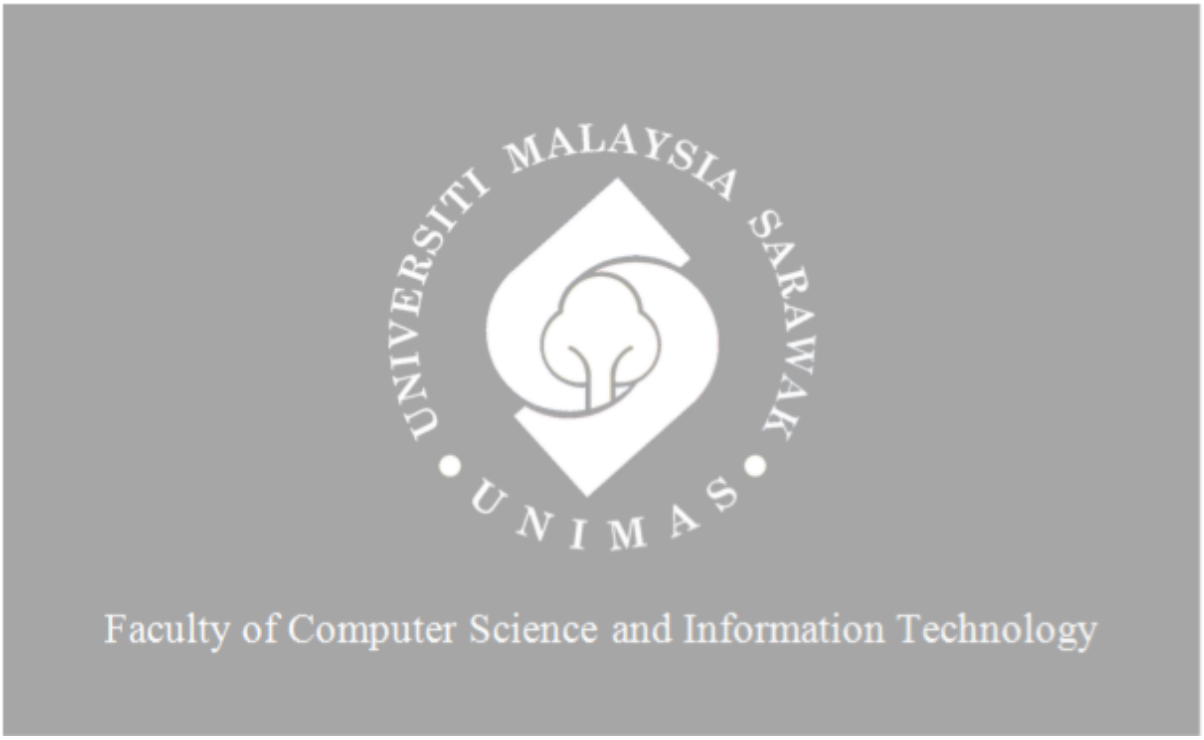


PENNYLOG: EXPENSE TRACKER PREDICTIVE BUDGET OPTIMISATION

JOAANE CHONG AIE YING

79681

Bachelor of Software Engineering with Honours 2025

UNIVERSITI MALAYSIA SARAWAK

THESIS STATUS ENDORSEMENT FORM

TITLE PENNYLOG: EXPENSE TRACKER PREDICTIVE BUDGET OPTIMISATION

ACADEMIC SESSION: Year 2025

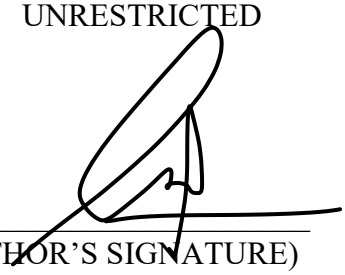
JOAANE CHONG AIE YING

(CAPITAL LETTERS)

hereby agree that this Thesis* shall be kept at the Centre for Academic Information Services, Universiti Malaysia Sarawak, subject to the following terms and conditions:

1. The Thesis is solely owned by Universiti Malaysia Sarawak
2. The Centre for Academic Information Services is given full rights to produce copies for educational purposes only
3. The Centre for Academic Information Services is given full rights to do digitization in order to develop local content database
4. The Centre for Academic Information Services is given full rights to produce copies of this Thesis as part of its exchange item program between Higher Learning Institutions [or for the purpose of interlibrary loan between HLI]
5. ** Please tick (√)

☐	CONFIDENTIAL	(Contains classified information bounded by the OFFICIAL SECRETS ACT 1972)
☐	RESTRICTED	(Contains restricted information as dictated by the body or organization where the research was conducted)
☑	UNRESTRICTED	



(AUTHOR'S SIGNATURE)

Validated by

(SUPERVISOR'S SIGNATURE)

Permanent Address

NO.21 PERSIARAN WIRA JAYA BARAT, 26

TAMAN AMPANG 31350 IPOH PERAK

Date: 28/7/2025

Date: _____

Note * Thesis refers to PhD, Master, and Bachelor Degree

** For Confidential or Restricted materials, please attach relevant documents from relevant organisations / authorities

Declaration

From this point forward, I hereby proclaim that this project and all of its contents are the result of my own effort and is original. Any information, materials, or ideas that have been adapted or gathered from other sources have been accurately referenced or acknowledged in the appropriate manner. The whole project has not been copied or written by any other person, with the exception of instances where appropriate citations have been provided explicitly.

JOAANE CHONG AIE YING

Faculty of Computer Science and Information Technology

Universiti Malaysia Sarawak.

Acknowledgement

I would like to express my deepest gratitude to everyone who has helped me finish my FYP project successfully. I am extremely thankful of everyone's contributions. I would like to begin by expressing my most sincere gratitude to my project supervisor, Ts. Nurfauza Jalil, for providing me with important guidance, support, and constructive feedback during every aspect of the project. Her knowledge and support have been extremely helpful in determining the path that my project would take and improving its overall quality. In addition, I would like to express my deepest gratitude to Mr Abdul Rahman Bin Mat, who served as my project examiner. His comprehensive examination, insightful comments, and recommendations were crucial in ensuring that my project was of a high overall quality.

In addition, I would like to express my gratitude to the Faculty of Computer Science and Information Technology for providing me the resources such as the FYP room which has an atmosphere that was favourable to learning and extremely helpful in the process of developing my project. Besides that, I would want to express my deepest gratitude to my parents and other members of my family for their unwavering support, patience, and understanding throughout this difficult yet rewarding process. The unwavering support and confidence that they have shown in my capabilities has been an unending wellspring of power.

Finally, I would want to express my deepest gratitude to my friends and classmates, whose support, encouragement, and companionship have been a source of inspiration for me during my project journey.

Abstract

PennyLog is a mobile application that was developed with the purpose of simplifying the process of financial management and providing advanced budgeting solutions. Through the utilisation of cutting-edge technology, PennyLog offers users the ability to scan and enhance receipts, automatically classify transactional details, and provide expense forecasting. PennyLog bridges the gap by providing an application that is both simple and easy to use. This allows it to address the issues that are faced by existing systems, such as the availability of advanced categorisation capabilities and predictive analytics, as well as the absence of optical character recognition for text extraction.

Utilising visualisations such as line graph, it provides users with assistance in monitoring their spending patterns and pie charts and expense breakdowns to optimising their expenses with budgeting option. PennyLog was developed with Flutter, for Android devices, and Firebase, which provides backend services for storing data. To ensure that the system meets user requirements and offers a seamless experience, the application will go through multiple testing, which will test all the requirements, functionality and user acceptance test. PennyLog will transform the way in which individuals handle their financial matters by integrating advanced deep learning model with user-centric design. This results in improved financial planning and financial management.

Keywords — Expense Tracking, Optical Character Recognition, Natural Language Processing, Recurrent Neural Networks, Mobile Application.

Abstrak

PennyLog merupakan sebuah aplikasi mudah alih yang dibangunkan bertujuan untuk memudahkan process pengurusan kewangan dan menyediakan penyelesaian belanjawan yang canggih. Dengan memanfaatkan teknologi terkini, PennyLog menyediakan pengguna dengan keupayaan untuk mengimbas hibrid sempadan, klasifikasi transaksi automatik dan perancangan kos.

PennyLog merapatkan jurang ini dengan menyediakan aplikasi yang ringkas dan mudah digunakan. Ini membolehkan untuk mengatasi masalah yang sering dihadapi oleh sistem semasa. Dengan menggunakan visual seperti carta garisan untuk menjejaki tabiat perbelanjaan serta carta pai dan analisis perbelanjaan untuk mengoptimumkan belanjawan, PennyLog membantu pengguna mengurus perbelanjaan mereka dengan lebih berkesan.

PennyLog dibangunkan menggunakan Flutter, bagi peranti Android, serta Firebase menyediakan storan data. Untuk memastikan sistem memenuhi keperluan pengguna dan memberikan pengalaman yang lancar, aplikasi melalui memastikan beberapa ujian dijalankan untuk memastikan tiada masalah dan selamat untuk digunakan. PennyLog akan mengubah cara orang ramai mengurus kewangan mereka dengan menyepadukan model pembelajaran mendalam yang canggih dengan reka bentuk berpusatkan pengguna. Ini membawa kepada perancangan dan pengurusan kewangan yang lebih baik.

Kata Kunci — Penjejakan Perbelanjaan, Pengecaman Aksara Optik, Pemprosesan Bahasa Semula Jadi, Rangkaian Neural Berulang, Aplikasi Mudah Alih

.

Contents

Declaration iii
Acknowledgement iv
Abstract v
Abstrak vi
List of Figure x
List of Table xvii
List of Abbreviations xviii
CHAPTER 1: INTRODUCTION 20
1.1 Introduction 20
1.2 Problem Statement 21
1.3 Scope 22
1.4 Aims and Objective 23
1.5 Brief Methodology 23
1.5.1 Phases in eXtreme Programming Model 25
1.6 Significance of Project 26
1.7 Project Schedule 27
1.8 Expected Outcome 27
1.9 Summary 28
CHAPTER 2: LITERATURE REVIEW 29
2.1 Introduction 29
2.2 Literature Study 29
2.2.1 Machine Learning (ML) 29
2.2.2 Deep Learning (DL) 31
2.2.3 Recurrent Neural Network (RNN) 32
2.2.4 Long Short-Term Memory (LSTM) 33
2.2.5 Natural Language Processing (NLP) 34
2.2.6 Optical Character Recognition (OCR) 35
2.3 Review on Existing Similar Application 36
2.3.1 Money Lover 37
2.3.2 Veryfi 40

2.3.3 Receipt Scanner: Spend Wise 43
2.3.4 Comparison of Existing Applications 46
2.4 Development Tools and Technologies 46
2.4.1 Software 46
2.5 Summary 57
CHAPTER 3: METHODOLOGY 58
3.1 Introduction 58
3.2 Exploration Phase 58
3.2.1 Section A: Demographic Information 59
3.2.2 Section B: Financial Management Habits and Challenges 60
3.2.3 Section C: Preferences and Expectations for an Expense Tracker 64
3.2.4 System Requirements 67
3.3 Planning Phase 71
3.3.1 Overall System Architecture Design 71
3.3.2 Use Case Diagram 72
3.3.3 Activity Diagram 72
3.3.5 Class Diagram 84
3.3.6 Prototype 85
3.5 Productionizing Phase 94
3.6 Maintenance Phase 95
3.7 Death Phase 95
3.8 Summary 95
CHAPTER 4: IMPLEMENTATION 96
4.1 Introduction 96
4.2 Installation and Configuration of System's Component 96
4.2.1 Visual Studio Code 98
4.2.2 Chocolatey, Dart SDK and Dart Plugin 99
4.2.3 Python 102
4.2.4 Android Studio IDE, Android SDK, Java Runtime Environment (JRE) and Java Development Kit (JDK) 103
4.2.5 Flutter SDK 106
4.2.5 Git and GitHub 107

4.2.5 Firestore and Node.js 109
4.2.6 Pytesseract 112
4.2.7 FAST API and Uvicorn 114
4.2.8 Google Cloud Run and Docker 116
4.3 Introducing Role Based Access 127
4.3.1 User 127
4.4 Modules for PennyLog Application 127
4.4.1 Home Module 127
4.4.2 Transaction Module 138
4.4.3 New Transaction Module 149
4.4.4 Insight Module 160
4.4.5 Settings Module 171
4.5 Summary 182
CHAPTER 5: TESTING 182
5.1 Introduction 182
5.2 Functional Testing 182
5.3 User Acceptance Test (UAT) 213
5.4 Summary 228
CHAPTER 6: CONCLUSION AND FUTURE WORK 229
6.1 Introduction 229
6.2 Objective Achievement 229
6.3 Project Limitations 230
6.4 Future Work 232
References 234
Appendix A - Schedule 240
Appendix B – Use Case Specification 241
Appendix C – Requirement Gathering Form 252
Appendix D – User Acceptance Test Form 256

List of Figure

Figure 1.1: eXtreme Programming (XP) Methodology Model 24
Figure 1.2: Project Schedule of FYP 27
Figure 2.1: Google Trend Machine Learning Algorithm 30
Figure 2.2: Deep Learning Workflow 31
Figure 2.3: Deep Learning Technique 32
Figure 2.4: Recurrent Neural Network (RNN) 32
Figure 2.5: Architecture of LSTM 33
Figure 2.6: General process of an OCR engine 36
Figure 2.7: Money Lover Logo 37
Figure 2.8: Money Lover Application 38
Figure 2.9: Veryfi Logo 40
Figure 2.10: Verify Application 42
Figure 2.11: Receipt Scanner: Spend Wise Logo 43
Figure 2.12: Receipt Scanner: Spend Wise Application ..... **Error! Bookmark not defined.**
Figure 2.13: StandardScaler (R, 2024) 55
Figure 3.1: Pie Chart Gender 59
Figure 3.2: Pie Chart Year of Study 59
Figure 3.3: Faculty 59
Figure 3.4: Tracking of Expenses and Income 60
Figure 3.5: Monthly Budget Opinion 60
Figure 3.6: Method to Track Expenses and Income 61
Figure 3.7: Method to Categorise Expenses and Income 61
Figure 3.8: Challenges Tracking Expenses 62
Figure 3.9: Difficulty in Analysing Spending pattern 62
Figure 3.10: Primary Challenge in Managing Finance 63
Figure 3.11: Lost Track of Spendings 63
Figure 3.12: Challenge to Predict Spendings 63
Figure 3.13: Personalising Financial Goal 64
Figure 3.14: Preference Of Expenses Category 64
Figure 3.15: Preference of Income Category 65
Figure 3.16: Preferred Features 65
Figure 3.17: Customisable Interface 66
Figure 3.18: OCR Functionality 66
Figure 3.19: NLP Functionality 66
Figure 3.20: Importance of Prediction 67
Figure 3.21: Optimisation Feature 67
Figure 3.22: Overall System Architecture Design 71
Figure 3.23: Use Case Diagram 72
Figure 3.24: Overall Activity Diagram 72
Figure 3.25: Activity Diagram Home Page 73
Figure 3.26: Activity Diagram Transaction Page 74
Figure 3.27: Activity Diagram Add New Transaction 75
Figure 3.28: Activity Diagram Insight Page 76
Figure 3.29:Activity Diagram Settings Page 77
Figure 3.30: Home Page Sequence Diagram 78

Figure 3.31: Transaction Page Sequence Diagram 79
Figure 3.32: Add Transaction Manually Sequence Diagram 80
Figure 3.33: Add Transaction Using OCR Sequence Diagram 81
Figure 3.34: Settings Sequence Diagram 83
Figure 3.35: Class Diagram 84
Figure 3.36: a) Launching Page and b) Home Page Prototype 85
Figure 3.37: a) Expense Transaction and b) Income Transaction Prototype 86
Figure 3.38: a) Successful and b) Failed Transaction Details Prototype 86
Figure 3.39: a) Transaction Method and b) Transaction Type Pop Up Prototype 87
Figure 3.40: a) Income and b) Expense New Transaction Manual Prototype 88
Figure 3.41: a) Receipt Scanning, b) Receipt Selection and c) photo selection Prototype 88
Figure 3.42:Receipt Scanning Transaction Form 89
Figure 3.43: a) Successful and b) Failed Transaction Status Prototype 90
Figure 3.44: a) Add Budget and b) Insight Generated On Insight Page Prototype 90
Figure 3.45: Settings Page Prototype 91
Figure 3.46: Select Theme Page Prototype 91
Figure 3.47: Currency Selector Page Prototype 92
Figure 3.48: Language Selector Page Prototype 92
Figure 4.1: fl_chart package 97
Figure 4.2: Flutter Pub Get Command 98
Figure 4.3: Pip Install Command 98
Figure 4.4: Pip install -r requirements.txt 98
Figure 4.5: Visual Studio Code 99
Figure 4.6: VS Code Installation Wizard 99
Figure 4.7: Install Dart SDK 100
Figure 4.8: Choco Version 100
Figure 4.9: choco install dart sdk 101
Figure 4.10: dart version 101
Figure 4.11: Dart SDK Environment path 101
Figure 4.12: Dart Availability 102
Figure 4.13: Dart Plugin 102
Figure 4.14: Python Installer 103
Figure 4.15: Python Installation File 103
Figure 4.16: Python Environment Path 103
Figure 4.17: JDK 24 104
Figure 4.18: Successful Installation JDK 104
Figure 4.19: Java Version 105
Figure 4.20: Download Android Studio 105
Figure 4.21: Android Studio Terms and Condition 105
Figure 4.22: Android Studio IDE 106
Figure 4.23: Install Flutter SDK 107
Figure 4.24: Flutter SDK Environment Variable 107
Figure 4.25: Flutter Doctor 107
Figure 4.26: Download Git 108
Figure 4.27: Git Set Up 108
Figure 4.28: Node.js Installation 109

Figure 4.29: Firebase Tool Installation 109
Figure 4.30: CLi firebase login 110
Figure 4.31: Firebase CLi Login Successful 110
Figure 4.32: Create firebase project 110
Figure 4.33: Firebase Project Name 111
Figure 4.34: Firebase Project Ready 111
Figure 4.35: Firebase Project 111
Figure 4.36: Dart pub global activate flutterfire_cli 112
Figure 4.37: Firebase platform configuration 112
Figure 4.38: Configuration successful 112
Figure 4.39: Tesseract Engine Installer 113
Figure 4.40: Tesseract OCR package 113
Figure 4.41: Traineddata Tesseract OCR 113
Figure 4.42: Tess data directory 114
Figure 4.43: Installation uvicorn and fastapi 114
Figure 4.44: Import FastAPI 115
Figure 4.45: Initialise new app 115
Figure 4.46: Run Server 115
Figure 4.47: Server Successful Running 115
Figure 4.48: JSON Response 115
Figure 4.49: FastAPI Swagger UI 115
Figure 4.50: Get Started for Cloud Run 116
Figure 4.51: Account Information Google Cloud 117
Figure 4.52: Verify Identity 117
Figure 4.53: Google Cloud Run Console 118
Figure 4.54: Project List 118
Figure 4.55: Create Project Cloud Run 119
Figure 4.56: Successful Project Creation Cloud Run 119
Figure 4.57: Project List Cloud Run 119
Figure 4.58: Cloud Run Dashboard 119
Figure 4.59: Cloud Run API Library 120
Figure 4.60: Enable API Cloud Run 120
Figure 4.61: Cloud Run SDK 121
Figure 4.62: Cloud Run SDK installer 121
Figure 4.63: Google Cloud CLI setup 122
Figure 4.64: gcloud init 122
Figure 4.65: CLI authenticated 122
Figure 4.66: Docker Installation Complete 123
Figure 4.67: Docker Version 123
Figure 4.68: WSL Error in Docker 123
Figure 4.69: wsl --update 123
Figure 4.70: Artifact registry 124
Figure 4.71: Dockerfile 124
Figure 4.72: Docker build 124
Figure 4.73: gcloud auth 125
Figure 4.74: docker push 125

Figure 4.75: firebase_init.py 125
Figure 4.76: Service Account PennyLog 126
Figure 4.77: Binding to service account 127
Figure 4.78: gcloud run deploy 127
Figure 4.79: Home Page Module 128
Figure 4.80: Drop Down 128
Figure 4.81: fetchTransactionData() function 129
Figure 4.82: Financial balance widget 129
Figure 4.83: Convert month and year 130
Figure 4.84: Format month and year to Firestore format 130
Figure 4.85: Query Firestore for income and expenses total 130
Figure 4.86: Sum the expenses and income 131
Figure 4.87: Update UI with total 131
Figure 4.88: Total Balance UI 131
Figure 4.89: Total Income UI 132
Figure 4.90: Total Expenses UI 132
Figure 4.91: Spending Trend Bar Graph 132
Figure 4.92: initState() for bar graph 132
Figure 4.93: fetchMonthlyExpenses function 133
Figure 4.94: Initialise an empty map 133
Figure 4.95: Query Expenses 133
Figure 4.96: Loop result expenses total 133
Figure 4.97: SetState chart 134
Figure 4.98: Load Bar Graph 134
Figure 4.99: Bar Graph Main Container 134
Figure 4.100: Chart Container 135
Figure 4.101: Alignment and Scaling Bar Chart 135
Figure 4.102: Bar Chart Interactive ToolTip 135
Figure 4.103: X-axis Bar Chart 135
Figure 4.104: Y-Axis Bar Chart 135
Figure 4.105: Black Boarder Bar Chart 136
Figure 4.106: Grid Lines Bar Chart 136
Figure 4.107: Bar chart main data 136
Figure 4.108: _getBarColor function 136
Figure 4.109: Recent Transaction 137
Figure 4.110: Section header for recent transaction 137
Figure 4.111: StreamBuilder latest transaction 137
Figure 4.112: Recent transaction _buildTransactionCard(...) 138
Figure 4.113: Transaction Page 138
Figure 4.114: Income Toggle 139
Figure 4.115: Expense Toggle 139
Figure 4.116: Search Transaction 139
Figure 4.117: TextField onChanged 139
Figure 4.118: Fetch from firestore the search 139
Figure 4.119: filter transaction using searchQuery 140
Figure 4.120: Filter searchQuery 141

Figure 4.121: Display transaction 141
Figure 4.122: Stream Builder for transaction 141
Figure 4.123: Group and Sort by date 142
Figure 4.124: Ui build for transaction list 142
Figure 4.125: Entry.key 142
Figure 4.126: buildStyledTransactionCard 143
Figure 4.127: InkWell 143
Figure 4.128: View transaction details 143
Figure 4.129: paymentMethod _buildDetailRow 144
Figure 4.130: Transaction Edit Mode 144
Figure 4.131: Choose New Image 145
Figure 4.132: _pickNewImage function 145
Figure 4.133: buildReceiptImage() 146
Figure 4.134:_showImagePreviewDialog() 146
Figure 4.135: showDialog() 147
Figure 4.136: Transform.scale 147
Figure 4.137: setState for zoom 147
Figure 4.138: Dialog height limit 147
Figure 4.139: ClipRRect 147
Figure 4.140: Save transaction 148
Figure 4.141: Update transaction in database 148
Figure 4.142: Delete Transaction 148
Figure 4.143: widget.transaction.reference.delete() 149
Figure 4.144: Bottom Navigation Bar 149
Figure 4.145: Transaction Type Pop Up 150
Figure 4.146: Transaction_detail_page.dart 150
Figure 4.147: saveTransaction() 150
Figure 4.148: Upload Image 151
Figure 4.149: crop photo 151
Figure 4.150: cropSelectedImage function 152
Figure 4.151: Preview Image 152
Figure 4.152: Flowchart part 1 153
Figure 4.153: Auto_enhance 154
Figure 4.154: auto_enhance() 154
Figure 4.155: Flowchart part 2 154
Figure 4.156: enhanced image 155
Figure 4.157: HTTP code 200 155
Figure 4.158: Perform OCR Dialog 156
Figure 4.159: Flowchart part 4 156
Figure 4.160: doc_id search for image 157
Figure 4.161: Clean text and categorisation 157
Figure 4.162: Load Excel for NLP 158
Figure 4.163: extract_category_from_text 158
Figure 4.164: Flowchart part 5 159
Figure 4.165: Show OCR result 159
Figure 4.166: Extracted information pop up 160

Figure 4.167: Extracted information in form 160
Figure 4.168: Insight Page 161
Figure 4.169: Mockaroo 161
Figure 4.170: Step 1 Load and Aggregate Expense Data 162
Figure 4.171: Step 2 Prepare Sequences for LSTM Model 162
Figure 4.172: Step 3 Chronologically Split and Save Scaler 163
Figure 4.173: Step 4 building LSTM Model 164
Figure 4.174: Step 5 Train The Model 165
Figure 4.175: Flowchart part 1 insight page 166
Figure 4.176: initState() 166
Figure 4.177: Month Drop down 166
Figure 4.178: selectedMonth 167
Figure 4.179:_loadPredictionForMonth() 167
Figure 4.180: Flowchart 2 insight page 168
Figure 4.181: HTTP 200 Code 168
Figure 4.182: Insight page pie chart and table 169
Figure 4.183: Budget 169
Figure 4.184: Flowchart part 3 insight page 170
Figure 4.185: Flowchart part 4 insight page 171
Figure 4.186: Update Budget and show original insight page 171
Figure 4.187: Settings 171
Figure 4.188 _buildSettingCard 172
Figure 4.189: Select Theme Page 172
Figure 4.190: Save Theme to firestore 173
Figure 4.191: _loadThemeFromFirestore 173
Figure 4.192: Theme GestureDetector 173
Figure 4.193: changeTheme 174
Figure 4.194: _generateThemeData and _getThemeJsonPath 174
Figure 4.195: Select Currency 174
Figure 4.196: Currency List 175
Figure 4.197: showCurrencyPicker 175
Figure 4.198: setCurrency 176
Figure 4.199: loadCurrency 176
Figure 4.200: Select Language 176
Figure 4.201: _showLanguageList 177
Figure 4.202: language_picker import 177
Figure 4.203: _languageList 177
Figure 4.204: Search Language List 178
Figure 4.205: _buildLanguageList 178
Figure 4.206: _saveSelectedLanguage 179
Figure 4.207: loadLanguage 179
Figure 4.208: setLanguage 180
Figure 4.209: load language from SharedPreferences 180
Figure 4.210: Skip English Translation 180
Figure 4.211: Get Device UUID and hash key 180
Figure 4.212: Check Cache 181

Figure 4.213: Send Translation to MyMemoryAPI 181
Figure 4.214: Successful Response Code 181
Figure 4.215: Save To Firebase 181
Figure 4.216: Return Translation 181
Figure 5.1: Section 1: Users Demographic 214

List of Table

Table 1.1: System Specification 22
Table 1.2: Project Constraint 23
Table 2.1 Types of Machine Learning Algorithm Categories 30
Table 2.2: Comparison of Existing Application 46
Table 3.1: Functional Requirement 68
Table 3.2: Non-functional Requirement 69
Table 3.3: Hardware Requirement 70
Table 3.4: Software Requirement 70
Table 4.1: Dependencies 97
Table 5.1: Test Case for Home Module Spending Overview 183
Table 5.2: Test Case for Home Module Spending Trend Graph 184
Table 5.3: Test Case for Home Module Recent Transaction 185
Table 5.4: Test Case for Transaction Page Module Transaction List 188
Table 5.5: Test Case for Transaction Page Module Manage Transaction 189
Table 5.6: Test Case for New Transaction Module Add Manual Transaction 196
Table 5.7: Test Case for New Transaction Module Add Transaction Using Receipt 199
Table 5.8: Test Case for Insight Page Module View Prediction 205
Table 5.9: Insight Page Module Insert Budget 206
Table 5.10: Settings Module Select Theme 207
Table 5.11: Test Case for Settings Module Search and Select Currency 209
Table 5.12: Test Case for Settings Module Search and Select Language 211
Table 5.13: Home Module User Acceptance Test Questions and Result 214
Table 5.14: Home Module Improvement and Suggestion 216
Table 5.15 Transaction Module User Acceptance Test Questions and Result 217
Table 5.16: Transaction Module Improvement and Suggestion 219
Table 5.17: Add New Transaction Module User Acceptance Test Questions and Result 220
Table 5.18: Add New Transaction Module Improvement and Suggestion 222
Table 5.19: Insight Module User Acceptance Test Questions and Result 224
Table 5.20: Insight Module Improvement and Suggestion 225
Table 5.21: Settings Module User Acceptance Test Questions and Result 226
Table 5.22: Settings Module Improvement and Suggestion 227
Table 6.1: Objective Achievement 229

List of Abbreviations

Abbreviation	Word/Phrase
AI	Artificial Intellegence
ANN	Artificial Neural Network
API	Application Programming Interface
ASGI	Asynchronous Server Gateway Interface
CNN	Convolutional Neural Networks
CNTK	Microsoft Cognitive Toolkit
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
CSS	Cascading Style Sheet
CSV	Comma Saperated Value
DDR5	Double Data Rate 5
DL	Deep Learning
ES7	ECMAScript 7
GB	Gigabyte
GPUs	Graphics Processing Unit
HD	High Definition
HP	Hewlett-Packard
Hz	Hertz
IDE	Integrated Development Environment
IIF	Intuit Interchange Format
iOS	iPhone Operating System
IoT	Internet of Things
JDK	Java Development Kit
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
LLVM	Low Level Virtual Machine
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MB	Megabyte
mcOS	Mac Operating System
ML	Machine Learning
MSE	Mean Square Error
MultiNLI	Multi-Genre Natural Language Inference
NDK	Native Development Kit
NLP	Natural Language Processing
NoSQL	Not Only SQL
OCR	Optical Character Recognition
OEM	Original Equipment Manufacturer
OS	Operating System
PC	Personal Computer
PDF	Portable Document Format
RAM	Random Access Memory
RNN	Recurrent Neural Network
RTX	Real-Time Ray Tracing
SDK	Software Development Kit
SQL	Structured Query Language

SSD	Solid State Drive
SVM	Support Vector Machines
tf	tensorflow
TFX	TensorFlow Extended
TL	Transfer Learning
TPUs	Teraflop Processors
TUF	The Ultimate Force
UI/UX	User Interface User Experience
UID	Unique Identifier
UNIMAS	Universiti Malaysia Sarawak
UUID	Device-Based Unique Identifier
VM	Virtual Machine
VS	Visual Studio
VS code	Visual Studio Code
WSL	Windows Subsystem for Linux
XP	eXtreme Programming

CHAPTER 1: INTRODUCTION

1.1 Introduction

Finance is a difficult task to manage in our life. Information regarding financial transactions has historically been recorded and stored manually on paper bills. Although this approach is reliable, looking through actual bills can be very time-consuming and difficult. Instead of the traditional approach to budget management, improvement can be made through technology. As smartphone usage rises, individuals are using mobile applications to assist with most, if not all, of their everyday chores, thus making their lives easier (Harsshita et al., 2024). Therefore, PennyLog will be designed to be a mobile application.

According to Harsshita et al. (2024), an expense tracker application falls under the finance category which is vital to the management of a person's finance and is necessary for keeping track of our day-to-day living spending. According to several surveys conducted across the globe, majority of the respondent found it difficult to give specific figures when asked about their expenses, which suggests that they are not fully aware of their spending habits. Therefore, this demonstrates the need for PennyLog to include an expense analysis section to better understand users' financial behaviours and facilitate efficient resource management. By providing an in-depth breakdown of monthly expenses, in the form of charts, this feature greatly reduces time and improves the accuracy of financial tracking. This helps users understand their spending patterns. Additionally, the application can provide users ways to monitor their expenses and incomes such as daily transaction records (Dadhich et al., 2022).

Additionally, many users prefer application that can allow them to multitask, therefore automation is essential to be included in the application. Features such as Optical Character Recognition (OCR) for text extraction from receipt and Natural Language Processing (NLP) for categorisation of expenses category during transaction can help create a more efficient process and streamlined day-to-day experience for users (Harsshita et al., 2024). Additionally, users will not fear losing the receipts after saving it in the application since they can view each receipt under each transaction, simplifying their task and saving their time compared to manually keeping each receipt, which could be tedious in the long term.

In addition to helping users plan, this proactive approach fosters financial mindfulness, enabling users to make well-informed decisions that support their long-term financial goals (Dadhich et al., 2022).

1.2 Problem Statement

Despite the availability of many tools and online resources, the financial management application and tool available in the market falls short in offering a comprehensive, automated solution (Harsshita et al., 2024). Users are left to deal with repetitious manual task which are prone to errors and mistakes (Bhatele et al., 2023). This can lead to inaccuracies in financial records. When users rely on these records to make financial decision, it can negatively impact their decisions in the future. This is because users are incorrectly interpreting their actual financial circumstances, making it harder to plan properly as well as unable to achieve their financial goals. Secondly, handling physical receipts make managing financing more complicated since paper receipts are quite fragile and can often be misplaced

or there are chances of it being damage in case of a water or fire incident Furthermore, the process of categorising expenses and income remains subjective and inconsistent, with users manually determining which category each transaction falls into. This lack of standardised categorisation makes it hard to see the spending pattern and therefore complicates the process of analysing and improving of the budgeting methods over time (Shelke et al., 2023).

According to Harsshita et al. (2024), the standard budgeting systems in the market have limitations in their scope, with little or no predictive analytics or budget optimisation capabilities. Without these tools, there is certain limitation such as user are not provided with the information on where to improve their expenses as well as the direction needed to plan for future spending, restricting their capacity to proactively manage their financial resources and optimise their budget for long-term stability.

1.3 Scope

The system will develop to:

1. Automate the expense and income tracking for UNIMAS student using OCR and NLP.
2. Building a Recurrent Neural Network (RNN) model using Stacked Long Short-Term Memory (LSTM) for expense prediction and budget optimisation feature.
3. Primary user will be UNIMAS students.

This system is developed as specified in Table **1.1**: System Specification:

Table 1.1: System Specification

Frontend	Using Flutter which is a mobile application development tool with Dart components to provide a user-friendly interface for Android devices
Backend	Python for server-side logic, FastAPI for REST API method creation, Uvicron for local server
Storage	Firebase for database management and cloud connection
Optical Character Recognition (OCR)	Pytesseract for image to text extraction
Natural Language Processing (NLP)	SentenceTransformer model for expenses and income categorisation.
Time series prediction	Recurrent Neural Network (RNN) with Stacked Long Short-Term Memory (LSTM) Model for monthly expenses prediction

Project Constraints:

Table 1.2: Project Constraint

Compatibility	Limited to devices capable of running Flutter applications.
OCR Accuracy	Depending on lighting and image quality, not all receipt types may be recognised with the same accuracy.
NLP Limitation	Categorisation will be based on pre-trained data and may not be fully accurate due to different standardisation.

1.4 Aims and Objective

The overall aim and objective of these projects are:

a. To analyse users' expenses before and after using the application expense prediction and budget optimisation features
b. To design and implement a system that accurately logs expenses using image-to-text (OCR) technologies.
c. To test and evaluate on the usability and functionality of the system solution through user testing.

1.5 Brief Methodology

When developing an application, a software development approach must be chosen to know how the project will kick-off and provides a guideline into the project's successful completion. It plays a vital role in making sure that the development and system's progress proceeds in a more rational and well-organised manner, ensuring that the development does not go beyond its planned scope (Al Fajar et al., 2022). This software technique operates inside the context Agile methodology.

According to Tashtoush et al. (2021), agile methodology focusses on creating functional software that is provided in manageable chunks. This methodology has a flexibility whereby it is acceptable for software requirement to change compared to the initial requirements after early prototyping because the developer will be able to understand and envision how the application will function better. Shorter iterations are used to mitigate this change risk, which lowers the risk associated with significant software integrations. Agile approaches' primary objective is to lower software development process overhead while enabling changes to be adopted without risking the process or requiring an excessive amount of rework (Alsaqqa et al., 2020).

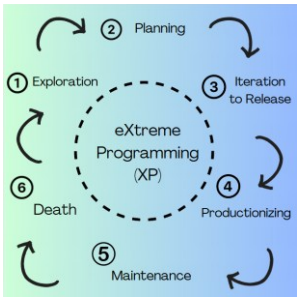


Figure 1.1: eXtreme Programming (XP) Methodology Model

The proposed Methodology to be used in this project is the eXtreme Programming (XP). Due to its ability to handle rapidly changing or complex

requirements, it is a more adaptable, low risk, disciplined, and lightweight approach which fits the project requirements for PennyLog. An overview of the six XP phases is stated as follows:

1.5.1 Phases in eXtreme Programming Model

a. **Exploration Phase**

The initial phase is when the core requirements for users are determined. The developer collects feedback to make sure the application offers useful features that efficiently handle real-world problem. Then a prioritised list of necessary features is produced, with an emphasis on both user needs and technical feasibility.

b. **Planning Phase**

The developer divides the development process into manageable iterations and develops a project roadmap during the planning phase. The problem statement, objectives, scope and limitation are firstly identified. Then, a timeline with distinct milestones is created using iteration cycles ranging from one to two weeks or up to a month. The system architecture and designs are also created during this phase.

c. **Iteration to Release Phase**

The developer starts developing in this phase. Delivering functional features, getting feedback, and adjusting based on observation that is done are the main goals of this phase. Each feature is developed in short, iterative phases. Every feature is examined separately to make sure it satisfies quality requirements and functions exactly as intended. A functional release is created at the end of each iteration to collect user input.

d. **Productionising Phase**

The goal of the productionising phase is to improve PennyLog's usability, stability, and performance to get it ready for a more completed release. Making sure final testing is done, performance optimisation, and user experience enhancements.

e. **Maintenance Phase**

To make sure the software stays responsive to user needs, the developer actively tracks PennyLog's performance during the maintenance period, fixes user-reported problems, and performs continuous enhancements.

f. **Death Phase**

The Death phase is when PennyLog's active development comes to an end whereby there is no more upgrades or new major changes are to be added. The developer concentrates on completing all project documentation during this phase.

1.6 Significance of Project

1. **Modernisation**: PennyLog will replace the manual methods of financial management with automated and optimised feature which are user friendly.
2. **Automation**: The implementation of OCR and NLP simplifies the transaction logging task ensuring a timelier process.
3. **Insights**: PennyLog provides structured, up to date summaries with visualisation for better financial tracking for users.
4. **Predictive Budget Optimisation**: Expense prediction model offers a more personalised prediction with budget optimisation functionality for user to better plan their finance.

5. **Practicality and Scalability**: PennyLog is designed that ensures every feature is useful and practical for users.

1.7 Project Schedule

Figure 1.2 shows the project schedule for PennyLog task that have been defined for this project which will be completed in two academic semester which is 2024/2025-1 and 2024/2025-2. Refer Appendix A for any further details.

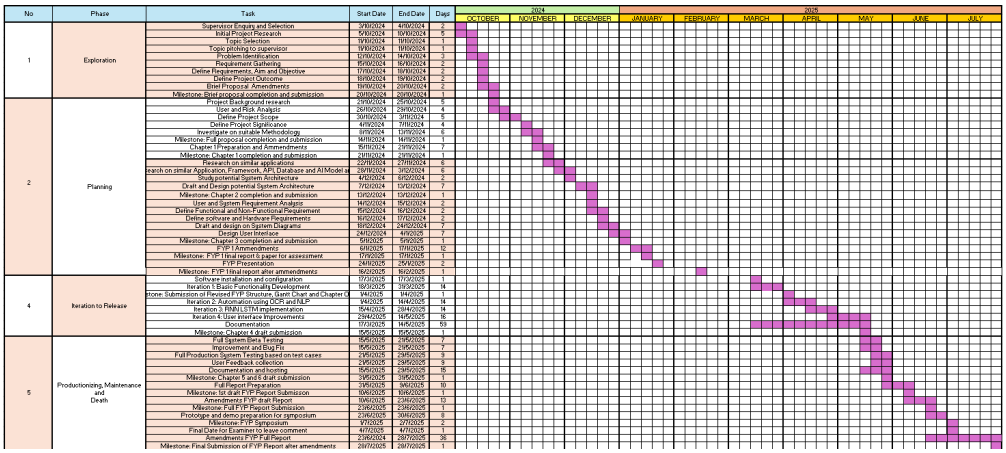


Figure 1.2: Project Schedule of FYP

1.8 Expected Outcome

The expected outcome for PennyLog is to create an application for mobile devices that uses image-to-text OCR technology such as Pytesseract to enable users to conveniently keep track of spending such as expenses and earnings like income. Users will be able to track their everyday expenditures more quickly and easily. Besides that, the application will automatically classify expenses and income using NLP. Users can better organise their transactions which guarantees accurate and consistent tracking while minimising manual tasks.

Additionally, PennyLog will build a RNN LSTM model. This feature offers insightful, data-driven suggestions for future expenses by leveraging deep learning

technique for analysing the trend of users' income and expense historical data. Users will be able to set their own personalised financial limit, and the app offers optimised suggestion for monthly expenses. With the help of this predictive analytics function, users may find areas for possible savings, cut back on wasteful spending, and more strategically manage their budgets to reach their financial goals.

Lastly, to provide users with a clear understanding of their income and expenses trends as well as their overall financial spending, the application will also include interactive charts and data visualisation capabilities.

1.9 Summary

PennyLog is a mobile application designed primarily to enhance and simplify existing financial management application. It gives users, especially students, a convenient way to keep track of their spending and make the most of their income. Financial management may be difficult, particularly when there are many obligations and the tendency for expenses to mount up quickly. Conventional approaches to tracking spending, including manually entering transactions into spreadsheets or on paper, are time consuming, prone to mistakes, and don't offer real-time information. The requirement for a more effective, automated solution that incorporates cutting-edge technologies to streamline expense and income tracking, minimise manual entry, and offer insightful data on spending patterns is addressed by PennyLog in response to this constraint.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

This chapter examines the foundational technologies, framework, and libraries that will be utilised in expense tracking. It also discusses the advantages and disadvantages of these technologies and libraries, and it offers a comparative study of existing application in the market to identify the disadvantages that PennyLog intends to improve. To adapt to the requirements of students, the suggested solution intends to provide an application that is not only user-friendly but also automated and predictive in its approach to tracking expenses.

2.2 Literature Study

2.2.1 Machine Learning (ML)

The field of artificial intelligence (AI), and more specifically machine learning (ML), experienced significant growth in recent years. This growth has enabled applications to function in an intelligent manner. These systems are typically able to learn and improve from experience automatically without being manually programmed. It is therefore essential to utilise machine learning algorithms to intelligently analyse these data and to construct the applications that correspond to them in the actual world. There are four primary types of machine learning algorithms, which are supervised, unsupervised, semi-supervised, and reinforcement learning (H. Sarker, 2022).

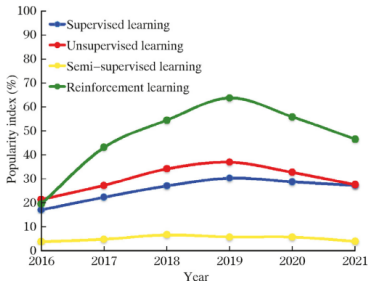


Figure 2.1: Google Trend Machine Learning Algorithm

The popularity of these machine learning algorithms is growing daily, as demonstrated in Figure 2.1, which is based on data obtained from Google Trends from the year 2016 to the year 2021 (Pugliese et al.,2021). Regarding ML, the effectiveness and efficiency of the algorithm are generally determined by the nature and qualities of the data as well as the performance of the learning algorithms. Different learning algorithms serve different purposes, and even the results of multiple learning algorithms that belong to the same category can vary depending on the qualities of the data (H. Sarker, 2022). Table 2.1 illustrates the different type of machine learning algorithm.

Table 2.1 Types of Machine Learning Algorithm Categories

No	Algorithm Type	Model Building	Type
1	Supervised Learning	Using a task-driven approach, algorithms or models learn from data that has been labelled.	1. Classification 2. Regression
2	Unsupervised Learning	The data-driven approach involves algorithms or models learning from data that has not been labelled.	1. Clustering 2. Associations 3. Dimensionality Reduction
3	Semi-Supervised Learning	Model are built with the combination of labelled and unlabelled data.	1. Classification 2. Clustering
4	Reinforcement Learning	Models are being built using the concept of reward and penalty.	1. Classification

2.2.2 Deep Learning (DL)

Deep learning is a subset of machine learning. One of the sources of inspiration for DL is the information processing patterns that may be seen in the human brain. Multiple layers of algorithms, also known as Artificial Neural Networks (ANNs), are utilised in the development of deep learning. Each of these algorithms offers a unique interpretation of the data that has been provided to them (Alzubaidi et al., 2021). In addition, deep learning makes use of transformers and graph technologies simultaneously to construct multi-layer learning models (Alzubaidi et al., 2021). It has been effectively used in varies and its breakthroughs are largely due to the development of robust frameworks that make it easier to create, train, and deploy neural networks (Samrat Medavarapu, 2024). DL modelling has the potential to improve performance in relation to the quantity of data because of its ability to process many features to construct an effective data-driven model (H. Sarker, 2022).

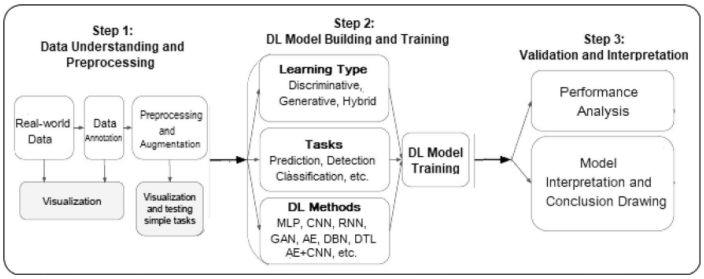


Figure 2.2: Deep Learning Workflow

Figure 2.2 illustrates a deep learning workflow that may be utilised to address real-world issues (H. Sarker, 2020).

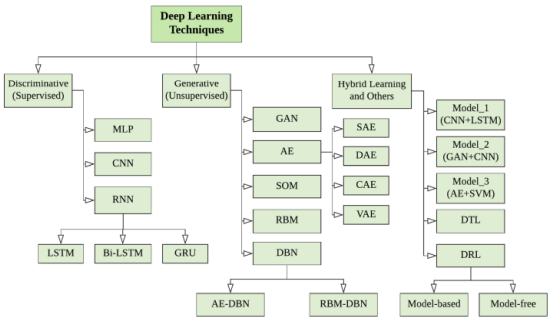


Figure 2.3: Deep Learning Technique

Figure 2.3 illustrates that there are three basic classifications of deep learning approaches, which are discriminative, generative and hybrid learning (H. Sarker, 2020).

2.2.3 Recurrent Neural Network (RNN)

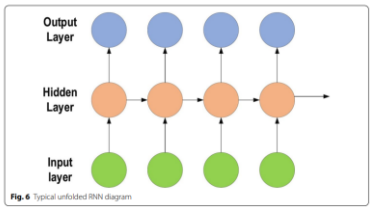


Figure 2.4: Recurrent Neural Network (RNN)

Recurrent Neural Networks (RNNs) is a well-known deep learning technique commonly used in Time series prediction and Natural Language Processing (NLP). The ability to process data in sequence and retain it making the technique effective for capturing contextual information like the meaning of the word or the patterns in financial trend. RNN consist of input, output and hidden layers as seen in Figure 2.4. It functions as short-term memory units that enables the model to learn dependencies over time. RNN can handle sequencing by maintaining memory of precious input which is well suited for time series prediction (Alzubaidi et al., 2021).

The issue with RNN is the vanishing and exploding gradient during training that limits their ability to retain long term dependencies which is crucial for time series prediction. The issue starts when the gradient becomes too small or too large through repeated backpropagation (Sunday, 2024). This causes the network to forget earlier inputs. To solve this, Long Short-Term Memory (LSTM) networks is introduced where memory cells and gated units are used to regulate the flow of information and preserve important data. Supports from framework like TensorFlow and PyTorch, RNN and LSTM architecture significantly improved the time series forecasting capabilities by enabling the model to learn complex and sequential patterns from large datasets (MRS.G.S.N Malleswari et al., 2024).

2.2.4 Long Short-Term Memory (LSTM)

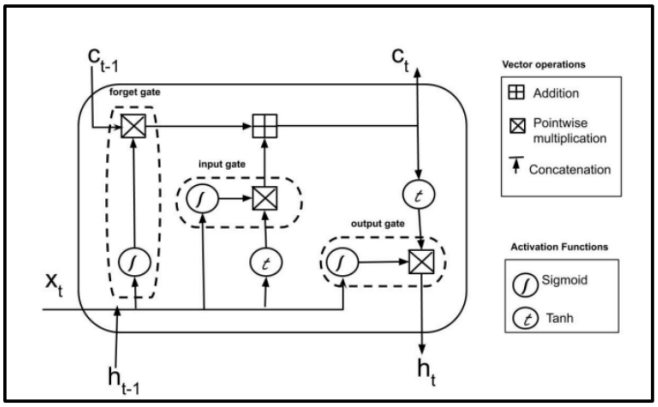


Figure 2.5: Architecture of LSTM

LSTM is RNN specialised architecture designed to effectively capture long term dependencies such as datasets in a sequential manner. LSTM uses feedback connections which allows them to process the entire sequence rather than isolating certain inputs (Saxena, 2024). This has made LSTM to be suitable for task where time series, text, and speech data are used where understanding the past data's is crucial to be able to predict accurately. The architecture can extract and retain relevant information temporal patterns (Singhal, 2024).

According to Pirani et al. (2022), the standard RNN has an issue with long term memory due to the vanishing gradient, whereby with LSTM, it is overcome by using gated memory cells. These gated memory cell which has input, forget and output gates as seen in Figure 2.5 which plays a primary role in controlling the flow of information, which allows the network to decide which data to keep or discard during the training process (Siami-Namini, Tavakoli, & Namin, 2019). This selective memory mechanism has allowed LSTM models to learn which feature is important over time, making it effective for applications like financial forecasting that PennyLog plans to implement. By maintaining and updating internal memory states with the appropriate information's, LSTM ensures that the learning process over extended sequences are stable which makes it an important model in deep learning sequential data.

2.2.5 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a subfield of computer science that focusses on the creation of computer algorithms that are capable of intelligently processing native languages. The virtual computer interface can be made more user-friendly for people through the application of natural language processing (NLP). In the 1950s, natural language processing (NLP) has been employed for the first time in linguistic interaction with artificial intelligence (AI). Speech analysis, machine translation, automatic summarisation, speech analysis, conference resolution, voice recognition, and categorisation are among the primary functions of natural language processing (NLP). A field of research and practice known as natural language processing (NLP) investigates the ways in which computers may be utilised to comprehend and handle the textual content of natural language or speech for the

purpose of practical applications. Natural language processing (NLP) can be thought of as an automatic processing of human language.

The field of natural language processing (NLP) is concerned with the communication that occurs between native human languages and computer technology. Through the application of this method, computer systems can analyse, comprehend, and interpret meaning in human language in a manner that is both practical and intelligent. This contact with the computer makes it possible to implement real-world applications like as search engine results pages (Google), translation systems (Google Translate), automatic question answering, grammar checking, text separation, and a huge number of other applications (Shivahare et al., 2022).

2.2.6 Optical Character Recognition (OCR)

According to Ahirwar et al. (2022), the need for textual information extraction from a variety of sources has increased recently. OCR is a computer vision technology that converts text from images or scanned documents such as pdf into a machine-readable format which then can be utilise for many different purposes. Among the many uses, Data Entry Automation is related to what PennyLog aims for which is to automate the expenses and income entry with the help of OCR and NLP.

Figure 2.6: General process of an OCR engine

Figure 2.6 illustrate how an OCR engine works. The process of optical character recognition (OCR) starts with the capture of an image that contains text by using a device such as a camera or a smartphone. A technique known as image processing is applied to the captured image to improve its quality. This processing may involve scaling, converting to greyscale, or thresholding to improve recognition. After that, the image that has been processed is sent to an OCR engine such as Tesseract OCR engine, which is responsible for extracting text from the image. The recovered text is then subjected to additional text processing, which may include cleaning, formatting, or tokenisation, to render it suitable for analysis or search. The text that has been processed is then saved in a database, where it may be searched or queried whenever it is required. The results are then sent back to the user or the system for further utilisation (V Geetha et al., 2022).

When preprocessing library such as with OpenCV are applied to optical character recognition (OCR), the accuracy of recognition has the potential to even reach 99% or greater levels of accuracy (Wang, 2023).

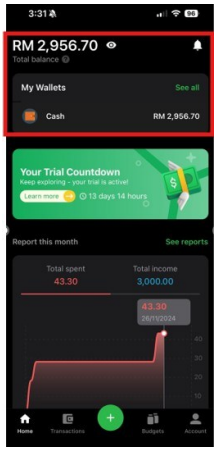
2.3 Review on Existing Similar Application

In the market of financial tracking and management, three application which is Money Lover, Veryfi, and Receipt Scanner are financial management tools. By reviewing these existing systems, we can identify their strengths and weaknesses, allowing us to optimise PennyLog.

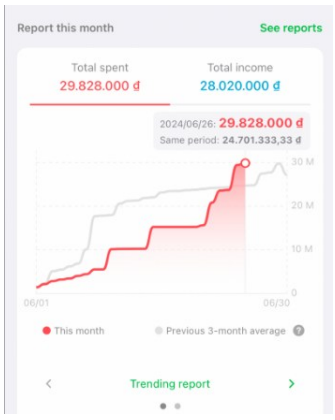
2.3.1 Money Lover

Figure 2.7: Money Lover Logo

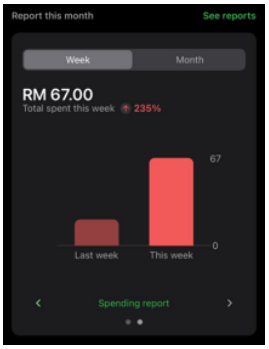
Money Lover is an application build for individuals to manage their financials and is developed by Bookmart Technology. User will need to create an account, and this application is available on Android, iOS, and web platforms. Despites its user-friendly design, several features are only available when user subscribe to their premium version. It is a great tool which are designed and aimed for any individuals. Figure 2.8 (a-i) shows the Money Lover application.



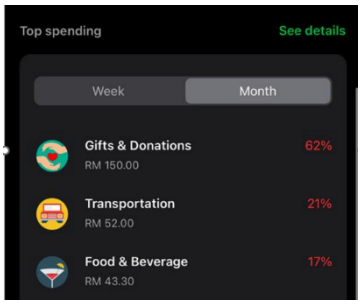
a) Money Lover Home



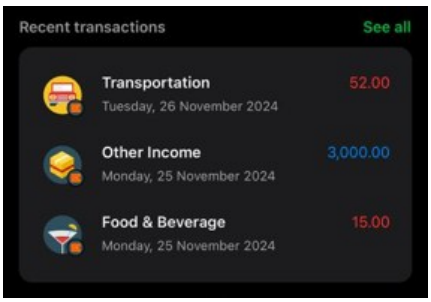
b) Report this month (*Home Screen*, 2024)



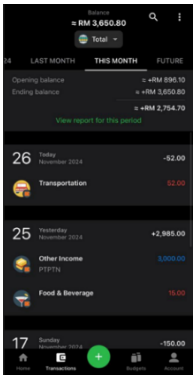
c) Week Spending Report



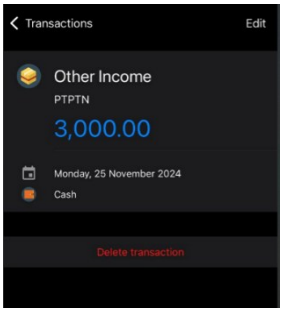
d) Top Spending



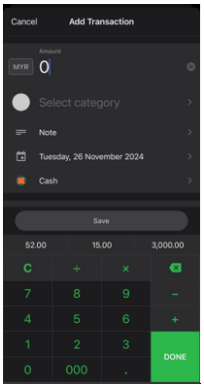
e) Recent Transaction



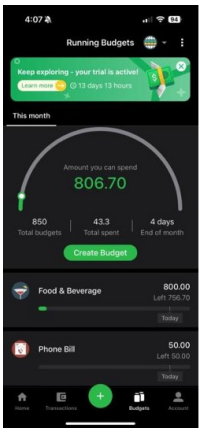
f) Transaction This Month



g) Income Transaction



h) Add New Transaction



i) Budget Tab

The Figure 2.8: Money Lover Application

first section of this application as seen in Figure 2.8 (a) is the Home tab where the total balance of their wallet is shown at the top followed by their wallets. The total is amount added from the wallet. For the wallet, users can add maximum up to 3 wallets. The wallet is customisable to be a basic wallet, a linked wallet, a credit wallet or a goal wallet. Each wallet can hold different currency depending on how they customise it. For user that have premium account, they can link their banks to the wallet.

The next section under the Home tab is the Report this month. The first report as seen in Figure 2.8 (b) is a line graph for total spending and total income for the current month and the previous 3 months. The second report as seen in Figure 2.8 Figure 2.8 (c) is the current weekly and monthly spendings in the form of bar graph. A percentage indicator is also shown to detect changes in their spending pattern. Both reports give a comparative perspective between weekly and monthly spendings that is designed for both short-term and long-term financial planning.

There is also a Top spending sections as seen in Figure 2.8 (d) showcasing the top spending categories by week and month as well as the recent transaction section as seen in Figure 2.8 (e) that records down the recent records of expenses and income recorded by their date of entry. A see all button is available to navigate user to the transaction tab.

The transaction tab as seen in Figure 2.8 (f) displays the total balance at the top with toggles between this month, last month or a particular spending period that they would like to view. This section is where user can track all their expenses and income as well as view individual transaction details as seen in Figure 2.8 (g).

Users will have options to edit or delete the individual transactions. There is also a search functionality available by using keywords like category or notes. This flexibility allows users to have control over their financial record and save time.

Furthermore, there is a section where user can add new transaction as seen in Figure 2.8 (h). Users must manually key in the fields such as amount select category from a list provided, note, data, time and payment type. Users have the freedom to choose category based on expense, income and Debt/Loan. They also can add new category if they wish to. The ability to choose and customise their transaction is to ensure that users have the freedom to organise their transactions according to personal preferences

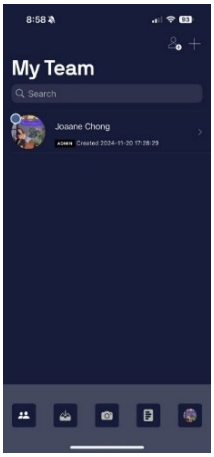
Lastly is the budget tab as seen in Figure 2.8 (i) where users can create their own budgets. There is a progress indicator showing the total budget alongside the total that they have spent, and the number of days left to reach the end of the month. This section provides users with real-time insights into their budgeting progress and ensures that they stay within the budget that they have set.

2.3.2 Veryfi

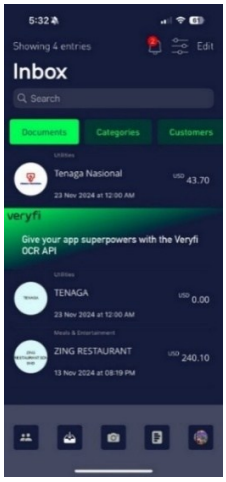
Figure 2.9: Veryfi Logo

Veryfi is a mobile application developed by Veryfi, Inc where the application is designed to facilitate expense tracking, document management as well as financial organisation. The application incorporates advanced technologies such as OCR and AI to automate the capturing, processes and organisation of receipts,

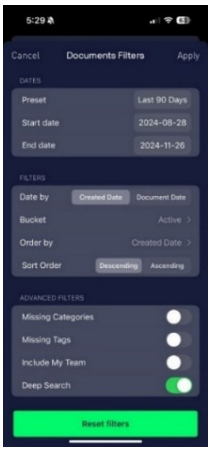
invoice and many other financial documents that are captured in real time or uploaded. It includes NLP for categorisation of the captured financial documents into categories. Veryfi target market group are small businesses, freelancers as well as individuals. User will need to create an account to use the application. **Figure 2.10** (a-f) shows the Verify application.



a) My Team Tab



b) Documents Inbox Tab



c) Document Filters

d) Virtual Scanned Receipt

e) Camera for Long Receipt **f) Report Preview**

Figure 2.10: Verify Application

In Veryfi, the first tab is the My Team tab as seen in **Figure 2.10** (a) whereby the admin is the user that create the account. They are the primary user that have full control over the management of the application. They can also add other user to use the same account, which encourage collaboration especially for business purposes.

Next is the inbox tab as seen in **Figure 2.10** (b). The application uses OCR to extract key information's, NLP to categorise upload receipt of document and all scanned receipts and documents are then uploaded and stored in the inbox tab. The inbox are organised into a specific section such as document, category, customer, tags, vendors and payment depending on how the users would like to view it.

At the top of the Inbox tab, user can filter specific document using start date, end date, date by, order by, sort order and others which eliminates the need to do so manually as seen in **Figure 2.10** (c). Each documents can be individually viewed when user clicks on it and it offers both original and virtual version of it as seen in **Figure 2.10** (d). For each documents, user can categorise, edit, delete,

archive, share and give a specific tag which allows user control on how they want to manage all their documents. These features also enhances and improves the accessibility and usability of all the uploaded documents for better management.

When user click on the camera tab as seen in **Figure 2.10** (e), they are allowed to capture receipt in many different sizes. Besides that user can also upload pre-existing image from their gallery. Furthermore, the application also provides functionality for users to upload pdf or files from their mobile devices if the documents are in different format.

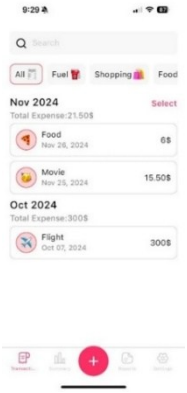
Lastly, the report tab is where user can generate reports of uploaded financial documents in Comma Separated Value (CSV), Portable Document Format (PDF) or Intuit Interchange Format (IIF) format as seen in **Figure 2.10** (f). The report can also be emailed, print or even export externally. This way it simplify task such as bookkeeping and financial planning.

2.3.3 Receipt Scanner: Spend Wise

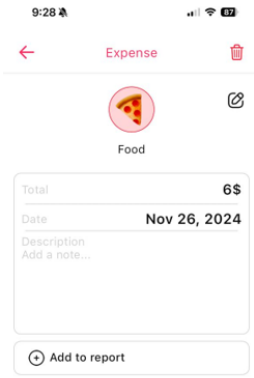
Figure 2.11: Receipt Scanner: Spend Wise Logo

Receipt Scanner: Spend Wise is a mobile application developed by Eywin Bilgi Teknologileri Anonim Sirketi. This application is designed to be able to streamline the management of expenses by allowing users to add expenses manually or using OCR feature to scan, store and categorise receipts. Additionally, the application also provides visualisation statistics so that users able to gain better

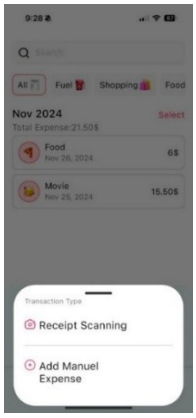
insights into their spendings expenses. However, many of these advanced features requires additional subscription. **Error! Reference source not found.** (a-f) shows the Verify application.



a) **Transaction Tab**

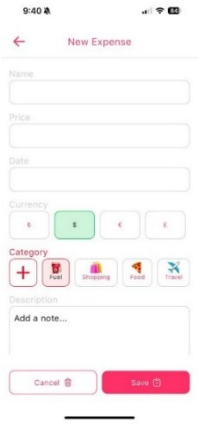


b) **Expense Preview**

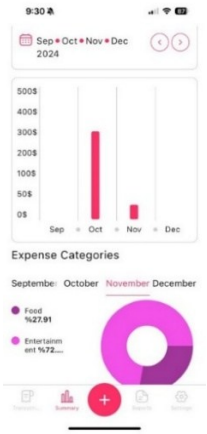


c) **New Transaction Entry**

d) **Camera View for Scanning**



e) **Manual Expense Entry**



f) **Summary Tab**

Figure 2.12: Receipt Scanner: Spend Wise Application

The first tab is the transaction tab as seen in **Error! Reference source not found.** (a) which stores all transaction according to month year and category that users can toggle between. Each transaction is displayed with a brief description such as the amount, category, date and the total for that month. There is also a search functionality at the top where user can search transaction user keyword. The functionalities help facilitate easier and more convenient finance overview

Each transaction can be viewed in detailed as seen in **Error! Reference source not found.** (b) such as category, total, date and description. User also have the flexibility to edit and delete the transaction in case there is a need to correct error or remove any unnecessary entry. For the premium account users, user can generate financial summary and perform analysis.

To add new expenses, user can use receipt scanning to perform OCR or manually add as seen in **Error! Reference source not found.** (c). For the receipt scanner, user will just need to take a picture using the camera as seen in **Error! Reference source not found.** (d) and the camera will automatically scan the section that covers the receipt to digitise the transaction. For manual entry, user will just need to fill in the necessary fields as seen in **Error! Reference source not found.** (e).

As seen in **Error! Reference source not found.** (f), the summary tab is where users are given an overview of their summarised financial transaction in the form of bar graph. The visualisation presentation will allow user to track and compare their current and previous spending pattern over several months. The summary can be viewed for a 4-month period at a time or individually each month.

Furthermore, there is also a summary break down for each expense by category where each expenses total is displayed in percentage format. Access to the last tab which is the report tab only can be accessed using the premium version.

2.3.4 Comparison of Existing Applications

Table 2.2: Comparison of Existing Application

Feature	Money Lover	Veryfi	Receipt Scanner	PennyLog
Premium features	Some features are limited to premium subscription	Some features are limited to premium subscription	Some features are limited to premium subscription	No premium features
Available in iOS	✓	✓	✓	×
Available in Android	✓	✓	×	✓
Login Functionality	Need to log in	Need to log in	Need to log in	No login required and usesdevice UUID
OCR Functionality for receipt scanning	×	Limited to premium subscription	Limited to premium subscription	Free for all to use
NLP Functionality for categorisation	×	✓	✓	✓
Graphs and Visualisation	✓	×	✓	✓
Reports and Analytics	✓	✓	✓	✓
Budgeting Tools	✓	×	×	✓
Multiple Currency Support	✓	✓	✓	✓

2.4 Development Tools and Technologies

2.4.1 Software

This section will provide the software needed to develop PennyLog.

Visual Studio Code (VSC)

According to Tan et al. (2024), Visual Studio (VS) Code is a free and open-source, lightweight Integrated Development Environment (IDE) that was created

and maintained by Microsoft. VS Code's primary feature is its extension support, which enables users to enhance their installation with other languages, debuggers, and tools to facilitate future development. In addition to Microsoft's standard extensions, third-party organisations and individual developers have submitted many extensions to the Visual Studio Code Extension Marketplace. Because of its cross-platform compatibility, Visual Studio Code may be used on a wide range of devices with the same user interface, including Windows, Linux, and macOS. Visual Studio Code is lightweight and ideal for all students using a variety of devices. Since the IDE supports multiple languages and nearly all the major programming languages have extensions, it is suitable to be used for PennyLog development that uses both Dart and Python programming language.

Flutter and Dart

Flutter is a cross-platform framework designed for creating mobile apps with exceptional performance. Google made Flutter available to users in 2016. Flutter apps are compatible with Android and iOS (Tashildar et al., 2020).

According to Tashildar et al. (2020), every application in Flutter is created using Dart. Dart is a programming language created and maintained by Google. It is widely utilised at Google and has demonstrated the ability to create huge web applications like AdWords. Dart was first created as a JavaScript replacement and successor. As such, it incorporates most of the core features of JavaScript's next standard (ES7), including the keywords "async" and "await." However, Dart's Java-like syntax is meant to appeal to those who are unfamiliar with JavaScript. Dart is a client-optimised programming language that can be used to create quick apps on any

platform. A lot of single-page applications can be made with Dart. One well-known example is Gmail, where the browser remains on the same page when you click on a message in your inbox (Sri Swathiga et al., 2021).

Firestore

Cloud Firestore is a NoSQL cloud database within Firebase that is both scalable and adaptable. Data synchronisation and storage are utilised for client-side and server-side development. It is utilised for server, browser, and mobile development using Firebase and Google Cloud Platform. The document-oriented, NoSQL database in the Cloud FireStore stores data in JavaScript object notation (JSON) format. NoSQL databases were developed to solve the problems associated with processing large amounts of data from several sources and formats in big data situations. NoSQL databases is a non-relational, horizontally scalable database management systems, operate on standard-configuration machines. By comparing with many different data models, NoSQL data stores are incredibly adaptable, scalable, and efficient as a data management system for big data. Instead of using the conventional schema-rigid SQL approach, relational data can be developed in a lightweight, flexible manner utilising JSON document storage and a straightforward NoSQL-style application programming interface (API). Collections of schema-flexible document entities can be created, read, updated, and deleted (CRUD) in these operational stores (Semma et al., 2022).

According to Srivastava et al. (2017), combining Firebase Authentication with Cloud Firestore offers a dependable way to secure application data and manage user identification. Firebase Authentication is flexible for a range of application

requirements because it allows several sign-in methods, such as email/password, social media providers, and anonymous authentication. Cloud Firestore which is a real-time, adaptable database that easily combines with Firebase Authentication to provide secure data access on a per-user basis. For example, developers can use device-generated UUIDs or anonymous authentication in applications when conventional email/password registration is not necessary which is what PennyLog is aiming for. By connecting these UUIDs to the Firebase UID, users may be securely and seamlessly identified across installations or sessions. Therefore, firestore will be used as the primary database for PennyLog.

Tensorflow

TensorFlow is an open-source library which was initially created by Google as an internal machine learning tool. In 2017, a stable version was made accessible under the Apache Open-Source License. The Apache Open-Source License permits the use, modification, and redistribution of libraries without requiring payment to Google. Because it is an open-source software, anyone can download, modify, and use the code because it was designed to be adaptable, effective, extendable, and portable (Abrahams et al., 2016). Developers who work on open-source projects are free to improve the code, and they can then propose their changes to be incorporated into a later version. TensorFlow is now being used for building and deploying both machine and deep learning applications. Its growing popularity has led to ongoing improvements from Google and other developers (Joseph et al., 2021).

TensorFlow is appropriate for a variety of applications, ranging from large-scale production systems to research projects, because it provides both high-level

and low-level APIs. Tools like TensorFlow Extended (TFX), TensorFlow Lite, and TensorFlow Serving are part of its vast ecosystem and are used for model training, optimisation, and deployment (Samrat Medavarapu, 2024). This tool can run on any type of computer, from smartphones to massive computing clusters. Essentially, it eliminates the need to reimplement models by providing lightweight software that may immediately create your trained model. Multidimensional arrays, or tensors, are the input that is supplied into TensorFlow. TensorFlow got its name from the flow of a multidimensional array. After entering the system as input at one end, these tensors undergo several procedures before producing output. The development phase and the run phase are its two stages. With the use of GPUs, the development phase is carried out on high-performance workstations, PCs, or laptops for training data. (Joseph et al., 2021).

TensorFlow will serve as the backend framework that powers the computations for building, training, and deploying the time series prediction model for PennyLog. For example, Adam optimiser, callbacks like EarlyStoppings and ModelCheckpoint as well as saving and loading model.

Keras

According to Samrat Medavarapu (2024), Keras is easier for beginners to use for its renowned simplicity and use, having been first created as a high-level API for creating and training neural networks. It enables developers and researchers to easily prototype and create complex models using minimal coding. Its flexibility and user-friendly API make it well-liked by developers and researchers. TensorFlow are among the backend that Keras supports.

Keras capabilities have been further enhanced by the incorporation of TensorFlow. When utilised with TensorFlow as the backend, Keras produced results that were on par with native TensorFlow but requiring a lot less code and development effort. This integration creates a smooth experience for creating and implementing deep learning models by fusing the robustness and scalability of TensorFlow with the simplicity of Keras. Deep learning required quick prototyping and simplicity, which Keras offered. Keras lets users concentrate on creating and training models by abstracting away the complex nature of backend engines like TensorFlow and Theano (Samrat Medavarapu, 2024).

TensorFlow became a well-rounded deep learning framework with an extensive range of capabilities, scalability, and flexibility. While high-level API like Keras, makes model development and training easier, TensorFlow's low-level API offers specific control over model architecture and optimisation. TensorFlow's integration with Keras has produced an efficient framework that takes advantage on both technologies' advantages. The ecosystem of TensorFlow consists of TensorFlow Serving for scalable model serving, TensorFlow Lite for model deployment on mobile and edge devices, and TensorFlow Extended (TFX) for end-to-end machine learning pipelines. By extending Keras' capabilities, these tools make it possible to produce and deploy deep learning models with ease. Model training, analysis, deployment, and data validation are all supported by the TensorFlow Extended (TFX) platform. This integration makes it easier to create scalable and reliable machine learning and deep learning pipelines, guaranteeing that models can be implemented and observed in real-world settings which is what PennyLog needs (Samrat Medavarapu, 2024).

PennyLog will be utilising sequential modelling, LSTM layers, Dense Layer, Dropout layer, model compilation and training as well as model summary for the time series forecasting.

Pytesseract OCR

Hewlett-Packard created Tesseract between 1984 and 1994. Since 2005, Tesseract has been an open-source program, and since 2006, Google has funded its advancement (Koo & Khor, 2023). A wrapper for Google's Tesseract-OCR Engine is called Pytesseract. Pytesseract is a tool written in python for OCR. In other words, it will recognise and "read" the text that is embedded in images. It is especially useful as a standalone invocation script for tesseract since it can read any picture type including jpeg, png, gif, bmp, tiff, and others that is supported by the Pillow and Leptonica imaging libraries. Furthermore, rather than saving the recognised text to a file, pytesseract will print it if it is used as a script (*Pytesseract*, 2024).

There are many practical applications for Tesseract, from enhancing accessibility for people with visual impairments to digitising old records and extracting text from bills, receipts, and forms. Because of its strength and adaptability, Tesseract is a vital tool for data scientists, creating new opportunities for machine learning and data analysis (Kumar, 2023). Pennylog implementation of OCR for image to text extraction is by using pytesseract from receipts uploaded or captured by users.

Sentence Transformer (“all-MiniLM-L6-v2”)

The All-MiniLM-L6-v2 model is an NLP model which is developed by Microsoft. It is part of the MiniLM family. MiniLM propose is to compress large

Transformer-based pre-trained models. The structure of this model includes a 6 Transformer-encoder-layers which is a simplified version of the large models such as the 12 layers of BERT-base model. The BERT model is a pre-trained language model using Transformer architecture. For each layer, it has smaller number of hidden units which is fewer than the 768 units of BERT-base. The model is designed to be lighter and load faster and reasoning by reducing the number of layers, hidden units and parameters. The model has been trained to support multiple languages which helps in the case of PennyLog for the usage of NLP task that support cross-language usage. The model can consider the context before and after the word to better capture the context information (Yin & Zhang, 2024).

Figma

According to Faroq Santoso (2024), Figma is one of the most popular software design tools frequently used for designing, wireframing, prototyping, and interface design for mobile applications and websites. Figma can be run on any operating system. In general, people who work in the application design industry, such as web designers and UI/UX designers, utilise this program extensively. It has a rather advanced graphic design editor with features that many are accustomed to (Greiner-Petter & Ibachb, 2021).

Figma is also a platform for communities to exchanging ideas and solutions. Figma is used by designers worldwide for collaborative projects, vector graphics, and graphic design for digital media in addition to interface creation. Figma also offers plugins made by other community members to help streamline processes. It is also possible for users to create their own plugin and distribute it to others

(Staiano, 2022). PennyLog will be utilising Figma to produce the initial prototype mobile application design.

Pandas and NumPy

The Pandas library in Python is a useful tool that efficiently work with data structures. NumPy, which is used for numerical computations are an example of library that it is compatible with the Pandas Library. Pandas offers a full platform for data analytics and statistical computing, which includes tools for data manipulation that are like SQL. These capabilities include the ability to merge datasets using a various type of joins, including inner, right, and left joins. Pandas, which is derived from the term "Panel Data," provides advanced capabilities such as hierarchical indexing and automatic data alignment (Snehkunj & Vachiyatwala, 2022).

In-built support for the NumPy library is included in Pandas. This library is designed to handle numerical data and operations such as indexing, sorting, and reshaping. The functionality of NumPy is extended by Pandas so that it can efficiently manage heterogeneous data, while NumPy is capable of handling homogeneous array data. Data visualisation is made possible with the help of Matplotlib, which is supported by Pandas. Additionally, files can be saved in a variety of formats, including Excel, CSV, and JSON. Pandas is equipped with three primary data structures. Firstly, is Series, which is a one-dimensional array for homogeneous data, second is DataFrame, which is a two-dimensional array that supports heterogeneous data and mutability and lastly Panel, which is a three-dimensional array for complex data representations. When it comes to managing, analysing, and visualising data in Python, Pandas is an exceptional library that is

necessary (Snehkunj & Vachiyatwala, 2022). In the case of PennyLog which has dataset saved in Excel sheets, both Pandas and NumPy are utilised.

Scikit-Learn

The Scikit-learn is a Python library that is open-source. While scikit-learn is primarily designed in a high-level language and places an emphasis on user-friendliness, careful consideration has been given to maximising computational performance. Preprocessing, model selection, and evaluation are some of the machine learning tools that are provided by Scikit-learn. StandardScaler which is part of Scikit-learn, is frequently utilised in the context of time series forecasting. StandardScaler or Standardisation is also known as Z-score scaling or zero-mean scaling is a common method used in data preprocessing steps to scale and centre features. The feature is beneficial for like PennyLog where the data have varying spread of values and the prediction model are sensitive to the features scale. It will transform the data to have a mean of 0 and a standard deviation of 1.

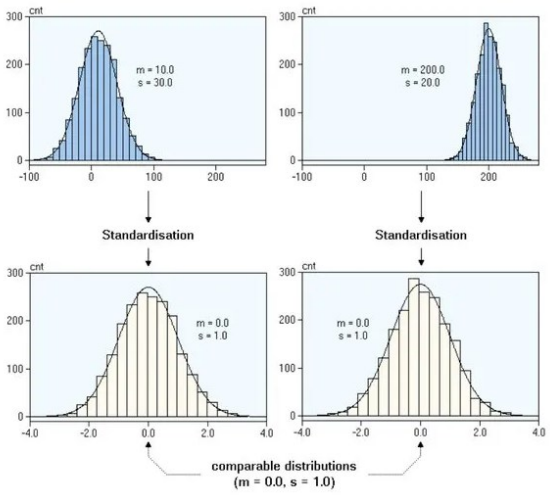


Figure 2.13: StandardScaler (R, 2024)

This is because when data is too high or too low, it will be challenging for the model to capture important patterns which are widely spread. By standardisation, the data will be centralised and can improve the training process of the model as seen in Figure 2.13 (De Amorim et al., 2022).

OpenCV

OpenCV is a python library which is widely used for image processing. It is available for free as it is an open-source library distributed by the Berkely Software Distribution license. Initially started as a research project by Intel, it is now a popular library that contains many tools that solves computer vision problems. It has low level image processing functionalities suitable for PennyLog image enhancement process and high-level algorithms for processing that involves face detection, feature matching and tracking as well as others. For PennyLog image enhancement, it will use the image filtering technique. OpenCV has both linear image filtering and nonlinear image filtering which PennyLog implements both (Mahamkali & Ayyasamy, 2015). For linear image filtering, PennyLog applies the sharpening Kernal technique whereas for the non-linear image filtering, PennyLog implements the Adaptive Thresholding, Denoising, Gamma correction and CLAHE.

FastAPI and Uvicorn

FastAPI is a high-performance python web framework that is used to build REST APIs. It is used to define routes, logics, models and validation with python. FastAPI can integrate seamlessly with Uvicorn which is a lightweight Asynchronous Server Gateway Interface (ASGI) server that can handle multiple connections and event that is incoming and outgoing synchronously

(GeeksforGeeks, 2023). PennyLog will be creating the GET and POST Rest API using FastAPI, while using uvicorn to run the API locally or in production inside a container.

Google Cloud Run

Google Cloud Run is a service provided by Google. It is a serverless computing platform that eliminates the need for extra expenses for onsite hardware, software and storage infrastructure. The platform run containerised application which are cost-effective with the help of Docker. Developers all around the world can build and run their applications without having to manage the servers. Using the free tier version, when the app is idle, it scales down to 0 and only charge for resources which is used. It has logging and monitoring features as a add on (Khatiwada & Dhakal, 2024). PennyLog will be using Google Cloud Run to host the backend services for the REST API.

2.5 Summary

In summary, Chapter 2 offered a comprehensive breakdown of the technologies and components that are the foundation of PennyLog. Furthermore, an analysis was conducted to determine the limitations of existing methods, which provided insights into the distinctive additions and enhancements that PennyLog can implement.

CHAPTER 3: METHODOLOGY

3.1 Introduction

Chapter 3 discusses PennyLog's exploration and planning phase. An online survey is conducted to gather user requirements and preferences, which are then used to create the system's architecture and design.The Exploration phase involves gathering user and system requirements through an online survey aimed at UNIMAS students. Functional requirements, Non-functional, Hardware and software requirements are also identified.

PennyLog is then represented in a technical and organised manner through system design in the planning phase, which includes activity, use case, sequence, and class diagrams. A system prototype is also designed to provide a visual representation of workflows and user interface, allowing for iterative feedback and improvement.

3.2 Exploration Phase

To obtain requirements and insights for the creation of the PennyLog application, an online survey was conducted aiming mainly for university students at UNIMAS. The survey aims to gather information that would help in understanding user demographics, financial management habits, and expectations of users on a AI based expense tracker mobile application. The questionnaire contained three sections which are demographic information, financial management habits and challenges as well as preferences and expectations for an expense tracker

3.2.1 Section A: Demographic Information

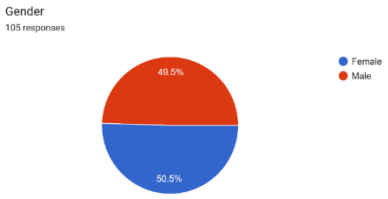


Figure 3.1: Pie Chart Gender

The pie chart in Figure 3.1 shows a near-equal distribution of responses, with females (49.5%) and males (50.5%).

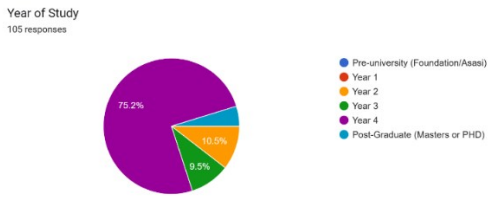


Figure 3.2: Pie Chart Year of Study

The pie chart in Figure 3.2 depicts the distribution of respondents at UNIMAS across several academic levels with 75.2% of respondents are in Year 4, indicating a strong representation of final-year students as the target user.

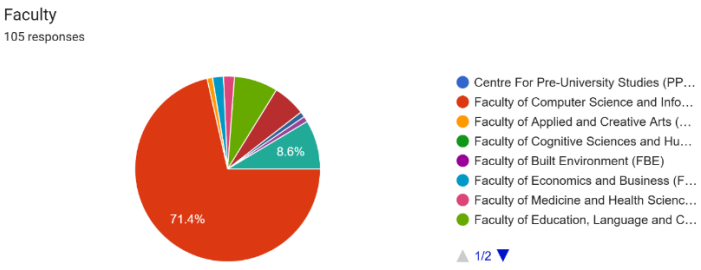


Figure 3.3: Faculty

Figure 3.3 illustrates the pie chart distribution of respondents across the various faculties of UNIMAS. The Faculty of Computer Science and Information Technology is the most significantly represented making up 71.4% of all responses.

3.2.2 Section B: Financial Management Habits and Challenges

Figure 3.4: Tracking of Expenses and Income

Figure 3.4 pie chart illustrates the regularity of respondents record their expenses and income. 27.6% respondence keep track on a weekly basis, 26.7% track them daily and 19% monthly demonstrating a sizable proportion of students take proactive financial management measures. A high proportion of 21%, track their accounts rarely, while 5% never do. These findings imply that, there is still a large opportunity to promote the benefits of constant tracking to others who track their finances occasionally or never.

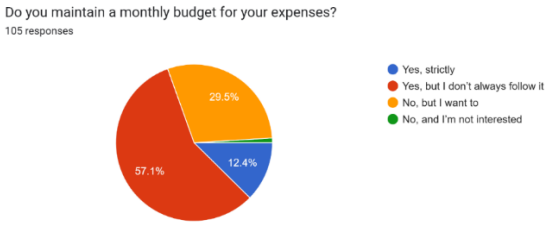


Figure 3.5: Monthly Budget Opinion

The pie chart in Figure 3.5 illustrates the responses of users on maintaining budget for expenses. Majority (57.1%) of respondents stated that they have a budget, but they don't always stick to it. This shows that strict methods of budgeting may not be appropriate for all the users. To acknowledge this, PennyLog will prioritise user freedom and flexibility.

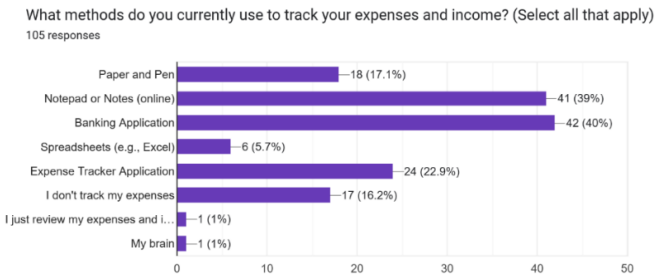


Figure 3.6: Method to Track Expenses and Income

The bar chart in Figure 3.6 illustrates the common method for tracking expenses and income. A majority with 40% uses banking application and 39% uses Notepad or notes. Only 22.9% respondents use an expense tracking application with a large proportion of 17.1% still uses paper and pen. Notably, 16.2% of respondents do not track their costs. The result of this survey question highlights the need for a standardised financial management tool due to the limitations that each method provide in which PennyLog can implement and overcome.

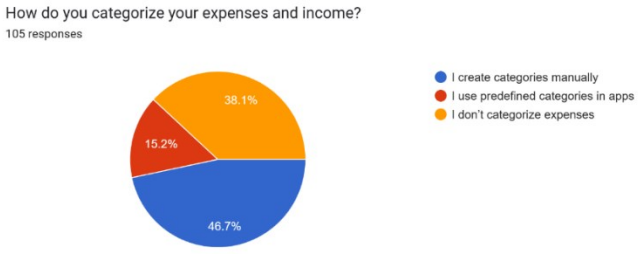


Figure 3.7: Method to Categorise Expenses and Income

The pie chart in Figure 3.7 reveals that 46.7% of respondents manually create their own categories, 38.1% do not categorise and 15.2% uses predefined categories in the application. This highlights the importance for PennyLog to have a flexible and user-friendly categorisation feature to accommodate a variety of preferences among the users such as manual for flexibility and automation using NLP to save time.

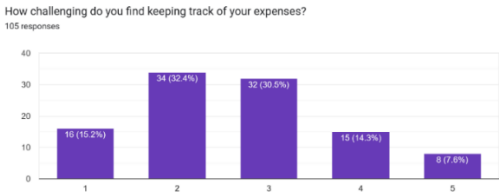


Figure 3.8: Challenges Tracking Expenses

Figure 3.8 shows that a significant percentage of respondents find keeping track of their expenses challenging, with 15.2% rating it as "Very Challenging," 32.4% rating it as "Challenging," and 30.5% rating it as "Average." This shows that many students struggle with expense tracking for a variety of reasons, therefore PennyLog should aim for a user-friendly mobile application with proper filtering.

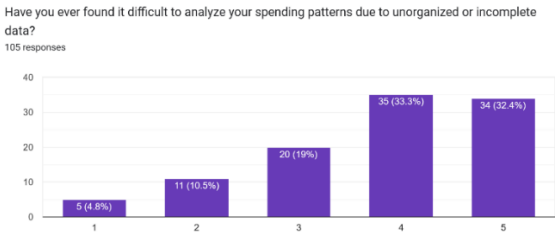


Figure 3.9: Difficulty in Analysing Spending pattern

The bar chart in Figure 3.9 shows the difficulties for users in understanding spending trends due to unorganised or incomplete data. A significant number of respondents, 33.3% (level 5) and 32.4% (Level 4) find it to be very difficult and difficult respectively. This suggests that many students are unable to understand their spending habit due to lack of visualisation which PennyLog can include.

Figure 3.10: Primary Challenge in Managing Finance

The bar chart in Figure 3.10 shows that the primary challenges respondents experience while managing their finances are forgetting to log expenses and income (81%), misplacing receipts (52.4%), manual data input (52.4%) and time-consuming processes (51.4%). These findings highlight the necessity for PennyLog to incorporate advanced features that can solve these challenges.

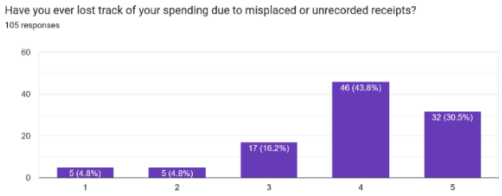


Figure 3.11: Lost Track of Spendings

In Figure 3.11, the bar chart shows the frequency of respondents lost track of their spending due to misplaced or unrecorded receipts. 43.8% respondents have frequently encountered this issue, 30.5% occasionally, 16.5% regularly, whereas 4.8% rarely or never. These findings highlight the need for PennyLog to include features that reduce dependency on physical receipts.

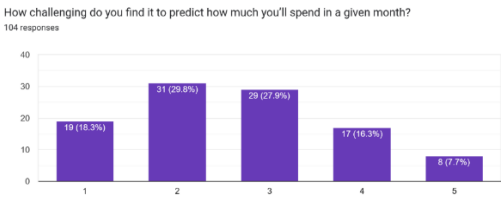


Figure 3.12: Challenge to Predict Spendings

The bar chart in Figure 3.12 shows that predicting monthly spending is a challenge for many students with 18.3% finds it Very Challenging, 29.8% "Challenging," and 27.9% as "Moderately Challenging". Therefore, there is a need for PennyLog to include the expense prediction with budget optimisation feature to help a vast majority of the students.

Figure 3.13: Personalising Financial Goal

Figure 3.13 illustrate a bar chart to determine the importance of having personalised financial goal. A significant percentage, 50.5%, believe it is very important. The results of this analysis suggest that PennyLog users finds it important to be able to set and track personalised financial goals.

3.2.3 Section C: Preferences and Expectations for an Expense Tracker

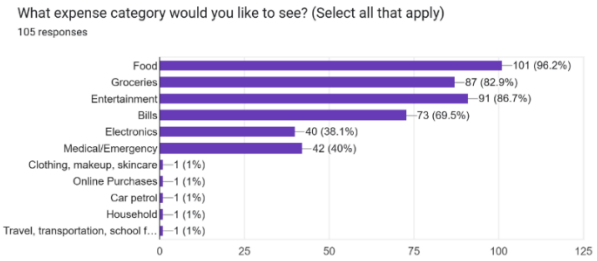


Figure 3.14: Preference Of Expenses Category

The bar chart in Figure 3.14 shows the expenses categories that respondents prefer to be tracked in PennyLog. A significant majority of respondents chose Food with 96.2%, Groceries with 82.9%, Entertainment with 86.7%, Bills with 69.5% and

Medical/Emergency with 40%. These findings suggest that students prioritise tracking spending linked to necessities and social activities.

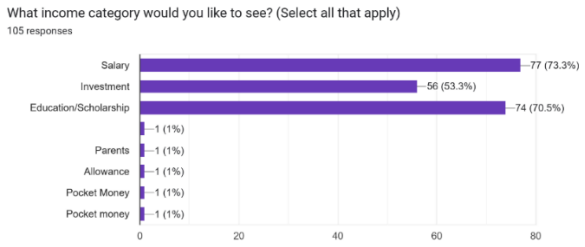


Figure 3.15: Preference of Income Category

The bar chart in Figure 3.15 shows the income categories that respondents prefer to be tracked in PennyLog. 73.3% selected Salary, Education/Scholarship with 70.5, Investment with 53.3%. These areas are significant sources of income for students, influencing their budgeting, savings, and long-term financial planning.

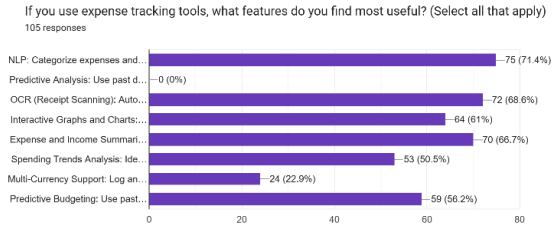


Figure 3.16: Preferred Features

The bar chart in Figure 3.16 illustrates the preferred feature chosen by respondents. NLP 71.4%, OCR (Receipt Scanning) 68.6%, Interactive Graphs and Charts 61%, Predictive Budgeting 56.2% and Expense and Income Summaries 66.7%. These findings highlight the necessity of implementing advanced features like NLP, OCR, interactive visualisations, and predictive budgeting capabilities into PennyLog.

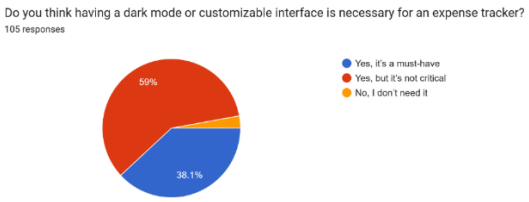


Figure 3.17: Customisable Interface

Figure 3.17 shows the importance of having a dark mode or customisable interface in an expense tracker. 59%, feels Yes but it's not critical, 38.1% Yes, it's a must-have and only 2.9% believe that No, I don't need it. The analysis reveals that while it may not be the most important feature it is preferred by a significant percentage of the target users.

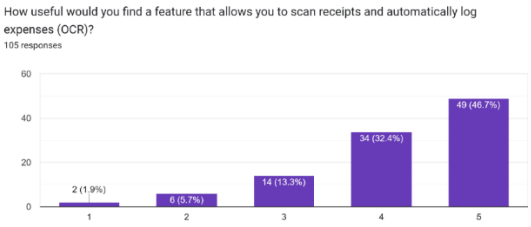


Figure 3.18: OCR Functionality

The bar chart in Figure 3.18 illustrates the usefulness of OCR functionality for users. 46.7% finds it extremely useful, 32.4% Very Useful, 13.3% Somewhat Useful, 5.7% slightly useful and 1.9% find it not useful. These results strongly suggest that users prefer having this functionality as it is useful.

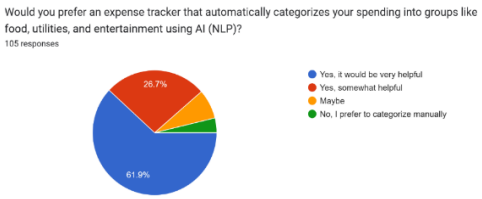


Figure 3.19: NLP Functionality

The pie chart in Figure 3.19 displays the preference for an expense tracker that automatically categorises expenses and income. Majority of respondents, 61.9%, preferred this feature as it is very helpful. The results of this analysis strongly indicate a large demand for AI-powered categorisation features.

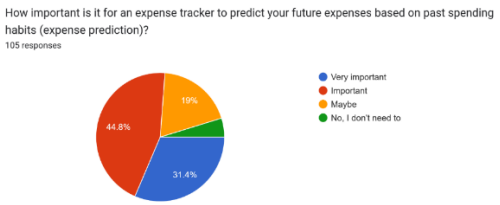


Figure 3.20: Importance of Prediction

The pie chart in Figure 3.20 represents users' opinion on importance of expense prediction feature. Majority of respondents, 44.8%, consider it as Very Important. These findings indicate a high demand for predictive analysis.

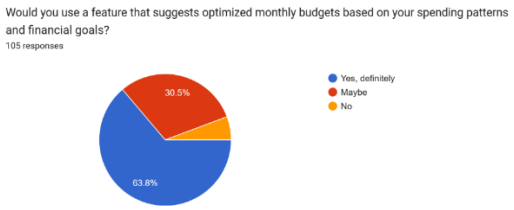


Figure 3.21: Optimisation Feature

The pie chart in Figure 3.21 illustrates respondents' willingness to use the expense prediction with budget optimisation feature. Majority of respondents 63.8% chose Yes, definitely. The results of this analysis strongly suggest that users are willing to try out the Expense prediction with budget optimisation feature that PennyLog wants to offer.

3.2.4 System Requirements

The PennyLog project is being developed as a mobile app intended for Android devices.

Functional Requirements

Functional requirements are the specific feature, capabilities, functionality, and the behaviour in which the system exhibit to fulfil the objective and requirements of users. They outline the actions or processes that the system carries out in response to user inputs and specify what the system should do to achieve a final goal (Alqurashi & Alawneh, 2024). Table 3.1 shows the functional requirements of PennyLog.

Table 3.1: Functional Requirement

No	Functional Requirement	Explanation
1	User Authentication	• Uses Device UUID so that users do not need registration and credential to use the application.
2	Customisation	• Users will be able to customise and manage the application settings such as theme, language and currencies.
3	Transaction Management and Data Entry	• Users will be able to manually enter transactions such as expenses and income. • Users will be able to automate data entry by using the receipt scanning capabilities which will enable the OCR capabilities to automatically extract transaction details from their receipts. • Users will be able to manage transactions such as view, edit, delete and save. • User will be able to filter and access their transactions.
4	Data Storing and Retrieval	• The application can save all the transaction data securely and accurately in firebase
4	Categorisation	• The application able to categorise transaction automatically using NLP • Users able to manually assign categories to their transaction from the list of categories available.
5	Receipt Scanning	• Utilise OCR capabilities to extract transaction details such as date, time, total, category, payment method from receipts.

		• Users will be able to edit extracted information before saving.
6	Expense Prediction with Budget Optimisation	• Users will be provided budget optimisation recommendation with expenses breakdown by category. • User will be able to input a specific budget to generate a new personalised prediction
7	Visualisation	• Display graphical insights, such as line graph, pie charts and tables, that highlight expense trends and category breakdowns. • Display overall financial status total.

Non-Functional Requirements

Non-functional requirements, which is the opposite of functional requirements, specify how the system should act or perform under specific scenarios. These standards ensure that the system is dependable, efficient, usable, and secure (Alqurashi & Alawneh, 2024). Table **3.2** shows the functional requirements of PennyLog.

Table 3.2: Non-functional Requirement

No	Non-Functional Requirement
1	PennyLog will ensure there is a reliable filtering, processing, storage, and synchronisation of transaction data from both the frontend and backend.
2	PennyLog will offer a user-friendly interface to ensure easier financial management for users.
3	PennyLog will provide high reliability for text extraction using OCR and categorisation using NLP.
4	PennyLog shall ensure operation of the application have minimal disruptions by providing high system availability.
5	PennyLog will provide a simple and uniform design to ensure easy navigation across the application functionalities.
6	Using charts and tables, PennyLog will present reliable and clear visualisations of spending trends and overviews.

Hardware and Software Requirements

Table 3.3: Hardware Requirement

No	Hardware	Requirements
1	Processor	Multicore Processor for a faster model training
2	RAM	16 GB or higher to do multitasking and to handle OCR as well as NLP workloads
3	Storage	512GB SSD to enable faster read and write speed as well as for backup or storing large dataset
4	Graphics	Integrated Graphics for basic UI development such as Intel UHD Graphics Dedicated GPU for rendering advanced UI or AI training such as NVIDIA GTX 1050 or equivalent
5	Operating System	Windows 10 or later
6	Display	Screen resolution of 1920x1080 or higher
7	Network	Basic network adapter with reliable internet access

Table 3.4: Software Requirement

No	Software	Requirements
a)	Integrated Development Environment (IDE)	Visual Studio Code and Android Studio
b)	Documentation and Gantt Chart	Microsoft 365
c)	Version Control	Github
d)	Prototyping	Figma
e)	Diagram and designing	draw.io
f)	Firebase	Latest Stable Version
g)	Pytesseract	Latest Stable Version
h)	Tensorflow and Keras	Latest Stable Version
i)	SentenceTransformer("all-MiniLM-L6-v2")	Latest Stable Version
j)	Pandas	Latest Stable Version
k)	NumPy	Latest Stable Version
l)	Scikit-Learn	Latest Stable Version
m)	FastAPI	Latest Stable Version
n)	Uvicorn	Latest Stable Version
o)	Google Cloud Run	Latest Stable Version

3.3 Planning Phase

This phase involves creating both logical architectures and physical representations of the system (Barr & Noble, 2004). Diagrams and specifications which define the system behaviour are part of the logical design, whereas a prototype is the physical design that will focus on how user will engage with the system.

3.3.1 Overall System Architecture Design

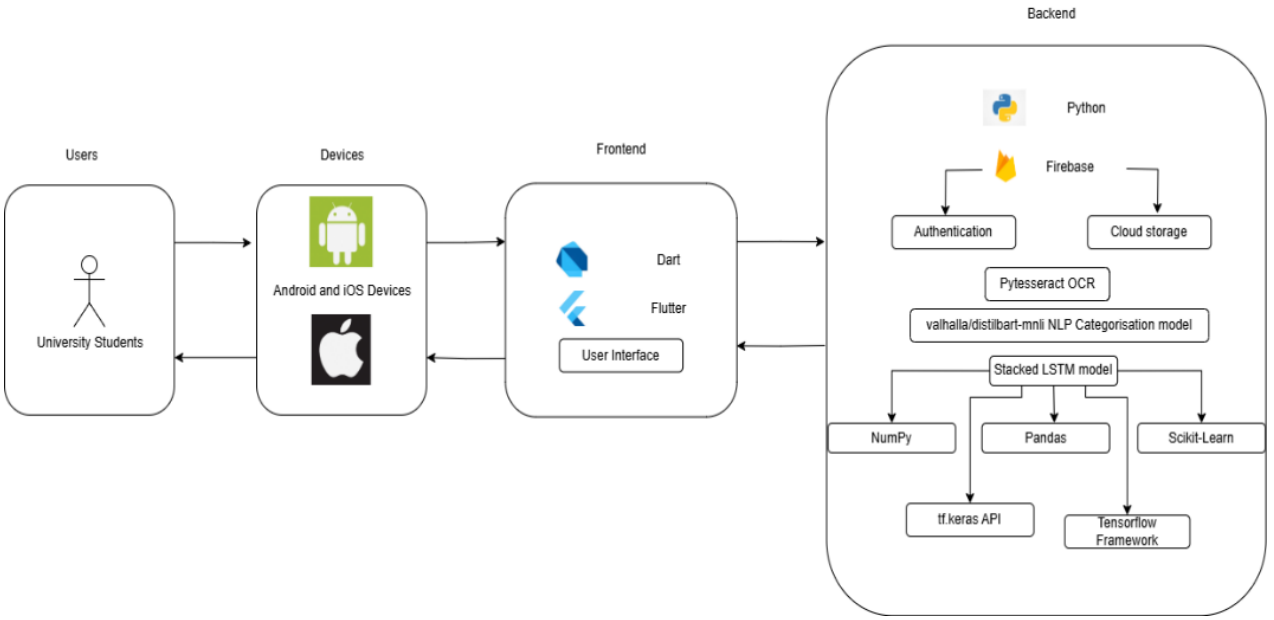


Figure 3.22: Overall System Architecture Design

3.3.2 Use Case Diagram

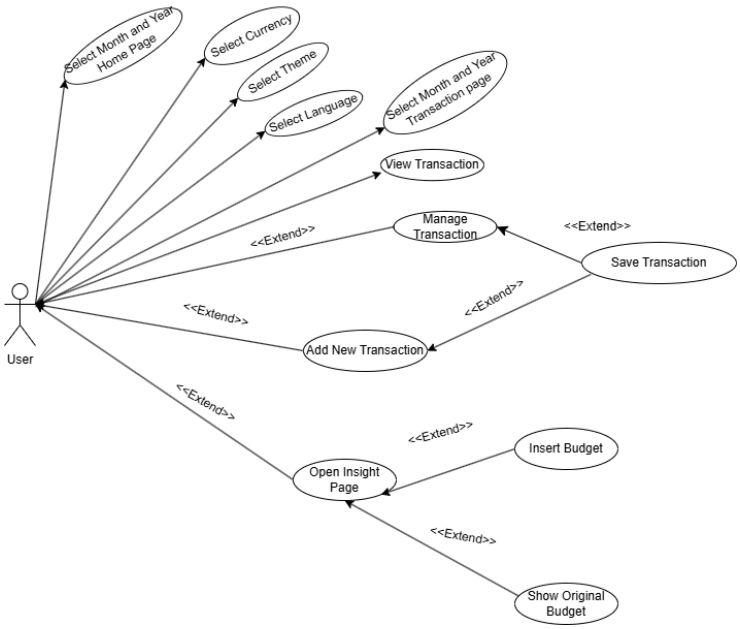


Figure 3.23: Use Case Diagram

Use Case Specification

The use case specification is included in the appendix for the purpose of providing a reference. Refer to Appendix B for any further detail.

3.3.3 Activity Diagram

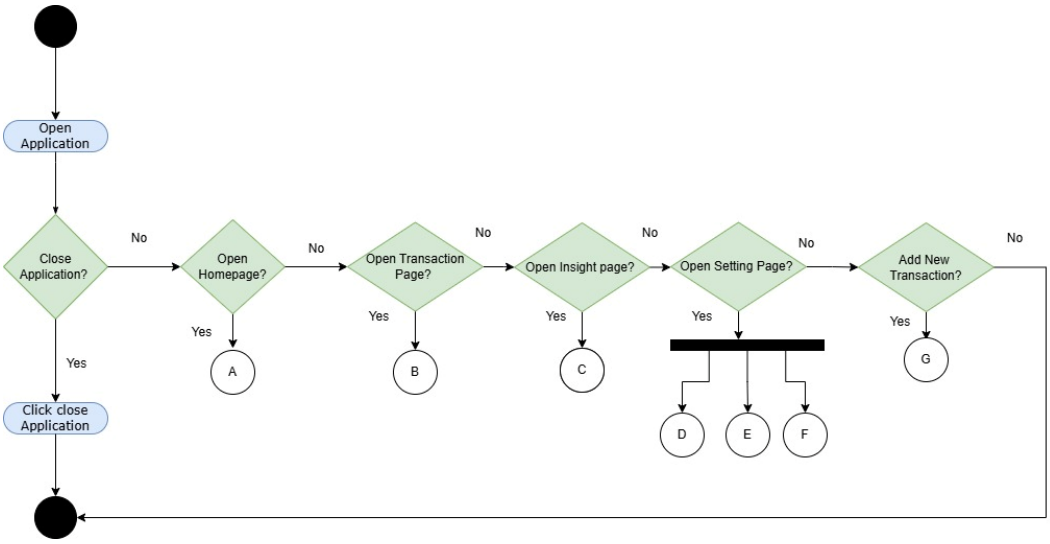


Figure 3.24: Overall Activity Diagram

Home Page

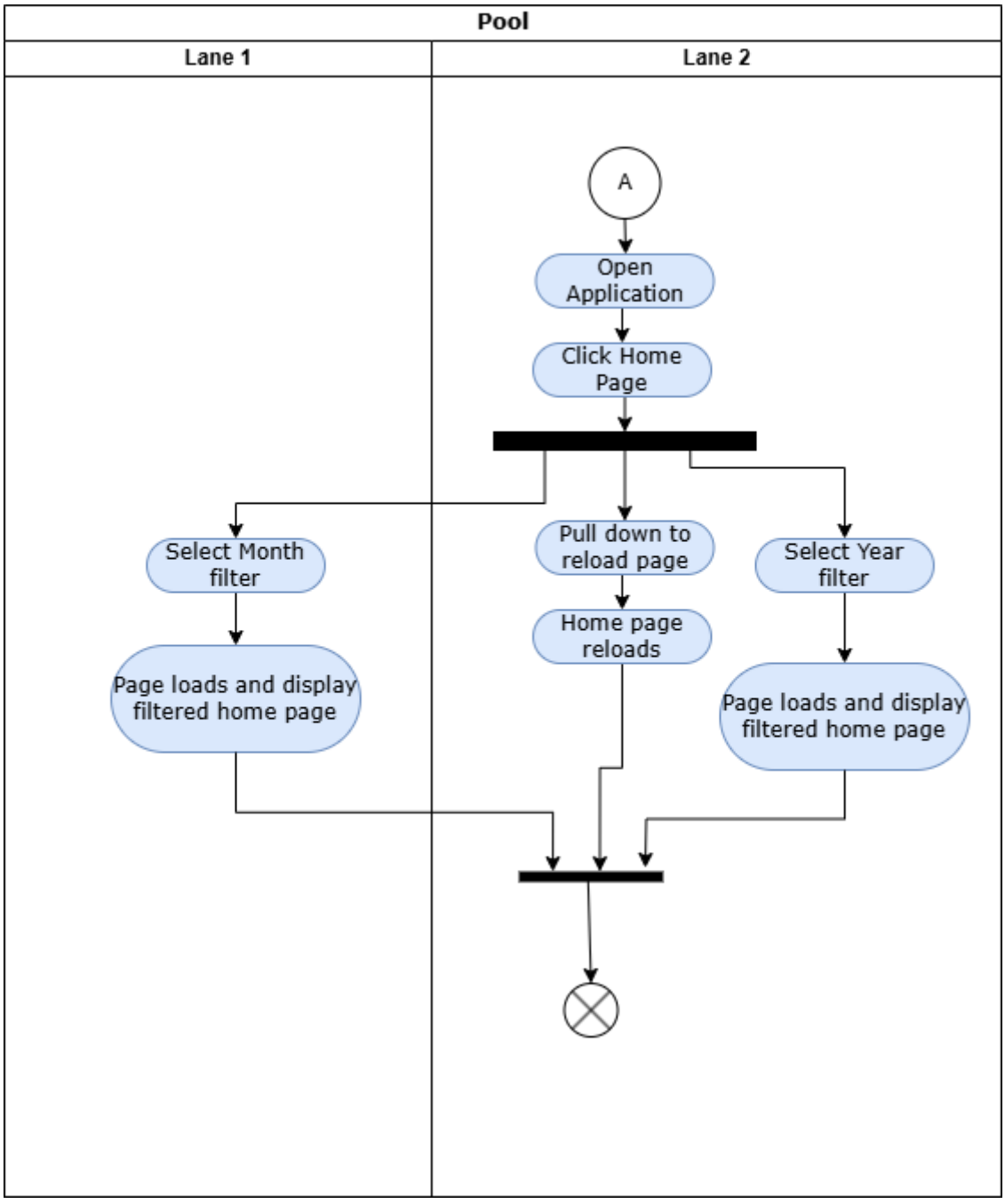


Figure 3.25: Activity Diagram Home Page

Transaction Page

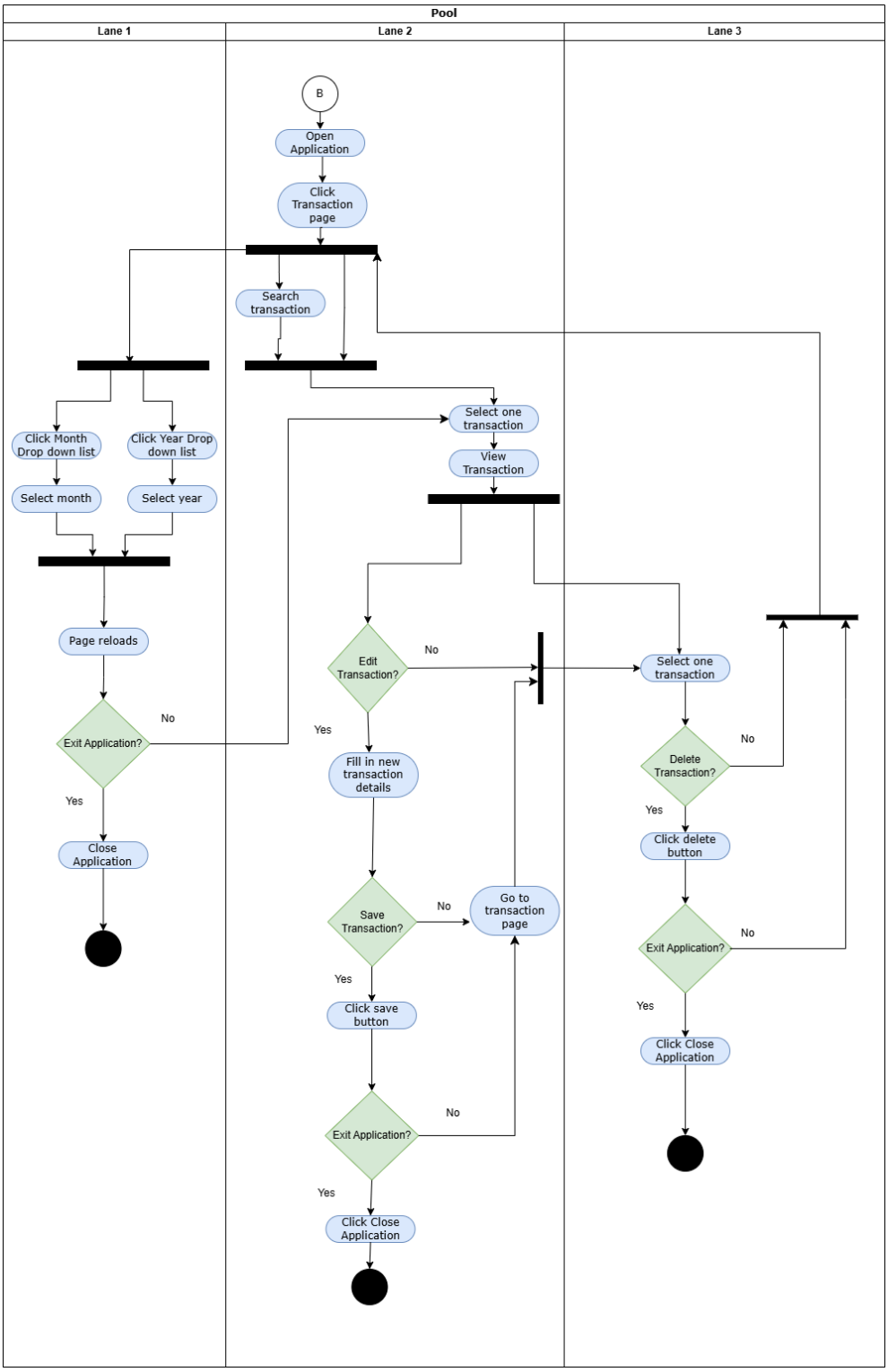


Figure 3.26: Activity Diagram Transaction Page

Add New Transaction Page

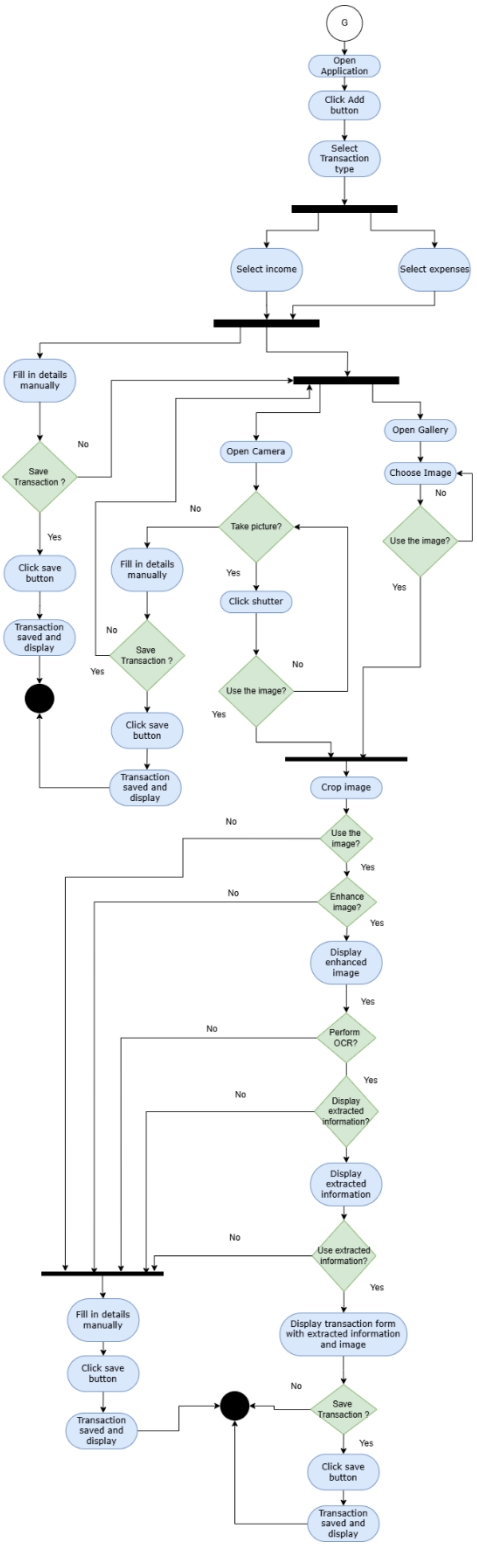


Figure 3.27: Activity Diagram Add New Transaction

Insight Page

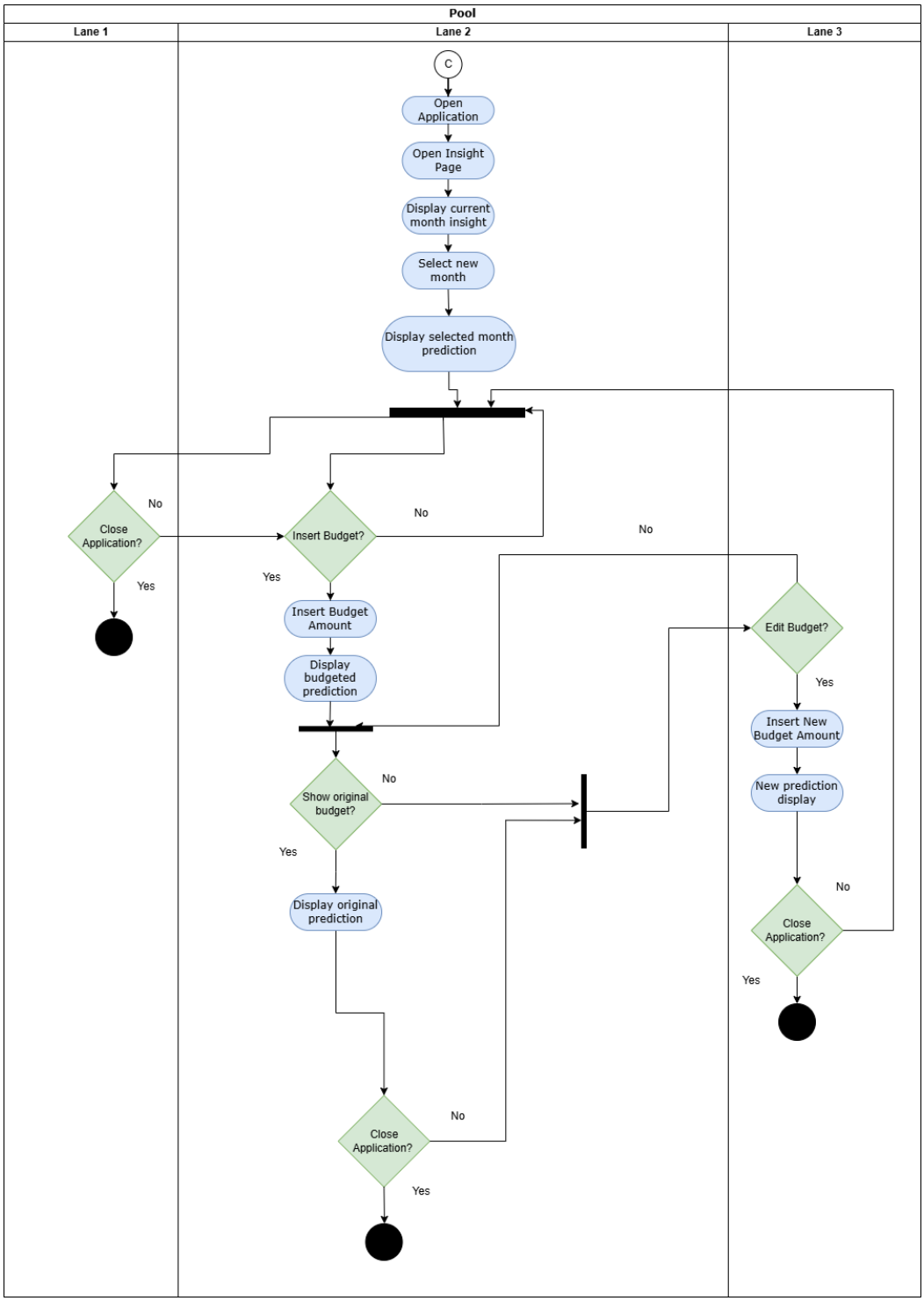


Figure 3.28: Activity Diagram Insight Page

Setting Page

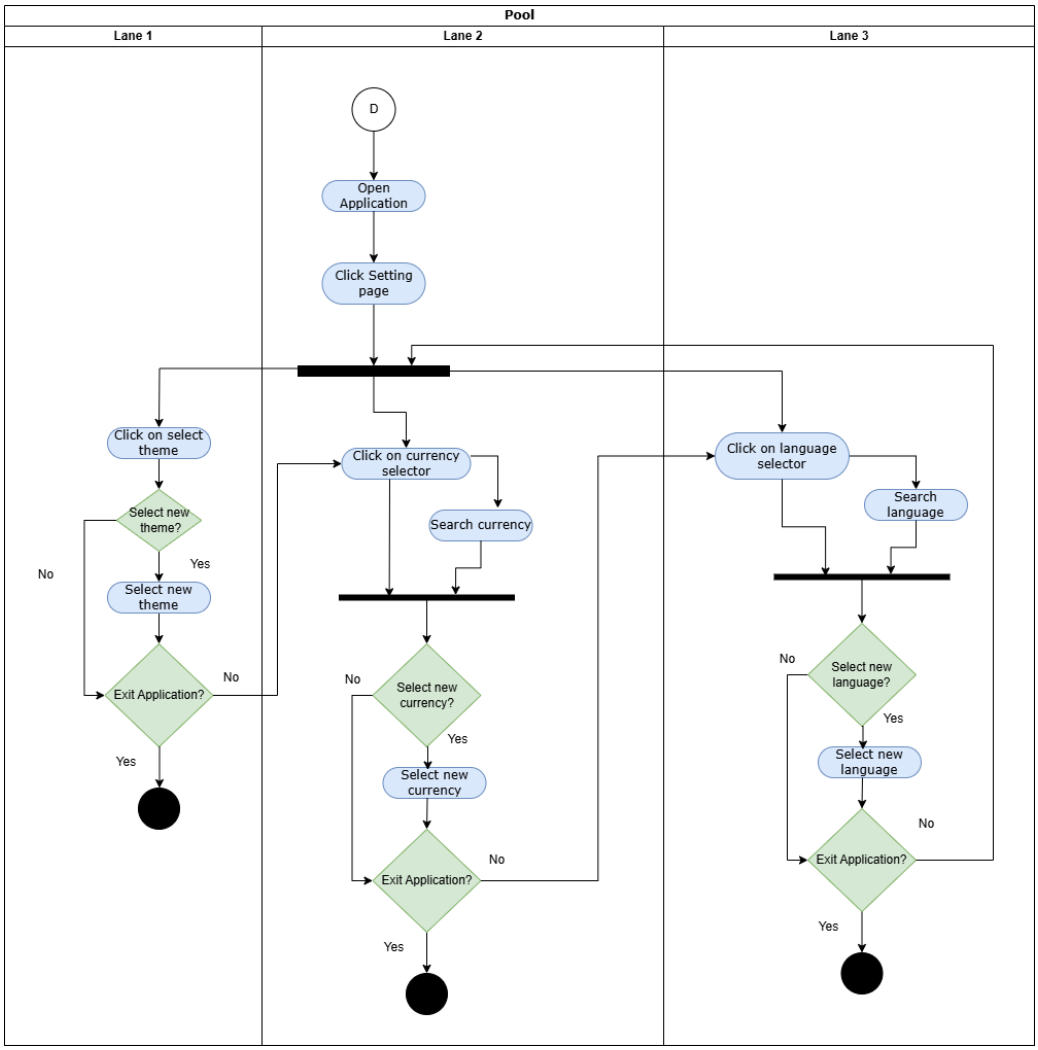


Figure 3.29:Activity Diagram Settings Page

3.3.4 Sequence Diagram

Home Page

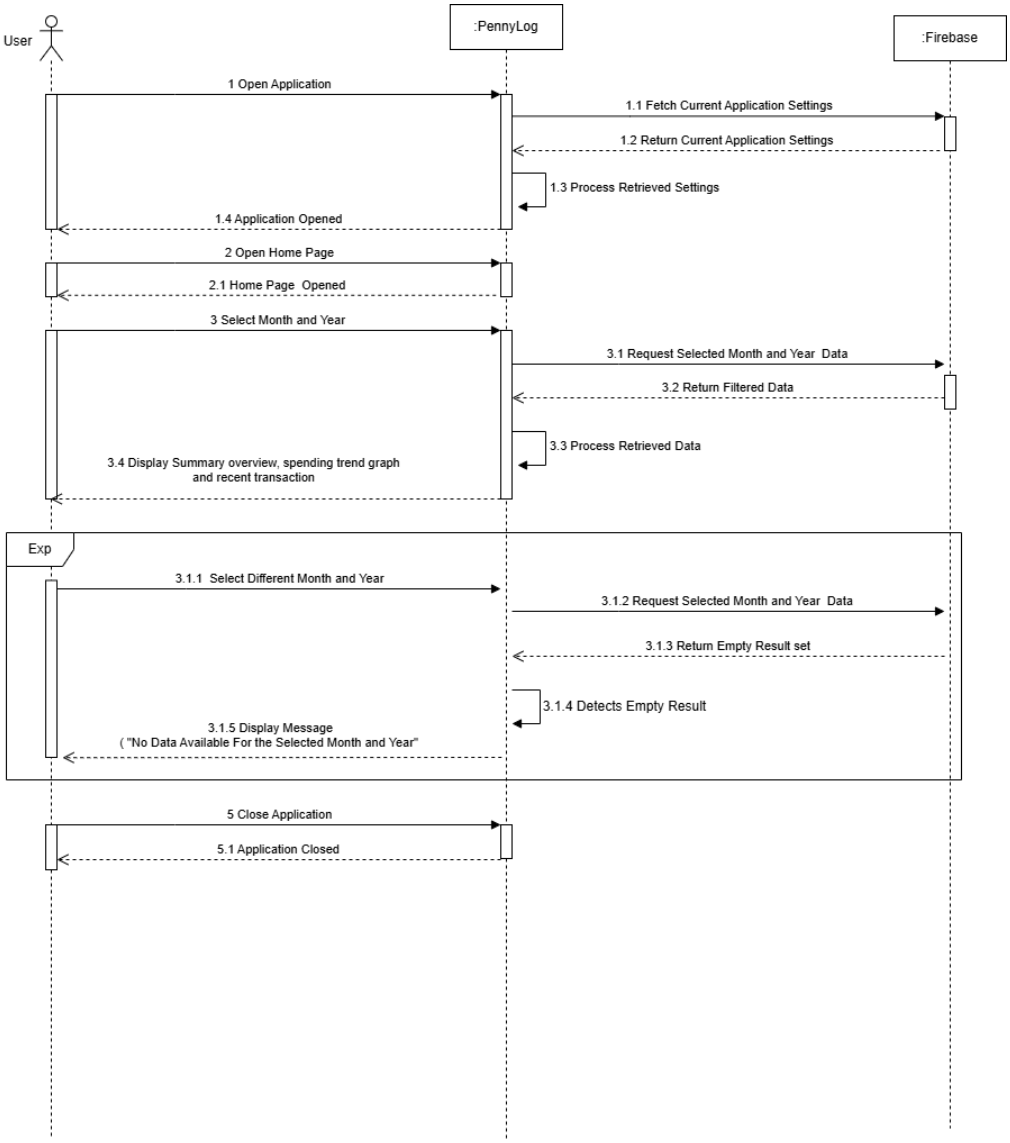


Figure 3.30: Home Page Sequence Diagram

Transaction Page

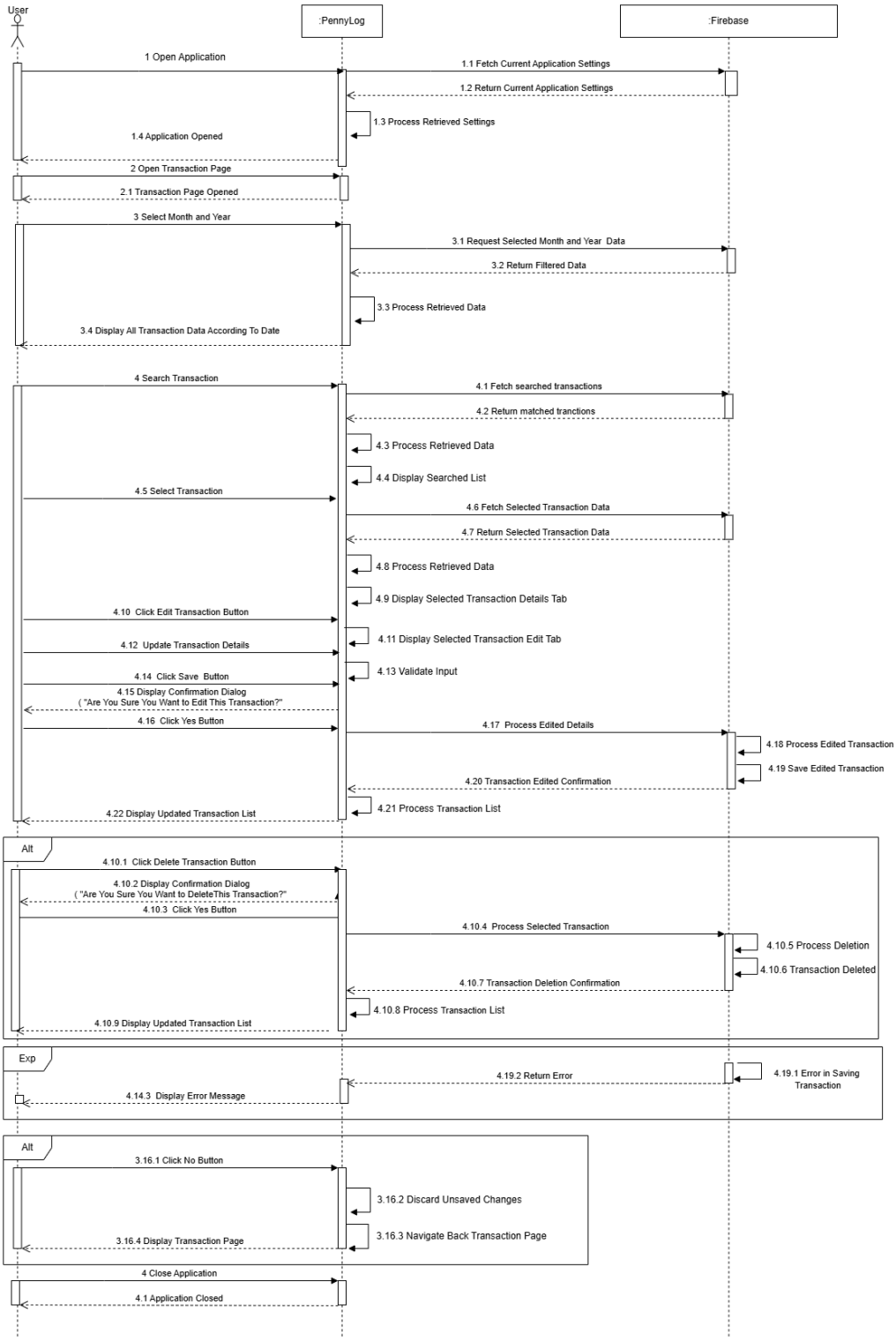


Figure 3.31: Transaction Page Sequence Diagram

Add New Transaction

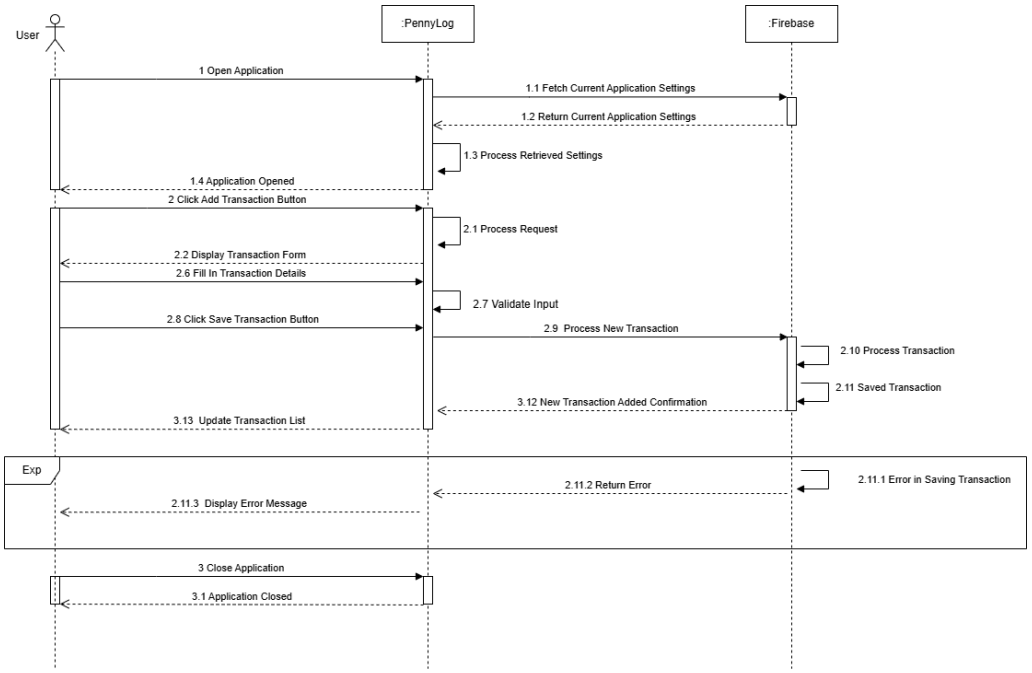


Figure 3.32: Add Transaction Manually Sequence Diagram

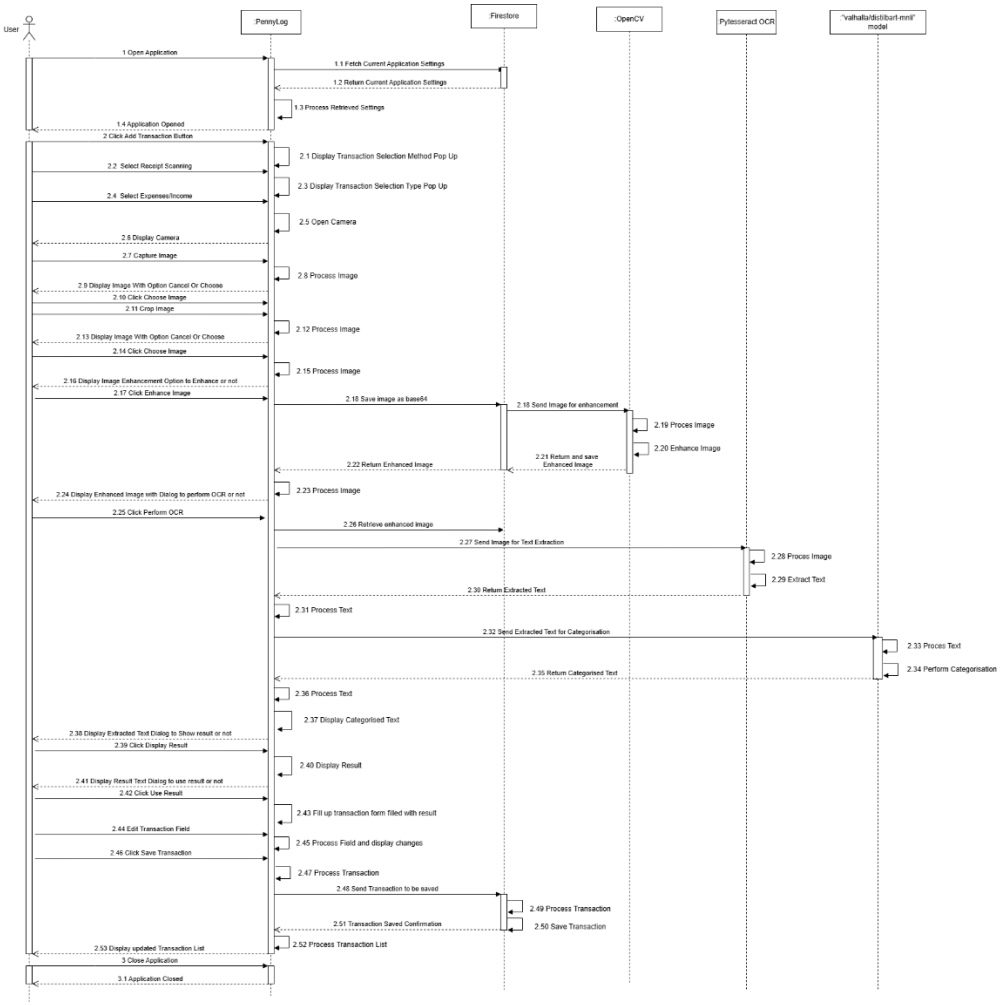
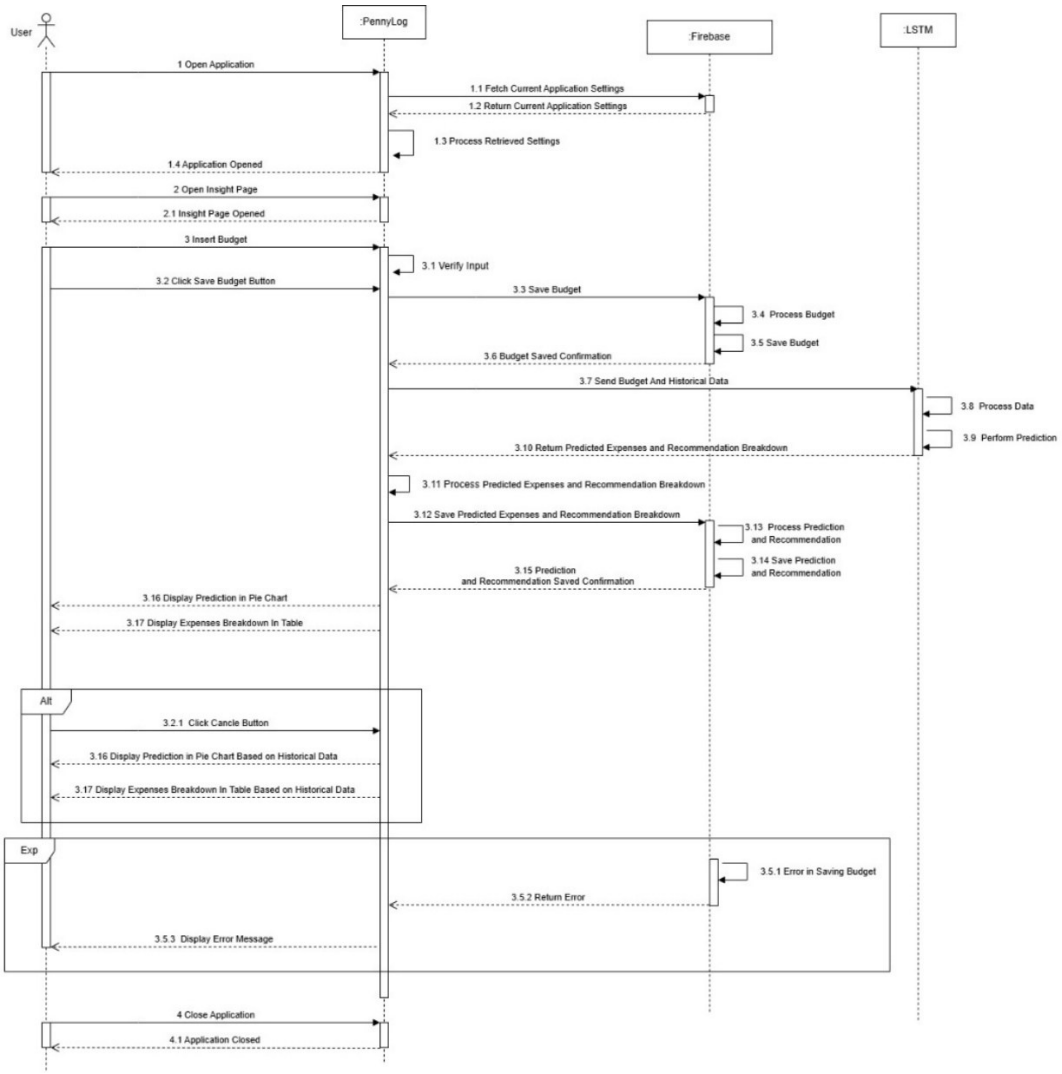


Figure 3.33: Add Transaction Using OCR Sequence Diagram

Insight Page



Insight Page Sequence Diagram

Setting Page

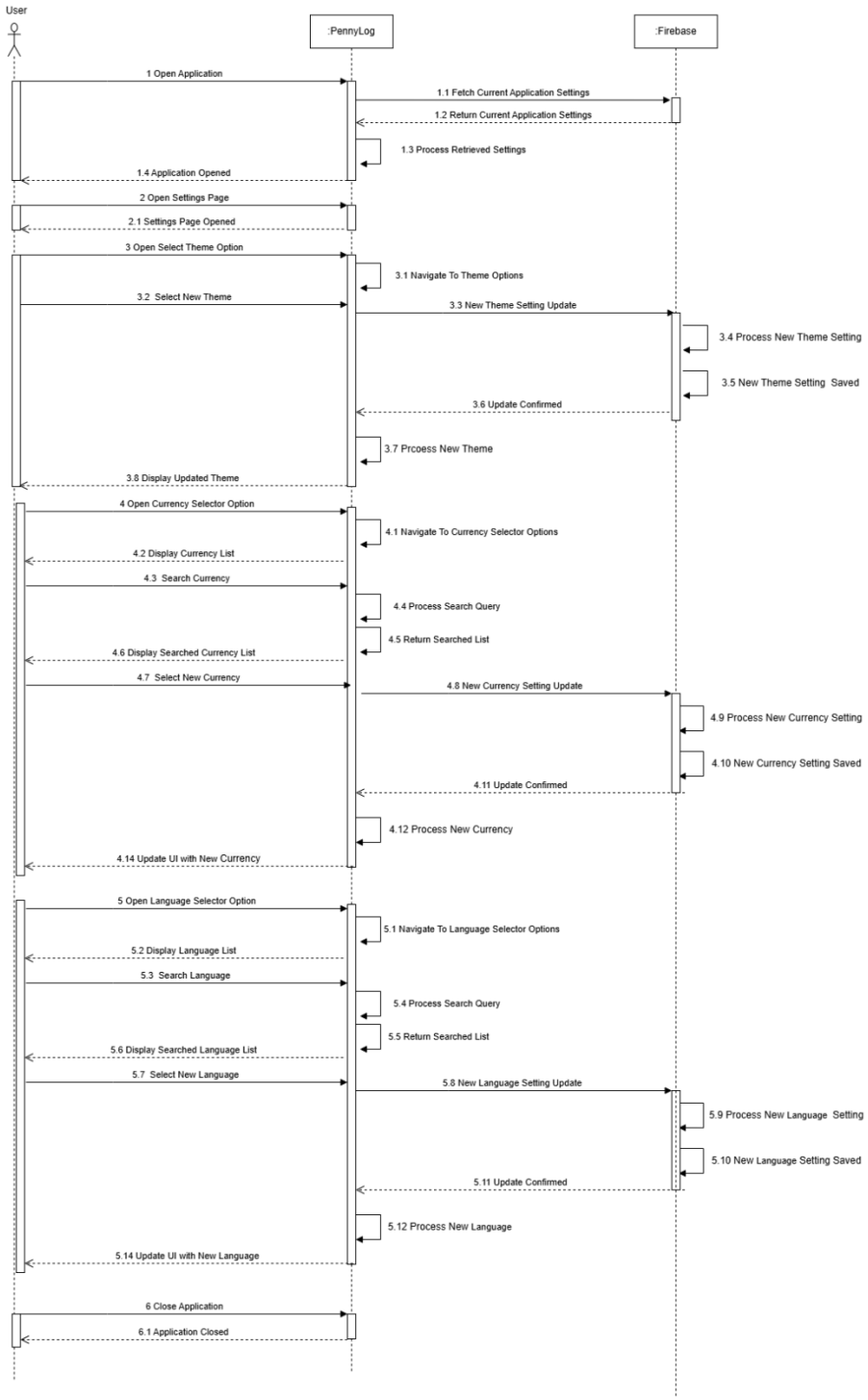


Figure 3.34: Settings Sequence Diagram

3.3.5 Class Diagram

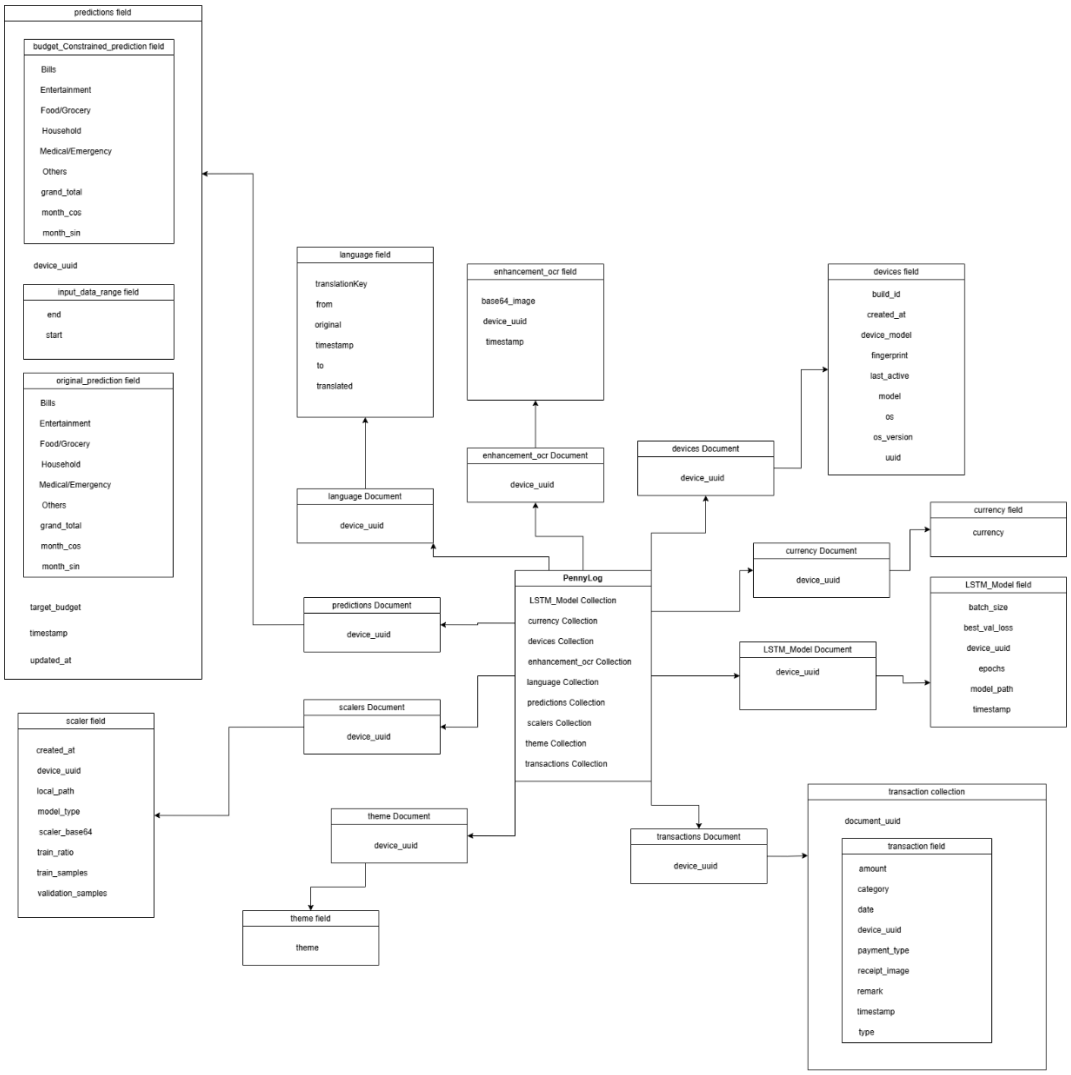


Figure 3.35: Class Diagram

3.3.6 Prototype

Home Page

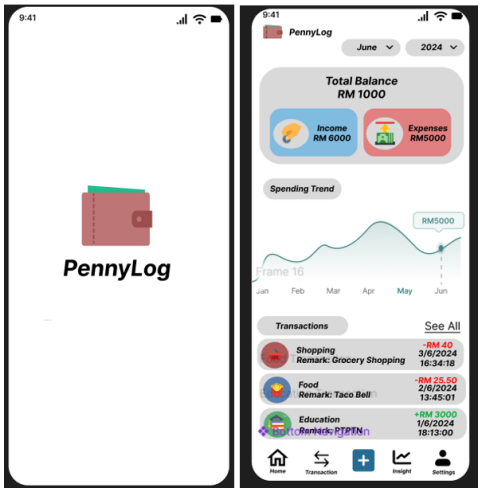


Figure 3.36: a) Launching Page and b) Home Page Prototype

This is the first screen that users see when they open the PennyLog the splash screen with a PennyLog logo and word. There will be no login and sign-up page as users are not requiring an account to use this application.

After launching, users are greeted with the landing page which is the home page. At the top, there is a PennyLog logo accompanied by two drop downs to filter the month and year. Below it then has the users overall financial balances with their "Total Balance", "Total Income" and "Total Expenses" for the selected month and year. Accompanying the financial balance is the spending trend graph that provide visual representation of the spending trend over the past 12 month. At the lower half, there is a recent transaction section along with a see all button that navigate user to the transaction page. There is also a bottom navigation bar that can navigate to the Home, Transaction, Add New Transaction, Insight and the Setting Page.

Transaction Page

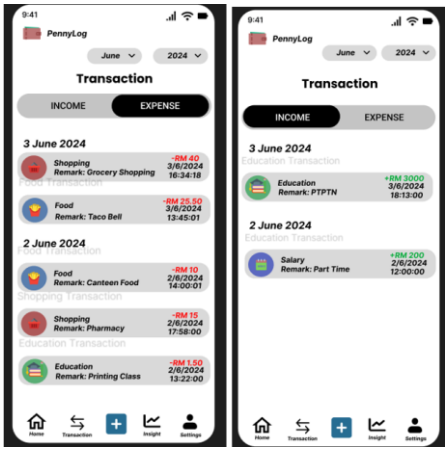


Figure 3.37: a) Expense Transaction and b) Income Transaction Prototype

Next, is the transaction page. Similarly to the home page, there is a PennyLog Logo with the month and year drop down for filtering. For this page, it is divided into section, where user can toggle between expense and income transaction type. All the list of transaction for both expense and income are listed in descending order. The transactions are grouped and displayed by the date of transaction. Each transaction is displayed alongside its date and time, category, amount and remark.

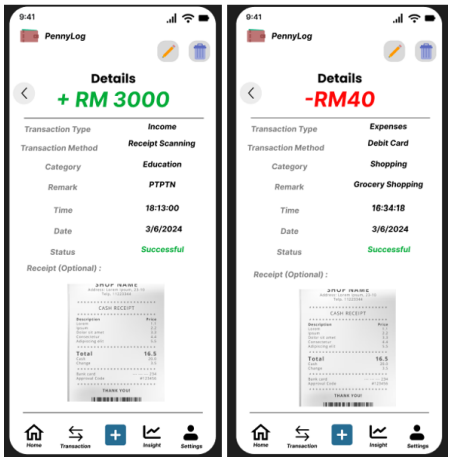


Figure 3.38: a) Successful and b) Failed Transaction Details Prototype

Users can view expended details of each transaction by clicking on it. A pop up alongside its details will be shown. The transaction is displayed in a structured manner that details each of the transaction information such as amount, transaction

type, transaction method, category, remark, time, date, status and receipt image. If the transaction is an income, it will be a positive value in green whereas if it is an expense, it will be a negative value in red. If the transaction is successful, the status will be remarked as “Successful” and in green, meanwhile if the transaction is unsuccessful, the status will be remarked as “unsuccessful” and in red. At the top of the page is also two buttons which allow user options to either edit the transaction or delete the transaction. There is also a left button for users to navigate back to the previous screen which is the transaction list.

Add New Transaction

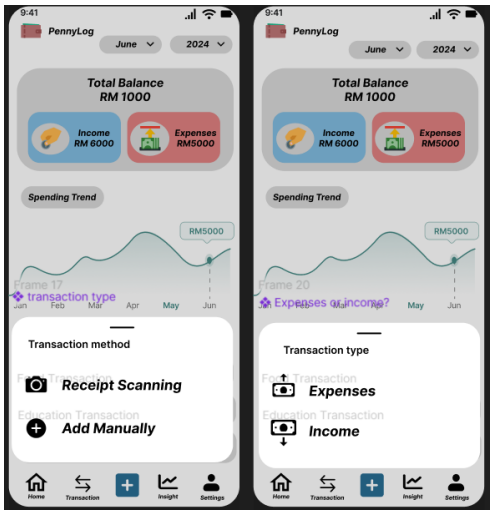


Figure 3.39: a) Transaction Method and b) Transaction Type Pop Up Prototype

To add new transactions, there is a prominent “+” add button at the bottom navigation bar. Regardless of where and when the “+” button is clicked, it will trigger pop ups which will prompt user to choose the transaction method and transaction type.

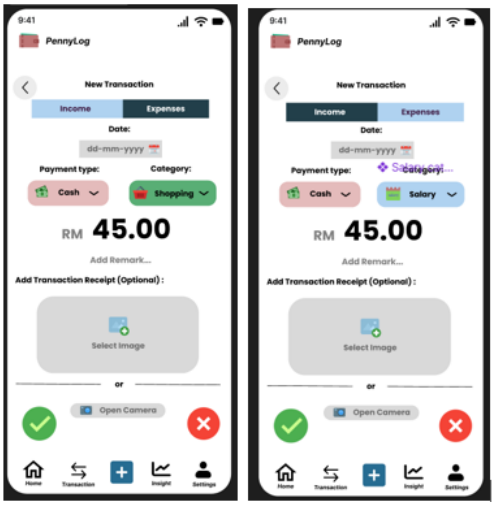


Figure 3.40: a) Income and b) Expense New Transaction Manual Prototype

If the user chose method as "Add Manually", they will be navigated to a transaction form. The page is properly divided depending on the transaction type at the top that labels "Income" and "Expenses". The toggle will allow user to easily switch the transaction type. Users will then need to fill in essential fields such as date, payment type, category, amount, remark and allow user to capture or upload receipt to be attached together. User can save the transaction by clicking on the green button and to cancel click on the red. There is a left button at the top for user to navigate back to the previous page.

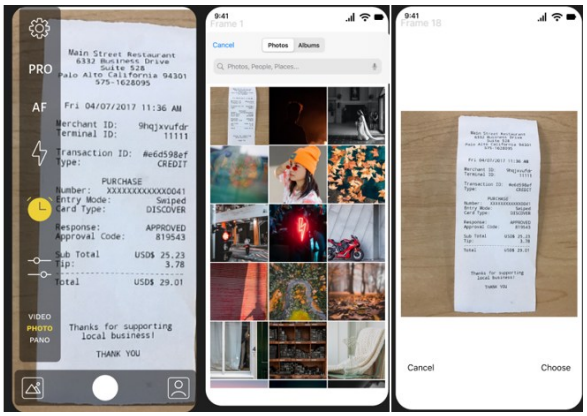


Figure 3.41: a) Receipt Scanning, b) Receipt Selection and c) photo selection Prototype

If user chose “Receipt Scanning” option as the transaction method, the application would utilise the device’s camera to capture the image of receipt. User can also use pre-existing image from their device by clicking the gallery icon on the bottom left. Once the user has finalised the image, the application will prompt user to choose the image or cancel it. If user cancel it, they will be navigated back to the previous page, if they choose to upload the image, the application will utilise the OCR technique to extract data, and NLP will be used to categorise the necessary information automatically at its dedicated field.

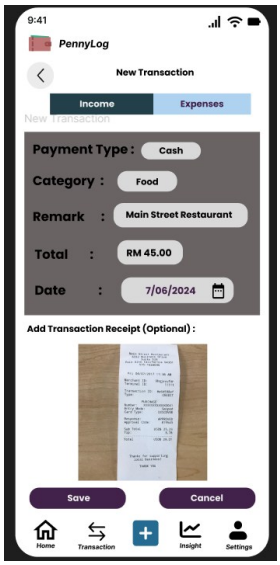


Figure 3.42:Receipt Scanning Transaction Form

After extraction, the application will open a new page that has dedicated transaction field but now has been pre-filled with information that was extracted The information such as payment type, category, remark, total, date and the receipt that was chosen will be uploaded together with the transaction. User still able to edit the information if necessary. User can click save button to save transaction or cancel button to not add the transaction.

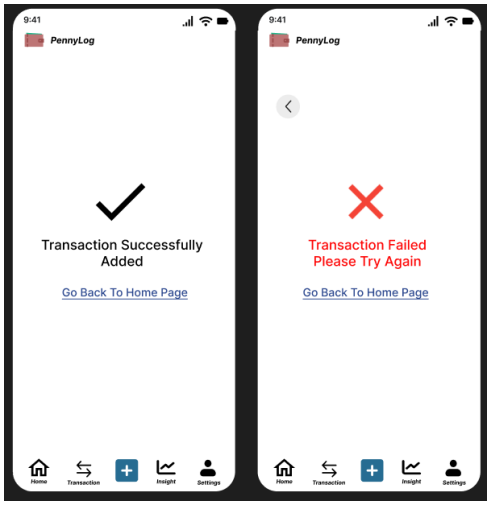


Figure 3.43: a) Successful and b) Failed Transaction Status Prototype

For the transaction that is newly being added, edited or deleted, there will be a validation screen shown if the transaction is added or if it fails. Then there will be a button to navigate back to the home screen.

Insight Page

Figure 3.44: a) Add Budget and b) Insight Generated On Insight Page Prototype

Next is the insight page where it will display financial insights with proper visualisation of an expense prediction with budget optimisation feature. The

application will take in the user budget, and predict future expenses based on past financial data. The predicted total expenses are divided by category will be displayed in pie chart form. There will also be a recommended expenses breakdown by category displayed in table form. User can add or remove the budget at the top by clicking on the check button or the cross button respectively. User can freely change the budget by clicking on the edit button at the top.

Setting Page

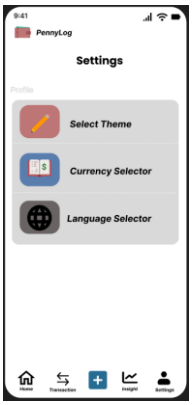


Figure 3.45: Settings Page Prototype

The last page for the application is the settings page. There are three choices that user can select which is “Select Theme”, “Currency Selector” and “Language Selector”.

Figure 3.46: Select Theme Page Prototype

The first selection is theme where user can choose between a dark or light theme which will automatically be applied to the whole application once chosen.

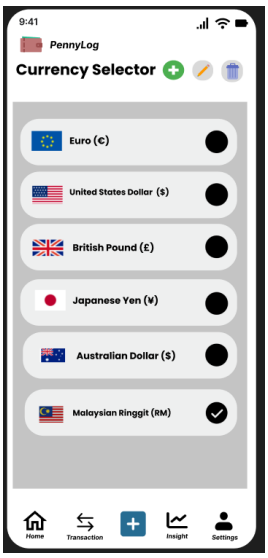


Figure 3.47: Currency Selector Page Prototype

The next settings choice is the currency selector. For the transaction that is being recorded in PennyLog, there is one currency that will be used as default. User can freely choose any currency from the available choices just by click on the currency. The application will reload and apply the selected currency. At the top there is 3 button which gives user three choices to manage the currency.

Figure 3.48: Language Selector Page Prototype

The last setting is the language selector. Like the currency, there is one language that will be used in the entire application which is by default. Users have the freedom to use any language simply by just clicking. At the top there is 3 button which gives user three choices to manage the language.

3.4 Iteration to Release Phase

PennyLog will be developed using an iterative process in Extreme Programming (XP), which allowed for continuous improvements based on user feedback to guarantee user satisfaction. Four iterations in total, each concentrating on improving one specific core module, will be carried out.

Developing PennyLog's basic features will be in the first iteration. Basic features such as validations, transaction displays, transaction filtering, data display for graphs, manage each individual transaction by viewing, editing, deleting and saving and changes as well changing settings.

The second iteration concentrated on automating expenses tracking using Natural Language Processing (NLP) for automatic expense categorisation and optical character recognition (OCR) for scanning receipts. To cut down on manual entry, the OCR module enabled users to scan or upload receipts and extract important information. There will also be integration with the backend to ensure model can be accessed, make request and display on the frontend the result.

The third iteration will be focusing on improving the OCR and NLP accuracy as well as add image processing. This phase also includes the implementation of RNN LSTM models for time series forecasting for the expense prediction

To improve usability, the fourth and final iteration concentrated on improving the interface design and user experience. Clearer financial insights were provided

via improved graph visualisations, which made it possible for users to efficiently monitor their spending patterns. To guarantee accessibility and easy navigation, the user interface (UI) will be modified in response to input from user acceptance test. The accuracy of expense forecasts and budget recommendations will also be improved by optimising the budget prediction model. Finally, final improvements were made to receipt scanning and categorisation, guaranteeing a flawless combination of OCR and NLP for transaction tracking.

The system will be developed incrementally during PennyLog's Iteration to Release phase by breaking down the features into more manageable, smaller iterations. Every iteration will concentrate on prioritising a particular set of features. Feedback will be continuously gathered, tested, fixed, and improvement will be made of the features. To ensure each feature and component in the application are working as expected, functional and user acceptance test will be conducted. The application with each feature that can be used will be made available for user review at the end of each iteration, providing opportunity to make necessary modifications and enhancements based on feedback from users before proceeding to the next one iteration (Tashtoush et al., 2021).

3.5 Productionising Phase

The main objective of the Productionising Phase is to guarantee that PennyLog is secure, reliable, and prepared for the final deployment. Final features and improvement will be made during this phase to ensure it fulfils all the requirements. During this phase, proper documentations will also be prioritised such as the full report, paper, source code, user manual and others. The system will

be ready for distribution through the sharing of APK files after it's been verified for its stability and dependability (Tashtoush et al., 2021).

3.6 Maintenance Phase

PennyLog will be constantly tested and maintained during the maintenance phase to ensure all functionality is working as expected without any error and can accommodate to the changing user requirements. In this stage, user-reported issues are fixed, the application is updated to stay compatible with new device operating systems, and any performance bottlenecks are optimised (Tashtoush et al., 2021).

3.7 Death Phase

When the development phase has completed and the system is reliable, error-free, and fully functional, PennyLog's lifecycle is completed in the Death Phase. During this period, there is no new features or updates required, and all bugs and issues have been resolved. The application needs little to no active maintenance from the development team because it is regarded as stable and reliable. Any last-minute documentation that is required will be completed during this phase (Tashtoush et al., 2021).

3.8 Summary

This chapter summarises the overall design and development plan of PennyLog. The application which was created to overcome the limitations of existing expense tracking systems by incorporating advanced features This chapter highlights the exploration and planning phases, which aims on developing a user friendly, functional, and scalable application.

CHAPTER 4: IMPLEMENTATION

4.1 Introduction

This chapter will cover the development of PennyLog in the Iteration to Release Phase of the eXtreme Programming (XP) methodology. The goal of this chapter is to ensure that the application is developed to achieve the objective of the project. The chapter will include details such as the setup, installation and configuration of the systems component that is needed to run the application in a testing and production environment. Besides that, the implementation will include the processes of connecting the frontend and backend of the application, integration of open-source libraries, external dependencies, as well as deployment of the application. There will also be a role-based access control (RBAC) to ensure all users able to interacts with the available features of the application. The chapter will also be down to cover the core application module that is designed for a specific functionality such as Home, Transaction, New Transaction, Insight and Settings. Each modules functionality and interactive design will be explained by sections to provide a better picture of how PennyLog is developed and will be operating.

4.2 Installation and Configuration of System's Component

To develop and implement PennyLog, there are dependencies that needs to be installed and configured in both frontend and backend. Table 4.1 shows the dependencies in pubspec.yaml file for the flutter application which is the frontend and Requirement.txt file for python-based dependencies for the backend.

Table 4.1: Dependencies

<table>
<tr>
<td>

```
dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.8
  firebase_core: ^3.12.1
  firebase_analytics: ^11.4.4
  cloud_firestore: ^5.6.5
  intl: ^0.20.2
  json_theme: ^8.0.0+2
  provider: ^6.1.2
  logger: ^2.5.0
  logging: ^1.3.0
  image_picker: ^1.0.7
  flutter_plugin_android_lifecycle: ^2.0.18
  fl_chart: ^1.0.0
  photo_manager: ^3.6.4
  photo_view: ^0.15.0
  permission_handler: ^12.0.0+1
  flutter_image_compress: ^2.4.0
  firebase_auth: ^5.5.2
  device_info_plus: ^11.4.0
  currency_picker: ^2.0.21
  http: ^1.3.0
  language_picker: ^0.4.5
  file_picker: ^10.1.9
  flutter_tesseract_ocr: ^0.4.28
  path_provider: ^2.1.5
  path: ^1.9.1
  http_parser: ^4.1.2
  crypto: ^3.0.6
  uuid: ^4.5.1
  shared_preferences: ^2.5.3
```

a) **Pubspec.yaml**

</td>
<td>

```
annotated-types==0.7.0
anyio==4.9.0
click==8.1.8
colorama==0.4.6
fastapi==0.115.12
filelock==3.18.0
fsspec==2025.3.2
h11==0.14.0
idna==3.10
Jinja2==3.1.6
MarkupSafe==3.0.2
mpmath==1.3.0
networkx==3.4.2
numpy>=1.26.0,<2.2.0
opencv-python==4.11.0.86
packaging==24.2
pillow==11.2.1
pydantic==2.11.3
pydantic_core==2.33.1
pytesseract==0.3.13
setuptools==78.1.0
sniffio==1.3.1
starlette==0.46.2
sympy==1.13.1
torch==2.6.0
typing-inspection==0.4.0
typing_extensions==4.13.2
uvicorn==0.34.1
torchvision==0.21.0
google-cloud-firestore==2.11.1
protobuf>=3.20.3,<6.0.0dev
pandas==2.2.3
scikit-learn== 1.6.1
tensorflow == 2.19.0
keras == 3.9.2
```

a) **Requirement.txt**

</td>
</tr>
<tr>
<td>Pubspec.yaml</td>
<td>Requirement.txt</td>
</tr>
<tr>
<td>

- Flutter configuration file
- Manges flutter packages and project assets

</td>
<td>

- Python dependencies file

</td>
</tr>
</table>

For us to be able to download the dependencies at the frontend, we will just need to paste the dependencies that can be found in the dart website for the dart packages. The website is https://pub.dev/ and by searching a package or dependencies name, we just by clicking the copy icon, the package are copied as shown in Figure 4.1.

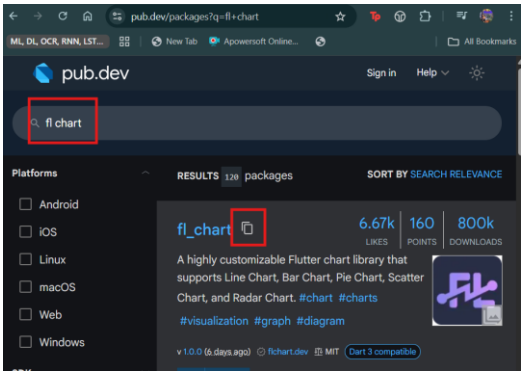


Figure 4.1: fl_chart package

After copying, we just need to paste the package in the pubspec.yaml file, save and run “flutter pub get” command in the terminal to download the packages into the project as shown in Figure 4.2.

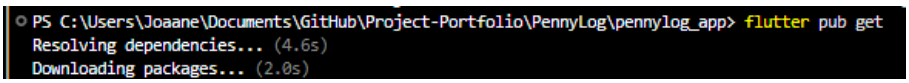


Figure 4.2: Flutter Pub Get Command

As for the backend, users need to run the command “pip install” along with the packages or dependencies name to download as shown in Figure 4.3.

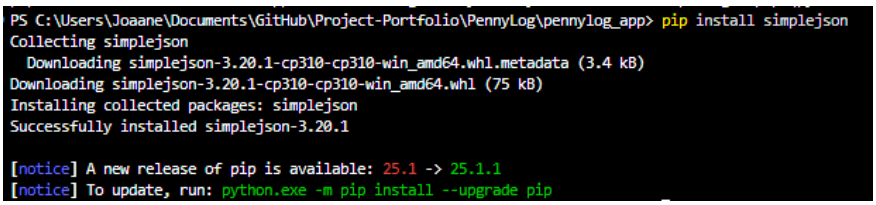


Figure 4.3: Pip Install Command

We can also paste the dependencies with the version number into the requirement.txt file and run the command “pip install -r requirements.txt” as shown in Figure 4.4.

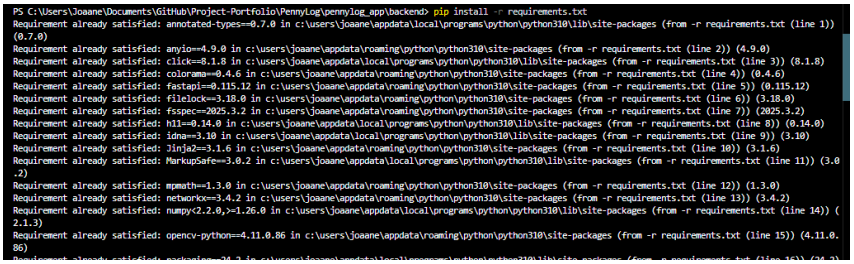


Figure 4.4: Pip install -r requirements.txt

4.2.1 Visual Studio Code

Visual Studio Code can be downloaded at the official website which is https://code.visualstudio.com/ and user will need to click “Download for Windows” to initiate the downloading as shown in Figure 4.5.

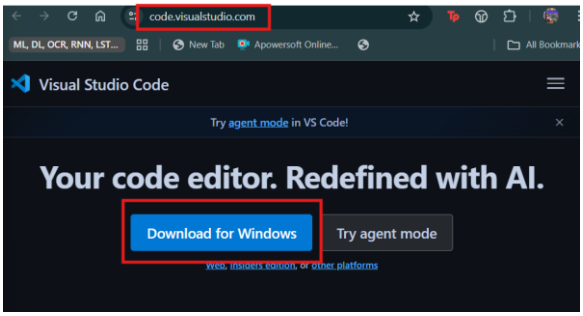


Figure 4.5: Visual Studio Code

Once the download has been completed, the execution file will appear in the recent download history and user will just need to click it to execute**.** To complete the setup, user will just need to accept the terms and proceed to save the visual studio in an appropriate location as well as other settings recommended by the installation wizard in Figure 4.6.

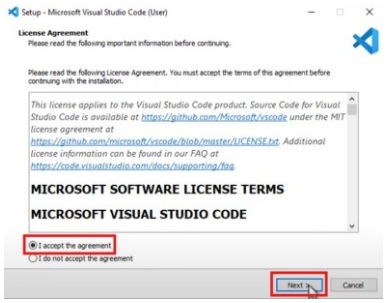


Figure 4.6: VS Code Installation Wizard

4.2.2 Chocolatey, Dart SDK and Dart Plugin

To use Dart for the frontend development of PennyLog using flutter framework, the Dart Software Development Kit (SDK) is needed since it provides the necessary tools for execution such as the Dart language compiler, Dart runtime, Dart Command Line Interface (Cli) tools, and standard libraries. The downloads can be done at this link https://dart.dev/get-dart, and downloaded using the windows operating system following the steps given as seen in Figure 4.7.

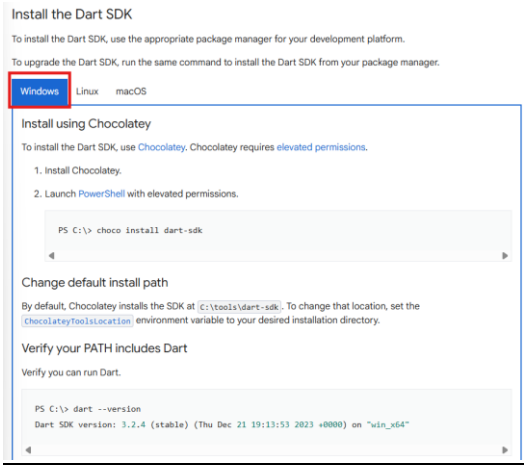


Figure 4.7: Install Dart SDK

However, before the dart SDK is downloaded, Chocolatey which is the package manager for windows will be needed to help simplify the installation of software's just by using the command line. Instead of manually downloading the installer, setting the environment variables and the configuration of paths, Chocolatey will be the one to automate the processes just by using a single command. The installation of Chocolatey will be via Powershell script using the administrative shell as shown in the steps at the website. To check if chocolatey has been downloaded and can be used, is by running the command choco as seen in Figure 4.8.

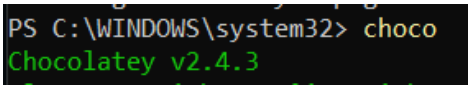


Figure 4.8: Choco Version

After successfully downloading chocolatey, in the powershell, dart sdk can be downloaded by just running this command "choco install dart sdk" as seen in Figure 4.9.

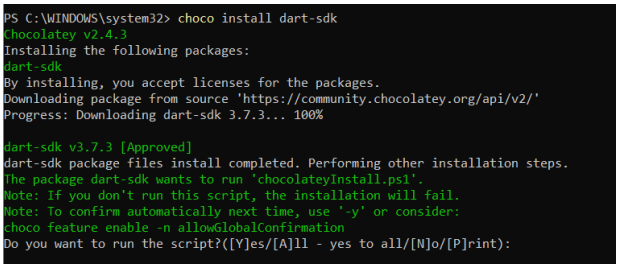


Figure 4.9: choco install dart sdk

After running the script, the dart sdk will successfully be downloaded. To check the dart installation is confimed and to check the version can be seen by running “dart –version” in the powershell in Figure 4.10.

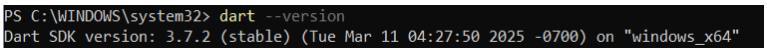


Figure 4.10: dart version

After successfully downloading the dart sdk, we will need to set the environment path to ensure that the dart tool can be accessible in any terminal window and has global access to dart command. Editor like Visual Studio Code can auto detect the Dart tool only if its set in the environment variable as seen in Figure 4.11.

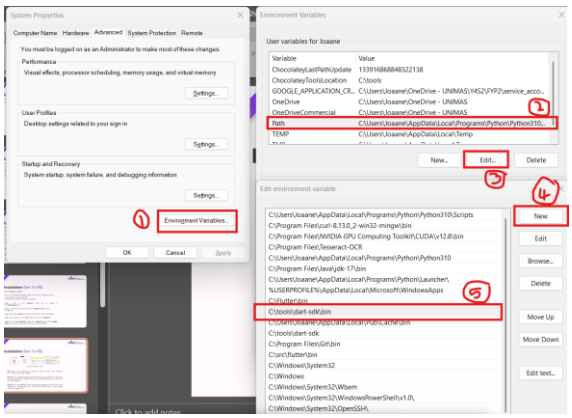


Figure 4.11: Dart SDK Environment path

To confirm the environment path has been set successfully, we can check via the command prompt by typing “dart”. The output will be as shown in Figure 4.12. If not, we will just have to restart the laptop to show the result.

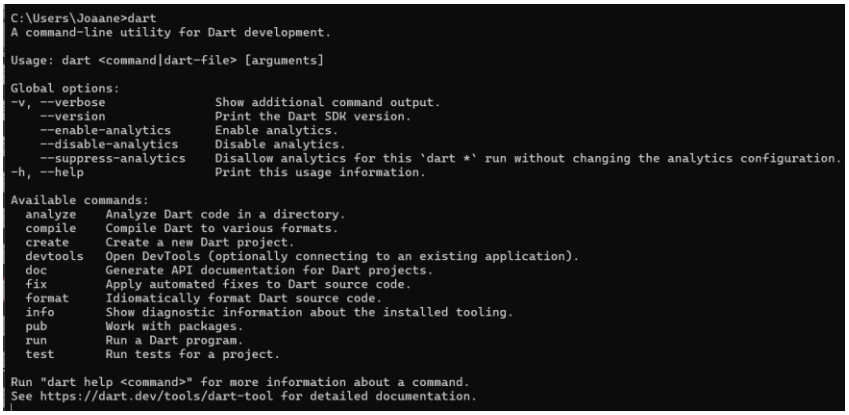


Figure 4.12: Dart Availability

The dart plugin in visual studio code needs to be downloaded as well to enable the full support of the dart programming language in the editor. The dart plugin can be found under the extensions section at the left navigation bar. We will just need to click “Install” and relaunch the vs code to start using dart as seen in Figure 4.13.

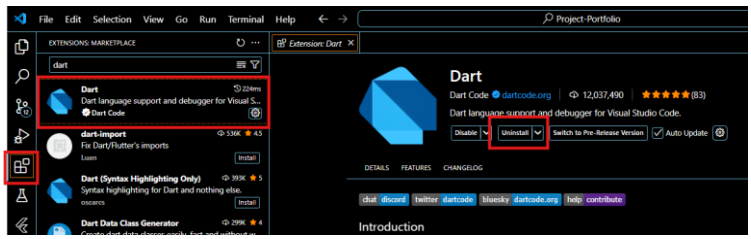


Figure 4.13: Dart Plugin

4.2.3 Python

Python is the main programming language for the backend connection. Python 3.10.11 was selected to be installed to ensure compatibility with a key machine learning library particularly TensorFlow. TensorFlow provide limited support for newer python releases for example python 3.11+, therefore by installing 3.10.11, it can help avoid compatibility issue and ensure that there can be a smooth integration of TensorFlow with other related dependencies such as pandas, scikit-learn and pytesseract. To download python, the installation can be found here https://www.python.org/downloads/release/python-31011/.

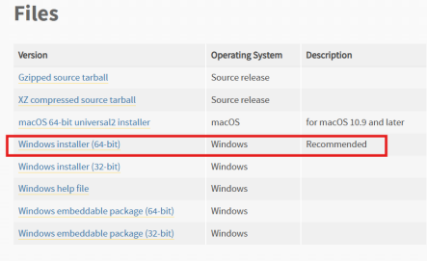


Figure 4.14: Python Installer

Based on the Figure 4.14, Windows installer (64-bit) is the recommended file to automatically run the execution file after downloading. After clicking the execution file, we will just need to click install now and allow the installation to be completed as seen in Figure 4.15.

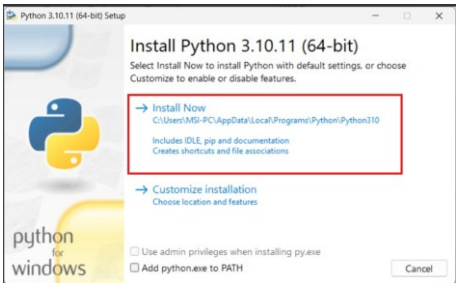


Figure 4.15: Python Installation File

After installation, similarly we need to insert the Script path of the python in the environment variable path. Once the path has been correctly set as seen in Figure 4.16, the installation for python has been completed

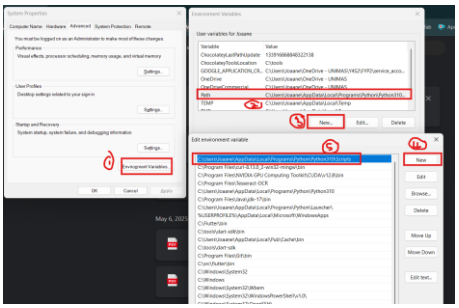


Figure 4.16: Python Environment Path

4.2.4 Android Studio IDE, Android SDK, Java Runtime Environment (JRE) and Java Development Kit (JDK)

PennyLog is a mobile application developed in VS Code and tested using the mobile emulator integrated in it. However, Android studio will be needed to

provide access to the android SDK, and it manages the emulator via AVD Manager. To compile and run the application on the Android emulator, both Java Runtime Environment (JRE) and Java Development Kit (JDK) are required by Gradle.

The JDK includes both the JRE and development tools need. Therefore, we need is to download the JDK in the official oracle website, since we are using windows, under JDK 24 for windows, download the x64 installer in https://www.oracle.com/java/technologies/downloads /?er=221886#jdk24-windows as seen in Figure 4.17.

Figure 4.17: JDK 24

After downloaded, we can run the execution file that was downloaded into the download history. Proceed to click Next in the next few wizards guide until the JDK has been successfully installed as shown in Figure 4.18.

Figure 4.18: Successful Installation JDK

To verify that the JDK has been successfully downloaded on my Microsoft Windows computer, use the command “java -version” in the terminal to check and

the version of java that has been downloaded successfully installed will be shown as seen in Figure 4.19.

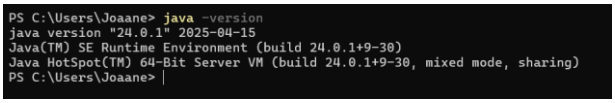


Figure 4.19: Java Version

After completely downloaded the pre-requisites needed, we can start the Android Studio IDE installation. It can be downloaded at https://developer.android.com/studio. Click on the download button shown in Figure 4.20.

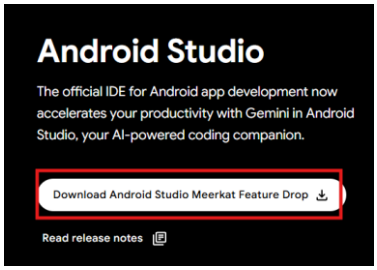


Figure 4.20: Download Android Studio

Make sure to read the terms and condition, then click on the agree and then click again on the download as seen in Figure 4.21.

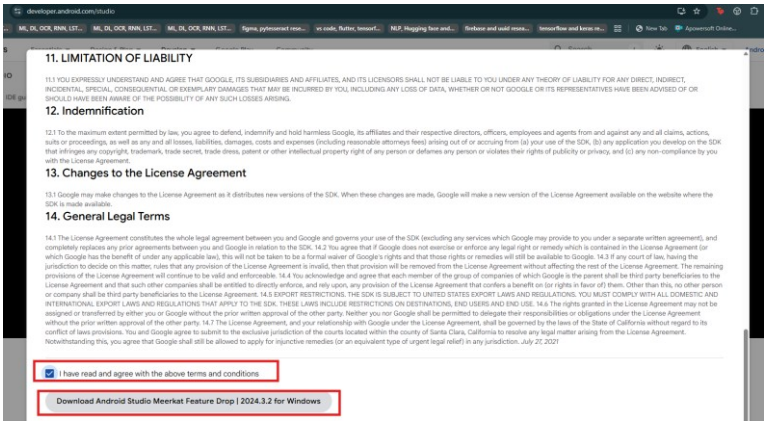


Figure 4.21: Android Studio Terms and Condition

Ensure the execution file is completely downloaded and run it and proceed with the set-up wizard like usual. After Android studio has been downloaded, the IDE will open to show as seen in Figure 4.22. Android Studio is ready to be used.

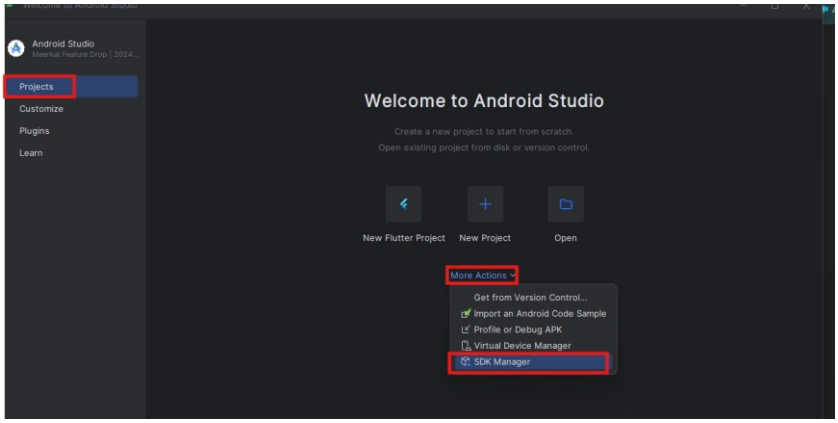


Figure 4.22: Android Studio IDE

The Android SDK will automatically be downloaded as well during the initial setup alongside the IDE. The android SDK is essential for compilation and running of the flutter application on the android devices. This approach is to simplify the environment configuration and for compatibility purposes with the android emulator and the system. It is managed directly in the Android Studio integrated SDK Manager. Under Project, just click more action and choose SDK Manager as shown in diagram.

4.2.5 Flutter SDK

Flutter SDK is also another essential that is needed because it has all the tools needed to develop the flutter app. By default, android studio is designed for native android application development. However, if we manually install the flutter SDK in our own windows OS and configure it as a plugin within other IDE software, developers will be able to develop a flutter application in other IDE such as visual studio code. The link to download flutter SDK is https://docs.flutter.dev/get-started/install/windows/mobile and click “Download and install” to download the SDK as shown in Figure 4.23.

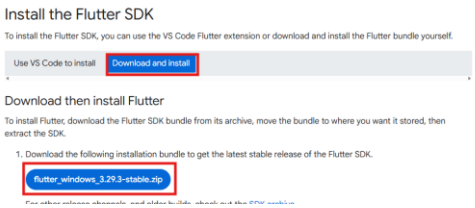


Figure 4.23: Install Flutter SDK

Once downloaded, unzip the file and choose a proper destination location to store the SDK. Once a suitable location is chosen, copy the bin destination folder for the flutter SDK and saved in the environment variable as shown in Figure 4.24. Now the flutter SDK are accessible in VS code.

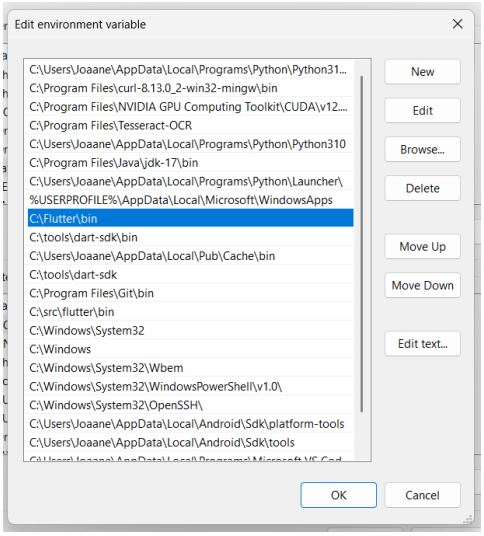


Figure 4.24: Flutter SDK Environment Variable

To check if other IDE will be able to develop a flutter application, run "flutter doctor" command in the terminal to check if everything has been successfully downloaded. It checks the environment and displays a status report of the flutter installation. If there are no issues, the output is shown in Figure 4.25.

```
PS C:\Users\Joaane> flutter doctor
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 3.29.3, on Microsoft Windows [Version 10.0.26100.4061], locale en-MY)
[√] Windows Version (11 Home Single Language 64-bit, 24H2, 2009)
[√] Android toolchain - develop for Android devices (Android SDK version 35.0.0)
[√] Chrome - develop for the web
[X] Visual Studio - develop Windows apps
    X Visual Studio not installed; this is necessary to develop Windows apps.
      Download at https://visualstudio.microsoft.com/downloads/.
      Please install the "Desktop development with C++" workload, including all of its default components
[√] Android Studio (version 2024.3.2)
[√] VS Code (version 1.100.1)
[√] Connected device (3 available)
[√] Network resources
```

Figure 4.25: Flutter Doctor

4.2.5 Git and GitHub

Git is the Command-line interface (Cli) tool that allows us to interact with GitHub which is a platform for hosting and provide version control for code repository. It ensures that all the code version that we push are saved and we will be able to retrieve the code version anytime. For PennyLog, Git will be used to track changes in the source code, commit progress locally and able to push the updates to the remote repository in GitHub. The link to download git is https://git-scm.com/downloads.

Figure 4.26: Download Git

Download the git version for the windows operating system as seen in Figure 4.26 and go through the installation wizard as seem in Figure 4.27 and continue to click Next without changing any of the default settings. Once everything is done, click “Finish”.

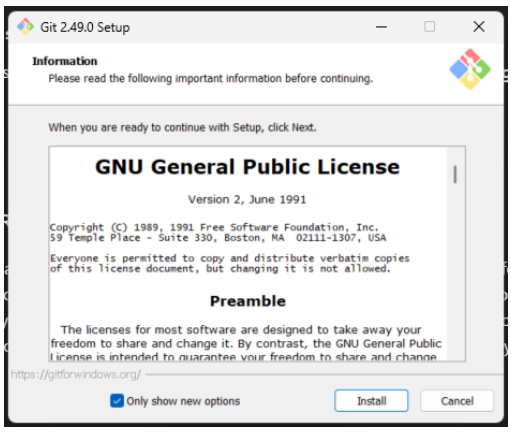


Figure 4.27: Git Set Up

To use GitHub as version control, the project, which was saved locally, are moved into the GitHub repository which was cloned into the local machine. Then

it is committed and pushed to GitHub. Every time there is new changes, using the command “git add” to add all modified/created files, “git commit =m “Your message”” to commit changes and “git push origin main” is to push to GitHub.

4.2.5 Firestore and Node.js

Firestore is the main database used for PennyLog. To automate the registration of PennyLog into firestore, we will need to download the firebase Cli tool. First, we will need to download Node.js because firebase Cli tool is distributed as a Node.js package. The link to download Node.js is at https://nodejs.org/en and download the Long-Term Support (LTS) version as seen in Figure 4.28.

Figure 4.28: Node.js Installation

Proceed with the installation wizard the same way. Click on accept the agreement, Next and Install. The downloaded package must be set in the environment as well. To check if Node.js has been successfully downloaded is through the terminal by the command “node –version”. The version that has been downloaded will be shown.

Next**,** to download the firebase Cli, run the command “npm install firebase tools” in the terminal as seen in Figure 4.29.

```
PS C:\Users\Joaane\Documents\GitHub\Project-Portfolio\PennyLog\pennylog_app> npm install -g firebase-tools

added 85 packages, removed 19 packages, and changed 610 packages in 32s

78 packages are looking for funding
  run `npm fund` for details
```

Figure 4.29: Firebase Tool Installation

After completed, then login into firebase via the cli by using “firebase login” as seen in Figure 4.30.

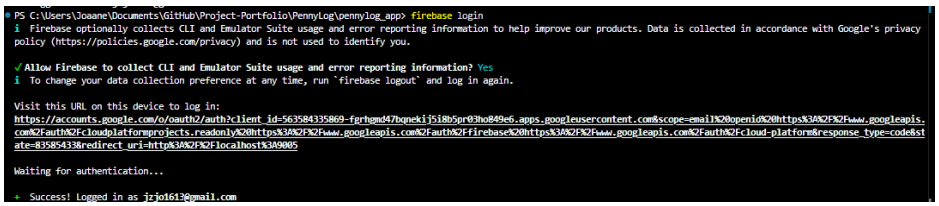


Figure 4.30: CLi firebase login

Once successfully login, it will display at the window as seen in Figure 4.31.

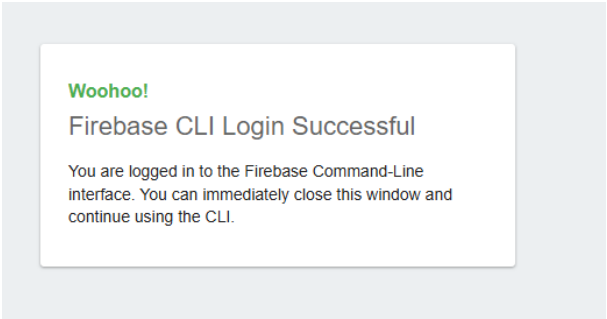


Figure 4.31: Firebase CLi Login Successful

Next is to create a new firebase project in https://console.firebase.google.com/u/0/ as seen in Figure 4.32.

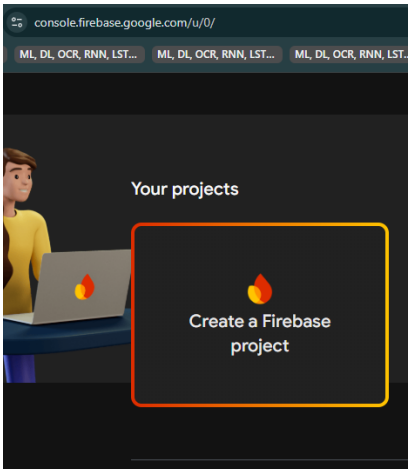


Figure 4.32: Create firebase project

In this case the project is called “PennyLog” as seen in Figure 4.33 and click continue and enable AI assistance for the firebase project. After that just create default account for firebase and click create project.

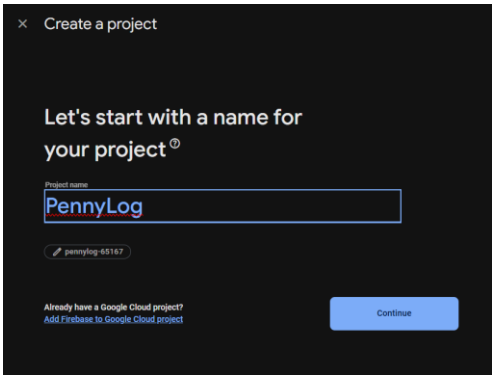


Figure 4.33: Firebase Project Name

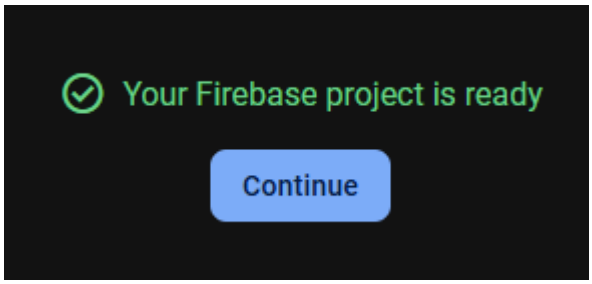


Figure 4.34: Firebase Project Ready

When the firebase project is ready, click continue as seen in Figure 4.34. To check all the projects available via the Cli, run the command “firebase projects: list”. All the available project created will be shown in Figure 4.35.

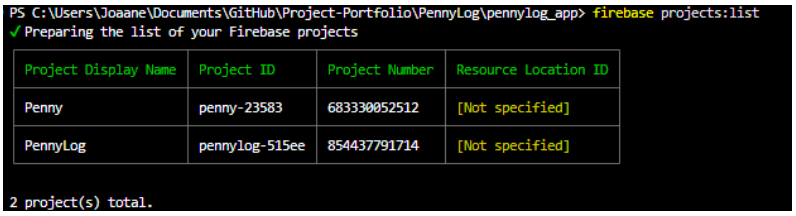


Figure 4.35: Firebase Project

To link within flutter and firebase, we will use the cli tool called flutter fire. Use the command “dart pub global activate flutterfire_cli” in the terminal as seen in Figure 4.36. Ensure the current directory is the working directory of the project. Once installed and there is no error, the Cli is ready to be used.

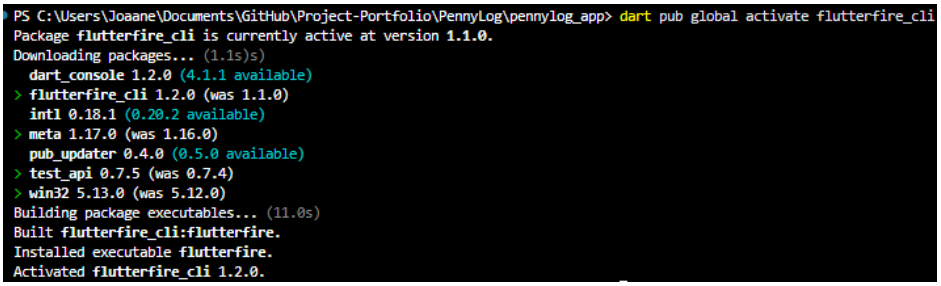


Figure 4.36: Dart pub global activate flutterfire_cli

To link the project, use the command “flutterfire configure” and choose the project that we want to link with. Then choose the platform for configuration support which is android as seen in Figure 4.37.

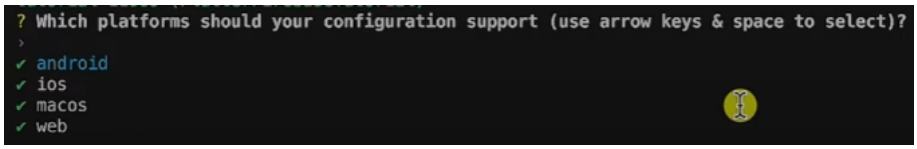


Figure 4.37: Firebase platform configuration

Then next the CLi will do all the registration automatically without any other steps. The successful message is shown below in Figure 4.38.

Figure 4.38: Configuration successful

To ensure there is no error in the firebase_option.dart file, the project will need firebase_core and firebase_analytics in the pubspec.yaml under dependencies.

4.2.6 Pytesseract

Since pytesseract is the python wrapper for Google’s Tesseract OCR engine, it allows users to use the Tesseract OCR capabilities directly in python code. For PennyLog, we will use the OCR capabilities via the backend. While pytesseract is the python interface for the OCR, it does not contain the engine itself. Instead, it will rely on the tesseract engine which needs to be installed in the windows OS to do the image processing and text recognition.

First is to download the Tesseract OCR engine. To download the OCR engine, it can be found in https://github.com/ub-mannheim/tesseract/wiki. Download the latest installer compatible with windows operating system as shown in Figure 4.39.

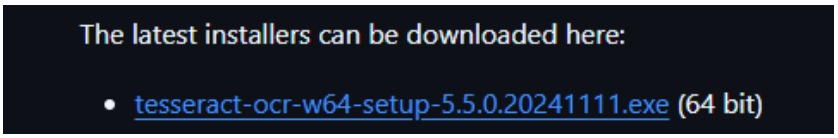


Figure 4.39: Tesseract Engine Installer

Once downloaded, set the PATH in environment variable. There is also a tesseract OCR dart package dependency needed in the pubspec.yaml file as seen in Figure 4.40.

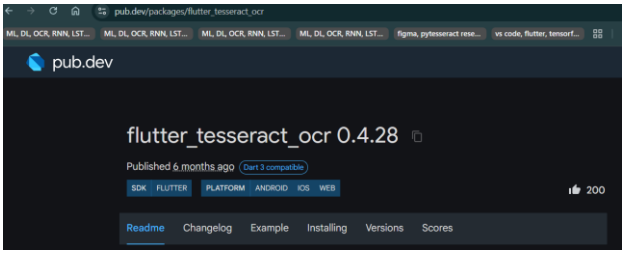


Figure 4.40: Tesseract OCR package

When using the OCR engine, it supports many languages. It has language file which the engine has trained on. To configure it to output only certain type of languages, the languages can be found in https://github.com/tesseract-ocr/tessdata and be saved in the project directory to be accessed. PennyLog will be using English, Malay, Chinese and Tamil for the languages. The selected traineddata file can be downloaded by click the download raw file as shown in Figure 4.41.

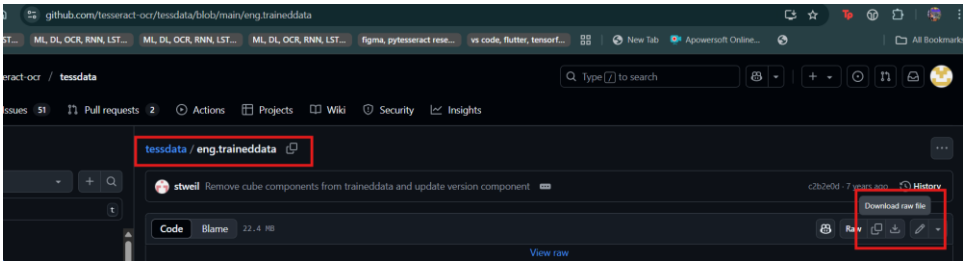


Figure 4.41: Traineddata Tesseract OCR

The traineddata are then saved under a assets folder as shown in Figure 4.42.

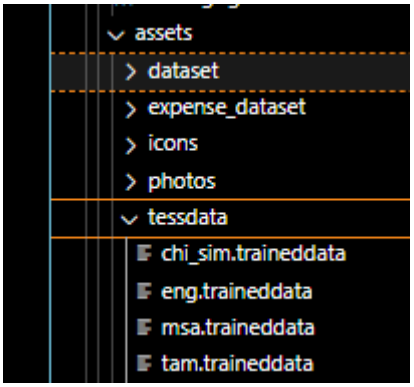


Figure 4.42: Tess data directory

4.2.7 FAST API and Uvicorn

For the initial phase of the development, a local server is used to test and run the backend API for PennyLog frontend and backend communication. Since API are only accessible to the frontend by using a server, PennyLog will be using Uvicorn which is a lightweight ASGI server that supports Python code. Uvicorn is used to host the backend locally, enabling data exchange between the frontend and backend of PennyLog. The server which rans the backend will be build using FastAPI, which is a lighweight framework. FastAPI creates REST API routes which will enable PennyLog frontend and backend communicate using standard HTTP methods such as GET and POST.

Firstly, to install uvicorn and FastAPI, the terminal is used by running “pip install uvicorn fastapi”. The installation process will be shown like in Figure 4.43.

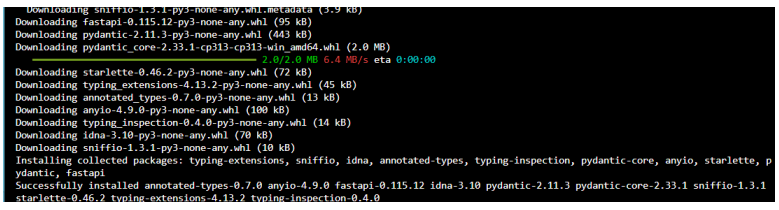


Figure 4.43: Installation uvicorn and fastapi

After downloaded, first we will need to import fastapi as shown in Figure 4.44.

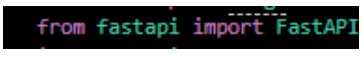


Figure 4.44: Import FastAPI

Then initialise new application as shown in Figure 4.45.

```
app = FastAPI()
```

Figure 4.45: Initialise new app

Then the next step is to run the server as shown in Figure 4.46.

```
> uvicorn main:app --reload
```

Figure 4.46: Run Server

If the server successfully run, the result is shown in Figure 4.47.

```
INFO:     Started server process [12864]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
```

Figure 4.47: Server Successful Running

JSON response in the form of key message that holds string “Hello World” which is returned from FASTAPI route at http://127.0.0.1:8000 in Figure 4.48.

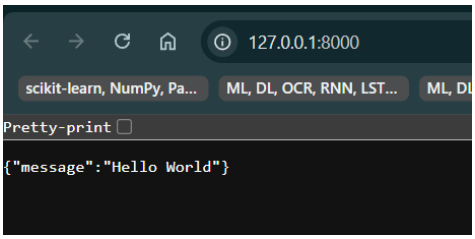


Figure 4.48: JSON Response

And Route for swagger UI which is an auto-generated interactive documentation interface for REST API routes at http://127.0.0.1:8000/docs can be seen in Figure 4.49.

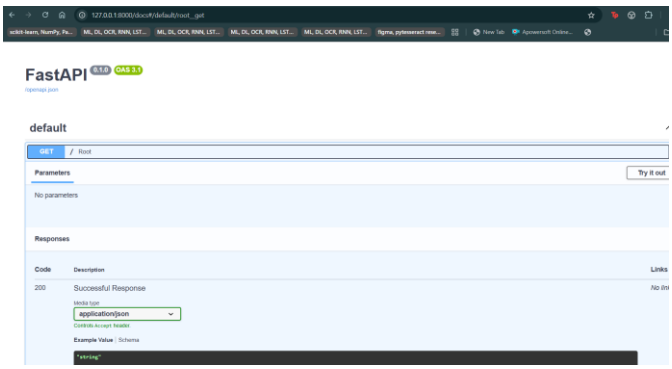


Figure 4.49: FastAPI Swagger UI

4.2.8 Google Cloud Run and Docker

After developing and testing the FastAPI backend locally using the Uvicorn server, the applications backend is deployed to the cloud using Google Cloud Run to make it accessible by others online. The reason why google cloud run is used because it is a fully managed platform that allows users to deploy containerised application. It is scaled automatically based on traffics and will not require server management. To run the FastAPI server on google cloud run, the application will need to be containerised using docker. The docker container will include the python environment, the require libraries, FastAPI backend code, uvicorn server and the configuration files to ensure the app behaves similarly in any environment. These are the steps to deploy.

The first step is to create a Google Cloud Account by navigating to the official Google Cloud website at https://cloud.google.com/?hl=en by clicking on the "Get started for Free" to begin the sign up process as seen in Figure 4.50.

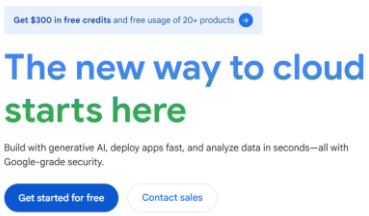


Figure 4.50: Get Started for Cloud Run

For new users, they are offered $300 in free credits and has access to over 20+ Google Cloud services and products. Once clicked, user will then be asked to sign in with a google account, select country and optionally choose to receive the email updates as seen in Figure 4.51.

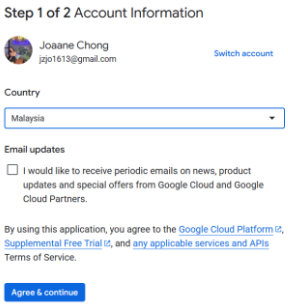


Figure 4.51: Account Information Google Cloud

User then need to click on "Agree & Continue" to proceed with the registration. To prevent their services from being abused by spammers and bots, Google will require a valid payment method to verify the user's identity. Users will not get charged unless they upgrade to a paid account. User will need to fill in their payment profile and payment method as seen in Figure 4.52 and click "Start free" to activate the free trial.

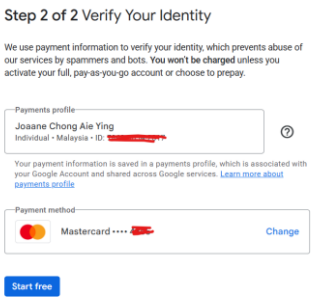


Figure 4.52: Verify Identity

Once the account has been created, to we need to create project for PennyLog to ensure we can organise the resources and manage permission for the project. First is to navigate to the console via https://console.cloud.google.com and make sure we are logged in as seen in Figure 4.53.

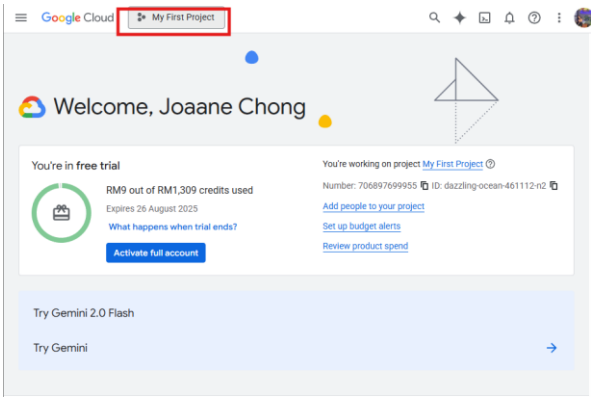


Figure 4.53: Google Cloud Run Console

Then open the project selector drop down at the top left corner of the dashboard as seen in Figure 4.53. It will open a list of existing projects and an object to create a new one as seen in Figure 4.54.

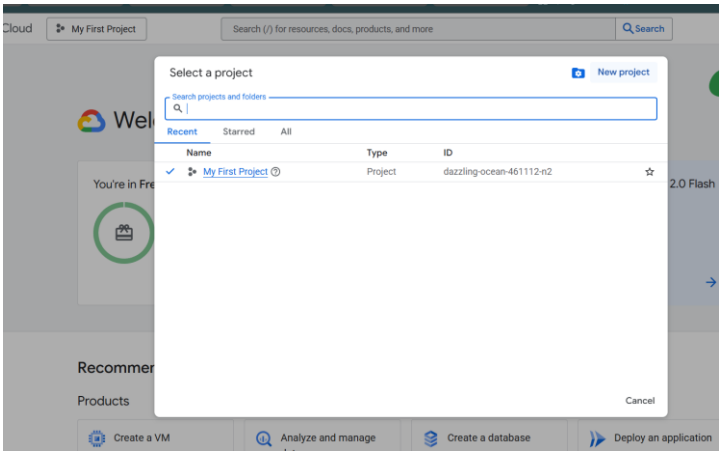


Figure 4.54: Project List

Click on the New Project to create a new project for PennyLog and a window will appear to fill in the project details as seen in Figure 4.55 such as the project name, name it as “fastapi-pennylog-backend” which is meaningful with no organisation since it is a private personal project. Then click “Create”.

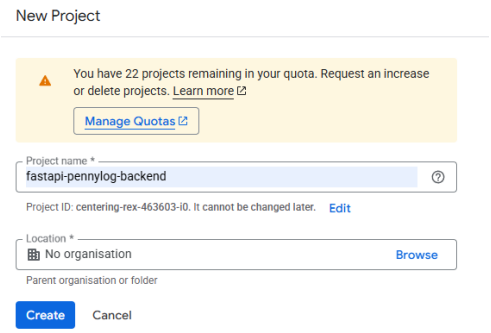


Figure 4.55: Create Project Cloud Run

There will be a confirmation message showing that the project has been created as seen in Figure 4.56.

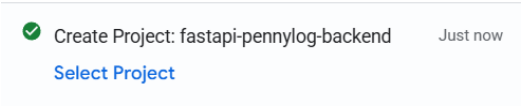


Figure 4.56: Successful Project Creation Cloud Run

The newly created project will also be shown in the project list and can be selected as seen in Figure 4.57.

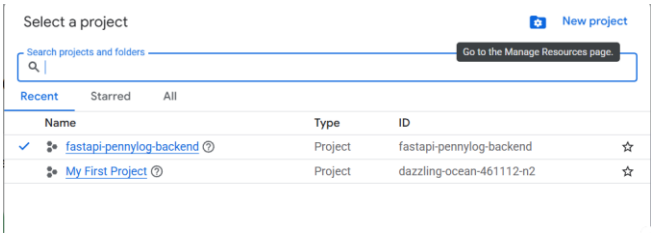


Figure 4.57: Project List Cloud Run

Once selected, the dashboard along the welcome message with the free trial credit details and the project ID will be shown in Figure 4.58. This confirms that the project has been set up and ready to be used.

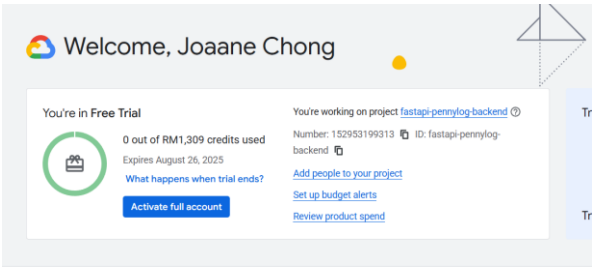


Figure 4.58: Cloud Run Dashboard

To deploy the backend successfully on Google cloud Run, there are several API that needs to be enabled in the project to ensure services like building Docker containers, storing container image and hosting the applications. User can enable the API from the console, by going to the google Cloud Console, choose APIs & Services, and choose library in the link given https://console.cloud.google.com/apis/library?inv=1&invt=Abyg1g as seen in Figure 4.59.

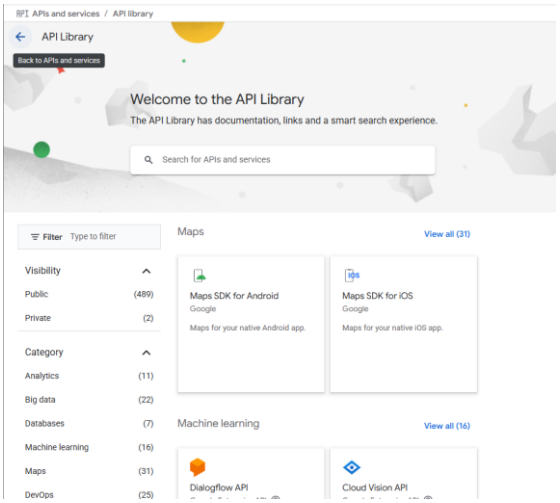


Figure 4.59: Cloud Run API Library

Then search and enable three APIs which is Cloud Run API, Artifact Registry API and Cloud Build API. After searching for the API, just click on the "Enable" button as seen in Figure 4.60.

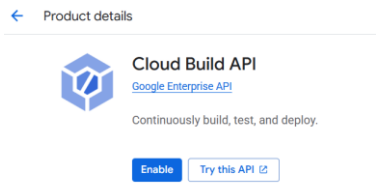


Figure 4.60: Enable API Cloud Run

After enabled it will appear in the APIs and Services and navigate to the Enabled APIs and services section in the console.

After the APIs been enabled, the next step is to install and initialised the Google cloud run CLI. The CLI will help make the process of building the docker image, creating repository, authenticate, docker push and everything easier with just using the command in the terminal. First is to install the CLI at https://cloud.google.com/sdk/docs/install. Choose the appropriate installer for the OS as seen in Figure 4.61.

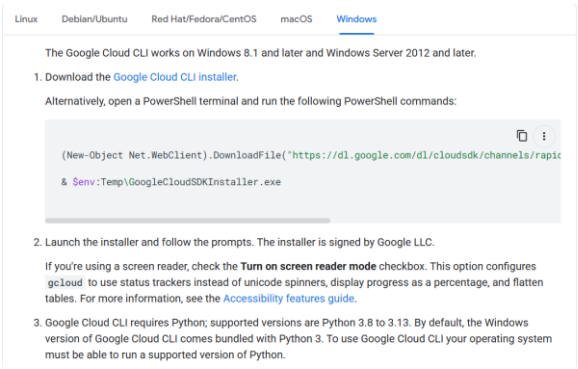


Figure 4.61: Cloud Run SDK

After the APIs been enabled, the next step is to install and initialised the Google cloud run CLI. The CLI will help make the process of building the docker image, creating repository, authenticate, docker push and everything easier with just using the command in the terminal. First is to install the CLI by running the SDK command in the terminal given at https://cloud.google.com/sdk/docs/install. Choose the appropriate installer for the OS as seen in Figure 4.61. Then run the installation using the PowerShell command as seen in Figure 4.62.

Figure 4.62: Cloud Run SDK installer

The installer wizard will pop up and we click next through the setup ensuring that we set up appropriately as seen in Figure 4.63.

Figure 4.63: Google Cloud CLI setup

Then after installation complete, we initialise by running gcloud init in the command prompt to sign in the google account, select the fastapi-pennylog-backend project and choose a default region which should be asia-southeast1 seen in Figure 4.64.

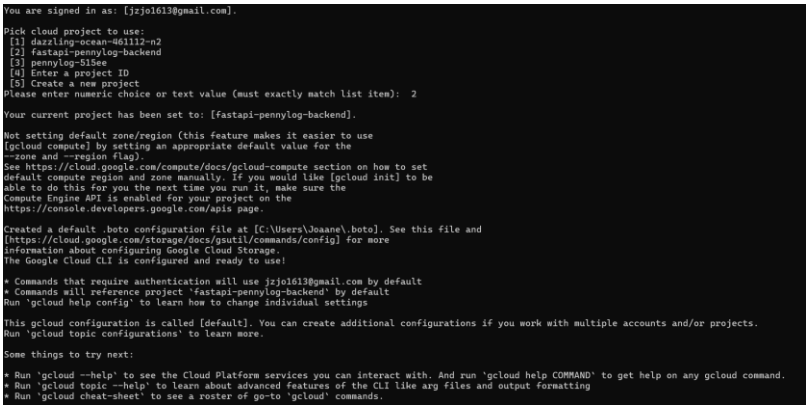


Figure 4.64: gcloud init

Once authenticated and configured, a pop up will show the CLI is ready to be used as seen in Figure 4.65.

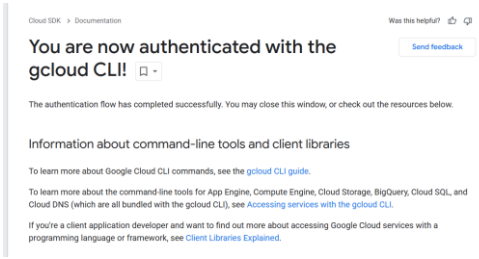


Figure 4.65: CLI authenticated

Next is to install Docker in Desktop to be used to build and containerise the backend image at https://www.docker.com/products/docker-desktop/ . Installation is completed when the pop up shown as seen in Figure 4.66.

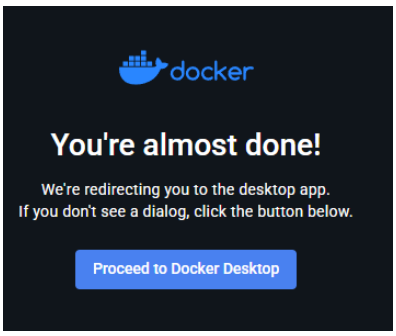


Figure 4.66: Docker Installation Complete

To check docker has been successfully installed is by running docker –version in the terminal as seen in Figure 4.67.

Figure 4.67: Docker Version

If encounter error as seen in Figure 4.68 when launching Docker, the Windows Subsystem for Linux (WSL) needs to be downloaded by running "wsl –update" in the terminal as seen in Figure 4.69

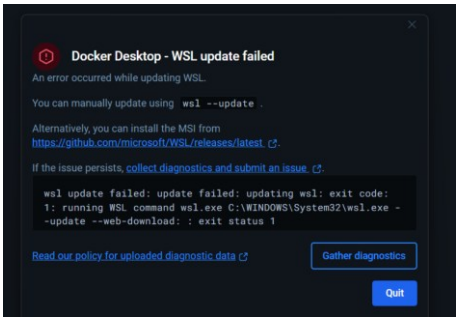


Figure 4.68: WSL Error in Docker

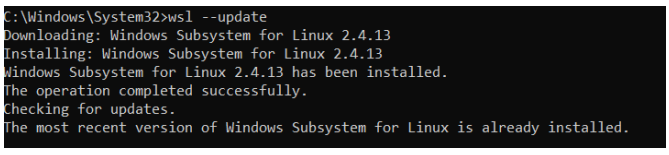


Figure 4.69: wsl --update

The next step is then to build and push the docker image to the google cloud artifact registry. Firstly, create artifact registry which is a onetime set up per project per region. This is to create a Docker compatible container registry on Google Cloud as seen in Figure 4.70.

```
C:\Users\Joaane\AppData\Local\Google\Cloud SDK>gcloud artifacts repositories create fastapi-repo --repository-format=docker --l
ocation=asia-southeast1 --description="Docker repo for FastAPI backend"
Create request issued for: [fastapi-repo]
Waiting for operation [projects/fastapi-pennylog-backend/locations/asia-southeast1/operations/c3f9f461-41ec-4009-b1f5-0fcfaadf
5950] to complete...done.
Created repository [fastapi-repo].
```

Figure 4.70: Artifact registry

Next is to navigate to the backend where the FastAPI project is located where the Dockerfile exist. The Dockerfile with all the necessary command for installation should be ready as seen in Figure 4.71.

```
1  FROM python:3.10-slim
2
3  # Set working directory
4  WORKDIR /app
5
6  # Install system dependencies
7  RUN apt-get update && apt-get install -y libglib2.0-0 libgl1-mesa-glx tesseract-ocr
8
9
10 # Copy files relative to build context (which is already pennylog_app/)
11 COPY backend /app
12 COPY assets/tessdata /app/assets/tessdata
13 COPY assets/dataset /app/assets/dataset
14 COPY assets/expense_dataset /app/assets/expense_dataset
15 COPY backend/requirements.txt ./requirements.txt
16
17 # Install Python dependencies
18 RUN pip install --prefer-binary --no-cache-dir --default-timeout=300 -r requirements.txt
19
20 # Copy the rest of the app code
21 COPY . .
22
23 # Run the FastAPI app using uvicorn
24 CMD ["python", "-m", "uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8080"]
```

Figure 4.71: Dockerfile

Then build the docker image locally as seen in Figure 4.72. This means that the docker image is built from the current directory and is tagged with full artifact registry path.

```
PS C:\Users\Joaane\Documents\GitHub\Project-Portfolio\PennyLog\pennylog_app\backend> docker build -t asia-southeast1-docker.pkg.dev/fastapi-pennylog-backend/fastapi-repo/ba
ckend .
[+] Building 1010.1s (7/11)                                                                                                    docker:desktop-linux
 => => transferring context: 28                                                                                                              0.0s
 => [1/6] FROM docker.io/library/python:3.11-slim@sha256:dbf1de478a55d6763afaa39c2f3d7b54b25230614980276de5cacdde79529d0c                  12.5s
 => => resolve docker.io/library/python:3.11-slim@sha256:dbf1de478a55d6763afaa39c2f3d7b54b25230614980276de5cacdde79529d0c                   0.0s
 => => sha256:9d545c45fb8c5b235b5b88114aeeefb48a96eface42af73e9545845852402e2d 16.21MB / 16.21MB                                            2.3s
 => => sha256:dbf1de478a55d6763afaa39c2f3d7b54b25230614980276de5cacdde79529d0c 9.13kB / 9.13kB                                              0.0s
 => => sha256:df52c7d12cc5bd9b0437abbf295ef7eb78f68948e906a68cec8741a585bb6df3 1.75kB / 1.75kB                                              0.0s
 => => sha256:a6e724de6234966d669af179631f7ba6d18527f7a49b7e4cbf0ef970ed2ead06 5.37kB / 5.37kB                                              0.0s
 => => sha256:61320b01ae5e8798393ef25f2dc72faf43703e60ba089b07d7170acbabbf8f62 28.23MB / 28.23MB                                            6.1s
 => => sha256:fa70febde0f65a8b721a3ff8c13b09826c281b025abd1a92b4b08eb839b7cbd1 3.51MB / 3.51MB                                              0.9s
 => => sha256:09c4893e5320a7c59acf82e87a07980214c1a955ef9f60cb9d0a48ba562c315a 2500 / 2500                                                  1.5s
 => => extracting sha256:61320b01ae5e8798393ef25f2dc72faf43703e60ba089b07d7170acbabbf8f62                                                   3.3s
 => => extracting sha256:fa70febde0f65a8b721a3ff8c13b09826c281b025abd1a92b4b08eb839b7cbd1                                                   0.4s
 => => extracting sha256:9d545c45fb8c5b235b5b88114aeeefb48a96eface42af73e9545845852402e2d                                                   2.0s
 => => extracting sha256:09c4893e5320a7c59acf82e87a07980214c1a955ef9f60cb9d0a48ba562c315a                                                   0.0s
 => [internal] load build context                                                                                                           0.4s
 => => transferring context: 2.86MB                                                                                                         0.3s
 => [2/6] WORKDIR /app                                                                                                                      0.3s
 => [3/6] RUN apt-get update && apt-get install -y build-essential                                                                        993.0s
 => => # Get:37 http://deb.debian.org/debian bookworm/main amd64 libquadmath0 amd64 12.2.0-14+deb12u1 [145 kB]
 => => # Get:38 http://deb.debian.org/debian bookworm/main amd64 libgcc-12-dev amd64 12.2.0-14+deb12u1 [2437 kB]
 => => # Get:39 http://deb.debian.org/debian bookworm/main amd64 gcc-12 amd64 12.2.0-14+deb12u1 [19.3 MB]
```

Figure 4.72: Docker build

After the Docker image has been build, we need to configure the Docker to use the correct google cloud credentials by running the “gcloud auth” command as seen in Figure 4.73. This command will update the local Docker configuration file config.json to use the gcloud for authentication when pushing to this domain.

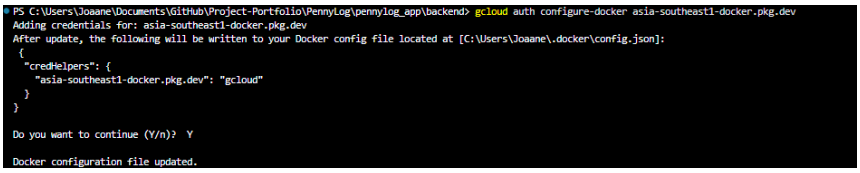


Figure 4.73: gcloud auth

Once the image has been built and tagged corrected, it can be pushed to google cloud as seen in Figure 4.74 by running the command docker push in the terminal.

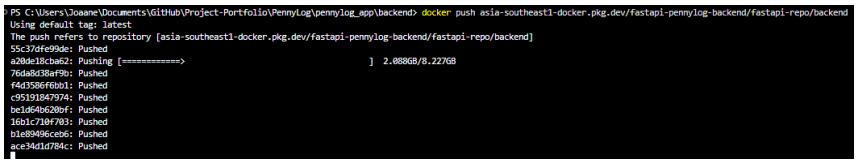


Figure 4.74: docker push

Before deploying our backend to Google Cloud Run, we need to ensure our backend has access to firestore and has the necessary permission. By creating firebase_init.py file as seem in Figure 4.75 and each file at the backend is referenced to this file, it ensures that the backend has access.

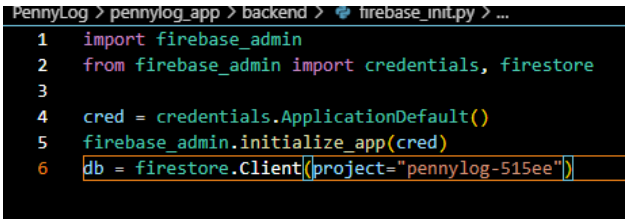


Figure 4.75: firebase_init.py

First is to import the firebase admin SDK. The firebase_admin is the official SDK for server-side firebase access. Then, credentials are used to authenticate PennyLog backend to firebase and firestore is the module used to interact with

Firestore. Then cred = credentials.ApplicationDefault() uses the Application Default Credentials (ADC). In Google Cloud Run, ADC will automatically pick up the service account credential that is assigned during deployment using the command fastapi-sa. This means we don't have to manually include the .json file that contains sensitive credential files for backend access to firestore. The command firebase_admin.initialize_app(cred) will connect the FastAPI backend to firebase services like firestore using the previously loaded credential. Then, db = firestore.Client(project="pennylog-515ee") creates a client db to read and write to firestore where we explicitly tell which firebase project to connect with id pennylog-515ee to use. This is necessary because the cloud run project fastapi-pennylog-backend is not configured directly to use the firestore project pennylog-515ee.

Moving on, the next step is to create a service account that will act on behalf of PennyLog to access other Google Cloud Services securely. The service account is named as fastapi-sa as seen in Figure 4.76.

Figure 4.76: Service Account PennyLog

Then bind the project and firestore permission to the service account. This will grant access to the service account permission to access firestore and the project that is in cloud run as seen in Figure 4.77.

Figure 4.77: Binding to service account

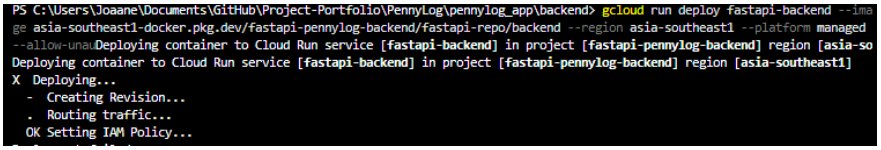


Figure 4.78: gcloud run deploy

Then the last step is to run "gcloud run deploy" as seen in Figure 4.78 using the service account and the necessary settings. The command is as shown below.

```
gcloud run deploy fastapi-backend --image asia-southeast1-
docker.pkg.dev/fastapi-pennylog-backend/fastapi-repo/backend --platform
managed --region asia-southeast1 --allow-unauthenticated --service-account
fastapi-sa@fastapi-pennylog-backend.iam.gserviceaccount.com --memory=6Gi --
cpu=2
```

4.3 Introducing Role Based Access

There is only one type of user in PennyLog. The users are aimed at UNIMAS Student who will be the primary and main user for PennyLog.

4.3.1 User

PennyLog user are mainly students in UNIMAS covering from student who are in foundation and up to masters or PHD students who are currently still studying in any faculty and studies offered. Users will have access to all modules available in PennyLog which is the home, transaction, new transaction, insight and settings module. They will not need to register for an account to use PennyLog.

4.4 Modules for PennyLog Application

PennyLog has been divided into five different modules with each module having its own functionality and purposes that can help user with their financial management activity. The modules are connected and play a big role in ensuring accuracy in the display.

4.4.1 Home Module

Figure 4.79: Home Page Module

The Home Module as seen in Figure 4.79 is the home page will be the landing page that users are redirected to after the splash page has been shown. The Home Module focuses on 3 different section which is the overall financial balance widget in main.dart, spending trend graph widget in spending_trend_graph.dart and the recent transaction widget in recent_transaction.dart. All widget dynamically adapts the theme, multi-language and multi-currency supports when changed at settings. At the top of the screen, there is two drop down options where user can select a month and year, which the application will dynamically filters and update all the widget using the selected time range.

```
Row(
  children: [
    // Month Dropdown
    Container(
      margin: const EdgeInsets.only(top: 4),
      padding: const EdgeInsets.symmetric(horizontal: 12, vertical: 0),
      decoration: BoxDecoration(
        color: Colors.grey[300],
        borderRadius: BorderRadius.circular(20),
      ), // BoxDecoration
      child: DropdownButton<String>(
        value: selectedMonth,
        underline: const SizedBox(),
        icon: const Icon(Icons.arrow_drop_down, size: 20),
        iconEnabledColor: Colors.black,
        dropdownColor: Colors.grey[300],
        style: const TextStyle(
          fontSize: 14,
          fontWeight: FontWeight.bold,
          color: Colors.black,
        ), // TextStyle
        onChanged: (value) async {
          setState(() => selectedMonth = value!);
          await fetchTransactionData();
        },
        items: List.generate(12, (i) {
          final englishMonth = DateFormat.MMMM().format(DateTime(0, i + 1));
          final display = translatedMonths[i].isEmpty ? englishMonth : translatedMonths[i];
          return DropdownMenuItem<String>(
            value: englishMonth,
            child: Text(display),
          ); // DropdownMenuItem
        }), // List.generate
      ), // DropdownButton
    ), // Container
    const SizedBox(width: 8),
```

Figure 4.80: Drop Down

Figure 4.80 shows the selection is handled by using two different DropdownButton widgets, one for month and another for year. For example, in diagram shows the source code for displaying the drop-down widget for month. The selectedMonth will hold the current selected month, and onChanged, it calls the fetchTransactionData() function to retrieve and refresh all the widget displayed data. It uses DataFormat.MMMM() to format each of the month name in the drop down as January, February, and so on. The dropdown is wrapped in a styled Container with rounded corners for standardise UI appearance which is the borderRadius: BorderRadius.circular. The same goes for the year drop down.

Figure 4.81: fetchTransactionData() function

When either of the drop-down value changes or a new transaction is added, the fetchTransactionData() function is called as seen in Figure 4.81.

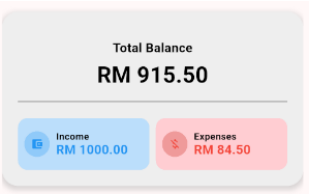


Figure 4.82: Financial balance widget

Firstly, for the overall financial balance widget as seen in Figure 4.82, the function will perform the following action: -

1) Convert the selected month and year to a date range as seen in Figure 4.83. Firstly, create a DateTime object which will represent the first day of the selected month. selectedYear is the year selected by user from the dropdown. Then, monthNumber is a number from 1 (January) to 12 (December), based on the selected month. 1 means the 1st day of the month. If the user selected, for example, April 2025 then it become DateTime(2025, 4, 1).

```
final firstDay = DateTime(selectedYear, monthNumber, 1);
final lastDay = DateTime(selectedYear, monthNumber + 1, 0);
```

Figure 4.83: Convert month and year

2) Format the month and year to match the firestore string format. It convert Dart DateTime objects into formatted string which are compatible with Firestore date filtering as seen in Figure 4.84.

```
final firstDayStr = DateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS").format(firstDay);
final lastDayStr = DateFormat("yyyy-MM-dd'T'HH:mm:ss.SSS").format(lastDay);
```

Figure 4.84: Format month and year to Firestore format

3) Query the firestore under 'transactions' collection for both income and expenses under the selected date range as seen in Figure 4.85. For the where('date', isGreaterThanOrEqualTo: firstDayStr) filters transaction on or after the first day of the selected month. Same goes for .where('date', isLessThanOrEqualTo: lastDayStr). Then, .where('type', isEqualTo: 'Income') ensures that the transaction is an income transaction, same goes for expenses. Lastly, .get() executes the query.

```
try {
  final incomeQuery = await FirebaseFirestore.instance
      .collection('transactions')
      .where('date', isGreaterThanOrEqualTo: firstDayStr)
      .where('date', isLessThanOrEqualTo: lastDayStr)
      .where('type', isEqualTo: 'Income')
      .get();

  final expenseQuery = await FirebaseFirestore.instance
      .collection('transactions')
      .where('date', isGreaterThanOrEqualTo: firstDayStr)
      .where('date', isLessThanOrEqualTo: lastDayStr)
      .where('type', isEqualTo: 'Expenses')
      .get();
```

Figure 4.85: Query Firestore for income and expenses total

4) Total up both the income and expenses by adding up all the amount values from firestore incomeQuery results. Firstly, .fold(0, ...) is a method that start with 0 and are repeatedly combines values using a function. Then, (sum, doc) => sum + (doc['amount'] ?? 0) is the function that adds up each amount field to the running sum. Lastly, doc['amount'] ?? 0 is when the amount field is null or missing, it will use 0 to avoid errors as seen in Figure 4.86.

```
double totalIncome = incomeQuery.docs.fold(0, (currentSum, doc) => currentSum + (doc['amount'] ?? 0));
double totalExpenses = expenseQuery.docs.fold(0, (currentSum, doc) => currentSum + (doc['amount'] ?? 0));
```

Figure 4.86: Sum the expenses and income

5) Update UI with the values. In flutter, every statefulWidget has a property which are called mounted. If mounted == true it means that the widget is currently part of the widget tree, meaning its visible or active on screen. When we execute setState(), which is a special method used inside a StatefulWidget to tell flutter that something in the UI has been changed, and it must rebuilt the widget. As seen in Figure 4.87, mounted will check if the widget is still visible on screen if user haven't navigated away from the screen, then its true. Then inside setState(), we are updating the value of totalIncome and totalExpenses based on the value of month and year selected.

```
if (mounted) {
  setState(() {
    this.totalIncome = totalIncome;
    this.totalExpenses = totalExpenses;
    isLoading = false;
  });
```

Figure 4.87: Update UI with total

When the totalIncome and totalExpenses is updated using setState(), on the UI, the widget will be rebuilt to reflect the latest values on screen. Firstly, the total balance widget as shown Figure 4.88.

```
Text(
    '$currencySymbol ${(totalIncome - totalExpenses).toStringAsFixed(2)}',
    style: const TextStyle(
    fontSize: 28,
    color: Colors.black,
    fontWeight: FontWeight.bold,
  ), // TextStyle
), // Text
```

Figure 4.88: Total Balance UI

Secondly is total income as seen in Figure 4.89.

```
Text(
  '$currencySymbol ${totalIncome.toStringAsFixed(2)}',
  style: const TextStyle(
    fontSize: 18,
    fontWeight: FontWeight.bold,
    color: Colors.blue,
  ), // TextStyle
), // Text
```

Figure 4.89: Total Income UI

And lastly, total expenses as seen in Figure 4.90.

```
Text(
  '$currencySymbol ${totalExpenses.toStringAsFixed(2)}',
  style: const TextStyle(
    fontSize: 18,
    fontWeight: FontWeight.bold,
    color: Colors.red,
  ), // TextStyle
), // Text
```

Figure 4.90: Total Expenses UI

The second widget on the home module is spending trend graph widget as seen in Figure 4.91. This widget is stored in spending_trend_graph.dart as shown in diagram.

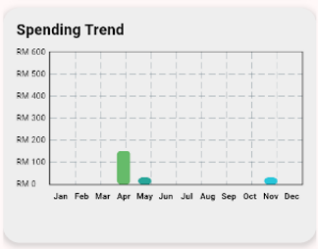


Figure 4.91: Spending Trend Bar Graph

The spending Trend Graph displays a bar chart visualisation of the total expenses for each month from January to December on the selected year. The chart is to help user spot seasonal trends as well as spending patterns. The spending trend graph are built once when the widget is first created using the initState() method as shown in Figure 4.92.

```
@override
Qodo Gen: Options | Test this method
void initState() {
  super.initState();
  fetchMonthlyExpenses();
  _translateLabels();
}
```

Figure 4.92: initState() for bar graph

In initState() is when the widget is first created. When the fetchMonthlyExpenses() function is triggered, it initilises the spending data for the graph as shown in Figure 4.93.

Figure 4.93: fetchMonthlyExpenses function

The function will perform the following operation: -

1) It will first initialise an empty map called tempExpenses of all 12 months with no value as seen in Figure 4.94.

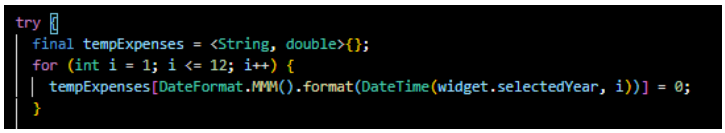


Figure 4.94: Initialise an empty map

2) Then it will query Firestore to fetch all Expenses values from 1st of January till 31st of December of the selected year as seen in Figure 4.95.

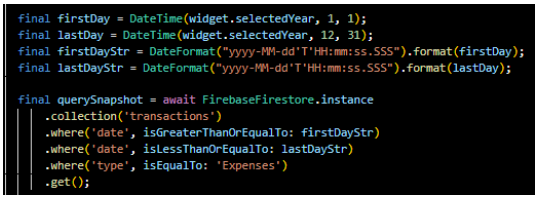


Figure 4.95: Query Expenses

3) The function will loop through the results which are returned to aggregate the total expenses value per month as seen in Figure 4.96.

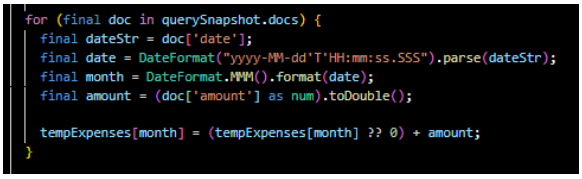


Figure 4.96: Loop result expenses total

4) Whenever there is a transaction added or a new year is selected at the drop down, the values on the chart UI will be updated with the new values as seen in Figure 4.97.

```
if (!mounted) return;
setState(() {
  monthlyExpenses = tempExpenses;
  isLoading = false;
});
```

Figure 4.97: SetState chart

The source code for the UI which builds the bar graph is as follows: -

1) The Bar graph will show a loading indicator when while the latest data is being fetched from firestore as seen in Figure 4.98.

```
if (isLoading) {
  return const Center(child: CircularProgressIndicator());
}
```

Figure 4.98: Load Bar Graph

2) The main container which holds the bar graph along with the Spending Trend title is as shown in Figure 4.99. It wraps the entire graph in a styled card with light background, rounded corners and a shadow effect. This is to help separate the widget between each other.

```
return Container(
  margin: const EdgeInsets.only(top: 20),
  padding: const EdgeInsets.all(16),
  decoration: BoxDecoration(
    color: Colors.grey[200],
    borderRadius: BorderRadius.circular(20),
    boxShadow: [
      BoxShadow(
        color: Colors.grey.withAlpha(102),
        spreadRadius: 2,
        blurRadius: 5,
        offset: const Offset(0, 3),
      ), // BoxShadow
    ],
  ), // BoxDecoration
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      if (isTranslationLoaded)
        Text(
          translatedSpendingTrend, // Display translated "Spending Trend"
          style: const TextStyle(
            fontSize: 18,
            color: Colors.black,
            fontWeight: FontWeight.bold,
          ), // TextStyle
        ), // Text
```

Figure 4.99: Bar Graph Main Container

3) First is the chart container. The SizedBox sets the height of the graph to 200 pixels and inside it, BarChart is then used to render the bar chart visual using the details configured provided by BarChartData as seen in Figure 4.100.

```
SizedBox(
  height: 200,
  child: BarChart(
    BarChartData(
```

Figure 4.100: Chart Container

4) Alignment and scaling of the bar as seen in Figure 4.101.

```
BarChartData(
  alignment: BarChartAlignment.spaceAround,
  maxY: _getMaxY(values) * 1.2,
```

Figure 4.101: Alignment and Scaling Bar Chart

5) Interactive tooltip where user is able to have touch interaction on bar. There will be a tooltip that displays month and value when user taps or hover on a bar as seen in Figure 4.102.

```
barTouchData: BarTouchData(
  enabled: true,
  touchTooltipData: BarTouchTooltipData(
    getTooltipItem: (group, groupIndex, rod, rodIndex) {
      final month = translatedMonths[months[group.x.toInt()]] ?? months[group.x.toInt()];
      return BarTooltipItem(
```

Figure 4.102: Bar Chart Interactive ToolTip

6) Bottom Titles which is the X-Axis that shows the month under each bar as seem in Figure 4.103.

```
titlesData: FlTitlesData(
  show: true,
  bottomTitles: AxisTitles(
    sideTitles: SideTitles(
      showTitles: true,
      getTitlesWidget: (value, meta) {
        return Padding(
          padding: const EdgeInsets.only(top: 8.0),
          child: Text(
            translatedMonths[months[value.toInt()]] ?? months[value.toInt()],
            style: const TextStyle(
              fontSize: 10,
              fontWeight: FontWeight.bold,
              color: Colors.black,
            ), // TextStyle
          ), // Text
        ); // Padding
      },
      reservedSize: 38,
    ), // SideTitles
  ), // AxisTitles
```

Figure 4.103: X-axis Bar Chart

7) Left titles which is the Y-Axis that shows expense value in the selected currency and values are formatted as integers as seen in Figure 4.104.

```
  leftTitles: AxisTitles(
    sideTitles: SideTitles(
      showTitles: true,
      getTitlesWidget: (value, meta) {
        return Text(
          '$currencySymbol ${value.toInt()}',
          style: const TextStyle(
            fontSize: 10,
            color: Colors.black,
          ), // TextStyle
        ); // Text
      },
      reservedSize: 40,
    ), // SideTitles
  ), // AxisTitles
```

Figure 4.104: Y-Axis Bar Chart

8) Add a black boarder around the chart Area as seen in Figure 4.105.

```
borderData: FlBorderData(
  show: true,
  border: Border.all(color: Colors.black,),
), // FlBorderData
```

Figure 4.105: Black Boarder Bar Chart

9) Add Grids lines to enhance readability of each month's value and help compare the bar heights with better visualisation as seen in Figure 4.106.

```
gridData: FlGridData(show: true),
```

Figure 4.106: Grid Lines Bar Chart

10) Displays the main data in the bar along with unique colour for each month based on _getBarColor() as seen in Figure 4.107.

```
barGroups: List.generate(
  months.length,
  (index) => BarChartGroupData(
    x: index,
    barRods: [
      BarChartRodData(
        toY: values[index],
        color: _getBarColor(months[index]),
        width: 16,
        borderRadius: BorderRadius.circular(4),
      ), // BarChartRodData
    ],
  ), // BarChartGroupData
), // List.generate
```

Figure 4.107: Bar chart main data

11) Each month has its own dedicated colour to differentiate values as seen in Figure 4.108.

```
Qodo Gen: Options | Test this method
Color _getBarColor(String month) {
  switch (month.toLowerCase()) {
    case 'jan': return Colors.red.shade400;
    case 'feb': return Colors.orange.shade400;
    case 'mar': return Colors.yellow.shade600;
    case 'apr': return Colors.green.shade400;
    case 'may': return Colors.teal.shade400;
    case 'jun': return Colors.blue.shade400;
    case 'jul': return Colors.indigo.shade400;
    case 'aug': return Colors.purple.shade400;
    case 'sep': return Colors.pink.shade400;
    case 'oct': return Colors.brown.shade400;
    case 'nov': return Colors.cyan.shade400;
    case 'dec': return Colors.deepOrange.shade400;
    default: return Colors.grey.shade400;
  }
}
```

Figure 4.108: _getBarColor function

The Third and last widget on the home module is the recent transaction widget as seen in Figure 4.109. The widget will display the three most recent transactions for both expenses and income queried from Firestore. The transaction details will include icon, category names, remark, formatted amount and timestamps. The filter is not applied here

since it shows the latest regardless of the selected year and month. It automatically applies update in real time when a new update is added to firestore.

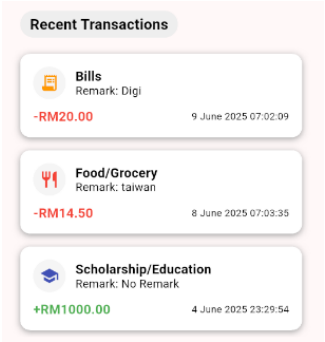


Figure 4.109: Recent Transaction

1) Section header for Recent transactions displayed in horizontal row with rounded styling and padding as seen in Figure 4.110.

```
Container(
  padding: const EdgeInsets.symmetric(horizontal: 12, vertical: 6),
  decoration: BoxDecoration(
    color: Colors.grey[200],
    borderRadius: BorderRadius.circular(20),
  ), // BoxDecoration
```

Figure 4.110: Section header for recent transaction

2) Next is StreamBuilder which is a flutter widget that automatically listens to a stream of data and rebuilds the UI when new data comes in. It queries from Firestore transaction collection, and order by the data in descending order, with limit to 3 transactions as seen in Figure 4.111.

```
StreamBuilder<QuerySnapshot>(
  stream: FirebaseFirestore.instance
      .collection('transactions')
      .orderBy('date', descending: true)
      .limit(3)
      .snapshots(),
```

Figure 4.111: StreamBuilder latest transaction

3) Each transaction is rendered using _buildTransactionCard(...) where data displayed are icon, category name, remark label, amount and formatted date and time. Each card is styled with a clean and colour coded based on category and transaction type as seen in Figure 4.112.

```
Widget _buildTransactionCard(
  ), // BoxDecoration
  child: Column(
    children: [
      Row(
        children: [
          CircleAvatar(
            backgroundColor: Colors.grey[100],
            child: Icon(getCategoryIcon(data['category']), color: _getCategoryColor(data['category'])),
          ), // CircleAvatar
          const SizedBox(width: 12),
          Expanded(
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(translatedCategory,
                 style: const TextStyle(
                  fontSize: 16,
                  fontWeight: FontWeight.bold,
                  color: Colors.black,
                  ) // TextStyle
                  ), // Text
                Text(
                  '$translatedRemarkLabel: ${translatedRemark.isNotEmpty ? translatedRemark : translatedNoRemark}',
                  style: TextStyle(fontSize: 14,color: Colors.black),
                  ), // Text
              ],
            ), // Column
          ), // Expanded
        ],
      ), // Row
      const SizedBox(height: 12),
      Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: [
          Text(
            '${isIncome ? '+' : '-'}$currencySymbol${amount.toStringAsFixed(2)}',
            style: TextStyle(
              fontSize: 16,
              fontWeight: FontWeight.bold,
              color: isIncome ? Colors.green : Colors.red,
            ), // TextStyle
          ), // Text
          Text(
            '$formattedDate $formattedTime',
            style: TextStyle(fontSize: 12, color: Colors.black),
          ), // Text
        ],
```

Figure 4.112: Recent transaction _buildTransactionCard(...)

4.4.2 Transaction Module

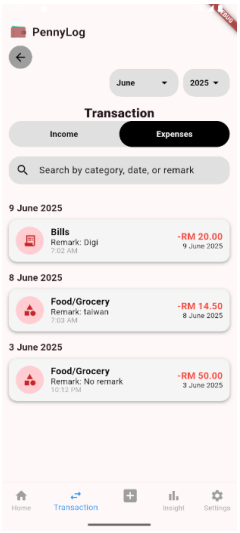


Figure 4.113: Transaction Page

The next module is the transaction module which is the transaction page as seen in Figure 4.113 for displaying all the transactions called transaction_page.dart. Like the home module, the transaction module has a month and year drop down to filter the information. Below the drop down, there is an Income and Expenses Toggle which allow users to view the financial records according to its type.

```
onTap: () => setState(() => selectedType = 'Income'),
```

Figure 4.114: Income Toggle

```
onTap: () => setState(() => selectedType = 'Expenses'),
```

Figure 4.115: Expense Toggle

Based on Figure 4.114 and Figure 4.115 respectively, when the toggle is tapped, the selectedType will be updated using setState(). The UI will update to change towards the selected transaction type.

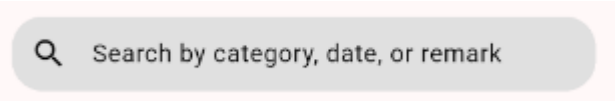


Figure 4.116: Search Transaction

Below the toggle is the search bar as seen in Figure 4.116. Every time users search something by typing in the TextField, onChanged it updates the searchQuery as seen in Figure 4.117.

```
onChanged: (value) {
  setState(() {
    searchQuery = value.trim().toLowerCase(); // Update search query
  });
```

Figure 4.117: TextField onChanged

The data will be fetched from Firestore as seen in Figure 4.118. It will use getDeviceUUID() to identify the device its query for and fetches the transaction document under the device in firestore. It will filter based on the selected type using .where(), order by date in descending order and it will listen to real-time updates with. snapshot() if there are changes to the query.

```
Expanded(
  child: FutureBuilder<String>(
    future: getDeviceUUID(),
    builder: (context, uuidSnapshot) {
      if (uuidSnapshot.connectionState != ConnectionState.done) {
        return const Center(child: CircularProgressIndicator());
      }
      if (uuidSnapshot.hasError || !uuidSnapshot.hasData) {
        return const Center(child: Text('Error loading device UUID'));
      }

      final uuid = uuidSnapshot.data!;
      final transactionStream = FirebaseFirestore.instance
          .collection('transactions')
          .doc(uuid)
          .collection('transaction')
          .where('type', isEqualTo: selectedType)
          .orderBy('date', descending: true)
          .snapshots();
```

Figure 4.118: Fetch from firestore the search

Next the transaction which are fetched from firestore will be preprocess before it is filtered as seen in Figure 4.119.

```
final filtered = snapshot.data!.docs.where((doc) {
  final data = doc.data() as Map<String, dynamic>;
  final date = DateTime.parse(data['date']);
  final rawRemark = data['remark']?.trim();
  final remark = (rawRemark?.isNotEmpty == true) ? rawRemark!.toLowerCase() : translatedNoRemark.toLowerCase();
  final category = data['category']?.toLowerCase() ?? '';
  final formattedDate = '${date.day} ${translatedMonths[DateFormat.MMMM().format(date)]} ${date.year}'.toLowerCase();
```

Figure 4.119: filter transaction using searchQuery

The final filtered = snapshot.data!.docs.where((doc) will loop through all the transaction documents fetched from firestore and filter them based on search input and selected month as well as year. Then final data = doc.data() as Map<String, dynamic>; will convert each firestore document into usable Map that allows access to field like date, remark and category. Following that, final date = DateTime.parse(data['date']); will convert the date string from firestore into DateTime object which is useful for month/year filtering and formatting the display. The final rawRemark = data['remark']?.trim(); will get the remark string from the transaction, .trim() removes any leading/trailing spcaes, ?. ensure no crash if remark is null. Then final remark = (rawRemark?.isNotEmpty == true) ? rawRemark!.toLowerCase() : translatedNoRemark.toLowerCase(); will check if remark has content, if yes it converts to lowercase, if empty/null, it fallback to a translated “No remark” string will also be convert to lowercase. The final category = data['category']?.toLowerCase() ?? ''; will convert the category field to lowercase and if its null, it will fallback to an empty string. The final formattedDate = '${date.day}${translatedMonths[DateFormat.MMMM().format(date)]} ${date.year}'.toLowerCase(); will build a readable date string like “9 June 2025” and it uses a translatedMonths that supports multilingual and converts the whole date string to lowercase as well.

Then for the filtering logic is that it will check if any of the following, which is the category, remark and formattedDate match the search query. The transaction must also match the selected month and year from the drop down as seen in Figure 4.120.

```
// Filter by search query
return (category.contains(searchQuery) ||
        remark.contains(searchQuery) ||
        formattedDate.contains(searchQuery)) &&
       date.month == months.indexOf(selectedMonth) + 1 &&
       date.year == selectedYear;
}).toList();
```

Figure 4.120: Filter searchQuery

Then after filtering, the transaction will be grouped by date in the UI as seen in Figure 4.121.

```
final grouped = <String, List<DocumentSnapshot>>{};
for (var doc in filtered) {
  final data = doc.data() as Map<String, dynamic>;
  final date = DateTime.parse(data['date']);
  final formattedDate = '${date.day} ${translatedMonths[DateFormat.MMMM().format(date)]} ${date.year}';
  grouped.putIfAbsent(formattedDate, () => []).add(doc);
}
```

Figure 4.121: Display transaction

```
child: StreamBuilder<QuerySnapshot>(
  stream: FirebaseFirestore.instance
      .collection('transactions')
      .where('type', isEqualTo: selectedType)
      .orderBy('date', descending: true)
      .snapshots(),
```

Figure 4.122: Stream Builder for transaction

After the search bar, the transactions displayed in the transaction module are fetched using StreamBuilder as seen in Figure 4.122. The stream is connected to Firestore which will listen for real-time updates, and it is filtered based on the selectedType. The results are also filtered by the dates that matches the selected month and year in descending order with the latest transaction at the top. The benefit is there is no need to reload the whole page when using Streambuilder if a new transaction comes in.

```dart
final grouped = <String, List<DocumentSnapshot>>{};
for (var doc in filtered) {
  final date = DateTime.parse(doc['date']);
  final formattedDate = '${date.day} ${translatedMonths[DateFormat.MMMM().format(date)]} ${date.year}';
  grouped.putIfAbsent(formattedDate, () => []).add(doc);
}
```

Figure 4.123: Group and Sort by date

Based on Figure 4.123, first is the declaration of an empty grouping map. It will act as a container to store all the transaction which are grouped by the day. It will loop through the filtered transaction which is the for (var doc in filtered) , then the date is converted to a readable string format which is like 30 April 2025 using the '${date.day} ${translatedMonths[monthName]} ${date.year}'. The grouped.putIfAbsent(...) will check if the particular date key exist or not, if not it will create an empty list for that date. Then the new transaction is added to that list. This is to ensure all similar dates are grouped together.

```dart
return ListView(
  children: grouped.entries.map((entry) {
    return Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Padding(
          padding: const EdgeInsets.fromLTRB(16, 12, 16, 4),
          child: Text(
            entry.key, // date header
            style: const TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
          ), // Text
        ), // Padding
        ...entry.value.map<Widget>((doc) {
          final data = doc.data() as Map<String, dynamic>;
          final String remark = data['remark'] ?? '';
          final langProvider = Provider.of<LanguageProvider>(context, listen: false);

          return FutureBuilder<String>(
            future: remark.isNotEmpty
                ? langProvider.translate(remark)
                : Future.value(''),
            builder: (context, snapshot) {
              final translatedRemarkText = snapshot.connectionState == ConnectionState.done
                  ? snapshot.data ?? remark
                  : remark; // fallback while loading

              return buildStyledTransactionCard(doc, currencySymbol, translatedRemarkText);
            },
          ); // FutureBuilder
```

Figure 4.124: Ui build for transaction list

Then the transaction list is displayed using ListView as seen in Figure 4.124 to build the UI. Each entry.key is the date string which for example 30 April 2025 in Figure 4.125 that will be the date header that separates each of the list.

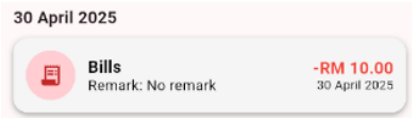


Figure 4.125: Entry.key

Each of the entry.value will be the transaction list for the date. Each transaction is shown in a transaction card style for a more consistent styling and view through buildStyledTransactionCard as shown in Figure 4.126.

```
Qodo Gen: Options | Test this method
Widget buildStyledTransactionCard(DocumentSnapshot doc, String currencySymbol, String translatedRemarkText) {
  final data = doc.data() as Map<String, dynamic>;
  final date = DateTime.parse(data['date']);
  final amount = double.tryParse(data['amount'].toString()) ?? 0.0;
  final category = data['category'];
  final isIncome = data['type'] == 'Income';
```

Figure 4.126: buildStyledTransactionCard

Transaction card is a UI component that displays the details of a single transaction such as icon, category name, remark, amount and date.

```
return InkWell(
  onTap: () {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => ManageTransaction(transaction: doc),
      ), // MaterialPageRoute
    );
  },
```

Figure 4.127: InkWell

Inkwell allow user to tap into the card, and it will navigate user to view in more detail of the transaction by navigating user to the ManageTransaction page called manage_transaction.dart where user can view, edit, delete and save the transaction as seen in Figure 4.127.

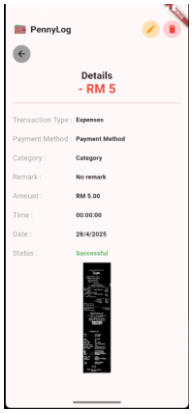


Figure 4.128: View transaction details

When user click to view each transaction card, the _buildDetailRow which is a helper method, it is used to recreate the reusable row layout for each of the transaction details field. It will standardise the design for each row as seen in Figure 4.128. The _buildDetailRow is used for both viewing mode and editing mode.

Figure 4.129: paymentMethod _buildDetailRow

An Example is the _buildDetailRow for the paymentType in Figure 4.129. The paymenttype variable will store all the method of payment for the transactions such as Cash, Credit Card, Debit Card, E-Wallet, Check, etc.

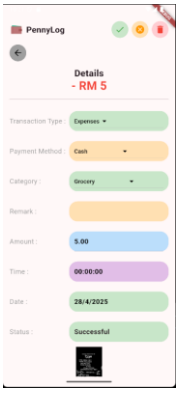


Figure 4.130: Transaction Edit Mode

When user want to edit the transaction and we click on the edit button, the if isEditing == true, will allow a dropdown menu to be shown which allows user to change the payment method as seen in Figure 4.130. If isEditing == false, user is on the viewing mode. The DropdownButton will always show the current paymentType. If the paymentType is not found, then by default it will show the first option paymentOptions[0]. When user chooses a new payment method, it triggers onChanged, where the paymentType will be updated using setState(). For the style, color: Colors.orange[100] light orange background with rounded corners

borderRadius: BorderRadius.circular is used. User will be able to edit all the information including choosing a new image as well as seen in Figure 4.131.

Figure 4.131: Choose New Image

When user want to change the image and click the change image button label, the onPressed is triggered for _pickNewImage function where user can pick a new image from gallery as shown in Figure 4.132.

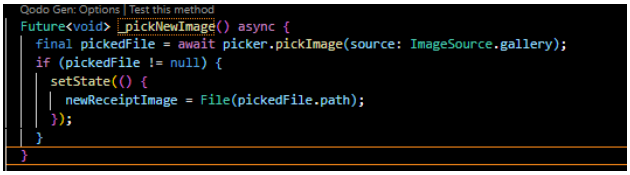


Figure 4.132: _pickNewImage function

Then the current image will be replaced by the newReceiptImage for the particular transaction as seen in Figure 4.132. In viewing mode, when isEditing ==false, when user wants to view the image more closely, the buildReceiptImage() widget will be triggered to display the image when user click on the image. Similarly, when in editing mode, when isEditing == false, buildReceiptImage() widget is also triggered to show the new image that was chosen.

```
Qodo Gen: Options | Test this method
Widget buildReceiptImage() {
 try {
  // Case 1: New image picked during editing
  if (newReceiptImage != null) {
   return GestureDetector(
    onTap: () async {
     final bytes = await newReceiptImage!.readAsBytes();
     if (!mounted) return;
     await _showImagePreviewDialog(context, bytes);
    },
    child: ClipRRect(
     borderRadius: BorderRadius.circular(10),
     child: Image.file(
      newReceiptImage!,
      height: 250,
      width: double.infinity,
      fit: BoxFit.contain
     ), // Image.file
    ), // ClipRRect
   ); // GestureDetector
   // Case 2: Existing image from database
  } else if (receiptImageUrl.isNotEmpty) {
   final bytes = base64Decode(receiptImageUrl);
   return GestureDetector(
    onTap: () => _showImagePreviewDialog(context, bytes),
    child: ClipRRect(
     borderRadius: BorderRadius.circular(10),
     child: Image.memory(
      bytes,
      height: 250,
      width: double.infinity,
      fit: BoxFit.contain
     ), // Image.memory
    ), // ClipRRect
   ); // GestureDetector
  }
 } catch (_) {}
 // Case 3: No image exists
 return Text(noreceiptLabel, style: TextStyle(color: Colors.grey));
}
```

Figure 4.133: buildReceiptImage()

If new image is picked and it's not empty in newReceiptImage != null, it shows the newly picked image file. Else if an existing image exist where its not empty in receiptImageUrl.isNotEmpty, it decodes the base64 string from firestore and display the existing image. Then if there is no image, it will display as 'No receipt available'. The image will be wrapped in a GestureDetector, therefore by tapping the image to view the image, it will trigger a zoomable preview which uses the _showImagePreviewDialog() as seen in Figure 4.133.

Figure 4.134:_showImagePreviewDialog()

When the image is tapped, it opens a model dialog box by using showDialog() as seen in Figure 4.134.

```
double scale = 1.0;

await showDialog<void>(
  context: context,
  barrierDismissible: true,
  builder: (BuildContext dialogContext) {
    return StatefulBuilder(
```

Figure 4.135: showDialog()

By default, the image scale is set at 1.0. Using the StatefulBuilder, it allows dynamic updates to happen in the dialog such as the changing of the zoom scale as seen in Figure 4.135.

```
child: Transform.scale(
  scale: scale,
  child: Image.memory(
    imageBytes,
    fit: BoxFit.contain,
  ), // Image.memory
```

Figure 4.136: Transform.scale

The Transform.scale will allow users to zoom in and out of the receipt image by wrapping it in 'Image.memory' seen in Figure 4.136.

```
onPressed: () {
  setState(() {
    scale = (scale + 0.1).clamp(0.5, 3.0);
  });
```

Figure 4.137: setState for zoom

The zoom in and out button will call setState seen in Figure 4.137 to adjust the scale which are clamped between min 0.5 and maximum 3.0. There are two buttons available where 'cancel' will close the dialog using Navigator.pop(), and 'Reset Zoom' will reset the zoom scale to 1.0.

```
maxHeight: MediaQuery.of(context).size.height * 0.7,
```

Figure 4.138: Dialog height limit

The dialog has a height limit of 70% of the screen as seen in Figure 4.138.

```
child: ClipRRect(
  borderRadius: BorderRadius.circular(12),
  child: SingleChildScrollView(
    child: Transform.scale(
      scale: scale,
      child: Image.memory(
        imageBytes,
        fit: BoxFit.contain,
      ), // Image.memory
```

Figure 4.139: ClipRRect

Then it will use ClipRRect to create a rounded image corner to the image container and Image.memory(...) will be used to show the base65 string images seen in Figure 4.139. The zoom buttons (+/-) are at the top right corner of the image using 'Positioned' for it to be on the top right corner and 'Stack' for it to be on the image.

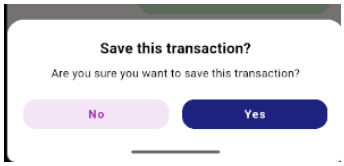


Figure 4.140: Save transaction

When user has edited the transaction, they have the option to save it or not as seen in Figure 4.140. The save transaction dialog will be triggered when they click the save button.

```
if (confirm == true && _formKey.currentState!.validate()) {
  try {
    String updatedImage = receiptImageUrl;
    if (newReceiptImage != null) {
      final bytes = await newReceiptImage!.readAsBytes();
      updatedImage = base64Encode(bytes);
    }
    await widget.transaction.reference.update({
      'type': selectedType,
      'payment_type': paymentType,
      'category': category,
      'remark': _remarkController.text,
      'amount': double.parse(_amountController.text),
      'date': date.toIso8601String(),
      'receipt_image': updatedImage,
```

Figure 4.141: Update transaction in database

The save button actually triggers the _updateTransaction() function that will show a pop up using the showModelBottomSheet where it shows the pop up as a RoundedRectangleBorder with two buttons which is yes or no. If yes, then confirm == true, with the changes made, in Firestore for transaction collection, the information will be updated as seen in Figure 4.141. If not, the information does not change.

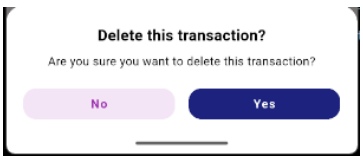


Figure 4.142: Delete Transaction

Users will also have the option to delete the transaction as seen in Figure 4.142. Similarly, like the save transaction, the delete transaction will trigger the _deleteTransaction() function.

```
if (confirm == true) {
  await widget.transaction.reference.delete();
  if (!mounted) return;
  Navigator.pop(context);
}
```

Figure 4.143: widget.transaction.reference.delete()

If user click yes, confirm == true, then it will delete the transaction document from firestore. The widget.transaction is the DocumentSnapshot passed to this widget. DocumentSnapshot is a Firebase Firestore class that will represent a document from a firestore database. It contains actual document data, reference to document location, metadata, check if document exist or provide methods to access document field. The. reference gets the document reference in Firestore. The .delete() removes the document from the database. Then the Navigator.pop(context) will return to the previous screen after successfully deleted as seen in Figure 4.143.

4.4.3 New Transaction Module

The New Transaction Module are separated into two flows for user which is user will be able to add new transaction manually or automatically using the OCR and NLP functionality. When user click on the "+" icon in the bottom navigation bar as seen in Figure 4.144, a pop up will appear, where user can choose either income or expenses as seen in Figure 4.145.

Figure 4.144: Bottom Navigation Bar

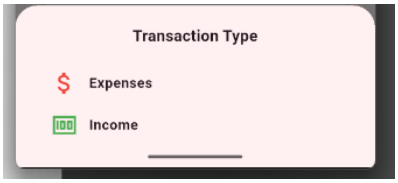


Figure 4.145: Transaction Type Pop Up

When user choose either one, they will be brough into a form to be filled in the transaction details. User can toggle between expenses and income depending on what transaction they wish to add as seen in Figure 4.146.

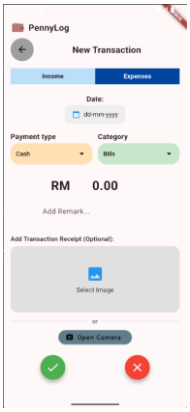


Figure 4.146: Transaction_detail_page.dart

If user wish to use the manual mode, we will just need to fill in all the fields and click on the save button below and the saveTransaction() function will save the necessary information to the database under transactions collection as seen in Figure 4.147.

```
Qodo Gen: Options | Test this method
Future<void> saveTransaction() async {
  try {
    final uuid = await getDeviceUUID();
    final doc = FirebaseFirestore.instance.collection('transactions').doc();

    String? base64Image;
    if (receiptImageBytes != null) {
      base64Image = base64Encode(receiptImageBytes!);
    } else if (receiptImage != null) {
      final bytes = await receiptImage!.readAsBytes();
      base64Image = base64Encode(bytes);
    }

    await doc.set({
      'device_uuid': uuid, // Storing UUID in the document too
      'type': selectedType,
      'date': selectedDate?.toIso8601String() ?? '',
      'payment_type': paymentType,
      'category': category,
      'amount': double.tryParse(amountController.text) ?? 0.0,
      'remark': remarkController.text.trim(),
      'receipt_image': base64Image ?? '',
      'method': widget.method,
      'timestamp': FieldValue.serverTimestamp(),
    });
```

Figure 4.147: saveTransaction()

If user do not wish to fill in the fields manually, user can perform automation to use the OCR and NLP functionality by clicking the upload image or open camera to take picture as seen in the form.

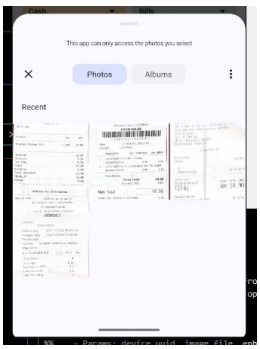


Figure 4.148: Upload Image

If user choose to upload image as seen in Figure 4.148, they can select any image that is available in the gallery. When they have selected the image, they wanted or if they use the camera to capture the image, the image can be cropped as seen in Figure 4.149.

Figure 4.149: crop photo

The function cropSelectedImage will be called, and it will pass the selected image to the ImageCropper() which is a using the image_cropper package for Dart to crop images.

```
Qodo Gen: Options | Test this method
Future<File?> cropSelectedImage(String imagePath) async {
  try {
    final croppedFile = await ImageCropper().cropImage(
      sourcePath: imagePath,
      compressFormat: ImageCompressFormat.jpg,
      uiSettings: [
        AndroidUiSettings(
          toolbarTitle: 'Crop Image',
          lockAspectRatio: false,
          initAspectRatio: CropAspectRatioPreset.original,
        ),
        IOSUiSettings(
          title: 'Crop Image',
        ),
      ],
    );
    return croppedFile != null ? File(croppedFile.path) : null;
  } catch (e) {
    print("❌ Error in cropping: $e");
    return null;
  }
}
```

Figure 4.150: cropSelectedImage function

The function will take image path as input using String imagePath and return file Future<File?> either to be a cropped image or null if the cropping is cancelled or fails. The final croppedFile = await ImageCropper().cropImage(sourcePath: imagePath, will open the cropping UI using the image as the provided path and compressFormat: ImageCompressFormat.jpg, this will compress the output as JPG format. The return croppedFile != null ? File(croppedFile.path) : null; shows that if the image is successfully cropped, the result path will be converted into a File object and return it. If user cancel or something fails, it will return null.

Then, a pop up will be shown using showDialog() asking user if they want to perform enhancement as seen in Figure 4.151.

Figure 4.151: Preview Image

When the user chooses to perform enhancement, and click on the “Yes” button, the application makes a HTTP POST request to the endpoint called /auto-enhance-image/.

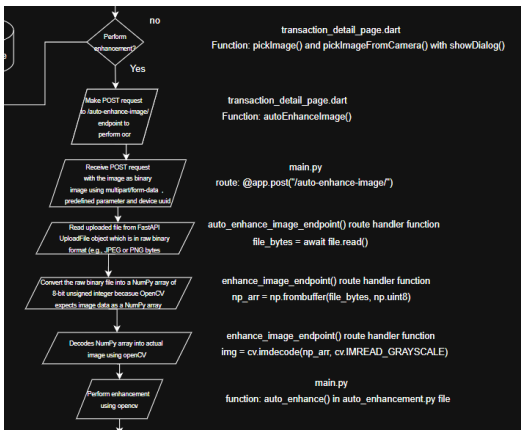


Figure 4.152: Flowchart part 1

As seen in Figure 4.152, when user click on the “yes” button, the function autoEnhanceImage() will make a HTTP POST request to the /auto-enhance-image/ endpoint to perform enhancement on the selected image. The backend in main.py, the server route @app.post("/auto-enhance-image/") receives the POST request with the image as binary using the multipart/form-data and the device uuid. The function auto_enhance_image_endpoint() which is the route handler for the POST request will read the uploaded file from the FastAPI UploadFile object for the image which is in raw binary format using file_bytes = await file.read(). Then np_arr = np.frombuffer(file_bytes, np.uint8) will convert the raw binary file unto a NumPy array of 8-bit unsigned integer because the OpenCV which is the python image processing library that requires imaga data as a Numpy array. After that img = cv.imdecode(np_arr, cv.IMREAD_GRAYSCALE) will decode the NumPy array into actual image using OpenCV. Then the function auto_enhance() will be called and the image will be passed for enhancement using OpenCV as seen in Figure 4.153.

```
enhanced_img = auto_enhance(img)
```

Figure 4.153: Auto_enhance

In the auto_enhance() function, it will automatically apply enhancement depending on the image properties as seen in Figure 4.154.

```
force_contrast = brightness > 200 or hist_skew > 220

if brightness < 100:
    gamma = 1.5 if brightness < 70 else 1.2
    invGamma = 1.0 / gamma
    table = np.array([(i / 255.0) ** invGamma * 255 for i in range(256)]).astype("uint8")
    result = cv2.LUT(result, table)

if contrast < 50 or force_contrast:
    lab = cv2.cvtColor(result, cv2.COLOR_BGR2LAB)
    l, a, b = cv2.split(lab)
    clahe = cv2.createCLAHE(clipLimit=2.5, tileGridSize=(8, 8))
    cl = clahe.apply(l)
    result = cv2.cvtColor(cv2.merge((cl, a, b)), cv2.COLOR_LAB2BGR)

if sharpness < 100 or force_contrast:
    kernel = np.array([[0, -1, 0],
                       [-1, 5, -1],
                       [0, -1, 0]])
    result = cv2.filter2D(result, -1, kernel)
```

Figure 4.154: auto_enhance()

The function will determine if the image is dark, it will apply gamma correction to brighten the image. If the contrast is low, it will apply CLAHE (Contrast Limited Adaptive Histogram Equalisation). If the image is too blurry it will apply sharpening filter to make the words clearer on the image.

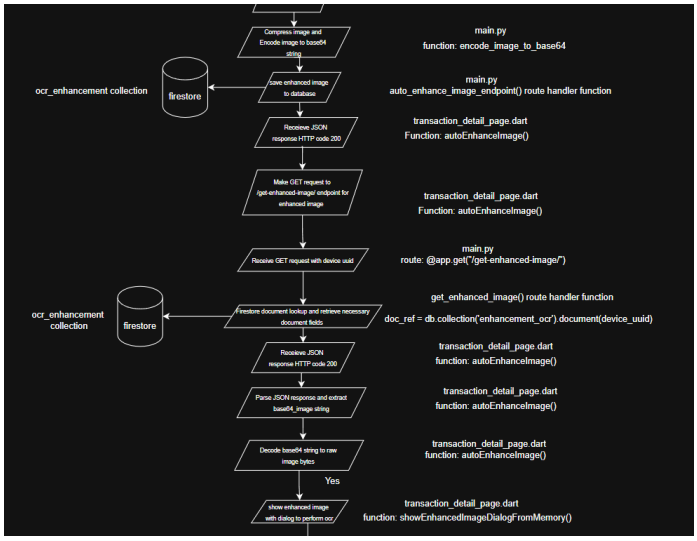


Figure 4.155: Flowchart part 2

After OpenCV performed the image enhancement as seen in Figure 4.155, the function encode_image_to_base64() will convert the image to base64 string so that the image can be saved into Firestore. The auto_enhance_image_endpoint()

route handler function will save the image into ocr_enhancement collection in Firestore. Then the backend will send a JSON HTPP code 200 which is a successful message to autoEnhanceImage() as seen in Figure 4.156.

```
// Fetch the enhanced image from Firestore using the doc_id
final getUri = Uri.parse('$baseUrl/get-enhanced-image/$docId');
final getResponse = await http.get(getUri);

if (getResponse.statusCode == 200) {
  final base64Image = json.decode(getResponse.body)['base64_image'];
  final bytes = base64Decode(base64Image);

  setState(() {
    receiptImage = null;              // No more using file
    receiptImageBytes = bytes;         // Store as memory
    enhancedReceiptDocId = docId;      // Save the doc id for OCR
  });
```

Figure 4.156: enhanced image

Then, the autoEnahnceImage() function will make a HTTP GET request to /get-enhanced-image/ endpoint for the enhanced image. The route @app.get("/get-enhanced-image/") receives the GET request with device uuid as parameter, then the get_enhanced_image() route handler function using doc_ref = db.collection('enhancement_ocr').document(device_uuid) which is Firestore document do a lookup and retrieve the necessary document field. The frontend will receive a JSON response HTTP code 200 for successful response for the enhanced image being passed as seen in Figure 4.157.

```
INFO:main:Processing image enhancement for device: 6273
a118-0fea-45cb-9c35-b8475f85edf0
INFO:main:Enhanced image saved for device: 6273a118-0fe
a-45cb-9c35-b8475f85edf0
INFO:     127.0.0.1:50063 - "POST /enhance-image/ HTTP/
1.1" 200 OK
INFO:main:Fetching enhanced image for device: 6273a118-
0fea-45cb-9c35-b8475f85edf0
INFO:     127.0.0.1:50069 - "GET /get-enhanced-image/62
73a118-0fea-45cb-9c35-b8475f85edf0 HTTP/1.1" 200 OK
```

Figure 4.157: HTTP code 200

The frontend will parse the JSON response and extract the base64-encoded image string, converts it to bytes and the bytes will represent the actual image data. The setState() will update the frontend with the enhancedPreviewBytes which is the decoded image bytes for display. The showEnhancedImageDialogFromMemory()

will display a pop-up dialog with the enhanced image and user is asked if they want to perform OCR on the enhanced image as shown in Figure 4.158.

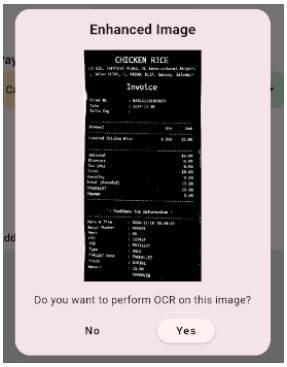


Figure 4.158: Perform OCR Dialog

If user choose not to perform OCR, then the empty transaction form with the enhanced image is shown. User then can choose to enter manually the transaction details, however if user choose to perform OCR and choose yes, the flow is as show in Figure 4.159.

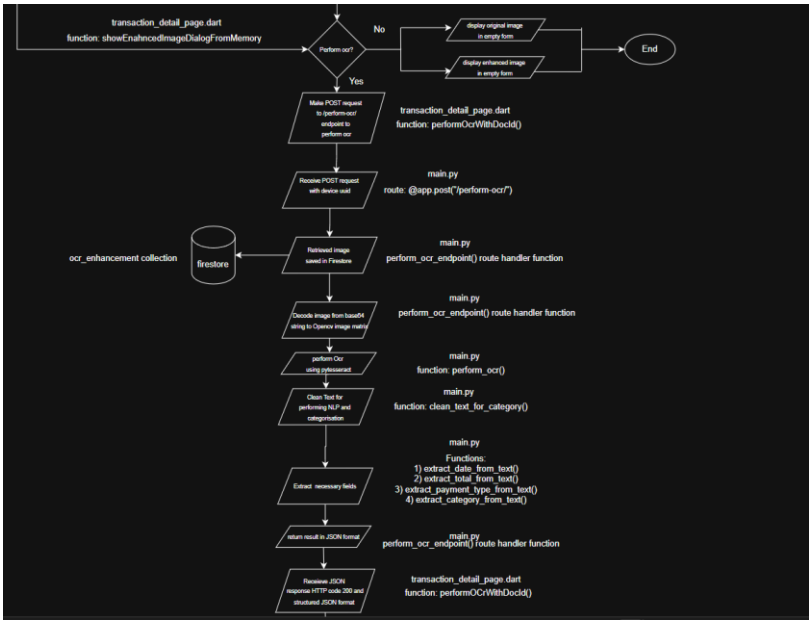


Figure 4.159: Flowchart part 4

The application will make a HTTP POST request to /perform-ocr/ endpoint using the function performOcrWithDocId. The route: @app.post("/perform-ocr/") receives the POST request with the necessary device uuid. Then the

perform_ocr_endpoint() route handler function will retrieve the image saved in ocr_enhancement collection in the database with the doc_id which is the device uuid as seen in Figure 4.160.

Figure 4.160: doc_id search for image

Then, the image will be decoded from the base64 string into OpenCV image. This is necessary because Firestore stores image as base64 string and OpenCV needs numpy arrays to process images. The OpenCV will help process the image into grayscale images as OCR works better with grayscale images. After that, OCR is performed on the image using perform_ocr() function for pyteserract. Then using clean_text_for_category() to clean the ocr extracted text for better categorisation by removing unnecessary spaces, special character, etc as seen in Figure 4.161.

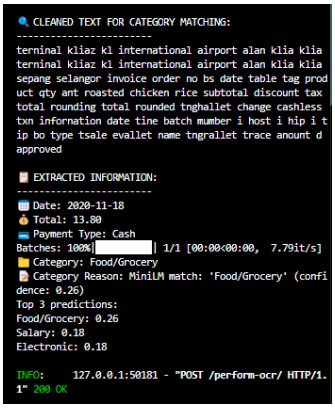
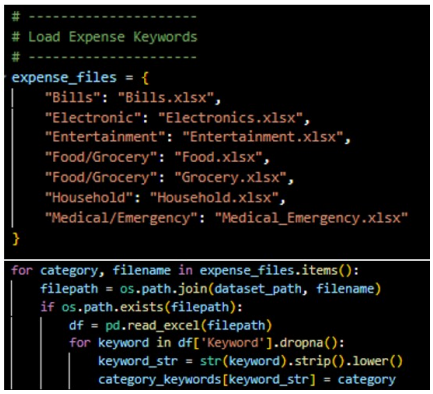


Figure 4.161: Clean text and categorisation

The respective functions will perform the extraction which is extract_date_from_text() for date, extract_total_from_text() for total, extract_payment_type_from_text() for payment type and extract_category_from_text() is where the NLP SentenceTransformer("all-MiniLM-L6-v2") will be used for categorisation.

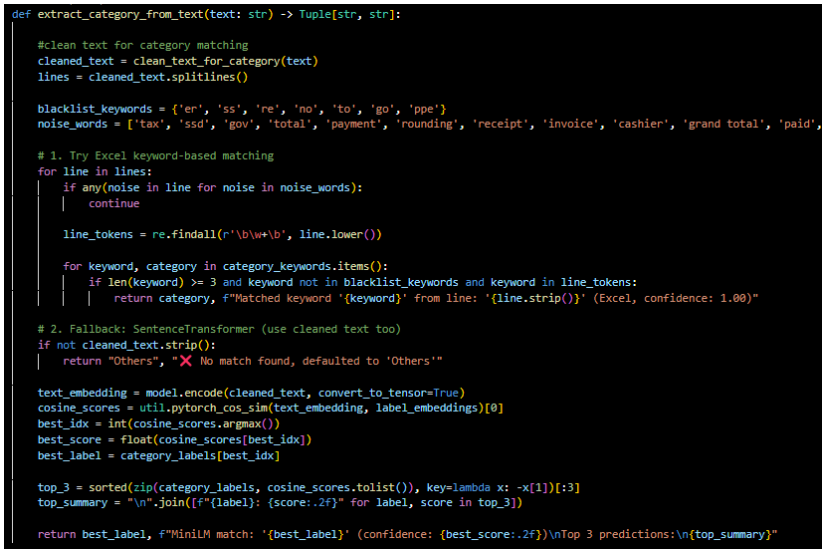


```
# ---------------------
# Load Expense Keywords
# ---------------------
expense_files = {
    "Bills": "Bills.xlsx",
    "Electronic": "Electronics.xlsx",
    "Entertainment": "Entertainment.xlsx",
    "Food/Grocery": "Food.xlsx",
    "Food/Grocery": "Grocery.xlsx",
    "Household": "Household.xlsx",
    "Medical/Emergency": "Medical_Emergency.xlsx"
}

for category, filename in expense_files.items():
    filepath = os.path.join(dataset_path, filename)
    if os.path.exists(filepath):
        df = pd.read_excel(filepath)
        for keyword in df['Keyword'].dropna():
            keyword_str = str(keyword).strip().lower()
            category_keywords[keyword_str] = category
```

Figure 4.162: Load Excel for NLP

For the NLP for category, first it will try to use keyword matching from the data provided in the excel in Figure 4.162.



```
def extract_category_from_text(text: str) -> Tuple[str, str]:

    #clean text for category matching
    cleaned_text = clean_text_for_category(text)
    lines = cleaned_text.splitlines()

    blacklist_keywords = {'er', 'ss', 're', 'no', 'to', 'go', 'ppe'}
    noise_words = ['tax', 'ssd', 'gov', 'total', 'payment', 'rounding', 'receipt', 'invoice', 'cashier', 'grand total', 'paid',

    # 1. Try Excel keyword-based matching
    for line in lines:
        if any(noise in line for noise in noise_words):
            continue

        line_tokens = re.findall(r'\b\w+\b', line.lower())

        for keyword, category in category_keywords.items():
            if len(keyword) >= 3 and keyword not in blacklist_keywords and keyword in line_tokens:
                return category, f"Matched keyword '{keyword}' from line: '{line.strip()}' (Excel, confidence: 1.00)"

    # 2. Fallback: SentenceTransformer (use cleaned text too)
    if not cleaned_text.strip():
        return "Others", "❌ No match found, defaulted to 'Others'"

    text_embedding = model.encode(cleaned_text, convert_to_tensor=True)
    cosine_scores = util.pytorch_cos_sim(text_embedding, label_embeddings)[0]
    best_idx = int(cosine_scores.argmax())
    best_score = float(cosine_scores[best_idx])
    best_label = category_labels[best_idx]

    top_3 = sorted(zip(category_labels, cosine_scores.tolist()), key=lambda x: -x[1])[:3]
    top_summary = "\n".join([f"{label}: {score:.2f}" for label, score in top_3])

    return best_label, f"MiniLM match: '{best_label}' (confidence: {best_score:.2f})\nTop 3 predictions:\n{top_summary}"
```

Figure 4.163: extract_category_from_text

If no keyword matches, then it will use the model to convert text to embedding meaning to 384-dimensional vectors using text_embedding = model.encode(cleaned_text, convert_to_tensor=True). Each words meaning will be captured in the numerical values. Similar words will have similar vector values. The categories predefined will also be converted to vector using label_embeddings = model.encode(category_labels, convert_to_tensor=True). So, each category has it vector value. If words have similar vector values meaning the higher the cosine similarity, the better the category matching. Then the text vector value is compared

with the category vector value using cosine_scores = util.pytorch_cos_sim(text_embedding, label_embeddings)[0]. The words with highest similarity score with the category will be the selected category as seen in Figure 4.163.

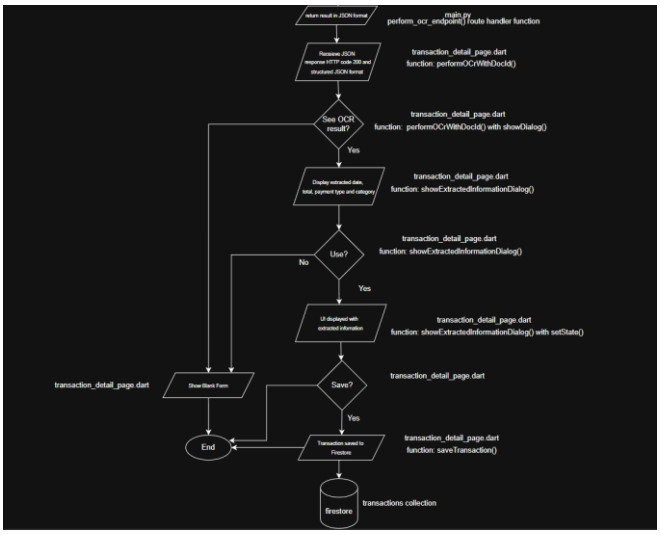


Figure 4.164: Flowchart part 5

Then the extracted informations are return in JSON format by the perform_ocr_endpoint() route handler function. The frontend function performOCrWithDocId() will receive the HTTP code 200 and the structured JSON format. The performOCrWithDocId() with showDialog() will show a pop up to ask user if they want to see the OCR result as shown in Figure 4.165.

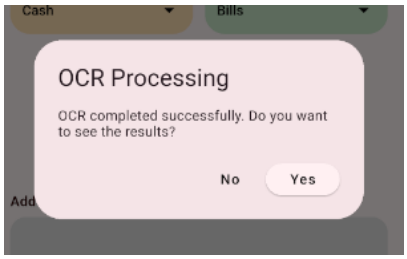


Figure 4.165: Show OCR result

If user choose yes, the showExtractedInformationDialog() function will display the extracted date, total, payment type and category as shown in Figure 4.166.

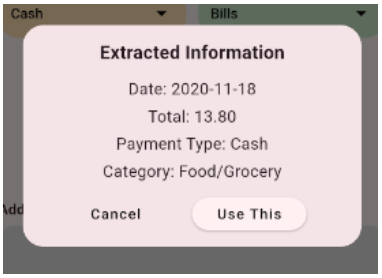


Figure 4.166: Extracted information pop up

If user choose "Use This" like in Figure 4.166 then the showExtractedInformationDialog() with setState() will update the form field with the necessary extracted information as shown in Figure 4.167 along with the enhanced image. User can choose to save the information if they wish which will trigger saveTransaction() function to save in database in transactions collection.

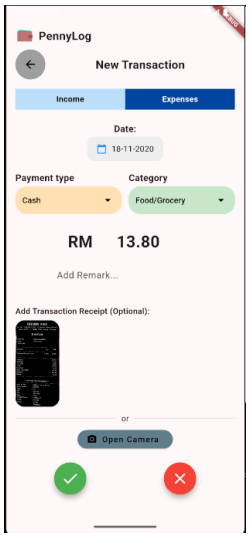


Figure 4.167: Extracted information in form

4.4.4 Insight Module

The insight module is where the time series forecasting using RNN LSTM model comes in. User will have access to a prediction model for their expenses for the past month in the year in the form of pie chart for better visualisation and the table will display the recommended expenses breakdown by expense category as seen in Figure 4.168. For budget optimisation where user have their own budget,

they can update the budget, and according to the original prediction, a new prediction with the budget will be updated.

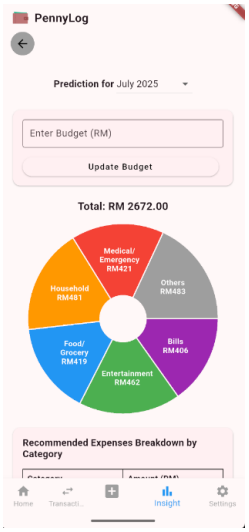


Figure 4.168: Insight Page

To perform the prediction, the model must be built first. The first process before the actual model building, we need to prepare data. For the data, random data are downloaded from Mockaroo which provide random data according to the field we customise in excel format. There are weights that can be applied to certain field to ensure certain value is heavier compared to others as seen in Figure 4.169.

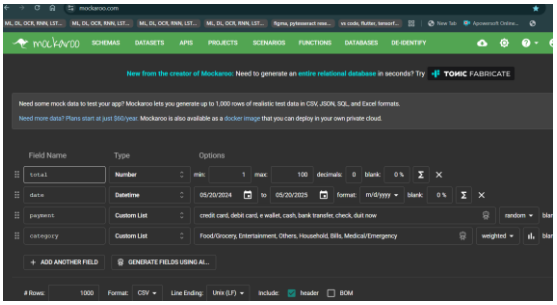


Figure 4.169: Mockaroo

Data are prepared from January 2018 to April 2025 and are saved in the "assets/expense_dataset" folder. After preparing the data, the model needs to be built. There are 5 steps to ensure the model is built properly.

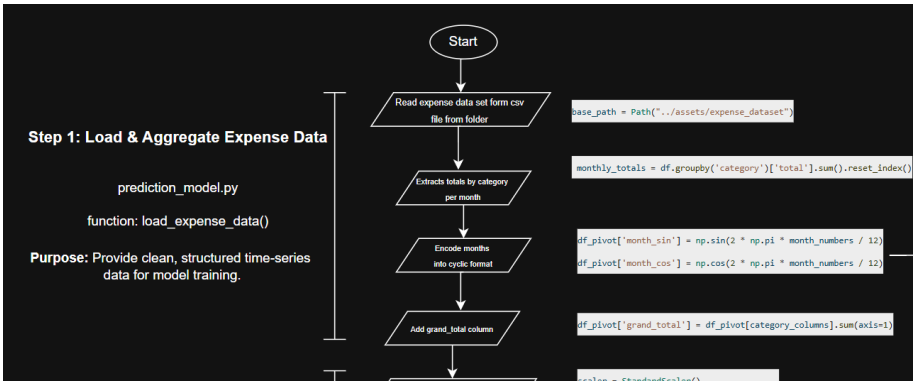


Figure 4.170: Step 1 Load and Aggregate Expense Data

The first step is to load and aggregate the expense the data as seen in Figure 4.170. Using the function load_expense_data(), the data grouped by year and month in csv format are processed and extracted. The total for each expense category is extracted and grouped together for the sum. The months are also encoded into cyclic format to ensure months like January and December who are close in the year cycle can be identified as well other yearly pattern. A grand total column is added to sum all the categories total to get the total for the month.

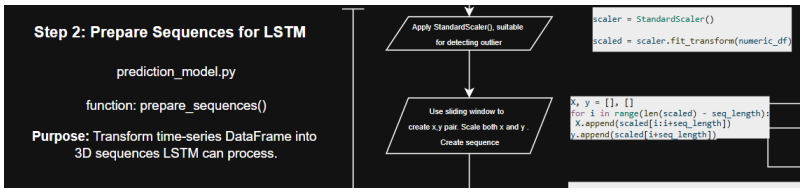


Figure 4.171: Step 2 Prepare Sequences for LSTM Model

The next step is to prepare sequence for the LSTM Model as seen in Figure 4.171. Since LSTM models are sensitive to the input ranges, to standardise the inputs to be in the same scale while still put into consideration any outlier for seasonal or new spending patterns, StandardScaler() are applied. By scaling the input value into the same range, it prevents larger input value from dominating the whole training process. Then sliding window are implemented where a seq_length of 12 months for the data are used to create training samples for the time series prediction. Meaning 1 sliding window contain 12 months for the model to learn and

predict the data for the following month. The LSTM model needs sequences to learn patterns and each sequence will show spending patterns. The X and Y pair are X for input that takes in num_samples, seq_length and num_ features where number of samples is the number of sliding windows, sequence length is 12 sequence per window and number of expense categories is the number of features which is 9 expense category features. Y is the output that produces num_samples and num_features for the results.

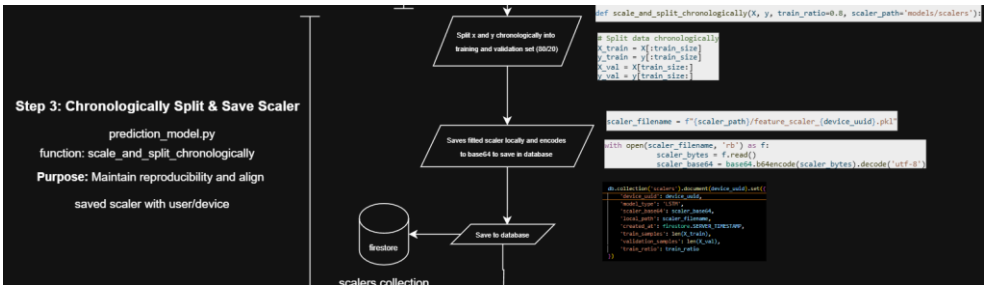


Figure 4.172: Step 3 Chronologically Split and Save Scaler

The third step is to chronologically split and save the scaler as seen in Figure 4.172. The scaler which is the input data that was scaled are chronologically split into training and validation sets which is 80 and 20 respectively. It is chronologically split because LSTM time series prediction considers sequence when performing predictions. Then the scale is saved locally and Firestore under the scalers collection.

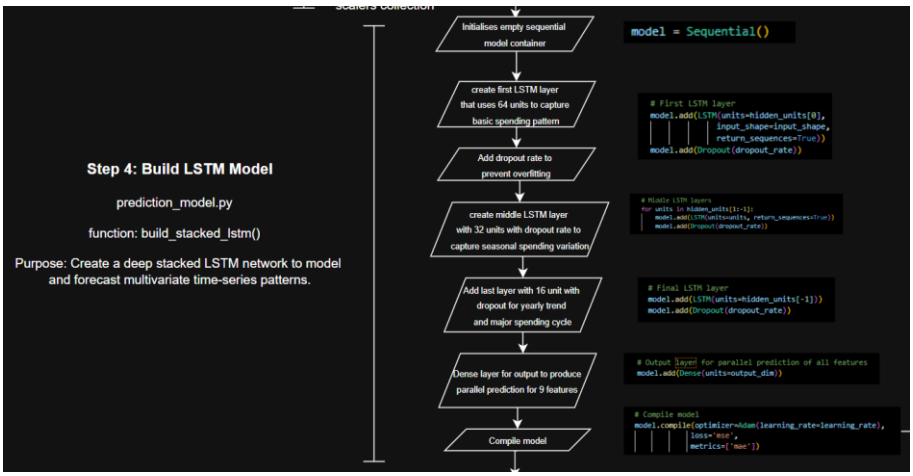


Figure 4.173: Step 4 building LSTM Model

Step 4 as seen in Figure 4.173 is the building of the LSTM model. A sequential model is initialised. Then since the LSTM model being build is a stacked LSTM, there will be 3 layers. The output of one layer will be the input of the next. Layer 1 is the first input layer with 64 hidden units that aims to capture basic patterns. Layer 2 has 32 hidden units and layer 3 has 16 hidden units. Layer 2 aims to capture higher level feature which is more abstract and dynamic and layer 3 aims for longer term dependencies and more complex pattern. All three layers have 0.2 dropout rate which is the neuron are randomly dropped during training to ensure the network doesn't rely on specific neurons, help generalise and prevent overfitting. Lastly is the dense output layer that outputs parallel prediction of 9 features for the expenses category. A dense layer means a neural network where layers a fully connected. Every input neuron is connected to every output neuron. It can learn complex patterns and map them to predict 9 features parallelly. The model is compiled using Adam as optimiser and loss function using Mean Square Error (MSE) as well as using metrics like Mean Absolute Error (MAE). Adam optimiser is used because it has adaptive learning rates and well suited for complex neural network. MSE is used for to measure how wrong the prediction is and MAE is used to evaluate the performance of the model.

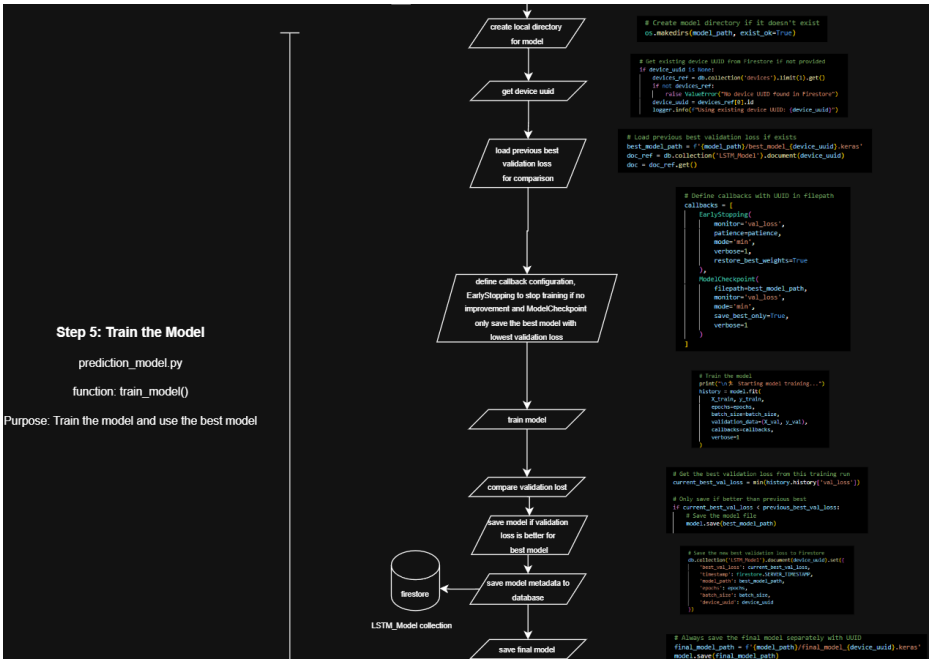


Figure 4.174: Step 5 Train The Model

The last step is to train the model as seen in Figure 4.174. Firstly, a local directory is created for the model. Then, the device uuid is loaded along with the best validation loss for the previous training to do comparison. This step is to ensure if any feature is changed and the model is re-trained, it will take and save the model only if the results are better. Then, callback configuration is defined for EarlyStopping to stop the model training if no improvement detected and ModelCheckpoint to save the best model with lowest validation loss. Then the model is trained using model.fit with the x and y training data, the number of epochs, batch size, validation data, callbacks and verbose. Then the validation loss is compared to the previous model if have. Then the better model and saved locally and in the database under LSTM_Model collection. The final model is saved as well. Once the model is ready to be used, it can be accessed from the frontend.

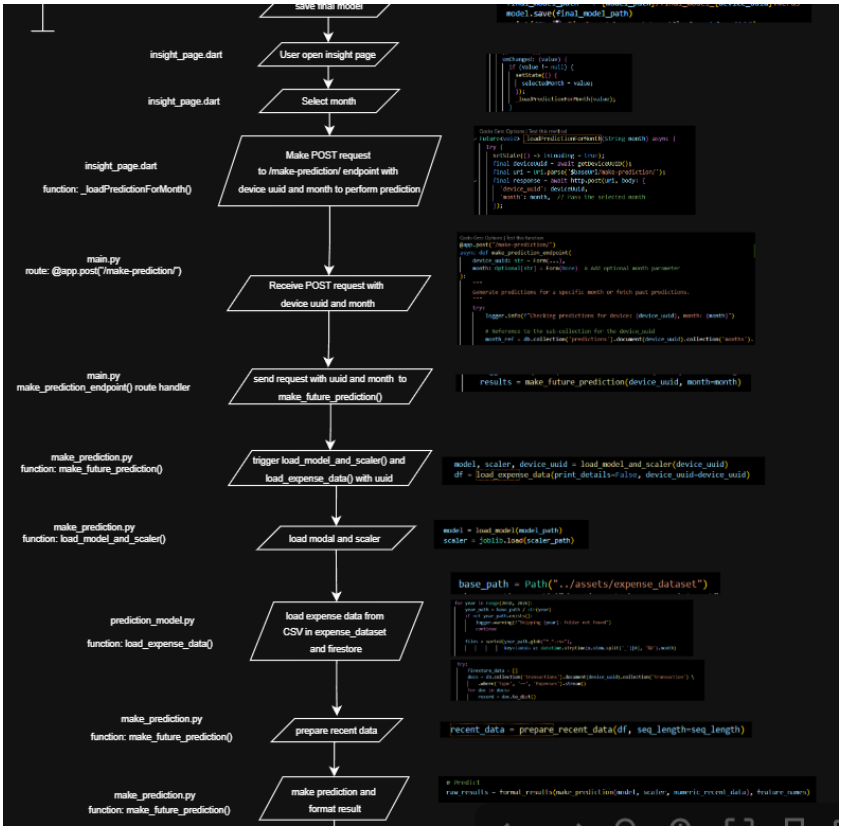


Figure 4.175: Flowchart part 1 insight page

The flow of the insight page is as shown in Figure 4.175. When user opens the insight page in the UI, the function _loadInitialPrediction() is called in the initState() as shown in Figure 4.176.

```
@override
Qodo Gen: Options | Test this method
void initState() {
  super.initState();
  _loadInitialPrediction();
}
```

Figure 4.176: initState()

On load it will load the current month prediction, but user can also load prediction from different month as seen in Figure 4.177. The drop down will include the months of the current year.

Figure 4.177: Month Drop down

The available months are listed by the function availablePredictionMonths and depending on the month chosen, the onchanged the function will set the selectedMonth as the value and pass the value to the function _loadPredictionForMonth(value) as seen in Figure 4.178.

Figure 4.178: selectedMonth

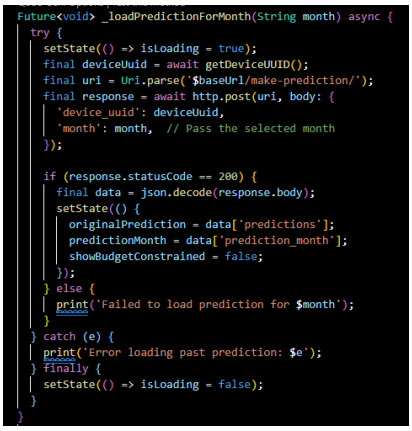


Figure 4.179:_loadPredictionForMonth()

If user changes the month, _loadPredictionForMonth will be called as seen in Figure 4.179 where the device uuid and month is passed as variable to the /make-prediction/ endpoint. If not, on load, the function _loadInitialPrediction() will be called where it makes a POST request to the /make-prediction/ endpoint with the device uuid as parameter for the current month prediction. The backend receives the POST request, and the make_prediction_endpoint() route handler sends request to the make_future_prediction() function with the uuid and month. Then make_future_prediction() loads the modal and scaler using

load_model_and_scaler() function and load the appropriate data using load_expense_data(). The data being loaded is the date, category, grand total, month sin and month cos. Then, make_future_prediction() prepare the most recent data based on the selected month using the function prepare_recent_data(), make prediction using make_prediction() function using the model, scaler and recent data as parameter, and then finally format the results using format_result().

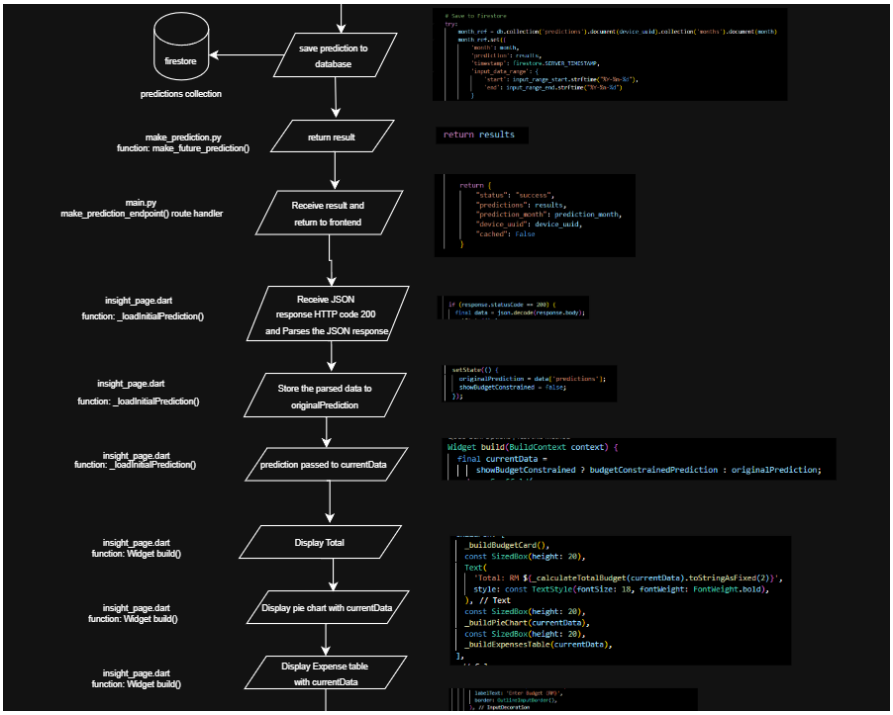


Figure 4.180: Flowchart 2 insight page

Then based on Figure 4.180 the formatted result is saved into the predictions collection in Firestore using the uuid, and the result are returned to the UI. The frontend receives the JSON response HTTP code 200 as seem in Figure 4.181. Currently, the model has 13.29 % of MAE. For example, the prediction of the month June 2025 uses the data from May 2024 to April 2025.

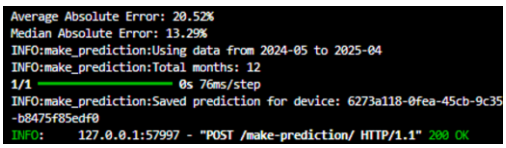


Figure 4.181: HTTP 200 Code

The frontend parses the JSON response containing the prediction result. The parsed data are store in originalPrediction and passed as currentData. Using Widget build(), the total, pie chart and Expense table are displayed with currentData as seen in Figure 4.182.

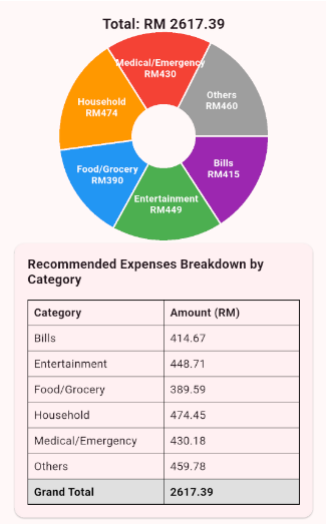


Figure 4.182: Insight page pie chart and table

At the top of the pie chart there is a buildBudgetCard() function is where the budget optimisation feature for the expense prediction comes in. User can enter a budget that they wish to spend on as seen in Figure 4.183.

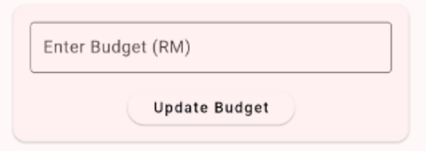


Figure 4.183: Budget

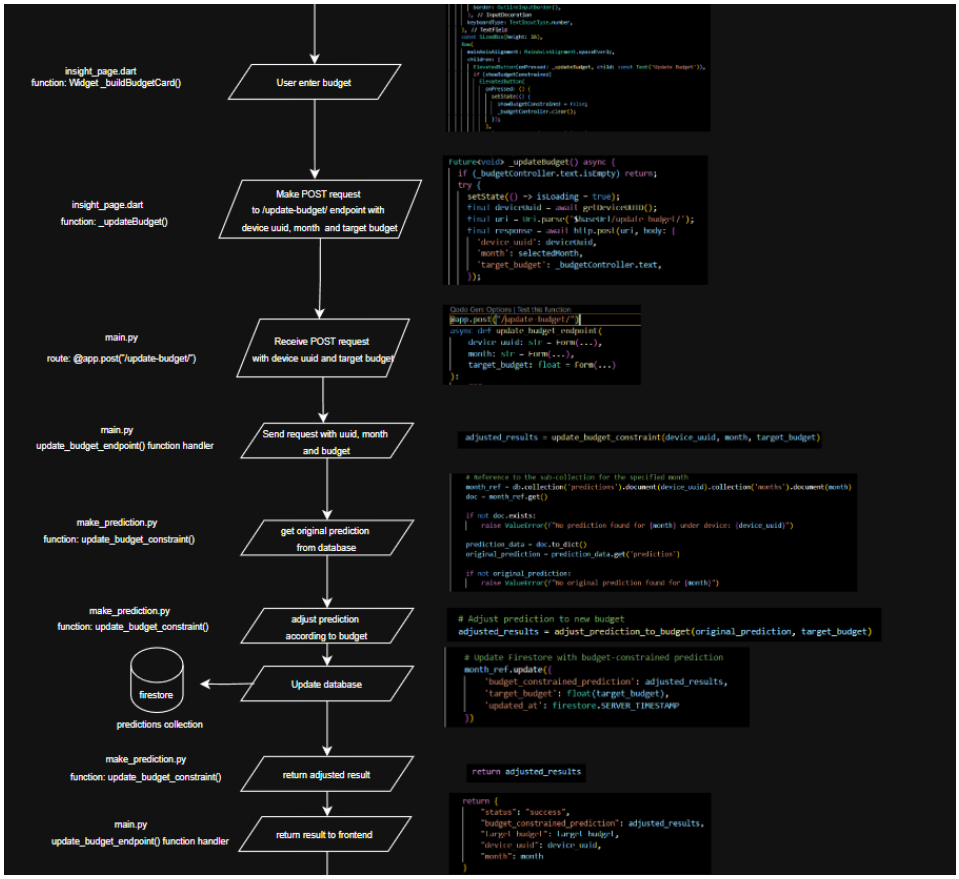


Figure 4.184: Flowchart part 3 insight page

As seen in Figure 4.184 when user adds a budget and click update budget, the function _updateBudget() makes a POST request to /update-budget/ endpoint with the month, device uuid and target budget as parameter. The backend receives the POST request and sends the request to update_budget_constraint() function. The function gets the original prediction from the database using the uuid and month as reference and make adjustment to the prediction according to the budget, so each expense category equally changes the value. Then the budget is updated again to the predictons collection in Firestore. The function update_budget_constraint() returns the adjusted prediction result, and the update_budget_endpoint() function handler returns the result to the frontend.

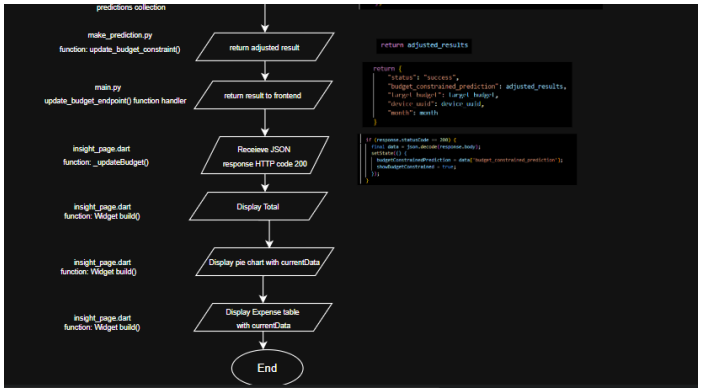


Figure 4.185: Flowchart part 4 insight page

The frontend function _updateBudget() receive the JSON response HTTP code 200 with the results. The Widget build() displays the total, pie chart and the Expense table with the updated value according to the budget. User can continue to change the budget by click on the Update Budget button or show the original prediction using the Show Original button as seen in Figure 4.186.

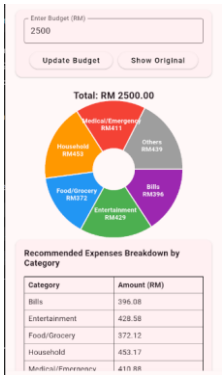


Figure 4.186: Update Budget and show original insight page

4.4.5 Settings Module

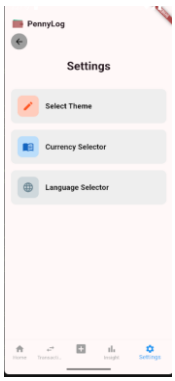


Figure 4.187: Settings

The last module is the settings module. In the settings as seem in Figure 4.187, users have the freedom to select a theme, currency and language which will automatically be applied to the application using the device UUID so that it only has one setting saved.

Figure 4.188 _buildSettingCard

All three settings are displayed as a custom setting card which is created using _buildSettingCard() function as seen in Figure 4.188. In the card, the icon is the left icon shown where each of it has it own icon like edit icon for theme, book icon for currency and language icon for language. bgColor is the background color for the icon container, iconColor is the icons own colour, label is the main label shown on the setting module page, and onTap triggers the Navigator.push to navigate to the appropriate selector page for all three settings.

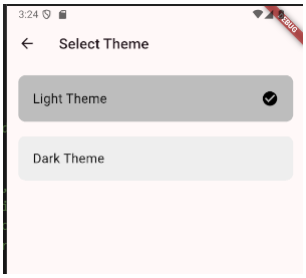


Figure 4.189: Select Theme Page

First is the theme page called theme_selection_page.dart where user can choose between the light or dark theme as seen in Figure 4.189.

```
Qodo Gen: Options | Test this method
Future<void> _saveThemeToFirestore(ThemeEnum theme) async {
  String uuid = await getDeviceUUID();
  await FirebaseFirestore.instance.collection('theme').doc(uuid).set({
    'theme': theme.name,
  }, SetOptions(merge: true));
}
```

Figure 4.190: Save Theme to firestore

Each theme is saved using the device uuid so that the device only saves one theme settings as seen in _saveThemetoFirestore function as seen in Figure 4.190.

```
Qodo Gen: Options | Test this method
Future<void> _loadThemeFromFirestore() async {
  String uuid = await getDeviceUUID();
  final doc = await FirebaseFirestore.instance.collection('theme').doc(uuid).get();
  if (!mounted) return;
  if (doc.exists && doc.data()!.containsKey('theme')) {
    final themeStr = doc['theme'] as String;
    final themeProvider = Provider.of<ThemeProvider>(context, listen: false);
    if (themeStr == ThemeEnum.dark.name) {
      themeProvider.changeTheme(ThemeEnum.dark);
    } else {
      themeProvider.changeTheme(ThemeEnum.light);
    }
  }
  if (mounted) {
    setState(() {
      _isLoading = false;
    });
  }
}
```

Figure 4.191: _loadThemeFromFirestore

When user relaunch the app, the application will trigger the _loadThemeFromFirestore to load last saved theme which is saved in Firestore as seen in Figure 4.191.

```
GestureDetector(
  onTap: () {
    themeProvider.changeTheme(ThemeEnum.light);
    _saveThemeToFirestore(ThemeEnum.light);
  },
  child: Container(
    padding: const EdgeInsets.all(20),
    decoration: BoxDecoration(
      color: currentTheme == ThemeEnum.light ? Colors.grey[400] : Colors.grey[800],
      borderRadius: BorderRadius.circular(10),
    ), // BoxDecoration
    child: Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        Text(
          translate['light']!,
          style: TextStyle(
            fontSize: 18,
            color: currentTheme == ThemeEnum.light ? Colors.black : Colors.white,
          ), // TextStyle
        ), // Text
        if (currentTheme == ThemeEnum.light)
          const Icon(Icons.check_circle, color: Colors.black),
      ],
    ), // Row
  ), // Container
), // GestureDetector
const SizedBox(height: 20),
```

Figure 4.192: Theme GestureDetector

When user choose a theme or change a theme in the settings page for theme selector, for example to light theme, the light theme is wrapped in a GestureDetector

where it detects taps upon changes. Then when tapped, it will call themeProvider.changeTheme(ThemeEnum.light) as seen in Figure 4.192.

```
Qodo Gen: Options | Test this method
Future<void> changeTheme(ThemeEnum theme) async {
  currentTheme = theme;
  await _generateThemeData();
  await _saveThemeToFirestore(); // Save when changed
  notifyListeners();
}
```

Figure 4.193: changeTheme

The function changeTheme in theme_provider.dart is triggered. It updates the currentTheme to light as chosen from user. It then calls _generateThemeData() to load the theme style from JSON files. It then saves the new theme into firestore by triggering _saveThemeToFirestore() and notifies the application to rerender with the new theme selected as seen in Figure 4.193.

```
Qodo Gen: Options | Test this method
Future<void> _generateThemeData() async {
  final themeStr = await rootBundle.loadString(_getThemeJsonPath());
  final Map themeJson = jsonDecode(themeStr);
  currentThemeData = ThemeDecoder.decodeThemeData(themeJson);
}

Qodo Gen: Options | Test this method
String _getThemeJsonPath() {
  switch (currentTheme) {
    case ThemeEnum.light:
      return "assets/themes/light_theme.json";
    case ThemeEnum.dark:
      return "assets/themes/dark_theme.json";
  }
}
```

Figure 4.194: _generateThemeData and _getThemeJsonPath

When _generateThemeData is called, it loads the selected theme file which is being retrieved by _getThemeJsonPath that is pointing to the path assets/themes/light_theme.json. The JSON file will contain all the appropriate settings for the light theme as seen in Figure 4.194.

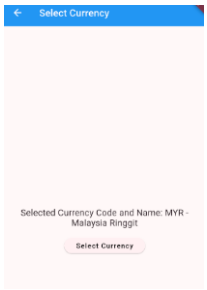


Figure 4.195: Select Currency

The second settings is the currency selector as seen in Figure 4.195 where when user click, they will be navigated to the page that indicates the selector currency code and name.

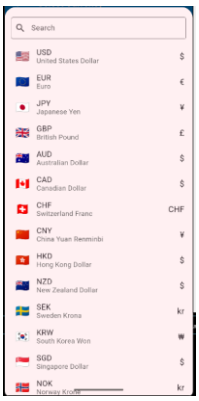


Figure 4.196: Currency List

If user wants to change the currency, they will need to click on the select currency button and a list of currency will appear where user can scroll to view all the currency and filter the currency using keyword as seen in Figure 4.196.

```
ElevatedButton(
  onPressed: () {
    showCurrencyPicker(
      context: context,
      showFlag: true,
      showCurrencyName: true,
      showCurrencyCode: true,
      onSelect: (Currency currency) {
        if (!mounted) return;
        setState(() {
          _selectedCurrency = currency;
        });
        context.read<CurrencyProvider>().setCurrency(currency);
```

Figure 4.197: showCurrencyPicker

When user click on the Select Currency buttons as seen in Figure 4.197, it opens a currency picker model with a searchable list of currencies available which is provided by the import called currency_picker.dart where it is a package that provide build-in list of all world currencies with code,name and flag. The ShowCurrencyPicker() functions will open a searchable list model. The model is a Currency model that gives access to fields like currency.code like MYR, currency.name like Malaysia Ringgit and currency.flag which is the country flag emoji MY. So, when user selects a currency, the setState updates the UI whereby

the _selectedCurrency is the current currency. The CurrencyProvider().setCurrency(currency) is called to save the choice globally.

```
Qodo Gen: Options | Test this method
Future<void> setCurrency(Currency currency) async {
  _currency = currency;
  notifyListeners();

  final uuid = await getDeviceUUID();
  await FirebaseFirestore.instance.collection('currency').doc(uuid).set({
    'currency': currency.code,
  }, SetOptions(merge: true));
  }
}
```

Figure 4.198: setCurrency

The setCurrency in Figure 4.198 will update the _currency and will trigger the UI to refresh using notifyListiners(). It will save the currency code to firestore under the currency collection using the device uuid.

```
Qodo Gen: Options | Test this method
Future<void> loadCurrency() async {
  String uuid = await getDeviceUUID();
  final doc = await FirebaseFirestore.instance.collection('currency').doc(uuid).get();

  if (doc.exists && doc.data()!.containsKey('currency')) {
    String currencyCode = doc['currency'] as String;
    _currency = CurrencyService().findByCode(currencyCode);
  } else {
    _currency = CurrencyService().findByCode('MYR'); // Default to MYR if not set
  }
  notifyListeners();
}
```

Figure 4.199: loadCurrency

Like theme, whenever the application is opened or runs again, it will load the currency from firestore using loadCurrency and display the current selected currency throughout the application as seen in Figure 4.199.

Lastly are the language settings, whereby the UI will first display the selected language and there is a button select language for user to change the current language as shown in Figure 4.200.

Figure 4.200: Select Language

When user clicks on the select language, a list of languages will appear. The select language button is in _showLanguageList function.

```
Qodo Gen: Options | Test this method
void _showLanguageList(BuildContext context, Map<String, String> translate) {
  showModalBottomSheet(
    context: context,
    isScrollControlled: true,
    builder: (context) {
      return StatefulBuilder(
        builder: (BuildContext context, StateSetter setModalState) {
          return SizedBox(
            height: MediaQuery.of(context).size.height * 0.85,
            child: Column(
              children: [
                Padding(
                  padding: const EdgeInsets.all(8.0),
                  child: TextField(
                    controller: _searchController,
                    decoration: InputDecoration(
                      labelText: translate['selectLanguage']!,
                      prefixIcon: const Icon(Icons.search),
                      border: OutlineInputBorder(
                        borderRadius: BorderRadius.circular(8),
                      ), // OutlineInputBorder
                    ), // InputDecoration
                    onChanged: (value) {
                      setModalState(() {}); // Trigger rebuild of the modal
                    },
                  ), // TextField
                ), // Padding
                const SizedBox(height: 8),
                Expanded(
                  child: _buildLanguageList(context),
                ), // Expanded
              ],
            ), // Column
          ); // SizedBox
        },
      ); // StatefulBuilder
```

Figure 4.201: _showLanguageList

When user click on the select language button in _showLanguageList function, it triggers the function _buildLanguageList as seen in Figure 4.201. In the _showLanguageList, there is a showModelBottomSheet. The function showModelBottomSheet is a popup UI which will appear from the bottom of the screen. It overlays the current screen with a list of language in the form of pop up. The list of language comes from _buildLanguageList.

```
import 'package:language_picker/languages.dart'; // Language picker
```

Figure 4.202: language_picker import

```
final List<Language> _languageList = Languages.defaultLanguages;
```

Figure 4.203: _languageList

The _buildLanguageList gets the _languageList that stores the languages from the import package language_picker/languages.dart as shown in Figure 4.202 and Figure 4.203 respectively.

Figure 4.204: Search Language List

The pop up will have a search Input Field included as seen in Figure 4.204 with the list of languages. The controller will track what the user types, the onChanged will trigger the setModelState to to rebuild the model with the filtered list that comes from _buildLanguageList.

```
Widget _buildLanguageList(BuildContext context) {
  final query = _searchController.text.toLowerCase();
  final filteredLanguages = query.isEmpty
      ? _languageList
      : _languageList
          .where((lang) => lang.name.toLowerCase().contains(query))
          .toList();

  return ListView.builder(
    itemCount: filteredLanguages.length,
    itemBuilder: (context, index) {
      final lang = filteredLanguages[index];
      return ListTile(
        title: Text('${lang.name} (${lang.isoCode})'),
        trailing: _selectedLanguageCode == lang.isoCode
            ? const Icon(Icons.check, color: Colors.blue)
            : null,
        onTap: () {
          _saveSelectedLanguage(lang.isoCode);
          Navigator.pop(context);
        },
      ); // ListTile
    },
  ); // ListView.builder
}
```

Figure 4.205: _buildLanguageList

The _buildLanguageList(context) function uses the ListView.builder() to filter the language list, show the results matched, display the lang.name (lang.isoCode) and save the new language lang.isoCode using _saveSelectedLanguage when onTap is triggered as seen in Figure 4.205. When user type “chi” in the beginning, the _searchController.text becomes “chi” then the _buildLanguageList() will do the filtering. The final query = _searchController.text.toLowerCase(); grab whats the user typed in the search box, convert to lowercase, then _languageList.where((lang) =>

lang.name.toLowerCase().contains(query)).toList(); will filter main list _languageList and only name contain the query will be included in the search. The return ListView.builder(is when flutter builds a scrollable list of the filtered result. For each filtered item, it will create a return ListTile(, where each tile shows the language name and ISO code. onTap it will trigger _saveSelectedLanguage and closes the model using Navigator.pop(context).

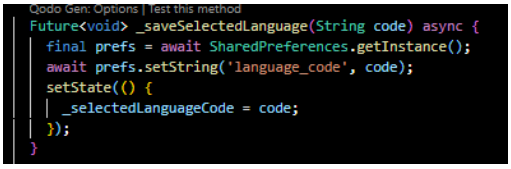


Figure 4.206: _saveSelectedLanguage

In the _saveSelectedLanguage function in Figure 4.206, the final prefs = await SharedPreferences.getInstance(); will initialises access to local device storage by using the shared_preference plugin. It will allow the application to save key_value pairs in this case is the language_code key and value. The await prefs.setString('language_code', code); will store the selected ISO code. Then setState(() { _selectedLanguageCode = code; }); will update the application so that the screen reflects the newly selected language.

When the value is saved in shared_preference, the device will trigger loadLanguage in language_provider.dart before it updates the application.

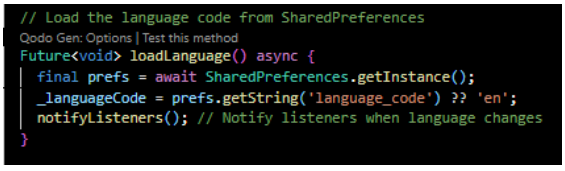


Figure 4.207: loadLanguage

In the loadLanguage as seen in Figure 4.207, it reads the language code from the device storage SharedPreference. If nothing is saved, it falls back to the original language which doesn't need translation which is 'en' for English. It will then use

notifyListeners() so all widgets using LanguageProvider will be rebuilt in the new language. The function is usually called in initiState() for every page.

```dart
Future<void> setLanguage(String code) async {
  final prefs = await SharedPreferences.getInstance();
  await prefs.setString('language_code', code);
  _languageCode = code;
  _cachedTranslations.clear(); // Clear cache when language changes
  notifyListeners(); // Notify listeners when language changes
}
```

Figure 4.208: setLanguage

The setLanguage function in Figure 4.208 is triggered when usually user chooses a new language in the language list. It will update the stored language in SharedPreferences, updates the _languageCode, clear cache translation so that new translations are generated, and notifies all listening widget for the updates.

Whenever in the source code there is a line used for translation like await langProvider.translate("Text"), the translator.dart is called where the actual translation happens.

```dart
final prefs = await SharedPreferences.getInstance();
final toLang = prefs.getString('language_code') ?? 'en';
```

Figure 4.209: load language from SharedPreferences

Firstly, it loads the target language that the user selected as seen in Figure 4.209.

```dart
if (toLang == 'en') {
  _logger.i("Translation skipped: Target language is English.");
  return originalText;
}
```

Figure 4.210: Skip English Translation

If the selected language is English, it will skip translation and use the normal label as seen in Figure 4.210.

```dart
final deviceUUID = await getDeviceUUID();
final translationKey = '${fromLang}_$toLang${originalText.hashCode}';
```

Figure 4.211: Get Device UUID and hash key

If not, it will get the device UUID along with a unique hash key to represent the specific translation as seen in Figure 4.211.

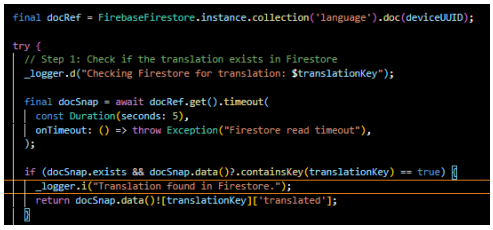


Figure 4.212: Check Cache

It will first look for cached translation in firestore, if there is an existing translation, it will return the translation as seen in Figure 4.212.

Figure 4.213: Send Translation to MyMemoryAPI

If no translation found, it will make a new request to MyMemory API for the translation as seen in Figure 4.213.

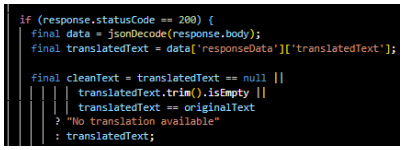


Figure 4.214: Successful Response Code

Once the translation is returned with the status code 200 which means successful, the response is extracted, and the results are clean. It is to handle any empty or invalid responses. It will also provide a proper fallback message if the result is not suitable as seen in Figure 4.214.

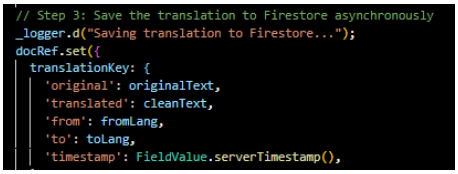


Figure 4.215: Save To Firebase

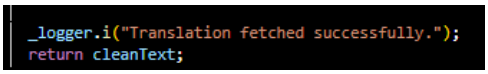


Figure 4.216: Return Translation

Lastly, the translation will be saved to firestore and the translated text are returned to the UI as seen in Figure 4.215 and Figure 4.216 respectively.

4.5 Summary

In summary, chapter 4 presented the installation of the configurations, library as well as how each module of PennyLog was developed. The purpose of this chapter is to fully understand how each functions work in all the module as well how the frontend and backend communicates. All the functionality works well with only limitation to the saiz of the image since the image is converted to base64 string to be saved in Firestore, the saiz of the file has to remain within 1MB.

CHAPTER 5: TESTING

5.1 Introduction

The XP methodology testing is involved in the Productionising and Maintenance phase. The testing performed for PennyLog: Expense Tracker Predictive Budget Optimisation will verify that the application has no bugs and error free, before and after deployment (Niranjanamurthy et al., 2018). The testing is done to ensure that the application meets the target user's requirement which was collected during the Exploration phase. Both functional and User Acceptance Test (UAT) will be performed on all modules.

5.2 Functional Testing

Functionality testing is conducted to ensure that all the functionality in PennyLog operates as specified by the requirements. Niranjanamurthy et al. (2018) specifies that the testing concentrates mostly on the main functions in the module, its basic usability, the accessibility of the system as well as testing the error conditions. The test cases are tabulated in table form below.

Table 5.1: Test Case for Home Module Spending Overview

Feature Name: **Test Case Description:** **Testing Objective:**		**Home Module – Spending Overview** To verify that the home module for spending overview section displays correctly the values for Total Balance, Income and Expenses based on the selected month and year. To ensure that the query made to retrieve information from Firestore are calculated correctly for total balance, total expense and total income. The amount is displayed correct and are dynamically updated when user changes the month or year as well as when new transactions are added.					
Test Case ID	**Test Scenario**	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status**	**Severity Failure**
HM_SO_01	New Users launch PennyLog to see home page	1. Launch PennyLog 2. Home page display by default as landing page 3. Displays current month and year spending overview	1. Device UUID 2. Default current month and year	No data for the current month spending overview.	The display matches the totals accurately.	Pass	High
HM_SO_02	Users change month drop down	1. Tap on month dropdown 2. Select May 3. Home Page Reloads	1. Device UUID 2. May 2025	Displays May 2025 spending overview for total balance, income and expenses.	The display matches the total accurately for the May 2025.	Pass	High
HM_SO_03	Users change year drop down	1. Tap on year dropdown 2. Select 2024 3. Home Page Reloads	1. Device UUID 2. June 2024	Displays June 2024 spending overview for total balance, income and expenses.	The display matches the total accurately for the June 2024.	Pass	High
HM_SO_04	Users select month and year with no data	1. Tap on year dropdown 2. Select 2016 3. Home Page Reloads 4. Tap on month dropdown 5. Select January	1. Device UUID 2. January 2016	Displays No data in January 2016 for spending overview.	Accurately display no data on January 2016 spending overview.	Pass	Medium

		6. Home Page Reloads					
HM_SO_05	Users select month and year with negative total balance	1. Select month and year with Expense total more than income total 2. Home Page Reloads	1. Device UUID 2. May 2025	Display negative total balance with expenses total more than income total.	Accurately display negative total balance data for May 2025 spending overview.	Pass	High

Table 5.2: Test Case for Home Module Spending Trend Graph

Feature Name: **Test Case Description:** **Testing Objective:**		**Home Module – Spending Trend Graph** To verify that the 12-month spending trend bar graph able to accurately display data. The graph will also help with visualisation on monthly expenses total for the selected year. To ensure that the bar graph displays data for all 12 months accurately by querying from Firestore and updates the graph to reflect any changes to the monthly expenses for the selected year.					
Test Case ID	**Test Scenario**	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status**	**Severity Failure**
HM_STG_01	New Users launch PennyLog to see home page	1. Launch PennyLog. 2. Home page display by default as landing page. 3. Displays current month and year spending trend graph	1. Device UUID 2. Default current month and year	No data shown for the current year graph.	Accurately display the spending trend graph.	Pass	High
HM_STG_02	Users change year drop down	1. Tap on year dropdown 2. Select 2024 3. App reloads	1. Device UUID 2. 2024	Displays 2024 spending trend graph.	Accurately display the spending trend graph.	Pass	High
HM_STG_03	Users select year with no data	1. Tap on year dropdown 2. Select 2016 3. App reloads	1. Device UUID 2. 2016	Displays 2016 spending trend graph that has no bar for selected month.	Accurately display the spending trend graph.	Pass	High

HM_STG_04	User attempts to press on a bar in the bar graph.	1. Tap on May 2025 bar 2. Displays total expenses as a pop up on press.	1. Device UUID 2. May 2025	Displays May 2025 expenses total as a pop up when user press or tap on the bar.	Accurately display the spending trend graph.	Pass	High

Table 5.3: Test Case for Home Module Recent Transaction

Feature Name: **Test Case Description:** **Testing Objective:**		**Home Module – Recent Transaction** To verify that recent transaction section always displays the latest 3 transaction. To ensure the recent transaction list consistently show the latest 3 transaction from all records saved in Firestore sorted by datetime in descending order.					
Test Case ID	**Test Scenario**	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status**	**Severity Failure**
HM_RT_01	New Users launch PennyLog to see home page	1. Launch PennyLog. 2. Home page display by default as landing page. 3. Displays recent 3 transaction list.	Device UUID	No data shown for recent category.	Accurately display the recent transaction.	Pass	High
HM_RT_02	Users add 2 new transactions	1. Displays 2 transactions only in recent transactions.	Transaction 1: Device UUID Category: Food/Grocery Remark: KFC Amount: 20 Date: 1 June 2025 Time: 1.30 PM Transaction 2: Device UUID	Display the 2 transactions in the transaction list sorted by datetime in descending order.	The 2 transactions are accurately listed in the recent transaction list in the correct order.	Pass	High

			Category: Bills Remark: Water Bill Amount: 60 Date: 5 June 2025 Time: 5.30 PM				
HM_RT_03	User attempt to launch the app for the first time.	1. Empty recent transaction list shown.	-	No transaction list shown in the recent transaction section.	Accurately display no data.	Pass	High
HM_RT_04	User adds 3 new transactions with different transaction category	1. Add 3 new transactions with different transaction category using the new transaction module 2. Recent transaction section displays the new transactions.	Transaction 1: Category: Food/Grocery Remark: Sambal Sambal Amount: 20 Date: 2 June 2025 Time: 8.00 PM Transaction 2: Category: Bills Remark: Electric Bill Amount: 100 Date: 3 June 2025 Time: 12.00 PM Transaction 3: Category: Entertainment Remark: Movie Amount: 40	Display transactions in recent transactions with the different categories' icons accurately.	Transaction lists are shown accurately with different categories icons and information's.	Pass	High

			Date: 4 June 2025 Time: 1.00 PM				
HM_RT_05	User adds 3 new transactions in the same date with different time	1. Add 3 new transactions with same date but different timing. 2. Recent transaction section displays the new transactions.	Transaction 1: Category: Food/Grocery Remark: Steamboat Amount: 40 Date: 2 June 2025 Time: 1.00 PM Transaction 2: Category: Bills Remark: Netflix Amount: 100 Date: 2 June 2025 Time: 3.00 PM Transaction 3: Category: Emergency/Medical Remark: Clinic Amount: 50 Date: 2 June 2025 Time: 8.00 PM	Display transactions in recent transactions sorted by datetime in descending order.	The 3 transactions are accurately sorted by DateTime in the recent transaction section.	Pass	High
HM_RT_06	User attempts to change month and year	1. Tap on year dropdown 2. Select 2024 3. App reloads 4. Tap on month dropdown 5. Select May 6. App reloads.	May 2024	The app displays the same recent transaction lists.	The app displays the same recent transaction list on every month and year.	Pass	High

Table 5.4: Test Case for Transaction Page Module Transaction List

Feature Name:		**Transaction Page Module – Transaction List**					
Test Case Description:		To verify that users can view all transaction list filtered based on selected month and year, as well as toggle between transaction type.					
Testing Objective:		To ensure that the drop-down filters and toggle button for transaction type correctly loads and display the transactions based on selection sorted by datetime in descending order.					
Test Case ID	**Test Scenario**	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status**	**Severity Failure**
TPM_TL_01	Users launch PennyLog to see Transaction page	1. Launch PennyLog. 2. Home page display by default as landing page 3. Select transaction page at bottom navigation bar 4. Page loads	Device UUID	Displays the default month and year transaction list for expenses on load.	Accurately display the current transaction list.	Pass	High
TPM_TL_02	Users change month drop down	1. Tap on month dropdown 2. Select May 3. Page Reloads	1. Device UUID 2. Month: May 3. Year: 2025	Display expense transaction list for May 2025 on load.	Accurately display the current transaction list.	Pass	High
TPM_TL_03	Users change year drop down	1. Tap on year dropdown 2. Select 2024 3. Page Reloads	1. Device UUID 2. Month: May 3. Year: 2024	Display expense transaction list for May 2024 on load.	Accurately display the current transaction list.	Pass	High
TPM_TL_04	Users choose month and year with no list	1. Tap on year dropdown 2. Selects 2016 3. Page reloads 4. Tap on month dropdown 5. Selects January 6. Page Reloads	1. Device UUID 2. Month: January 3. Year: 2016	No data displayed on load.	Accurately display the empty list.	Pass	High

TPM_TL_05	Users select income toggle	1. Tap on income toggle. 2. Loads and display income transaction list.	Device UUID	Displays the default month and year transaction list for income on load.	Accurately display the current transaction list.	Pass	High
TPM_TL_06	Users select income toggle then expense toggle	1. Taps on income toggle. 2. Page loads and display income transaction list. 3. Taps on expense toggle 4. Page loads and display expense transaction list.	Device UUID	Displays the default month and year transaction list for income and then for expense on load.	Accurately display the current transaction list.	Pass	High

Table 5.5: Test Case for Transaction Page Module Manage Transaction

Feature Name: **Test Case Description:** **Testing Objective:**		**Transaction Page Module – Manage Transaction (View, Edit, Delete, Save)** To verify that users can view transaction details, edit values, save changes made or delete transactions. To ensure that the application is reliable and able to handle users request on the transaction management and the UI updates the reflected CRUD operations accordingly. Firestore will dynamically store the changes as well.					
Test Case ID	**Test Scenario**	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status**	**Severity Failure**
TPM_MT_01	Users View Expense Transaction detail	1. Launch PennyLog. 2. Home page display by default as landing page 3. Select transaction page at bottom navigation bar 4. Page loads and display expense transaction list for default month and year 5. User selects and taps on one of the transactions in the available list.	Device UUID	Application displays the selected expense details such as amount, transaction type, payment method, category, remark, amount, time, date, status and image if available.	Accurately display the selected transaction details.	Pass	High

		6. Transaction detail page loads and displays.					
TPM_MT_02	Users View Income Transaction detail	1. User taps on income toggle 2. Application load and display income transaction list 3. User selects and taps on one of the transactions in the available list. 4. Transaction detail page loads and displays.	Device UUID	Application displays the selected income details such as amount, transaction type, payment method, category, remark, amount, time, date, status and image if available.	Accurately display the selected transaction details.	Pass	High
TPM_MT_03	Users View image on income transaction	1. User taps on income toggle 2. Application load and display income transaction list 3. User selects and taps on one of the transactions in the available list. 4. Transaction detail page loads and displays. 5. Taps on available image in the detail page.	Device UUID	The transaction detail page displays a pop up with the image. Users can tap on "+" to zoom in, "-" to zoom out, "Reset Zoom" button to show original image saiz. Users can also move to view image by tapping and dragging the photo around.	The image is shown in the pop up and the functionality works as intended.	Pass	High
TPM_MT_04	Users edit expense transaction amount with valid value and save	1. User selects and taps on one of the transactions in the available list. 2. Transaction detail page loads and displays. 3. Tap on the edit icon	1. Device UUID 2. Amount: 50	Transaction detail page becomes editable. User able to change the amount and a save transaction prompt pop up appears asking user	The application successfully edits the changes and accurately	Pass	High

		4. Change the transaction amount with a valid value 5. Tap on the save button icon 6. Save transaction prompt pops up 7. Tap on “Yes” button 8. Application reloads and display expense transaction list for default month and year		for confirmation. Once user click yes, a validation message appears showing “Transaction Updated Successfully”. The information on the transaction list changes and firebase value are also updated.	shows the displays.		
TPM_MT_05	Users edit, enter invalid amount value for expense transaction and save	1. User selects and taps on one of the transactions in the available list. 2. Transaction detail page loads and displays. 3. Tap on the edit icon 4. Change the transaction amount with an invalid value 5. Tap on the save button icon 6. Validation error appears	1. Device UUID 2. Amount: aa	Transaction detail page becomes editable. Validation error showing the message “Enter a valid amount > 0” appears when user try to save the transaction.	The application successfully shows the validation error message.	Pass	High
TPM_MT_06	Users edit, leaves blank amount value for expense transaction and save	1. User selects and taps on one of the transactions in the available list. 2. Transaction detail page loads and displays. 3. Tap on the edit icon 4. Leave the transaction amount empty	1. Device UUID 2. Amount: 0	Transaction detail page becomes editable. Validation error showing the message “Please enter an amount” appears when user try to save the transaction.	The application successfully shows the validation error message.	Pass	High

		5. Tap on the save button icon 6. Validation error appears					
TPM_MT_07	Users edit, change transaction type and save	1. User selects and taps on one of the transactions in the available list. 2. Transaction detail page loads and displays. 3. Tap on the edit icon 4. Change the transaction type from expenses to income 5. Tap on the save button icon 6. Save transaction prompt pops up 7. Tap on "Yes" button 8. Application reloads and displays the changes on the income transaction list.	1. Device UUID 2. Transaction Type: Income	Transaction detail page becomes editable. User able to change the transaction type and a save transaction prompt pop up appears asking user for confirmation. Once user click yes, a validation message appears showing "Transaction Updated Successfully". The information on the transaction list changes and firebase value are also updated.	The application successfully edits the changes and accurately shows the displays.	Pass	High
TPM_MT_08	Users edit, change payment type and save	1. User selects and taps on one of the transactions in the available list. 2. Transaction detail page loads and displays. 3. Tap on the edit icon 4. Change the transaction type to "Bank Transfer" 5. Tap on the save button icon	1. Device UUID 2. Payment Type: Bank Transfer	Transaction detail page becomes editable. User able to change the payment type and a save transaction prompt pop up appears asking user for confirmation. Once user click yes, a	The application successfully edits the changes and accurately shows the displays.	Pass	High

		6. Save transaction prompt pops up 7. Tap on “Yes” button 8. Application reloads and display expense transaction list for default month and year		validation message appears showing “Transaction Updated Successfully”. The information on the transaction list changes and firebase value are also updated.			
TPM_MT_09	Users edit, change date and save	1. User selects and taps on one of the transactions in the available list. 2. Transaction detail page loads and displays. 3. Tap on the edit icon 4. Change the date to “9/6/2025” 5. Tap on the save button icon 6. Save transaction prompt pops up 7. Tap on “Yes” button 8. Application reloads and display expense transaction list for default month and year	1. Device UUID 2. Date = 9/6/2025	Transaction detail page becomes editable. Date picker appears and user able to change the date. A save transaction prompt pop up appears asking user for confirmation. Once user click yes, a validation message appears showing “Transaction Updated Successfully”. The information on the transaction list changes and firebase value are also updated. The transaction time will also change according to the time the transaction is saved.	The application successfully edits the changes and accurately shows the displays.	Pass	High

TPM_MT_10	Users edit, upload a photo and save	1. User selects and taps on one of the transactions in the available list. 2. Transaction detail page loads and displays. 3. Tap on the edit icon 4. Tap on “Change Image” button 5. A gallery appears 6. Selects the image 7. Tap on the save button icon 8. Save transaction prompt pops up 9. Tap on “Yes” button 10. Application reloads and display expense transaction list for default month and year	1. Device UUID 2. Receipt1.png	Transaction detail page becomes editable. User able add an image and a save transaction prompt pop up appears asking user for confirmation. Once user click yes, a validation message appears showing “Transaction Updated Successfully”. The information on the transaction list changes and firebase saves the image as base65 with the remaining transaction information.	The application successfully edits the changes and accurately shows the displays.	Pass	High
TPM_MT_11	Users edit, change a photo and save	1. User selects and taps on one of the transactions in the available list. 2. Transaction detail page loads and displays. 3. Tap on the edit icon 4. Tap on “Change Image” button 5. A gallery appears 6. Selects the image 7. Tap on the save button icon	1. Device UUID 2. Receipt2.png	Transaction detail page becomes editable. User able change the image and a save transaction prompt pop up appears asking user for confirmation. Once user click yes, a validation message appears showing	The application successfully edits the changes and accurately shows the displays	Pass	High

		8. Save transaction prompt pops up 9. Tap on “Yes” button 10. Application reloads and display expense transaction list for default month and year		“Transaction Updated Successfully”. The information on the transaction list changes and firebase saves the new image base64 with the remaining transaction information.			
TPM_MT_12	Users edit, change amount and cancel editing	1. User selects and taps on one of the transactions in the available list. 2. Transaction detail page loads and displays. 3. Tap on the edit icon 4. Change the transaction amount with a valid value 5. Press back button or X button 6. The page loads and shows the transaction list page with no changes.	Device UUID Original Amount: 50 New Amount: 70	Transaction detail page becomes editable. User able change the amount. The transaction page appears again with no changes.	No changes are made, and all transaction details displays accurately.	Pass	High
TPM_MT_13	Users delete transaction	1. User selects and taps on one of the transactions in the available list. 2. Transaction detail page loads and displays. 3. Tap on the delete icon 4. Delete transaction prompt pops up 5. Tap on “Yes” button	Device UUID	Transaction detail page becomes editable. A delete transaction prompt pop up appears asking user for confirmation. Once user click yes, a validation message appears showing	The application successfully deletes the transaction, and the UI accurately displays the changes.	Pass	High

		6. Application reloads and display expense transaction list for default month and year		"Transaction Deleted Successfully". The transaction is deleted from the list and firebase removes the transaction details.			

Table 5.6: Test Case for New Transaction Module Add Manual Transaction

Feature Name: **Test Case Description:** **Testing Objective:**		**New Transaction Module – Add Manual Transaction** To verify that user can add new transaction manually for both income and expenses. The system supports image cropping. To ensure that manual process functions accurately with proper validation and image cropping. The process functions with backend support and database to save transaction.					
Test Case ID	**Test Scenario**	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status**	**Severity Failure**
NTM_AT_01	Users Click on Add New Transaction	1. Launch PennyLog. 2. Home page display by default as landing page 3. Tap on "+" on the bottom navigation bar to add new transaction	Device UUID	A pop up to prompt user for transaction type appears showing Expense and income selection.	The pop up accurately displays.	Pass	Medium
NTM_AT_02	Users choose income type	1. Users select income transaction type 2. App loads and display income transaction form	Device UUID	The application navigates to income form.	The accurate form is displayed.	Pass	High
NTM_AT_03	Users choose expense type	1. Users select expense transaction type 2. App loads and display expense transaction form	Device UUID	The application navigates to	The accurate form is displayed.	Pass	High

				expense form.			
NTM_AT_04	Users fill in all valid transaction details and save for expenses	1. Users select expense transaction type 2. App loads and display expense transaction form 3. Fills in all field with valid values 4. Click on save icon 5. Validation appears and page redirect to previous page	Device UUID Transaction type: Expenses Payment Method: Cash Category: Bills Remark: Digi Amount: 20 Date: 9/6/2025 Receipt: No Receipt	Validation showing "Transaction saved successfully" is displayed. Firebase saves new transaction.	The application successfully saved new transaction and accurately reflects on UI.	Pass	High
NTM_AT_05	Users fill in all valid transaction details and save for income	1. Users select income transaction type 2. App loads and display income transaction form 3. Fills in all field with valid values 4. Click on save icon 5. Validation appears and page redirect to previous page	Device UUID Transaction type: Income Payment Method: Bank Transfer Category: Salary Remark: June Salary Amount: 4000 Date: 26/6/2025 Receipt: No Receipt	Validation showing "Transaction saved successfully" is displayed. Firebase saves new transaction	The application successfully saved new transaction and accurately reflects on UI.	Pass	High
NTM_AT_06	Users fill in with invalid transaction details for expenses	1. Users select expense transaction type 2. App loads and display expense transaction form 3. Fills in all field with invalid values 4. Click on save icon 5. Error validation appears	Device UUID Transaction type: Expenses Payment Method: Cash Category: Bills Remark: Digi Amount: aa Date: - Receipt: No Receipt	Validation error shows "Please select a date" and "Enter a valid amount greater than 0".	The application successfully shows the validation error on the invalid field that was filled in.	Pass	High

NTM_AT_07	Users fill in with invalid transaction details for income	1. Users select income transaction type 2. App loads and display income transaction form 3. Fills in all field with invalid values 4. Click on save icon 5. Error validation appears	Device UUID Transaction type: Income Payment Method: Cash Category: Investment Remark: Crypto Amount: - Date: - Receipt: No Receipt	Validation error shows "Please select a date" and "Enter a valid amount greater than 0".	The application successfully shows the validation error on the invalid field that was filled in.	Pass	High
NTM_AT_08	User upload image and fill in transaction details manually for expense	1. Users select expense transaction type 2. App loads and display expense transaction form 3. Tap on Select Image 4. Permission to use gallery appears 5. Tap Yes 6. Select Image 7. Resize and Crop Image 8. Select "✓" to choose the cropped image 9. Preview Image appears with cropped photo alongside prompt for enhancing image 10. Tap No 11. Fill in transaction details manually. 12. Click on save icon 13. Validation appears and page redirect to previous page	Device UUID Transaction type: Expense Payment Method: Debit Card Category: Medical/Emergency Remark: Crypto Amount: 4000 Date: 26/6/2025 Receipt: Receipt1.png	Validation showing "Transaction saved successfully" is displayed. Firebase saves new transaction.	The application successfully saved new transaction and accurately reflects on UI.	Pass	Medium

NTM_AT_09	User upload image and fill in transaction details manually for income	1. Users select income transaction type 2. App loads and display income transaction form 3. Tap on Select Image 4. Permission to use gallery appears 5. Tap Yes 6. Select Image 7. Resize and Crop Image 8. Select "✓" to choose the cropped image 9. Preview Image appears with cropped photo alongside prompt for enhancing image 10. Tap No 11. Transaction detail form appears with the cropped image. 12. Fill in transaction details manually. 13. Click on save icon 14. Validation appears and page redirect to previous page	Device UUID Transaction type: Income Payment Method: Bank Transfer Category: Salary Remark: Part Time Amount: 1500 Date: 16/6/2025 Receipt: Receipt3.png	Validation showing "Transaction saved successfully" is displayed. Firebase saves new transaction.	The application successfully saved new transaction and accurately reflects on UI.	Pass	Medium

Table 5.7: Test Case for New Transaction Module Add Transaction Using Receipt

Feature Name:	**New Transaction Module – Add Transaction Using Receipt**
Test Case Description:	To verify that user can add new transaction through receipt image for both income and expenses. The system supports image cropping, enhancement, OCR and applying extracted data.
Testing Objective:	To ensure that the automated process functions accurately with proper validation, image processing and extraction. The process is automated with backend support and database to save transaction.

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Result	Actual Result	Status	Severity Failure
NTM_ATUR_01	Users add new transaction using select image for expenses or income	1. Users select expense or income transaction type 2. App loads and display expense or income transaction form 3. Tap on Select Image 4. Permission to use gallery appears 5. Tap Yes 6. Select Image 7. Resize and Crop Image 8. Select "✓" to choose the cropped image 9. Preview Image appears with cropped photo alongside prompt for enhancing image 10. Tap Yes 11. Enhancing loading indicator appears 12. Enhanced image preview appears alongside prompt to perform OCR on the image 13. Tap Yes 14. Performing OCR loading indicator appears 15. OCR completion pops up appear alongside prompt to see the result 16. Extracted information shown alongside prompt to use automate the filling in process 17. Tap Use This 18. Information is automatically filled in with the extracted information	Device UUID Image: Receipt1.png	Permission for gallery appears. Pop up for selection appears. Validation showing "Transaction saved successfully" is displayed. Firebase saves new transaction.	Permission appears for new user. The application successfully saved new transaction and accurately reflects on UI. All pop ups accurately displayed.	Pass	High

		19. Fill in missing fields 20. Tap on save icon 21. Validation appears and page redirect to previous page					
NTM_ATUR_02	Users add new transaction using Open Camera for expenses or income	1. Tap on Open Camera 2. Permission to use camera appears 3. Tap Yes 4. Camera opens 5. Take picture 6. Select "✓" to choose image captured 7. Resize and Crop Image 8. Select "✓" to choose the cropped image 9. Preview Image appears with cropped photo alongside prompt for enhancing image 10. Tap Yes 11. Enhancing loading indicator appears 12. Enhanced image preview appears alongside prompt to perform OCR on the image 13. Tap Yes 14. Performing OCR loading indicator appears 15. OCR completion pops up appear alongside prompt to see the result 16. Extracted information shown alongside prompt to use automate the filling in process 17. Tap Use This 18. Information is automatically filled in with the extracted information	Device UUID Image: Receipt2.png	Permission for camera appears. Pop up for selection appears. Validation showing "Transaction saved successfully" is displayed. Firebase saves new transaction.	Permission appears for new user. The application successfully saved new transaction and accurately reflects on UI. All pop ups accurately displayed.	Pass	High

		19. Fill in missing fields 20. Tap on save icon 21. Validation appears and page redirect to previous page					
NTM_ATUR_03	Users' open camera to capture image, crop and enhance image for expenses or income	1. Take picture 2. Select "✓" to choose image captured 3. Resize and Crop Image 4. Select "✓" to choose the cropped image 5. Preview Image appears with cropped photo alongside prompt for enhancing image 6. Tap Yes 7. Enhancing loading indicator appears 8. Enhanced image preview appears alongside prompt to perform OCR on the image 9. Tap No 10. Transaction detail form appears with the enhanced image 11. Fill in transaction details manually. 12. Click on save icon 13. Validation appears and page redirect to previous page	Device UUID Image: Receipt2.png	Permission to use camera appears. Pop up for selection appears. Transaction detail form appears. Validation showing "Transaction saved successfully" is displayed. Firebase saves new transaction.	Accurately display all permission and pop up. The form appears with the accurate enhanced image. The application successfully saved new transaction and accurately reflects on UI.	Pass	High
NTM_ATUR_04	Users upload, crop and enhance image for expenses or income	1. Select Image 2. Resize and Crop Image 3. Select "✓" to choose the cropped image 4. Preview Image appears with cropped photo alongside prompt for enhancing image 5. Tap Yes	Device UUID Image: Receipt2.png	Permission for gallery appears. Pop up for selection appears. Transaction detail form	Accurately display all permission and pop up. The form appears with the accurate enhanced	Pass	High

		6. Enhancing loading indicator appears 7. Enhanced image preview appears alongside prompt to perform OCR on the image 8. Tap No 9. Transaction detail form appears with the enhanced image 10. Fill in transaction details manually. 11. Click on save icon 12. Validation appears and page redirect to previous page		appears. Validation showing "Transaction saved successfully" is displayed. Firebase saves new transaction.	image. The application successfully saved new transaction and accurately reflects on UI.		
NTM_ATUR_05	Users upload, crop, enhance and perform OCR for expenses or income	1. Select Image 2. Resize and Crop Image 3. Select "✓" to choose the cropped image 4. Preview Image appears with cropped photo alongside prompt for enhancing image 5. Tap Yes 6. Enhancing loading indicator appears 7. Enhanced image preview appears alongside prompt to perform OCR on the image 8. Tap Yes 9. Performing OCR loading indicator appears 10. OCR completion pops up appear alongside prompt to see the result 11. Extracted information shown alongside prompt to use automate the filling in process	Device UUID Image: Receipt2.png	Permission for gallery appears. Pop up for selection appears. Transaction detail form appears. Validation showing "Transaction saved successfully" is displayed. Firebase saves new transaction.	Accurately display all permission and pop up. The form appears with the accurate enhanced image. The application successfully saved new transaction and accurately reflects on UI.	Pass	High

		12. Tap No to not use extracted information 13. Transaction detail form appears with the enhanced image 14. Fill in transaction details manually. 15. Click on save icon 16. Validation appears and page redirect to previous page					
NTM_ATUR_06	Users' open camera to capture image, crop, enhance and perform OCR for expenses or income	1. Take picture 2. Select "✓" to choose image captured 3. Resize and Crop Image 4. Select "✓" to choose the cropped image 5. Preview Image appears with cropped photo alongside prompt for enhancing image 6. Tap Yes 7. Enhancing loading indicator appears 8. Enhanced image preview appears alongside prompt to perform OCR on the image 9. Tap Yes 10. Performing OCR loading indicator appears 11. OCR completion pops up appear alongside prompt to see the result 12. Extracted information shown alongside prompt to use automate the filling in process 13. Tap No to not use extracted information	Device UUID Image: Receipt2.png	Permission to use camera appears. Pop up for selection appears. Transaction detail form appears. Validation showing "Transaction saved successfully" is displayed. Firebase saves new transaction.	Accurately display all permission and pop up. The form appears with the accurate enhanced image. The application successfully saved new transaction and accurately reflects on UI.	Pass	High

		14. Transaction detail form appears with the enhanced image 15. Fill in transaction details manually. 16. Click on save icon 17. Validation appears and page redirect to previous page					

Table 5.8: Test Case for Insight Page Module View Prediction

Feature Name: **Test Case Description:** **Testing Objective:**		**Insight Page Module – View Prediction** To verify that the predicted monthly expenses are generated and displayed accurately. To ensure that the prediction for the next month is automatically loaded or newly generated and displayed accurately in pie chart and table form with category breakdown and total.					
Test Case ID	**Test Scenario**	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status**	**Severity Failure**
IPM_VP_01	Users newly launch PennyLog to see insight page	1. Launch PennyLog 2. Home page display by default as landing page 3. Tap on insight page on bottom navigation bar 4. Page loads and displays predictions.	1. Device UUID 2. 12 Months data from expense_dataset and Firestore database	Auto generate prediction in pie chart and table format.	Prediction shows accurately.	Pass	High
IPM_VP_02	Users relaunch PennyLog on the same month	1. Relaunch PennyLog 2. Home page display by default as landing page 3. Tap on insight page on bottom navigation bar 4. Page loads and displays predictions.	1. Device UUID 2. Month June 2025 prediction saved in Firestore	System finds existing prediction for the month July and loads data from Firestore.	Predictions displayed automatically and accurately.	Pass	High

Table 5.9: Insight Page Module Insert Budget

Feature Name: **Test Case Description:** **Testing Objective:**		**Insight Page Module – Insert Budget** To verify that entering custom budgets will prompt the model to recalculate predicted values within the same categories. To ensure that the prediction model can recalculate the values accurately to adjust to the user specific budget, updates the Firestore and reflects on the UI.					
Test Case ID	**Test Scenario**	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status**	**Severity Failure**
IPM_IB_01	Users enter valid budget	1. Enter a valid budget 2. Click on "Update Budget" Button 3. App reloads and shows predictions.	1. Device UUID 2. Original: RM 2672 3. Budget: RM 2600	The page reloads and displays the adjusted predictions of the pie chart, table and total. Firestore saves the budget prediction as well.	The page reloads and displays the adjusted predictions accurately. Firestore successfully saved the budget prediction accurately.	Pass	High
IPM_IB_02	Users enter same budget	1. Enter a valid budget same as the prediction 2. Click on "Update Budget" Button 3. App reloads and shows predictions.	1. Device UUID 2. Original: RM 2672 3. Budget: RM 2672	The page reloads and displays the same predictions.	The page reloads and displays the same predictions accurately.	Pass	Medium
IPM_IB_04	Users enter higher budget	1. Enter a valid budget with higher value 2. Click on "Update Budget" Button 3. App reloads and shows predictions.	1. Device UUID 2. Original: RM 2672 3. Budget: RM 2700	The page reloads and displays the adjusted predictions of the pie chart, table and total. Firestore saves the budget prediction as well.	The page reloads and displays the adjusted predictions accurately. Firestore successfully saved the budget prediction accurately.	Pass	Medium

IPM_IB_04	Users enter invalid budget in alphabetic format	1. Enter an invalid budget in alphabetic format 2. Click on "Update Budget" Button	1. Device UUID 2. Original: RM 2672 3. Budget: abc	Displays validation error "Please enter a valid number"	Accurately displays validation error message.	Pass	High
IPM_IB_05	Users enter invalid budget in special character format	1. Enter an invalid budget in special character format 2. Click on "Update Budget" Button	1. Device UUID 2. Original: RM 2672 3. Budget: #@@	Displays validation error "Please enter a valid number"	Accurately displays validation error message.	Pass	High
IPM_IB_06	Users enter no value	1. Click on "Update Budget" Button with no value	1. Device UUID 2. Original: RM 2672 3. Budget: (empty)	Displays validation error "Budget is required"	Accurately displays validation error message.	Pass	High
IPM_IB_07	Users click on "Show original"	1. Enter a valid budget 2. Click on "Update Budget" Button 3. App reloads and shows predictions. 4. Click on "Show Original" button.	1. Device UUID 2. Original: RM 2672 3. Budget: RM2400	Immediately displays the original prediction stored in Firestore.	Accurately displays the original prediction.	Pass	High

Table 5.10: Settings Module Select Theme

Feature Name:	**Settings Module – Select Theme**
Test Case Description:	To verify that user can switch between light and dark themes. The selected theme should be applied in the entire app and are persistent across the app sessions.
Testing Objective:	To ensure that the theme selection is functional and can be applied through the UI as well as the preferences are saved in Firestore with the device UUID for consistency.

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Result	Actual Result	Status	Severity Failure
SM_ST_01	Users newly launch PennyLog to see theme setting page	1. Launch PennyLog 2. Home page display by default as landing page 3. Tap on settings page on bottom navigation bar 4. Selects "Select Theme" option under settings page	Device UUID	By default, the app displays light theme. Users are navigated to the settings page.	The app accurately displays with light theme with proper navigation.	Pass	High
SM_ST_02	Users select dark theme	1. Selects dark theme 2. App automatically apply changes 3. Firestore save theme preference.	1. Dark theme 2. Device UUID	The app automatically applies the dark theme changes across the whole app. Firestore saved latest preference for theme.	The app changes to dark theme across the whole app accurately. Firestore successfully saved preference.	Pass	High
SM_ST_03	Users select light theme	1. Selects light theme 2. App automatically apply changes 3. Firestore save theme preference.	1. Light theme 2. Device UUID	The app automatically applies the light theme changes across the whole app from the previously saved dark theme. Firestore saved latest preference for theme.	The app changes to light theme across the whole app accurately. Firestore successfully saved preference.	Pass	High
SM_ST_04	Users select dark theme and restart the app	1. Selects dark theme 2. App automatically apply changes 3. Users quit the app	1. Dark theme 2. Device UUID	The app relaunches with the latest saved theme preference	The app accurately relaunches with	Pass	High

		4. Users relaunch the app		which is the dark theme.	the applied dark theme.		
SM_ST_05	Users using light theme reselect light theme	1. Selects light theme 2. App automatically apply changes 3. Reselect light theme 4. No changes happen.	1. Light theme 2. Device UUID	No changes happen to the app as it is in light theme.	No changes happen.	Pass	Low

Table 5.11: Test Case for Settings Module Search and Select Currency

Feature Name: **Test Case Description:** **Testing Objective:**		**Settings Module – Search and Select Currency** To verify that user can view, search and select currencies. The selected currencies symbol should be applied in the entire app and are persistent across the app sessions. To ensure that the currency selection is functional and can be applied through the UI as well as the preferences are saved in Firestore with the device UUID for consistency. Currencies are provided by the currency_picker package that provides a list of ISO 4217 currencies.					
Test Case ID	**Test Scenario**	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status**	**Severity Failure**
SM_SASC_01	Users newly launch PennyLog to see currency setting page	1. Launch PennyLog 2. Home page display by default as landing page 3. Tap on settings page on bottom navigation bar 4. Selects “Currency Selector” option under settings page 5. Selected currency page are displayed	Device UUID	By default, the app uses Malaysian Ringgit (RM) currency.	The app accurately displays with currency.	Pass	High
SM_SASC_02	Users view currency list	1. Click on “Select Currency” Button 2. A list of currencies is shown in order.	Device UUID	A list of currencies provided by	The list is accurately displayed.	Pass	High

				currency_picker package			
SM_SASC_03	Users search USD currency	1. Click on “Select Currency” Button 2. A list of currencies is shown in order. 3. Users search for United States Dollar. 4. List are filtered and displays the list.	1. Device UUID 2. Keyword “USD” 3. Keyword “United States Dollar”	A list of filtered currencies provided by currency_picker package are shown that matches the keywords.	The list is accurately displayed.	Pass	High
SM_SASC_04	Users select USD currency	1. Users search for United States Dollar. 2. List are filtered and displays the list. 3. Select New currency. 4. App shows the selected currency page.	1. Device UUID 2. Keyword “USD” 3. Keyword “United States Dollar”	USD currency are picked and reflected in selected currency page. The currency is also updated across the whole app. Firestore saves the preference with the device UUID.	The currency is accurately reflected across the whole app. Firebase accurately stores the selected currency.	Pass	High
SM_SASC_05	Users select Hong Kong Dollar (HKD) currency and relaunch app	1. Users search for Hong Kong Dollar. 2. List are filtered and displays the list. 3. Select New currency. 4. App shows the selected currency page. 5. Restarts app.	1. Device UUID 2. Keyword “HKD” 3. Keyword “Hong Kong Dollar”	The app relaunches and the saved HKD preference are still displayed.	The app accurately shows the HKD currency upon app restart.	Pass	High

Table 5.12: Test Case for Settings Module Search and Select Language

Feature Name:		**Settings Module – Search and Select Language**					
Test Case Description:		To verify that user can view, search and select language. The selected language translation should be applied in the entire app and are persistent across the app sessions.					
Testing Objective:		To ensure that the language selection is functional and can be applied through the UI as well as the preferences are saved in Firestore with the device UUID for consistency. To speed up translation, translation is cached using SharedPreference and Static translation are used for fallback on missing translation.					
Test Case ID	**Test Scenario**	**Test Steps**	**Test Data**	**Expected Result**	**Actual Result**	**Status**	**Severity Failure**
SM_SASL_01	Users newly launch PennyLog to see language setting page	1. Launch PennyLog 2. Home page display by default as landing page 3. Tap on settings page on bottom navigation bar 4. Selects “Language Selector” option under settings page 5. Selected language page is displayed	Device UUID	By default, the app uses English Language. The selected language page displays “Selected Language: en”.	The app accurately displays with language.	Pass	High
SM_SASL_02	Users view language list	1. Click on “Select Language” Button 2. 4 languages are shown in the list.	Device UUID	List provided by language_picker package	The list is accurately displayed.	Pass	High
SM_SASL_03	Users search Chinese language	1. Click on “Select Language” Button 2. 4 languages are shown in the list. 3. Users search for Chinese language. 4. List are filtered and displays in the list.	1. Device UUID 2. Keyword “Chinese”	A list of filtered languages provided by language_picker package are shown that matches the keywords.	The list is accurately displayed.	Pass	High
SM_SASL_04	Users select Chinese Language as a new user	1. Users search for Chinese language. 2. List are filtered and displays in the list.	1. Device UUID 2. Keyword “Chinese”	Chinese Language are picked and reflected in selected language page. The	The Chinese Language is accurately translated across	Pass	High

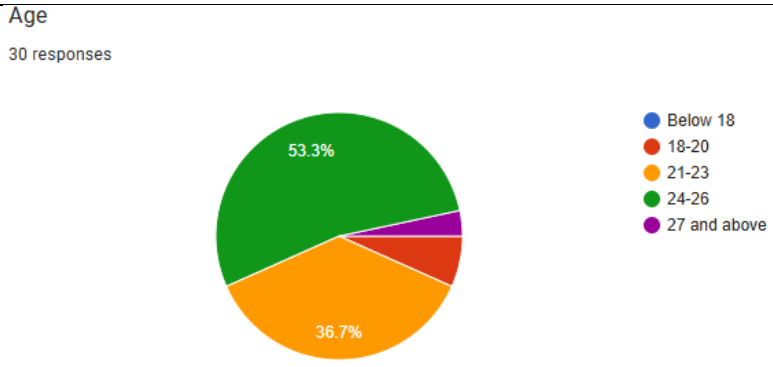
		3. Select Chinese language 4. Application loads for users using the language for the first time 5. The app applies translation across the whole app.		Chinese translation is also applied across the whole app. Firestore saves the preference with the device UUID.	the whole app. Firebase accurately stores the selected language.		
SM_SASL_04	Users select Tamil Language and restart app	1. Users search for Tamil language. 2. List are filtered and displays in the list. 3. Select Tamil language 4. The app applies translation across the whole app. 5. Restarts the app	1. Device UUID 2. Keyword "Tamil"	The app restarts with the Tamil Language across the whole app.	The Tamil Language is accurately used across the whole app.	Pass	High
SM_SASL_05	Users reselect English Language	1. Select English language 2. The app shows default language 3. Restarts the app	1. Device UUID 2. English Language	The app doesn't use translation and show default English Language.	The app accurately shows the default English Language across the app.	Pass	High
SM_SASL_06	Users reselect Chinese Language	1. Select Chinese language 2. The app reloads with Chinese Translation	1. Device UUID] 2. Chinese Language	The loads the app faster and doesn't retranslate because Shared Preference for cache translation.	The app accurately shows the Chinese Language Translation across the app.	Pass	High

5.3 User Acceptance Test (UAT)

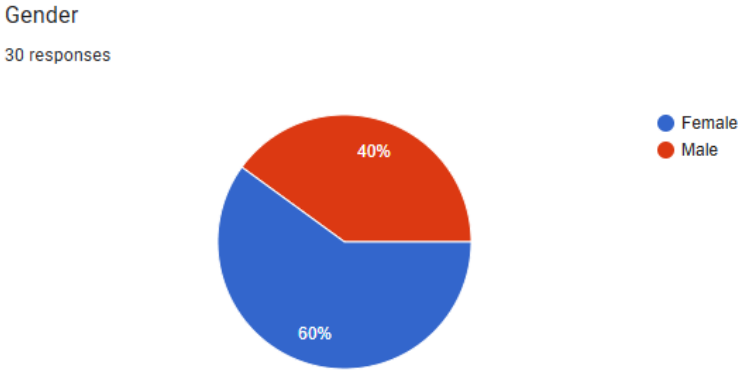
User Acceptance Test (UAT) is the final testing being conducted in a software development. When evaluating the testing result, if the software or application meets the acceptance criteria, it is ready for production and release. The procedure of this test is for users to perform testing according to the test plan.

To conduct this testing, the application is used by user by downloading the APK file and a total of 30 respondents feedback is collected and analysed below. The feedback will truly help with future improvements and to understand the limitations that the application has.

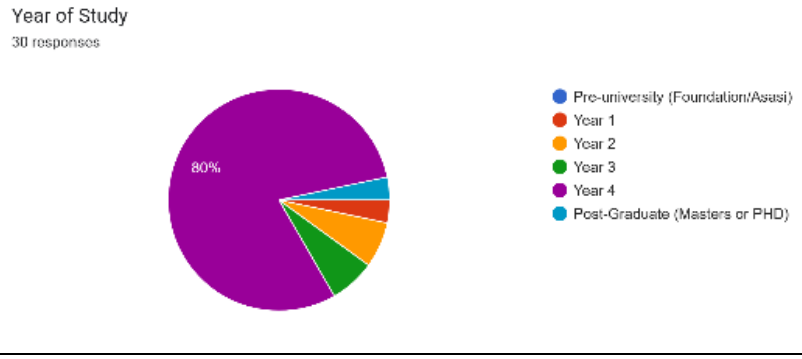
Section 1 is the demographic of the respondents that are chosen to conduct the testing as shown in Figure 5.1.



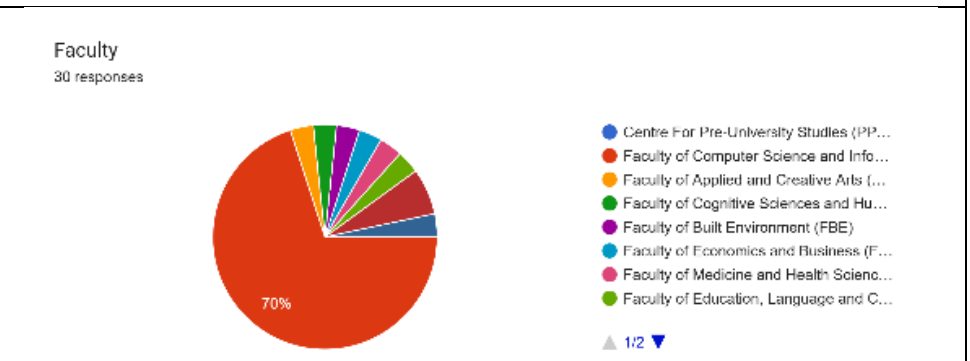
a) Age



b) Gender



c) Year of Study



d) Faculty

Figure 5.1: Section 1: Users Demographic

Overall, most of the respondents, 53.3% are users of age 24-26 based on Figure 5.1 (a), 60% are females based on Figure 5.1 (b), 80% are year 4 students based on Figure 5.1 (c) and 70% are from the faculty of Computer Science and Information Technology based on Figure 5.1 (d).

Section 2-6 are divided according to PennyLog Modules. The questionnaire has multiple choice grid where user need to rate between 1(strongly Disagree) to 5(Strongly Agree). Section 2 is the PennyLog Home Module. Table 5.13 presents the result of the multiple-choice grid questions grouped by each section of the Module while Table 5.14 compiles and summarises all the users' suggestions and comments for improvements on the module.

Table 5.13: Home Module User Acceptance Test Questions and Result

Section	Question	Result
Spending Overview	The Spending Overview gives a quick and clear summary of my current financial balance.	1: 0, 2: 1, 3: 2, 4: 8, 5: 19
	It is easy to distinguish between income and expenses in the overview.	1: 0, 2: 0, 3: 1, 4: 12, 5: 17
	The colour usage (e.g., blue for income, red for expenses) helps in understanding at a glance.	1: 0, 2: 1, 3: 2, 4: 11, 5: 16
	The amount displayed is accurate.	1: 0, 2: 1, 3: 2, 4: 8, 5: 18
Spending Trend Graph	The graph clearly shows my monthly spending trend.	1: 0, 2: 0, 3: 4, 4: 7, 5: 19
	It is easy to understand the data presented in the graph.	1: 0, 2: 0, 3: 3, 4: 6, 5: 21
	The graph helps me reflect on my spending patterns.	1: 0, 2: 0, 3: 2, 4: 9, 5: 19
Recent Transaction	The list of recent transactions is easy to read	1: 0, 2: 0, 3: 2, 4: 7, 5: 21
	The categories and amounts are clearly displayed.	1: 0, 2: 1, 3: 2, 4: 6, 5: 21
	I find it useful to see recent transactions directly on the home screen.	1: 0, 2: 0, 3: 2, 4: 7, 5: 21
	The timestamp and remark information provided is clear and helpful.	1: 0, 2: 0, 3: 1, 4: 7, 5: 22
Performance & System Quality of Home Page	The app loads quickly and responds without noticeable delay.	1: 1, 2: 0, 3: 4, 4: 12, 5: 13
	The data shown updates in real-time after changes.	1: 1, 2: 1, 3: 3, 4: 11, 5: 14
	The layout is efficient and helps me quickly understand my financial status.	1: 1, 2: 0, 3: 1, 4: 12, 5: 16
	The information shown is accurate	1: 0, 2: 1, 3: 0, 4: 13, 5: 16

	The app gives me enough control over my privacy and data.	<table><tr><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th></tr><tr><td>0</td><td>0</td><td>3</td><td>12</td><td>15</td></tr></table>

The Home module is evaluated based on three sections which is spending overview, spending trend graph and recent transactions as well as performance and system quality which includes speed, accuracy, privacy and responsiveness. The testing of home module received high levels of user satisfaction for all sections. It performed well in terms of clarity, accuracy and visual organisation. For performance wise, users found the module to be fast and reliable in terms of data displayed. There is minor concern on privacy and graph usefulness for reflection, which can be further improved.

Table 5.14: Home Module Improvement and Suggestion

Question	Answer
Any suggestion/comments for improving Home Module?	No.
	None
	The home module is easy to navigate so I personally think there's none
	-
	looks clean and easy on the eyes
	Personally, for me I would want the incomes to be green instead of blue and suggested to look into wrapping either in scaffold or streambuilder to ensure the UI works for every devices
	Add a label: Instead of just "Total Balance", consider: Net Balance (Income - Expenses) Optionally add a tooltip or info icon (ℹ) that explains how it's calculated.
	Add a legend or distinguish income vs expense in the chart with different colors.
	Add on features like scroll down to load data.
	Example: Current month is June but users wants to add on miss out spending's from April. After adding the expenses for April, users can scroll down to load data and the total balance will be change without having to tap on the month.
	1. The values for income and total balance take too long to update causing some confusion and could cause user to double register their income/expenses. 2. Spending Trend should include the total amount spent maybe just a little text to indicate how much spending per month

	(optional). 3. Recent transactions are very clear and helpful, however I tried to register an expense that I made and it registered two times instead of once. 4. The UI and responsiveness for the top bar could be better as it's not really aligned (I'm using the POCO X7 Pro for references).
	Fix the UI issue when pressing add button

For the improvement and suggestions of the home module, most users provided positive remarks with no improvement needed. Some improvement suggestions are on the visual clarity such as colour and labels, UI responsiveness across devise, loading behaviour for data, and a few functional bugs like duplicate transaction logging are reported and should be prioritised for amendment in upcoming fixes.

Section 3 is the PennyLog Transaction Module. Table 5.15 presents the result of the multiple-choice grid questions grouped by each section of the Module while Table 5.16 compiles and summarises all the users' suggestions and comments for improvements on the module.

Table 5.15 Transaction Module User Acceptance Test Questions and Result

Section	**Question**	**Results**
Transaction List & Filtering	The toggle between Income and Expenses is easy to use and works as expected.	**1** | **2** | **3** | **4** | **5** 0 | 0 | 2 | 8 | 20
	It is easy to find a specific transaction in the list.	**1** | **2** | **3** | **4** | **5** 0 | 1 | 1 | 11 | 17
	The transaction cards are clear and easy to read.	**1** | **2** | **3** | **4** | **5** 0 | 0 | 0 | 10 | 20
	The displayed information (date, amount, category, remark) is accurate.	**1** | **2** | **3** | **4** | **5** 0 | 0 | 0 | 10 | 20
Viewing Transaction Details	I can easily view full details of each transaction.	**1** | **2** | **3** | **4** | **5** 0 | 0 | 0 | 10 | 20

	The layout of transaction details is neat and easy to understand.	**1**	**2**	**3**	**4**	**5**
		0	0	1	10	19
	The attached receipt image is helpful in verifying transactions.	**1**	**2**	**3**	**4**	**5**
		0	0	0	9	21
	Transactions are displayed accurately	**1**	**2**	**3**	**4**	**5**
		0	0	0	8	22
Image Preview	The receipt image can be zoomed and moved easily.	**1**	**2**	**3**	**4**	**5**
		0	0	2	12	16
	The preview image function is responsive and accurate.	**1**	**2**	**3**	**4**	**5**
		0	0	0	9	21
Editing, Saving, and Deleting	I can edit the transaction details without difficulty.	**1**	**2**	**3**	**4**	**5**
		0	0	2	7	21
	The Save function is clear and confirms my changes successfully.	**1**	**2**	**3**	**4**	**5**
		0	0	1	9	20
	The Delete option is easy to find and provides a confirmation step.	**1**	**2**	**3**	**4**	**5**
		0	0	1	7	22
	I can upload new image	**1**	**2**	**3**	**4**	**5**
		0	0	0	9	21
Performance & System Quality of Home Page	The Page loads quickly when I switch between income and expenses.	**1**	**2**	**3**	**4**	**5**
		1	0	3	7	19
	Toggling between income and expense views is smooth and responsive.	**1**	**2**	**3**	**4**	**5**
		0	1	0	9	20
	Transaction data is displayed accurately and updates properly after edits or deletions.	**1**	**2**	**3**	**4**	**5**
		0	0	0	7	23
	The image preview and zoom functions work reliably and without lag.	**1**	**2**	**3**	**4**	**5**
		0	0	2	7	21
	Editing and saving a transaction completes quickly without issues.	**1**	**2**	**3**	**4**	**5**
		0	1	0	9	20
	I feel confident that my edits and deletions are successfully saved in the system.	**1**	**2**	**3**	**4**	**5**
		0	1	0	9	20
	Overall, the Transaction Page is stable, efficient, and performs well across all functions.	**1**	**2**	**3**	**4**	**5**
		0	0	2	10	18

The testing for transaction module receive good feedback as users finds it to be performing well, fast and reliable across all major functions. Users have confidence in the accuracy, applications stability and the security in transaction handling. Some minor consideration can be made on refinement in zoom and responsiveness on specific devices.

Table 5.16: Transaction Module Improvement and Suggestion

Question	**Answer**
Any suggestion/comments for improving Home Module?	No.
	Add search functionality
	None
	I like that I am able to scan receipt, so it is good
	very intuitive and very easily maneuvered
	-
	Use color-coded buttons (e.g., red for Expense, green or blue for Income) and maybe add icons next to "Income" and "Expense"?
	Category tag or color dot for fast scanning (e.g., red for bills, yellow for food).
	While editing the transactions, I found that the options are not visible when using the app in dark mode. I'll send you the video.
	1. You can include a search bar to easily find certain transactions as scrolling through the transactions would be quite tiring especially when I have a lot of transactions going on. 2. Displayed amounts are accurate but I have a previous problem of the transactions being double when I only input it once. 3. I would be good if the zoom in/zoom out function could be used using finger gestures as well instead of just using buttons to zoom in/zoom out (optional). 4. Editing is fine and clear, however in my opinion, I would love for the buttons especially save to be at the bottom especially when I scroll down and finished everything, I think it's nicer for me to see the button at the bottom instead of on top (optional). 5. I think the delete button shouldn't be at the top right corner as usually that is a place for buttons that users usually press to confirm instead of delete (optional). 6. In dark mode, the dropdowns to edit the transactions are not visible (i.e the color of the background of the dropdown is black - should be white or other colors that make the options visible).

	7. The status font in edit transaction is grey and is not really visible.

For the improvement and suggestions of the transaction module, users believe that the module are functioning well. Some suggestions on improvement can be made like adding search functionality, improving dark mode support, UI adjustment for button placement, fixing interaction flow, visual enhancement and fixes on minor bugs and visibility issues.

Section 4 is the PennyLog Add New Transaction Module. Table 5.17 presents the result of the multiple-choice grid questions grouped by each section of the Module while Table 5.18 compiles and summarises all the users' suggestions and comments for improvements on the module.

Table 5.17: Add New Transaction Module User Acceptance Test Questions and Result

Section	Question	Results				
Add New Transaction	The '+' button in the bottom navigation is easily visible and accessible.	**1**	**2**	**3**	**4**	**5**
		1	0	2	7	20
	The transaction type selection (Income / Expenses) is clear and easy to use.	**1**	**2**	**3**	**4**	**5**
		0	1	2	6	21
	I understand the difference between income and expense input.	**1**	**2**	**3**	**4**	**5**
		0	0	2	6	22
Filling the Transaction Form	The process of adding a new transaction is straightforward.	**1**	**2**	**3**	**4**	**5**
		0	1	1	7	21
	The layout of the transaction form is clear and easy to follow.	**1**	**2**	**3**	**4**	**5**
		0	0	1	10	19
	The input fields are easy to understand and complete.	**1**	**2**	**3**	**4**	**5**
		0	0	2	6	22
	The image upload or receipt feature is functional and helpful.	**1**	**2**	**3**	**4**	**5**
		1	0	1	8	20
	Transaction is saved and displayed accurately in the transaction module	**1**	**2**	**3**	**4**	**5**
		1	0	1	7	21

Adding a Receipt (Image Upload or Camera)	It is easy to upload a receipt image from my gallery or camera.	**1**	**2**	**3**	**4**	**5**
		2	0	0	4	24
	The process is straightforward	**1**	**2**	**3**	**4**	**5**
		1	0	1	7	21
	The app clearly shows the selected image before proceeding.	**1**	**2**	**3**	**4**	**5**
		1	0	0	3	26
Image Enhancement	The prompt asking whether to enhance the image is helpful.	**1**	**2**	**3**	**4**	**5**
		0	0	1	5	24
	The auto enhancement is useful	**1**	**2**	**3**	**4**	**5**
		0	0	1	9	20
	I noticed a clear improvement after enhancing the image.	**1**	**2**	**3**	**4**	**5**
		0	0	2	9	19
	Loading indicator keeps me inform of the process	**1**	**2**	**3**	**4**	**5**
		0	0	1	7	22
OCR (Text Extraction)	The prompt asking whether to perform OCR is clear and timely.	**1**	**2**	**3**	**4**	**5**
		1	0	1	7	21
	The OCR completed successfully without issues.	**1**	**2**	**3**	**4**	**5**
		1	1	1	10	17
	The OCR extraction is accurate	**1**	**2**	**3**	**4**	**5**
		1	1	1	12	15
	The option to view extracted results before using them is useful.	**1**	**2**	**3**	**4**	**5**
		1	0	0	9	20
	Loading indicator keeps me inform of the process	**1**	**2**	**3**	**4**	**5**
		1	0	1	6	22
Using Extracted Information (Autofill)	The extracted information (date, total, payment type, category) was mostly accurate.	**1**	**2**	**3**	**4**	**5**
		0	1	3	11	15
	The NLP-generated category matched the content of the receipt correctly.	**1**	**2**	**3**	**4**	**5**
		0	1	3	10	16
	Auto-fill feature is helpful in reducing manual work.	**1**	**2**	**3**	**4**	**5**
		0	2	1	9	18
	The extracted information is reflected accurately on the form	**1**	**2**	**3**	**4**	**5**
		0	1	1	12	16

Performance & System Quality of Add New Transaction	The Add Transaction form opens quickly and is responsive.	**1**	**2**	**3**	**4**	**5**
		0	0	1	8	21
	Uploading or capturing a receipt image works smoothly.	**1**	**2**	**3**	**4**	**5**
		2	1	0	6	21
	The cropping tool is responsive and easy to use.	**1**	**2**	**3**	**4**	**5**
		1	1	1	8	19
	Cropped images are saved and passed to the next step without issues.	**1**	**2**	**3**	**4**	**5**
		1	0	2	5	22
	The auto enhancement performs without error	**1**	**2**	**3**	**4**	**5**
		0	0	1	8	21
	The OCR (Text Extraction) process completes within a reasonable time.	**1**	**2**	**3**	**4**	**5**
		0	0	1	9	20
	The extracted OCR text is loaded correctly and displayed without issues.	**1**	**2**	**3**	**4**	**5**
		1	0	2	8	19
	Categorisations are accurate	**1**	**2**	**3**	**4**	**5**
		0	0	1	12	17
	The autofill from extracted data works reliably and populates the form correctly.	**1**	**2**	**3**	**4**	**5**
		0	0	3	8	19
	Saving the transaction (manually or auto filled) is quick and error-free.	**1**	**2**	**3**	**4**	**5**
		0	1	1	10	18

The testing for add new transaction module received well with users accepting well on the navigation, form layout, receipt upload and the overall flow. User appreciates the clear previews, smooth upload, image enhancement, OCR and NLP capabilities that helps in automation. Some areas of improvement are on the OCR precision and enhancing NLP categorisation accuracy

Table 5.18: Add New Transaction Module Improvement and Suggestion

Question	**Answer**
Any suggestion/comments	No.
	perhaps more AI, and less button clicks to complete a single task

for improving Home Module?	Perhaps the OCR module might be a bit inaccurate when uploading incorrect/blurry images
	The RM 0.00 input lacks visual affordance (doesn't look editable). Suggestion: Make it a text field with a border or underline to show it's tappable/editable. The same case goes for the "Add Remark" field.
	Consider making the date field tappable with a visible border or calendar icon button.
	1. The layout of the transaction form is easy to understand, however I think if the open camera function is to be integrated along with the upload image function (i.e when pressing upload image, the option to open camera is also available in the popup - optional) would be better. 2. The transactions sometimes display wrong values as it sometimes confuses between table number, etc. instead of the actual total (on some receipts I had to cover these numbers in order to get it to detect the right value). 3. Loading time of the OCR is a bit long (subjective) and I think it would be better if the loading popup has a cancel button just in case I want to cancel it when it is loading for too long. 4. On some receipts, it is categorized differently as it should be food/groceries.

For the improvement and suggestions of the add new transaction module, users generally notes that the modules layout are clear and functions well, with the uploading and image-based features comes out as helpful and integrated well. Some areas of improvement would be the OCR accuracy on low clarity receipts, fixing some visual clarify and make it editable for some form fields, streamline the camera and upload image access, enable the image processing to be cancelled, refine on the NLP classification and loading time.

Section 5 is the PennyLog Insight Module. Table 5.19 presents the result of the multiple-choice grid questions grouped by each section of the Module while Table 5.20 compiles and summarises all the users' suggestions and comments for improvements on the module.

Table 5.19: Insight Module User Acceptance Test Questions and Result

Section	Question	Results
Expense Prediction	The predicted total expense for the upcoming month is clearly shown.	1: 2, 2: 1, 3: 2, 4: 4, 5: 21
	The pie chart helps me understand how my predicted expenses are distributed by category.	1: 1, 2: 0, 3: 2, 4: 9, 5: 18
	The colour coding in the pie chart makes the categories easy to distinguish.	1: 1, 2: 0, 3: 1, 4: 6, 5: 22
	The accompanying table provides useful details that complement the chart.	1: 1, 2: 1, 3: 3, 4: 7, 5: 18
	The displayed month and year are accurate	1: 1, 2: 1, 3: 3, 4: 3, 5: 22
	The insights provided help me understand my spending habits.	1: 1, 2: 1, 3: 2, 4: 7, 5: 19
	The graphs and data visualisations are easy to read.	1: 0, 2: 1, 3: 2, 4: 3, 5: 24
	I find the forecasting feature useful.	1: 0, 2: 2, 3: 4, 4: 5, 5: 19
Budget Optimisation	Entering a budget amount is simple and straightforward.	1: 0, 2: 0, 3: 0, 4: 5, 5: 25
	The system responds quickly after I update the budget.	1: 1, 2: 0, 3: 1, 4: 5, 5: 23
	The new prediction based on the updated budget feels relevant and helpful.	1: 1, 2: 0, 3: 2, 4: 6, 5: 21
	I understand how the budget input affects the forecasted spending distribution.	1: 0, 2: 0, 3: 4, 4: 8, 5: 18
	I find the budget forecasting feature useful.	1: 1, 2: 0, 3: 3, 4: 8, 5: 18
Performance & System Quality of Insight Module	The Insight page loads quickly when accessed.	1: 1, 2: 0, 3: 5, 4: 4, 5: 20
	The pie chart and table render correctly and without lag.	1: 3, 2: 0, 3: 2, 4: 8, 5: 17

	Budget input is accepted instantly and processed without delay.	**1** **2** **3** **4** **5** 3 0 1 5 21
	The updated prediction based on budget input appears quickly and without glitches.	**1** **2** **3** **4** **5** 3 0 1 9 17
	The displayed prediction values (total and category breakdown) are consistent and accurate.	**1** **2** **3** **4** **5** 3 0 2 7 18

The testing for insight page has high satisfaction rate on how visually engaging the content, informative prediction, having a responsive and stable budget adjustment feature, and high clarify in the charts, tables and labels. Some improvement areas would be the trust on the forecasting algorithm, provide better user guidance on how the budget input affects the prediction and considering tooltips for explaining the prediction logic.

Table 5.20: Insight Module Improvement and Suggestion

Question	**Answer**
Any suggestion/comments for improving Home Module?	Add another feature where users can see previous month's pie chart or prediction for the current month based on the updated budget.
	None
	It can predict my spending, I think it is useful
	quite insightful
	I LOVE THIS FEATURE!
	For table, can add sortable columns and percentage columns
	Adding a legend or labels directly on the chart can help users quickly identify categories without referring to the table.
	No

For the improvement and suggestions of the insight module, the module is well received by users with several leaving comments like its useful, insightful and

they love the feature. They have a strong positive perception of the forecasting functionality. Some enhancement opportunity will be to add previous predictions for comparison, improve the data readability like sorting table and percentage value as well as adding chart legends and hover info.

Section 6 is the PennyLog Setting Module. Table 5.21 presents the result of the multiple-choice grid questions grouped by each section of the Module while Table 5.22 compiles and summarises all the users' suggestions and comments for improvements on the module.

Table 5.21: Settings Module User Acceptance Test Questions and Result

Section	**Question**	**Results**
Theme	Switching between light and dark theme is easy and instant.	**1** 0 | **2** 0 | **3** 0 | **4** 9 | **5** 21
	The selected theme is applied consistently across the app.	**1** 0 | **2** 0 | **3** 0 | **4** 8 | **5** 22
	My theme preference is retained even after restarting the app.	**1** 0 | **2** 0 | **3** 0 | **4** 7 | **5** 23
Currency	Searching for a specific currency is simple and effective.	**1** 0 | **2** 0 | **3** 0 | **4** 4 | **5** 26
	The selected currency is applied correctly across the app (e.g., totals, transactions).	**1** 1 | **2** 0 | **3** 2 | **4** 6 | **5** 21
	My currency preference is saved even after restarting the app.	**1** 0 | **2** 0 | **3** 0 | **4** 5 | **5** 25
Language	It is easy to search and select a preferred language.	**1** 0 | **2** 0 | **3** 0 | **4** 5 | **5** 25
	The app updates the language immediately and accurately after selection.	**1** 0 | **2** 1 | **3** 3 | **4** 4 | **5** 22
	My language preference is saved even after restarting the app.	**1** 0 | **2** 0 | **3** 0 | **4** 4 | **5** 26
	Translation speed is fast	**1** 0 | **2** 1 | **3** 2 | **4** 5 | **5** 22

Performance & System Quality of Settings Module	The Settings page opens quickly and is easy to navigate.	**1**	**2**	**3**	**4**	**5**
		0	0	0	4	26
	The Theme applied instantly and without issues.	**1**	**2**	**3**	**4**	**5**
		0	0	0	6	24
	The currency selection responds quickly and updates across the app.	**1**	**2**	**3**	**4**	**5**
		0	0	1	9	20
	The language selection is applied correctly and immediately throughout the app.	**1**	**2**	**3**	**4**	**5**
		0	0	3	4	23
	The selected settings persist after closing and reopening the app.	**1**	**2**	**3**	**4**	**5**
		0	0	0	5	25
	The search functions are fast and accurate.	**1**	**2**	**3**	**4**	**5**
		0	0	0	6	24

The testing for insight page which has the highest user satisfaction rating due to it being highly accessible and the preference are applied across the whole application with correct selection saved. The search and switching operations are fast and accurate as well. The mode, translation and currency switch are also responsive. No major issues were detected.

Table 5.22: Settings Module Improvement and Suggestion

Question	**Answer**
Any suggestion/comments for improving Home Module?	No
	No.
	None
	more ability to use like a hybrid colour scheme and more different language supports like local malaysian languages
	Add exchange currency
	-
	May consider data export function.
	Provide Tooltip Explanations
	1. Some of the UI's are not consistent and the options are not visible. 2. The selected currency is applied but is not converted to their own value when the user has inputted transactions (will make it harder for the user especially if the user has a lot of transactions).

	3. The change in language and currency sometimes needs the user to restart the app in order for the changes to take place - restarting took some time when this is applied. 4. Currency selection sometimes needs app restart as well to apply. 5. Some of the words in the app changes to 'null' when language is changed (i.e in my case I changed from English to Chinese and some words in the Insights page turns into 'null').

Lastly for the improvement and suggestions of the settings module, users are generally satisfied with the current feature due to the ease of use. Some improvement can be made to fixing the translation bug that turns the content to null or no translation. Besides that, adding informative tooltips and new features like exporting documents. Furthermore, improve on the customisation like having more language translation.

5.4 Summary

The chapter has outlined the testing which was proposed for PennyLog: Expense Tracker Predictive Budget Optimisation. The functionality testing which was conducted using black box testing technique ensure that every functionality is working as defined. The details are outlined using the test case template to show the result for each feature under every module. Besides that, user acceptance test was also conducted by involving 30 participants of varying courses and backgrounds. Their feedback is recorded and analyse to better understand the limitation and improvement which was needed to make PennyLog better

CHAPTER 6: CONCLUSION AND FUTURE WORK

6.1 Introduction

This chapter will be concluding the implementation, development and testing of the proposed application. The objective of the project will be reviewed to make sure that all objective has been achieved. Furthermore, the constraints encountered and potential improvements for future work to improve PennyLog will be discussed in this chapter as well.

The final year project PennyLog by integrating artificial intelligence (AI) for text extraction, text categorisation and time series forecasting has been successfully completed. The project which is executed for two semesters has successfully been hosted on the backend on Google Cloud Run and the frontend is available through downloading the APK file. All the chapters in the report have showcased the dynamic process in implementing, developing and enhancing PennyLog. Each of the chapters and processes in the XP methodology are important in leading to the successfully implementing the application

6.2 Objective Achievement

Table 6.1 shows the project objectives alongside the accomplishment made.

Table 6.1: Objective Achievement

Objective	**Accomplishments**
To analyse users' expenses before and after using the app's expense prediction and budget optimisation features	Successfully designed and implemented a time series forecasting model that performs monthly expense prediction alongside a budget optimisation feature which accepts user defined budgets as input that recalculates the total expenses. Users can now compare their actual spendings before and after using this feature which are aimed to improve financial awareness and encourage better planning.
To design and implement a system that accurately logs expenses and	Developed and deployed PennyLog as a mobile application which integrates OpenCV for image enhancement,

income using image-to-text (OCR) and NLP technologies.	Pytesseract for OCR and Sentence Transformer for NLP-based categorisation. User can upload or capture receipt, crop, enhance image, extract text and auto-fill transaction details more efficiently.
To test and evaluate on the usability and functionality of the system solution through user testing.	Conducted user testing across all 5 modules. The system met both functional and usability expectations with users leaving helpful and insightful feedback in terms of the apps accuracy, performance security, reliability and ease of use.

6.3 Project Limitations

Despite the successful implementation and development of PennyLog, there were several limitations that were identified:

1. **Auto Enhancement Limitation**

   Implementing auto enhancement algorithm using OpenCV python library has its limitation because the library may not be able to consistently produce the best result for all type of receipt images. Considering the variation in lightning, blur and background noise on the image, the enhancement quality is affected that leads to poor OCR results.

2. **OCR Accuracy**

   The text extraction accuracy relies on the pytesseract OCR, which uses the Tesseract Engine. The Tesseract depends highly on the image quality and how clear it is. Any noise, fadedness, or skewed presented in the image may result in the inaccurate or incompletely text extraction.

3. **Prediction Model Accuracy**

   The time series prediction model for forecasting the future month expenses might predict inaccurate results if there is not enough data presented during training and during prediction. Additionally, external factors such as

inflation, sudden changes in lifestyle, inconsistent spending pattern might confuse the model and limits the model reliability.

4. **Availability on backend server running on Google Cloud Run**

   The backend services is running and hosted on Google Cloud Run Free Tier plan which has limitation. The plan causes delay in the result returned by the server as well as a cold start issue if the server is idle for a period. This affects the real time interaction of the application with the server.

5. **Cross Platform Limitation**

   Currently, the application is only deployed to Android devices. While flutter is flexible and enables cross platform by design, the application has yet to be adapted for iOS or web platform which limits the accessibility form reaching a wider audience.

6. **Currency Conversion Limitation**

   The current currency implementation supports only changing the currency symbol. It doesn't perform real-time currency conversion based on real time exchange rates, which causes inaccuracies for users that wants to handle transactions in different currencies.

7. **Limited Image Upload**

   Currently the application only allows users to upload one image per transaction which is not convenient since some receipts can be two pages long.

8. **Lack of Insight for year**

   The prediction is limited to the current year from January to December. However, some users may want to compare predictions from previous year to better understand if there is improvement in their spending patterns.

6.4 Future Work

To overcome the limitations presented and to improve the performance as well as the user experience of PennyLog Application in the future, there is several future developments that can be implemented:

1. **Ai-Assisted Image Enhancement Using Gemini:**

   Future improvement may include using a different technique such as implementing Google's Gemini Large Language Model (LLM) to help in analysing image quality and dynamically recommend or apply enhancement on the image automatically. Through the combination of Gemini reasoning capabilities with the OpenCV techniques, the application can adaptively enhance image based on each images context which can lead to better OCR outputs.

2. **OCR improvement**

   The Pytesseract OCR improvement can parallelly be improved alongside the image enhancement using Gemini. Besides that, is considering exploring a different OCR engine available in the market such as PaddleOCR, EasyOCR, Microsoft Azure OCR, AWS Textract and Google Cloud Vision OCR or customising a new OCR model.

3. **Model Accuracy Improvement**

   To improve on the model accuracy, some additional improvements can be made such as incorporate input features such as transaction frequency, category wise breakdown, and rolling averages which can help in learning external factors that affects the models. Besides that, a more advanced architecture such as GRU, Transformer based model or hybrid model can be considered. Some other suggestions such a time series cross validation,

hypermeter tuning and regular retraining with updated user data is also recommended.

4. **Improve backend limitation**

   To eliminate the cold start issue, future consideration can me made into using a cloud schedular API that can trigger the backend server to run without it going onto a cloud start (sleeping).

5. **Expand to cross platform support**

   Development can be considered to extend to iOS and web platforms to increase accessibility. This was more users can access the application and help in a more device specific testing.

6. **Adding real time currency conversion**

   Future updates can integrate a live exchange for currency by including API such as ExchangeRate-API or Open Exchange Rate. This will allow more users to track their expenses in multiple currencies making the application a more globally adaptable financial management tool.

7. **Upload more than one image or merge image**

   Allowing users to upload more than one image will give them flexibility and make the application more appealing compared to the ones in the market. Another capability to be considered is to be able to merge the images into one image if it's the same receipt.

8. **Include insight from previous year**

   Add on previous years predictions in the drop down menu to allows user filter the prediction based on year so that users can look back on previous prediction and compare it to their total spendings.

References

Abrahams, S., Scarpinelli, A., Hafner, D., & Erwitt, E. (2016). *TensorFlow for machine intelligence*. http://katalog.ub.uni-heidelberg.de/titel/68278188

Ahirwar, R., Saxena, R., & Sunil Sangle, H. (2022). Real Time Text Detection and Recognition using Pytesseract. *International Journal of Advanced Research in Science, Communication and Technology (IJARSCT)*, *2*(2), 431–434. https://ijarsct.co.in/Paper2769.pdf

Al Fajar, M., Dar, M. H., & Rohani, R. (2022). Application of the Waterfall model in the development of family planning participants information system. *Sinkron, 7*(2). https://doi.org/10.33395/sinkron.v7i2.11387

Alqurashi, A., & Alawneh, L. (2024). Stacked ensemble deep learning for the classification of nonfunctional requirements. *IEEE Transactions on Reliability*, 1–15. https://doi.org/10.1109/tr.2024.3513834

Alsaqqa, S., Sawalha, S., & Abdel-Nabi, H. (2020). Agile Software Development: Methodologies and Trends. *International Journal of Interactive Mobile Technologies (iJIM)*, *14*(11), 246. https://doi.org/10.3991/ijim.v14i11.13269

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, *8*(1). https://doi.org/10.1186/s40537-021-00444-8

Babu S, S. (2024). Trends in IT Project Management: Agile Methodologies and Their Impact on Project Success. *Journal of Recent Trends in Computer Science and Engineering (JRTCSE)*, *12*(2), 73–81. https://www.researchgate.net/publication/383373919_Trends_in_IT_Project_Management_Agile_Methodologies_and_Their_Impact_on_Project_Success

Bannigidad, P., & Sajjan, S. P. (2024). Ancient Kannada Handwritten Character Recognition from Palm Leaf Manuscripts Using PyTesseract-OCR Technique. In *Communications in computer and information science* (pp. 154–162). https://doi.org/10.1007/978-3-031-60725-7_12

Barr, P., & Noble, J. (2004). *Extreme programming system metaphor: A semiotic approach*. Retrieved from https://compart.uni-bremen.de/content/4-teaching/0-winter-2020-21/1-semiotics-media/3-material/12_khaled_extremeprogramming.pdf

Bhatele, P., Mahajan, D., Mahajan, B., Mahajan, D., Mahajan, N., & Mahajan, P. (2023). TrackEZ Expense Tracker. *2023 4th International Conference for Emerging Technology (INCET)*, *9*, 1–5. https://doi.org/10.1109/incet57972.2023.10170735

Chahal, A., & Gulia, P. (2018). Machine Learning and Deep Learning. *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, *8*(12), 2278–3075.

https://www.researchgate.net/publication/364097061_Machine_Learning_and_Deep_Learning

Dadhich, A., Jain, S., Jain, S., & Mathur, S. (2022). Expense Tracker. *International Journal of Research and Analytical Reviews(IJRAR)*, *10*(2), 777–783. https://doi.org/10.13140/RG.2.2.11881.47204

De Amorim, L. B., Cavalcanti, G. D., & Cruz, R. M. (2022). The choice of scaling technique matters for classification performance. *Applied Soft Computing*, *133*, 109924. https://doi.org/10.1016/j.asoc.2022.109924

Dhruv, A. J., Patel, R., & Doshi, N. (2020). Python: The Most Advanced Programming Language for Computer Science Applications. *Proceedings of the International Conference on Culture Heritage, Education, Sustainable Tourism, and Innovation Technologies (CESIT 2020)*, 292–299. https://doi.org/10.5220/0010307902920299

ELHAG, A. (2024). ENHANCING UNDERGRADUATE UX EDUCATION WITH WIZARD OF OZ AND PAPER PROTOTYPING. *Global and Lokal Distance Education- GLOKALde*, *10*(1), 3. https://www.glokalde.com/pdf/issues/25/Article3.pdf

Faroq Santoso, M. (2024). Implementation of UI/UXConcepts and techniques In Web Layout Design with FIGMA. *Jurnal Teknologi Dan Sistem Informasi Bisnis*, *6*(2), 279–285. http://103.241.192.17/~jurnalunidha/index.php/jteksis/article/view/1223/758

Greiner-Petter, M., & Ibachb, M. (2021). *Digital tools for collaborative design processes*. https://irf.fhnw.ch/server/api/core/bitstreams/0eda7328-941b-4c34-be9f-53df60891809/content

Hamid Shoukry, T. A., & Aly Gaber, R. (2023). A Proposed Best Practices for Agile Approach - XP. *International Journal for Scientific Research (IJSR)*, *2*(3). https://vsrp.co.uk/wp-content/uploads/9-IJSR-Vol.-2-No.-3-Mar-2023-All.pdf#page=24

Harsshita, N. S., Shruthi, N. A., Srinidhi, N. B. K., & Sandhya, N. D. A. (2024). Advanced Personal Budget Analytics: combining optical character recognition and natural language processing for automated budget categorization and insight extraction. *Deleted Journal*, *2*(10), 2463–2469. https://doi.org/10.47392/irjaeh.2024.0337

H. Sarker, I. (2020). Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions. *Springer Nature Link*, *2*(140). https://link.springer.com/article/10.1007/s42979-021-00815-1

H. Sarker, I. (2022). Machine Learning: Algorithms, Real-World Applications and Research Directions. *Springer Nature*, *2*(160). https://doi.org/10.1007/s42979-021-00592-x

Joseph, F. J. J., Nonsiri, S., & Monsakul, A. (2021). Keras and TensorFlow: A Hands-On Experience. In *EAI/Springer Innovations in Communication*

and Computing (pp. 85–111). https://doi.org/10.1007/978-3-030-66519-7_4

Khatiwada, P., & Dhakal, P. (2024). Evaluating Serverless Machine Learning Performance on Google Cloud Run. *Distributed, Parallel, and Cluster Computing*. https://doi.org/10.48550/arXiv.2406.16250

Khurana, D., Koli, A., Khatter, K., & Singh, S. (2022). Natural language processing: state of the art, current trends and challenges. *Multimedia Tools and Applications*, *82*(3), 3713–3744. https://doi.org/10.1007/s11042-022-13428-4

K.N. Leung, H., & W.L. Wong, P. (1997). A study of user acceptance tests. *Software Quality Journal*, *6*, 136–149. https://doi.org/10.1023/A:1018503800709

Koo, X., & Khor, K. (2023). Expense tracking with Tesseract Optical Character Recognition v5: a mobile application development. *2023 IEEE Symposium on Industrial Electronics & Applications (ISIEA)*, *44*, 1–5. https://doi.org/10.1109/isiea58478.2023.10212265

Kumar, A. (2023, May 10). *Python Tesseract PDF & OCR Example - Analytics Yogi*. Analytics Yogi. https://vitalflux.com/python-tesseract-pdf-ocr-example/

Mahamkali, N., & Ayyasamy, V. (2015). OpenCV for Computer Vision Applications. *Proceedings of National Conference on Big Data and Cloud Computing (NCBDC'15)*, 52–56. https://www.researchgate.net/publication/301590571_OpenCV_for_Computer_Vision_Applications

Money Lover. (2024, December 5). Google Play. https://play.google.com/store/apps/details?id=com.bookmark.money

Moreno-Garcia, C. F., Jayne, C., Elyan, E., & Aceves-Martins, M. (2023). A novel application of machine learning and zero-shot classification methods for automated abstract screening in systematic reviews. *Decision Analytics Journal*, *6*, 100162. https://doi.org/10.1016/j.dajour.2023.100162

MRS.G.S.N Malleswari, P. Manikanta, R.Venkata Lavanya, A.Avinash, & B.Tarak. (2024). TIME SERIES ANALYSIS USINGPYTHON. *International Journal of Techno-Engineering*, *16*(2), 234–241. http://ijte.uk/archive/2024/TIME-SERIES-ANALYSIS-USING-PYTHON.pdf

Niranjanamurthy, M., Navale, S., Jagannatha, S., & Chakraborty, S. (2018). Functional software testing for web applications in the context of industry. *Journal of Computational and Theoretical Nanoscience*, *15*(11), 3398–3404. https://doi.org/10.1166/jctn.2018.7632

Ohlsson, V. (2016). Optical Character and Symbol Recognition using Tesseract. *Master of Science in Engineering Technology Computer Science and Engineering*. http://ltu.diva-portal.org/smash/record.jsf?pid=diva2:1019846

Omonije, A. (2024). Agile Methodology: A comprehensive impact on modern business operations. *International Journal of Science and Research (IJSR)*, *13*(2), 132–138. https://doi.org/10.21275/sr24130104148

Panja, M., Chakraborty, T., & Kumar, U. (2022). Data Dictionary. *Springer Nature Switzerland AG 2022*. https://doi.org/10.1007/978-3-030-26050-7_75-2

Pirani, M., Thakkar, P., Jivrani, P., Bohara, M. H., & Garg, D. (2022). A Comparative Analysis of ARIMA, GRU, LSTM and BiLSTM on Financial Time Series Forecasting. *2022 IEEE International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE)*, 1–6. https://doi.org/10.1109/icdcece53908.2022.9793213

Pugliese, R., Regondi, S., & Marini, R. (2021). Machine learning-based approach: global trends, research directions, and regulatory standpoints. *Data Science and Management*, *4*, 19–29. https://doi.org/10.1016/j.dsm.2021.12.002

pytesseract. (2024, August 16). PyPI. https://pypi.org/project/pytesseract/?source=post_page-----cd670ee38052----------------------------

Receipt Scanner: Spend Wise. (2024). Apple App Store. https://apps.apple.com/th/app/receipt-scanner-spend-wise/id6478102657

R, V. K. G. (2024, January 20). 07] Standardization and normalization techniques in machine learning: StandardScaler(), MinMaxScaler(), Normalizer()&RobustScaler(). *Medium*. https://medium.com/@vinodkumargr/07-standardization-and-normalization-techniques-in-machine-learning-standardscaler-3890a89bddb

Saadeldin Awadalla, M. M., Satam, P. M., Mohamed Said Al Rahbi, S. S., & Kumaran Nair, S. S. (2023). A Mobile Application for Expenditure Tracker. *6th Middle East College Student Research Conference Proceeding*. https://www.jsr.org/index.php/path/article/view/2153

Samrat Medavarapu, S. (2024). Advancements in Deep Learning: A review of KERAS and TensorFlow frameworks. *Journal of Scientific and Engineering Research*, *11*(5). https://jsaer.com/download/vol-11-iss-5-2024/JSAER2024-11-5-282-286.pdf

Saxena, S. (2024, November 26). *What is LSTM? Introduction to Long Short-Term Memory*. Analytics Vidhya. Retrieved December 7, 2024, from https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/

Semma, A. B., Ali, M., Saerozi, M., Mansur, M., & Kusrini, K. (2022). Cloud computing: google firebase firestore optimization analysis. *Indonesian Journal of Electrical Engineering and Computer Science*, *29*(3), 1719. https://doi.org/10.11591/ijeecs.v29.i3.pp1719-1728

Shelke, S., Shingre, M., Lebisha, S., & Shaikh, S. (2023). Smart BAT - Smart Budget Analyzer and Tracker. *2023 International Conference on Advanced Computing Technologies and Applications (ICACTA)*, *04*, 1–5. https://doi.org/10.1109/icacta58201.2023.10392452

Shihab Ahmed, D., Ibrahim Al-badri, R. S., Ali Hashim, F., & Mahmood Hussien, N. (2023). KERAS DEEP LEARNING PACKAGE IN PYTHON: A REVIEW. *European Journal of Interdisciplinary Research and Development*, *19*, 24–29. https://www.ejird.journalspark.org

Siami-Namini, S., Tavakoli, N., & Namin, A.S. (2019). A Comparative Analysis of Forecasting Financial Time Series Using ARIMA, LSTM, and BiLSTM. *ArXiv, abs/1911.09512*.

Singla, S., Kaur, A., Anju, Soni, A., Dhaiya, R., & Kaur, G. K. (2022). Unveiling Financial insights: The Daily Expense Tracker System approach. *2024 International Conference on Emerging Innovations and Advanced Computing*. https://doi.org/10.1109/INNOCOMP63224.2024.00079

Singhal, R. (2024). *FINANCIAL PERFORMANCE ANALYSIS USING MACHINE LEARNING ALGORITHMS: POST-IPO OF NYKAA*. Dublin Business School. Retrieved January 10, 2025, from https://esource.dbs.ie/server/api/core/bitstreams/b971eccd-5ffe-4803-8d9b-caf18278edba/content

Smith, R. (2007). An overview of the Tesseract OCR engine. *Proceedings of the International Conference on Document Analysis and Recognition*, 629–633. https://doi.org/10.1109/icdar.2007.4376991

Snehkunj, R., & Vachiyatwala, K. (2022). Data Analysis Using Pandas Library of Python. *Acta Scientific COMPUTER SCIENCES*, *4*(3). https://actascientific.com/ASCS/pdf/ASCS-04-0236.pdf

Shivahare, B. D., Singh, A. K., Uppal, N., Rizwan, A., Vaathsav, V. S., & Suman, S. (2022). Survey Paper: Study of Natural Language Processing and its Recent Applications. *2022 2nd International Conference on Innovative Sustainable Computational Technologies (CISCT)*, 1–5. https://doi.org/10.1109/cisct55310.2022.10046440

Sri Swathiga, U. U. A., P. Vinodhin, & V. Sasikala. (2021). AN INTERPRETATION OF DART PROGRAMMING LANGUAGE. *Dogo Rangsang Research Journal*, *11*(10). https://www.researchgate.net/publication/358661479_AN_INTERPRETATION_OF_DART_PROGRAMMING_LANGUAGE

Srivastava, N., Shree, U., Ram Chauhan, N., & Kumar Tiwari, D. (2017). FIREBASE CLOUD MESSAGING (ANDROID). *International Journal of Innovative Research in Science, Engineering and Technology*, *6*(9). https://www.ijirset.com/upload/2017/cotii/3_CS_COTII_2017_Firebase_cloud.pdf

Staiano, F. (2022). *Designing and prototyping interfaces with FIGMA*. Packet Publishing Ltd.

https://books.google.com.my/books?hl=en&lr=&id=GOBeEAAAQBAJ&oi=fnd&pg=PP1&dq=figma&ots=elasmL7V4L&sig=fxW0Pm7GrOboVw8Ct6oZ8EO4DvQ&redir_esc=y#v=onepage&q=figma&f=false

Sunday, D. O. (2024). Application of Long Short-Term Memory (LSTM) in stock price prediction. *International Journal of Development and Economic Sustainability*, *12*(3), 36–45. https://doi.org/10.37745/ijdes.13/vol12n33645

Tan, J., Chen, Y., & Jiao, S. (2024). Visual Studio code in Introductory Computer Science Course: An Experience Report. *Proceedings of ACM Conference (Conference'17)*. https://doi.org/10.18260/1-2--48259

Tashildar, A., Shah, N., Gala, R., Giri, T., & Chavhan, P. (2020). APPLICATION DEVELOPMENT USING FLUTTER. *International Research Journal of Modernization in Engineering Technology and Science*, *02*(08). https://www.irjmets.com/uploadedfiles/paper/volume2/issue_8_august_2020/3180/1628083124.pdf

Tashtoush, Y. M., Darweesh, D. A., Husari, G., Darwish, O. A., Darwish, Y., Issa, L. B., & Ashqar, H. I. (2021). Agile approaches for cybersecurity systems, IoT and intelligent transportation. *IEEE Access*, *10*, 1360–1375. https://doi.org/10.1109/access.2021.313686

Vasilakos, S., Iacobellis, G., Stylios, C. D., & Fanti, M. P. (2012). Decision Support systems based on a UML description approach. *2012 IEEE 6th International Conference 'Intelligent Systems,'* 041–046. https://doi.org/10.1109/is.2012.6335188

V Geetha, V Sudheer, & Gomathy. (2022). OPTICAL CHARACTER RECOGNITION. *JOURNAL OF ENGINEERING, COMPUTING & ARCHITECTURE*, 1934–7197. https://www.researchgate.net/publication/360620085_OPTICAL_CHARACTER_RECOGNITION

Veryfi. (2024, December 2). Google Play. https://play.google.com/store/search?q=Veryfi&c=apps

Wang, J. (2023). A study of the OCR Development History and Directions of Development. *Highlights in Science Engineering and Technology*, *72*, 409–415. https://doi.org/10.54097/bm665j77

Yin, C., & Zhang, Z. (2024). A Study of Sentence Similarity Based on the All-minilml6-v2 Model With "Same Semantics, Different Structure" After Fine TuningA Study of Sentence Similarity Based on the All-minilml6-v2 Model With "Same Semantics, Different Structure" After Fine Tuning. *Atlantis Press*. https://doi.org/10.2991/978-94-6463-540-9_69

Appendix A - Schedule

No	Phase	Task	Start Date	End Date	Days
1	Exploration	Supervisor Enquiry and Selection	3/10/2024	4/10/2024	2
		Initial Project Research	5/10/2024	10/10/2024	5
		Topic Selection	11/10/2024	11/10/2024	1
		Topic pitching to supervisor	11/10/2024	11/10/2024	1
		Problem Identification	12/10/2024	14/10/2024	3
		Requirement Gathering	15/10/2024	16/10/2024	2
		Define Requirements, Aim and Objective	17/10/2024	18/10/2024	2
		Define Project Outcome	18/10/2024	19/10/2024	2
		Brief Proposal Amendments	19/10/2024	20/10/2024	2
		Milestone: Brief proposal completion and submission	20/10/2024	20/10/2024	1
2	Planning	Project Background research	21/10/2024	25/10/2024	5
		User and Risk Analysis	26/10/2024	29/10/2024	4
		Define Project Scope	30/10/2024	3/11/2024	5
		Define Project Significance	4/11/2024	7/11/2024	4
		Investigate on suitable Methodology	8/11/2024	13/11/2024	6
		Milestone: Full proposal completion and submission	14/11/2024	14/11/2024	1
		Chapter 1 Preparation and Ammendments	15/11/2024	21/11/2024	7
		Milestone: Chapter 1 completion and submission	21/11/2024	21/11/2024	1
		Research on similar applications	22/11/2024	27/11/2024	6
		earch on similar Application, Framework, API, Database and AI Model a	28/11/2024	3/12/2024	6
		Study potential System Architecture	4/12/2024	6/12/2024	2
		Draft and Design potential System Architecture	7/12/2024	13/12/2024	7
		Milestone: Chapter 2 completion and submission	13/12/2024	13/12/2024	1
		User and System Requirement Analysis	14/12/2024	15/12/2024	2
		Define Functional and Non-Functional Requirement	15/12/2024	16/12/2024	2
		Define software and Hardware Requirements	16/12/2024	17/12/2024	2
		Draft and design on System Diagrams	18/12/2024	24/12/2024	7
		Design User Interface	24/12/2024	4/1/2025	7
		Milestone: Chapter 3 completion and submission	5/1/2025	5/1/2025	1
		FYP 1 Ammendments	6/1/2025	17/1/2025	12
		Milestone: FYP 1 final report & paper for assessment	17/1/2025	17/1/2025	1
		FYP Presentation	24/1/2025	25/1/2025	2
		Milestone: FYP 1 final report after ammendments	16/2/2025	16/2/2025	1
4	Iteration to Release	Software installation and configuration	17/3/2025	17/3/2025	1
		Iteration 1: Basic Functionality Development	18/3/2025	31/3/2025	14
		stone: Submission of Revised FYP Structure, Gantt Chart and Chapter D	1/4/2025	1/4/2025	1
		Iteration 2: Automation using OCR and NLP	1/4/2025	14/4/2025	14
		Iteration 3: RNN LSTM implementation	15/4/2025	28/4/2025	14
		Iteration 4: User interface Improvements	29/4/2025	14/5/2025	16
		Documentation	17/3/2025	14/5/2025	59
		Milestone: Chapter 4 draft submission	15/5/2025	15/5/2025	1
5	Productionizing, Maintenance and Death	Full System Beta Testing	15/5/2025	21/5/2025	7
		Improvement and Bug Fix	15/5/2025	21/5/2025	7
		Full Production System Testing based on test cases	21/5/2025	29/5/2025	9
		User Feedback collection	21/5/2025	29/5/2025	9
		Documentation and hosting	15/5/2025	29/5/2025	15
		Milestone: Chapter 5 and 6 draft submission	31/5/2025	31/5/2025	1
		Full Report Preparation	31/5/2025	9/6/2025	10
		Milestone: 1st draft FYP Report Submisson	10/6/2025	10/6/2025	1
		Amendments FYP draft Report	10/6/2025	23/6/2025	13
		Milestone: Full FYP Report Submission	23/6/2025	23/6/2025	1
		Prototype and demo preparation for symposium	23/6/2025	30/6/2025	8
		Milestone: FYP Symposium	1/7/2025	2/7/2025	2
		Final Date for Examiner to leave comment	4/7/2025	4/7/2025	1
		Amendments FYP Full Report	23/6/2024	28/7/2025	36
		Milestone: Final Submission of FYP Report after amendments	28/7/2025	28/7/2025	1

Appendix B – Use Case Specification

B1: Select Month and Year Home Page Use Case

Use Case:	Select Month and Year Home Page
Short Description:	A PennyLog user who has opened their application can click the drop-down menu in the home page to choose a specific month and year that they would like to preview the home page.
Actors:	User
Pre-conditions:	1. The user has a mobile device. 2. The user is a new user or an existing user 3. The user has an internet connection.
Post-conditions:	The user can preview the spending overview, spending trend graph and recent transaction for selected month and year.
Main Flow of Events:	
1. The use case is initiated when the user opens the application by clicking the application icon in their mobile device. 2. The user now can navigate to the home page by clicking on the home page icon at the bottom navigation panel. 3. In the home page, the user clicks the drop-down menu located at the top of the page. 4. The user can select a particular month and year as listed in the drop-down menu. 5. The page will be refreshed, and the chosen year and month expenses and income logs will be shown. 6. The user can now view the balance, total expenses and total income overview shown for that chosen month and year. 7. The user also has access to the spending trend bar chart that compares the total expenses spent from January to December of that particular year. 8. The user can also view the recent 3 most recently logged transaction. 9. The use case is terminated.	
Alternative Flow(s):	[A1] User Changes Month and Year multiple times. 1. The user clicks the drop-down menu in the home page and select a particular month and year. 2. The application loads and displays the data. 3. After previewing the page, the user clicks the drop-down menu again and chooses a different month and year to compare the logs. 4. The system refreshes the page and displays the selected year and month data each time a new selection is made. 5. The use case is terminated.
Exception Flow(s):	[E1] No available data for the month. 1. The user clicks the drop-down menu in the home page and select a particular month and year where there are no logs recorded.

	2. The home page refreshes 3. The application will display empty values of the home page. 4. The use case is terminated.

B2: Select Month and Year Transaction Page Use Case

Use Case:	Select Month and Year Transaction Page
Short Description:	A PennyLog user who has opened their application can click the drop-down menu in the Transaction page to choose the month and year that they would like to view the filtered transactions.
Actors:	User
Pre-conditions:	1. The user has a mobile device. 2. The user is a new user or an existing user 3. The user has an internet connection.
Post-conditions:	The user can view all the transaction for the chosen month and year filtered based on income and expenses.
Main Flow of Events:	
1. The use case is initiated when the user opens the application by clicking the application icon in their mobile device. 2. The user now can navigate to the Transaction page by clicking on the Transaction page icon at the bottom navigation panel. 3. In the Transaction page, the user clicks the drop-down menu located at the top of the page. 4. The user can select a particular month and year as listed in the drop-down menu. 5. The page will be refreshed, and the chosen year and month transactions will be shown. 6. User can view the transactions based on transaction type by clicking on the toggle at the top. 7. The user can also now choose to search a transaction, view the transaction, edit the transaction, delete the transaction or save the edited transaction. 8. The use case is terminated.	
Alternative Flow(s):	[A1] User Changes Month and Year multiple times. 1. The user clicks the drop-down menu in the Transaction page and selects a particular month and year. 2. The application loads and displays the data. 3. After previewing the transactions, the user clicks the drop-down menu again and chooses a different month and year to compare the logs. 4. The system refreshes the page and displays the selected year and month data each time a new selection is made. 5. The use case is terminated.
Exception Flow(s):	[E1] No available data for the month. 1. The user clicks the drop-down menu on the home page and select a particular month and year where there is Transactions recorded.

	2. The application will display an empty transaction page 3. The use case is terminated.

B3: View Transaction Use Case

Use Case:	View Transaction
Short Description:	A PennyLog user who has opened their application can view transaction by clicking on an individual transaction that is displayed in the transaction page.
Actors:	User
Pre-conditions:	1. The user has a mobile device. 2. The user is a new user or an existing user 3. The user has an internet connection.
Post-conditions:	The user can view selected transaction in details as a pop up with an option to edit or delete the transaction or to navigate back to the transaction page.
Main Flow of Events:	
1. The use case is initiated when the user opens the application by clicking the application icon in their mobile device. 2. The user now can navigate to the Transaction page by clicking on the Transaction page icon at the bottom navigation panel. 3. In the Transaction page, the user can choose to click one of the transactions displayed in the income toggle or the expense toggle. 4. Once the user clicks on the transaction, the system will load and display all the information regarding the transaction chosen such as transaction type, transaction method, date, time, total, category, status, remark, and image in a pop up. 5. The use case is terminated.	
Alternative Flow(s):	[A1] User views a recently added transaction 1. After the user chooses to add a new transaction, the system updates the transaction page to include the newly added transaction. 2. The user then navigates to the transaction page and clicks on the newly added transaction to view the details that has just been added. 3. The system loads and displays the selected transaction details. 4. The use case is terminated.
Exception Flow(s):	[E1] No transaction available 1. The user navigated to the transaction page and selected the month and year that they would like to view the transactions 2. Empty transaction page displayed. 3. The use case is terminated.

B4: Manage Transaction Use Case

Use Case:	Manage Transaction

Short Description:	A user who has navigated to the transaction page will be able to search a transaction, edit transaction, delete transaction and save transaction.
Actors:	User
Pre-conditions:	1. The user has a mobile device. 2. The user is a new user or an existing user 3. The user has an internet connection.
Post-conditions:	1. The user can search for transactions. 2. The user can edit existing transactions. 3. The user can delete existing transactions. 4. The user can save the edited transactions.
Main Flow of Events:	
1. The use case is initiated when the user opens the application by clicking the application icon in their mobile device. 2. The user now can navigate to the transaction page by clicking on the transaction page icon at the bottom navigation panel. 3. The user can search a transaction by typing in keyword like date, remark or category to filter the transaction. 4. The user type in "Food/Grocery" and the system filters all category under it. 5. The user chooses to open one of the transactions displayed and the application opens the transaction and show all the details in a pop up. 6. The user edits the transaction by changing the category to "Others". 7. The user chooses to save the transactions. 8. The application saves the changes and displays the new changes in the transaction page. 9. The user clicks again on the edited transaction and the applications open the transaction in a pop up to show the changed details. 10. The user chooses to delete the transaction by clicking on the delete button. 11. The system will pop up a prompt to validate whether user wishes to continue their actions by clicking yes or no. 12. If yes, the system will save the new information or action taken, if not the system will terminate the session, and no changes will be made. 13. The use case is terminated.	
Alternative Flow(s):	[A1] User cancel a deletion 1. The user selected a transaction and chose to delete the transaction by clicking the delete button. 2. The system displays a confirmation message to delete the transaction. 3. The user realise it's a mistake and click no. 4. The confirmation disappeared and the transaction remains in the application transaction page. 5. The use case is terminated.
Exception Flow(s):	[E1] Missing data during editing transaction 1. The user attempts to save an edited transaction with an empty date field that is required.

	2. The system will display a message when the user tries to save the transaction which is “Please select a date” at the missing field. 3. The user then corrects the information and tries to save again or cancel. 4. The use case is terminated.

B5: Add New Transaction Use Case

Use Case:	Add New Transaction
Short Description:	A PennyLog user has opened the application and will be able to add new transactions for both expenses and income using manual method or through receipt scanning. The new transaction will be saved into the applications database and be displayed in the application.
Actors:	User
Pre-conditions:	1. The user has a mobile device. 2. The user is a new user or an existing user 3. The user has an internet connection.
Post-conditions:	The system displays the new transactions in the recent transaction and in the transaction page. The totals in the home page and the spending trend graph will also be updated.
Main Flow of Events:	
1. The use case is initiated when the user opens the application by clicking the application icon in their mobile device. 2. The user now can add a new transaction by clicking the add icon button at the bottom navigation bar. 3. An interface pop up prompting user to choose a transaction type such as expenses or income. 4. User chooses and the transaction form for user to fill in the transaction details. 5. User can open camera to capture image or choose image from gallery to automate the transaction process where user can crop the image, enhance image, perform OCR and use the extracted information in the fields. 6. The user can then choose to save the transaction or cancel the transaction process. 7. The use case is terminated.	
Alternative Flow(s):	[A1] User cancel adding a transaction 1. The user clicks the cancel button or closes the application after filling in the transaction form. 2. The application will discard all the changes made or any unsaved data. 3. The use case is terminated.
Exception Flow(s):	[E1] Application crash during transaction creation 1. The user clicks the add button icon to add new transaction.

	2. The application crashes while the user is entering or trying to save the transaction. 3. The system displays a message which is “Application has stopped working, please wait or restart the application”. 4. The user can either wait or restart the application to add a new transaction. 5. The user tries to restart the application and proceed to repeat the adding new transaction process. 6. The use case is terminated.

B6: Save Transaction Use Case

Use Case:	Save Transaction
Short Description:	A PennyLog user will be able to save a new transaction or edited transaction.
Actors:	User
Pre-conditions:	1. The user has a mobile device. 2. The user is a new user or an existing user 3. The user has an internet connection.
Post-conditions:	The application will be updated with the new transaction at the transaction page and the recent transaction list. The totals and the graphs will also be updated.
Main Flow of Events:	
1. The use case is initiated when the user opens the application by clicking the application icon in their mobile device. 2. The user now can add a new transaction by clicking the add icon button at the bottom navigation bar. 3. An interface pops up prompting the user to choose a transaction type. 4. After choosing and the information has been displayed in their respective field, the user can fill in the transaction details. 5. After filling in all the necessary field, the user clicks on the save button. 6. A pop up appears to validate if user would like to proceed with their action. 7. Users click yes. 8. The application reloads and the new transaction are added to the application. 9. The use case is terminated.	
Alternative Flow(s):	N/A
Exception Flow(s):	[E1] Application crash when saving transaction 1. The user attempt to save an edited transaction. 2. The user attempt to save the transaction by clicking on the save icon button. 3. The application crashes when attempting to save the transaction. 4. The system displays a message which is “Application has stopped working, please wait or restart the application”.

	5. The user can either wait or restart the application to edit again the transaction and save it 6. The user tries to restart the application and proceed to repeat the same process. 7. The use case is terminated.

B7: Open Insight Page Use Case

Use Case:	Open Insight Page
Short Description:	A New or existing PennyLog user can open the application Insight page by launching the application and select insight page in their mobile device.
Actors:	User
Pre-conditions:	1. The user has a mobile device. 2. The user is a new user or an existing user 3. The user has an internet connection.
Post-conditions:	The user can open the insight page and view the prediction for the following month successfully.
Main Flow of Events:	
1. The use case is initiated when the user opens the application by clicking the application icon in their mobile device. 2. The user now can navigate to the insight page by clicking on the insight page icon at the bottom navigation panel. 3. The application refreshes and display the insight page with the current month prediction in pie chart form and tabulated in table. 4. The use case is terminated.	
Alternative Flow(s):	[A1] User Open application in offline mode. 1. The user opens the application insight page without an active internet connection. 2. The application won't be able to open without an internet connection and crashes with a system message indicating the application failed to launch. 3. The use case is terminated.
Exception Flow(s):	[E1] User did not successfully open the insight page. 1. The application displays a page with no value of the current month insight. 2. The use case is terminated.

B8: Manage Insight Page Use Case

Use Case:	Manage Insight Page
Short Description:	A PennyLog user can filter and choose a month in the current year to view the prediction, insert a budget to create a new prediction and to view the original prediction.
Actors:	User
Pre-conditions:	1. The user has a mobile device. 2. The user is a new user or an existing user 3. The user has an internet connection.

Post-conditions:	1. User can filter and view a particular month in 2025 predictions 2. Users insert a budget, and the application generate a optimised budget prediction. 3. Users can view back the original prediction for the month.
Main Flow of Events:	
1. The use case is initiated when the user opens the application by clicking the application icon in their mobile device. 2. The user now can navigate to the insight page by clicking on the insight page icon at the bottom navigation panel. 3. The page reloads and displays current month predictions in pie chart and table format. 4. User can change to view a different month's predictions by clicking on the month drop down at the top. 5. The page reloads and shows the months prediction. 6. Users can insert a budget into the budget field and click "Update Budget" button. 7. The application reloads and displays an optimised budget prediction. 8. Users can click the "View Original" button to view the original budget for the month. 9. The application clears the budget and displays the original prediction. 10. The use case is terminated.	
Alternative Flow(s):	N/A
Exception Flow(s):	[E1] User insert an invalid format of budget 1. The user attempts to insert a budget with special character and click update budget. 2. The system displays an error message which is "Please input a valid budget amount" at the field. 3. The use case is terminated.

B9: Select Theme Use Case

Use Case:	Select Theme
Short Description:	A PennyLog user who has opened their application and navigated to the settings page can click the select theme menu and select the theme that they would like PennyLog to use.
Actors:	User
Pre-conditions:	1. The user has a mobile device. 2. The user is a new user or an existing user 3. The user has an internet connection.
Post-conditions:	The user can select a dark theme or a light theme.
Main Flow of Events:	
1. The use case is initiated when the user opens the application by clicking the application icon in their mobile device.	

2. The user now can navigate to the settings page by clicking on the settings page icon at the bottom navigation panel. 3. In the setting page, the user clicks the Select Theme menu. 4. Users can choose to select a new theme or stick with the current default theme. 5. The application will load and now use the new theme that is selected. 6. The use case is terminated.	
Alternative Flow(s):	[A1] User does not change a new theme. 1. The user navigates to the setting page and clicks the Select Theme menu. 2. The user previews the available theme but does not change to a new theme. 3. The user kept the current setting for the currency as default. 4. No changes made to the application. 5. The use case is terminated.
Exception Flow(s):	[E1] Theme list fails to load 1. The user navigates to the setting page and clicks the Select Theme. 2. The list of the available theme fails to load. 3. The system displays an error message “Unable to load theme options. Please check your internet connection and try again”. 4. The user can reconnect to the internet and relaunch the application. 5. The user repeats the steps. 6. The use case is terminated.

B10: Manage Currency Use Case

Use Case:	Manage Currency
Short Description:	A PennyLog user who has opened their application and navigated to the settings page can click the currency selection menu to search or select a new currency.
Actors:	User
Pre-conditions:	1. The user has a mobile device. 2. The user is a new user or an existing user 3. The user has an internet connection.
Post-conditions:	The user can search a particular currency or apply a new currency to the application.
Main Flow of Events:	
1. The use case is initiated when the user opens the application by clicking the application icon in their mobile device. 2. The user now can navigate to the settings page by clicking on the settings page icon at the bottom navigation panel. 3. In the setting page, the user clicks the currency selector. 4. The user clicks on the select new currency button. 5. User search for a particular currency by using the keyword.	

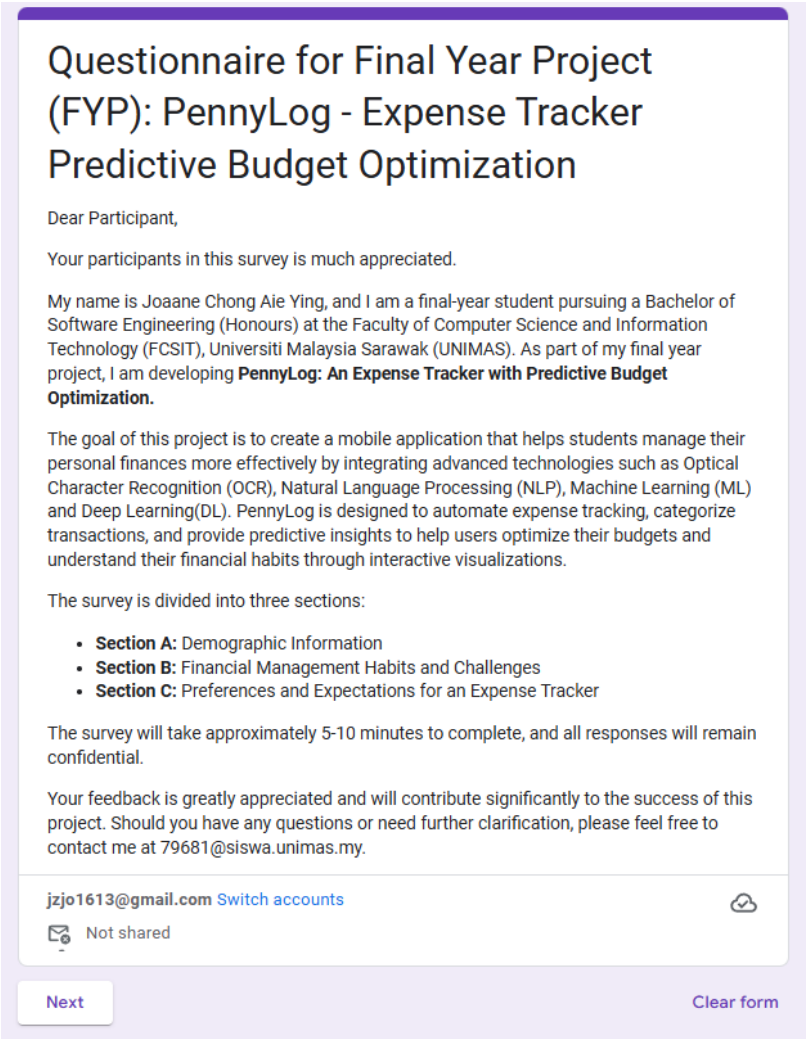
6. Users can then choose to select a new currency or stick with the current default currency. 7. The application will load and now use the new currency that is selected. 8. The use case is terminated.	
Alternative Flow(s):	[A1] User does not change a new currency. 1. The user navigates to the setting page and selects the currency selector. 2. The user previews the available currency but does not change to a new currency. 3. The user kept the current setting for the currency as default. 4. The use case is terminated.
Exception Flow(s):	[E1] Currency list fails to load 1. The user navigates to the setting page and selects the currency selector. 2. The list of the available currency fails to load. 3. The system displays an error message “Unable to load currency options. Please check your internet connection and try again”. 4. The user can reconnect to the internet and relaunch the application. 5. The use case is terminated.

B11: Manage Language Use Case

Use Case:	Manage Language
Short Description:	A PennyLog user who has opened their application and navigated to the settings page and click the language selector menu where user can search and select a new language to be applied in the application.
Actors:	User
Pre-conditions:	1. The user has a mobile device. 2. The user is a new user or an existing user 3. The user has an internet connection.
Post-conditions:	The user can search and select a new language to be used in the application.
Main Flow of Events:	
1. The use case is initiated when the user opens the application by clicking the application icon in their mobile device. 2. The user now can navigate to the settings page by clicking on the settings page icon at the bottom navigation panel. 3. In the setting page, the user clicks the language selector menu. 4. The user clicks on the select new currency button. 5. User can now search for a new language in the list by typing the keyword. 6. Users can then select a language that they would like the application to use or remain the default English language. 7. The use case is terminated.	

Alternative Flow(s):	[A1] User didn't select a new language 1. The user opens the language selector menu to choose a new language to be used. 2. However, the user changes their mind and doesn't change the default settings. 3. The application continues to use the default settings which is the English language. 4. The use case is terminated.
Exception Flow(s):	[E1] Language list fails to load 1. The user navigates to the setting page and clicks the language selector menu. 2. The list of the available language fails to load. 3. The system displays an error message "Unable to load language options. Please check your internet connection and try again". 4. The user can reconnect to the internet and relaunch the application. 5. The user repeats the steps. 6. The use case is terminated.

Appendix C – Requirement Gathering Form



Questionnaire for Final Year Project (FYP): PennyLog - Expense Tracker Predictive Budget Optimization

Dear Participant,

Your participants in this survey is much appreciated.

My name is Joaane Chong Aie Ying, and I am a final-year student pursuing a Bachelor of Software Engineering (Honours) at the Faculty of Computer Science and Information Technology (FCSIT), Universiti Malaysia Sarawak (UNIMAS). As part of my final year project, I am developing **PennyLog: An Expense Tracker with Predictive Budget Optimization.**

The goal of this project is to create a mobile application that helps students manage their personal finances more effectively by integrating advanced technologies such as Optical Character Recognition (OCR), Natural Language Processing (NLP), Machine Learning (ML) and Deep Learning(DL). PennyLog is designed to automate expense tracking, categorize transactions, and provide predictive insights to help users optimize their budgets and understand their financial habits through interactive visualizations.

The survey is divided into three sections:

- **Section A:** Demographic Information
- **Section B:** Financial Management Habits and Challenges
- **Section C:** Preferences and Expectations for an Expense Tracker

The survey will take approximately 5-10 minutes to complete, and all responses will remain confidential.

Your feedback is greatly appreciated and will contribute significantly to the success of this project. Should you have any questions or need further clarification, please feel free to contact me at 79681@siswa.unimas.my.

jzjo1613@gmail.com Switch accounts

Not shared

Next

Clear form

Questionnaire for Final Year Project (FYP): PennyLog - Expense Tracker Predictive Budget Optimization

jzjo1613@gmail.com Switch accounts

Not shared

* Indicates required question

Section A: Demographic Information

This section aims to gather basic demographic details to gain insights into our respondents. Your answers will help ensure that the data collected is representative of the university community. Kindly provide information about your gender, age, year of study, and faculty. Participation is entirely voluntary, and all responses will be kept strictly confidential.

Gender *

- Female
- Male

Age *

- Below 18
- 18–20
- 21–23
- 24–26
- 27 and above

Year of Study *

- Pre-university (Foundation/Asasi)
- Year 1
- Year 2
- Year 3
- Year 4
- Post-Graduate (Masters or PHD)

Are you a student in UNIMAS? *

- Yes
- No

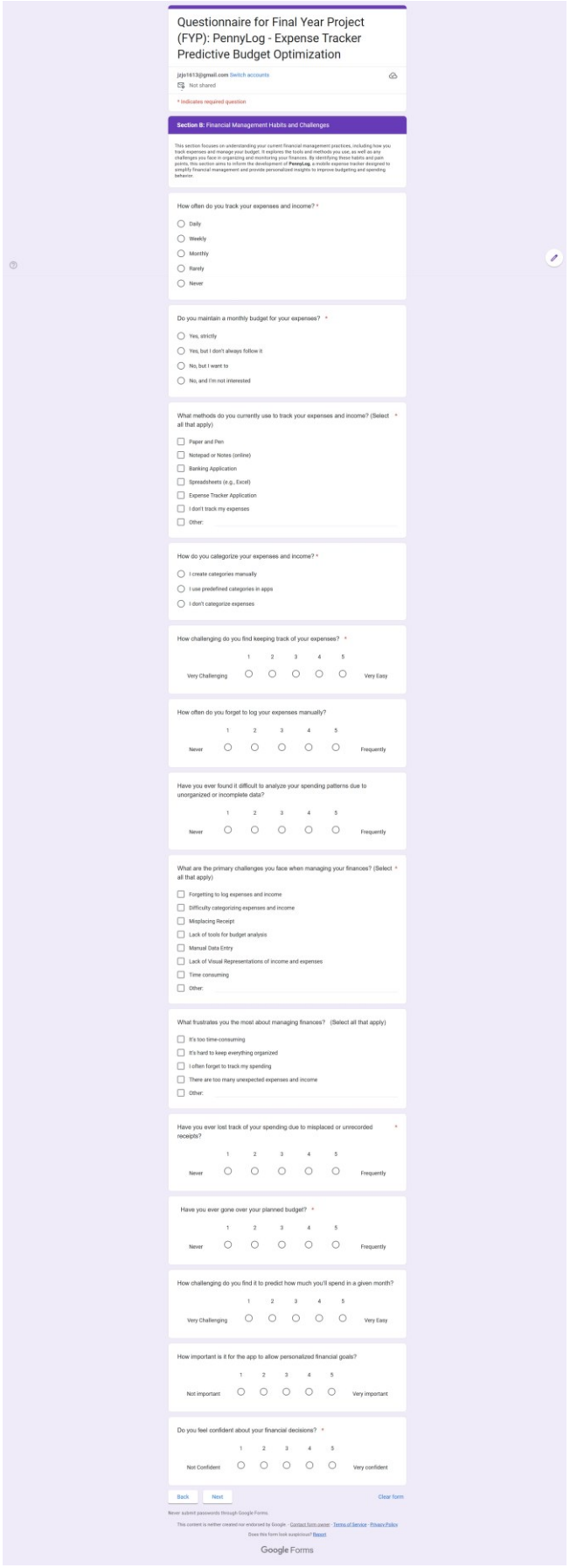
Which University Are You From?

Your answer

Faculty *

- Centre For Pre-University Studies (PPPU)
- Faculty of Computer Science and Information Technology (FCSIT)
- Faculty of Applied and Creative Arts (FACA)
- Faculty of Cognitive Sciences and Human Development (FCSHD)
- Faculty of Built Environment (FBE)
- Faculty of Economics and Business (FEB)
- Faculty of Medicine and Health Sciences (FMHS)
- Faculty of Education, Language and Communication (FELC)
- Faculty of Engineering (FENG)
- Faculty of Resource Science and Technology (FRST)
- Faculty of Social Sciences and Humanities (FSSH)
- Others

Back Next Clear form



Questionnaire for Final Year Project (FYP): PennyLog - Expense Tracker Predictive Budget Optimization

jqjo1613@gmail.com Switch accounts

Not shared

* Indicates required question

Section B: Financial Management Habits and Challenges

This section focuses on understanding your current financial management practices, including how you track expenses and manage your budget. It explores the tools and methods you use, as well as any challenges you face in organizing and monitoring your finances. By identifying these habits and pain points, this section aims to inform the development of **PennyLog**, a mobile expense tracker designed to simplify financial management and provide personalized insights to improve budgeting and spending behavior.

How often do you track your expenses and income? *

- Daily
- Weekly
- Monthly
- Rarely
- Never

Do you maintain a monthly budget for your expenses? *

- Yes, strictly
- Yes, but I don't always follow it
- No, but I want to
- No, and I'm not interested

What methods do you currently use to track your expenses and income? (Select all that apply) *

- Paper and Pen
- Notepad or Notes (online)
- Banking Application
- Spreadsheets (e.g., Excel)
- Expense Tracker Application
- I don't track my expenses
- Other:

How do you categorize your expenses and income? *

- I create categories manually
- I use predefined categories in apps
- I don't categorize expenses

How challenging do you find keeping track of your expenses? *

	1	2	3	4	5	
Very Challenging	○	○	○	○	○	Very Easy

How often do you forget to log your expenses manually?

	1	2	3	4	5	
Never	○	○	○	○	○	Frequently

Have you ever found it difficult to analyze your spending patterns due to unorganized or incomplete data?

	1	2	3	4	5	
Never	○	○	○	○	○	Frequently

What are the primary challenges you face when managing your finances? (Select all that apply) *

- Forgetting to log expenses and income
- Difficulty categorizing expenses and income
- Misplacing Receipt
- Lack of tools for budget analysis
- Manual Data Entry
- Lack of Visual Representations of income and expenses
- Time consuming
- Other:

What frustrates you the most about managing finances? (Select all that apply)

- It's too time-consuming
- It's hard to keep everything organized
- I often forget to track my spending
- There are too many unexpected expenses and income
- Other:

Have you ever lost track of your spending due to misplaced or unrecorded receipts? *

	1	2	3	4	5	
Never	○	○	○	○	○	Frequently

Have you ever gone over your planned budget? *

	1	2	3	4	5	
Never	○	○	○	○	○	Frequently

How challenging do you find it to predict how much you'll spend in a given month?

	1	2	3	4	5	
Very Challenging	○	○	○	○	○	Very Easy

How important is it for the app to allow personalized financial goals?

	1	2	3	4	5	
Not important	○	○	○	○	○	Very important

Do you feel confident about your financial decisions? *

	1	2	3	4	5	
Not Confident	○	○	○	○	○	Very confident

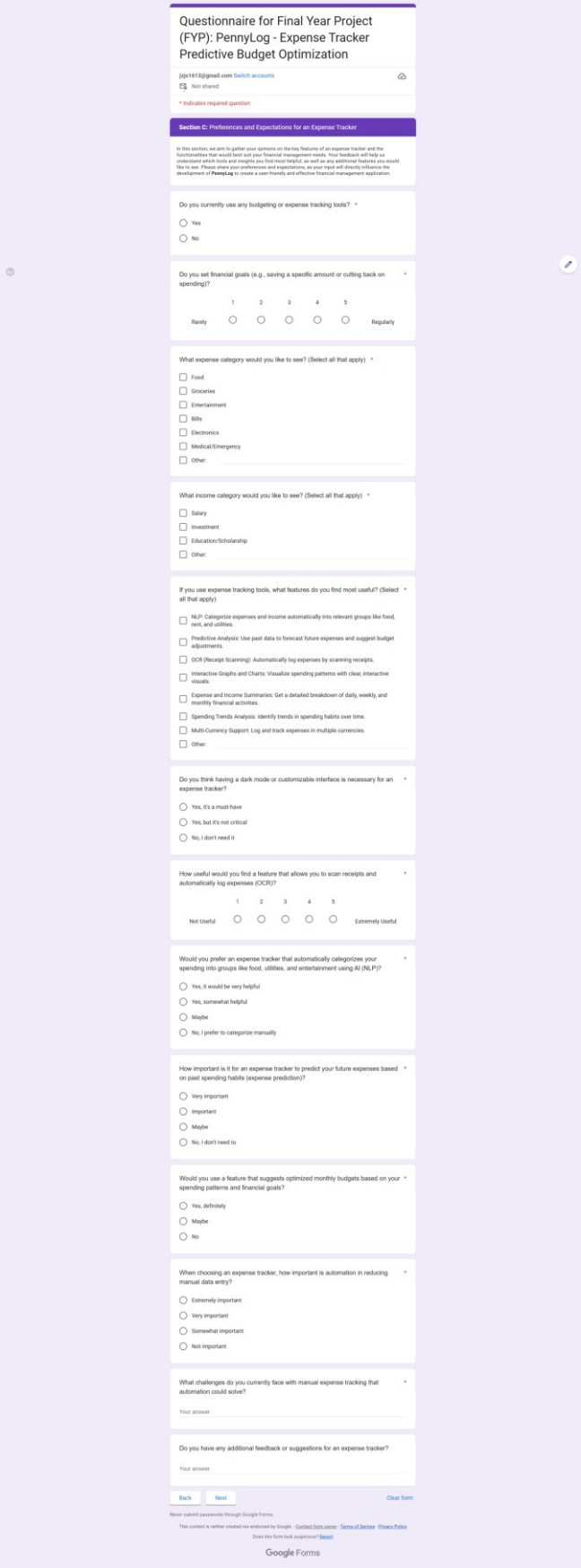
Back Next Clear form

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. - Contact form owner - Terms of Service - Privacy Policy

Does this form look suspicious? Report

Google Forms



Questionnaire for Final Year Project (FYP): PennyLog - Expense Tracker Predictive Budget Optimization

jzjo1613@gmail.com Switch accounts

Not shared

* Indicates required question

Section C: Preferences and Expectations for an Expense Tracker

In this section, we aim to gather your opinions on the key features of an expense tracker and the functionalities that would best suit your financial management needs. Your feedback will help us understand which tools and insights you find most helpful, as well as any additional features you would like to see. Please share your preferences and expectations, as your input will directly influence the development of **PennyLog** to create a user-friendly and effective financial management application.

Do you currently use any budgeting or expense tracking tools? *

- Yes
- No

Do you set financial goals (e.g., saving a specific amount or cutting back on spending)? *

	1	2	3	4	5	
Rarely	○	○	○	○	○	Regularly

What expense category would you like to see? (Select all that apply) *

- [ ] Food
- [ ] Groceries
- [ ] Entertainment
- [ ] Bills
- [ ] Electronics
- [ ] Medical/Emergency
- [ ] Other:

What income category would you like to see? (Select all that apply) *

- [ ] Salary
- [ ] Investment
- [ ] Education/Scholarship
- [ ] Other:

If you use expense tracking tools, what features do you find most useful? (Select all that apply) *

- [ ] NLP: Categorize expenses and income automatically into relevant groups like food, rent, and utilities.
- [ ] Predictive Analysis: Use past data to forecast future expenses and suggest budget adjustments.
- [ ] OCR (Receipt Scanning): Automatically log expenses by scanning receipts.
- [ ] Interactive Graphs and Charts: Visualize spending patterns with clear, interactive visuals.
- [ ] Expense and Income Summaries: Get a detailed breakdown of daily, weekly, and monthly financial activities.
- [ ] Spending Trends Analysis: Identify trends in spending habits over time.
- [ ] Multi-Currency Support: Log and track expenses in multiple currencies.
- [ ] Other:

Do you think having a dark mode or customizable interface is necessary for an expense tracker? *

- Yes, it's a must-have
- Yes, but it's not critical
- No, I don't need it

How useful would you find a feature that allows you to scan receipts and automatically log expenses (OCR)? *

	1	2	3	4	5	
Not Useful	○	○	○	○	○	Extremely Useful

Would you prefer an expense tracker that automatically categorizes your spending into groups like food, utilities, and entertainment using AI (NLP)? *

- Yes, it would be very helpful
- Yes, somewhat helpful
- Maybe
- No, I prefer to categorize manually

How important is it for an expense tracker to predict your future expenses based on past spending habits (expense prediction)? *

- Very important
- Important
- Maybe
- No, I don't need to

Would you use a feature that suggests optimized monthly budgets based on your spending patterns and financial goals? *

- Yes, definitely
- Maybe
- No

When choosing an expense tracker, how important is automation in reducing manual data entry? *

- Extremely important
- Very important
- Somewhat important
- Not important

What challenges do you currently face with manual expense tracking that automation could solve? *

Your answer

Do you have any additional feedback or suggestions for an expense tracker?

Your answer

Back Next Clear form

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. - Contact form owner - Terms of Service - Privacy Policy

Does this form look suspicious? Report

Google Forms

Appendix D – User Acceptance Test Form

PennyLog App User Experience Feedback

Greetings, My name is Joaane Chong Aie Ying, a final year Software Engineering Student in Universiti Malaysia Sarawak (UNIMAS) at the Faculty of Computer Science & Information Technology (FCSIT).

For my Final Year Project, I developed PennyLog, a mobile based finance management application which are enhanced with Artificial Intelligence (AI) features.

Overview of Project:
PennyLog is a cross-platform mobile application developed using Flutter and FastAPI, designed to help users track, analyze, and optimize their personal finances. The home page presents an intuitive overview of spending, including a yearly trend graph, recent transactions, and filtering by month and year. Users can view transactions by type (income or expense), with full functionality to edit, delete, and preview associated images with zoom and reset controls. For adding new transactions, users can either input details manually or utilize image-based OCR (Optical Character Recognition) powered by OpenCV, which includes image cropping and automatic enhancement features to improve text extraction. Extracted data can then autofill the transaction form to save time. One of the core features of PennyLog is a time series forecasting model in the Insight page that predicts future expenses based on historical transaction data. These predictions are presented in a structured table format, offering users clear visibility into upcoming financial trends. Complementing this, the app includes a budget optimization tool that allows users to set a monthly budget, which is then recalculated dynamically to show how spending aligns with the set limit. Personalization settings allow users to switch between light/dark themes, choose preferred currencies, and select application language.
This survey is structured into **6 short sections**, beginning with **User Demographics**, followed by focused evaluations on each key module of the PennyLog app:

1. **User Demographics**
2. **Home Module**
3. **Transaction Module**
4. **Add New Transaction Module**
5. **Insight Module**
6. **Settings Module**

Your participation will help assess the usability, accuracy, and overall user experience of PennyLog.
The survey takes approximately **10-15 minutes** to complete. Your honest feedback is crucial in refining the application for future users.

If you have any questions, feel free to reach out at **79681@siswa.unimas.my**

Thank you so much for your time and input!

jzjo1613@gmail.com Switch accounts

Not shared

Scan The QR Code to Download The APK File for PennyLog

PennyLog

Next

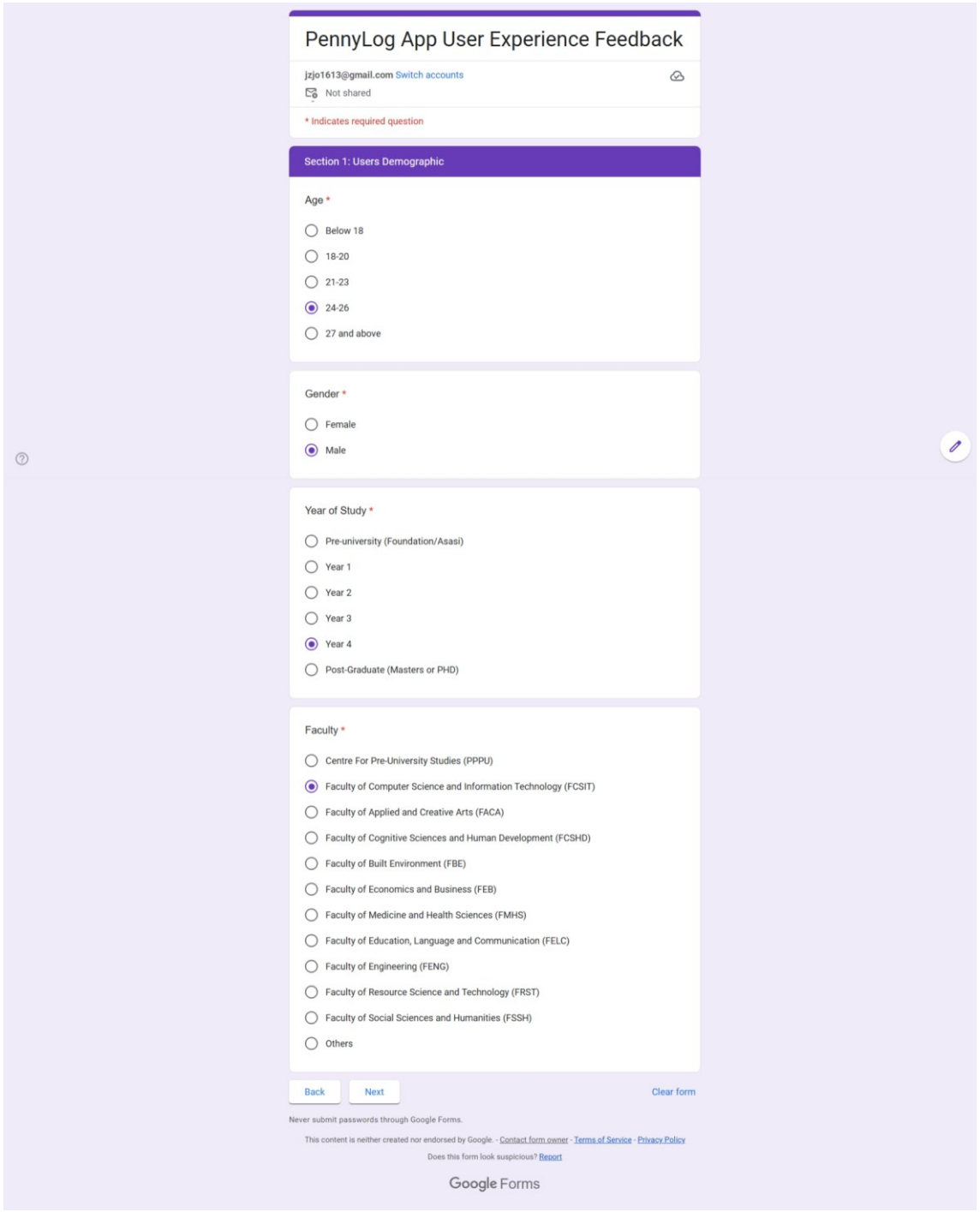
Clear form

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. - Contact form owner - Terms of Service - Privacy Policy

Does this form look suspicious? Report

Google Forms



PennyLog App User Experience Feedback

jzjo1613@gmail.com Switch accounts

Not shared

* Indicates required question

Section 1: Users Demographic

Age *

- Below 18
- 18-20
- 21-23
- 24-26
- 27 and above

Gender *

- Female
- Male

Year of Study *

- Pre-university (Foundation/Asasi)
- Year 1
- Year 2
- Year 3
- Year 4
- Post-Graduate (Masters or PHD)

Faculty *

- Centre For Pre-University Studies (PPPU)
- Faculty of Computer Science and Information Technology (FCSIT)
- Faculty of Applied and Creative Arts (FACA)
- Faculty of Cognitive Sciences and Human Development (FCSHD)
- Faculty of Built Environment (FBE)
- Faculty of Economics and Business (FEB)
- Faculty of Medicine and Health Sciences (FMHS)
- Faculty of Education, Language and Communication (FELC)
- Faculty of Engineering (FENG)
- Faculty of Resource Science and Technology (FRST)
- Faculty of Social Sciences and Humanities (FSSH)
- Others

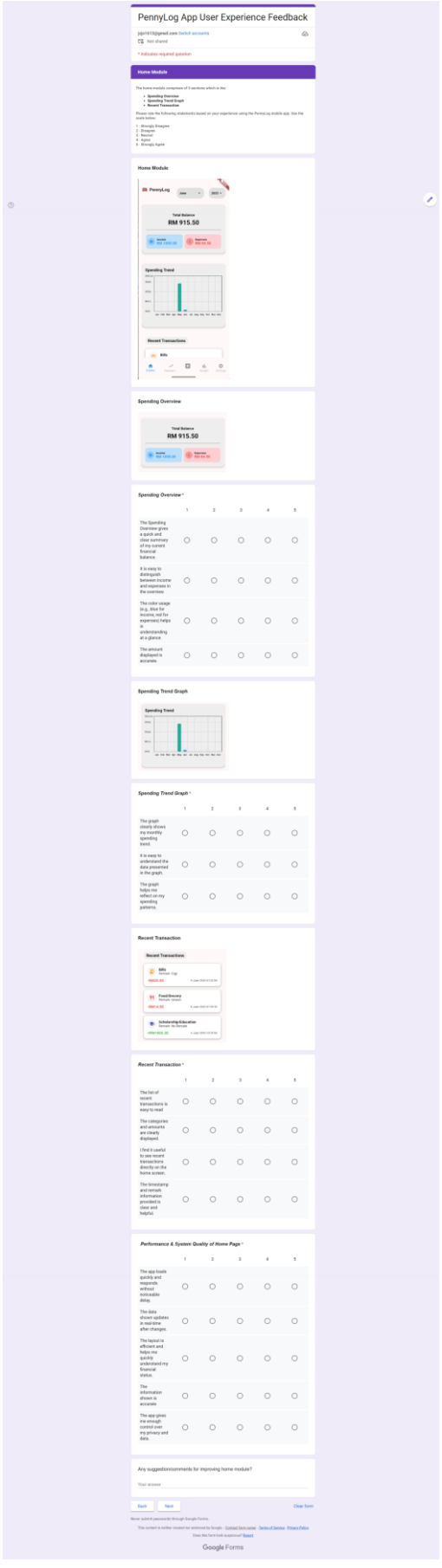
Back Next Clear form

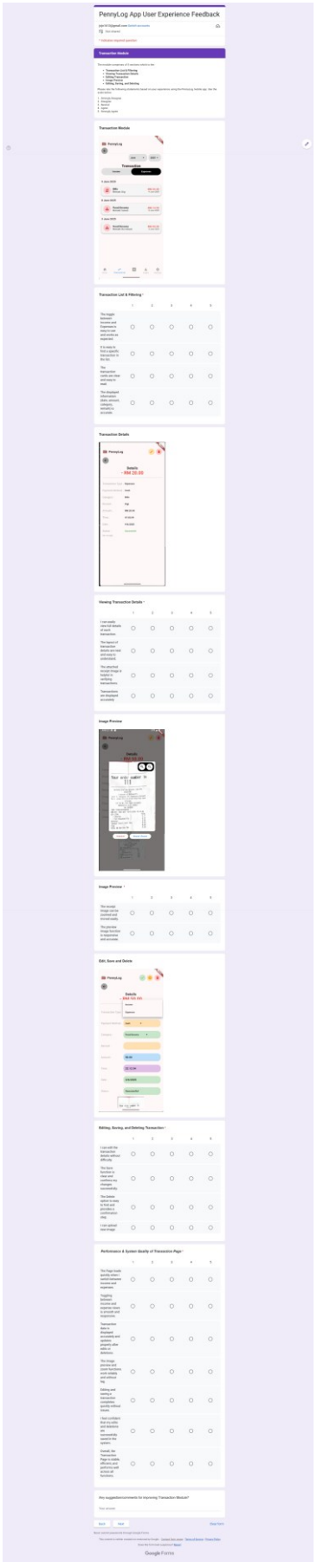
Never submit passwords through Google Forms.

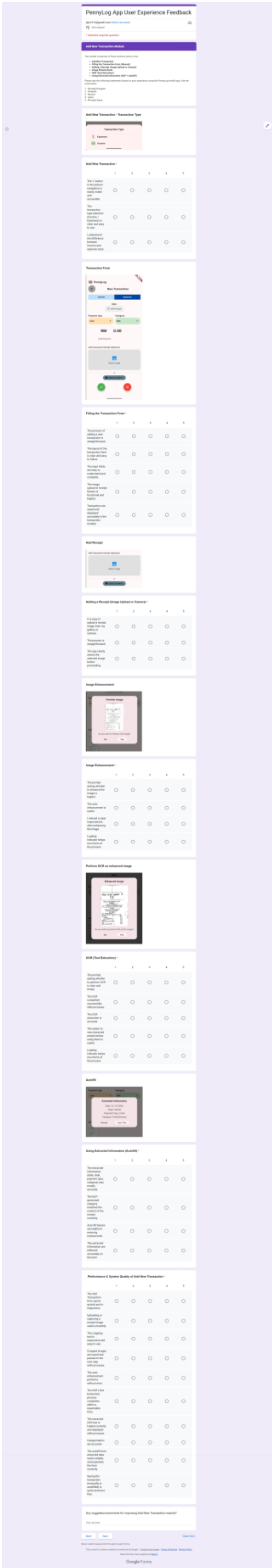
This content is neither created nor endorsed by Google. - Contact form owner - Terms of Service - Privacy Policy

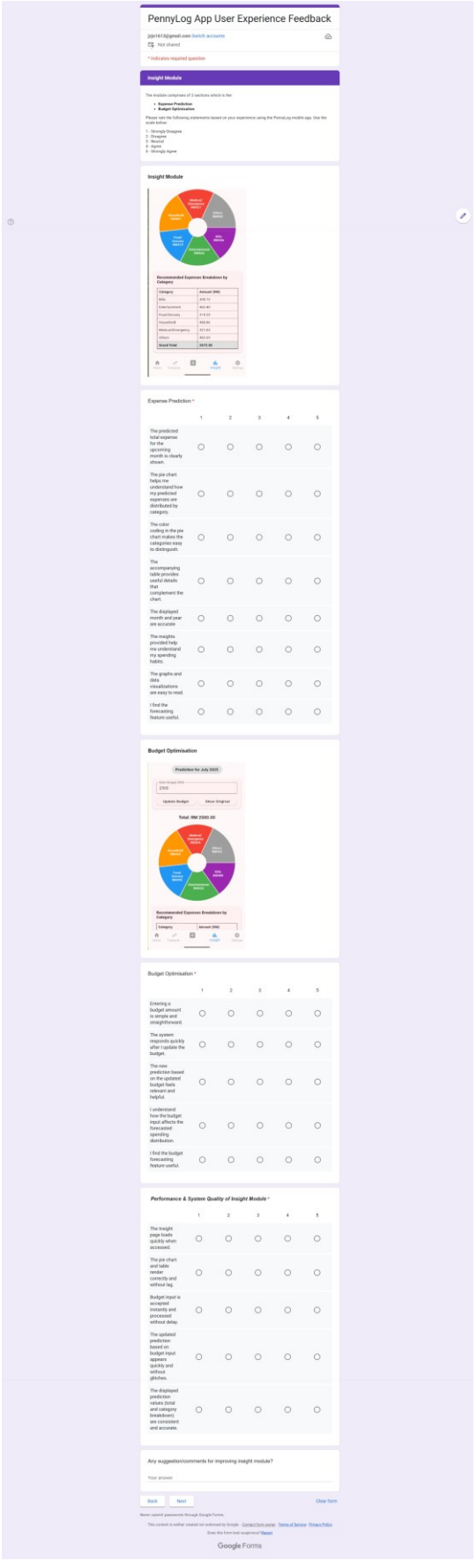
Does this form look suspicious? Report

Google Forms


PennyLog App User Experience Feedback
Not shared
* Indicates required question
Home Module
The home module comprises of 3 sections which is the:
Spending Overview
Spending Trend Graph
Recent Transaction
Please rate the following statements based on your experience using the PennyLog mobile app. Use the scale below:
1 - Strongly Disagree
2 - Disagree
3 - Neutral
4 - Agree
5 - Strongly Agree
Home Module
PennyLog
Total Balance
RM 915.50
Spending Trend
Recent Transactions
Bills
Spending Overview
Total Balance
RM 915.50
Spending Overview *
1
2
3
4
5
The Spending Overview gives a quick and clear summary of my overall financial balance.
It is easy to distinguish between income and expenses in the overview.
The color usage (e.g., blue for income, red for expenses) helps in understanding at a glance.
The amount displayed is accurate.
Spending Trend Graph
Spending Trend
Spending Trend Graph *
1
2
3
4
5
The graph clearly shows my monthly spending trend.
It is easy to understand the data presented in the graph.
The graph helps me reflect on my spending patterns.
Recent Transaction
Recent Transactions
Bills
Food/Grocery
Scholarship/Education
Recent Transaction *
1
2
3
4
5
The list of recent transactions is easy to read.
The categories and amounts are clearly displayed.
I find it useful to see recent transactions directly on the home screen.
The timestamp and remark information provided is clear and helpful.
Performance & System Quality of Home Page *
1
2
3
4
5
The app loads quickly and responds without noticeable delay.
The data shown updates in real-time after changes.
The layout is consistent and helps me quickly understand my financial status.
The information shown is accurate
The app gives me enough control over my privacy and data.
Any suggestion/comments for improving home module?
Your answer
Back
Next
Clear form
Google Forms


PennyLog App User Experience Feedback
Transaction Module
Transaction List & Filtering
Transaction Details
Viewing Transaction Details
Image Preview
Image Preview
Edit, Save and Delete
Editing, Saving, and Deleting Transactions
Performance & System Quality of Transaction Page
Any suggestion/comments for improving Transaction Module?
Google Forms


PennyLog App User Experience Feedback


PennyLog App User Experience Feedback

jipr163@gmail.com Switch accounts

Not shared

* Indicates required question

Insight Module

The module comprises of 2 sections which is the:

- Expense Prediction
- Budget Optimisation

Please rate the following statements based on your experience using the PennyLog mobile app. Use the scale below:

1 - Strongly Disagree
2 - Disagree
3 - Neutral
4 - Agree
5 - Strongly Agree

Insight Module

Recommended Expenses Breakdown by Category

Category	Amount (RM)
Bills	635.74
Entertainment	462.40
Food/Groceries	618.25
Household	459.85
Medical/Emergency	421.83
Others	452.83
Grand Total	3472.90

Expense Prediction *

	1	2	3	4	5
The predicted total expense for the upcoming month is clearly shown.					
The pie chart helps me understand how my predicted expenses are distributed by category.					
The color coding in the pie chart makes the categories easy to distinguish.					
The accompanying table provides useful details that complement the chart.					
The displayed month and year are accurate					
The insights provided help me understand my spending habits.					
The graphs and data visualizations are easy to read.					
I find the forecasting feature useful.					

Budget Optimisation

Prediction for July 2025

2500

Update Budget | Show Original

Total: RM 2500.00

Recommended Expenses Breakdown by Category

Category	Amount (RM)

Budget Optimisation *

	1	2	3	4	5
Entering a budget amount is simple and straightforward.					
The system responds quickly after I update the budget.					
The new prediction based on the updated budget feels relevant and helpful.					
I understand how the budget input affects the forecasted spending distribution.					
I find the budget forecasting feature useful.					

Performance & System Quality of Insight Module *

	1	2	3	4	5
The insight page loads quickly when accessed.					
The pie chart and table render correctly and without lag.					
Budget input is accepted instantly and processed without delay.					
The updated prediction based on budget input appears quickly and without glitches.					
The displayed prediction values (total and category breakdown) are consistent and accurate.					

Any suggestion/comments for improving insight module?

Your answer

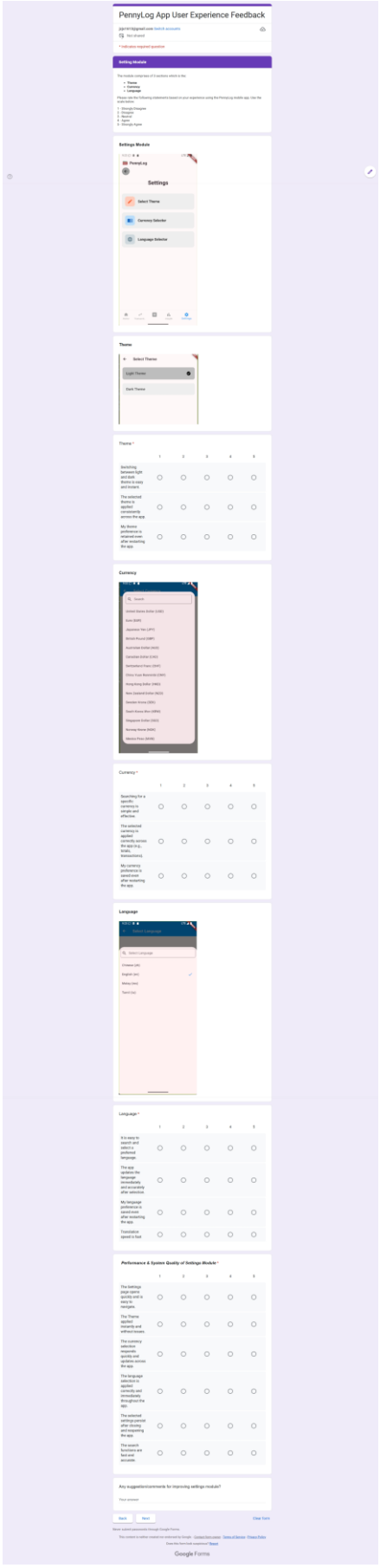
Back | Next | Clear form

Never submit passwords through Google Forms.

This content is neither created nor endorsed by Google. - Contact form owner - Terms of Service - Privacy Policy

Does this form look suspicious? Report

Google Forms


PennyLog App User Experience Feedback

Settings Module

Settings Module

Theme

Theme *

Currency

Currency *

Language

Language *

Performance & System Quality of Settings Module *

Any suggestion/comments for improving settings module?

Google Forms