IMAGE RECOGNITION EXPENSE EXTRACTION

BY

KOK WEI JIN

A REPORT SUBMITTED TO
Universiti Tunku Abdul Rahman
in partial fulfilment of the requirements
for the degree of
BACHELOR OF COMPUTER SCIENCE
(HONOURS) Faculty of Information and
Communication Technology (Kampar Campus)

JANUARY 2021

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: IMAGE RECOGNITION EXPENSE EXTRACTION

Academic Session: January 2021

I KOK WEI JIN

(CAPITAL LETTER)

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

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

Verified by,

(Author's signature)

(Supervisor's signature)

Address:
15, Jalan Bukit Rahman Putra 1/8
47000 Sungai Buloh, Selangor
Malaysia

Yap Seok Gee

Supervisor's name

Date: 16/4/2021

Date: 16/4/2021

IMAGE RECOGNITION EXPENSE EXTRACTION

BY

KOK WEI JIN

A REPORT SUBMITTED TO
Universiti Tunku Abdul Rahman
in partial fulfilment of the requirements
for the degree of
BACHELOR OF COMPUTER SCIENCE
(HONOURS) Faculty of Information and
Communication Technology (Kampar Campus)

JANUARY 2021

DECLARATION OF ORIGINALITY

I declare that this report entitled "**IMAGE RECOGNITION EXPENSE EXTRACTION**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature : 

Name : KOK WEI JIN

Date : 16/4/2021

ACKNOWLEDGEMENTS

I shall begin with thanking my supervisor, Ms. Yap Seok Gee. She has helped me immensely in clarifying the various questions I had about the scope and objectives of this project, and has played a pivotal role in my journey of transforming this theoretical idea of a project into a concrete one. Many thanks also to Ts. Phan Koo Yuen, who helped critique my project in a constructive manner, such that I may improve upon it.

I would also like to thank my family and Ms. Yeo, for they have stood by me in the hardest of times when it seemed as if my world as I knew it would perish before my eyes. I am eternally grateful for their unyielding support, and shall never be able to repay their kindness.

And also to UTAR, which has expended a mighty amount of resources to adapt to this pandemic, such that their students may continue with their education.

ABSTRACT

This is a mobile application development project developed for academic purposes. The topics covered are mobile development and OCR. Keeping track of income and expenses both in the short and long term is integral for long-term financial growth, as evidenced by the resources allocated for income and expense tracking in large organizations. Both accounting staff as well as personal assistants to managers may perform resource tracking work. In order to achieve long-term financial goals, families may also want to keep track of financial resources. However, while it may be an essential behaviour for long-term financial growth, income and expense tracking is generally a behaviour that takes effort and discipline. All parties can benefit if the effort and discipline required for tracking is lessened through the development of this application. The examined research includes discussion on the suitability of OCR and Spectral clustering, as well as the pre-processing steps before using Spectral clustering, alongside proposed improvements. Research on training neural networks using transformation and training data of deformed receipt images has examined. The report also details the process of how deformations may be synthetically added to the training to the training dataset, which type of neural network is trained to remove deformations from receipts, and how the receipts may be further processed. The proposed methodology is rapid application development (RAD). The four deliverables of each phase include: a list of functional and non-requirements, a sequence diagram, application iterations, as well as the complete application.

UNIVERSITI TUNKU ABDUL RAHMAN

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

GUI	Graphical User Interface
OCR	Optical Character Recognition
MBR	Minimum Bounding Rectangles
3D	Three-Dimensional
CSV	Comma-Separated Values
IDE	Integrated Development Environment
GPS	Global Positioning System
RAD	Rapid Application Development
XML	Extensible Mark-up Language
UML	Unified Modeling Language
XLS	Excel Spreadsheet

Chapter 1: Introduction

1.1 Problem Statement

Keeping track of expenses and income is integral for long-term financial growth. This statement is true for organizations and families. In organizations, this responsibility is fulfilled by employees at different roles. In families, it is usually up to the parents. Commonly encountered problems include: a lack of discipline and willingness to track expenses, the accuracy of OCR functionality in applications which are supposed to ease the burden, and a lack of functionality to track the expenses of family members.

There are various types of people who keep track of expenses like accounting staff, personal assistants, and individuals and families who want to follow a budget. For example, a personal assistant to a manager may need to records various expenses for his boss as well as help out in errands. If there are many receipts, it is challenging for the personal assistant find out if the boss is spending at a desired rate, and to notify the boss. An accounting staff may need to record expenses for all relevant staff to produce accurate expense reports. It is challenging for accounting staff to use current applications to accurately record expenses through OCR software. Individuals and family members who intend to follow a budget would also want to record their expenses, but may find it hard to maintain the discipline and effort required to do that. Heads of families also have difficulty in finding an application that allows managing various dependants' expenses at a glance from an application.

All types of people and roles that would keep track of expenses find it hard and cumbersome to maintain the discipline and effort required to manually type and enter expense records into the various popular budgeting applications available today. This is especially true if recording expenses is not a person's job responsibility, but rather a way for that person, for example a family member to determine if they are following a budget. Decreasing the discipline and effort required to track expenses and income will help the above mentioned types of users to actually perform required expenses or income tracking

Of those applications that make recording expenses easier by reducing manual entry of expenses and including OCR functionality, many applications are inaccurate. It is not that OCR functionality like those provided through Google's Mobile Vision API are inaccurate, it is that the methods used by the various applications to extract expense data from the text detected by APIs, be they custom algorithms or proprietary expense data extraction APIs, are inaccurate.

It is important that the above mentioned types of users maintain a high degree of accuracy in their expense recording, so as to obtain accurate information regarding the state of personal or organisational expenses and income.

Heads of families may need to be able to view and manage expenses of their dependants and spouse. To the author's knowledge, of the few most popular applications today, none of them have this functionality. It is important that families who want to follow a budget be able to view and manage the expenses of their dependants so that they are accountable.

1.2 Background and motivation

The result of this project will be a mobile-based software system.

The scope of this project is to create a mobile Android application that implements the Google Vision API to accurately identify the text in a receipt image. A custom algorithm will be used to accurately extract the expense value and date of a whole individual receipt. However, the scope of this project will not include extracting the expense values of individual expense items within a whole individual receipt. The scope will include recording the location of where the expense was incurred through the device's GPS coordinates when adding a record, but not through extracting the location from the receipt image.

The project will implement SQLite as the database management system of choice, and extract the location where the expenses was incurred by detecting the GPS coordinates of the device's location when recording the expense with a single tap.

After extracting expense data from receipts, the application will display the data through a GUI to allow users to confirm the data that is to be recorded or reject it. This application will also allow users to backup and restore their data through Google Drive, according to the Google account currently signed in within the application. This application will not implement various sub-functions present in the reviewed applications because of concerns relating to the ratio of time and effort required versus the frequency of use. Examples of functions not included are: allowing users to see how much they have spent compared to the same period last year, tracking of credit limits, and the tracking of gross income spent on debt, as well as monthly and yearly views of expense and income data.

If time permits, the project may include functions to extract the expense values of individual expense items within a whole individual receipt, to provide a more detailed description of each expense.

My motivation is to help people align their spending behaviour with their financial goals. From personal experience, it is not unusual to spend money in a way that does not strictly follow a pre-planned budget. It is important for accounting staff, personal assistants, and individuals and families to be able to know how to spend money in a way that re-aligns them with their spending goals, in the event that they have exceeded their budget. In these demographics with busy schedules and many tasks, a more efficient method perform their task will surely be a valuable addition to their lives and help allocate more time to solve other problems.

The problems mentioned in section 1.1 are not trivial. Decreasing the effort required for recording expenses and income, having an accurate enough OCR functionality as well as being able to view and manage dependent expenses at a glance may help users be more financially disciplined and aware of their financial situation.

When the above mentioned type of users are aware of financial situation, they may be able to make more informed decisions regarding their financial resources, and in consequence increase their financial literacy. Based on the report on Standard & Poor's rating services Global Financial Literacy Survey, only 37% of adults in Malaysia are financially literate (Klapper, Lusardi & van Oudheusden 2014). From this fact, it is

clear that there is room for improvement in financial literacy for Malaysians, and this project seeks to aid in that goal.

1.3 Project Objectives

The aim of the project is to use OCR in identifying text, and then to use a custom algorithm to extract the expense value (also known as the total value of a receipt) and date from a whole individual receipt as part of the budgeting application, as well as to develop a way to view each individual family member income through the device of the head of family, and also to derive supposed rates of spending intended to achieve a previously planned budget by developing a custom algorithm. All this proposed functionality will be implemented on top of basic functions for budgeting applications, such as the manual recording of income and expenses. Implementing these functions, the application will able to solve the problems previously stated in 1.1.

Each family member's income can be represented by an account. At any point in time, the head of the family can choose to view and manage the expense data of each of each of these separate family members.

Supposed rates of spending can be determined by allowing a user to determine a budget for spending for a specific period and category. At any date, a user may records expenses. The difference between the planned spending of an amount within a period and the actual amount of money spent, and then divided by days within that period without any spending will be the supposed spending rate, which gives users an idea of how much to spend to maintain their budget.

The contribution of this project is to develop a software system as to lower the effort and discipline required to keep track of a person or organisation's financial status to gain knowledge that helps promote financial literacy.

1.4 Proposed approach

Attached below is the activity diagram or system flowchart which illustrates major decisions and actions related to the each of the major functions in a high-level manner.

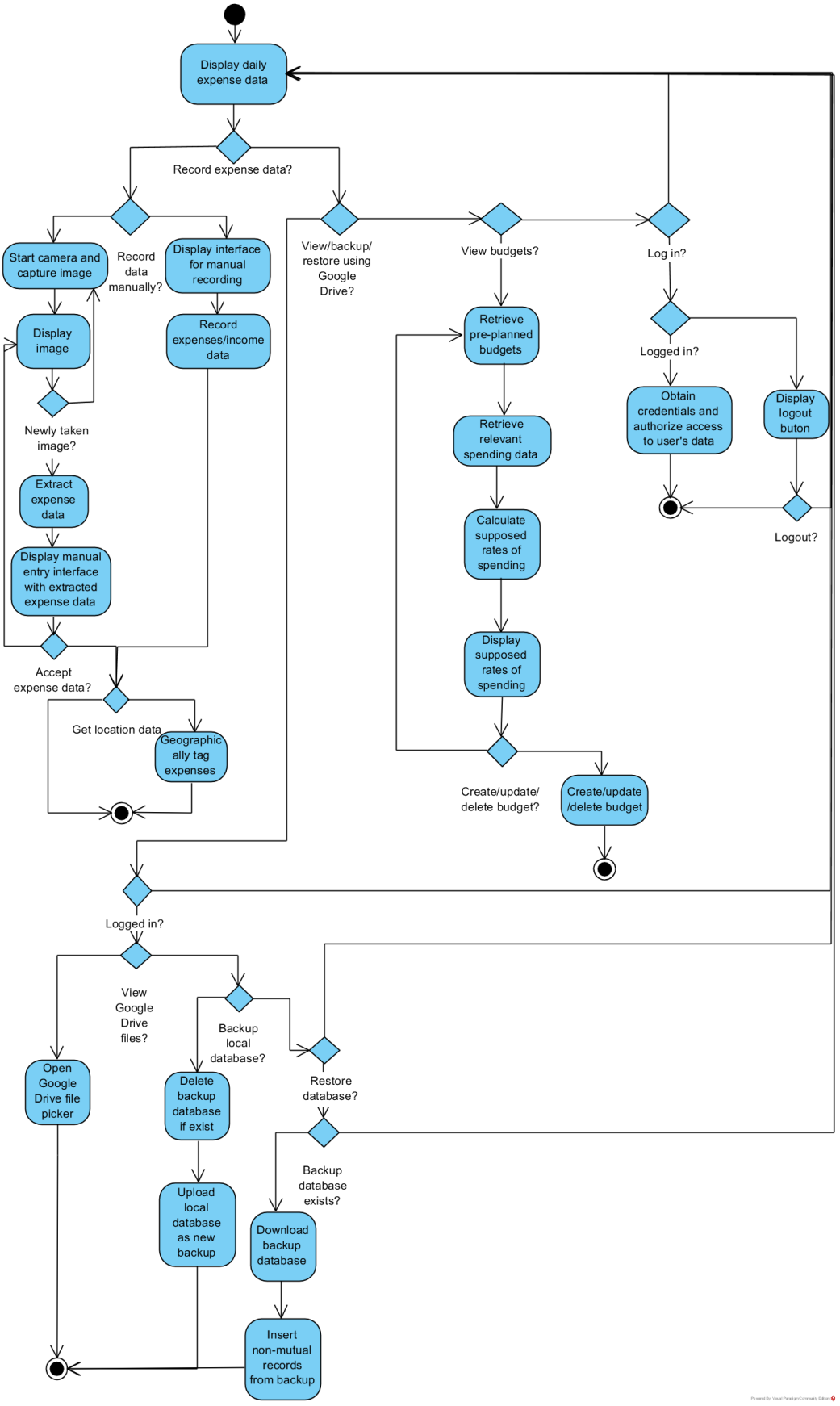


Figure 1.1 Activity Diagram of the application

The Rapid Application Development methodology which has four phases, namely: requirements planning, user design, construction and cutover phases, will be used in this project.

The major functions shown in the activity diagram above and their corresponding user requirements are listed in the following table. Unless stated, a number represents a functional requirement:

Major functions	**Corresponding user requirements**
Record expenses	1, 5, 6 Non-functional: 1, 2
Drive and login	2, 3
Budget	4

Table 1.1 Major functions and corresponding user requirements

1.5 Highlight of what have been achieved

All user requirements have been achieved. User requirement 1 has been achieved such that the total amount and date of a whole individual receipt can be extracted through the use of a custom algorithm, after text from a receipt has been detected and extracted by using a Google's Mobile Vision API. After automatic extraction of the total amount and date, the extracted data is displayed using the GUI that was used for manual entry, which allows users so confirm the expense by pressing the "Add" button or reject the expense by pressing the back button, after expense details have been automatically detected and extracted by the custom algorithm, which satisfies user requirement 2. Of course, users can also manually enter expense data through the implemented GUI and activities, which utilize the SQLite DBMS for storage. In both manual and OCR cases, users can use the in-built function that obtains the device's location when the user is recording the expense for location tagging. Thus, user requirements 6 and 7 are achieved, in addition to 1 and 2.

Users can set a spending limit for a category and for a set period, or a budget. The application then derives the supposed rate of spending for that budget so that users will

know how much to spend per day for days that do not yet have any expenses, given the amount they have already spent within that period. Finally, users can login and backup their expense and budget data, implemented using the Drive API, thereby representing their own expense data using an account, satisfying requirement 3. Additionally, users can switch between different accounts in the same application on the device, which means that any user which has dependents can request them to back up their data on their separate device. The head of family can now log into their dependent's account on his own phone, restore the backed up data which allows him to add or remove expense or budget records and upload any modifications, thereby allowing him to view or manage his dependent's expense data. This functionality satisfies requirement 4.

Thus, all user requirements from 1 to 7 have been satisfied, and aligns with the project's motivation. The project also succeeds in helping the demographics mentioned in the problem statement, as well as in its objectives.

Also worth mentioning is the work put into making the project conform to material design standards, such that the application may be more user-friendly. Below are a few screenshots of the expense and income daily view as well as the budget list. Measurements of interest are listed below.

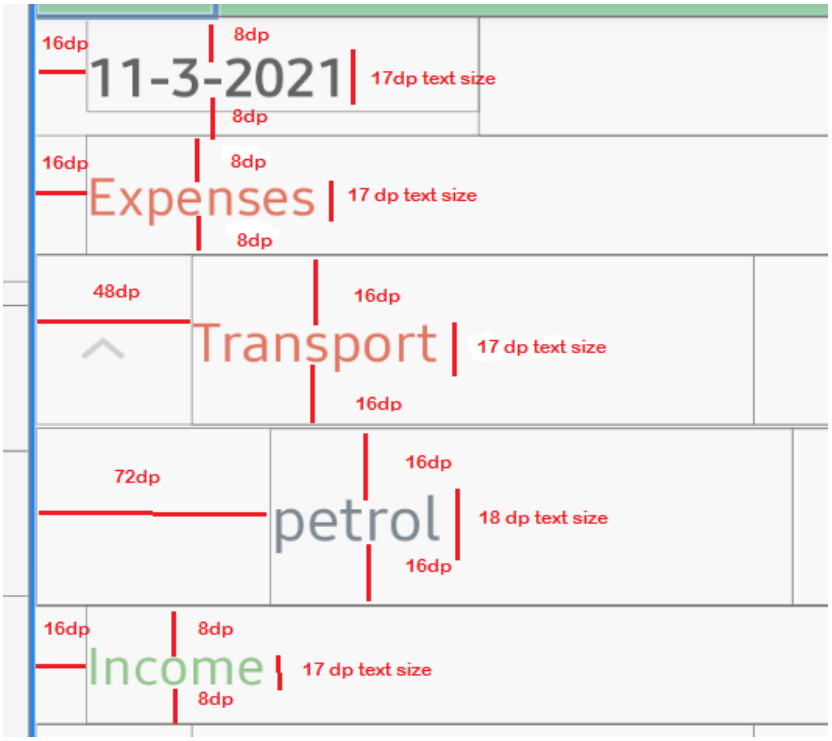


Figure 1.2 Dimensions of daily expense or income view

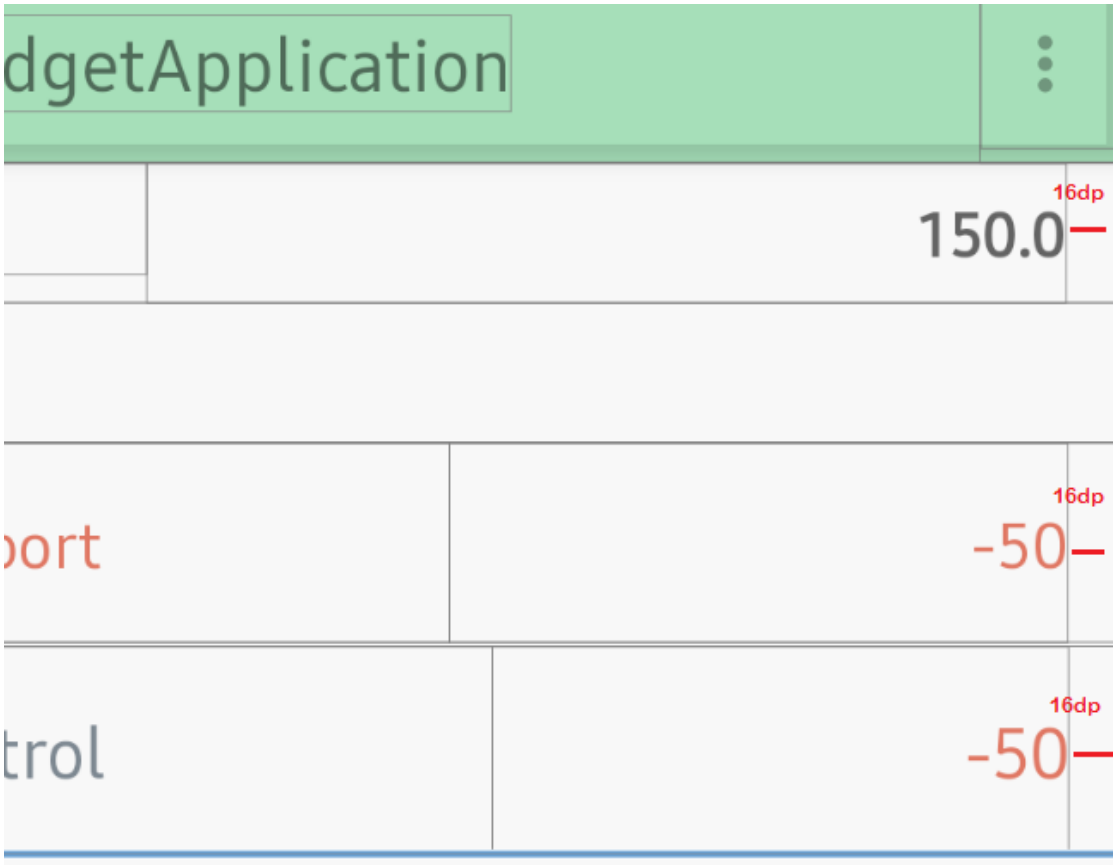


Figure 1.3: Dimensions of daily expense or income view 2

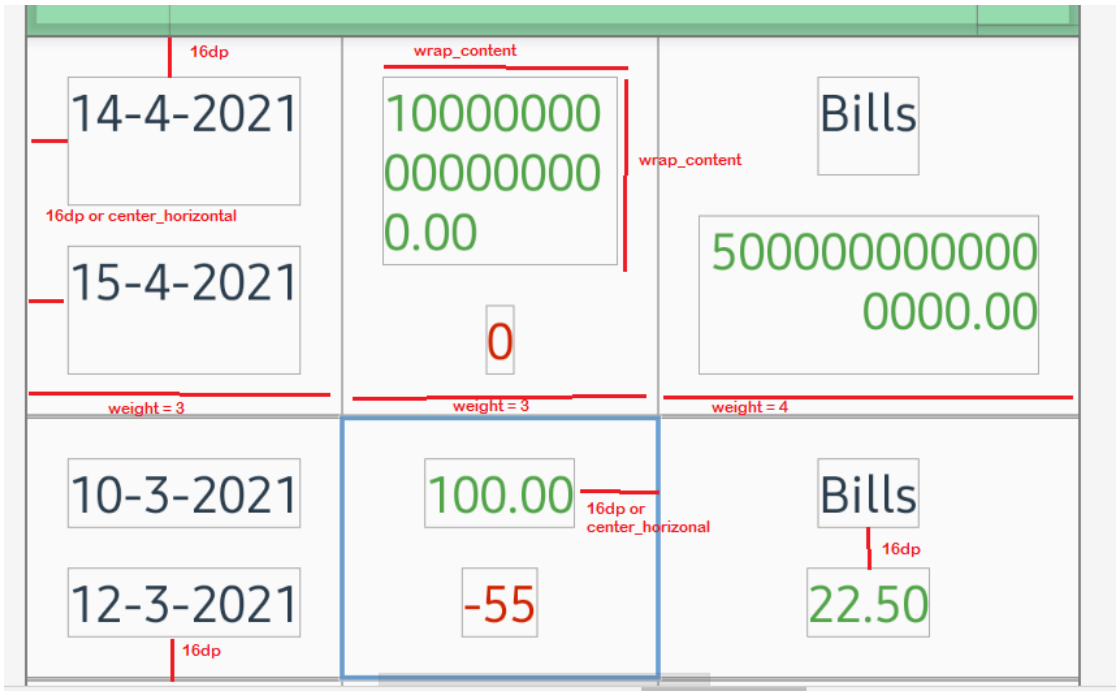


Figure 1.4 Dimensions of budget list view

1.6 Report organization

This report will provide an overview of the system design methodology of the proposed system, deliverables in each stage, tools used for its development, as well as any design diagrams that enhance understanding of how to use the application for different use-cases. In chapter 2, various similar applications will be reviewed in regards to their functionality, as well as strengths and weaknesses. In chapter 3, the various aspects of system design will be fleshed out, including the various UML diagrams.

Chapter 2: Literature Review

The author has considered reviewing different text recognition or OCR APIs for this project. However, from observation Google's Mobile Vision API detects text accurately enough, at least for expense total amounts and dates for whole individual receipt images. This means that for the expense total amounts and dates in most receipts, there was not a difference large enough between the text in the physical receipt and the text detected from images of those physical receipts by the API such that the author was hampered in developing a custom algorithm to extract expense data from the detected text.

In a practical sense, this meant that if the receipt contained the word "total" or the number "23.50", much more than not, the API would detect the text accurately. However, there was no free API available online as of this report to extract expense data. This solidified the author's view that the previous works compared should be about the expense extraction functionality instead of OCR APIs.

The author has developed a custom algorithm to extract expense data from detected text. However, due to the nature of examined mobile applications, there is no way of examining the source code and methods that are part of other products since they are not open source.

2.1 Application review

2.1.1 Easy Expense

While using Easy Expense (Easy Expense Tracker, 2021), there are 2 ways to record expenses: "Manual Expense Entry" and "Smart Receipt Capture", wherein the latter uses OCR functionality as part of its effort for expense data extraction. From first capturing the image to saving the expense total amount and date of a whole individual receipt, the time amounts to about 5 seconds per receipt. However, the Smart Receipt Capture functionality often times fails to properly identify the bounding area in an image for which a receipt occupies, needing to user to manually adjust boundaries.

In regards to account related functionality to view and manage different family member expense data, Easy Expense allows users to login using an Easy Expense account, and

export and import .csv data for free. However, if the user want to use the in-build backup feature, it is unfortunately a paid one.

The application provides some visualization of expense data, breaking down expenses by groups, and allowing monthly, quarterly and yearly views of a summary. However, it does not provide any functionality to calculate the supposed rate of spending.

Functionality (free)	**Automated data extraction**	**View and manage family members**	**Supposed rate of spending**
Featured in application	✓	×	×

Table 2.1: Available features for free in Easy Expense

2.1.2 Smart Receipts

Smart Receipts (Smart Receipts LLC, 2021) records expenses by first creating "Expense reports" with a start and end date, within which a user can record expenses manually either through text or images, which the user can then generate a report containing a table for all expenses during this period, as well as any attached receipts if present. The user can also use the "Automatic Scans (OCR)" functionality to record expenses automatically needing only to capture receipt images. The methods used to implement this functionality include "Google Vision and Machine Learning" as well as integration with the Taggun API, which is a paid proprietary API for expense data extraction.

For account-related functions for viewing and managing dependent related expense data, the application includes functionality to backup and restore expense data for free to Google Drive, similar to what this project's author has done.
However, it needs to be said that there is no functionality to calculate the supposed rate of spending for a user, nor set a budget for a predefined period. For helping the user make better decisions about their spending behaviour, there is only functionality to graphically plot total expenses per day on a graph for a set period, as well as visualizing the proportion of each expense category using a pie chart.

Functionality (free)	Automated data extraction	View and manage family members	Supposed rate of spending
Featured in application	×	✓	×

Table 2.2: Available features for free in Smart Receipts

2.1.3 Spending Tracker

In using Spending Tracker (MH Riley Ltd, 2021), there is only one method to record expenses, and that is manually. There are no features that help automate the extraction of expense data.

An upside is that for account-related functions, this Spending Tracker actually does quite well. Not only is it possible to create multiple accounts under one device to represent different people or different accounts under the same person, it is possible to back up and restore expense data to a Dropbox account, for free. If a need arises to obtain a dependent's expense data for viewing and managing, it can be done quickly.

A downside to this application is that there is essentially no functionality to help the user make more informed spending decisions. There is even no functionality to manually create budgets, much less calculate the supposed rate of spending for any period. The only upside in this area for this application is that it allows the viewing of transactions according to the different categories.

Functionality (free)	Automated data extraction	View and manage family members	Supposed rate of spending
Featured in application	×	✓	×

Table 2.3: Available features for free in Spending Tracker

2.1.4 TimelyBills

There is only one way to add an expense record to this TimelyBills (TimelyBills, 2021), and that is manually. There is automated expense data extraction support through any other means, whether custom algorithms or proprietary APIs.

For account-related functions, it does quite well. This is because it allows users to backup and restore their expense data to Google Drive, similar to what is provided by the author's application. Not only that, TimelyBills also provides special functionalities for budgeting and money management for family members. Unfortunately, the big downside is that for the above mentioned account-related functions, both functions are paid features, and do not even have a trial option for new users to try out before paying.

TimelyBills provides many features to help the user in making better financial decisions. In addition to yearly views of expenses, income and balances, it allows the setting of expense reports to be viewable by family groups, and even tries to predict trends according to spending behaviour. It is unsure how many of those features are paid or are provided for free. In addition, another downside of TimelyBills is that it does not help calculate the supposed rate of spending.

Functionality (free)	**Automated data extraction**	**View and manage family members**	**Supposed rate of spending**
Featured in application	×	×	×

Table 2.4: Available features for free in TimelyBills

2.1.5 Wallet

Similar to many of the previous applications the Wallet (BudgetBakers.com, 2021) application only has one way to record expenses, and that is manually. There is no support for automated expense data extraction from receipt images.

For functionality related to viewing and managing dependent expense data, the application allows the user to create an account to login. What is there to help backup and restore expense data? To backup expense data, users have to export either a PDF, XLS or CSV file using the settings contained within the application. Not only that, the user still has to confirm the backup process by opening an email which contains a verification code, which is another step to complicate the back up process. Finally, after confirming the back up request, the user can go into the application and press import, so that the expense data can be restored. However, the most frustrating part of this

application is that in addition to the complicated steps required for expense data back up and restore, the functions required for them are actually paid, and are not free to use.

For functions that help users improve spending behaviour, this application has the most comprehensive suite of services available. Not only does it allow users to create budgets for a set period, amount and category while analysing spent amounts and extrapolating spending totals into the future while plotting all that info on a graph, it also automatically updates the daily recommended amount to be spent given current expense records. The daily recommended value in this application is also known as the supposed spending rate in the author's application.

Functionality (free)	**Automated data extraction**	**View and manage family members**	**Supposed rate of spending**
Featured in application	×	×	✓

Table 2.5: Available features for free in Wallet

2.1.6 Money Lover

For Money Lover (Finsify, 2021), there is only one method of adding expenses records, and that is manually. There is no support for automated expense data extraction from images.

For functionality to view and manage dependent expense data, it is possible to export either CSV or excel files, and to import them manually after the fact. However, this process is unnecessarily cumbersome. Not only that, the export functionality requires that the user upgrade and pay for the privilege to do so.

On the bright side, one of the redeeming features of this application is that it not only allows users to create budgets, events, track recurring transactions and bills to help users spend more responsibly, it automatically calculated the recommended daily amount that is to be spent within a period given a certain amount of income and expenses, which is similar to the function to calculate the supposed rate of spending in the author's application.

Functionality (free)	**Automated data extraction**	**View and manage family members**	**Supposed rate of spending**
Featured in application	×	×	✓

Table 2.6: Available features for free in Money Lover

2.2 Summary table of features

Below is a table that summarizes the available features for free in each application reviewed.

Functionality (free)/ Featured in application	**Automated data extraction**	**View and manage family members**	**Supposed rate of spending**
Easy Expense	✓	×	×
Smart Receipts	×	✓	×
Spending Tracker	×	✓	×
TimelyBills	×	×	×
Wallet	×	×	✓
Money Lover	×	×	✓

Table 2.7: Available features for free in each application reviewed

Chapter 3: System Design

3.1.1 Proposed Methodology

The author proposes that this project be developed using the rapid application development (RAD) method. Since the author's proposed system has five well-defined objectives, where only one person develops the system and makes decisions, this development methodology suits the proposed system.

The RAD method has four phases: requirements planning, user design, construction and cutover phases.

Requirements planning phase:

Since there is only one person developing this application, all decisions are made unilaterally. Project scope, constraints, and system requirements are decided by developer or author. There are no consensuses to be reached.

Deliverable: List of functional and non-functional requirements

User design phase:

In this phase, the author will develop models that represent the flow of the whole system, including major actions and decisions. Since the system is not intended to be deployed in a commercial sense, the sole user in this case will be the author. An activity diagram will be created using Visual Paradigm to illustrate the system flow and provide a high level view of how each available major activity is affected by various decisions.

Deliverable: Activity diagram

Construction phase:

In this phase, the author will utilise Android Studio to develop a mobile application with OCR functionality, in addition to functionalities mentioned in the project scope. Unit testing of the mobile application will also be carried out.

Deliverable: Application iterations.

Cutover phase:

As there are no outside users to train, there will be no user training done other than required for the author to be proficient in using the mobile application. Final testing will be done to make sure application functions as intended.

Deliverable: Complete application.

3.1.2 Technologies used

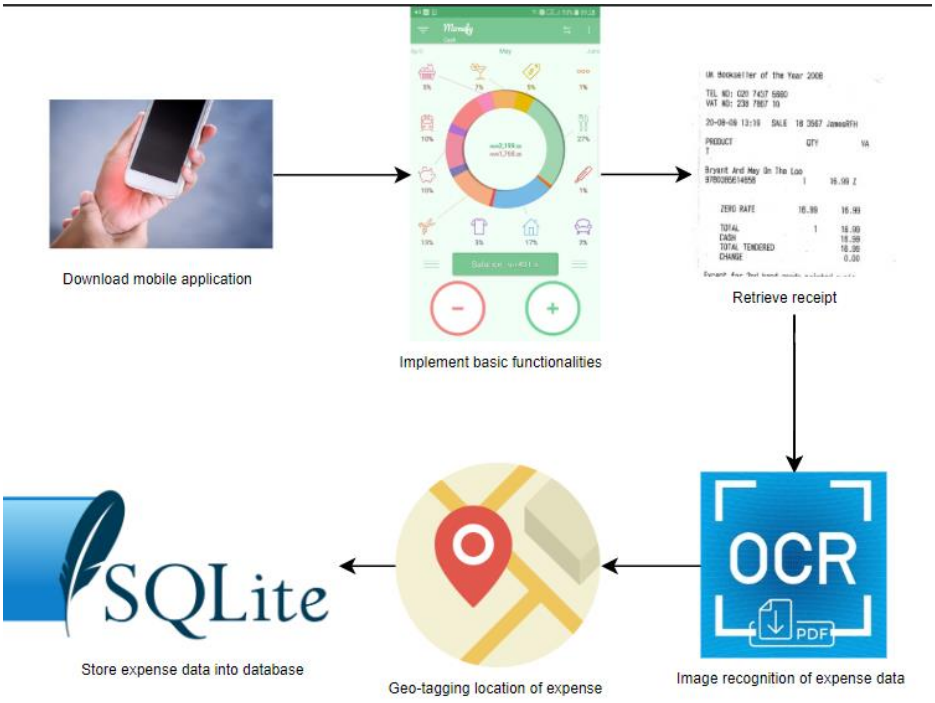


Figure 3.1: Tools used

Through Android Studio 4.1, the application will have multiple interactive components, such as buttons, and textboxes throughout the layout of the application. If expenses of several family members are combined under a master account, the user will be able to access the various pages in the application to inspect the expense details. The basic functions of manually recording income and expenses and exporting data will be developed with this IDE. Separate expenses can only be obtained by synchronization through an the use of the Drive API (v3-rev110-1.23.0), by backing up dependent expense data on his or her respective device, and then logging in to the relevant Google account on the head of family's device to initiate a restore process, which downloads the previously backed up database on the latter's device.

Budget and expense records will be stored inside an SQLite database, version 3.22.0. The data needed for the calculation of the supposed rate of spending will be stored in the "expense" table, while all budgets will be stored in the "budgets" table of the "expenses" database.

The recording of the expense location can be done manually by the user by recording it in the expense description, or it can be done automatically through the detection of the receipt's venue through the API. Furthermore, the Google Mobile Vision API will also be implemented through Android Studio. For the Google Mobile Vision API (firebase-ml-vision:19.0.3), Internet connectivity is first required for usage.

3.2 User Requirements

User requirements include:

1. Extraction of total amount and date of expense from a receipt image locally by using a custom algorithm, after using Google's Mobile Vision Text API to detect text in individual pictures after it has been taken.
2. Represent each family member's income by an account.
3. View and manage the expense data of each separate family member.
4. Derive supposed rates of spending intended to achieve a previously planned budget.
5. Manual recording of income, fixed income and expenses by implementing relevant user interfaces, activities, and a database for storing such expenses.
6. Location tagging by detecting the device location when a record is added.

Non-functional user requirements include:

1. Display expense data through a GUI to allow users to confirm or reject the data that is to be recorded after expense details have been detected and extracted automatically by the custom algorithm.
2. A processing time of 5 seconds per receipt which is the time elapsed from the capturing of the picture by the user, to displaying expense data in the GUI for users to confirm or reject.

If time permits, the application would also allow the extraction of individual expenses from a receipt, and not only the total amount of all expenses.

3.3 Use-case diagram

The image recognition expense extraction system has 3 actors, namely family members, personal assistants, and accountants.

The system has four use cases:

1. Extract expenses through images
2. Manually record income and expenses
3. View and manage family member expense data
4. Derive supposed rates of spending

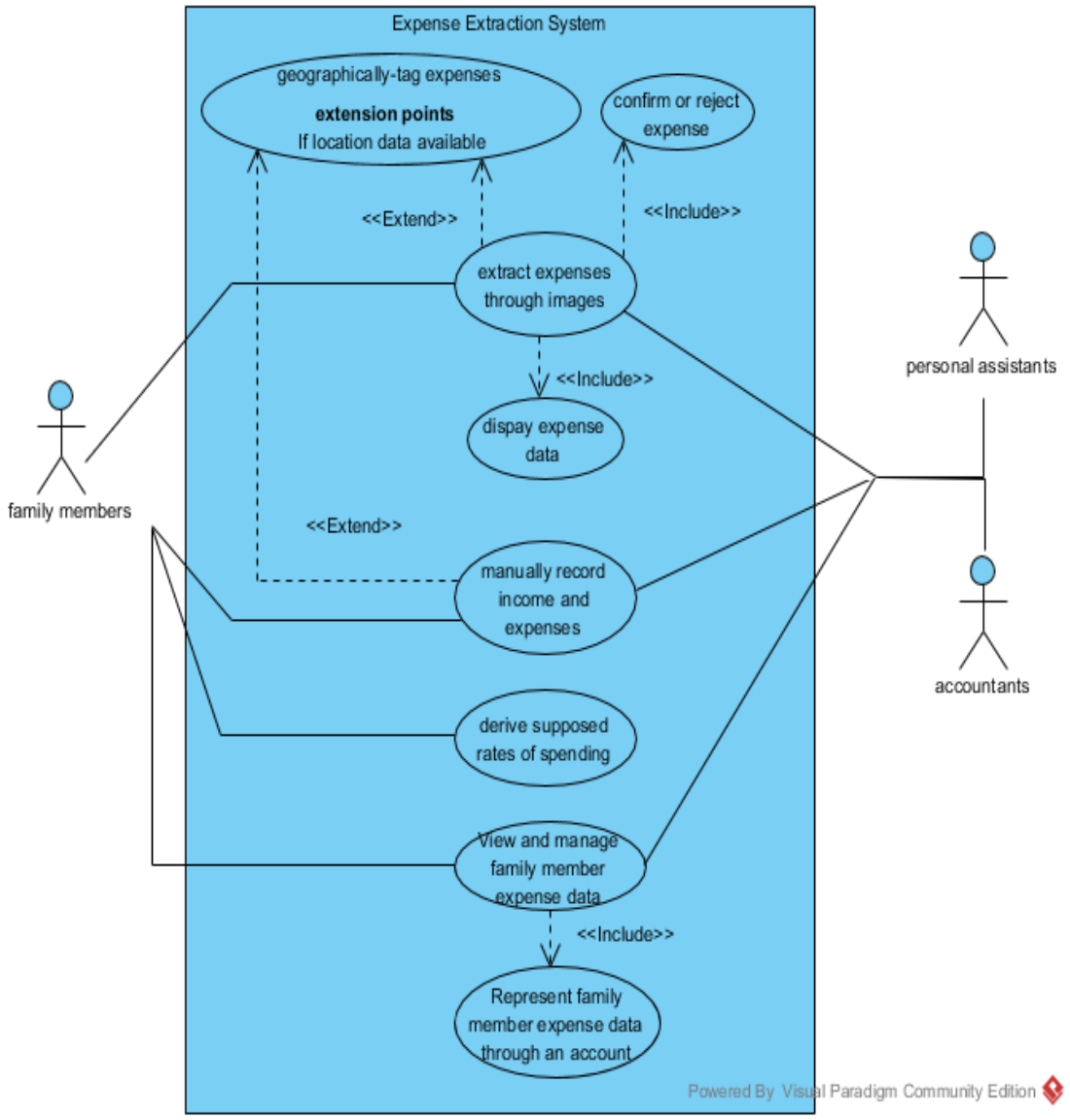


Figure 3.2 Use-Case diagram of the application

The two use cases of extract expenses through images and manually record income and expenses can extend their functionality to geographically-tag expenses if location services are permitted by the user. If any of the actors choose to record expenses through image recognition, the extracted expense data will first be displayed and confirmed by the user before being recorded.

3.4 Activity diagram

Below is an activity diagram for the overall execution of the system. It details how the flow of the system is affected by various inputs, such as the decision of the user to record expense data using either of the two methods or log in to Google services to view, manage family member expense data, or to create budgets and view supposed rates of spending. It also details how the flow is affected if certain data such as location and budget data are non-existent.

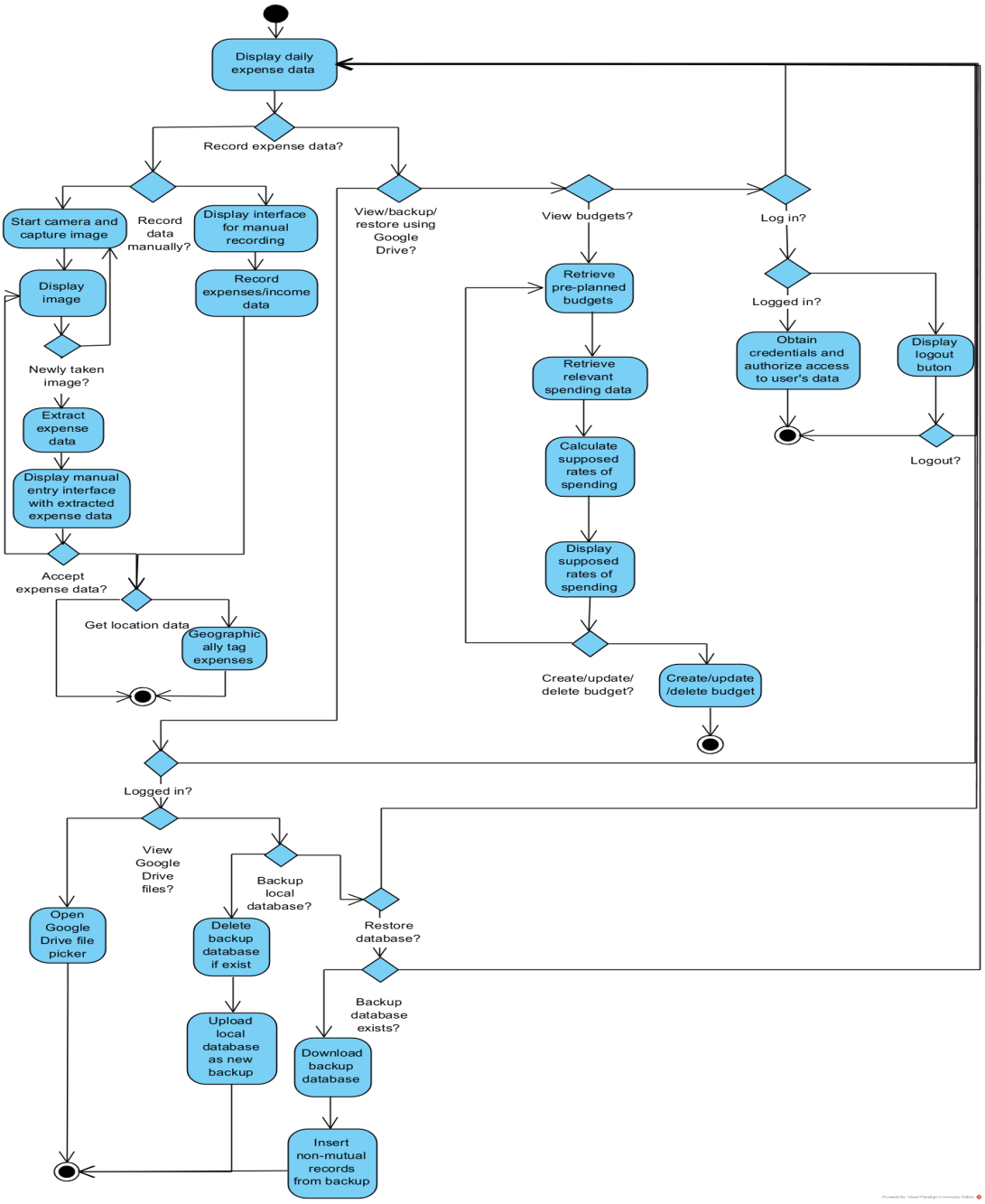


Figure 3.3: Activity Diagram of the application

Chapter 4: Display Daily Expense Data

4.1 Methodology, tools, libraries and classes of interest

For displaying the daily expense data on the main activity when the user first starts the application, a PagerAdapter (SliderAdapter) is used to represent the expense of each day, with one page representing one day's expenses. Within one page or day, there further exists 2 BaseExpandableListAdapters (OneLevelExpenseAdapter), each representing an expandable list of either expenses or incomes, with the 0^{th} level being expense or income categories such as Bills, Transport, Food, Eating Out, and so on, and the 1^{st} level being records under that category for that day. Both the SliderAdapter and OneLevelExpenseAdapter are integral in organizing the daily expense views.

4.2 Requirements

The display functionality is the most basic of functions in a budget application. A budgeting application simply cannot exist without it. It is not included as part of the functional or non-functional user requirements. However, the author has tried to conform to material design guidelines so that information access will be easier and the application will be user friendly overall.

4.3 Implementation and testing

In implementing this functionality, the author has performed a reasonable set of tests to solve possible erroneous situations or cases that might arise that would try to ensure the viewing function works as designed. A few examples of erroneous cases tested for are:

1. What if expense descriptions are unusually long? Is it possible to ensure the containing views do not affect neighbouring ones?
2. What if there is only one of expense or income on a day? Will the expense option view still be positioned properly such as to not negatively affect the others?
3. If an expense or income is deleted or updated, will the changes show when the user returns to this activity?
4. If there are a mix of incomes and expenses on a day, will the sum of both options be correct when listing it alongside the same level as the date?

Below is the class diagram to illustrate the classes relevant to implementing this functionality:

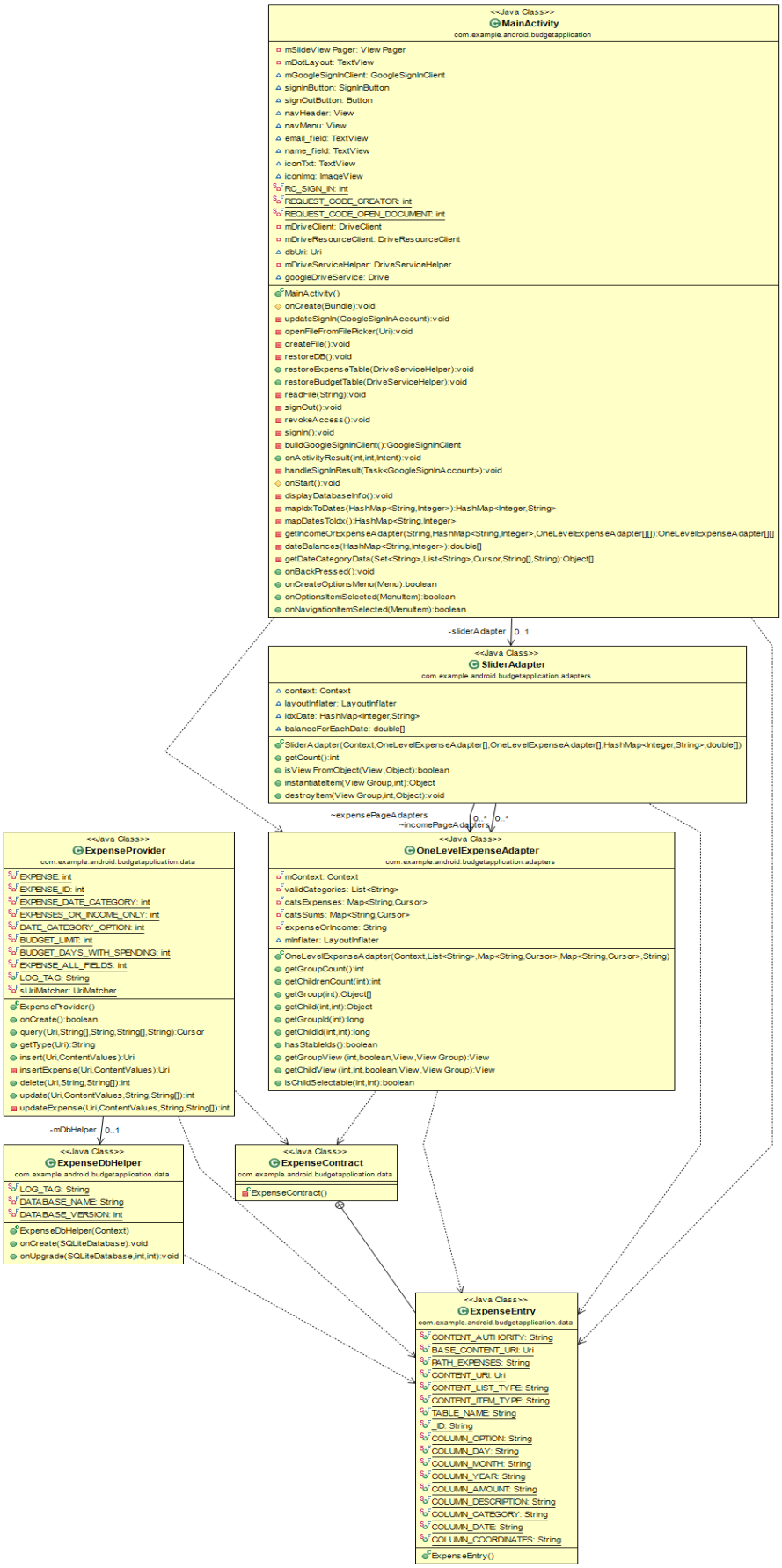


Figure 4.1: Class diagram, display daily expense data

Chapter 5: Manual Expense Entry

5.1: Methodology, tools, libraries and classes of interest

For manually recording expenses, the author has used a varied amount of more uncommon classes. The list is: DatePickerDialog, LocationManager, LocationListener, ScheduledExecutorService, and LocationService, and BroadcastReceiver. As is self-explanatory, the DatePickerDialog is for users to set a date for the current expense. LocationManager and LocationListener are used to obtain user permissions regarding location services and to obtain the user's device location respectively. LocationService is actually a service that implements the LocationListener interface, and together with BroadcastReceiver it can broadcast and relay the obtained location back to ManualEntryActivity. Not only that, when BroadcastReceiver is used with ScheduledExecutorService, the author managed to implement a timer function that counts down and displays on screen the seconds elapsed starting form when the user requests for location data to be obtained and ending when the result is returned. To note, the timer function actually runs in a background thread such that it does not interfere with more time-sensitive operations like UI updates that should have as much leeway as possible in the main thread to run. The integral classes are ManualEntryActivity and LocationService.

5.2 Requirements

With this functionality implemented together with the location gathering service, the functional requirements of 5 and 6 respectively are satisfied.

5.3 Implementation and testing

In implementing this functionality, the author has performed a reasonable set of tests to solve possible erroneous situations. A few examples of erroneous cases tested for are:

1. Whether the day, month, and year selected by the user is inserted correctly into the database.
2. Whether the amounts recorded by the user have the correct sign, negative for expenses and positive for incomes.

3. Whether there is a prompt for permissions when the user wants to record the location.

Below is the class diagram to illustrate the classes relevant in implementing this functionality:

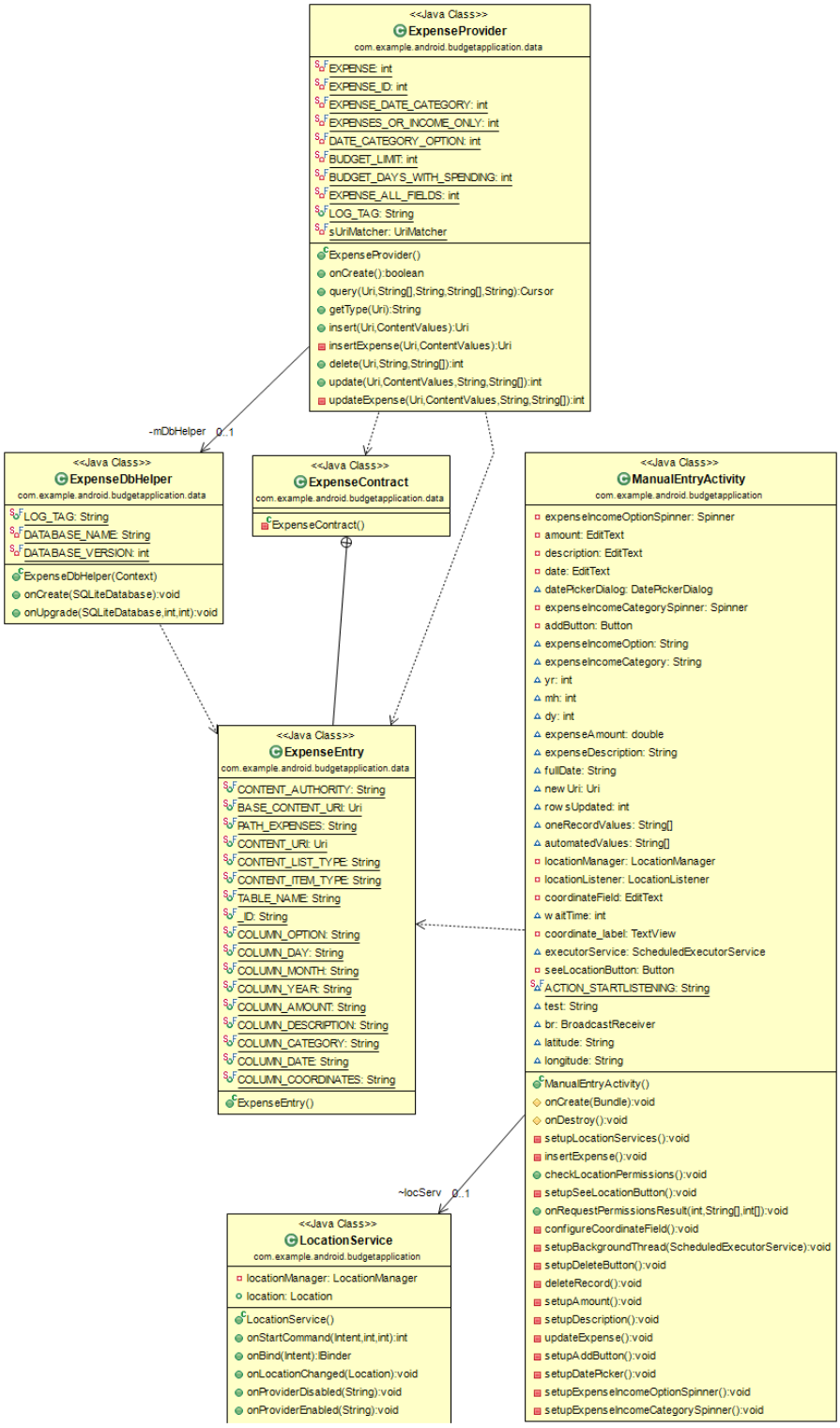


Figure 5.1: Class diagram, manual expense entry

Chapter 6: Automated Expense Entry

6.1: Methodology, tools, libraries and classes of interest

Classes of interest include ExifInterface and Matrix for creation of file objects to store captured images. For classes related to OCR and expense data extraction, the FirebaseVisionDocumentTextRecognizer allows the use of the MobileVision API to run machine learning models to detect and extract text from an image such that a FirebaseVisionDocumentText is returned on successful extraction of text. A FirebaseVisionDocumentText contains successfully smaller divisions of objects, such as Blocks, Paragraphs, Words and Symbols, where objects like Words have fields such as BoundingBox, which in turn have properties such as Bottom, or Top that contains the Y-coordinates or latitude of the bottom and top corners of the BoundingBox that contains the Word. The above classes of interest all exist in the ImageRecognitionEntryActivity class, and together with LocationService, are the two unique classes and services which contain the bulk of the integral code that supports this functionality.

6.2 Requirements

With this functionality implemented, functional requirement 1 is satisfied, as users no need to manually extract the expense date and amount of a whole individual receipt. Non-functional requirement 1 is satisfied, since the extracted data is filled into a form, where the user can reject it by pressing the back button. Not only that, non-functional requirement 2 is also satisfied since the processing time is about 5 seconds per receipt, achievable in a constant and sustainable manner. For clarification, it is worth reiterating that the processing time is defined as the number of seconds elapsed between the capturing of an image by the user, and the displaying of the extracted expense data on the ManualExpenseEntry GUI for confirmation or rejection. This of course includes processing time for the Mobile Vision API to detect the text and the custom algorithm to extract the expense data.

6.3 Implementation and testing

In implementing this functionality, the author has performed a reasonable set of tests to solve possible erroneous situations. A few examples of erroneous cases tested for are:

1. Whether dates that do not symbols as delimiters between the day, month and year can be detected for example: 01/02/2020 and 3 Jan 20
2. Whether terms representing months such as “Jan”, “february”, “May”, “mar” can be accurately identified such that other words that contain subsets of characters that entirely match those terms can be successfully filtered out.
3. Whether an impossibly wide array of sum key words such as “total”, “amount”, “due”, “subtotal”, “sub total”, “totl”, “nett” and others can be successfully identified, so that the corresponding expense sum can be extracted.

The process of extracting expense data is separated into two main groups of effort. Both start by iterating over all Blocks, Paragraphs and Words.

The first group is for extracting the expense sum by first detecting possible sum keywords like “total”, “amount” and “due” by iterating over an array and checking for any matches for any Word, which is then stored. After that, the application iterates over all the previously stored Words. For each stored Word (sum keyword), it is checked against all the Words in the receipt. The bottom threshold of the sum keyword is calculated by increasing the Bottom coordinates of its Bounding Box by 25% of the Bounding Box’s height. The top threshold of the sum keyword is calculated by increasing the Top coordinates of its Bounding Box by 40% of the Bounding Box’s height. The result is that the sum keyword’s top and bottom corners have moved down of the screen, so to speak, essentially moving the enclosing bounding box in a downward direction.

The application will then obtain the current word’s Bounding Box bottom coordinates, which is compared with the sum keyword’s top and bottom threshold. If the current Word has a bottom coordinates that are more than the top threshold and less than the bottom threshold, it proceeds to the next check. The checks after this ensures that the Word is a 2 decimal place number that is more than 0. The last Word in the whole rceipt that passes this check will be the expense’s total sum amount.

Secondly, the application moves on to extract the expense date. Similar to the sum amount, the process takes place when iterating over all Words. There are two broad categories of dates normally found in receipts. One is with symbols that delimit the boundary between the day, month and year like “20/01/2020” or “05-Jan-2020”, and the other consists of no delimiting symbols, or the “symbol” and “no symbol” cases respectively. However, the “no-symbol” case is further divided into 2 sub-categories, where one is essentially delimited by a blank space character like “01 Jan 19”, and the other has no delimiting symbols at all like “01Jan20”, or the “no-space” case. The “symbol” case is first checked for, where the current Word is checked against a reasonably exhaustive list of symbols like “/”, “-“ and “,” to see if either one is a substring of the current Word. Further checks ensure that there are exactly 2 of those symbols located in different positions, before checking if the year is located in the first or last third of the Word in question.

“No symbol” cases are separated into 2 cases, the “space” or “no-space” cases. If a “symbol” case date is not found first, the application checks for “no symbol” cases. The application then regex matches the current Word against terms like “jan”, “january”, “feb”, “february” and so on. If a match is found, it is then checked to see if the matching substring is part of a larger Word where the first and final characters are numbers. If so, the whole Word is a “no-space” date, and it has finished retrieving the date for that case.

If the whole Word is only the matching substring, and if in the previous iteration the Word is a number, then it is a “space” date. By the current iteration, the year has not been recorded yet. Only after the subsequent iteration can the application obtain the year.

With this, the expense total amount and date for a receipt has been successfully extracted from the text. When the author refers to the “custom algorithm” for expense data extraction, it is in reference to the above mentioned processes. After some formatting, the results are displayed in the same GUI used for manual addition of expense records. Users can press back to re-take a picture for better results, or proceed with adding the expense record.

Located on the next page is a class diagram for all classes relevant to this function:

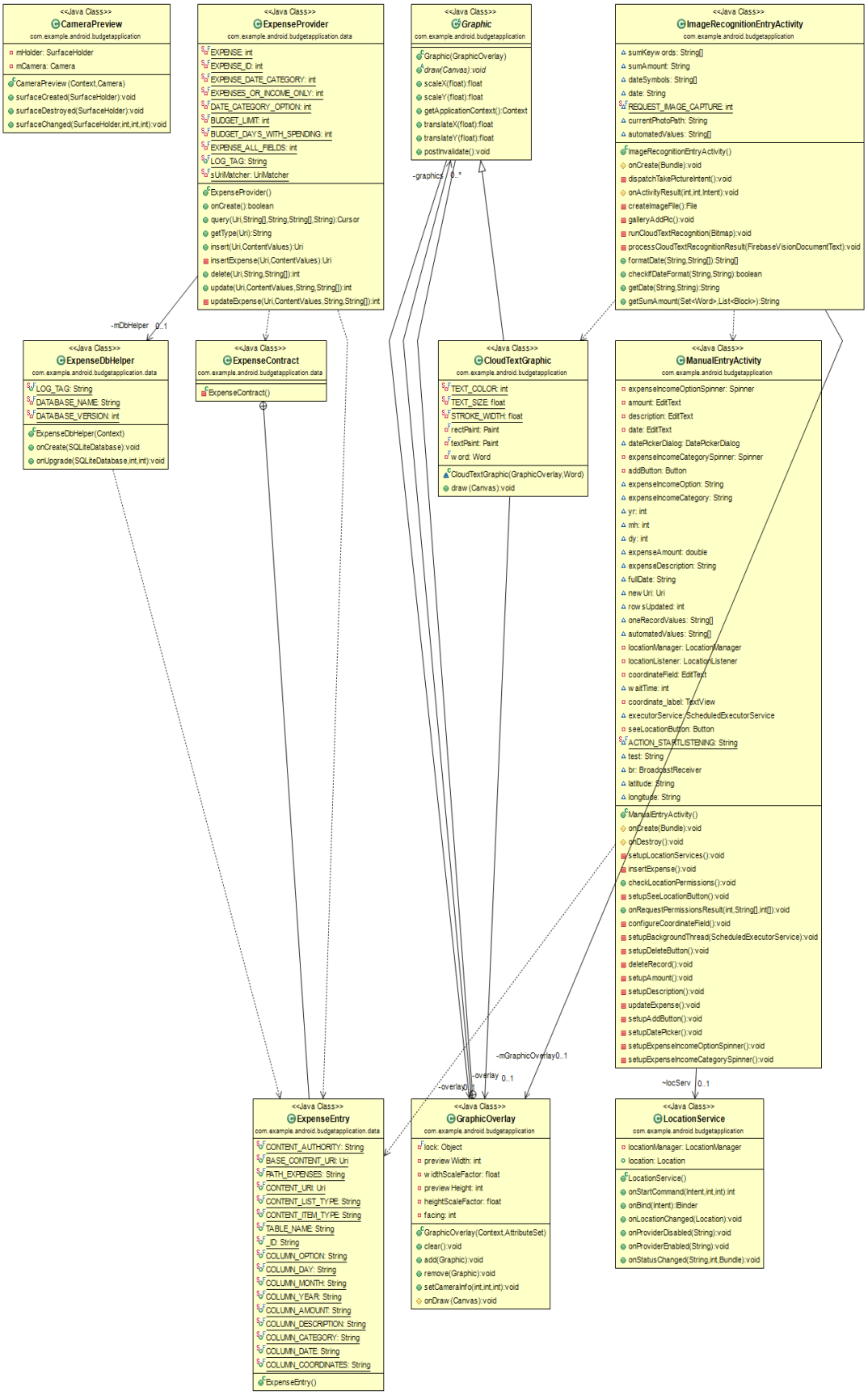


Figure 6.1: Class diagram, automated expense entry

Chapter 7: Represent, view and manage dependent expenses

7.1: Methodology, tools, libraries and classes of interest

Of special consideration is the Drive API, and the GoogleSignInClient, GoogleAccountCredential, GoogleSignInAccount, GoogleSignInOptions, and DriveServiceHelper classes. The GoogleSignInClient allows us to launch a separate intent for user-login, if the GoogleSignInButton is pressed. On return, the application tries to get a GoogleSignInAccount. If it succeeds, a GoogleAccountCredential is created, allowing us to use the Drive API to create a DriveServiceHelper, which contains all the tasks necessary to back up and restore expense date to the user's Google Drive. The above mentioned classes of interest are located in the MainActivity class. The DriveServiceHelper class is the unique integral class that supports this functionality, with the bulk of the code for the restore and back-up tasks defined there.

7.2: Requirements

Through the Drive API, the application can represent each family member's expenses by backing the expense data to their respective Google Drive storage. Furthermore, the head of family can log in to a dependent's Google account to view and manage their expense records and budgets. Thus, functional requirements 2 and 3 are achieved.

7.3: Implementation and testing:

In implementing this functionality, the author has performed a reasonable set of tests to solve possible erroneous situations or cases that might arise when trying to ensure these account-related functions work as intended A few examples of erroneous cases tested for are:

1. Whether earlier versions of back-ups are deleted before backing up the current local expenses database.
2. Whether the application alerts the user to login before accessing any account-related functionalities.
3. Whether the back-up expenses file can be downloaded successfully.
4. Whether record values are assigned into correct data types when first extracted from the back-up database.

It is worth noting that when backing-up, the uploaded database name is appended with the current time in milliseconds since January 1, 1970, to ensure a unique file name,

since it will be compared to another previous back-up. For the restore function, the application will iterate over all of the back-ups records, and query the local database for any matching records. If there are no results returned, the new unique record will be inserted into the local database. Otherwise, nothing will be done with the current back-up record. Contained below is a class diagram for relevant classes:

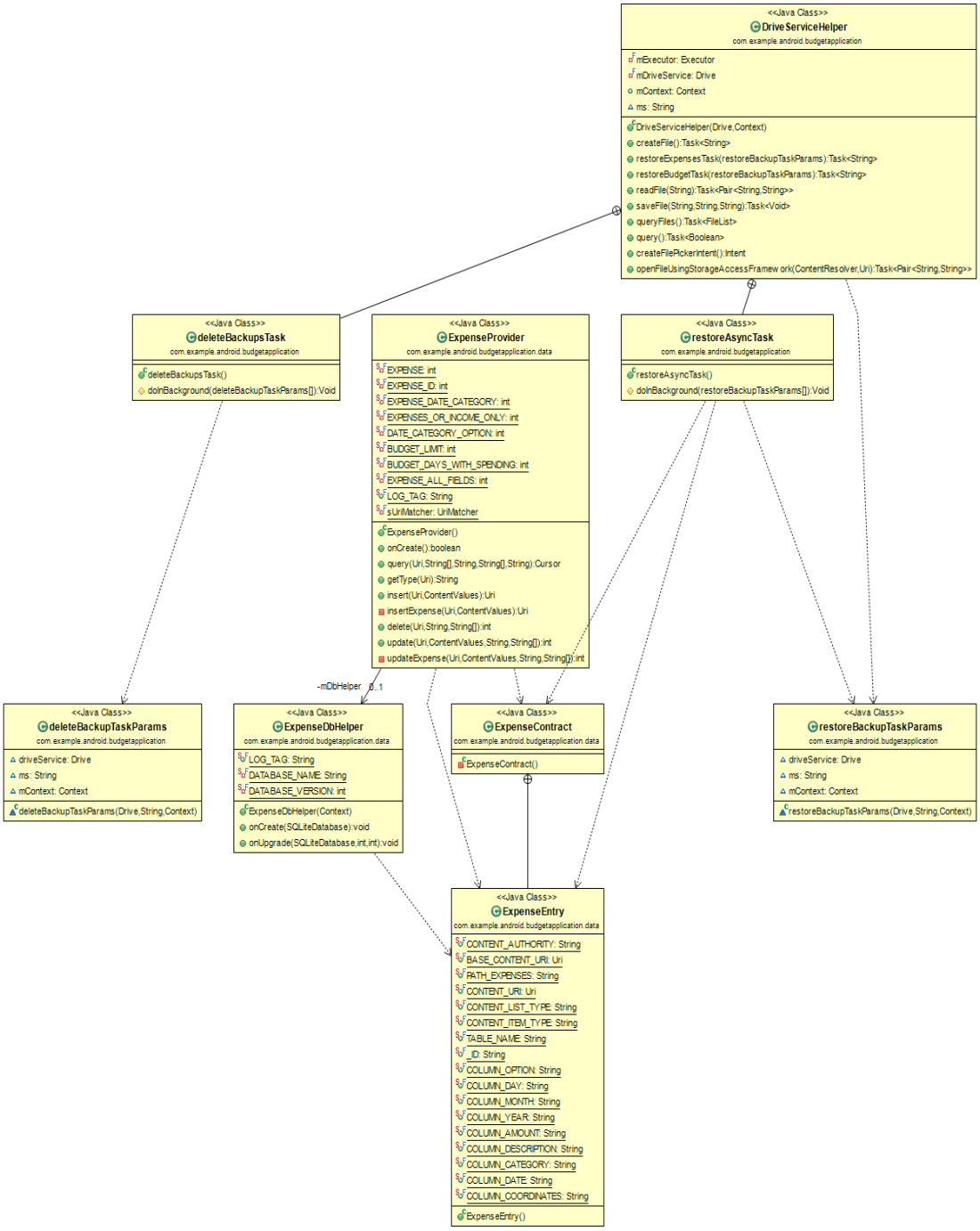


Figure 7.1: Class diagram, represent, view and manage dependent expenses

Chapter 8: Supposed spending rate

8.1: Methodology, tools, libraries and classes of interest

There are 3 main integral classes that support this functionality, namely BudgetListActivity, MnaualBudgetActivity, and BudgetAdapter. As its namesake, the BudgetListActivity displays created budgets using a ListView, while ManualBudgetActivity allows the user to create a budget. The more unique class here is the BudgetAdapter, which calculates supposed rates of spending using each budget record. As an adapter, it will then bind views with the budget records and supposed rates of spending in BudgetListActivity. Please note the need for more unique SQL queries used to retrieve data needed for the supposed rate of spending. The SQL queries will be explained in further detail below.

8.2: Requirements:

With the BudgetAdapter calculating the suppsed rate of spending and BudgetListActivtiy displaying each budget, functional requirement 4 is achieved, as the supposed rate of spending is derived and displayed to users to help in achieving a previously planned budget.

8.3: Implementation and testing:

In implementing this functionality, the author has performed a reasonable set of tests to solve possible erroneous situations or cases that might arise when trying to ensure supposed rates of spending are calculated correctly, as well as ensuring budget views are not distorted due to large values. As stated earlier in the project objective, the supposed rate of spending is the difference between the total amount spent in a period under a category and the allocated budget, divided by the number of days with no spending. In BudgetAdapter, the number of days in a period listed by a budget is obtained by calculating the difference in milliseconds between those two dates, converting that value later to days. We then obtain the amount spent within those two dates by using the following query on the expense table, where the selectionArg array contains data from the user's own budget:

```
String sql_budget_limit = "select sum(" + ExpenseContract.ExpenseEntry.COLUMN_AMOUNT + ") from (" +
        "select " + ExpenseContract.ExpenseEntry.COLUMN_AMOUNT + " from " + ExpenseContract.ExpenseEntry.TABLE_NAME +
        " where " + ExpenseContract.ExpenseEntry.COLUMN_DAY + "<=" + selectionArgs[0] + " AND " + ExpenseContract.ExpenseEntry.COLUMN_MONTH + "<= " + selectionArgs[1] +
        " AND " + ExpenseContract.ExpenseEntry.COLUMN_YEAR + "<= " + selectionArgs[2] + " AND " + ExpenseContract.ExpenseEntry.COLUMN_CATEGORY + "= " + selectionArgs[6] +
        " AND " + ExpenseContract.ExpenseEntry.COLUMN_OPTION + "= " + "'Expense'" + " INTERSECT " +
        "select " + ExpenseContract.ExpenseEntry.COLUMN_AMOUNT + " from " + ExpenseContract.ExpenseEntry.TABLE_NAME +
        " where " + ExpenseContract.ExpenseEntry.COLUMN_DAY + ">=" + selectionArgs[3] + " AND " + ExpenseContract.ExpenseEntry.COLUMN_MONTH + ">= " + selectionArgs[4] +
        " AND " + ExpenseContract.ExpenseEntry.COLUMN_YEAR + ">= " + selectionArgs[5] + " AND " + ExpenseContract.ExpenseEntry.COLUMN_CATEGORY + "= " + selectionArgs[6] +
        " AND " + ExpenseContract.ExpenseEntry.COLUMN_OPTION + "= " + "'Expense'" +
        ")";
```

Figure 8.1: SQL Query, spent amount within a period under a category

The above SQL query first obtains all records for a spending category starting from the budget's start date to the latest possible date. It then repeats the same query but in a different range, starting from the earliest possible date to the provided budget end date. It then obtains the intersection between those two results, which is all spending records within a period under of a particular spending category. Finally, it extracts the sum from that final table.

After getting the amount spent within a period, the application then queries the expense table in similar fashion as used to obtain the amount spent within a period. The difference is, this time the intersection of the two tables is queried for unique dates. The number of records in the result are the number of unique days of spending within a period. Below is the aforementioned SQL query

```
String sql_budget_days_with_spending = "select DISTINCT " + ExpenseContract.ExpenseEntry.COLUMN_DATE +  " from (" +
        "select " + ExpenseContract.ExpenseEntry.COLUMN_DATE + " from " + ExpenseContract.ExpenseEntry.TABLE_NAME +
        " where " + ExpenseContract.ExpenseEntry.COLUMN_DAY + "<=" + selectionArgs[0] + " AND " + ExpenseContract.ExpenseEntry.COLUMN_MONTH + "<= " + selectionArgs[1] +
        " AND " + ExpenseContract.ExpenseEntry.COLUMN_YEAR + "<= " + selectionArgs[2] + " AND " + ExpenseContract.ExpenseEntry.COLUMN_CATEGORY + "= " + selectionArgs[6] +
        " AND " + ExpenseContract.ExpenseEntry.COLUMN_OPTION + "= " + "'Expense'" + " INTERSECT " +
        "select " + ExpenseContract.ExpenseEntry.COLUMN_DATE + " from " + ExpenseContract.ExpenseEntry.TABLE_NAME +
        " where " + ExpenseContract.ExpenseEntry.COLUMN_DAY + ">=" + selectionArgs[3] + " AND " + ExpenseContract.ExpenseEntry.COLUMN_MONTH + ">= " + selectionArgs[4] +
        " AND " + ExpenseContract.ExpenseEntry.COLUMN_YEAR + ">= " + selectionArgs[5] + " AND " + ExpenseContract.ExpenseEntry.COLUMN_CATEGORY + "= " + selectionArgs[6] +
        " AND " + ExpenseContract.ExpenseEntry.COLUMN_OPTION + "= " + "'Expense'" +
        ")";
```

Figure 8.2: SQL Query, distinct dates with expenses within a period under a category

We then derive the balance left for a particular period by subtracting the amount spent within a period, from the user-defined spending limit for that same period. This spending limit value is obtained from budget records. After obtaining the balance left, the application simply divides it by the number of days within that period which have no spending. Thus, the supposed rate of spending is derived, and displayed together with each Budget-representing-row in BudgetListActivity.

Located in the following page is the class diagram of classes crucial in implementing this functionality:

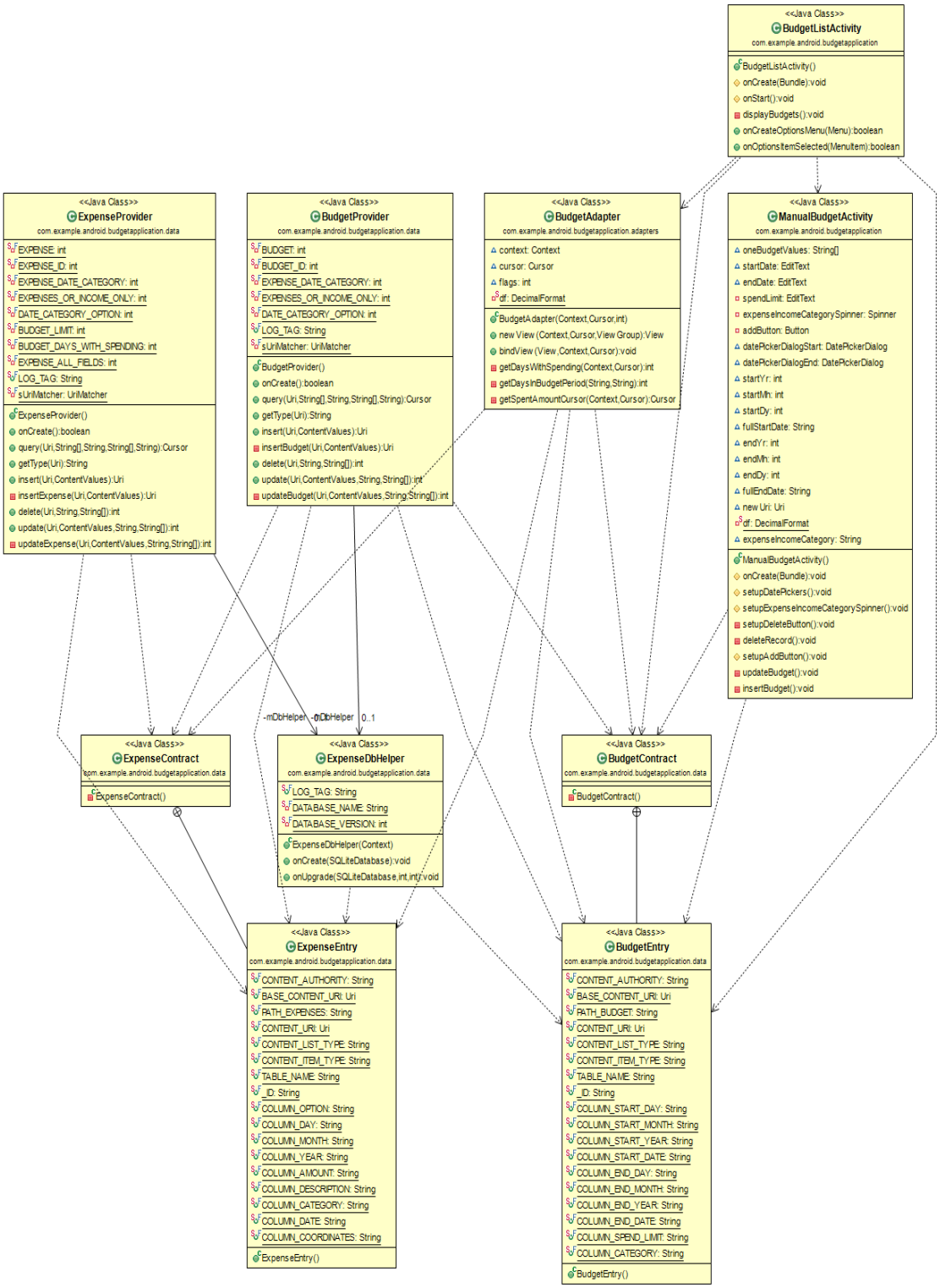


Figure 8.3: Class diagram, supposed spending rate

Chapter 9: Conclusion

9.1: Achievements and objectives

As mentioned in previous chapters, the broad categories of functions that include those used for the recording of expenses, the derivation of the supposed spending rate, and the representing of user expense data by accounts have been successfully implemented. With this, the application manages to cater for all use-cases defined in Chapter 3. Not only that, the application also satisfies all functional and non-functional requirements defined in Chapter 3. As a further note, the achievement of all requirements aligns with the project's motivation. The project also succeeds in helping the demographics mentioned in the problem statement, and ultimately, in its objectives.

9.2 Problems encountered

The author has encountered innumerable problems in the process of completing this project, the amount of which might be able to fill another report of this length. The major hurdles early on where the lack of knowledge in regards to Android development, as the author had not taken any course before the before the commencement of project 1, and had to make due with online courses and resources that served as a crash course of sorts. The online course provided sufficient introduction to various important topics like views, data adapters, and database functionality in regards to Android development, but only insofar as to complete the project in the online course, and in nowhere near sufficient amounts to aid in even completing any major functionality in this project 2.

One of the most time consuming problems in completing this project is identifying the correct views and view groups to use in displaying the daily expense data such that the data can be displayed in a user-friendly manner. There seemed to be innumerable choices for use in displaying daily expense data, both known and unknown to the author at the time. The hopelessness and lack of hand-holding in that regard induced quite a bit of desperation, nervousness, and uncertainty within the author, wondering whether this project can even have a good basic foundation. For example, the author initially had the idea to use a 3 level ExpandableListView, with level 0, 1 and 2 representing

dates, expense/income categories and expense/income records respectively. First of all, the author had to learn how the basic 2 level ExpanableListViews work, which was far beyond covered material in the online course. This took about 1 or 2 weeks.

Second, the author had to learn how to properly populate and display 3 level ExpandableListViews e, which took weeks more to adapt to the application's own data and by learning solely through online resources. Finally, with the technical aspect done, the author sought to conform the views to material design guidelines, to increase user-friendliness.

Unfortunately, the author did not find a way to edit the relevant view dimensions, and had to start over in re-thinking the best way to present the data. This was quite a blow to motivation, as a lot of work had been put in that direction. Fortunately, with more observation of other applications and extensive research and testing, suitable view groups and views were identified, namely a PageAdapter to represent each day's expense data in a separate page, populated with two ExpandableListViews, but this time they had only 2 levels instead of 3. With much effort and persistence the author manage to learn how to achieve technical feasibility in conforming the views to material design guidelines. Issues related to the appropriate identification and feasible utilization of views occupied the author's attention for about 3 months as the sole issue being focused on for this project. There were other major issues afterwards too, but none as time-consuming as this one.

The author's personal insight into the total research experience is that the whole process requires a lot of self-discipline, and certainly gives a new meaning to independent learning. Furthermore, it has expanded the horizon's of the author, by observing and detailing the approaches of other similar applications, their pros and cons, and engendering a certain respect for the development work put into each of those alternate applications. However, the author is also no less surprised and flummoxed by the features that are usable only after payment in these alternate applications, as they have not caused the user money to include many of those features in this project.

9.3 Novelties and contributions

The project has achieved in combining the three major functions

1. Accurate enough automated expense data extraction through OCR and a custom algorithm,
2. Viewing and managing separate family member expense data through accounts and backups, and
3. Deriving supposed rates of spending in one application,

All 3 above functionalities are free of charge, with none of the above functionalities requiring payment. From the alternative applications examined in Chapter 2, none of the applications have all 3 at once, paid or unpaid. Even those that might have 2 at once, many of them require payment. There might be other applications out there that have all 3 functionalities for free, but it is not feasible to examine all possible existing applications in Android ecosystem, thus this is only a comparison to the most popular applications and the topmost results returned in the Google Play Store. In that sense, this project contributes a novel combination of all 3 of the above functionalities, free of charge.

9.4 Improvements that can be made

The one major improvement that can instantly increase the utility of this project is a way to extract expense data of individual expense items that are part of a whole individual receipt. For example, extracting the expense amount of items like “bread” or “jam” within a grocery receipt. This will further decrease the burden of those who buy large quantities of items in a single trip, like families, helping them identify large and unnecessary expenditures in a more granular fashion and improve their spending behaviour while being able to spend time in other facets of their lives.

References:

BudgetBakers.com, 2021. *Wallet*. Mobile app. Version 8.2.271. Available from: https://play.google.com/store/apps/details?id=com.droid4you.application.wallet

Easy Expense Tracker, 2021. *Receipt Scanner: smart receipts & expense tracker*. Mobile app. Version 3.27.4. Available from: https://play.google.com/store/apps/details?id=com.easyexpense

Finsify, 2019. *Money Lover: Money Manager, Budget Expense Tracker*. Mobile app. Version 5.18.0.2021031707. Available from: https://play.google.com/store/apps/details?id=com.bookmark.money

Klapper, L., Lusardi, A. and van Oudheusden, P. (2014) "*Insights from the Standard & Poor's Ratings Services Global Financial Literacy*" Available from: < https://gflec.org/wp-content/uploads/2015/11/3313-Finlit_Report_FINAL-5.11.16.pdf?x37292 > [27 July 2019].

Mh Riley Ltd, 2021. *Spending Tracker*. Mobile app. Version 2.4.1. Available from: https://play.google.com/store/apps/details?id=com.mhriley.spendingtracker

Smart Receipts LLC, 2021. *Smart Receipts.* Mobile app. Version 4.21.0.2321. Available from: https://play.google.com/store/apps/details?id=wb.receipts

TimelyBills, 2021. *Money Manager, Budget, Expense Tracker, Bills, Loan.* Mobile app. Version 1.21.115. Available from: https://play.google.com/store/apps/details?id=in.usefulapp.timelybills

Poster

Image recognition expense extraction

Introduction

- The proposed application is an image recognition system that allows users to manually record expense data, as well as automate expense data recording through optical character recognition. Another proposed feature is the combination of different family member expenses into one account.

Methods

- The application is developed using Java with an SQLite Database using the Rapid Application Development methodology. The phases involved are requirements planning, user design, construction and cutover, where the construction and user design phases are iterative. Gantt charts were used to estimate the timeline for various deliverables.

Results

- The result is a budgeting application that is able to manually and automatically extract the expense results from receipt images, as well as combine family member income data. This application combines the convenience of mobile applications with the accuracy of human-based data extraction to further aid organisations and people in reducing the labour required to record expenses and incomes.

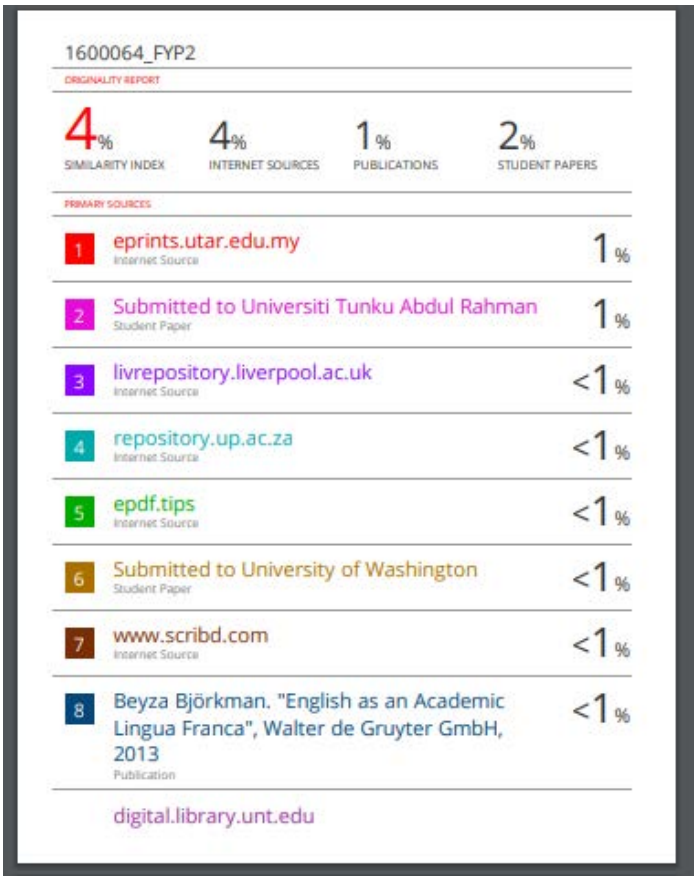
Discussion

- The application is innovative since there are not many free applications that possess accurate image recognition functionality together with relatively fast processing. In the future, the author hopes to increase the user-friendliness as well as enhance the overall style of the system's elements.

Conclusion

- This budgeting application is an offline Android-based Java application which is able to extract and record expense data into a database, as well as display the recorded expense data. It is able to extract expense data manually and automatically through OCR.

Turnitin Summary of Plagiarism Check



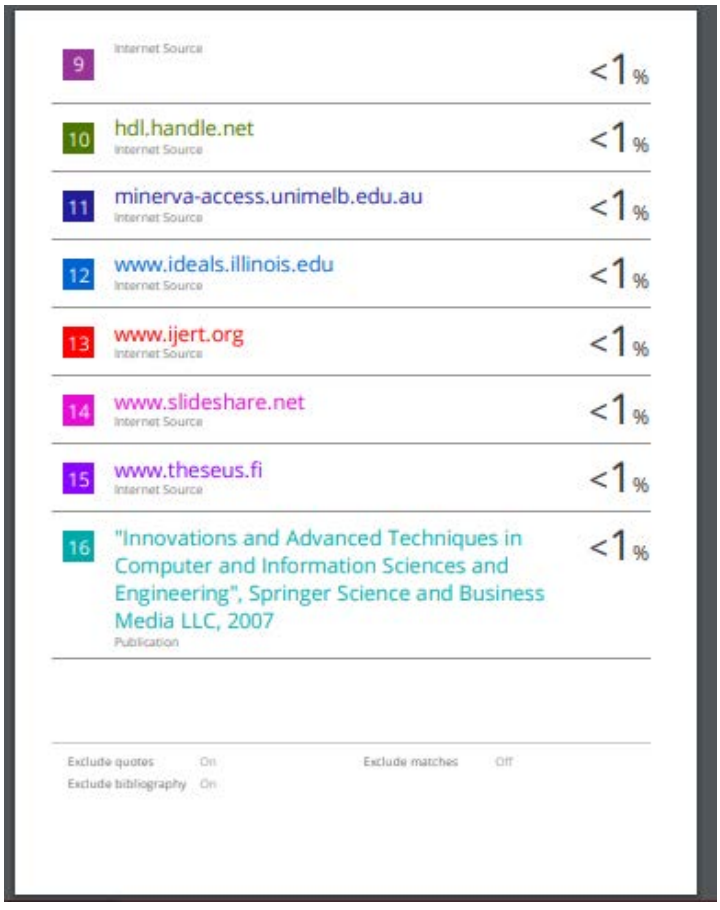
1600064_FYP2

ORIGINALITY REPORT

4%	4%	1%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	eprints.utar.edu.my Internet Source	1%
2	Submitted to Universiti Tunku Abdul Rahman Student Paper	1%
3	livrepository.liverpool.ac.uk Internet Source	<1%
4	repository.up.ac.za Internet Source	<1%
5	epdf.tips Internet Source	<1%
6	Submitted to University of Washington Student Paper	<1%
7	www.scribd.com Internet Source	<1%
8	Beyza Björkman. "English as an Academic Lingua Franca", Walter de Gruyter GmbH, 2013 Publication	<1%
	digital.library.unt.edu	



9	Internet Source	<1%
10	hdl.handle.net Internet Source	<1%
11	minerva-access.unimelb.edu.au Internet Source	<1%
12	www.ideals.illinois.edu Internet Source	<1%
13	www.ijert.org Internet Source	<1%
14	www.slideshare.net Internet Source	<1%
15	www.theseus.fi Internet Source	<1%
16	"Innovations and Advanced Techniques in Computer and Information Sciences and Engineering", Springer Science and Business Media LLC, 2007 Publication	<1%

Exclude quotes On
Exclude bibliography On
Exclude matches Off

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

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	KOK WEI JIN
ID Number(s)	16ACB00064
Programme / Course	BACHELOR OF COMPUTER SCIENCE (HONOURS)
Title of Final Year Project	IMAGE RECOGNITION EXPENSE EXTRACTION

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceeds the limits approved by UTAR)**
Overall similarity index: 4 % **Similarity by source** Internet Sources: 4 % Publications: 1 % Student Papers: 2 %	
Number of individual sources listed of more than 3% similarity: 0	

Parameters of originality required and limits approved by UTAR are as Follows:

(i) Overall similarity index is 20% and below, and

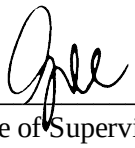
(ii) Matching of individual sources listed must be less than 3% each, and

(iii) Matching texts in continuous block must not exceed 8 words

Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.

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

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



Signature of Supervisor

Name: Yap Seok Gee

Date: 16/04/2021

Signature of Co-Supervisor

Name: _____

Date: _____

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	16ACB00064
Student Name	KOK WEI JIN
Supervisor Name	YAP SEOK GEE

TICK (√)	DOCUMENT ITEMS Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
√	Front Cover
✓	Signed Report Status Declaration Form
√	Title Page
✓	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
√	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Poster
✓	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)

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

I, the author, have checked and confirmed all the items listed in the table are included in my report. _____ (Signature of Student) Date: 16/04/2021	Supervisor verification. Report with incorrect format can get 5 mark (1 grade) reduction. _____ (Signature of Supervisor) Date: 16/04/2021