

Universiti Tunku Abdul Rahman

Bachelor of Information Systems (Honours) Business Information System
Prepared by: Tan Le Jie (20ACB04995)

PLAGIARISM CHECK RESULT

Personal Finance Management Application

ORIGINALITY REPORT

1%

SIMILARITY INDEX

0%

INTERNET SOURCES

1%

PUBLICATIONS

0%

STUDENT PAPERS

PRIMARY SOURCES

1

Ton Duc Thang University

Publication

<1%

2

Greg Heiberger. "SOCIAL MEDIA IN THE CURRICULUM AND CO-CURRICULUM: PRE-SERVICE TEACHERS AND THEIR COLLEGIATE PEERS", Thesis Commons, 2018

Publication

<1%

3

Sachi Gupta, Gaurav Agarwal, Shivani Agarwal, Dilkeshwar Pandey. "Depression detection using cascaded attention based deep learning framework using speech data", Multimedia Tools and Applications, 2024

Publication

<1%

4

Pawan Singh Mehra, Dharendra Kumar Shukla. "Artificial Intelligence, Blockchain, Computing and Security - Volume 2", CRC Press, 2023

Publication

<1%

5

Quoc-Tai Duong. "Efficient Integrated Circuits for Wideband Wireless Transceivers", Linkoping University Electronic Press, 2016

<1%

Publication

13/9/2024

6 Mohamad Fazli Sabri, Mervin Anthony, Siong Hook Law, Husniyah Abdul Rahim, Nik Ahmad Sufian Burhan, Muslimah Ithnin. "Impact of financial behaviour on financial well-being: evidence among young adults in Malaysia", *Journal of Financial Services Marketing*, 2023
Publication

<1%

7 Jeswin Jose, Nabanita Ghosh. "chapter 6 Digital Financial Literacy and Its Impact on Financial Behaviors", IGI Global, 2024
Publication

<1%

8 Gianni Nicolini, Brenda J. Cude. "The Routledge Handbook of Financial Literacy", Routledge, 2021
Publication

<1%

Exclude quotes On
Exclude bibliography On

Exclude matches Off

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)	Tan Le Jie
ID Number(s)	20ACB04995
Programme / Course	Business Information Systems
Title of Final Year Project	Personal Finance Management Application

Similarity	Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)
Overall similarity index: <u> 1 </u> % Similarity by source Internet Sources: <u> 0 </u> % Publications: <u> 1 </u> % Student Papers: <u> 0 </u> %	
Number of individual sources listed of more than 3% similarity: <u> - </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.

wong

Signature of Supervisor

Signature of Co-Supervisor

Name: _____
Ts Dr Wong Pei Voon

Name: _____

Date: 13/9/2024

Date: _____



UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	20ACB04995
Student Name	Tan Le Jie
Supervisor Name Ts	Dr Wong Pei Voon

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)
	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
	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.

(Tan Le Jie)

